OpenKasugai-Controller Install Manual

v1.1.0

目次

0. はじめに

本書では、OpenKasugai のデモの実行環境を構築する手順を記載する。

0.1 注意点

本書を使用する際に注意すべき点を示す。
・資材一覧のバージョンに差分がないこと。
・強調文字は強調するポイント、強調文字はその中で特に注意するポイントを示す。
・章タイトル下の「対象」表記は、その操作手順を実施すべき物理サーバ種別を示す。
・各種操作の実行アカウントは特に明記が無い限り、”ubuntu”にて実行する想定である。従って、ディレクトリに関して”$HOME”は”/home/ubuntu/”を想定している。

・対象システム：
- 対象の K8s node ノード：アクセラレータ未搭載、Xilinx FPGA のみ搭載、NVIDIA GPU のみ搭載、Xilinx FPGA と Nvidia GPU の両方が搭載
  ※なお、各種 PCIe デバイスのホットプラグや FPGA の動的再構成には未対応。

0.2 手順概要

本書で記述している環境構築手順の概要を以下に示す。

環境構築手順一覧

章	大項目	説明
1	機器設置	物理サーバを設置し、拡張スロットを搭載する。
2	機器設定	100Gb スイッチの設定と接続を行う。
3	OS インストール	物理サーバに OS をインストールし設定を行う。
4	FPGA セットアップ	FPGA カードをセットアップする。
5	コンテナ管理基盤セットアップ	コンテナ管理基盤（K8s）をセットアップする。
6	GPU セットアップ	GPU カードをセットアップする。
7	各種処理モジュールのセットアップ	CPUFunction や GPUFunction 上で実行する各種処理モジュールや映像配信・映像受信ツール等の各コンテナをセットアップする。
8	コントローラのセットアップ	データフローを配備するためのコントローラをセットアップする。
9	評価手順	データフローを配備して、データ疎通を評価する。
10	付録	FAQ 関連。

0.3 想定機器一覧

本書で想定している環境の機器構成を以下に示す。

想定環境での使用機器一覧

項	構成物	種別	機器
1	K8s control plane	物理サーバ	• 富士通 PRIMERGY RX2540M6
2	K8s node		
3	100Gb Switch (OS：Mellanox ONYX)	Switch	Mellanox SN2100-CB2F
4	FPGA カード (フィルタリサイズ用)	FPGA カード	Xilinx Alveo U250
5	GPU カード (高度推論用)	GPU カード	NVIDIA A100（80GB）
6	100G NIC（Intel）	NIC	Intel E810CQDA2
7	100G NIC（Mellanox）		Mellanox ConnectX 5
8	AOC ケーブル	ケーブル	Mellanox MFA1A00-C003 互換 AOC ケーブル

物理環境構成図とソフトウェア構成図は、別紙「OpenKasugai-Controller-InstallManual_Attachment1」の「1. 想定環境図」シートを参照。

0.4　使用ソフトウェア・FPGA 回路バージョン一覧

本書で構築する環境で使用しているソフトウェアと FPGA 回路のバージョン一覧を以下に示す。

使用ソフトウェアバージョン一覧

ソフトウェア		バージョン
Ubuntu		22.04.5 LTS
Vivado		2023.1.0
DPDK		23.11.1 LTS
k8s		1.31.1
コンテナランタイム		CRI-O v1.31.0
CNI		Calico v3.28.1
Multus		4.1.1
SR-IOV デバイスプラグイン		3.7.0
SR-IOV CNI プラグイン		2.8.1
CRC（コントローラ）		1.1.0
kubebuilder		3.12.0
go		1.23.0
NVIDIA Driver		550.90.12
nvidia-container-toolkit		1.16.2
推論アプリコンテナ	DeepStream	7.0
	CUDA	12.2
	TensorRT	8.6.1.6-1+cuda12.0
	gcc	11.4.0
	GStreamer	1.20.3

FPGA 回路バージョン一覧

回路		ファイル名
フィルタ・リサイズ回路	mcs	OpenKasugai-fpga-example-design-1.0.0-1.mcs
	bit	OpenKasugai-fpga-example-design-1.0.0-2.bit

0.5 資材一覧

本書で構築する環境に必要な提供資材の一覧を以下に示す。

資材一覧

項	品名	説明	提供
1	FPGA 書き込みスクリプト	phase3 Flash へ FPGA バイナリデータ（MCS ファイル）書き込みスクリプト	・github(OpenKasugai) の hardware-drivers リポジトリ(v1.0.0)より取得
2	FPGA バイナリデータ	・mcs ファイル ・フィルタ・リサイズの bit ファイル	・github(OpenKasugai) の hardware-design リポジトリ(v1.0.0)より取得 ・ mcs ファイル ・ F/R の bit ファイル
3	FPGA ソフトウェア一式	FPGA ドライバおよびライブラリ	・github(OpenKasugai) の hardware-drivers リポジトリ(v1.0.0)より取得 ・FPGA ライブラリ・ドライバ
4	CRC ソースコード & テストデータ	カスタムリソースコントローラソフトのソースコード一式とテストデータ(YAML と JSON)	・github(OpenKasugai)の controller リポジトリ(v1.1.0)より取得 ・スケジューラ&各種 CRC のソース ・自動取得&CM 作成機能 ・FPGA 手動書込みツール ・FPGA 内リソースの回収確認ツール ・テストデータ ・試験スクリプト一式
5	推論処理モジュール	GPU を利用する推論 Pod のソースコードおよびコンテナイメージ	・ソースコード：github(OpenKasugai)の controller リポジトリ(v1.1.0)の sample-functions/functions/より取得 ・dma 版：gpu_infer_dma_plugins/ ・tcp 版：gpu_infer_tcp_plugins/
6	デコード処理モジュール	CPU でデコードを行う Pod のコンテナイメージ	・ソースコード：github(OpenKasugai)の controller リポジトリ(v1.1.0)の sample-functions/functions/cpu_decode/より取得
7	フィルタリサイズ処理モジュール (CPU 用)	CPU でフィルタリサイズを行う Pod のコンテナイメージ	・ソースコード：github(OpenKasugai)の controller リポジトリ(v1.1.0)の sample-functions/functions/cpu_filter_resize/ より取得
8	コピー分岐処理モジュール(CPU 用)	CPU でコピー分岐処理を行う Pod のコンテナイメージ	・ソースコード：github(OpenKasugai)の controller リポジトリ(v1.1.0)の sample-functions/functions-ext/cpu_copy_branch/ より取得
9	glue 処理モジュール(CPU 用)	CPU で glue 変換(dma→tcp)処理を行う Pod のコンテナイメージ	・ソースコード：github(OpenKasugai)の controller リポジトリ(v1.1.0)の sample-functions/functions-ext/ cpu_glue_dma_tcp/ より取得
10	デモ個別機能	PoC 用デモの映像配信ツール	・github(OpenKasugai)の controller リポジトリ(v1.1.0)の sample-functions/utils/より取得 ・ 配信サーバ：send_video_tool/

			・ 受信サーバ：rcv_vide_tool/
11	デモ用動画	配信時に利用する動画ファイル	以下などの外部サイトより取得 • https://pixabay.com/ja/videos/%E3%83%AA%E3%83%90%E3%83%97%E3%83%BC%E3%83%AB-%E6%A9%8B%E8%84%9A-%E9%A0%AD-46098/ • https://www.pexels.com/video/a-busy-downtown-intersection-6896028/

1. 機器設置

1.1 物理サーバ・Switch の設置

本書で例示するシステム物理構成および本章の設定対象を図 1 に示す。

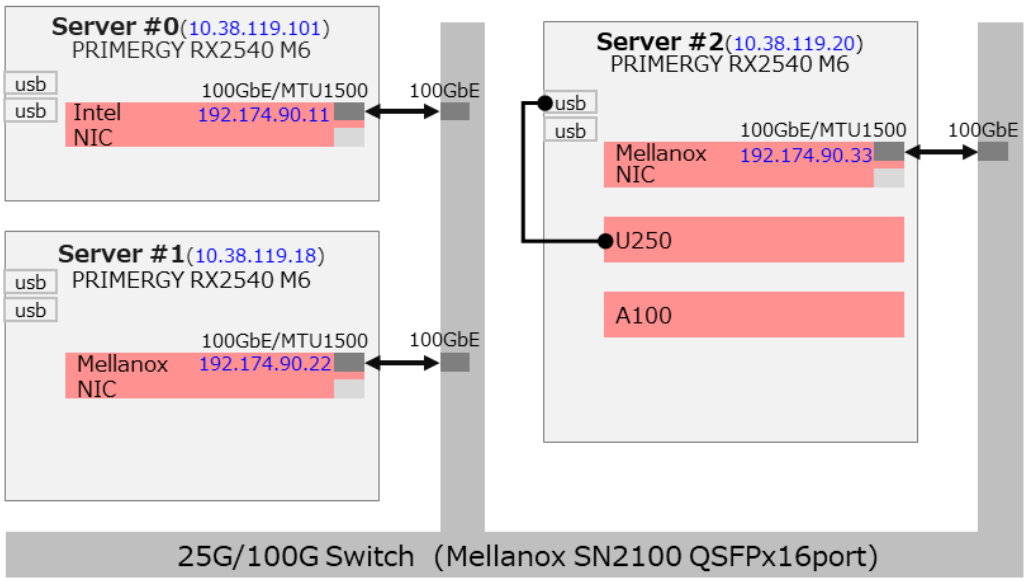


図 1 システム物理構成およびカード搭載スロット

上図のスロット位置に FPGA カード、GPU カード、NIC カードを搭載する。

サーバ名	用途	カード	枚数
Server #0	K8s control plane	Intel 100G NIC	1
Server #1	K8s node 1	Mellanox 100G NIC	1
Server #2	K8s node 2	Mellanox 100G NIC	1
		Alveo U250(FPGA カード)	1
		NVIDIA A100(GPU カード)	1

2. 機器設定

2.1 100Gb Switch の設定確認

システム物理構成における、本章の設定対象を図 2.1 に示す。本章では 100Gb スイッチの設定確認を行う。

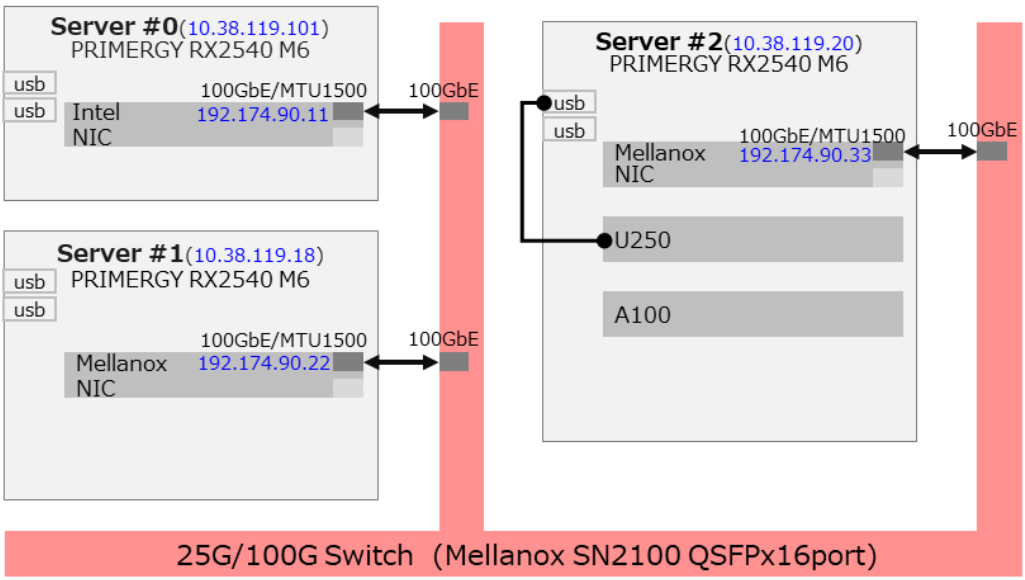


図 2.1 設定対象スイッチ

対象機器は、Mellanox SN2100-CB2F (OS：Mellanox ONYX)である。前面図を以下に示す。

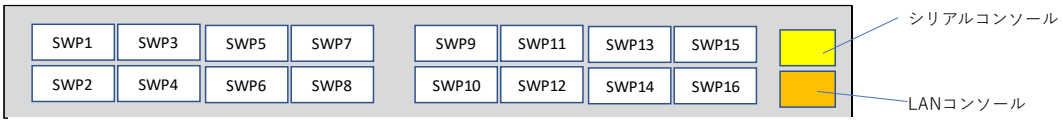


図 2.2 Mellanox SN2100-CB2F 前面図

Mellanox SN2100-CB2F の通信ポートは全部で 16 ポートあり、デフォルトは 100GbE となっている。
100G NIC と接続する場合は、特に設定変更は不要。

2.1.1 事前準備

シリアルコンソールにシリアル接続して、LAN コンソールの IP 設定を行う。

物理接続イメージは以下

・シリアルコンソール----LAN-シリアル変換---RS232C ケーブル----シリアル USB 変換----WinPC

WinPC におけるシリアルコンソール設定は以下

・ボーレート：115200

他はデフォルト設定

LAN コンソールに IP アドレス設定したら、LAN コンソールに LAN ケーブルを接続して、SSH 接続が可能となる。ID/PWD は管理者に確認する。

2.1.2 設定確認

(1) 全ポート設定確認コマンド例

100G NIC を接続しているポート(下の例：Eth1/1~1/4)が 100G で **Up** となっていることを確認する。

```
switch- sn2100-1 [standalone: K8s controller plane component] > show interfaces ethernet
status

Port                Operational state          Speed                Negotiation
----                -----------------          -----                -----------
Eth1/1              Up                        100G                 Auto
Eth1/2              Up                        100G                 Auto
Eth1/3              Up                        100G                 Auto
Eth1/4              Up                        100G                 Auto
Eth1/5              Down                      Unknown              Auto
Eth1/6              Down                      Unknown              Auto
Eth1/7              Down                      Unknown              Auto
Eth1/8              Down                      Unknown              Auto

. . .
```

2.1.3 接続確認

SW のポートと 100GNIC をケーブル接続する。

接続時に、SW の該当ポートのリンク LED が点灯することを確認する。

3. OS インストール

3.1 OS インストール

対象：K8s control plane、全ての K8s node

ソフトウェア全体構成における、本章の設定対象を図 3 に示す。
本章では物理サーバに対して OS のインストールを行う。

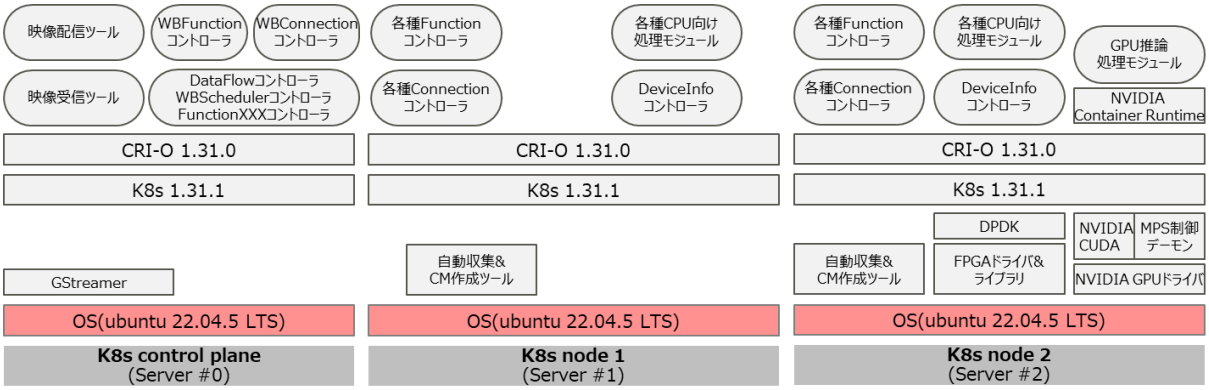


図 3 OS インストール対象サーバ

3.1.1 Ubuntu 22.04.5 インストール

以下の OS をインストールする。

・OS：ubuntu: 22.04.5
・ISO イメージ：ubuntu-22.04.5-live-server-amd64.iso

OS インストール時の推奨パラメータを以下に示す。なお、追加パッケージは不要である。
※注意：自動アップデートによるバージョン更新が行われないように注意すること。
自動アップデートオフ設定は後述（3.2.1 を参照）

OS インストール用パラメータシート

言語	English
キーボード設定(Layout/Variant)	Japanese
ベースインストール	Ubuntu Server ※”minimized”ではない
ネットワーク設定	自動アップデートを防ぐため、OS インストール後に設定する
プロキシ設定	プロキシサーバの有無に応じて設定
ミラーサーバ設定	特に変更しない (http://archive.ubuntu.com/ubuntu のまま)
ストレージ設定	・全ディスク選択 (カスタムストレージレイアウトは選択しない) （特にパーティション切る必要無し） （SWAP 領域の設定変更は不要） ・LVM 設定有効（Encrypt は無効）
ユーザ作成	それぞれに任意のパラメータを設定 ※本手順書では、「ログインユーザ名：ubuntu」で設定する
Ubuntu Pro	スキップ
SSH サーバインストール	有効(“Install OpenSSH server”を選択) それ以外(SSH 公開鍵の import など)は変更しない

インストール後に、インストールした OS 情報、カーネル情報を確認する。

```
$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.5 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy

$ uname -r
5.15.0-119-generic
```

3.2 OS 設定

3.2.1 自動アップデートオフ設定

対象：K8s control plane、全ての K8s node

Ubuntu の自動アップデートがデフォルト設定で ON になっているため OFF に設定する。
エディタで、/etc/apt/apt.conf.d/20auto-upgrades を下記内容に修正する。

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
  ↓
APT::Periodic::Update-Package-Lists "0";
APT::Periodic::Unattended-Upgrade "0";
```

3.2.2 ネットワーク設定

対象：K8s control plane、全ての K8s node

各サーバとも以下の 2 種類のネットワークを設定する。

・C-plane ネットワーク：OS の操作や kubernetes の制御用として使用するネットワーク
・D-plane ネットワーク：別紙「OpenKasugai-Demo」にて配備する DataFlow に流す映像を送信するためのネットワーク

●D-Plane 用ネットワークについて

推論を実施するための各処理を行う K8s node とは別のサーバに入力映像を流す映像配信ツールや推論結果（映像）を受け取る映像受信ツールが配備される構成をとる。上記構成のもとで「映像配信サーバ→各処理を行うサーバ→映像受信サーバ」の映像送信経路を D-plane ネットワークとして 100Gb Switch を経由して構築する。

本書では図 1 の各サーバの役割を次のようにする。

・映像配信サーバ：K8s control plane（Server #0）
・各処理サーバ：K8s node 1（Server #1）、K8s node 2（Server #2）
※全処理が 1 台のサーバに配備される場合も 2 台のサーバにまたがって配備される場合もある。
・映像受信サーバ：K8s control plane（Server #0）

このようなサーバ構成の下で，映像送信経路用の D-plane ネットワーク（192.174.90.XX）を構築する。
従って、図 1 における全サーバの 100GNIC に対して、D-plane ネットワークの設定を行う。

※別紙「OpenKasugai-Demo」を参考に DF を配備する際に、別紙「OpenKasugai-Demo」に記載している以上に DF を配備した場合、配備する DF の本数の増加によって映像受信ツールが受信した映像に、乱れやブロックノイズが見られるようになる可能性がある。その場合には、9.7 節の「Intel/Mellanox100/25GNIC の MTU9000 設定」の手順を実施する。

●各サーバでの設定手順

Intel 製/Mellanox 製の 25G/100G の NIC で共通の設定となる。

以下では、図 1 の想定環境（C-plane 用ネットワークは 10.38.119.0 系のネットワークで、D-plane 用は 192.174.90.0 系のネットワーク)のうち、Server#0 向けのネットワーク設定手順を例に記載する。

なお、Server#0 での C-plane 用 NIC のインターフェースは”**ens1f0**”、D-plane 用 NIC のインターフェースは” **ens7f0**”とする。

/etc/netplan/に、90-installer-config.yaml というファイルを作成し以下の様に編集する。

```
network:
  ethernets:
    ens1f0:
      addresses: [10.38.119.101/24]
      nameservers:
        addresses: [設定する DNS サーバ]
      routes:
        - to: default
          via: デフォルトゲートウェイアドレス(必要なら設定する)
    ens7f0:
      addresses: [192.174.90.11/24]
      mtu: 1500
  version: 2
```

作成したファイルのモードを以下に変更する。適用する。

```
$ sudo chmod 600 /etc/netplan/90-installer-config.yaml
$ ll /etc/netplan/
drwxr-xr-x  2 root root 4096 Nov 26 10:40 ./
drwxr-xr-x 96 root root 4096 Nov 26 08:57 ../
-rw-------  1 root root  354 Nov 26 08:57 50-cloud-init.yaml
-rw-------  1 root root  200 Nov 26 10:40 90-installer-config.yaml
```

システムへ適用する。以下コマンドで仮適用する。

```
$ sudo netplan try
[sudo] password for ubuntu:
Do you want to keep these settings?

