MEC Service Federation for Location-aware IoT with DevOps MEC Infra Orchestration

ETSI – LF Edge Hackathon
Team DOMINO
Solution Submission

Oleg Berzin, oberzin@equinix.com
Vivekanandan Muthukrishnan, vmuthukrishnan@aarnanetworks.com

September 2022

Introduction

In our solution we use Akraino Public Cloud Edge Interface (PCEI) blueprint to demonstrate orchestration of federated MEC infrastructure and services, including 5G Control and User Plane Functions, MEC and Public Cloud IaaS/SaaS, across two operators/providers (a 5G operator and a MEC provider), as well as deployment and operation of end-to-end cloud native IoT application making use of 5G access and distributed both across geographic locations and across hybrid MEC (edge cloud) and Public Cloud (SaaS) infrastructure.

We first demonstrate a solution to critical issues of deployment and interconnection of MEC Federation, such as activation of physical bare metal servers, deployment of operating systems and virtualization layers (Docker/K8s) on the servers, interconnection of participants of MEC Federation using a production global network fabric and virtual networking functions as well as activation of public cloud SaaS and MEC-to-Cloud interconnection for extending MEC Federation to cloud resources.

We then show deployment of to-be-federated (or to-be-shared) services in the respective domains of the two operators by deploying a cloud native implementation of ETSI MEC Location API server (MEC013) in the 5G operator's domain, and a cloud native IoT Edge Gateway (based on Azure IoT Edge) in the MEC provider's domain.

Finally, at the IoT application layer, we provide a reference IoT client emulating several sensors and capable of communicating with a cloud native IoT Edge Gateway across the federated infrastructure using low power encoding for IoT messages (temperature, humidity, and pressure), as well as the interaction between the distributed IoT Edge Gateway and the Location API service to enable insertion of obtained UE location data into IoT sensor data showing location-aware IoT across federated MEC infrastructure.

By orchestrating, bare metal servers and their software stack, 5G control plane and user plane functions, interconnection between the 5G provider and MEC provider, connectivity to a public cloud as well as the IoT application and the MEC Location API service, we show how it is possible for providers to enable sharing of their services in a MEC Federation environment.

Summary of contributions and innovations

In this solution we provide the following contributions and innovations:

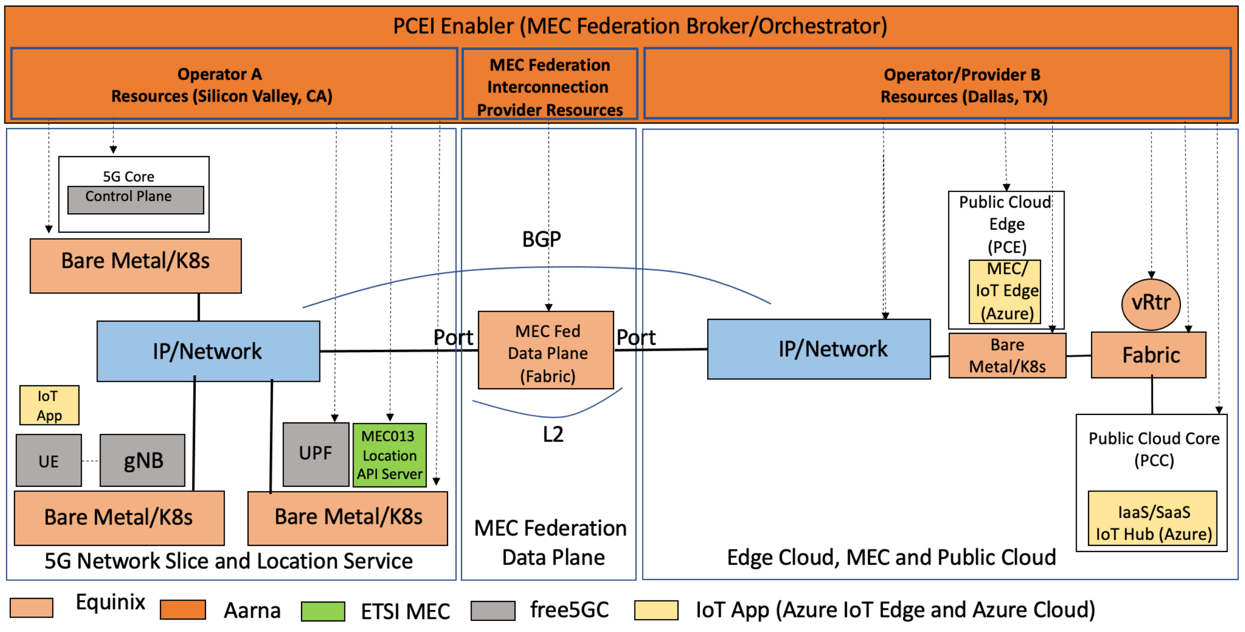
- A practical use case showing a realization of ETSI MEC Federation architecture
- An introduction and a functioning demonstration of MEC Federation Data Plane
- Implementation of the GSMA OPG Edge Node sharing scenario using MEC Federation
- Implementation of ETSI MEC Location API Service and its integration with a MEC application
- Implementation of a combined MEC Federation Broker and MEC Orchestrator with unique capabilities for infrastructure orchestration in multiple domains such as public cloud, edge/MEC cloud, network operator, 5G control plane and user plane cloud native function deployment as well as cloud native service and application deployment
- Implementation of integrated Terraform Infrastructure-as-Code module into the orchestrator enabling DevOps infrastructure orchestration
- Implementation of integrated Ansible Infrastructure Configuration and Installation module into the orchestrator
- Cloud native 5G Control Plane and Distributed UPF deployment design and the correspondent Helm Charts
- Use of production services (by Equinix) such as bare metal cloud, virtual network functions, public cloud access and a global interconnection fabric as dynamically orchestratable infrastructure components for the realization of the MEC Federation use case
- Implementation of a reference IoT client
- Implementation of a custom software module for Azure IoT Edge that enables its integration with ETSI MEC Location API service
- An end-to-end demonstration of the infrastructure orchestration, 5G control plane and user plane functions deployment, ETSI MEC Location API service deployment and the location aware, distributed IoT application operation

Use case description

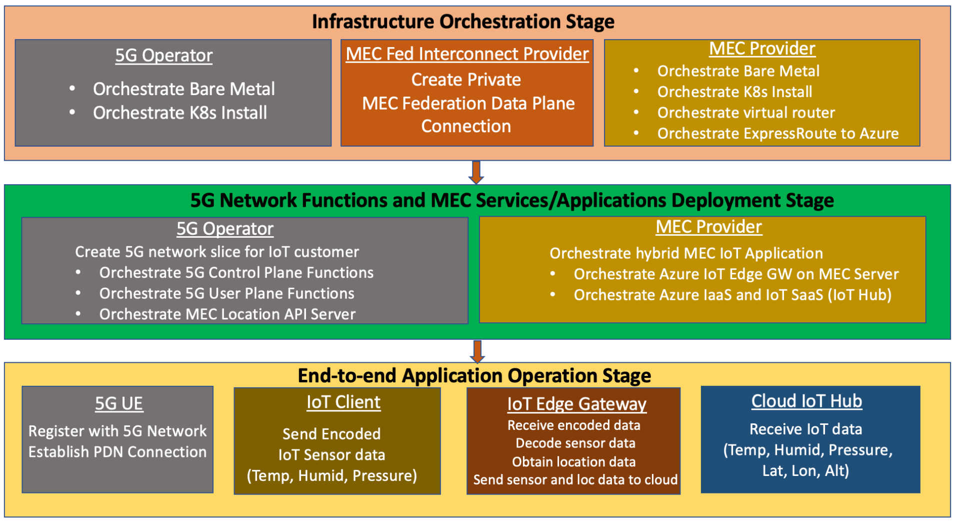
Our use case involves a 5G operator offering 5G access in the Silicon Valley, CA area and a MEC operator offering an edge cloud service for edge applications as well as connectivity to public clouds in the Dallas, TX area.

Both operators/providers use services of a MEC Federation Interconnection Provider (MFIP) to enable several critical functions such as interconnection between the 5G operator domain and the MEC provider domain using a global private interconnection fabric (also referred to as MEC Federation Data Plane), colocation services to host bare metal compute resources as well as private connectivity to public clouds using a virtual network function (VNF) service integrated with the private interconnection fabric.

In our scenario, the MFIP also runs an orchestration service enabling the 5G operator and the MEC provider to activate their respective infrastructure and interconnection components and a subsequent deployment of functions, services, and applications. The scenario is shown below.


PCEI Enabler (MEC Federation Broker/Orchestrator)
Operator A Resources (Silicon Valley, CA)
MEC Federation Interconnection Provider Resources
Operator/Provider B Resources (Dallas, TX)
5G Core Control Plane
Bare Metal/K8s
IP/Network
IoT App
UE
gNB
Bare Metal/K8s
UPF
MEC013 Location API Server
Bare Metal/K8s
5G Network Slice and Location Service
BGP
Port
MEC Fed Data Plane (Fabric)
Port
L2
MEC Federation Data Plane
IP/Network
Public Cloud Edge (PCE)
MEC/ IoT Edge (Azure)
Bare Metal/K8s
vRtr
Fabric
Public Cloud Core (PCC)
IaaS/SaaS IoT Hub (Azure)
Edge Cloud, MEC and Public Cloud
Equinix
Aarna
ETSI MEC
free5GC
IoT App (Azure IoT Edge and Azure Cloud)

Our use case then proceeds in stages as shown and described below.



Infrastructure Orchestration Stage

5G Operator
- Orchestrate Bare Metal
- Orchestrate K8s Install

MEC Fed Interconnect Provider
Create Private MEC Federation Data Plane Connection

MEC Provider
- Orchestrate Bare Metal
- Orchestrate K8s Install
- Orchestrate virtual router
- Orchestrate ExpressRoute to Azure

5G Network Functions and MEC Services/Applications Deployment Stage

5G Operator
Create 5G network slice for IoT customer
- Orchestrate 5G Control Plane Functions
- Orchestrate 5G User Plane Functions
- Orchestrate MEC Location API Server

MEC Provider
Orchestrate hybrid MEC IoT Application
- Orchestrate Azure IoT Edge GW on MEC Server
- Orchestrate Azure IaaS and IoT SaaS (IoT Hub)

End-to-end Application Operation Stage

5G UE
Register with 5G Network
Establish PDN Connection

IoT Client
Send Encoded IoT Sensor data (Temp, Humid, Pressure)

IoT Edge Gateway
Receive encoded data
Decode sensor data
Obtain location data
Send sensor and loc data to cloud

Cloud IoT Hub
Receive IoT data (Temp, Humid, Pressure, Lat, Lon, Alt)

The infrastructure and interconnection orchestration stage:

1. The 5G operator accesses the orchestration service offered by the MFIP to deploy a 5G network slice for their IoT customer in the Silicon Valley, CA area (the left side of the above diagram):
    a. Deploy bare metal servers, using a bare metal cloud offered by a colocation provider (we use a production Equinix Metal service in our demonstration). This includes a server for 5G Control Plane Functions (based on Free5GC) and a server for the User Plane Function. Note that the local IP/Network connectivity is orchestrated as part of the bare metal orchestration service.
    b. Install Kubernetes on the bare metal servers and register their Kubernetes clusters with the MEC Federation orchestration service.
2. The MEC provider accesses the orchestration service offered by the MFIP to deploy the edge cloud and the private connectivity to the public cloud in the Dallas, TX area (the right-hand side of the picture):
    a. Deploy bare metal servers, using a bare metal cloud offered by a colocation provider (we use a production Equinix Metal service in our demonstration). This includes a server for the edge cloud/MEC. Note that the local IP/Network connectivity is orchestrated as part of the bare metal orchestration service.
    b. Install Kubernetes on the bare metal server and register their Kubernetes clusters with the MEC Federation orchestration service.
    c. Deploy a Virtual Network Function (VNF) to enable access to multiple public clouds from the edge cloud/MEC server. Note that in our demonstration we use a production Network Edge service offered by Equinix to deploy a Cisco CSR1000v virtual router VNF) and Azure public cloud. Also note that we use private connectivity between the VNF and the Azure cloud using Azure ExpressRoute with BGP routing across Equinix Fabric.
3. The 5G operator accesses the orchestration service offered by the MFIP to create a private MEC Federation Data Plane connection to the MEC provider:
    a. The connection is orchestrated using a production Equinix Fabric service between data centers in Silicon Valley, CA and Dallas, TX. Note that BGP routing between the 5G provider's network and the MEC operator's network is also orchestrated as part of the connection creation.

The 5G network functions and MEC services/applications deployment stage:

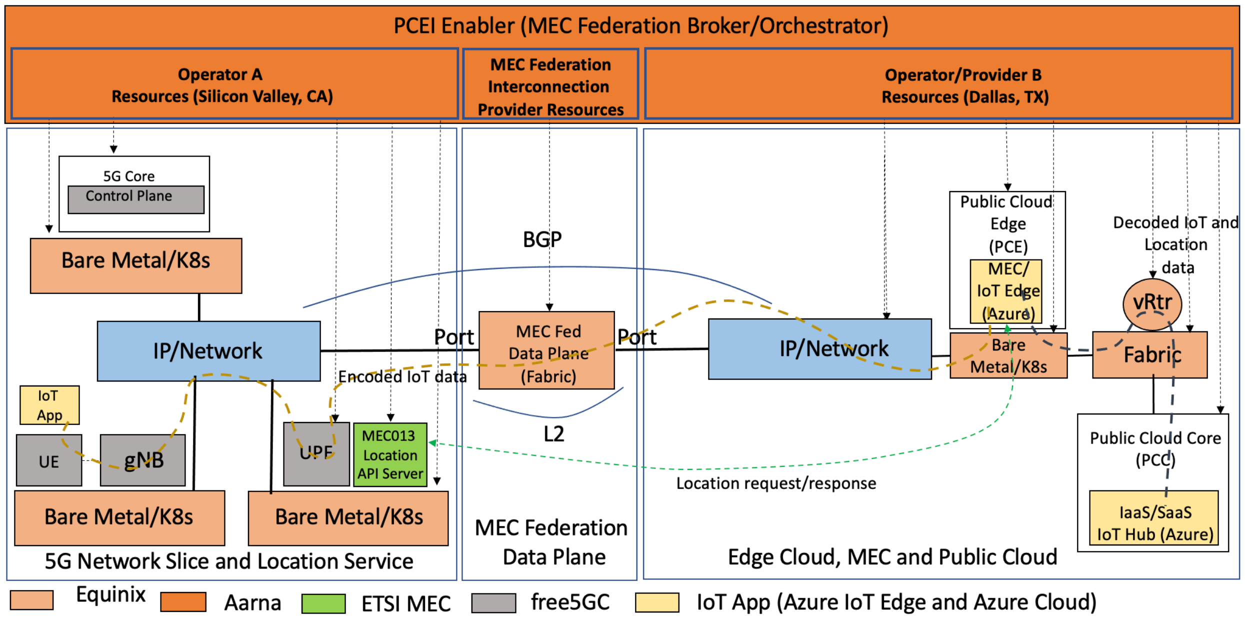
1. The 5G operator deploys a network slice for their IoT customer:
    a. Deploy a 5G network slice for the IoT customer by deploying cloud native 5G Control Plane and User Plane Functions (based on Free5GC) on respective Kubernetes clusters/servers activated in the Silicon Valley data center.
    b. Enable access for 5G UEs provisioned with low power IoT clients to the customer's network slice. Note that in our demonstration we use a simulated gNB, a simulated UE and a simulated 5G New Radio network (based on Free5GC).
2. The 5G operator deploys MEC Location API server as part of the customer's network slice:
    a. The cloud native implementation of the MEC013 Location API server is deployed by accessing the MEC Federation orchestrator and placed on the same cluster/server that is hosting the customer's 5G UPF.
3. The MEC provider deploys a hybrid MEC application on the edge cloud and public cloud for the use by the 5G operator's IoT customer:
    a. Deploy Azure IoT Edge cloud native IoT Gateway on the edge cloud/server.
    b. Activate IaaS and IoT SaaS components in Azure Cloud:
        i. IaaS components: Azure ExpressRoute, Azure Private Peering/BGP, Azure VNET and VNET Gateway, a test VM.
        ii. IoT SaaS components: Azure IoT Hub, IoT Edge with private connectivity, Private Endpoint for IoT Edge.

