OpenKasugai-Controller Install Manual

v1.1.0

Contents

0. Introduction

This document describes how to build a running environment for the OpenKasugai demo.

0.1 Notes

The following points should be noted when using this manual.

・There is no difference in the material list version.

・Emphasis letters indicate points of emphasis, and emphasis letters indicate points of particular attention.

・"Target" under the chapter title indicates the type of physical server to which the operation procedure should be performed.

・Unless otherwise specified, the execution account for various operations is assumed to be "ubuntu." Thus, for directories, “$HOME ” assumes“/home/ubuntu/.”

・Target Systems:

- Target K8s node nodes: no accelerator, only Xilinx FPGAs, only NVIDIA GPUs, both Xilinx FPGAs and NVIDIA GPUs

  *It does not support hot-plugging of various PCIe devices or dynamic reconfiguration of FPGAs.

0.2 Procedure summary

The following is an overview of the environment construction procedures described in this document.

List of environment configuration procedures

Chapter	Major item	Description
1	Equipment installation	Install physical servers with expansion slots.
2	Equipment settings	Configure and connect the 100Gb switch.
3	OS installation	Install and configure the OS on the physical server.
4	FPGA setup	Set up the FPGA card.
5	Container management base setup	Set up the container management infrastructure (K8s).
6	GPU setup	Set up the GPU card.
7	Setup of various processing modules	Various processing modules executed on CPUFunction and GPUFunction, and containers such as video distribution and video reception tools are set up.
8	Controller setup	Set up a controller for deploying data flows.
9	Evaluation procedure	Deploy dataflows to evaluate data communication.
10	Appendix	FAQ related.

0.3 List of assumed devices

The following shows the configuration of equipment in the environment assumed in this document.

List of devices used in the assumed environment

Paragraph	construct	Type	Equipment
1	K8s control plane	Physical server	• Fujitsu PRIMERGY RX2540M6
2	K8s node		
3	100Gb Switch (OS:Mellanox ONYX)	Switch	Mellanox SN2100-CB2F
4	FPGA card (for filter size)	FPGA card	Xilinx Alveo U250
5	GPU card (for advanced inference)	GPU card	NVIDIA A100(80GB)
6	100G NIC(Intel)	NIC	Intel E810CQDA2
7	100G NIC(Mellanox)		Mellanox ConnectX 5
8	AOC cable	Cable	Mellanox MFA1A00-C003 Compatible AOC Cable

Refer to the "1. Estimated Environment Chart" sheet in the separate document "OpenKasugai-Controller-InstallManual_Attachment1" for the physical environment configuration diagram and software configuration diagram.

0.4 List of Software and FPGA Circuit Versions Used

The following is a list of versions of software and FPGA circuits used in the environment to be constructed in this document.

List of Software Versions Used

Software		Version
Ubuntu		22.04.5 LTS
Vivado		2023.1.0
DPDK		23.11.1 LTS
k8s		1.31.1
container runtime		CRI-O v1.31.0
CNI		Calico v3.28.1
Multus		4.1.1
SR-IOV device plug-in		3.7.0
SR-IOV CNI plug-in		2.8.1
CRC (Controller)		1.1.0
kubebuilder		3.12.0
go		1.23.0
NVIDIA Driver		550.90.12
nvidia-container-toolkit		1.16.2
Inference Function Container	DeepStream	7.0
	CUDA	12.2
	TensorRT	8.6.1.6-1+cuda12.0
	gcc	11.4.0
	GStreamer	1.20.3

List of FPGA circuit versions

Circuit		File name
Filte・resize circuit	mcs	OpenKasugai-fpga-example-design-1.0.0-1.mcs
	bit	OpenKasugai-fpga-example-design-1.0.0-2.bit

0.5 Material list

The following is a list of materials required for the environment to be built in this document.

Material list

chapter	Product name	Description	Provision
1	FPGA write scripts	Script to write FPGA binary data (MCS file) to phase3 Flash	・Retrieved from hardware-drivers repogitory (v1.0.0) on github(OpenKasugai)
2	FPGA binary data	・mcs file ・Filter Resize bit file	・Retrieved from hardware-design repository (v1.0.0) on github(OpenKasugai) ・ mcs file ・ F/R bit files
3	FPGA software suite	FPGA drivers and libraries	・Retrieved from hardware-drivers repository (v1.0.0) on github(OpenKasugai) ・FPGA library driver
4	CRC source code & test data	Custom resource controller software source code and test data (YAML and JSON)	・Retrieved from controller repository (v1.1.0) on github(OpenKasugai) ・Scheduler and various CRC sources ・Automatic acquisition and CM creation function ・FPGAReconfigrationTool ・FPGAClearCheckTool ・test data & test script set
5	Inference processing module	Source code and container images for GPU-powered inference Pod	・Source code: Retrieved from sample-functions/functions/in controller repository(v1.1.0) on github(OpenKasugai) ・dma version: gpu_infer_dma_plugins / ・tcp version: gpu_infer_tcp_plugins /
6	Decode processing module	Container image of the Pod decoded by the CPU	・Source code: from the controller repository(v1.1.0) on github(OpenKasugai) Retrieved from sample-functions/functions/cpu_decode /
7	Filter size processing module (for CPU)	Container image of a Pod with CPU filtering	・Source code: from the controller repository(v1.1.0) on github(OpenKasugai) sample-functions/functions/cpu_filter_resize/
8	Copy branch processing module (for CPU)	Container image of a Pod that performs copy branch processing on the CPU	・Source code: from the controller repository(v1.1.0) on github(OpenKasugai) sample-functions/functions-ext/cpu_copy_branch/
9	Glue processing module (for CPU)	Container image of a Pod that performs a glue conversion (dma→tcp) on the CPU	・Source code: from the controller repository(v1.1.0) on github(OpenKasugai) sample-functions/functions-ext/cpu_glue_dma_tcp /
10	Demo individual features	Video distribution tool for PoC demonstration	・Retrieved from sample-functions/utils/in controller repository(v1.1.0) on github(OpenKasugai) ・ Send server: send_video_tool / ・ Receive server: rcv_vide_tool /

11	demonstration video	Movie files used for distribution	Retrieved from external sites such as • 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. Equipment installation

1.1 Installing physical server・Switch

Figure 1 System Physical Configuration and Card Loading Slots shows the physical configuration of the system exemplified in this document and the setting targets in this chapter.

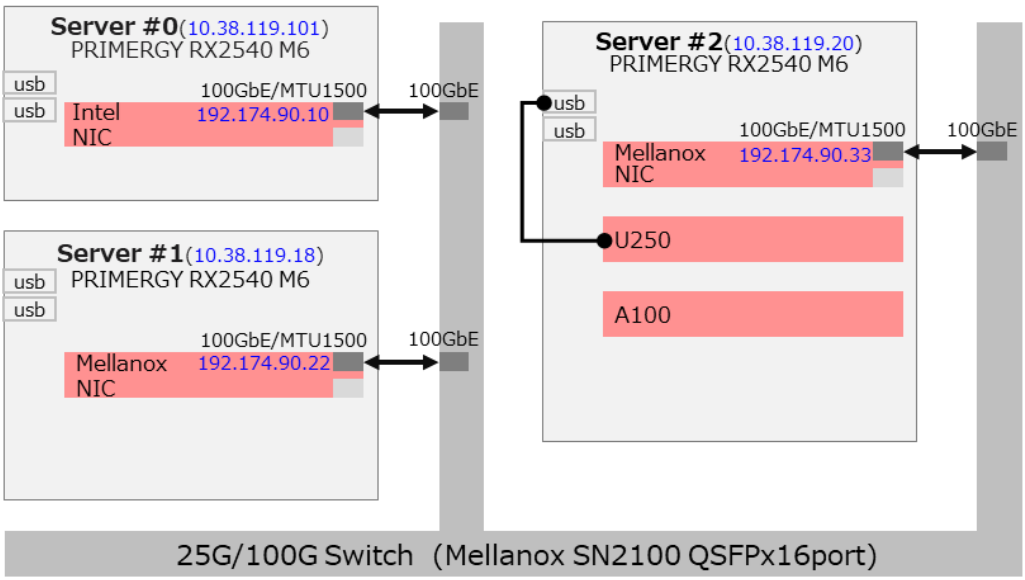

Figure 1 System Physical Configuration and Card Loading Slots

The FPGA card, GPU card, and NIC card are installed in the slot position shown above.

Server name	Use	Card	Number of copies
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 Alveo U250 (FPGA card) NVIDIA A100 (GPU card)	1 1 1

2. Equipment settings

2.1 Verifying the 100Gb Switch configuration

Fig. 2.1 shows the setting targets of this chapter in the system physical configuration. This chapter verifies the configuration of the 100Gb switch.

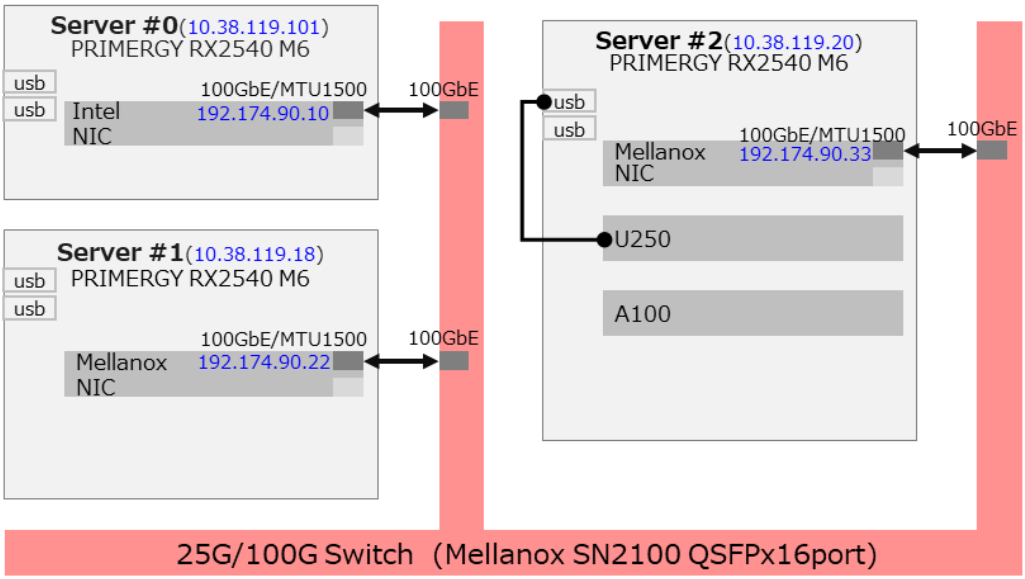

Figure 2.1 Switch to be configured

Serial Console The device is a Mellanox SN2100 CB2F (OS: Mellanox ONYX). The front view is shown below.

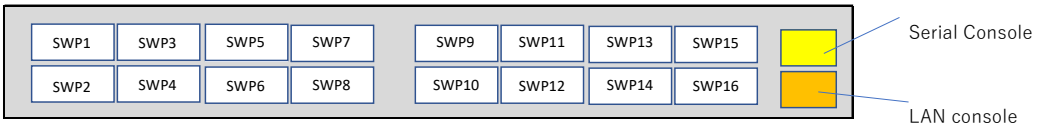

Figure 2.2 Mellanox SN2100-CB2F Front View

The Mellanox SN2100 CB2F has a total of 16 communication ports, with a default of 100GbE.
No configuration changes are required when connecting to a 100G NIC.

2.1.1 Advance preparation

Make a serial connection to the serial console and configure the LAN console IP settings.

Physical connection images are:

- **Serial Console ---- LAN-to-Serial Conversion ---RS232C Cable ---- Serial USB Conversion ---- WinPC**

Serial console settings on WinPC are:

- Baud rate: 115200

Other default settings

After setting the IP address on the LAN console, connect the LAN cable to the LAN console to enable SSH connection. Check the ID/PWD with the administrator.

