Starter Kit Plus

Demo kit with smart components and periMICA

User Guide

Perinet
Seamless IoT Connectivity

Abstract

This user guide presents the *demo* application delivered with the Starter Kit Plus, explains the Zero Configuration network and the security infrastructure applied between all the components which are controled by PKI2go. The intend is to help the user to use and configure the periNODE smart components and the periMICA containers included in the kit and setup a new secured application or rollback with the default configuration.

Doc-No.: PRN.100.548 rev: 2 2022-09-30

Document Information

Title	Starter Kit Plus
Subtitle	Demo kit with smart components and periMICA
Type	User Guide
Status	Release
Version	2
Date	2022-09-30
Disclosure Restriction	

Contents

1 Overview

The purpose of the Perinet Starter Kit Plus is to present a complete example *application* on how to collect field sensor data via periNODE distance smart adapter, how to use the periNODE GPIO to send commands on the network to control the status of a contactor and finally to present the distributed *DistanceControl* periMICA container which is an example on how to use the IIoT environment to automate a simple task.

The *application* is taking advantage of a security infrastructure, and it is also possible to the allowed users to visualize all the information with the help of periMICA containers (Figure 1).

Figure 1: Dashboard Starter Kit Plus

1.1 Demo Application

The distance sensor values are received by *periNODE distance smart adapter* (Figure 2) and commands to turn on/off can be sent to the contactor using the *periNODE GPIO smart adpter* (Figure 3). The data communication is made via MQTT protocol using the MQTT Broker container running in *periMICA edge computer*.

Figure 2: periNODE 0-10V with attached distance sensor

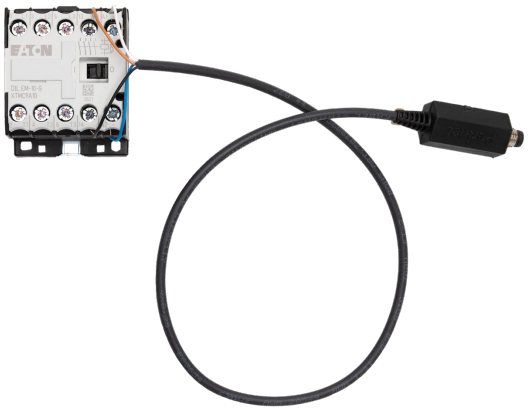

Figure 3: periNODE GPIO with attached contactor

Figure 4: Bottom view of periMICA edge computer

A periMICA edge computer (Figure 4) is used for receiving and collocating the data through a MQTT broker container, that serves the data to a Dashboard container working as a MQTT subscriber client. On reception, the Dashboard container stores the sensor data inside an InfluxDB database, finally preparing it to be visualized with the built-in Grafana tool.

The *periNODE GPIO smart adapter* is subscribed to the broker and interprets the messages coming which can switch the contactor status (on/off) for the OUTPUT port and show the status of the contactor in case of an INPUT configured port.

Additionally, the *DistanceControl* container is running a python application developed to read the sensor values, switches on the contactor when the distance value read in the sensor is less than 100 mm and switches off otherwise. This is a practical example on how to automate a simple task using the IIoT protocols like mDNS, MQTT and using the *Perinet* API to read and communicate commands to the contactor included in the Starter Kit Plus.

A group of components, periNODEs and periMICA containers working for the same purpose, is called an *application*. In the Starter Kit Plus, the pre-configured application is named *demo*.

Each participating application component, like periMICA container or periNODE smart adapter, can be directly accessed via its own HTTP web user interface (Figure 5). This UI allows monitoring sensor information, checking the status or sending the commands to switch the OUTPUT port as well as re-configuring the application.

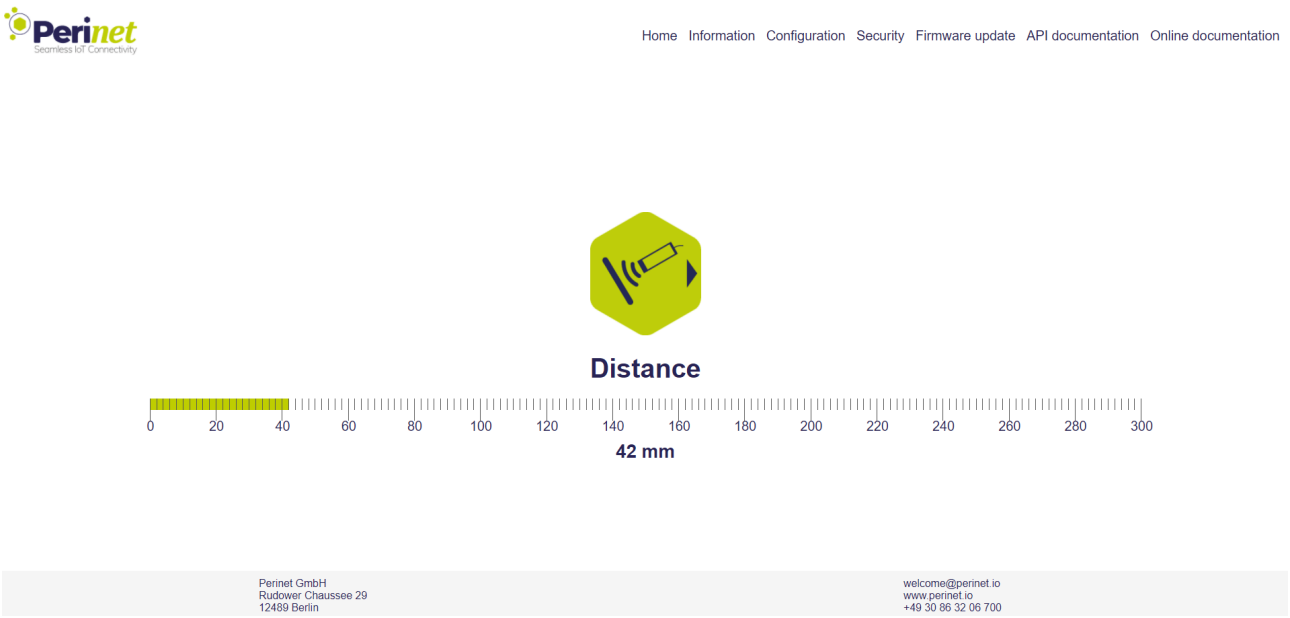

Figure 5: periNODE distance homepage

The network communication through both, MQTT and HTTP protocols is secured with the help of mutual TLS (mTLS), which means that the web servers extend the TLS handshake by requesting the clients to authenticate themselves with certificates. It can be seen as an authentication of the client to access certain resources of the corresponding web service.

For this reason, the *GettingStarted* container delivers a Web UI to provide client certificates, in order to allow the access of each application component. Hence, the *GettingStarted* container is the starting point and the only application component that can be accessed without client certificates.

After the client certificates are installed successfully into the browser, the data can be viewed by accessing the Grafana website, provided through the *Dashboard* container Web UI.

1.2 Zero Configuration Networking

A seamless configuration approach is desired. Therefore the communication between periNODEs and containers is based on IPv6 link-local addresses with the help of multicast DNS (mDNS) and DNS service discovery (DNS-SD). After wiring all components together, each single device can be accessed directly via the browser by using the mDNS hostname:

```
https://<device-name>.local
```

Although periNODEs and periMICA containers are IPv6-only devices, periMICA allows IPv4 communication as well. In case IPv4-only clients need to access the smart adapters or containers, the periMICA forwards the traffic to them through domain forwarding.

Please see the *periMICA User Guide* [14] for more details on the domain forwarding feature.

1.3 Dashboards Visualization

After logging in to the periMICA and successfully installing the required client certificates (as instructed in the *Starter Kit Plus Quick Start Guide* [24]), the access to the Grafana dashboards can be done by clicking on the *Open dashboard* button in the Dashboard container Web UI:

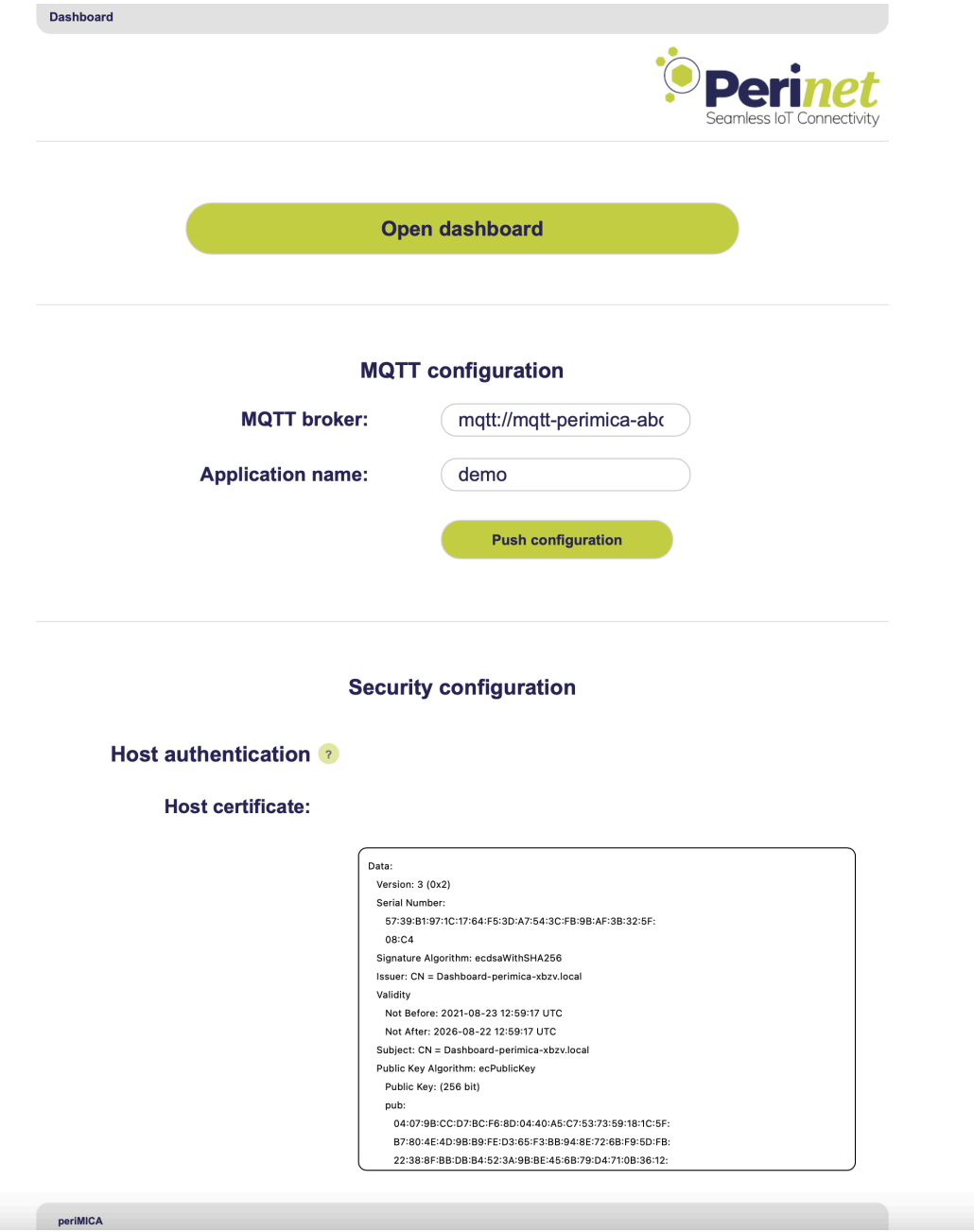

Figure 6: Access Grafana dashboards

For more details on the dashboards, please refer to the *Dashboard Container User Guide* [3].

1.4 DistanceControl Container

Another component of *application demo* is the DistanceControl container which is developed to verify each distance value read by the sensor and switches on/off the contactor automatically. The initial state of the container is *active*, so the user can easily check the functionality placing an obstacle in front of the sensor or removing it and it would be possible to hear the sound of contactor switching the state. It can be checked on the periNODE GPIO WebUI as well.

This container is presented to show an example on how to use the Starter Kit Plus components and the IIoT environment in order to automate a task and it is better explained in Section 4.

2 Security with PKI2go

Application-specific Certificate Authority (CA), a root of a Public Key Infrastructure (PKI), can be generated without an in-house PKI. We are eliminating the complexity and costs of a PKI build-out, by providing a proof of concept PKI2go with the Starter Kit Plus. The PKI2go is a patent-pending security service that Perinet offers in form of a periMICA container, to secure a complete IoT application, as represented in Figure 7:

Figure 7: PKI2go provides certificates for each application component and users

The PKI2go provides its own application Certificate Authority (CA) for creating and signing root, host and client certificates. Furthermore, users can register to the application with specific user roles and corresponding client certificates. Administrators can use this mechanism to grant different levels of authorization to specific users.

For further information on PKI2go, please refer to the *PKI2go Container User Guide* [20].

2.1 Host Management

Hosts, like Perinet Smart Components or periMICA containers, will be discovered in the network by the *PKI2go* container and are displayed in its Web UI in the *Hosts discovered* list (Figure 8):

Figure 8: PKI2go container overview

By selecting a host and pressing the "> > >" button, the host will be added to the application.

With this step, the following settings will be enabled/applied automatically to the added host:

Root certificate: The certificate that identifies the root CA (Certificate Authority), which is trusted by a periNODE or a container. It is used by the periNODE/container to authenticate remote clients (e.g. web browser) before they connect to it, when mutual TLS (mTLS) has been enabled.

Host certificate: A unique certificate used by the periNODE/container to authenticate itself when communicating with a remote client (e.g. a web browser). It holds the host name and, for consistency purposes, it has been issued by a trusted root CA. This certificate proves the originality of the host itself.

Client certificate: The certificate used by a periNODE/container to authenticate itself when connecting as a client to a remote server. It is used, for example, when the periNODE/-container is connecting to an MQTT broker.

mTLS: Required in order to grant access to host servers only for clients that provide valid client certificates.

Application name: Required by the *PKI2go* to identify which hosts are part of the application.

2.2 User Management

After the initialization of the *PKI2go* Root CA, the container always provides a default user named **application_admin_user**, with corresponding client certificates that gives full access rights. This is necessary in order to access the PKI2go after it has been initialized, as it secures its https webserver using mTLS. In the case of the Starter Kit Plus, these client certificates can be downloaded from the *GettingStarted* container.

However, other users can be created by clicking on the *Add new user* button (Figure 8), allowing user access control based on certain levels of permissions:

Reader: Only allowed to read information.

Super: Allowed to configure basic settings of periNODE smart adapters or periMICA containers.

Admin: Full access, changing security configuration and even the right for firmware updates of periNODE smart adapters.

Note: After creating a new user, a package *.zip* file is provided. The bundle *.p12* file inside the package is protected by password which is "*password*" by default.

