



中国移动
China Mobile

Microservice UPF design experience and interactions with PaaS platform

Qihui Zhao
January 13, 2022

www.10086.cn

1 Overview

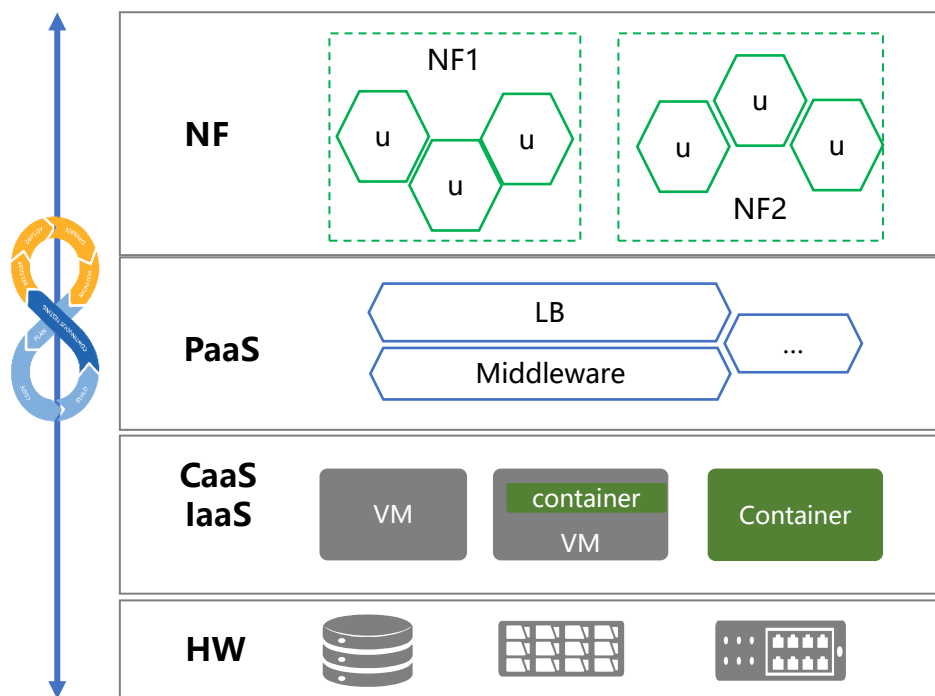
2 Microservice Design of UPF

3 PaaS capabilities for cloud native network functions

4 Conclusions

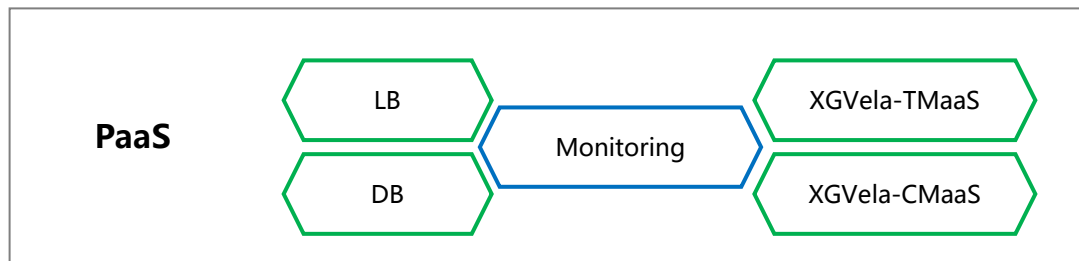
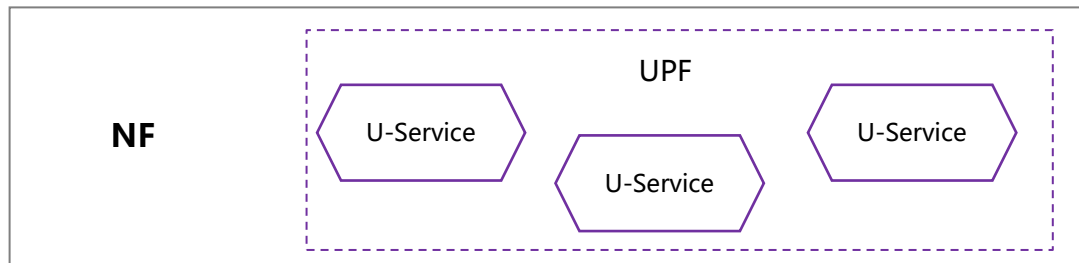
Background

- Cloud Native is the widely recognized direction of telecom network cloud evolution. Global telecom operators and vendors have been practicing ever since 2019. Container, microservice, DevOps are popular start points. But there is no standard answer on how to practice cloud native in real network cloud.
- To seek the answer to above question, from October to December 2021, **CMCC collaborated with Inspur**, completed a **cloud native experiment that covers both microservice design of NF and applying of PaaS platform**.



- **Why doing this?**
 - Going deeply into application lifecycle can help telecom operators know on which aspects we should involve cloud native.
- **Cut in point: NF microservice design and PaaS platform**
 - The primary object of cloud native evolution is application. It generates requirements of agility on design/development/delivery/operation process, while the platform provides capabilities to support those requirements, which is reflected in the left potential/partial architecture. So, application and platform are two important and tightly connected points that worth being researched together.
 - **Aligned with the goal of XGVela – Telecom PaaS platform for network cloud /network functions**

- The experiment chose UPF as the target NF, which is packaged as containers running on bare metal.
- The experiment mainly focuses on the **microservice design of UPF, common/reusable capabilities within UPF, and how UPF uses PaaS.**



1. Application Layer: **Promote microservice design of UPF**

- ✓ Redesign the monolithic UPF as microservice UPF based on telco microservice design principle
- ✓ Pick common and reusable capabilities from UPF microservices, and put them onto PaaS platform
- ✓ Microservice UPF integrates capabilities provided by PaaS platform to achieve complete functionalities

2. Platform Layer: **Develop/Verify PaaS capabilities according to UPF requirements**

- ✓ Develop Telco PaaS capabilities (e.g. LB, DB) to satisfy special protocol processing requirement and high reliability requirement
- ✓ Integrate General PaaS capabilities (e.g. Monitoring)
- ✓ Verifying existing XGVela seed code functionalities (CMaaS and TMaaS)

