

**Docap数据采集系统云端部署手册**

**当前版本号：1.1**

**2024/05/20**

**变更履历**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **版本号** | **修改人员** | **修改目的** | **维护日期** | **审查人员** | **审查日期** |
| 1.0 | 崔晓乐 | 初版编写 | 2024/1/12 |  |  |
| 1.1 | 娄伟 | 增加文件服务器部署操作 | 2024/5/27 |  |  |
| 1.2 | 娄伟 | 更新云端部署手册 | 2024/8/7 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[1. 文档保密等级说明 4](#_Toc187324510)

[2. 文档编写目的 4](#_Toc187324511)

[3. 文档适用对象 4](#_Toc187324512)

[4. 系统概述 4](#_Toc187324513)

[4.1. 部署架构图 5](#_Toc187324514)

[5. 服务部署 6](#_Toc187324515)

[5.1. Core部署 6](#_Toc187324516)

[5.1.1环境准备 6](#_Toc187324517)

[5.1.2部署详细 6](#_Toc187324518)

[5.2. MQTT Broker部署 8](#_Toc187324519)

[5.2.1环境准备 8](#_Toc187324520)

[5.2.2部署详细 9](#_Toc187324521)

[5.3. 时序数据库部署 11](#_Toc187324522)

[5.3.1 Cassandra部署 11](#_Toc187324523)

[5.3.1.1环境准备 11](#_Toc187324524)

[5.3.2部署详细 12](#_Toc187324525)

[5.4. 文件服务器部署 13](#_Toc187324526)

[5.4.1环境准备 13](#_Toc187324527)

[5.4.2部署详细 13](#_Toc187324528)

