OpenKasugai Demo for all-in-one

This document describes the design, deployment methods, etc. regarding the following scenario.

1. Arithmetic operation scenario
2. Video inference scenario

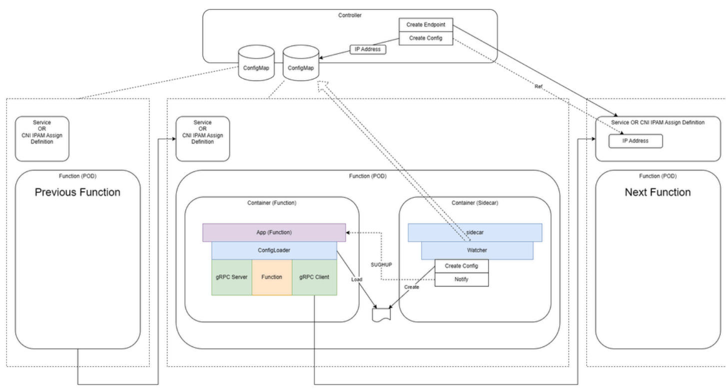
Positioning

The scenario introduced on this page is an improvement of the scenarios included in the OpenKasugai-controller. The existing scenarios require explicit specification of the connection destination information for the next Function (e.g., IP address), but this scenario eliminates the need for address specification by allowing the Connection controller to provide the connection destination information for the Function and automatically create Service resources based on the connection relationships between Functions.

Currently, it only supports CPU/GPU Functions and Ethernet Connections.

Function Internal Design

The mechanism by which the controller automatically sets the IP address for the next Function is shown below.



1. The Connection controller refers to the Pod information of the Function and creates a ConfigMap containing information about the connection destination.
2. The Function starts in a way that includes a sidecar. The sidecar watches the ConfigMap and saves the ConfigMap related to its Function as Config. The Config is stored in a shared directory within the Pod.
3. When the application starts, it loads the Config from the shared directory to set the destination for the next Function.
4. When a change in the Config is detected, the sidecar issues a signal (SIGHUP) to the Function process.
5. The application that receives the signal updates the destination for the next Function by reloading the Config.

Communication between Functions is done using Service resources, except in cases where communication occurs directly via external NW (e.g., SR-IOV NIC). The Connection controller automatically creates Service resources according to the connection relationships of the Function.

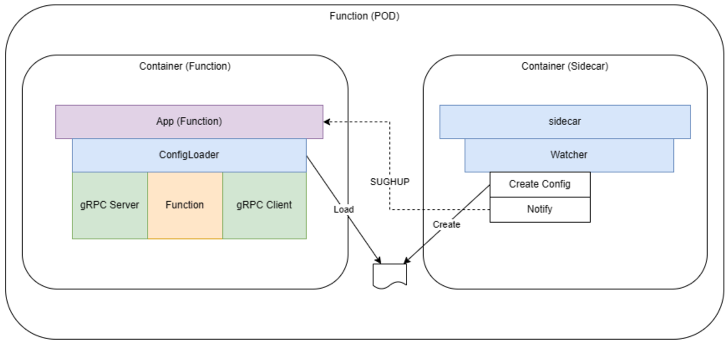
1. If it is the first Function in the DataFlow, a Service(type=LoadBalancer) is created.
2. If the preceding Function communicates using internal NW, a Service(type=ClusterIP) is created.

Differences in Structure Between Arithmetic Scenarios and Video Inference Scenarios

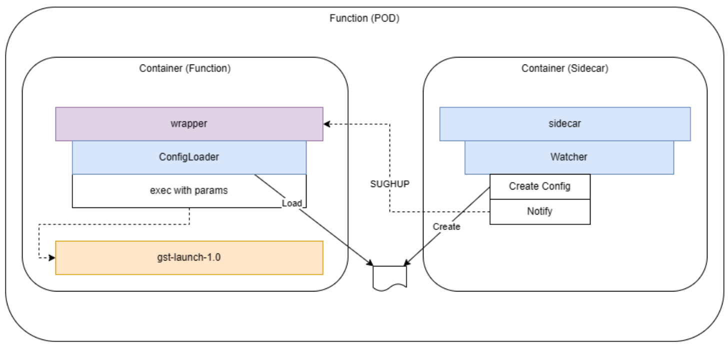
The arithmetic scenario has a mechanism in the Function program to receive signals from the sidecar. When manufacturing the Function as a standalone program, refer to the implementation of this scenario.

