解决方案实践

映云科技车联网数据基础设施解决方案实践

文档版本 1.0

发布日期 2023-12-07

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 方案概述

应用场景

车联网数据基础设施解决方案

随着新能源汽车数量的增加，车机管理难的问题日益突出。

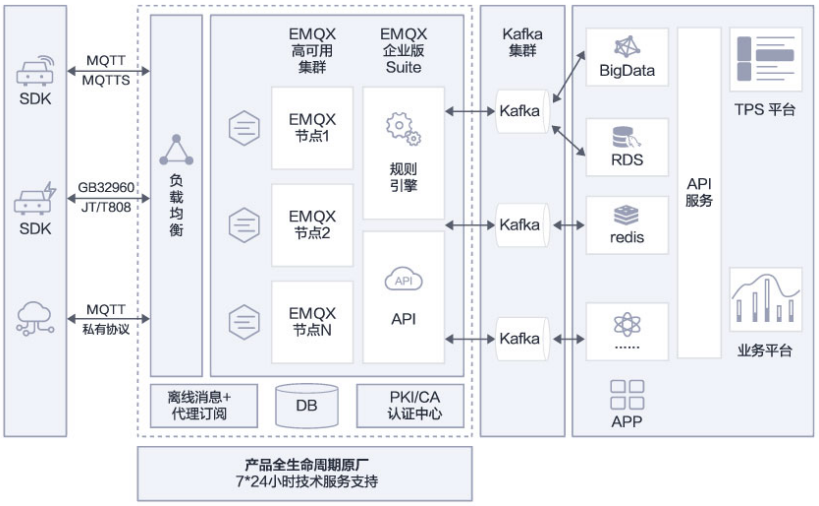
- 无法支撑海量车机系统安全接入
- 车机管理效率低下且存在众多安全问题
- 在复杂网络环境下保证消息实时性与可靠性
- 无法保障大并发、高可用消息通信
- 难以快速实现业务系统对接
- 难以支撑包含整车部分、调度系统和运维系统，需要稳定、低时延、高性能的车云数据接入服务
- 无法支持数据实时采集与分析实现生产线运行数据可视化、生产装备运行数据分析实现产线质量预警、生产预测性维护等智能制造场景

本章节将介绍基于 EMQ，构建以自动驾驶为核心的新一代车联网数据平台以支撑主机厂智能网联为核心发展战略，开启造车新时代。

方案架构

车联网数据基础设施解决方案

图 1-1 车联网数据基础设施解决方案架构



1. 基于Huawei Cloud部署EMQX Enterprise集群实现海量车端、手机app的安全接入
2. 通过规则引擎实现百万级的消息吞吐能力与应用集成
3. 轻松适配和对接各种应用集成技术栈，为用户应用提供保障
4. 通过离线消息存储、北向API接口等实现车控消息可靠、低时延双向通信

方案优势

- 高性能

  单节点支持每秒实时接收、处理与分发数百万条的 MQTT 消息，毫秒级消息时延。
- 高可用、易运维

  高可用集群支持弹性伸缩，无单点故障；支持热升级、热配置。
- 保证物联安全

  TLS/DTLS 加密协议保证数据传输安全，支持国密加密算法；支持对设备的 ACL 访问控制细粒度控制；支持基于 X.509 证书, JWT Token 认证。
- 稳固的基础设施

