BAETYL Documentation

BAETYL

Sep 24, 2019

Contents

Baetyl, extend cloud computing, data and service seamlessly to edge devices.

CHAPTER 1

What is Baetyl

Baetyl is an open edge computing framework of Linux Foundation Edge that extends cloud computing, data and service seamlessly to edge devices. It can provide temporary offline, low-latency computing services, and include device connect, message routing, remote synchronization, function computing, video access pre-processing, AI inference, device resources report etc. The combination of Baetyl and the **Cloud Management Suite** of BIE(Baidu IntelliEdge) will achieve cloud management and application distribution, enable applications running on edge devices and meet all kinds of edge computing scenario.

About architecture design, Baetyl takes **modularization** and **containerization** design mode. Based on the modular design pattern, Baetyl splits the product to multiple modules, and make sure each one of them is a separate, independent module. In general, Baetyl can fully meet the conscientious needs of users to deploy on demand. Besides, Baetyl also takes containerization design mode to build images. Due to the cross-platform characteristics of docker to ensure the running environment of each operating system is consistent. In addition, **Baetyl also isolates and limits the resources of containers**, and allocates the CPU, memory and other resources of each running instance accurately to improve the efficiency of resource utilization.

1.1 Advantages

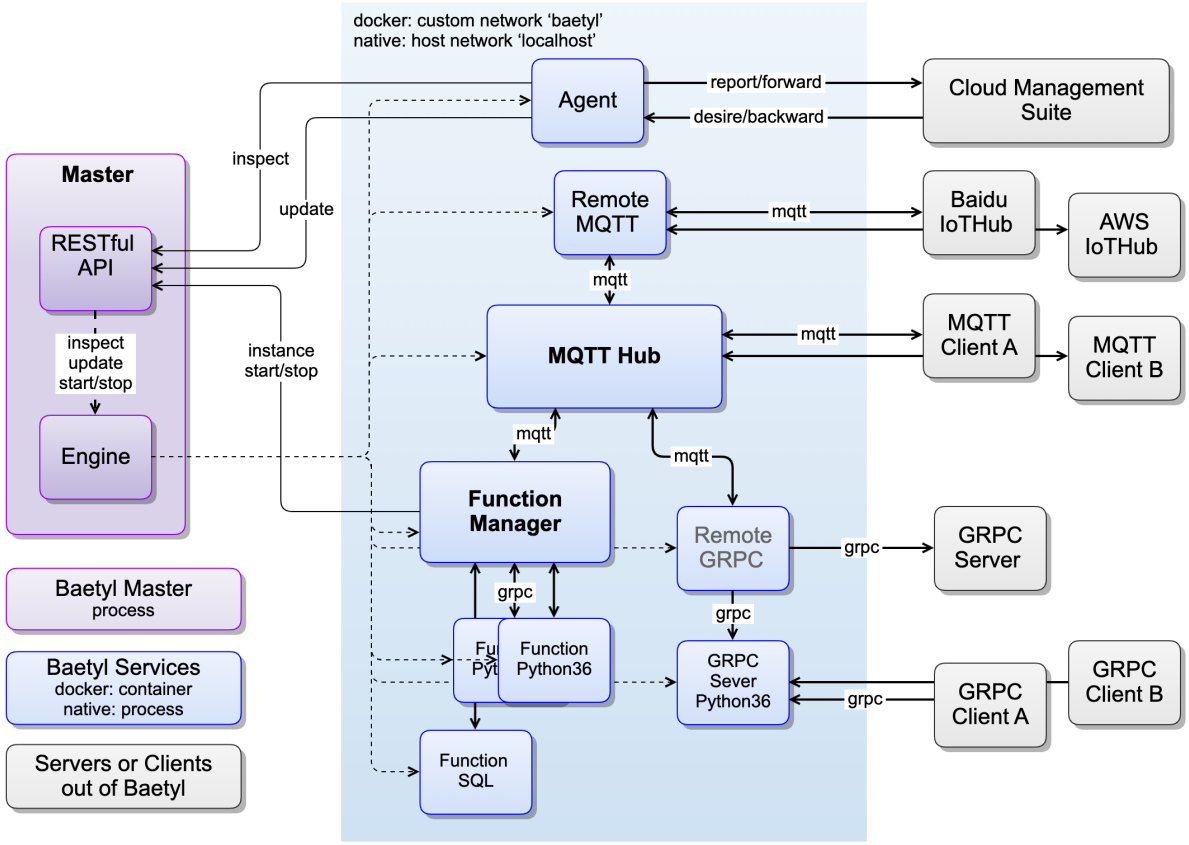
- **Shielding Computing Framework**: Baetyl provides two official computing modules(**Local Function Module** and **Python Runtime Module**), also supports customize module(which can be written in any programming language or any machine learning framework).
- **Simplify Application Production**: Baetyl combines with **Cloud Management Suite** of BIE and many other productions of Baidu Cloud(such as CFC, Infinite, EasyEdge, TSDB, IoT Visualization) to provide data calculation, storage, visible display, model training and many more abilities.
- **Service Deployment on Demand**: Baetyl adopts containerization and modularization design, and each module runs independently and isolated. Developers can choose modules to deploy based on their own needs.
- **Support multiple platforms**: Baetyl supports multiple hardware and software platforms, such as X86 and ARM CPU, Linux and Darwin operating systems.

1.2 Components

As an edge computing platform, **Baetyl** not only provides features such as underlying service management, but also provides some basic functional modules, as follows:

- Baetyl Master is responsible for the management of service instances, such as start, stop, supervise, etc., consisting of Engine, API, Command Line. And supports two modes of running service: **native** process mode and **docker** container mode
- The official module baetyl-agent is responsible for communication with the BIE cloud management suite, which can be used for application delivery, device information reporting, etc. Mandatory certificate authentication to ensure transmission security;
- The official module baetyl-hub provides message subscription and publishing functions based on the MQTT protocol, and supports four access methods: TCP, SSL, WS, and WSS;
- The official module baetyl-remote-mqtt is used to bridge two MQTT Servers for message synchronization and supports configuration of multiple message route rules. ;
- The official module baetyl-function-manager provides computing power based on MQTT message mechanism, flexible, high availability, good scalability, and fast response;
- The official module baetyl-function-python27 provides the Python2.7 function runtime, which can be dynamically started by `baetyl-function-manager`;
- The official module baetyl-function-python36 provides the Python3.6 function runtime, which can be dynamically started by `baetyl-function-manager`;
- The official module baetyl-function-node85 provides the Node 8.5 function runtime, which can be dynamically started by `baetyl-function-manager`;
- SDK (Golang) can be used to develop custom modules.

1.2.1 Architecture



Architecture

1.3 Installation

- *Quick Install Baetyl*
- *Build Baetyl From Source*

1.4 Development

- Baetyl design
- *Baetyl config interpretation*
- *How to write Python script for Python runtime*
- *How to write Node script for Node runtime*
- *How to import third-party libraries for Python runtime*
- *How to import third-party libraries for Node runtime*
- *How to develop a customize runtime for function*
- *How to develop a customize module for Baetyl*

1.5 Contributing

If you are passionate about contributing to open source community, Baetyl will provide you with both code contributions and document contributions. More details, please see: How to contribute code or document to Baetyl.

1.6 Contact us

As the first open edge computing framework in China, Baetyl aims to create a lightweight, secure, reliable and scalable edge computing community that will create a good ecological environment. In order to create a better development of Baetyl, if you have better advice about Baetyl, please contact us:

- Welcome to join Baetyl's Wechat
- Welcome to join Baetyl's LF Edge Community
- Welcome to send email to baetyl@lists.lfedge.org
- Welcome to submit an issue

CHAPTER 2

Baetyl Design

2.1 Concepts

- **System**: Refers to the Baetyl system, including **Master**, **Service**, **Volume** and system resources used.
- **Master**: Refers to the core part of the Baetyl, responsible for managing **Volume** and **Service**, built-in **Engine**, external RESTful API and command line.
- **Module**: Provides an executable package for **Service**, such as a docker image, to launch instances of **Service**.
- **Service**: Refers to a set of running programs that managed by Baetyl to provide specific functions such as message routing services, function computing services, micro-services, etc.
- **Instance**: Refers to the specific running program or container launched by the **Service**, a **Service** can start multiple instances, or can be dynamically started by other services. For example, the instances of function runtime service are dynamically started and stopped by the function manager service.
- **Volume**: Refers to the directory used by the **Service**, can be a read-only directory, such as a directory for placing resources such as configuration, certificates, scripts, etc., or a writable directory to persist data, such as logs and database.
- **Engine**: Refers to the operational abstractions and concrete implementations of the various running modes of the **Service**, such as the docker container mode and the native process mode.
- **Services** and **System** Relationships: Baetyl systems can start multiple services, there is no dependency between services, and their startup order should not be assumed (although it is currently started sequentially). All data generated by the service at runtime is temporary and will be deleted when the service is stopped, unless it is mapped to a persistent directory. The program in the service may stop for various reasons, and the service will restart the program according to the user's configuration. This situation is not equal to the stop of the service, so the temporary data will not be deleted.

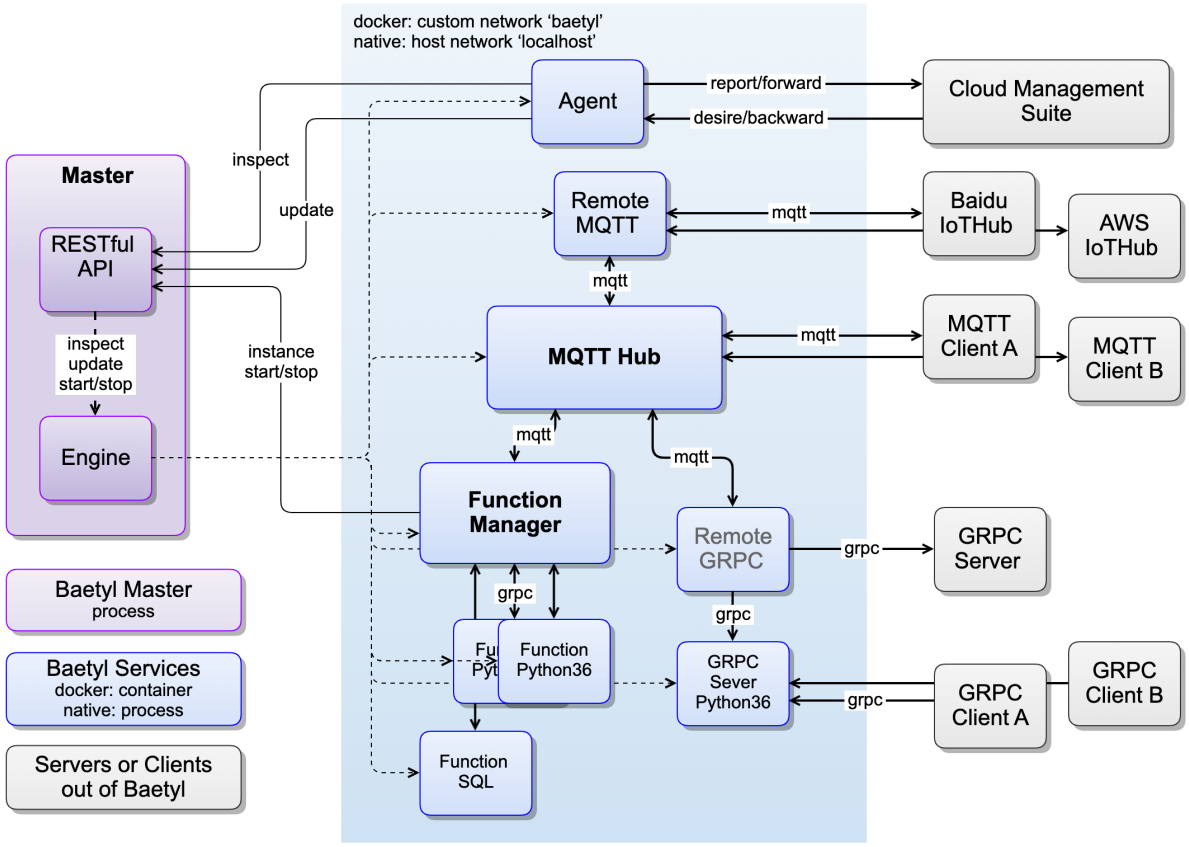
2.2 Components

A complete Baetyl system consists of **Master**, **Service**, **Volume** and system resources used. The Master loads all modules according to the application configuration to start the corresponding services, and a service can start several instances, all of which are managed and supervised by Master. NOTE that the instances of the same service shares the storage volume bound to the service. Therefore, if an exclusive resource exists, such as listening to the same port, only one instance can be successfully started.

At present, Baetyl has the following official modules:

- *baetyl-agent*: Provides cloud agent service for status reporting and application OTA.
- *baetyl-hub*: Provides an MQTT-based message routing service.
- *baetyl-remote-mqtt*: Provides a bridge services for synchronizing messages between Hub and remote MQTT services.
- *baetyl-function-manager*: Provides function services for function instance management and message-triggered function calls.
- *baetyl-function-python27*: Provides a GRPC micro-service that loads Python scripts based on Python2.7 runtime that can be managed by baetyl-function-manager as a function instance provider.
- *baetyl-function-python36*: Provides a GRPC micro-service that loads Python scripts based on Python3.6 runtime that can be managed by baetyl-function-manager as a function instance provider.
- *baetyl-function-node85*: Provides a GRPC micro-service that loads javascripts based on Node8.5 runtime that can be managed by baetyl-function-manager as a function instance provider.

Structure Diagram:



Structure Diagram

2.3 Master

Master as the core of the Baetyl system, it manages all storage volumes and services, has a built-in runtime engine system, and provides RESTful APIs and command lines.

The start and stop process of the Master is as follows:

1. Execute the startup command: `sudo systemctl start baetyl` to start Baetyl in Docker mode and then execute the command `sudo systemctl status baetyl` to check whether baetyl is running. In darwin, excute `sudo baetyl start` to run the Baetyl in the foreground of the terminal.
2. The Master will first load etc/baetyl/conf.yml in the working directory, initialize the running mode, API server, log and exit timeout, etc. These configurations can not be changed during application OTA. If no error is reported, the baetyl.sock (only on Linux) file is generated in the var/run/ directory.
3. The Master will then attempt to load the application configuration var/db/baetyl/application.yml and will not start any service if the configuration does not exist, otherwise the list of services and storage volumes in the application configuration will be loaded. This file will be updated during application OTA, and the system will update the services according to the new configuration.
4. Before starting all services, the Master will first call the Engine interface to perform some preparatory work. For example, in container mode, it will try to download the image of all services first.
5. After the preparation is completed, start all services in sequence, and if the service fails to start, the Master will exit. In the container mode, the storage volumes are mapped to the inside of the container; in the process mode,

a temporary working directory is created for each service, and the storage volumes are soft linked to the working directory. If the service is stopped, the temporary working directory will be cleaned up, and the behavior is the same with container mode.

6. Finally, you can stop baetyl by `ctrl + c`, and the Master will notify all service instances to exit and wait. If it times out, it will force the instance to be killed. Then clean up baetyl.sock and exit.

The complete application.yml configuration as follows:

```
// AppConfig application configuration
type AppConfig struct {
        // specifies the version of the application configuration
        Version  string        `yaml:"version" json:"version"`
        // specifies the service information of the application
        Services []ServiceInfo `yaml:"services" json:"services" default:"[]"`
        // specifies the storage volume information of the application
        Volumes  []VolumeInfo  `yaml:"volumes" json:"volumes" default:"[]"`
}

// VolumeInfo storage volume configuration
type VolumeInfo struct {
        // specifies a unique name for the storage volume
        Name     string `yaml:"name" json:"name" validate:"regexp=^[a-zA-Z0-9][a-zA-
↪Z0-9_-]{0\\,63}$"`
        // specifies the directory where the storage volume is on the host
        Path     string `yaml:"path" json:"path" validate:"nonzero"`
}

// MountInfo storage volume mapping configuration
type MountInfo struct {
        // specifies the name of the mapped storage volume
        Name     string `yaml:"name" json:"name" validate:"regexp=^[a-zA-Z0-9][a-zA-
↪Z0-9_-]{0\\,63}$"`
        // specifies the directory where the storage volume is in the container
        Path     string `yaml:"path" json:"path" validate:"nonzero"`
        // specifies the operation permission of the storage volume, read-only or␣
↪writable
        ReadOnly bool   `yaml:"readonly" json:"readonly"`
}

// ServiceInfo service configuration
type ServiceInfo struct {
        // specifies the unique name of the service
        Name      string            `yaml:"name" json:"name" validate:"regexp=^[a-zA-
↪Z0-9][a-zA-Z0-9_-]{0\\,63}$"`
        // specifies the image of the service, usually using the docker image name
        Image     string            `yaml:"image" json:"image" validate:"nonzero"`
        // specifies the number of instances started
        Replica   int               `yaml:"replica" json:"replica" validate:"min=0"`
        // specifies the storage volumes that the service needs, map the storage␣
↪volume to the directory in the container
        Mounts    []MountInfo       `yaml:"mounts" json:"mounts" default:"[]"`
    // specifies the port bindings which exposed by the service, only for docker␣
↪container mode
        Ports     []string          `yaml:"ports" json:"ports" default:"[]"`
        // specifies the device bindings which used by the service, only for docker␣
↪container mode
        Devices   []string          `yaml:"devices" json:"devices" default:"[]"`
```

(continues on next page)

(continued from previous page)

```
        // specifies the startup arguments of the service program, but does not␣
↪include `arg[0]`
        Args      []string          `yaml:"args" json:"args" default:"[]"`
        // specifies the environment variable of the service program
        Env       map[string]string `yaml:"env" json:"env" default:"{}"`
        // specifies the restart policy of the instance of the service
        Restart   RestartPolicyInfo `yaml:"restart" json:"restart"`
        // specifies resource limits for a single instance of the service,  only for␣
↪docker container mode
        Resources Resources         `yaml:"resources" json:"resources"`
}
```

2.3.1 Engine

Engine is responsible for the storage volume mapping of services, instance start and stop, daemon, etc.. It abstracts the service operation, can implement different service running modes. Depending on the capabilities of the device, different running modes can be selected to run the services. The docker container mode and the native process mode are currently supported, and the k3s container mode will be supported later.

Docker Engine

The docker engine interprets the service Image as a docker image address and starts the service by calling the `Docker Engine` client. All services use a custom network provided by `Docker Engine` (default is baetyl), and the ports are exposed according to the `Ports` information. The directories are mapped according to the `Mounts` information, the devices are mapped according to the `Devices` information, and the resources that the containers can use, such as CPU, memory, etc., are configured according to the `Resources` information. Services can be accessed directly using the service name, which is routed by docker's DNS server. Each instance of the service corresponds to a container, and the engine is responsible for starting and stopping the container.

Native Engine

On platforms that do not provide container services (such as older versions of Windows), the Native engine simulates the container's experience as much as possible. The engine interprets the service image as the package name. The package is provided by the storage volume and contains the program required by the service, but the dependencies of this program (such as Python interpreter, Node interpreter, lib, etc.) need to be installed on the host in advance. All services use the host network directly, all ports are exposed, and users need to be careful to avoid port conflicts. Each instance of the service corresponds to a process, and the engine is responsible for starting and stopping the process.

***NOTE**: Process mode does not support resource restrictions, no need to expose ports, map devices.*

At present, the above two modes basically achieve unified configuration, leaving only the difference in service address configuration, so the configuration in example is divided into two directories, native and docker, but will eventually be unified.

2.3.2 RESTful API

The Baetyl Master exposes a set of RESTful APIs, adopts HTTP/1. By default, Unix Domain Socket is used on Linux systems, and the fixed address is `/var/run/baetyl.sock`. Other environments use TCP. The default address is `tcp://127.0.0.1:50050`. At present, the authentication mode of the interface adopts a simple dynamic token. When the Master starts the services, it will dynamically generate a Token for each service, and the service name and Token are transmitted to the service instance as environment variables which can be read by instance and sent to the

Master in request header. It should be noted that the dynamically launched instance cannot obtain the Token, so the dynamic instance cannot dynamically start other instances.

For the service instance, after the instance is started, you can get the API Server address of the Baetyl Master, the name and Token of the service, and the name of the instance from the environment variable. For details, see *Environment Variable*.

The Header key is as follows:

- x-openedge-username: service name as username
- x-openedge-password: dynamic token as password

The following are the currently available interfaces:

- GET /v1/system/inspect gets system information and status
- PUT /v1/system/update updates system and services
- GET /v1/ports/available gets available port on host
- PUT /v1/services/{serviceName}/instances/{instanceName}/start starts an instance of a service dynamically
- PUT /v1/services/{serviceName}/instances/{instanceName}/stop stops an instance of a service dynamically
- PUT /v1/services/{serviceName}/instances/{instanceName}/report reports the custom info or stats of the instance of the service

System Inspect

This interface is used to obtain the following information and status:

```
// Inspect all baetyl information and status inspected
type Inspect struct {
        // exception information
        Error    string    `json:"error,omitempty"`
        // inspect time
        Time     time.Time `json:"time,omitempty"`
        // software information
        Software Software  `json:"software,omitempty"`
        // hardware information
        Hardware Hardware  `json:"hardware,omitempty"`
        // service information, including service name, instance running status, etc.
        Services Services  `json:"services,omitempty"`
        // storage volume information, including name and version
        Volumes  Volumes   `json:"volumes,omitempty"`
}

// Software software information
type Software struct {
        // operating system information of host
        OS          string `json:"os,omitempty"`
        // CPU information of host
        Arch        string `json:"arch,omitempty"`
        // Baetyl process work directory
        PWD         string `json:"pwd,omitempty"`
        // Baetyl running mode of application services
        Mode        string `json:"mode,omitempty"`
        // Baetyl compiled Golang version
        GoVersion   string `json:"go_version,omitempty"`
```

(continues on next page)

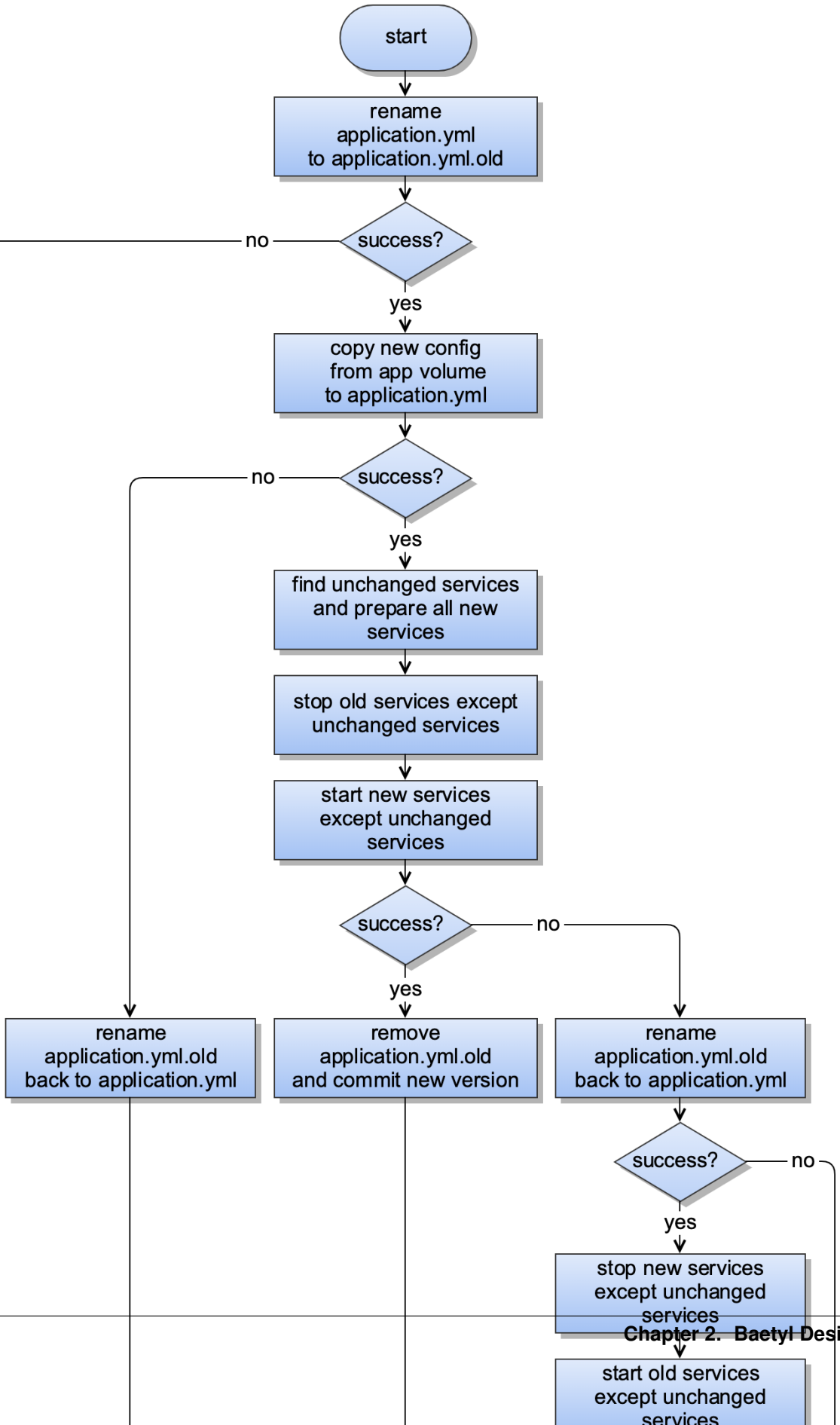
(continued from previous page)

```
        // Baetyl release version
        BinVersion  string `json:"bin_version,omitempty"`
        // Baetyl git revision
        GitRevision string `json:"git_revision,omitempty"`
        // Baetyl loaded application configuration version
        ConfVersion string `json:"conf_version,omitempty"`
}

// Hardware hardware information
type Hardware struct {
        // memory usage information of host
        MemInfo  *utils.MemInfo  `json:"mem_stats,omitempty"`
        // CPU usage information of host
        CPUInfo  *utils.CPUInfo  `json:"cpu_stats,omitempty"`
        // disk usage information of host
        DiskInfo *utils.DiskInfo `json:"disk_stats,omitempty"`
        // CPU usage information of host
        GPUInfo  []utils.GPUInfo `json:"gpu_stats,omitempty"`
}
```

System Update

This interface is used to update the application or the master binary in the system, which called the application OTA or the master OTA. The configuration of the volumes, networks, and services will be compared during application OTA. If the service and its related configuration are not changed, the service will not be restarted, otherwise it will be restarted.

The process of application OTA is as follows:


start
rename application.yml to application.yml.old
success?
no
yes
copy new config from app volume to application.yml
success?
no
yes
find unchanged services and prepare all new services
stop old services except unchanged services
start new services except unchanged services
success?
no
yes
rename application.yml.old back to application.yml
remove application.yml.old and commit new version
rename application.yml.old back to application.yml
success?
no
yes
stop new services except unchanged services
start old services except unchanged services

Instance Start&Stop

This interface is used to dynamically start and stop an instance of a service. You need to specify the service name and instance name. If you repeatedly launch an instance of the same name with the same service, the previously started instance will be stopped first, and then the new instance will be started.

This interface supports the dynamic configuration of the service to cover the static configuration in the storage volume. The overlay logic adopts the environment variable. When the instance starts, the environment variable can be loaded to overwrite the configuration in the storage volume to avoid resource conflicts. For example, in the native process mode, when the function manager service starts the function runtime instance, the free ports are allocated in advance, so that the function runtime instances can listen to different ports.

Instance Report

This interface is used to periodically report the custom status information of the service instance to the Baetyl Master. The content of the report is placed in the body of the request, and JSON format is used. The first layer of the JSON field is used as the key and its value will be overwritten if it is reported multiple times. For example:

If the instance of the service `infer` reports the following information for the first time, including `info` and `stats`:

```
{
    "info": {
        "company": "baidu",
        "scope": "ai"
    },
    "stats": {
        "msg_count": 124,
        "infer_count": 120
    }
}
```

The subsequent JSON that `baetyl-agent` reports to the cloud is as follows:

```
{
    ...
    "time": "0001-01-01T00:00:00Z",
    "services": [
        {
            "name": "infer",
            "instances": [
                {
                    "name": "infer",
                    "start_time": "2019-04-18T16:04:45.920152Z",
                    "status": "running",
                    ...

                    "info": {
                        "company": "baidu",
                        "scope": "ai"
                    },
                    "stats": {
                        "msg_count": 124,
                        "infer_count": 120
                    }
                }
            ]
        },
```

(continues on next page)

(continued from previous page)

```
    ]
    ...
}
```

If the instance of the service `infer` reports the following information for the second time, containing only `stats`, the old `stats` will be overwritten:

```
{
    "stats": {
        "msg_count": 344,
        "infer_count": 320
    }
}
```

The subsequent JSON that `baetyl-agent` reports to the cloud is as follows, the old `info` is kept and the old `stats` is overwritten:

```
{
    ...
    "time": "0001-01-01T00:00:00Z",
    "services": [
        {
            "name": "infer",
            "instances": [
                {
                    "name": "infer",
                    "start_time": "2019-04-18T16:04:46.920152Z",
                    "status": "running",
                    ...

                    "info": {
                        "company": "baidu",
                        "scope": "ai"
                    },
                    "stats": {
                        "msg_count": 344,
                        "infer_count": 320
                    }
                }
            ]
        },
    ]
    ...
}
```

2.3.3 Environment Variable

Baetyl currently sets the following system environment variables for the service instance:

- BAETYL_HOST_OS: Operate system of the device (host) where Baetyl is located
- BAETYL_HOST_ID: ID of the device (host) where Baetyl is located, can be used as device fingerprint
- BAETYL_MASTER_API_ADDRESS: API Server address of the Baetyl Master
- BAETYL_MASTER_API_VERSION: API version of the Baetyl Master

- BAETYL_SERVICE_MODE: Service running mode adopted by the Baetyl Master
- BAETYL_SERVICE_NAME: The name of the service
- BAETYL_SERVICE_TOKEN: Dynamically assigned Token
- BAETYL_SERVICE_INSTANCE_NAME: The name of the instance of the service
- BAETYL_SERVICE_INSTANCE_ADDRESS: The address of the instance of the service

The official function manager service is to connect to the Baetyl Master by reading `BAETYL_MASTER_API_ADDRESS`. For example, the `BAETYL_MASTER_API_ADDRESS` under Linux system is `unix:///var/run/baetyl.sock`; In the container mode under other systems, the default value of `BAETYL_MASTER_API_ADDRESS` is `tcp://host.docker.internal:50050`; In the process mode under other systems, the default value of `BAETYL_MASTER_API_ADDRESS` is `tcp://127.0.0.1:50050`.