The video inference scenario does not have a mechanism for the process (gstreamer) that performs video inference to receive signals from the sidecar, thus control is carried out through a program called a wrapper. When relying on external programs for the Function's processing, refer to the implementation of this scenario.

App Side



Wrapper Side



Implementation Preconditions

To execute this scenario, it is necessary to additionally introduce components into the execution environment of OpenKasugai-controller.

Creating a Kubernetes Namespace

Create a namespace to be used for the scenario.

```
$ kubectl create namespace cpufunc-caclapp
$ kubectl create namespace cpufunc-sample
```

Adding Reference Permissions to Kubernetes Resources

Add reference permissions so that Kubernetes resources can be referenced from within the function.

```
$ kubectl create clusterrolebinding cpufunc-caclapp-default-view --clusterrole=view --serviceaccount=cpufunc-calcapp:default
$ kubectl create clusterrolebinding cpufunc-sample-default-view --clusterrole=view --serviceaccount=cpufunc-sample:default
```

Installation of MetalLB

MetalLB is introduced to set external network IP addresses for functions in Kubernetes.

```
$ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.14.8/config/manifests/metallb-native.yaml
```

Set to L2 mode

```
$ cat <<EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: example
  namespace: metallb-system
EOF
```

To define the IPPool used in the scenario, input the definition of the IPPool.
Make sure `addresses` matches the actual environment.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: cpufunc-pool
  namespace: metallb-system
spec:
  addresses:
```

```
  - 192.168.91.240-192.168.91.249
EOF
```

Installation of nvidia-k8s-ipam

nvidia-k8s-ipam is introduced to automatically assign IP addresses to Function's SR-IOV NIC.

```
$ kubectl kustomize https://github.com/mellanox/nvidia-k8s-ipam/deploy/overlays/no-webhook?ref=v0.3.5 | kubectl apply -f -
```

To define the IP Pool used in the external connection (SR-IOV) scenario, input the definition of the IP Pool.
`subnet`, `perNodeBlockSize`, and `exclusions` should be adjusted to fit the actual environment.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: nv-ipam.nvidia.com/v1alpha1
kind: IPPool
metadata:
  name: nv-pool1
  namespace: kube-system
spec:
  subnet: 192.168.91.0/24
  perNodeBlockSize: 250
  gateway: 192.168.91.1
  exclusions: # optional
  - startIP: 192.168.91.0
    endIP: 192.168.91.239
  - startIP: 192.168.91.250
    endIP: 192.168.91.255
  nodeSelector:
    nodeSelectorTerms:
    - matchExpressions:
        - key: node-role.kubernetes.io/control-plane
          operator: DoesNotExist
EOF
```

Creation of NetworkAttachmentDefinition

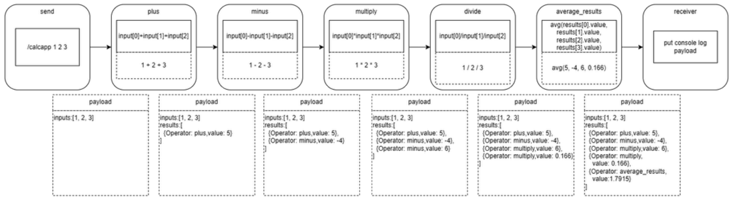
Define the NetworkAttachmentDefinition to assign an IP address to the POD in external connection scenarios.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-ipam-config
  namespace: cpufunc-caclapp
  annotations:
    k8s.v1.cni.cncf.io/resourceName: nvidia.com/mlnx_sriov_netdevice
spec:
  config: '{
    "type": "sriov",
    "cniVersion": "0.3.1",
    "name": "sriov-ipam-config",
    "ipam": {
        "type": "nv-ipam",
        "poolName": "nv-pool1"
      }
    }'
EOF
```