  华为云提供的计算、网络和存储系统为车运一体化消息传递提供了安全、坚固和灵活的基础设施服务。
- 丰富的能力服务

华为云提供基于KAFKA/Redis/RDS/大数据等多种技术栈，可同EMQX集成后形成车联网整体技术服务解决方案，助力主机厂和第三方应用开发商快速构建应用。

2 资源和成本规划

车联网数据基础设施解决方案

表 2-1 资源和成本规划

云资源	规格	数量	每月费用（元）
VPC	网段选择192.168.0.0/16，其他采用默认配置	1	00.00
Subnet	网段选择192.168.0.0/24，其他采用默认配置	1	00.00
安全组	根据需要开通入方向1883、8883等端口	1	00.00
ECS	【弹性云服务器 ECS 】X86计算 | 通用计算型 | c7.large.2 | 2核 | 4GB; CentOS | CentOS 7.8 64bit;【系统盘】通用型SSD | 40GB	3	/
Kafka	【分布式消息服务Kafka版】kafka.4u8g.cluster | 代理个数：3 单个代理存储空间: 超高IO | 200GB	1	/
ELB	【弹性负载均衡】共享型负载均衡 全动态BGP 带宽: 全动态BGP | 带宽 | 30Mbit/s	2	/
企业主机安全	【企业主机安全】企业版	1	/
Anti-DDoS流量清洗	【Anti-DDoS流量清洗】基础DDoS 防护能力，5Gbps	2	/
云备份	【云备份】云服务器备份存储库 | 100GB;	1	/
总计：/(EMQX 5万线)			

3 实施步骤

3.1 搭建 EMQX 集群环境

安装 openssl

步骤1 安装依赖gcc、perl5

```
yum install gcc perl -y
```

步骤2 安装配置openssl

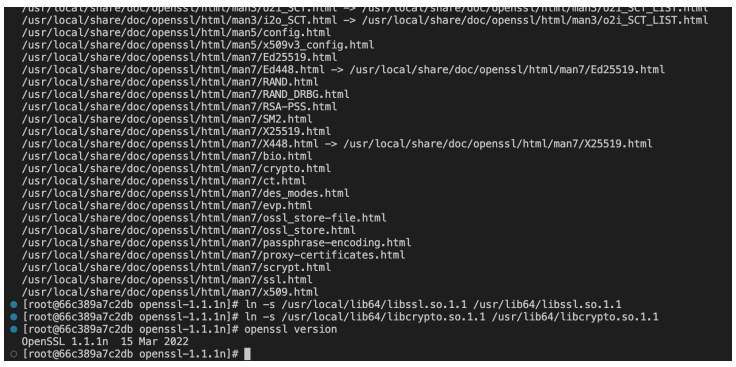
1. 下载openssl二进制包

```
wget
tar -zxvf openssl-1.1.1n.tar.gz
```

2. 手动编译并创建软连接

```
cd openssl-1.1.1n
./config
make && make install
$ ln -s /usr/local/lib64/libssl.so.1.1 /usr/lib64/libssl.so.1.1
$ ln -s /usr/local/lib64/libcrypto.so.1.1 /usr/lib64/libcrypto.so.1.1
```

图 3-1 手动编译并创建软连接



----结束

部署 EMQX

步骤1 下载EMQX企业版安装包

步骤2 安装

步骤3 运行

sudo systemctl start emqx

----结束

节点集群创建 (如修改后端 ECS 实例页面配置文件)

多节点集群，每台机器上的emqx都需要进行如下修改

步骤1 修改节点配置

```
vi /etc/emqx/emqx.conf
node.name = node1@x.x.x.x
```

步骤2 运行EMQX

```
sudo systemctl start emqx
```

步骤3 加入集群

```
emqx_ctl cluster join node2@x.x.x.x
```

图 3-2 加入集群

```
root@d2b7aacaa9f0:~# emqx_ctl cluster join node2@172.17.0.5
=CRITICAL REPORT==== 15-Dec-2022::09:56:42.391748 ===
[EMQ X] emqx shutdown for join
Join the cluster successfully.
Cluster status: #{running_nodes => ['node1@172.17.0.4','node2@172.17.0.5'],
                  stopped_nodes => []}
```

