Deep Dive: WeOnPrem
Portable Edge Computing Solution

Overview

The document presents the whole end-to-end system called WeOnPrem, which includes both hardware and software solution. We utilized embedded devices(including raspberry pi and jetson nano) as a portable local data center in wework building, deployed an edged cloud-native solution named kubeedge to schedule containerized application, used zookeeper and kafka as middleware to support device management and event-driven applications.
Refer to some previous works(CV Edge Computing Infra & On Prem AI and deployed CV models), there still exists some pain points:

- CV models cannot deployed into an embedded solution because of the hardware constraint.
- Easy system deployment wasn't figured out clearly in previous works.
- A thoughtful cloud-edge synchronization infrastructure wasn't considered as high-priority.

To solve the pain points mentioned above, a lot of works have been done:

- At first, we refactored the whole system, and transferred the software into the embedded environment. It was important to highlight that a limited Yolov3 model developed by tensorRT & C++ could accelerate the model inference. An end-to-end system could also run and sample data in CNHQ.
- Then, we developed some statistical reports and a visualized panel to make our solution better.
- Recently, an easy-to-deploy solution was designed and developed to make sure we can deploy our system as quickly as possible and manage software deployment & update well.

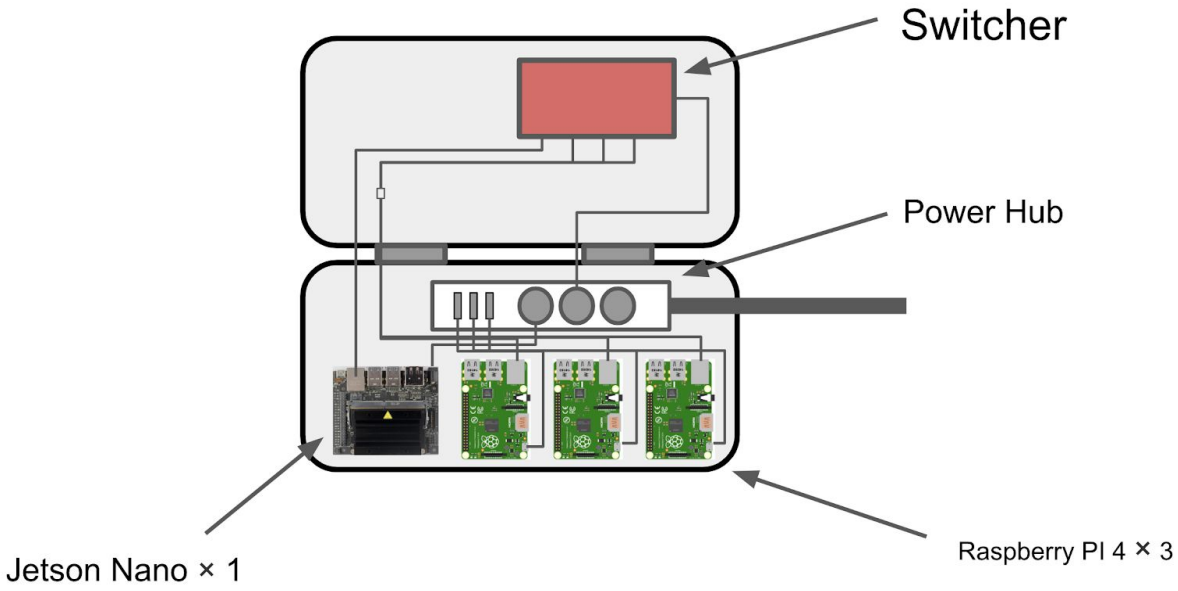
Hardware

As we all know, it is important to deploy our edge system as quickly as possible. So that requires us less configurable ports, stable networks and power.
In our solution, the WeOnPrem hardware is regarded as a black box. It can support the capability of model inference and logical computing. So this solution plays a role of local data center.

The design principles are:

- An isolated network specifically for on-prem processing devices (sensor devices use. IOT network as well as bluetooth). The motivation is not only about security but also avoiding unpredictable turbulence of public WIFI.

- An API gateway and network gateway on a same device, serving sensor & cloud connections.



The physical goods are shown in the following pictures:



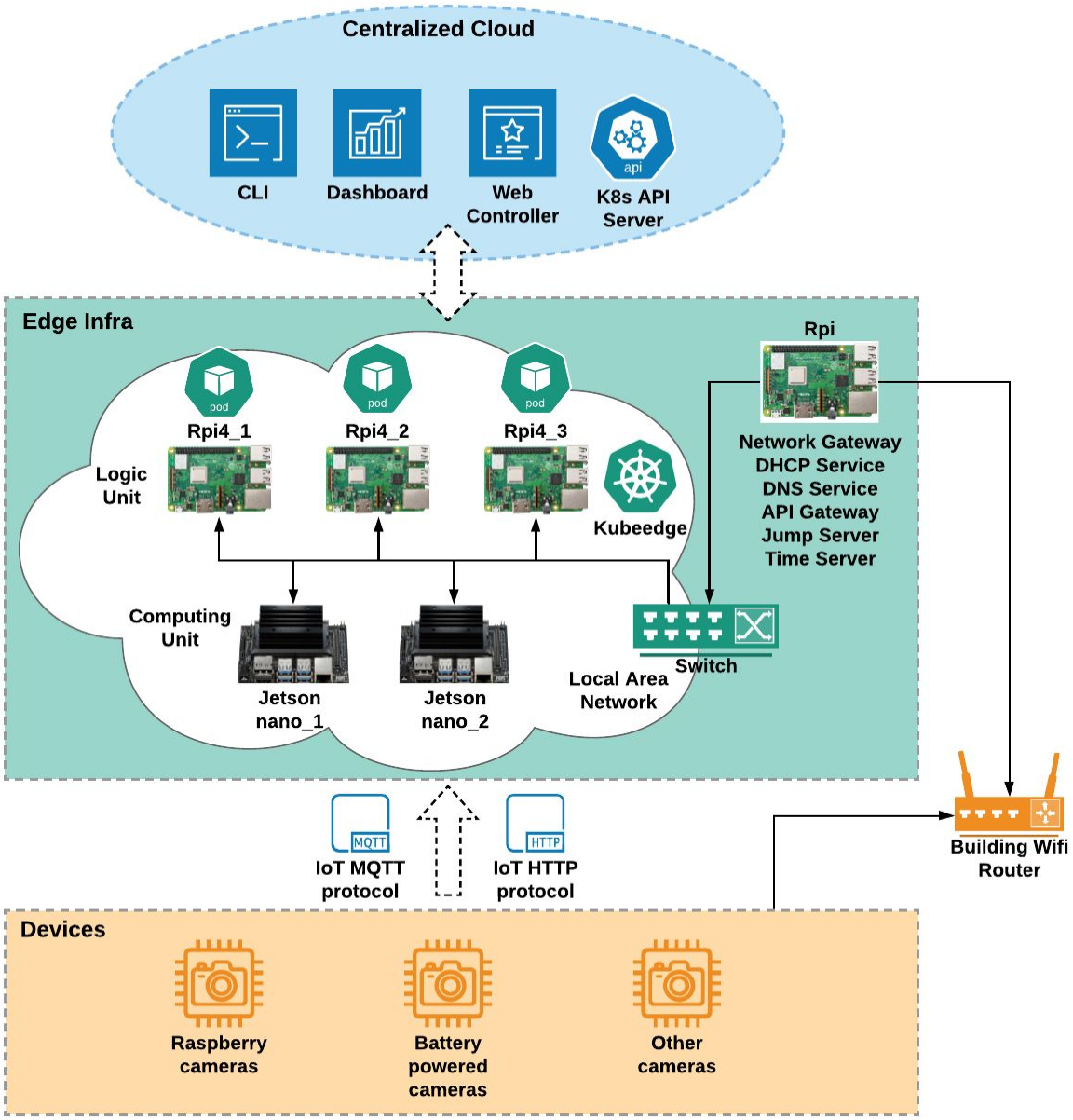
Infra deployment

The whole infrastructure contains three parts: cloud side, edge side and device side.

- The cloud side as a centralized control center will monitor every edge side's status in each building.
- The edge side as a local data center will support model computing capability and device management.
- The device side will do data sampling.

So the edge part plays a key role in this solution, which supplies the logical and computational functions for specific application scenario. The edge side will monitor devices' status and conduct device management as well. What's more, it is as a bridge to joint devices and control panel in centralized cloud.

It is illustrated in the plot below according to our design:



Network Deployment

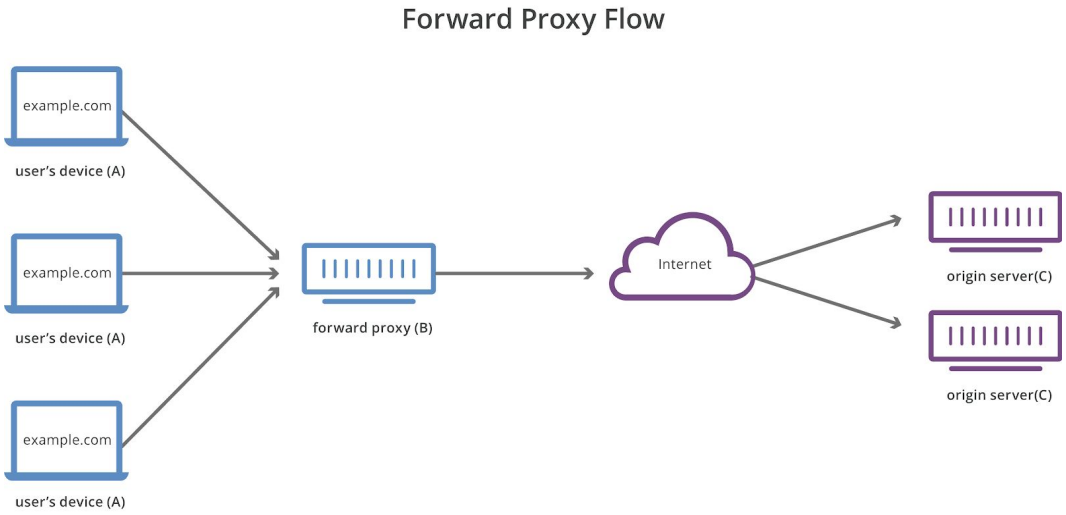
DHCP & DNS Services

DHCP and dns service are aimed to give every device an alias name for better memorizing. Dnsmasq provides network infrastructure for small networks: DNS, DHCP, router advertisement and network boot. It is designed to be lightweight and have a small footprint, suitable for resource constrained routers and firewalls.