1 Overview

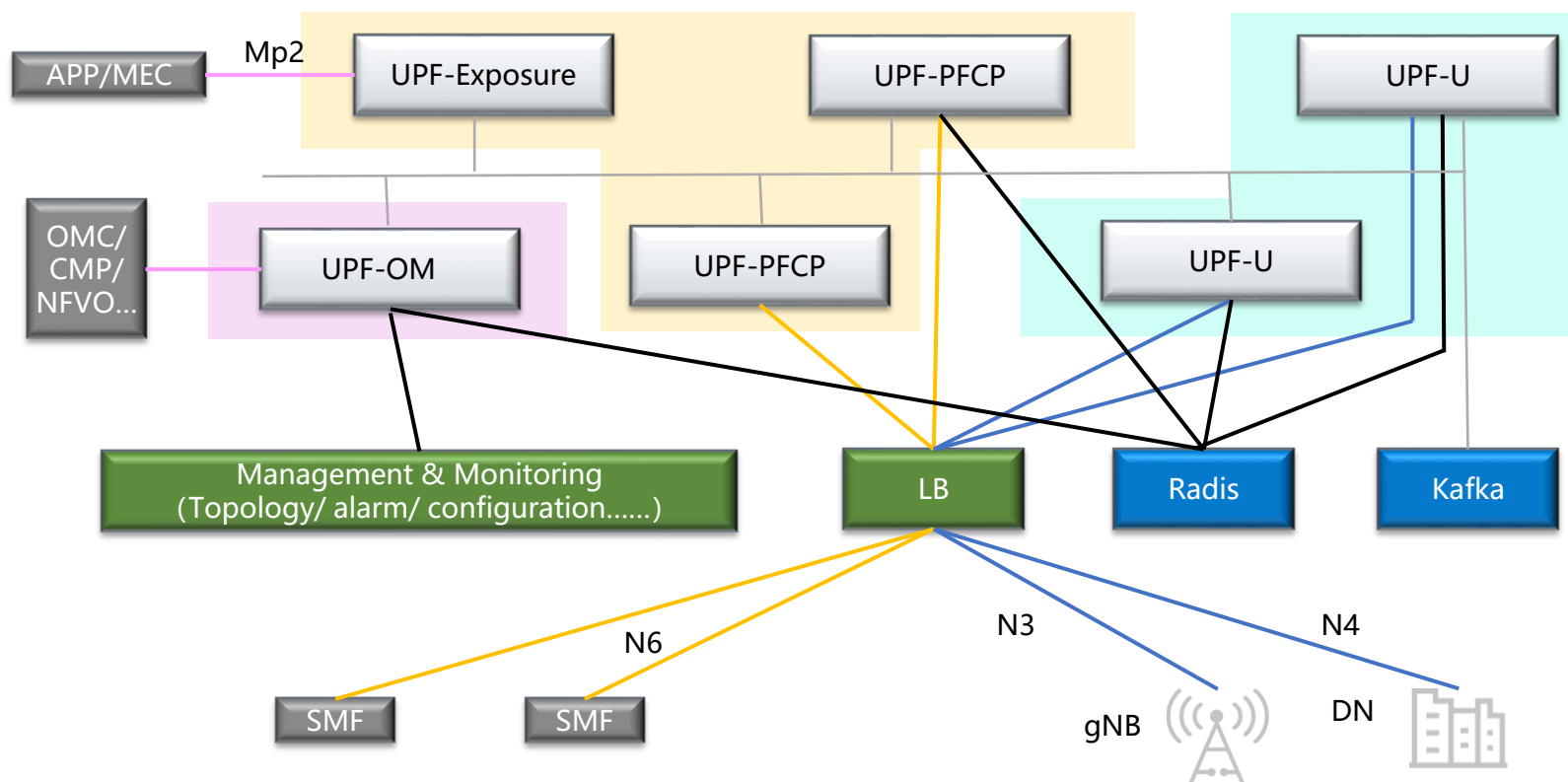
2 **Microservice Design of UPF**

3 PaaS capabilities for cloud native network functions

4 Conclusions

Microservice Design of UPF

- Microservice design of UPF maintains all 3GPP defined interfaces and refers to classical design paradigm of CT applications for internal functionality block design.
- It follows “Single responsibility principle”, and separate UPF functions into three major types: management plane service, control plane service, and data plane service.



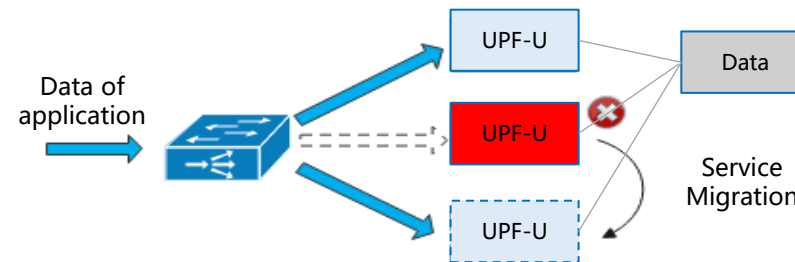
Microservice design principle:

- ✓ External interfaces follows 3GPP requirements
- ✓ Separate microservice into management plane, control plane and data plane
- ✓ Separate interface module and processing module
- ✓ Stateless
- ✓ Use container and K8S

Benefit of Microservice UPF

High availability

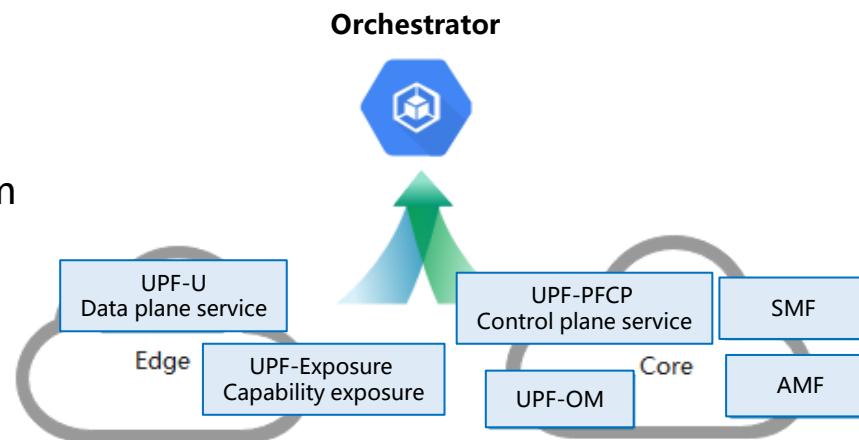
Cloud native UPF separates data processing units and data storage. When service instance fails, the user traffic can be quickly moved to running service instances.



Elastic Deployment

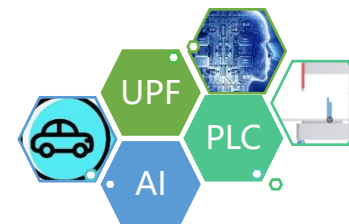
Applications in edge manufacturing industry has different requirements on data processing delay and throughput. And the compute power on edge is different from site to site.

Microservice UPF can be orchestrated and dispatched among edge and core sites based on application requirements and resource conditions.



Customized Service

The separation of interface module and processing module make it easy to create new services through service combination and new interface creation.



