OpenKasugai デモ for all-in-one

本ドキュメントでは以下のシナリオに関する設計、デプロイ方法等について記載する

1. 四則演算シナリオ
2. 映像推論シナリオ

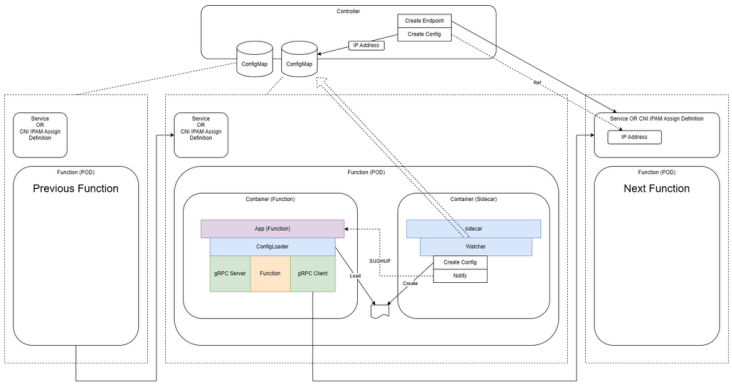
位置づけ

本ページで紹介するシナリオは、OpenKasugai-controllerに含まれるシナリオを改善したものである。既存のシナリオは、次Functionの接続先情報（例：IPアドレス）を明示的に指定する必要があるが、本シナリオは、Connectionコントローラが Functionの接続先情報を提供したり、Function間の接続関係に応じてServiceリソースを自動作成することにより、アドレス指定を不要とする改修を行っている。

現状、CPU/GPU FunctionとEthernet Connectionのみに対応している

Function内設計

コントローラが次FunctionのIPアドレスを自動設定する仕組みを以下に示す。



1. Connectionコントローラは、FunctionのPod情報を参照し、接続先に関する情報を含むConfigMapを作成する
2. Functionはsidecarを含む形で起動する。sidecarはConfigMapをwatchしており、自Functionに関連するConfigMapをConfigとして保存する。Configは、Pod内の共有ディレクトリに保存する。
3. アプリが起動するときは、共有ディレクトリ上のConfigをロードして、次Functionの宛先を設定する
4. Configの変更を検知したとき、sidecarはFunctionプロセスにシグナル（SIGHUP）を発行する
5. シグナルを受信したアプリは、Configをリロードすることにより、次Functionの宛先を更新する

Function間の通信は、外部NW（例：SR-IOV NIC）を介して直接行う場合を除き、Serviceリソースを用いて通信する。Connectionコントローラは、Functionの接続関係に応じて、Serviceリソースを自動作成する。

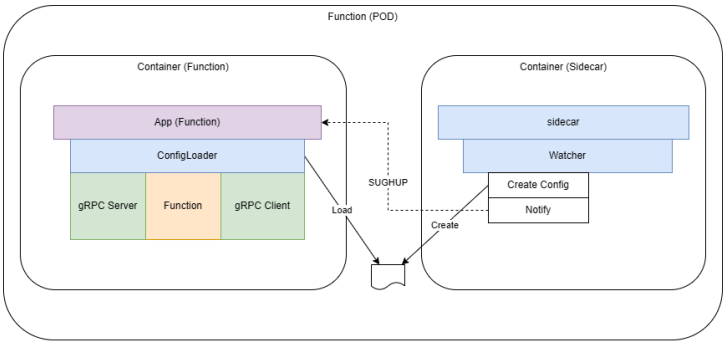
1. 自身がDataFlowにおける最初のFunctionの場合、Service(type=LoadBalancer)を作成する
2. 直前のFunctionが内部NWを使用して通信を行う場合、Service(type=ClusterIP)を作成する

四則演算シナリオと映像推論シナリオの構造の違い

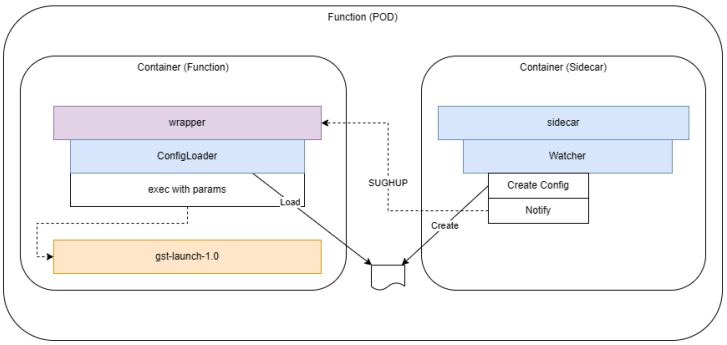
四則演算シナリオは、Functionプログラム内に、sidecarからシグナルを受け取る仕組みを用意している。Functionを単独プログラムとして製造する場合、本シナリオの実装を参考に行う。