2.1.2 Configuration confirmation

(1) Example of All Ports Configuration Check Command

Verify that the port to which the 100G NIC is connected (example: Eth1/1- 1/4 below) is 100G and **Up**.

```
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 Connection check

Cable the SW port to the 100GNIC.

Make sure that the link LED of the corresponding port of the switch is lit when connecting.

3. OS installation

3.1 OS installation

Target: K8s control plane, all K8s nodes

Fig. 3 shows the setting targets of this chapter in the overall software configuration.
In this chapter, the OS is installed on the physical server.

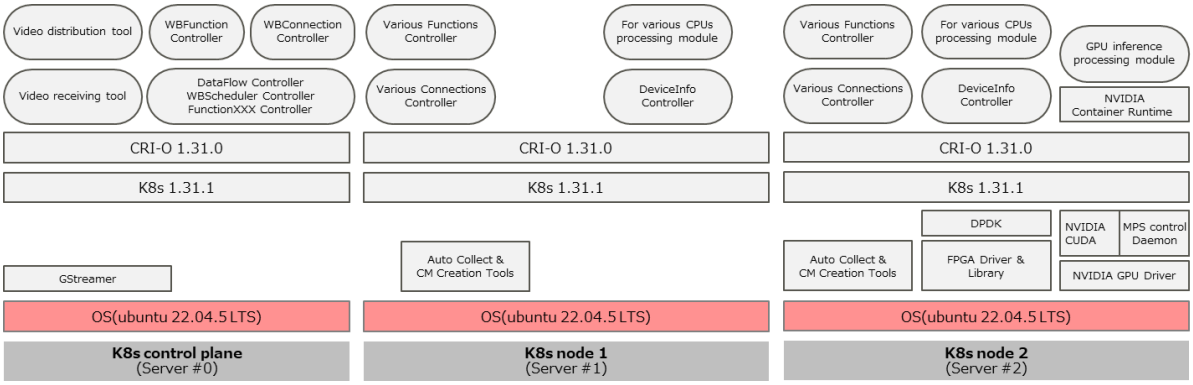

Figure 3 OS Installation Target Server

3.1.1 Ubuntu 22.04.5 Installation

Install the following OS:

・OS:ubuntu: 22.04.5

・ISO image: ubuntu-22.04. 5-live-server-amd64.iso

The recommended parameters for OS installation are as follows: An additional package is not required.
*Note: Be careful not to use automatic updates to update the version.
Automatic update off setting is described below (see 3.2.1).

OS installation parameter sheet

Language	English
Keyboard Settings (Layout/Variant)	Japanese
Base installation	Ubuntu Server * Not "minimized"
Network Settings	Set after OS installation to prevent automatic update
Proxy Settings	Set according to the presence or absence of a proxy server
Mirror Server Settings	Do not change in particular (Keep http://archive.ubuntu.com/ubuntu)
Storage Settings	・Select All Disks (Do not select custom storage layout) (no need to partition) (No SWAP region configuration changes required) ・LVM configuration enabled (Encrypt disabled)
User Creation	Set any parameters for each *In this manual, the login user name is ubuntu.
Ubuntu Pro	Skip
SSH server installation	Enabled (select “Install OpenSSH server ”) Do not change anything else (such as importing the SSH public key)

After installation, check the installed OS information and kernel information.

```
$ 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
5.15.0-122-generic
```

3.2 OS settings

3.2.1 Automatic update off setting

Target: K8s control plane, all K8s nodes

Set Ubuntu's automatic update to OFF because it is ON by default.

In the editor, modify/etc/apt/apt.conf.d/20auto-upgrades to:

```
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 Network settings

Target: K8s control plane, all K8s nodes

Configure the following two types of networks for each server.

- C-plane network: network used for operating the OS and controlling kubernetes
- D-plane network: Video to be fed to DataFlow deployed in the separate document "OpenKasugai-Demo" network for transmitting

●About Networking for D-Plane

A video distribution tool for transmitting an input video and a video reception tool for receiving an inference result (video) are arranged on a server separate from the K8s nodes for performing each process for performing inference. Under the above structure, the video transmission route of “video distribution server → server for each processing → video reception server ” is constructed as a D-plane network via a 100Gb Switch.

In this document, the roles of the servers shown in FIG. 1 are as follows.

- Video distribution server: K8s control plane (Server #0)
- Processing Servers: K8s node 1 (Server #1), K8s node 2 (Server #2)
  *The entire process may be deployed on a single server or across two servers.
- Video receiving server: K8s control plane (Server #0)

Under such a server configuration, a D-plane network (192.174.90.XX) for video transmission routes is constructed. Therefore, the D-plane network is set for the 100 GNICs of all the servers in FIG. 1.

*When deploying DFs with reference in the separate document "OpenKasugai-Demo”, if more DFs are deployed than are described in the separate document "OpenKasugai-Demo”, there is a possibility that due to an increase in the number of deployed DFs, turbulence or block noise may be observed in the video received by the video receiving tool. If so, follow the steps in Section 9.7, "Configuring MTU9000 for Intel/Mellanox100/25 GNIC."

●Configuration steps on each server

This is a common setting for Intel/Mellanox 25G/100G NICs.

The network setup procedure for Server#0 in the assumed environment (The network for the C-plane is the 10.38.119.0 network, and the network for the D-plane is the 192.174.90.0 network.) shown in FIG. 1 will be described below as an example.

The interface of the C-plane NIC in Server#0 is "**ens1f0** ", and the interface of the D-plane NIC is" **ens7f0**."

Create a file called 90-installer-config.yaml in/etc/netplan/and edit it as follows:

```
network:
  ethernets:
    ens1f0:
      addresses: [10.38.119.101/24]
      nameservers:
        addresses – [DNS server to configure]
      routes:
        - to: default
          via – Default gateway address (set if necessary)
    ens7f0:
      addresses: [192.174.90.33/24]
      mtu: 1500
  version: 2
```

Change the mode of the created file to: apply.

```
$ 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
```

Apply to the system. Temporarily apply with the following command:

```
$ 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
```

If there is no particular error as shown in the above message (even if there is a Warning), pressing the return key at this point will output the message "Configuration accepted. " and the provisional application will become the actual application.

Check whether the IP address/subnet mask width (192.174.90.11/24) is set for the applicable interface as follows:

```
$ 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
```

Perform the above procedure on the remaining servers (Server#1, Server#2 in Figure 1).

●Verify NetworkSwitch (SN2100) connection port Speed on each server

For 100GNIC, verify that the link speed is 100GbE with "Speed: 100,000 Mb/s" as shown in the following command example.

```
$ sudo ethtool ens7f0
Settings for ens93f0:
       Supported ports: [ ]
       Supported link modes: 100000baseCR4/Full
       (Omitted)
       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 settings

Target: K8s control plane, all K8s nodes

If using Proxy, set the following in/etc/environment:

```
$ 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@ [proxy server]: [port]"
https_proxy="http://user:password@ [proxy server]: [port]"
HTTP_PROXY="http://user:password@ [proxy server]: [port]"
HTTPS_PROXY="http://user:password@ [proxy server]: [port]"
no_proxy="127.0.0.1, localhost, (IP of host)"
NO_PROXY="127.0.0.1, localhost, (IP of host)"
```

3.2.4 Upgrading the Kernel

Target: K8s control plane, all K8s nodes

If your kernel version is older than the tested version of 5.15.0-122-generic, upgrade.

*It's supposed to work with different versions, but it hasn't been tested.

```
$ 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

(Confirm Kernel Upgrade After Reboot)
$ uname -r
5.15.0-122-generic
```

3.2.5 Time synchronization

Target: K8s control plane, all K8s nodes

Configuring an NTP Server

```
Edit /etc/systemd/timesyncd.conf as follows
[Time]
NTP= [Specify NTP server to configure]


$ 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 (configured NTP
server name)."
      Tasks: 2 (limit: 154194)
     Memory: 19.4M
     CGroup: /system.slice/systemd-timesyncd.service
             mq2240 /lib/systemd/systemd-timesyncd
```

Verify Time Synchronization Is Configured

```
$ timedatectl timesync-status
        Server: XX.XX.XX.XX (configured NTP server name)  *This is the NTP server that you
configured.
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 settings

Target: All K8s nodes

The size of one page is increased from the standard for memory management in order to reduce the load on OS and CPU. Therefore, set up Hugepage on the server.
Here is an example of how to configure and verify Hugepage on ubuntu:

Setting method

Editing/etc/default/grub

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

*Add hugepage-related boot options to GRUB_CMDLINE_LINUX_DEFAULT.
With the above setting, when hugepagesz=1G, 32 pages are allocated at startup.

*Page size: On x86 systems, 1 page size is 4kb. The size of HugePage can be set to 2MB or 1GB on x86 systems. Set the size of 1GB from allocating it to the process that processes the data stream.
*Number of pages (hugepages): Set according to the number of processes processing the data stream.

- Grub configuration and reboot

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

- How to Check

Enter the following command:

```
$ cat /proc/meminfo
...
HugePages_Total: 32 <- Reserve 32 pages
HugePages_Free: 32
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 1048576 kB <- hugepagesize is 1GB
```

4. FPGA setup

This chapter is performed only when FPGAs are used. If an FPGA is not used (FPGA-equipped server is not used), the procedure of this chapter is not necessary.

Fig. 4.1 shows the targets of this chapter in the physical configuration of the assumed system.
In this chapter, the following procedure is used to write data to the FPGA card (AlveoU250) installed in the K8s node 2.

- First, install Vivado on the host machine connected to the FPGA by USB, and write the MCS file.
- Then, the procedure for installing Bitstream writing software (Mcap) in the K8s node and writing Bitstream (BIT file) will be described in the following chapter.

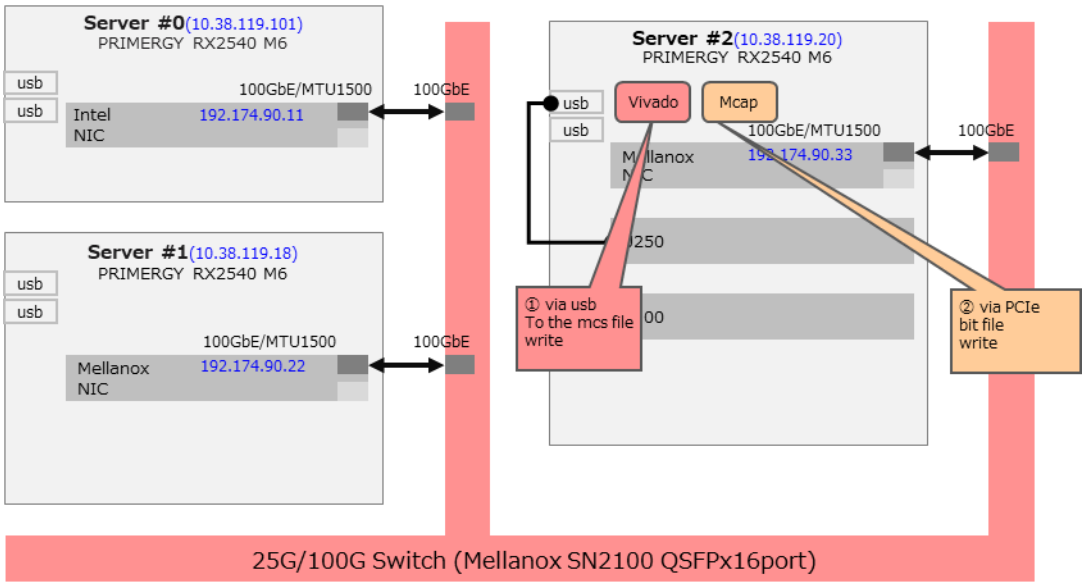

Figure 4.1 Writable FPGA Card and Writing Tool

4.1 Installing Vivado

Target: Host that writes the mcs file (Server #2 is used in the assumed environment)

Install Vivado to write MCS files to the FPGA card.

●Preparation: Getting Vivado
Go to https://japan.xilinx.com/support/download.html/content/xilinx/ja/downloadNav/vivado-design-tools/archive.html and download "AMD Integrated Installer for FPGA and Adaptive SoC) 2023.1 SFD" from the 2023.1 archive.
*Download requires an AMD account, so if you don't have one, get one.
*Be aware that downloading takes a very long time.

4.1.1 Install required packages

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

4.1.2 Generating a configuration file for installation

Extract the acquired Vivado, enter the extracted directory, and generate an installation configuration file.