Press ENTER before the timeout to accept the new configuration

Changes will revert in 102 seconds
```

上記メッセージのように特にエラーが出なければ(Warning は出ても構わない)、ここでリターンキーを押すと、” Configuration accepted.”というメッセージが出力されて仮適用が本適用となる。

以下の様に対象のインターフェースに IP アドレス/サブネットマスク幅（192.174.90.11/24）が設定されているか確認する。

```
$ ip addr show ens1f0
3: ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default
qlen 1000
    link/ether b4:96:91:9d:85:8c brd ff:ff:ff:ff:ff:ff
    altname enp66s0f0
    inet 10.38.119.101/24 brd 10.38.119.255 scope global ens1f0
       valid_lft forever preferred_lft forever
    inet6 fe80::b696:91ff:fe9d:858c/64 scope link
       valid_lft forever preferred_lft forever

$ ip addr show ens7f0
4: ens7f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default
qlen 1000
    link/ether b4:96:91:ca:54:b8 brd ff:ff:ff:ff:ff:ff
    altname enp174s0f0
    inet 192.174.90.11/24 brd 192.174.90.255 scope global ens7f0
       valid_lft forever preferred_lft forever
    inet6 fe80::b696:91ff:feca:54b8/64 scope link
       valid_lft forever preferred_lft forever
```

上記手順を残りのサーバ(図 1 の場合は Server#1, Server#2)でも実施する。

●各サーバでの NetworkSwitch(SN2100)接続ポート Speed 確認

100GNIC の場合は、以下のコマンド例のように、「Speed: 100000Mb/s」という表示でリンクスピードが 100GbE となっていることを確認する。

```
$ sudo ethtool ens7f0
Settings for ens93f0:
       Supported ports: [ ]
       Supported link modes: 100000baseCR4/Full
       (中略)
       Speed: 100000Mb/s
       Duplex: Full
       Port: Other
       PHYAD: 0
       Transceiver: internal
       Auto-negotiation: off
       Current message level: 0x00000007 (7)
                              drv probe link
       Link detected: yes
```

3.2.3 Proxy の設定

対象：K8s control plane、全ての K8s node

Proxy を使用する場合、/etc/environment に以下の設定を行う。

```
$ sudoedit /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin"
http_proxy="http://user:password@[プロキシサーバ]:[port]"
https_proxy="http://user:password@[プロキシサーバ]:[port]"
HTTP_PROXY="http://user:password@[プロキシサーバ]:[port]"
HTTPS_PROXY="http://user:password@[プロキシサーバ]:[port]"
no_proxy="127.0.0.1,localhost,(ホストの IP)"
NO_PROXY="127.0.0.1,localhost,(ホストの IP)"
```

3.2.4 Kernel のアップグレード

対象：K8s control plane、全ての K8s node

Kernel のバージョンが動作検証済みの 5.15.0-122-generic より古い場合はアップグレードする。
※異なるバージョンでも動作する想定だが、動作は未検証。

```
$ sudo apt update
$ sudo apt install linux-image-5.15.0-122-generic linux-headers-5.15.0-122-generic
linux-modules-extra-5.15.0-122-generic
$ sudo reboot

(リブート後、Kernel がアップグレードされたことを確認)
$ uname -r
5.15.0-122-generic
```

3.2.5 時刻同期

対象：K8s control plane、全ての K8s node

NTP サーバを設定する

```
/etc/systemd/timesyncd.conf を下記で編集する
[Time]
NTP= [設定する NTP サーバを記載]


$ sudo systemctl restart systemd-timesyncd
$ sudo systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
     Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled; vendor
preset: enabled)
     Active: active (running) since Wed 2022-10-19 15:39:05 JST; 1 months 9 days ago
       Docs: man:systemd-timesyncd.service(8)
   Main PID: 2240 (systemd-timesyn)
     Status: "Initial synchronization to time server XX.XX.XX.XX:XX (設定した NTP サーバ
名)."
      Tasks: 2 (limit: 154194)
     Memory: 19.4M
     CGroup: /system.slice/systemd-timesyncd.service
             mq2240 /lib/systemd/systemd-timesyncd
```

時刻同期が設定されていることを確認する

```
$ timedatectl timesync-status
      Server: XX.XX.XX.XX (設定した NTP サーバ名)  ※設定した NTP サーバとなっていること
Poll interval: 34min 8s (min: 32s; max 34min 8s)
         Leap: normal
      Version: 4
      Stratum: 5
    Reference: A00EE46
    Precision: 1us (-20)
Root distance: 74.431ms (max: 5s)
       Offset: +163us
        Delay: 479us
       Jitter: 298us
 Packet count: 1690
    Frequency: -0.415ppm
```

3.2.6 Hugepage の設定

対象：全ての K8s node

OS や CPU の負担を減らすためにメモリ管理上 1 ページのサイズを標準から大きくする。その為、サーバに Hugepage の設定を実施する。
以下に、ubuntu での Hugepage 設定方法と確認の例を示す。

設定方法

/etc/default/grub の編集

```
GRUB_CMDLINE_LINUX_DEFAULT="default_hugepagesz=1G hugepagesz=1G hugepages=32"
```

※GRUB_CMDLINE_LINUX_DEFAULT に hugepage 関連のブートオプションを追記する。
上記設定で hugepagesz=1G で 32 ページ分が起動時に確保される設定となる。

※ページサイズ：x86 系システムでは 1 ページのサイズは 4kb である。HugePage のサイズは x86 系システムでは 2MB や 1GB が設定可能である。データストリームを処理するプロセスに割り当てる事から 1GB のサイズを設定する。
※ページ数(hugepages)：データストリームを処理するプロセス数に応じて設定する。

- Grub 設定の反映と再起動

```
$ sudo update-grub
$ sudo reboot
```

- 確認方法

下記コマンドを入力する。

```
$ cat /proc/meminfo
...
HugePages_Total: 32 <- 32 ページ分確保されていること
HugePages_Free: 32
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 1048576 kB <- hugepage サイズが 1GB になっている
```

4. FPGA セットアップ

本章は FPGA を使用する場合にのみ実行する手順となる。FPGA を使用しない(FPGA 搭載サーバを用いない)場合は本章の手順は実施不要である。

想定システムの物理構成における、本章の設定対象を図 4.1 に示す。
本章では K8s node 2 に搭載された FPGA カード（AlveoU250）に対して以下の手順で書き込みを行う。

・まず FPGA と USB 接続したホストマシンに Vivado をインストールして MCS ファイルの書込みを行う。
・その後、K8s node に Bitstream 書込みソフト（Mcap）をインストールして Bitstream（BIT ファイル）の書込みを行う手順については、後章にて記載する。

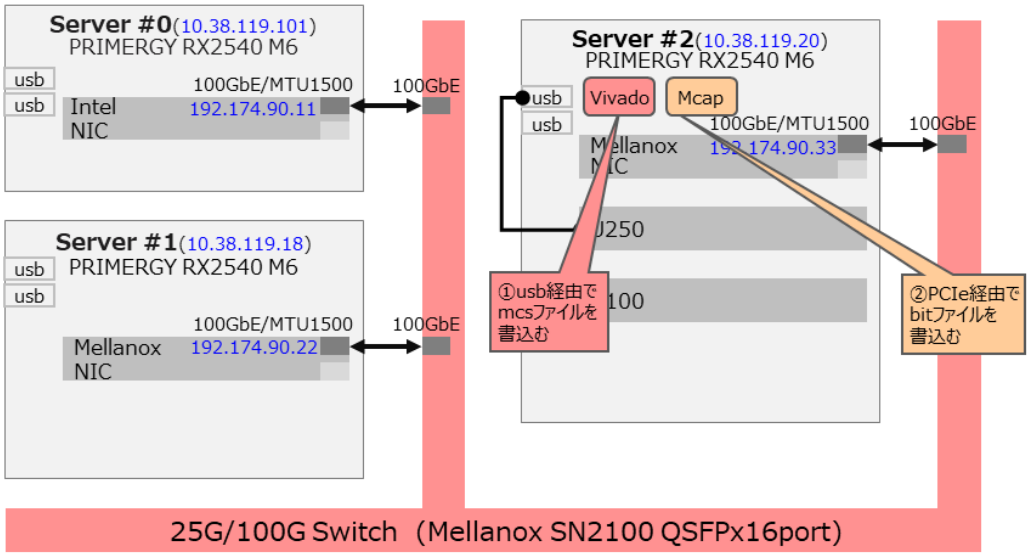


図 4.1 書込み対象 FPGA カードと書込みツール

4.1 Vivado のインストール

対象：mcs ファイルの書込みを行うホスト（想定環境では Sever #2 で実施）

FPGA カードに MCS ファイルを書込むための Vivado をインストールする。

●事前準備：Vivado の入手
https://japan.xilinx.com/support/download.html/content/xilinx/ja/downloadNav/vivado-design-tools/archive.html にアクセスし、2023.1 のアーカイブから「AMD 統合インストーラー FPGA およびアダプティブ SoC 用) 2023.1 SFD」をダウンロードする。
※ダウンロードに AMD のアカウントが必要なため、AMD のアカウントがない場合は取得すること
※ダウンロードには非常に時間がかかるため注意

4.1.1 必要パッケージのインストール

```
$ sudo apt update
$ sudo apt install dpkg-dev libtinfo5 libncurses5
```

4.1.2　インストール用コンフィグファイルの生成

取得した Vivado を展開し、展開されたディレクトリに入って、インストール用コンフィグファイルの生成を実施する。

```
$ tar xvfz Xilinx_Unified_2023.1_0507_1903.tar.gz
$ cd Xilinx_Unified_2023.1_0507_1903
$ ./xsetup -b ConfigGen
```

以下の選択肢が提示されるので、最初は「2. Vivado」を、２番目は「1. Vivado ML Standard」を選ぶ。

```
Running in batch mode...
Copyright (c) 1986-2022 Xilinx, Inc.  All rights reserved.
INFO : Log file location - /$HOME//.Xilinx/xinstall/xinstall_1666235568828.log
Select a Product from the list:
1. Vitis
2. Vivado
3. On-Premises Install for Cloud Deployments (Linux only)
4. BootGen
5. Lab Edition
6. Hardware Server
7. PetaLinux
8. Documentation Navigator (Standalone)
Please choose: 2
INFO : Config file available at /$HOME//.Xilinx/install_config.txt. Please use -c
<filename> to point to this install configuration.

Select an Edition from the list:
1. Vivado ML Standard
2. Vivado ML Enterprise

Please choose: 1

INFO : Config file available at /home/ubuntu/.Xilinx/install_config.txt. Please
use -c <filename> to point to this install configuration.
```

4.1.3 Vivado のインストール

生成された install_config.txt を指定して以下でインストールを実行する。

```
$ sudo ./xsetup --agree XilinxEULA,3rdPartyEULA --batch Install –config
$HOME/.Xilinx/install_config.txt
```

※$HOME/.Xilinx/install_config.txt は--config で指定するパラメータ

上記手順実行時、以下のエラーが発生した場合には、以下の対処を実施し再度 Vivado のインストールを実行する。

エラー内容

```
ERROR: Program group entry, Xilinx Design Tools, already exists for 2023.1. Specify
a different program group entry.
```

対処内容

```
$ cd ~/.config/menus/applications-merged
$ rm Xilinx¥ Design¥ Tools.menu
```

4.1.4 Vivado コマンド実行に必要なシェルスクリプトの実行

```
$ source /tools/Xilinx/Vivado/2023.1/settings64.sh
```

再起動してもすぐに使える様にするために.bashrc と.bash_profile に追記しておく。

- ~/.bash_profile：以下を追記。(~/.bash_profile が存在しなければ新規作成する)

```
if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
```

- ~/.bashrc：以下を追記

```
source /tools/Xilinx/Vivado/2023.1/settings64.sh
```

※root でも実行するので、root でも同様の設定をしておくと良い

(1) Vivado がインストールされていることを確認

下記コマンドを入力し、下記の赤文字部(Vivado v2023.1 (64-bit))が表示されることを確認する。
バージョン(v2023.1)が正しいこともチェックする。

```
$ vivado -version
vivado v2023.1 (64-bit)
Tool Version Limit: 2023.05
SW Build 3865809 on Sun May  7 15:04:56 MDT 2023
IP Build 3864474 on Sun May  7 20:36:21 MDT 2023
SharedData Build 3865790 on Sun May 07 13:33:03 MDT 2023
Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.
```

(2) JTAG ドライバをインストール

下記コマンドを入力し、JTAG ドライバをインストールする

```
$ cd /tools/Xilinx/Vivado/2023.1/data/xicom/cable_drivers/lin64/install_script/install_drivers/
$ ls
52-xilinx-digilent-usb.rules      52-xilinx-ftdi-usb.rules      52-xilinx-pcusb.rules
install_digilent.sh  install_drivers  setup_pcusb  setup_xilinx_ftdi
$ sudo ./install_drivers
INFO: Installing cable drivers.
INFO: Script name = ./install_drivers
INFO: HostName = server2
INFO: RDI_BINROOT= .
INFO:                Current                working                dir                =
/tools/Xilinx/Vivado/2023.1/data/xicom/cable_drivers/lin64/install_script/install_dr
ivers
INFO: Kernel version = 5.15.0-86-generic.
INFO: Arch = x86_64.
Successfully installed Digilent Cable Drivers
--File /etc/udev/rules.d/52-xilinx-ftdi-usb.rules does not exist.
--File version of /etc/udev/rules.d/52-xilinx-ftdi-usb.rules = 0000.
--Updating rules file.
--File /etc/udev/rules.d/52-xilinx-pcusb.rules does not exist.
--File version of /etc/udev/rules.d/52-xilinx-pcusb.rules = 0000.
--Updating rules file.