After setup, we can modify the configuration file of dnsmasq service. Then enable it as a daemon process and edit `/etc/hostname` file and define the devices' domain name. As a result, we can route the device nodes by domain names.

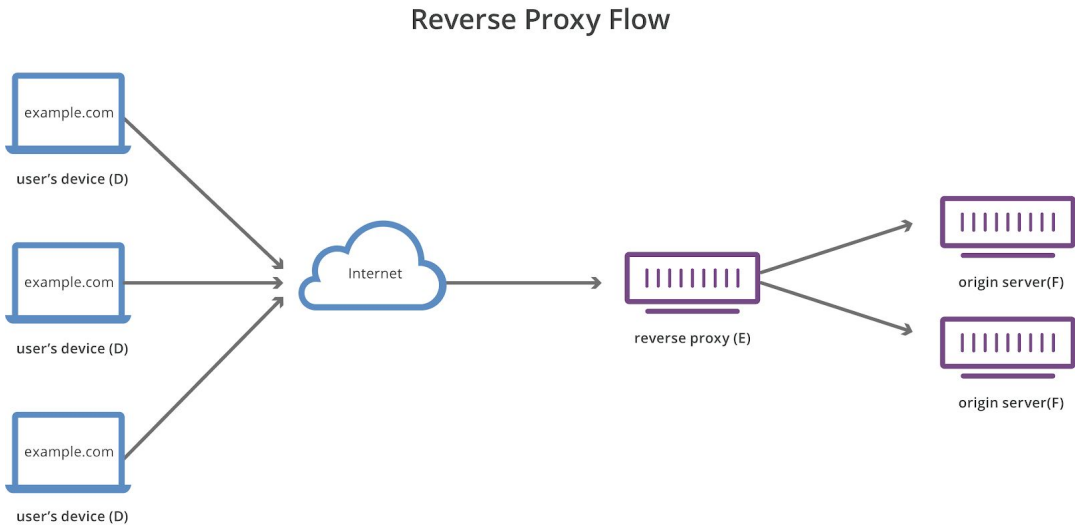
Forward Proxy

As to enable the edge nodes have access to outer internet, we deploy a forward proxy in gateway node.
After setup of forward proxy, edge nodes in the inner local network can connect with the cloud side and pull images from docker hub.



Reverse Proxy

To allow sensors sending data to application server in the inner network, we deployed a nginx as reverse proxy to expose a http port and transfer requests to the destination node.



Native Containerized Environment

To accelerate software deployment, our system need a cloud-native solution to manage our application, schedule applications by configuration, and recover from breakdown.