1 Overview

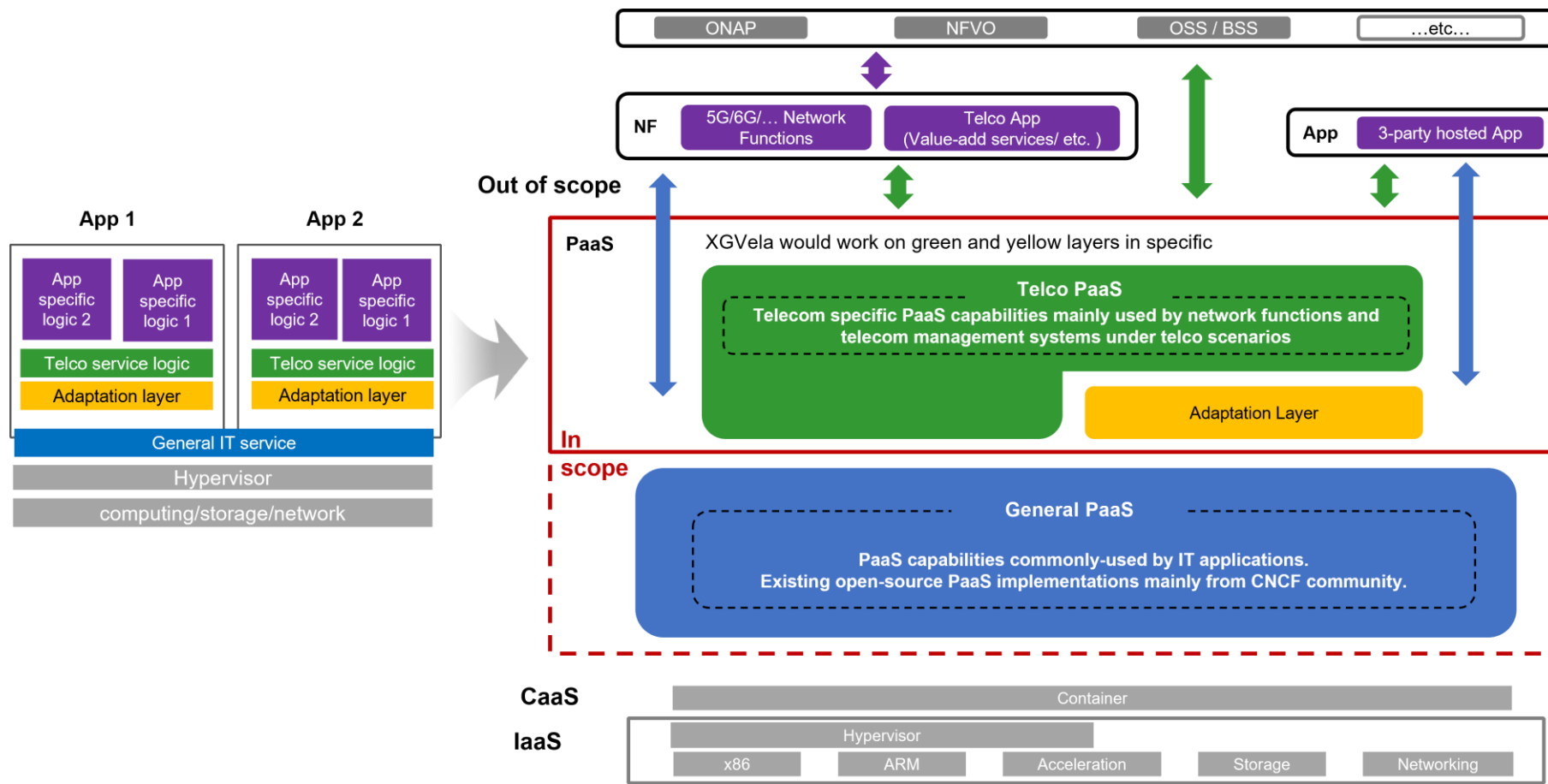
2 Microservice Design of UPF

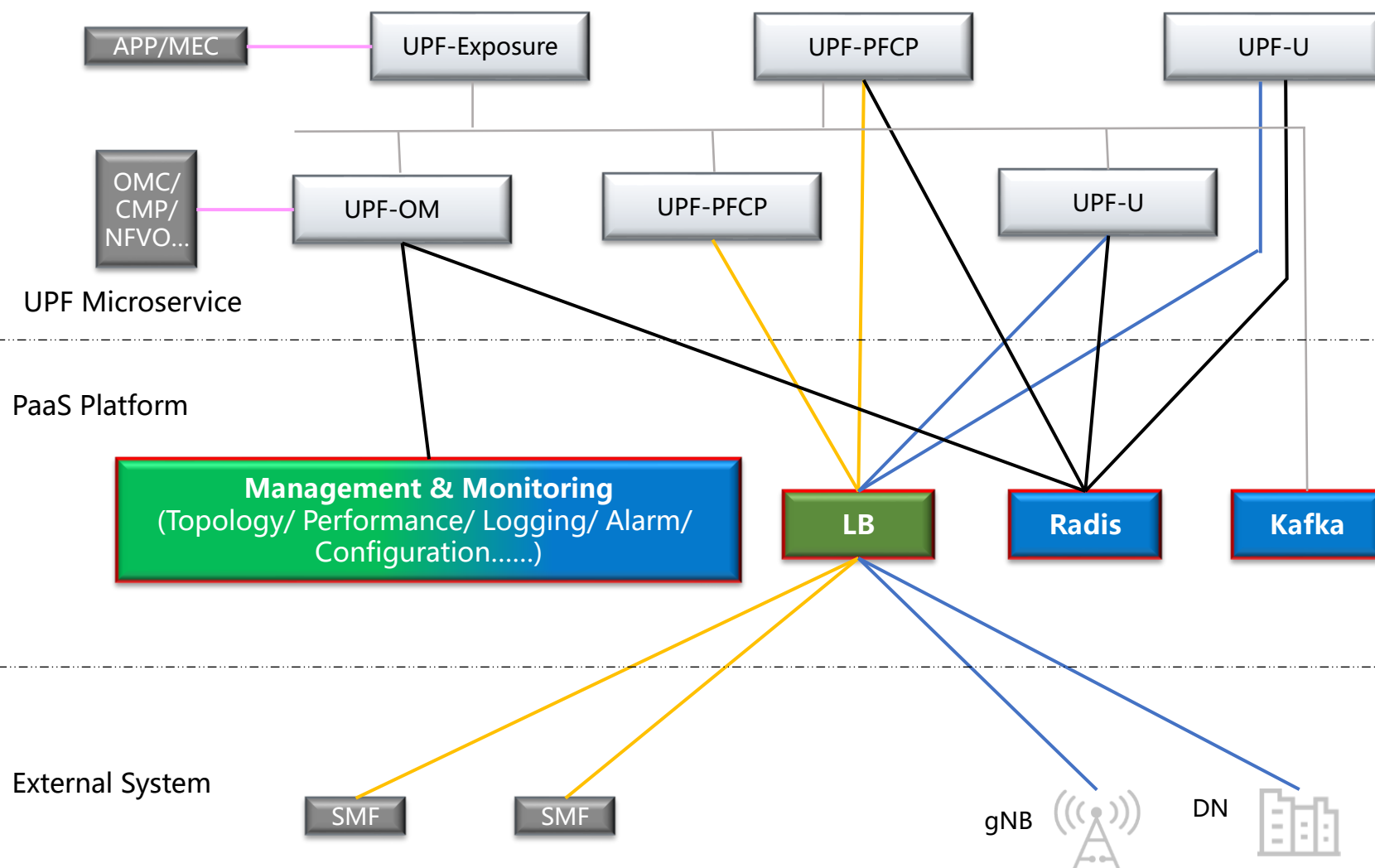
3 **PaaS capabilities for cloud native network functions**

4 Conclusions



- XGVela was launched in Linux Foundation in April 2020, is now LFN Sandbox project.
- XGVela aims to construct a **telecom cloud native PaaS platform**





Management & Monitoring

- ✓ Topology, performance, logging, alarm, configuration.....
- ✓ **Highly reusable functions in development**
- ✓ **Provide local & lightweight management**