INFO: Digilent Return code = 0
INFO: Xilinx Return code = 0
INFO: Xilinx FTDI Return code = 0
INFO: Return code = 0
INFO: Driver installation successful.
CRITICAL WARNING: Cable(s) on the system must be unplugged then plugged back in order
for the driver scripts to update the cables.
```

(3) サーバのリブート

```
$ sudo reboot
```

4.2 FPGA カード書込み

フィルタリサイズ FPGA に対するコンフィグレーションデータの書込み手順について述べる。

● 使用条件

ダウンロード用 USB ケーブルがホストと FPGA カード 間で接続されていること
（USB 経由で MCS ファイルをホストから FPGA に書込むため。）

● 事前準備

○Xilinx ツールインストールパスの確認（FPGA と USB ケーブルで接続しているホスト）：Vivado を参照するのでインストールパスを確認する。

※本ドキュメントでは**"/tools/Xilinx/Vivado/2023.1/"**にインストールされているとする。

○FPGA カードの状態を確認

・MCS ファイルが書き込まれていない場合の出力例

```
$ lspci |grep Xilinx
1f:00.0 Memory controller: Xilinx Corporation Device 903f
```

・MCS ファイルが書き込まれている場合の出力例

```
$ lspci |grep Xilinx
1f:00.0 Processing accelerators: Xilinx Corporation Device 5004
```

このように MCS ファイルが書き込まれている場合でも、そのまま以降の手順を進めて問題無い。

4.2.1 Vivado で MCS ファイル書き込み

対象：mcs ファイルの書込みを行うホスト（想定環境では Sever #2 で実施）

(1) Vivado 用 USB ドライバのインストール

4.1.4 の(2)を実施。既に実施している場合はスキップ。

(2) リポジトリの clone

資材一覧に記載の github の hardware-design と hardware-drivers を取得する。

```
$ cd ~
$ git clone https://github.com/openkasugai/hardware-design.git
$ git clone https://github.com/openkasugai/hardware-drivers.git
```

(3) MCS ファイルの用意

ビルド済 MCS/Bitstream を使用する場合は、(2)で取得したリポジトリの配下に含まれているため準備は不要。

自身でビルドを実施する場合は、ビルド手順書(~/hardware-design/BUILD.md)を参照してビルドすること。

(4) MCS ファイル書込み用コマンド(run_flash.sh)の用意と MCS ファイルのコピー

tools/run_flash ディレクトリに移動し、書き込む mcs ファイルをコピーする。

(MCS ファイルは 0 で準備したものを使用する想定)

```
$ cd ~/hardware-drivers/tools/run_flash
$ cp ~/hardware-design/example-design/bitstream/OpenKasugai-fpga-example-design-1.0.0-1.mcs .
```

※run_flash.sh コマンドについて

- コマンド概要
  - FPGA カード上の FlashMemory へデータの書き込みが行われる。1 度書き込んだら、コールドリブートしても自動で FPGA へ書き込まれる。（リブートのたびに書き込み作業を行う必要がない。）
- コマンド引数
  - -t： 書きこむファイル名(拡張子 mcs)
    ※指定する MCS ファイルは、run_flash.sh と同じディレクトリに配置すること
  - -i：書込み先 FPGA を指定。
    0 を指定した場合は、以降の手順③の jtag target コマンドで確認する「1 個目検知 FPGAID」の FPGA に書き込まれる
    1 を指定した場合は、以降の手順③の jtag target コマンドで確認する「2 個目検知 FPGAID」の FPGA に書き込まれる

(5)　MCS ファイルの ROM への書込み

run_flash.sh を使って FPGA に MCS ファイルを書き込む。-t オプションで MCS ファイルを指定し、-i オプションでデバイスのインデックスを指定する。

“Flash programming completed successfully” が表示されれば成功。

```
$ ./run_flash.sh -t OpenKasugai-fpga-example-design-1.0.0-1.mcs -i 0
# (中略)
INFO: [Labtoolstcl 44-377] Flash programming completed successfully
program_hw_cfgmem: Time (s): cpu = 00:00:04 ; elapsed = 00:32:58 . Memory
(MB): peak = 3755.586 ; gain = 252.000 ; free physical = 490216 ; free virtual
= 493877
INFO: [Common 17-206] Exiting Vivado at Fri Oct 28 16:55:10 2022...
```

FPGA が複数枚ある場合は各 FPGA に対して書込みが必要。書込む際にはデバイスのインデックスを変更すること。以下は 2 枚目の FPGA への書込みの例。

```
$ ./run_flash.sh -t OpenKasugai-fpga-example-design-1.0.0-1.mcs -i 1
# (中略)
INFO: [Labtoolstcl 44-377] Flash programming completed successfully
program_hw_cfgmem: Time (s): cpu = 00:00:04 ; elapsed = 00:32:58 . Memory
(MB): peak = 3755.586 ; gain = 252.000 ; free physical = 490216 ; free virtual
= 493877
INFO: [Common 17-206] Exiting Vivado at Fri Oct 28 16:55:10 2022...
```

完了後、ホストをコールドリブートする。その後、IPMITool や手動などで電源を投入する。

```
$ sudo poweroff
```

● 1 章の図 1 の想定環境における MCS ファイル書込み手順の例

K8s node2 用サーバ(Server #2)に FPGA カードが 1 枚搭載されており、ホスト側の USB I/F も同じサーバである。従って、Server #2 に Vivado をインストールし、1 枚の FPGA カードに MCS ファイルを書込む。事前に 4.2.1 節の(4)までを実施しておくこと。

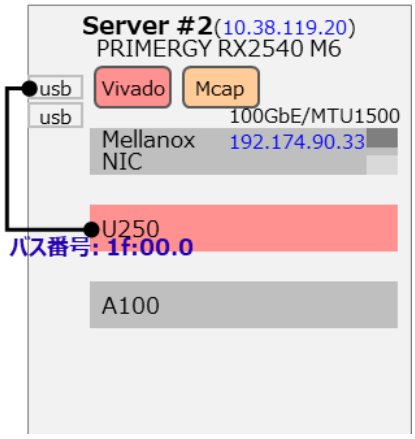


図 4.3 USB 経由で FPGA に mcs ファイルを書込む

① usb ケーブルの接続

FPGA と K8s node のホスト側 USB I/F を接続する。既に接続している場合はスキップ。

② FPGA の FPGAID を確認(xsdb コマンドで connect して確認)

Vivado コマンドを実行するための設定を行う。4.1.4 節にて~/.bashrc に定義済みであれば不要。

```
$ source /tools/Xilinx/Vivado/2023.1/settings64.sh
```

xsdb コマンドコンソール起動

```
$ xsdb
  rlwrap: warning: your $TERM is 'xterm' but rlwrap couldn't find it in the
  terminfo database. Expect some problems.

  ****** Xilinx System Debugger (XSDB) v2023.1
    **** Build date : Oct 19 2021-03:13:42
      ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.

xsdb%
```

connect コマンドで FPGA カードと接続

```
xsdb% connect
  attempting to launch hw_server

  ****** Xilinx hw_server v2023.1.0
    **** Build date : Oct  6 2021 at 23:40:43
      ** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.

  INFO: hw_server application started
  INFO: Use Ctrl-C to exit hw_server application

  INFO: To connect to this hw_server instance use url: TCP:127.0.0.1:3121

  tcfchan#0
xsdb%
```

jtag target コマンド実行。サーバに搭載している数だけ FPGA が検出されることを確認する。

```
xsdb% jtag target
  1  Xilinx A-U250-P64G FT4232H 21330621T00YA
     2  xcu250 (idcode 04b57093 irlen 24 fpga)
        3  bscan-switch (idcode 04900101 irlen 1 fpga)
           4  debug-hub (idcode 04900220 irlen 1 fpga)

xsdb% exit
```

「検知した FPGAID」

③ 検出した FPGA に書込み

“Flash programming completed successfully” が表示されれば成功

```
$ cd ~/hardware-drivers/tools/run_flash
$ ./run_flash.sh -t OpenKasugai-fpga-example-design-1.0.0-1.mcs -i 0
※中略
INFO: [Labtoolstcl 44-377] Flash programming completed successfully
program_hw_cfgmem: Time (s): cpu = 00:00:04 ; elapsed = 00:32:58 . Memory
(MB): peak = 3755.586 ; gain = 252.000 ; free physical = 490216 ; free virtual
= 493877
INFO: [Common 17-206] Exiting Vivado at Fri Oct 28 16:55:10 2022...
```

・ ./run_flash.sh の引数”-i”に指定する値は jtag target コマンドで検出された順番に依存し、1 番目に検出した FPGA デバイスを指定する場合は”0”、2 番目に検出した FPGA デバイスを指定する場合は”1”を指定する。今回は FPGA は 1 枚のみなので”0”

(6) FPGA カードの状態確認

MCS ファイル書き込みしてコールドリブートした後は`Device 903f となることを確認。

```
$ lspci |grep Xilinx
1f:00.0 Processing accelerators: Xilinx Corporation Device 903f
```

4.2.2 Mcap で Bitstream（子 bs）書き込み

対象：FPGA を搭載している全ての K8s node (想定環境では Sever #2 で実施)

FPGA への子 bs の書き込みは、8.4.1 節にて詳細を説明する。

5. コンテナ管理基盤セットアップ

ソフトウェア全体構成における、本章の設定対象を図 5 に示す。
本章では各サーバに K8s 関連コンポーネントをインストールし、K8s クラスターの構築を行う。

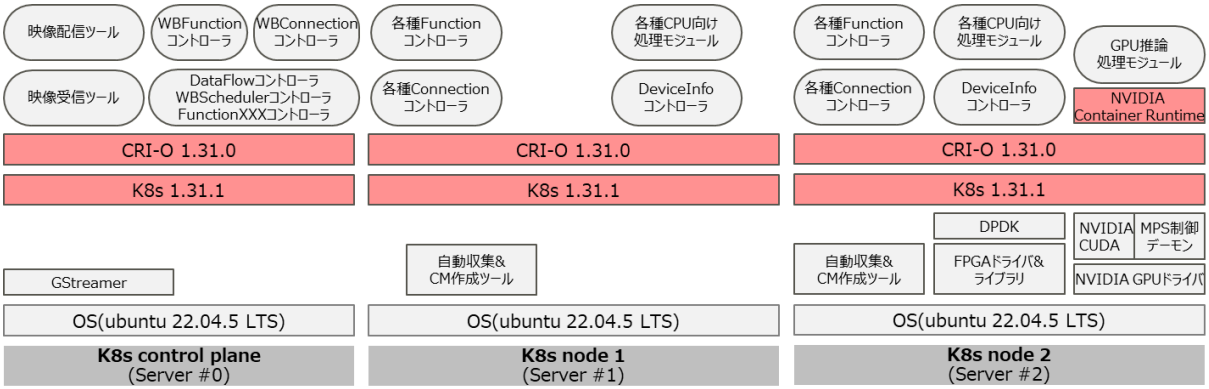


図 5 K8s 関連コンポーネントインストール対象

また本章において K8s control plane／K8s node に対して必要な手順の流れを以下一覧に示す。

	K8s control plane	K8s node
5.1 事前準備	5.1.1 各種設定	
	5.1.2 ソフトインストール	
	5.1.3 iptables 設定	
5.2 K8s のインストール	5.2.1 K8s 1.31.1 のインストール	
5.3 CRI-O のインストール	5.3.1 CRI-O v1.31.0 インストール	
	—	5.3.2 NVIDIA container-toolkit インストール（GPU あり）
	—	5.3.3 CRI-O 側の設定ファイルの編集 （GPU なし）
	5.3.4 CRI-O を再起動	
5.4 K8s クラスター構築	5.4.1 calico の manifest をダウンロード	—
	5.4.2 K8s クラスター構築	—
	5.4.3 calico の適用	—
	—	5.4.4 K8s クラスターに参加
5.5 SR-IOV CNI プラグインセットアップ	—	5.5.1～5.5.3 の各手順
5.6 Multus のインストール	5.6.1～5.6.3 の各手順	—

5.1 事前準備

5.1.1 各種設定

対象：K8s control plane、全ての K8s node

1. Swap が有効になっていると kubelet が起動エラーとなるため必ず swap 領域を無効化する。

```
$ sudo free -m
              total        used        free      shared  buff/cache   available
Mem:           3789         185        3491           8         112        3426
Swap:          3071           0        3071
$ sudo swapoff -a
$ sudo free -m
              total        used        free      shared  buff/cache   available
Mem:           3789         184        3493           8         112        3427
Swap:             0           0           0
```

再起動すると再度有効化してしまうため、/etc/fstab を下記に修正して永続的に swap を無効化。

```
$ sudoedit /etc/fstab

#/swap.img     none  swap  sw    0     0     （この行をコメントアウト）
```

2. ホスト名を設定する。ホスト名は任意だが、それぞれわかりやすい名前にすること。
   ここでは、図 1 に合わせてホスト名を設定した場合の例を示す。

```
=== K8s controller plane 用サーバ（Server #0）の例 ===
$ hostnamectl set-hostname server0
$ hostnamectl
   Static hostname: server0
   (中略)
  Operating System: Ubuntu 22.04.5 LTS
（以下略）

=== K8s node 1 用サーバ（Server #1）の例 ===
$ hostnamectl set-hostname server1
$ hostnamectl
   Static hostname: server1
   (中略)
  Operating System: Ubuntu 22.04.5 LTS
（以下略）

=== K8s node 2 用サーバ（Server #2）の例 ===
$ hostnamectl set-hostname server2
$ hostnamectl
   Static hostname: server2
   (中略)
  Operating System: Ubuntu 22.04.5 LTS
（以下略）
```

3. K8s の K8s control plane と K8s node のノードの IP アドレスに設定する IP アドレスを確認し、次の手順 4)にて/etc/hosts に設定する。
   ここでは、図 1 に合わせてホスト名を設定した場合の例を示す。

```
$ ip addr show

=== K8s controller plane 用サーバ（Server #0）の出力例 ===
  （途中省略）
8: ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default qlen 1000
    link/ether b4:96:91:9d:79:80 brd ff:ff:ff:ff:ff:ff
    inet 10.38.119.101/24 brd 10.38.119.255 scope global ens1f0
       valid_lft forever preferred_lft forever
    inet6 fe80::4ab:1eff:feaa:7e28/64 scope link
       valid_lft forever preferred_lft forever
  （以下省略）

=== K8s node 1 用サーバ（Server #1）の出力例 ===
（途中省略）
7: ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default qlen 1000
    link/ether 3c:ec:ef:f6:0e:9e brd ff:ff:ff:ff:ff:ff
    inet 10.38.119.18/24 brd 10.38.119.255 scope global ens1f0
       valid_lft forever preferred_lft forever
    inet6 fe80::247e:8ff:fe43:2ccb/64 scope link
       valid_lft forever preferred_lft forever
 （以下省略）

=== K8s node 2 用サーバ（Server #2）の出力例 ===
（途中省略）
7: ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default qlen 1000
    link/ether 3c:ec:ef:f6:0e:9e brd ff:ff:ff:ff:ff:ff
    inet 10.38.119.20/24 brd 10.38.119.255 scope global ens1f0
       valid_lft forever preferred_lft forever
    inet6 fe80::247e:8ff:fe43:2ccb/64 scope link
       valid_lft forever preferred_lft forever
 （以下省略）
```

4. /etc/hosts に、K8s control plane と K8s node の両ノード（クラスタ内ノード）を環境にあわせて登録する。
   ここでは、図 1 に合わせてホスト名を設定した場合の例を示す。

```
sudoedit /etc/hosts

10.38.119.101 server0
10.38.119.18 server1
10.38.119.20 server2
```

5.1.2 ソフトインストール

対象：K8s control plane、全ての K8s node

1. wget が入っていない場合はインストールを行う。

```
$ sudo apt install wget
```

2. git が入っていない場合はインストールを行う。

```
$ sudo apt update
$ sudo apt install git-all
```

```
$ git  config  --global  http.sslVerify  false
$ cat ~/.gitconfig
（設定されているか確認）
```

5.1.3 iptables 設定

対象：K8s control plane、全ての K8s node

1. br_netfilter をロードする。

```
$ lsmod | grep br_netfilter
（br_netfilter がロードされているかを確認。ロードされていない場合は以下コマンドを実行）
$ sudo modprobe br_netfilter
```

2. /etc/sysctl.d/k8s.conf を下記内容に修正する。

```
$ sudoedit /etc/sysctl.d/k8s.conf
---
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.conf.all.rp_filter=2
---
$ sudo sysctl --system
（設定されているか確認）
```

5.2 K8s のインストール

5.2.1 K8s 1.31.1 のインストール

対象：K8s control plane、全ての K8s node

1. K8s レポジトリを編集する。

```
$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg
--dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list
$ sudo apt update -y
```

2. 必須コンポーネントをインストールする

```
$ sudo apt-get install kubelet=1.31.1-1.1 kubeadm=1.31.1-1.1 kubectl=1.31.1-1.1
-y
```

3. kubelet の自動起動を設定して起動指示。

```
$ sudo systemctl enable kubelet
$ sudo systemctl start kubelet
$ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor
preset: enabled)
    Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: activating (auto-restart) (Result: exit-code) since Fri 2023-09-08
14:16:26 JST; 4s ago
（以下略）

（Active:activating になっていることを確認、現時点では running にはならない）
```

5.3 CRI-O のインストール

5.3.1 CRI-O v1.31.0 インストール

対象：K8s control plane、全ての K8s node

1. 必要なカーネルパラメータの設定を行う。

```
$ sudo modprobe overlay
$ sudo modprobe br_netfilter

（root 権限で実行する）
$ sudo -s
# cat <<EOF | sudo tee /etc/modules-load.d/kubernetes.conf
Overlay
br_netfilter
EOF

# cat > /etc/sysctl.d/99-kubernetes-cri.conf <<EOF
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

# sysctl --system
```

2. CRI-O バージョンを変数に設定する。※引き続き root 権限で実行する

```
# CRIO_VERSION=v1.31
```

3. Repository を取得する。※引き続き root 権限で実行する

```
# curl -fsSL https://pkgs.k8s.io/addons:/cri-
o:/stable:/$CRIO_VERSION/deb/Release.key | gpg --dearmor -o
/etc/apt/keyrings/cri-o-apt-keyring.gpg

# echo "deb [signed-by=/etc/apt/keyrings/cri-o-apt-keyring.gpg]
https://pkgs.k8s.io/addons:/cri-o:/stable:/$CRIO_VERSION/deb/ /" | tee
/etc/apt/sources.list.d/cri-o.list
```

4. CRI-O インストール

```
# apt-get update

# apt-get install cri-o=1.31.0-1.1

# exit
```

※CRI-O のインストールで 404 になる場合は、apt-get clean して apt-get update からやり直すこと。

※apt-get で認証の期限切れのエラーが発生した場合は apt install ca-certificates を実行してから再度実施。

5. CRI-O の設定

他ソフトが使用するサブネットアドレスとかぶらないように、必要であれば subnet を変更。

```
$ sudoedit /etc/cni/net.d/11-crio-ipv4-bridge.conflist

    "ranges": [
        [{ "subnet": "172.35.0.0/16" }],（サブネットアドレスを変更）
        [{ "subnet": "1100:200::/24" }]
    ]
```

calico を CNI として使用するため CRI-O の bridge は退避させる必要がある。（calico 起動後、/etc/cni/net.d 配下に 10-calico.conflist が作成され、元のファイルを上書きしてしまうため）

```
$ cd /etc/cni/net.d
$ sudo mkdir /etc/crio/net.d
$ sudo mv 11-crio-ipv4-bridge.conflist /etc/crio/net.d/
$ sudo rm -rf /etc/cni/net.d/
```

（プロキシ環境下で構築する場合は以下赤字の設定を行う(途中で改行せずに 1 行で記載する)）。

```
$ sudoedit /lib/systemd/system/crio.service

[Service]
Type=notify
EnvironmentFile=-/etc/default/crio
Environment="HTTP_PROXY=http://$(user):$(password)@(プロキシサーバ):(port)"
"HTTPS_PROXY=http://$(user):$(password)@ (プロキシサーバ):(port)"
"NO_PROXY=127.0.0.1,localhost,(ホストの IP),10.96.0.1"
（元からある Environment=GOTRACEBACK=crash は削除）
```

5.3.2 NVIDIA container-toolkit インストール（GPU あり）

対象：GPU を搭載している K8s node （想定環境では Sever #2 で実施）

本節は GPU を使用しない(GPU 搭載サーバを用いない)場合は本章の手順は実施不要である。

(1) NVIDIA container-toolkit（NVIDIA Cntainer Runtime が含まれる）をインストール