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

You'll be presented with the following options, the first is "2. Vivado" and the second is "1. Vivado ML Standard."

```
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 Installing Vivado

Run the installation below, specifying the generated install_config.txt.

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

*$HOME/.Xilinx/install_config.txt is the parameter specified by --config

If any of the following errors occurs during execution of the above procedure, take the following actions and install Vivado again.

Description of the error

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

Actions to be taken

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

4.1.4 Running shell scripts required to run Vivado commands

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

Add it to your .bashrc and .bash_profile so that it can be used immediately after a reboot.

- ~/.bash_profile: Add:(Create ~/.bash_profile if it does not exist)

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

- ~/.bashrc: Add the following

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

*It also runs as root, so it's recommended to do the same for root.

(1) Verify Vivado is installed

Enter the following command and confirm that the red text shown below (Vivado v2023.1 (64 bit)) is displayed.

Also check that the version (v2023.1) is correct.

```
$ 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) Install JTAG driver

Enter the following command to install the JTAG driver

```
$ 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) Rebooting the server

```
$ sudo reboot
```

4.2 FPGA card write

This paper describes a procedure for writing configuration data to a filter size FPGA.

●Conditions of Use

A download USB cable is connected between the host and the FPGA card.
(To write MCS files from the host to the FPGA via USB.)

● advance preparation

○ Checking the Xilinx tool installation path (host connected to the FPGA by USB cable): Check the installation path because you are referring to Vivado.

*This document assumes that it is installed in **"/ tools/Xilinx/Vivado/2023.1/."**

○ Check the status of the FPGA card.

・Sample Output When No MCS File Has Been Written

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

・Sample Output When an MCS File Has Been Written

Even if the MCS file is written in this way, there is no problem to proceed with the following steps.

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

4.2.1 Writing MCS files in Vivado

Target: Host that writes the mcs file (Server #2 is used in the assumed environment)

(1) Installing the USB driver for Vivado

Implement (2) of 4.1.4. Skip if already done.

(2) Cloning the repository

Get hardware-design repository and hardware-drivers repository from github in the Materials List.

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

(3) Preparing the MCS File

If you use pre-built MCS/Bitstream, you do not need to prepare it because it is included under the repository obtained in (2).

If you are building by yourself, refer to the build instructions (~/hardware-design/BUILD.md).

(4) Preparing commands for writing MCS files (run_flash.sh) and copying MCS files

Navigate to the tools/run_flash directory and copy the mcs file to be written.

(The MCS file prepared in 0 is assumed to be used.)

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

*About the run_flash.sh Command

- Command Overview
  - ✧ Data is written to the FlashMemory on the FPGA card.
    Once written, it is automatically written to the FPGA even if cold reboot. (There is no need to write every reboot.)
- Command Arguments
  - ✧ -t: Name of the file to write (extension mcs)
    *The MCS file you specify must be in the same directory as run_flash.sh
  - ✧ -i: Specifies the write destination FPGA.
    If 0 is specified, the data is written to the FPGA of "1st detection FPGAID" which is checked by the jtag target command in the following step (5-3).
    If 1 is specified, the data is written to the FPGA of the "second detected FPGAID" that is checked by the jtag target command in the following step (5-3).

(5) Writing MCS Files to ROM

Write the MCS file to the FPGA using run_flash.sh. The -t option specifies the MCS file and the -i option specifies the device index.

If "Flash programming completed successfully" appears, it is successful.

```
$ ./run_flash.sh -t OpenKasugai-fpga-example-design-1.0.0-1.mcs -i 0
# (Omitted)
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...
```

If there are multiple FPGAs, each FPGA must be written. Change the device index when writing. The following is an example of writing to the second FPGA.

```
$ ./run_flash.sh -t OpenKasugai-fpga-example-design-1.0.0-1.mcs -i 1
# (Omitted)
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...
```

After completion, cold-reboot the host. Then, turn on the power by IPMITool or manually.

```
$ sudo poweroff
```

●Example of MCS file writing procedure in the assumed environment shown in Figure 1 in chapter 1

One FPGA card is installed in the server for K8s node2 (Server #2), and USB I/F on the host side is also from the same server. Therefore, Vivado is installed on Server #2 and the MCS file is written to one FPGA card. Perform up to (4) of Section 4.2.1 in advance.

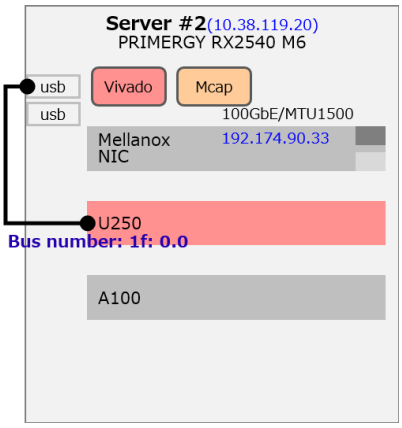

Figure 4.3 Writing an mcs file to the FPGA via USB

(5-1) Connecting the usb cable

Connect the FPGA to the host USB I/F of the K8s node. Skip if already connected.

(5-2) Verify FPGAID on FPGA (connect with xsdb command)

Specify settings for executing the Vivado command. Not required if defined in ~/.bashrc in Section 4.1.4.

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

xsdb command starting console

```
$ 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 to an FPGA card with the connect command

```
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%
```

Execute the jtag target command. Verify that as many FPGAs are discovered in the server.

```
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
```

Detected FPGAID

(5-3) Write to discovered FPGAs

Success if "Flash programming completed successfully" is displayed

```
$ cd ~/hardware-drivers/tools/run_flash
$ ./run_flash.sh -t OpenKasugai-fpga-example-design-1.0.0-1.mcs -i 0
*Omitted
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 argument "-i" is the order in which the device was detected by the jtag target command. Specify“ 0” to specify the first detected FPGA device, and specify “1 ” to specify the second detected FPGA device. This time there is only one FPGA, so it is "0."

(6) Checking the FPGA Card Status

Verified `Device 903f` after writing MCS file and cold reboot.

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

4.2.2 Bitstream (child bs) writing with Mcap

Target: K8s nodes with FPGA installed (performed with Server #2 in the assumed environment)

Writing child bs to an FPGA is described in detail in Section 8.4.1.

5. Container management base setup

FIG. 5 shows the setting targets of this chapter in the overall software configuration.
This chapter installs the K8s-related components on each server and builds a K8s cluster.

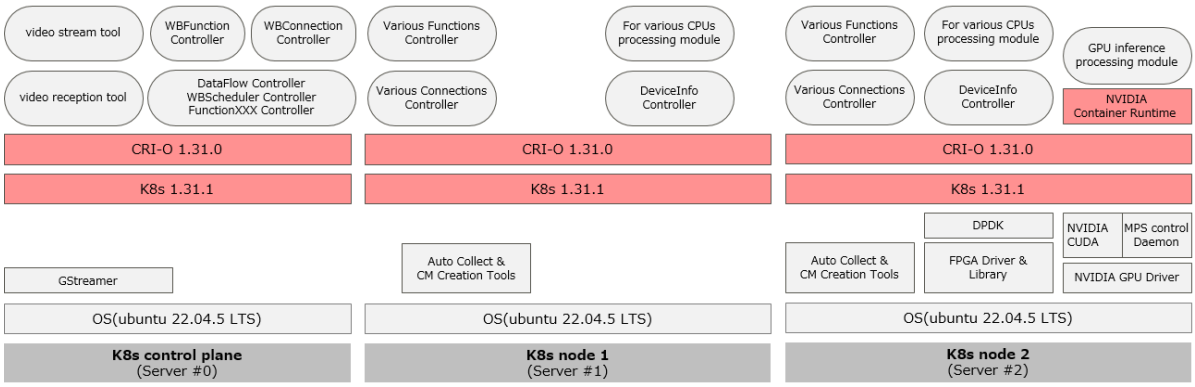

Figure 5 K8s-Related Components What to Install

The procedure flow required for the K8s control plane/K8s node in this chapter is listed below.

	K8s control plane	**K8s node**
5.1 Advance preparation	5.1.1 Various settings	
	5.1.2 Software installation	
	5.1.3 Iptables settings	
5.2 Installing K8s	5.2.1 Installing K8s 1.31.1	
5.3 Installing CRI-O	5.3.1 CRI-O v1.31.0 installation	
	—	5.3.2 NVIDIA container-toolkit install with GPU
	—	5.3.3 Editing the CRI-O configuration file (No GPU)
	5.3.4 Restart CRI-O	
5.4 Building K8s clusters	5.4.1 Download calico manifest	—
	5.4.2 Building K8s clusters	—
	5.4.3 Applying calico	—
	—	5.4.4 Join K8s Cluster
5.5 SR-IOV CNI plug-in setup	—	Steps 5.5.1 through 5.5.3
5.6 Installing Multus	Steps 5.6.1 through 5.6.3	—

5.1 Advance preparation

5.1.1 Various settings

Target: K8s control plane, all K8s nodes

1. If swap is enabled, kubelet will fail to start, so be sure to disable the swap region.

```
$ 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
```

Since it is re-enabled when rebooting, permanently disable swap by modifying/etc/fstab to:

```
$ sudoedit /etc/fstab

#/swap.img     none  swap  sw    0     0  (Comment out this line)
```

2. Set the host name. The host name is optional but should be descriptive.
Here, an example is shown in which the host name is set according to FIG. 1.

```
=== Example server for K8s controller plane component (Server #0) ===
$ hostnamectl set-hostname server0
$ hostnamectl
   Static hostname: server0
   (Omitted)
  Operating System: Ubuntu 22.04.5 LTS
(The rest is omitted)

=== Example of server for K8s node component1 (Server #1) ===
$ hostnamectl set-hostname server1
$ hostnamectl
   Static hostname: server1
   (Omitted)
  Operating System: Ubuntu 22.04.5 LTS
(The rest is omitted)

=== Example server for K8s node component 2 (Server #2) ===
$ hostnamectl set-hostname server2
$ hostnamectl
   Static hostname: server2
   (Omitted)
  Operating System: Ubuntu 22.04.5 LTS
(The rest is omitted)
```

3. Check the IP address to be set as the IP address of the K8s control plane of the K8s and the node of the K8s node, and set it in/etc/hosts in step 4) below.
   Here, an example is shown in which the host name is set according to FIG. 1.

```
$ ip addr show

=== Sample output from server for K8s controller plane component (Server #0) ===
  (Omitted)
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
 (hereinafter omitted)

=== Sample output from server for K8s node component1 (Server #1) ===
(Omitted)
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
 (hereinafter omitted)

=== Sample output from server for K8s node component2 (Server #2) ===
(Omitted)
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
 (hereinafter omitted)
```

4. Register the K8s control plane and K8s nodes (nodes in the cluster) in/etc/hosts according to the environment.
   Here, an example is shown in which the host name is set according to FIG. 1.

```
sudoedit /etc/hosts

10.38.119.101 server0
10.38.119.18 server1
10.38.119.20 server2
```

5.1.2 Software installation

Target: K8s control plane, all K8s nodes

1. If wget is not installed, install it.

```
$ sudo apt install wget
```

2. If git is not installed, install it.

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

```
$ git  config  --global  http.sslVerify  false
$ cat ~/.gitconfig
(Check if it is set)
```

5.1.3 Iptables settings

Target: K8s control plane, all K8s nodes

1. Load br_netfilter.

```
$ lsmod | grep br_netfilter
(Verified that br_netfilter is loaded. If not, run the following command)
$ sudo modprobe br_netfilter
```

2. Modify/etc/sysctl.d/k8s.conf to:

```
$ 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
(Check if it is set)
```

5.2 Installing K8s

5.2.1 Installing K8s 1.31.1

Target: K8s control plane, all K8s nodes

1. Edit the K8s repository.

```
$ 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. Install Required Components

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

3. Set kubelet to start automatically and instruct to start.

```
$ 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
(The rest is omitted)

(Active: Make sure it is activating, it is not running at this time)
```

5.3 Installing CRI-O

5.3.1 CRI-O v1.31.0 installation

Target: K8s control plane, all K8s nodes

1. Set the required kernel parameters.

```
$ sudo modprobe overlay
$ sudo modprobe br_netfilter

(Run as root)
$ 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. Set the CRI-O version to a variable. *Continue to run as root

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

3. Get Repository. *Continue to run as root

```
# 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 installation

```
# apt-get update

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

# exit
```

*If the CRI-O installation results in 404, apt-get clean and start over with apt-get update.