3 Configure and Reset Application

3.1 Configure periNODE Smart Adapters

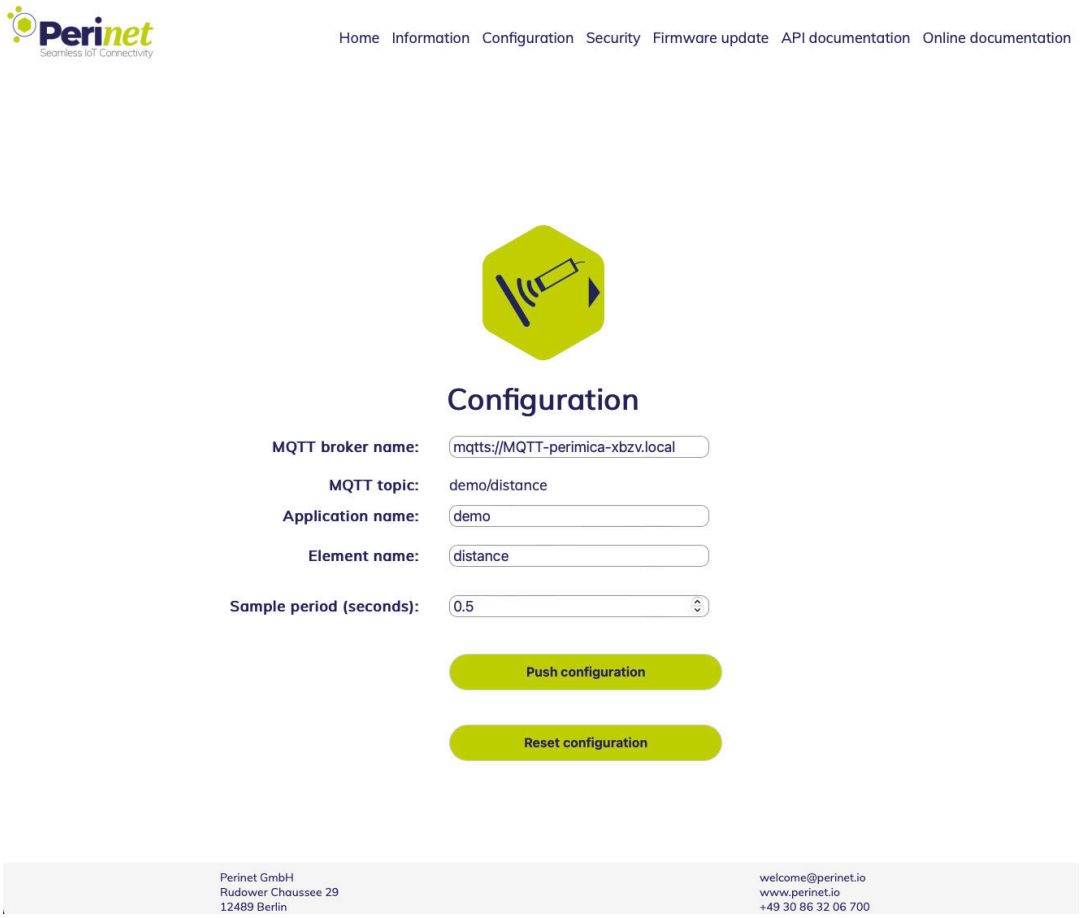

Figure 9: periNODE distance configuration web page

An example configuration for the periNODE distance sensor is shown in Figure 9. The *Application name* defines a subset of IoT devices that belong to one application. It allows to structure elements that belong together and, in the same time, it defines the prefix of the MQTT topics under which periNODE smart adapters publish information. The MQTT protocol allows the use of this prefix as general topic, to subscribe to the data that belongs in one application (*demo/#* in our example).

However, in the Starter Kit Plus, we recommended to change the *Application name* only when non-mTLS HTTPS and MQTT connections should be established, because the periNODEs are part of the infrastructure managed by the *PKI2go*.

The *Element name* mostly refers to the actual periNODE in use, e.g. for the Starter Kit Plus, "distance" is used for periNODE distance smart adapater and GPIO 1 / GPIO 2 distinguishing the periNODE GPIO smart adapter ports. It defines the MQTT topic suffix under which the periNODE smart adapter publishes its messages.

In order to change the frequency of the sensor data transfers, the parameter *Sample period*

(seconds) represents the interval between consecutive messages in seconds. Therefore, if the *Sample period (seconds)* is set to 0.5, the periNODE smart adapter sends an MQTT message every 0.5 seconds.

The configuration for periNODE GPIO smart adapter is shown in the Figure 10. The *Application name* is also present and it works the same as already presented in the periNODE distance sensor. The different in the periNODE GPIO configuration is that there are two *Element names* because each one refer to an INPUT/OUTPUT periNODE port.

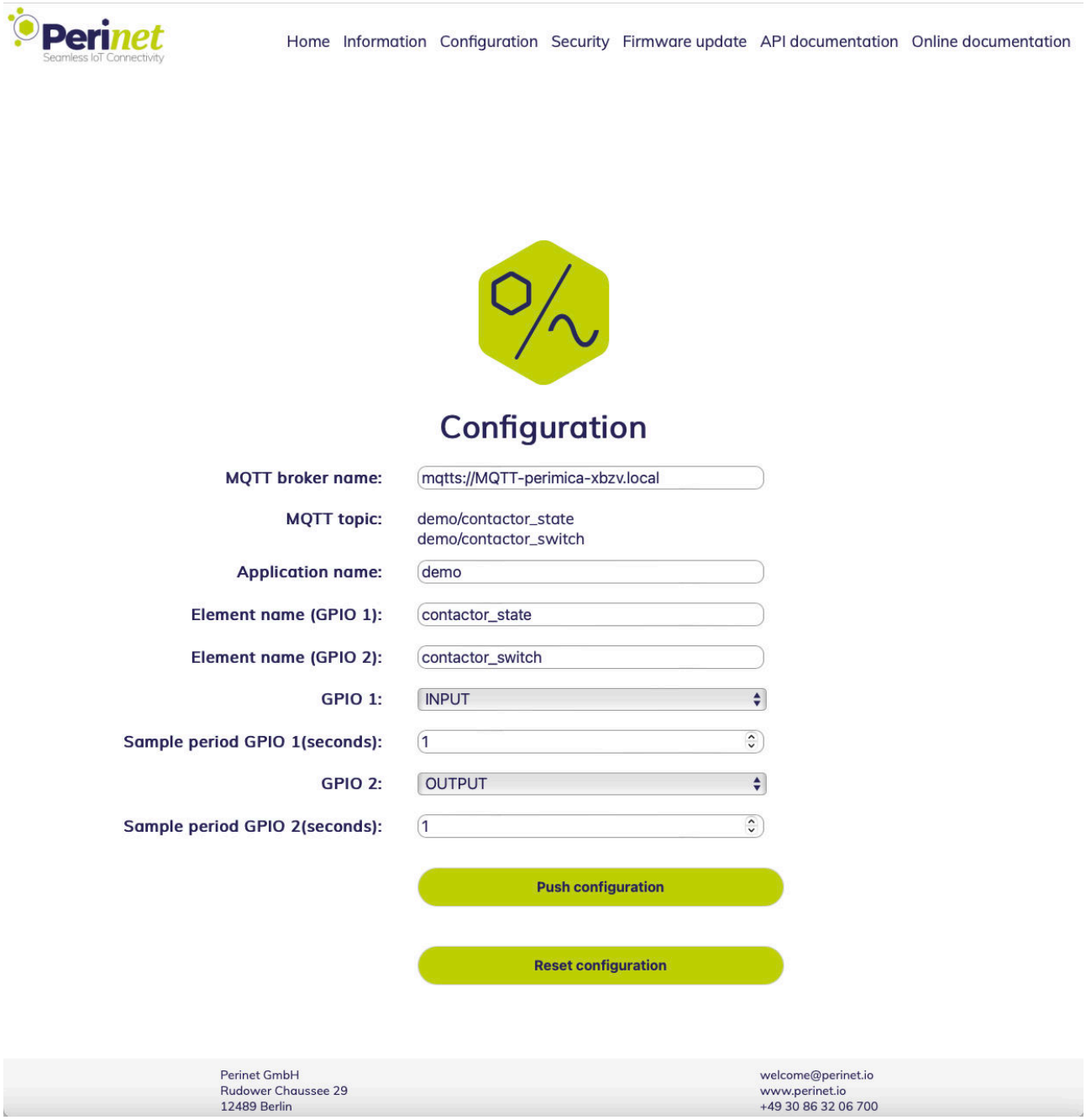

Figure 10: periNODE GPIO configuration web page

In the Starter Kit Plus there is the GPIO 1 configured as an INPUT which means the periNODE

will receive the status of the contactor (On/Off) and GPIO 2 is configured as an OUTPUT and it can be used to switch the status of the contactor. The status of the OUTPUT port can be set pressing the button in the home page of the *periNODE GPIO smart adapter* as shown in the Figure 11.

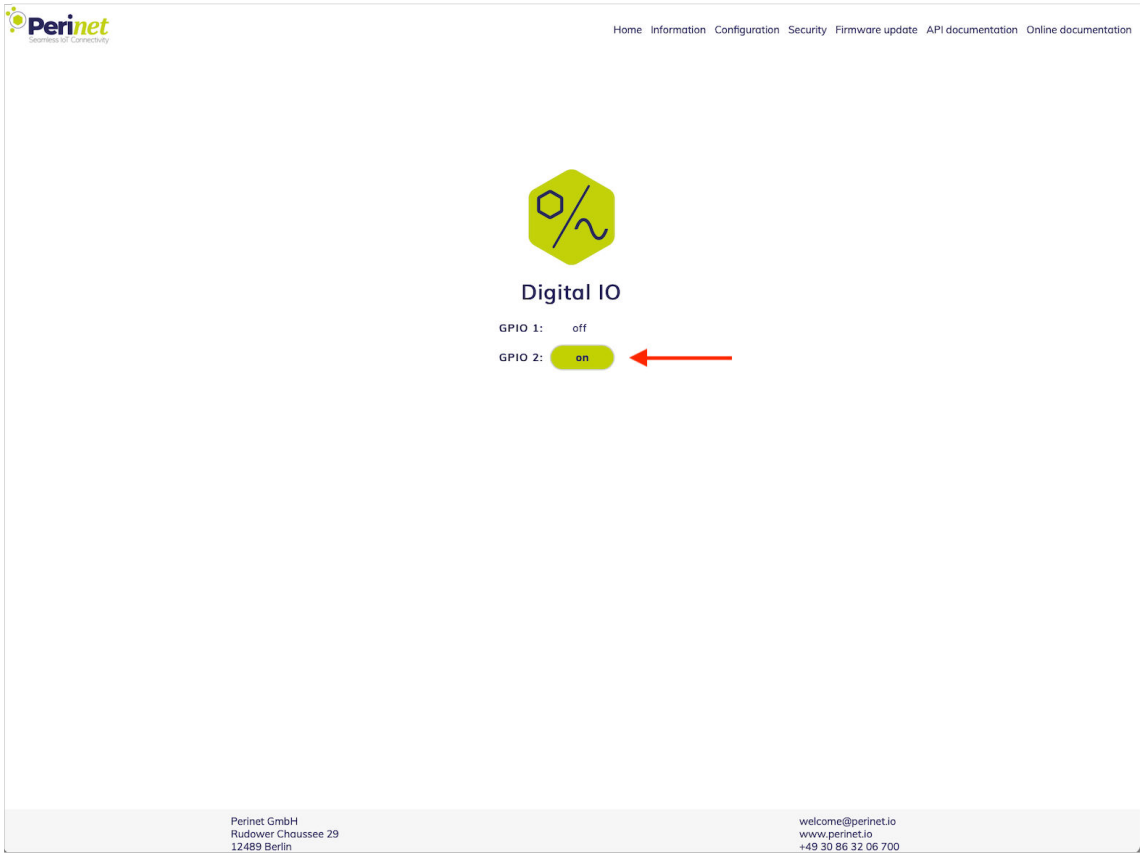

Figure 11: periNODE GPIO Home page

3.2 Setup a New Application

In order to get a better understanding on how to setup your own secure application from scratch, this section presents a guided step by step installation of a new application with the Starter Kit Plus.

1. The already configured periNODEs need to be removed from the *demo* application, because they will be used in your new application. This means that you need to reset the configuration (including security) of the two periNODE smart adapters through their Web UI. Therefore, please navigate to the *Configuration* page (Figure 12) and click on the *Reset configuration* button (Figure 13). Next, go to *Security* and click on *Reset security*. Repeat these steps for both periNODEs:

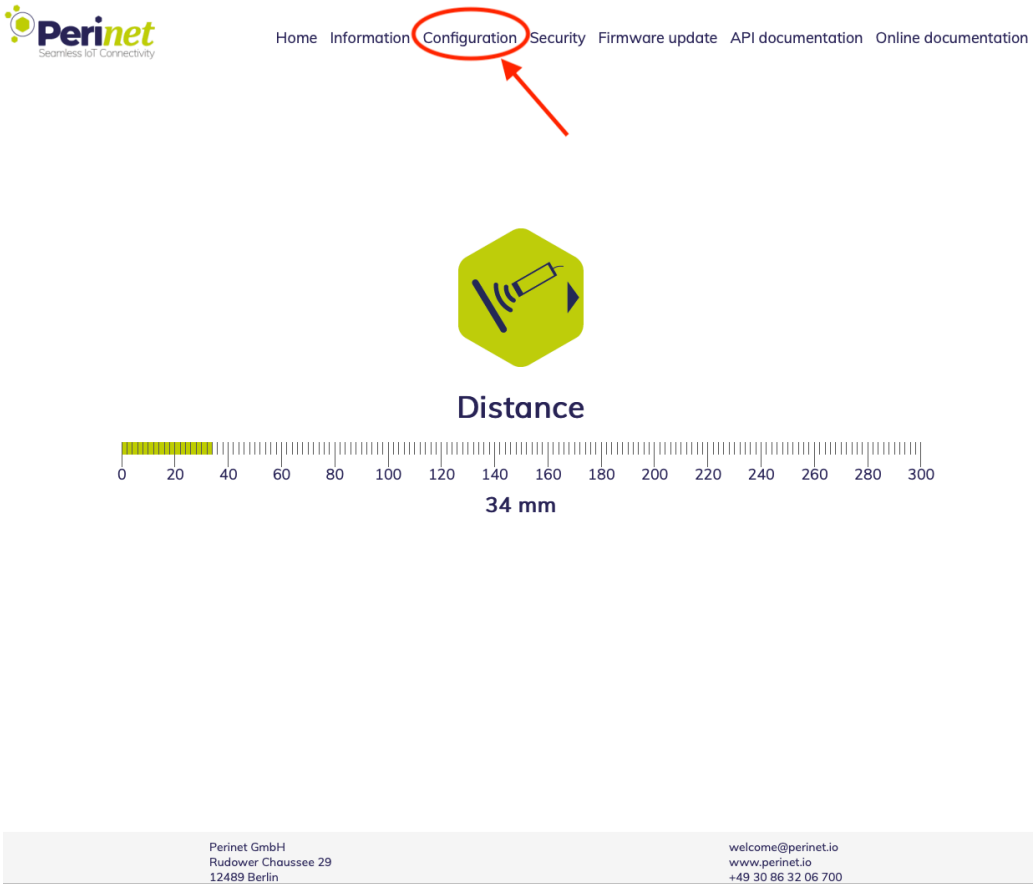

Figure 12: Go to *Configuration* on the periNODE Web UI

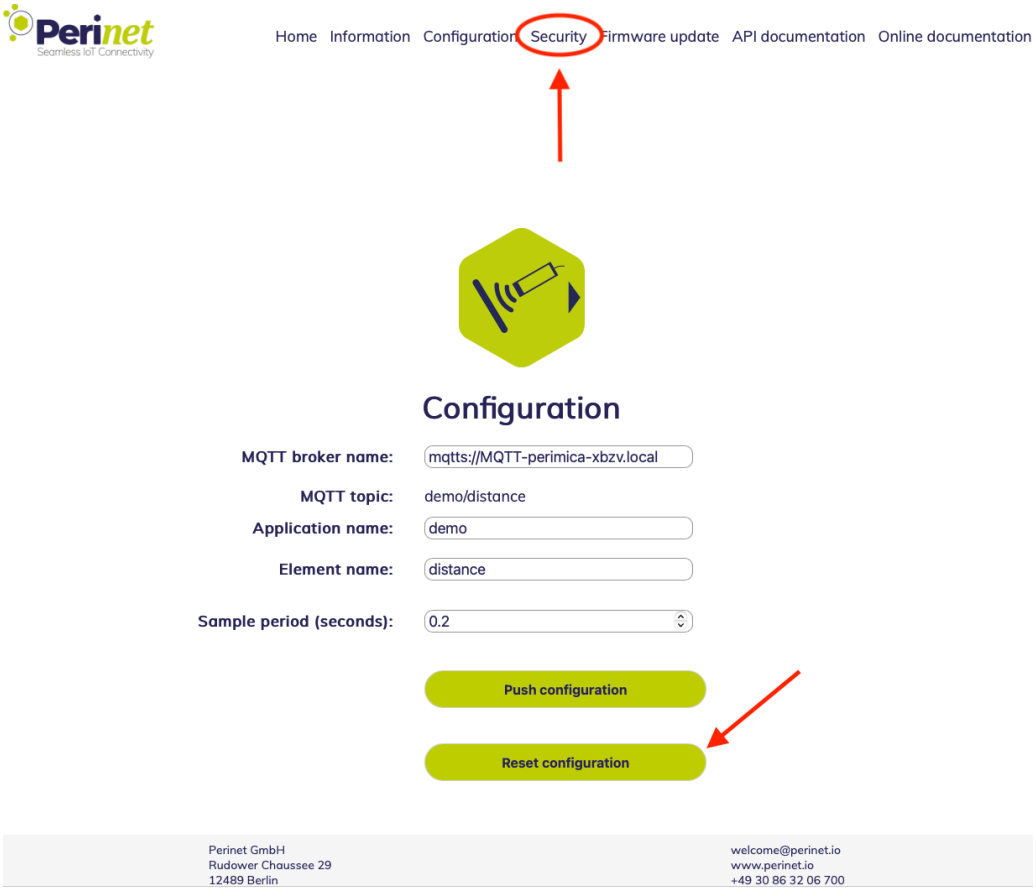

Figure 13: Reset configuration and next go to *Security*

Figure 14: Reset security

2. The containers installed by default on the periMICA included in the Starter Kit Plus cannot be deleted and a reset would put the containers back to Starter Kit Plus default factory state. Therefore, they need to be reinstalled. Please go to `https://downloads.perinet.io` and download the four containers: PKI2go, MQTT, Dashboard and DistanceControl.
3. Install the containers by logging into the periMICA and by clicking on the *Install* icon (see Figure 15). On the periMICA *Install* webpage, upload and install the downloaded container *.tar* files (Figure 16).

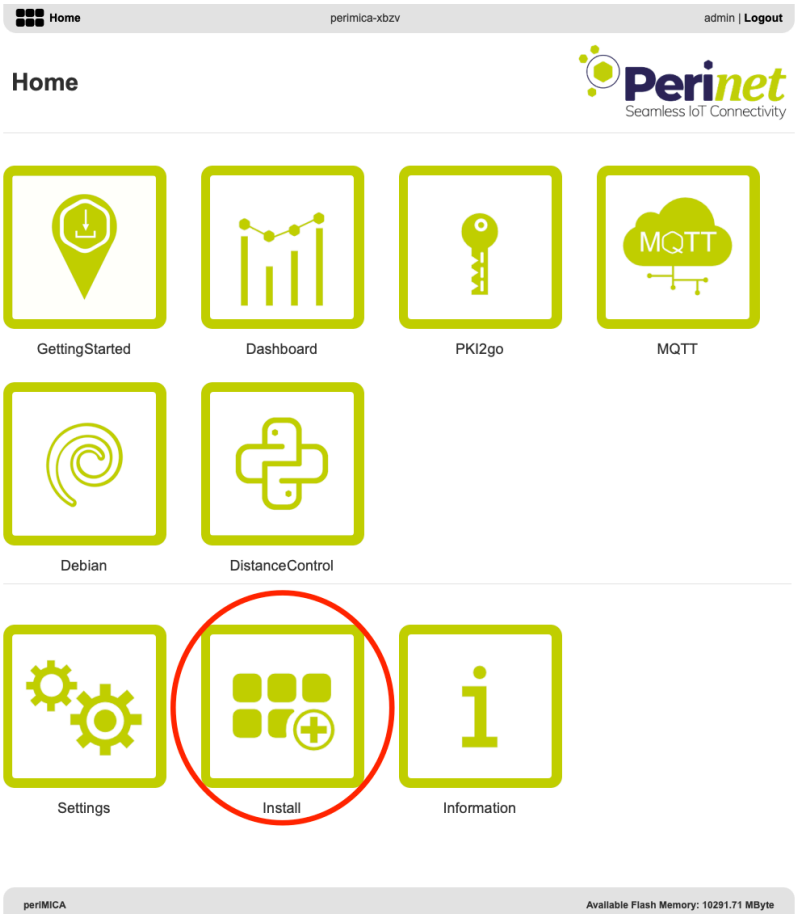

Figure 15: Go to periMICA *Install* webpage

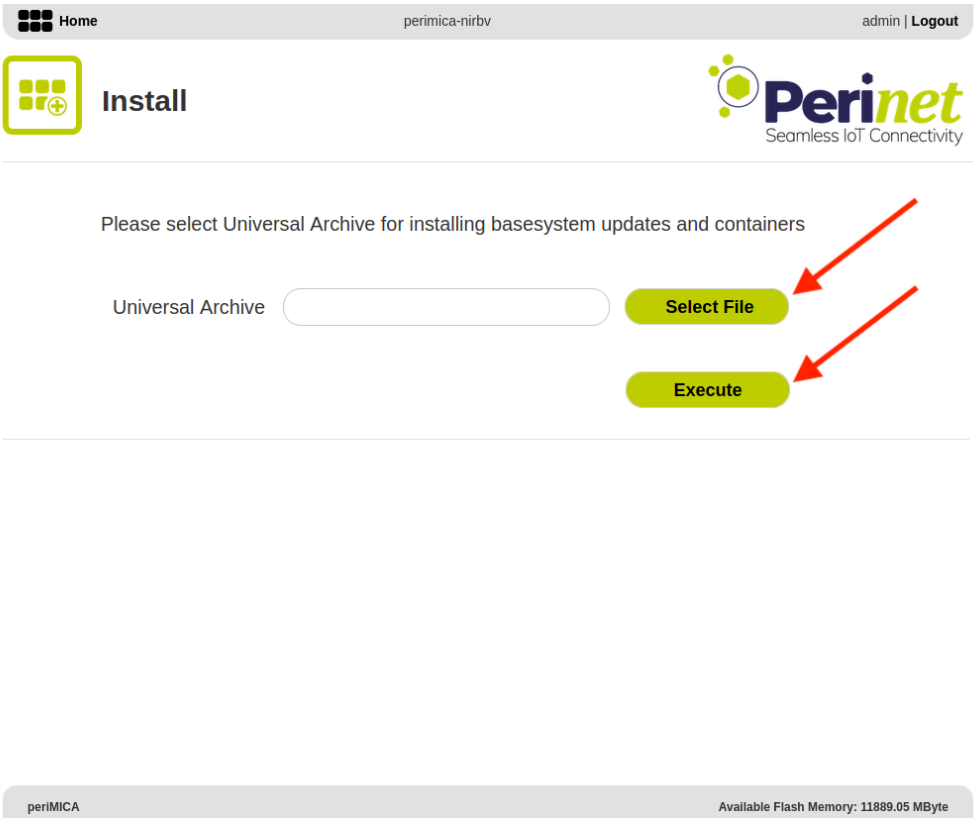

Figure 16: Select the file and execute

4. After installation, the containers are inactive by default (Figure 17). Start each new container by using the container context menu, by right-clicking on the corresponding container icon. Click on the *Start* button like in Figure 18:

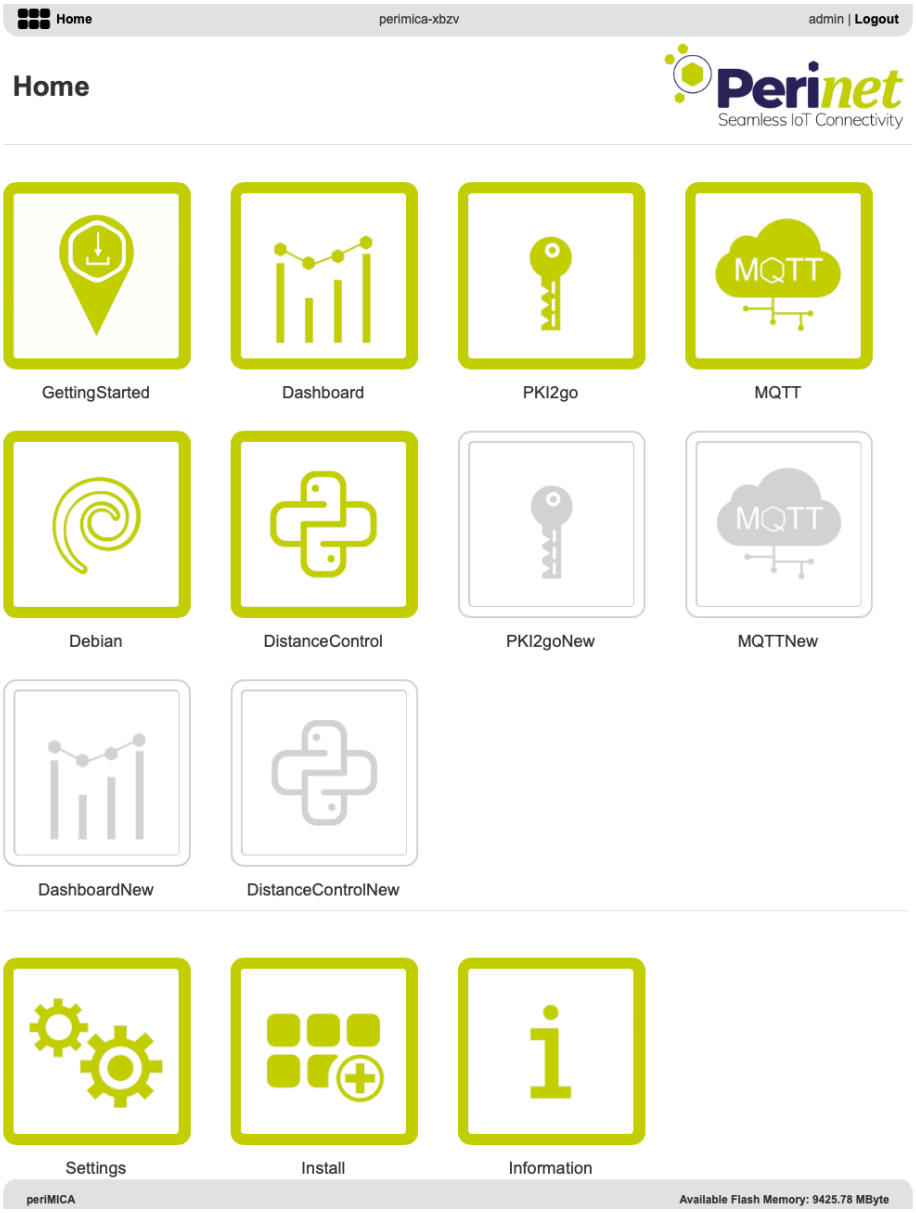

Figure 17: periMICA home page after installing the new containers

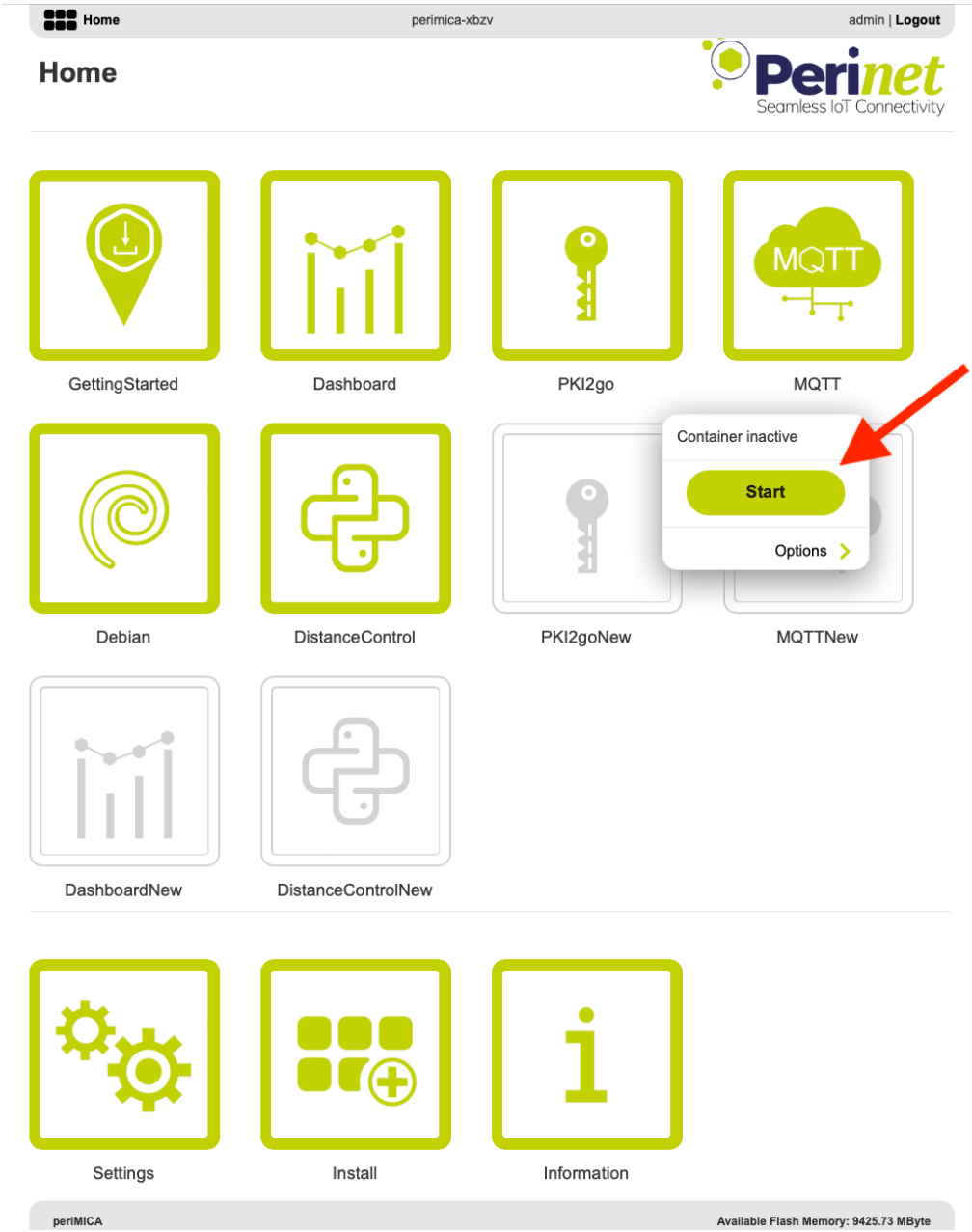

Figure 18: Start container

5. Open the newly installed PKI2go Web UI. By clicking on the *Application* field, a drop-down list will show all applications published on the same network. You can select the desired application name from the list or type a new name and then click on the *Create Root CA* button.