```
$ curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --
dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg ¥
  && curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-
container-toolkit.list | ¥
    sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-
toolkit-keyring.gpg] https://#g' | ¥
    sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
$ sudo apt-get update
$ NV_TOOLKIT_VERSION=1.16.2-1
$ sudo apt-get install nvidia-container-toolkit=${NV_TOOLKIT_VERSION} nvidia-
container-toolkit-base=${NV_TOOLKIT_VERSION}
```

(2) CRI-O/NVIDIA Cntainer Runtime 連携

① 連携用設定ファイルを生成して編集

連携用設定ファイルを生成

```
$ sudo nvidia-ctk runtime configure --runtime=crio --set-as-default --
config=/etc/crio/crio.conf.d/99-nvidia.conf
INFO[0000] Loading config: /etc/crio/crio.conf.d/99-nvidia.conf
INFO[0000] Config file does not exist; using empty config
INFO[0000] Successfully loaded config
INFO[0000] Wrote updated config to /etc/crio/crio.conf.d/99-nvidia.conf
INFO[0000] It is recommended that crio daemon be restarted.
```

/etc/crio/crio.conf.d/99-nvidia.conf を編集(末尾に赤字の行を追加)

```
$ sudoedit /etc/crio/crio.conf.d/99-nvidia.conf

[crio]

  [crio.runtime]
    default_runtime = "nvidia"

    [crio.runtime.runtimes]

      [crio.runtime.runtimes.nvidia]
        runtime_path = "/usr/bin/nvidia-container-runtime"
        runtime_type = "oci"
        monitor_path = "/usr/libexec/crio/conmon"
```

② NVIDIA Container Runtime 側の設定ファイル(config.toml)の編集

```
$ sudoedit /etc/nvidia-container-runtime/config.toml
・・・
（リストの中身を以下に編集する）
runtimes = ["/usr/libexec/crio/crun", "docker-runc", "runc", "crun"]

・・・
```

5.3.3 CRI-O の設定ファイルの編集（GPU なし）

対象：GPU を非搭載の K8s node （想定環境では Sever #1 で実施）

GPU が差さっていないサーバでは CRI-O の設定ファイルの編集は不要。

5.3.4 CRI-O を起動

対象： K8s control plane、全ての K8s node

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable crio

$ sudo systemctl start crio
$ sudo systemctl status crio
● crio.service - Container Runtime Interface for OCI (CRI-O)
     Loaded: loaded (/lib/systemd/system/crio.service; enabled; vendor preset:
enabled)
     Active: active (running) since Tue 2023-03-28 17:26:04 JST; 4s ago
       Docs: https://github.com/cri-o/cri-o
   Main PID: 46657 (crio)
      Tasks: 17
     Memory: 18.3M
     CGroup: /system.slice/crio.service
             mq46657 /usr/bin/crio
```

（crio.service の active(running)（起動）を確認）

5.4 K8s クラスター構築

5.4.1 calico の manifest のダウンロードと編集

対象：K8s control plane

1. calico v3.28.1 の manifest をダウンロードする

```
$ cd ~/
$ curl
https://raw.githubusercontent.com/projectcalico/calico/v3.28.1/manifests/calico.
yaml -O
```

2. 取得した calico.yaml の編集
   取得した calico.yaml を開き、”IP_AUTODETECTION_METHOD”の追記を行う

```
$ vi calico.yaml
```

   “autedetect”で検索をかけ、その直後に以下の様に赤字の内容を追記する

```
# Auto-detect the BGP IP address.
- name: IP
  value: “autodetect”
# Set Auto-Detection Method of Network Interface ※コメントなので追記しなくても良い
- name: IP_AUTODETECTION_METHOD
  value: kubernetes-internal-ip
#Enable IPIP
- name: CALICO_IPV4POOL_IPIP
```

5.4.2 K8s control plane で K8s クラスター構築

対象：K8s control plane

下記の kubeadm コマンドの青字は環境に合わせて適宜変更する。下記は本書での例。

--pod-network-cidr： 環境に合わせて適宜設定する

--apiserver-advertise-address：K8s control plane の IP アドレス

“Your Kubernetes control-plane has initialized successfully!”が出力されたら成功。

```
$ sudo swapoff -a

$ sudo kubeadm init --kubernetes-version 1.31.1 --pod-network-cidr=182.16.0.0/16
--apiserver-advertise-address=10.38.119.101 --cri-
socket=unix:///var/run/crio/crio.sock

(中略)
Your Kubernetes control-plane has initialized successfully!
※5.4.4 節の 2 で使用するため出力された以下コマンド（例）をコピーしておく
kubeadm join 10.38.119.22:6443 --token cark16.ammkfw7y16p1oieq ¥
        --discovery-token-ca-cert-hash
sha256:65e1490066504e60121266e9348ec5ee177c50778b654e559bccbd558668ddec

$ rm -rf $HOME/.kube
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf  ~/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

5.4.3 K8s control plane で calico の適用

対象：K8s control plane

```
$ kubectl apply -f calico.yaml
(中略)
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created

$ kubectl get pod -A -o wide
NAMESPACE     NAME                                       READY   STATUS    ...
kube-system   calico-kube-controllers-74677b4c5f-vs4wl   1/1     Running   ...
kube-system   calico-node-7vjdd                           1/1     Running   ...
kube-system   coredns-565d847f94-779p5                    1/1     Running   ...
kube-system   coredns-565d847f94-n9bxg                    1/1     Running   ...
kube-system   etcd-server0                                1/1     Running   ...
kube-system   kube-apiserver-server0                      1/1     Running   ...
kube-system   kube-controller-manager-server0             1/1     Running   ...
kube-system   kube-proxy-bqf8w                           1/1     Running   ...
kube-system   kube-scheduler-server0                      1/1     Running   ...
（しばらく待った後に上記のように Running になっていれば OK）
```

5.4.4 K8s node を K8s クラスターに参加

対象：全ての K8s node

1. K8s node を join させる準備
   青字は環境に合わせて適宜変更する。下記は本書での例。

```
$ sudo swapoff -a

$ rm -rf $HOME/.kube
$ mkdir -p $HOME/.kube
$ scp ubuntu@10.38.119.101:~/.kube/config ~/.kube/
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

2. 5.4.2 節実施時にコピーしたコマンドを使って join
   ※先頭に sudo、末尾に --cri-socket=unix:///var/run/crio/crio.sock オプションを追加

```
 $ sudo kubeadm join 10.38.119.22:6443 --token cark16.ammkfw7y16p1oieq ¥
        --discovery-token-ca-cert-hash
sha256:65e1490066504e60121266e9348ec5ee177c50778b654e559bccbd558668ddec --cri-
socket=unix:///var/run/crio/crio.sock
（上記は手順例、コピーしたコマンドを使うこと）

 $ sudo systemctl status kubelet
（kubelet の status が Active:active(running)なことを確認）
```

3. クラスター参加と pod の Running を確認

```
$ kubectl get node -o wide
NAME       STATUS   ROLES          AGE    VERSION   INTERNAL-IP    ...
server1    Ready    <none>         118s   v1.28.3   10.38.119.18   ...
server2    Ready    <none>         114s   v1.28.3   10.38.119.20   ...
server0    Ready    control-plane   9m7s   v1.28.3   10.38.119.101  ...

$ kubectl get pod -A -o wide
NAMESPACE     NAME                                       READY   STATUS    ...
kube-system   calico-kube-controllers-74677b4c5f-vs4wl   1/1     Running   ...
kube-system   calico-node-42lpk                          1/1     Running   ...
kube-system   calico-node-7vjdd                          1/1     Running   ...
kube-system   calico-node-h79jp                          1/1     Running   ...
kube-system   coredns-565d847f94-779p5                   1/1     Running   ...
kube-system   coredns-565d847f94-n9bxg                   1/1     Running   ...
kube-system   etcd-server0                               1/1     Running   ...
kube-system   kube-apiserver-server0                     1/1     Running   ...
kube-system   kube-controller-manager-server0            1/1     Running   ...
kube-system   kube-proxy-994ll                           1/1     Running   ...
kube-system   kube-proxy-bqf8w                           1/1     Running   ...
kube-system   kube-proxy-cgpkw                           1/1     Running   ...
kube-system   kube-scheduler-server0                     1/1     Running   ...

(しばらく待っても Running にならない場合、各 Node を再起動すると解決することがある。)
```

5.5 SR-IOV CNI プラグインセットアップ

別紙「OpenKasugai-Demo」で配備する DF の処理モジュールのうち、Pod 上で動作する処理モジュールは、Pod に追加した 2nd NIC から、K8s node の 100GNIC に作成した SR-IOV の VF を利用して Ethernet 通信を行うため、5.5～5.6 の手順で、それに必要な SR-IOV CNI プラグイン、Multus のセットアップおよびインストールを行う

対象：全ての K8s node

K8s の Pod が SR-IOV デバイスを利用した通信を行えるようにするため、SR-IOV CNI プラグインをセットアップする

5.5.1 Go 言語のインストール

下記コマンドで Go 言語をインストールする

```
$ cd ~/
$ wget https://go.dev/dl/go1.23.0.linux-amd64.tar.gz
$ tar xvfz ~/go1.23.0.linux-amd64.tar.gz
```

下記コマンドで gopath ディレクトリを作成する

```
$ mkdir ~/gopath
```

下記コマンドで gopath を設定する
※下記の設定は、~/.bashrc に記載する。

```
$ export GOPATH="$HOME/gopath"
$ export GOROOT="$HOME/go"
$ export PATH="$GOROOT/bin:$PATH"
$ source ~/.bashrc
```

5.5.2 SR-IOV CNI プラグインの入手

```
$ cd ~/
$ git clone https://github.com/k8snetworkplumbingwg/sriov-cni.git -b v2.8.1
```

5.5.3 SR-IOV CNI プラグインのビルド

```
$ cd sriov-cni/
$ make build
$ sudo cp build/sriov /opt/cni/bin
$ sudo chown root:root /opt/cni/bin/sriov
$ sudo chmod 755 /opt/cni/bin/sriov
```

もし、”make build”実行時に ”Command 'make' not found, …” で失敗した場合は dpkg-dev をインストールすること

```
$ sudo apt update
$ sudo apt install dpkg-dev
```

5.6 Multus のインストール

対象：K8s control plane

DF の各処理モジュールの Pod に 2nd NIC を作成するため、Multus をインストールする

5.6.1 Multus の入手

1. Multus の GitHub リポジトリを clone する

```
$ cd ~/
$ git clone https://github.com/k8snetworkplumbingwg/multus-cni.git -b v4.1.1
```

2. Multus の manifest を編集する

```
$ cd ~/multus-cni/deployments
$ vi multus-daemonset-thick.yml

---
resources:
   requests:
     cpu: "400m"（値を”400m”に設定）
     memory: "200Mi"（値を”200Mi”に設定）
   limits:
     cpu: "400m"（値を”400m”に設定）
     memory: "200Mi"（値を”200Mi”に設定）
```

※Multus の Pod で OOM killed が発生する場合は、適宜、上記の設定値を調整する。limits を requests の任意の倍数の値に設定することで、OOM killed が解消される場合がある。

5.6.2 Multus の manifest を適用（DaemonSet として配備）

```
$ cd ~/multus-cni/deployments
$ kubectl apply -f multus-daemonset-thick.yml
```

5.6.3 NetworkAttachmentDefinition の Manifest の作成および適用

1) NetworkAttachmentDefinition の Manifest の作成
この後の 8.9 節で、各 K8s node の 100GNIC に VF を作成する。
SR-IOV の VF を作成する NIC が存在する K8s node の台数分だけ(想定環境では Server #1 と Server #2 の 2 台分)、以下の内容で NetworkAttachmentDefinition の Manifest を作成する。

以下には、Server #1 向けの manifest の設定例を示す。

```
$ cd ~/
$ vi server1-config-net-sriov.yaml ※yaml のファイル名は任意
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: server1-config-net-sriov
  namespace: test01
  annotations:
    k8s.v1.cni.cncf.io/resourceName: nvidia.com/mlnx_sriov_device
spec:
  config: '{
  "type": "sriov",
  "cniVersion": "0.3.1",
  "name": "server1-net-sriov",
  "ipam": {
    "type": "static"
  }
}'
```

※metadata.name の「server1」部分に、対象の K8s node のホスト名を指定。
metadata.namespace に、別紙「OpenKasugai-Demo」で配備する DataFlow の namespace と同じ namespace を指定。
k8s.v1.cni.cncf.io/resourceName に、SR-IOV デバイスプラグインの ConfigMap 上の resourceName を指定。
spec.name の「server1」部分に、対象の K8s node のホスト名を指定。

2) 上記 1)で作成した NetworkAttachmentDefinition の Manifest の適用
以下には、Server #1 向けの manifest の適用例を示す。

```
$ cd ~/
$ kubectl create namespace test01
$ kubectl apply -f server1-config-net-sriov.yaml
```

※上記 1)で複数の K8s node ぶんの NetworkAttachmentDefinition を作成した場合は、全て適用する(想定環境では、Server #2 向けの manifest の適用も行う。)

6. GPU セットアップ

本章は GPU を使用する場合にのみ実行する手順となる。GPU を使用しない(GPU 搭載サーバを用いない)場合は本章の手順は実施不要である。

6.1 NVIDIA GPU ドライバのインストール

対象：GPU を搭載している K8s node （想定環境では Sever #2 で実施）

ソフトウェア全体構成における、本章の設定対象を図 6.1 に示す。
本章では GPU 搭載 K8s node に対して GPU ドライバ関連をインストールする。

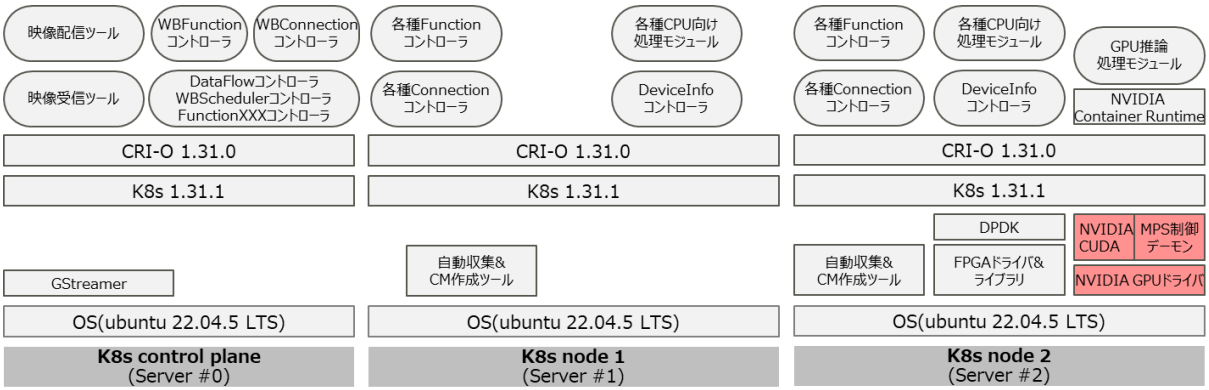


図 6.1 GPU ドライバインストール対象

1. run ファイルをダウンロードする
   https://www.nvidia.com/Download/Find.aspx?lang=en-us にアクセスし、図 6.2 の様に「Product Type」などを指定して、「Search」ボタンを押す。Search 結果の Version が「550.90.12」のリンクから run ファイルをダウンロードする。

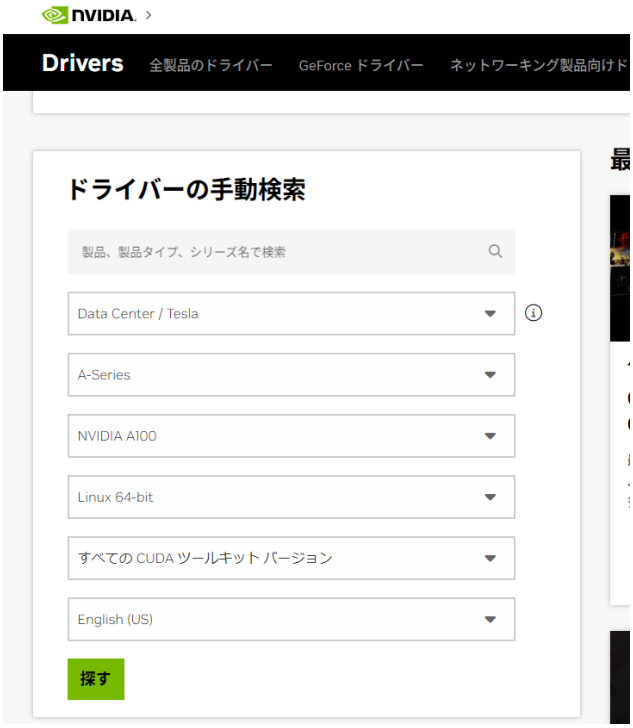


図 6.2 NVIDIA ドライバダウンロードサイト

2. NVIDIA ドライバファイル（NVIDIA-Linux-x86_64-550.90.12.run）をホームディレクトリで実行後、アップデートを確認。

```
$ cd ~/
$ sudo sh NVIDIA-Linux-x86_64-550.90.12.run -q --ui=none

$ nvidia-smi
（ドライババージョン 550.90.12 を確認）
```

NVIDIA GPU ドライバファイルの実行の際に以下の ERROR メッセージが出力された場合は、Nouveau kernel driver を無効化する必要がある。

```
ERROR: The Nouveau kernel driver is currently in use by your system.  This
driver is incompatible with the NVIDIA driver, and must be disabled before
proceeding.
```

上記の ERROR が出た場合は、以下のコマンドを実行して、Nouveau kernel driver を無効化してから、再度 NVIDIA GPU ドライバファイルを実行する。

```
$ sudoedit /etc/modprobe.d/blacklist-nouveau.conf
---
blacklist nouveau
options nouveau modeset=0
---
$ sudo update-initramfs -u
$ sudo reboot
```

3. OS の再起動を実施

```
$ sudo reboot
```

6.2 MPS 制御デーモンの起動

対象：GPU を搭載している K8s node

1. /root/.bashrc への設定追加
   以下の内容を/root/.bashrc に追記する（例）

```
export CUDA_DEVICE_ORDER="PCI_BUS_ID"
export CUDA_VISIBLE_DEVICES=0
export CUDA_MPS_PIPE_DIRECTORY=/tmp/nvidia-mps
export CUDA_MPS_LOG_DIRECTORY=/tmp/nvidia-mps
```

- CUDA_DEVICE_ORDER=”PCI_BUS_ID”について：
  CUDA_VISIBLE_DEVICES で指定する ID を nvidia-smi で確認できる ID とするために設定する。
- CUDA_VISIBLE_DEVICES について：
  対象 K8s node に搭載されている GPU デバイスのうち、MPS で使用するデバイスの ID を指定する。
  ”CUDA_DEVICES_ORDER=”PCI_BUS_ID”を指定することで、nvidia-smi で確認できる ID(0, 1, …)で指定可能。
  なお、対象 K8s node に搭載している全 GPU デバイスを使用する場合は省略可能