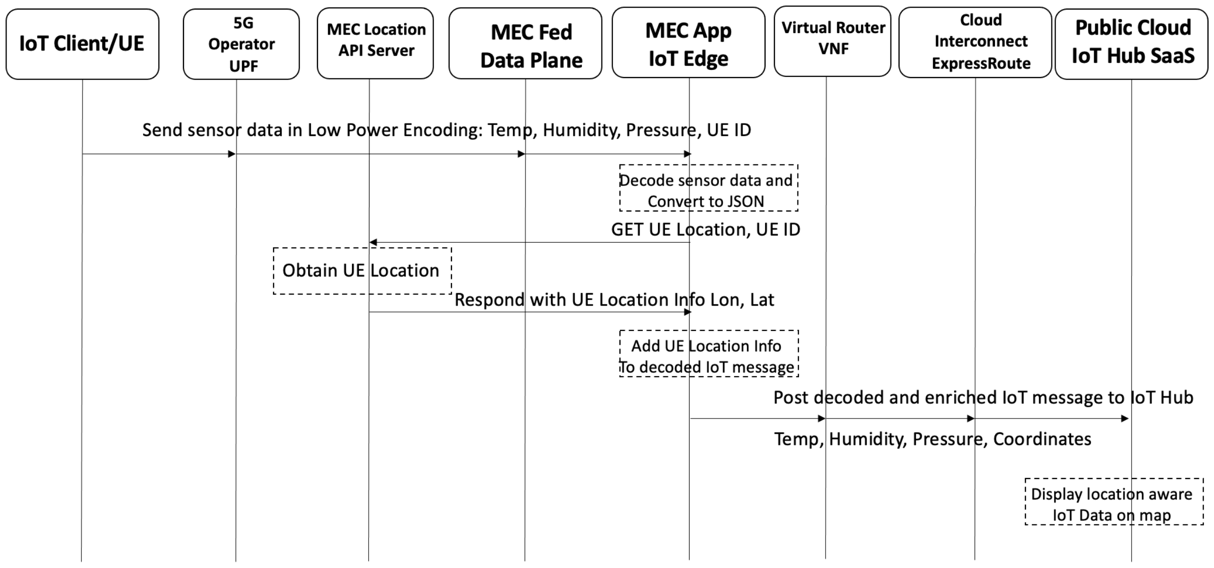
The end-to-end application operation stage:

1. The customer's UE in the Silicon Valley, CA area, connects to the 5G network, registers with the 5G control plane, receives an IPv4 address and establishes a Packet Data Network (PDN) session with the 5G UPF. 5G UPF forwards data to the MEC Federation data plane connection towards the MEC provider's edge cloud.

2. The customer's IoT client running on the 5G UE collects Temperature, Humidity and Pressure measurements, encodes them into the low power IoT message format and sends encoded messages to the IoT Edge Gateway running on the MEC provider's edge cloud in the Dallas, TX data center.
3. The IoT Edge Gateway receives the encoded sensor data, decodes the data to convert the format from low power encoding to the JSON representation.
4. The IoT Edge Gateway in Dallas, TX sends an API call to the Location API server running in the 5G operator data center in Dallas, TX and obtains Latitude, Longitude and Altitude location data for the UE from the 5G operator's MEC Location API server.
5. The IoT Edge Gateway adds location data to the IoT sensor data and publishes the combined message to the IoT Hub running in the Azure Cloud. Note that this data is forwarded from the edge cloud server in the Dallas, TX data center to the virtual router VNF (Cisco CSR1000v) and then routed over the private peering on the Azure ExpressRoute virtual circuit to the customer's VNET and the IoT Hub SaaS. The combined data is stored and posted for processing/viewing.

The data path and the application interactions are shown below:


PCEI Enabler (MEC Federation Broker/Orchestrator)
Operator A Resources (Silicon Valley, CA)
MEC Federation Interconnection Provider Resources
Operator/Provider B Resources (Dallas, TX)
5G Core Control Plane
Bare Metal/K8s
IP/Network
IoT App
UE
gNB
UPF
MEC013 Location API Server
Encoded IoT data
BGP
Port
MEC Fed Data Plane (Fabric)
L2
Port
IP/Network
Public Cloud Edge (PCE)
MEC/ IoT Edge (Azure)
Bare Metal/K8s
Decoded IoT and Location data
vRtr
Fabric
Public Cloud Core (PCC)
IaaS/SaaS IoT Hub (Azure)
Location request/response
5G Network Slice and Location Service
MEC Federation Data Plane
Edge Cloud, MEC and Public Cloud
Equinix
Aarna
ETSI MEC
free5GC
IoT App (Azure IoT Edge and Azure Cloud)


IoT Client/UE
5G Operator UPF
MEC Location API Server
MEC Fed Data Plane
MEC App IoT Edge
Virtual Router VNF
Cloud Interconnect ExpressRoute
Public Cloud IoT Hub SaaS
Send sensor data in Low Power Encoding: Temp, Humidity, Pressure, UE ID
Decode sensor data and Convert to JSON
GET UE Location, UE ID
Obtain UE Location
Respond with UE Location Info Lon, Lat
Add UE Location Info To decoded IoT message
Post decoded and enriched IoT message to IoT Hub
Temp, Humidity, Pressure, Coordinates
Display location aware IoT Data on map

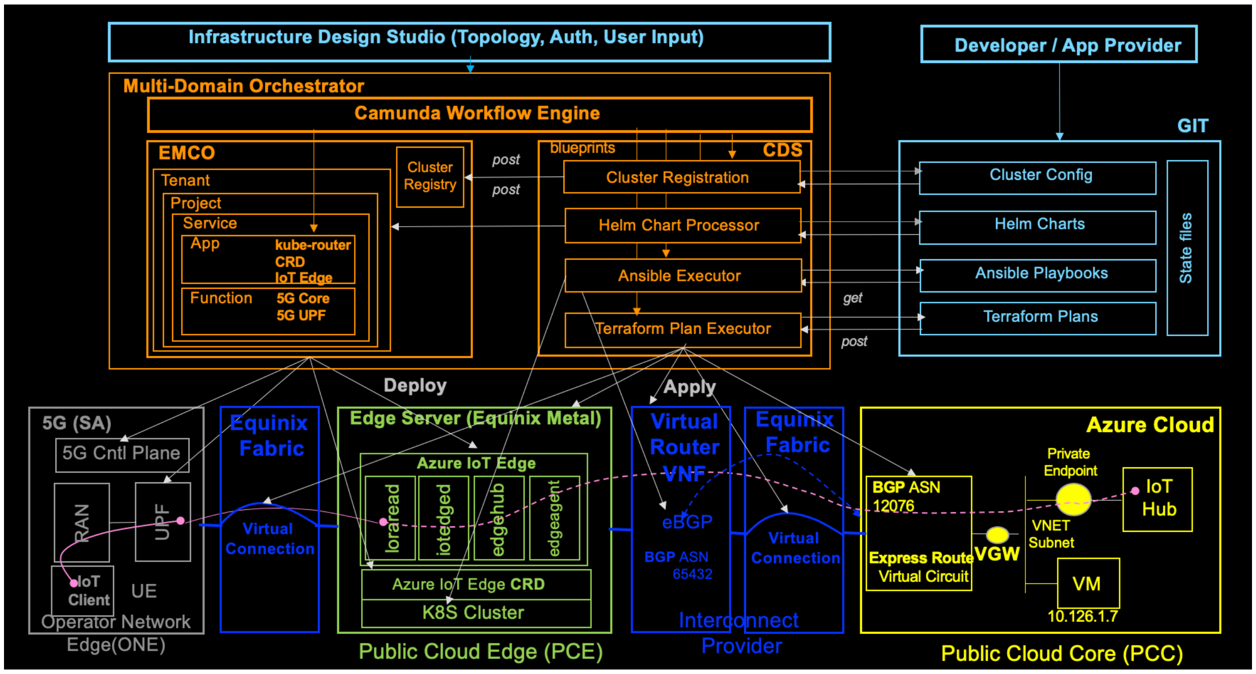
Solution components

MEC Federation Orchestrator

The orchestrator is based on Akraino Public Cloud Edge Interface (PCEI) blueprint. The PCEI blueprint provides the multi-domain orchestrator to enable infrastructure orchestration and cloud native application deployment across public clouds (core and edge), edge clouds, interconnection providers and network operators. The notable innovations in PCEI are the integration of Terraform as a microservice to enable DevOps driven Infrastructure-as-Code provisioning, integration of Ansible as a microservice to enable automation of configuration of infrastructure resources (e.g., servers) and deployment of Kubernetes and its critical components (e.g., CNIs) on the edge cloud, and introduction of a workflow engine to manage the stages and parameter exchange for infrastructure orchestration and application deployment as part of a composable workflow. PCEI can help simplify the process of multi-domain orchestration by enabling uniform representation of diverse services, features, attributes, and APIs used in individual domains as resources and data in the code that can be written by developers and executed by the orchestrator, effectively making the infrastructure orchestration across multiple domains DevOps-driven.

Orchestrator architecture

The structure of the orchestrator is shown below (in orange):

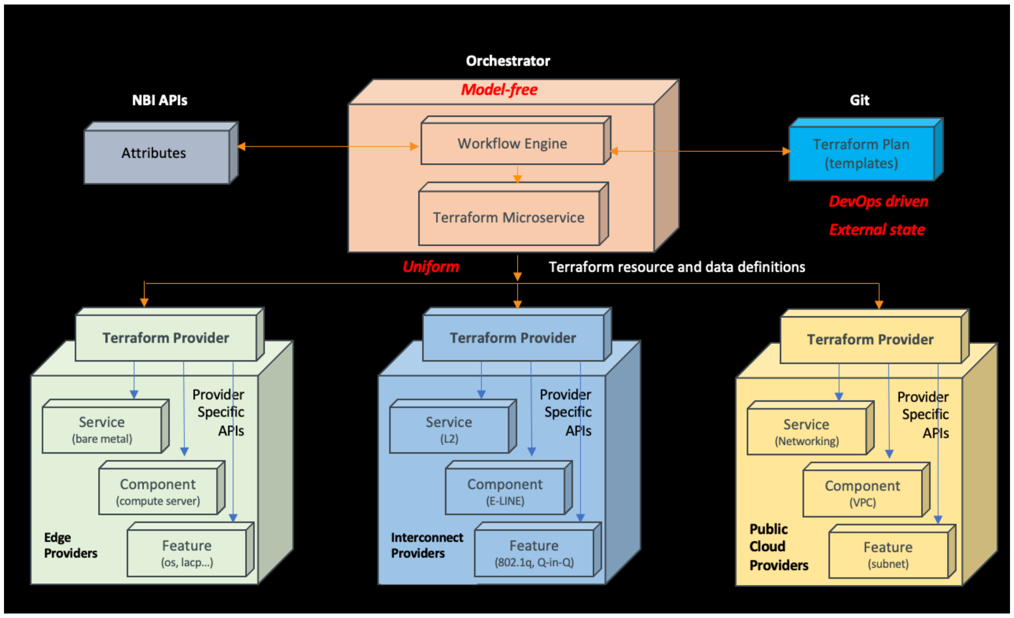


The orchestrator consists of three major parts:

- **The infrastructure deployment part**. It is based on the Controller Design Studio (CDS) open source software and enables the orchestrator to use Infrastructure-as-Code (Terraform) for orchestrating cloud IaaS/SaaS, bare metal, and interconnection/networking, as well as Ansible for installing Kubernetes and configuring network functions.
- **The application deployment part**. It is based on the Edge Multi-Cluster Orchestrator (EMCO) open-source LFN project and enables to deploy distributed cloud native applications and functions on Kubernetes clusters.
- **The workflow engine part**. It is based on the Camunda open-source BPMN project and allows to create sequences of workflows that in turn use the infrastructure and application deployment parts to implement desired deployments.

Terraform module

The orchestrator features a Terraform based Infrastructure-as-Code module that allows execution of Terraform plans in a programmatic manner driven by the Workflow Engine. In our implementation, the orchestrator becomes like a CPU. It retrieves the code and data it needs, then executes it, all without the overhead of maintaining a model." Most important, this enables a DevOps-driven approach to infrastructure orchestration, allowing to build and adjust physical and virtual infrastructure quickly.

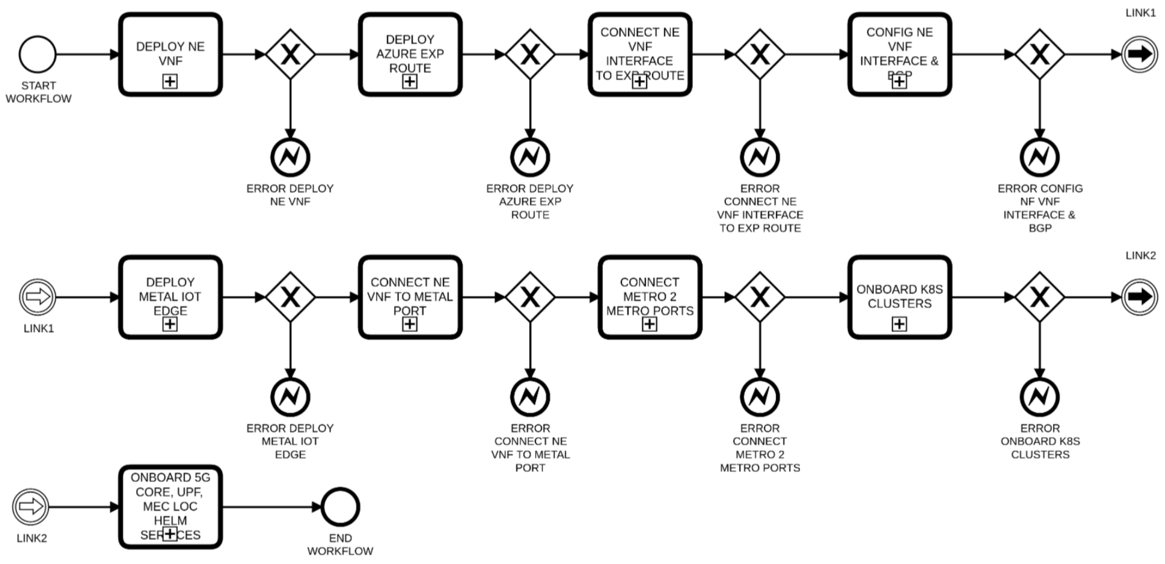


This approach has the following benefits:

- **Uniform** - use of the same infrastructure orchestration methods across public clouds, edge clouds and interconnection domains.
- **Model-free** – the orchestrator does not need to understand the details of the individual infrastructure domains (i.e., implement their models). It only needs to know where to retrieve the Terraform plans for the domain in question and execute the plans using the specified provider.
- **External state** – the state of infrastructure resources created by the orchestrator is stored outside of the orchestrator itself, making it stateless with respect to the infrastructure
- **DevOps driven** – the Terraform plans can be developed and evolved using DvOps tools and processes.

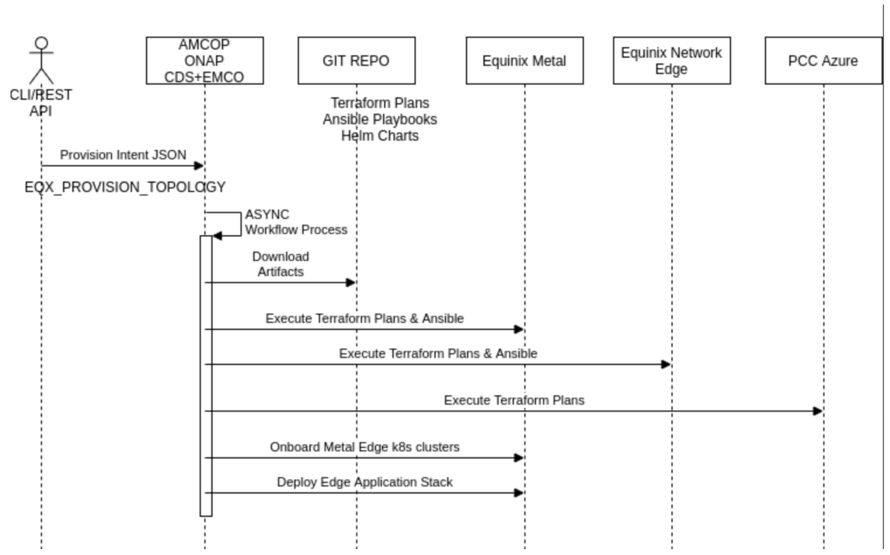
Workflow Engine

The orchestration workflow and action sequencing used in our implementation is shown below.