***NOTE**: Environment variables configured in the application will be overwritten if they are the same as the above system environment variables.*

2.4 Official Modules

Currently, several modules are officially provided to meet some common application scenarios. Of course, developers can also develop their own modules.

2.4.1 baetyl-agent

The `baetyl-agent`, also known as the cloud agent module, is responsible for communicating with the BIE Cloud Management Suite. It has MQTT and HTTPS channels. MQTT enforces two-way authentication for SSL/TLS certificates. HTTPS enforces one-way authentication for SSL/TLS certificates. Developers can refer to this module to implement their own Agent module to connect their own cloud platform.

The cloud agent do three things at the moment:

1. After the startup, periodically obtain status information from the Master and report it to the cloud.
2. Listen to the events sent by the cloud, trigger the corresponding operations, and currently only process the application OTA event.
3. Responsible for cleaning the volume directory, the master will not be notified to do APP OTA during the volume cleaning period.

After receiving the application OTA command from the BIE Cloud Management Suite, the cloud agent first downloads the storage volume data packets used in all configurations and decompresses them to the specified location. If the storage volume data packets already exist and the MD5 is the same, the download will be skipped. After all storage volumes are ready, the cloud agent module will call the Master's `/update/system` interface to trigger the Master to update the system.

_**NOTE**: If the device cannot connect to the external network or needs to leave the cloud management suite, you can remove the Agent module from the application configuration and run offline. _

2.4.2 baetyl-hub

The `baetyl-hub` is a stand-alone version of the message subscription and distribution center that uses the MQTT3.1.1 protocol to provide reliable messaging services in low-bandwidth, unreliable networks. It acts as a messaging middleware for the Baetyl system, providing message-driven interconnect capabilities for all services.

Currently supports 4 access methods: TCP, SSL (TCP + SSL), WS (Websocket) and WSS (Websocket + SSL). The MQTT protocol support is as follows:

- Support `Connect`, `Disconnect`, `Subscribe`, `Publish`, `Unsubscribe`, `Ping`, etc.
- Support QoS levels 0 and 1
- Support `Retain`, `Will`, `Clean Session`
- Support topics subscribed with wildcards such as `+`, `#`
- Support validation of ClientID and Payload
- **Not Support** topics subscribed with prefix `$`
- **Not Support** Client's Keep Alive feature and QoS Level 2

NOTE:

- The maximum number of separators `/` in the publish and subscribe topics is no more than 8, and the topic name can be up to 255 characters in length.
- The maximum length of the message message is 32k. The maximum length that can be supported is 268, 435, 455 (Byte), about 256 MB, which can be modified by the `message` configuration item.
- ClientID supports uppercase and lowercase letters, numbers, underscores, hyphens (minus sign), and empty characters (not allowed to be empty if CleanSession is false), up to 128 characters in length
- The QoS of the message can only be dropped. For example, when the QoS of the original message is 0, even if the subscription QoS is 1, the message is sent at the level of QoS 0.
- If certificate mutual authentication is used, the client must send a **non-empty** username and **empty** password when connecting, username will be used for topic authentication. If password is not empty, it will further check if the password is correct.

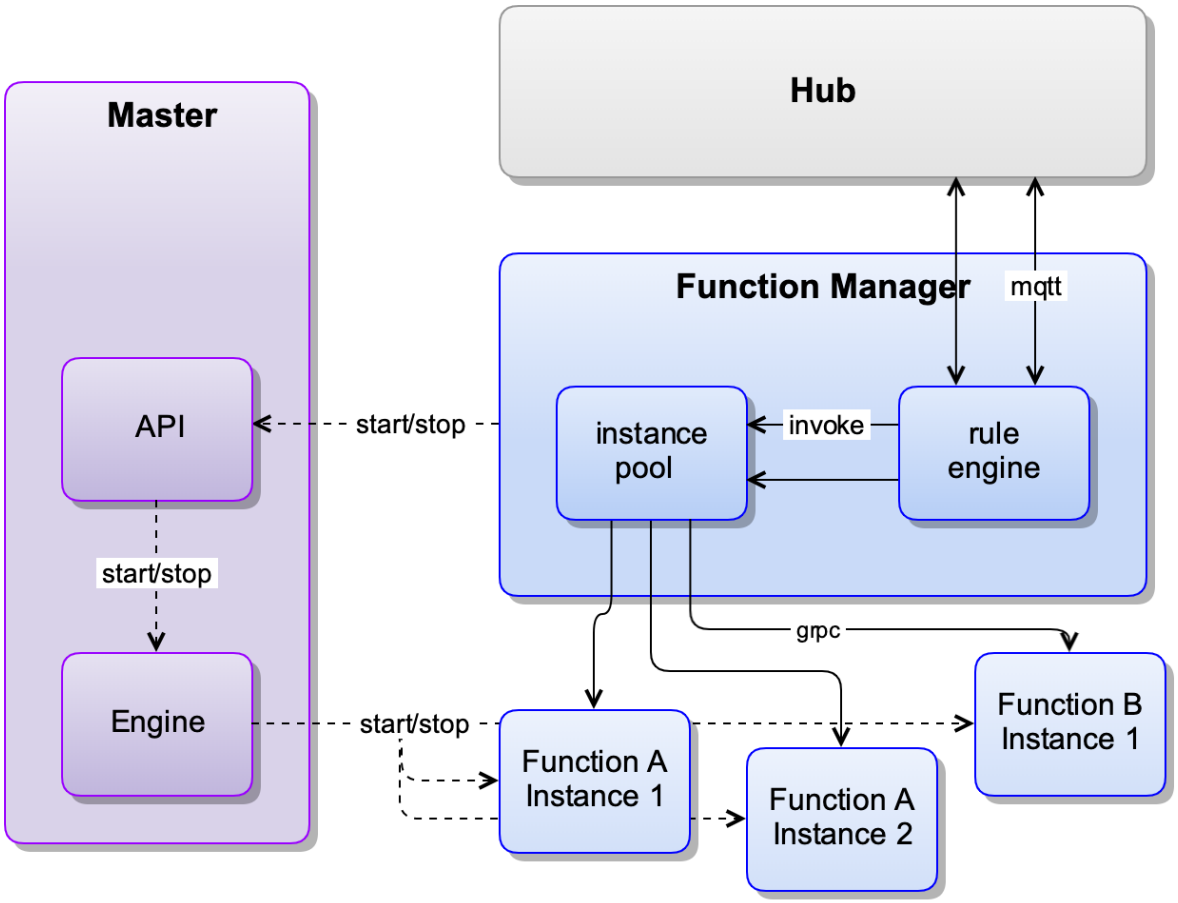
The Hub supports simple topic routing, such as subscribing to a message with the topic `t` and publishing it back with a new topic `t/topic`.

If this module does not meet your requirements, you can also use a third-party MQTT Broker/Server to replace it.

2.4.3 baetyl-function-manager

The `baetyl-function-manager`, also known as the function manager module, provides the computing power based on the MQTT message mechanism, flexible, highly available, scalable, and responsive, and compatible Baidu CFC. **It is important to note that function service do not guarantee message order, unless only one function instance is started**.

The function manager module is responsible for managing all function instances and message routing rules, and supports automatic scaling. The structure diagram is as follows:



Structure Diagram

If the function executes incorrectly, the function server returns a message in the following format for subsequent processing. Where `functionMessage` is the message input by the function (the message being processed), not the message returned by the function. An example is as follows:

```
{
    "errorMessage": "rpc error: code = Unknown desc = Exception calling application",
    "errorType": "*errors.Err",
    "functionMessage": {
        "ID": 0,
        "QOS": 0,
        "Topic": "t",
        "Payload": "eyJpZCI6MSwiZGV2aWNlIjoiMTExIn0=",
        "FunctionName": "sayhi",
        "FunctionInvokeID": "50f8f102-2b8c-4904-86df-0728811a5a4b"
    }
}
```

2.4.4 baetyl-function-python27

The design motion of module `baetyl-function-python27` is the same as the module `baetyl-function-python36` but their python runtime are different. The module `baetyl-function-python27` is based on python27 runtimeand provide the libs protobuf3grpcio based on Python2.7.

2.4.5 baetyl-function-python36

`baetyl-function-python36` provides Python functions similar to Baidu CFC, where users can handle messages by writing their own functions. It is very flexible to use for filtering, converting and forwarding messages. This module can be started separately as a GRPC service or as a function instance provider for the function manager module.

The input and output of a Python function can be either JSON or binary. The message Payload will try a JSON decoding (`json.loads(payload)`) before passing it as a parameter. If it succeeds, it will pass the dictionary type. If it fails, it will pass the original binary data.

Python functions support reading environment variables such as os.environ['PATH'].

Python functions support reading contexts such as context['functionName'].

An example is shown below:

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-
"""
module to say hi
"""

def handler(event, context):
    """
    function handler
    """
    event['functionName'] = context['functionName']
    event['functionInvokeID'] = context['functionInvokeID']
    event['messageQOS'] = context['messageQOS']
    event['messageTopic'] = context['messageTopic']
    event['sayhi'] = 'hello, world'
    return event
```

_**NOTE**: In the native process mode, to run sayhi.py provided in the example of this project, you need to install **Python3.6** and its packages **pyyaml**, **protobuf3** and **grpcio** (pip installation can be used, `**pip3** install pyyaml protobuf grpcio`). _

2.4.6 baetyl-function-node85

The design motion of module `baetyl-function-node85` is the same as the module `baetyl-function-python36`, and provide Node8.5 runtime for Baetyl, where users can write javascripts to handle messages in JSON or binary format. An example is shown below:

```
#!/usr/bin/env node

exports.handler = (event, context, callback) => {
  result = {};

  if (Buffer.isBuffer(event)) {
      const message = event.toString();
      result["msg"] = message;
      result["type"] = 'non-dict';
  }else {
      result["msg"] = event;
      result["type"] = 'dict';
  }
```

(continues on next page)

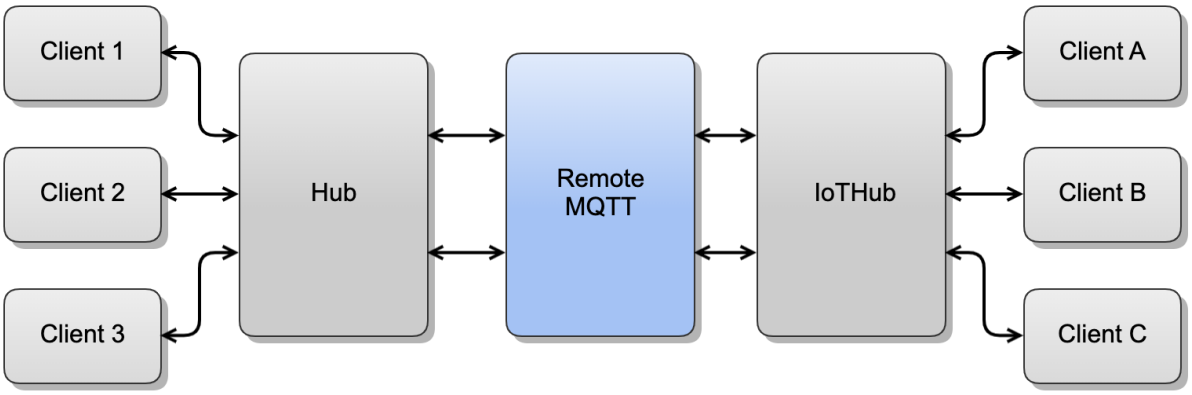
(continued from previous page)

```
  result["say"] = 'hello world';
  callback(null, result);
};
```

***NOTE**: In the native process mode, to run index.js provided in the example of this project, you need to install **Node8.5**.*

2.4.7 baetyl-remote-mqtt

`baetyl-remote-mqtt`, also known as the remote MQTT communication module, bridges two MQTT Servers for message synchronization. Currently, you can configure multiple message routing rules. The structure is as follows:



Remote MQTT Communication Example

As shown in the figure above, the Baetyl remote communication module is used to forward and synchronize messages between the Baetyl Local Hub Module and the remote cloud Hub. Further, the edge-cloud collaborative message forwarding and delivery can be realized by accessing the MQTT Client at both ends.

CHAPTER 3

Contributing

Welcome to Baetyl Open Source Project. To contribute to Baetyl, please follow the process below.

We sincerely appreciate your contribution. This document explains our workflow and work style.

3.1 Workflow

Baetyl use this Git branching model. The following steps guide usual contributions.

1. Fork

   Our development community has been growing fast, so we encourage developers to submit code. And please file Pull Requests from your fork. To make a fork, please refer to Github page and click on the "Fork" button.

2. Prepare for the development environment

```
go get github.com/baetyl/baetyl # clone baetyl official repository
cd $GOPATH/src/github.com/baetyl/baetyl # step into baetyl
git checkout master  # verify master branch
git remote add fork https://github.com/<your_github_account>/baetyl  # specify␣
↪remote repository
```

3. Push changes to your forked repository

```
git status   # view current code change status
git add .    # add all local changes
git commit -c "modify description"  # commit changes with comment
git push fork # push code changes to remote repository which specifies your␣
↪forked repository
```

4. Create pull request

   You can push and file a pull request to Baetyl official repository https://github.com/baetyl/baetyl. To create a pull request, please follow these steps. Once the Baetyl repository reviewer approves and merges your pull request, you will see the code which contributed by you in the Baetyl official repository.

3.2 Code Review

- About Golang format, please refer to Go Code Review Comments.
- Please feel free to ping your reviewers by sending them the URL of your pull request via email. Please do this after your pull request passes the CI.
- Please answer reviewers' every comment. If you are to follow the comment, please write "Done"; please give a reason otherwise.
- If you don't want your reviewers to get overwhelmed by email notifications, you might reply their comments by in a batch.
- Reduce the unnecessary commits. Some developers commit often. It is recommended to append a sequence of small changes into one commit by running `git commit --amend` instead of `git commit`.

3.3 Merge Rule

- Please run command `govendor fmt +local` before push changes, more details refer to govendor
- Must run command `make test` before push changes(unit test should be contained), and make sure all unit test and data race test passed
- Only the passed(unit test and data race test) code can be allowed to submit to Baetyl official repository
- At least one reviewer approved code can be merged into Baetyl official repository

Note: The document's contribution rules are the same as the rules above.

- English document homepage
- Chinese document homepage
- English Document Project
- Chinese Document Project

CHAPTER 4

Quick Install Baetyl

Compared to manually download software in previous version, it supports installing `Baetyl` through package manager in newer version. With this method, users can quickly install `Baetyl` by simply typing a few commands at terminal.

Installation packages are provided for Ubuntu16.04, Ubuntu18.04, Debian9, CentOS7 and Raspbian-stretch currently. The supported platforms are amd64, i386, armv7l, and arm64.

Baetyl supports two running modes: **docker** container mode and **native** process mode. This document will be described in **docker** container mode.

4.1 Install the container runtime

Baetyl relies on docker container runtime in **docker** container mode. Users can install docker (for Linux-like systems) with the following command if it's not installed yet:

```
curl -sSL https://get.docker.com | sh
```

View the version of installed docker:

```
docker version
```

NOTEAccording to the Official Release Log, the version of docker lower than 18.09.2 has some security implications. It is recommended to install/update the docker to 18.09.2 and above.

For more details, please see the official documentation.

4.2 Install Baetyl

The rpm and deb packages will be released accordingly when Baetyl releases a new version. Users can install Baetyl to the device through package manager with following command:

```
curl -sSL http://dl.baetyl.io/install.sh | sudo -E bash -
```

If everything is ok, Baetyl will be installed on the `/usr/local` directory after the execution is complete.

4.3 Import the example configuration (optional)

As an edge computing framework, Baetyl provides MQTT connect service through hub module, provides local functional service through function manager module and some runtime modules like python27, python36, nodejs85, sql and so on. What's more, all the modules are started by Baetyl main program through a configuration file. More detailed contents about the module's configuration please refer to *Configuration Interpretation* for further information.

Baetyl officially provides an example configuration for some module which can be imported using following command:

```
curl -sSL http://dl.baetyl.io/install_with_docker_example.sh | sudo -E bash -
```

The example configuration is for learning and testing purposes only. Users should perform on-demand configuration according to actual working scenarios.

There is no need to import any configuration files if no modules need to launch.

4.4 Start Baetyl

The newer version of Baetyl uses `Systemd` as a daemon, and users can start Baetyl with the following command:

```
sudo systemctl start baetyl
```

If you have previously installed Baetyl or imported a new configuration file, it is recommended to use the reboot method:

```
sudo systemctl restart baetyl
```

Stop Baetyl:

```
sudo systemctl stop baetyl
```

If users only want to run Baetyl in the foreground, execute the following command::

```
sudo baetyl start
```

4.5 Verify successful installation

After installation, users can verify whether Baetyl is successfully installed or not by the following steps:

- executing the command `sudo systemctl status baetyl` to check whether `baetyl` is running, as shown below. Otherwise, `baetyl` fails to start.

```
baetyl@baetyl:~$ sudo systemctl status baetyl
● baetyl.service - Baetyl
   Loaded: loaded (/lib/systemd/system/baetyl.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2019-09-23 11:31:57 CST; 5h 55min ago
 Main PID: 997 (baetyl)
    Tasks: 10 (limit: 4623)
   CGroup: /system.slice/baetyl.service
           └─997 /usr/local/bin/baetyl start

9月 23 11:31:57 baetyl systemd[1]: Started Baetyl.
```

Baetyl

- Executing the command `docker stats` to view the running status of docker containers. Since the main program baetyl will first pull required images from docker mirror repository, it will take 2~5 minutes to see the baetyl starts successfully. Take the example configurations as above, the running status of containers are as shown below. If some containers are missing, it means they failed to start.

```
CONTAINER ID        NAME                                        CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
10131b874143        function-node85-sayhi.node85-sayhi.1        0.00%               0B / 0B             0.00%               72.5kB / 58.7kB     0B / 0B             0
ff1f55107a12        function-sql-filter.sql-filter.1            0.00%               0B / 0B             0.00%               75.8kB / 43.1kB     2.23MB / 0B         0
781838b42c4b        function-python36-sayhi.python36-sayhi.1    0.09%               0B / 0B             0.00%               72.5kB / 60.6kB     28.7kB / 0B         0
0618c0c49e45        function-python27-sayhi.python27-sayhi.1    0.09%               0B / 0B             0.00%               72.1kB / 60.4kB     1.72MB / 0B         0
2a29e8e17192        timer                                       0.00%               0B / 0B             0.00%               29kB / 13kB         135kB / 0B          0
e8c36a1d92aa        function-manager                            0.00%               0B / 0B             0.00%               321kB / 335kB       922kB / 0B          0
23b8aa709fee        localhub                                    13.05%              0B / 0B             0.00%               153kB / 87.6kB      229kB / 156kB       0
```

docker stats

- Under the condition of two above failures, you need to view the log of main program. And the log file which is stored in /usr/local/var/log/baetyl/baetyl.log by default. Once found errors in the log file, users can refer to *FAQ*. If necessary, just Submit an issue.

CHAPTER 5

Build Baetyl From Source

Compared to the quick installation of Baetyl, users can compile Baetyl from source to get the latest features.

Before compiling, users should configure the build environment. So this article consist of two parts: **environment configuration** and **source code compilation**.

5.1 Environment Configuration

5.1.1 Linux Platform

Install Go

Go to *related resources* to complete the download, then:

```
tar -C /usr/local -zxf go$VERSION.$OS-$ARCH.tar.gz  # Decompress the Go archive to␣
↪the /usr/local directory
export PATH=$PATH:/usr/local/go/bin # Configuring environment variables
export GOPATH=yourpath  # GOPATH setting
go env  # View Go's environment variables
go version # View Go's version
```

NOTE: Baetyl requires that the compiled version of Go should be above 1.10.0.

Install the container runtime

In **docker** container mode, Baetyl relies on docker container runtime. If `docker` is not installed yet, users can install the latest version of docker (for Linux-like systems) with the following command:

```
curl -sSL https://get.docker.com | sh
```

View the version of installed docker:

```
docker version
```

NOTEAccording to the Official Release Log, the version of docker lower than 18.09.2 has some security implications. It is recommended to install/update the docker to 18.09.2 and above.

For more details, please see the official documentation.

5.1.2 Darwin Platform

Install Go

- Install by using HomeBrew

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"  # install homebrew
brew install go
```

Modify the environment configuration file after the installation is complete.(e.g: ~/.bash_profile)

```
export GOPATH="${HOME}/go"
export GOROOT="$(brew --prefix golang)/libexec"
export PATH="$PATH:${GOPATH}/bin:${GOROOT}/bin"
```

Make the environment variable take effect:

```
source yourfile
```

Create the GOPATH specified directory:

```
test -d "${GOPATH}" || mkdir "${GOPATH}"
```

- Install by using binary file

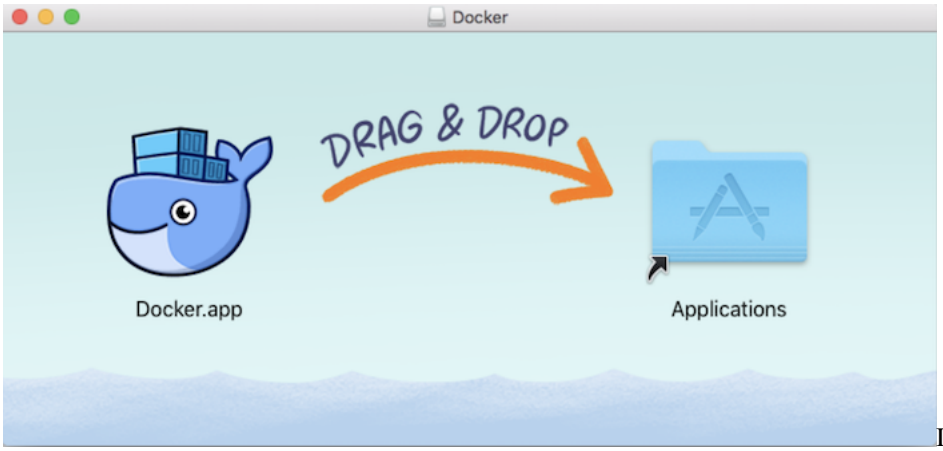
Go to *related resources* to complete the download, then:

```
tar -C /usr/local -zxf go$VERSION.$OS-$ARCH.tar.gz  # Decompress the Go archive to␣
↪the /usr/local directory
export PATH=$PATH:/usr/local/go/bin # Configuring environment variables
export GOPATH=yourpath  # GOPATH setting
go env  # View Go's environment variables
go version # View Go's version
```

NOTE: Baetyl requires that the compiled version of Go should be above 1.10.0.

Install the container runtime

Go to official page to download the .dmg file you need. Once done, double-click to open and drag docker into the application folder.



Install On Darwin

View the version of installed docker:

```
docker version
```

5.2 Source Code Compilation

5.2.1 Download Source Code

After completing the compilation environment configuration according to the corresponding environment, go to the Baetyl Github Page to download source code of baetyl.

```
go get github.com/baetyl/baetyl
```

5.2.2 Build Docker Image

In container mode, docker starts the module by running the image corresponding to each module, so build the mirror first with the following command:

```
cd $GOPATH/src/github.com/baetyl/baetyl
make clean
make image # build image
```

Note: Under the Darwin system, users need to specify the compilation parameters because the compiled images themselves are based on Linux system:

```
env GOOS=linux GOARCH=amd64 make image
```

The following docker images are generated by the above command:

```
baetyl-agent:latest
baetyl-hub:latest
baetyl-function-manager:latest
baetyl-remote-mqtt:latest
baetyl-timer:latest
baetyl-function-python27:latest
baetyl-function-python36:latest
baetyl-function-node85:latest
```

View the generated images with the following command:

```
docker images
```

5.2.3 Compile

```
cd $GOPATH/src/github.com/baetyl/baetyl
make rebuild
```

Note: You need to install `node` and `npm` beforehand because the Node 8.5 runtime module will call `npm install` command to install dependencies during `make`. For details, please refer to [Nodejs official website] (https://nodejs.org/en/download /).

After the compilation is completed, the following executable files will be generated in the root directory and each module directory, respectively:

```
baetyl
baetyl-agent/baetyl-agent
baetyl-hub/baetyl-hub
baetyl-function-manager/baetyl-function-manager
baetyl-remote-mqtt/baetyl-remote-mqtt
baetyl-timer/baetyl-timer
```

In addition, `package.zip` files are generated in each module directory.

5.2.4 Install

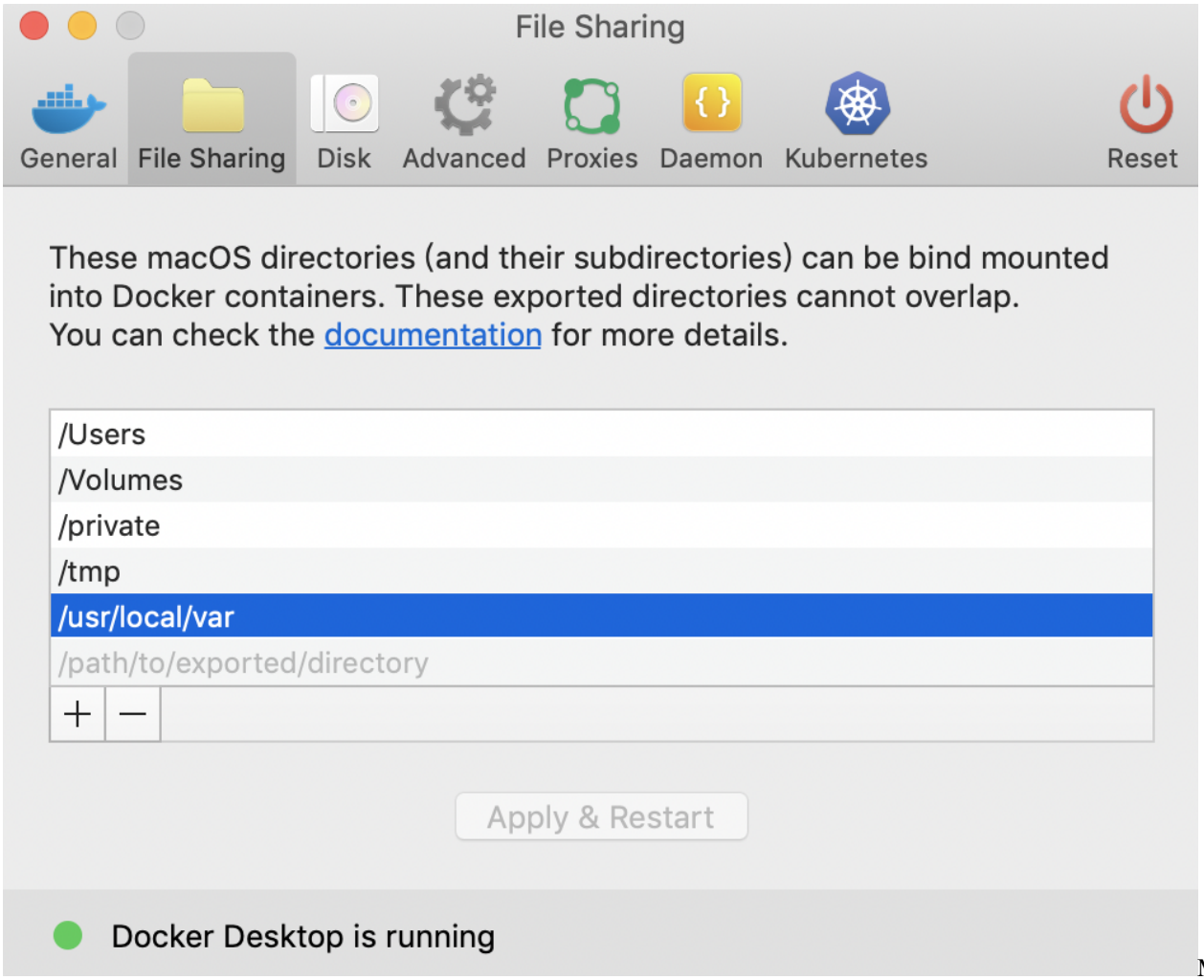
Install to default path: `/usr/local`

```
cd $GOPATH/src/github.com/baetyl/baetyl
make install # install for docker mode with example configuration
make install-native # install for native mode with example configuration
```

Specify the installation path, such as installing into the output directory:

```
cd $GOPATH/src/github.com/baetyl/baetyl
make install PREFIX=output
```

On the Darwin platform, you need to set the `/usr/local/var` directory to make it (and it's subdirectories) can be bind mounted into Docker containers which would be used by Baetyl.