```
$ cat <<EOF | kubectl apply -f -
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-ipam-config
  namespace: cpufunc-sample
  annotations:
    k8s.v1.cni.cncf.io/resourceName: nvidia.com/mlnx_sriov_netdevice
spec:
  config: '{
    "type": "sriov",
    "cniVersion": "0.3.1",
    "name": "sriov-ipam-config",
    "ipam": {
        "type": "nv-ipam",
        "poolName": "nv-pool1"
      }
    }'
EOF
```

Arithmetic Operations Scenario

Overview



This document outlines the overview of the Function addressed in this scenario.

Function	Overview
send	Data Input Function Connection information to the next Function is set, and by executing `/calcapp [input value]`, input data will be sent to the next Function.
plus	Addition Function Sum all the contents of the input data using `+` to create the result. Send the input data as it is, along with the result, to the next function.
minus	Subtraction Function Create a result by subtracting all the contents of the input data with `-` At this time, the first input data will not have a `-` attached, and `-` will be inserted between the input data to create the result Send the input data as it is along with the result to the next Function
multiply	Multiplication Function Create a result by multiplying all the contents of the input data by `*` Send the input data as it is along with the result to the next Function
divide	Division Function Perform `/` on all contents of the input data to create the result Send the input data as is along with the result to the next Function
average_results	Result Average Function Calculate the average value of all values in the result data and create the result Send the input data as it is along with the result to the next Function
receiver	Output Function Log the input data and the contents of the results.

Sample Scenario Content

Scenario Description

As a sample for the arithmetic operation scenario, the following is stored within the repository.

Configuration Destination	Scenario Overview	Notes
test/sample-data/sample-data-for-all-in-one/calc-func/basic	Arithmetic operation scenario internal connection pattern	Connecting each Function using internal cluster communication.
test/sample-data/sample-data-for-all-in-one/calc-func/sriov	Arithmetic operation scenario external connection pattern	Connect each function using SR-IOV.

Resources, ConfigMap Description

Each directory contains the following files, each serving the following roles.

File Name	k8s Resource Type	Explanation
cm-cpufunc-config.yaml	ConfigMap	Store detailed configuration information necessary for the execution of the function.
functioninfo.yaml	ConfigMap	Stores information about the type of accelerator that executes the Function and the connection methods supported by the Function.
functiontype.yaml	FunctionType	Define the Function available on OpenKasugai from the Config and Info of the Function.
functionchain.yaml	FunctionChain	Define the combination of functions
df-cpufunc.yaml	DataFlow	Specify the FunctionChain to deploy the DataFlow.

Building the Image

Build the sidecar on each worker node.

```
$ cd sample-functions/functions/for_all_in_one/cpugpu-func/cpufunc_sidecar
$ buildah bud -t cpufunc_sidecar .
```

Build the arithmetic function on each worker node.

```
$ cd sample-functions/functions/for_all_in_one/calc-func/cpufunc_calcapp
$ buildah bud -t cpufunc_calcapp .
```

Deployment Method

The deployment of DataFlow is carried out by performing resource allocation in the following order. The resource allocation uses the `kubectl apply -f` command.

Application Order	Applicable file name	k8s Resource Types	Explanation
1	cm-cpufunc-config.yaml	ConfigMap	Store detailed configuration information necessary for the execution of the function.
2	functioninfo.yaml	ConfigMap	Stores information about the type of accelerator that executes the Function and the connection methods supported by the Function.
3	functiontype.yaml	FunctionType	Define the Function available on OpenKasugai from the Config and Info of the Function.
4	functionchain.yaml	FunctionChain	Define the combination of functions.

	Wait until various resources are OK, Running, or Deployed after execution.
	`$ kubectl get crd | grep example.com | cut -d ' ' -f 1 | paste -s -d ',' | \`
	`xargs kubectl get -A`		

5	df-cpufunc.yaml	DataFlow	Specify the FunctionChain to

			deploy the DataFlow.

Result Confirmation Procedure

Input and Output Confirmation Procedure

The execution of arithmetic operation scenarios is carried out by inputting data via gRPC and obtaining the results from the standard output of the Pod.

Input Implementation Procedure