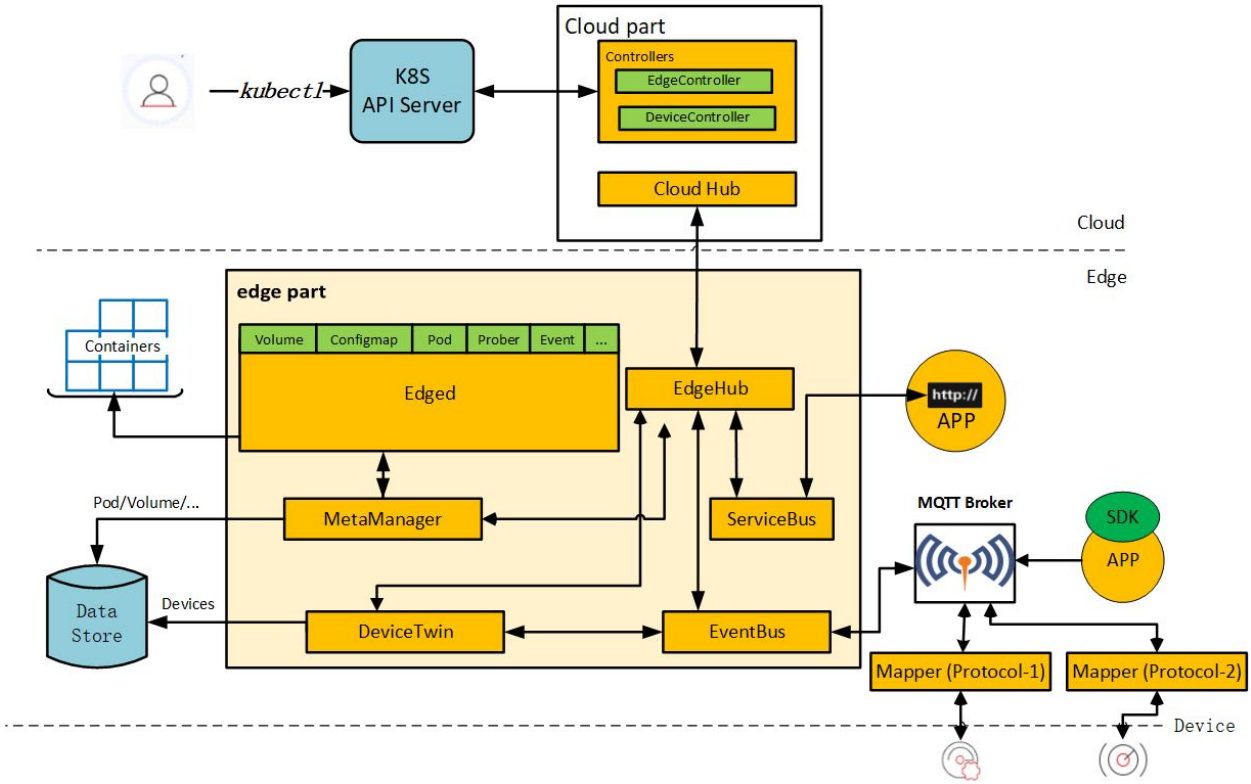
Scheme Selection

There are some open source solutions for edge computing. In the cloud solution, Kubernetes has become a standard for scaling up IT infrastructure to achieve cloud-native capabilities. So we mainly compare two solutions: k3s and kubeedge.
As a result, we chose kubeedge. Reasons are listed below:

1. K3s is wrapped in a simple packagethe reduces the dependecies and steps needed to run as production Kubernetes cluster. But Kubeedge is a extension for Kubernetes and enables optimization in edge resource utilization.
2. K3s doesn't contain cloud part. It is a lightweight version. Kubeedge contains a control plane resides in the cloud. So the cloud part can manage the edge nodes.
3. Kubeedge is born for edge computing scenarios, which is also supports device management and meta-data management.

Kubeedge

KubeEdge is an open source system extending native containerized application orchestration and device management to hosts at the Edge. It is built upon Kubernetes and provides core infrastructure support for networking, application deployment and metadata synchronization between cloud and edge. It also supports MQTT and allows developers to author custom logic and enable resource constrained device communication at the Edge. Kubeedge consists of a cloud part and an edge part. Both edge and cloud parts are now opensourced.
Kubeedge arhitecture can be seen below and more details can be referred in official documents.



Constraints & Risks

There are some risks in kubeedge:

- Kubeedge has been opensourced for a short time. So lots of features need to be added in the future.
- Micro services in kubeedge is not supported very well. So we still use some third-party solutions for service discovery.
- The software community is small and inactive. So the bug fixing is slow.

Deploy Resources

After setup[refer to this document], we can make device nodes join in the cluster.
Type in command `kubectl get nodes` to inspect the nodes' status:

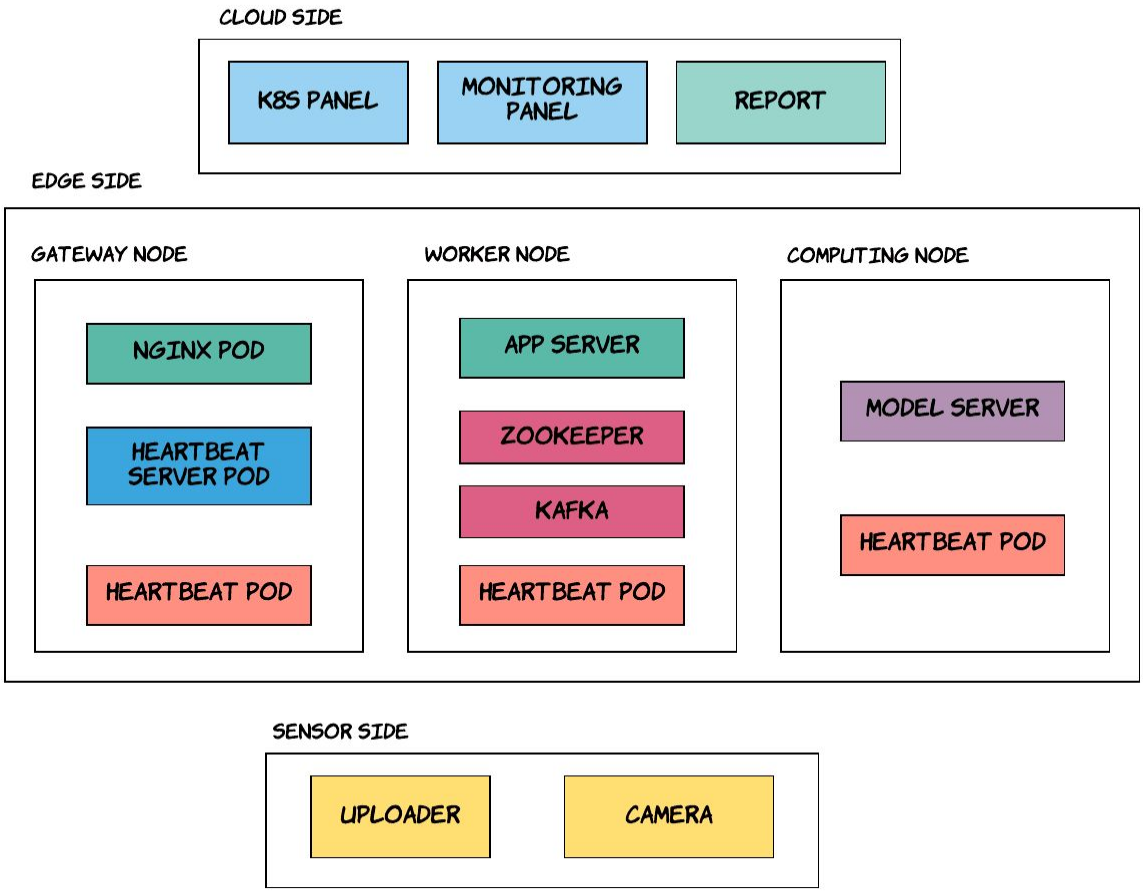
```
NAME             STATUS   ROLES    AGE    VERSION
comp-node0       Ready    edge     5h2m   v1.15.3-kubeedge-v0.0.0-master+$Format:%h$
docker-desktop   Ready    master   5h6m   v1.14.3
gateway-node     Ready    edge     5h2m   v1.15.3-kubeedge-v0.0.0-master+$Format:%h$
worker-node0     Ready    edge     5h2m   v1.15.3-kubeedge-v0.0.0-master+$Format:%h$
worker-node1     Ready    edge     5h2m   v1.15.3-kubeedge-v0.0.0-master+$Format:%h$
```