Messaging

- ✓ Internal communication between different microservice instance

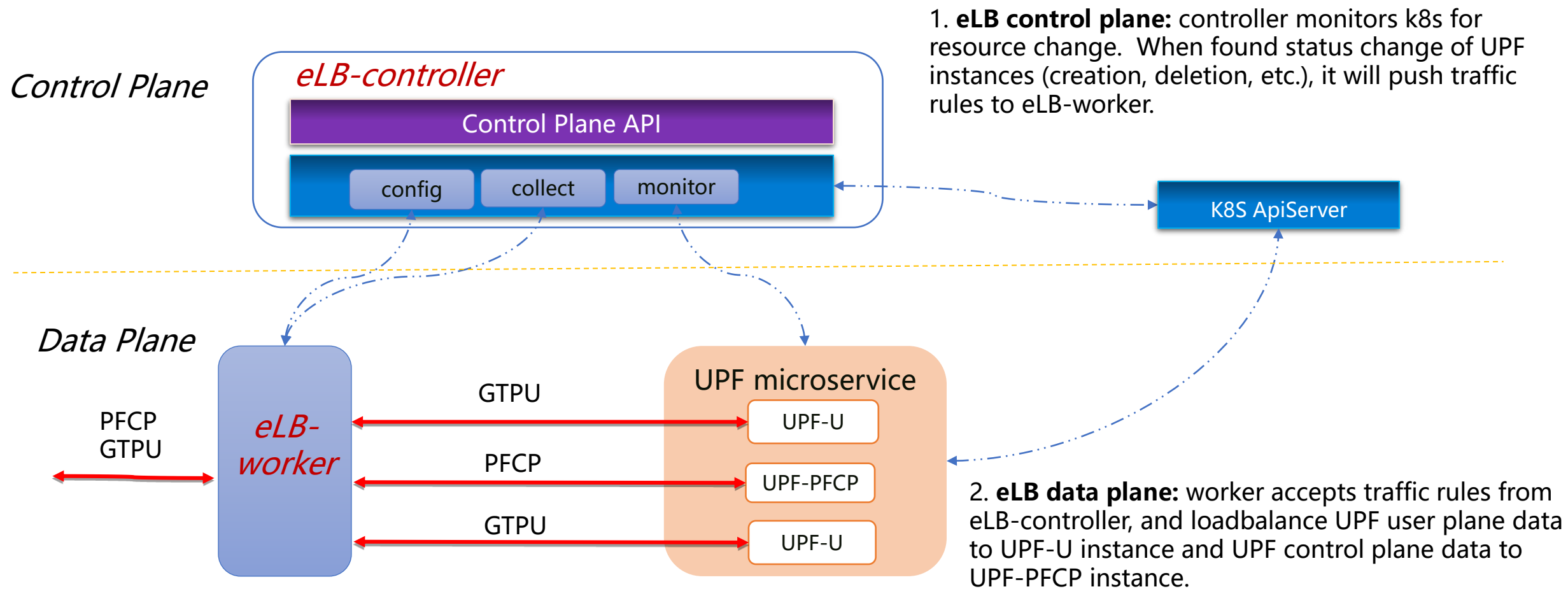
High performance DB

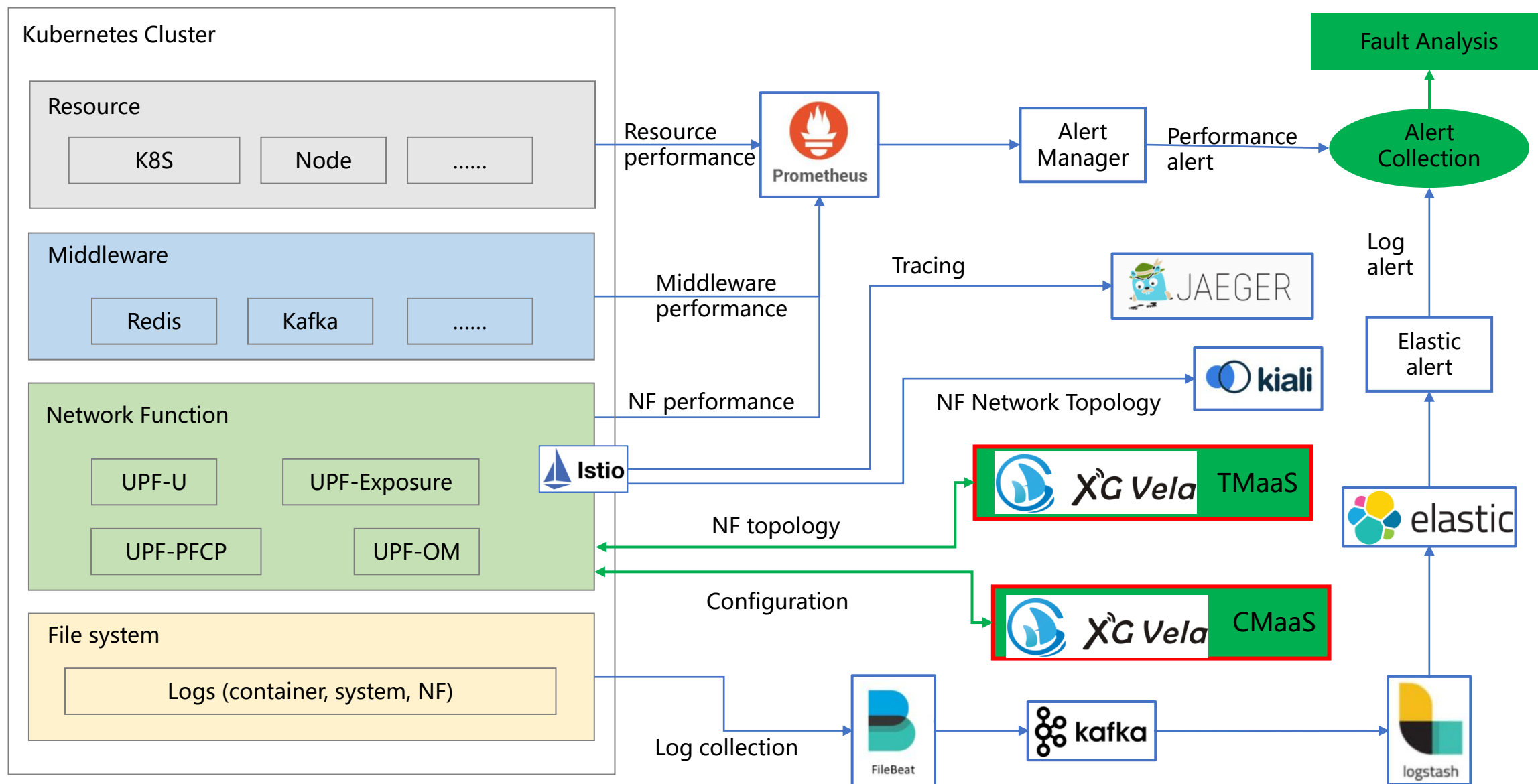
- ✓ Redis cluster to store PDU session

Enhanced LB

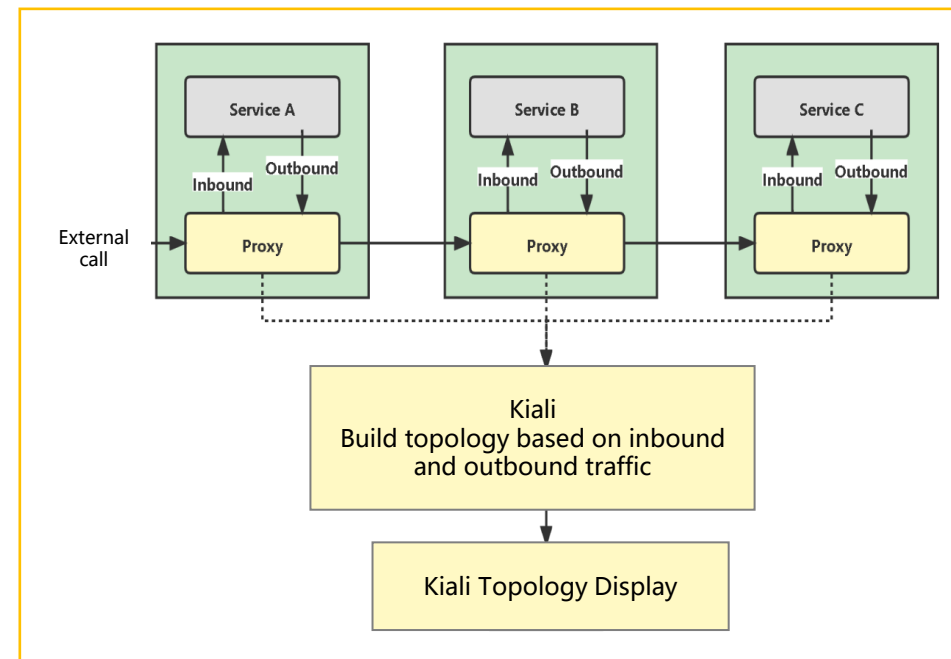
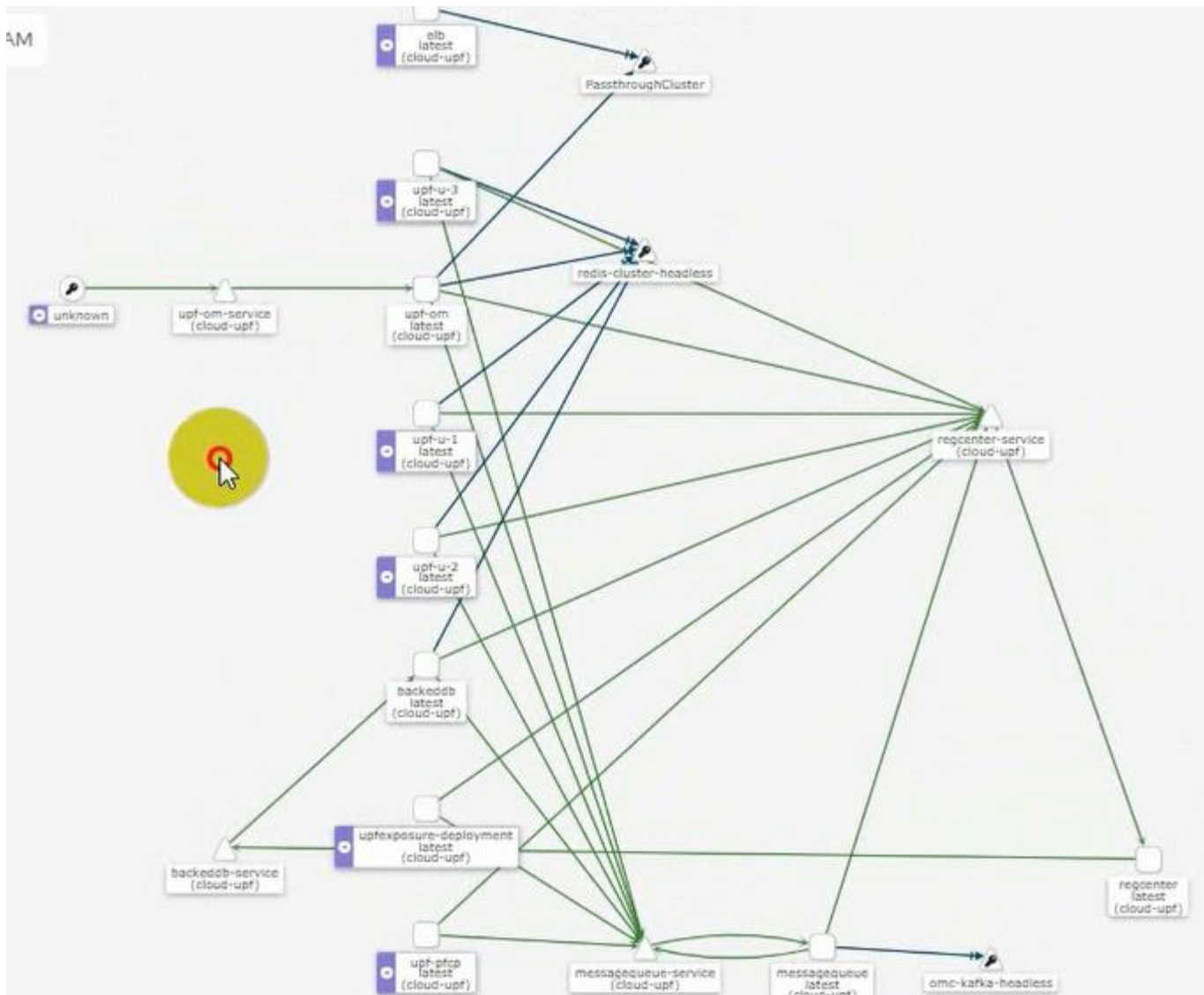
- ✓ Load balancing among microservice instances