START WORKFLOW
DEPLOY NE VNF
ERROR DEPLOY NE VNF
DEPLOY AZURE EXP ROUTE
ERROR DEPLOY AZURE EXP ROUTE
CONNECT NE VNF INTERFACE TO EXP ROUTE
ERROR CONNECT NE VNF INTERFACE TO EXP ROUTE
CONFIG NE VNF INTERFACE & BGP
ERROR CONFIG NF VNF INTERFACE & BGP
LINK1
LINK1
DEPLOY METAL IOT EDGE
ERROR DEPLOY METAL IOT EDGE
CONNECT NE VNF TO METAL PORT
ERROR CONNECT NE VNF TO METAL PORT
CONNECT METRO 2 METRO PORTS
ERROR CONNECT METRO 2 METRO PORTS
ONBOARD K8S CLUSTERS
ERROR ONBOARD K8S CLUSTERS
LINK2
LINK2
ONBOARD 5G CORE, UPF, MEC LOC HELM SERVICES
END WORKFLOW

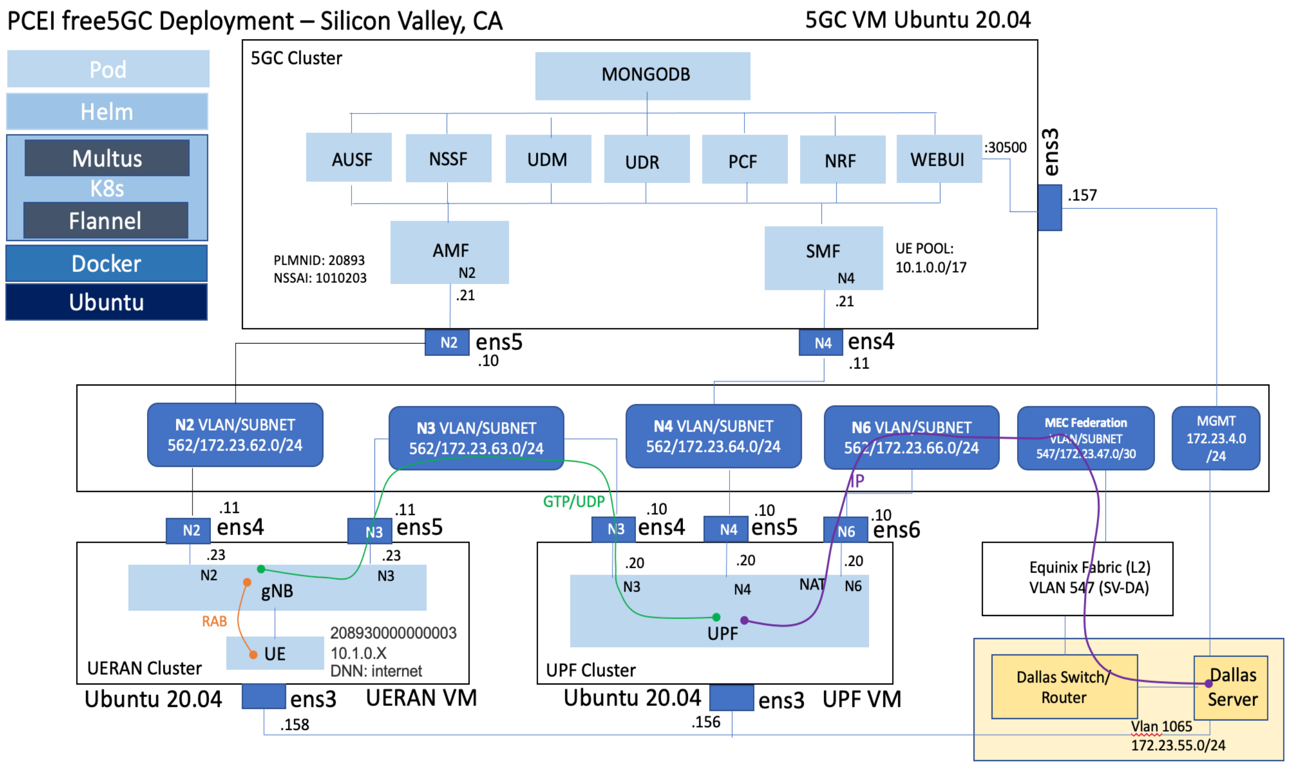
The end to end infra orchestration Camunda workflow steps to demonstrate Public Cloud Azure, Equinix Network Edge and Metal IoT k8s cluster Infrastructure orchestration:

- DEPLOY NE VNF
    - Deploys NE Cisco CSR 1000v VNF (We will reuse pre-deployed VNF as it takes more time to come up)
- DEPLOY AZURE EXP ROUTE
    - Deploys Azure Express Route Circuit
- CONNECT NE VNF INTERFACE TO EXP ROUTE
    - Connect Network Edge Cisco CSR Interface 5 connection with the Azure Express Route Circuit
- CONFIG NE VNF INTERFACE & BGP
    - Configure Cisco CSR 1000v interfaces and BGP configurations
- DEPLOY METAL & K8S
    - Deploys Equinix Metal for the IoT Edge application deployment
- CONNECT NE VNF TO METAL PORT
    - Creates a connection between Equinix Metal and to Cisco CSR Interface 7 connection
- CONNECT METRO 2 METRO PORTS
    - Connects Equinix metro location ports
- ONBOARD K8S CLUSTERS
    - Onboard k8s clusters across multiple edge locations
    - Create the below service models from the Helm3 charts
        - PCEI-MEC-5G-CORE-CP-SVC
        - PCEI-MEC-UPF-LOCAPI-SVC
        - PCEI-MEC-UE-RAN-SIM-SVC
        - PCEI-MEC-METAL-IOT-EDGE-SVC



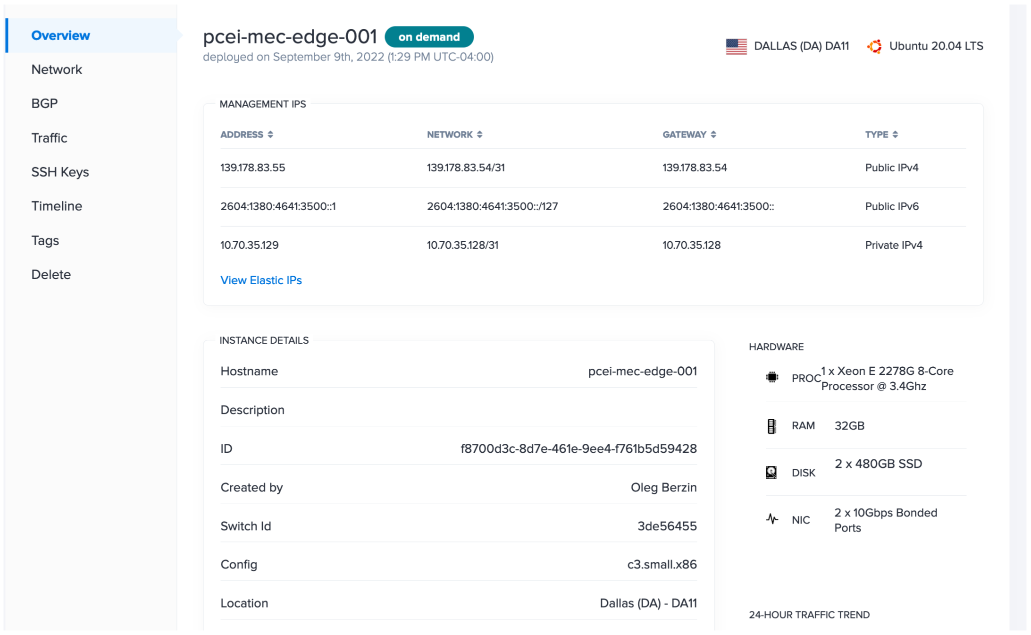
5G Control Plane, User Plane and UE/gNB

We used Free5GC to implement and deploy a network slice that consists of the Control Plane Functions and a distributed User Plane Function. Note that we used a simulated UE/gNB package from Free5GC to run UE and the IoT client within the UE container. The following diagram shows details of our 5G deployment and configuration. Note that all 5G functions were deployed using the orchestrator.



Bare Metal Cloud

We used Equinix Metal production bare metal orchestration platform to activate bare metal servers in Silicon Valley, CA and Dallas, TX. The servers were orchestrated using workflow driven Terraform component of the PCEI orchestrator. After the deployment the bare metal server with the installed Ubuntu OS can be viewed from the Equinix Metal portal and connected to via SSH.



pcei-mec-edge-001 on demand
deployed on September 9th, 2022 (1:29 PM UTC-04:00)

DALLAS (DA) DA11 Ubuntu 20.04 LTS

MANAGEMENT IPS

ADDRESS	NETWORK	GATEWAY	TYPE
139.178.83.55	139.178.83.54/31	139.178.83.54	Public IPv4
2604:1380:4641:3500::1	2604:1380:4641:3500::/127	2604:1380:4641:3500::	Public IPv6
10.70.35.129	10.70.35.128/31	10.70.35.128	Private IPv4

View Elastic IPs

INSTANCE DETAILS

Hostname	pcei-mec-edge-001
Description	
ID	f8700d3c-8d7e-461e-9ee4-f761b5d59428
Created by	Oleg Berzin
Switch Id	3de56455
Config	c3.small.x86
Location	Dallas (DA) - DA11

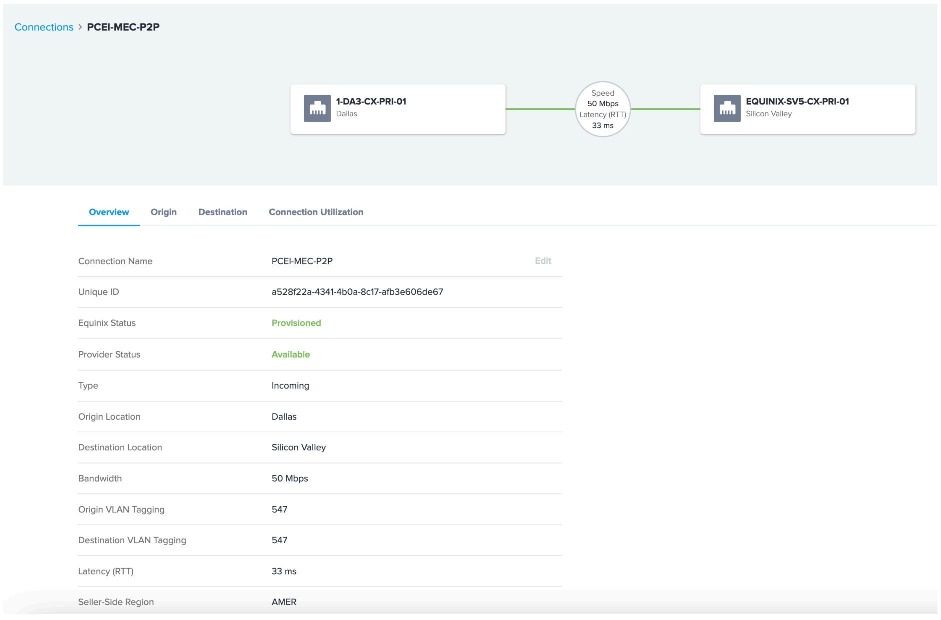
HARDWARE

PROC	1 x Xeon E 2278G 8-Core Processor @ 3.4Ghz
RAM	32GB
DISK	2 x 480GB SSD
NIC	2 x 10Gbps Bonded Ports

24-HOUR TRAFFIC TREND

MEC Federation Data Plane

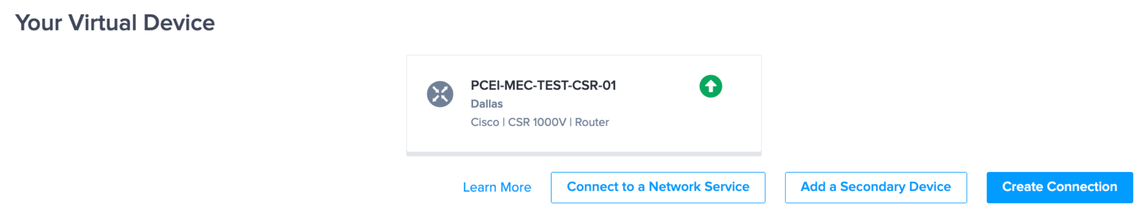
We use a production Equinix Fabric private interconnection service to implement the MEC Federation Data Plane connectivity between the 5G operator domain in Silicon Valley, CA and the MEC provider domain in Dallas, TX. The connectivity was orchestrated using Terraform component of the PCEI orchestrator. After the deployment the MEC Federation Data Plane connection can be viewed from the Equinix Fabric portal.



Connection Name	PCEI-MEC-P2P
Unique ID	a528f22a-4341-4b0a-8c17-afb3e606de67
Equinix Status	Provisioned
Provider Status	Available
Type	Incoming
Origin Location	Dallas
Destination Location	Silicon Valley
Bandwidth	50 Mbps
Origin VLAN Tagging	547
Destination VLAN Tagging	547
Latency (RTT)	33 ms
Seller-Side Region	AMER

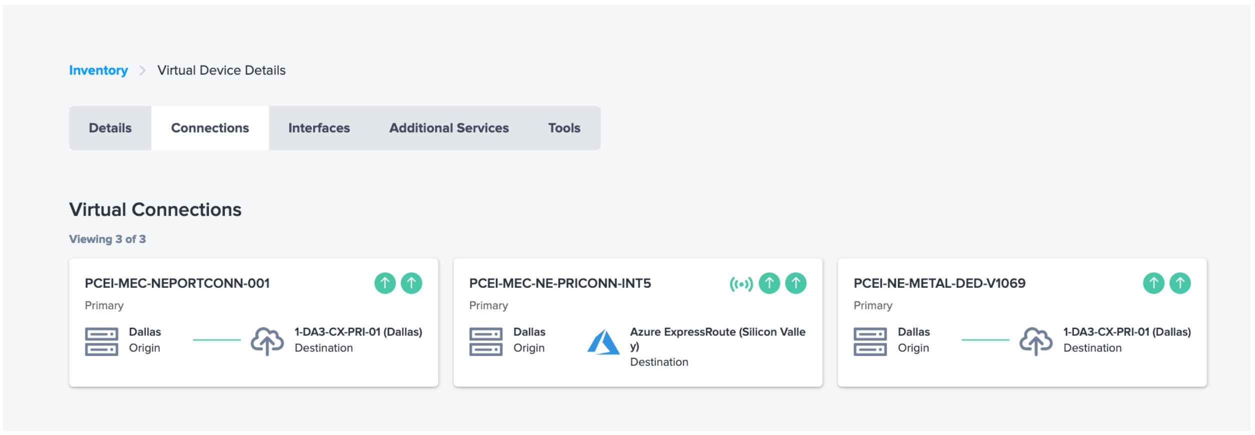
Virtual Network Function and Public Cloud access

We use a production Equinix Network Edge VNF service to implement the virtual router (Cisco CSR1000v) to enable multi-cloud connectivity between the MEC server in Dallas, TX to public clouds. The VNF was orchestrated using Terraform component of the PCEI orchestrator. After the deployment the VNF can be viewed from the Equinix Fabric portal. Note that we have also used the Ansible part of the PCEI orchestrator to configure the VNF with the appropriate interface, IP addressing and BGP parameters to correctly connect to the Azure Cloud with private peering.



Your Virtual Device

PCEI-MEC-TEST-CSR-01
Dallas
Cisco | CSR 1000V | Router



Virtual Connections

Viewing 3 of 3

PCEI-MEC-NEPORTCONN-001
Primary
Dallas Origin — 1-DA3-CX-PRI-01 (Dallas) Destination

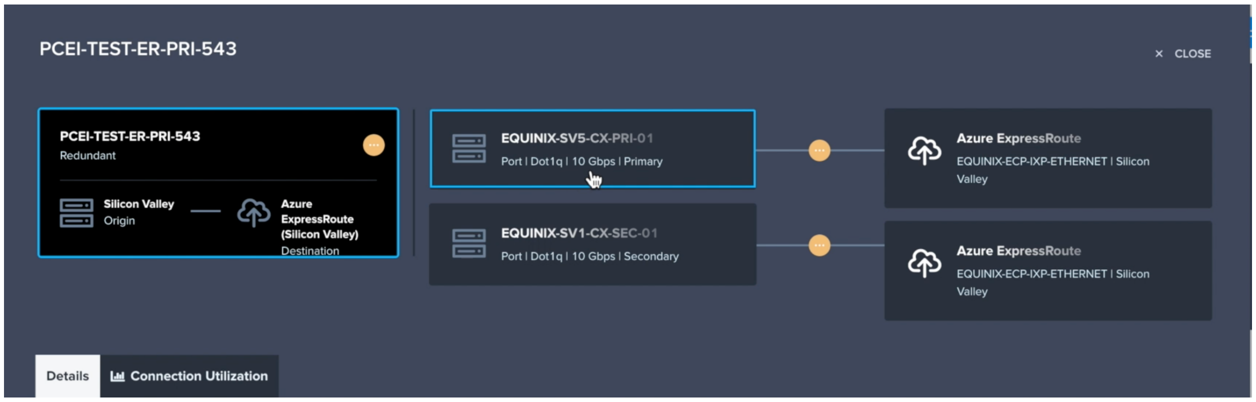
PCEI-MEC-NE-PRICONN-INT5
Primary
Dallas Origin — Azure ExpressRoute (Silicon Valley) Destination

PCEI-NE-METAL-DED-V1069
Primary
Dallas Origin — 1-DA3-CX-PRI-01 (Dallas) Destination

Below is the virtual router configuration to connect to Azure Cloud and to MEC server implemented via Ansible orchestration:

```
interface GigabitEthernet5
 description Azure ExpressRoute
 mtu 8950
 vrf forwarding cloud
 ip address 172.23.43.1 255.255.255.252
 negotiation auto
!
interface GigabitEthernet7
 description MEC Server VLAN
 mtu 8950
 vrf forwarding cloud
 ip address 172.23.67.1 255.255.255.0
 negotiation auto
 !
 router bgp 65432
 bgp router-id 1.2.3.4
 !
 address-family ipv4 vrf cloud
  redistribute connected
  neighbor 172.23.43.2 remote-as 12076
  neighbor 172.23.43.2 password 7 120A0014000E18
  neighbor 172.23.43.2 activate
  neighbor 172.23.43.2 remove-private-as
  maximum-paths eibgp 8
 exit-address-family
```

For the public cloud access, we use Azure ExpressRoute service and create a connection using Equinix Fabric between the virtual router and Azure. This connection was implemented using the Terraform component of the PCEI orchestrator. After the deployment the ExpressRoute connection can be viewed from the Equinix Fabric portal.