Describe the schema configuration you wanna orchestrate your resources(including node, pods, daemonset, deployment, etc.), and you can manage your applications similar to interact with k8s.
Type in command `kubectl get pods` to inspect the pods' status:

```
NAME                                READY   STATUS    RESTARTS   AGE
app-server-665cf4c6fd-zf4ws         1/1     Running   2          4h53m
heartbeat-jn-daemon-4pmh6           1/1     Running   1          4h54m
heartbeat-pi-daemon-6nbn2           1/1     Running   0          4h54m
heartbeat-pi-daemon-7tsb8           1/1     Running   0          4h54m
heartbeat-pi-daemon-jgpf5           1/1     Running   0          4h54m
heartbeat-server-6b7b6df84d-88rjc   1/1     Running   0          4h53m
kafka-deployment-5467fd9d78-296zp   1/1     Running   0          4h54m
zk-deployment-6c6d44d55d-kdwwz      1/1     Running   0          4h55m
```

Software

Architecture



In this section, we mainly talk about some key points in the edge side:

- Every node deployed heartbeat pod to monitor the physical status in devices and send heartbeat to zookeeper.
- In the gateway node, heartbeat server would receive the heartbeat http requests from sensor to register heartbeat timestamp into zookeeper. This node also deployed a nginx

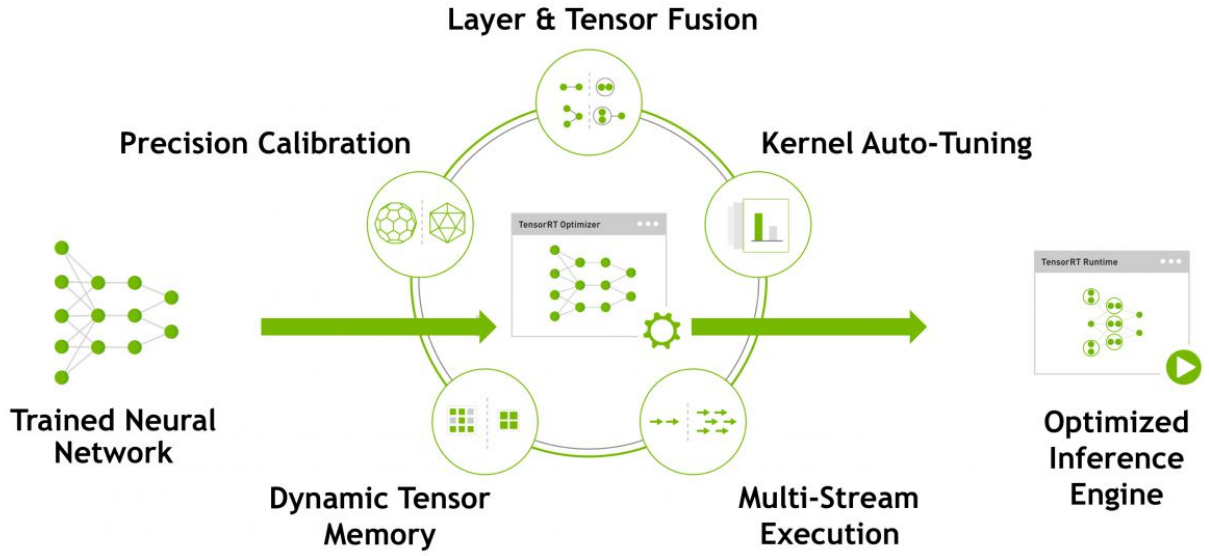
server as a proxy to transfer picture uploading post requests to the app server deployed in worker node.

- Worker nodes deployed middleware like zookeeper and kafka, application to implement the logical workflow.
- In the computing node, a model server is used to subscribe kafka topic and do model inference.