- General load balancer can neither recognize CT protocols like PFCP or GTP, nor stably loadbalance PDU sessions to fixing backend processing unit based on UE and session features.
- Enhanced LB is designed to solve the weaknesses of general load balancer.



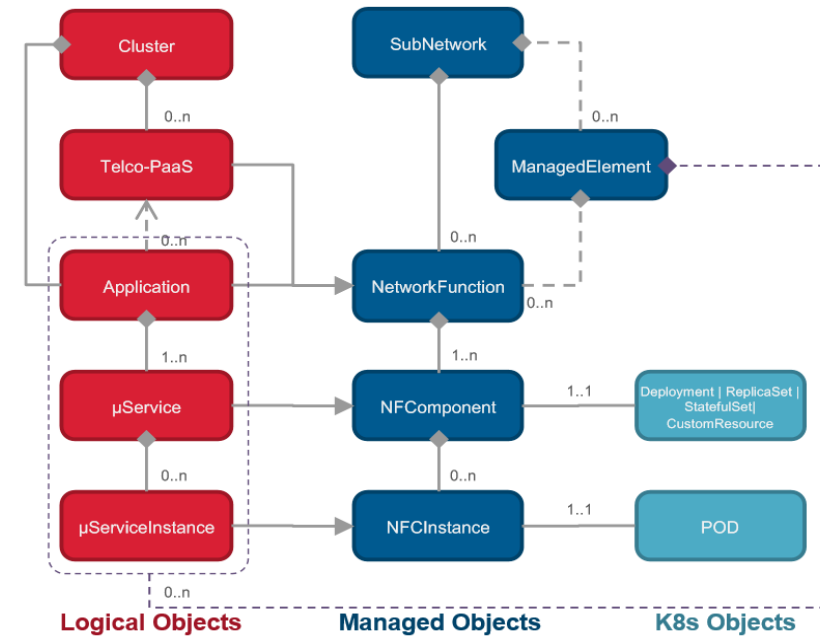
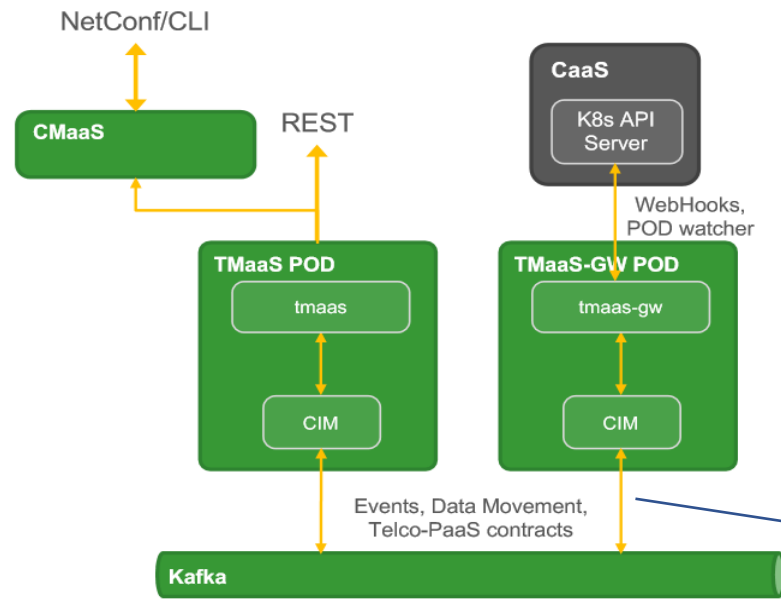