Mount path on Mac

5.2.5 Run

If the program is already installed to the default path: `/usr/local`

```
sudo baetyl start
```

If the program has been installed to the specified path, such as installing into the output directory:

```
cd $GOPATH/src/github.com/baetyl/baetyl
sudo ./output/bin/baetyl start
```

NOTE:

1. After the baetyl is started, you can check if the baetyl has run successfully by `ps -ef | grep "baetyl"` and determine the parameters used at startup. And you can check the log file for details. Log files are stored by default in the `var/log/baetyl` directory of the working directory.
2. If run in docker container mode, the container runtime status can be viewed via the `docker stats` command.
3. To use your own image, you need to modify the **image** of the modules and functions in the application configuration to specify your own image.

4. For custom configuration, follow the instructions in *Configuration Interpretation* to make the relevant settings.

5.2.6 Uninstall

If it is the default installation:

```
cd $GOPATH/src/github.com/baetyl/baetyl
make clean # Can be used to clean executable files generated by compilation
make uninstall # Uninstall if you install in docker mode
make uninstall-native # Uninstall if you install in native mode
```

If the installation path is specified, for example, it is installed into the output directory.

```
cd $GOPATH/src/github.com/baetyl/baetyl
make clean # Can be used to clean executable files generated by compilation
make uninstall PREFIX=output # Uninstall if you install in docker mode
make uninstall-native PREFIX=output # Uninstall if you install in native mode
```

CHAPTER 6

Baetyl Configuration Interpretation

Supported units:

- Size unit: b(byte), k(kilobyte), m(megabyte), g(gigabyte)
- Time unit: s(second), m(minute), h(hour)

Configuration examples can be found in the `example` directory of this project.

6.1 Master Configuration

The Master configuration and application configuration are separated. The default configuration file is `etc/baetyl/conf.yml` in the working directory. The configuration is interpreted as follows:

```
mode: The default value is `docker`, running mode of services. **docker** container␣
↪mode or **native** process mode
grace: The default value is `30s`, the timeout for waiting services to gracefully␣
↪exit.
server: API Server configuration of Master.
  address: The default value can be read from environment variable `BAETYL_MASTER_API_
↪ADDRESS`, address of API Server.
  timeout: The default value is `30s`, timeout of API Server.
logger: Logger configuration
  path: The default is `empty` (none configuration), that is, it does not write to␣
↪the file. If the path is specified, it writes to the file.
  level: The default value is `info`, log level, support `debug``info``warn` and␣
↪`error`.
  format: The default value is `text`, log print format, support `text` and `json`.
  age:
    max: The default value is `15`, means maximum number of days the log file is kept.
  size:
    max: The default value is `50`, log file size limit, default unit is `MB`.
  backup:
    max: The default value is `15`, the maximum number of log files to keep.
```

6.2 Application Configuration

The default configuration file for the application configuration is `var/db/baetyl/application.yml` in the working directory. The configuration is interpreted as follows:

```
version: Application version
services: Service list configuration
  - name: [MUST] Service name, must be unique in the service list
    image: [MUST] Service entry. In the docker container mode, which means the␣
↪address of image. In the native process mode indicates where the service program␣
↪package is located.
    replica: The default value is 0, the number of service copies, indicating the␣
↪number of service instances started. Usually the service only needs to start one.␣
↪The function runtime service is generally set to 0, not started by the Master, but␣
↪is dynamically started by the function manager service.
    mounts: Storage volume mapping list
      - name: [MUST] The volume name, corresponding to one of the storage volume lists
        path: [MUST] The path mapped by the volume in the container
        readonly: The default value is false, whether the storage volume is read-only
    ports: Ports exposed in docker container mode, for example
      - 0.0.0.0:1883:1883
      - 0.0.0.0:1884:1884/tcp
      - 8080:8080/tcp
      - 9884:8884
    devices: Device mapping in docker container mode, for example
      - /dev/video0
      - /dev/sda:/dev/xvdc:r
    args: Service instance startup arguments, for example
      - '-c'
      - 'conf/conf.yml'
    env: Environment variables of service instance, for example
      version: v1
    restart: Service restart policy configuration
      retry:
        max: The default is `empty`(none configuration), which means always retry. If␣
↪not, which means the maximum number of service restarts.
      policy: The default value is `always`, restart policy, support `no`, `always`␣
↪and `on-failure`. And `no` means none restart, `always` means always restart, `on-
↪failure` means restart the service if it exits abnormally.
      backoff:
        min: The default value is `1s`, minimum interval of restart.
        max: The default value is `5m`, maximum interval of restart.
        factor: The default value is `2`, factor of interval increase.
    resources: Service instance resource limit configuration in docker container mode
      cpu:
        cpus: The percentage of CPU available of the service instance, for example `1.
↪5`, means that `1.5` CPU cores can be used.
        setcpus: The CPU core available for the service instance, for example `0-2`,␣
↪means that `0` to `2` CPU cores can be used; `0` means that the 0th CPU core can be␣
↪used; `1`, which means the 1st CPU core can be used.
      memory:
        limit: The available memory of the service, for example `500m`, means that␣
↪500 megabytes of memory can be used.
        swap: The swap space available to the service, for example `1g`, means that␣
↪1G of memory can be used.
      pids:
        limit: Number of processes the service can create.
```

(continues on next page)

(continued from previous page)

```
volumes: Storage volume list
  - name: [MUST] The volume name, must be unique in the list of storage volumes
    path: [MUST] The path of the storage volume on the host, relative to the working␣
↪directory of the Master
```

6.3 baetyl-agent Configuration

```
remote:
  mqtt: MQTT channel configuration
    clientid: [MUST] The Client ID, must be the id of cloud core device.
    address: [MUST] The endpoint address for client to connect with cloud management␣
↪suit, must use ssl endpoint.
    username: [MUST] The client username, must be the username of cloud core device.
    ca: [MUST] The CA path for client to connect with cloud management suit.
    key: [MUST] The private key path for client to connect with cloud management suit.
    cert: [MUST] The public key path for client to connect with cloud management suit.
    timeout: The default value is `30s`, means timeout of the client connects to␣
↪cloud.
    interval: The default value is `1m`, means maximum interval of client␣
↪reconnection, doubled from 500 microseconds to maximum.
    keepalive: The default value is `1m`, means keep alive time between the client␣
↪and cloud after connection has been established.
    cleansession: The default value is `false`, , means whether keep session in cloud␣
↪after client disconnected.
    validatesubs: The default value is `false`, means whether the client checks the␣
↪subscription result. If it is true, client exits and return errors when␣
↪subscription is failure.
    buffersize: The default value is `10`, means the size of the memory queue sent by␣
↪the client to the cloud management suit. If found exception, the client will exit␣
↪and lose message.
  http: HTTPS channel configuration
    address: This address is automatically inferred based on the address of the MQTT␣
↪channel. No configuration required
    timeout: The default value is `30s`, connection timeout period
  report: Agent report configuration.
    url: The report URL. No configuration required
    topic: The template of report topic. No configuration required
    interval: The default value is `20s`, interval of reporting.
  desire: Agent desire configuration.
    topic: The template of desire topic. No configuration required
```

6.4 baetyl-hub Configuration

```
listen: [MUST] Listening address, for example
  - tcp://0.0.0.0:1883
  - ssl://0.0.0.0:1884
  - ws://:8080/mqtt
  - wss://:8884/mqtt
certificate: SSL/TLS certificate authentication configuration, if `ssl` or `wss` is␣
↪enabled, it must be configured.
  ca: Server CA certificate path
```

(continues on next page)

(continued from previous page)

```
  key: Server private key path
  cert: Server public key path
principals: ACL configuration. If not configured, client cannot connect to this Hub,␣
↪support username/password and certificate authentication.
  - username: Username for client non-ssl connection
    password: Password for client connection
    permissions:
      - action: Operation type of permission. `pub` means publish permission, `sub`␣
↪means subscription permission.
        permit: List of topics allowed by the operation type, support `+` and `#`␣
↪wildcards.
  - username: Username for client ssl connection
    permissions:
      - action: Operation type of permission. `pub` means publish permission, `sub`␣
↪means subscription permission.
        permit: List of topics allowed by the operation type, support `+` and `#`␣
↪wildcards.
subscriptions: Topic routing configuration
  - source:
      topic: subscribe topic
      qos: QoS of topic
    target:
      topic: publish topic
      qos: QoS of topic
message: MQTT message related configuration
  length:
    max: The default value is `32k`, which means maximum message length that can be␣
↪allowed to be transmitted. The maximum can be set to 268,435,455 Byte(about 256MB).
  ingress: Message receive configuration
    qos0:
      buffer:
        size: The default value is `10000`, means the number of messages that can be␣
↪cached in memory with QoS0. Increasing the cache can improve the performance of␣
↪message reception. If the device loses power, it will directly discard the message␣
↪with QoS0.
    qos1:
      buffer:
        size:  The default value is `100`, means the message cache size of waiting␣
↪for persistent with QoS1. Increasing the cache can improve the performance of␣
↪message reception, but the potential risk is that the service will exit␣
↪abnormally(such as device power failure), it will lose the cached message, and will␣
↪not reply(puback). The service exits normally and waits for the cached message to␣
↪be processed without losing data.
      batch:
        max:  The default value is `50`, means the maximum number of messages with␣
↪QoS1 can be insert into the database (persistence). After the message is persisted,␣
↪it will reply with confirmation(ack).
      cleanup:
        retention:  The default value is `48h`, means the time that the message with␣
↪QoS1 can be saved in the database. Messages that exceed this time will be␣
↪physically deleted during cleanup.
        interval:  The default value is `1m`, means cleanup interval with QoS1.
  egress: Message publish configuration
    qos0:
      buffer:
        size:  The default value is `10000`, means the number of messages to be sent␣
↪in the in-memory cache wit QoS0. If the device is powered off, the message will be␣
↪discarded directly. After the buffer is full, the newly pushed message will be␣
↪discarded directly.
```

(continues on next page)

(continued from previous page)

```
    qos1:
      buffer:
        size:  The default value is `100`, means the size of the message buffer is␣
↪not confirmed(ack) after the message with QoS1 is sent. After the buffer is full,␣
↪the new message is no longer read, and the message in the cache is always␣
↪acknowledged(ack). After the message with QoS1 is sent to the client, it waits for␣
↪the client to confirm(puback). If the client does not reply within the specified␣
↪time, the message will be resent until the client replies or the session is closed.
      batch:
        max:  The default value is `50`, means the maximum number of messages read␣
↪from the database in batches.
      retry:
        interval:  The default value is `20s`, means the re-publish interval of␣
↪message.
  offset: Message serial number persistence related configuration
    buffer:
      size:  The default value is `10000`, means the size of the cache queue for the␣
↪serial number of the message that was acknowledged(ack). For example, three␣
↪messages with QoS1 and serial numbers 1, 2, and 3 are sent to the client in batches.
↪ The client confirms the messages of sequence numbers 1 and 3. At this time,␣
↪sequence number 1 will be queued and persisted. Although sequence number 3 has been␣
↪confirmed, it still has to wait for the serial number 2 to be confirmed before␣
↪entering the column. This design can ensure that the message can be recovered from␣
↪the persistent serial number after the service restarts abnormally, ensuring that␣
↪the message is not lost, but the message retransmission will occur, and therefore␣
↪the message with QoS 2 is not supported.
    batch:
      max:  The default value is `100`, means the maximum number of batches of␣
↪message serial numbers can be insert into the database.
logger: Logger configuration
  path: The default is `empty` (none configuration), that is, it does not write to␣
↪the file. If the path is specified, it writes to the file.
  level: The default value is `info`, log level, support `debug``info``warn` and␣
↪`error`.
  format: The default value is `text`, log print format, support `text` and `json`.
  age:
    max: The default value is `15`, means maximum number of days the log file is kept.
  size:
    max: The default value is `50`, log file size limit, default unit is `MB`.
  backup:
    max: The default value is `15`, the maximum number of log files to keep.
status: Service status configuration
  logging:
    enable: The default value is `false`, means whether to print baetyl status␣
↪information.
    interval: The default value is `60s`, means interval of printing baetyl status␣
↪information.
storage: Database storage configuration
  dir: The default value is `var/db/baetyl/data`, means database storage directory.
shutdown: Service exit configuration
  timeout: The default value is `10m`, means timeout of service exit.
```

6.5 baetyl-function-manager Configuration

```
hub:
  clientid: [MUST] The Client ID for the client to connect with the local Hub.
  address: [MUST] The endpoint address for the client to connect with the local Hub.
  username: The username for the client to connect with the local hub.
  password: The password for the client to connect with the local hub.
  ca: The CA path for the client to connect with the local hub.
  key: The private key path for the client to connect with the local hub.
  cert: The public key path for the client to connect with the local hub.
  timeout: The default value is `30s`, means timeout of the client connection with␣
↪the local hub.
  interval: The default value is `1m`, means maximum interval of client reconnection,␣
↪doubled from 500 microseconds to maximum.
  keepalive: The default value is `1m`, means keep alive time between the client and␣
↪the local hub after connection has been established.
  cleansession: The default value is `false`, , means whether keep session in the␣
↪local Hub after client disconnected.
  validatesubs: The default value is `false`, means whether the client checks the␣
↪subscription result. If it is true, client exits and return errors when␣
↪subscription is failure.
  buffersize: The default value is `10`, means the size of the memory queue sent by␣
↪the client to the local Hub. If found exception, the client will exit and lose␣
↪messages.
rules: Router rules configuration
  - clientid: [MUST] The Client ID for client to connect with the local Hub
    subscribe:
      topic: [MUST] The message topic subscribed from the local Hub.
      qos: The default value is `0`, the message QoS subscribed from the local Hub.
    function:
      name: [MUST] The name of the function that processes the message.
    publish:
      topic: [MUST] The message topic published to the local Hub.
      qos: The default value is `0`, means the message QoS published to the local Hub.
functions:
  - name: [MUST] The function name, must be unique in the function list.
    service: [MUST] The service name which provides the function runtime instance.
    instance: function instance configuration
      min: The default value is `0`, means the minimum number of function instance.␣
↪And the minimum configuration allowed to be set is `0`, the maximum configuration␣
↪allowed to be set is `100`.
      max: The default value is `1`, means the maximum number of function instance.␣
↪And the minimum configuration allowed to be set is `1`, the maximum configuration␣
↪allowed to be set is `100`.
      idletime: The default value is `10m`, maximum idle time of function instance.
      evicttime: The default value is `1m`, interval time between two evict␣
↪operations.
      message:
        length:
          max: The default value is `4m`, means the maximum message length allowed␣
↪for function instances to be received and publish.
    backoff:
      max: The default value is `1m`, the maximum reconnection interval of the client␣
↪connection function instance
    timeout: The default value is `30s`, Client connection function instance timeout
```

6.6 baetyl-function-python Configuration

```
# the configurations of the two modules are the same, so we can follow this sample␣
↪below
server: GRPC Server configuration; Do not configure if the instances of this service␣
↪are managed by baetyl-function-manager
  address: GRPC Server address, <host>:<port>
  workers:
    max: The default value is the number of CPU core multiplied by 5, the maximum␣
↪capacity of the thread pool
  concurrent:
    max: The default value is `empty`, means no limit, the maximum number of␣
↪concurrent connections
  message:
    length:
      max: The default value is `4m`, the maximum message length allowed for function␣
↪instances to receive and send
  ca: Server CA certificate path
  key: Server private key path
  cert: Server public key path
functions: function list
  - name: [MUST] The function name, must be unique in the function list.
    handler: [MUST] The function of Python code to handle message, for example,
↪'sayhi.handler'
    codedir: [MUST] The path of Python code
logger: Logger configuration
  path: The default is `empty` (none configuration), that is, it does not write to␣
↪the file. If the path is specified, it writes to the file.
  level: The default value is `info`, log level, support `debug``info``warn` and␣
↪`error`.
  format: The default value is `text`, log print format, support `text` and `json`.
  age:
    max: The default value is `15`, means maximum number of days the log file is kept.
  size:
    max: The default value is `50`, log file size limit, default unit is `MB`.
  backup:
    max: The default value is `15`, the maximum number of log files to keep.
```

6.7 baetyl-remote-mqtt Configuration

```
hub:
  clientid: [MUST] The Client ID for the client to connect with the local Hub.
  address: [MUST] The endpoint address for the client to connect with the local Hub.
  username: The username for the client to connect with the local hub.
  password: The password for the client to connect with the local hub.
  ca: The CA path for the client to connect with the local hub.
  key: The private key path for the client to connect with the local hub.
  cert: The public key path for the client to connect with the local hub.
  timeout: The default value is `30s`, means timeout of the client connection with␣
↪the local hub.
  interval: The default value is `1m`, means maximum interval of client reconnection,␣
↪doubled from 500 microseconds to maximum.
  keepalive: The default value is `1m`, means keep alive time between the client and␣
↪the local hub after connection has been established.
```

(continues on next page)

(continued from previous page)

```
  cleansession: The default value is `false`, , means whether keep session in the␣
↪local Hub after client disconnected.
  validatesubs: The default value is `false`, means whether the client checks the␣
↪subscription result. If it is true, client exits and return errors when␣
↪subscription is failure.
  buffersize: The default value is `10`, means the size of the memory queue sent by␣
↪the client to the local Hub. If found exception, the client will exit and lose␣
↪messages.
rules: Message routing rules configuration
  - hub:
      clientid: The client ID for the client to connect with the local Hub.
      subscriptions: The topics subscribed by client from Hub, for example
        - topic: say
          qos: 1
        - topic: hi
          qos: 0
    remote:
      name: [MUST] The remote name, must be one of the remote list
      clientid: The client ID for the client to connect with remote Hub.
      subscriptions: The topics subscribed by client from remote Hub, for example
        - topic: remote/say
          qos: 0
        - topic: remote/hi
          qos: 0
remotes: The remote list
  - name: [MUST] The remote name, must be unique in this list.
    clientid: The client ID for the client to connect with the remote Hub.
    address: [MUST] The address for the client connect with the remote Hub.
    username: The username for the client connect with the remote Hub.
    password: The password for the client connect with the remote Hub.
    ca: The CA path for the client connect with the remote Hub.
    key: The private key path for the client connect with the remote Hub.
    cert: The public key path for the client connect with the remote Hub.
    timeout: The default value is `30s`, means timeout of the client connect to the␣
↪remote Hub.
    interval: The default value is `1m`, means maximum interval of client␣
↪reconnection, doubled from 500 microseconds to maximum.
    keepalive: The default value is `1m`, means keep alive time between the client␣
↪and the local hub after connection has been established.
    cleansession: The default value is `false`, , means whether keep session in the␣
↪local Hub after client disconnected.
    validatesubs: The default value is `false`, means whether the client checks the␣
↪subscription result. If it is true, client exits and return errors when␣
↪subscription is failure.
    buffersize: The default value is `10`, means the size of the memory queue sent by␣
↪the client to the local Hub. If found exception, the client will exit and lose␣
↪messages.
logger: Logger configuration
  path: The default is `empty` (none configuration), that is, it does not write to␣
↪the file. If the path is specified, it writes to the file.
  level: The default value is `info`, log level, support `debug``info``warn` and␣
↪`error`.
  format: The default value is `text`, log print format, support `text` and `json`.
  age:
    max: The default value is `15`, means maximum number of days the log file is kept.
  size:
    max: The default value is `50`, log file size limit, default unit is `MB`.
```

(continues on next page)

(continued from previous page)

```
  backup:
    max: The default value is `15`, the maximum number of log files to keep.
```

6.8 baetyl-timer Configuration

```
hub: Hub configuration
  address: The address for the client to connect with the Hub.
  username: The username for the client to connect with the Hub.
  password: The password for the client to connect with the Hub.
  clientid: The client id for the client to connect with the Hub.
timer: timer configuration
  interval: Timing interval
publish:
  topic: The message topic published to the Hub.
  payload: The payload data, for example
    id: 1
logger: Logger configuration
  path: The default is `empty` (none configuration), that is, it does not write to␣
↪the file. If the path is specified, it writes to the file.
  level: The default value is `info`, log level, support `debug``info``warn` and␣
↪`error`.
```

CHAPTER 7

Device connect to Baetyl with Hub service

Statement:

- The device system used in this test is Ubuntu 18.04
- MQTT.fx and MQTTBox are MQTT Clients in this test, which *MQTT.fx* used for TCP and SSL connection test and *MQTTBox* used for WS (Websocket) connection test.
- The hub service image used is the official image published in the Baetyl Cloud Management Suite: `hub.baidubce.com/baetyl/baetyl-hub`
- You can also compile the required Hub service image by using Baetyl source code. Please see *How to build image from source code*

The complete configuration reference for Hub Module Configuration.

NOTEDarwin can install Baetyl by using Baetyl source code. Please see *How to build image from source code*.

7.1 Workflow

- Step 1: Install Baetyl and its example configuration, more details please refer to *How-to-quick-install-Baetyl*
- Step 2: Modify the configuration according to the usage requirements, and then execute `sudo systemctl start baetyl` to start the Baetyl in Docker container mode, or execute `sudo systemctl restart baetyl` to restart the Baetyl. Then execute the command `sudo systemctl status baetyl` to check whether baetyl is running.
- Step 3: Configure the MQTT Client according to the connection protocol selected.
  - If TCP protocol was selected, you only need to configure the username and password(see the configuration option username and password of principals) and fill in the corresponding port.
  - If SSL protocol was selected, username, private key, certificate and CA should be need. then fill in the corresponding port;
  - If WS protocol was selected, you only need to configure the username, password, and corresponding port.

- Step 4: If all the above steps are normal and operations are correct, you can check the connection status through the log of Baetyl or MQTT Client.

7.2 Connection Test

If the Baetyl's example configuration is installed according to `Step 1`, to modify the configuration of the application and Hub service.

7.2.1 Baetyl Application Configuration

If the official installation method is used, replace the Baetyl application configuration with the following configuration:

```
# /usr/local/var/db/baetyl/application.yml
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub
    replica: 1
    ports:
      - 1883:1883
      - 8883:8883
      - 8080:8080
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-cert
        path: var/db/baetyl/cert
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
volumes:
  - name: localhub-conf
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-cert
    path: var/db/baetyl/localhub-cert-only-for-test
  - name: localhub-log
    path: var/db/baetyl/localhub-log
```

Replace the configuration of the Baetyl Hub service with the following configuration:

```
# /usr/local/var/db/baetyl/localhub-conf/service.yml
listen:
  - tcp://0.0.0.0:1883
  - ssl://0.0.0.0:8883
  - ws://0.0.0.0:8080/mqtt
certificate:
  ca: var/db/baetyl/cert/ca.pem
  cert: var/db/baetyl/cert/server.pem
  key: var/db/baetyl/cert/server.key
```

(continues on next page)

(continued from previous page)

```
principals:
  - username: two-way-tls
    permissions:
      - action: 'pub'
        permit: ['tls/#']
      - action: 'sub'
        permit: ['tls/#']
  - username: test
    password: hahaha
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']
subscriptions:
  - source:
      topic: 't'
    target:
      topic: 't/topic'
logger:
  path: var/log/baetyl/service.log
  level: 'debug'
```

7.2.2 Baetyl Startup

According to `Step 2`, execute `sudo systemctl start baetyl` to start Baetyl in Docker mode and then execute the command `sudo systemctl status baetyl` to check whether baetyl is running. The normal situation is shown as below.

```
baetyl@baetyl:~$ sudo systemctl status baetyl
● baetyl.service - Baetyl
   Loaded: loaded (/lib/systemd/system/baetyl.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2019-09-23 11:31:57 CST; 5h 55min ago
 Main PID: 997 (baetyl)
    Tasks: 10 (limit: 4623)
   CGroup: /system.slice/baetyl.service
           └─997 /usr/local/bin/baetyl start

9月 23 11:31:57 baetyl systemd[1]: Started Baetyl.
```

Baetyl status

NOTEDarwin can install Baetyl by using Baetyl source code, and excute `sudo baetyl start` to start the Baetyl in Docker container mode.

Look at the log of the Baetyl master by executing `sudo tail -f /usr/local/var/log/baetyl/baetyl.log` as shown below:

```
baetyl@baetyl:~$ tail -f /usr/local/var/log/baetyl/baetyl.log
time="2019-09-23T17:55:39+08:00" level=debug msg="instance (localhub) not found" baetyl=master engine=docker service=localhub
time="2019-09-23T17:55:42+08:00" level=debug msg="container (bc82f4ade99e:localhub) started" baetyl=master engine=docker
time="2019-09-23T17:55:42+08:00" level=info msg="instance started" baetyl=master cid=bc82f4ade99e engine=docker instance=localhub service=localhub
time="2019-09-23T17:55:42+08:00" level=info msg="service (localhub) started" baetyl=master
time="2019-09-23T17:55:42+08:00" level=info msg="app version (v0) committed" baetyl=master
time="2019-09-23T17:55:42+08:00" level=info msg="services started" baetyl=master
time="2019-09-23T17:56:42+08:00" level=info msg="[PUT] /v1/services/localhub/instances/localhub/report" api=server baetyl=master
time="2019-09-23T17:56:42+08:00" level=info msg="[PUT] /v1/services/localhub/instances/localhub/report" api=server baetyl=master
time="2019-09-23T17:57:42+08:00" level=info msg="[PUT] /v1/services/localhub/instances/localhub/report" api=server baetyl=master
time="2019-09-23T17:57:42+08:00" level=info msg="[PUT] /v1/services/localhub/instances/localhub/report" api=server baetyl=master
```

Baetyl startup

As you can see, the image of Hub service has been loaded after Baetyl starts up normally. Alternatively, you can use `docker ps` command to check which docker container is currently running.

```
baetyl@baetyl:~$ docker ps
CONTAINER ID        IMAGE                                       COMMAND             CREATED             STATUS              PORTS
                    NAMES
bc82f4ade99e        hub.baidubce.com/baetyl/baetyl-hub:latest   "baetyl-hub"        6 minutes ago       Up 6 minutes        0.0.0.0:1883->1883/tcp, 0.0.0.0:8080->8080
/tcp, 0.0.0.0:8883->8883/tcp   localhub
```

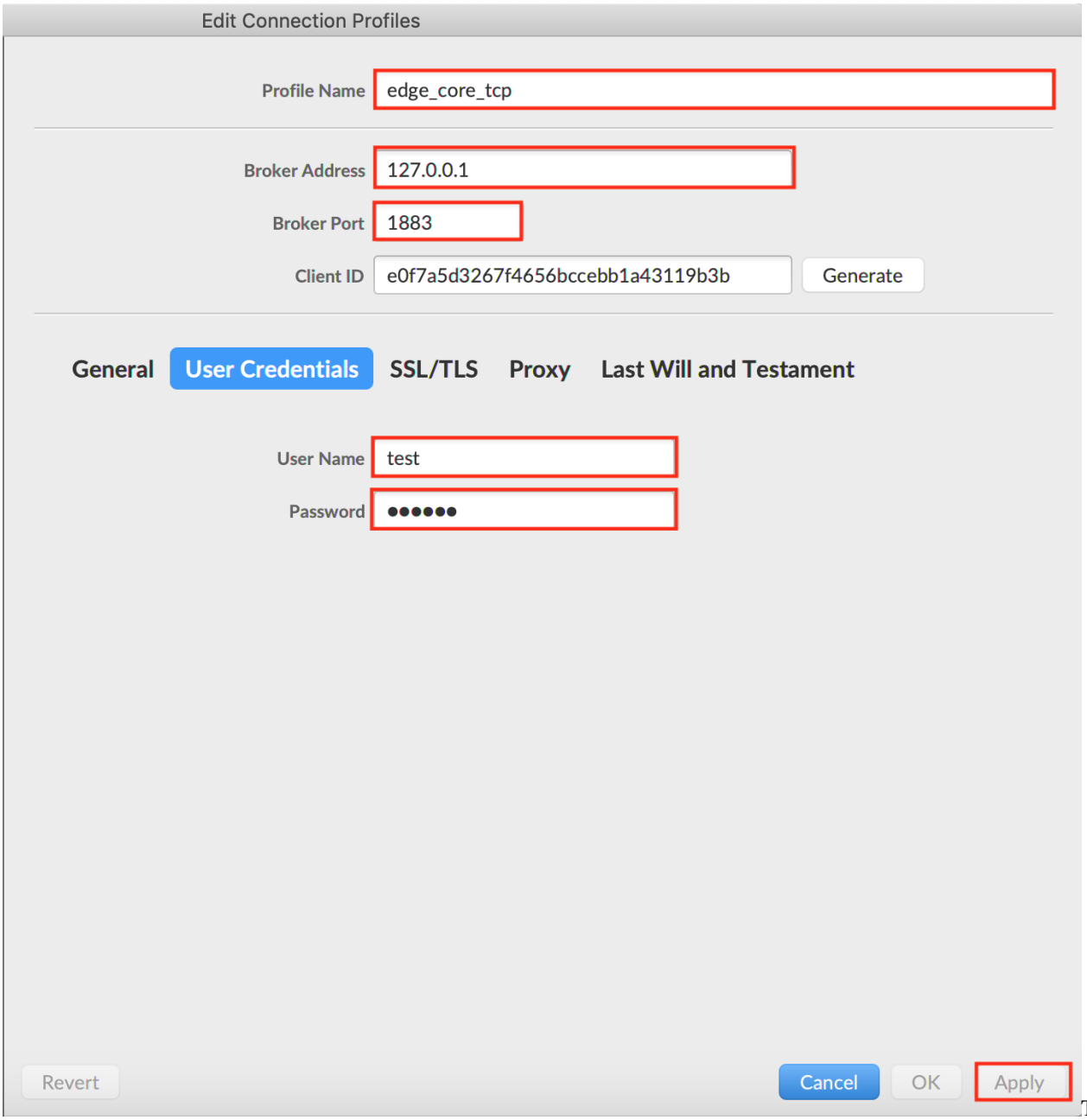
docker ps

Container mode requires port mapping, allowing external access to the container, the configuration item is the `ports` field in the main program configuration file.

As mentioned above, when the Hub Module starts, it will open ports 1883, 8883 and 8080 at the same time, which are used for TCP, SSL, WS (Websocket) protocol. Then we will use MQTTBox and MQTT.fx as MQTT client to check the connection between MQTT client and Baetyl.

