KubeEdge BP Source Code Analysis

Yin Ding

[KubeEdge](https://kubeedge.io/en/) is an open source framework for edge computing built on Kubernetes that is an incubation-level [CNCF](https://www.cncf.io/projects/kubeedge/) project. KubeEdge helps developers deploy and manage containerized applications in the cloud and on the edge using the same unified platform. KubeEdge handles networking, deployment and data synchronization between edge and cloud infrastructure.

blocked URL

KubeEdge cloud components

KubeEdge cloud components all fall under what is called “CloudCore” which handles communication between the Kubernetes cluster via API and then communicates with the edge devices.

- **CloudHub** – Works by establishing a websocket connection with EdgeHub on edge devices and passes changes from the cloud to the edge
- **EdgeController** – Handles metadata for nodes and pods on the edge and allows data from cloud to be sent to specific edge nodes
- **DeviceController** – Similar to EdgeController and handles metadata for specific devices so data can be synced between edge and cloud

KubeEdge edge components

KubeEdge edge components fall under “EdgeCore” and handle communication between application containers, devices, and the cloud.

- **EdgeHub** – Connects to cloud via websocket and is responsible for passing data from devices back to the cloud and cloud data to devices
- **Edged** – The agent that runs on edge nodes and what manages the actual containers and pods running on edge devices
- **MetaManager** – MetaManager handles message processing between Edged and EdgeHub. MetaManager also provides persistence and querying of metadata via SQLite
- **EventBus** – MQTT client that allows edge devices to interact with MQTT servers and gives KubeEdge pub/sub capabilities
- **ServiceBus** – HTTP client that allows edge devices to interact with other services over HTTP
- **DeviceTwin** – Stores device status and syncs device status with cloud. DeviceTwin also provides the ability to query devices connected to KubeEdge
- **Mappers** – KubeEdge Mappers allow edge nodes to communicate over common IoT protocols like Modbus, OPC-UA, and Bluetooth.

Code Architecture Overview

KubeEdge has many modules, users can choose to turn on or off some modules according to their needs. These modules are managed through beehive. Beehive is the core message communication framework in KubeEdge, which is used for registration of different modules and communication between modules. Both CloudCore and EdgeCore components in KubeEdge depend on the beehive framework.

The module definition is an interface. As long as this interface is implemented, it can become a module. Common modules in KubeEdge, such as cloudhub, edgehub, edgeController, etc., have already implemented this interface.

```
// Module interface
type Module interface {
   Name() string
   Group() string
   Start()
   Enable() bool
}
```

The Register function of each module calls the Register function of beehive to register the module in beehive. According to whether the module is enabled (modules.enabled) in the configuration file, beehive adds the module to the internal modules map or disabledModules map for management.

```
        cloudhub.Register(c.Modules.CloudHub)
        edgecontroller.Register(c.Modules.EdgeController)
        devicecontroller.Register(c.Modules.DeviceController)
        nodeupgradejobcontroller.Register(c.Modules.NodeUpgradeJobController)
        synccontroller.Register(c.Modules.SyncController)
        cloudstream.Register(c.Modules.CloudStream, c.CommonConfig)
        router.Register(c.Modules.Router)
        dynamiccontroller.Register(c.Modules.DynamicController)
```

Beehive uses the golang channel to realize inter-module communication. The communication methods include "unicast" and "multicast", that is, the message can be sent to a certain module alone, or the message can be sent to the module group (that is, the edged, hub, bus and other groups mentioned above). Beehive uses context to manage communication between groups and modules. When using channel as the communication method, ChannelContext implements two interfaces related to context: ModuleContext and MessageContext.

```
// ModuleContext is interface for context module management
type ModuleContext interface {
        AddModule(info *common.ModuleInfo)
        AddModuleGroup(module, group string)
        Cleanup(module string)
}
```

```
// MessageContext is interface for message syncing
type MessageContext interface {
        // async mode
        Send(module string, message model.Message)
        Receive(module string) (model.Message, error)
        // sync mode
        SendSync(module string, message model.Message, timeout time.Duration) (model.Message, error)
        SendResp(message model.Message)
        // group broadcast
        SendToGroup(group string, message model.Message)
        SendToGroupSync(group string, message model.Message, timeout time.Duration) error
}
```