CV models

Inference Framework

Deploy trained deep learning models for inference on embedding devices are quite challenging. Computational capacity is limited while fulfilling the requirements of real-time applications on QPS and latency.
Firstly, we choose TensorRT as our inference framework, since it is the official release designed for jetson nao. NVIDIA TensorRT™ is a platform for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications. This framework conduct a neural network optimization procedure to trained deep model before for acceleration and it optimizes the runtime efficiency on its hardware-related fundamental. Applications were observed to get a raise about 100% on speed performance.



Secondly, the performance of code running on CPU is also taken into account. A roughly implemented pipeline will also cause high latency while idling GPU device. We wrote the whole

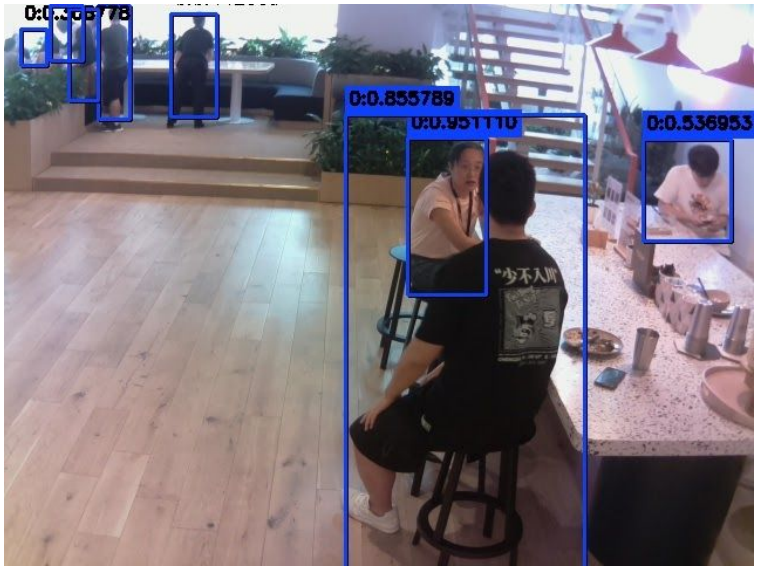
pipeline with C++ and parallelize the GPU processing and CPU processing with libml, accelerating it by 30%.

Model Selection

Currently, where are 3 applications in out project ---- people counting, messy detection and sink cleanness detection. For the first and third application, we formulate the problem into object detection tasks. Then we choose YOLOv3, a famous object localization framework with both high accuracy and efficiency. This model has fewer parameters such that it can be successfully deployed on edge devices. By rounding of tuning, we successfully deployed the full version model on jetson nano, which produces highest accuracy on people detection. For messy detection, we do not detect all individual objects in the image, instead, we directly predict the messy probability for the room. We chose an InceptionV3 model to implement this.
The application of sink cleanness prediction can also be regarded as a messy detection task, however, we have further considerations on it such as that the objects in the sink may have different weights for cleanness.