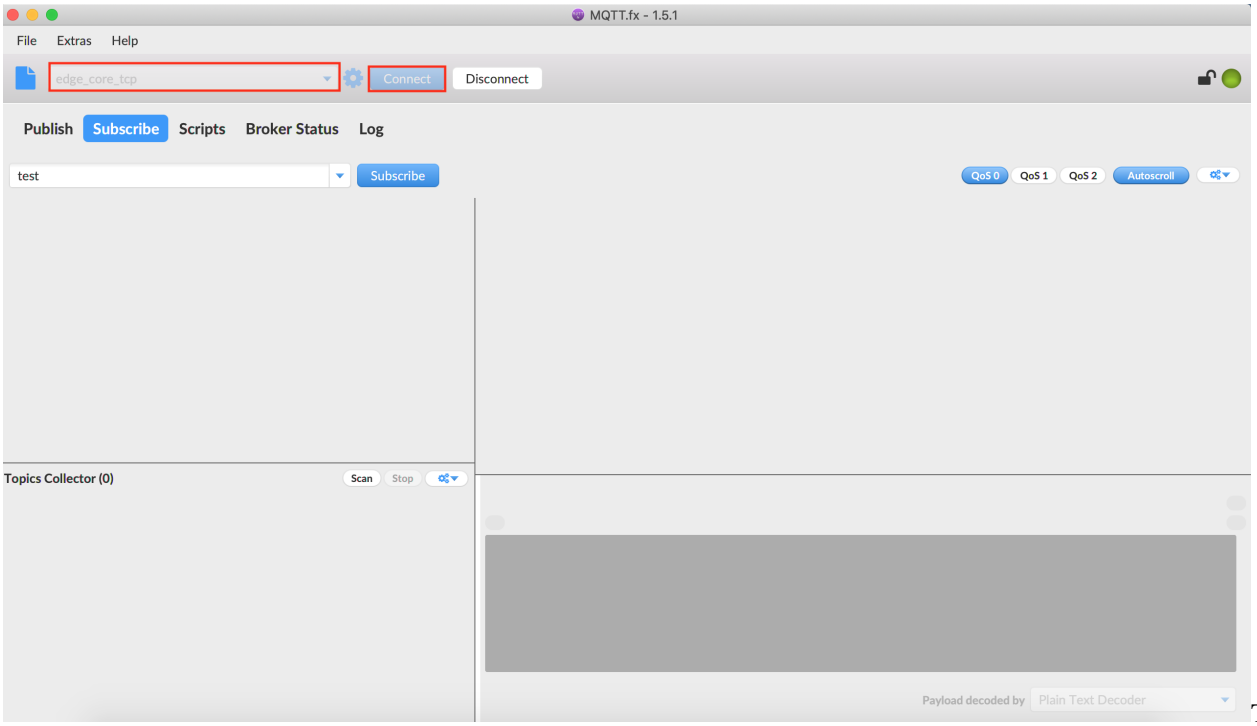
TCP Connection Test

Startup MQTT.fx, enter the `Edit Connection Profiles` page, fill in the `Profile Name`, `Broker Address` and `Port` according to the connection configuration of Baetyl Hub service, and then configure the `username` & `password` in User Credentials according to the `principals` configuration. Then click `Apply` button to complete the connection configuration of MQTT.fx with TCP protocol.



TCP connection configuration

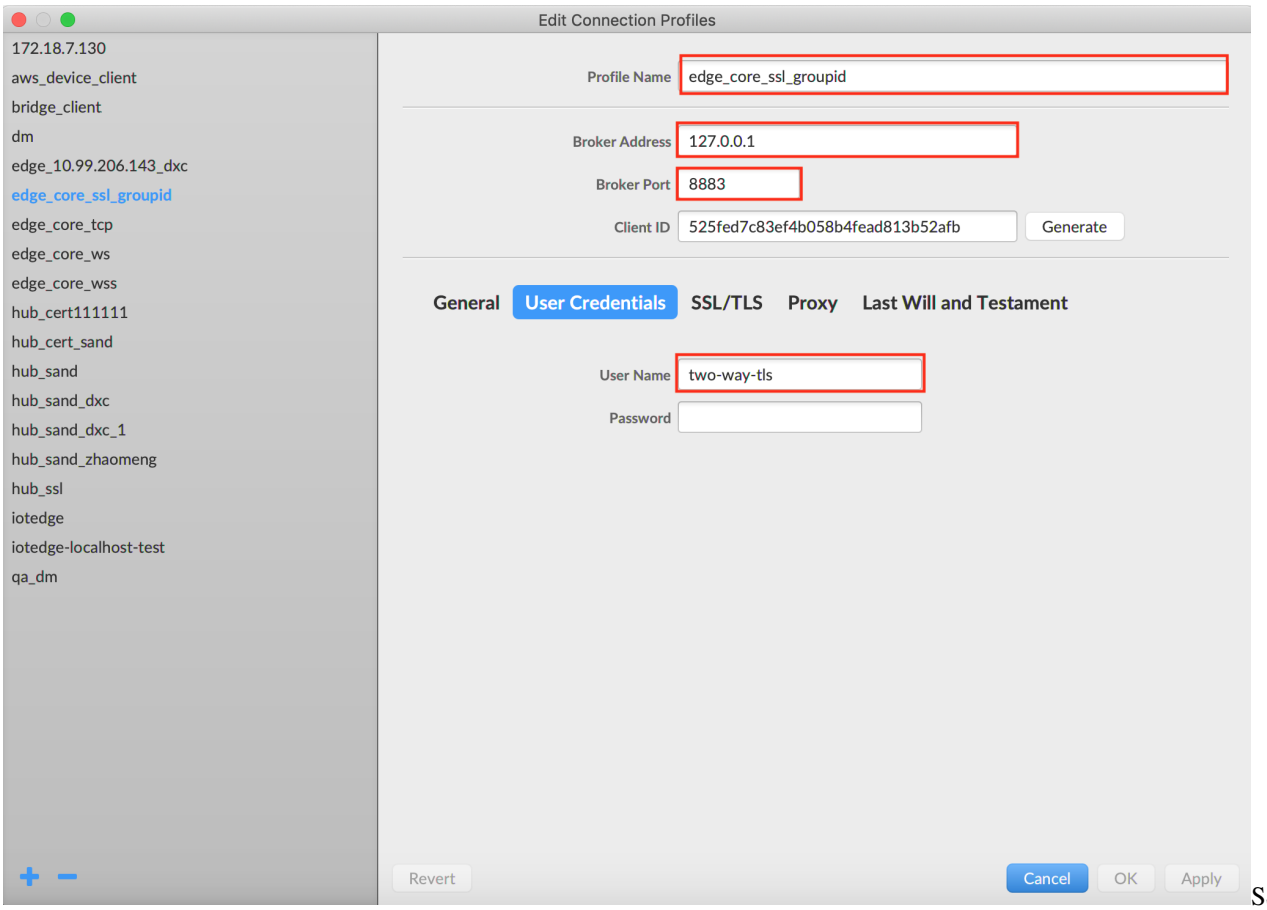
Then close the configuration page, select the Profile Name configured, then click `Connect` button, if the connection configuration information matches the `principals` configuration of Baetyl Hub service, you can see the connection success flag which as shown below.



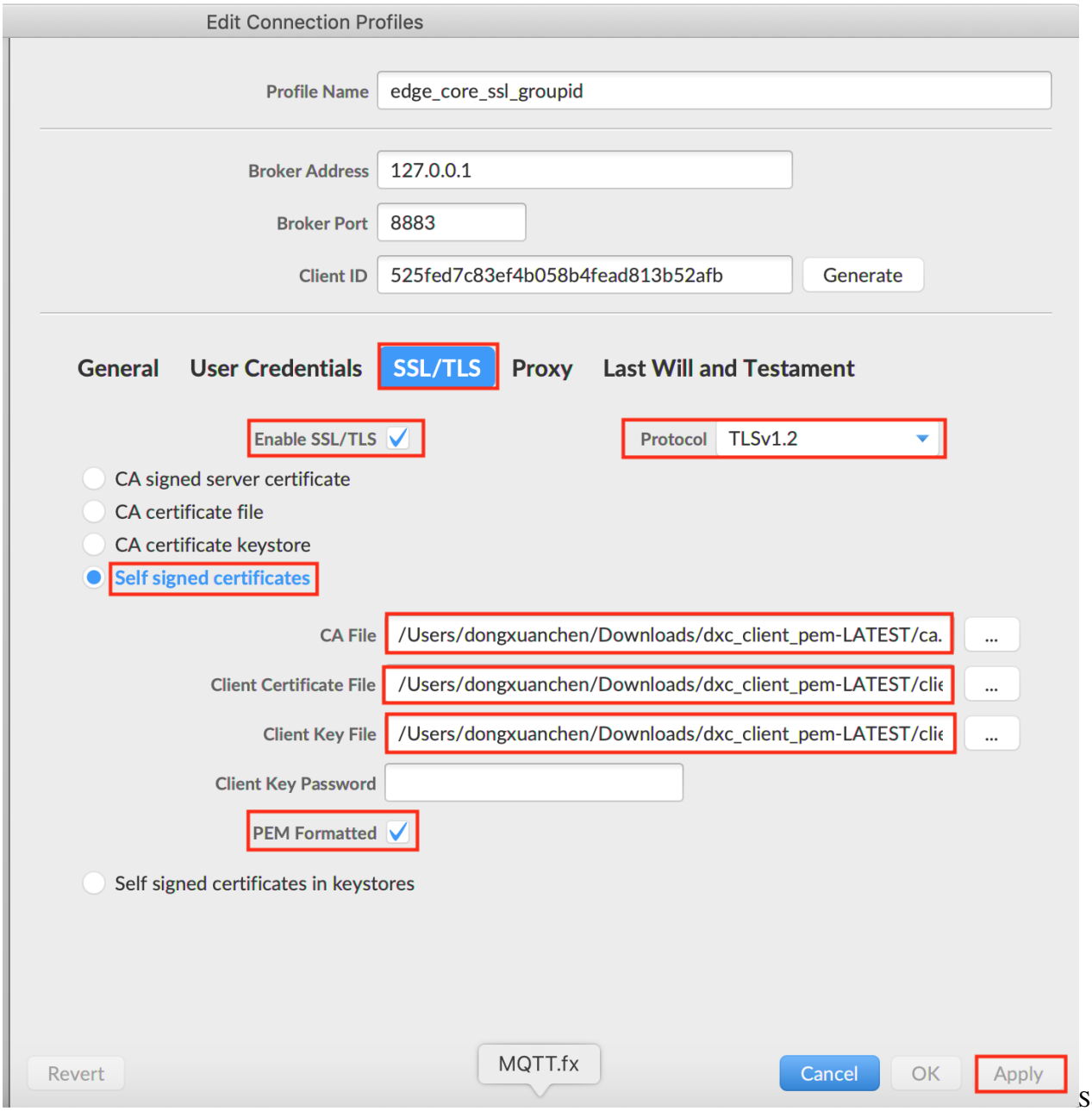
TCP connection success

SSL Connection Test

Startup MQTT.fx and enter the Edit Connection Profiles page. Similar to the TCP connection configuration, fill in the profile name, broker address, and port. For SSL protocol, you need to fill in the username in `User Credentials` and configure SSL/TLS option as shown below. Then click the `Apply` button to complete the connection configuration of MQTT.fx in SSL connection method.

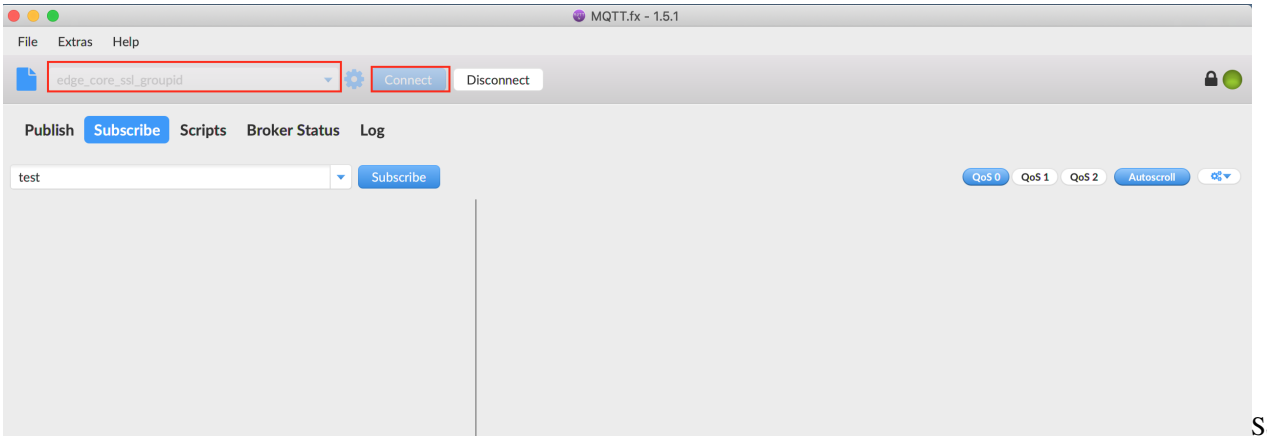


SSL connection configuration1



SSL connection configuration2

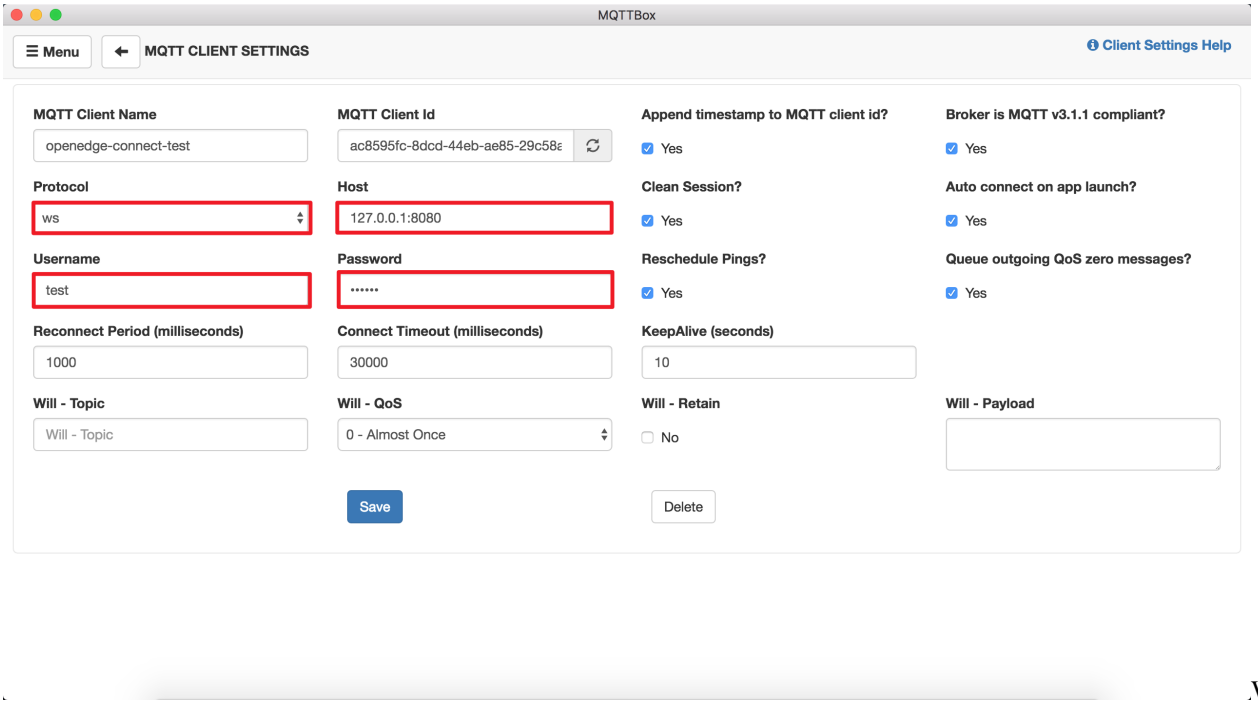
Then close the configuration page, select the Profile Name configured, then click `Connect` button, if the connection configuration information matches the `principals` configuration of Baetyl Hub service, you can see the connection success flag which as shown below.



SSL connection success

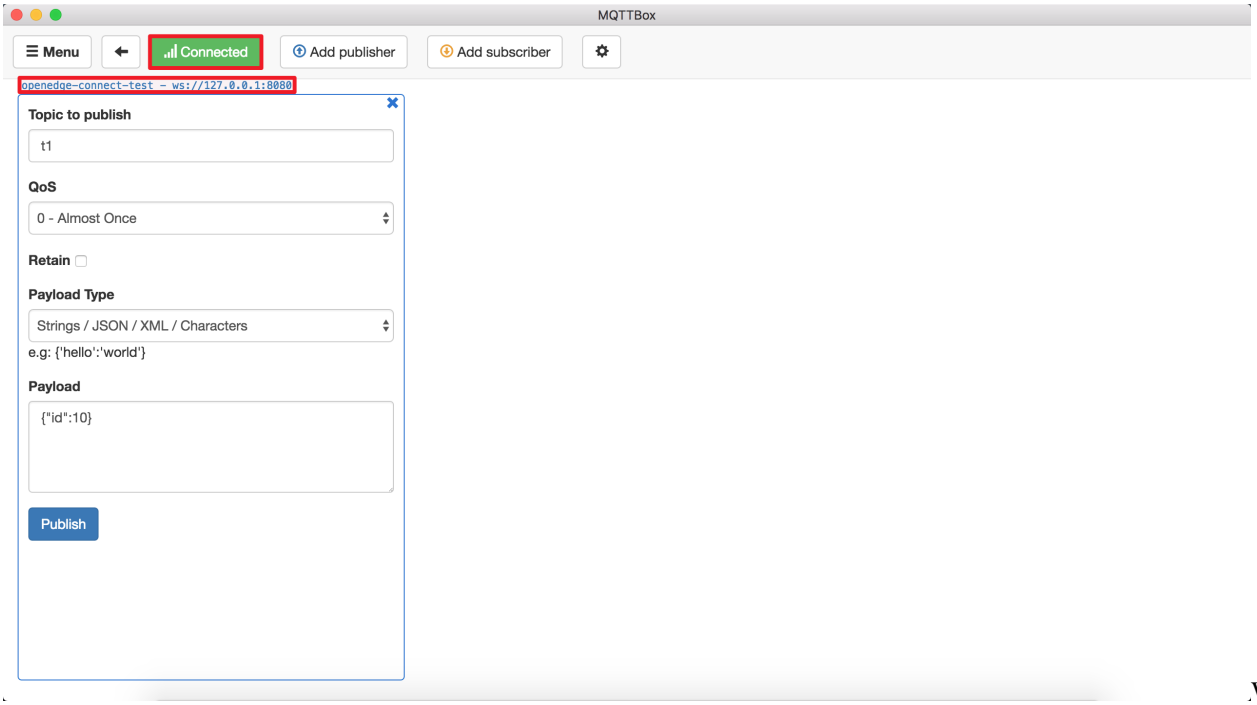
WS (Websocket) Connection Test

Startup MQTTBox, enter the Client creation page, select the `ws` protocol, configure the broker address and port according to the Baetyl Hub service, fill in the username and password according to the `principals` configuration option, and click the `save` button. Then complete the connection configuration of MQTTBox in WS connection method which as shown below.



WSWebsocketcon configuration

Once the above operation is correct, you can see the sign of successful connection with Baetyl Hub in MQTTBox, which is shown in the figure as below.



.WSWebsocketcon success

In summary, we successfully completed the connection test for the Baetyl Hub service through MQTT.fx and MQTTBox. In addition, we can also write test scripts to connect to Baetyl Hub through Paho MQTT. For details, please refer to *Related Resources Download*.

CHAPTER 8

Message transferring among devices with Local Hub Service

Statement

- The operating system as mentioned in this document is Ubuntu 18.04.
- The MQTT client toolkit as mentioned in this document is MQTTBox.

NOTEDarwin can install Baetyl by using Baetyl source code. Please see *How to build image from source code*.

Different from Device connect to Baetyl with Hub service, if you want to transfer MQTT messages among multiple MQTT clients, you need to configure the connect information, topic permission, and router rules. More detailed configuration of Hub service, please refer to Hub service configuration.

This document uses the TCP connection method as an example to test the message routing and forwarding capabilities of the Hub service.

8.1 Workflow

- Step 1: Install Baetyl and its example configuration, more details please refer to *How-to-quick-install-Baetyl*
- Step 2: Modify the configuration according to the usage requirements, and then execute `sudo systemctl start baetyl` to start the Baetyl in Docker container mode, or execute `sudo systemctl restart baetyl` to restart the Baetyl. Then execute the command `sudo systemctl status baetyl` to check whether baetyl is running.
- Step 3MQTTBox connect to Hub Service by TCP connection method, more detailed contents please refer to Device connect to Baetyl with Hub service.
  - If connect successfully, then subscribe the MQTT topic due to the configuration of Hub Service.
  - If connect unsuccessfully, then retry `Step 3` operation until it connect successfully.
- Step 4Check the publishing and receiving messages via MQTTBox.

8.2 Message Routing Test

The Baetyl application configuration is replaced with the following configuration:

```
# /usr/local/var/db/baetyl/application.yml
version: V2
services:
  - name: hub
    image: 'hub.baidubce.com/baetyl/baetyl-hub'
    replica: 1
    ports:
      - '1883:1883'
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub_data
        path: var/db/baetyl/data
      - name: log-V1
        path: var/log/baetyl
volumes:
  - name: localhub-conf
    path: var/db/baetyl/localhub-conf/V1
  - name: log-V1
    path: var/db/baetyl/log
  - name: localhub_data
    path: var/db/baetyl/localhub_data
```

The Baetyl Hub service configuration is replaced with the following configuration:

```
# /usr/local/var/db/baetyl/localhub-conf/service.yml
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: 'test'
    password: 'hahaha'
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']
subscriptions:
  - source:
      topic: 't'
    target:
      topic: 't/topic'
logger:
  path: var/log/baetyl/service.log
  level: 'debug'
```

As configured above, message routing rules depends on the `subscriptions` configuration item, which means that messages published to the topic `t` will be forwarded to all devices(users, or mqtt clients) that subscribe the topic `t/topic`.

***NOTE**: In the above configuration, the permitted topics which are configured in `permit` item support the + and # wildcards configuration. More detailed contents of + and # wildcards will be explained as follows.*

 # wildcard

For MQTT protocol, the number sign(# U+0023) is a wildcard character that matches any number of levels within a topic. The multi-level wildcard represents the parent and any number of child levels. The multi-level wildcard character MUST be specified either on its own or following a topic level separator(/ U+002F). In either case it MUST be the last character specified in the topic.

For example, the configuration of `permit` item of `sub` action is `sport/tennis/player1/#`, it would receive messages published using these topic names:

- `sport/tennis/player1`
- `sport/tennis/player1/ranking`
- `sport/tennis/player1/score/wimbledon`

Besides, topic `sport/#` also matches the singular `sport`, since # includes the parent level.

For Baetyl, if the topic # is configured in the `permit` item list(whether `pub` action or `sub` action), there is no need to configure any other topics. And the specified account(depends on `username/password`) will have permission to all legal topics of MQTT protocol.

+ wildcard

As described in the MQTT protocol, the plus sign(+ U+002B) is a wildcard character that matches only one topic level. The single-level wildcard can be used at any level in the topic, including first and last levels. Where it is used it MUST occupy an entire level of the topic. It can be used at more than one level in the topic and can be used in conjunction with the multi-level wildcard.

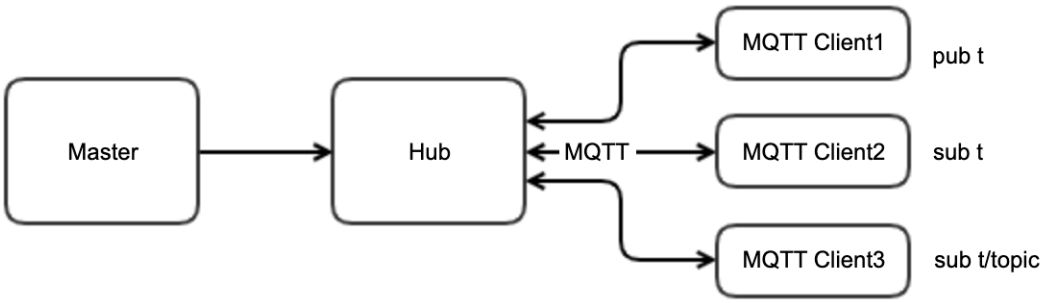
For example, topic `sport/tennis/+` matches `sport/tennis/player1` and `sport/tennis/player2`, but not `sport/tennis/player1/ranking`. Also, because the single-level wildcard matches only a single level, `sport/+` does not match `sport` but it does match `sport/`.

For Baetyl, if the topic + is configured in the `permit` item list(whether `pub` action or `sub` action), the specified account(depends on `username/password`) will have permission to all single-level legal topics of MQTT protocol.

NOTE: For MQTT protocol, wildcard **ONLY** can be used in Topic Filter(`sub` action), and **MUST NOT** be used in Topic Name(`pub` action). But in the design of Baetyl, in order to enhance the flexibility of the topic permissions configuration, wildcard configured in `permit` item(whether in `pub` action or `sub` action) is valid, as long as the topic of the published or subscribed meets the requirements of MQTT protocol is ok. In particular, wildcards (# and +) policies are recommended for developers who need to configure a large number of publish and subscribe topics in the `principals` configuration.

8.2.1 Message Transfer Test Among Devices

The message transferring and routing workflow among devices are as follows:



Message

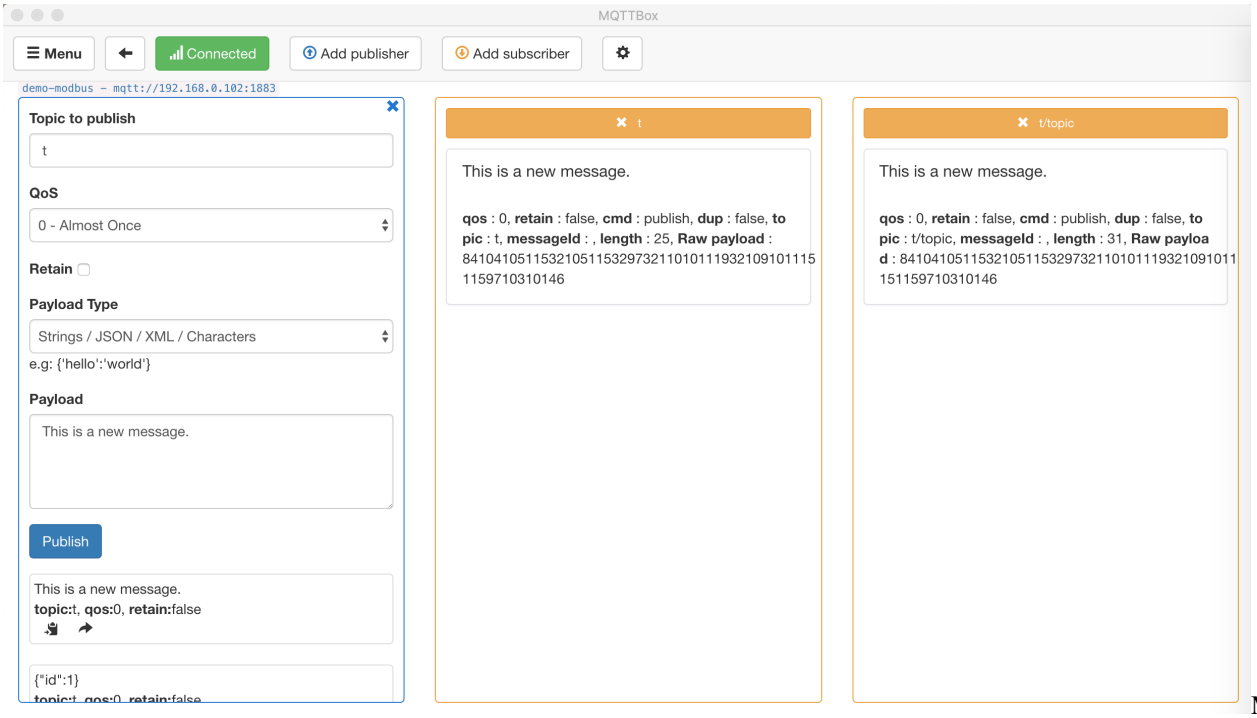
transfer test among devices

Specifically, as shown in the above figure, **client1**, **client2**, and **client3** respectively establish a connection to Baetyl with Hub Service, **client1** has the permission to publish messages to the topic `t`, and **client2** and **client3** respectively have the permission to subscribe topic `t` and `t/topic`.

Once the connection to Baetyl for the above three clients with Hub Service is established, as the configuration of the above three clients, **client2** and **client3** will respectively get the message from **client1** published to the topic `t` to Hub Service.

In particular, **client1**, **client2**, and **client3** can be combined into one client, and the new client will have the permission to publish messages to the topic `t`, with permissions to subscribe messages to the topic `t` and `t/topic`. Here, using MQTTBox as the new client, click the `Add subscriber` button to subscribe the topic `t` and `t/topic`.

Then clicks the `Publish` button to publish message with the payload `This is a new message.` and with the topic `t` to Hub service, you will find this message is received by MQTTBox with the subscribed topics `t` and `t/topic`. More detailed contents are as below.



MQTTBox received message successfully

In summary, the message forwarding and routing test between devices based on Hub service is completed through MQTTBox.