*If apt-get encounters an expired certificate error, run apt install ca-certificates and try again.

5. Configuring CRI-O

Change subnet if necessary so that it does not conflict with the subnet address used by other software.

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

    "ranges": [
        [{"subnet": "172.35.0.0/16"}], (change subnet address)
        [{ "subnet": "1100:200::/24" }]
    ]
```

In order to use calico as CNI, the bridge of CRI-O must be evacuated. (After calico starts, 10 calico.conflist is created under/etc/cni/net.d and overwrites the original file.)

```
$ 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/
```

(When building in a proxy environment, make the following settings in red (write on one line without breaking a line in the middle).)

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

[Service]
Type=notify
EnvironmentFile=-/etc/default/crio
Environment="HTTP_PROXY=http://$(user):$(password)@ (proxy server): (port)"
"HTTPS_PROXY=http://$(user):$(password)@ (proxy server): (port)"
"NO_PROXY=127.0.0.1, localhost, (host IP), 10.96.0.1"
(Remove the original Environment=GOTRACEBACK= crash)
```

5.3.2 NVIDIA container-toolkit installation with GPU

Target: K8s nodes with GPU (Server #2 in the assumed environment)

In this section, the procedure of this chapter is not necessary if GPU is not used (GPU-equipped server is not used).

(1) Install NVIDIA container-toolkit (includes NVIDIA Container Runtime)

```
$ 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 integration

(2-1) Generating and editing a configuration file for linkage

Create a configuration file for linkage

```
$ 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.
```

Edit `/etc/crio/crio.conf.d/99 nvidia.conf` (add a red line at the end)

```
$ 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"
```

(2-2) Editing the NVIDIA Container Runtime Side Configuration File (config.toml)

```
$ sudoedit /etc/nvidia-container-runtime/config.toml
. . .
(Edit the contents of the list as follows)
runtimes = ["/usr/libexec/crio/crun," "docker-runc," "runc," "crun"]

. . .
```

5.3.3 Editing the CRI-O configuration file (No GPU)

Target: K8s nodes without GPU　(Server #1 is used in the assumed environment)

There is no need to edit the CRI-O configuration file on servers that do not have different GPUs.

5.3.4 Start CRI-O

Target: K8s control plane, all K8s nodes

```
$ 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
(Verify that crio.service is active (running))
```

5.4 Building K8s clusters

5.4.1 Downloading and editing the calico manifest

Target: K8s control plane

1. Download the manifest for calico v3.28.1

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

2. Editing the retrieved calico.yaml

   Open the obtained calico.yaml and add "IP _AUTODETECTION_ METHOD."

```
$ vi calico.yaml
```

   Search with "autedetect" and add the contents of the red as follows immediately after.

```
# Auto-detect the BGP IP address.
- name: IP
  value: “autodetect”
# Set Auto-Detection Method of Network Interface *
- name: IP_AUTODETECTION_METHOD
  value: kubernetes-internal-ip
#Enable IPIP
- name: CALICO_IPV4POOL_IPIP
```

   * This is a comment line, so you do not need to add it.

5.4.2 Build K8s cluster with K8s control plane

Target: K8s control plane

Change the blue text of the kubeadm command below to suit your environment. The following are examples in this book.

--pod-network-cidr: Set appropriate values for your environment

--apiserver-advertise-address: IP address of the K8s control plane

Success if "Your Kubernetes control-plane has initialized successfully!" is printed.

```
 $ 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

 (Omitted)
 Your Kubernetes control-plane has initialized successfully!
 *Copy the following command (example) output for use in 2 of section 5.4.4
 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 Applying calico in the K8s control plane

Target: K8s control plane

```
$ kubectl apply -f calico.yaml
(Omitted)
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   ...
(After waiting for a while, it will be OK if it shows Running as shown above.)
```

5.4.4 Join K8s nodes to K8s cluster

Target: All K8s nodes

1. Preparing the K8s nodes to join

   Change blue letters as appropriate for the environment. The following are examples in this book.

```
$ 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 join using the command copied at the time of section 2

   *Add sudo at the beginning and --cri-socket=unix:///var/run/crio/crio.sock at the end

```
 $ 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
 (The above is an example of the procedure, use the copied command)

 $ sudo systemctl status kubelet
 (Ensure kubelet status is Active: active (running))
```

3. Verify cluster join and pod running

```
 $ 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   ...

(If it is not running after some time, restarting each Node may solve it.)
```

5.5 SR-IOV CNI plug-in setup

Of the DF processing modules deployed in the evaluation procedure in the separate document “OpenKasugai-Demo”, the processing module running on the Pod uses the VF of the SR-IOV created in the 100GNIC of the K8s nodes from the 2nd NIC added to the Pod to perform Ethernet communication. Therefore, the SR-IOV CNI plug-in, and Multus necessary for this are set up and installed in step 5.5~5.6.

Target: All K8s nodes

Set up the SR-IOV CNI plug-in to enable K8s pods to communicate using SR-IOV devices

5.5.1 Installing the Go language

Use the following command to install Go language

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

Use the following command to create the gopath directory

```
$ mkdir ~/gopath
```

Set gopath with the following command

*The following settings are in ~/.bashrc.

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

5.5.2 Obtaining the SR-IOV CNI plug-in

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

5.5.3 Building the SR-IOV CNI plug-in

```
$ 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
```

If "make build" fails with "Command 'make' not found, .." install dpkg-dev.

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

5.6 Installing Multus

Target: K8s control plane

Install Multus to create a 2nd NIC on each processing module pod in DF

5.6.1 Getting Multus

1. Clone the GitHub repository for Multus

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

2. Edit the manifest for Multus

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

---
resources:
   requests:
     cpu: "400m" (set the value to "400m")
     memory: "200Mi" (value set to "200Mi")
   limits:
     cpu: "400m" (set the value to "400m")
     memory: "200Mi" (value set to "200Mi")
```

*If OOM killed occurs in Pod, Multus, adjust the above settings accordingly. Setting limits to any multiple of requests may eliminate OOM killed.

5.6.2 Apply Multus manifest (deployed as DaemonSet)

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

5.6.3 Creating and applying a Manifest for a NetworkAttachmentDefinition

1) Creating a Manifest for a NetworkAttachmentDefinition

In Section 8.9, we will create a VF for each K8s node's 100GNIC.

Create a Manifest of the NetworkAttachmentDefinition with the following contents for the number of K8s nodes where the NICs on which the SR-IOV VF is to be created exist (in the assumed environment, two NICs, Server #1 and Server # 2).

Here is an example of configuring a manifest for Server # 1:

```
$ cd ~/
$ vi server1-config-net-sriov.yaml # Any yaml file name
---
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"
  }
}'
```

*Specify the host name of the target K8s node in the “server1” part of metadata.name.metadata.namespace is the same namespace as the DataFlow namespace deployed in the evaluation procedure in the separate document “OpenKasugai-Demo”.
k8s.v1.cni.cncf.io/resourceName is the resourceName on the ConfigMap of the SR-IOV device plug-in.
Specify the host name of the target K8s node in the "server1" part of spec.name.

2) Applying the Manifest of the NetworkAttachmentDefinition created in 1) above
Here is an example of the application of manifest for Server # 1:

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

*If Network Attachment Definitions for multiple K8s nodes were created in 1) above, apply them all (In the expected environment, it also applies the manifest for Server #2.).

6. GPU setup

This chapter is only for GPUs. If GPUs are not used (GPU-equipped servers are not used), the procedure in this chapter is not necessary.

6.1 Installing the NVIDIA GPU driver

Target: K8s nodes with GPU　(Server #2 in the assumed environment)

FIG. 6.1 shows the setting targets of this chapter in the overall software configuration.
In this chapter, we will install the GPU driver for the GPU-equipped K8s nodes.

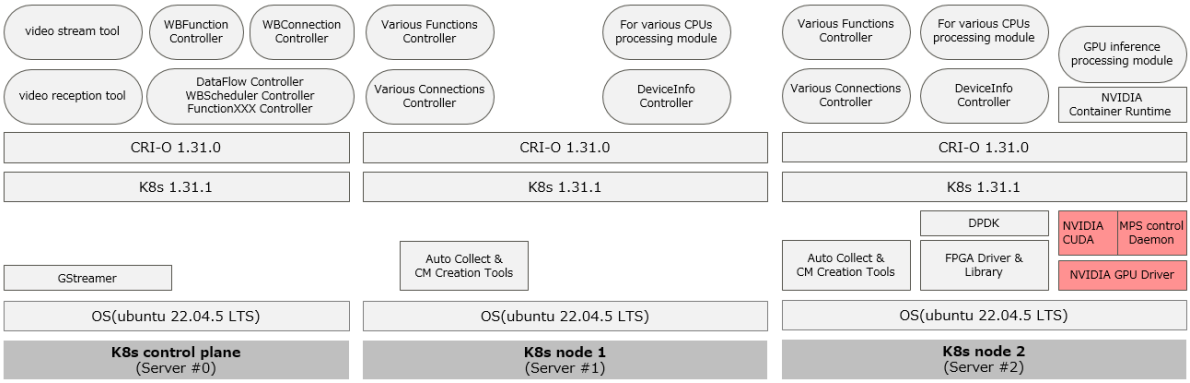

Figure 6.1 GPU Driver Installation Target

1. Download the run file
   Access https://www.nvidia.com/Download/Find.aspx?lang=en-us, specify "Product Type" as shown in Fig. 6.2, and press the "Search" button. Download the run file from the link with Version "550.90.12" in the search results.

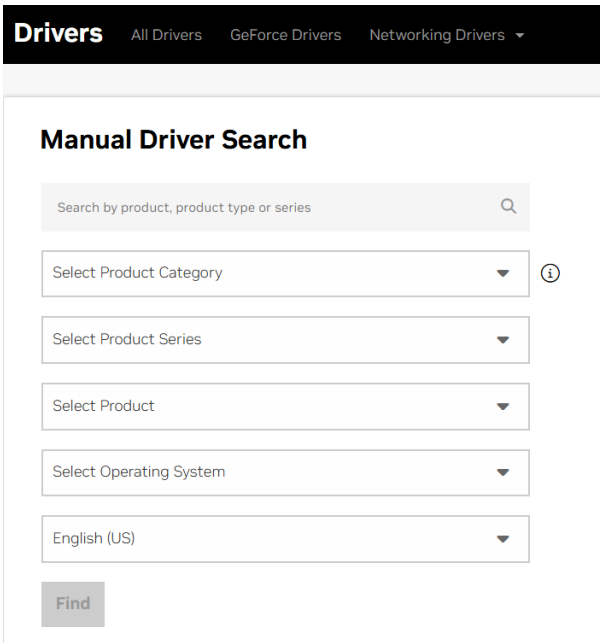

Figure 6.2 NVIDIA Driver Download Site

2. Run the NVIDIA driver file (NVIDIA-Linux-x86_64-550.90.12.run) in your home directory and check for updates.

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

$ nvidia-smi
(Check driver version 550.90.12)
```

If you get the following ERROR message when running the NVIDIA GPU driver file, you must disable the Nouveau kernel driver.

```
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.
```

If you get the above ERROR, disable the Nouveau kernel driver and run the NVIDIA GPU driver file again by running the following command:

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

3. Reboot the OS

```
$ sudo reboot
```

6.2 Starting the MPS control daemon

Target: K8s nodes with GPU

1. Adding settings to/root/.bashrc

Add the following to/root/.bashrc (example):

```
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
```

- For CUDA_DEVICE_ORDER="PCI_BUS_ID ":
  Set the ID specified in CUDA_VISIBLE_DEVICES so that it can be checked by nvidia-smi.
- About CUDA_VISIBLE_ DEVICES:
  Of the GPU devices installed in the target K8s node, specify the ID of the device to be used by the MPS.
  By specifying "CUDA _DEVICES_ORDER="PCI_BUS_ID," you can specify an ID that can be verified by nvidia-smi (0, 1, ...).
  It can be omitted when all GPU devices installed in the target K8s node are used.

2. Starting and checking the MPS control daemon

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

$ ps aux | grep mps
Check that the following control daemon is displayed:
root  783951  0.0  0.0  76472   140 ? Ssl  10:00   0:00 nvidia-cuda-mps-control -d
```

3. Configuring the MPS Control Daemon to Start Automatically
   Configure the MPS control daemon to run automatically at boot time so that you do not need to run MPS each time you reboot. First, create the following "MpsAutoStart.service" file.