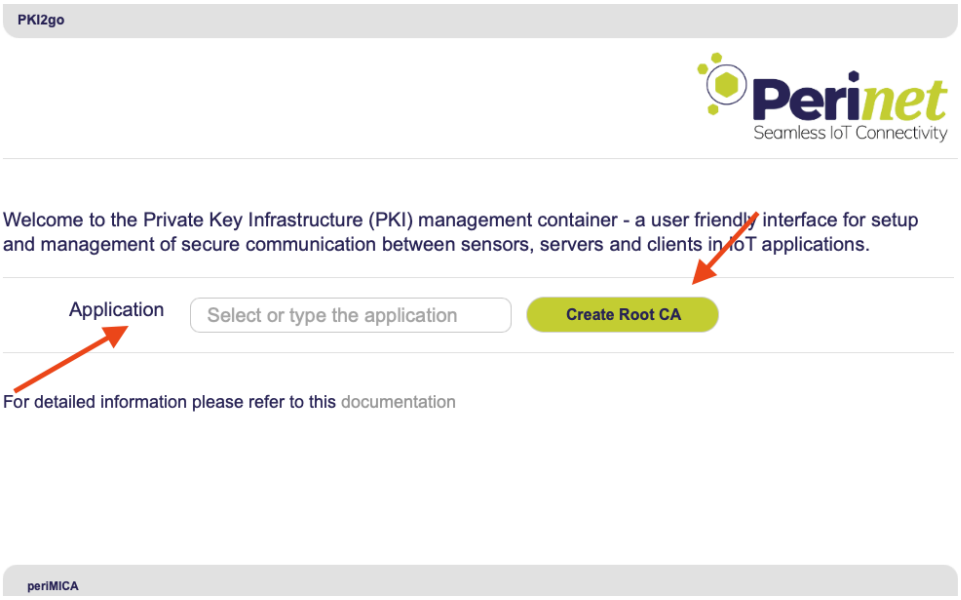

Figure 19: Create a new Root CA

6. PKI2go creates a new *Root CA* and provides a new *PKI2go.zip* file that contains the new certificates, which need to be installed into your browser/operating system. Please refer to the *Starter Kit Plus Security Certificates Installation Guide* [25] for more details.

   Note: The browser can cache the client certificates to speed up the connection. This is the reason why you can get *access denied* after installing a new certificate. Please restart your browser after installing new certificates, in order to successfully access again the periNODEs and the containers.

7. Refresh the PKI2go Web UI page. PKI2go is now ready to sign and distribute certificates to the hosts in the new application. The *Hosts discovered* list shows all hosts publishing services on the network. Select the new MQTT container installed in step 2 and add it in the application by clicking on the "> > >" button. Do the same for the Dashboard and DistanceControl containers and for both periNODEs. This will setup the application name, enable mTLS and install the corresponding certificates for each container and periNODE smart adapter. The following components should now appear in the *Hosts in this application* list (Figure 20):

Figure 20: Setup application in PKI2go

From now on, the periNODE smart adapters and containers will authenticate themselves with the host certificates provided by the PKI2go container. In the same time, with the newly received client certificates, the periNODE smart adapters, MQTT, Dashboard and DistanceControl containers are capable to communicate securely via MQTT protocol using mTLS.

8. Open the MQTT container Web UI and turn mTLS authentication *on*. Activate the new configuration by clicking on the *Push configuration* button (Figure 21):

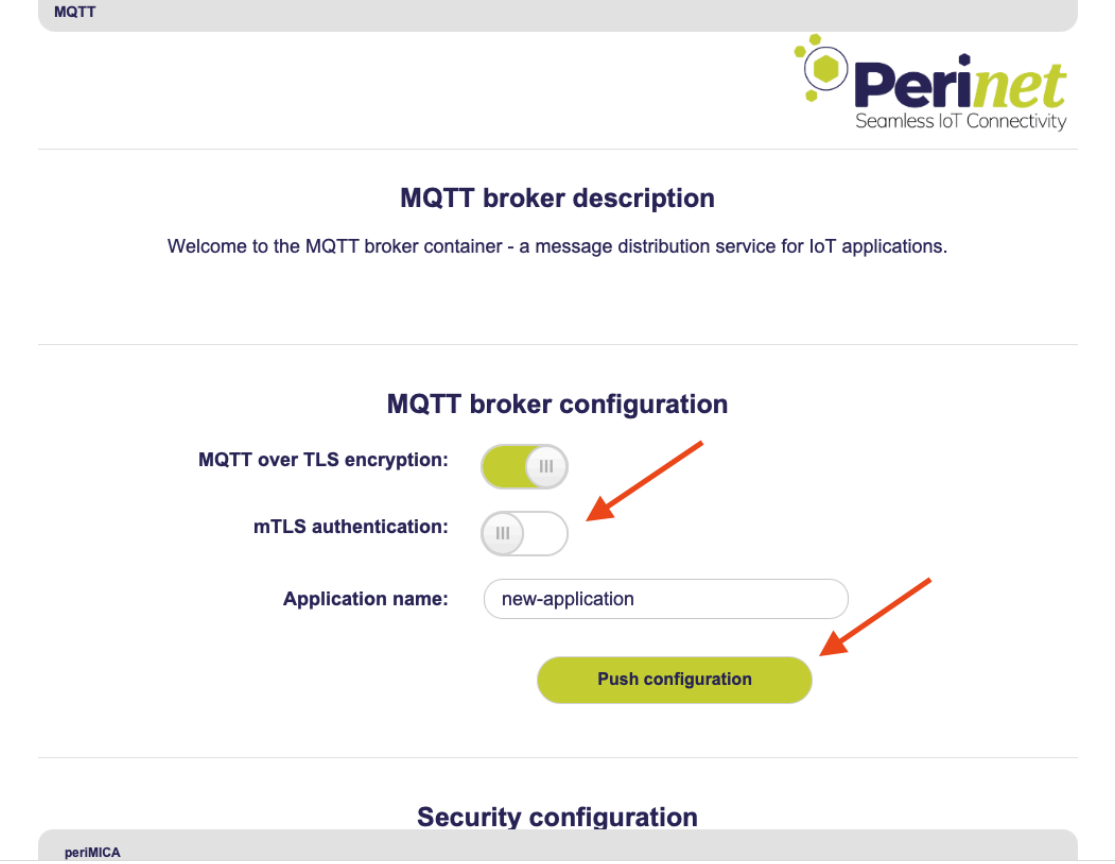

Figure 21: Enable mTLS on the MQTT broker

9. Now connect both periNODEs to the new MQTT broker. For each periNODE, open its Web UI configuration page and setup the MQTT broker. Activate the new configuration by clicking on the *Push configuration* button (Figure 22):

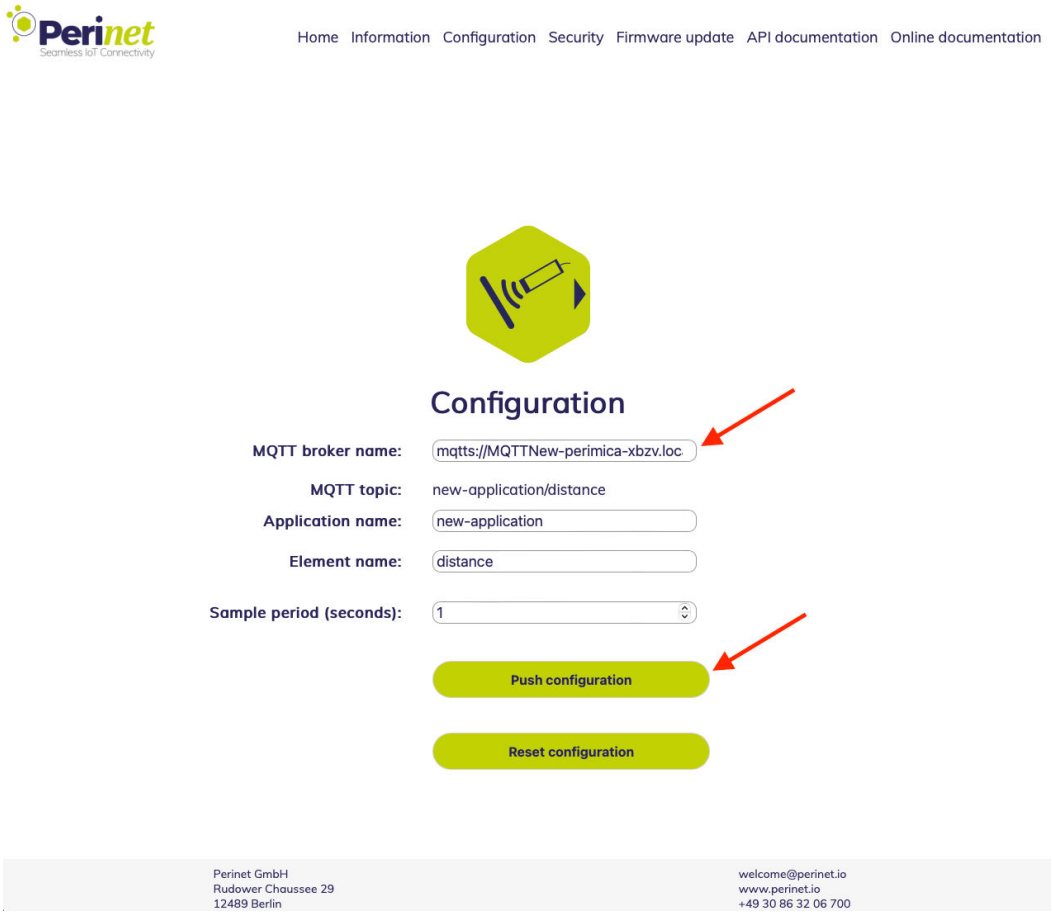

Figure 22: Setup MQTT broker on the periNODE

10. The new DistanceControl container needs to be connected to the MQTT broker. Open the DistanceControlNew container Web UI and configure the new MQTT broker as in Figure 23. Click on *Push configuration*:

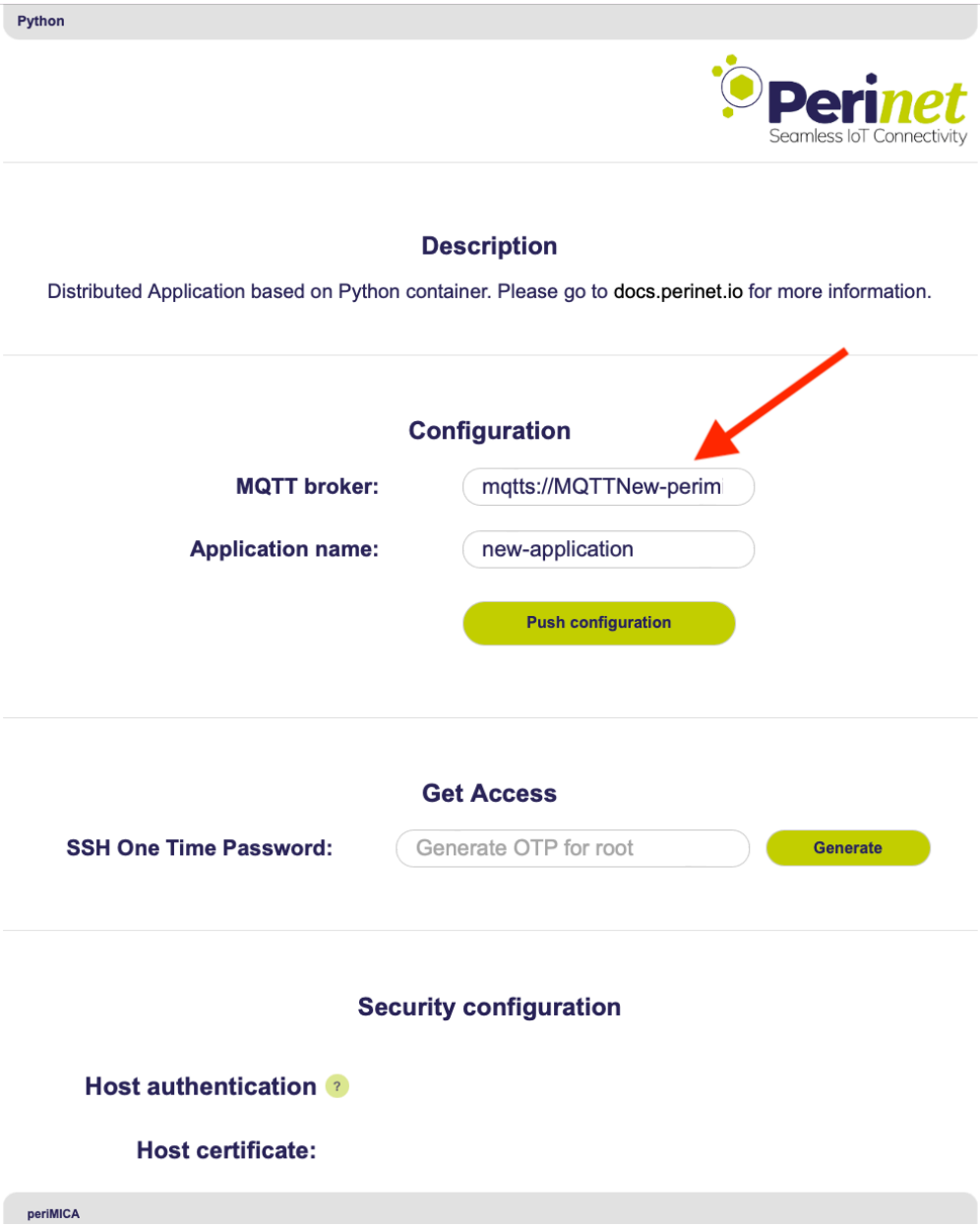

Figure 23: Setup MQTT broker in the DistanceControl

11. The new Dashboard container needs to be connected to the MQTT broker as well. Open the DashboardNew container Web UI and configure the new MQTT broker as in Figure 24. Click on *Push configuration*:

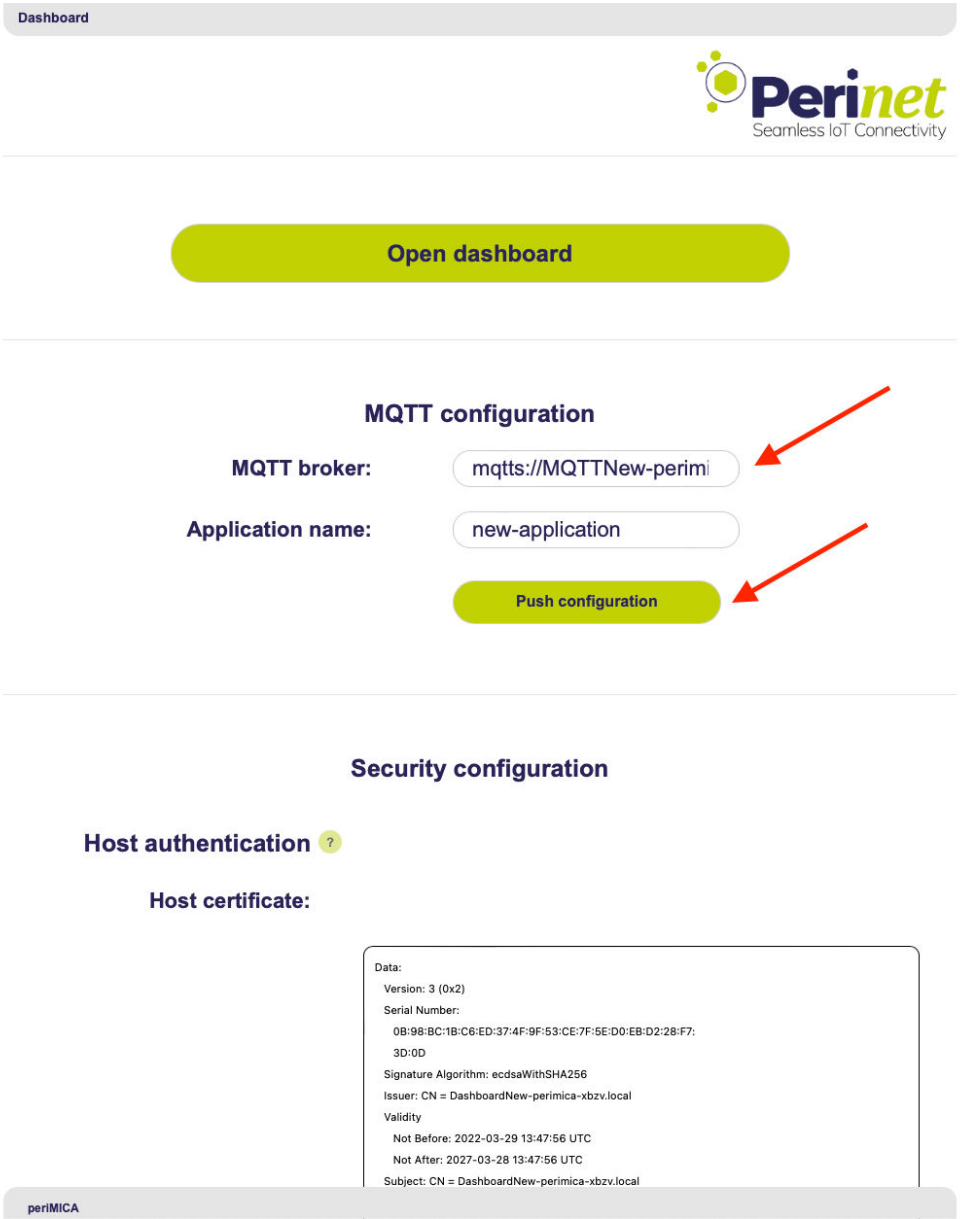

Figure 24: Setup MQTT broker in the Dashboard

12. Finally access the dashboard via the *Open dashboard* button to visualize the sensors data.

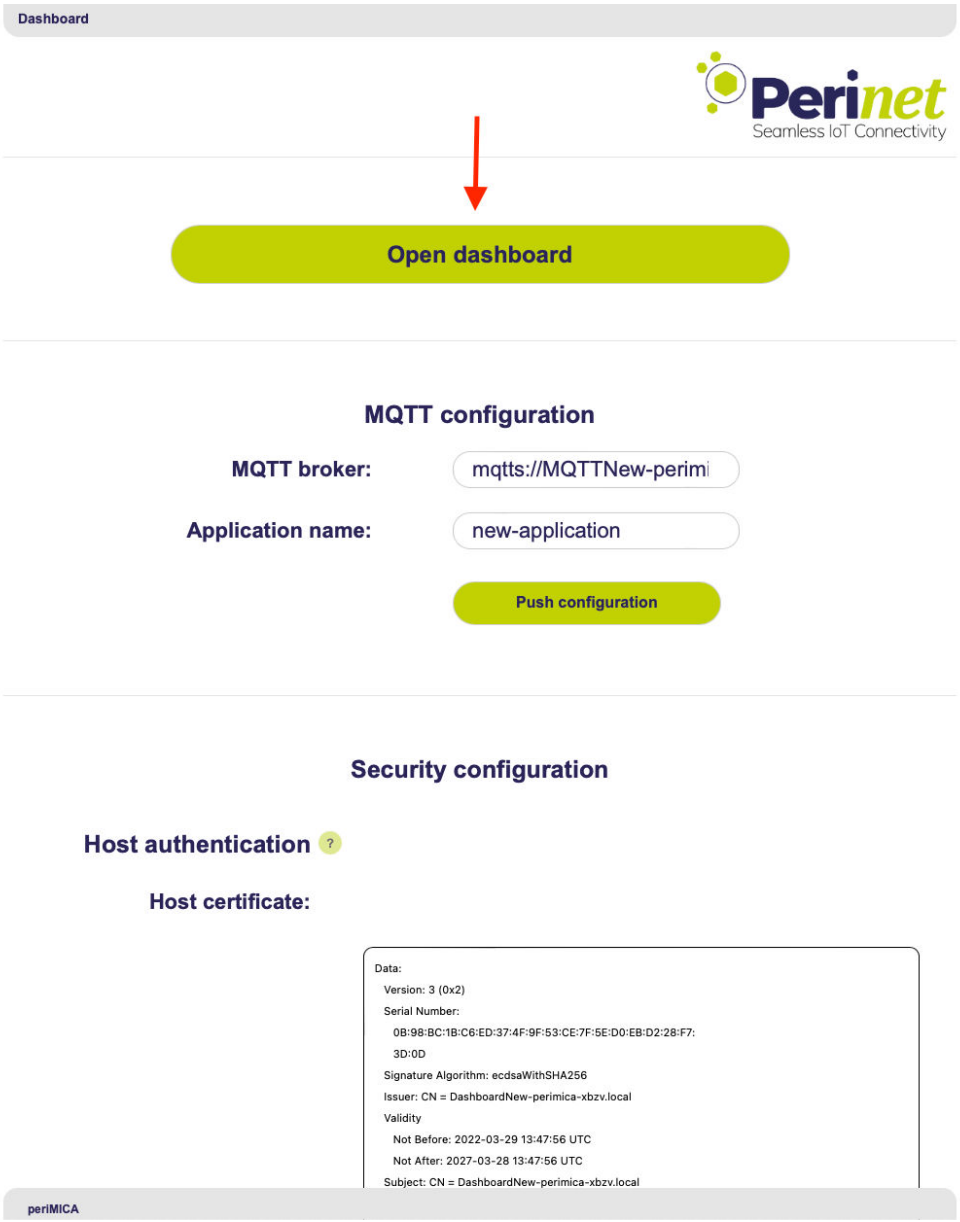

Figure 25: Open dashboard

You should be able to see the data coming from periNODEs again in your dashboard panels. The components are now protected by the new Application Root CA created by the PKI2go. Only users holding certificates issued by the same Application Root CA have access to periNODEs and containers. The configuration could be done separately as well, for example, to allow to some users only the access to the periNODE GPIO and to others only the access to the distance sensor.

The new DistanceControl container should be also prepared to work conform the defined rule implemented and the periNODE GPIO will switch the contactor accordingly to the rule (see Section 4 for more details).

For more information on different types of users and granting specific roles to the certificates, please read the *PKI2go Container User Guide* [20].

Figure 26: Dashboard panels

3.3 Rollback periNODEs to Starter Kit Plus Initial State

After a security reset, the periNODEs will have the Perinet GmbH factory certificates and not the ones signed by the *demo Application CA*, as in the initial state of the Starter Kit Plus. Therefore it is necessary to manually reconfigure the periNODEs:

1. Open the *initial* PKI2go container Web UI. This is the default PKI2go container of the Starter Kit Plus.

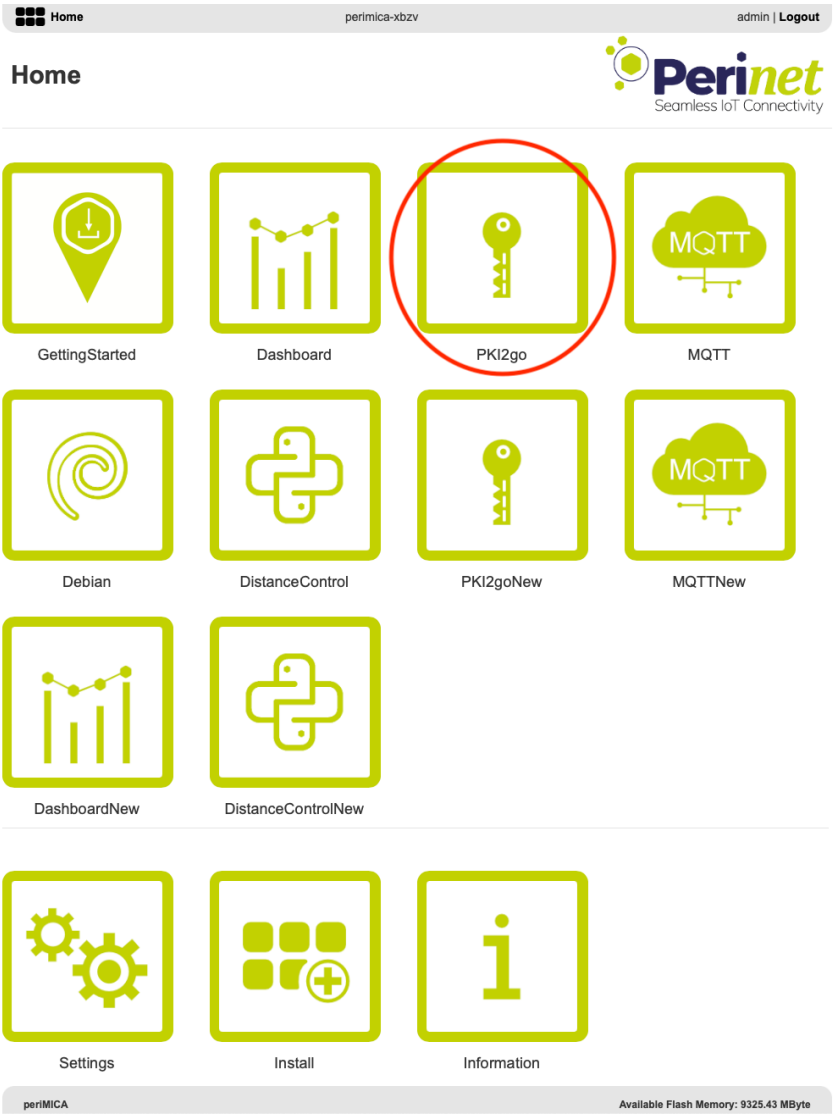

Figure 27: Click on the PKI2go icon

2. Download the demo Root CA certificate by clicking on the certificate icon:

Figure 28: Download demo root CA certificate

3. Download both periNODEs host certificates by double-clicking on the corresponding periNODE in the *Hosts in this application* list.

Figure 29: Download the certificate by double-clicking on the periNODEs

Note: The *periNODE-names* used here are example purposes only. Find the *periNODE-name* installed in your kit.

4. Download the *demo_admin_user* client certificates by double-clicking on the user named *demo_admin_user* in the *Users* list.

Figure 30: Download the user certificate by double-clicking on the *demo_admin_user*

5. Extract the downloaded *demo_admin_user.zip* archive, which contains two files: one *.p12* certificate file (only needed for the browser - not relevant for now) and one *.pem* certificate file (which needs to be uploaded as a client certificate - to be used in step 17).