CHAPTER 9

Message handling with Local Function Service

Statement

- The operating system as mentioned in this document is Ubuntu 18.04.
- The version of runtime is Python3.6, and for Python2.7, configuration is the same except fot the language difference when coding the scripts
- The MQTT client toolkit as mentioned in this document is MQTTBox.
- The docker image used in this document is compiled from the Baetyl source code. More detailed contents please refer to *Build Baetyl from source*.
- In this article, the service created based on the Hub service is called Hub service.

NOTEDarwin can install Baetyl by using Baetyl source code. Please see *How to build image from source code*.

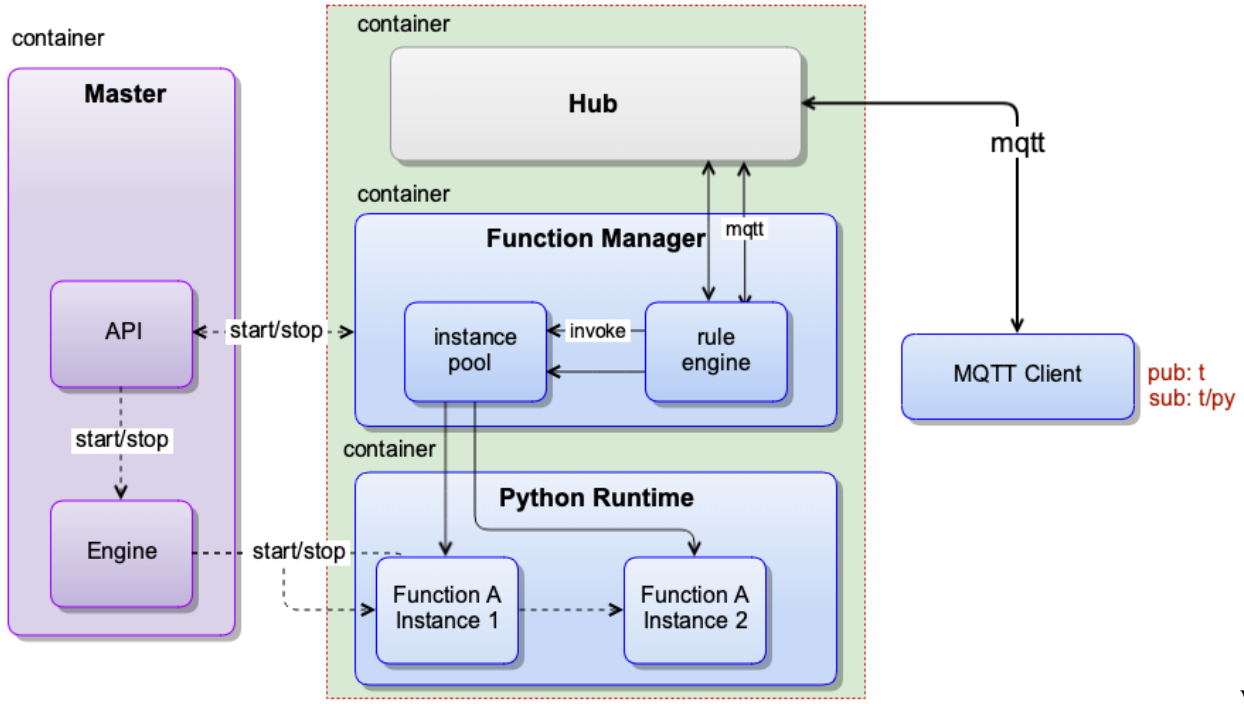
Different from the Hub service to transfer message among devices(mqtt clients), this document describes the message handling with Local Function Manager service(also include Hub service and Python3.6 runtime service). In the document, Hub service is used to establish connection between Baetyl and mqtt client, Python3.6 runtime service is used to handle MQTT messages, and the Local Function Manager service is used to combine Hub service with Python3.6 runtime service with message context.

This document will take the TCP connection method as an example to show the message handling, calculation and forwarding with Local Function Manager service.

9.1 Workflow

- Step 1: Install Baetyl and its example configuration, more details please refer to *How-to-quick-install-Baetyl*
- Step 2: Modify the configuration according to the usage requirements, and then execute `sudo systemctl start baetyl` to start the Baetyl in Docker container mode, or execute `sudo systemctl restart baetyl` to restart the Baetyl. Then execute the command `sudo systemctl status baetyl` to check whether baetyl is running.

- Step 3MQTTBox connect to Hub Service by TCP connection method, more detailed contents please refer to Device connect to Baetyl with Hub service
  - If connect successfully, then subscribe the MQTT topic due to the configuration of Hub Service, and observe the log of Baetyl.
    * If the Baetyl's log shows that the Python Runtime Service has been started, it indicates that the published message was handled by the specified function.
    * If the Baetyl's log shows that the Python Runtime Service has not been started, then retry it until the Python Runtime Service has been started.
  - If connect unsuccessfully, then retry `Step 3` operation until it connect successfully
- Step 4Check the publishing and receiving messages via MQTTBox.



Workflow of using Local Function Manager Service to handle MQTT messages

9.2 Message Handling Test

If the Baetyl's example configuration is installed according to `Step 1`, to modify the configuration of the application, Hub service and function services.

Change the Baetyl application configuration to the following configuration:

```
# /usr/local/var/db/baetyl/application.yml
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub
    replica: 1
    ports:
      - 1883:1883
```

(continues on next page)

(continued from previous page)

```
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
  - name: function-manager
    image: hub.baidubce.com/baetyl/baetyl-function-manager
    replica: 1
    mounts:
      - name: function-manager-conf
        path: etc/baetyl
        readonly: true
      - name: function-manager-log
        path: var/log/baetyl
  - name: function-python27-sayhi
    image: hub.baidubce.com/baetyl/baetyl-function-python27
    replica: 0
    mounts:
      - name: function-sayhi-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayhi-code
        path: var/db/baetyl/function-sayhi
        readonly: true
  - name: function-python36-sayhi
    image: hub.baidubce.com/baetyl/baetyl-function-python36
    replica: 0
    mounts:
      - name: function-sayhi-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayhi-code
        path: var/db/baetyl/function-sayhi
        readonly: true
  - name: function-node85-sayhi
    image: hub.baidubce.com/baetyl/baetyl-function-node85
    replica: 0
    mounts:
      - name: function-sayjs-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayjs-code
        path: var/db/baetyl/function-sayhi
        readonly: true
  - name: function-sql-filter
    image: hub.baidubce.com/baetyl/baetyl-function-sql
    replica: 0
    mounts:
      - name: function-filter-conf
        path: etc/baetyl
        readonly: true
volumes:
  # hub
  - name: localhub-conf
```

(continues on next page)

(continued from previous page)

```
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-cert
    path: var/db/baetyl/localhub-cert-only-for-test
  - name: localhub-log
    path: var/db/baetyl/localhub-log
  # function
  - name: function-manager-conf
    path: var/db/baetyl/function-manager-conf
  - name: function-manager-log
    path: var/db/baetyl/function-manager-log
  - name: function-sayhi-conf
    path: var/db/baetyl/function-sayhi-conf
  - name: function-sayhi-code
    path: var/db/baetyl/function-sayhi-code
  - name: function-sayjs-conf
    path: var/db/baetyl/function-sayjs-conf
  - name: function-sayjs-code
    path: var/db/baetyl/function-sayjs-code
  - name: function-filter-conf
    path: var/db/baetyl/function-filter-conf
```

Change the Baetyl Hub service configuration to the following configuration:

```
# /usr/local/var/db/baetyl/localhub-conf/service.yml
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: test
    password: hahaha
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']
subscriptions:
  - source:
      topic: 't'
    target:
      topic: 't/topic'
logger:
  path: var/log/baetyl/service.log
  level: "debug"
```

The configuration of the Baetyl local function services do not need to be modified. The specific configuration is as follows:

```
# /usr/local/var/db/baetyl/function-manager-conf/service.yml
hub:
  address: tcp://localhub:1883
  username: test
  password: hahaha
rules:
  - clientid: func-python27-sayhi-1
    subscribe:
```

(continues on next page)

(continued from previous page)

```
      topic: t
    function:
      name: python27-sayhi
    publish:
      topic: t/py2hi
  - clientid: func-sql-filter-1
    subscribe:
      topic: t
      qos: 1
    function:
      name: sql-filter
    publish:
      topic: t/sqlfilter
      qos: 1
  - clientid: func-python36-sayhi-1
    subscribe:
      topic: t
    function:
      name: python36-sayhi
    publish:
      topic: t/py3hi
  - clientid: func-node85-sayhi-1
    subscribe:
      topic: t
    function:
      name: node85-sayhi
    publish:
      topic: t/node8hi
functions:
  - name: python27-sayhi
    service: function-python27-sayhi
    instance:
      min: 0
      max: 10
  - name: sql-filter
    service: function-sql-filter
  - name: python36-sayhi
    service: function-python36-sayhi
  - name: node85-sayhi
    service: function-node85-sayhi
logger:
  path: var/log/baetyl/service.log
  level: "debug"

# /usr/local/var/db/baetyl/function-filter-conf/service.yml
functions:
  - name: sql-filter
    handler: 'select qos() as qos, topic() as topic, * where id < 10'

# /usr/local/var/db/baetyl/function-sayhi-conf/service.yml
functions:
  - name: 'python27-sayhi'
    handler: 'index.handler'
    codedir: 'var/db/baetyl/function-sayhi'
  - name: 'python36-sayhi'
    handler: 'index.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

(continues on next page)

(continued from previous page)

```
# /usr/local/var/db/baetyl/function-sayjs-conf/service.yml
functions:
  - name: 'node85-sayhi'
    handler: 'index.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

Python function code does not need to be changed. `/usr/local/var/db/baetyl/function-sayhi-code/index.py` is implemented as follows:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
"""
function to say hi in python
"""

def handler(event, context):
    """
    function handler
    """
    res = {}
    if isinstance(event, dict):
        if "err" in event:
            raise TypeError(event['err'])
        res = event
    elif isinstance(event, bytes):
        res['bytes'] = event.decode("utf-8")

    if 'messageQOS' in context:
        res['messageQOS'] = context['messageQOS']
    if 'messageTopic' in context:
        res['messageTopic'] = context['messageTopic']
    if 'messageTimestamp' in context:
        res['messageTimestamp'] = context['messageTimestamp']
    if 'functionName' in context:
        res['functionName'] = context['functionName']
    if 'functionInvokeID' in context:
        res['functionInvokeID'] = context['functionInvokeID']

    res['Say'] = 'Hello Baetyl'
    return res
```

The Node function code does not need to be changed. `/usr/local/var/db/baetyl/function-sayjs-code/index.js` is implemented as follows:

```
#!/usr/bin/env node

const hasAttr = (obj, attr) => {
    if (obj instanceof Object && !(obj instanceof Array)) {
        if (obj[attr] != undefined) {
            return true;
        }
    }
    return false;
};
```

(continues on next page)

(continued from previous page)

```
const passParameters = (event, context) => {
    if (hasAttr(context, 'messageQOS')) {
        event['messageQOS'] = context['messageQOS'];
    }
    if (hasAttr(context, 'messageTopic')) {
        event['messageTopic'] = context['messageTopic'];
    }
    if (hasAttr(context, 'messageTimestamp')) {
        event['messageTimestamp'] = context['messageTimestamp'];
    }
    if (hasAttr(context, 'functionName')) {
        event['functionName'] = context['functionName'];
    }
    if (hasAttr(context, 'functionInvokeID')) {
        event['functionInvokeID'] = context['functionInvokeID'];
    }
};

exports.handler = (event, context, callback) => {
    // support Buffer & json object
    if (Buffer.isBuffer(event)) {
        const message = event.toString();
        event = {}
        event["bytes"] = message;
    }
    else if("err" in event) {
        return callback(new TypeError(event['err']))
    }

    passParameters(event, context);
    event['Say'] = 'Hello Baetyl'
    callback(null, event);
};
```

As configured above, if the MQTTBox has established a connection with the Hub service based on the above configuration, a message with the topic `t` is sent to the Hub, and the function service will route the message to `python27-sayhi`, `python36-sayhi`, `node85-sayhi` and `sql-filter` functions to process, and messages with topic `t/py2hi`, `t/py3hi`, `t/node8hi`, and `t/sqlfilter` are output separately. At this time, the MQTT client subscribed to the topic `#` will receive these messages, as well as the original message `t` and the message with topic `t/topic` which is renamed by Hub service directly .

***NOTE**: Any function that appears in the `rules` configuration must be configured in the `functions` configuration, otherwise the function runtime instances can not be started normally.*

9.2.1 Baetyl Start

According to `Step 2`, execute `sudo systemctl start baetyl` to start Baetyl in Docker mode and then execute the command `sudo systemctl status baetyl` to check whether baetyl is running.

NOTEDarwin can install Baetyl by using Baetyl source code, and excute `sudo baetyl start` to start the Baetyl in Docker container mode.

Look at the log of the Baetyl master by executing `sudo tail -f -n 40 /usr/local/var/log/baetyl/baetyl.log` as shown below:

```
time="2019-09-23T21:49:08+08:00" level=info msg="mode: docker; grace: 30000000000; pwd: /usr/local; api: unix:///var/run/baetyl.sock" baetyl=master
time="2019-09-23T21:49:08+08:00" level=info msg="engine started" baetyl=master
time="2019-09-23T21:49:08+08:00" level=error msg="no such file or directory" api=server baetyl=master
time="2019-09-23T21:49:08+08:00" level=info msg="server started" baetyl=master
time="2019-09-23T21:49:08+08:00" level=debug msg="network (4c71d71eeab3:baetyl) exists" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="image (hub.baidubce.com/baetyl/baetyl-function-python36) pulled" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="image (hub.baidubce.com/baetyl/baetyl-function-manager) pulled" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="image (hub.baidubce.com/baetyl/baetyl-function-node85) pulled" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="image (hub.baidubce.com/baetyl/baetyl-function-sql) pulled" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="image (hub.baidubce.com/baetyl/baetyl-function-python27) pulled" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="image (hub.baidubce.com/baetyl/baetyl-hub) pulled" baetyl=master engine=docker
time="2019-09-23T21:49:09+08:00" level=debug msg="localhub replica: 1" baetyl=master engine=docker service=localhub
time="2019-09-23T21:49:09+08:00" level=debug msg="instance (localhub) not found" baetyl=master engine=docker service=localhub
time="2019-09-23T21:49:11+08:00" level=debug msg="container (52baddd7a1cf:localhub) started" baetyl=master engine=docker
time="2019-09-23T21:49:11+08:00" level=info msg="instance started" baetyl=master cid=52baddd7a1cf engine=docker instance=localhub service=localhub
time="2019-09-23T21:49:11+08:00" level=info msg="service (localhub) started" baetyl=master
time="2019-09-23T21:49:11+08:00" level=debug msg="function-manager replica: 1" baetyl=master engine=docker service=function-manager
time="2019-09-23T21:49:11+08:00" level=debug msg="instance (function-manager) not found" baetyl=master engine=docker service=function-manager
time="2019-09-23T21:49:14+08:00" level=debug msg="container (db939cb5cdfc:function-manager) started" baetyl=master engine=docker
time="2019-09-23T21:49:14+08:00" level=info msg="instance started" baetyl=master cid=db939cb5cdfc engine=docker instance=function-manager service=function-manager
time="2019-09-23T21:49:14+08:00" level=info msg="service (function-manager) started" baetyl=master
time="2019-09-23T21:49:14+08:00" level=debug msg="function-python27-sayhi replica: 0" baetyl=master engine=docker service=function-python27-sayhi
time="2019-09-23T21:49:14+08:00" level=info msg="service (function-python27-sayhi) started" baetyl=master
time="2019-09-23T21:49:14+08:00" level=debug msg="function-python36-sayhi replica: 0" baetyl=master engine=docker service=function-python36-sayhi
time="2019-09-23T21:49:14+08:00" level=info msg="service (function-python36-sayhi) started" baetyl=master
time="2019-09-23T21:49:14+08:00" level=debug msg="function-node85-sayhi replica: 0" baetyl=master engine=docker service=function-node85-sayhi
time="2019-09-23T21:49:14+08:00" level=info msg="service (function-node85-sayhi) started" baetyl=master
time="2019-09-23T21:49:14+08:00" level=debug msg="function-sql-filter replica: 0" baetyl=master engine=docker service=function-sql-filter
time="2019-09-23T21:49:14+08:00" level=info msg="service (function-sql-filter) started" baetyl=master
time="2019-09-23T21:49:14+08:00" level=info msg="app version (v0) committed" baetyl=master
time="2019-09-23T21:49:14+08:00" level=info msg="services started" baetyl=master
time="2019-09-23T21:50:11+08:00" level=info msg="[PUT] /v1/services/localhub/instances/localhub/report" api=server baetyl=master
```

Baetyl start

Also, we can execute the command `docker ps` to view the list of docker containers currently running.

```
baetyl@baetyl:/usr/local/var$ docker ps
CONTAINER ID        IMAGE                                          COMMAND                  CREATED             STATUS              PORTS
 NAMES
db939cb5cdfc        hub.baidubce.com/baetyl/baetyl-function-manager   "baetyl-function-man…"   4 minutes ago       Up 4 minutes
 function-manager
52baddd7a1cf        hub.baidubce.com/baetyl/baetyl-hub             "baetyl-hub"             4 minutes ago       Up 4 minutes        0.0.0.0:1883->1883/tcp
 localhub
```

View the list of docker containers currently running

After comparison, it is not difficult to find that the Hub service and the function service have been successfully loaded when Baetyl starts. The function runtime instance is not started because the function runtime instance is dynamically started by function service when a message is triggered.

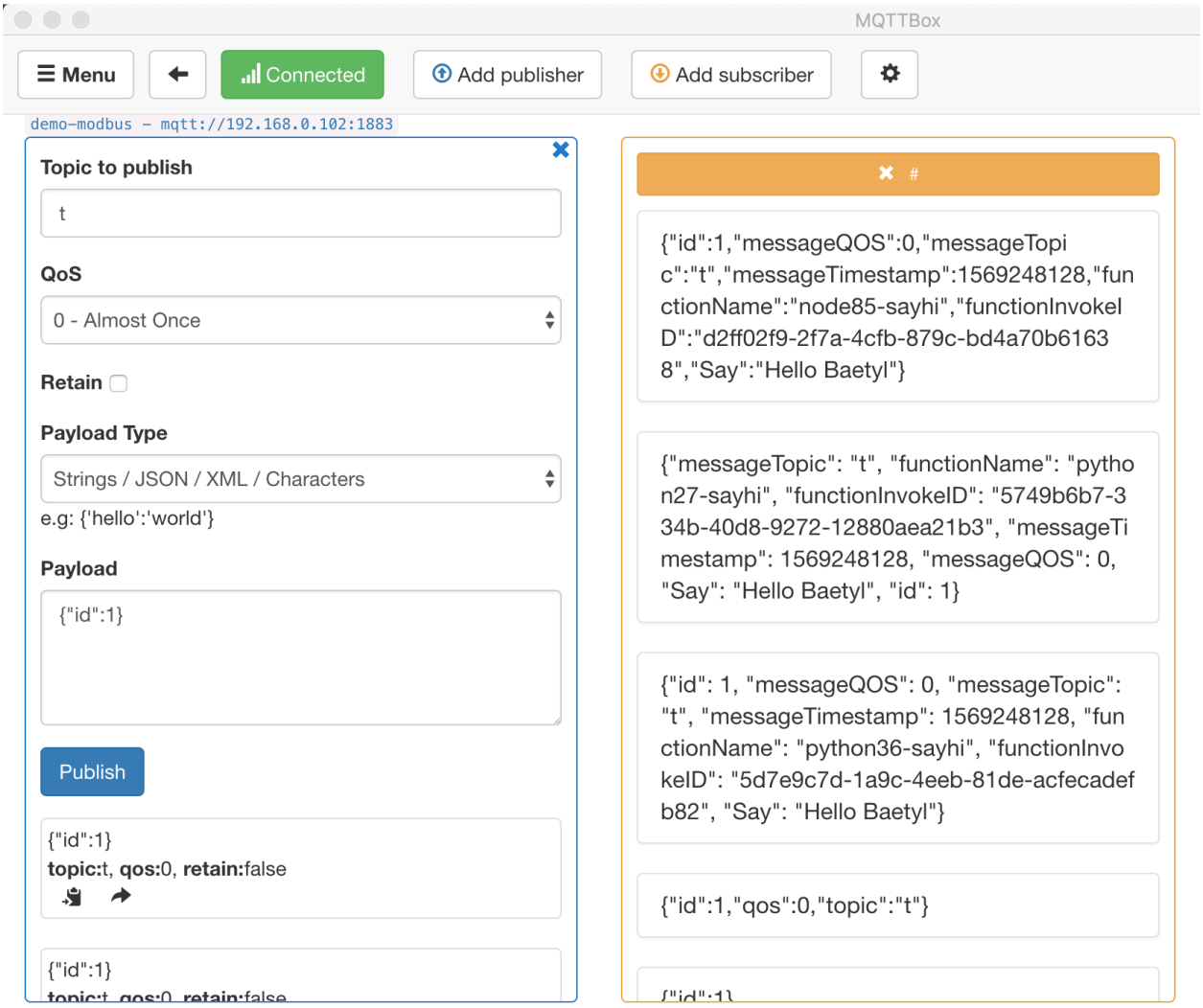
9.2.2 MQTTBox Establish Connection

In this test, we configured the connection information of MQTTBox by TCP connection, and then clicked the `Add subscriber` button to subscribe to the topic `#`, which is used to receive all messages received by the Hub services.

9.2.3 Message Handling Check

By looking at the `/usr/local/var/db/baetyl/function-sayhi-code/index.py` code file, you can see that after receiving a message, the function `handler` will perform a series of processes and return the result. The returned results include some context information, such as `messageTopic`, `functionName`, and so on.

Here, we publish the message `{"id":1}` with the topic `t` to Hub service via MQTTBox, and then observe the receiving messages as follows.



MQTTBox received messages

After sending the message, we quickly execute the command `docker ps` to see the list of the currently running containers. All function runtime service instances are started. The result is shown below.

```
baetyl@baetyl:/usr/local/var$ docker ps
CONTAINER ID        IMAGE                                                COMMAND                  CREATED             STATUS              PORTS
  NAMES
7410649788db        hub.baidubce.com/baetyl/baetyl-function-python36     "function-python36.py"   8 seconds ago       Up 4 seconds
  function-python36-sayhi.python36-sayhi.3
56073223ab5a        hub.baidubce.com/baetyl/baetyl-function-python27     "function-python27.py"   8 seconds ago       Up 3 seconds
  function-python27-sayhi.python27-sayhi.3
5d64c1e02c24        hub.baidubce.com/baetyl/baetyl-function-node85       "function-node85.js"     8 seconds ago       Up 4 seconds
  function-node85-sayhi.node85-sayhi.3
925ab2b2bdda        hub.baidubce.com/baetyl/baetyl-function-sql          "baetyl-function-sql"    14 minutes ago      Up 14 minutes
  function-sql-filter.sql-filter.1
db939cb5cdfc        hub.baidubce.com/baetyl/baetyl-function-manager      "baetyl-function-man…"   32 minutes ago      Up 32 minutes
  function-manager
52baddd7a1cf        hub.baidubce.com/baetyl/baetyl-hub                   "baetyl-hub"             32 minutes ago      Up 32 minutes       0.0.0.0:1883->1883/tcp
  localhub
```

View the list of docker containers

In summary, we simulated the process of local processing of messages through the Hub service and function services. It can be seen that the framework is very suitable to process message flows at edge.

CHAPTER 10

Message Synchronize between Baetyl-Hub and Baidu IoTHub via Baetyl-Remote-MQTT module

Statement

- The operating system as mentioned in this document is Ubuntu18.04.
- It should be installed for Baetyl when you read this document, more details please refer to *Quick-install-Baetyl*
- The MQTT client toolkit which is used to connect to Baidu IoTHub is MQTT.fx
- The MQTT client toolkit which is used to connect to Baetyl-Hub is MQTTBox.
- The hub and Baetyl-Remote-MQTT module images used have published by BIE Cloud Management Suite: `hub.baidubce.com/baetyl/baetyl-hub:latest`hub.baidubce.com/baetyl/baetyl-remote-mqtt:latest
- Docker images compiled from the Baetyl source code also can be used. More detailed contents please refer to *Build Baetyl from source*
- The Remote Hub as mentioned in this document is Baidu IoTHub

The Baetyl-Remote-MQTT module was developed to meet the needs of the IoT scenario. The Baetyl(via Baetyl-Hub module) can synchronize message with Remote Hub services(Baidu IoTHub) via the Baetyl-Remote-MQTT module. That is to say, through the Baetyl-Remote-MQTT module, we can either subscribe the message from Remote Hub and publish it to the Baetyl-Hub module or subscribe the message from Baetyl-Hub module and publish it to Remote Hub service. The configuration of Baetyl-Remote-MQTT module can refer to Baetyl-Remote-MQTT module Configuration.

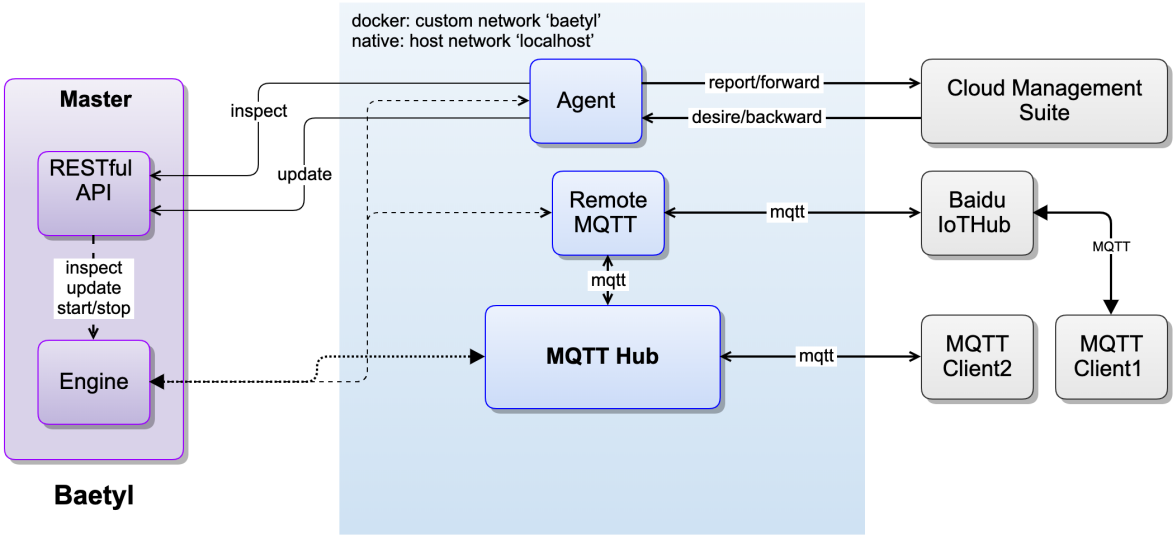
10.1 Workflow

- Step 1Create device(MQTT client) connection info(include `endpoint`, `user`, `principal`, `policy`, etc.) via Baidu IoTHub.
- Step 2Select MQTT.fx as the MQTT client that used to connect to Baidu IoTHub.
  - If connect successfully, then do the following next.

  - If connect unsuccessfully, then retry it until it connect successfully. More detailed contents can refer to How to connect to Baidu IoTHub via MQTT.fx

- Step 3Startup Baetyl in docker container mode, and observe the log of Baetyl.
  - If the Baetyl-Hub module and Baetyl-Remote-MQTT module start successfully, then do the following next.
  - If the Baetyl-Hub module and Baetyl-Remote-MQTT module start unsuccessfully, then retry `Step 3` until they start successfully.
- Step 4Select MQTTBox as the MQTT client that used to connect to the Baetyl-Hub.
  - If connect successfully, then do the following next.
  - If connect unsuccessfully, then retry `Step 4` until it connect successfully.
- Step 5Due to the configuration of Baetyl-Remote-MQTT module, using MQTTBox publish message to the specified topic, and observing the receiving message via MQTT.fx. Similarly, using MQTT.fx publish message to the specified topic, and observing the receiving message via MQTTBox.
- Step 6If both parties in `Step 5` can receive the message content posted by the other one, it indicates the Remote function test passes smoothly.

The workflow diagram are as follows.



using Baetyl-Remote-MQTT module to synchronize message

10.2 Message Synchronize via Baetyl-Remote-MQTT module

Configuration file location for the Baetyl main program is: `var/db/baetyl/application.yml`.

The configuration of Baetyl Master are as follows:

```
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub:latest
    replica: 1
```

(continues on next page)

(continued from previous page)

```
    ports:
      - 1883:1883
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
   - name: remote-iothub
    image: hub.baidubce.com/baetyl/baetyl-remote-mqtt:latest
    replica: 1
    mounts:
      - name: remote-iothub-conf
        path: etc/baetyl
        readonly: true
      - name: remote-iothub-cert
        path: var/db/baetyl/cert
        readonly: true
      - name: remote-iothub-log
        path: var/log/baetyl
volumes:
  # hub
  - name: localhub-conf
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-log
    path: var/db/baetyl/localhub-log
  # remote mqtt
  - name: remote-iothub-conf
    path: var/db/baetyl/remote-iothub-conf
  - name: remote-iothub-cert
    path: var/db/baetyl/remote-iothub-cert
  - name: remote-iothub-log
    path: var/db/baetyl/remote-iothub-log
```

Configuration file location for Baetyl-Hub module is: `var/db/baetyl/localhub-conf/service.yml`.

The configuration of Baetyl-Hub module is as follow:

```
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: test
    password: hahaha
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']
logger:
  path: var/log/baetyl/localhub-service.log
  level: "debug"
```

Configuration file location for Baetyl-Remote-MQTT module is: `var/db/baetyl/remote-iothub-conf/service.yml`.

The configuration of Baetyl-Remote-MQTT module is as follow:

```
name: remote-iothub
hub:
  address: tcp://localhub:1883
  username: test
  password: hahaha
remotes:
  - name: iothub
    address: 'ssl://xxxxxx.mqtt.iot.bj.baidubce.com:1884'
    clientid: remote-iothub-1
    username: xxxx/test
    ca: var/db/baetyl/cert/ca.pem
    cert: var/db/baetyl/cert/client.pem
    key: var/db/baetyl/cert/client.key
rules:
  - hub:
      subscriptions:
        - topic: t1
    remote:
      name: iothub
      subscriptions:
        - topic: t2
          qos: 1
logger:
  path: var/log/baetyl/remote-service.log
  level: 'debug'
```

According to the configuration of the above, it means that the Baetyl-Remote-MQTT module subscribes the topic `t1` from the Baetyl-Hub module, subscribes the topic `t2` from Baidu IoTHub. When MQTTBox publishes a message to the topic `t1`, the Baetyl-Hub module will receive this message and forward it to Baidu IoTHub via Baetyl-Remote-MQTT module, and MQTT.fx will also receive this message(suppose MQTT.fx has already subscribed the topic `t1` before) from Baidu IoTHub. Similarly, When we use MQTT.fx to publish a message to the topic `t2`, then Baidu IoTHub will receive it and forward it to the Baetyl-Hub module via Baetyl-Remote-MQTT module. Finally, MQTTBox will receive this message(suppose MQTTBox has already subscribed the topic `t2` before).

In a word, from MQTTBox publishes a message to the topic `t1`, to MQTT.fx receives the message, the routing path of the message are as follows.

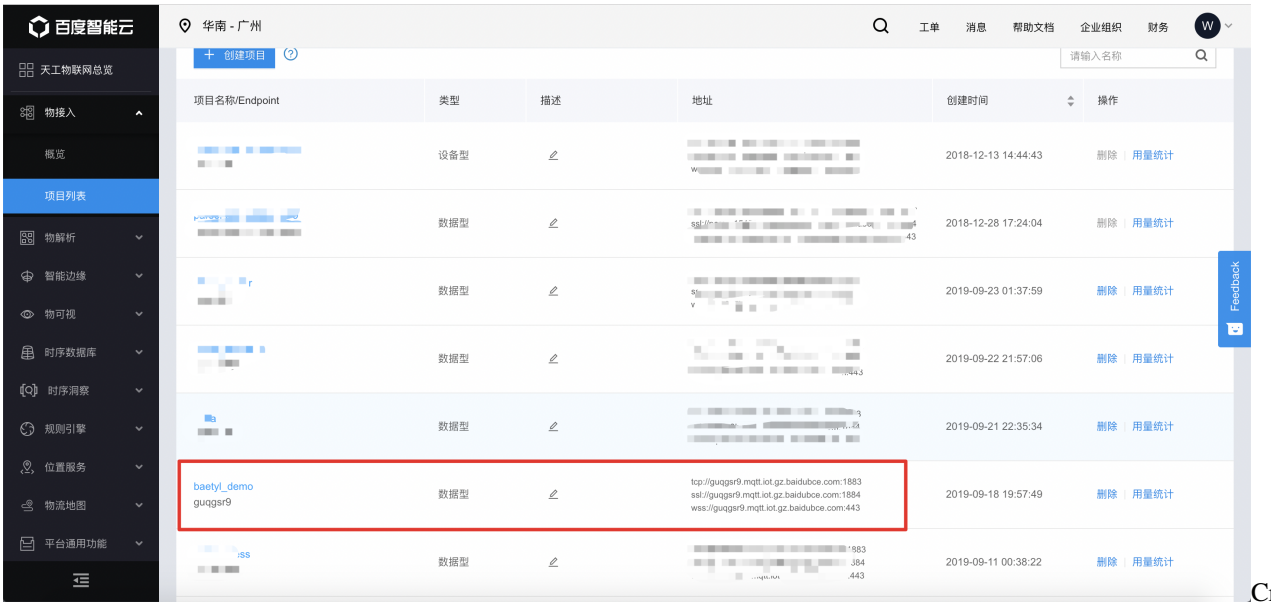
MQTTBox -> Baetyl-Hub module -> Baetyl-Remote-MQTT module -> Baidu IoTHub -> MQTT.fx

Similarly, from MQTT.fx publishes a message to the topic `t2`, to MQTTBox receives the message, the routing path of the message are as follows.

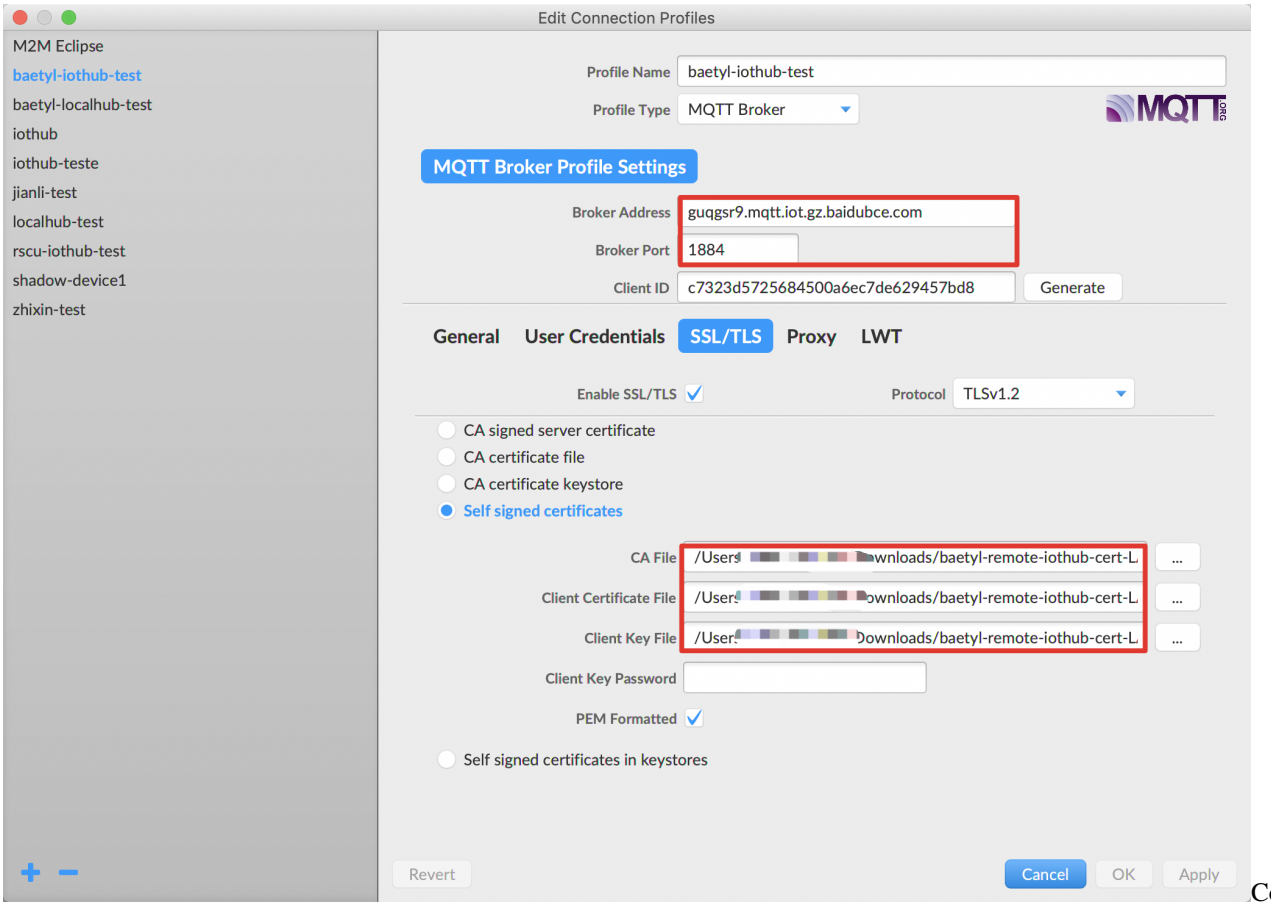
MQTT.fx -> Baidu IoTHub -> Baetyl-Remote-MQTT module -> Baetyl-Hub module -> MQTTBox

10.2.1 Establish a Connection between MQTT.fx and Baidu IoTHub

As described in `Step 1, Step 2`, the detailed contents of the connection between MQTT.fx and Baidu IoTHub are as follows.

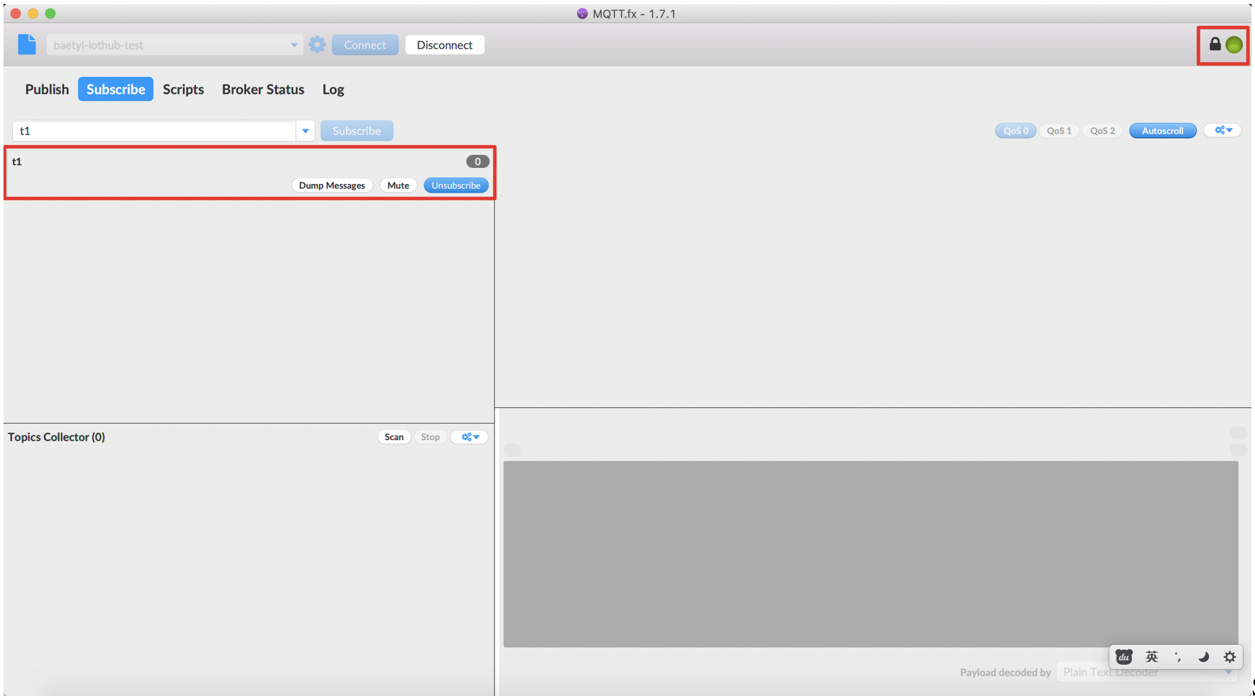


Create endpoint via Baidu IoTHub



Configuration of MQTT.fx

After set the configuration of MQTT.fx, click `OK` or `Apply` button, then click `Connect` button, and wait for the connecting. Also, we can check if the connection status is OK via the color button. When the button's color change to **Green**, that is to say, the connection is established. Then switch to the `Subscribe` page and subscribe the topic `t1`. More detailed contents are shown below.



Successfully establish a connection between MQTT.fx and Baidu IoTHub

10.2.2 Establish a Connection between MQTTBox and the Baetyl-Hub module

As described in `Step 3`, the Baetyl-Hub module and Baetyl-Remote-MQTT module also loaded when Baetyl started. Also, we can lookup the running status of Baetyl through the command `sudo systemctl status baetyl`.

```
baetyl@baetyl:~$ sudo systemctl status baetyl
● baetyl.service - Baetyl
   Loaded: loaded (/lib/systemd/system/baetyl.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2019-09-23 11:31:57 CST; 5h 55min ago
 Main PID: 997 (baetyl)
    Tasks: 10 (limit: 4623)
   CGroup: /system.slice/baetyl.service
           └─997 /usr/local/bin/baetyl start

9月 23 11:31:57 baetyl systemd[1]: Started Baetyl.
```

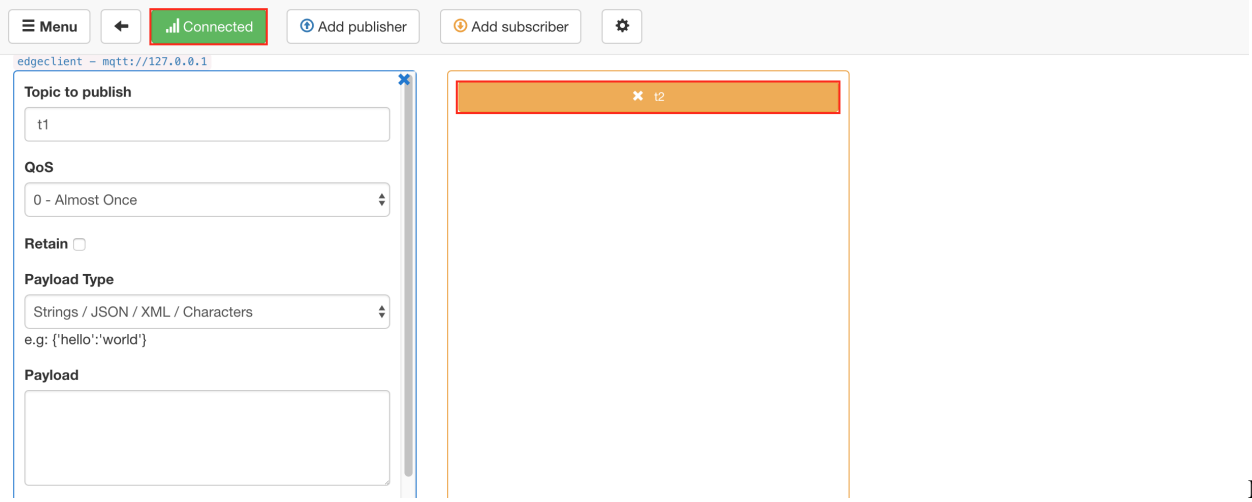
lookup the running status of Baetyl

In addition, we can execute the command `docker stats` to view the list of docker containers currently running on the system.

```
→ baetyl-demo docker ps
CONTAINER ID        IMAGE                                                   COMMAND                CREATED          STATUS          PORTS
                        NAMES
8bc6410e943d        [redacted]/baetyl-remote-mqtt:0.1.6    "baetyl-remote-mqtt"   12 minutes ago   Up 12 minutes
                        remote-mqtt
9fd900c43a89        [redacted]/baetyl-hub:0.1.6            "baetyl-hub"           12 minutes ago   Up 12 minutes   0.0.0.0:1883->1883/tcp
, 0.0.0.0:8883->8883/tcp   localhub
```

View the list of docker containers currently running

After Baetyl successfully startup, set the configuration of connection, then establish the connection with the Baetyl-Hub module and subscribe the topic `t2`.



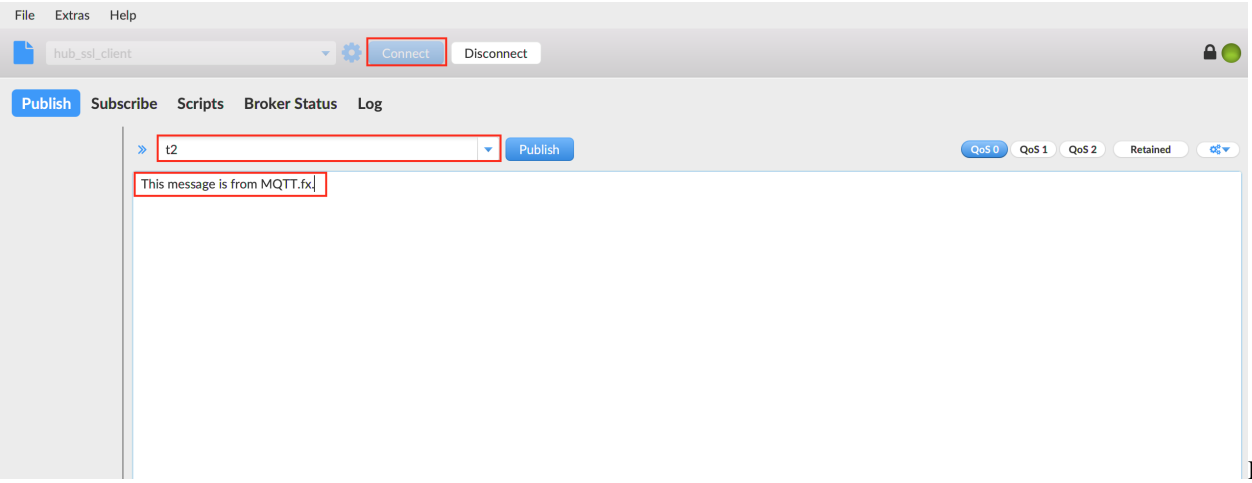
MQTTBox successfully subscribe the topic t2

10.2.3 Message Synchronize Test

Here, MQTT.fx and MQTTBox will be used as message publishers, and the other one will be used as a message receiver.

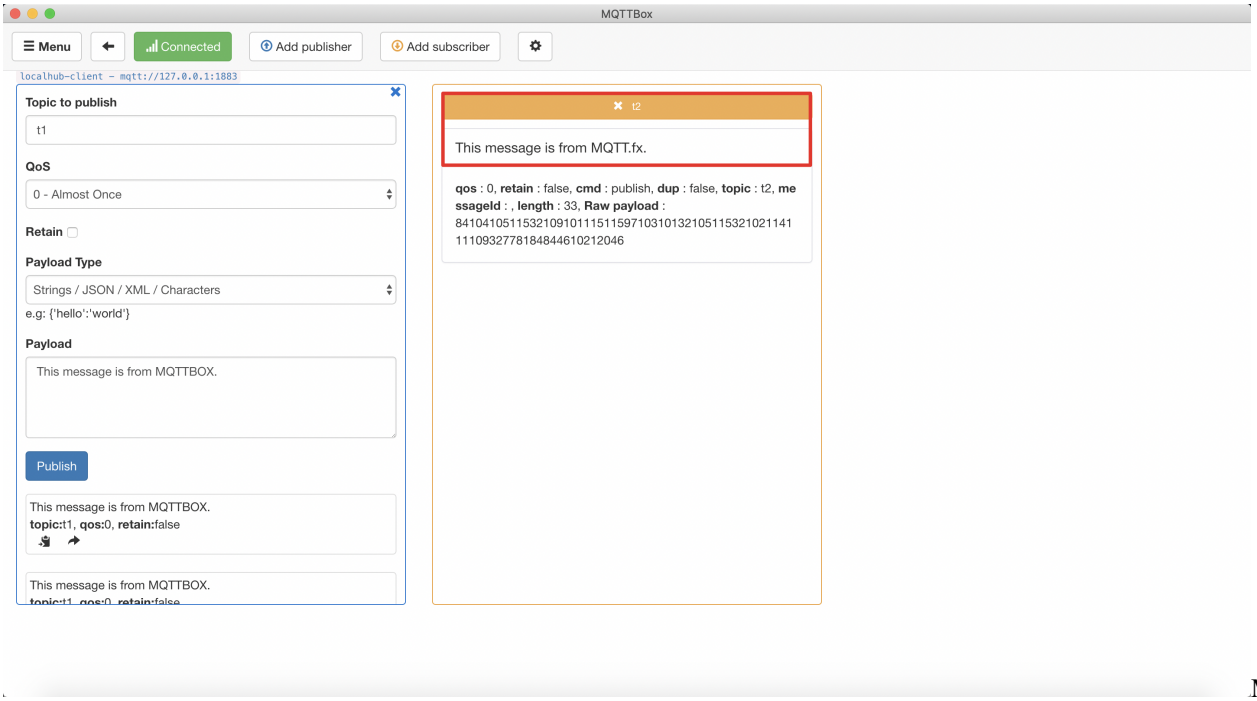
MQTT.fx publishes message, and MQTTBox receives message

Firstly, using MQTT.fx publishes a message `This message is from MQTT.fx.` to the topic `t2`.



Publishing a message to the topic t2 via MQTT.fx

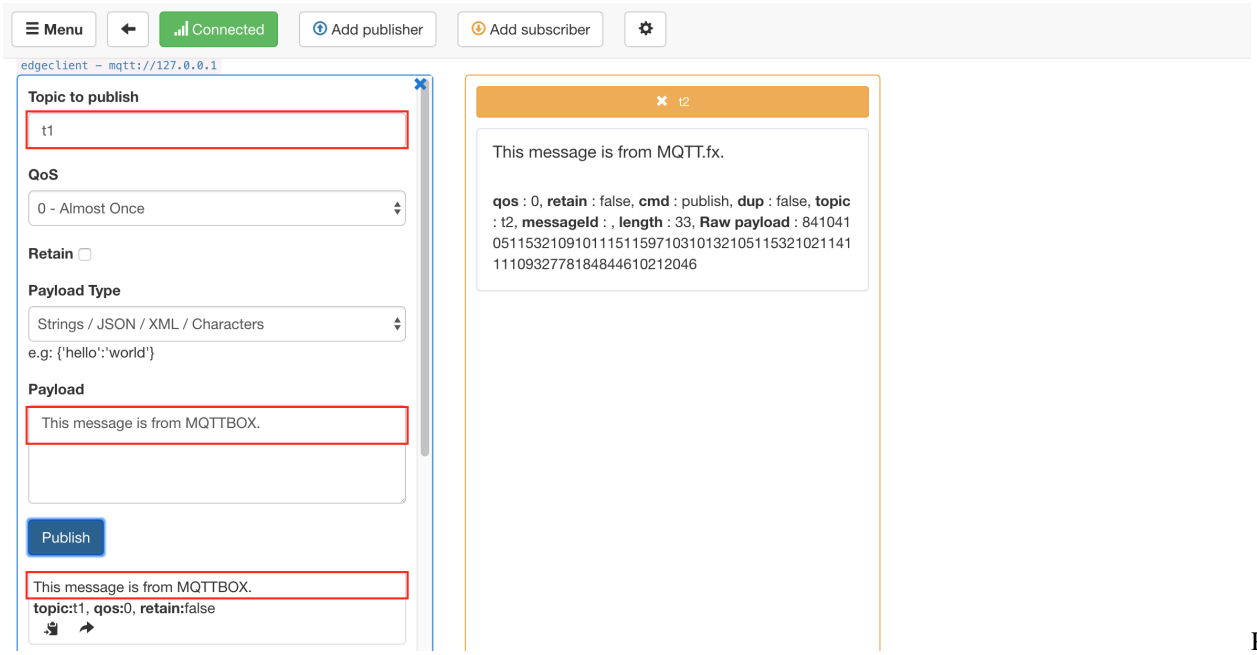
At the same time, observing the message receiving status of MQTTBox via the topic `t2`.



MQTTBox successfully received the message

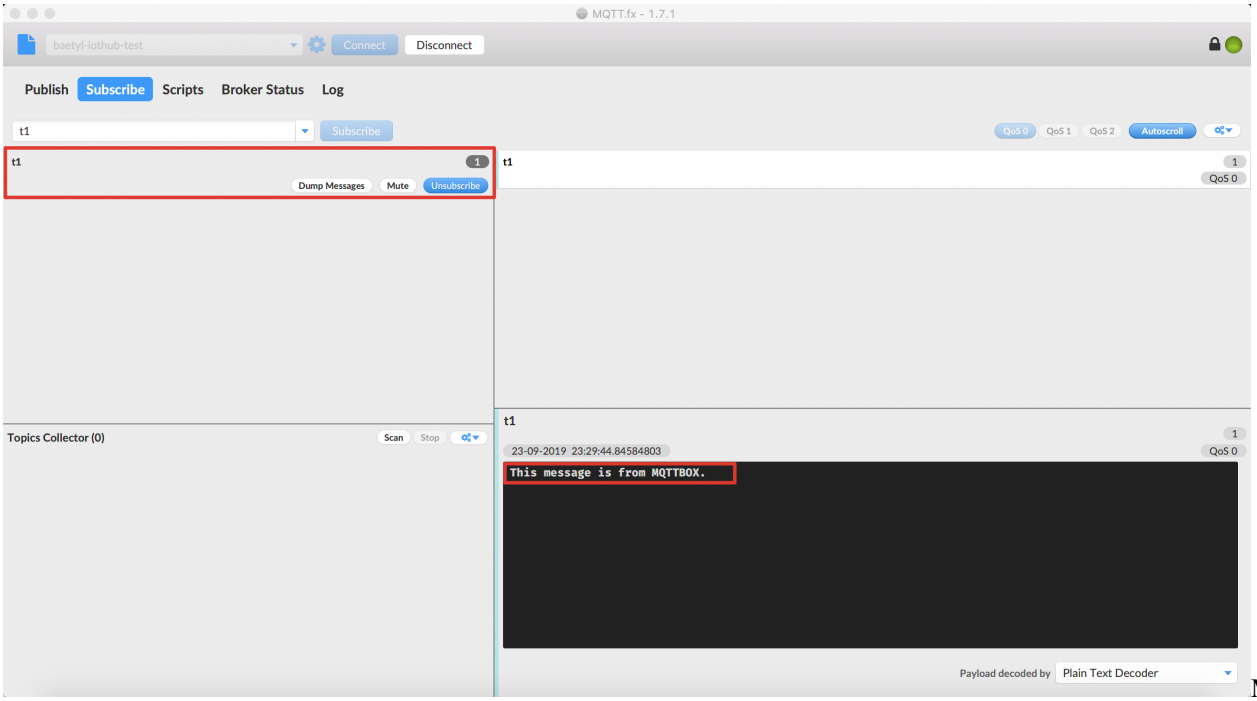
MQTTBox publishes message, and MQTT.fx receives message

Similarly, publishing the message `This message is from MQTTBox.` to the topic `t1` via MQTTBox.



Publishing a message to the topic t1 via MQTTBox

Then we can observe the message receiving status of MQTT.fx via the topic `t1`.



MQTT.fx successfully received the message

In summary, both MQTT.fx and MQTTBox have correctly received the specified message, and the content is consistent.

CHAPTER 11

How to write a python script for Python runtime

Statement

- The operating system as mentioned in this document is Ubuntu16.04.
- The version of runtime is Python3.6, and for Python2.7, configuration is the same except for the language difference when coding the scripts
- The MQTT client toolkit as mentioned in this document is MQTTBox.
- In this article, the service created based on the Hub module is called `localhub` service. And for the test case mentioned here, the `localhub` service, function calculation service, and other services are configured as follows:

```
# The configuration of Local Hub service
# Configuration file location is: `var/db/baetyl/localhub-conf/service.yml`.
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: 'test'
    password: 'hahaha'
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']

# The configuration of Local Function Manager service
# Configuration file location is: var/db/baetyl/function-manager-conf/service.yml
hub:
  address: tcp://localhub:1883
  username: test
  password: hahaha
rules:
  - clientid: localfunc-1
    subscribe:
```

(continues on next page)

(continued from previous page)

```
      topic: py
    function:
      name: sayhi3
    publish:
      topic: py/hi
functions:
  - name: sayhi3
    service: function-sayhi3
    instance:
      min: 0
      max: 10
      idletime: 1m

# The configuration of python function runtime
# Configuration file location is: var/db/baetyl/function-sayhi-conf/service.yml
functions:
  - name: 'sayhi3'
    handler: 'sayhi.handler'
    codedir: 'var/db/baetyl/function-sayhi'

# The configuration of application.yml
# Configuration file location is: var/db/baetyl/application.yml
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub
    replica: 1
    ports:
      - 1883:1883
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
  - name: function-manager
    image: hub.baidubce.com/baetyl/baetyl-function-manager
    replica: 1
    mounts:
      - name: function-manager-conf
        path: etc/baetyl
        readonly: true
      - name: function-manager-log
        path: var/log/baetyl
  - name: function-sayhi3
    image: hub.baidubce.com/baetyl/baetyl-function-python36
    replica: 0
    mounts:
      - name: function-sayhi-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayhi-code
        path: var/db/baetyl/function-sayhi
        readonly: true
volumes:
```

(continues on next page)

(continued from previous page)

```
  # hub
  - name: localhub-conf
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-log
    path: var/db/baetyl/localhub-log
  # function manager
  - name: function-manager-conf
    path: var/db/baetyl/function-manager-conf
  - name: function-manager-log
    path: var/db/baetyl/function-manager-log
  # function python runtime sayhi
  - name: function-sayhi-conf
    path: var/db/baetyl/function-sayhi-conf
  - name: function-sayhi-code
    path: var/db/baetyl/function-sayhi-code
```

Baetyl officially provides the Python runtime to load python scripts written by users. The following description is about the name of the python script, the execution function name, input, output parameters, and so on.

11.1 Function Name Convention

The name of a python script can refer to Python's universal naming convention, which Baetyl does not specifically limit. If you want to apply a python script to handle an MQTT message, the configuration of Python3.6 runtime service is as follows:

```
functions:
  - name: 'sayhi3'
    handler: 'sayhi.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

Here, we focus on the `handler` attribute, where `sayhi` represents the script name and the `handler` represents the entry function called in the file.

```
function-sayhi-code/
├── __init__.py
└── sayhi.py
```

More detailed configuration of Python runtime, please refer to *Python runtime configuration*.

11.2 Parameter Convention

```
def handler(event, context):
    # do something
    return event
```

The Python runtime provided by Baetyl supports two parameters: `event` and `context`, which are described separately below.