# 文档保密等级说明

公开：无特殊限制可公开使用

秘密：仅限于公司内部人员使用

仅限项目人员：仅限于项目相关人员使用

机密：仅限于公司总经理，高层管理人员，部门经理、副经理及事件相关负责人

绝密：仅限于公司总经理，高层管理人员及事件相关负责人

# 文档编写目的

通过该文档，使用户可以自由的部署Docap的云端平台，并通过

# 文档适用对象

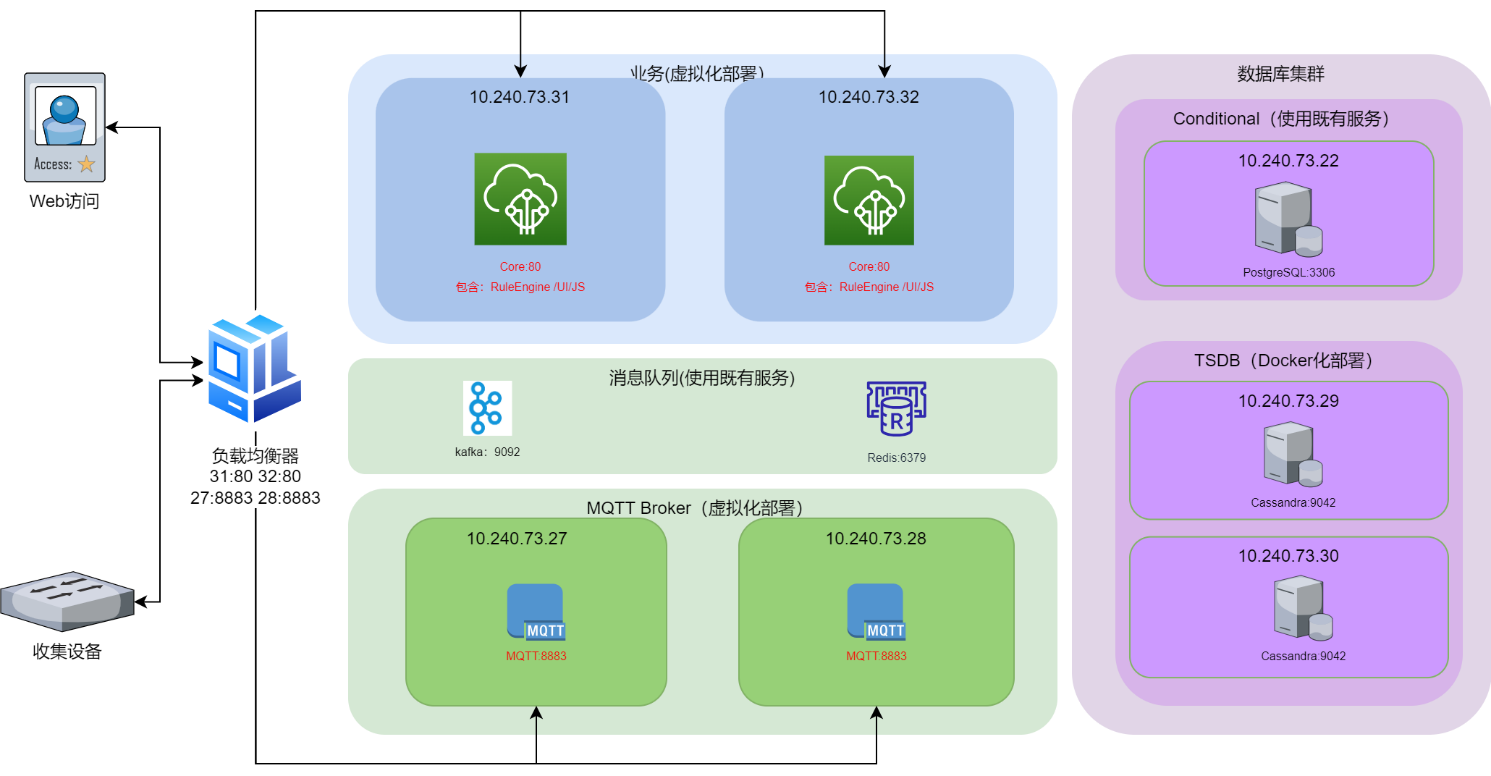
Docap数据采集系统的云端部署人员以及相关的云端系统维护人员。

# 系统概述

Docap云端系统是Docap数据采集系统的云端服务，用于支持Docap边缘设备进行数据的收集，指令的下发，设备的维护等功能，通过Docap云端系统，用户可以进行CAN数据采集，XCP数据采集，UDS采集，UDS Flash烧写，OTA升级等功能。

## 部署架构图

基于后期维护考虑，进行容器化和虚拟化结合的部署方式，这其中包括各个子服务的容器，以及第三方服务的容器，另外，考虑高可用以及负载均衡，核心服务均部署双份（后期根据接入设备可动态添加）



|  |  |  |  |
| --- | --- | --- | --- |
| **服务名** | **部署方式** | **作用** | **部署位置及端口** |
| Core | 虚拟化 | 【自研服务】负责核心业务处理包括RuleEngine、UI、Js-Executor，负责消息处理，页面呈现，JS执行，REST API以及Web Socket的处理 | 10.240.73.31:80  10.240.73.32:80 |
| MQTT Broker | 虚拟化 | 【自研服务】MQTT服务支持 | 10.240.73.27:8883  10.240.73.28:8883 |
| Kafka | N/A | 【Geely服务】逻辑数据存储 | 10.240.73.XX:9092 |
| Redis | N/A | 【Geely服务】逻辑数据存储 | 10.240.73.XX:6379 |
| PostgreSQL | N/A | 【Geely服务】逻辑数据存储 | 10.240.73.22:3306 |
| Cassandra | 容器化 | 【第三方服务】时序数据存储  包含热备 | 10.240.73.29:9042  10.240.73.30:9042 |

# 服务部署

首先，编译工程源码，会生成thingsboard.deb和tb-mqtt-transport.deb

## Core部署

### 5.1.1环境准备

部署环境默认为ubuntu环境，版本为20.04

1. 首先更新系统并安装java

sudo apt update

sudo apt install openjdk-11-jdk

1. 上述命令执行完成后，校验java是否安装成功

java -version

输出以下内容表示安装成功

openjdk version "11.0.xx"

OpenJDK Runtime Environment (...)

OpenJDK 64-Bit Server VM (build ...)

