TABLE OF CONTENTS

Where’s Edge workloads running?

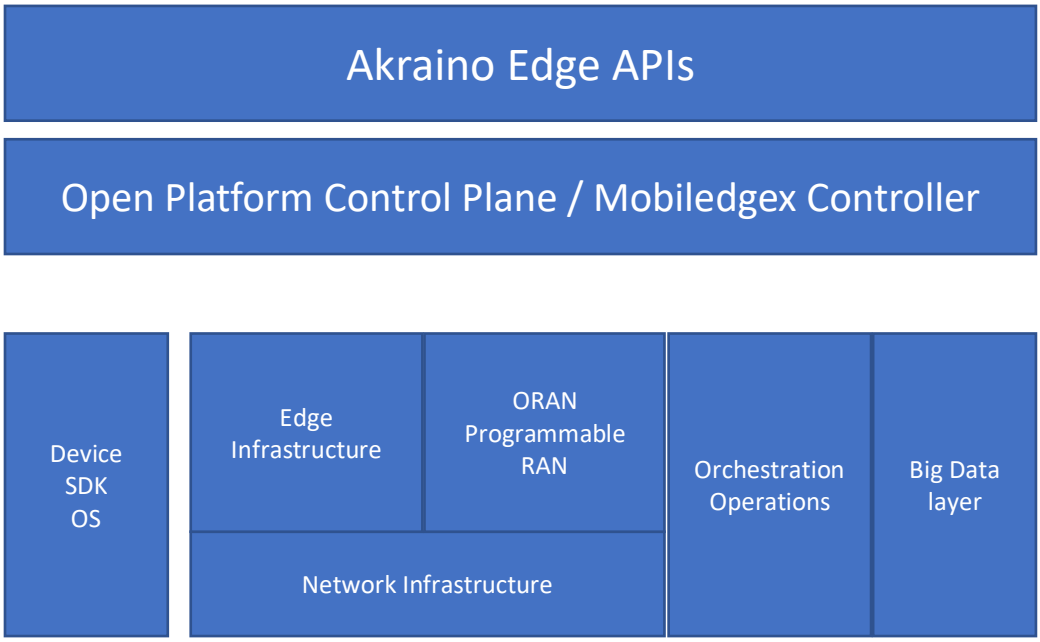
Is edge on the device or on premise or nearest cache on internet from where you are consuming the content. No matter where the edge workload runs it is apparent across all the domains of projects under Linux foundation edge that you need certain basic functionality to manage and run a fleet of devices or computers. The challenges across all edge projects relate to distributed workloads processing and possibly mobility.

What we need is an end to end service creation framework where in the developers can write new edge applications. A software development and deployment platform which offers open API Library for any developer to discover and consume the services the edge has to offer. We can learn from cloud model today how to develop, orchestrate and manage workloads/applications and offer developers similar CI/CD pipeline functionality. Not to say that as devices and machines come online all the needs will be met but at least it provides a framework to look at.

In principle whoever is developing a solution for edge will appreciate resources scaling desired to addressing power limitation either on device or on premise or near the far edge telco Radios. Hence the dynamic workload, specialized resources for acceleration, server less models are emerging in most of the Akraino Blueprints.

From API summit we learned a few use cases which we would like to present the Architecture and APIs for a continued learning in the community to develop open APIs for next generation of developers to write applications for next generation of internet at edge.

API framework



The above diagram shows a high-level view of what would be an ideal open API Platform to build applications serving real edge environment. Please note that RAN access is optional in the stack for on premise deployments.

Open RAN: Software Defined, Programmable Radio Access Network

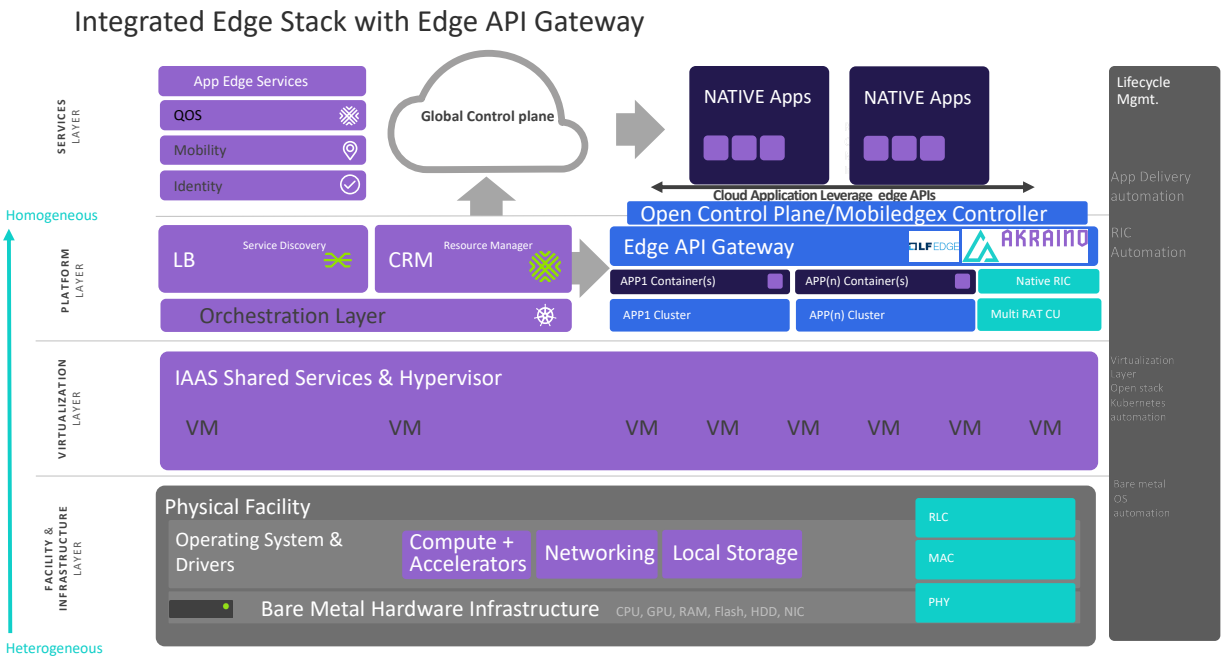
Network Infrastructure: Software Defined Network Layer for Control and Data Plane