```
$ kubectl exec -it -n cpufunc-calcapp df-calcapp-wbfunction-send-cpu-pod -- /calcapp 1 2 3 4 5
Defaulted container "cpu-container0" out of: cpu-container0, cpu-container1
2024-12-10T06:58:35.571Z	INFO	workspace/main.go:104	load config	{"found": true, "config": {"Next":{"Host":"10.96.134.3","Port":8080}}}
2024-12-10T06:58:35.571Z	INFO	workspace/main.go:271	send message	{"address": "10.96.134.3:8080", "request": "inputs:1 inputs:2 inputs:3 inputs:4 inputs:5"}
2024-12-10T06:58:35.571Z	INFO	workspace/main.go:282	connecting	{"address": "10.96.134.3:8080"}
2024-12-10T06:58:35.590Z	INFO	workspace/main.go:276	recive message	{"address": "10.96.134.3:8080", "response": "status:OK"}
```

Output Confirmation Procedure

```
$ kubectl logs -n cpufunc-calcapp df-calcapp-wbfunction-rcv-cpu-pod -c cpu-container0
2024-12-10T06:58:19.381Z	INFO	workspace/main.go:130	start notify loader
2024-12-10T06:58:19.381Z	INFO	workspace/main.go:223	start gRPC server	{"port": 8080}
2024-12-10T06:58:19.381Z	INFO	workspace/main.go:152	start config loader
2024-12-10T06:58:20.959Z	INFO	workspace/main.go:158	loading config
2024-12-10T06:58:20.960Z	INFO	workspace/main.go:168	load config	{"found": true, "config": {"Next":{"Host":"localhost","Port":8080}}}
2024-12-10T06:58:35.585Z	INFO	workspace/main.go:312	start call Do	{"req": "inputs:1 inputs:2 inputs:3 inputs:4 inputs:5 results:{operator:\"pluse\" value:15}
 results:{operator:\"minus\" value:-13} results:{operator:\"multiply\" value:120} results:{operator:\"divide\" value:0.008333333333333333} results:{operator:\"average\" value:30.502083333333335}"}
```

Customization Procedure for FunctionChain

By modifying FunctionChain, confirm that the execution results obtained in arithmetic scenarios change.

Edit `functionchain.yaml` as follows

(an example changed to send → plus → minus → rcv)

```
$ cat functionchain.yaml
apiVersion: example.com/v1
kind: FunctionChain
metadata:
  name: calcapp-chain
  namespace: cpufunc-calcapp
spec:
  functionTypeNamespace: "cpufunc-calcapp"
  connectionTypeNamespace: "cpufunc-calcapp"
  functions:
    send:
      functionName: "calcapp-send"
      version: "1.0.0"
    plus:
      functionName: "calcapp-plus"
      version: "1.0.0"
    minus:
      functionName: "calcapp-minus"
      version: "1.0.0"
    rcv:
      functionName: "calcapp-rcv"
      version: "1.0.0"
  connections:
  - from:
      functionKey: "wb-start-of-chain"
      port: 0
    to:
      functionKey: "send"
      port: 0
    connectionTypeName: "auto"
  - from:
      functionKey: "send"
      port: 0
    to:
      functionKey: "plus"
      port: 0
    connectionTypeName: "auto"
  - from:
      functionKey: "plus"
      port: 0
    to:
      functionKey: "minus"
      port: 0
    connectionTypeName: "auto"
```