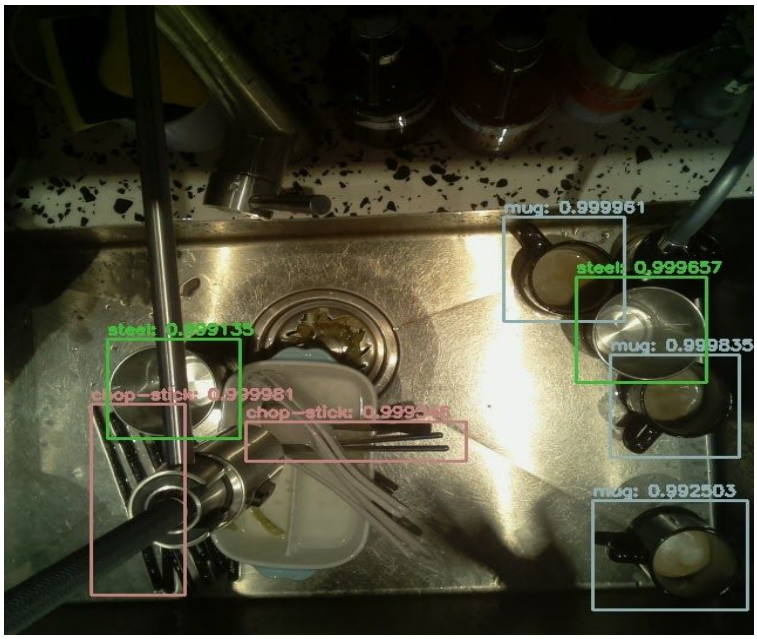
Current Application Scenarios

People Counting



Messy Detection

Sink cleanness

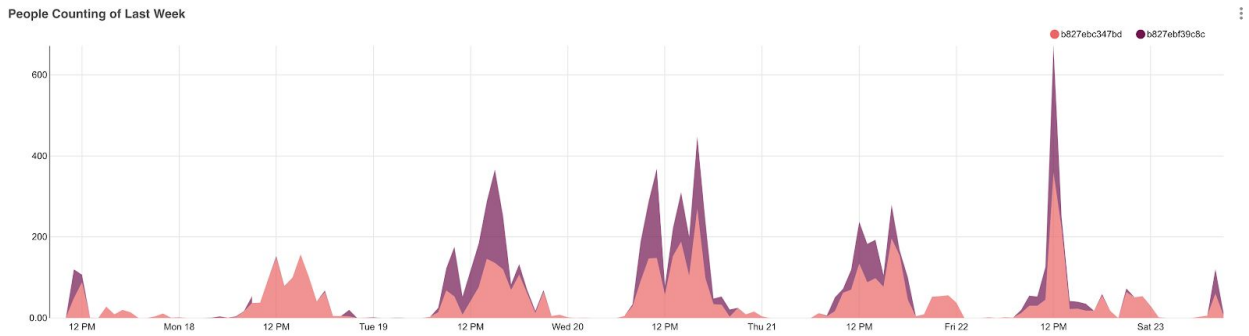


Visualization

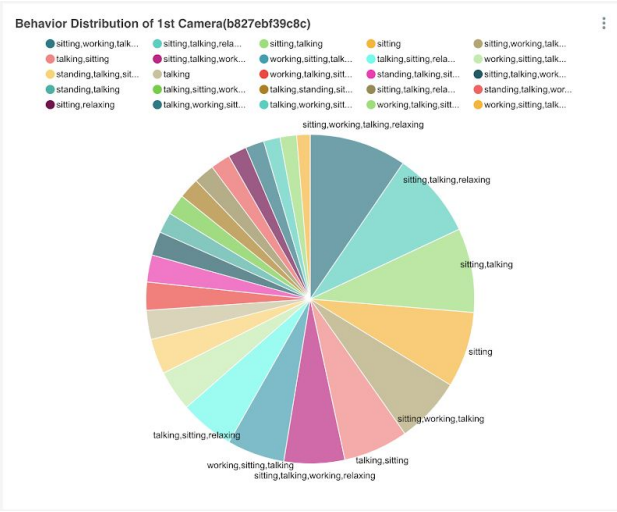
Superset statistical report

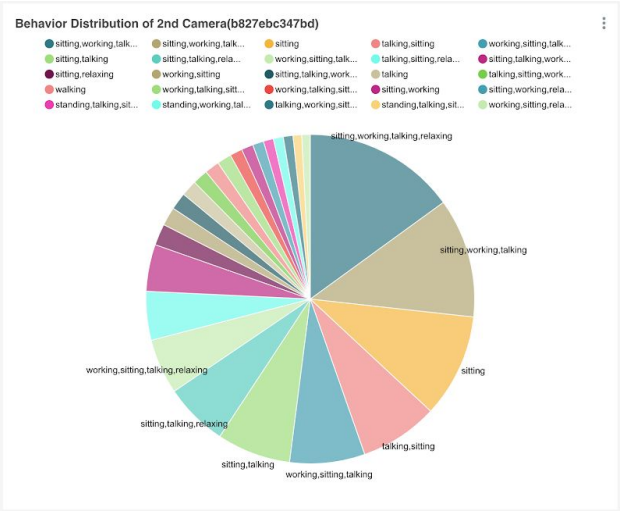
A statistical report demo based on Superset is shown below, which contains some statistical graph including engagement comparison in time series plot, behavior distribution and area distribution in pie chart.