Edge infrastructure: Low-latency Multi-Access Edge Computing Platform

Orchestration and Operations: Service Orchestration & Management: Application Management and Service Orchestration

Data & Services Marketplace: Collection of Data from network, edge infra, application for a comprehensive monitoring panel and big data intelligence for Developers to see their Application execution in real-time

What's this Open Platform stack would look like?



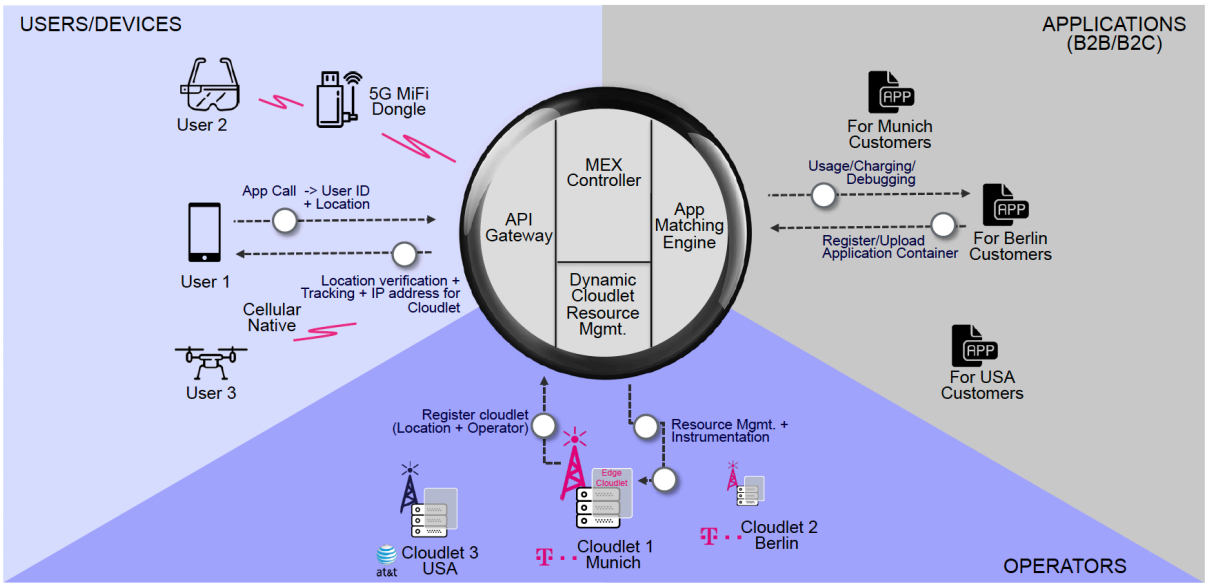
The open platform control plane shall provide API libraries Run-time Environment and package all the following APIs in a usable format which normal developers can build applications on. we shall at high level include the following set of APIs across the functional blocks described above:

- SDK: Device Side centric API set will include
  - ›Register
  - ›Metrics
  - ›Discovery
  - ›Advertise resource capability
  - ›Publish and subscribe events
- Orchestration and operations API set will include:
  - ›Client Registration API
  - ›Cellular Control Plane API for identity, location, QOE
  - ›Infrastructure Orchestration API
  - ›Application on-boarding API
  - ›Resource Isolation API
  - ›Metrics API
  - ›Metering API
  - ›Application-focused Privacy Policy API

- Network Control RAN or Access network
  - ›RAN slice Programmability APIs
  - ›Monitor per session APIs
  - ›Programmability per session APIs
- Edge Infrastructure
  - ›Programmable Data path: Match Action Pipeline APIs

- ›Datapath Acceleration APIs
- ›Resource Inventory APIs

MobiledgeX: Open Platform deployed for application Orchestration and operations:

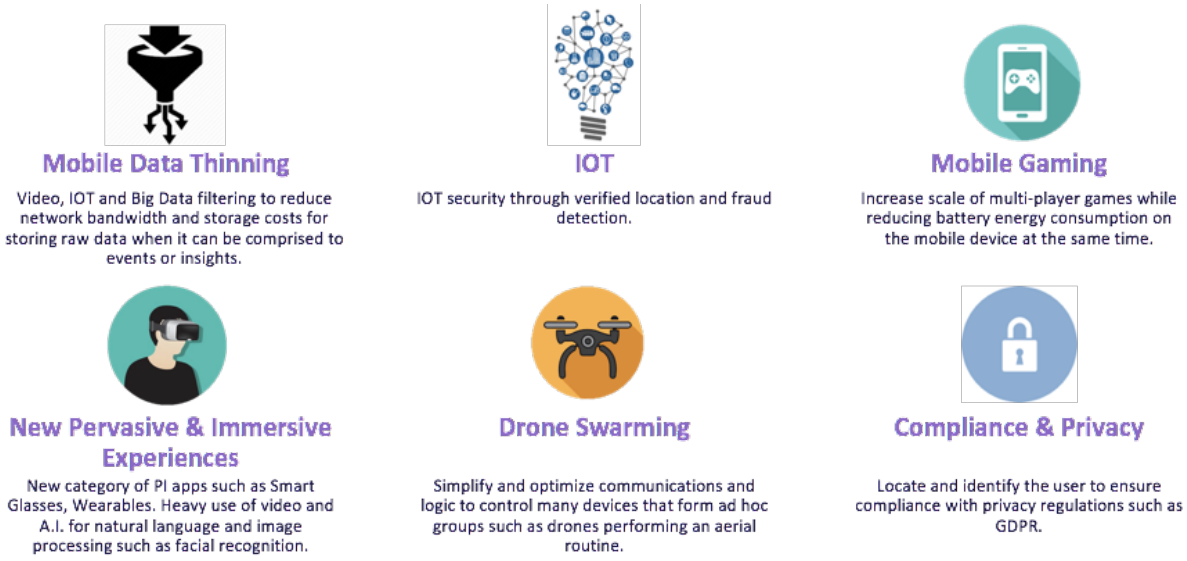


MobiledgeX Controller today is aligned with the above strategy helps with backend application deployment and plans to build out a policy control place to specify the application needs to RIC on south bound side. Controller specifies policies at the granularity of minutes to hours (when it onboard/offboards an application backend) and the xapp does the more real time adjustments of the policy.

Markets use cases:

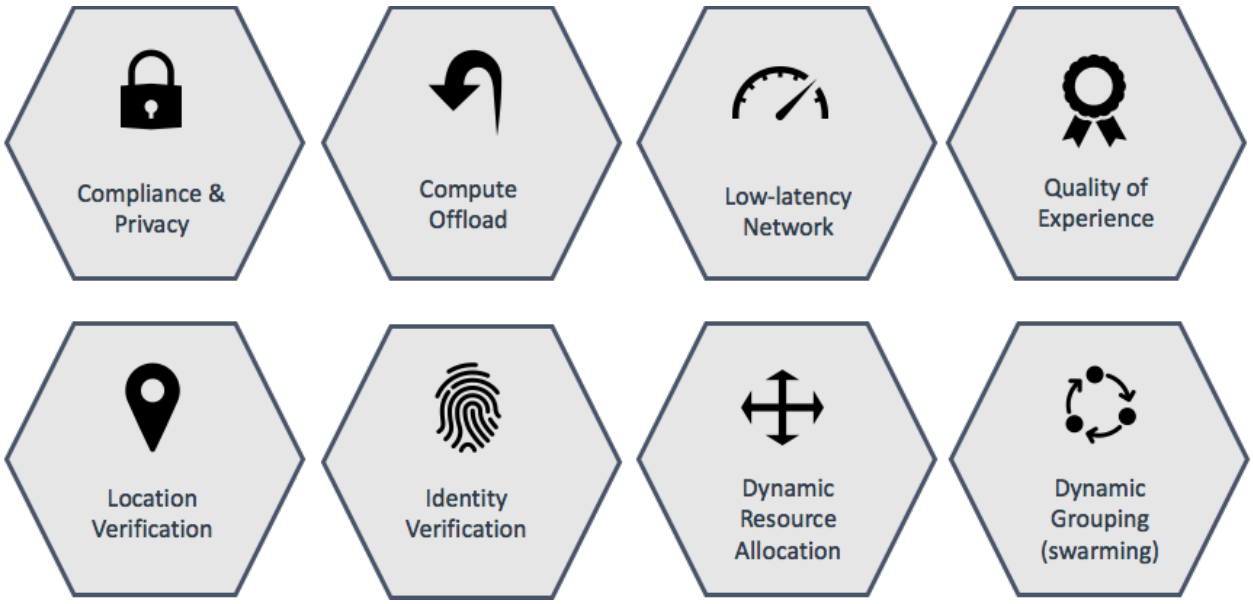
Edge with Mobility: Telco Edge

Some of the use cases described below desire a telco edge to serve these workloads.