Solution 1: General PaaS – Istio + Kiali



1. Deploy Istio proxy together with UPF microservices as sidecar.
2. Traffic going in and out microservice instance should go through proxy.
3. Proxy analyze the incoming and outgoing traffic, get the source IP and destination IP of the traffic and report to Kiali.
4. Kiali analyze the microservice interaction relationship based on IP information, generate microservice topology and display.

Solution 2: Telco PaaS –XGVela TMaaS



POD annotation

```

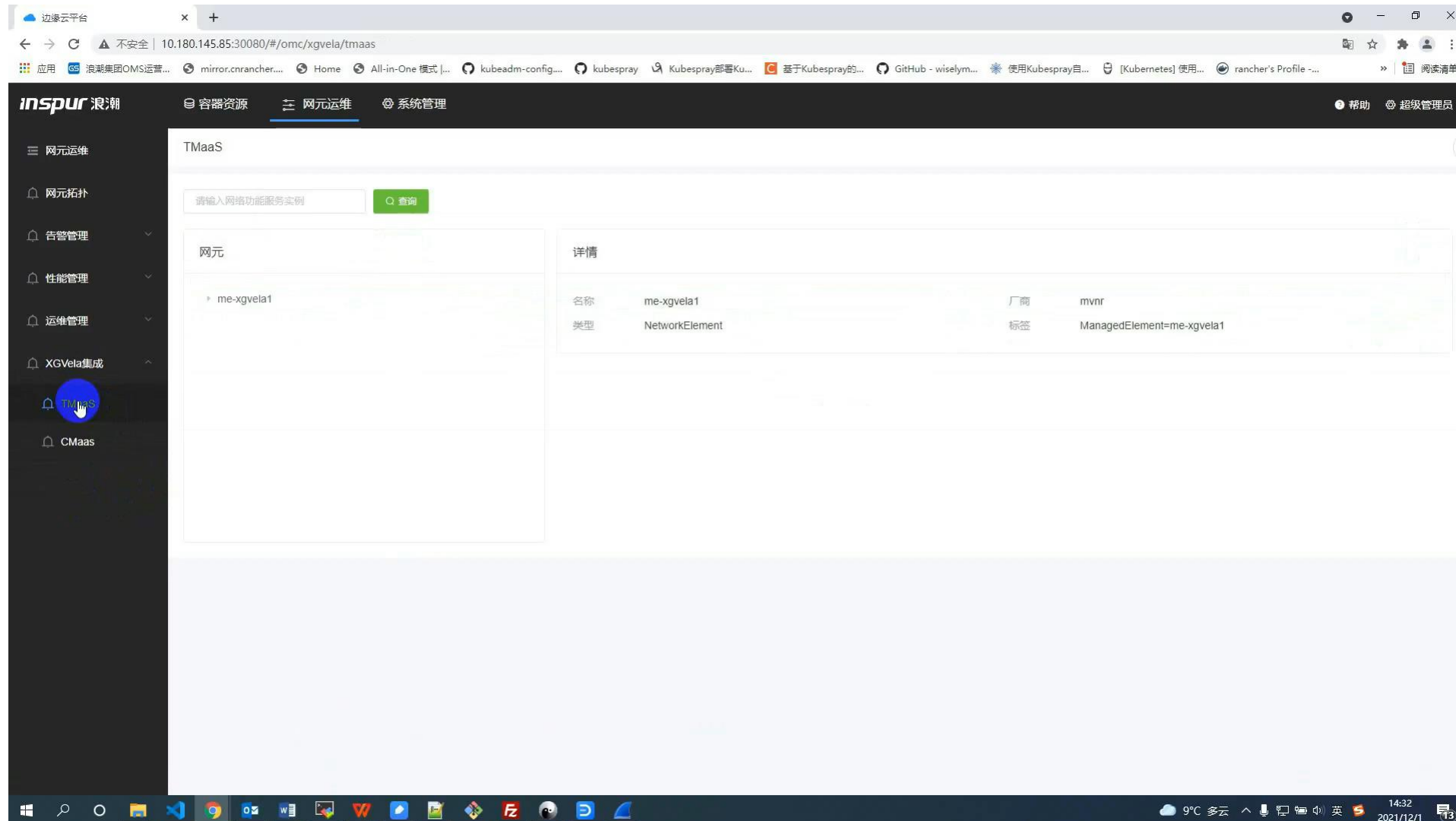
xgvela.org/tmaas {
  telcoPaasId: <>
  dnPrefix: <>
  nfId: <>
  nfType: <>
  vendorName: <>
}
    
```

```

pushToKafka(
  new PodDetails(action, podName, namespace, nfName, nfType, nfServiceName,
    nfServiceType),nfName);
    
```

1. Developer adds TMaaS annotation to UPF microservice yaml files (Deployment, StatefulSet, CustomerResource, etc.)
2. TMaaS-GW is a k8s client, and listens to the k8s PODs changing EVENT. When it find the EVENT's POD annotations contains NF_ID, NF_TYPE, NF_SERVICE_ID and NF_SERVICE_TYPE, it will prepare the PodDetails and send it to Kafka.
3. TMaaS listen to KAFKA EVENT' s according to pre-defined TOPIC, and then generate logic topology tree based on PodDetails.
4. Every update of Pod (adding/ deleting) will cause the update of topology tree simultaneously.