映像推論シナリオは、映像推論を行うプロセス（gstreamer）がsidecarのシグナルを受け取る仕組みを持っていないため、wrapperと呼ばれるプログラムを介して制御を行う。Functionの処理を外部プログラムに頼る場合、本シナリオの実装を参考に行う。

App Side



Wrapper Side



実施前提

本シナリオを実行するために、OpenKasugai-controllerの実行環境にコンポーネントを追加で導入する必要がある。

kubernetes namespace の作成

シナリオに用いる namespace を作成する

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

kubernetesリソースへの参照権限追加

Function内からkubernetesリソースを参照可能なよう参照権限を追加する

```
$ 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
```

MetalLBのインストール

MetalLBは、Functionにkubernetesの外部のNWのIPアドレスを設定するために導入する

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

L2モードに設定

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

シナリオにて使用するIPPoolの定義を行う為、IPPoolの定義を投入する
`addresses` は実際の環境に合せる事

```
$ 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
```

nvidia-k8s-ipamのインストール

nvidia-k8s-ipamは、FunctionのSR-IOV NICのIPアドレスを自動的に付与するために導入する。

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

外部接続(SR-IOV)シナリオにて使用するIPPoolの定義を行う為、IPPoolの定義を投入する
`subnet`、`perNodeBlockSize`、`exclusions` は実際の環境に合せる事

```
$ 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
```

NetworkAttachmentDefinitionの作成

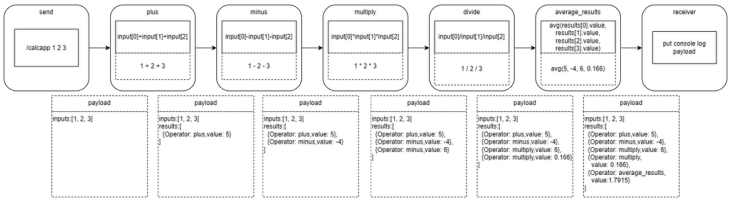
外部接続シナリオにてPODにIPアドレスを割り当てる為にNetworkAttachmentDefinitionの定義を行う

```
$ 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
```

四則演算シナリオ

概要



本シナリオで扱うFunctionの概要を記載する

Function	機能
send	データ入力Function 次のFunctionへの接続情報が設定されており `/calcapp [入力値]`を実行する事で次のFunctionに入力データを送信する
plus	加算Function 入力データの内容を全て `+` し結果を作成 入力データをそのまま入力データとし結果と共に次のFunctionに送信する
minus	減算Function 入力データの内容を全て `-` し結果を作成 この際、先頭の入力データには `-` が付与されず、入力データ間に `-` を挿入して結果を作成する 入力データをそのまま入力データとし結果と共に次のFunctionに送信する
multiply	乗算Function 入力データの内容を全て `*` し結果を作成 入力データをそのまま入力データとし結果と共に次のFunctionに送信する
divide	除算Function 入力データの内容を全て `/` し結果を作成 入力データをそのまま入力データとし結果と共に次のFunctionに送信する
average_results	結果平均Function 結果データの全ての値の平均値を算出し結果を作成 入力データをそのまま入力データとし結果と共に次のFunctionに送信する
receiver	出力Function 入力データ、結果の内容をログに出力する

サンプルシナリオの内容

シナリオ説明

四則演算シナリオのサンプルとして、リポジトリ内に以下を格納している。

配置先	シナリオ概要	備考
test/sample-data/sample-data-for-all-in-one/calc-func/basic	四則演算シナリオ内部接続パターン	クラスタ内部通信を用いて各Functionの接続を行う
test/sample-data/sample-data-for-all-in-one/calc-func/sriov	四則演算シナリオ外部接続パターン	SR-IOVを用いて各Functionの接続を行う

リソース、ConfigMap説明

各ディレクトリには以下ファイルを格納しており、それぞれ以下の役割を持つ

ファイル名	k8sリソース種別	説明
cm-cpufunc-config.yaml	ConfigMap	Functionの実行に必要な詳細な設定情報を格納

functioninfo.yaml	ConfigMap	Functionを実行するアクセラレータ種別や、Functionがサポートする接続方式に関する情報を格納
functiontype.yaml	FunctionType	FunctionのConfigとInfoから、OpenKasugai上で利用可能なFunctionを定義する
functionchain.yaml	FunctionChain	Functionの組み合わせを定義する
df-cpufunc.yaml	DataFlow	FunctionChainを指定して、DataFlowの配備を行う