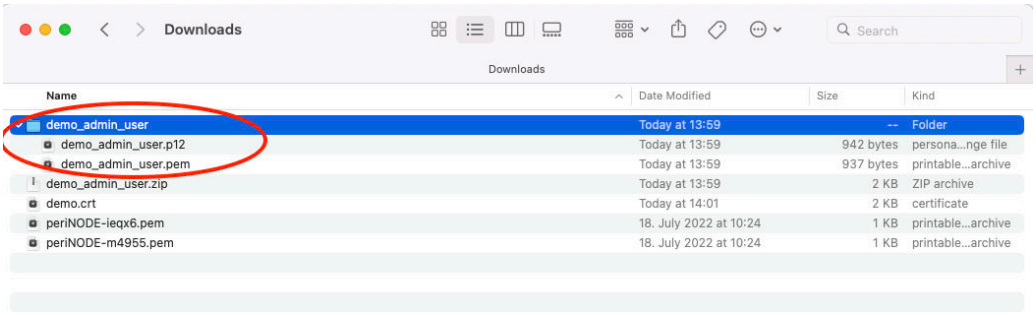

Figure 31: Extract the *demo_admin_user.zip*

6. In the periNODE Web UI, go to *Configuration*:

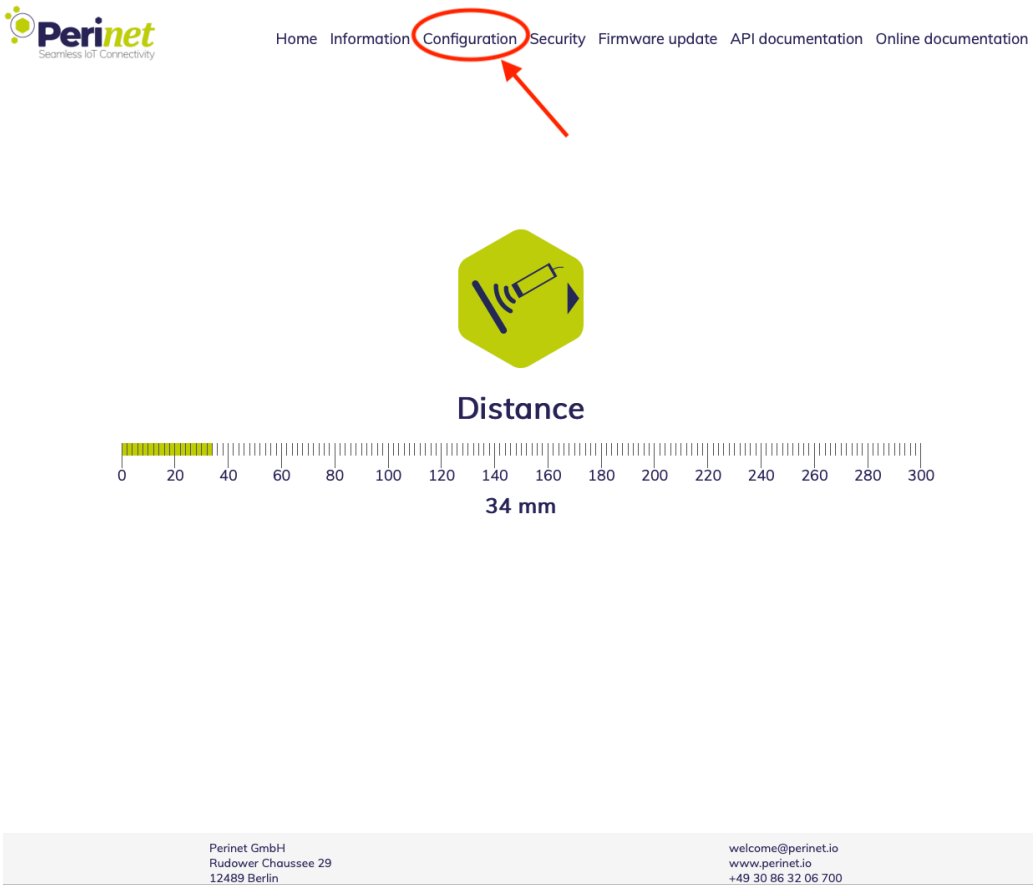

Figure 32: Go to *Configuration* on the periNODE Web UI

7. Setup the MQTT broker name with *mqtts://MQTT-perimica-name.local* and the application to *demo*.

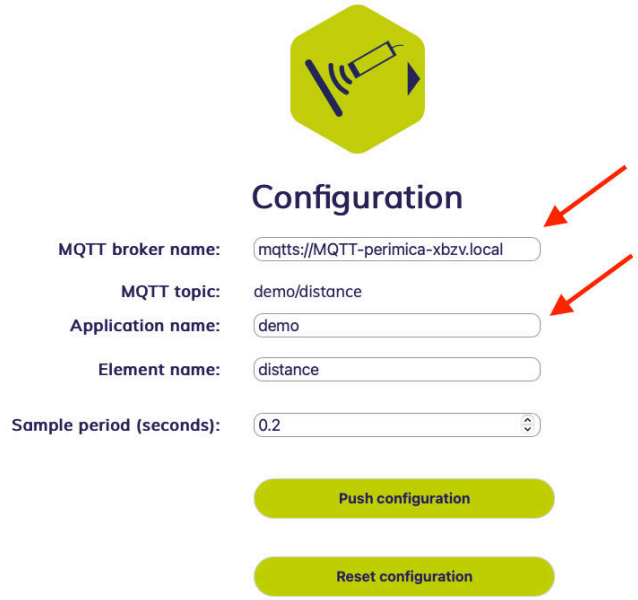

Figure 33: Select the MQTT broker name and the application name *demo*

8. Click on *Push configuration*:

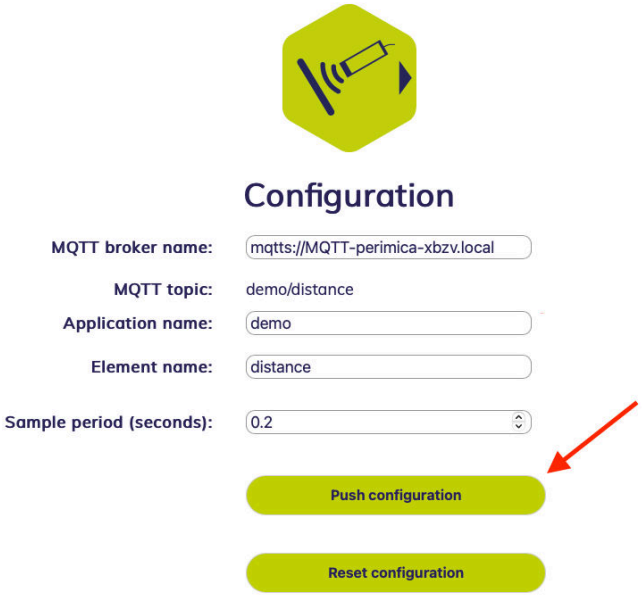

Figure 34: Click on *Push configuration* and confirm the message

9. Navigate to the Security page by clicking on *Security*:

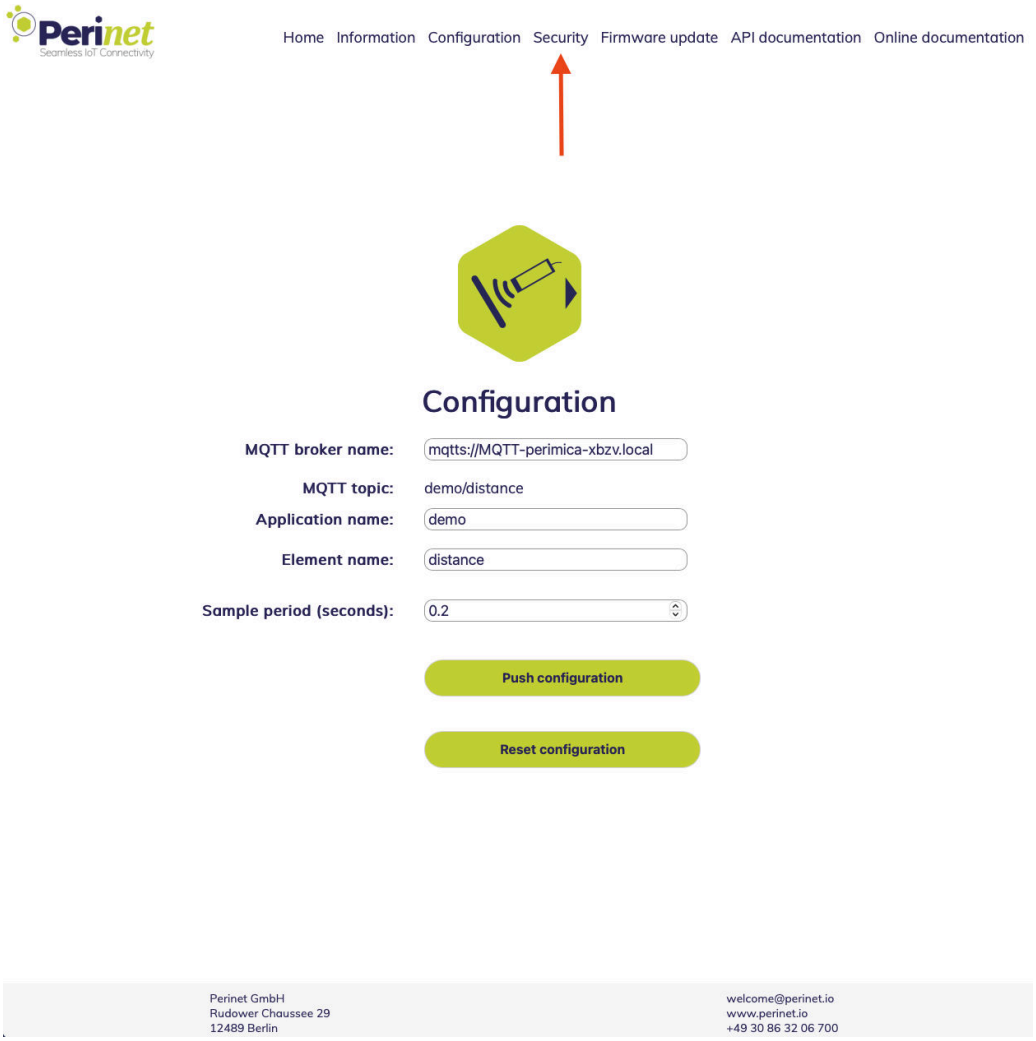

Figure 35: Go to *Security* on the periNODE Web UI

10. Configure the host certificate by clicking on *Upload host certificate*:

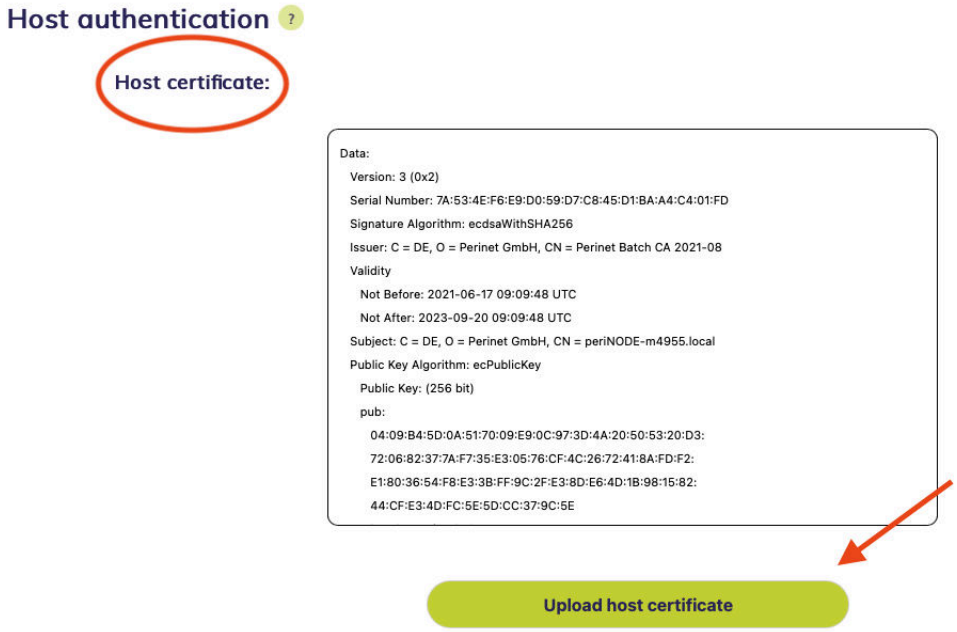

Figure 36: Upload host certificate

11. Locate the downloaded *periNODE-name.pem* file certificate:

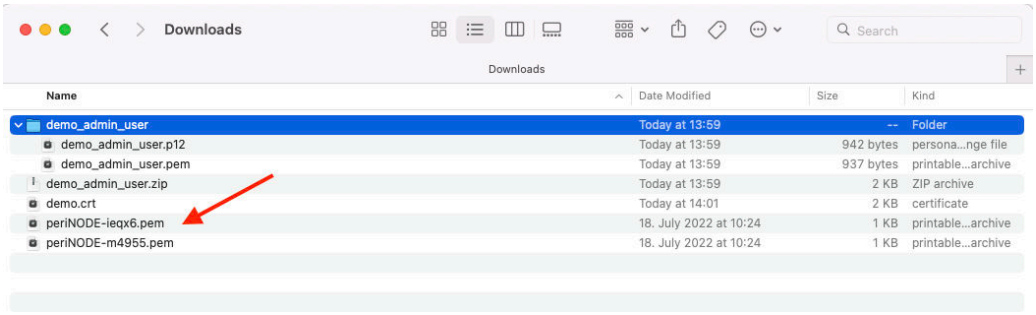

Figure 37: Upload the *periNODE-name.pem* file

12. Confirm the upcoming popup message:

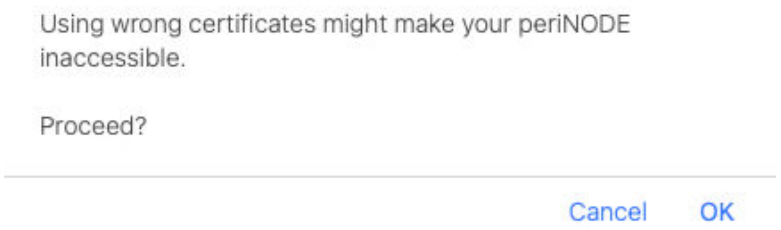

Figure 38: Confirm the certificate upload

13. Upload the root certificate:

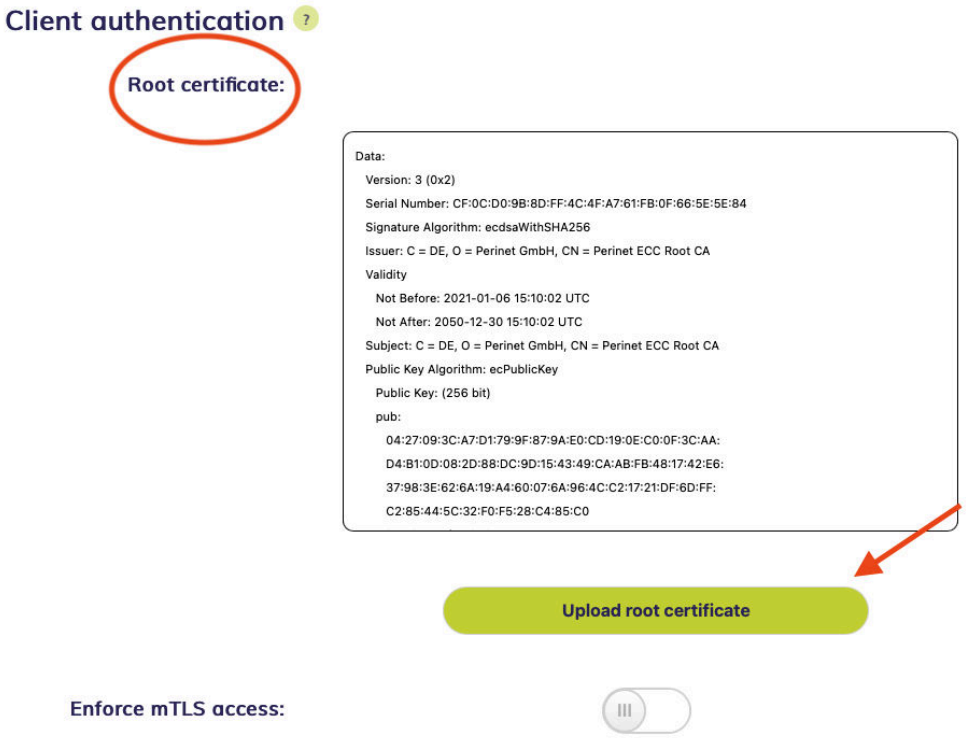

Figure 39: Upload root certificate

14. Locate the downloaded *demo.crt* certificate:

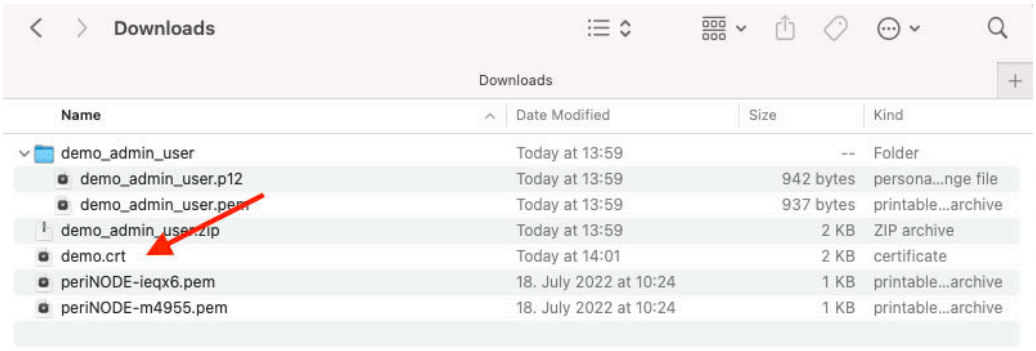

Figure 40: Upload the *demo.crt* file

15. Confirm the upcoming popup message:

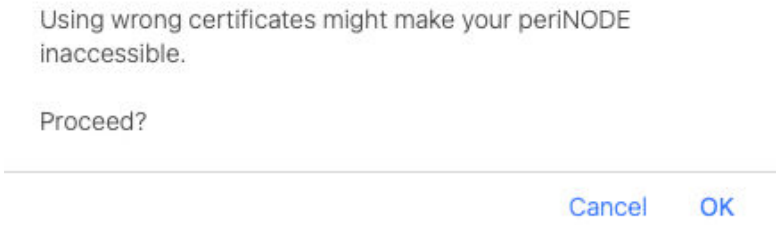

Figure 41: Confirm the certificate upload

16. Upload the client certificate:

Figure 42: Upload client certificate

17. Locate the downloaded *demo_admin_user.pem* certificate:

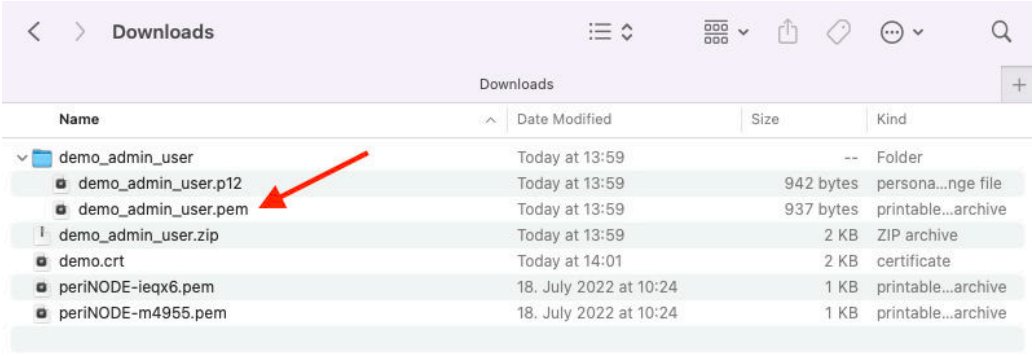

Figure 43: Upload the *demo_admin_user.pem* file

18. Confirm the upcoming popup message:

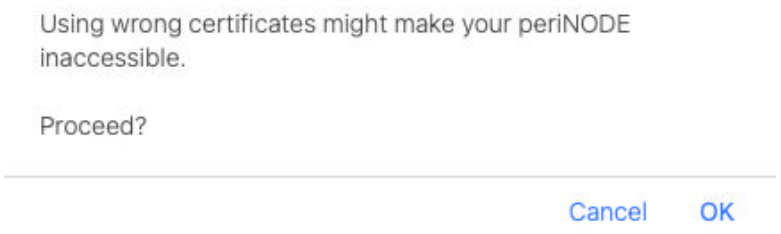

Figure 44: Confirm the certificate upload

19. *Enforce mTLS access* by turning the switch on:

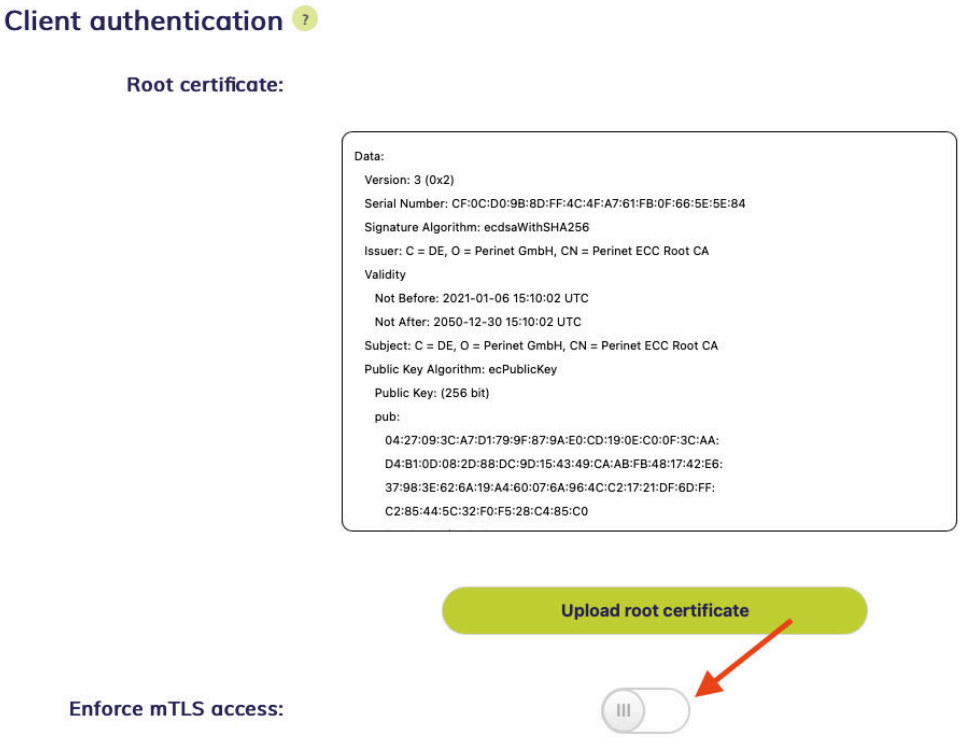

Figure 45: Enforce mTLS access

20. Confirm the upcoming popup message:

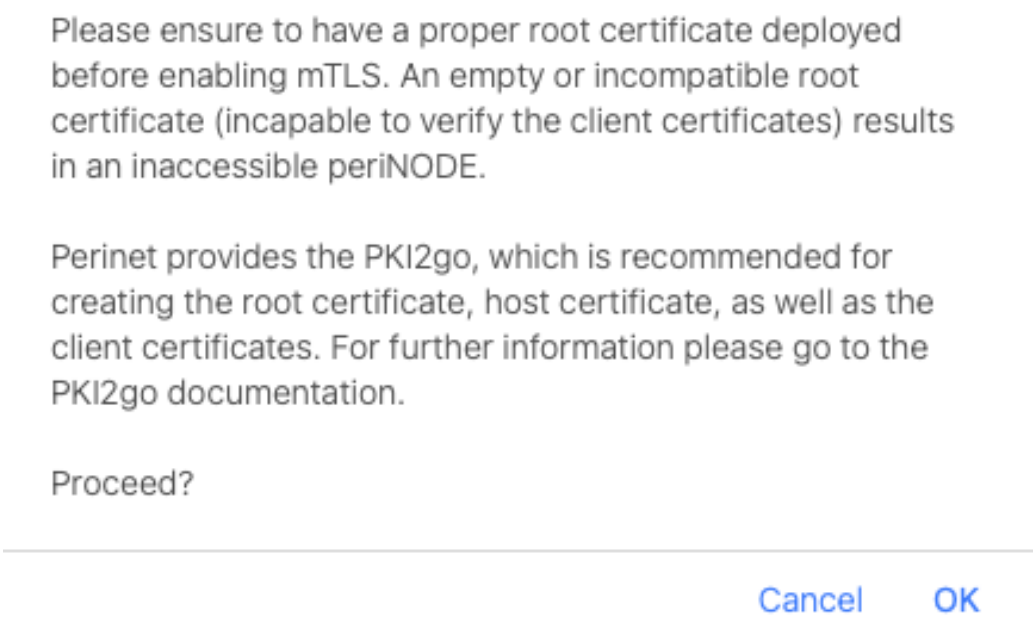

Figure 46: Confirm *Enforce mTLS*

21. Repeat the steps 6-20 for the remaining periNODE.

4 Distributed DistanceControl Container

The DistanceControl container is an example application to demonstrate the usage of IIoT protocols like MQTT and mDNS to automate tasks. Data is read from a sensor connected to the periNODE distance adapter and processed. The process result commands an actuator connected to another adapter: periNODE GPIO.

In the Figure 47 below, the DistanceControl container WebUI is shown.

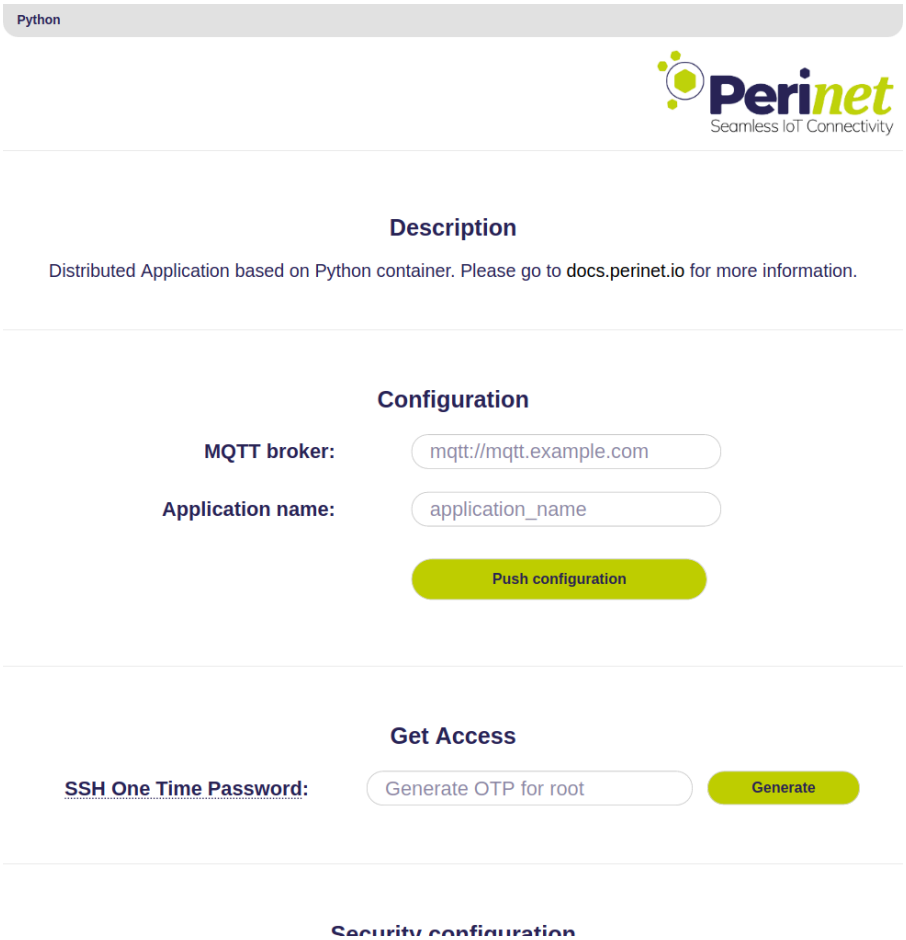

Figure 47: DistanceControl Starter Kit Plus

4.1 Basic Functionality

The *DistanceControl* container is a Debian container running a Python script which:

- receives distance values