Solution 2 Demo: Telco PaaS –XGVela TMaaS



The screenshot displays the Inspur TMaaS web interface. The browser address bar shows the URL `10.180.145.85:30080/#/omc/xgvela/tmaas`. The interface includes a navigation menu on the left with options like '网元运维', '网元拓扑', '告警管理', '性能管理', '运维管理', 'XGVela集成', and 'CMAas'. The main content area is titled 'TMaaS' and features a search bar with the text '请输入网络功能服务实例' and a '查询' button. Below the search bar, there are two panels: '网元' (Network Element) and '详情' (Details). The '网元' panel lists a single entry 'me-xgvela1'. The '详情' panel displays the following information:

名称	me-xgvela1	厂商	mvnr
类型	NetworkElement	标签	ManagedElement=me-xgvela1

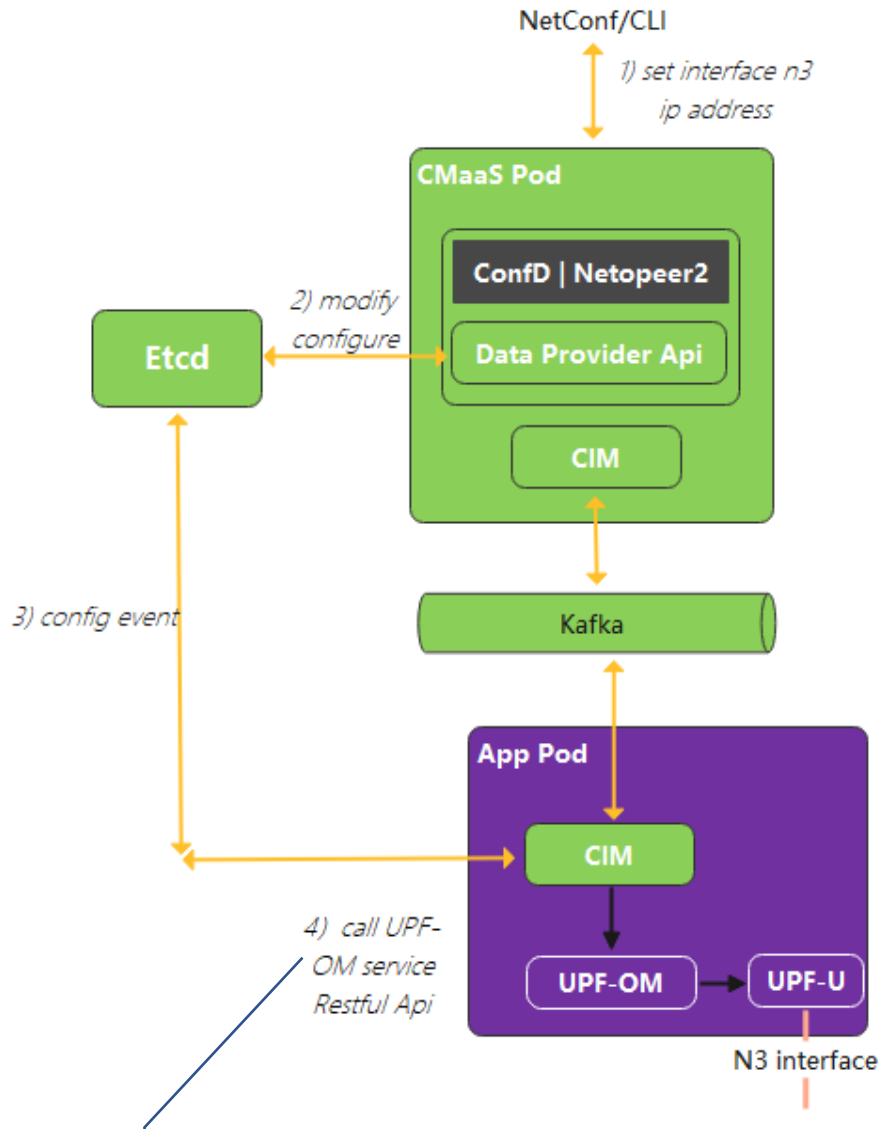
The Windows taskbar at the bottom shows the system time as 14:32 on 2021/12/1.

Solution 1 VS Solution 2

Istio + Kialis	TMaaS
<p>Dynamic topology</p> <ul style="list-style-type: none"> • Topology is generated based on traffic between microservices. • If no workload carried and no interactions happened within NF, then no topology would be generated. 	<p>Static topology</p> <ul style="list-style-type: none"> • Topology is generated based on pod lifecycle event. • As long as the status of pod (with annotation) changed, topology can be updated.
<p>Change the deployment file</p> <ul style="list-style-type: none"> • Add proxy as sidecar to UPF deployment files. • Update iptables. 	<p>Change the deployment file</p> <ul style="list-style-type: none"> • Add annotation to UPF deployment files.
<p>Protocol sensitive</p> <ul style="list-style-type: none"> • As all of UPF traffic will go through proxy, it either requires proxy can recognize all telco protocol, or only use istio proxy for non-telco-protocol traffic. • Existing open-source Istio+Kiali solution cannot provide complete topology due to protocol limit (PFCP, Diameter, SCTP.....). 	<p>Protocol insensitive</p> <ul style="list-style-type: none"> • Topology is generated through static pod information, whose description format is standard in K8S, and the format of annotation is standardized by TMaaS.
<p>Post-topology</p> <ul style="list-style-type: none"> • Topology generated through analysis on real-time traffic between microservices. 	<p>Pre-topology</p> <ul style="list-style-type: none"> • Topology generated through pre-defined annotations.

- TMaaS can help create “static” topology through pod annotations without limitations of protocol recognition.
- TMaaS solution and Istio+Kiali solution can be use together to create complete and real-time topology.

Solution: Telco PaaS –XGVela CMaaS



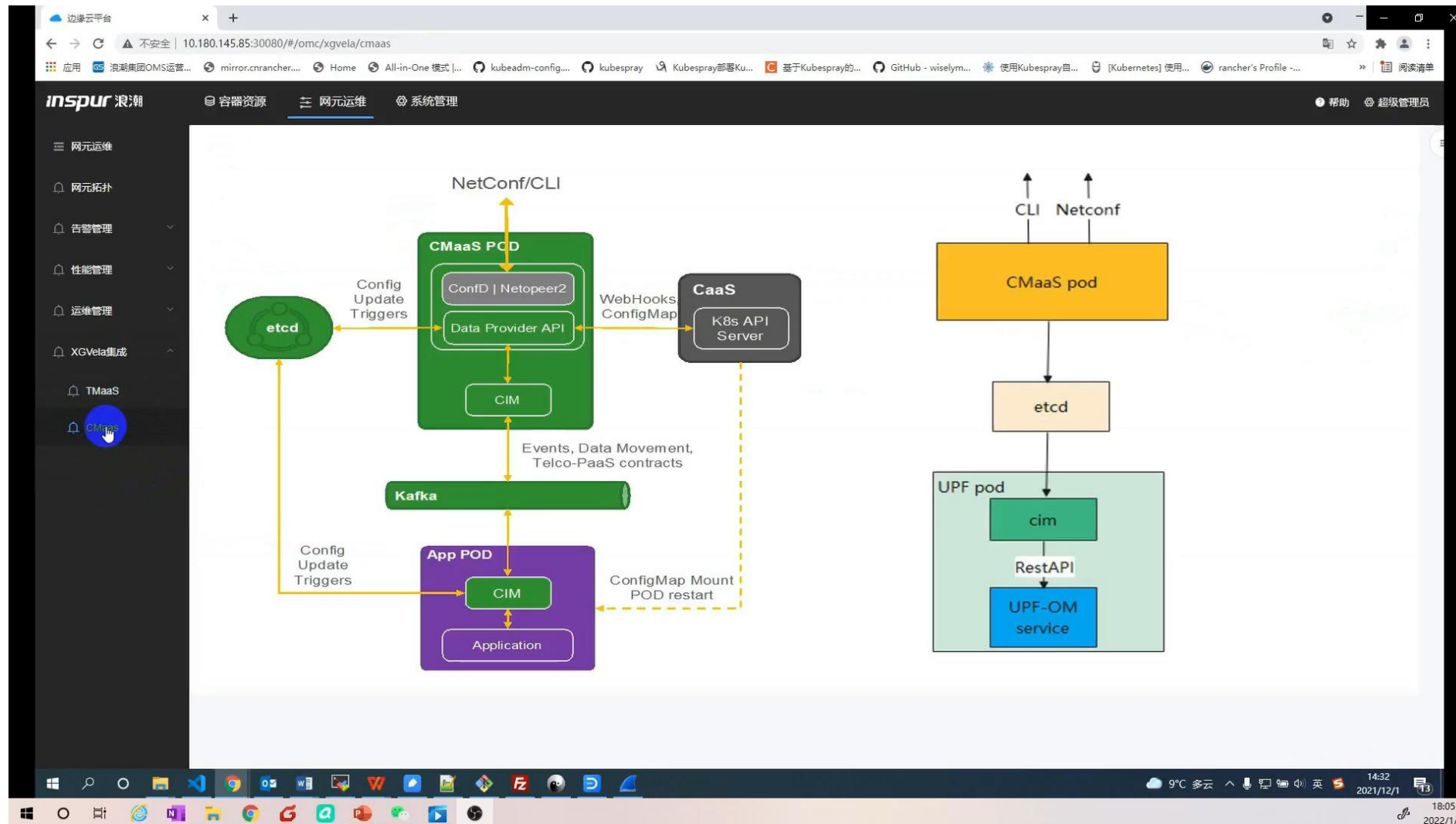
CMaaS integration:

- UPF-OM implements a configuration API to communicate with CIM
- CIM is deployed together with UPF-OM as sidecar
- Set UPF configuration through CMaaS northbound API through CLI (e.g. set the upf interface status or IP of interface)
- CMaaS will translate the request body into key-value configmap and store into etcd
- CIM will monitor the configuration change in etcd
- CIM will call the UPF-OM configuration API to update configuration

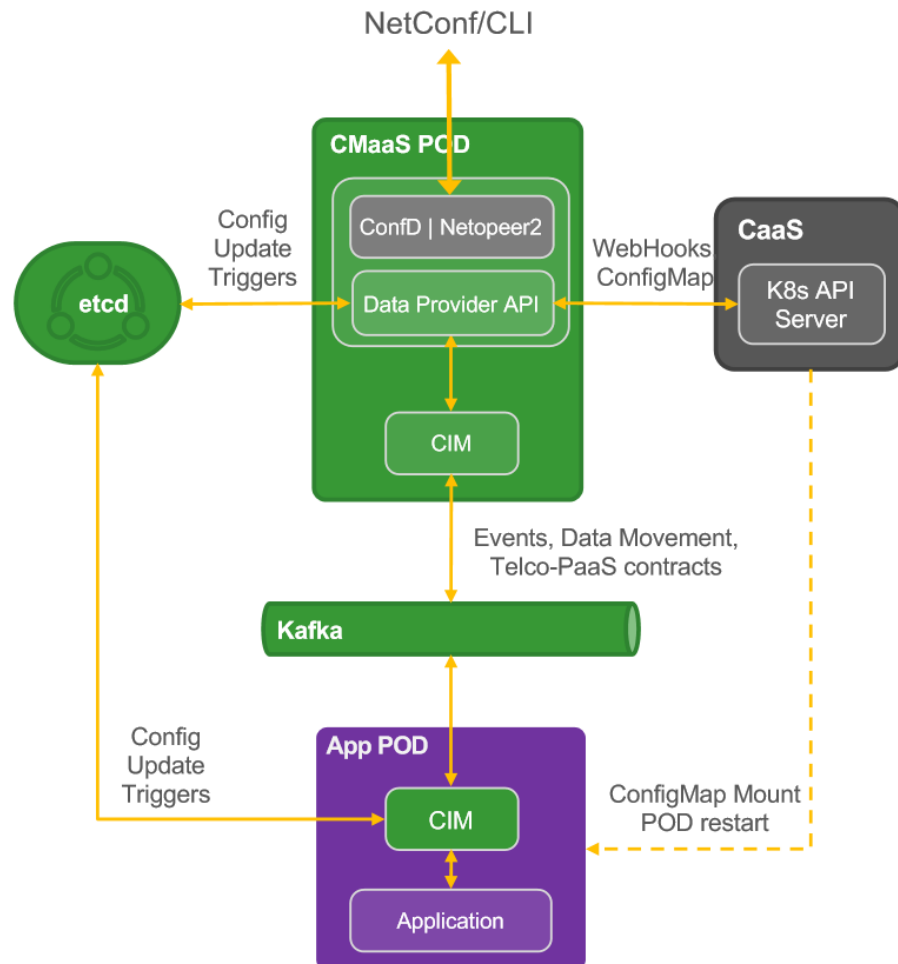
CMaaS northbound API:

	data Type	Cardinality	Remarks
Request body	data-key	1	Specifies the data key Eg: "interfaces"
	config-patch	1	config information Eg: "[{ "op" : "replace" , "path" : "/upf-interfaces/eht0/status" , "value" : "UP" }]"
	revision	1	revision number
Response body	status	1	200 ok 4xx error
	ProblemDetails	0..1	if status is 4xx

Solution Demo: Telco PaaS –XGVela CMaaS



Advantage of Telco PaaS –XGVela CMaaS



- CNF and uService configuration are stored in ConfigMap. Addresses Day-0, 1 delivery.
- CMaaS discovers new CNF and configuration changes using k8s service discovery mechanism.
- On new CNF deployment, loads any management configuration yang and json from ConfigMap and provisions the NetConf server module (not implemented yet).
- **Day-2 configuration changes are delivered via k8s rolling update or by direct API calls to application containers via etcd and CIM per application need.**

1 Overview

2 Microservice Design of UPF

3 PaaS capabilities for cloud native network functions

4 Conclusions

Microservice related conclusions

1. Telco devices naturally have distributed characteristics. It is good to continually follow the classic NF architecture, which may contain LB, OAM, storage, processing unit.
2. Designing NF into smaller microservices is also good for service flexibility and new service customization. Microservice design principle can refer to Page 6.
3. Theoretically, telco operators only care about NF functionalities, interfaces and running status, without caring about internal implementation. But as potential requirements on precise management and orchestration may emerge in the future, it is possible to standardize some of the NF implementation, but what should be standardized is not clear.

PaaS related conclusions

1. PaaS, especially open-source PaaS functionalities, can effectively help to simplify NF development and standardize operator management.
2. Currently, four types of PaaS capabilities are common PaaS for NFs. Please refer to page 10 for details.
3. NF logic related PaaS functionalities (LB, DB) usually be integrated at development phase and packaged together with NF image. O&M functionalities are loosely-coupled with NF logic, can be integrated at development phase while deployed on-demand with independent images after NF deployment.
4. Telecom protocol, telecom reliability and telecom operating habits are different from IT services, which may require telco enhancement /implementation on PaaS capabilities.

Whether the above conclusions are right is worth further investigation. If you are interested in researching cloud native evolution of telecom network cloud, please join us in XGVela and the future experiment.

Thanks!