イメージのビルド

各ワーカーノードでサイドカーのビルドを行う

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

各ワーカーノードで四則演算Functionのビルドを行う

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

デプロイ方法

DataFlowのデプロイは、リソース配置を以下の順で行うことで実施する。リソース配置には、 `kubectl apply -f` コマンドを使用する。

適用順	適用ファイル名	k8sリソース種別	説明
1	cm-cpufunc-config.yaml	ConfigMap	Functionの実行に必要な詳細な設定情報を格納
2	functioninfo.yaml	ConfigMap	Functionを実行するアクセラレータ種別や、Functionがサポートする接続方式に関する情報を格納
3	functiontype.yaml	FunctionType	FunctionのConfigとInfoから、OpenKasugai上で利用可能なFunctionを定義する
4	functionchain.yaml	FunctionChain	Functionの組み合わせを定義する

	実行後に各種リソースがOK又はRunning又はDeployedとなるまで待機する
	`$ kubectl get crd | grep example.com | cut -d ' ' -f 1 | paste -s -d ',' | \`
	`  xargs kubectl get -A`		

5	df-cpufunc.yaml	DataFlow	FunctionChainを指定して、DataFlowの配備を行う

結果確認手順

インプット、アウトプット確認手順

四則演算シナリオの実行は、gRPCによるデータ投入と、Podの標準出力の結果を取得することにより行う。

インプット実施手順

```
$ 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"}
```

アウトプット確認手順

```
$ 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}"}
```

FunctionChainのカスタマイズ手順

FunctionChainを変更することにより、四則演算シナリオで得られる実行結果が変わることを確認する。

`functionchain.yaml` を以下のように編集する
(send → plus → minus → rcv に変更した例)

```
$ 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"
```

デプロイ方法を実施しデプロイした後に、インプットを実行しアウトプットの確認を行う

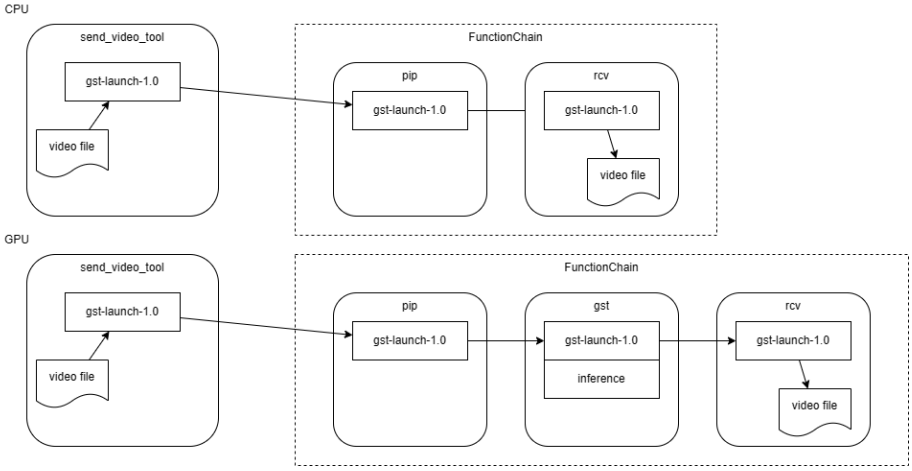
```
$ kubectl exec -it -n cpufunc-calcapp df-calcapp-wbfunction-send-cpu-pod -- /calcapp 1 2 3 4 5
Defaulted container "cpu-container0" out of: cpu-container0, cpu-container1
2024-12-13T07:03:54.634Z	INFO	workspace/main.go:104	load config	{"found": true, "config": {"Next":{"Host":"10.96.203.93","Port":8080}}}
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}"}
```

映像推論シナリオ

概要

映像推論シナリオは、CPU FunctionとGPU Functionの組み合わせにより実行する形で実装している。



本シナリオで扱うFunctionの概要を記載する

Function	機能
pip	映像転送Function 入力映像をそのまま次のFunctionに送信する
gst	映像推論Function 入力映像に推論処理を施しFunctionに送信する
rcv	映像受信Function 入力映像をそのままファイルに保存する