- **event**Depend on the `Payload` in the MQTT message

  - If the original `Payload` is a json format data, then pass in the data handled by `json.loads(Payload)`
  - If the original `Payload` is Byte, string(not Json), then pass in the original `Payload`

- **context**MQTT message context
  - context.messageQOS // MQTT QoS
  - context.messageTopic // MQTT Topic
  - context.functionName // MQTT functionName
  - context.functionInvokeID //MQTT function invokeID
  - context.invokeid // as above, be used to compatible with CFC

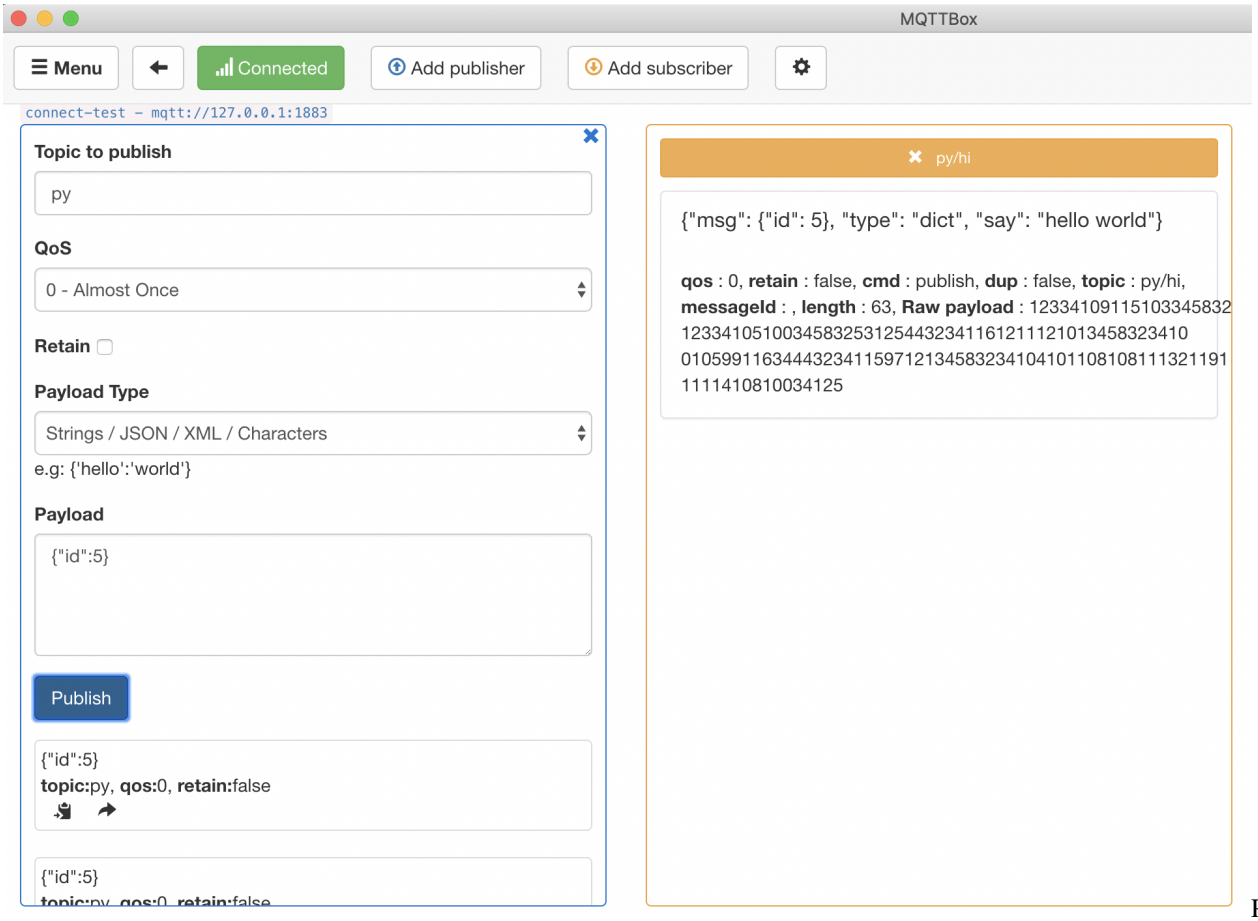
***NOTE**: When testing in the cloud CFC, please don't use the context defined by Baetyl directly. The recommended method is to first determine whether the field is exists or not in the `context`. If exists, read it.*

11.3 Hello World

Now we will implement a simple python script with the goal of appending a `hello world` message to each MQTT message. For a dictionary format message, return it directly, and for an none dictionary format message, convert it to string and return.

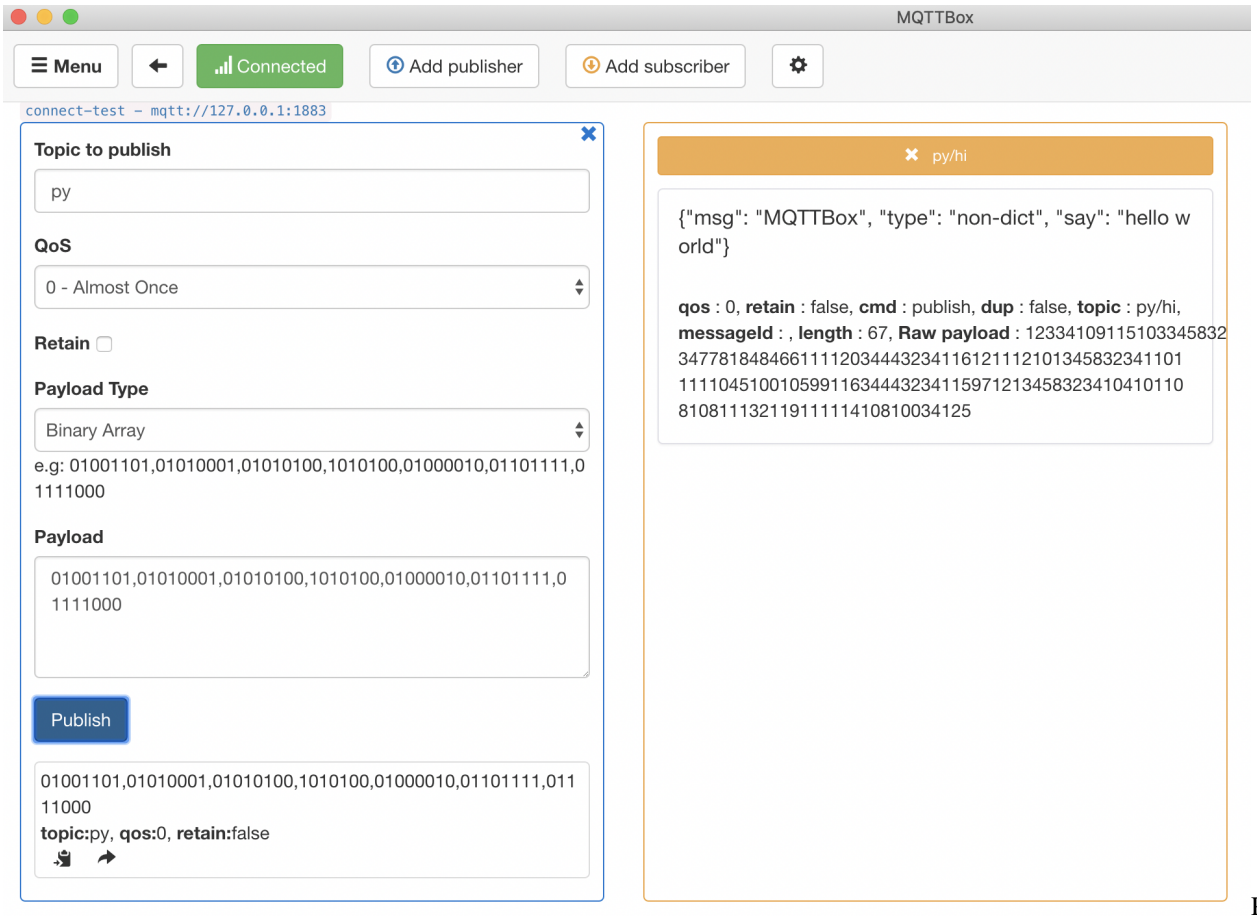
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def handler(event, context):
    result = {}
    if isinstance(event, dict):
        result['msg'] = event
        result['type'] = 'dict'
        result['say'] = 'hello world'
    else:
        result['msg'] = event.decode("utf-8")
        result['type'] = 'non-dict'
        result['say'] = 'hello world'

    return result
```

Publish a dict format message:



Publish a dict format message

Publish an non-dict format message:



Publish an non-dict format message

As above, for some general needs, we can implement it through the Python Standard Library. However, for some more complex demands, it is often necessary to import third-party libraries to complete. How to solve the problem? We've provided a general solution in How to import third-party libraries for Python runtime.

CHAPTER 12

How to write a javascript for Node runtime

Statement

- The operating system as mentioned in this document is Ubuntu16.04.
- The version of runtime is Node8.5
- The MQTT client toolkit as mentioned in this document is MQTTBox.
- In this article, the service created based on the Hub module is called `localhub` service. And for the test case mentioned here, the `localhub` service, function calculation service, and other services are configured as follows:

```
# The configuration of Local Hub service
# Configuration file location is: var/db/baetyl/localhub-conf/service.yml
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: 'test'
    password: 'hahaha'
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']

# The configuration of Local Function Manager service
# Configuration file location is: var/db/baetyl/function-manager-conf/service.yml
hub:
  address: tcp://localhub:1883
  username: test
  password: hahaha
rules:
  - clientid: localfunc-1
    subscribe:
      topic: node
    function:
```

(continues on next page)

(continued from previous page)

```
      name: sayhi
    publish:
      topic: t/hi
functions:
  - name: sayhi
    service: function-sayhi
    instance:
      min: 0
      max: 10
      idletime: 1m

# The configuration of Node function runtime
# Configuration file location is: var/db/baetyl/function-sayjs-conf/service.yml
functions:
  - name: 'sayhi'
    handler: 'index.handler'
    codedir: 'var/db/baetyl/function-sayhi'

# The configuration of application.yml
# Configuration file location is: var/db/baetyl/application.yml
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub
    replica: 1
    ports:
      - 1883:1883
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
  - name: function-manager
    image: hub.baidubce.com/baetyl/baetyl-function-manager
    replica: 1
    mounts:
      - name: function-manager-conf
        path: etc/baetyl
        readonly: true
      - name: function-manager-log
        path: var/log/baetyl
  - name: function-sayhi
    image: hub.baidubce.com/baetyl/baetyl-function-node85
    replica: 0
    mounts:
      - name: function-sayjs-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayjs-code
        path: var/db/baetyl/function-sayhi
        readonly: true
volumes:
  # hub
  - name: localhub-conf
```

(continues on next page)

(continued from previous page)

```
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-log
    path: var/db/baetyl/localhub-log
  # function manager
  - name: function-manager-conf
    path: var/db/baetyl/function-manager-conf
  - name: function-manager-log
    path: var/db/baetyl/function-manager-log
  # function node runtime sayhi
  - name: function-sayjs-conf
    path: var/db/baetyl/function-sayjs-conf
  - name: function-sayjs-code
    path: var/db/baetyl/function-sayjs-code
```

Baetyl officially provides the Node runtime to load javascripts written by users. The following description is about the name of a javascript, the execution function name, input, output parameters, and so on.

12.1 Function Name Convention

The name of a javascript can refer to universal naming convention, which Baetyl does not specifically limit. If you want to apply a javascript to handle an MQTT message, the configuration of Node runtime service is as follows:

```
functions:
  - name: 'sayhi'
    handler: 'index.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

Here, we focus on the `handler` attribute, where `index` represents the script name and the `handler` represents the entry function called in the file.

```
function-sayjs-code/
└── index.js
```

More detailed configuration of Node runtime, please refer to *Node runtime configuration*.

12.2 Parameter Convention

```
exports.handler = (event, context, callback) => {
    callback(null, event);
};
```

The Node runtime provided by Baetyl supports two parameters: `event` and `context`, which are described separately below.

- **event**Depend on the `Payload` in the MQTT message
  - If the original `Payload` is a json format data, then pass in the data handled by `json.loads(Payload)`
  - If the original `Payload` is Byte, string(not Json), then pass in the original `Payload`
- **context**MQTT message context

- context.messageQOS // MQTT QoS
- context.messageTopic // MQTT Topic
- context.functionName // MQTT functionName
- context.functionInvokeID //MQTT function invokeID
- context.invokeid // as above, be used to compatible with CFC

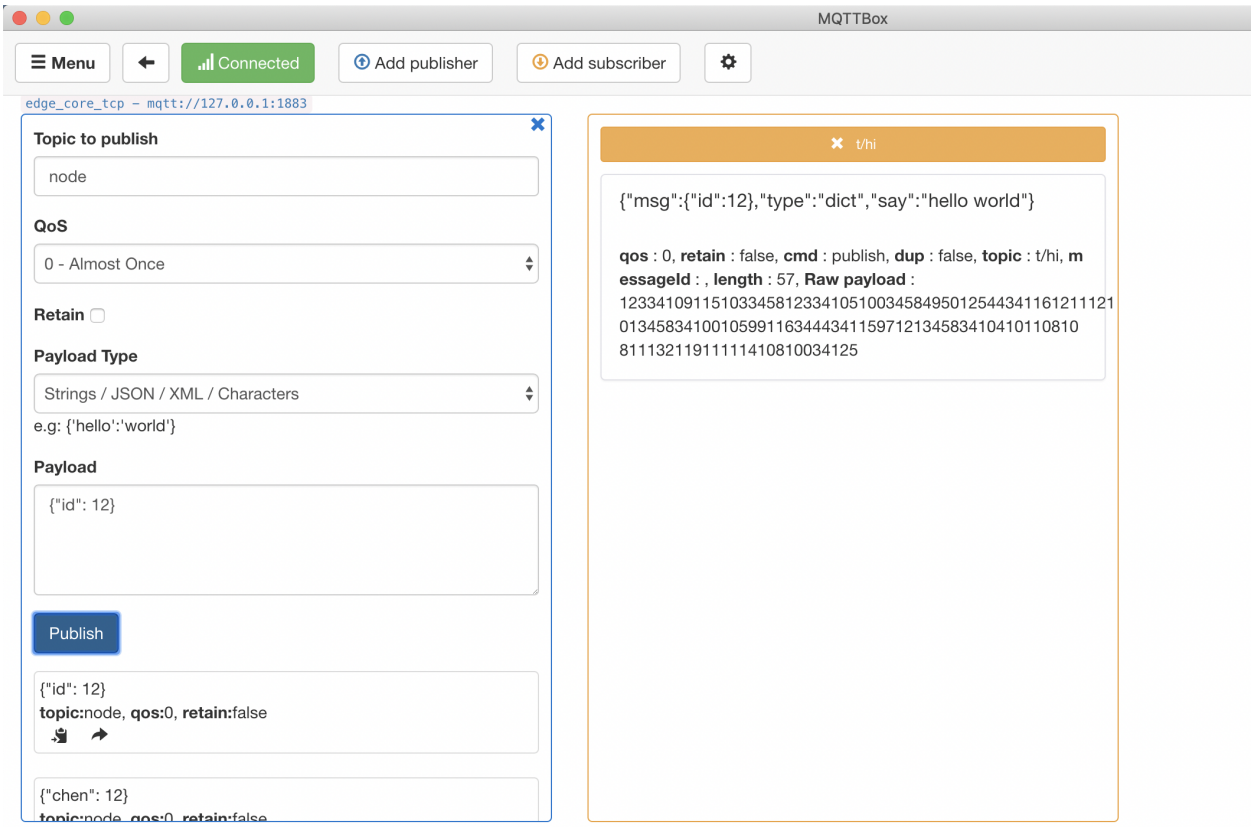
***NOTE**: When testing in the cloud CFC, please don't use the context defined by Baetyl directly. The recommended method is to first determine whether the field is exists or not in the `context`. If exists, read it.*

12.3 Hello World

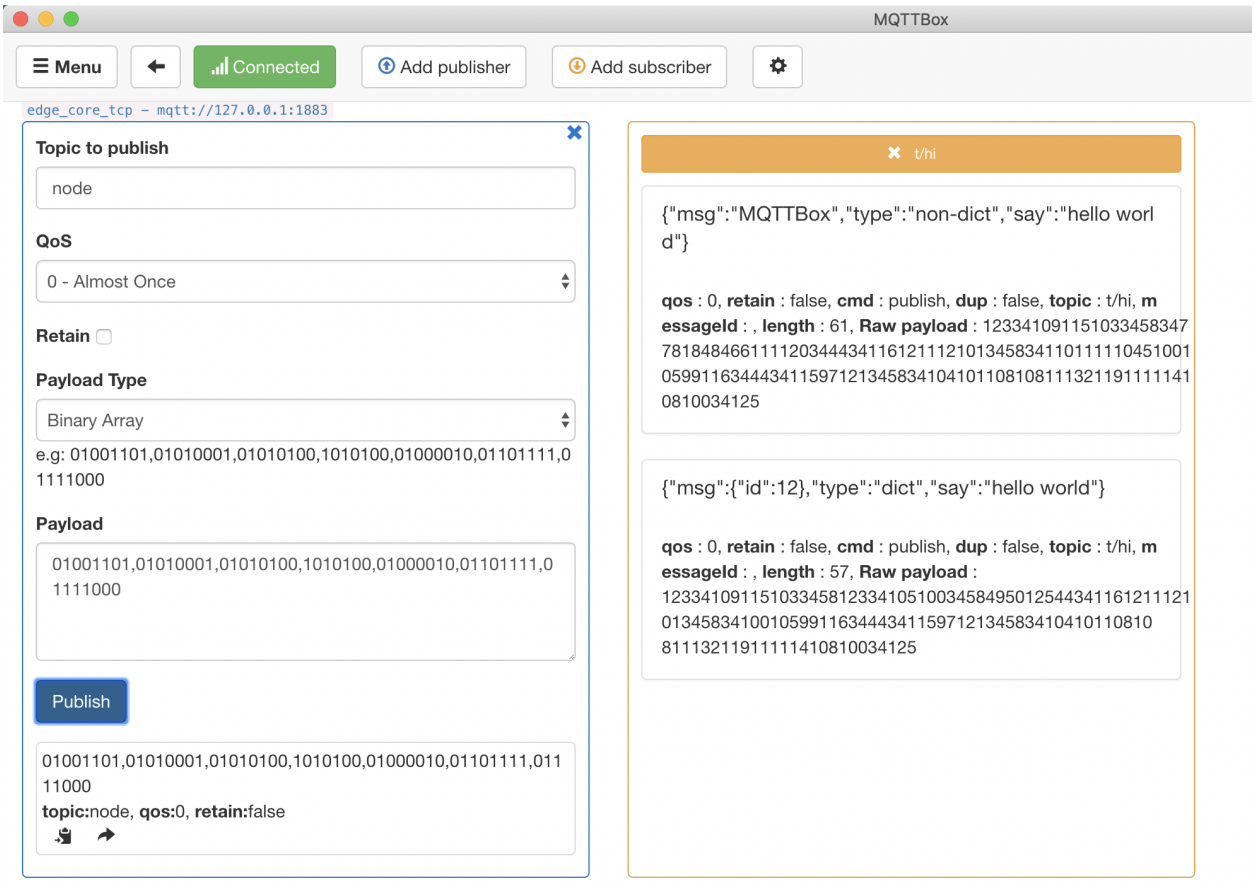
Now we will implement a simple javascript with the goal of appending a `hello world` message to each MQTT message. For a dictionary format message, return it directly, and for an none dictionary format message, convert it to string and return.

```
#!/usr/bin/env node

exports.handler = (event, context, callback) => {
  result = {};

  if (Buffer.isBuffer(event)) {
      const message = event.toString();
      result["msg"] = message;
      result["type"] = 'non-dict';
  }else {
      result["msg"] = event;
      result["type"] = 'dict';
  }

  result["say"] = 'hello world';
  callback(null, result);
};
```

Publish a dict format message:



Publish an non-dict format message:



As above, for some general needs, we can implement it through the Node Standard Library. However, for some more complex demands, it is often necessary to import third-party libraries to complete. How to solve the problem? We've provided a general solution in How to import third-party libraries for Node runtime.

CHAPTER 13

How to import third-party libraries for Python runtime

Statement

- The operating system as mentioned in this document is Ubuntu16.04.
- The version of runtime is Python3.6, and for Python2.7, configurations are the same except for the language differences when coding the scripts.
- The MQTT client toolkit as mentioned in this document is MQTTBox.
- In this document, the third-party libraries we'll import are `requests` and `Pytorch`.
- In this article, the service created based on the Hub module is called `localhub` service. And for the test case mentioned here, the `localhub` service, function calculation service, and other services are configured as follows:

```
# The configuration of localhub service
# Configuration file location is: var/db/baetyl/localhub-conf/service.yml
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: 'test'
    password: 'hahaha'
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']

# The configuration of Local Function Manager service
# Configuration file location is: var/db/baetyl/function-manager-conf/service.yml
hub:
  address: tcp://localhub:1883
  username: test
  password: hahaha
rules:
  - clientid: localfunc-1
```

(continues on next page)

(continued from previous page)

```
    subscribe:
      topic: py
    function:
      name: sayhi3
    publish:
      topic: py/hi
functions:
  - name: sayhi3
    service: function-sayhi3
    instance:
      min: 0
      max: 10
      idletime: 1m

# The configuration of application.yml
# Configuration file location is: var/db/baetyl/application.yml
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub
    replica: 1
    ports:
      - 1883:1883
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
  - name: function-manager
    image: hub.baidubce.com/baetyl/baetyl-function-manager
    replica: 1
    mounts:
      - name: function-manager-conf
        path: etc/baetyl
        readonly: true
      - name: function-manager-log
        path: var/log/baetyl
  - name: function-sayhi3
    image: hub.baidubce.com/baetyl/baetyl-function-python36
    replica: 0
    mounts:
      - name: function-sayhi-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayhi-code
        path: var/db/baetyl/function-sayhi
        readonly: true
volumes:
  # hub
  - name: localhub-conf
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-log
```

(continues on next page)

(continued from previous page)

```
    path: var/db/baetyl/localhub-log
  # function manager
  - name: function-manager-conf
    path: var/db/baetyl/function-manager-conf
  - name: function-manager-log
    path: var/db/baetyl/function-manager-log
  # function python runtime sayhi
  - name: function-sayhi-conf
    path: var/db/baetyl/function-sayhi-conf
  - name: function-sayhi-code
    path: var/db/baetyl/function-sayhi-code
```

Generally, using the Python Standard Library may not meet our needs. In fact, it is often necessary to import some third-party libraries. Two examples are given below.

13.1 Import `requests` third-party libraries

Suppose we want to crawl a website and get the response. Here, we can import a third-party library `requests`. How to import it, as shown below:

- Step 1: change path to the directory of Python scripts, then download `requests` package and its dependency packages(idnaurllib3chardetcertifi)

```
cd /directory/of/Python/script
pip download requests
```

- Step 2: inflate the downloaded `.whl` files for getting the source packages, then remove useless `.whl` files and package-description files

```
unzip \*.whl
rm -rf *.whl *.dist-info
```

- Step 3: make the current directory be a package

```
touch __init__.py
```

- Step 4: import the third-party library `requests` in the Python script as shown below:

```
import requests
```

- Step 5: execute your Python script

```
python your_script.py
```

If the above operations are normal, the resulting script directory structure is as shown in the following figure.

```
[root@instance-80v5cs0x function-sayhi-code]# tree -L 1
.
├── certifi
├── chardet
├── get.py
├── idna
├── __init__.py
├── requests
└── urllib3

5 directories, 2 files
```

the directory of the Python script

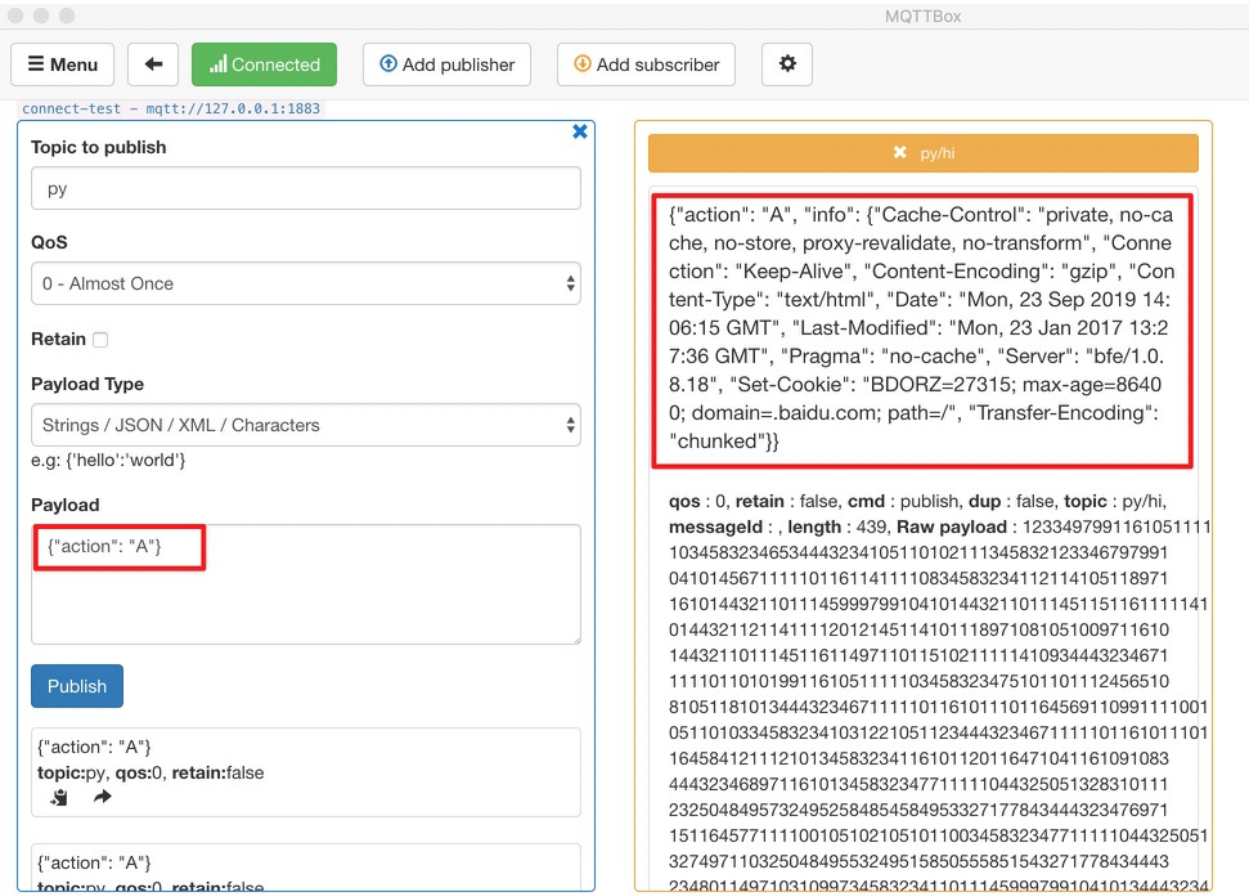
Now we write the Python script `get.py` to get the headers information of https://baidu.com, assuming the trigger condition is that Python3.6 runtime receives the "A" command from the `localhub` service. More detailed contents are as follows:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import requests

def handler(event, context):
    """
    data: {"action": "A"}
    """
    if 'action' in event:
        if event['action'] == 'A':
            r = requests.get('https://baidu.com')
            if str(r.status_code) == '200':
                event['info'] = dict(r.headers)
            else:
                event['info'] = 'exception found'
        else:
            event['info'] = 'action error'
    else:
        event['error'] = 'action not found'

return event
```

The configuration of Python function runtime is as below:

```
# The configuration of Python function runtime
functions:
  - name: 'sayhi3'
    handler: 'get.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

As above, after receiving the message publish to the topic `py`, the `localhub` service will call the `get.py` script to handle, and following it publish the result to the topic `py/hi`. So in the test case, we use MQTTBox to subscribe the topic `py/hi` and publish the message `{"action": "A"}` to the `localhub` service by the topic `py`. If everything works correctly, MQTTBox can receive the message of the topic `py/hi` which contains the headers information of https://baidu.com as shown below.