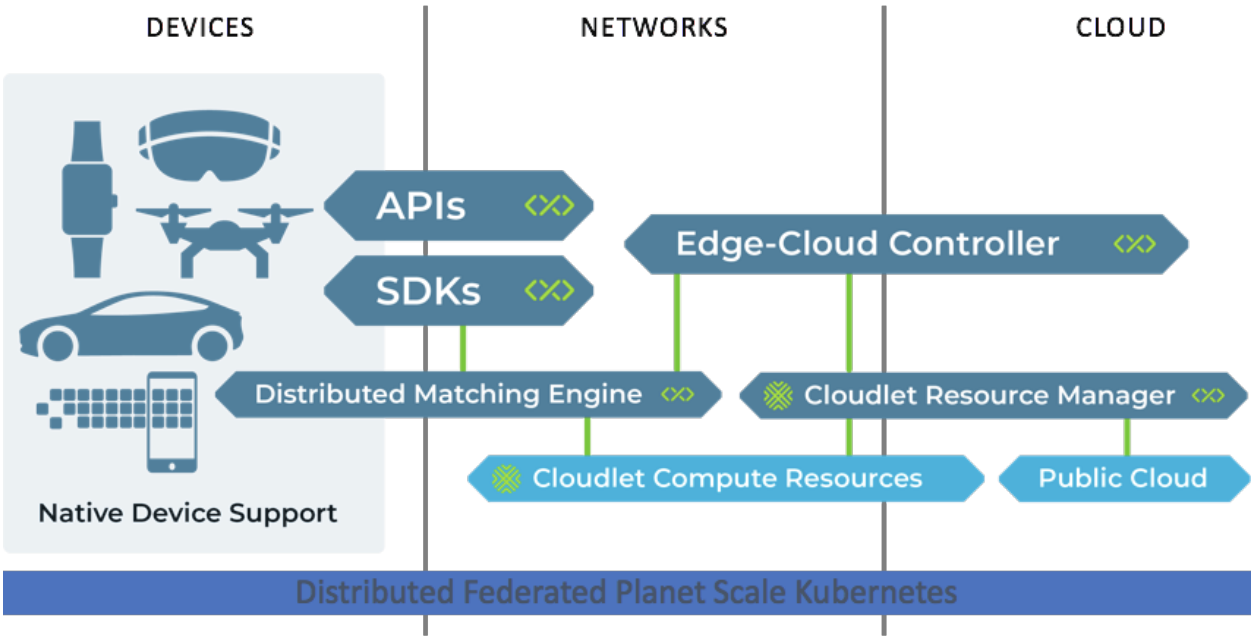
Edge Features

What features are high value to serve the above cases are as follows:



Architecture

what Architecture would allow us to serve the distributed and application mobility workload



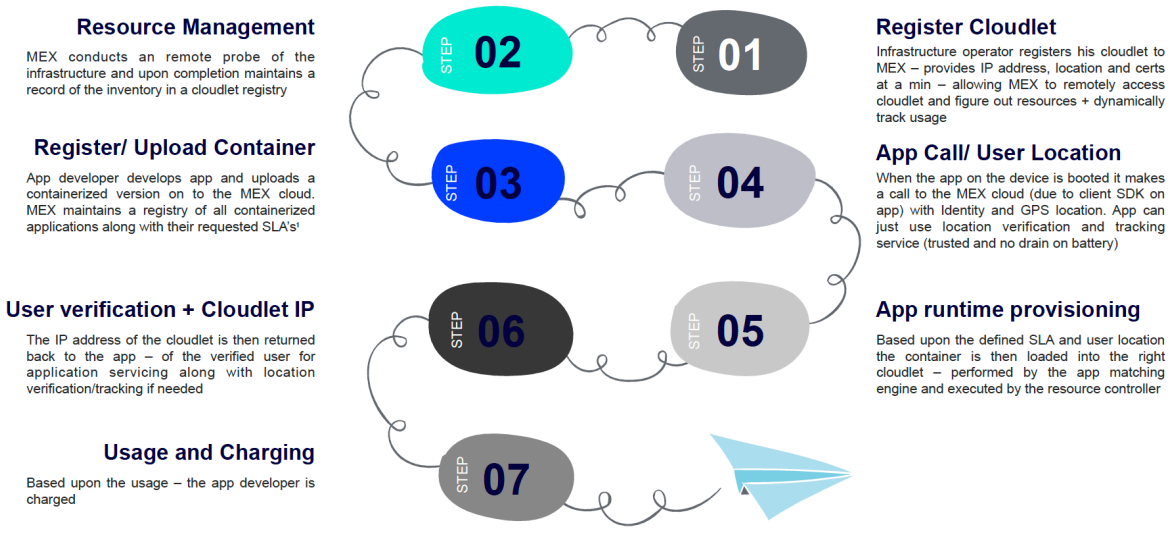
Workflow:

Operational workflow:

- UE downloads the App with client SDK
- Upon activation, UE's App is connected to edge cloud
- App intelligently programs (e.g performance metrics such as connectivity bandwidth, latency, degradation parameters etc.) that the App needs to be connected to edge platform
- App discovers the 'nearby' edge services - Mobiledgex Platform uses the location checks and some form of 'requirements check/resource needs' to determine 'best' edge platform to execute the App on.
- App performs a handshake between UE and Edge platform (authentication, secure channel, transactional handshake here)
- App is instantiated and execution begins on the Edge platform
- Mobiledgex Platform handles the Application Mobility scenario when the App maybe on a moving UE and now needs to connect to another Edge (e.g Edge Handover)
- Mobiledgex Platform monitors the App throughout, from the instance it is created on the edge, mobility to another edge if happens and till the App finishes execution

Complete Orchestration workflow:

App developer specifies SLA (e.g. region etc.) when uploading his containerized app. App is then pre-loaded onto cloudlet based upon the SLA. Developer may be charged based upon tenancy. As of today, there isn't significant contention for cloudlet resources – hence a static system allows us to provide reserved instances to developers to ensure a robust and high quality user experience



Client-Side API set

Key APIs

Available Today	In Development
`registerClient`	`watchLocation`
`findCloudlet`	`addUserToDynLocationGrp`
`verifyLocation`	`predictQualityOfExperience`

registerClient –

- Registers client with closest Distributed Matching Engine
- Validates legitimacy of mobile subscriber

- Verifies that billing agreement is in place
- All session information is encrypted

This is the first call that the client makes to the SDK. The SDK code gathers UDI information, service provider and application identification information. The SDK will connect with the appropriate Distributed Matching Engine (i.e., the nearest edge location) in the operator network over the cellular control plane. The validity of the subscriber automatically gets established, and the DME further verifies the edge billing agreement and has input on edge usage by the client for billing reasons. DME will then return a session Key/Cookie encrypted on the DME public key to validate the future calls. The DME's has well-known entries based on carrier.dme.mobiledgeX.com (managed by MobiledgeX) that the SDK will resolve to the appropriate IP address. Any user consent (for privacy, tracking, etc.) is gathered as part of this call including any pop-ups and setting the device OS options. This is the one and only time the SDK interacts with the user.