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

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


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

   Then, as root, move the above files to the/etc/systemd/system directory and enable the service.

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

7. Setup of various processing modules

FIG. 7 shows the setting targets of this chapter in the overall software configuration.
In this chapter, you set up the various processing modules for evaluation.

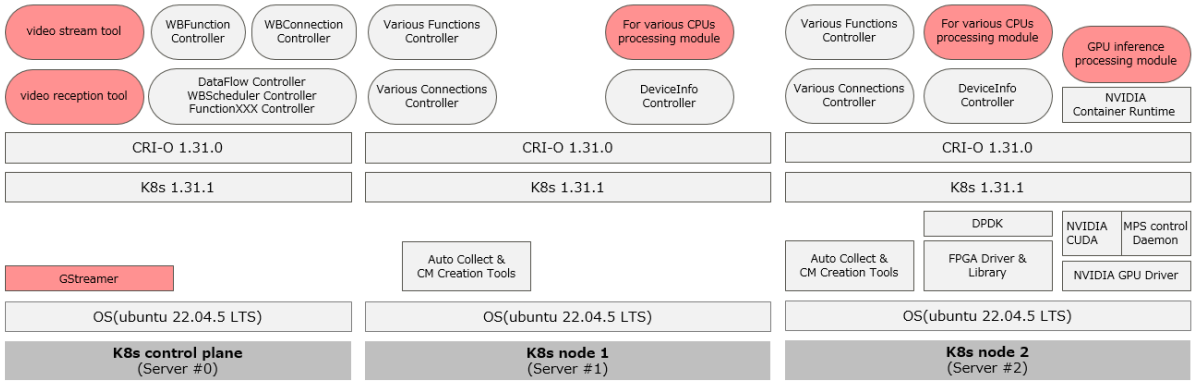

Figure 7 Processing Module for Setup

The flow of setup of various processing modules and video distribution tools to be performed in this chapter is described below.

	K8s control plane	K8s node
7.1 Advance preparation	○	
7.2 GPU inference processing module setup	—	○(with GPU)
7.3 Setting up the CPU decoding module	—	○
7.4 Setting up the CPU filter resize processing module	—	○
7.5 Setting up the CPU copy branch processing module	—	○
7.6 Setting up the CPU Glue processing module	—	○
7.7 Setting up the video distribution tool	○	—
7.8 Preparing the video for the OpenKasugai Demo	○	—

7.1 Advance preparation

Target: K8s control plane, all K8s nodes

7.1.1 Changing and restarting CRI-O settings

1. Use the following command to install buildah

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

2. Modify /usr/share/containers/containers.conf.

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

# [machine] (comment out [machine])
```

3. Modify /etc/containers/registries.conf to add container registry information and restart CRI-O.

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

$ sudo systemctl restart cri-o
```

*If a container registry other than "docker.io" or "quay.io" is used, specify it in the same way as "docker.io" or "quay.io" above.

7.1.2 Acquisition of material (FPGA library drivers, controllers, and sample apps)

Store the source (controller) in your home directory.

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

7.2 GPU inference processing module setup

This section is performed only when using GPUs. If GPUs are not used (GPU-equipped servers are not used), the procedure in this chapter is not necessary.

Temporarily disable MPS because it can cause GPU processing modules to fail to build when it is enabled.

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

7.2.1 GPU inference processing module (FPGA-enabled version) setup

Target: K8s nodes with GPU

1. Use the following steps to build a container

```
$ 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
```

*Be aware that the build takes a very long time

*Run on a server equipped with the GPU (T4 or A100) that this application will use.

7.2.2 Setting up the GPU inference processing module (TCP-Enabled)

Target: K8s nodes with GPU

1. Use the following steps to build a container

```
$ 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 ../../../../../
```

*This app does not include an FPGA library

*Run on a server equipped with the GPU (T4 or A100) that this application will use.

Make sure that the above GPU-based processing modules have been built.

```
$ 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
```

Enable the MPS that was disabled at the beginning of Section 7.2.

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

$ ps aux | grep mps
Check that the following control daemon is displayed:
root  783951  0.0  0.0  76472   140 ? Ssl  10:00   0:00 nvidia-cuda-mps-control -d
```

7.3 Setting up the CPU decoding module

Target: All K8s nodes

1. Use the following steps to build a container

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

7.4 Setting up the CPU filter resize processing module

Target: All K8s nodes

1. To build the container:

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

7.5 Setting up the CPU copy branch processing module

Target: All K8s nodes

1. To build the container:

```
$ 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 Setting Up the CPU Glue processing module

Target: All K8s nodes

1. Use the following steps to build a container

```
$ 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
```

Confirm that the above CPU processing modules have been built.

```
$ 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 Setting up the video distribution tool/video reception tool

Target: K8s control plane

Editing the build specification in the Video stream tool build file with the editor

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

Editing the build specification in the Video reception tool build file with the editor

*Be aware that the build takes a very long time

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

Verify that the distribution reception tool image is built

```
$ 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 Preparing the video for the OpenKasugai Demo

Target : K8s control plane

Acquire video movies from the URL described in section 0.5 “11 Movie for Demo”, and store them into /opt/DATA/video/ directory.

Create the/opt/DATA/video directory if it does not exist.

```
$ sudo mkdir -p /opt/DATA/video   *Run if/opt/DATA/video is missing
```

When all the movies are acquired, 2 movies are stored in the directory as follows.

```
$ 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
```

In the test conducted in the separate document "OpenKasugai-Demo”, the video should be 4K and 15fps or less, preferably 30 to 60 seconds in length.

None of the above videos fit the criteria, so they need to be edited.

Here, the procedure for adjusting "6896028-uhd_3840_2160_30fps.mp4" of the above moving images to the above conditions is described.

The video is about 46 seconds at 4 K, 30fps, so you can change the frame rate to 15fps. Here, ffmpeg is used for video editing. If not, install it first.

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

*"input_4 K_15fps.mp4" is the file name of the edited video, which is used as an input video in the separate document “OpenKasugai-Demo”.

"46098-447095422_small.mp4" is full HD, 25fps, 30 second video, so if you want to use it, you'll need to change the resolution and frame rate.

8. Controller setup

FIG. 8 shows the setting targets of this chapter in the overall software configuration.
In this chapter, you install various controllers for data flow deployment (Custom Resource Controllers (CRC)), run tools that automatically collect the necessary settings for the controllers, and prepare the FPGAs.

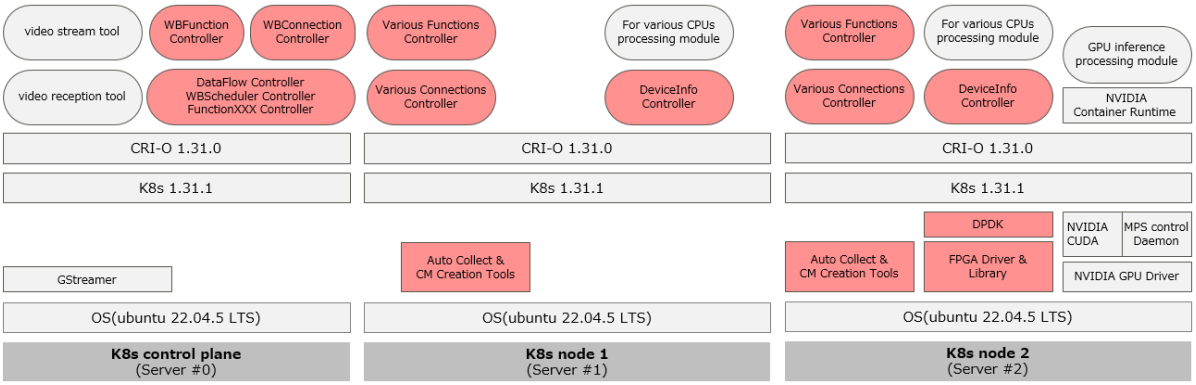

Figure 8 Setup Target Components

In this chapter, the flow of necessary procedures for the K8s control plane/K8s node is listed below.

	K8s control plane	K8s node
8.1 Advance preparation	8.1.1 Installing the Go language	
		8.1.2 Building the FPGA library
	—	8.1.3 Building the FPGADB library
8.2 Building the CRC container	8.2.1 K8s control plane side build	8.2.2 Build on the K8s node side
8.3 Preparing the CRC to start	—	○
8.4 Preparation for automatic collection and creation of ConfigMap	—	8.4.1 FPGA Bitstream write
	—	8.4.2 Installing DCGM
8.5 Preparing input data for creating various types of information (ConfigMap)	8.5.1 Editing input data (environment-dependent data)	
	—	8.5.2 Aggregation of input data
8.6 Acquisition and deployment of various types of information (external information, ConfigMap)	—	○
8.7 FPGAReconfigrationTool builds	—	○
8.8 FPGAClearCheckTool builds	—	○
8.9 8.9 SR-IOV VF creation and management	8.9.3 SR-IOV device plug-in setup 8.9.4 Running the SR-IOV device plugin	8.9.1 Creating a VF to a 100GNIC 8.9.2 Create VF

The roles of each CRC and a list of starting servers are described below.

CRC Name	Role	Startup Server	
		K8s control plane	K8s node
FunctionType	Makes available the registered custom resource of FunctionType (CR).	○	
FunctionChain	Enables registered FunctionChain CR.		
FunctionTarget	Produces FunctionTarget CRs based on the ComputeResource CR.		
DataFlow	Creates and deletes WBFunction CRs and WBConnection CRs based on the DataFlow CR.		
WBscheduler	Determines the where each WBFunction CR and WBConnection CR of the DataFlow CR is deployed, and set results in the DataFlow CR.		
WBFunction	Identifies FunctionKind(GPUFunction, FPGAFunction, or CPUFunction) of each Function CR based on each WBFunction CR and creates the identified CR, also deletes the WBFunction CR.	○	
WBConnection	Identifies ConnectionKind(EthernetConnection or PCIeConnection) of each Connection CR based on each WBConnection CR and creates the identified CR, also deletes the WBConnection CR.	○	
GPUFunction	Launches and deletes Gstreamer plug-in containers (with inference apps) based on the GPUFunction CR.		○
FPGAFunction	Writes child bs to specified FPGA device. Also allocates and recovers resources in the FPGA based on the FPGAFunction CR.		○
CPUFunction	Launches and deletes plug-in containers for Gstreamer (with decoding app) based on the CPUFunction CR.		○
Ethernet Connection	Controls a route of the Ethernet network for video distribution processing based on the EthernetConnection CR.		○
PCIeConnection	Controls a route of the internal network (PCIe Path) for video distribution processing based on the PCIeConnection CR.		○
DeviceInfo	Generates an initial ComputeResource CR when building the system, also updates ComputeResource as each DataFlow CR is deployed or removed.		○

8.1 Advance preparation

To create a container image for each CRC, we set up a Go language environment and show how to create and register images.

8.1.1 Installing the Go language

Target: K8s control plane, all K8s nodes

*If done for K8s node on section 5.5.1, only done for K8s control plane

Use the following command to install Go language

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

Use the following command to create the gopath directory

```
$ mkdir ~/gopath
```

Set gopath with the following command

*Put the following settings in ~/.bashrc.

```
$ vi ~/.bashrc
Add the following
export GOPATH="$HOME/gopath"
export GOROOT="$HOME/go"
export PATH="$GOROOT/bin:$PATH"

$ source ~/.bashrc
```

8.1.2 Building the FPGA library

Target: All K8s nodes

1. Get Source (FPGA library)
   If not, perform section 7.1.2 to obtain the source.

2. Install the required packages (skip if already installed)

```
$ 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. Building the DPDK
   Enter the following command to build:

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

4. Building and installing MCAP tools
   Type the following command to build and store the mcap tool in the directory passed by sudo's PATH environment variable: (* In the following example,/usr/local/bin is selected after checking with printenv.)