Get the header information of https://baetyl.io

13.2 Import `Pytorch` third-party libraries

`Pytorch` is a widely used deep learning framework for machine learning. We can import a third-party library `Pytorch` to use its functions. How to import it, as shown below:

- Step 1: change path to the directory of Python scripts, then download `Pytorch` package and its dependency packages(PILcaffee2numpysixtorchvision)

```
cd /directory/of/Python/script
pip3 download torch torchvision
```

- Step 2: inflate the downloaded `.whl` files for getting the source packages, then remove useless `.whl` files and package-description files

```
unzip \*.whl
rm -rf *.whl *.dist-info
```

- Step 3: make the current directory be a package

```
touch __init__.py
```

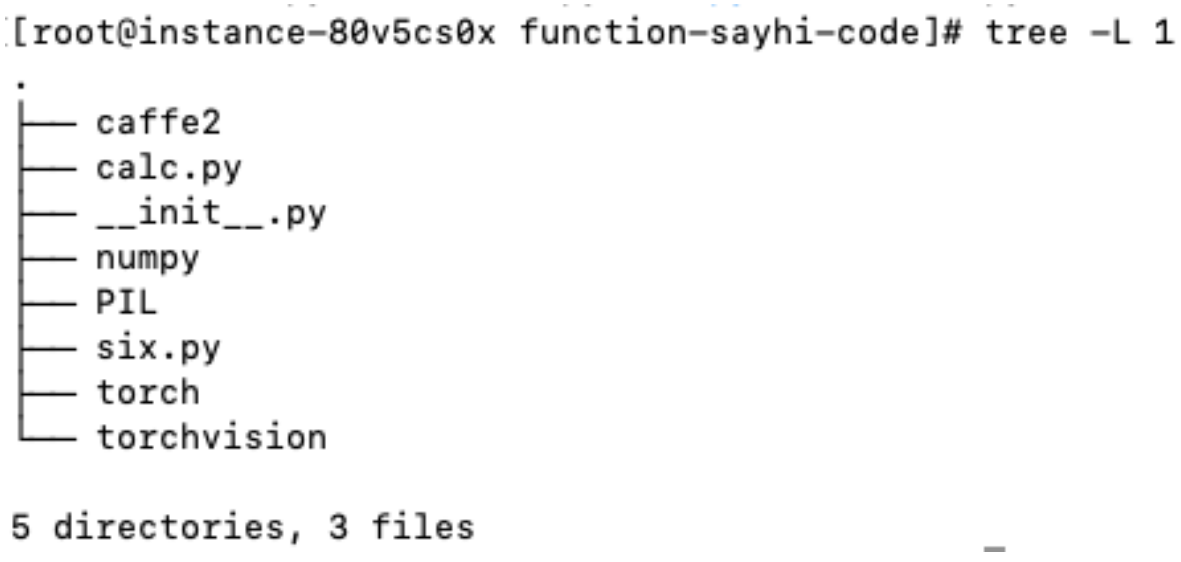
- Step 4: import the third-party library `Pytorch` in the Python script as shown below:

```
import torch
```

- Step 5: execute your Python script

```
python your_script.py
```

If the above operations are normal, the resulting script directory structure is as shown in the following figure.



the directory of the Python script

Now we write the Python script `calc.py` to use functions provided by `Pytorch` for generating a random tensor, assuming the trigger condition is that Python3.6 runtime receives the "B" command from the `localhub` service. More detailed contents are as follows:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import torch

def handler(event, context):
  """
  data: {"action": "B"}
  """
  if 'action' in event:
    if event['action'] == 'B':
      x = torch.rand(5, 3)
      event['info'] = x.tolist()
    else:
      event['info'] = 'exception found'
  else:
    event['error'] = 'action not found'

  return event
```

The configuration of Python function runtime is as below:

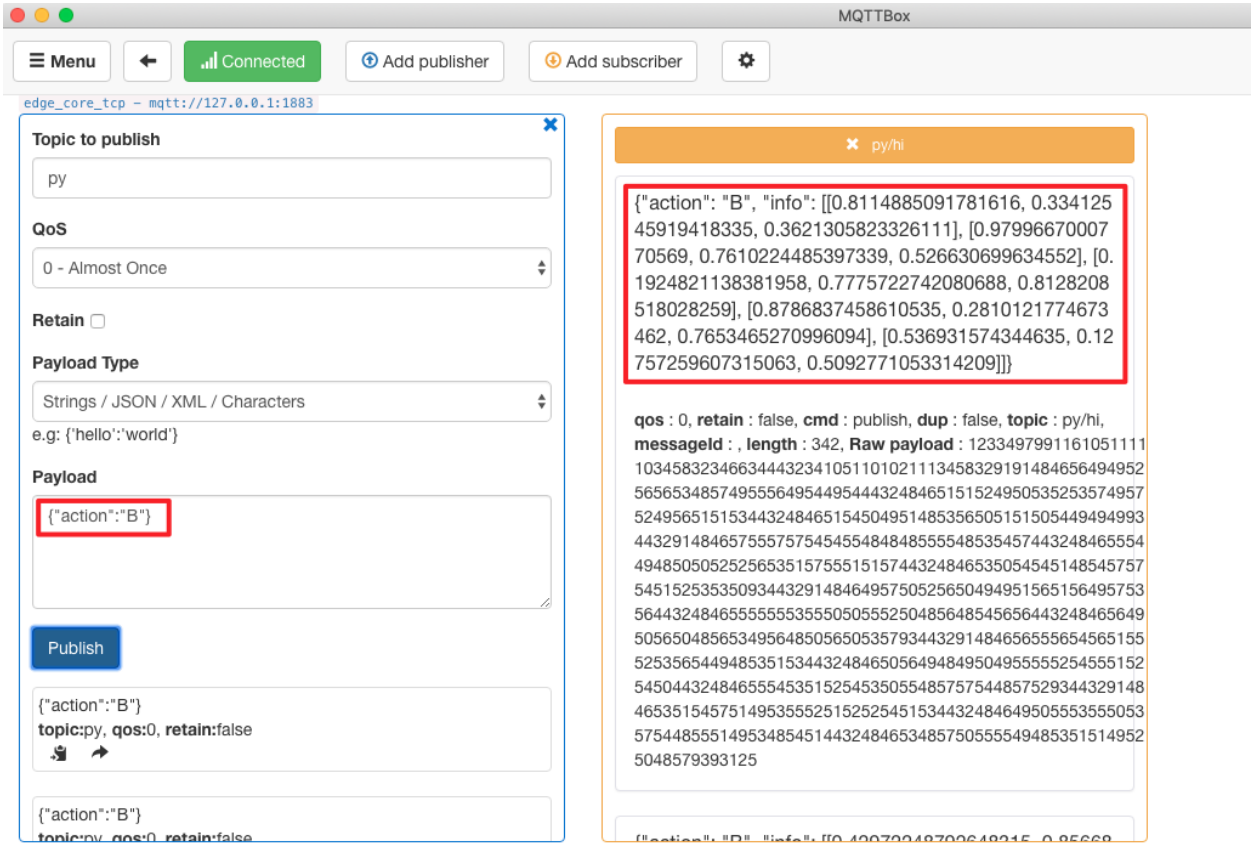
```
# The configuration of Python function runtime
functions:
  - name: 'sayhi3'
```

(continues on next page)

(continued from previous page)

```
    handler: 'calc.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

As above, after receiving the message publish to the topic `py`, the `localhub` service will call the `calc.py` script to handle, and following it publish the result to the topic `py/hi`. So in the test case, we use MQTTBox to subscribe the topic `py/hi` and publish the message `{"action": "B"}` to the `localhub` service by the topic `py`. If everything works correctly, MQTTBox can receive the message of the topic `py/hi` in which we can get a random tensor as shown below.



generate a random tensor

CHAPTER 14

How to import third-party libraries for Node runtime

Statement

- The operating system as mentioned in this document is Ubuntu16.04.
- The version of runtime is Node8.5
- The MQTT client toolkit as mentioned in this document is MQTTBox.
- In this document, we give an example about how import the third-party library `Lodash`.
- In this article, the service created based on the Hub module is called `localhub` service. And for the test case mentioned here, the `localhub` service, function calculation service, and other services are configured as follows:

```
# The configuration of Local Hub service
# Configuration file location is: var/db/baetyl/localhub-conf/service.yml
listen:
  - tcp://0.0.0.0:1883
principals:
  - username: 'test'
    password: 'hahaha'
    permissions:
      - action: 'pub'
        permit: ['#']
      - action: 'sub'
        permit: ['#']

# The configuration of Local Function Manager service
# Configuration file location is: var/db/baetyl/function-manager-conf/service.yml
hub:
  address: tcp://localhub:1883
  username: test
  password: hahaha
rules:
  - clientid: localfunc-1
    subscribe:
```

(continues on next page)

(continued from previous page)

```
      topic: node
    function:
      name: sayhi
    publish:
      topic: t/hi
functions:
  - name: sayhi
    service: function-sayhi
    instance:
      min: 0
      max: 10
      idletime: 1m

# The configuration of application.yml
# Configuration file location is: var/db/baetyl/application.yml
version: v0
services:
  - name: localhub
    image: hub.baidubce.com/baetyl/baetyl-hub
    replica: 1
    ports:
      - 1883:1883
    mounts:
      - name: localhub-conf
        path: etc/baetyl
        readonly: true
      - name: localhub-data
        path: var/db/baetyl/data
      - name: localhub-log
        path: var/log/baetyl
  - name: function-manager
    image: hub.baidubce.com/baetyl/baetyl-function-manager
    replica: 1
    mounts:
      - name: function-manager-conf
        path: etc/baetyl
        readonly: true
      - name: function-manager-log
        path: var/log/baetyl
  - name: function-sayhi
    image: hub.baidubce.com/baetyl/baetyl-function-node85
    replica: 0
    mounts:
      - name: function-sayjs-conf
        path: etc/baetyl
        readonly: true
      - name: function-sayjs-code
        path: var/db/baetyl/function-sayhi
        readonly: true
volumes:
  # hub
  - name: localhub-conf
    path: var/db/baetyl/localhub-conf
  - name: localhub-data
    path: var/db/baetyl/localhub-data
  - name: localhub-log
    path: var/db/baetyl/localhub-log
```

(continues on next page)

(continued from previous page)

```
  # function manager
  - name: function-manager-conf
    path: var/db/baetyl/function-manager-conf
  - name: function-manager-log
    path: var/db/baetyl/function-manager-log
  # function node runtime sayhi
  - name: function-sayjs-conf
    path: var/db/baetyl/function-sayjs-conf
  - name: function-sayjs-code
    path: var/db/baetyl/function-sayjs-code
```

Generally, using the Node Standard Library may not meet our needs. In fact, it is often necessary to import some third-party libraries. We'll give one example below.

14.1 Import `Lodash` third-party libraries

`Lodash` is a modern JavaScript utility library delivering modularity, performance & extras. Baetyl support import third-party libraries such as `Lodash` to use its functions. How to import it, as shown below:

- Step 1: change path to the directory of javascripts, then install `Lodash` package

```
cd /directory/of/Node/script
npm install --save lodash
```

- Step 2: import `Lodash` in a javascript:

```
const _ = require('lodash');
```

- Step 3: execute your javascript:

```
node your_script.js
```

If the above operations are normal, the resulting script directory structure is as shown in the following figure.

```
[root@instance-80v5cs0x function-sayjs-code]# tree -L 2
.
├── index.js
└── node_modules
    └── lodash

1 directory, 2 files
```

_ the directory of Lodash

Now we write the script `index.js` to use functions provided by `Lodash`. More detailed contents are as follows:

```
#!/usr/bin/env node

const _ = require('lodash');

exports.handler = (event, context, callback) => {
  result = {}
```

(continues on next page)

(continued from previous page)

```
  //remove repeating elements in array
  result["unique_array"] = _.uniq(event['array']);
  //sort
  result['sorted_users'] = _.sortBy(event['users'], function(o) { return o.age; });
  //filter
  result['filtered_users'] = _.filter(event['users'], function(o) { return !o.active;␣
↪});

  callback(null, result);
}
```

The configuration of Node function runtime is as below:

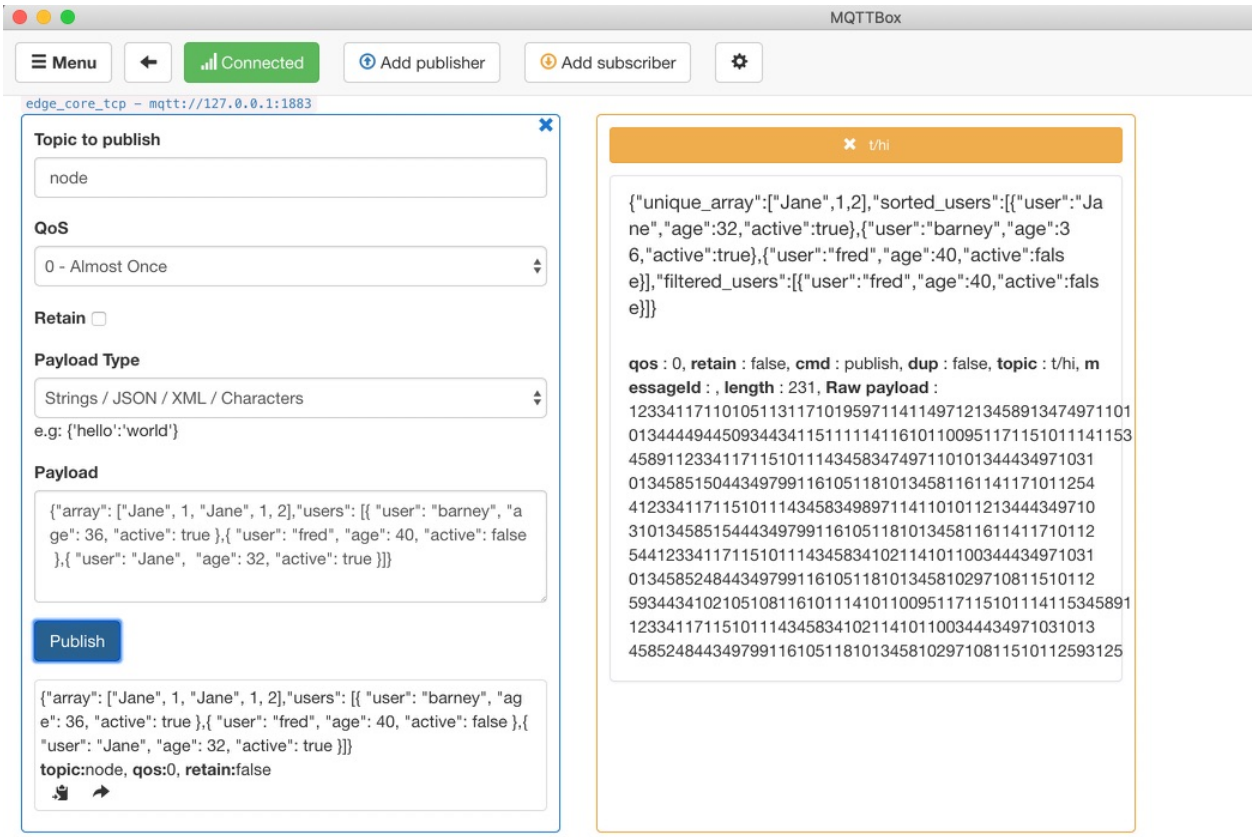
```
# The configuration of Node function runtime
functions:
  - name: 'sayhi'
    handler: 'index.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

First define the following json data as an input message:

```
{
    "array": ["Jane", 1, "Jane", 1, 2],
    "users": [
        { "user": "barney", "age": 36, "active": true  },
        { "user": "fred",   "age": 40, "active": false },
        { "user": "Jane",   "age": 32, "active": true  }
    ]
}
```

As above, after the `localhub` service receives the message sent to the topic `node`, it calls `index.js` script to execute the concrete logic to remove repeated elements, filter, sort of array in input data. The result is then fed back to the topic `t/hi` as an MQTT message. We subscribe to the topic `t/hi` via MQTTBox and we can observe the following message:

```
{
    "unique_array": ["Jane", 1, 2],
    "sorted_users": [
        { "user": "Jane",   "age": 32, "active": true  },
        { 'user': 'barney', "age": 36, "active": true  },
        { "user": "fred",   "age": 40, "active": false }
    ],
    "filtered_users": [
        { "user": "fred",   "age": 40, "active": false }
    ],
}
```



using_lodash

CHAPTER 15

Customize Runtime Module

The function runtime is the carrier of the function execution. The function is executed by dynamically loading the function code, which is strongly related to the language of the function implementation. For example, Python code needs to be called using the Python runtime. This is a multi-language issue. In order to unify the interface and protocol, we finally chose GRPC to create a flexible functional computing framework with its powerful cross-language IDL and high-performance RPC communication capabilities.

In the function compute service (FaaS), `baetyl-function-manager` is responsible for the management and invocation of function instances. The function instance is provided by the function runtime service, and the function runtime service only needs to meet the conventions described below.

15.1 Protocol Convention

Developers can use the `function.proto` in `sdk/baetyl-go` to generate messages and service implementations for their respective programming languages, as defined below. For the usage of GRPC, refer to GRPC Official Documents.

```
syntax = "proto3";

package baetyl;

// The function server definition.
service Function {
    rpc Call(FunctionMessage) returns (FunctionMessage) {}
    // rpc Talk(stream Message) returns (stream Message) {}
}

// FunctionMessage function message
message FunctionMessage {
    uint64 ID                = 1;
    uint32 QOS               = 2;
    string Topic             = 3;
    bytes  Payload           = 4;
```

(continues on next page)

(continued from previous page)

```
    string  FunctionName      = 11;
    string  FunctionInvokeID  = 12;
}
```

***NOTE**: In docker container mode, the resource limit of the function instance should not be lower than `50M` memory and 20 threads.*

15.2 Configuration Convention

The function runtime module does not enforce the configuration. However, for the unified configuration mode, the following configuration items are recommended.

- name: function name
- handler: function processing interface
- codedir: The path to the function code, if any.

The following is a configuration example of a Python function runtime service:

```
functions:
  - name: 'sayhi'
    handler: 'sayhi.handler'
    codedir: 'var/db/baetyl/function-sayhi'
```

15.3 Start/Stop Convention

The function runtime service is the same as other services, the only difference is that the instance is dynamically started by other services. For example, to avoid listening port conflicts, you can specify the port dynamically. The function runtime module can read `BAETYL_SERVICE_INSTANCE_ADDRESS` from the environment variable as the address that the GRPC Server listens on. In addition, dynamically launched function instances do not have permission to call the main program's API. Finally, the module listens for the `SIGTERM` signal to gracefully exit. A complete implementation can be found in the Python2.7Python3.6 runtime module (`baetyl-function-python27baetyl-function-python36`).

CHAPTER 16

Customize Module

Read *Build Baetyl From Source* before developing custom modules to understand Baetyl's build environment requirements.

Custom modules do not limit the development language. Understand these conventions below to integrate custom modules better and faster.

The custom module does not limit the development language. As long as it is a runnable program, you can even use the image already on hub.docker.com, such as `eclipse-mosquitto`. But understanding the conventions described below will help you develop custom modules better and faster.

16.1 Directory Convention

At present, the native process mode, like the docker container mode, opens up a separate workspace for each service. Although it does not achieve the effect of isolation, it can guarantee the consistency of the user experience. The process mode creates a separate directory for each service in the `var/run/baetyl/services` directory, using service name. When the server starts, it specifies the directory as the working directory, and the service-bound storage volumes will be mapped (soft link) to the working directory. Here we keep the definition of the docker container mode, the workspace under the directory is also called the container, then the directory in the container has the following recommended usage:

- Default working directory in the container: `/`
- Default configuration file in the container: `/etc/baetyl/service.yml`
- Default persistence path in the container: `/var/db/baetyl`
- Default log directory in the container: `/var/log/baetyl`

NOTE: If the data needs to be persisted on the device (host), such as database and log, the directory in the container must be mapped to the host directory through the storage volume, otherwise the data will be lost after the service is stopped.

16.2 Start/Stop Convention

There is no excessive requirement for the module to be started. But it is recommended to load the YMAL format configuration from the default file, then run the module's business logic, and finally listen to the `SIGTERM` signal to gracefully exit. A simple `Golang` module implementation can refer to the MQTT remote communication module (`baetyl-remote-mqtt`).

16.3 SDK

If the module is developed using `Golang`, you can use the SDK provided by Baetyl, located in the sdk directory of the project, and the functional interfaces are provided by `Context`. At present, the SDK capabilities provided are still not enough, and the follow-up will be gradually strengthened.

The list of `Context` interfaces are as follows:

```
// returns the system configuration of the service, such as hub and logger
Config() *ServiceConfig
// loads the custom configuration of the service
LoadConfig(interface{}) error
// creates a Client that connects to the Hub through system configuration,
// you can specify the Client ID and the topic information of the subscription.
NewHubClient(string, []mqtt.TopicInfo) (*mqtt.Dispatcher, error)
// returns logger interface
Log() logger.Logger
// check running mode
IsNative() bool
// waiting to exit, receiving SIGTERM and SIGINT signals
Wait()
// returns wait channel
WaitChan() <-chan os.Signal

// Master RESTful API

// updates system and
UpdateSystem(string, bool) error
// inspects system stats
InspectSystem() (*Inspect, error)
// gets an available port of the host
GetAvailablePort() (string, error)
// reports the stats of the instance of the service
ReportInstance(stats map[string]interface{}) error
// starts an instance of the service
StartInstance(serviceName, instanceName string, dynamicConfig map[string]string) error
// stop the instance of the service
StopInstance(serviceName, instanceName string) error
```

The following uses the simple timer module implementation as an example to introduce the usage of the SDK.

```
package main

import (
        "encoding/json"
        "time"

        "github.com/baetyl/baetyl/protocol/mqtt"
```

(continues on next page)

(continued from previous page)

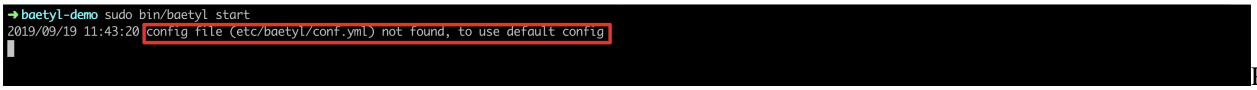
```
        baetyl "github.com/baetyl/baetyl/sdk/baetyl-go"
)

// custom configuration of the timer module
type config struct {
        Timer struct {
                Interval time.Duration `yaml:"interval" json:"interval" default:"1m"`
        } `yaml:"timer" json:"timer"`
        Publish mqtt.TopicInfo `yaml:"publish" json:"publish" default:"{\"topic\":\
↪"timer\"}"`
}

func main() {
        // Running module in baetyl context
        baetyl.Run(func(ctx baetyl.Context) error {
                var cfg config
                // load custom config
                err := ctx.LoadConfig(&cfg)
                if err != nil {
                        return err
                }
                // create a hub client
                cli, err := ctx.NewHubClient("", nil)
                if err != nil {
                        return err
                }
                // start client to keep connection with hub
                cli.Start(nil)
                // create a timer
                ticker := time.NewTicker(cfg.Timer.Interval)
                defer ticker.Stop()
                for {
                        select {
                        case t := <-ticker.C:
                                msg := map[string]int64{"time": t.Unix()}
                                pld, _ := json.Marshal(msg)
                                // send a message to hub triggered by timer
                                err := cli.Publish(cfg.Publish, pld)
                                if err != nil {
                                        // log error message
                                        ctx.Log().Errorf(err.Error())
                                }
                        case <-ctx.WaitChan():
                                // wait until service is stopped
                                return nil
                        }
                }
        })
}
```

CHAPTER 17

FAQ

This document mainly provides related issues and solutions for Baetyl deployment and startup in various platforms.

Q1: Prompt missing startup dependency configuration item when starting Baetyl in docker container mode.



Picture

Suggested Solution: As shown in the above picture, Baetyl startup lacks configuration dependency files, refer to GitHub-Baetyl `example` folder(the location is `etc/baetyl/conf.yml`).

Q2: Execute the command `docker info` get the following result on Ubuntu/Debian: "WARNING: No swap limit support"

Suggested Solution:

1. Open `/etc/default/grub` with your favorite text editor. Make sure the following lines are commented out or add them if they don't exist:

   GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"

1. Save and exit and then run: `sudo update-grub` and reboot.

***NOTE**: If you got some error when you execute `step2`, it may be that the `grub` setting is incorrect. Please repeat `steps 1 and 2`.*

Q3: Found "WARNING: Your kernel does not support swap limit capabilities. Limitation discarded" when Baetyl start.

Suggested Solution: Refer to Q2.

Q4: Found "Got permission denied while trying to connect to the docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.38/images/json: dial unix /var/run/docker.sock: connect: permission denied" when Baetyl start.

Suggested Solution: Add the docker group if it doesn't already exist:

```
sudo groupadd docker
```

Add the current user to the docker group:

```
sudo usermod -aG docker ${USER}
su - ${USER}
```

Q5: Found "Cannot connect to the docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?" when Baetyl start.

Suggested Solution: If you still report this issue after the solution of Q4 solution is executed, restart the docker service.

For example, execute the following command on CentOS:

```
systemctl start docker
```

Q6: Found "failed to create master: Error response from daemon: client version 1.39 is too new. Maximum supported API version is 1.38" when Baetyl start.

Suggested Solution: Workaround is to pass API version via environment variable:

`DOCKER_API_VERSION=1.38`

For example:

```
sudo vim ~/.bash_profile
export DOCKER_API_VERSION=1.38
source ~/.bash_profile
```

Q7: How does Baetyl connect to NB-IOT network?

Suggested Solution: NB-IoT is a network standard similar to 2/3/4G with low bandwidth and low power consumption. NB-IoT supports TCP-based MQTT protocol, so you can use NB-IoT card to connect to Baidu Cloud IotHub, deploy Baetyl application and communicate with BIE Cloud Management Suite. However, among the three major operators in China, Telecom have imposed whitelist restrictions on their NB cards, and only allow to connect to Telecom Cloud service IP. Therefore, only Mobile NB cards and Unicom NB cards can be used to connect to Baidu Cloud service.

Q8: Does Baetyl support to push data to Kafka?

Suggested Solution: Support, you can refer to *How to write a python script for python runtime*, and subscribe messages from the local Hub module and writing them to Kafka service. Besides, you can also refer to *How to develop a customize module for Baetyl*, which subscribes message from the local Hub module and then writes it to Kafka.

Q9: What are the ways to change Baetyl configurations? Can I only make configuration changes through the BIE Cloud Management Suite?

Suggested Solution: Currently, we recommend changing configurations through the BIE Cloud Management Suite, but you can also manually change the configuration file on the core device and then restart Baetyl to take effect.

Q10I download MQTTBox client, extract it to a directory, and copy/move the executable file `MQTTBox` to `/usr/local/bin`(other directory is similar, such as `/usr/bin`, `/bin`, `/usr/sbin`, etc.). But it reports an error of "error while loading shared libraries: libgconf-2.so.4: cannot open shared object file: No such file or directory`when`MQTTBox" start.

Suggested Solution As above description, this is because the lack of `libgconf-2.so.4` library when `MQTTBox` start, and the recommended use is as follows:

- Step 1: Download and extract the MQTTBox software package;
- Step 2: `cd /pat/to/MQTTBox/directory and sudo chmod +x MQTTBox`;
- Step 3`sudo ln -s /path/to/MQTTBox /usr/local/bin/MQTTBox`;
- Step 4Open terminal and execute the command `MQTTBox`.

Q11: `localfunc` can't process the message, check `funclog` has the following error message:

> level=error msg="failed to create new client" dispatcher=mqtt error="dial tcp 0.0.0.0:1883:connect:connection refused"

Suggested Solution: If you are using the BIE Cloud Management Suite to deliver the configuration, there are a few points to note:

1. Cloud delivery configuration currently only supports container mode.
2. If the configuration is sent in the cloud, the hub address configured in `localfunc` should be `localhub` instead of `0.0.0.0`.

According to the above information, the actual error is judged, and the configuration is delivered from the cloud as needed, or by referring to *Configuration Analysis Document* for verification and configuration.

Q12 How can i use BIE Cloud Management Suite with CFC(Cloud Function Compute)?

Suggested Solution:

1. Make sure your BIE configuration and CFC functions in the same region, such as beijing/guangzhou.
2. Make sure your CFC functions are published.
3. Select `CFC function template` when volume create, more detailed contents please refer to How-to-apply-volume-in-the-right-way

Q13 What's the relationship between the parameter ports and the parameter listen which in the hub configuration file?

Suggested Solution:

1. ports: Port exposed configuration in docker container mode.
2. listen: Which address the hub module will listen on. In docker container mode, it's means container address. In native process mode, it's means host address.
3. By referring to *Configuration Analysis Document*

Q14: How to process data in the cloud platform after message send to Baidu IoT Hub by Baetyl?

Suggested Solution: In the cloud platform, the Rule Engine can be used to transmit data to other cloud services, such as CFC(Cloud Function Compute), TSDB.

Q15: How to connect the Device management of Baidu IoT Hub?

Suggested Solution: The Device management of Baidu IoT Hub does not support ssl authentication. As a temporary solution, you can configure *Remote Feature* to connect the Device management with username and password authentication manually.

Q16If I don't want to lose messages and want to ensure all messages are synchronized to cloud, how can I do?

Suggested Solution:

You must meet the following 2 conditions:

- To make sure messages will be persist in local disk which are sent to local hub, the topic's QoS must be set to 1.
- To make sure messages will be sent to cloud successful, the QoS of `rules` configuration of Remote module must be set to 1, which includes remote sub's QoS and the pub's QoS. By referring to *Configuration Analysis Document*

Q17: After the configuration is sent from the cloud to the edge, the default startup mode is `docker` container mode. After modifying `mode: native` in `etc/baetyl/conf.yml` the startup error is similar to the following: "failed to update system: open

/Users/ Xxx/baetyl_native/var/run/baetyl/services/agent/lib/baetyl/hub.baidubce.com/baetyl/baetyl-agent:latest/package.yml: no such file or directory".

Suggested Solution: At present, our cloud management does not support the process mode. If you need to start `Baetyl` in process mode locally, please refer to the configuration content in `example/native` and execute the command `make install-native`. Install and start by process with the command `sudo baetyl start`.

Q18: There is a similar error when downloading the image: "error="Error response from daemon: Get https://hub.baidubce.com/v2/: x509: failed to load system roots and no roots provided" baetyl= Master".

Suggested Solution: This is because the `ca-certificates` package is missing from the system and can be installed to solve this problem. For example, if the host system is Debian, you can use the following command to install it:

```
sudo apt-get update
sudo apt-get install ca-certificates
```

For other systems, please check the relevant installation operations yourself.

CHAPTER 18

Download

18.1 Golang download

Official websitehttps://golang.org/dl/

Golang download link of BOS(Baidu-Object-Storage)

18.2 MQTT download

18.2.1 MQTT client sample

C sample of MQTT clientC-sample-of-MQTT-client

Python sample of MQTT clientPython-sample-of-MQTT-client

18.2.2 MQTT.fx download

Official websitehttp://www.jensd.de/apps/mqttfx/1.7.1/

MQTT.fx download link of BOS

18.2.3 MQTTBox download

Official websitehttp://workswithweb.com/html/mqttbox/downloads.html

MQTTBox download link of BOS

18.2.4 Paho MQTT Client SDK

Official website: http://www.eclipse.org/paho

Paho MQTT Client Comparison