PCEI-TEST-ER-PRI-543

× CLOSE

PCEI-TEST-ER-PRI-543
Redundant

Silicon Valley
Origin

Azure ExpressRoute (Silicon Valley)
Destination

EQUINIX-SV5-CX-PRI-01
Port | Dot1q | 10 Gbps | Primary

EQUINIX-SV1-CX-SEC-01
Port | Dot1q | 10 Gbps | Secondary

Azure ExpressRoute
EQUINIX-ECP-IXP-ETHERNET | Silicon Valley

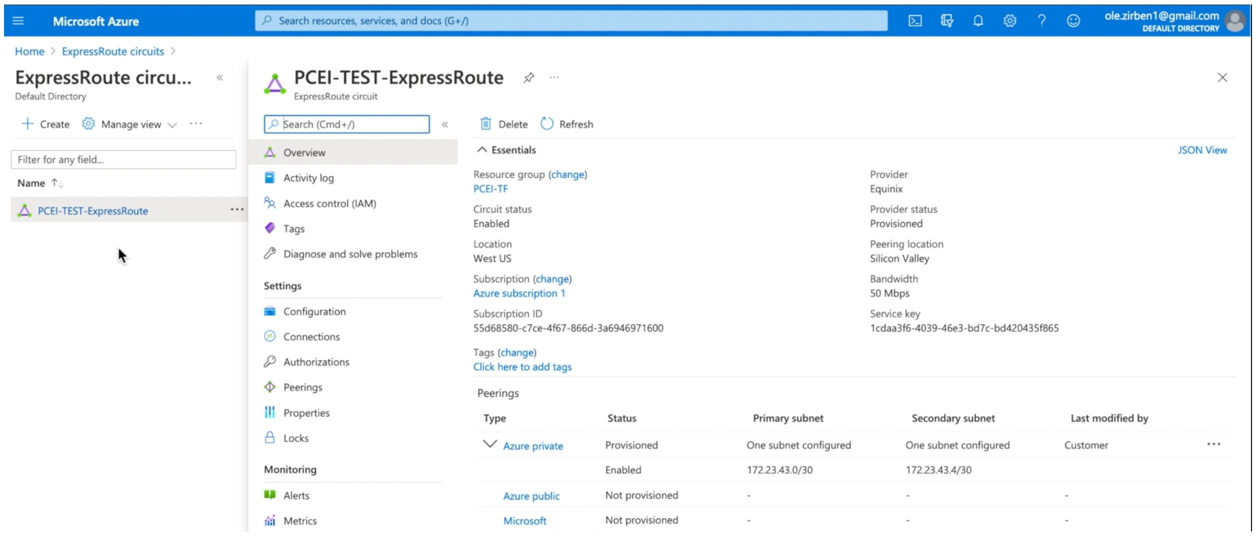
Azure ExpressRoute
EQUINIX-ECP-IXP-ETHERNET | Silicon Valley

Details | Connection Utilization

Public Cloud

Private Interconnect

We use Microsoft Azure cloud to implement IaaS and IoT SaaS infrastructure. The access to Azure Cloud from the MEC provider is implemented using Azure ExpressRoute with Private BGP peering over Equinix Fabric. The ExpressRoute connection and the Private BGP peering were created using Terraform driven from the orchestrator and can be viewed in Azure portal.


Microsoft Azure | Search resources, services, and docs (G+/) | ole.zirben1@gmail.com DEFAULT DIRECTORY

Home > ExpressRoute circuits >

ExpressRoute circu...
Default Directory

+ Create | Manage view

Filter for any field...

Name ↑↓

PCEI-TEST-ExpressRoute

PCEI-TEST-ExpressRoute
ExpressRoute circuit

Search (Cmd+/)

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems

Settings
Configuration
Connections
Authorizations
Peerings
Properties
Locks

Monitoring
Alerts
Metrics

Delete | Refresh

Essentials | JSON View

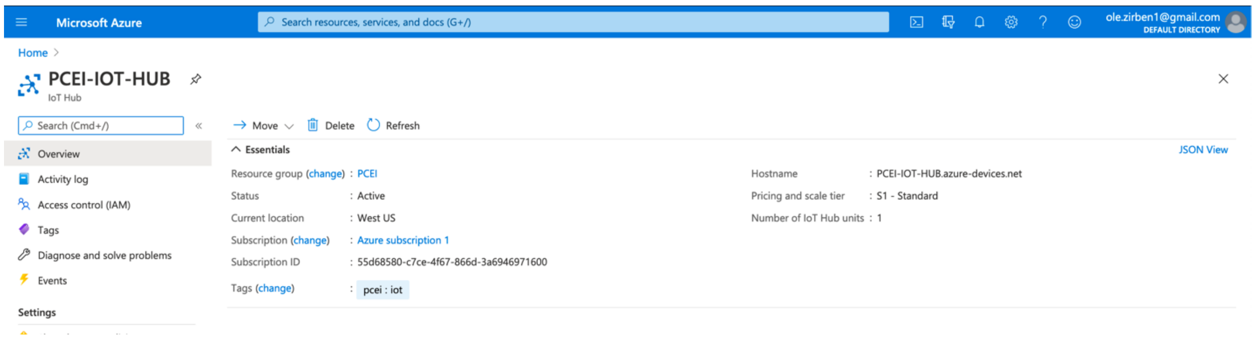
Resource group (change): PCEI-TF	Provider: Equinix
Circuit status: Enabled	Provider status: Provisioned
Location: West US	Peering location: Silicon Valley
Subscription (change): Azure subscription 1	Bandwidth: 50 Mbps
Subscription ID: 55d68580-c7ce-4f67-866d-3a6946971600	Service key: 1cdaa3f6-4039-46e3-bd7c-bd420435f865
Tags (change): Click here to add tags	

Peerings

Type	Status	Primary subnet	Secondary subnet	Last modified by
Azure private	Provisioned	One subnet configured	One subnet configured	Customer
	Enabled	172.23.43.0/30	172.23.43.4/30	
Azure public	Not provisioned	-	-	-
Microsoft	Not provisioned	-	-	-

IoT SaaS

For the IoT SaaS component in the Azure Cloud we used Azure IoT Hub.


Microsoft Azure | Search resources, services, and docs (G+/) | ole.zirben1@gmail.com DEFAULT DIRECTORY

Home >

PCEI-IOT-HUB
IoT Hub

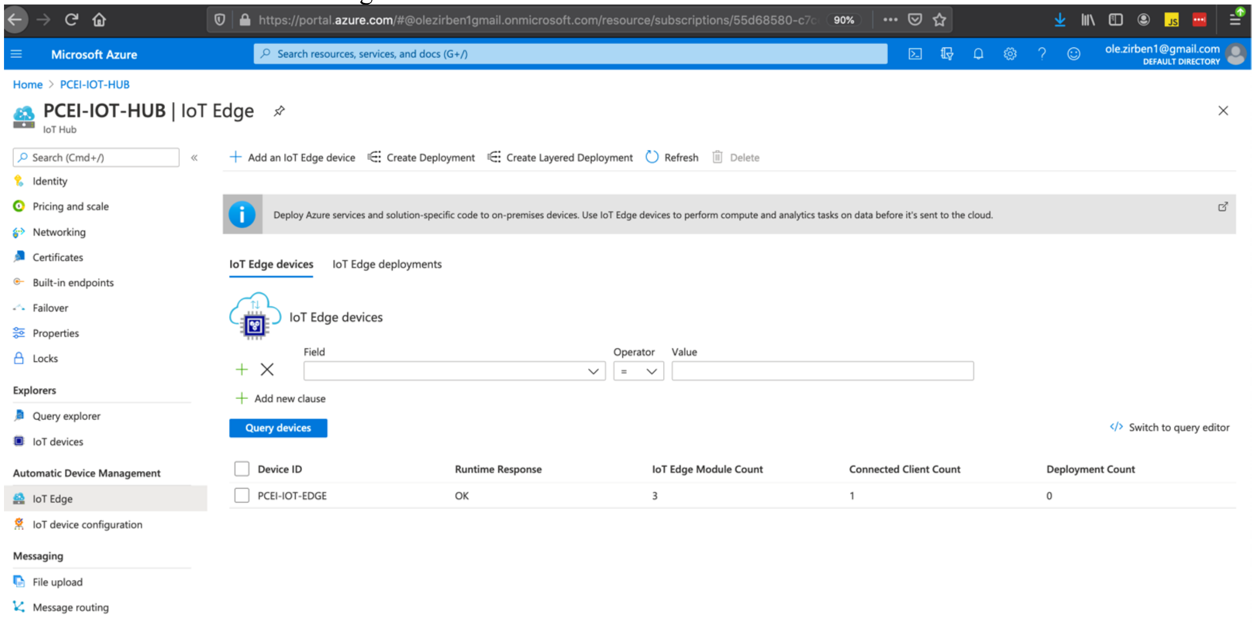
Search (Cmd+/) | Move | Delete | Refresh

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Events
Settings

Essentials | JSON View

Resource group (change)	: PCEI	Hostname	: PCEI-IOT-HUB.azure-devices.net
Status	: Active	Pricing and scale tier	: S1 - Standard
Current location	: West US	Number of IoT Hub units	: 1
Subscription (change)	: Azure subscription 1		
Subscription ID	: 55d68580-c7ce-4f67-866d-3a6946971600		
Tags (change)	: pcei : iot		

We also created an Azure IoT Edge device shadow within the IoT Hub.


https://portal.azure.com/#@olezirben1gmail.onmicrosoft.com/resource/subscriptions/55d68580-c7c | 90%

Microsoft Azure | Search resources, services, and docs (G+/) | ole.zirben1@gmail.com DEFAULT DIRECTORY

Home > PCEI-IOT-HUB

PCEI-IOT-HUB | IoT Edge
IoT Hub

Search (Cmd+/) | Add an IoT Edge device | Create Deployment | Create Layered Deployment | Refresh | Delete

Identity
Pricing and scale
Networking
Certificates
Built-in endpoints
Failover
Properties
Locks
Explorers
Query explorer
IoT devices
Automatic Device Management
IoT Edge
IoT device configuration
Messaging
File upload
Message routing

Deploy Azure services and solution-specific code to on-premises devices. Use IoT Edge devices to perform compute and analytics tasks on data before it's sent to the cloud.

IoT Edge devices | IoT Edge deployments

IoT Edge devices

Field | Operator = | Value

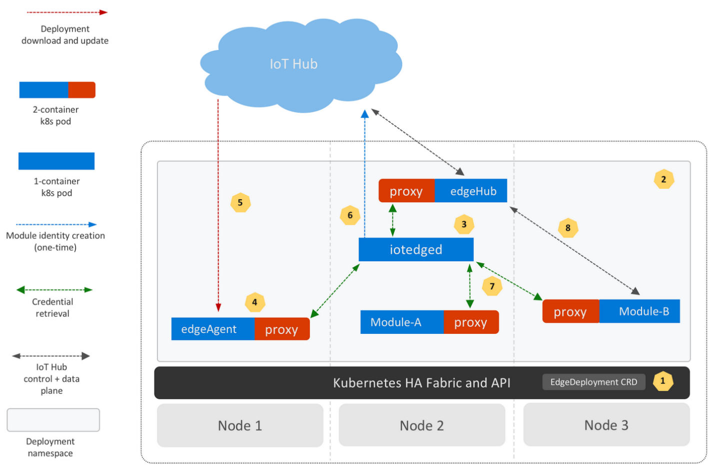
Add new clause

Query devices | Switch to query editor

Device ID	Runtime Response	IoT Edge Module Count	Connected Client Count	Deployment Count
PCEI-IOT-EDGE	OK	3	1	0

Azure IoT Edge Gateway

For the MEC application we used Azure IoT Edge software and deployed it on the Kubernetes cluster running on the edge cloud server in Dallas, TX in the MEC provider domain. The architecture of the cloud native Azure IoT Edge Gateway is shown below and can be found at this link https://microsoft.github.io/iotedge-k8s-doc/architecture.html.



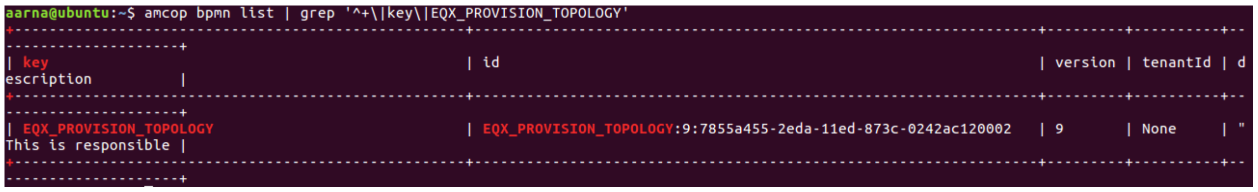
Solution implementation

MEC Federation Orchestrator

Infrastructure orchestration

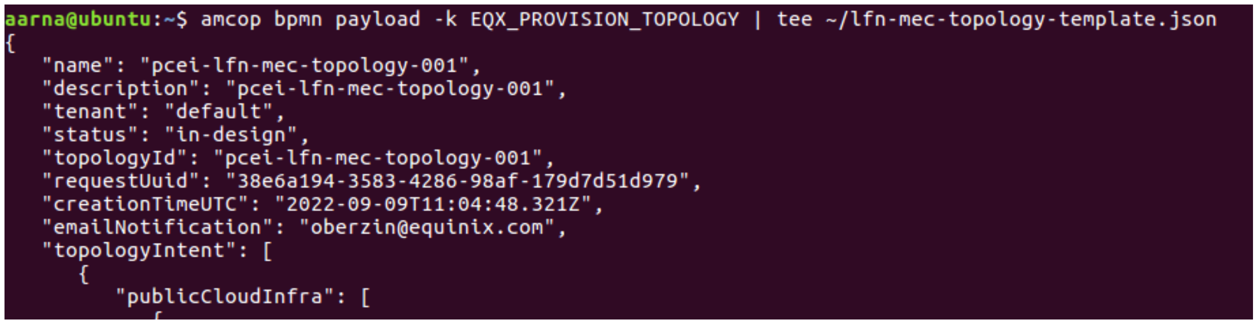
In our implementation we used the AMCOP package provided by Aarna Networks to run the PCEI orchestrator.

```
# Get the AMCOP workflow details
amcop bpmn list | grep '^+\|key\|EQX_PROVISION_TOPOLOGY'
```



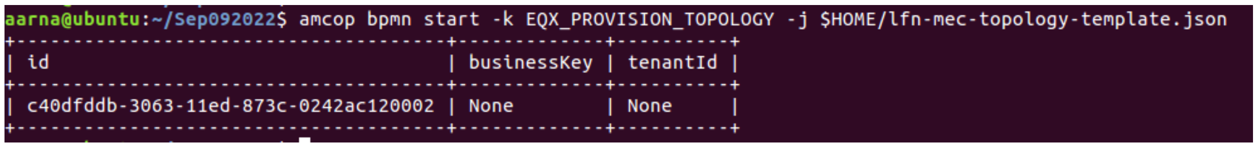
```
aarna@ubuntu:~$ amcop bpmn list | grep '^+\|key\|EQX_PROVISION_TOPOLOGY'
+-----------------------------------------------------+------------------------------------------------------------------+---------+----------+--
--------------------+
| key                                                 | id                                                               | version | tenantId | d
escription          |
+-----------------------------------------------------+------------------------------------------------------------------+---------+----------+--
--------------------+
| EQX_PROVISION_TOPOLOGY                              | EQX_PROVISION_TOPOLOGY:9:7855a455-2eda-11ed-873c-0242ac120002    | 9       | None     | "
This is responsible |
+-----------------------------------------------------+------------------------------------------------------------------+---------+----------+--
--------------------+
```

```
# Review the AMCOP workflow input JSON payload template
amcop bpmn payload -k EQX_PROVISION_TOPOLOGY | tee ~/lfn-mec-topology-template.json

# Take some time to review the JSON payload to understand the topology provisioning
# intent
```



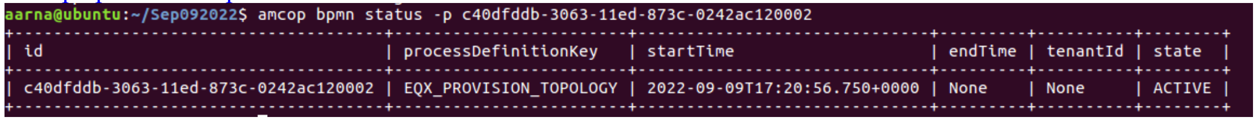
```
aarna@ubuntu:~$ amcop bpmn payload -k EQX_PROVISION_TOPOLOGY | tee ~/lfn-mec-topology-template.json
{
   "name": "pcei-lfn-mec-topology-001",
   "description": "pcei-lfn-mec-topology-001",
   "tenant": "default",
   "status": "in-design",
   "topologyId": "pcei-lfn-mec-topology-001",
   "requestUuid": "38e6a194-3583-4286-98af-179d7d51d979",
   "creationTimeUTC": "2022-09-09T11:04:48.321Z",
   "emailNotification": "oberzin@equinix.com",
   "topologyIntent": [
      {
         "publicCloudInfra": [
```

# Execute the AMCOP workflow to deploy **pcei-lfn-mec-topology-001**
# This command will output the Camunda workflow process instance ID
```
amcop bpmn start -k EQX_PROVISION_TOPOLOGY -j $HOME/lfn-mec-topology-template.json
```



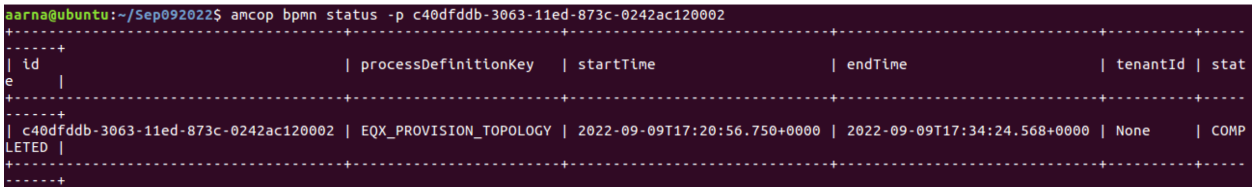
```
aarna@ubuntu:~/Sep092022$ amcop bpmn start -k EQX_PROVISION_TOPOLOGY -j $HOME/lfn-mec-topology-template.json
+--------------------------------------+-------------+----------+
| id                                   | businessKey | tenantId |
+--------------------------------------+-------------+----------+
| c40dfddb-3063-11ed-873c-0242ac120002 | None        | None     |
+--------------------------------------+-------------+----------+
```

```
# We can use the above process instance to check the status
# Running state = ACTIVE
amcop bpmn status -p c40dfddb-3063-11ed-873c-0242ac120002
```



```
aarna@ubuntu:~/Sep092022$ amcop bpmn status -p c40dfddb-3063-11ed-873c-0242ac120002
+--------------------------------------+------------------------+-------------------------------+---------+----------+--------+
| id                                   | processDefinitionKey   | startTime                     | endTime | tenantId | state  |
+--------------------------------------+------------------------+-------------------------------+---------+----------+--------+
| c40dfddb-3063-11ed-873c-0242ac120002 | EQX_PROVISION_TOPOLOGY | 2022-09-09T17:20:56.750+0000 | None    | None     | ACTIVE |
+--------------------------------------+------------------------+-------------------------------+---------+----------+--------+
```

```
# Completion state = COMPLETED
amcop bpmn status -p c40dfddb-3063-11ed-873c-0242ac120002
```