2. MPS 制御デーモンの起動・確認

```
$ sudo -s
# nvidia-cuda-mps-control -d
# exit

$ ps aux | grep mps
下記の様な制御デーモンが表示されることを確認
root  783951  0.0  0.0  76472   140 ? Ssl  10:00   0:00 nvidia-cuda-mps-control -d
```

3. MPS 制御デーモンの自動起動設定

reboot 時に毎回の MPS 実行が不要となるように、起動時に自動で MPS 制御デーモンが実行されるように設定する。まずは、下記の「MpsAutoStart.service」のファイルを作成する。

```
[Unit]
Description=Start MPS at boot

[Service]
ExecStart=nvidia-cuda-mps-control -d
Type=oneshot
RemainAfterExit=yes


[Install]
WantedBy=multi-user.target
```

続いて、root 権限で上記のファイルを/etc/systemd/system ディレクトリに移動し、サービスを有効化する。

```
$ sudo -s
# mv MpsAutoStart.service /etc/systemd/system
# systemctl enable MpsAutoStart.service
```

7. 各種処理モジュールのセットアップ

ソフトウェア全体構成における、本章の設定対象を図 7 に示す。
本章では評価に用いる各種処理モジュールのセットアップを行う。

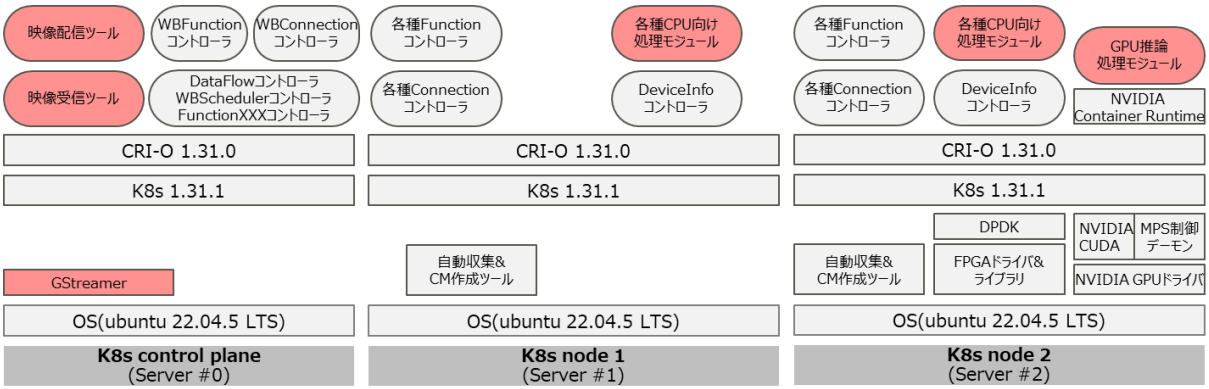


図 7 セットアップ対象処理モジュール

また本章で実施する各種処理モジュール／映像配信ツールのセットアップの流れを以下に示す。

	K8s control plane	K8s node
7.1 事前準備	○	
7.2 GPU 推論処理モジュールのセットアップ	—	○（GPU あり）
7.3 CPU デコード処理モジュールのセットアップ	—	○
7.4 CPU フィルタリサイズ処理モジュールのセットアップ	—	○
7.5 CPU コピー分岐処理モジュールのセットアップ	—	○
7.6 CPU Glue 処理モジュールのセットアップ	—	○
7.7 映像配信ツールのセットアップ	○	—
7.8 デモ用動画の準備	○	—

7.1 事前準備

対象：K8s control plane、全ての K8s node

7.1.1 CRI-O の設定変更・再起動

1. 下記コマンドで buildah をインストールする

```
$ sudo apt-get update
$ sudo apt-get -y install buildah
```

2. /usr/share/containers/containers.conf を修正する。

```
$ sudoedit /usr/share/containers/containers.conf

#[machine]（[machine]をコメントアウト）
```

3. /etc/containers/registries.conf を修正してコンテナリジストリの情報を追加し、CRI-O を再起動する。

```
$ sudoedit /etc/containers/registries.conf
---
unqualified-search-registries = ["docker.io", "quay.io"]
---

$ sudo systemctl restart cri-o
```

※"docker.io","quar.io"以外のコンテナレジストリを利用する場合は、上記の"docker.io", "quay.io"と同様に記載する。

7.1.2 資材(FPGA ライブラリ・ドライバ、コントローラ、サンプルアプリ群)の取得

ホームディレクトリにソース(controller)を格納する。

```
$ cd ~/
$ git clone --recursive https://github.com/openkasugai/controller.git -b v1.1.0
```

7.2 GPU 推論処理モジュールのセットアップ

本節は GPU を使用する場合にのみ実行する手順となる。GPU を使用しない(GPU 搭載サーバを用いない)場合は本章の手順は実施不要である。

MPS が有効だと GPU の処理モジュールのビルドに失敗する可能性があるため、一時的に無効にする。

```
$ sudo bash -c "echo quit | nvidia-cuda-mps-control"
```

7.2.1 GPU 推論処理モジュール(FPGA 対応版)のセットアップ

対象：GPU を搭載している K8s node

1. 以下の手順でコンテナをビルドする

```
$ cd ~/
$ cp ~/controller/sample-
functions/functions/gpu_infer_dma_plugins/fpgasrc/build_docker/gpu-
deepstream/Dockerfile .
$ sudo buildah bud --runtime=/usr/bin/nvidia-container-runtime -t
gpu_infer_dma:1.1.0 -f Dockerfile
```

※ビルドには非常に時間がかかるため注意

※本アプリが使用する予定の GPU（T4 か A100）が搭載されたサーバで実行すること

7.2.2 GPU 推論処理モジュール(TCP 対応版)のセットアップ

対象：GPU を搭載している K8s node

1. 以下の手順でコンテナをビルドする

```
$ cd ~/controller/sample-
functions/functions/gpu_infer_tcp_plugins/fpga_depayloader
$ chmod a+x build_app.sh
$ cd build_docker/gpu-deepstream
$ chmod a+x generate_engine_file.sh
$ chmod a+x find_gpu.sh
$ chmod a+x check_gpus.sh
$ sudo buildah bud --runtime=/usr/bin/nvidia-container-runtime -t
gpu_infer_tcp:1.1.0 -f Dockerfile ../../../../../
```

※本アプリには FPGA ライブラリは含まれない
※本アプリが使用する予定の GPU（T4 か A100）が搭載されたサーバで実行すること

上記の GPU 系の処理モジュール群がビルドされていることを確認する。

```
$ sudo buildah images | grep gpu_
REPOSITORY                  TAG         IMAGE ID      CREATED         SIZE
localhost/gpu_infer_tcp     1.1.0       63540ccdd609  22 minutes ago  22.2 GB
localhost/gpu_infer_dma     1.1.0       b385b25ccf27  22 minutes ago  23.5 GB
```

7.2 節の冒頭で無効化した MPS を有効に戻す。

```
$ sudo nvidia-cuda-mps-control -d

$ ps aux | grep mps
下記の様な制御デーモンが表示されることを確認
root  783951  0.0  0.0  76472   140 ? Ssl  10:00   0:00 nvidia-cuda-mps-control -d
```

7.3 CPU デコード処理モジュールのセットアップ

対象：全ての K8s node

1. 以下の手順でコンテナをビルドする

```
$ cd ~/controller/sample-functions/functions/cpu_decode/docker
$ sudo ./buildah_bud.sh 1.1.0
```

7.4 CPU フィルタリサイズ処理モジュールのセットアップ

対象：全ての K8s node

1. 以下の手順でコンテナをビルドする。

```
$ cd ~/controller/sample-functions/functions/cpu_filter_resize
$ sudo buildah bud -t cpu_filter_resize:1.1.0 -f containers/cpu/Dockerfile
```

7.5 CPU コピー分岐処理モジュールのセットアップ

対象：全ての K8s node

1. 以下の手順でコンテナをビルドする。

```
$ cd ~/
$ cp ~/controller/sample-functions/functions-
ext/cpu_copy_branch/build_docker/Dockerfile .
$ sudo buildah bud -t cpu_copy_branch:1.1.0 -f Dockerfile
```

7.6 CPU Glue 処理モジュールのセットアップ

対象：全ての K8s node

1. 以下の手順でコンテナをビルドする

```
$ cd ~/
$ cp ~/controller/sample-functions/functions-
ext/cpu_glue_dma_tcp/build_docker/Dockerfile .
$ sudo buildah bud -t cpu_glue_dma_tcp:1.1.0 -f Dockerfile
```

上記の CPU 系の処理モジュール群がビルドされていることを確認する。

```
$ sudo buildah images | grep cpu_
localhost/cpu_glue_dma_tcp      1.1.0      e675c9c54d2b    4 hours ago     1.61 GB
localhost/cpu_copy_branch       1.1.0      68b0b864ce36    5 hours ago     1.47 GB
localhost/cpu_filter_resize     1.1.0      1654f3935c8d    5 hours ago     885 MB
localhost/cpu_decode            1.1.0      5392a69e3376    5 hours ago     1.48 GB
```

7.7 映像配信ツール/映像受信ツールのセットアップ

対象：K8s control plane

映像配信ツールのビルドファイルのビルド指定をエディタで修正

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ sudo buildah bud -t send_video_tool:1.1.0 ./
```

映像受信ツールのビルドファイルのビルド指定をエディタで修正
※ビルドには非常に時間がかかるため注意

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ sudo buildah bud -t rcv_video_tool:1.1.0 ./
```

配信・受信ツールのイメージがビルドされていることを確認

```
$ sudo buildah images | grep _video_tool
localhost/rcv_video_tool        1.1.0      b8c8ada9cf5c    10 minutes ago  1.29 GB
localhost/send_video_tool       1.1.0      8257cf3e4389    1 minutes ago   1.29 GB
```

7.8 デモ用動画の準備

対象：K8s control plane

デモ用動画を 0.5 節の「11 デモ用動画」に記載した URL から取得して、/opt/DATA/video ディレクトリ配下に格納する。
/opt/DATA/video ディレクトリが無い場合は先に作成すること。

```
$ sudo mkdir -p /opt/DATA/video  ※/opt/DATA/video がない場合に実行
```

全ての動画を取得した場合、当該デイレクトリには以下の様に 2 本の動画が格納される。

```
$ ls -l /opt/DATA/video/
total 139528
-rw-r--r-- 1 ubuntu ubuntu   6319396 Dec  5 01:55 46098-447095422_small.mp4
-rw-r--r-- 1 ubuntu ubuntu 130259129 Dec  5 01:56 6896028-uhd_3840_2160_30fps.mp4
```

別紙「OpenKasugai-Demo」で実施する試験では、動画は 4K かつ 15fps 以下の必要があり、30～60 秒程度の長さが望ましい。
上記動画はいずれもその条件に当てはまらないため、編集を行う必要がある。
ここでは上記動画のうち「6896028-uhd_3840_2160_30fps.mp4」を上記条件に合わせる手順を示す。
対象の動画は 4K、30fps、46 秒程度のため、フレームレートを 15fps に変更すれば良い。ここでは動画の編集には ffmpeg を用いる。インストールされていない場合は先にインストールすること。

```
$ sudo apt install ffmpeg
$ ffmpeg -i /opt/DATA/video/6896028-uhd_3840_2160_30fps.mp4 -r 15
input_4K_15fps.mp4
```

※「input_4K_15fps.mp4」は編集後の映像のファイル名で、別紙「OpenKasugai-Demo」において入力映像として使用している。

なお、「46098-447095422_small.mp4」はフル HD かつ 25fps、30 秒程度の動画のため、こちらを用いたい場合は、解像度とフレームレートの変更が必要になる(手順は割愛する)。

8. コントローラのセットアップ

ソフトウェア全体構成における、本章の設定対象を図 8 に示す。
本章ではデータフロー配備を行う各種コントローラ（Custom Resource Controller(CRC)）のインストールと、コントローラに必要な設定を自動収集するツールの実行、および FPGA の準備を行う。

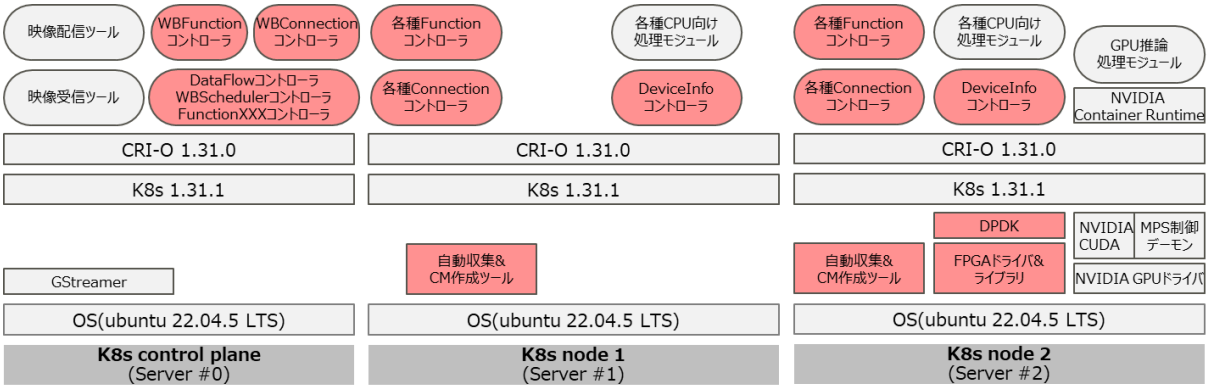


図 8 セットアップ対象コンポーネント

本章において K8s control plane／K8s node の必要な手順の流れを以下一覧に示す。

	K8s control plane	**K8s node**
8.1 事前準備	8.1.1 Go 言語のインストール	
		8.1.2 FPGA ライブラリのビルド
	—	8.1.3 FPGADB ライブラリのビルド
8.2 CRC コンテナのビルド	8.2.1 K8s control plane 側のビルド	8.2.2 K8s node 側のビルド
8.3 CRC の起動準備	—	○
8.4 自動収集&ConfigMap 作成実施に向けた準備	—	8.4.1 FPGA Bitstream 書き込み
	—	8.4.2 DCGM のインストール
8.5 各種情報(ConfigMap)の作成に向けた入力データの準備	8.5.1 入力データ(環境依存のデータ)の編集	
	—	8.5.2 入力データの集約
8.6 各種情報(外部情報、ConfigMap)の取り込み・配備	—	○
8.7 手動書込みツールのビルド	—	○
8.8 FPGA 内リソースの回収確認ツールのビルド	—	○
8.9 SR-IOV の VF 作成および管理	8.9.3 SR-IOV デバイスプラグインセットアップ 8.9.4 SR-IOV デバイスプラグインの実行	8.9.1 100GNIC への VF の作成 8.9.2 VF の作成

また、以下に各 CRC の役割、起動サーバの一覧を記載する。

CRC 名	役割	起動サーバ	
		K8s control plane	K8s node
FunctionType	登録された FunctionType のカスタムリソース(CR)を使用可能にする。	○	
FunctionChain	登録された FunctionChain CR を使用可能にする。		
FunctionTarget	ComputeResource CR を元に FunctionTarget CR を生成する。		
DataFlow	DataFlow CR を元に当該 CR を構成する WBFunction、WBConnection の CR を生成や削除を行う。		
WBscheduler	DataFlow CR の各 Function、Connection の配備先を決定し、その結果を当該 DataFlow CR に設定する。		
WBFunction	WBFunction CR を元に GPUFunction や FPGAFunction や CPUFunction の CR の生成や削除を行う。	○	
WBConnection	WBConnection CR を元に EthernetConnection や PCIeConnection の CR を生成や削除を行う。	○	
GPUFunction	GPUFunction CR を元に Gstreamer 用プラグインコンテナ(推論アプリ搭載)の起動や削除を行う。		○
FPGAFunction	FPGA への子 bs 書込みを行う。また FPGAFunction CR を元に FPGA 内リソースの割当てや回収を行う。		○
CPUFunction	CPUFunction CR を元に Gstreamer 用プラグインコンテナ(デコードアプリ搭載)の起動や削除を行う。		○
Ethernet Connection	EthernetConnection CR を元に映像配信処理用の Ethernet 経路を制御する。		○
PCIeConnection	PCIeConnection CR を元に映像配信処理の PCIe 経路を制御する。		○
DeviceInfo	システム構築時に初期状態の ComputeResource を生成する。また DataFlow の配備や削除に伴い ComputeResource を更新する。		○

8.1 事前準備

各 CRC のコンテナイメージ作成のため、Go 言語環境を設定してイメージ作成から登録方法を示す。

8.1.1 Go 言語のインストール

対象：K8s control plane、全ての K8s node

***5.5.1 節にて K8s node について実施済みの場合は、K8s control plane のみ実施**

下記コマンドで Go 言語をインストールする

```
$ cd ~/
$ wget https://go.dev/dl/go1.23.0.linux-amd64.tar.gz
$ tar xvfz ~/go1.23.0.linux-amd64.tar.gz
```

下記コマンドで gopath ディレクトリを作成する

```
$ mkdir ~/gopath
```

下記コマンドで gopath を設定する
※下記の設定は、~/.bashrc に記載する。

```
$ vi ~/.bashrc
以下を追加する
export GOPATH="$HOME/gopath"
export GOROOT="$HOME/go"
export PATH="$GOROOT/bin:$PATH"

$ source ~/.bashrc
```

8.1.2 FPGA ライブラリのビルド

対象：全ての K8s node

1. ソースを取得(FPGA ライブラリ)する
   未取得の場合、7.1.2 節を実施し、ソースを取得する。

2. 必要なパッケージをインストールする（インストール済みであればスキップ）

```
$ sudo apt-get update
$ sudo apt-get install build-essential python3-pip pkg-config libnuma-dev
zlib1g-dev libpciaccess-dev
$ sudo pip3 install meson ninja pyelftools
```

3. DPDK のビルド
   下記コマンドを入力し、ビルドを行う。

```
$ cd ~/controller/src/submodules/fpga-software/lib/
$ make dpdk
```

4. MCAP ツールのビルド及びインストール
   下記コマンドを入力し、ビルドを行い、sudo の環境変数 PATH が通っているディレクトリに mcap ツールを格納する。(※以下の例では printenv で確認した後に/usr/local/bin を選択している)

```
$ cd ~/controller/src/submodules/fpga-software/lib/
$ make mcap
$ sudo printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
$ sudo cp MCAP/mcap /usr/local/bin
```