サンプルシナリオの内容

シナリオ説明

映像推論シナリオのサンプルとして、リポジトリに以下を格納している。複数の接続パターンの動作確認が行えるよう、シナリオを用意している。

配置先	シナリオ概要	備考
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p1c1	内部NW 接続パターン	クラスタ内部通信を用いて各Functionの接続を行う
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p1c2	外部NW 接続パターン	クラスタ外部通信を用いて各Functionの接続を行う
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p3c1	SR-IOV 接続パターン	SR-IOVを用いて各Functionの接続を行う
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c1	GPU連携 内部NW 接続パターン	クラスタ内部通信を用いて各Functionの接続を行う
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c2	GPU連携 外部NW 接続パターン	クラスタ外部通信を用いて各Functionの接続を行う
test/sample-data/sample-data-for-all-in-one/cpugpu-	GPU連携 SR-IOV 接続パターン	SR-IOVを用いて各Functionの接続を行う

func/p2c3		
test/sample-data/sample-data-for-all-in-one/cpugpu-func/p2c4	GPU連携 内部NW複数 接続パターン	クラスタ内部通信を用いて各Functionの接続を行うChainに複数のFunctionが登録されている

リソース、ConfigMap説明

各ディレクトリには以下ファイルを格納しており、それぞれ以下の役割を持つ

ファイル名	k8sリソース種別	説明
cm-cpufunc-config.yaml	ConfigMap	Functionの実行に必要な詳細な設定情報を格納
functioninfo.yaml	ConfigMap	Functionを実行するアクセラレータ種別や、Functionがサポートする接続方式に関する情報を格納
functiontype.yaml	FunctionType	FunctionのConfigとInfoから、OpenKasugai上で利用可能なFunctionを定義する
functionchain.yaml	FunctionChain	Functionの組み合わせを定義する
df-cpufunc.yaml	DataFlow	FunctionChainを指定して、DataFlowの配備を行う

イメージのビルド

gpu_infer_tcp_plugins のビルド

各ワーカーノードで、以下のディレクトリの README に従い gpu_infer_tcp_plugins のビルドを行う
この際、タグは `localhost/gpu_infer_tcp:1.0.0` とする事

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

rcv_video_tool のビルド
各ワーカーノードで、以下のディレクトリの README に従い rcv_video_tool のビルドを行う
この際、タグは `localhost/rcv_video_tool:1.0.0` とする事

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

各ワーカーノードでサイドカーのビルドを行う

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

各ワーカーノードで映像推論のビルドを行う

```
$ 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 .
```

デプロイ方法

配信コンテナコンテナの起動

send_video_toolのPODが起動していない場合、send_video_toolの起動を行う

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

DataFlowのデプロイは、リソース配置を以下の順で行うことで実施する。リソース配置には、 `kubectl apply -f` コマンドを使用する。

適用順	適用ファイル名	k8sリソース種別	説明
1	cm-cpufunc-config.yaml	ConfigMap	Functionの実行に必要な詳細な設定情報を格納
2	functioninfo.yaml	ConfigMap	Functionを実行するアクセラレータ種別や、Functionがサポ

			ートする接続方式に関する情報を格納
3	functiontype.yaml	FunctionType	FunctionのConfigとInfoから、OpenKasugai上で利用可能なFunctionを定義する
4	functionchain.yaml	FunctionChain	Functionの組み合わせを定義する

	実行後に各種リソースがOK又はRunning又はDeployedとなるまで待機する
	`$ kubectl get crd | grep example.com | cut -d ' ' -f 1 | paste -s -d ',' | \`
	`  xargs kubectl get -A`		

5	df-cpufunc.yaml	DataFlow	FunctionChainを指定して、DataFlowの配備を行う

結果確認手順

インプット、アウトプット確認手順

DataFlowに対する映像投入は、映像配信Podから映像データを送信することにより実施する。結果の出力は、データ受信Functionを実行するPod内に保存するmp4ファイルを再生することにより行う。

インプット実施手順

配信コンテナ内に入り映像データの送信を行う

映像データはホストの `/opt/DATA/video` 以下に配置する必要がある。このフォルダはコンテナ内で `/opt/video` としてマウントされる。以下の例はsample.mp4 を `/opt/DATA/video` に配置した場合である。

```
$ 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
```

`udpsink host` の部分は以下のようにIPアドレスの確認を行う

p1c1

`EXTERNAL-IP` を宛先とする

```
$ 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

p1c1と同様

p3c1

PODリソース内 `annotations` :: `k8s.v1.cni.cncf.io/network-status` 項目の `"name": "cpufunc-sample/sriov-ipam-config"` となるオブジェクトのIPアドレスを宛先とする

```
$ 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

p1c1と同様

p2c2

p1c2と同様

p2c3

p3c1と同様

p2c4

p1c1と同様

アウトプット確認手順

配信コンテナにて配信が完了した後、受信Function(POD)内にてgst-launch-1.0のプロセスを `kill -2` で終了させる

```
$ 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
```

受信Function外へ映像データをコピーする

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