```
$ 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. Get JSON source file
   Enter the following command to get the JSON source file:

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

6. Building the FPGA library
   Type the following command to build the FPGA library:

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

7. Copy the header file of the FPGA library to the specified directory
   Copy the following directories into/usr/local/include/fpgalib: (If the/usr/local/include/fpgalib directory does not exist, use mkdir to create it.)

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

8. Copy the library body of the FPGA library to the specified directory.
   Copy the following into/usr/local/lib/fpgalib: (If the/usr/local/lib/fpgalib directory does not exist, use mkdir to create it.)

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

9. Link fpga-software/lib/DPDK/dpdk to/usr/local/lib/fpgalib/dpdk

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

8.1.3 Building the FPGADB library

Target: All K8s nodes

1. Building the FPGADB Library
   Type the following command to build the FPGADB library:
   (Build the 8.1.2 FPGA library in advance)

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

2. Copy the header files of the FPGADB library to the specified directory
   Copy the following directories (Directory containing header files required for using the FPGADB library under fpgadb/build) into/usr/local/include/fpgalib:

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

3. Copy the library body of the FPGADB library to the specified directory.
   Copy the following (libfpgadb.a under fpgadb/build) to/usr/local/lib/fpgalib:

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

8.2 Building the CRC container

●Assumptions

Select docker.io/library/golang: 1.23 if the following options appear in the subsequent controller container image creation:

```
[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 side build

Target: K8s Control Plane

(1) Add the shortname “golang ” to/etc/containers/registries.conf.d/shortnames.conf.

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

(2) Build the DataFlow, WBscheduler, FunctionTarget, FunctionType, and FunctionChain controllers.

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

If "make dokcer-build ..." fails with "Command 'make' not found, .," install dpkg-dev as follows and run "make dovker-build ..." again.

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

(3) Build the WBFunction controller.

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

(4) Build the WBConnection controller.

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

8.2.2 Build on the K8s node side

Target: All K8s nodes

(1) Add the shortname "golang " to/etc/containers/registries.conf.d/shortnames.conf.

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

(2) Building the DeviceInfo Controller

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

(3) Building the PCIeConnection Controller

```
$ 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) Building an EthernetConnection Controller

```
$ 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) Building the FPGAFunction Controller

```
$ 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 ..
```

If "OpenKasugai-fpga-example-design-1.0. 0-2. bit" is not available, git clone hardware-design repository and copy it again.

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

(6) Building GPUFunction controllers

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

(7) Building the CPUFunction Controller

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

8.3 Preparing the CRC to start

Target: All K8s nodes

1. Specify the settings required for CRC operation.
   In the case of a multiple K8s nodes configuration, perform this on each K8s node.

```
$ 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
```

   The following steps need only be performed once on any K8s nodes (not necessarily on each K8s node).

2. CRD Registration for DeviceInfo

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

3. CRD registration for PCIeConnection

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

4. CRD Registration for EthernetConnection

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

5. CRD Registration for FPGAFunctions, FPGAs, and ChildBs, and FPGAReconfiguration

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

6. CRD Registration for GPUFunction

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

7. CRD registration for CPUFunction

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

8.4 Preparation for automatic collection and creation of ConfigMap

8.4.1 FPGA Bitstream write

Target: All K8s nodes with FPGA

This chapter is performed only when FPGAs are used. If an FPGA is not used (FPGA-equipped server is not used), the procedure of this chapter is not necessary.

It is necessary to write Bitstream (child bs) in advance in order to acquire FPGA information by automatic collection and CM creation function.

(1) Write child bs

Since a BIT file corresponding to the MCS file used in section 4.2.1 is required, section 4.2.1 must be implemented in advance. This section assumes that the built BIT file under hardware-design is used. Also, if a BIT file that does not correspond to a written MCS file is written by mistake, perform section 4.2.1 again.

If you did not store the mcap in the build and sudo path in section 8.1.2, perform section 8.1.2 beforehand.

In this procedure, writing is performed by using the mcap tool, but since the -x (PCI device ID specification) and -s (BDF specification) options in the mcap command are environment-dependent values, it is necessary to change them according to the environment to be constructed.

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

```
$ lspci |grep Xilinx
xx:xx.x Processing accelerators: Xilinx Corporation Device 903f
1f:00.0 Memory controller: Xilinx Corporation Device 903f
...
```

Write child bs using the mcap command. It takes about 15 seconds to write.

*This procedure must be performed after each system boot and reboot.

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

If the write is successful, the following log is displayed:

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

(2) Installing the FPGA Driver

Load the FPGA driver from hardware-drivers on github.

This section must be followed by (1).

*When the OS is restarted, it is necessary to perform the procedure from (1) again.

If the xpcie driver is already loaded, unload it. If there is no lsmod output, it is not loaded.

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

Build and load the driver.

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

~ Images of ls/dev/xpcie* Results ~
/ dev/xpcie_ [UUID of FPGA]
```

8.4.2 Installing DCGM

Target: All K8s nodes with GPU

This chapter is only for GPUs. If GPUs are not used (GPU-equipped servers are not used), the procedure in this chapter is not necessary.

The automatic collection and CM creation function utilizes NVIDIA DCGM [*1] to acquire GPU information. Therefore, DCGM must be installed.

*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 Preparing input data for creating various types of information (ConfigMap)

Target: All K8s nodes

In order to run CRC, it is necessary to create a ConfigMap of the various information used by these CRCs. In order to create a ConfigMap, information about the infrastructure needs to be provided as input data. This section describes the procedure for preparing the input data required to create a ConfigMap.

The input data is classified into the following A and B.

A. Data common to all DFs:

For example use cases, a file that basically allows you to use the supplied material as is.

The target files are as follows:

File name	Description	Remarks
devicetypemap.json	Map information to convert device name (model) to DeviceType	*1
region-unique-info.json	region-specific information	*1
functionkindmap.json	Used to determine the CR that matches the device type	
connectionkindmap.json	Used to determine the CR that matches the connection type	
function-unique-info.json	More information about Functions	
filter-resize-childbs.json	Internal CH information for FPGA Filter/Resize	
functionnamemap.json	Judges whether FPGAFilter/Resize is an advanced or lightweight inference	
premadefilelist.json	Link files needed by automatic collection and CM creation function with internal structures	
gpufunc-config-high-infer.json	Config information for advanced inference GPUFunction	
gpufunc-config-low-infer.json	Config information for lightweight inference GPUFunction	
fpgafunc-config-filter-resize-high-infer.json	Config information for advanced inference filter/resizeFPGAFunction	
fpgafunc-config-filter-resize-low-infer.json	Config information for lightweight inference filter/resizeFPGAFunction	
cpufunc-config-decode.json	Config Information for Decode CPUFunction	
cpufunc-config-filter-resize-high-infer.json	Config Information for Advanced Inference filter/resize CPUFunction	
cpufunc-config-filter-resize-low-infer.json	Config Information for lightweight Inference filter/resize CPUFunction	
cpufunc-config-copy-branch.json	Config information for copy branch CPUFunction	
cpufunc-config-glue-fdma-to-tcp.json	Config information for GlueCPUFunction	

The storage location of the sample data is "~/controller/test/sample-data/sample-data-common/."

As an exception, when using devices other than the four types of devices used in the assumed environment (Alveo U250, NVIDIA GPU T4, NVIDIA GPU A100, Intel(R) Xeon(R) Gold 6346 CPU

@ 3.10 GHz, Intel(R) Xeon(R) Gold 6348 CPU @ 2.60 GHz), it is necessary to add a note to the file marked “*1” in the remarks column of the above table. For the method of adding, refer to the description of the relevant file in "3. Description of the input data (JSON) used to create CM" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1."

B. DF-dependent data:Files that need to be edited to match the content of individual DFs, where they are deployed, and so on.

The target files and directories are as follows.

file name, directory name	Description	Remarks
~/controller/test/ sample-data/sample-data-demo/json/ predetermined-region.json	A file that defines the RegionType for each region in the system.	*2

*2. A file that defines the area type of each area in the "Lane fixed method," in which the area type of each system is predetermined beforehand.

For the parts that need to be changed from the sample data and how to change them (which values should be written), refer to the sheet "3. Explanation of the input data (JSON) used for CM creation" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1”. (The policy of the change method is described in the remarks column.)

8.5.1 Editing input data (environment-dependent data)

Target: All K8s nodes

Edit predetermined-region.json, which is the sample data corresponding to B above, according to the implementation environment. The base files are:
~/controller/test/sample-data/sample-data-demo/json/predetermined-region.json

The contents to be set in the corresponding file are described below. The contents of the relevant file and the values to be defined for each parameter are also described in "B. Files that need to be edited from the supplied material" in the "3. Explanation of the input data (JSON) used for CM creation" sheet in the separate document "OpenKasugai-Controller-InstallManual_Attachment1", so it is recommended to refer to them as appropriate.

The predetermined-region.json contains information about each region created on the device (FPGA, GPU, CPU) installed in all K8s nodes. When the FPGA circuit described in Section 0.4 is used, there are two regions on each FPGA, so that two regions are described for each FPGA device. Further, since it is assumed that the region on the GPU/CPU device is one, one region for each CPU/GPU is described.

The values described in each region are as follows.

- nodeName: The node name of the node where the region is located.
  - Specifies the node name of the K8s node that contains the device.
- deviceUUID — Identifies the device on which the region is located.
  - For FPGA: Use the results of the "ls/dev/*" command.
    After installing the FPGA driver in Section 8.4.1, the FPGA device "xpcie _${FPGA-ID}" is displayed. Enter the value of ${FPGA-ID}.
  - For GPU: Use the result of the "nvidia-smi-L" command.
    Because UUID(Example: GPU-b8b4f1f5-bf51-eaa3-6ec4-97190b7f6c98) is output for each device, enter its value.
  - For CPU: Specify "node name" + "-cpu0."
    (Currently, each server is regarded as one virtual server, so a fixed values after the node name are acceptable.)
- subDeviceSpecRef: Region name of the region
  - For FPGA: Enter "${lane number}."
  - For CPU/GPU: Specify the same value as deviceType.
- regionType: Region type of the region
  - For FPGAs: Use the following format.
    ""${device type}" + "-" + "${parent bs-id}" + "-" + "${number of lanes}" + "lanes" + "-" + "${number of NICs}" + "nics""
    *If the FPGA circuit described in Section 0.4 is used, it is "alveou 250-0100001c-2lanes-0 nics."
  - For CPU/GPU: Specify the same value as deviceType.

8.5.2 Aggregation of input data

Target: All K8s nodes

The automatic collection and CM creation tool executed in Section 8.6 assumes that all the data in A and B above are collected in a specific directory (~/controller/src/tools/InfoCollector/infrainfo /) for each K8s node. Therefore, the input data is collected under ~/controller/src/tools/InfoCollector/infrainfo.
The following is an example of data creation in an assumed environment.

```
# Aggregation of above A (data that can be used commonly in all DF)
$ cp -r ~/controller/test/sample-data/sample-data-common/json/*
  ~/controller/src/tools/InfoCollector/infrainfo/.

# Aggregation of above B (DF dependent data)
$ cp -r ~/controller/test/sample-data/sample-data-demo/json/*
  ~/controller/src/tools/InfoCollector/infrainfo/.
```

8.6 Acquisition and deployment of various types of information (external information, ConfigMap)

Target: All K8s nodes

Deploy a ConfigMap of various information used by the CRC.

1. Delete old ConfigMap

*In the case of a multiple K8s nodes configuration, it should be run only once on any K8s nodes (not necessarily on each K8s node).

```
$ 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

*If you have a ConfigMap other than kube-root-ca.crt, delete it all using
$kubectl delete cm <ConfigMap name other than kube-root-ca.crt>
```

*1. If you delete the existing ConfigMap, but build it from the OS installation (from Chapter 3 of this document), at this point, there is no ConfigMap except kube-root-ca.crt, so the following error is output in all cases, but you can ignore it.
"Error from server (NotFound): configmaps" *ConfigMap name to delete* "not found

2. Deploying ConfigMap

*In the case of a multiple K8s nodes configuration, this should be done only once on any K8s nodes (not necessarily on each K8s node).