5. JSON ソースファイルを取得する
   下記コマンドを入力し、JSON ソースファイルを取得する。

```
$ cd ~/controller/src/submodules/fpga-software/lib/
$ make json
```

6. FPGA ライブラリのビルド
   下記コマンドを入力し、FPGA ライブラリのビルドを行う。

```
$ cd ~/controller/src/submodules/fpga-software/lib/
$ make
```

7. FPGA ライブラリのヘッダファイルを所定のディレクトリにコピーする
/usr/local/include/fpgalib に下記のディレクトリをディレクトリごとコピーする。
(/usr/local/include/fpgalib ディレクトリがない場合は mkdir で作成する)

```
$ cd ~/controller/src/submodules/fpga-software/lib/build
$ sudo cp -RT include/libfpga /usr/local/include/fpgalib
```

8. FPGA ライブラリのライブラリ本体を所定のディレクトリにコピーする
/usr/local/lib/fpgalib に下記をコピーする。(/usr/local/lib/fpgalib ディレクトリがない場合は mkdir で作成する)

```
$ cd ~/controller/src/submodules/fpga-software/lib/build
$ sudo cp libfpga.a /usr/local/lib/fpgalib/
```

9. fpga-software/lib/DPDK/dpdk を /usr/local/lib/fpgalib/dpdk にリンク

```
$ sudo ln -s ~/controller/src/submodules/fpga-software/lib/DPDK/dpdk
/usr/local/lib/fpgalib/dpdk
```

8.1.3 FPGADB ライブラリのビルド

対象：全ての K8s node

1. FPGADB ライブラリのビルド
下記コマンドを入力し、FPGADB ライブラリのビルドを行う。
(予め 8.1.2 FPGA ライブラリのビルドを実施すること)

```
$ cd ~/controller/src/fpgadb
$ make
```

2. FPGADB ライブラリのヘッダファイルを所定のディレクトリにコピーする
/usr/local/include/fpgalib に下記のディレクトリ（fpgadb/build 配下の FPGADB ライブラリ使用時に必要なヘッダファイルの入ったディレクトリ）をディレクトリごとコピーする。

```
$ cd ~/controller/src/fpgadb/build
$ sudo cp -RT include /usr/local/include/fpgalib
```

3. FPGADB ライブラリのライブラリ本体を所定のディレクトリにコピーする。
/usr/local/lib/fpgalib に下記（fpgadb/build 配下の libfpgadb.a）をコピーする。

```
$ cd ~/controller/src/fpgadb/build
$ sudo cp libfpgadb.a /usr/local/lib/fpgalib/
```

8.2 CRC コンテナのビルド

● **前提事項**

この後のコントローラのコンテナイメージ作成で以下の選択肢が出た場合は、docker.io/library/golang:1.23 を選択する。

```
[1/2] STEP 1/13: FROM golang:1.23 AS builder
? Please select an image:
  ▸ docker.io/library/golang:1.23
    quay.io/golang:1.23
```

8.2.1 K8s control plane 側のビルド

対象：K8s control plane

(1) /etc/containers/registries.conf.d/shortnames.conf に短縮名”golang”の設定を追加する。

```
$ sudoedit /etc/containers/registries.conf.d/shortnames.conf
---
# golang
"golang" = "docker.io/library/golang"
```

(2) DataFlow コントローラ、WBscheduler コントローラ、FunctionTarget コントローラ、FunctionType コントローラ、FunctionChain コントローラのビルドを実施する。

```
$ cd ~/controller/src/whitebox-k8s-flowctrl
$ make docker-build IMG=localhost/whitebox-k8s-flowctrl:1.1.0
```

もし、”make dokcer-build ...”実行時に ”Command 'make' not found, …” で失敗した場合は以下の

```
$ sudo apt update
$ sudo apt install dpkg-dev
```

様に dpkg-dev をインストールした後に改めて”make dovker-build ...”を実行すること。

(3) WBFunction コントローラのビルドを実施する。

```
$ cd ~/controller/src/WBFunction
$ make docker-build IMG=localhost/wbfunction:1.1.0
```

(4) WBConnection コントローラのビルドを実施する。

```
$ cd ~/controller/src/WBConnection
$ make docker-build IMG=localhost/wbconnection:1.1.0
```

8.2.2 K8s node 側のビルド

対象：全ての K8s node

(1) /etc/containers/registries.conf.d/shortnames.conf に短縮名”golang”の設定を追加する。

```
$ sudoedit /etc/containers/registries.conf.d/shortnames.conf
---
# golang
"golang" = "docker.io/library/golang"
```

(2) DeviceInfo コントローラのビルド

```
$ cd ~/controller/src/DeviceInfo
$ sudo buildah bud -t deviceinfo:1.1.0 -f ./Dockerfile ..
```

(3) PCIeConnection コントローラのビルド

```
$ cd ~/controller/src/PCIeConnection
$ cp -pr ../submodules/fpga-software openkasugai-hardware-drivers
$ sudo buildah bud¥
 --build-arg=PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/workspace/openkasugai-hardware-
drivers/lib/DPDK/dpdk/lib/x86_64-linux-gnu/pkgconfig:/workspace/openkasugai-
hardware-drivers/lib/build/pkgconfig¥
 -t pcieconnection:1.1.0 -f ./Dockerfile ..
```

(4) EthernetConnection コントローラのビルド

```
$ cd ~/controller/src/EthernetConnection
$ cp -pr ../submodules/fpga-software openkasugai-hardware-drivers
$ sudo buildah bud¥
 --build-arg=PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/workspace/openkasugai-hardware-
drivers/lib/DPDK/dpdk/lib/x86_64-linux-gnu/pkgconfig:/workspace/openkasugai-
hardware-drivers/lib/build/pkgconfig -t ethernetconnection:1.1.0 -
f ./Dockerfile ..
```

(5) FPGAFunction コントローラのビルド

```
$ cd ~/controller/src/FPGAFunction
$ cp -pr ../submodules/fpga-software openkasugai-hardware-drivers
$ cp -pr ../fpgadb .
$ cp -p $HOME/hardware-design/example-design/bitstream/OpenKasugai-fpga-example-
  design-1.0.0-2.bit .
$ sudo buildah bud¥
 --build-arg=PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/workspace/openkasugai-hardware-
drivers/lib/DPDK/dpdk/lib/x86_64-linux-gnu/pkgconfig:/workspace/openkasugai-
hardware-drivers/lib/build/pkgconfig -t fpgafunction:1.1.0 -f ./Dockerfile ..
```

もし、”OpenKasugai-fpga-example-design-1.0.0-2.bit”が無くてコピーできなかった場合は、hardware-design リポジトリを git clone で取得した後に改めてコピーを行う。

```
$ cd ~
$ git clone https://github.com/openkasugai/hardware-design.git
```

(6) GPUFunction コントローラのビルド

```
$ cd ~/controller/src/GPUFunction
$ sudo buildah bud -t gpufunction:1.1.0 -f ./Dockerfile ..
```

(7) CPUFunction コントローラのビルド

```
$ cd ~/controller/src/CPUFunction
$ sudo buildah bud -t cpufunction:1.1.0 -f ./Dockerfile ..
```

8.3 CRC の起動準備

対象：全ての K8s node

1. CRC の動作に必要な設定を行う。
   複数 K8s node 構成の場合、各 K8s node で実施すること。

```
$ sudo mkdir -p /etc/k8s_node
$ sudo cp -p $HOME/.kube/config /etc/k8s_node/.
$ sudo chmod 666 /etc/k8s_node/config

$ export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:$HOME/controller/src/submodules/fpga-
software/lib/DPDK/dpdk/lib/x86_64-linux-gnu/pkgconfig
$ export
PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:$HOME/controller/src/submodules/fpga-
software/lib/build/pkgconfig
$ export
LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/local/lib/fpgalib/dpdk/lib/x86_64-linux-
gnu
$ export CGO_CFLAGS_ALLOW=-mrtm
```

   以降の手順は(各 K8s node で実施する必要は無く)どこかの K8s node で 1 度だけ実行すれば良い。

2. DeviceInfo の CRD 登録

```
$ cd ~/controller/src/DeviceInfo
$ make install
```

3. PCIeConnection の CRD 登録

```
$ cd ~/controller/src/PCIeConnection
$ make install
```

4. EthernetConnection の CRD 登録

```
$ cd ~/controller/src/EthernetConnection
$ make install
```

5. FPGAFunction、FPGA、ChildBs、FPGAReconfiguration の CRD 登録

```
$ cd ~/controller/src/FPGAFunction
$ make install
```

6. GPUFunction の CRD 登録

```
$ cd ~/controller/src/GPUFunction
$ make install
```

7. CPUFunction の CRD 登録。

```
$ cd ~/controller/src/CPUFunction
$ make install
```

8.4 自動収集&ConfigMap 作成実施に向けた準備

8.4.1 FPGA Bitstream 書き込み

作業対象：FPGA を搭載した全ての K8s node
本章は FPGA を使用する場合にのみ実行する手順となる。FPGA を使用しない(FPGA 搭載サーバを用いない)場合は本章の手順は実施不要である。

自動収集&CM 作成機能で FPGA の情報を取得するために事前に Bitstream(子 bs)を書込む必要がある。

(1) 子 bs を書き込む

4.2.1 節で使用した MCS ファイルに対応した BIT ファイルが必要であるため、予め 4.2.1 節を実施すること。本節では、hardware-design 配下のビルド済み BIT ファイルを使用する想定で記載する。なお、誤って書き込み済の MCS ファイルに対応していない BIT ファイルを書き込んでしまった場合も再度 4.2.1 節を実施すること。
8.1.2 節で mcap をビルド、sudo のパスに格納していなければ、予め 8.1.2 節を実施すること。

本手順では、mcap ツールにて書込みを行うが、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 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
1f:00.0 Memory controller: Xilinx Corporation Device 903f
...
```

mcap コマンドを使用して子 bs を書き込む。書き込みには 15 秒程度がかかる。
※この手順は、システム起動後および再起動時は毎回実施する必要がある。

```
$ cd ~/hardware-design/example-design/bitstream/
$ sudo mcap -x 903f -s xx:xx.x -p OpenKasugai-fpga-example-design-1.0.0-2.bit
```

書込みに成功した場合、以下のようなログが表示される。

```
Xilinx MCAP device found (xx:xx.x)
FPGA Configuration Done!!
```

(2)　FPGA ドライバのインストール

github の hardware-drivers に含まれる FPGA ドライバをロードする。
本節は(1)を実施後に実施する必要がある。
※OS を再起動した際も、(1)から再度実施する必要がある。

既に xpcie ドライバがロード済である場合、アンロードする。下記の lsmod の出力が無ければロードされていない。

```
$ lsmod | grep xpcie
xpcie                110592  0
$ sudo rmmod xpcie
$ lsmod | grep xpcie
```

ドライバをビルドし、ロードする。

```
$ cd ~/controller/src/submodules/fpga-software/driver
$ make
$ sudo insmod xpcie.ko
$ ls /dev/xpcie*

~ ls /dev/xpcie*を実行後のイメージ~
/dev/xpcie_[FPGA の UUID]
```

8.4.2 DCGM のインストール

作業対象：GPU を搭載した全ての K8s node
本章は GPU を使用する場合にのみ実行する手順となる。GPU を使用しない(GPU 搭載サーバを用いない)場合は本章の手順は実施不要である。

自動収集&CM 作成機能では、GPU 情報を取得するために NVIDIA DCGM※1 を活用している。そのため、DCGM のインストールを行う必要がある。

※1. https://docs.nvidia.com/datacenter/dcgm/3.1/index.html

```
$ cd ~/
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID | sed -e 's/¥.//g')
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86_64/cu
da-keyring_1.1-1_all.deb
$ sudo dpkg -i cuda-keyring_1.1-1_all.deb
$ sudo apt-get update && sudo apt-get install -y datacenter-gpu-manager
```

8.5 各種情報(ConfigMap)の作成に向けた入力データの準備

作業対象：全ての K8s node、K8s control plane

CRC を動かすためには、これらが使用する各種情報の ConfigMap を作成する必要がある。ConfigMap を作成するためには、インフラに関する情報を入力データとして用意する必要がある。本節では、ConfigMap 作成に必要な入力データの準備の手順について述べる。

入力データは以下の A, B に分類される。

A. 全ての DF で共通的に使えるデータ：

サンプルのユースケースであれば、基本的に提供した資材をそのまま使用することが可能なファイル。対象ファイルは以下になる。

ファイル名	説明	備考
devicetypemap.json	デバイス名(モデル)を DeviceType に変換するためのマップ情報	※1
region-unique-info.json	領域固有情報	※1
functionkindmap.json	デバイス種別に合う CR を判定する際に使用	
connectionkindmap.json	接続種別に合う CR を判定する際に使用	
function-unique-info.json	Function の詳細情報	
filter-resize-childbs.json	FPGA の Filter/Resize に関する内部の CH 情報	
functionnamemap.json	FPGA の Filter/Resize の高度/軽量推論判別用	
premadefilelist.json	自動収集&CM 作成機能が必要とするファイルと内部で保持する構造体を紐づける	
gpufunc-config-high-infer.json	高度推論 GPUFunction 用 Config 情報	
gpufunc-config-low-infer.json	軽量推論 GPUFunction 用 Config 情報	
fpgafunc-config-filter-resize-high-infer.json	高度推論フィルタリサイズ FPGAFunction 用 Config 情報	
fpgafunc-config-filter-resize-low-infer.json	軽量推論フィルタリサイズ FPGAFunction 用 Config 情報	
cpufunc-config-decode.json	デコード CPUFunction 用 Config 情報	
cpufunc-config-filter-resize-high-infer.json	高度推論フィルタリサイズ CPUFunction 用 Config 情報	
cpufunc-config-filter-resize-low-infer.json	軽量推論フィルタリサイズ CPUFunction 用 Config 情報	
cpufunc-config-copy-branch.json	コピー分岐 CPUFunction 用 Config 情報	
cpufunc-config-glue-fdma-to-tcp.json	GlueCPUFunction 用 Config 情報	

サンプルデータの格納場所は「~/controller/test/sample-data/sample-data-common/」になる。

例外として、想定環境で使用している 4 種類のデバイス(Alveo U250, NVIDIA GPU T4, NVIDIA GPU A100, Intel(R) Xeon(R) Gold 6346 CPU @ 3.10GHz , Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz)以外のデバイスを使用する場合は、上記表の備考欄に「※1」のついたファイルに追記が必要になる。追

記方法については別紙「OpenKasugai-Controller-InstallManual_Attachment1」の「3.CM 作成に使用する入力データ(JSON)の説明」の当該ファイルの記載部分を参照。

B. DF に依存したデータ：個々の DF の内容や DF の配備先などに合わせて編集が必要なファイル。
対象ファイル、ディレクトリは以下になる。

ファイル名, ディレクトリ名	説明	備考
~/controller/test/ sample-data/sample-data-demo/json/ predetermined-region.json	システム内の各領域の RegionType を定義したファイル。	※2

※2. 各システムの領域種別を事前に決め打ちする「Lane 固定方式」において、各領域がどの領域種別なのかを定義しておくファイル。

サンプルデータからの変更が必要な箇所や変更方法(どの値を記載すれば良いか)については、別紙「OpenKasugai-Controller-InstallManual_Attachment1」の「3.CM 作成に使用する入力データ(JSON)の説明」の当該ファイルの記載部分を参照。（変更方法の方針は備考欄に記載あり）

8.5.1　入力データ(環境依存のデータ)の編集

作業対象：全ての K8s node

上記 B に相当するサンプルデータである predetermined-region.json を実施環境に合わせて編集する。ベースとなるファイルは以下のファイルになる。
~/controller/test/sample-data/sample-data-demo/json/predetermined-region.json

以下に該当ファイルに設定する内容を記載する。なお、該当ファイルの記載内容と各パラメータに定義する値については、別紙「OpenKasugai-Controller-InstallManual_Attachment1」の「3.CM 作成に使用する入力データ(JSON)の説明」シート内の「B.提供下資材から編集が必要なファイル」にも記載しているので、適宜参照すると良い。

predetermined-region.json には全ての K8s node に搭載されたデバイス(FPGA、GPU、CPU)について、デバイス上に作られる各領域に関する情報を記載する。0.4 節に記載した FPGA 回路を用いる場合各 FPGA 上の領域は 2 つとなるので FPGA デバイス毎に 2 つ分の領域を記載することになる。また、GPU/CPU デバイス上の領域は 1 つと想定しているため CPU/GPU 毎に 1 つ分の領域を記載することになる。

各領域に記載する値は以下になる。

- nodeName：当該領域があるノードのノード名。
  - デバイスを搭載している K8s node のノード名を記載する
- deviceUUID：当該領域があるデバイスの識別情報。
  - FPGA の場合：”ls /dev/*”コマンドの結果を利用。
    8.4.1 節で FPGA ドライバをインストールすると FPGA デバイスは
    ”xpcie_${FPGA-ID}"と表示されるので、${FPGA-ID}の値を記載する。
  - GPU の場合："nvidia-smi -L"コマンドの結果を利用。
    デバイス毎に UUID(例：GPU-b8b4f1f5-bf51-eaa3-6ec4-97190b7f6c98)が出力されるのでその値を記載する
  - CPU の場合："ノード名”+”-cpu0"を記載する。
    (現状各サーバで仮想的に 1 つとみなしているため、ノード名以降は固定値で良い)
- subDeviceSpecRef：当該領域の領域名
  - FPGA の場合："${lane 番号}"を記載する
  - CPU/GPU の場合：deviceType と同じ値を記載する
- regionType：当該領域の領域種別
  - FPGA の場合：以下のフォーマットで記載する。
    「"${デバイス種別}" + "-" + "${親 bs-id}" + "-" + "${lane 数}" + "lanes" + "-" + "${NIC 数}" + "nics"」
    ※0.4 節に記載した FPGA 回路を用いる場合は” alveou250-0100001c-2lanes-0nics”となる。
  - CPU/GPU の場合：deviceType と同じ値を記載する

8.5.2　入力データの集約

作業対象：全ての K8s node

8.6 節で実行する自動収集&CM 作成ツールは、各 K8s node で、上記 A, B の全てのデータが特定のディレクトリ（~/controller/src/tools/InfoCollector/infrainfo/）にまとまっている前提で処理を行うため、入力データを~/controller/src/tools/InfoCollector/infrainfo 配下にまとめる。
以下では想定環境でのデータ作成の例を示す。

```
#上記 A(全ての DF で共通的に使えるデータ)の集約
$ cp -r ~/controller/test/sample-data/sample-data-common/json/*
  ~/controller/src/tools/InfoCollector/infrainfo/.