Verify-location

- Determines true subscriber location, leveraging:
    - Unique device id entifier
    - Carrier ID
    - Access point ID
    - Location of cell towers
- Return codes:
    - Location confirmed – User location is not spoofed
    - Location not found – Carrier has not provided supplemental triangulation info
    - Location spoofed

Send the Unique Device Identifier, Carrier ID, AccessPointID and Location Info to the DME where the location is determined by the on-device GPS or some other means. The DME tries to verify the location is within its accuracy range and device is indeed connected to the specified AccessPoint and returns following codes:

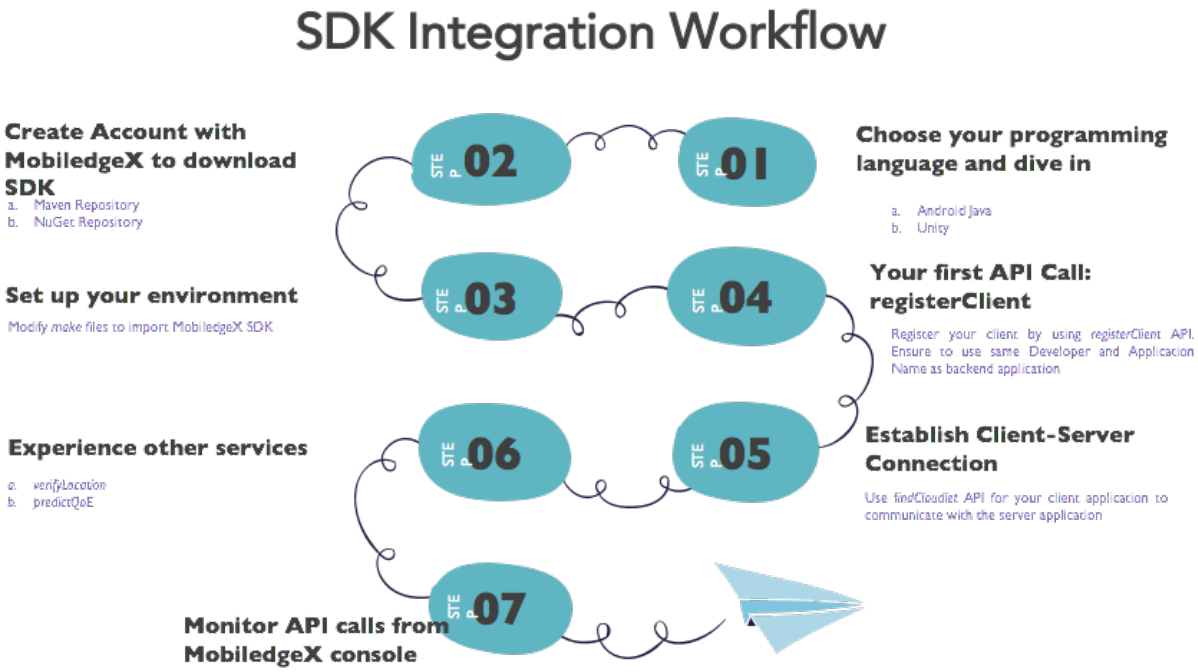
- Location Confirmed – If the device location is confirmed
- Location Not Found – If the Carrier is unable to comply with the request
- Location Spoofed – If the Carrier can confirm the no such user/device is connected to the specified access point or the location of the device is different then specified in location Info

Find-Cloudlet –

- Finds optimal edge computing footprint for workload, leveraging:

    - Location
    - Application subscription
    - Service agreements in place with mobile operator
- Client can fall back onto public cloud application instance

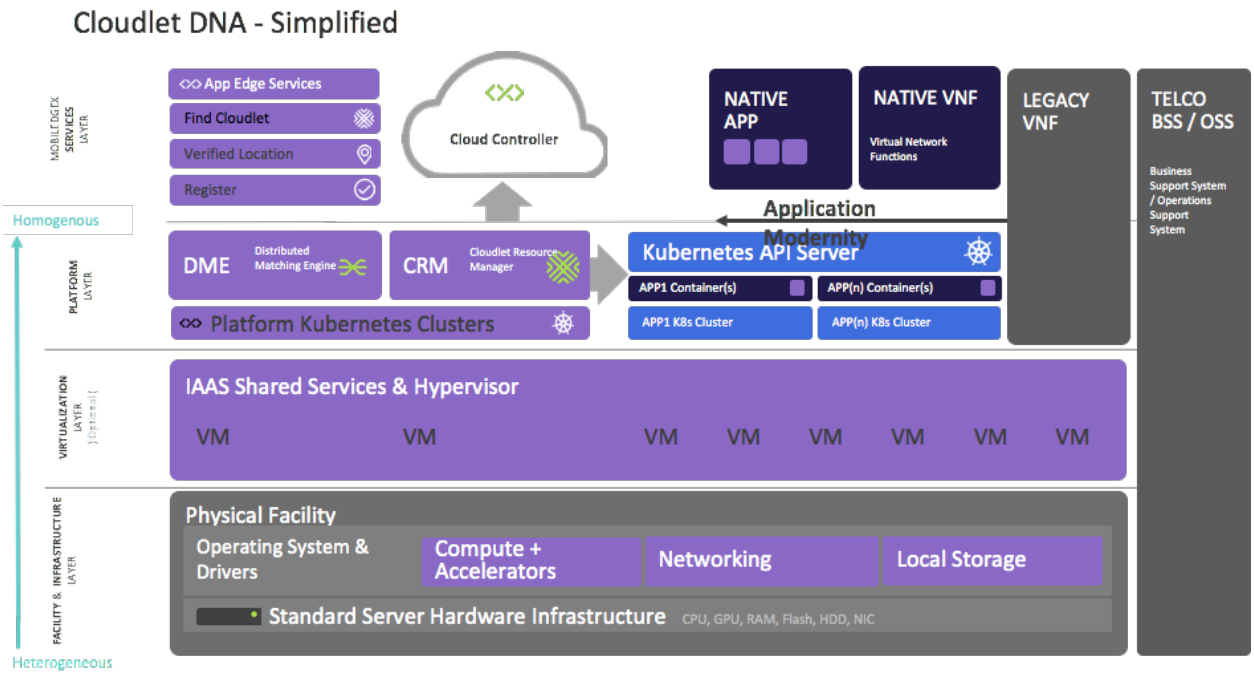
Find-Cloudlet: This is per application/user/device service discovery call that allows the application to find the application backend for offload services based on its location, application subscription, and it's service provider agreement. Note: In case of no suitable cloudlet instance available, the client has the option to connect to the application server in the public cloud.