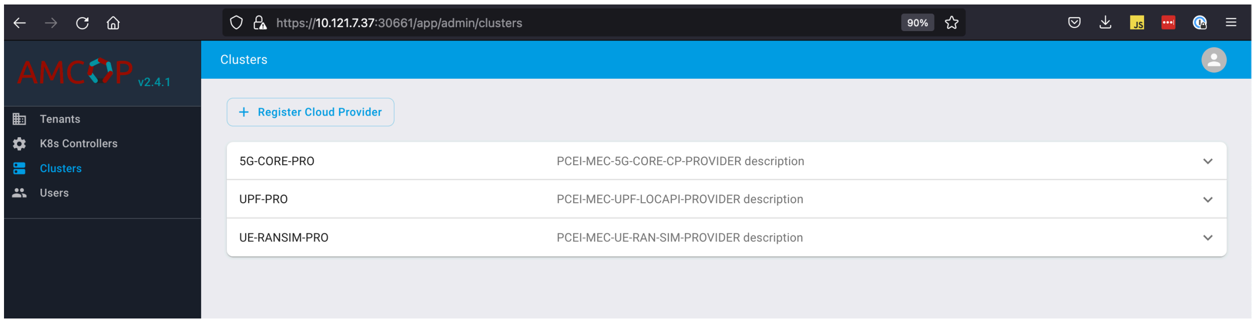


```
aarna@ubuntu:~/Sep092022$ amcop bpmn status -p c40dfddb-3063-11ed-873c-0242ac120002
+--------------------------------------+------------------------+------------------------------+------------------------------+----------+-----
------+
| id                                   | processDefinitionKey   | startTime                    | endTime                      | tenantId | stat
e     |
+--------------------------------------+------------------------+------------------------------+------------------------------+----------+-----
------+
| c40dfddb-3063-11ed-873c-0242ac120002 | EQX_PROVISION_TOPOLOGY | 2022-09-09T17:20:56.750+0000 | 2022-09-09T17:34:24.568+0000 | None     | COMP
LETED |
+--------------------------------------+------------------------+------------------------------+------------------------------+----------+-----
------+
```

Services, 5G functions and IoT application orchestration

For the service, 5G functions and IoT application orchestration we used the GUI part of the orchestrator.

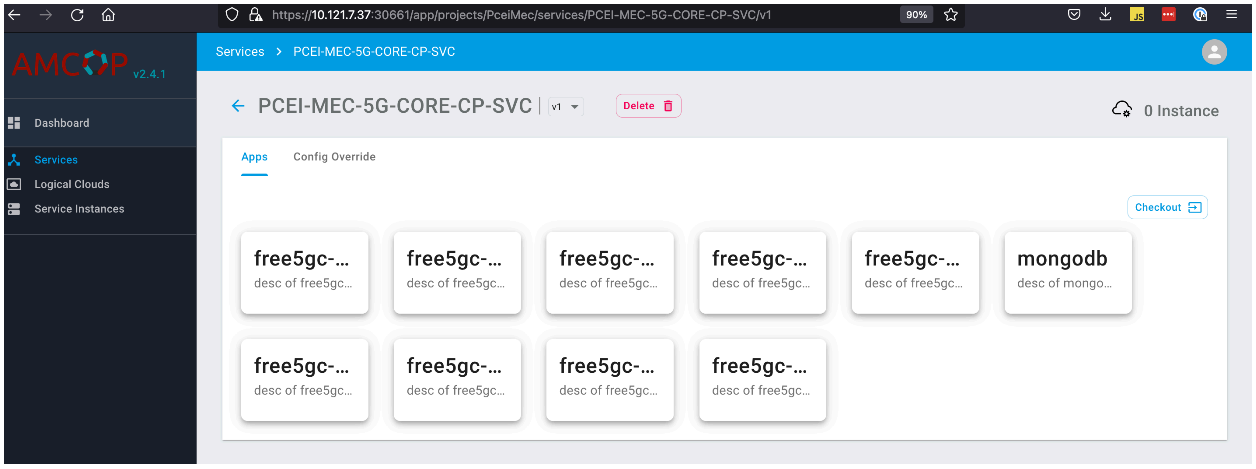
First, we onboarded the Kubernetes clusters for 5G Control Plane, User Plane and IoT Edge:



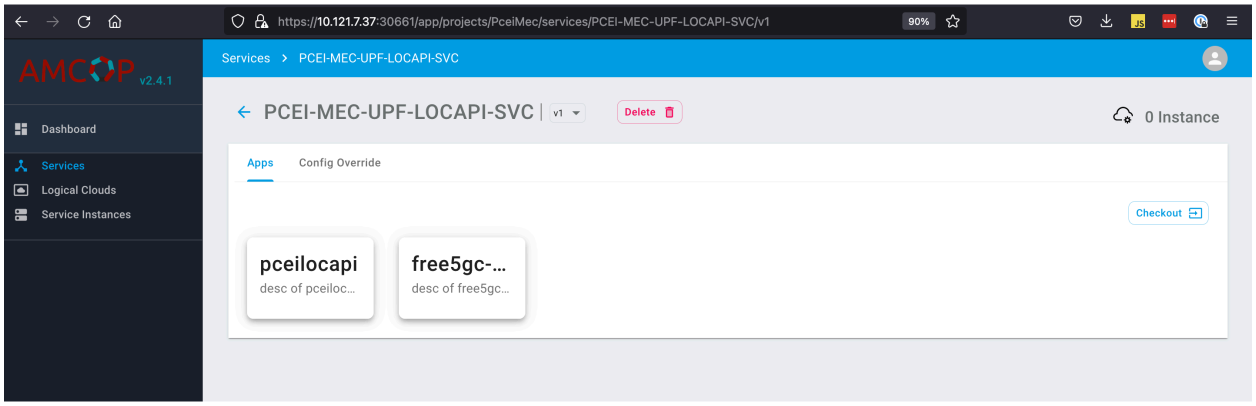
5G-CORE-PRO	PCEI-MEC-5G-CORE-CP-PROVIDER description
UPF-PRO	PCEI-MEC-UPF-LOCAPI-PROVIDER description
UE-RANSIM-PRO	PCEI-MEC-UE-RAN-SIM-PROVIDER description

The we created Service and Service Instances based on the Helm Charts for the Free5GC, Location API, and IoT Edge deployments:

5G Control Plane:



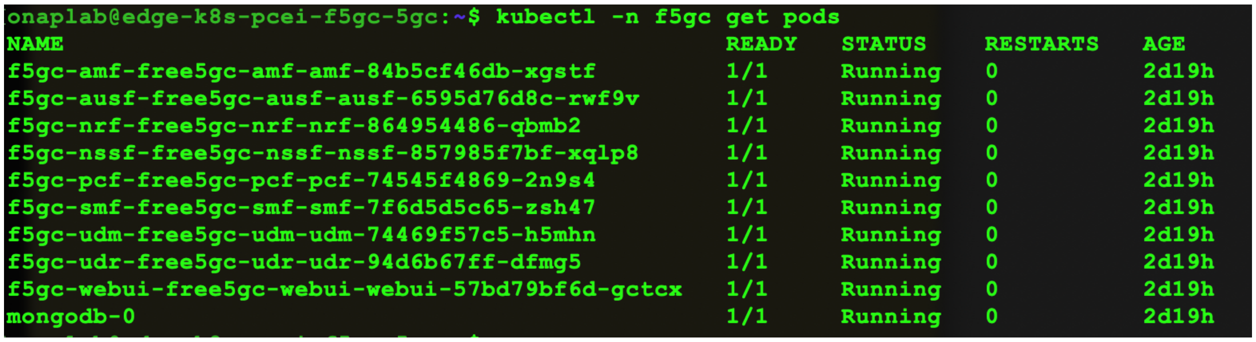
5G User Plane and Location API:



The Service Instances were deployed to the respective Kubernetes clusters.

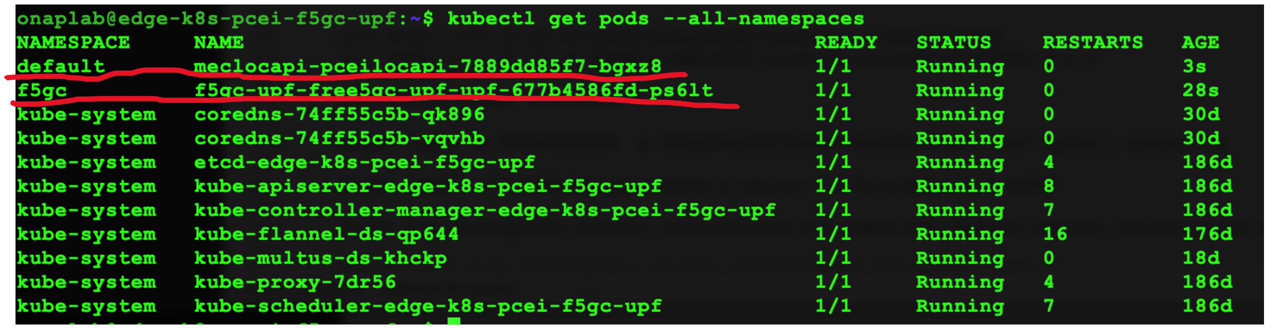
Free5GC

Control Plane cluster:



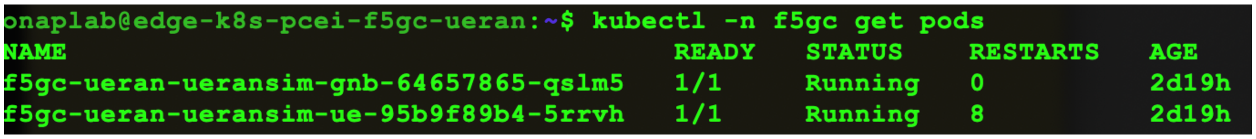
```
onaplab@edge-k8s-pcei-f5gc-5gc:~$ kubectl -n f5gc get pods
NAME                                          READY   STATUS    RESTARTS   AGE
f5gc-amf-free5gc-amf-amf-84b5cf46db-xgstf     1/1     Running   0          2d19h
f5gc-ausf-free5gc-ausf-ausf-6595d76d8c-rwf9v  1/1     Running   0          2d19h
f5gc-nrf-free5gc-nrf-nrf-864954486-qbmb2      1/1     Running   0          2d19h
f5gc-nssf-free5gc-nssf-nssf-857985f7bf-xqlp8  1/1     Running   0          2d19h
f5gc-pcf-free5gc-pcf-pcf-74545f4869-2n9s4     1/1     Running   0          2d19h
f5gc-smf-free5gc-smf-smf-7f6d5d5c65-zsh47     1/1     Running   0          2d19h
f5gc-udm-free5gc-udm-udm-74469f57c5-h5mhn     1/1     Running   0          2d19h
f5gc-udr-free5gc-udr-udr-94d6b67ff-dfmg5      1/1     Running   0          2d19h
f5gc-webui-free5gc-webui-webui-57bd79bf6d-gctcx 1/1   Running   0          2d19h
mongodb-0                                     1/1     Running   0          2d19h
```

User Plane and MEC Location API cluster:



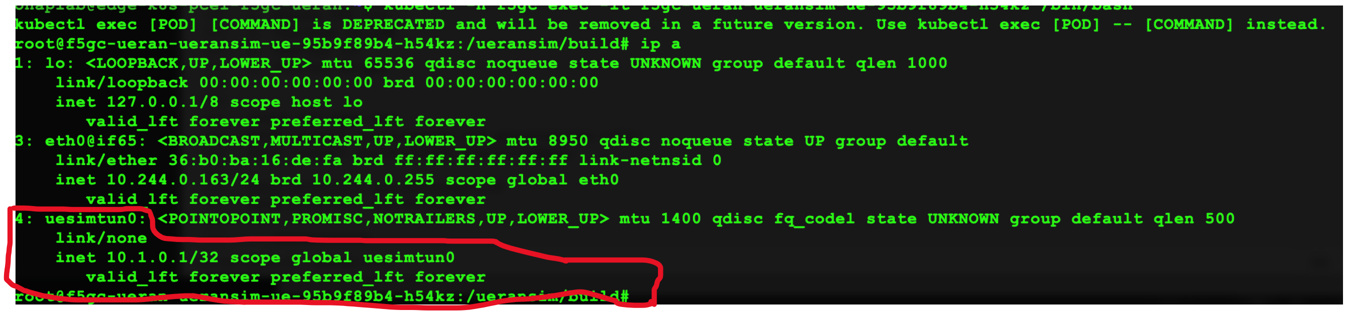
```
onaplab@edge-k8s-pcei-f5gc-upf:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY   STATUS    RESTARTS   AGE
default       meclocapi-pceilocapi-7889dd85f7-bgxz8         1/1     Running   0          3s
f5gc          f5gc-upf-free5gc-upf-upf-677b4586fd-ps6lt     1/1     Running   0          28s
kube-system   coredns-74ff55c5b-qk896                       1/1     Running   0          30d
kube-system   coredns-74ff55c5b-vqvhb                       1/1     Running   0          30d
kube-system   etcd-edge-k8s-pcei-f5gc-upf                   1/1     Running   4          186d
kube-system   kube-apiserver-edge-k8s-pcei-f5gc-upf         1/1     Running   8          186d
kube-system   kube-controller-manager-edge-k8s-pcei-f5gc-upf 1/1    Running   7          186d
kube-system   kube-flannel-ds-qp644                         1/1     Running   16         176d
kube-system   kube-multus-ds-khckp                          1/1     Running   0          18d
kube-system   kube-proxy-7dr56                              1/1     Running   4          186d
kube-system   kube-scheduler-edge-k8s-pcei-f5gc-upf         1/1     Running   7          186d
```

UE/gNB cluster:



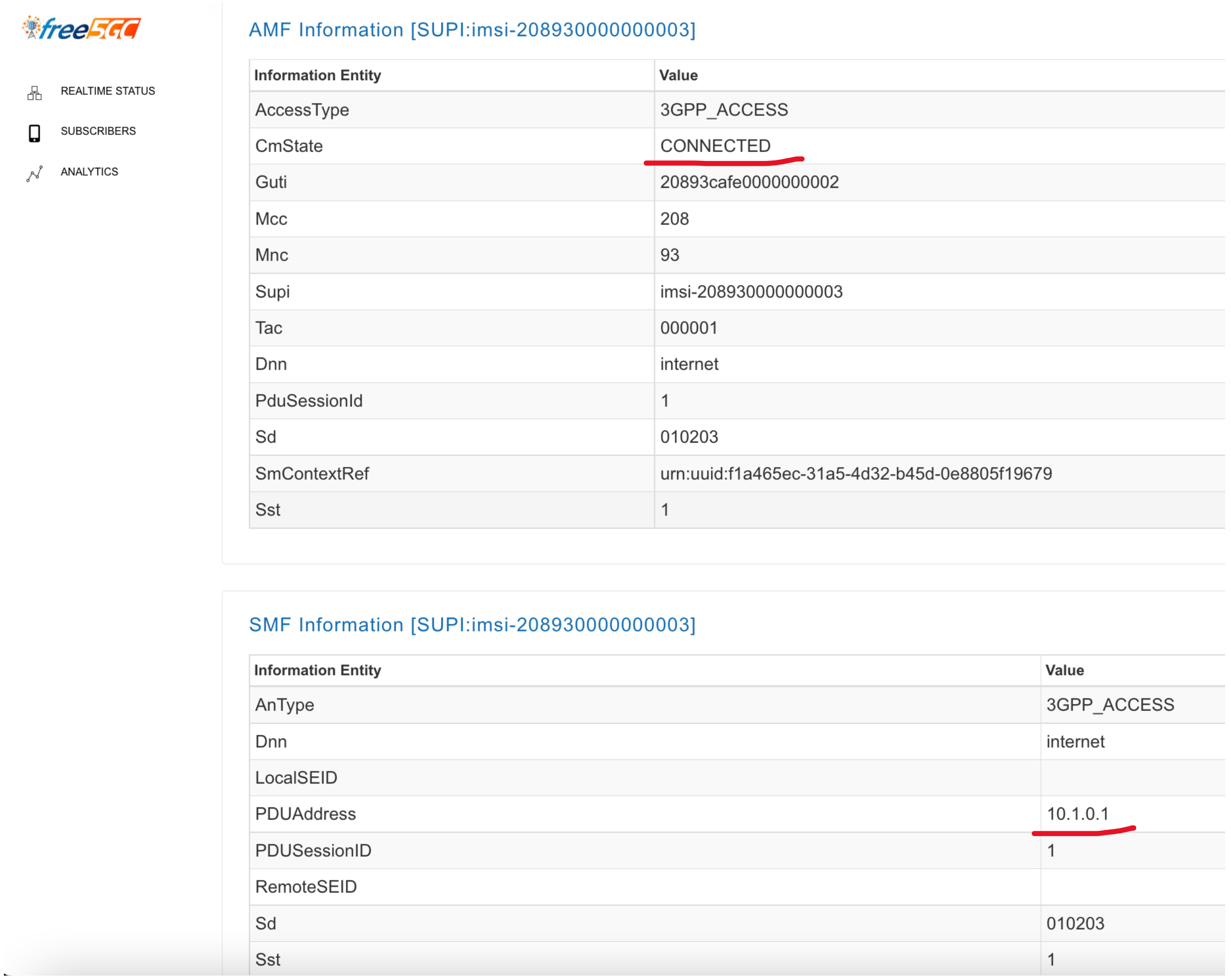
```
onaplab@edge-k8s-pcei-f5gc-ueran:~$ kubectl -n f5gc get pods
NAME                                      READY   STATUS    RESTARTS   AGE
f5gc-ueran-ueransim-gnb-64657865-qslm5    1/1     Running   0          2d19h
f5gc-ueran-ueransim-ue-95b9f89b4-5rrvh    1/1     Running   8          2d19h
```