----结束

3.2 开启 redis 认证

redis 配置

步骤1 **认证**

默认配置下示例数据如下，添加一条认证：

```
HMSET mqtt_user:emqx password public
```

步骤2 添加ACL认证

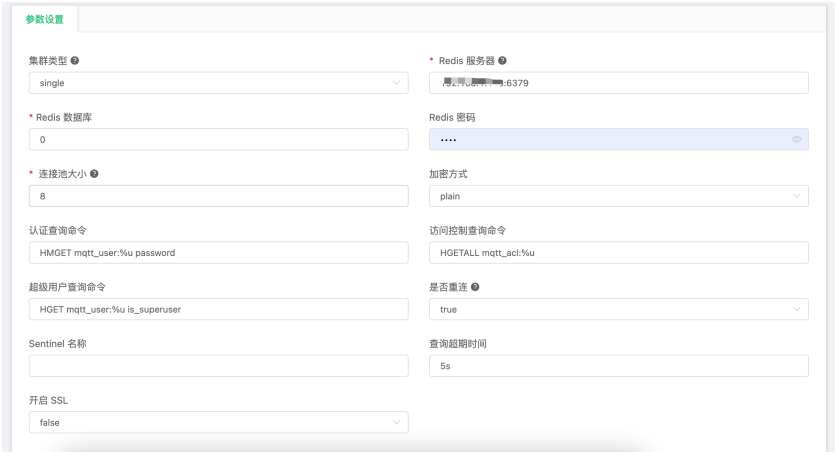
```
HSET mqtt_acl:testtopic/2 2
```

----结束

EMQX redis 认证模块配置

填写配置完成，最后单击“添加按钮”模块即可添加成功

图 3-3 EMQX redis 认证模块配置



3.3 验证操作是否成功

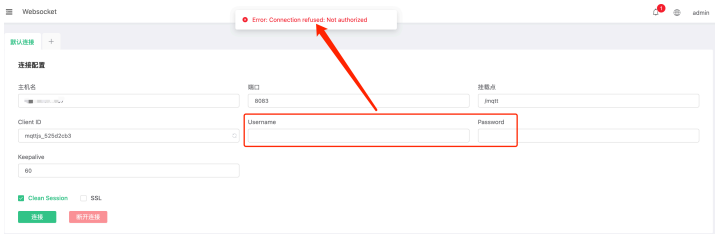
步骤1 关闭匿名登录

图 3-4 关闭匿名登录

步骤2 客户端连接

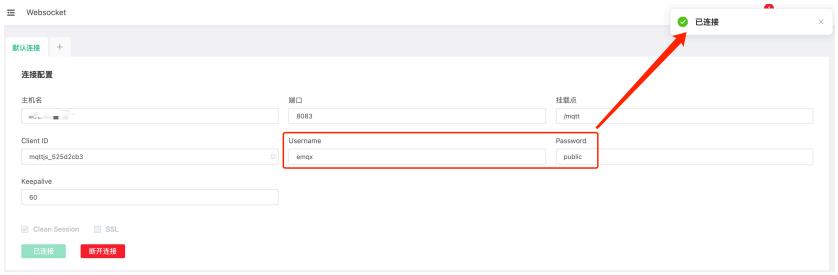
错误用户名密码客户端：

图 3-5 客户端连接



正确用户名密码客户端：

图 3-6 客户端连接



----结束

4 附录

4.1 源代码

redis 认证

```
-module(emqx_authz_redis).
-include("emqx_authz.hrl").
-include_lib("emqx/include/emqx.hrl").
-include_lib("emqx/include/logger.hrl").
-include_lib("emqx/include/emqx_placeholder.hrl").
-behaviour(emqx_authz).
%% AuthZ Callbacks
-export([
    description/0,
    create/1,
    update/1,
    destroy/1,
    authorize/4
]).
-ifdef(TEST).
-compile(export_all).
-compile(nowarn_export_all).
-endif.
-define(PLACEHOLDERS, [
    ?PH_CERT_CN_NAME,
    ?PH_CERT_SUBJECT,
    ?PH_PEERHOST,
    ?PH_CLIENTID,
    ?PH_USERNAME
]).
description() ->
    "AuthZ with Redis".
create(#{cmd := CmdStr} = Source) ->
    Cmd = tokens(CmdStr),
    ResourceId = emqx_authz_utils:make_resource_id(?MODULE),
    CmdTemplate = emqx_authz_utils:parse_deep(Cmd, ?PLACEHOLDERS),
    {ok, _Data} = emqx_authz_utils:create_resource(ResourceId, emqx_connector_redis, Source),
    Source#{annotations => #{id => ResourceId}, cmd_template => CmdTemplate}.
update(#{cmd := CmdStr} = Source) ->
    Cmd = tokens(CmdStr),
    CmdTemplate = emqx_authz_utils:parse_deep(Cmd, ?PLACEHOLDERS),
    case emqx_authz_utils:update_resource(emqx_connector_redis, Source) of
```

```
        {error, Reason} ->
            error({load_config_error, Reason});
        {ok, Id} ->
            Source#{annotations => #{id => Id}, cmd_template => CmdTemplate}
    end.
destroy(#{annotations := #{id := Id}}) ->
    ok = emqx_resource:remove_local(Id).
authorize(
    Client,
    PubSub,
    Topic,
    #{
        cmd_template := CmdTemplate,
        annotations := #{id := ResourceID}
    }
) ->
    Cmd = emqx_authz_utils:render_deep(CmdTemplate, Client),