```
  - from:
      functionKey: "minus"
      port: 0
    to:
      functionKey: "rcv"
      port: 0
    connectionTypeName: "auto"
  - from:
      functionKey: "rcv"
      port: 0
    to:
      functionKey: "wb-end-of-chain"
      port: 0
    connectionTypeName: "auto"
```

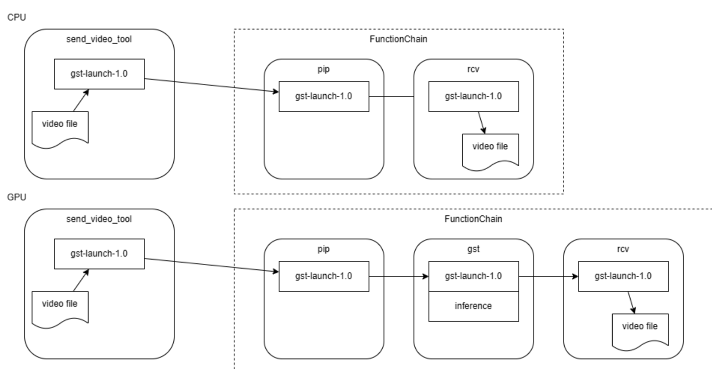
After implementing the deployment method and deploying, execute the input and check the output.

```
$ kubectl exec -it -n cpufunc-calcapp df-calcapp-wbfunction-send-cpu-pod -- /calcapp 1 2 3 4 5
Defaulted container "cpu-container0" out of: cpu-container0, cpu-container1
2024-12-13T07:03:54.634Z	INFO	workspace/main.go:104	load config	{"found": true, "config": {"Next":{"Host":"10.96.203.93","Port":8080}}}
2024-12-13T07:03:54.634Z	INFO	workspace/main.go:271	send message	{"address": "10.96.203.93:8080", "request": "inputs:1 inputs:2 inputs:3 inputs:4 inputs:5"}
2024-12-13T07:03:54.634Z	INFO	workspace/main.go:282	connecting	{"address": "10.96.203.93:8080"}
2024-12-13T07:03:54.644Z	INFO	workspace/main.go:276	recive message	{"address": "10.96.203.93:8080", "response": "status:OK"}
$ kubectl logs -n cpufunc-calcapp df-calcapp-wbfunction-rcv-cpu-pod -c cpu-container0
2024-12-13T07:02:36.473Z	INFO	workspace/main.go:152	start config loader
2024-12-13T07:02:36.473Z	INFO	workspace/main.go:130	start notify loader
2024-12-13T07:02:36.473Z	INFO	workspace/main.go:223	start gRPC server	{"port": 8080}
2024-12-13T07:02:38.036Z	INFO	workspace/main.go:158	loading config
2024-12-13T07:02:38.036Z	INFO	workspace/main.go:168	load config	{"found": true, "config": {"Next":{"Host":"localhost","Port":8080}}}
2024-12-13T07:03:54.641Z	INFO	workspace/main.go:312	start call Do	{"req": "inputs:1 inputs:2 inputs:3 inputs:4 inputs:5 results:{operator:\"pluse\" value:15} results:{operator:\"minus\" value:-13}"}
```

Video Inference Scenario

Overview

The video inference scenario is implemented in a way that it is executed by combining CPU Function and GPU Function.



This document outlines the overview of the Function addressed in this scenario.

Function	Overview
pip	Video Transfer Function Send the input video directly to the next Function.
gst	Image Inference Function Perform inference processing on the input image and send it to the Function.
rcv	Image Reception Function Save the input image directly to a file.

Content of Sample Scenario

Scenario Description

As a sample of the video inference scenario, the following is stored in the repository. A scenario has been prepared to verify the operation of multiple connection patterns.

Configuration Destination	Scenario Overview	Notes

test/sample-data/sample-data-for-all-in-one/cpugpu-func/p1c1	Internal NW connection pattern	Connect each function using internal communications of the cluster.
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p1c2	External NW Connection Pattern	Connect each Function using external communication of the cluster.
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p3c1	SR-IOV connection pattern	Connecting each function using SR-IOV
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c1	GPU collaboration internal NW connection pattern	Connect each Function using internal cluster communication.
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c2	GPU collaboration external network connection pattern	Connect each function using external communication of the cluster.
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c3	GPU collaboration SR-IOV connection pattern	Connect each function using SR-IOV.
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c4	GPU collaboration multiple internal network connection patterns	Using internal communication within the cluster to connect each function Multiple functions are registered in Chain

Resource, ConfigMap Explanation

Each directory contains the following files, each serving the following roles.

File Name	k8s Resource Type	Explanation
cm-cpufunc-config.yaml	ConfigMap	Store detailed configuration information necessary for the execution of the function.
functioninfo.yaml	ConfigMap	Stores information about the type of accelerator that executes the Function and the connection methods supported by the Function.
functiontype.yaml	FunctionType	Define the Function available on OpenKasugai from the Config and Info of the Function.
functionchain.yaml	FunctionChain	Define the combination of functions
df-cpufunc.yaml	DataFlow	Specify the FunctionChain to deploy the DataFlow.

Building the Image

Building gpu_infer_tcp_plugins

On each worker node, build gpu_infer_tcp_plugins according to the README in the following directory.
At this time, set the tag to `localhost/gpu_infer_tcp:1.0.0`.