- implements a decision level
- controls an actuator

To receive the distance values it uses an MQTT client to subscribe to the topic *application_name/distance* on the broker that has been configured via the textfield *MQTT broker:* on the

WebUI. When the values are below the decision level the actuator gets switched on, otherwise it's going to be switched off. To control the actuator the MQTT client publishes the command to the topic *application_name/contactor_switch*.

The rule implemented in the container basically is:

- Distance value < 100 mm → switch on the contactor
- Distance value >= 100 mm → switch off the contactor

The DistanceControl container is also part of the application called *demo*, which can be configured on the "Configuration" section on the WebUI. It can also be changed to be part of another application when necessary, just remember to connect the periNODE distance and periNODE GPIO to the same *application* to make it work.

The user can enable or disable the container in order to have the automation available. To do it, right click on the container and click on *Stop* as shown in Figure 48.

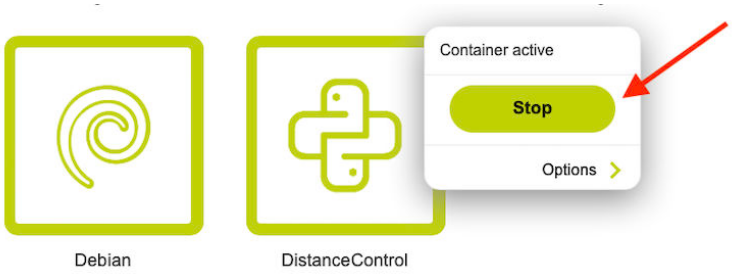

Figure 48: DistanceControl Starter Kit Plus - Stop Container

After stopping the container, it will be in inactive state, and you can see the container like in the Figure 49 below:

Figure 49: DistanceControl Starter Kit Plus - Inactive Container

Finally, when the container is not running, the switch will not change the state when changing the distance.

4.2 SSH Access

The DistanceControl container is based on Debian Bullseye distribution and it provides an SSH server which offers a secure access to the container using One Time Password. To access the container go to the DistanceControl WebUI and click on "Generate" like you can see in the Figure 50.

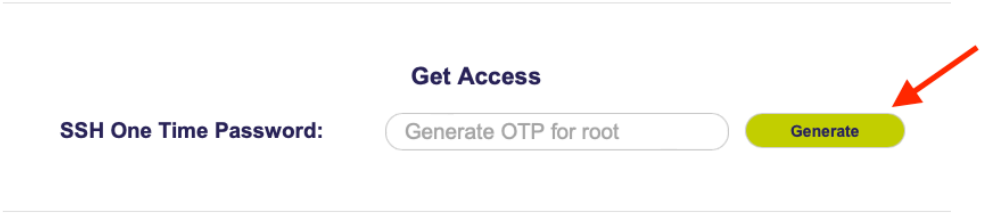

Figure 50: DistanceControl Get Access

Using an SSH client installed in your machine, provide the password generated and you will be logged in. As the name suggested the One time password just can be use once! You can see an example of the access messages you will receive during the access:

```
$ ssh root@DistanceControl-perimica-xbzv.local
root@DistanceControl-perimica-xbzv.local's password:
Linux DistanceControl-perimica-xbzv.corp.perinet.io 5.4.133 #1 SMP Mon Jul 26 11:19:13 UTC
↪ 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar 30 10:16:26 2022 from fe80::897:bbf4:890b:b178%eth0
root@DistanceControl-perimica-xbzv:~#
```

Listing 1: Access DistanceControl container using SSH

4.3 Security Configuration

The security configuration is part of periNODEs and periMICA containers and it is also present on the DistanceControl. It can be configured automatically using a PKI2go container or manually. The process to configure the security is explained on the Section 3.