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

* If the Namespace "test01" has already been created, the following error will be output when "./k8s-config.sh create" is executed, but it can be ignored.
"Error from server (AlreadyExists): namespaces "test01" already exists"

3. Launch Automatic Collection & CM Creation Tool

* For multiple K8s nodes configurations, this must be done on each K8s node

```
$ 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 *
```

*In case of K8s Node without NVIDIA GPU, since DCGM is not installed, the following error message is output in the InfoCollector log, but it can be ignored.
"INFO infocollect/infocollect.go:405 dcgm.Init() Error but Maybe there are NOT any GPU. {"error": "libdcgm.so not Found"}"

8.7 Building FPGAReconfigrationTool (FPGAReconfigurationTool)

Target: Any K8s nodes with an FPGA card

Build FPGAReconfigrationTool so that people can write child bs to the FPGA at any time. The following three types of writing are possible by using this tool.

●FPGA Bitstream write: Write child bs to FPGA in child bs unwritten state (same state as immediately after Section 8.4.1)

●Child Bitstream reset: Returns child bs in use by the FPGA specified by the argument to the state it was just written to.

●FPGA reset: Returns the FPGA specified in the argument to child bs unwritten state (the same state as immediately after the implementation of section 8.4.1).

To build FPGAReconfigrationTool, do the following:

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

8.8 Building FPGAClearCheckTool (FPGAClearCheckTool)

Target: Any K8s nodes with an FPGA card

OpenKasugai-Controller v1.1.0 adds the ability to recover various resources in the FPGA used by DataFlow being deleted when DataFlow is deleted. With the addition of this function, the recovered resources in the FPGA can be reused except for some resources in the FPGA (resources in the Function). *Resources inside the Function (resources inside filter/resize circuit in the case of child bs for filter/resize) are not subject to collection because state and setting information may remain depending on the logic of the Function.)

Here, we build a tool (FPGAClearCheckTool) that is used to confirm whether various resources in the FPGA used by DataFlow in question were actually recovered when DataFlow was deleted according to the separate document "OpenKasugai-Demo".

The FPGAClearCheckTool consists of the following two tools:

a) fpga-chk-connection: Call the FPGA library to check the state of the resource in the FPGA specified by the argument

b) FPGACheckPerDF: Identify the resources in the FPGA used by the deleted DataFlow and its DataFlow, and execute fpga-chk-connection to check the recovery status of the resources in the FPGA in DF units.

The build process is as follows:

1. Build fpga-chk-connection: Type the following command to build the fpga-chk-connection tool: (Build the 8.1.2 FPGA library in advance)

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

2. Building FPGACheckPerDF

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

8.9 SR-IOV VF creation and management

In order to use the VF of the SR-IOV created in 100GNIC to perform Ethernet communication between each Pod in DataFlow, VF is created for all 100GNIC in each K8s node with 100GNIC. After the VF is created, K8s control plane obtains the SR-IOV device plug-in and executes it as daemoset, so that the created VF is recognized as a resource available on each K8s node.
*If the OS is restarted, re-implement sections 0 and 0.4.

8.9.1 Creating a VF to 100 Giga NIC used

Work on: All K8s nodes

The settings vary depending on the vendor of the 100G NIC in the target server. Here are the settings for the Intel 100GNIC and the Mellanox 100GNIC.

●For Mellanox 100G NICs (here is the expected K8s node)

1) Download the MFT binaries (mft-4.30. 0-139-x86_64-deb.tgz) from the NVIDIA Mellanox Firmware Tools page (https://network.nvidia.com/products/adapter-software/firmware-tools/) and place them in your home directory.

```
$ 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
```

If the following error is output when running install.sh, install dkms according to the output, and then run install.sh again.

"-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) Download the MLNX_OFED binaries (MLNX_OFED_LINUX-24.10-1.1.4.0-ubuntu22.04-x86_64.tgz) from NVIDIA's Linux Drivers page at https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/ and place them in your home directory.

The PCI address of the NIC (red part below) is specified in a series of operations, so know it beforehand.

```
$ 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 # See (b) if the result is different from
MST modules:
------------
    MST PCI module loaded
    MST PCI configuration module is not loaded
~ Omitted ~
# mstconfig -d ae:00.0 set SRIOV_EN=1 NUM_OF_VFS=8
# reboot
```

・If the following error occurs in (a) above,

```
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
```

Just delete mft-autocomplete and redo (a) as shown.

・If is the following in the output result of (b) above,

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

PCI Devices:
~ Omitted ~
```

Additional commands need to be executed as follows:

```
# modprobe mst_pci # Additional Commands
# mst status
MST modules:
------------
    MST PCI module loaded
    MST PCI configuration module is not loaded
~ Omitted ~
```

If you execute # mst status after reboot, the message "MST PCI module is not loaded" may appear, but no action is required and you can proceed to the next step.

3) Reimplement Section 8.4.1 with this K8s node rebooting.

●If the target NIC is an Intel 100 GNIC:

No steps. Proceed to Section 8.9.2.

8.9.2 Create VF

Work on: All K8s nodes

*If the OS is restarted, re-execute this section and the SR-IOV device plug-in.

The number of VFs to be created must satisfy the number of VFs used by the DataFlow to be deployed. Basically, it is sufficient to set a value equal to or greater than the total number of VFs used by DataFlow which can be simultaneously deployed. Specifically, the number of VFs may be created by multiplying the number of VFs used in the DataFlow for which the maximum number of VFs is used by the maximum number of DataFlows that can be deployed simultaneously.
The number of VFs used in each DataFlow corresponds to the number of Pod with Ethernet connections. For example, in the case of "A) Flow using FPGA F/R (Filter Resize) (FIG. 1) " deployed in chapter 1 in the separate document "OpenKasugai-Demo", the number of VFs used is two because the Decode Pod for CPUFunction and the Reasoning Pod for GPUFunction are Pod through Ethernet connection (Decode Pod to Video stream tool, Inference Pod to Video reception tool to Ethernet).
The number of VFs to be created is 40 based on the following assumptions.

- Maximum number of simultaneous deployment DataFlow is 8
- The maximum number of VFs is used. DataFlow uses 5 VFs in the "copy branch Flow (Fig. 16)" deployed in Chapter 5 in the separate document "OpenKasugai-Demo"
- Decoding for CPUFunction Pod, filter/resize Pod, copy branch Pod, Because each of the two inference Pod for GPUFunction uses VF)
- The maximum number of VFs used is when 8 "C) copy branch Flow" are deployed simultaneously.

●For Mellanox 100G NICs (here is the expected K8s node)

1) Create a VF on the target 100GNIC

Change ens8np0 in the following command to match the interface name of the target 100GNIC.

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

2) Verify that the number of VFs specified in 2 above have been created on the target 100GNIC.

```
$ 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
    (Omitted)
    vf 6     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
…
```

●For Intel 100G NICs

1) Create a VF on the target 100GNIC

Change ens93f0 in the following command to match the interface name of the target 100GNIC.

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

2) Verify that the number of VFs specified in 2 above have been created on the target 100GNIC.

```
# 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 Device Plug-in Setup

Subject: K8s control plane

Set up the SR-IOV device plug-in to work with SR-IOV devices on the K8s

(1) Obtaining the SR-IOV Device Plug-in

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

(2) Apply SR-IOV device plug-in ConfigMap

(1)Apply configMap. yaml included in the SR-IOV device plug-in obtained in.

In configMap. yaml, the management name, vendor code, device driver, and device code information of the NIC for which the SR-IOV VF is to be created must be described.

●Using the Mellanox 100GNIC used in the assumed environment in this document

Since the default configMap. yaml does not include a description for Mellanox 100GNIC, it is necessary to add a description for Mellanox 100GNIC in configMap. yaml before applying.

*Not only Mellanox 100GNIC but also NIC not listed in configMap. yaml can be used by following the same procedure.

・Acquisition of necessary information

First, obtain information necessary for adding. On the server (K8s Node) that has the NIC that creates the VF of the SR-IOV, obtain the information described in vendors and devices of the NIC.

- Vendor information and device information (both physical and virtual devices): Obtain with the following command:

```
 $ lspci -nn | grep Mellanox
 -Output example-
 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]
 (Omitted)
 ae:00.7 Ethernet controller [0200]: Mellanox Technologies MT27800 Family
[ConnectX-5 Virtual Function] [15b3:1018]
```

- driver information: Obtain the interface name of the NIC and use the following command to obtain it.

```
$ ethtool -i ens8np0  #ens8np0 is the interface name of the NIC
-Output example-
driver: mlx5_core
--Omitted --
```

・configMap.Edit yaml

Edit (add) configMap. yaml based on the information obtained above.

```
$ cd ~/sriov-network-device-plugin/deployments
$ vi configMap.yaml
(add the following at the end)
----
            {
              "resourceName": "mlnx_sriov_device," # resourceName can be any
name
              "resourcePrefix": "nvidia.com",
              "selectors": {
                 "vendors": ["15b3"],
                 "devices": ["1017", "1018"],
                 "drivers": ["mlx5_core"]
              }
             }
----
```

・Apply configMap. yaml:

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

●When using NICs listed in configMap. yaml (such as Intel 100 GNIC)

If the configuration of the target NIC is described in the default configMap. yaml, there is no need to edit configMap. yaml, and the default configMap. yaml can be applied.

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

For reference, here is the default configMap. yaml configuration (in red) for the Intel 100 GNIC:

```
...
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 Running the SR-IOV Device Plugin

Subject: K8s control plane

1) Run the SR-IOV device plug-in as daemonset

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

*If the daemonset of the SR-IOV device plug-in is already running after restarting the OS, use the following command to delete the daemonset and then apply it again.

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

2) Verify that the VF created in a K8s node is recognized as an assignable resource on a K8s node. The red part below shows the number of VFs that a K8s node can allocate, as long as it is equal to the number of VFs created.

●If you followed the steps in Section 8.9.1, "If the target NIC is an Intel 100 GNIC"

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

● If you followed the steps in Section 8.9.1, “If the applicable NIC is a Mellanox 100GNIC”

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

*If you want to check the number of VFs currently in use, you can look at K8s node's "Allocated resources" as follows:. The value of Requests or Limits in the red is the number of VFs in use.

● If you followed the steps in Section 8.9.1, "If the target NIC is an Intel 100 GNIC"

```
 $ 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%)
   (Omitted)
   intel.com/intel_sriov_netdevice   40            40
```

●If you followed the steps in Section 8.9.1, "If the target NIC is a Mellanox 100 GNIC"