Please visit for more information: http://swagger.mobiledgex.net/client/

Backend orchestration and management

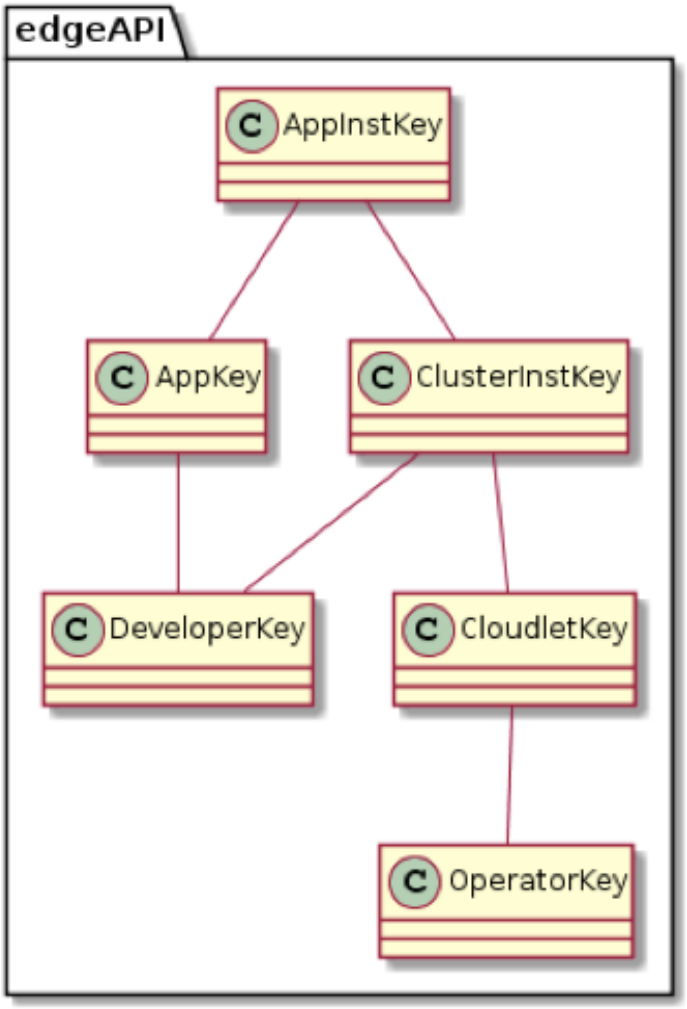
The edge stack shown below is virtualization and chip neutral and allows to bring commercial applications to edge to understand the application and developer needs to bring them to telco edge.



Key CSUD{Create/Show/Update/Delete} APIs

Resources Centric	Applications Centric
Operator|developer	Cluster inst
cloudlet	app
flavor	App inst

Key Relationships



createcloudlet -

A Cloudlet is a set of compute resources at a particular location, provided by an Operator.

```
edgectl --addr mexdemo.ctrl.mobiledgex.net:55001 --tls $TLS/mex-client.crt
controller CreateCloudlet --key-name pac --location-latitude 41.878 --
location-longitude -87.629 --numdynamicips 10
```

createdeveloper-

A developer defines a customer that can create and manage applications, clusters, instances, etc. Applications and other objects created by one Developer cannot be seen or managed by other Developers. Billing will likely be done on a per-developer basis.Creating a developer identity is likely the first step of (self-)registering a new customer.

```
edgectl --tls $TLS/mex-client.crt –addr mexdemo.ctrl.mobiledgex.net:55001
CreateDeveloper --key-name nianticdev
```

createApp-

apps are applications that may be instantiated on Cloudlets, providing a back-end service to an application client (using the SDK) running on a user device such as a cell phone, wearable, drone, etc.