```
sample-functions/functions/gpu_infer_tcp_plugins/fpga_depayloader
```

Build of rcv_video_tool
On each worker node, build rcv_video_tool according to the README in the following directory.
At this time, the tag should be `localhost/rcv_video_tool:1.0.0`.

```
sample-functions/utils/rcv_video_tool
```

Build the sidecar on each worker node.

```
$ cd sample-functions/functions/for_all_in_one/cpugpu-func/cpufunc_sidecar
$ buildah bud -t cpufunc_sidecar .
```

Build video inference on each worker node.

```
$ cd sample-functions/functions/for_all_in_one/cpugpu-func/cpufunc_gst
$ buildah bud -t cpufunc_gst .
$ cd sample-functions/functions/for_all_in_one/cpugpu-func/gpufunc_dsa
$ buildah bud -t gpufunc_dsa .
```

Deployment Method

Starting the delivery container

If the POD of send_video_tool is not running, start the send_video_tool.

```
$ cd sample-functions/utils/send_video_tool
$ kubectl apply -f send_video_tool.yaml
```

The deployment of DataFlow is carried out by performing resource allocation in the following order. The resource allocation uses the `kubectl apply -f` command.

Application Order	Applicable File Name	sk8 Resource Types	Explanation
1	cm-cpufunc-config.yaml	ConfigMap	Storing detailed configuration information necessary for the execution of the function.
2	functioninfo.yaml	ConfigMap	Stores information about the type of accelerator that executes the Function and the connection methods supported by the Function.
3	functiontype.yaml	FunctionType	Define the Function available on OpenKasugai from the Config and Info of the Function.
4	functionchain.yaml	FunctionChain	Define the combination of functions

	Wait until each resource is OK, Running, or Deployed after execution.
	`$ kubectl get crd | grep example.com | cut -d ' ' -f 1 | paste -s -d ',' | \`
	`xargs kubectl get -A`		

5	df-cpufunc.yaml	DataFlow	Specify the FunctionChain to deploy the DataFlow.

Result Confirmation Procedure

Input and Output Confirmation Procedure

The input of video to DataFlow is conducted by sending video data from the video distribution Pod. The output of the results is performed by playing the mp4 file saved within the Pod executing the data reception Function.

Input Execution Procedure

Enter the distribution container and send the video data.

Video files should be located in `/opt/DATA/video/` on the node, this directory is mounted as `/opt/video/` in the container. This example is that `sample.mp4` is located in `/opt/DATA/video/`.