Inside the UE container we can see an established 5G interface/session with the assigned IPv4:



```
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:/ueransim/build# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if65: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8950 qdisc noqueue state UP group default
    link/ether 36:b0:ba:16:de:fa brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.244.0.163/24 brd 10.244.0.255 scope global eth0
       valid_lft forever preferred_lft forever
4: uesimtun0: <POINTOPOINT,PROMISC,NOTRAILERS,UP,LOWER_UP> mtu 1400 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.1.0.1/32 scope global uesimtun0
       valid_lft forever preferred_lft forever
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:/ueransim/build#
```

The Free5GC WebUI shows the connected UE:



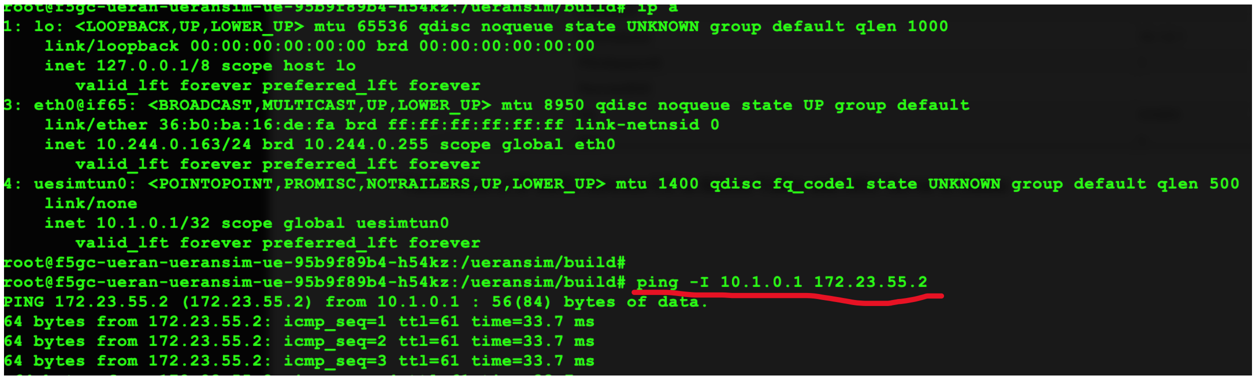
AMF Information [SUPI:imsi-208930000000003]

Information Entity	Value
AccessType	3GPP_ACCESS
CmState	CONNECTED
Guti	20893cafe0000000002
Mcc	208
Mnc	93
Supi	imsi-208930000000003
Tac	000001
Dnn	internet
PduSessionId	1
Sd	010203
SmContextRef	urn:uuid:f1a465ec-31a5-4d32-b45d-0e8805f19679
Sst	1

SMF Information [SUPI:imsi-208930000000003]

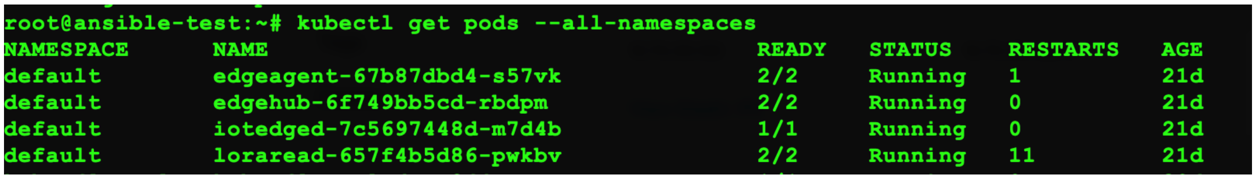
Information Entity	Value
AnType	3GPP_ACCESS
Dnn	internet
LocalSEID	
PDUAddress	10.1.0.1
PDUSessionID	1
RemoteSEID	
Sd	010203
Sst	1

Verification of IP traffic between UE in Silicon Valley and MEC Server in Dallas:



```
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:/ueransim/build# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if65: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8950 qdisc noqueue state UP group default
    link/ether 36:b0:ba:16:de:fa brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.244.0.163/24 brd 10.244.0.255 scope global eth0
       valid_lft forever preferred_lft forever
4: uesimtun0: <POINTOPOINT,PROMISC,NOTRAILERS,UP,LOWER_UP> mtu 1400 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.1.0.1/32 scope global uesimtun0
       valid_lft forever preferred_lft forever
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:/ueransim/build#
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:/ueransim/build# ping -I 10.1.0.1 172.23.55.2
PING 172.23.55.2 (172.23.55.2) from 10.1.0.1 : 56(84) bytes of data.
64 bytes from 172.23.55.2: icmp_seq=1 ttl=61 time=33.7 ms
64 bytes from 172.23.55.2: icmp_seq=2 ttl=61 time=33.7 ms
64 bytes from 172.23.55.2: icmp_seq=3 ttl=61 time=33.7 ms
```

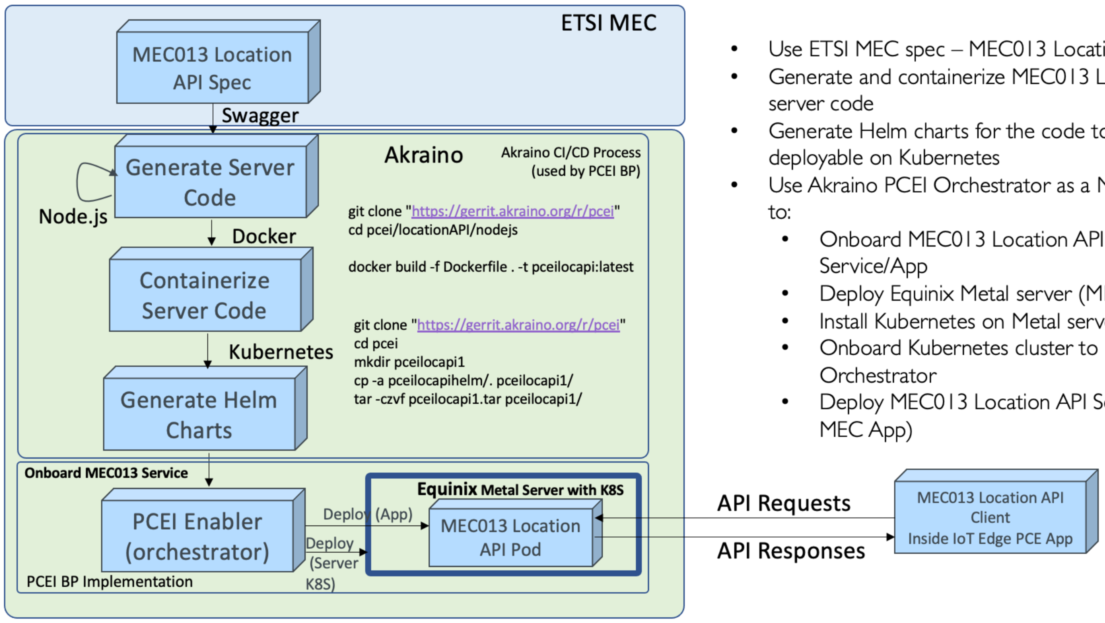
Azure IoT Edge on the MEC Server



```
root@ansible-test:~# kubectl get pods --all-namespaces
NAMESPACE       NAME                              READY   STATUS    RESTARTS   AGE
default         edgeagent-67b87dbd4-s57vk         2/2     Running   1          21d
default         edgehub-6f749bb5cd-rbdpm          2/2     Running   0          21d
default         iotedged-7c5697448d-m7d4b         1/1     Running   0          21d
default         loraread-657f4b5d86-pwkbv         2/2     Running   11         21d
```

MEC Location API cloud native implementation

We have used the ETSI MEC013 Location API specification and implemented a cloud native deployment of a test Location API server using the below method.



```
git clone "https://gerrit.akraino.org/r/pcei"
cd pcei/locationAPI/nodejs

docker build -f Dockerfile . -t pceilocapi:latest

git clone "https://gerrit.akraino.org/r/pcei"
cd pcei
mkdir pceilocapi1
cp -a pceilocapihelm/. pceilocapi1/
tar -czvf pceilocapi1.tar pceilocapi1/
```

- Use ETSI MEC spec – MEC013 Location API
- Generate and containerize MEC013 Location API server code
- Generate Helm charts for the code to make it deployable on Kubernetes
- Use Akraino PCEI Orchestrator as a MEO/MEPM to:
  - Onboard MEC013 Location API server as a Service/App
  - Deploy Equinix Metal server (MEP/MEC Host)
  - Install Kubernetes on Metal server
  - Onboard Kubernetes cluster to PCEI Orchestrator
  - Deploy MEC013 Location API Service (as MEC App)

Enabling Azure IoT Edge Custom Module

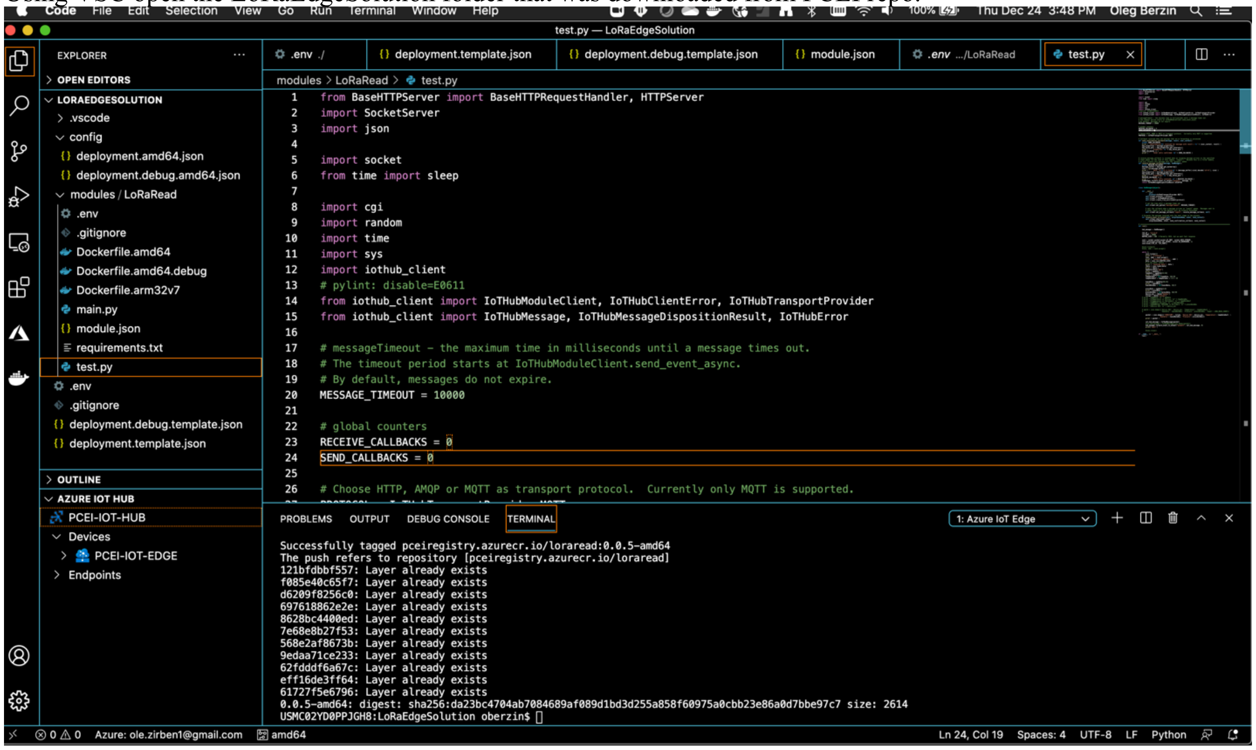
Use this link for information on developing custom software modules for Azure IoT Edge:
https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-develop-for-linux?view=iotedge-2018-06#set-up-vs-code-and-tools
The example below is optional. It shows how to build a custom module for Azure IoT Edge to read and decode Low Power IoT messages from a simulated LPWA IoT device. Follow the above link to:

1. Install Docker.
2. Download and install Visual Studio Code (VSC).
3. Setup VSC with Azure IoT Tools.
4. Setup Azure Container Registry in Azure Cloud.
5. Create Module Project. For this step, please refer to instructions below on downloading the LoRaEdgeSolution from PCEI repo.
6. Build and push solution to Azure Container Registry.

The steps below show how to build custom IoT module for Azure IoT Edge using "LoRaEdgeSolution" code from PCEI repo:
Download PCEI repo to the machine that has VSC and Docker installed (per above instructions):

```
git clone "https://gerrit.akraino.org/r/pcei"

cd pcei
ls -l
total 0
drwxr-xr-x  8 oberzin  staff  256 Dec 24 15:44 LoRaEdgeSolution
drwxr-xr-x  3 oberzin  staff   96 Dec 24 15:44 iotclient
drwxr-xr-x  5 oberzin  staff  160 Dec 24 15:44 locationAPI
```

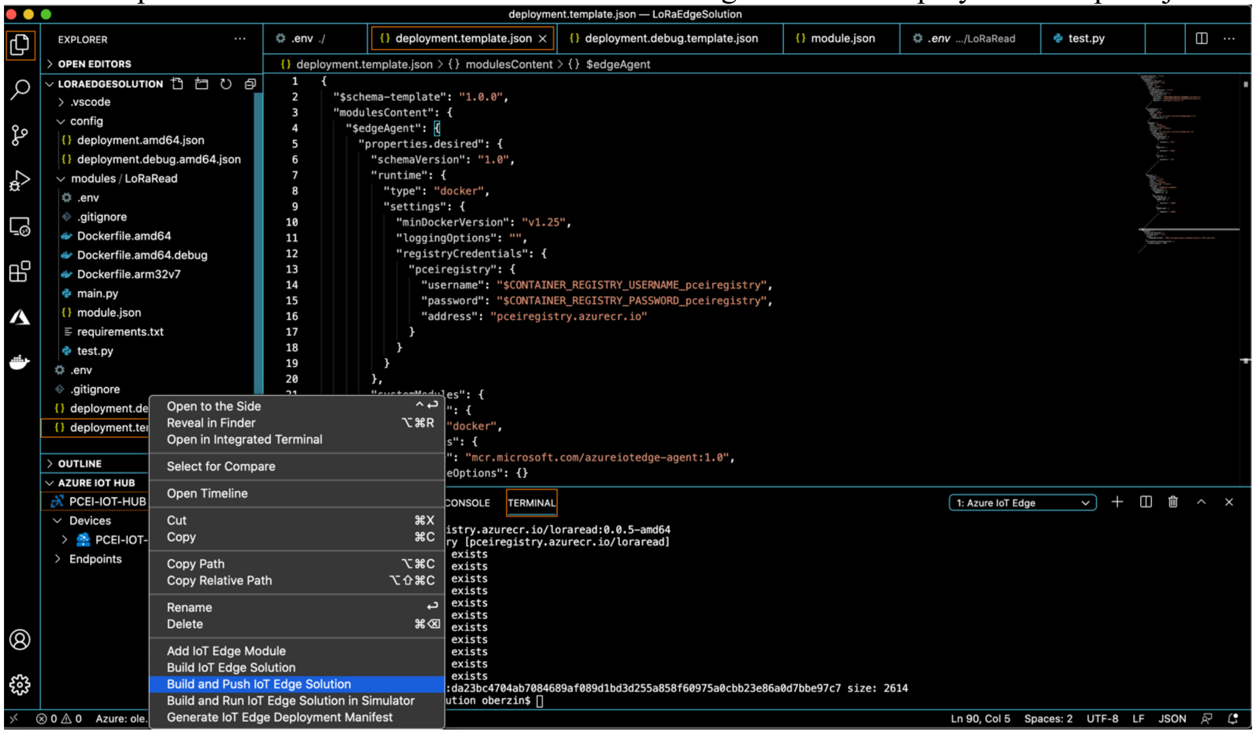
Using VSC open the LoRaEdgeSolution folder that was downloaded from PCEI repo.



```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
import SocketServer
import json

import socket
from time import sleep

import cgi
import random
import time
import sys
import iothub_client
# pylint: disable=E0611
from iothub_client import IoTHubModuleClient, IoTHubClientError, IoTHubTransportProvider
from iothub_client import IoTHubMessage, IoTHubMessageDispositionResult, IoTHubError

# messageTimeout - the maximum time in milliseconds until a message times out.
# The timeout period starts at IoTHubModuleClient.send_event_async.
# By default, messages do not expire.
MESSAGE_TIMEOUT = 10000

# global counters
RECEIVE_CALLBACKS = 0
SEND_CALLBACKS = 0

# Choose HTTP, AMQP or MQTT as transport protocol.  Currently only MQTT is supported.
```

```
Successfully tagged pceiregistry.azurecr.io/loraread:0.0.5-amd64
The push refers to repository [pceiregistry.azurecr.io/loraread]
121bfdbbf557: Layer already exists
f085e40c65f7: Layer already exists
d6209f8256c0: Layer already exists
697618862e2e: Layer already exists
8628bc4400ed: Layer already exists
7e68e8b27f53: Layer already exists
568e2af8673b: Layer already exists
9edaa71ce233: Layer already exists
62fdddf6a67c: Layer already exists
eff16de3ff64: Layer already exists
61727f5e6796: Layer already exists
0.0.5-amd64: digest: sha256:da23bc4704ab7084689af089d1bd3d255a858f60975a0cbb23e86a0d7bbe97c7 size: 2614
USMC02YD0PPJGH8:LoRaEdgeSolution oberzin$
```