Time Series People Counting Plot

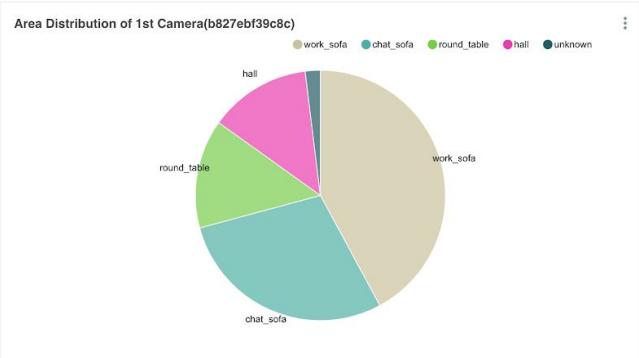


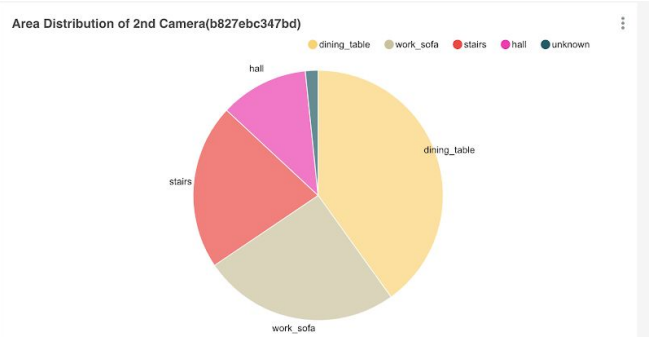
Behavior Distribution





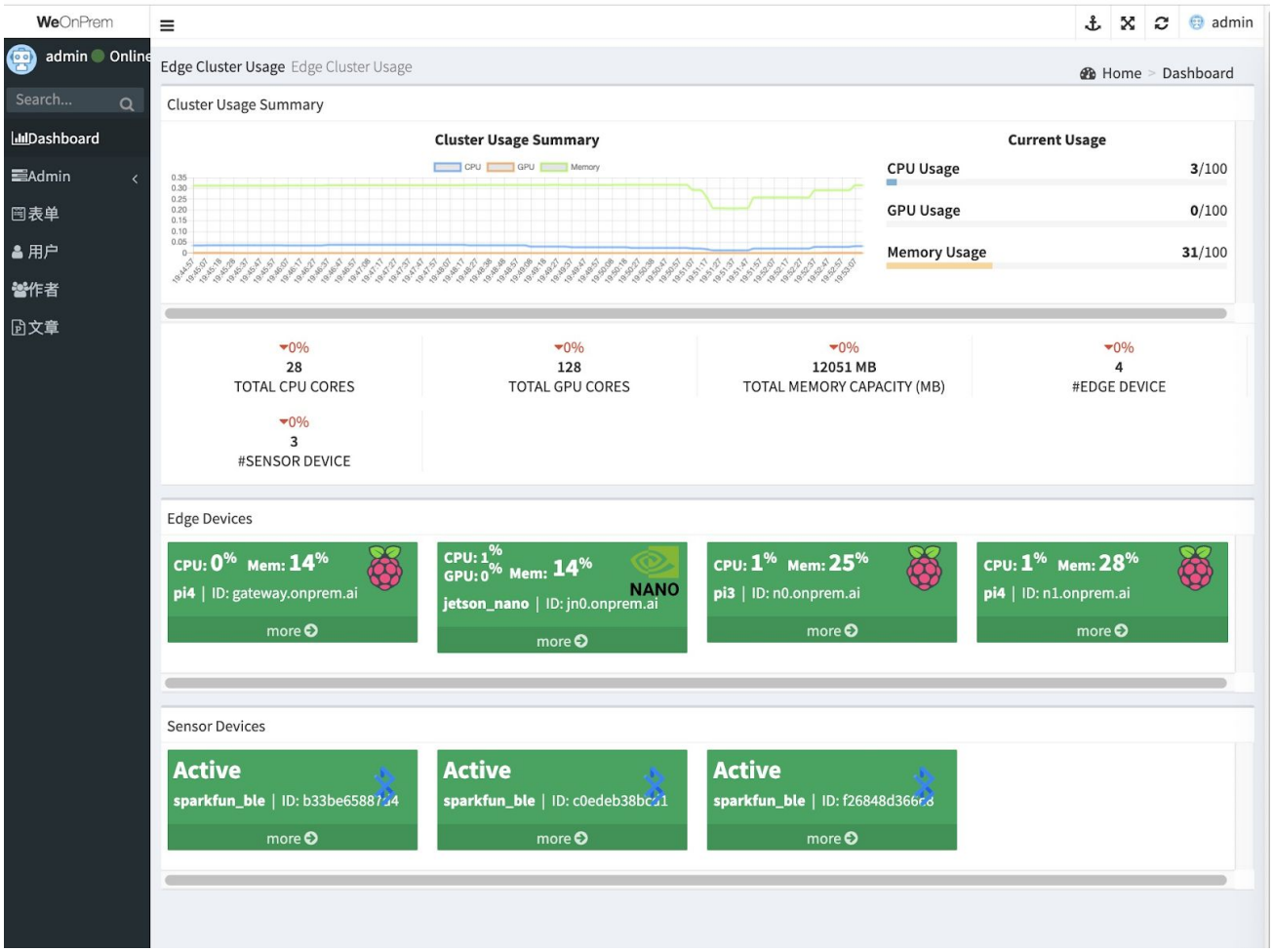
Engagement Area Distribution





Device Management Visualized Panel

To better monitor device lifecycle and status, we developed a panel web page to visualize device management. As a demo, devices' status(including cpu * memory) will be sent to zookeeper. Whether the devices are active or not can also be shown in this panel.



What's the next

About scenarios

- On premise deployment of sink cleanness detection for WeWork Japan.
- Some other add-ons to sink cleanness detection.(URL)
- Keep on working with China Tech to explore people counting on event evaluation, design evaluation, etc.
- If we can change the elevators.
- Working with US research team to check out more scenarios globally.

About hardwares

- Keep on working with demand site, collecting more design requirements for sensors and edge cluster. Improve the structure design of on-prem devices.
- Dig into pure offline cluster mode, developing device plugins such as local timing service, emergency power supply(UPS), etc.
- Study more on low-power-cost hardware solution.
- Cutting costs.

About edge clusters

- Currently, our deployment based kubeedge relies badly on cloud-site service. Once the service comes to some problems. The edge-site will be affected. Thus self-cluster-management on edge-site shall be developed.
- Edge cluster may face lots of on-prem situation which cannot instantly fixed by human-beings, thus, self cluster recovery mechanism is required.
- An automatic device initialization pipeline. All initial parameters can be set while flashing the system SD card. It is useful when manufacturing our solution package by batch.
- Make it more compatible to all kinds of AI solutions. Turning WeOnPrem into a general AI computational unit.

About models

- Efficiency improving: 1) Exploring uint8 mode for tensorRT. 2) Trying other network size compression algorithms to YoloV3. 3) Tuning people counting / sink cleanness model on small network-structure such as Yolo-Tiny and Yolo-Nano.
- Model accuracy: 1) Better preprocessing (debluring, illumination balancing, etc) 2) Data augmentation.
- Optimizing Activity Detection: Trying it on shot video inputs.