Applications belong to Developers and must specify their image and accessibility. Applications are analogous to Pods in Kubernetes. An application in itself is not tied to a Cloudlet but provides a definition that can be used to instantiate it on a Cloudlet. AppInsts are applications instantiated on a particular Cloudlet."

```
edgectl --tls $TLS/mex-client.crt –addr mexdemo.ctrl.mobiledgex.net:55001
CreateApp --key-developerkey-name niantic --key-name neon --key-version 1.0 -
-accessports tcp:7777 --imagetype ImageTypeDocker --deployment kubernetes —
imagepath docker.mobiledgex.net/niantic/images/neon:1.0
```

createClusterInst-

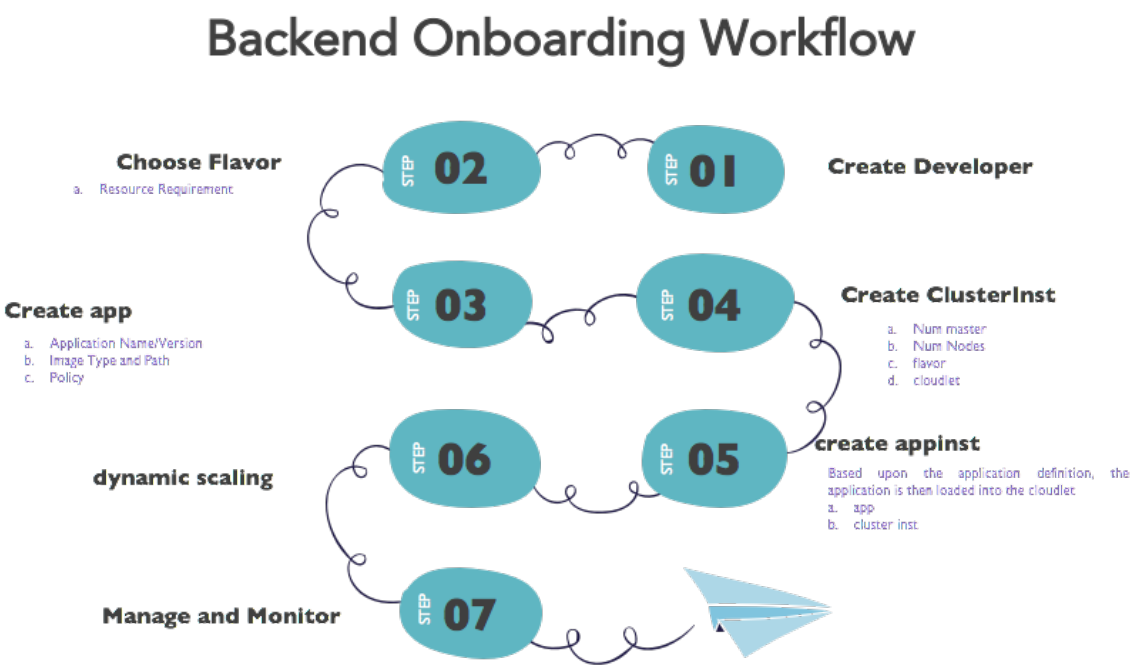
ClusterInst is an instance of a Cluster on a Cloudlet. It is defined by a Cluster plus a Cloudlet key. This separation of the definition of the Cluster versus its instance is unique to Mobiledgex, and allows the Developer to provide the Cluster definition, while either the Developer may statically define the instances, or the Mobiledgex platform may dynamically create and destroy instances in response to demand. When a Cluster is instantiated on a Cloudlet, the user may override the default Cluster Flavor of the Cluster. This allows for an instance in one location to be provided more resources than an instance in other locations, in expectation of different demands in different locations.

```
edgectl --tls $TLS/mex-client.crt --addr mexdemo.ctrl.mobiledgex.net:55001
controller CreateClusterInst --ipaccess IpAccessShared --key-cloudletkey-name
tmocloud-1 --key-cloudletkey-operatorkey-name tmus --key-clusterkey-name
nianticcluster --key-developer ninatic --flavor-name x1.medium --nummasters 1
--numnodes 3
```

createAppinst-

"AppInst is an instance of an App on a Cloudlet where it is defined by an App plus a ClusterInst key. Many of the fields here are inherited from the App definition."

```
edgectl --tls $TLS/mex-client.crt –addr mexdemo.ctrl.mobiledgex.net:55001
CreateAppInst --key-appkey-developerkey-name niantic --key-appkey-name neon -
-key-appkey-version 1.0 --key-clusterinstkey-cloudletkey-name tmocloud-1 --
key-clusterinstkey-cloudletkey-operatorkey-name tmus --key-clusterinstkey-
clusterkey-name nianticcluster --key-clusterinstkey- developer nianticdev
```



Please visit for more information: http://swagger.mobiledgex.net/client/

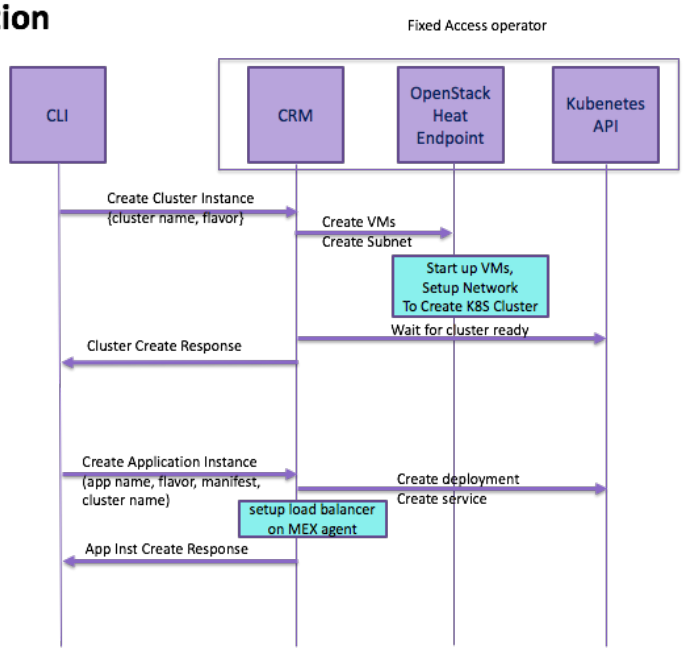
How to use the above API set?

you can refer to JSON and yaml files in case you wish to integrate in your edge stack.