#上記 B(DF に依存したデータ)の集約
$ cp -r ~/controller/test/sample-data/sample-data-demo/json/*
  ~/controller/src/tools/InfoCollector/infrainfo/.
```

8.6 各種情報(外部情報、ConfigMap)の取り込み・配備

作業対象：全ての K8s node
CRC が使用する各種情報の ConfigMap を配備する。

1. 古い ConfigMap の削除
   ※複数 K8s node 構成の場合は、(各 K8s node で実施する必要は無く)どこかの K8s node で 1 度だけ実行すること。

```
$ cd ~/controller/src/tools/InfoCollector/infrainfo/
$ ./k8s-config.sh delete ※1
$ kubectl get cm
NAME                                    DATA   AGE
connectionkindmap                       1      33d
functionkindmap                         1      33d
…
kube-root-ca.crt                        1      63d

※kube-root-ca.crt 以外の ConfigMap があれば以下で全て削除する
$ kubectl delete cm <kube-root-ca.crt 以外の ConfigMap 名>
```

   ※1. 既存の ConfigMap の削除を行っているが、OS インストールから(本書の 3 章から)構築している場合はこの時点では ConfigMap は kube-root-ca.crt 以外配備されていないので、全て以下のエラーが出力されるが無視して良い。
   “Error from server (NotFound): configmaps "*削除対象の ConfigMap 名*" not found

2. ConfigMap の配備
   ※複数 K8s node 構成の場合、(各 K8s node で実施する必要は無く)どこかの K8s node で 1 度だけ実行すること。

```
$ cd ~/controller/src/tools/InfoCollector/infrainfo/
$ ./k8s-config.sh create ※
```

   ※既に”test01”という Namespace が作成済みの場合、”./k8s-config.sh create”実行時に以下のエラーが出力されるが無視して良い。
   “Error from server (AlreadyExists): namespaces "test01" already exists”

3. 自動収集&CM 作成ツールの起動
   ※複数 K8s node 構成の場合は各 K8s node で実施すること

```
$ 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
$ make all ※
```

   ※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"}”

8.7 手動書込みツール(FPGAReconfigurationTool)のビルド

対象：FPGA カードを搭載している全ての K8s node

人が任意のタイミングで FPGA に子 bs を書込むための手動書込みツールのビルドを行う。本ツールを用いることで、以下の 3 種類の書込みが可能になる。

●子 bs の書込み：子 bs 未書込み状態(8.4.1 節実施直後と同じ状態)の FPGA に子 bs を書込む

●子 bs のリセット：引数で指定した FPGA で使用中の子 bs を書込み直後の状態に戻す。

●FPGA のリセット：引数で指定した FPGA を子 bs 未書込み状態(8.4.1 節実施直後と同じ状態)に戻す。

手動書込みツールのビルドを行うには以下を実施する。

```
$ cd ~/controller/src/tools/FPGAReconfigurationTool/
$ make build
```

8.8 FPGA 内リソースの回収確認ツール(FPGAClearCheckTool)のビルド

対象：FPGA カードを搭載している全ての K8s node

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子 bs ならフィルタリサイズ回路内のリソース)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。）

ここでは、別紙「OpenKasugai-Demo」にて DataFlow を削除した際に、当該 DataFlow が使用していた FPGA 内の各種リソースが本当に回収されたのかどうかを確認する際に使用するツール(FPGAClearCheckTool)のビルドを行う。
FPGAClearCheckTool は以下の 2 種類のツールから構成される。

a) fpga-chk-connection：FPGA ライブラリを呼出して、引数で指定された FPGA 内リソースの状態を確認する

b) FPGACheckPerDF：削除された DataFlow とその DataFlow が使用していた FPGA 内リソースを識別し上記 fpga-chk-connection を実行することで、DF 単位で FPGA 内リソースの回収状態を確認する

ビルド手順は以下になる。

1. fpga-chk-connection のビルド：以下のコマンドを入力し、fpga-chk-connection ツールのビルドを行う。(予め 8.1.2 FPGA ライブラリのビルドを実施すること)

```
$ cd ~/controller/src/tools/FPGAClearCheckTool/fpga-chk-connection/
$ make
```

2. FPGACheckPerDF のビルド

```
$ cd ~/controller/src/tools/FPGAClearCheckTool/FPGACheckPerDF
$ make build
```

8.9 SR-IOV の VF 作成および管理

100GNIC に作成する SR-IOV の VF を利用して DataFlow の各 Pod 間の Ethernet 通信を行うため、全 K8s node の 100GNIC に対して、VF の作成を行う。また、VF の作成後に、K8s control plane において、SR-IOV デバイスプラグインを入手して daemoset として実行することで、作成した VF を K8s node 上で利用可能なリソースとして認識させる。
※OS を再起動した場合は、8.9.2 節及び 8.9.4 節を再実施する。

8.9.1 100GNIC への VF の作成

作業対象：全ての K8s node

対象サーバの 100G NIC のベンダによって設定内容が異なる。ここでは Intel 100GNIC の場合の設定と Mellanox 100G NIC の場合の設定内容を示す。

- Mellanox 100G NIC の場合（想定環境の K8s node はこちら）

1) NVIDIA ファームウェアツール（MFT：Mellanox Firmware Tools）のページ（https://network.nvidia.com/products/adapter-software/firmware-tools/）から MFT のバイナリ（mft-4.30.0-139-x86_64-deb.tgz）をダウンロードし、ホームディレクトリ配下に格納する。

```
$ cd ~
$ tar xzvf mft-4.30.0-139-x86_64-deb.tgz
$ cd ~/mft-4.30.0-139-x86_64-deb
$ chmod +x install.sh
$ sudo ./install.sh
```

install.sh の実行時に以下のエラーが出力されたら出力に従って dkms をインストールした後、再度 install.sh を実行する。

“-E- There are missing packages that are required for installation of MFT.
-I- You can install missing packages using: apt-get install dkms”

```
$ sudo apt-get install dkms
```

2) NVIDIA の Linux Drivers のページ（https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/）から MLNX_OFED のバイナリ（MLNX_OFED_LINUX-24.10-1.1.4.0-ubuntu22.04-x86_64.tgz）をダウンロードし、ホームディレクトリ配下に格納する。
一連の操作の中で NIC の PCI アドレス(以下の赤字部分)を指定するので事前に把握しておく。

```
$ lspci | grep Mellanox
ae:00.0 Ethernet controller: Mellanox Technologies MT27800 Family [ConnectX-5]
$ cd ~
$ tar xzvf MLNX_OFED_LINUX-24.10-1.1.4.0-ubuntu22.04-x86_64.tgz
$ cd ~/MLNX_OFED_LINUX-24.10-1.1.4.0-ubuntu22.04-x86_64
$ sudo apt update
$ sudo apt install gfortran automake flex libgfortran5 libltdl-dev autoconf
autotools-dev libnl-route-3-200 quilt libnl-3-dev chrpath libnl-route-3-dev
graphviz swig bison m4 libfuse2 debhelper mstflint
$ sudo ./mlnxofedinstall (a)
$ sudo -s
# mst start
# mst status # 結果が以下と異なる場合は(b)を参照
MST modules:
------------
    MST PCI module loaded
    MST PCI configuration module is not loaded
～略～
# mstconfig -d ae:00.0 set SRIOV_EN=1 NUM_OF_VFS=8
# reboot
```

・上記(a)で以下の様なエラーが発生した場合、

```
Removing old packages...

Error: One or more packages depends on MLNX_OFED_LINUX.
Those packages should be removed before uninstalling MLNX_OFED_LINUX:

mft-autocomplete
```

表示通り、mft-autocomplete を削除して(a)をやり直せば良い。

・上記(b)の出力結果でが以下の場合、

```
# mst status
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module is not loaded

PCI Devices:
～略～
```

以下の様に追加でコマンドを実施する必要がある。

```
# modprobe mst_pci  #追加コマンド
# mst status
MST modules:
------------
    MST PCI module loaded
    MST PCI configuration module is not loaded
～略～
```

なお、reboot 後に# mst status を実行すると”MST PCI module is not loaded”となる場合があるが特に対処は不要であり、次の手順に進める。

3) K8s node が再起動されたので、8.4.1 節を再実施する。

● 対象 NIC が Intel 100GNIC の場合
手順なし。8.9.2 節に進む。

8.9.2 VF の作成

作業対象：全ての K8s node

※OS を再起動した場合は、本節並びに 8.9.4 節の SR-IOV デバイスプラグインの実行を再実施する

作成する VF 数は、配備する DataFlow が使用する VF 数を満たすだけの値を設定する。基本的には同時配備し得る DataFlow が使用する VF 数の総和以上の値を設定すれば良い。具体的には、VF 使用数が最大となる DataFlow での使用 VF 数と同時配備可能な最大 DataFlow 数の積で良い。
各 DataFlow で使用する VF の数は Ethernet 接続を行う Pod の数に相当する。例えば別紙「OpenKasugai-Demo」の 1 章で配備する「A)FPGA F/R(フィルタ・リサイズ)を用いたフロー(図 1)」の場合、CPUFunction 用のデコード Pod と GPUFunction 用の推論 Pod が Ethernet 接続を行う Pod のため、使用する VF 数は 2 となる(デコード Pod は映像配信ツールとの間、推論 Pod は映像受信ツールとの間が Ethernet 接続になる)。
なお、以下の想定により作成する VF 数を 40 としている。

・同時配備可能な DataFlow の最大数は 8 本
・使用する VF 数が最大となる DataFlow は別紙「OpenKasugai-Demo」の 5 章で配備する「コピー分岐フロー(図 16)」で、5 つの VF を使用
　・CPUFunction 用のデコード Pod、フィルタリサイズ Pod、コピー分岐 Pod、GPUFunction 用の 2 つの推論 Pod がそれぞれ VF を使用するため)
・使用する VF 数が最大となるのは「C)コピー分岐フロー」を同時に 8 本配備する場合

● Mellanox 100G NIC の場合（想定環境の K8s node はこちら）

1) 対象の 100GNIC に VF を作成する
以下コマンドの ens8np0 は対象の 100GNIC のインターフェース名に合わせて変更すること

```
$ sudo -s
# ibdev2netdev
# echo 40 > /sys/class/net/ens8np0/device/sriov_numvfs
# ibdev2netdev -v
# exit
```

2) 対象の 100GNIC に、上記 2 で指定した数の VF が作成されたことを確認する

```
$ ip link show
6: ens8np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 24:8a:07:92:3b:b8 brd ff:ff:ff:ff:ff:ff
    vf 0     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking
off, link-state auto, trust off, query_rss off
    vf 1     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking
off, link-state auto, trust off, query_rss off
    vf 2     link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking
off, link-state auto, trust off, query_rss off
…
```

● Intel 100G NIC の場合

1) 対象の 100GNIC に VF を作成する
以下コマンドの ens93f0 は対象の 100GNIC のインターフェース名に合わせて変更すること

```
$ sudo -s
# echo 40 > /sys/class/net/ens93f0/device/sriov_numvfs
```

2) 対象の 100GNIC に、上記 2 で指定した数の VF が作成されたことを確認する

```
# lspci -nn | grep Intel |grep Virtual
17:01.0 Ethernet controller [0200]: Intel Corporation Ethernet Adaptive Virtual
Function [8086:1889] (rev 02)
17:01.1 Ethernet controller [0200]: Intel Corporation Ethernet Adaptive Virtual
Function [8086:1889] (rev 02)
17:01.2 Ethernet controller [0200]: Intel Corporation Ethernet Adaptive Virtual
Function [8086:1889] (rev 02)
17:01.3 Ethernet controller [0200]: Intel Corporation Ethernet Adaptive Virtual
Function [8086:1889] (rev 02)
…
```

8.9.3 SR-IOV デバイスプラグインセットアップ

対象：K8s control plane

K8s 上で SR-IOV デバイスを扱えるようにするため、SR-IOV デバイスプラグインをセットアップする

(1) SR-IOV デバイスプラグインの入手

```
$ cd ~/
$ git clone https://github.com/k8snetworkplumbingwg/sriov-network-device-
plugin.git -b v3.7.0
```

(2) SR-IOV デバイスプラグインの ConfigMap を適用

(1)で取得した SR-IOV デバイスプラグインに含まれる configMap.yaml を適用する。
configMap.yaml には SR-IOV の VF を作成する NIC に関して、NIC の管理上の名称、ベンダーコード、デバイスドライバ、デバイスコードの情報を記載する必要があるため、記載が無い場合は適用前に追記が必要になる。

● 本書の想定環境で用いている Mellanox 100GNIC を利用する場合
デフォルトの configMap.yaml には Mellanox 100GNIC 向けの記載が無いため、適用前に configMap.yaml に Mellanox 100GNIC 向けの追記する必要がある。
※Mellanox 100GNIC に限らず configMap.yaml に記載の無い NIC を用いる場合は同様の手順を踏めばよい。

・必要な情報の取得
まずは追記に必要な情報を取得する。SR-IOV の VF を作成する NIC を持つサーバ(K8s Node)にて、当該 NIC の vendors および devices に記載する情報を取得する。
- ベンダ情報、デバイス情報(物理デバイスと仮想デバイスの両方)：下記コマンドにて取得する。

```
 $ lspci -nn | grep Mellanox
 -出力例-
 ae:00.0 Ethernet controller [0200]: Mellanox Technologies MT27800 Family
[ConnectX-5] [15b3:1017]
 ae:00.1 Ethernet controller [0200]: Mellanox Technologies MT27800 Family
[ConnectX-5 Virtual Function] [15b3:1018]
 ae:00.2 Ethernet controller [0200]: Mellanox Technologies MT27800 Family
[ConnectX-5 Virtual Function] [15b3:1018]
 ae:00.3 Ethernet controller [0200]: Mellanox Technologies MT27800 Family
 [ConnectX-5 Virtual Function] [15b3:1018]
 （中略）
 ae:00.7 Ethernet controller [0200]: Mellanox Technologies MT27800 Family
[ConnectX-5 Virtual Function] [15b3:1018]
```

- driver 情報：当該 NIC のインターフェース名を取得したうえで下記コマンドにて取得する。

```
$ ethtool -i ens8np0  #ens8np0 は当該 NIC のインターフェース名
-出力例-
driver: mlx5_core
--略--
```

・configMap.yaml の編集

上記で取得した情報に基づき configMap.yaml を編集(追記)する。

```
$ cd ~/sriov-network-device-plugin/deployments
$ vi configMap.yaml
（末尾に以下を追加）
----
            {
              "resourceName": "mlnx_sriov_device", #resouceName は任意の名称で良い
              "resourcePrefix": "nvidia.com",
              "selectors": {
                 "vendors": ["15b3"],
                 "devices": ["1017", "1018"],
                 "drivers": ["mlx5_core"]
              }
             }
----
```

・configMap.yaml の適用：

```
$ cd ~/sriov-network-device-plugin/deployments
$ kubectl apply -f configMap.yaml
```

● configMap.yaml に記載のある NIC（Intel 100GNIC 等）を用いる場合

対象 NIC の設定がデフォルトの configMap.yaml に記載されている場合は configMap.yaml の編集は不要で、デフォルトの configMap.yaml を適用すれば良い。

```
$ cd ~/sriov-network-device-plugin/deployments
$ kubectl apply -f configMap.yaml
```

参考として、デフォルトの configMap.yaml に記載済みの Intel 100GNIC 用の設定（赤字の部分）を以下に示す。

```
…
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
        "resourceList": [{
               "resourceName": "intel_sriov_netdevice",
               "selectors": {
                  "vendors": ["8086"],
                  "devices": ["154c", "10ed", "1889"],
                  "drivers": ["i40evf", "iavf", "ixgbevf"]
               }
          },
…
```

8.9.4 SR-IOV デバイスプラグインの実行

対象：K8s control plane

1) SR-IOV デバイスプラグインを daemonset として実行する

```
$ cd ~/sriov-network-device-plugin/deployments
$ kubectl apply -f sriovdp-daemonset.yaml
```

※OS 再起動後の本手順の再実行において、既に SR-IOV デバイスプラグインの daemonset が実行されている場合は、以下コマンドにて一度 daemonset を delete してから、再度 apply する

```
$ cd ~/sriov-network-device-plugin/deployments
$ kubectl delete -f sriovdp-daemonset.yaml
$ kubectl apply -f sriovdp-daemonset.yaml
```

2) K8s node で作成した VF が、当該 K8s node 上で割り当て可能なリソースとして認識されたことを確認する。以下の赤字部分が K8s node が割り当て可能な VF 数であり、作成した VF 数と同じであれば良い。

- 8.9.1 節で「対象 NIC が Intel 100GNIC の場合」の手順を実施していた場合

```
$ kubectl describe node swb-sm7 | grep -A 10 Allocatable
Allocatable:
  cpu:                                64
  (中略)
  intel.com/intel_sriov_netdevice:   40
  (後略)
```

- 8.9.1 節で「対象 NIC が Mellanox 100GNIC の場合」の手順を実施していた場合

```
$ kubectl describe node swb-sm7 | grep -A 10 Allocatable
Allocatable:
  cpu:                                64
  (中略)
  nvidia.com/mlnx_sriov_device:      40
  (後略)
```

3) また、現在使用中の VF 数を確認したい場合は以下の様に K8s node の” Allocated resources”を見れば良い。赤字部の Requests または Limits の値が使用中の VF 数となる。

● 8.9.1 節で「対象 NIC が Intel 100GNIC の場合」の手順を実施していた場合

```
 $ kubectl describe node swb-sm7 | grep -A 10 “Allocated resources”
Allocated resources:
    (Total limits may be over 100 percent, i.e., overcommitted.)
   Resource                          Requests      Limits
   --------                          --------      ------
   cpu                               900m (0%)     1400m (1%)
   (中略)
   intel.com/intel_sriov_netdevice   40            40
```

● 8.9.1 節で「対象 NIC が Mellanox 100GNIC の場合」の手順を実施していた場合

