KubeEdge-AI Architecture
Service Architecture

- **GlobalCoordinator @ Cloud:**
  1. unportal of EdgeAI,
  2. across-edges coordination

- **LocalController @ Edge:**
  1. local controller
  2. manage local dataset and models

- **Workers:**
  1. do inference or training, based on existing ML framework;
  2. launch on demand, imagine they are docker containers;
  3. different workers for different features;
  4. could run on edge or cloud.

- **Lib:**
  1. expose the Edge AI features to applications, i.e. training or inferencing programs.

---

**EdgeAI Workers:**
- Applications <-> AI features
  - Facial Recognition, ASR, etc
- Training Programs:
  - Build-in Training Algo, 3rd-party Training Algo
- Dataset Mgmt/Labeling
- Peers Mgmt/Status Sync.
- Batch Job Mgmt
- Resource Mgmt
- Drivers for cloud service backends

---

**LocalController@Edge**
- Incremental learning
- Joint inference
- Collaborative Training
- Dataset Mgmt/Labeling
- Model Mgmt
- Peers Mgmt/Status Sync.
- Batch Job Mgmt
- Resource Mgmt
- Drivers for cloud service backends

---

**KubeEdge@Edge**
- Arm Server (Atlas), x86 Server, smart camera (Hi Lane), etc

---

**GlobalCoordinator@Cloud**
- Cross-Edges Collaborative Training
- 3rd-Party Training Algo Mgmt
- Incremental learning
- Joint-Models Building
- Dataset Mgmt/Labeling
- Model Mgmt
- Drivers for cloud service backends

---

**EdgeAI Workers:**
- Inferencing Backend @ Cloud
- Inferencing API
- MindSpore / TensorFlow / Pytorch
- EdgeAI Lib
- ML Models

---

**EdgeAI Workers:**
- Cloud Training/Aggregation
- Training Programs:
  - Build-in Training Algo, 3rd-party Training Algo
- MindSpore / TensorFlow / Pytorch
- EdgeAI Lib

---

**KubeEdge@Cloud**
- IaaS/PaaS @ Cloud, e.g. CCE/ECS

---

**Arm Server (Atlas), x86 Server**
- ML Services @ Cloud, e.g. ModelArts
Edge AI Architecture based KubeEdge

- Cloud Core
- Edge Core
- Device

Cloud Core Components:
- GC (Global Coordinator)
- EdgeController
- DeviceController
- SyncController
- CloudHub

Edge Core Components:
- WebSocket (Default)
- QUIC (Alternate)

Device Components:
- Pod 1
- Pod 2
- Pod 3

EdgeMesh Components:
- LC (Local Controller)
- Containerd
- Cri-o
- Docker

Devices:
- Device 1
- Device 2
- Device 3
- Device 4
- ...
Due to current implementation of kubeedge, GC will be as a module in cloudcore of kubeedge. Another choice: standalone controllers out of cloudcore for decoupling.
Architecture@edge

Due to current implementation of kubeedge, we use mqtt in LC to interact with GC. Better choice: edgemesh?
Edge AI Feature: Federated training

```yaml
apiVersion: edgeai.io/v1alpha1
kind: FederatedTask
metadata:
  name: example-federatedtask
spec:
  aggregationWorker:
    name: aggregation-worker
    model:
      name: resnet50
    workerSpec:
      aggregationAlgorithm: FedAvg
      parameters:
      - key: batch_size
      - key: learning_rate
      - key: min_node_number
      - key: rounds_between_validations
    trainingWorkers:
    - name: work0
      nodeName: edge0
    workerSpec:
      dataset:
        name: dataset0
        trainScriptDir: /code/
        trainScriptBootFile: /code/main.py
        frameworkType: tensorflow
        frameworkVersion: 1.12
        parameters:
        - key: batch_size
          value: 32
        - key: learning_rate
          value: 0.001
        - name: work1
        - name: work2
```

- Federated training CRD
- Aggregation worker learning CRD
- Edge training workers
0. User creates a federated-task crd
1. GC watches this crd, syncs to LCs.
2. GC creates the aggregation worker, and publishes it as a service.
3. GC creates these edge training workers
4. GC waits these workers’ to complete (not showed in diagram)
Thank You