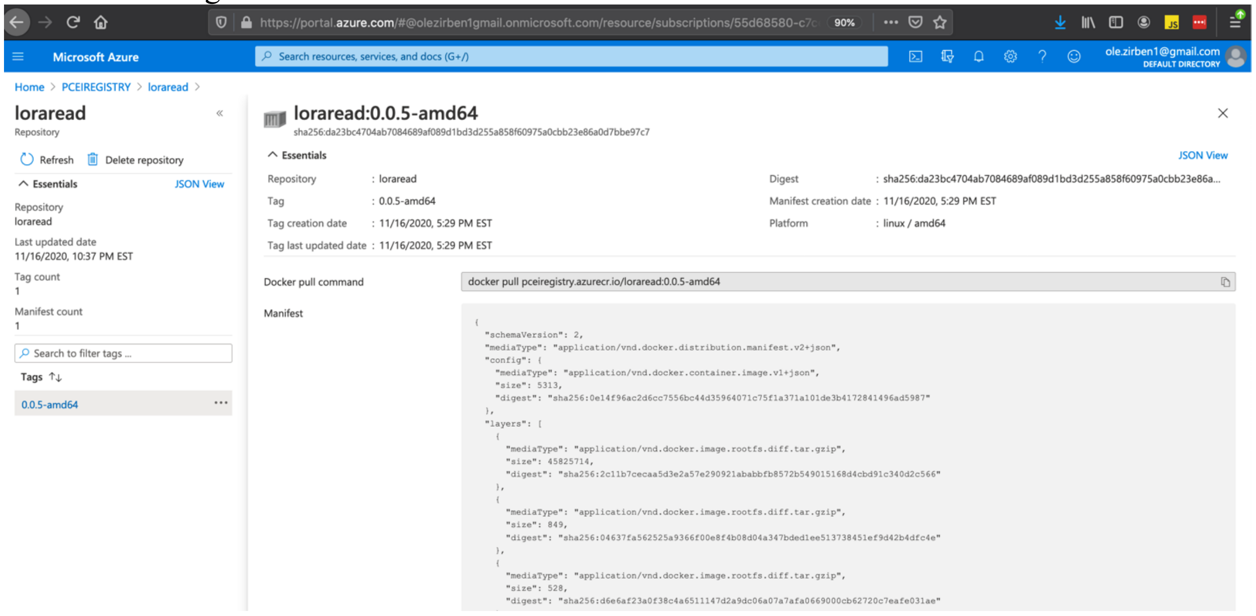
Add required credentials for Azure Container Registry (ACR) using .env file.
Build and push the solution to ACR as shown below. Righ-click on "deployment.template.json":



```
{
  "$schema-template": "1.0.0",
  "modulesContent": {
    "$edgeAgent": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "runtime": {
          "type": "docker",
          "settings": {
            "minDockerVersion": "v1.25",
            "loggingOptions": "",
            "registryCredentials": {
              "pceiregistry": {
                "username": "$CONTAINER_REGISTRY_USERNAME_pceiregistry",
                "password": "$CONTAINER_REGISTRY_PASSWORD_pceiregistry",
                "address": "pceiregistry.azurecr.io"
              }
            }
          }
        },
```

```
istry.azurecr.io/loraread:0.0.5-amd64
ry [pceiregistry.azurecr.io/loraread]
 exists
 exists
 exists
 exists
 exists
 exists
 exists
 exists
 exists
 exists
 exists
:da23bc4704ab7084689af089d1bd3d255a858f60975a0cbb23e86a0d7bbe97c7 size: 2614
ution oberzin$ 
```

The docker image for the custom module should now be visible in Azure Cloud ACR:


https://portal.azure.com/#@olezirben1gmail.onmicrosoft.com/resource/subscriptions/55d68580-c7c
Microsoft Azure
Search resources, services, and docs (G+/)
ole.zirben1@gmail.com
DEFAULT DIRECTORY
Home > PCEIREGISTRY > loraread >
Refresh
Delete repository
JSON View
Search to filter tags ...
Tags

loraread
Repository

Essentials
Repository
loraread
Last updated date
11/16/2020, 10:37 PM EST
Tag count
1
Manifest count
1

0.0.5-amd64

loraread:0.0.5-amd64
sha256:da23bc4704ab7084689af089d1bd3d255a858f60975a0cbb23e86a0d7bbe97c7

Essentials

Repository	: loraread	Digest	: sha256:da23bc4704ab7084689af089d1bd3d255a858f60975a0cbb23e86a...
Tag	: 0.0.5-amd64	Manifest creation date	: 11/16/2020, 5:29 PM EST
Tag creation date	: 11/16/2020, 5:29 PM EST	Platform	: linux / amd64
Tag last updated date	: 11/16/2020, 5:29 PM EST		

Docker pull command

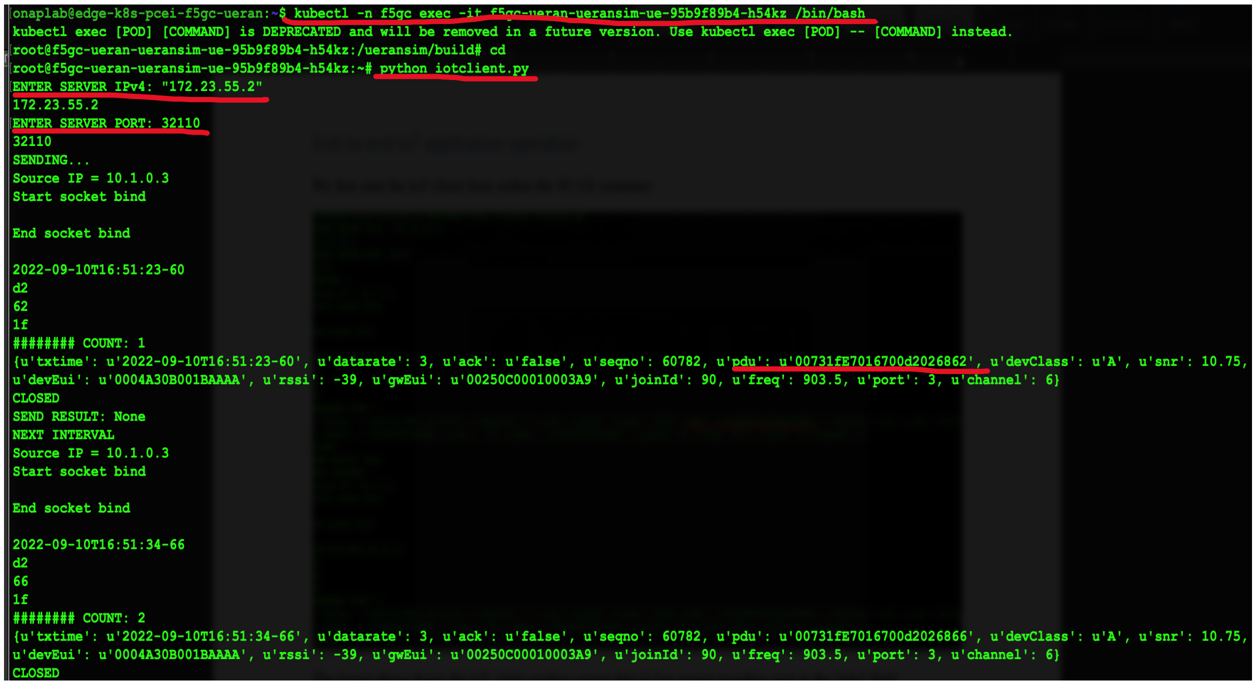
```
docker pull pceiregistry.azurecr.io/loraread:0.0.5-amd64
```

Manifest

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 5313,
    "digest": "sha256:0e14f96ac2d6cc7556bc44d35964071c75f1a371a101de3b4172841496ad5987"
  },
  "layers": [
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 45825714,
      "digest": "sha256:2c11b7cecaa5d3e2a57e290921ababbfb8572b549015168d4cbd91c340d2c566"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 849,
      "digest": "sha256:04637fa562525a9366f00e8f4b08d04a347bded1ee513738451ef9d42b4dfc4e"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 528,
      "digest": "sha256:d6e6af23a0f38c4a6511147d2a9dc06a07a7afa0669000cb62720c7eafe031ae"
```

End-to-end IoT application operation

We first start the IoT client from within the 5G UE container:

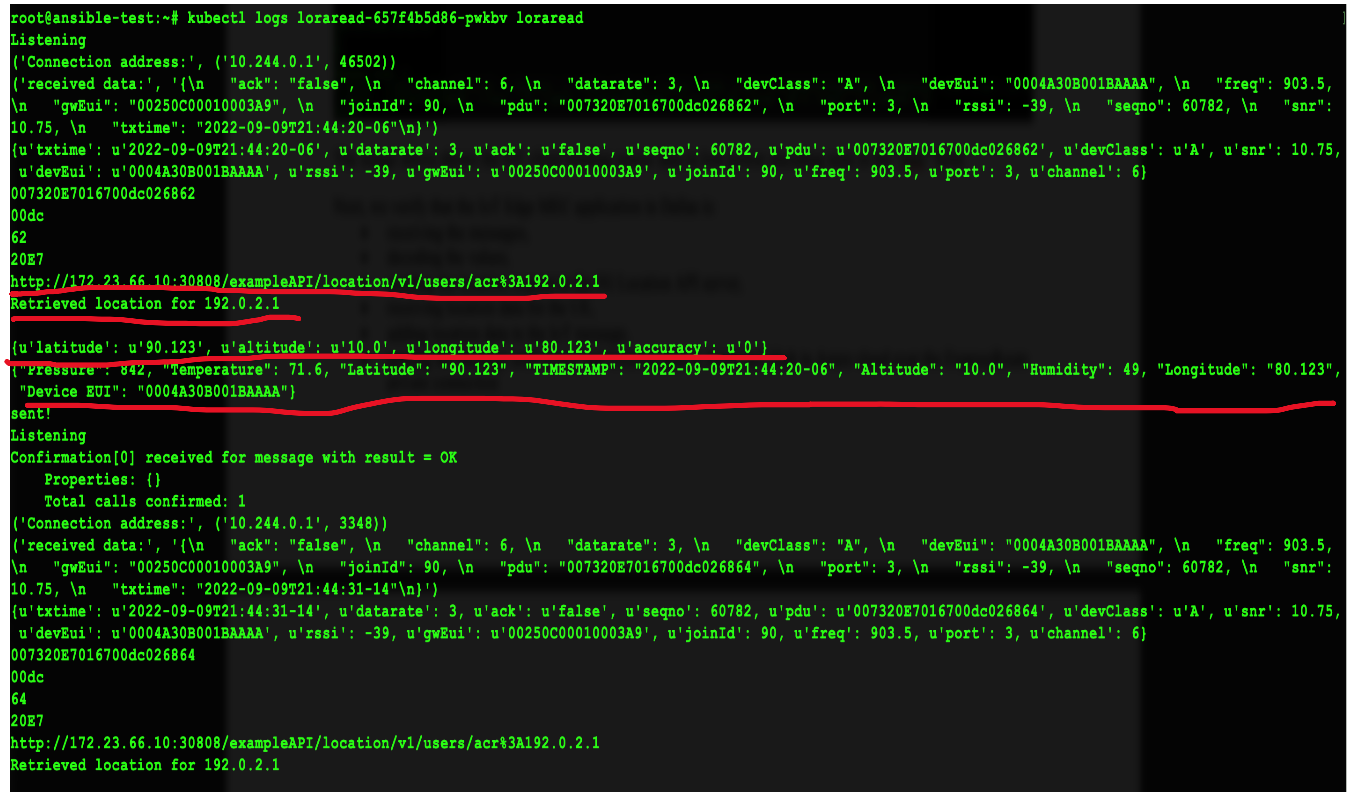


```
onaplab@edge-k8s-pcei-f5gc-ueran:~$ kubectl -n f5gc exec -it f5gc-ueran-ueransim-ue-95b9f89b4-h54kz /bin/bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:/ueransim/build# cd
root@f5gc-ueran-ueransim-ue-95b9f89b4-h54kz:~# python iotclient.py
ENTER SERVER IPv4: "172.23.55.2"
172.23.55.2
ENTER SERVER PORT: 32110
32110
SENDING...
Source IP = 10.1.0.3
Start socket bind

End socket bind

2022-09-10T16:51:23-60
d2
62
1f
######## COUNT: 1
{u'txtime': u'2022-09-10T16:51:23-60', u'datarate': 3, u'ack': u'false', u'seqno': 60782, u'pdu': u'00731fE7016700d2026862', u'devClass': u'A', u'snr': 10.75,
u'devEui': u'0004A30B001BAAAA', u'rssi': -39, u'gwEui': u'00250C00010003A9', u'joinId': 90, u'freq': 903.5, u'port': 3, u'channel': 6}
CLOSED
SEND RESULT: None
NEXT INTERVAL
Source IP = 10.1.0.3
Start socket bind

End socket bind

2022-09-10T16:51:34-66
d2
66
1f
######## COUNT: 2
{u'txtime': u'2022-09-10T16:51:34-66', u'datarate': 3, u'ack': u'false', u'seqno': 60782, u'pdu': u'00731fE7016700d2026866', u'devClass': u'A', u'snr': 10.75,
u'devEui': u'0004A30B001BAAAA', u'rssi': -39, u'gwEui': u'00250C00010003A9', u'joinId': 90, u'freq': 903.5, u'port': 3, u'channel': 6}
CLOSED
```

The image above shows the IoT client sending sensor data in low power encoding seen in the "pdu" field.

Next, we verify that the IoT Edge MEC application in Dallas is:

- receiving the messages,
- decoding the values,
- sending the location request to the 5G Location API server,
- receiving location data for the UE,
- adding location data to the IoT message,
- posting the message with sensor and location data to Azure IoT Hub in Azure cloud over the ExpressRoute private connection

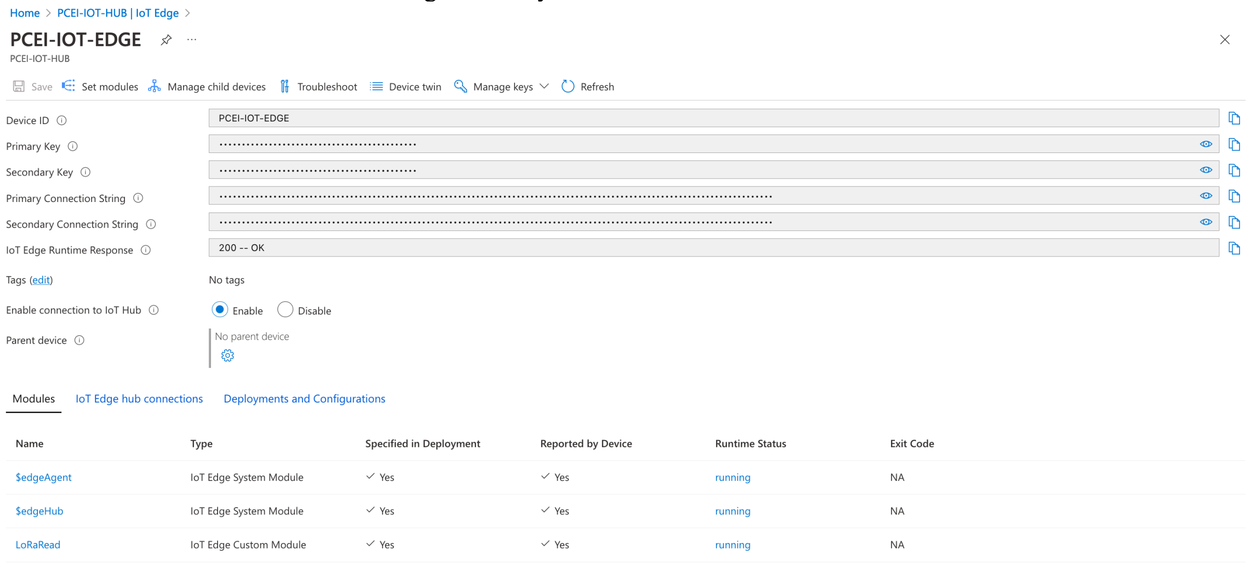


```
root@ansible-test:~# kubectl logs loraread-657f4b5d86-pwkbv loraread
Listening
('Connection address:', ('10.244.0.1', 46502))
('received data:', '{\n   "ack": "false", \n   "channel": 6, \n   "datarate": 3, \n   "devClass": "A", \n   "devEui": "0004A30B001BAAAA", \n   "freq": 903.5,
\n   "gwEui": "00250C00010003A9", \n   "joinId": 90, \n   "pdu": "007320E7016700dc026862", \n   "port": 3, \n   "rssi": -39, \n   "seqno": 60782, \n   "snr":
10.75, \n   "txtime": "2022-09-09T21:44:20-06"\n}')
{u'txtime': u'2022-09-09T21:44:20-06', u'datarate': 3, u'ack': u'false', u'seqno': 60782, u'pdu': u'007320E7016700dc026862', u'devClass': u'A', u'snr': 10.75,
 u'devEui': u'0004A30B001BAAAA', u'rssi': -39, u'gwEui': u'00250C00010003A9', u'joinId': 90, u'freq': 903.5, u'port': 3, u'channel': 6}
007320E7016700dc026862
00dc
62
20E7
http://172.23.66.10:30808/exampleAPI/location/v1/users/acr%3A192.0.2.1
Retrieved location for 192.0.2.1