```
 $ kubectl describe node swb-sm7 | grep -A 10 “Allocated resources”
Allocated resources:
    (Total limits may be over 100 percent, i.e., overcommitted.)
   Resource                          Requests      Limits
   --------                          --------      ------
   cpu                               900m (0%)     1400m (1%)
   (中略)
   nvidia.com/mlnox_sriov_device     40            40
```

以上を持って環境構築は完了となる。構築した環境にて DataFlow の配備や疎通を行う手順は別紙「OpenKasugai-Demo」に記載されているので、そちらを参照して実施すれば良い。

9. 付録

ここでは FAQ として本書 8 章までにおける補足事項や例外ケースの手順について記載する。

9.1 ConfigMap を作成し直したい場合

8.6 節にて ConfigMap を作成した後に ConfigMap を作り直したい場合は 8.6 節をやり直すだけでは正しく ConfigMap を作成出来ないケースもある。ConfigMap の再作成の手順は、ConfigMap を作成した後にどこまで作業を進めているかで変わってくる。
パターンは以下の 2 種類になる。

- ConfigMap 作成直後に作り直したい場合（8.6 節実施直後の場合）
- DataFlow(以下、DF)を配備後に作り直したい場合
  （別紙「OpenKasugai-Demo」の「1.3.1 DataFlow 配備」を実施済みの場合）

9.1.1 ConfigMap 作成直後に作り直したい場合（8.6 節実施直後の場合）

~/controller/src/tools/InfoCollector 配下の ConfigMap
DF 配備前なのでそのまま 8.6 節をやり直せば良い。
複数 K8s node 構成の場合は、8.6 節の 1.はスキップして、自動収集&CM 取得をやり直したい K8s node でのみ 2.以降を実施すれば良い。

9.1.2 DataFlow 配備後に作り直したい場合

このパターンの場合、DF を配備しているため各種コントローラ(CRC)を初期状態に戻す必要がある。
自動生成&CM 作成ツールで作成する CM 群は、ComputeResource（CR）の新規作成時に使用する。
ComputeResource の新規作成は、DeviceInfo コントローラの起動時に行う想定で、DeviceInfo コントローラは各種コントローラと同じタイミングで起動させる想定である。

1. 各種コントローラ(CRC)の停止：別紙「OpenKasugai-Demo」の「1.4.1 DataFlow 削除と各種 CR の削除、各 CRC の停止」を実施
2. ~/controller/src/tools/InfoCollector 配下の ConfigMap の作り直し：8.6 節を実施
3. 各種 CRC の起動：別紙「OpenKasugai-Demo」の「1.2.2 CRC 起動」を実施

9.2 DataFlow を 1 本だけ流したい場合

別紙「OpenKasugai-Demo」にて複数の DataFlow を流すデモにおいて DataFlow を 1 本のみ流したい場合は、使用する DataFlow ファイルを 1 つのみとして、該当の手順を実施すれば良い。
ただし、別紙「OpenKasugai-Demo」の「1.3.2 映像受信開始」と「1.3.3 映像配信開始」については、使用する DataFlow のタイプや配備する DataFlow の本数によって映像受信スクリプト、映像配信スクリプトの設定内容が変わるので、実施内容に合わせたスクリプト編集を行うこと

9.3 FPGA を環境構築直後の初期状態に戻したい場合

FPGA をシステム構築直後の初期状態に戻したい場合は、別紙「OpenKasugai-Demo」の「1.2.1 の(1)FPGA Bitstream の書込み」を実施

9.4 CRC を更新したい場合

CRC（カスタムリソースコントローラ）の更新が必要になった場合は以下の手順を実施する。評価を実施していた場合は、必ず別紙「OpenKasugai-Demo」の「1.4.1 DataFlow 削除と各種 CR の削除、各 CRC の停止」の手順にて各種 CR の削除と CRC の停止が行われた状態から始めること。
なお、CRC を更新したいケースとして以下の 2 つを想定し、各ケースについて手順を記載した。

- CRC のソースが更新されたケース➡9.4.1 節
- 別紙「OpenKasugai-Demo」の「1.3.1 DataFlow 配備」を実施した際に不具合が発生したケース➡9.4.2 節
  - CRC が正常に起動しない、もしくは CRC は正常に起動したが CR が正常に起動しない様なケースで、CRC(全体もしくは正常起動しない CRC のみ)を入れ直したい場合

9.4.1 提供ファイルの更新に伴う CRC の更新

(1) 全 CRC を更新したい場合

資材一覧の CRC ソースコードを用いて CRC を一括して更新したい場合は、8.1.2 節、8.2 節、8.3 節を実施することで更新が完了する。

(2) 特定の CRC のみを更新したい場合

資材一覧の CRC ソースコードを用いて特定の CRC を更新したい場合は、8.1.2 節、8.2 節の該当の CRC の作業、8.3 節の該当の CRC 起動準備作業を実施することで更新が完了する。

9.4.2 CRC の正常起動が確認できない等の不具合による CRC の更新

(1) 全 CRC を更新したい場合

CRC を一括して更新したい場合は、8.2 節、8.3 節を実施することで更新が完了する。

(2) 特定の CRC のみの更新を行いたい場合

特定の CRC を更新したい場合は、8.2 節の該当の CRC の作業、8.3 節の該当の CRC 起動準備作業を実施することで更新が完了する。

9.5 評価環境をリセットしたい場合

別紙「OpenKasugai-Demo」のに従ってデモを実施している際に、不具合が発生する等の理由で評価環境を初期状態にリセットしたい場合は、次の手順を実施する。

1. 別紙「OpenKasugai-Demo」の「1.3.4 映像配信・受信停止」と「1.4 環境停止」を実施
   映像配信が実施中であればそれを停止し、各種 CR 削除と各種 CRC 停止を行う。
2. 9.4.2 節を実施
   全ての CRC の更新を再度行うことで CRC が初期化される。
3. 別紙「OpenKasugai-Demo」の「1.2 環境初期化」を実施
   評価前の環境初期化を行うことで評価環境の初期状態にリセットされる。

上記によりデモの再実施再評価(別紙「OpenKasugai-Demo」の「1.3 映像配信」以降)を行うことが出来る。

9.6 ghcr のコンテナイメージを使う場合

ghcr(GitHub Container Registry)からコンテナイメージを取得して使う場合は、当該コンテナイメージのビルド手順は不要になる。取得可能なコンテナイメージを下表に示す。

	コンテナイメージ名
CPU デコード処理モジュール	ghcr.io/openkasugai/controller/cpu_decode:1.1.0
CPU フィルタリサイズ処理モジュール	ghcr.io/openkasugai/controller/cpu_filter_resize:1.1.0
映像受信ツール	ghcr.io/openkasugai/controller/rcv_video_tool:1.1.0
映像配信ツール	ghcr.io/openkasugai/controller/send_video_tool:1.1.0
DataFlow コントローラ、WBscheduler コントローラ、FunctionTarget コントローラ、FunctionType コントローラ、FunctionChain コントローラ	ghcr.io/openkasugai/controller/whitebox-k8s-flowctrl:1.1.0
WBConnection コントローラ	ghcr.io/openkasugai/controller/wbconnection:1.1.0
WBFunction コントローラ	ghcr.io/openkasugai/controller/wbfunction:1.1.0
CPUFunction コントローラ	ghcr.io/openkasugai/controller/cpufunction:1.1.0
GPUFunction コントローラ	ghcr.io/openkasugai/controller/gpufunction:1.1.0
FPGAFunction コントローラ	ghcr.io/openkasugai/controller/fpgafunction:1.1.0
EthernetConnection コントローラ	ghcr.io/openkasugai/controller/ethernetconnection:1.1.0
PCIeConnection コントローラ	ghcr.io/openkasugai/controller/pcieconnection:1.1.0
DeviceInfo コントローラ	ghcr.io/openkasugai/controller/deviceinfo:1.1.0

対象サーバにてコンテナイメージを取得し、コンテナイメージ名の ghcr.io/openkasugai/controller の部分を localhost に変更する。
以下に CPU デコード処理モジュールのコンテナイメージの取得・変更のコマンド例を示す。

```
$ sudo buildah pull ghcr.io/openkasugai/controller/cpu_decode:1.1.0
$ sudo buildah tag ghcr.io/openkasugai/controller/cpu_decode:1.1.0 localhost/cpu_decode:1.1.0
```

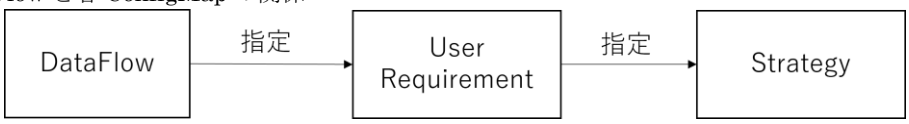
DataFlow のスケジューリング戦略を設定する場合
作業対象：K8s control plane

DataFlow のスケジューリング戦略を設定または変更する場合は、別紙「OpenKasugai-Demo」の「1.3.1 DataFlow 配備」の前に、以下の操作を実行して、必要な ConfigMap の作成や、DataFlow の YAML の編集を行う

DataFlow のスケジューリング戦略の設定内容の概要

名称	設定する内容
Strategy	・DataFlow の配備先候補のフィルタリング/スコアリングに利用する filter を指定
UserRequirement	・DataFlow の配備先の条件を指定 ・DataFlow が利用する Strategy の ConfigMap を指定
DataFlow	・UserRequirement の ConfigMap を指定

※DataFlow と各 ConfigMap の関係



●必要な操作

1. Strategy の ConfigMap の作成（記載する内容については、以下の 9.6.1 節を参照）
   別紙「OpenKasugai-Demo」の「1.3.1 DataFlow 配備」における各 DataFlow 共通の Strategy として、1.2.3 節(1)の run_controllers.sh によりデフォルトの Strategy の ConfigMap が配備される。
   別の Strategy を利用する場合は、新たな ConfigMap を別途作成する。

2. UserRequirement の ConfigMap の作成（記載する内容については、以下の 9.6.2 節を参照）
   別紙「OpenKasugai-Demo」の「1.3.1 DataFlow 配備」における各 DataFlow 共通の UserRequirement として、「1.2.2 節(1)各 CRC の起動」の run_controllers.sh により /home/ubuntu/k8s-software/src/tools/InfoCollector/testdata_day/user_requirement/user_requirement.yaml が配備される。別の UserRequirment を利用する場合は、新たな ConfigMap を別途作成する。

3. 配備対象の DataFlow にて、上記 2 の UserRequirement を指定（指定方法は、以下の 9.6.3 節を参照）

9.6.1 Strategy の ConfigMap の設定内容

●設定項目

項目名	データ型	Required	内容
referenceParameter	string	No	・Strategy の設定のために参照する、別の Strategy の ConfigMap の metadata.Name を指定
filterPipeline	[]string	No	・DataFlow のスケジューリングで利用する filter を指定 ・filter の種類は以下の 6 つ ①GenerateCombinations：配備先デバイスを考慮した各 filter の処理に必要な情報の生成 ②TargetResourceFit：配備先デバイスの種別を考慮したフィルタリング ③TargetResourceFitScore：配備先のデバイスの容量を考慮したスコアリング ④GenerateRoute：トポロジ情報を考慮した各 filter の処理に必要な情報の生成

			⑤ConnectionResourceFit：トポロジ情報上の接続経路を考慮したフィルタリング ⑥RouteScore：トポロジ情報上の接続の容量を考慮したスコアリング ・トポロジ情報を考慮したスケジューリングを行う場合は、上記の①～⑤を全て指定。トポロジ情報を考慮しないスケジューリングを行う場合は、上記の①～③のみを指定 ・Strategy および UserRequirement を DataFlow で利用しない場合は、デフォルトの filter として上記の①～③が実行される
selectTop	int	No	・filter のフィルタリング結果について、Score が<設定値>番目のものまでを取得する
<N>.referencePara meter	string	No	・filterPipeline の<N>番目の filter について、Strategy の設定のために参照する別の Strategy の ConfigMap の metadata.Name を指定
<N>.selectTop	int	No	・filterPipeline の<N>番目の filter のフィルタリング結果について、Score が<設定値>番目のものまでを取得する

●設定例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: strategy
  namespace : default
data:
  filterPipeline: |
    - GenerateCombinations
    - TargetResourceFit
    - TargetResourceFitScore
    - GenerateRoute
    - ConnectionResourceFit
    - RouteScore
  selectTop : "5"
```

9.6.2 UserRequirement の ConfigMap の設定内容

●設定項目

項目名	データ型	Required	内容
strategy	string	Yes	・Strategy の設定のために参照する、別の Strategy の ConfigMap の metadata.Name を指定
requestNodeNames	map[string][]string	No	・"map のキーに指定したファンクションの配備先/非配備先の nodeName を指定する。 ・map のキーは Function を示す FunctionChain.FunctionChainSpec.Functions の key 値。

			・map の値は nodeName の配列。'-'を付与することで非配備先としての指定となる
requestDeviceTypes	map[string][]string	No	・map のキーに指定したファンクションの配備先/非配備先の deviceType を指定する。 ・map のキーは Function を示す FunctionChain.FunctionChainSpec.Functions の key 値。 ・map の値は deviceType の配列。'-'を付与することで非配備先としての指定となる
requestFunctionTargets	map[string][]string	No	・map のキーに指定したファンクションの配備先/非配備先の FunctionTarget を指定する。 ・map のキーは Function を示す FunctionChain.FunctionChainSpec.Functions の key 値。 ・map の値は FunctionTarget の配列。'-'を付与することで非配備先としての指定となる
requestRegionNames	map[string][]string	No	・filterPipeline の<N>番目の filter のフィルタリング結果について、Score が<設定値>番目のものまでを取得する
requestFunctionIndexes	map[string][]string	No	・map のキーに指定したファンクションの配備先/非配備先の deviceType を指定する。 ・map のキーは Function を示す FunctionChain.FunctionChainSpec.Functions の key 値。 ・map の値は regionName の配列。'-'を付与することで非配備先としての指定となる
functionTargetNameSpace	string	No	デバイス情報を利用する Filter を実行する際に参照する FunctionTarget の metadata.namespace を指定

●設定例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-requirement
  namespace : default
data:
  strategy : strategy
  requestNodeNames : |
    decode-main:
      - server1
    filter-resize-main:
      - server1
      - server2
  requestFunctionTargets: |
    high-infer-main:
      - node1.a100-0.gpu
  requestRegionNames : |
    decode-main:
      - lane0
  requestDeviceTypes : |
    filter-resize-main:
      - cpu
```

9.6.3 DataFlow での UserRequirement の指定

●設定例

以下の赤字部分（Spec.UserRequiremet）にて、UserRequirement の ConfigMap の metadata.name を指定する

```
apiVersion: example.com/v1
kind: DataFlow
metadata:
  name: "df-ext-2-1"
  namespace: "test01"
spec:
  functionChainRef:
    name: "cpu-decode-filter-resize-glue-high-infer-chain"
    namespace: "chain-imgproc"
  requirements:
    all:
      capacity: 15
  functionUserParameter:
  - functionKey: decode-main
    userParams:
      ipAddress: 192.174.90.101/24
      inputPort: 5004
  - functionKey: glue-fdma-to-tcp-main
    userParams:
      ipAddress: 192.174.90.131/24
      glueOutputIPAddress: 192.174.90.141
      glueOutputPort: 16000
  - functionKey: high-infer-main
    userParams:
      ipAddress: 192.174.90.141/24
      inputIPAddress: 192.174.90.141
      inputPort: 16000
      outputIPAddress: 192.174.90.10
      outputPort: 2001
  userRequirement: user-requirement
```

9.7 Intel/Mellanox100/25GNIC の MTU9000 設定

対象：高度推論アプリが配備される K8s node、映像受信ツールが動いているサーバ

以下では、図 1 の想定環境を例として設定手順を示す。

● 送信側(K8s node 2)の設定

1) NIC の物理ポートの MTU の設定を行う。
/etc/netplan/00-installer-config.yaml を以下の様に編集する。

```
ens93f0:
    addresses:
    - 192.174.90.33/24
    mtu: 9000          ★mtu を 9000 に設定
```

2) システムへ適用する。以下コマンドで仮適用する。

```
$ sudo netplan try
[sudo] password for ubuntu:
Warning: Stopping systemd-networkd.service, but it can still be activated by:
  systemd-networkd.socket
Do you want to keep these settings?

Press ENTER before the timeout to accept the new configuration

Changes will revert in 120 seconds
```

上記メッセージのように特にエラーが出なければ、ここでリターンキーを押すと仮適用が本適用となる。

3) mtu 9000 と表示されていることを確認する。

```
$ ip addr show ens93f0
2: ens93f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group
default qlen 1000
 link/ether b4:96:91:ad:8f:70 brd ff:ff:ff:ff:ff:ff
 inet 192.174.90.33/24 brd 192.174.90.255 scope global ens93f0
      valid_lft forever preferred_lft forever
 inet6 fe80::b696:91ff:fead:8708/64 scope link
      valid_lft forever preferred_lft forever
```

● 受信側(K8s control plane)の設定

1) NIC の物理ポートの IP アドレスの設定、および MTU の設定を行う。
/etc/netplan/00-installer-config.yaml を以下の様に編集する。

```
ens7f0:
    match:
        macaddress: b4:96:91:ca:54:b8
    addresses:
    - 192.174.90.11/24
    mtu: 9000          ★mtu を 9000 に設定
```

2) システムへ適用する。以下コマンドで仮適用する。

```
$ sudo netplan try
[sudo] password for ubuntu:
Warning: Stopping systemd-networkd.service, but it can still be activated by:
  systemd-networkd.socket
Do you want to keep these settings?

Press ENTER before the timeout to accept the new configuration

Changes will revert in 120 seconds
```

上記メッセージのように特にエラーが出なければ、ここでリターンキーを押すと仮適用が本適用となる。

3) mtu 9000 と表示されていることを確認する。

```
$ ip addr show ens7f0
4: ens7f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group
default qlen 1000
    link/ether b4:96:91:ca:54:b8 brd ff:ff:ff:ff:ff:ff
    altname enp174s0f0
    inet 192.174.90.11/24 brd 192.174.90.255 scope global ens7f0
       valid_lft forever preferred_lft forever
    inet6 fe80::b696:91ff:feca:54b8/64 scope link
       valid_lft forever preferred_lft forever
```

9.8 映像配信実施後に K8s node をコールドリブートして映像配信をやり直したい場合

以下の手順に従って実施することで、映像配信可能な状態（別紙「OpenKasugai-Demo」の「1.3 映像配信」が実行できる状態）に戻せる。

1．各種 CR の削除、各 CRC の停止：別紙「OpenKasugai-Demo」の「1.4.1 DataFlow 削除と各種 CR の削除、各 CRC の停止」を実施

2．コールドリブート実施

```
$ sudo poweroff
```

3．環境の初期化（FPGA の書込み、CRC 起動）：別紙「OpenKasugai-Demo」の「1.2 環境初期化」を実施