[DeveloperAPIDocument]

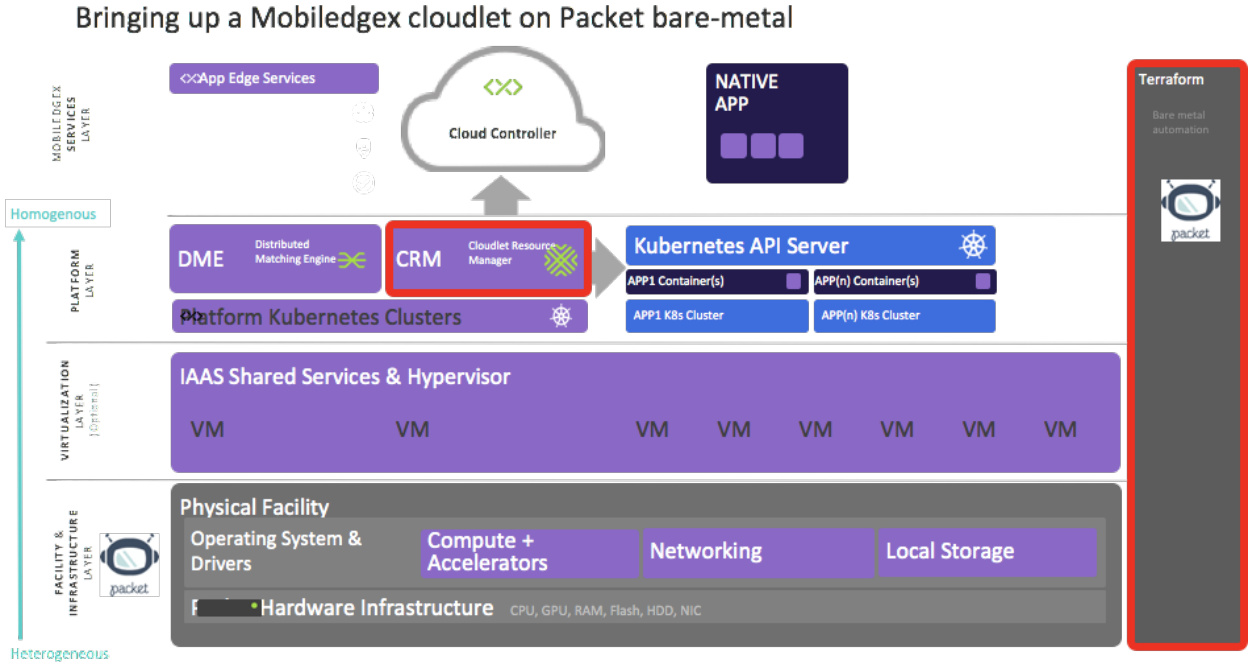
In case you are using a cloud provider and would like to spin application using our code you can use the following call flow to see how this works

Application Orchestration



14

We have automated the bare metal and virtualization layer open stack and kubernetes cluster with terraform script and edge stack looks as below. As you can see we have open sourced the basic CRM API set for bringing application over the above stack in 4 simple control commands.



Example commands set:

```
edgectl --addr mexdemo.ctrl.mobiledgex.net:55001 --tls $TLS/mex-client.crt
controller CreateApp --imagepath
docker.mobiledgex.net/mobiledgex/images/mobiledgexsdkdemo --imagetype
ImageTypeDocker --deployment kubernetes --key-developerkey-name MobiledgeX --
key-name "Example App" --key-version 1.0 --defaultflavor-name m1.medium --
accessports tcp:7777

edgectl --addr mexdemo.ctrl.mobiledgex.net:55001 --tls $TLS/mex-client.crt
controller CreateClusterInst --key-cloudletkey-operatorkey-name packet --key-
cloudletkey-name packet-sjc1 --key-clusterkey-name mexdemo-cluster --key-
developer MobiledgeX --flavor-name  m1.medium --nummasters 1 --numnodes 3

edgectl --addr mexdemo.ctrl.mobiledgex.net:55001 --tls $TLS/mex-client.crt
controller CreateAppInst --key-clusterinstkey-cloudletkey-name packet-sjc1 --
key-clusterinstkey-cloudletkey-operatorkey-name packet --key-clusterinstkey-
clusterkey-name mexdemo-cluster --key-clusterinstkey-developer MobiledgeX --
key-appkey-developerkey-name MobiledgeX --flavor-name x1.medium --key-appkey-
name "Example App" --key-appkey-version 1.0
```

For more information on Infrastructure you can refer the following links

›https://www.terraform.io/docs/providers/packet/index.html

›https://support.packet.com/kb/articles/terraform

›https://www.packet.com/developers/guides/mesos-dcos-with-terraform/

›https://docs.ansible.com/ansible/latest/scenario_guides/guide_packet.html

›https://support.packet.com/kb/articles/ansible