{u'latitude': u'90.123', u'altitude': u'10.0', u'longitude': u'80.123', u'accuracy': u'0'}
{"Pressure": 842, "Temperature": 71.6, "Latitude": "90.123", "TIMESTAMP": "2022-09-09T21:44:20-06", "Altitude": "10.0", "Humidity": 49, "Longitude": "80.123",
 "Device EUI": "0004A30B001BAAAA"}
sent!
Listening
Confirmation[0] received for message with result = OK
    Properties: {}
    Total calls confirmed: 1
('Connection address:', ('10.244.0.1', 3348))
('received data:', '{\n   "ack": "false", \n   "channel": 6, \n   "datarate": 3, \n   "devClass": "A", \n   "devEui": "0004A30B001BAAAA", \n   "freq": 903.5,
\n   "gwEui": "00250C00010003A9", \n   "joinId": 90, \n   "pdu": "007320E7016700dc026864", \n   "port": 3, \n   "rssi": -39, \n   "seqno": 60782, \n   "snr":
10.75, \n   "txtime": "2022-09-09T21:44:31-14"\n}')
{u'txtime': u'2022-09-09T21:44:31-14', u'datarate': 3, u'ack': u'false', u'seqno': 60782, u'pdu': u'007320E7016700dc026864', u'devClass': u'A', u'snr': 10.75,
 u'devEui': u'0004A30B001BAAAA', u'rssi': -39, u'gwEui': u'00250C00010003A9', u'joinId': 90, u'freq': 903.5, u'port': 3, u'channel': 6}
007320E7016700dc026864
00dc
64
20E7
http://172.23.66.10:30808/exampleAPI/location/v1/users/acr%3A192.0.2.1
Retrieved location for 192.0.2.1
```

Note the Latitude, Longitude, Altitude, Temperature, Humidity and Pressure values in the decoded message.

We can also see the IoT messages received by the IoT Hub service in the Azure IoT cloud from the IoT Edge Gateway running on the MEC server in Dallas:

Below is the status of the Azure IoT Edge Gateway as seen from the Azure Cloud



PCEI-IOT-EDGE

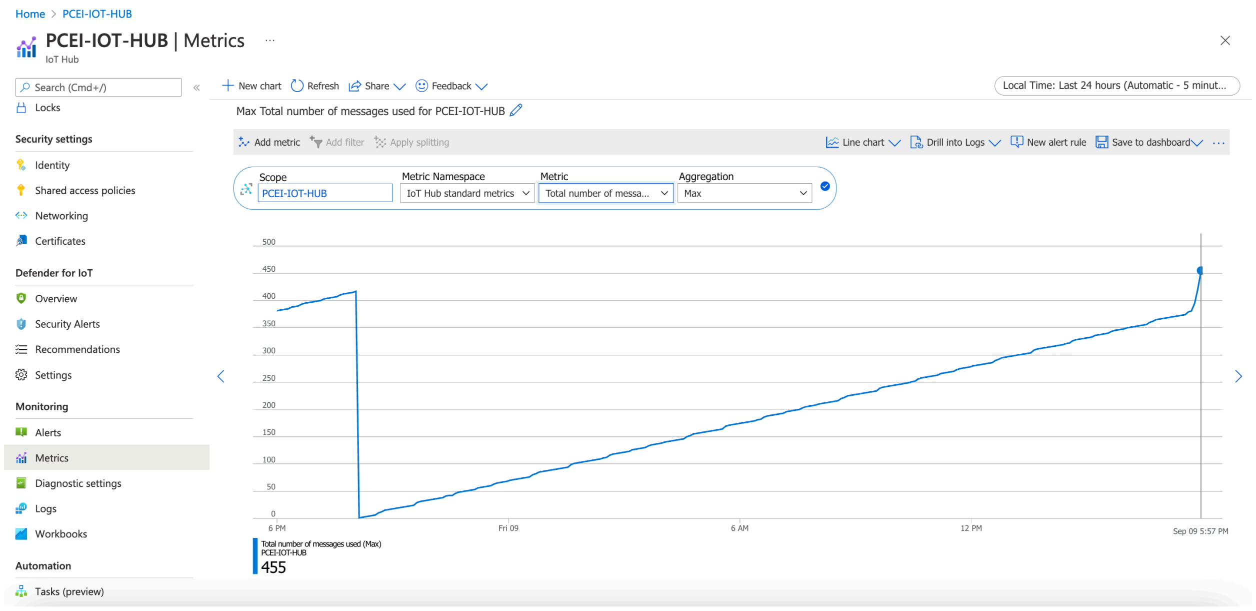
PCEI-IOT-HUB

Device ID	PCEI-IOT-EDGE
Primary Key	•••
Secondary Key	•••
Primary Connection String	••
Secondary Connection String	••
IoT Edge Runtime Response	200 -- OK
Tags (edit)	No tags
Enable connection to IoT Hub	[✓ Enable] Disable
Parent device	No parent device

Modules | IoT Edge hub connections | Deployments and Configurations

Name	Type	Specified in Deployment	Reported by Device	Runtime Status	Exit Code
$edgeAgent	IoT Edge System Module	✓ Yes	✓ Yes	running	NA
$edgeHub	IoT Edge System Module	✓ Yes	✓ Yes	running	NA
LoRaRead	IoT Edge Custom Module	✓ Yes	✓ Yes	running	NA

The IoT message count reception is shown below


Home > PCEI-IOT-HUB

PCEI-IOT-HUB | Metrics

IoT Hub

Max Total number of messages used for PCEI-IOT-HUB

Local Time: Last 24 hours (Automatic - 5 minut...

Scope: PCEI-IOT-HUB
Metric Namespace: IoT Hub standard metrics
Metric: Total number of messa...
Aggregation: Max

Total number of messages used (Max)
PCEI-IOT-HUB
455

Implementation details

IoT Client script

```
'''

@author: oberzin
'''
import socket
from time import sleep
import json
import os
import random
import datetime
import struct
import subprocess

SERVER_ADDRESS = '10.121.7.149'
SERVER_PORT = 30834
BUFFER_SIZE = 500
DEVICE_EUI = '0004A30B001BAAAA'
COUNT = 0;
INTERVAL = 10
STM_PLOAD = '007327E7016700CB02683C'

def client_send(SERVER, PORT, BUFFER):
    TCP_IP = SERVER
    TCP_PORT = PORT
    BUFFER_SIZE = BUFFER  # Normally 1024, but we want fast response

    cmd = 'ip a |grep uesim | grep \'10.1\' | cut -b 10-17'
    SRC_IP = str(subprocess.check_output(cmd, shell=True)).strip()
    #SRC_IP = "10.1.0.4"
    SRC_PORT = 32000

    print 'Source IP = %s' % SRC_IP

    global DEVICE_EUI
    global COUNT

#"{"Count": 53227, "Temperature": 71.6, "Device EUI": "0004A30B001B5E47"}"

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    print "Start socket bind \n"
    s.bind((SRC_IP, SRC_PORT))
    print "End socket bind \n"
    s.connect((TCP_IP, TCP_PORT))

    COUNT = COUNT + 1
    #s.connect((TCP_IP, TCP_PORT))
    now = datetime.datetime.now()
    curr_ts = now.strftime('%Y-%m-%dT%H:%M:%S') + ('-%02d' % (now.microsecond / 10000))
    print curr_ts
    tempDataDec = random.randint(21, 23) * 10
    tempDataHex = hex(tempDataDec)
    tempDataHexStr = str(bytearray([tempDataDec])).encode('hex')
    print tempDataHexStr

    humidDataDec = random.randint(49, 51) * 2
    humidDataHex = hex(humidDataDec)
    humidDataHexStr = str(bytearray([humidDataDec])).encode('hex')
    print humidDataHexStr

    pressDataDec = random.randint(30, 33)
    pressDataHex = hex(pressDataDec)
    pressDataHexStr = str(bytearray([pressDataDec])).encode('hex')
    print pressDataHexStr
    raw_message = {
                    "ack":"false","channel":6,"datarate":3,"devClass":"A",
                    "devEui":DEVICE_EUI,"freq":903.5,"gwEui":"00250C00010003A9",
                    "joinId":90,
                    'pdu': "0073" + pressDataHexStr + "E7016700" + tempDataHexStr + "0268" +
humidDataHexStr,
                    "port":3,"rssi":-39,"seqno":60782,"snr":10.75,
                    "txtime":curr_ts
                    }

    #print raw_message
    print '######## COUNT: %s' % COUNT
        #json.dumps(myDictObj, sort_keys=True, indent=3)
    ser_message = json.dumps(raw_message, sort_keys=True, indent=3)
    #print ser_message
    message = json.loads(ser_message)
    print message

    s.send(str(ser_message))
#    result = s.recv(BUFFER_SIZE)
    s.close()
    print ("CLOSED")
#    return result
    return

def main():

    #s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #s.connect((SERVER_ADDRESS, SERVER_PORT))
    SERVER_ADDRESS = input ("ENTER SERVER IPv4: ")
    print (SERVER_ADDRESS)
    SERVER_PORT = input ("ENTER SERVER PORT: ")
    print (SERVER_PORT)
    print 'SENDING...'
    while True:
        try:
            res = client_send(SERVER_ADDRESS, SERVER_PORT, BUFFER_SIZE)
            print 'SEND RESULT: %s' % res
            #sleep(INTERVAL)
        except Exception as ex:
            print ex
            pass
        sleep(INTERVAL)
        print ("NEXT INTERVAL")

    #s.close()

if __name__ == '__main__':
        main()
```

IoT Edge Gateway custom module with Location API integration

```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
import SocketServer
import json

import socket
from time import sleep

import requests

import cgi
import random
import time
import sys
import iothub_client
# pylint: disable=E0611
from iothub_client import IoTHubModuleClient, IoTHubClientError, IoTHubTransportProvider
from iothub_client import IoTHubMessage, IoTHubMessageDispositionResult, IoTHubError

# messageTimeout - the maximum time in milliseconds until a message times out.
# The timeout period starts at IoTHubModuleClient.send_event_async.
# By default, messages do not expire.
MESSAGE_TIMEOUT = 10000

# global counters
RECEIVE_CALLBACKS = 0
SEND_CALLBACKS = 0

# Choose HTTP, AMQP or MQTT as transport protocol.  Currently only MQTT is supported.
PROTOCOL = IoTHubTransportProvider.MQTT

# Callback received when the message that we're forwarding is processed.
def send_confirmation_callback(message, result, user_context):
    global SEND_CALLBACKS
    print ( "Confirmation[%d] received for message with result = %s" % (user_context, result) )
    map_properties = message.properties()
    key_value_pair = map_properties.get_internals()
    print ( "    Properties: %s" % key_value_pair )
    SEND_CALLBACKS += 1
    print ( "    Total calls confirmed: %d" % SEND_CALLBACKS )


# receive_message_callback is invoked when an incoming message arrives on the specified
# input queue (in the case of this sample, "input1").  Because this is a filter module,
# we will forward this message onto the "output1" queue.
def receive_message_callback(message, hubManager):
    global RECEIVE_CALLBACKS
    message_buffer = message.get_bytearray()
    size = len(message_buffer)
    print ( "    Data: <<<%s>>> & Size=%d" % (message_buffer[:size].decode('utf-8'), size) )
    map_properties = message.properties()
    key_value_pair = map_properties.get_internals()
    print ( "    Properties: %s" % key_value_pair )
    RECEIVE_CALLBACKS += 1
    print ( "    Total calls received: %d" % RECEIVE_CALLBACKS )
    hubManager.forward_event_to_output("output1", message, 0)
    return IoTHubMessageDispositionResult.ACCEPTED


class HubManager(object):

    def __init__(
            self,
            protocol=IoTHubTransportProvider.MQTT):
        self.client_protocol = protocol
        self.client = IoTHubModuleClient()
        self.client.create_from_environment(protocol)

        # set the time until a message times out
        self.client.set_option("messageTimeout", MESSAGE_TIMEOUT)

        # sets the callback when a message arrives on "input1" queue.  Messages sent to
        # other inputs or to the default will be silently discarded.
        self.client.set_message_callback("input1", receive_message_callback, self)

    # Forwards the message received onto the next stage in the process.
    def forward_event_to_output(self, outputQueueName, event, send_context):
        self.client.send_event_async(
            outputQueueName, event, send_confirmation_callback, send_context)
```

```
def get_location(LOC_SRV, LOC_SRV_PORT, device_name):

    check_url = 'http://' + str(LOC_SRV) + ':' + str(LOC_SRV_PORT) +
'/exampleAPI/location/v1/users/acr%3A' + str(device_name)

    print check_url

    response = requests.get(check_url)
    # sleep(10)
    if (response.status_code == 200):
        print 'Retrieved location for %s\n' % device_name
        location = json.loads(response.text)
        # print alarms
        return location
    else:
        print '### Location Request Timeout %s\n' % device_name
        print 'Response code = %d ... Retrying ...\n' % response.status_code
        location = []



#
*************************************************************************************************
******
def run():

    hub_manager = HubManager()

    TCP_IP = '0.0.0.0'
    TCP_PORT = 50005
    BUFFER_SIZE = 500  # Normally 1024, but we want fast response

    LOC_SRV = "172.23.66.10"
    LOC_SRV_PORT = 30808
    device_name = "192.0.2.1"

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((TCP_IP, TCP_PORT))

    #sock.listen(1)
    #conn, addr = sock.accept()

    while 1:
        sock.listen(1)
        print ( "Listening" )
        conn, addr = sock.accept()
        print ( 'Connection address:', addr )
        data = conn.recv(BUFFER_SIZE)
        # if not data: return
        print ( "received data:", data )
        jData = json.loads(data)
        print ( jData )
        appData=jData["pdu"]
        print ( appData )
        tempData = appData[12:16]
        print ( tempData )
        tempDataDec = int(tempData, 16)/10
        tempDataDecF = tempDataDec * 1.8 + 32

        humidData = appData[20:22]
        print ( humidData )
        humidDataDec = int(humidData, 16)/2

        pressData = appData[4:8]
        print ( pressData )
        pressDataDec = int(pressData, 16)/10
        device_eui = jData["devEui"]
        txtime = jData["txtime"]
    # print "\nAppData: %s" % appData
    # print "\nTemperature in Celsius: %s" % tempDataDec
    # print "\nTemperature in Fahrenheit: %s" % tempDataDecF
    # print "\nHUMIDITY Percents: %s" % humidDataDec
    # print "\nBAROMETRIC PRESSURE in Millibars: %s" % pressDataDec
    # print "\nDevice EUI: %s" % device_eui

    # payld = json.dumps({'Device EUI': device_eui, 'Temperature': tempDataDecF, \
    #                      'Humidity': humidDataDec, 'Pressure': pressDataDec, 'Count':
LORA_READ_COUNT})


           location = get_location(LOC_SRV, LOC_SRV_PORT, device_name)

           print location["userInfo"]["locationInfo"]

        lat = location["userInfo"]["locationInfo"]["latitude"]
        lon = location["userInfo"]["locationInfo"]["longitude"]
        alt = location["userInfo"]["locationInfo"]["altitude"]

        packet = json.dumps({'TIMESTAMP': txtime, 'Device EUI': device_eui, 'Temperature':
tempDataDecF, \
                             'Humidity': humidDataDec, 'Pressure': pressDataDec, \
                             'Latitude': lat, 'Longitude': lon, 'Altitude': alt})

        #packet = json.dumps({'TIMESTAMP': txtime, 'Device EUI': device_eui, 'Temperature':
tempDataDecF, \
        #                'Humidity': humidDataDec, 'Pressure': pressDataDec})

           print ( packet )

        iot_hub_message = IoTHubMessage(packet)
        #iot_hub_message = IoTHubMessage(json.dumps(packet))
        hub_manager.forward_event_to_output("output1", iot_hub_message, 0)
        print('sent!')

        #conn.close()

if __name__ == "__main__":
    run()
```

Terraform, Ansible, Camunda and Helm repositories

You can find all artifacts hosted in the following public GIT repositories

Terraform plans
https://gitlab.com/akraino-pcei-onap-cds/terraform-plans/-/tree/main/etsi-lfedge-hackathon-2022

Ansible playbooks
https://gitlab.com/akraino-pcei-onap-cds/ansible-scripts/-/tree/main/etsi-lfedge-hackathon-2022

Helm3 charts
https://gitlab.com/akraino-pcei-onap-cds/equinix-pcei-poc/-/tree/main/helm3-charts/etsi-lfedge-hackathon-2022

Camunda workflows
https://gitlab.com/akraino-pcei-onap-cds/camunda-bpmn-samples/-/tree/main/etsi-lfedge-hackathon-workflow

Acknowledgements

The authors would like to acknowledge the following individuals for their critical contributions to the implementation and validation of this project:

- Kavitha Papanna <pkavitha@aarnanetworks.com>
- Premkumar Subramaniyan <premkumar@aarnanetworks.com>
- Sai Lakshmi Cheedella <sailakshmi@aarnanetworks.com>
- Namachi S <namachi@aarnanetworks.com>