```
$ sudo docker exec -it send_video_tool bash
$ gst-launch-1.0 -e -v filesrc location=/opt/video/sample.mp4  \
  ! qtdemux  \
  ! video/x-h264  \
  ! h264parse  \
  ! rtph264pay config-interval=-1 seqnum-offset=1  \
  ! udpsink host=192.168.91.241 port=5678 buffer-size=2048
```

The part of `udpsink host` checks the IP address as follows.

p1c1

Addressing to `EXTERNAL-IP`

```
$ kubectl get service -n cpufunc-sample df-cpu-p1c1-wbfunction-pip-service
```

```
NAME                              TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
df-cpu-p1c1-wbfunction-pip-service   LoadBalancer   10.103.16.154   192.168.91.240   5678:30751/UDP   6m6s
```

p1c2

Like p1c1

p3c1

In the POD resource, the destination is the IP address of the object with `"name": "cpufunc-sample/sriov-ipam-config"` in the `annotations` :: `k8s.v1.cni.cncf.io/network-status` item.

```
$ kubectl get pod -n cpufunc-sample df-cpu-p3c1-wbfunction-pip-cpu-pod -o yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.projectcalico.org/containerID: a1c0691186339958a73e5091a90ce9bbee938c6dfa3ec2c136d423f3dd0711ed
    cni.projectcalico.org/podIP: 182.16.162.184/32
    cni.projectcalico.org/podIPs: 182.16.162.184/32
    ethernet.swb.example.com/network: sriov
    k8s.v1.cni.cncf.io/network-status: |-
      [{
          "name": "k8s-pod-network",
          "ips": [
              "182.16.162.184"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "cpufunc-sample/sriov-ipam-config",
          "interface": "net1",
          "ips": [
              "192.168.91.248" <-- chek IP Address
          ],
          "mac": "0a:99:a4:f4:31:56",
          "dns": {},
          "device-info": {
              "type": "pci",
              "version": "1.1.0",
              "pci": {
                  "pci-address": "0000:89:00.4"
              }
          }
      }]
    k8s.v1.cni.cncf.io/networks: sriov-ipam-config
  creationTimestamp: "2024-08-26T07:38:27Z"
  labels:
    swb/func-name: df-cpu-p3c1-wbfunction-pip-cpu-pod
    swb/func-type: cpufunc
  name: df-cpu-p3c1-wbfunction-pip-cpu-pod
  namespace: cpufunc-sample
  ownerReferences:
  - apiVersion: example.com/v1
    blockOwnerDeletion: true
    controller: true
    kind: CPUFunction
    name: df-cpu-p3c1-wbfunction-pip
    uid: c5c5e0ad-1964-4ca0-80e3-9e111ff25ad6
  resourceVersion: "23205212"
  uid: 8ea3247d-1497-4f31-b1b3-09d5f5961352
~~~
```

p2c1

Same as p1c1

p2c2

Same as p1c2

p2c3

Same as p3c1

p2c4

Same as p1c1

Output Confirmation Procedure

After the delivery is completed in the delivery container, terminate the gst-launch-1.0 process within the receiving Function (POD) using `kill -2`.

```
$ kubectl exec -it -n cpufunc-sample df-cpu-p3c1-wbfunction-rcv-cpu-pod -c cpu-container0 -- bash
```

```
# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0   1028     4 ?        Ss   07:38   0:00 /pause
root          7  0.0  0.0   5148  3436 ?        Ss   07:38   0:00 bash -c cp /config-rcv.yaml.tmpl /config/config.yaml.tmpl; /wrapper
root         20  0.0  0.0 1673568 6792 ?        Sl   07:38   0:00 /wrapper
root         26  0.0  0.0 2442784 28044 ?       Ssl  07:38   0:00 /sidecar
root         43  0.0  0.0 346932 44320 ?        SLl  07:38   0:00 gst-launch-1.0 -e -vvv udpsrc buffer-size=21299100 port=5678 ! application/x-rtp, media=video, encoding-name=H264, clock-rate=90000, payload=
root         83  0.0  0.0   5412  3944 pts/0    Ss   07:53   0:00 bash
root         90  0.0  0.0   7068  3360 pts/0    R+   07:53   0:00 ps aux
# kill -2 43
```

Copy video data outside the receiving function.

```
kubectl cp -n cpufunc-sample -c cpu-container0 df-cpu-p3c1-wbfunction-rcv-cpu-pod:rcv_video.mp4 rcv_video.mp4
```