```
 $ 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%)
   (Omitted)
   nvidia.com/mlnox_sriov_device     40            40
```

This completes the environment construction. The procedure for deploying and communicating DataFlow in the created environment is described in the separate document "OpenKasugai-Demo" so you can refer to it and carry out the procedure.

9. Appendix

The supplementary items and exception handling up to Chapter 8 of this document are explained as FAQ.

9.1 If you want to re-create the ConfigMap

If you want to re-create a ConfigMap after creating it in Section 8.6, you may not be able to create a ConfigMap correctly by simply redoing Section 8.6. The procedure for recreating a ConfigMap depends on how far you have gone after creating a ConfigMap.

There are two patterns as follows.

- ●If you want to re-create a ConfigMap immediately after it is created (immediately after the implementation of section 8.6)
- ●You want to recreate a DataFlow(hereinafter DF) after deployment.
  (If "1.3.1 DataFlow deployment" in the separate document "OpenKasugai-Demo" has already been performed)

9.1.1 If you want to re-create a ConfigMap immediately after it is created (immediately after the implementation of section 8.6)

ConfigMap under ~/controller/src/tools/InfoCollector

Since it is before the DF deployment, we can just repeat the section 8.6.

In the case of a multiple K8s nodes configuration, it is sufficient to skip step 1 of section 8.6 and perform step 2 and subsequent steps only on the K8s node for which you want to redo automatic collection and CM acquisition.

9.1.2 If you want to re-create a DataFlow after it has been deployed

In this pattern, since DF is deployed, it is necessary to restore the various controllers (CRC) to the initial state.

The CM group created by the automatic generation and CM creation tool is used when creating a new ComputeResource (CR). It is assumed that the new ComputeResource is created when the DeviceInfo controller is started, and that the DeviceInfo controller is started at the same timing as the various controllers.

1. Stopping Various Controllers (CRC): Implementation of "1.4.1 Delete DataFlow, Delete various CRs, Stop each CRC" in the separate document “OpenKasugai-Demo”
2. Rebuild ConfigMap under ~/controller/src/tools/InfoCollector: Implement Section 8.6
3. Launch of various CRCs: Implementation of "1.2.2 CRC start up" in the separate document "OpenKasugai-Demo"

9.2 When you want to flow only one DataFlow

If you want to stream only one DataFlow in a demo that streams multiple DataFlows in the separate document “OpenKasugai-Demo”, you can use only one DataFlow file and perform the corresponding procedure.

However, for "1.3.2 Start video reception" and "1.3.3 Start of video stream" in the separate document "OpenKasugai-Demo”, the settings of the video reception script and video distribution script will change depending on the type of DataFlow and the number of DataFlows to be deployed, so edit the script according to the flow pattern of the DataFlow to be used.

9.3 Returning the FPGA to the initial state immediately after the environment is created

If you want to return the FPGA to the initial state immediately after the system was built, perform "1.2.1 (1)Write FPGA Bitstream" in the separate document "OpenKasugai-Demo"

9.4 If you want to update the CRC

If the CRC (Custom Resource Controller) needs to be updated, If the evaluation has been carried out, be sure to start with removing CR and stopping CRC in accordance with the procedure described in "1.4.1 Remove DataFlow, CR, and CRC" in the separate document "OpenKasugai-Demo".

The following two cases are assumed as cases in which CRC is to be updated, and the procedure is described for each case.

- CRC Source Updated Case ➡ Section 9.4.1
- A problem occurred during "1.3.1 DataFlow deployment" in the separate document "OpenKasugai-Demo" ➡ Section 9.4.2
  - If the CRC does not start normally, or if the CRC starts normally but the CR does not start normally, and you want to re-insert the CRC (all or only the CRC that does not start normally)

9.4.1 Updating the CRC when the supplied file is updated

(1) If you want to update all CRCs

If you want to update the CRC in a batch using the CRC source code of the material list, you can complete the update by performing sections 8.1.2, 8.2, and 8.3.

(2) You want to update only specific CRCs

If you want to update the CRC in a batch using the CRC source code of the material list, you can complete the update by performing sections 8.1.2, 8.2, and 8.3.

9.4.2 Updating the CRC due to problems such as not being able to confirm the normal startup of the CRC

(4) If you want to update all CRCs

If you want to update the CRC in a batch, you can complete the update by performing Sections 8.2 and 8.3.

(5) You want to update only specific CRCs

If you want to update a specific CRC by using the CRC source code of the material list, you can complete the update by performing the corresponding CRC tasks in Sections 8.2 and the corresponding CRC startup preparation tasks in Section 8.3.

9.5 If you want to reset the evaluation environment

If you want to reset the evaluation environment to the initial state due to problems or other reasons while performing a demo according to the separate document "OpenKasugai-Demo", perform the following procedure.

1. Implementation of "1.3.4 Video stream and reception stop" and "1.4 Environmental shutdown" in the separate document "OpenKasugai-Demo"
   If video distribution is being performed, it is stopped, and various CR deletions and various CRC stops are performed.
2. Implement Section 9.4.2
   The CRCs are initialized by updating all CRCs again.
3. Initialization of "1.2 Environment initialization" in the separate document "OpenKasugai-Demo" before evaluation, it is reset to the initial state of the evaluation environment.

With the above, it is possible to perform the reimplementation and reevaluation of the demonstration (after "1.3 Video stream" in the separate document "OpenKasugai-Demo").

9.6 When to use a ghcr container image

If you use a container image from ghcr (GitHub Container Registry), you don't need to build the container image. The following table shows the container images that can be obtained.

	Container image name
CPU decoding processing module	ghcr.io/openkasugai/controller/cpu_decode:1.1.0
CPU filter/resize processing module	ghcr.io/openkasugai/controller/cpu_filter_resize:1.1.0
Image reception tool	ghcr.io/openkasugai/controller/rcv_video_tool:1.1.0
Image stream tool	ghcr.io/openkasugai/controller/send_video_tool:1.1.0
DataFlow controller, **WBscheduler controller,** **FunctionTarget controller,** **FunctionType controller,** **FunctionChain controller**	ghcr.io/openkasugai/controller/whitebox-k8s-flowctrl:1.1.0
WBConnection controller	ghcr.io/openkasugai/controller/wbconnection:1.1.0
WBFunction controller	ghcr.io/openkasugai/controller/wbfunction:1.1.0
CPUFunction controller	ghcr.io/openkasugai/controller/cpufunction:1.1.0
GPUFunction controller	ghcr.io/openkasugai/controller/gpufunction:1.1.0
FPGAFunction controller	ghcr.io/openkasugai/controller/fpgafunction:1.1.0
EthernetConnection controller	ghcr.io/openkasugai/controller/ethernetconnection:1.1.0
PCIeConnection controller	ghcr.io/openkasugai/controller/pcieconnection:1.1.0
DeviceInfo controller	ghcr.io/openkasugai/controller/deviceinfo:1.1.0

On the target server, acquire a container image, and change "ghcr.io/openkasugai/controller" in the container image name to localhost.

An example of the command for acquiring and changing the container image of the CPU decoding module is shown below.

```
$ 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
```

To configure a scheduling strategy for DataFlow

Target: K8s control plane, all K8s nodes

If you want to set or change DataFlow's scheduling strategy, do the following to create the required ConfigMap and edit DataFlow's YAML before "1.3.1 DataFlow deployment" in the separate document "OpenKasugai-Demo"

Overview of Configuring DataFlow Scheduling Strategies

Name	Settings
Strategy	・Specifies the filter used to filter and score DataFlow deployment candidates
UserRequirement	・Specify criteria for deploying DataFlow to ・Specify the Strategy ConfigMap used by DataFlow
DataFlow	・Specify ConfigMap for UserRequirement

*Relationship between DataFlow and each ConfigMap

●Required Actions

1. Creating a Strategy ConfigMap (See section 0 below for details.)
   As a common Strategy for each DataFlow in "1.3.1 DataFlow deployment" in the separate document "OpenKasugai-Demo", the default Strategy ConfigMap is deployed by run_controllers.sh in Section 1.2.3 (1). If you want to use a different strategy, create a new ConfigMap.

2. Creating a ConfigMap for a UserRequirement (See section 9.6.2 below for details.)
   As a common UserRequirement for each DataFlow in "1.3.1 DataFlow deployment" in the separate document "OpenKasugai-Demo", run_controllers.sh in section 1.2.2 (1)/ home/ubuntu/k8s-software/src/tools/InfoCollector/testdata_day/user_requirement/user_requirement.yaml is deployed. If you want to use a different UserRequirement, create a new ConfigMap.

3. In DataFlow to be deployed, specify UserRequirement as described in 2 above (See section 9.6.3 below for how to specify)

9.6.1 Strategy ConfigMap settings

●Setting item

Item name	Data type	Required	Contents
referenceParameter	string	No	・Specify metadata.Name of another Strategy's ConfigMap to reference for Strategy configuration
filterPipeline	[]string	No	・Specifies the filter to use for scheduling DataFlow ・There are six types of filters (1) GenerateCombinations: Generates information required to process each filter, taking into account the device to which it is deployed.

			(2) TargetResourceFit: Filtering based on destination device type (3) TargetResourceFitScore: Scoring based on the capacity of the device to be deployed (4) GenerateRoute – Generates the information required to process each filter, taking into account the topology information (5) ConnectionResourceFit: Filtering for connection paths on topology information (6) RouteScore – Scoring for capacity of connections on topology information ・If scheduling takes topology information into account, specify all of the above (1) to (5). When scheduling without considering topology information, specify only (1) through (3) above. ・If Strategy and UserRequirement are not used in DataFlow, the default filter (1) - (3) above will be executed
selectTop	int	No	・Get up to < set value > th score of filter results
<N>.referencePara meter	string	No	・For the <N> th filter in filterPipeline, specify metadata.Name of another Strategy ConfigMap to reference for Strategy configuration
<N>.selectTop	int	No	・Get the filter result of the <N> th filter in the filterPipeline up to the < set value > th Score

●Configuration Example

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

9.6.2 ConfigMap settings for UserRequirement

●Setting item

Item name	Data type	Required	Contents
strategy	string	Yes	・Specify metadata.Name of another Strategy's ConfigMap to reference for Strategy configuration

requestNodeNames	map[string][]string	No	・"Specify the nodeName to which the function specified in the map key is to be deployed or not deployed. ・The key of map is the key value of FunctionChain .FunctionChainSpec.Functions that indicates the Function. ・The value of map is an array of nodeName. ' Designate as non-deployable by appending -'
requestDeviceTypes	map[string][]string	No	・Specify the deployment-destination/non-deployment-destination deviceType of the function specified in the map key. ・The key of map is the key value of FunctionChain .FunctionChainSpec.Functions that indicates the Function. ・The value of map is an array of deviceType. ' Designate as non-deployable by appending -'
requestFunctionTargets	map[string][]string	No	・Specifies the FunctionTarget to which the function specified in the map key is to be deployed or not deployed. ・The key of map is the key value of FunctionChain .FunctionChainSpec.Functions that indicates the Function. ・The value of map is an array of FunctionTargets. ' Designate as non-deployable by appending -'
requestRegionNames	map[string][]string	No	・Get the filter result of the <N> th filter in the filterPipeline up to the < set value > th Score
requestFunctionIndexes	map[string][]string	No	・Specify the deployment-destination/non-deployment-destination deviceType of the function specified in the map key. ・The key of map is the key value of FunctionChain .FunctionChainSpec.Functions that indicates the Function. ・The value of map is an array of regionName. ' Designate as non-deployable by appending -'
functionTargetNameSpace	string	No	Specifies the metadata.namespace of the FunctionTarget to reference when executing a Filter that uses device information

●Configuration Example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-requirement
  namespace : default
data:
  strategy : strategy
  requestNodes : |
    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 Specifying UserRequirement in DataFlow

●Configuration Example

In the red (Spec.UserRequirement) below, specify metadata.name in UserRequirement's ConfigMap.

```
apiVersion: example.com/v1
kind: DataFlow
metadata:
  name: "df-test-4-1-1-1"
  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 MTU9000 settings for Intel/Mellanox100/25GNIC

Target: K8s nodes where the advanced inference app is deployed,
server where the video receiving tool is running

In the following, the setting procedure will be described using the assumed environment shown in FIG. 1 as an example.

●Configuring the Sender (K8s node 2)

1) Set the MTU of the physical port of the NIC.
Edit/etc/netplan/00-installer-config.yaml as follows:

```
ens93f0:
    addresses:
    - 192.174.90.33/24
    mtu: 9000          ★Set mtu to 9000
```

2) Apply to the system. Temporarily apply with the following command:

```
$ 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
```

If there is no particular error as shown in the above message, pressing the return key here sets the temporary application to the actual application.

3) Verify that mtu 9000 is displayed.

```
$ 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
```

●Receiver (K8s control plane) settings

1) Set the IP address of the physical port of the NIC and set the MTU.
Edit/etc/netplan/00-installer-config.yaml as follows: [*]

```
ens7f0:
    match:
        macaddress: b4:96:91:ca:54:b8
    addresses:
    - 192.174.90.11/24
    mtu: 9000          ★Set mtu to 9000
```

2) Apply to the system. Temporarily apply with the following command:

```
$ 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
```

If there is no particular error as shown in the above message, pressing the return key here sets the temporary application to the actual application.

3) Verify that mtu 9000 is displayed.

```
$ 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 If you want to cold reboot the K8s node after performing video distribution and restart video distribution

By following the procedure below, you can return to the state in which video distribution is enabled (the state in which "1.3 Video stream" in the separate document "OpenKasugai-Demo" can be executed).

1. Remove CRs, stop CRCs: "1.4.1 Delete DataFlow, Delete various CRs, Stop each CRC" in the separate document "OpenKasugai-Demo"
2. cold reboot implementation

```
$ sudo poweroff
```

3. Initializing the environment (Write FPGA, CRC Boot): Implementation of "1.2 Environment Initialization" in the separate document "OpenKasugai-Demo"