    case emqx_resource:query(ResourceID, {cmd, Cmd}) of
        {ok, []} ->
            nomatch;
        {ok, Rows} ->
            do_authorize(Client, PubSub, Topic, Rows);
        {error, Reason} ->
            ?SLOG(error, #{
                msg => "query_redis_error",
                reason => Reason,
                cmd => Cmd,
                resource_id => ResourceID
            }),
            nomatch
    end.
do_authorize(_Client, _PubSub, _Topic, []) ->
    nomatch;
do_authorize(Client, PubSub, Topic, [TopicFilter, Action | Tail]) ->
    case
        emqx_authz_rule:match(
            Client,
            PubSub,
            Topic,
            emqx_authz_rule:compile({allow, all, Action, [TopicFilter]})
        )
    of
        {matched, Permission} -> {matched, Permission};
        nomatch -> do_authorize(Client, PubSub, Topic, Tail)
    end.
tokens(Query) ->
    Tokens = binary:split(Query, <<" ">>, [global]),
    [Token || Token <- Tokens, size(Token) > 0].
```

4.2 常见问题

SSL 连接失败

现象：客户端无法与 EMQX 建立 SSL 连接。

解决方法：可以借助 EMQX 日志中的关键字来进行简单的问题排查，EMQX 日志相关内容请参考：日志与追踪。

1. certificate_expired

   日志中出现 certificate_expired 关键字，说明证书已经过期，请及时续签。

2. no_suitable_cipher

   日志中出现 no_suitable_cipher 关键字，说明握手过程中没有找到合适的密码套件，可能原因有证书类型与密码套件不匹配、没有找到服务端和客户端同时支持的密码套件等等。

3. handshake_failure

   日志中出现 handshake_failure 关键字，原因有很多，可能要结合客户端的报错来分析，例如，可能是客户端发现连接的服务器地址与服务器证书中的域名不匹配。

4. unknown_ca

   日志中出现 unknown_ca 关键字，意味着证书校验失败，常见原因有遗漏了中间 CA 证书、未指定 Root CA 证书或者指定了错误的 Root CA 证书。在双向认证中可以根据日志中的其他信息来判断是服务端还是客户端的证书配置出错。如果是服务端证书存在问题，那么报错日志通常为：

   {ssl_error,{tls_alert,{unknown_ca,"TLS server: In state certify received CLIENT ALERT: Fatal - Unknown CA\n"}}}

   看到 CLIENT ALERT 就可以得知，这是来自客户端的警告信息，服务端证书未能通过客户端的检查。

   如果是客户端证书存在问题，那么报错日志通常为：

   {ssl_error,{tls_alert,{unknown_ca,"TLS server: In state certify at ssl_handshake.erl:1887 generated SERVER ALERT: Fatal - Unknown CA\n"}}}

   看到 SERVER ALERT 就能够得知，表示服务端在检查客户端证书时发现该证书无法通过认证，而客户端将收到来自服务端的警告信息。

5. protocol_version

   日志中出现 protocol_version 关键字，说明客户端与服务器支持的 TLS 协议版本不匹配。

5 修订记录

表 5-1 修订记录

发布日期	修订记录
2023-2-15	第一次正式发布。