C1 Module Proxy

Introduction

C1 Core requires a constant TCP connection for bidirectional communication. This is hard to do using standard webservers like `Nginx` or `Apache`. C1 Module Proxy is a small service taking over exactly this task. It connects to C1 Core and handles all communication with it.

Installation

Using Docker

C1 Module Proxy provides a Docker template image with Ubuntu Focal and C1 Module Proxy preinstalled. You can pull this image with:

```
docker pull gitit.de:5005/sensaru/c1/c1-module-proxy:latest
```

Within the image you just need to modify the settings and files in `/etc/c1-module-proxy/` (or mount them).

Kubernetes

Here's a template you can use to deploy c1-module-proxy in Kubernetes. A secret with the C1 Core client certificates must exist in the namespace in `c1-core-client-tls`:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: c1-module-proxy
  name: c1-module-proxy
  namespace: <your namespace>
spec:
  clusterIP: None
  ports:
    - port: 80
  selector:
    app: c1-module-proxy
--
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: c1-module-proxy
  namespace: <your namespace>
spec:
```

```
  replicas: 1
  minReadySeconds: 60
  selector:
    matchLabels:
      app: c1-module-proxy
  strategy:
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: c1-module-proxy
        network-policy: permit-local-traffic
    spec:
      containers:
      - name: c1-module-proxy
        image: gitit.de:5005/sensaru/c1/c1-module-proxy:latest
        ports:
        - containerPort: 80
          protocol: TCP
        livenessProbe:
          failureThreshold: 3
          initialDelaySeconds: 60
          periodSeconds: 30
          successThreshold: 1
          tcpSocket:
            port: 80
          timeoutSeconds: 1
        readinessProbe:
          failureThreshold: 3
          initialDelaySeconds: 5
          periodSeconds: 30
          successThreshold: 1
          tcpSocket:
            port: 80
          timeoutSeconds: 1
        imagePullPolicy: Always
        volumeMounts:
        - name: c1-module-proxy-config
          mountPath: /etc/c1-module-proxy/main.conf
          subPath: main.conf
          readOnly: true
        - name: c1-core-client-tls
          mountPath: /etc/c1-module-proxy/c1-core-client-tls
          readOnly: true
      imagePullSecrets:
      - name: gitit-de
      volumes:
      - name: c1-module-proxy-config
        configMap:
          defaultMode: 288
          optional: false
          name: c1-module-proxy-config
      - name: c1-core-client-tls
        secret:
```

```
        defaultMode: 292
        optional: false
        secretName: c1-core-client-tls
---
apiVersion: v1
kind: ConfigMap
binaryData:
  main.conf: <base64-encoded main.conf>
metadata:
  name: c1-module-proxy-config
  namespace: <your namespace>
```

Install manually

Requirements

C1 Module Proxy requires

- `libhomegear-base` and
- `libc1-module`.

In addition

- `GnuTLS` and
- `GCrypt`

are required.

`libhomegear-base` can be downloaded as a Debian package for Debian and Ubuntu from https://downloads.homegear.eu/nightlies/. `libc1-module` needs to be compiled manually.

Compilation of `libc1-module` requires `g++`, `automake` and `libtool`. Under Debian and Ubuntu install the packages `automake, libtool and build-essential`.

To compile `libc1-module`, clone the repository and execute:

```
cd /path/to/libc1-module
./makeRelease.sh
```

`GnuTLS` and `GCrypt` can be installed by installing the packages `gnutls-dev` and `libgcrypt-dev`.

`c1-module-proxy` requires `cmake` for compilation, so this package needs to be installed as well.

Compilation

To compile C1 Module Proxy, clone the repository and execute:

```
cd /path/to/c1-module-proxy
mkdir build
```

```
cd build
cmake ..
make -j4
strip c1-module-proxy
cp c1-module-proxy /usr/bin
cd ..
```

Now create the settings and log directory and copy the example configuration:

```
mkdir /etc/c1-module-proxy
cp misc/* /etc/c1-module-proxy
mkdir /var/log/c1
```

Create a user, e. g. `c1` to run the service as user. Then set permissions on the log directory:

```
adduser --system --no-create-home --shell /bin/false --group c1
chown c1:c1 /var/log/c1
```

Configure module

Module configuration is also done in C1 Module Proxy. You can find the required entries in `/etc/c1-module-proxy/main.conf` in section `Module information`:

```
moduleName = {"de-DE": "Mein Modul", "en-US": "My module"}
moduleDescription = {"de-DE": "", "en-US": ""}
moduleUiMenu = {"label":{"de-DE":"Mein Modul","en-US":"My module"},"icon":"fontawesome/fa-heartbeat","url":"https://my-module.sensaru.net"}
moduleUiCategory =
moduleAppUiUrl =
moduleAppUiCategory =
moduleAclInfo = {"version":1,"rpcMethods":{"method1":"read","method2":"write"}}
moduleEventSubscriptions = ["c1-core/bp/+/au/+/association"]
```

`moduleName` and `moduleDescription` are self-explaining.

`moduleUiMenu` is only required for modules providing an user interface for C1 Core. This creates a menu entry in C1 Core with the provided information.

`moduleUiCategory` specifies under which category or heading the menu entry is sorted in.

`moduleAppUiUrl` is only required for modules that provide an user interface for the inhabitant smartphone app. The specified page is opened when the module is selected in the app.

`moduleAppUiCategory` specifies under which category the app entry is placed.

`moduleAclInfo` is mandatory and contains the list of all RPC methods implemented by the module. Every method must be added to the property `rpcMethods` and requires one of:

- `read`: To execute this method, read permissions are required.

- `write` : To execute this method, write permissions are required.
- `event` : To execute this method, event permissions are required.
- `admin` : To execute this method, administrative permissions are required.

It is not required to specify REST methods here. REST methods are associated to the permissions above by their HTTP method. Only `read` and `write` are supported at the moment for REST methods.

Warning

When the ACL info is updated after the first registration, a restart of C1 Core is required after the update to the ACL info. The reason is that C1 Core regards this info as static information and caches it in memory.

`moduleEventSubscriptions` contains all topics the module wants to subscribe to. See Events.

First connection

On first connection the module is registered to Sensaru Cloud. To be able to connect there is one more thing required: A client certificate. The client certificate needs to be requested from one of the certificate administrators. To create the certificate a module ID (e. g. `c1-device-management`) needs to be defined. This ID is part of the common name of the signed certificate and this ID is used to identify the module within Sensaru Cloud.

Note

The common name is a base64-encoded JSON. The JSON looks like this:
`{"type":"module","id":"c1-device-management","index":1,"date":1578005399878,"version":1,"environment":"dev"}`

Place the certificate and key in the locations specified in `/etc/c1-module-proxy/main.conf` (parameters `coreClientCert` and `coreClientKey`).

Warning

Never ever upload these certificates to a code repository and never ever store them within Docker images or containers. To use them within containers, use mounts.

High availability and scalability

Any module is required to be able to run in multiple instances over multiple datacenters. C1 Module Proxy and C1 Core already do a lot of the backend work, so you can start multiple instances of C1 Module Proxy. When multiple module instances are available, C1 Core randomly selects a module instance to direct requests to.

RPC

Requests originating from C1 Core

Your application needs to listen on the hostname and port specified in `/etc/c1-module-proxy/main.conf` (parameters `localRpcHost` and `localRpcPort`) and handle JSON-RPC requests on an arbitrary path. The path C1 Module Proxy sends the requests to can be specified in `/etc/c1-module-proxy/main.conf` (parameter `localRpcPath`).

Every request coming from C1 Core is forwarded to your application. The response from your application is forwarded back to C1 Core.

Requests originating from your application

When you want to call a RPC method on C1 Core or another module, send this request to the listen address and the listen port specified in `/etc/c1-module-proxy/main.conf` (parameters `httpListenAddress` and `httpListenPort`). The used path is irrelevant.

Calling moduleMethod

In contrast to the description here, when calling `moduleMethod` the response is returned directly when C1 Module Proxy is used. This makes usage much easier. The underlying call is still asynchronous.

More information

For more information read the section about inter-module communication.

REST

First read the section about module REST API's.

C1 Module Proxy takes the tunneled RPC call and translates it back to a REST call including HTTP headers. The REST request is called on the same hostname and port as specified in `localRpcHost` and `localRpcPort` in `/etc/c1-module-proxy/main.conf`.

The RPC metadata Struct which includes for example the logged in user and principal is passed as a base64-encoded JSON in the header field `C1-VERIFIED-METADATA`. Like this you can control data access and only return data the user has access to.

An example header might look like this:

```
C1-VERIFIED-METADATA:
eyJpYnNIb21lQ2xpZW50SWQiOiIiLCJpYnNIb21lQ2xpZW50VXNlcnMiOltdLCJzb3VyY2VNb2R1bGVJZCI6Imlicy1jb3
```

The decoded base64 from above contains the following JSON:

```
{
 "homeClientId": "",
 "homeClientUsers": [],
 "sourceModuleId": "c1-core",
 "userHomeClients": {
  "active": {},
  "all": {}
 },
 "userId": {
  "bp": "",
  "id": "157d9350-1db8-11e9-8e66-2f71a0be4cc5",
  "sd": "",
  "sp": "48109350-1db6-11e9-8e66-2f71a0be4cc5",
  "type": 1
 }
}
```

See the chapter about verified metadata for more information.

Authorization

It is mandatory that the logged-in user is only able to access data from it's own principal and data of principals that are hirarchically below the user's principal level (e. g. a system distributor user is allowed to access data from the system distributor's business partners).

Apart from that a user can be granted or denied permission

1. per RPC method or
2. per REST method/path combination.

Warning

Never ever allow a user access to data outside of it's own principal scope!

Permission types

There are four possible permission types that can be assigned to a RPC method:

1. `read`
2. `write`
3. `event`
4. `isAdmin`

For REST currently no permission types can be assigned manually to REST methods or paths. This is on our issue list. Currently permissions are set automatically in C1 Core. See below.

RPC permissions

The per RPC method permission is given by the group ACLs assigned to the user in C1 Core. This permission is checked within C1 Core, so there are no checks required within the module.

For this to work, the module must pass ACL information to C1 Core upon module registration. This ACL information is a JSON defined in `main.conf` in C1 Module Proxy (setting `moduleAclInfo`, see above).

REST permissions

REST permission types are assigned automatically based on the HTTP `method`. The association is as follows:

HTTP method	Permission type
`DELETE`	`write`
`GET`	`read`
`PATCH`	`write`
`POST`	`write`
`PUT`	`write`

The permission is checked within C1 Core, so there are no checks required within the module.