3.查看6.1章节安装yarn,nodejs

### 5.1.2部署详细

1. 安装包准备

将编译好的thingsboard.deb拷贝到ubuntu的目录/tmp中

1. 安装

sudo dpkg -i /tmp/thingsboard.deb

1. 更改配置

进入/usr/share/thingsboard/conf目录

更改配置, 将下述服务配置内容添加到thingsboard.conf中

图形用户界面, 文本, 应用程序

描述已自动生成

分别更改kafka, redis, cassandra, postgresql服务的地址、端口、账号密码等信息

1. 初始化postgresql和cassandra数据库表

此步骤是初始化postgresql和cassandra数据库表，注意本步骤如果已做过，无需再执行。

注意cassandra需要根据后续章节要预先安装。

此步骤需要先配置好redis, postgresql（需要先创建thingsboard数据库表）, cassandra, kafka等基础服务，配置完上述内容后，则可执行本步骤。

1. 创建postgresql中数据库表, 连接进入数据库，CREATE DATABASE thingsboard；
2. 进入/usr/share/thingsboard/conf
3. 执行sudo dos2unix thingsboard.conf
4. 执行sudo dos2unix thingsboard.yml
5. 进入/usr/share/thingsboard/bin目录
6. 执行sudo dos2unix install.sh
7. 执行sudo chmod +x install.sh
8. 执行sudo ./install.sh
9. 命令执行完毕后，会提示安装成功，此步骤任务完成

注:如果只是更新部署deb包(之前已经初始化过数据库相关)

进入/usr/share/thingsboard/conf/liquibase目录

sudo rm master.yml

sudo rm -r changelog

1. 启动服务

service thingsboard start

1. 关闭服务

service thingsboard stop

1. 重启服务

service thingsboard restart

1. Log查看

vim /var/log/thingsboard/thingsboard.log

## MQTT Broker部署

### 5.2.1环境准备

部署环境默认为ubuntu环境，版本为20.04

1. 更新系统并安装java

sudo apt update

sudo apt install openjdk-11-jdk

1. 校验java是否安装成功

java -version

输出以下内容表示安装成功

openjdk version "11.0.xx"

OpenJDK Runtime Environment (...)

OpenJDK 64-Bit Server VM (build ...)

### 5.2.2部署详细

1. 文件准备

将编译生成的tb-mqtt-transport.deb拷贝到目标机/tmp/目录下

1. 安装执行

sudo dpkg -i /tmp/tb-mqtt-transport.deb

1. 更改配置

上述命令执行完成后，进入/usr/share/tb-mqtt-transport/conf/目录

更改tb-mqtt-transport.conf文件，添加以下内容

图形用户界面, 文本, 应用程序

描述已自动生成

1. 启动服务

service tb-mqtt-transport start

1. 关闭服务

service tb-mqtt-transport stop

1. 重启服务

service tb-mqtt-transport restart

1. Log查看

vim /var/log/tb-mqtt-transport/tb-mqtt-transport.log

## 时序数据库部署

以下部署方案二选一：使用Cassandra或者iotdb（推荐）

### Cassandra部署

#### 1.1 环境准备

由于客户(Geely)堡垒机网络不与外部网络连通，无法从cassandra官网拉取安装包来安装，所以cassandra的安装采用docker镜像的方式部署安装。

部署环境默认为ubuntu环境，版本为20.04。

1. 安装docker

sudo apt-get update

apt install docker.io

通过docker -v检查docker是否已安装, 输出

Docker version 24.0.7, build afdd53b 表示已安装

1. 安装docker compose

sudo apt-get update

sudo apt-get install docker-compose-plugin

通过docker compose version检查版本

输出Docker Compose version v2.21.0 表示安装成功

#### 1.2部署详细

1. 准备cassandra镜像

将准备好的cass.tar.gz拷贝到目标机器/tmp目录

1. Load docker 镜像

进入cass.tar.gz所在目录，依次执行如下命令

gzip -d cass.tar.gz

docker load < cassandra\_latest.tar

1. 编写docker compose文件

考虑cassandra热备，需要在两台堡垒机上重复上面load image操作。

在两台堡垒机上分别编写docker-compose.yml文件

version: '3.0'

services:

cassandra:

restart: always

image: "cassandra:4.0.4"

ports:

- "9042:9042"

- "7199:7199"

- "7000:7000"

volumes:

- ./data:/var/lib/cassandra

environment:

CASSANDRA\_CLUSTER\_NAME: Thingsboard Cluster

CASSANDRA\_BROADCAST\_ADDRESS: {current\_node\_ip}

CASSANDRA\_SEEDS: {all\_cassandra\_nodes\_ip\_list}

其中current\_node\_ip是本机IP,

all\_cassandra\_nodes\_ip\_list是seeds的主机IP, 以”,”隔开

1. 启动cassandra服务

docker compose -f {yml direactory} up -d

通过sudo docker ps指令查看运行状态

### IotDb部署

#### 1.1 环境准备

使用docker compose方式部署。

部署环境默认为ubuntu环境，版本为20.04。

1. 安装docker

sudo apt-get update

apt install docker.io

通过docker -v检查docker是否已安装, 输出

Docker version 24.0.7, build afdd53b 表示已安装

1. 安装docker compose

sudo apt-get update

sudo apt-get install docker-compose-plugin

通过docker compose version检查版本

输出Docker Compose version v2.21.0 表示安装成功

#### 1.2部署详细

1. 准备iotdb镜像

将准备好的iotbd-standalone.tar.gz拷贝到目标机器/tmp目录

1. Load docker 镜像

进入iotbd-standalone.tar.gz所在目录，依次执行如下命令

gzip -d iotbd-standalone.tar.gz

docker load < iotbd-standalone.tar

1. 编写docker compose文件

1C1D的iotdb配置，在单节点上编写docker-compose.yml文件。

image: dockerproxy.cn/apache/iotdb:1.3.2-standalone #The image used

hostname: iotdb

container\_name: iotdb

restart: always

ports:

- "6667:6667"

- "9091:9091"

- "18080:18080"

environment:

- cn\_internal\_address=iotdb

- cn\_internal\_port=10710

- cn\_consensus\_port=10720

- cn\_seed\_config\_node=iotdb:10710

- dn\_rpc\_address=iotdb

- dn\_internal\_address=iotdb

- dn\_rpc\_port=6667

- dn\_internal\_port=10730

- dn\_mpp\_data\_exchange\_port=10740

- dn\_schema\_region\_consensus\_port=10750

- dn\_data\_region\_consensus\_port=10760

- dn\_seed\_config\_node=iotdb:10710

privileged: true

volumes:

- /home/devdata/iotdb/standalone/data:/iotdb/data

- /home/devdata/iotdb/standalone/logs:/iotdb/logs

- ./conf/iotdb-datanode.properties:/iotdb/conf/iotdb-datanode.properties

- ./conf/iotdb-common.properties:/iotdb/conf/iotdb-common.properties

- /dev/mem:/dev/mem:ro

networks:

- iotdb

prometheus:

image: dockerproxy.cn/prom/prometheus:latest

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml

ports:

- "9090:9090"

networks:

- iotdb

depends\_on:

- iotdb-service

grafana:

image: dockerproxy.cn/grafana/grafana:latest

ports:

- "3000:3000"

volumes:

- /home/devdata/grafana/data:/var/lib/grafana

networks:

- iotdb

depends\_on:

- prometheus

networks:

iotdb:

driver: bridge

1. 更新iotdb配置文件

更新iotdb-datanode.properties：

dn\_metric\_prometheus\_reporter\_port=9091

enable\_rest\_service=true

rest\_service\_port=18080

iotdb-common.properties：

timestamp\_precision=us

1. 新建promethues.yml:

# prometheus.yml

global:

scrape\_interval: 15s # 抓取指标的时间间隔

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['192.168.244.88:9090']

1. 启动iotdb

docker compose -f {yml direactory} up -d

通过sudo docker ps指令查看运行状态

## 文件服务器部署

### 5.4.1环境准备

安装docker compose

sudo apt-get update

sudo apt-get install docker-compose-plugin

通过docker compose version检查版本

输出Docker Compose version v2.21.0 表示安装成功

### 5.4.2部署详细

创建一个yml,填入如下内容

version: '3.0'

services:

minio:

restart: always

image: "minio/minio:latest"

ports:

- "9000:9000"

- "9001:9001"

volumes:

- /home/devdata/minio/data:/data

environment:

MINIO\_ROOT\_USER: 'root123'

MINIO\_ROOT\_PASSWORD: '12345678'

command: server /data --console-address ":9001"

docker-compose -f xxx.yml up -d

创建成功后，9001是web端口，9000是api端口

用户名: 'root123'

密码: '12345678'