4.4 Example of Automation

The `trigger_gpio.py` script is setup on Debian Service and will run as soon as the container is started. The script connects to the configured MQTT Broker and provides the certificates needed to connect. Once connected, it subscribes to the topic *demo/distance* in order to take the distance values provided by the periNODE distance. Each message received is evaluated and can trigger the contactor sending the message to the OUTPUT that has configured in the *demo/application*, so that the topic used is *demo/contactor_switch*. In the Figure 51 is shown the procedure.

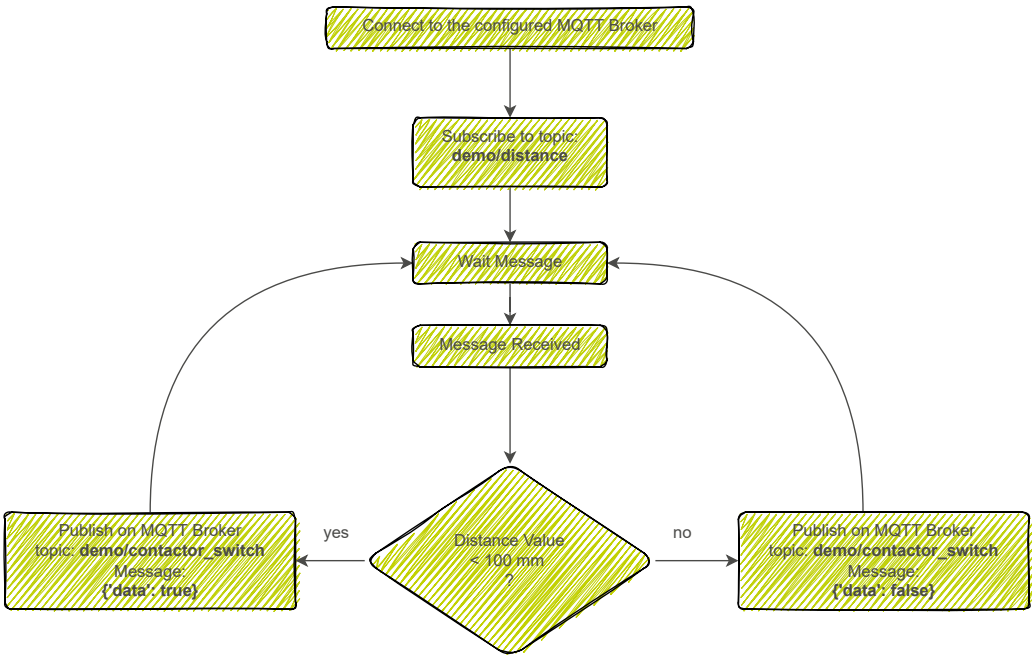

Figure 51: DistanceControl Fluxogram

The script is running as a Debian service. It is located in `/usr/bin/trigger_gpio.py`. You can access the container using SSH and configure the script if necessary using the Debian tools installed. For further information about Debian container, please refer to Debian User Guide [10].

The complete python script as a reference is presented in the Appendix A.

5 Troubleshooting

- In case periNODEs are misconfigured either by uploading wrong host or root certificates or when the periNODE configuration is not functional anymore by inserting wrong values, each periNODE can be put into reset mode. In the reset mode, the periNODE smart adapter falls back to factory default configuration, including security default configuration.

  To perform this reset, connect the periNODE with the reset cable (included in the Starter Kit), either from the periSTART or from the periSWITCH, for at least 20 seconds.

  At the end of the 20 seconds, replace the reset cable by the periLINE cable again and access again the resetted periNODE smart adapter.

  After this operation, the periNODE smart adapter needs to be reconfigured. In case of the Starter Kit Plus, this also means that it has to be added again to the default **demo** application, as explained in Section 3.3.

- Browsers can cache SSL certificates in order to speed up the access. But caching can cause issues, because a protected server will refuse the connection if the browser sent the incorrect certificate. In that case, refreshing the web page might help. If the problem still occurs, make sure to have the correct client certificates imported and restart the browser.

- When the periMICA is not able to reach any NTP (Network Time Protocol) server, its Time & Date are not automatically synchronized and it can generate security errors when trying to access the containers installed. To fix this problem you can configure the Time & Date manually in *periMICA Homepage* → *Settings* → *Time & Date*.

6 Further Documentation

6.1 Starter Kits

Document Name	Description
Starter Kit Plus Product Summary [23]	A product features summary documentation for the *Starter Kit Plus*.
Starter Kit Plus Quick Start Guide [24]	A setup guide for the *Starter Kit Plus* product.
Starter Kit Plus User Guide [26]	A detailed description of the setup *Starter Kit Plus*.
Starter Kit Plus Security Certificates Installation Guide [25]	A small guide on how to install security prerequisites for accessing the various entities of the *Starter Kit Plus* setup.
Starter Kit Product Summary [27]	A product features summary documentation for the available *Starter Kits*.
Starter Kit Quick Start Guide [28]	A setup guide for the *Starter Kit* products.

6.2 periMICA

Document Name	Description
periMICA Product Summary [12]	A product features summary documentation for the *periMICA*.
periMICA Quick Start Guide [13]	A setup guide for the product *periMICA*. The starting point when you are new to the product.
periMICA User Guide [14]	A detailed description and reference documentation of the product *periMICA*.
PKI2go Container User Guide [20]	A detailed description of the public key infrastructure (PKI) periMICA container 'PKI2GO'.
Security Certificates Installation Guide [21]	A guide on how to install prerequisites certificates on various platforms (MAC, Windows and linux).
periMICA Debian Container User Guide [10]	A detailed description of the base container of a periMICA lxc-based container. It's the starting point for container development.
periMICA Product ReBranding Application Note [11]	A detailed description for development to rebrand the product periMICA to a dedicated customer corporate design.

6.3 Smart Components

Document Name	Description
PeriLINE Product Summary [9]	A product features summary documentation for the product *periLINE*.
periNODE 0-10V Product Summary [15]	A product features summary documentation for the product *periNODE 0-10V*.
periNODE Pt100 Product Summary [17]	A product features summary documentation for the product *periNODE Pt100*.
periNODE GPIO Product Summary [16]	A product features summary documentation for the product *periNODE GPIO*.
periSWITCH 3-port Product Summary [19]	A product features summary documentation for the product *periSWITCH 3-port*.
periSTART Standard Product Summary [18]	A product features summary documentation for the product *periSTART standard*.
Smart Components Datasheet [22]	A detailed reference documentation of the Smart Components products (*periLINE*, *periNODE 0-10V*, *periNODE Pt100*, *periNODE GPIO*, *periSWITCH 3-port*, *periSTART standard*).

6.4 periCORE

Document Name	Description
periCORE Product Summary [8]	A product features summary documentation for the product *periCORE*.
periCORE Datasheet [4]	A detailed reference documentation of the product *periCORE*.
periCORE Development Kit Product Summary [5]	A product features summary documentation for the product *periCORE Development Kit*.
periCORE Development Kit Setup Application Note [6]	A setup guide for the product *periCORE Development Kit*. The starting point when you are new to the product which describes how to quickly set up a development environment for firmware development for a periCORE based product.
periCORE Development Kit User Guide [7]	A guide and reference documentation for the product *periCORE Development Kit*.

7 Contact & Support

For customer support, please call us at **+49 30 863 206 701** or send an e-mail to *support@perinet.io*.

For complete contact information visit us at `www.perinet.io`

A Distance Control Script Code Example

Listing 1: Example Python Script

```
import paho.mqtt.client as mqtt
import time
import json
import sys
import subprocess
import ssl
import logging


# Configuration files
SECURITY_CONFIGURATION = "/META/security_configuration.json"
SERVICE_CONFIGURATION = "/etc/avahi/services/https.service"
USER_CONFIGURATION = "/META/user_configuration.json"

# Default webserver certificates path
WEBSERVER_CERT_PATH = "/etc/webserver"
ROOT_CERT = WEBSERVER_CERT_PATH + "/root-ca.pem"
CLIENT_CERT = WEBSERVER_CERT_PATH + "/client.pem"
CLIENT_KEY = WEBSERVER_CERT_PATH + "/client.key"

# Output port element names
SINK_ELEMENT_NAME = "contactor_switch"
# Distance element name
DISTANCE_ELEMENT_NAME = "distance"

# Path to the log file
LOG_FILE = "/var/log/trigger_gpio.log"
# Basic Logging Configuration
logging.basicConfig(filename=LOG_FILE, filemode='w', level=logging.CRITICAL)

# MQTT ports
MQTT_PORT = 1883
SECURE_MQTT_PORT = 8883

SHELL_FAILURE_MESSAGE = "Shell call failed with error."

class trigger_gpio():
    """
        Trigger_gpio is prepared to run as a service in a python container
        which is based on a bullseye debian distribution.
        It connects to the MQTT Broker that is configured in the
   ↪ application,
        subscribe on the topic application name/distance and according to
```

```
    ↪ the
        following rule switch the contactor connected to the periNODE GPIO.
            - distance < 100 mm switch on the contactor,
            - distance <= 100 mm switch off the contactor.
        The application is implemented to use security TLS certificates to
        authenticate the broker and the client provides the granted
    ↪ certificate
        to read and send messages on the broker.
     """

    def __init__(self) -> None:
        """
            Setup the initial needed parametes.
            * Application Name. It can be configured in the container Web
    ↪ UI manuallly or automatically
            using the PKI2go container.
            * Check if the container prepared to run in a secure way. This
    ↪ means it is configured
            as secure and has the certificates needed to follow with the
    ↪ secure connections.
        """
        logging.info('init trigger_gpio()')
        self.application_name = self.get_application_from_service()
        self.secure = self.is_security_enabled()
        #sleep before call go_discover again and avoid flooding the network
        time.sleep(0.5)
        self.broker = self.get_broker_application()
        self.element_name_gpio = SINK_ELEMENT_NAME
        #sleep before call go_discover again and avoid flooding the network
        time.sleep(0.5)
        self.element_name_distance = DISTANCE_ELEMENT_NAME
        self.status = False

    def get_application_from_service(self) -> str:
        """
            Read the application_name configured in the container service.
            Once configured automatically through PKI2go or even manually
            in the container Web UI, the application_name is part of the
            avahi service txt-record field.
            It returns the string 'application_name'.
        """
        try:
            # read the configuration file to take the application_name
            service_configuration_file = open(SERVICE_CONFIGURATION, "r")
            content = service_configuration_file.read()
            service_configuration_file.close()
```

```
            start = content.find("application_name=")
            len_application = len("application_name=")

            # if application_name was found, find the end
            if start > 0:
                end = content.find('</txt-record>',\
                    start + len_application)
                logging.debug('get_application_from_service: returns ' +
↪ content[start + len_application : end])
                return content[start + len_application : end]
            else:
                logging.debug('get_application_from_service: returns empty')
                return ""
        except:
            logging.error("Oops! ", sys.exc_info()[0], " occurred.")
            raise

    def is_security_enabled(self) -> bool:
        """
            Read the configuration file to check if the mTLS is enabled.
            The param enable_user_role is part of the security API and
            keep in a json file inside the container.
            It returns a boolean true if mTLS is enabled or false if not.
        """
        try:
            security_configuration_file = open(SECURITY_CONFIGURATION, "r")
            security_configuration_content =
↪ security_configuration_file.read()
            security_configuration_file.close()
            info = json.loads(security_configuration_content)
            logging.debug('is_security_enabled: ' +
↪ str(bool(info["enable_user_role"])))
            return bool(info["enable_user_role"])
        except:
            logging.error("Oops! ", sys.exc_info()[0], " occurred.")
            raise

    def publish_mqtt_message(self, message: bool):
        """
            It publishes message on the broker.
            Prepare the message to be published in the mqtt client.
            In order to send a message to be read by periNODE GPIO and
            switch the port the topic should have compound like:
            - "application name/element name"
            And the json with the boolean received would be passed like:
```

```
            {"data": true}
        """
        try:
            if message:
                logging.debug('publish_mqtt_message: ' +
↪ self.application_name + '/' + self.element_name_gpio + '{"data": true}')
                self.client.publish(self.application_name + '/' +
↪ self.element_name_gpio, '{"data": true}')
                self.status = True
            else:
                logging.debug('publish_mqtt_message: ' +
↪ self.application_name + '/' + self.element_name_gpio + '{"data":
↪ false}')
                self.client.publish(self.application_name + '/' +
↪ self.element_name_gpio, '{"data": false}')
                self.status = False
        except:
            logging.error("Oops! ", sys.exc_info()[0], " occurred.")
            raise

    def on_message(self, client, userdata, message):
        """
            Callback function when received message from broker.
            Evaluate the distance received on the broker that contains
            the sensor data.
            When distance < 100 send message to switch on the contactor
            if distance >= 100 send message to switch off.
        """
        try:
            logging.debug('on_message: received')
            # parse the json message received
            data = json.loads(message.payload.decode("utf-8"))

            logging.debug('value -> ' + str(data['data'][0]['distance']))

            # evaluate the data to switch the contactor accordinly
            if (self.status == False and data['data'][0]['distance'] < 100):
                logging.debug('on_message: send true')
                self.publish_mqtt_message(True)
            elif (self.status == True and data['data'][0]['distance'] >= 100):
                logging.debug('on_message: send false')
                self.publish_mqtt_message(False)
        except:
            logging.error("Error publishing MQTT!")
            raise
```

```
    def __shell_call(self, cmd: str,  input: str="", raise_err: bool=False) -
↪ > str:
        """
            Auxilliary method to execute shell commands.
            Returns the string output.
        """
        ret = subprocess.run(cmd, shell=True, stdout=subprocess.PIPE,
↪ input=input,\
            encoding='utf-8',stderr=subprocess.PIPE)
        err =  ret.stderr
        if raise_err and err != "":
            logging.error(str(err))
        return ret.stdout

    def get_broker_application(self):
        """
            Get the mqtt_broker_name from user configuration file
        """
        # Find the mqtt broker configured in the application.
        # Fetch it from user configuration json file
        if (len(self.application_name) > 0):
            user_configuration_file = open(USER_CONFIGURATION, "r")
            user_configuration_content = user_configuration_file.read()
            user_configuration_file.close()
            info = json.loads(user_configuration_content)
            logging.debug('info read from file: ' + str(info))

            # remove tags (mqtt:// or mqtts:///) if needed
            broker = info["mqtt_broker_name"]
            if (broker.find("//") > 0):
                broker = broker[broker.find("//")+2:]
            logging.debug('get_broker_application() returns: ' + broker)
            return broker
        else:
            return ""

    def on_disconnect(self, client, userdata, rc):
        logging.error("Unexpected MQTT disconnection! Error: ",
↪ sys.exc_info()[0], " occurred.")
        raise

    def run(self):
        """
            Prepare the mqtt client.
            - Setup the callback function
            - Connect to the broker secure or insecure
```

```
            - Subscribe to the specific topic: application name/element name
            - Stay connected to receive the incoming messages
        """
        try:
            # when the broker is already found
            if (self.broker != ''):

                #create new instance
                self.client = mqtt.Client("client1")
                #attach function to callback
                self.client.on_message = self.on_message
                self.client.on_disconnect = self.on_disconnect

                # prepare and connect
                if (self.is_security_enabled()):
                    self.client.tls_set(ca_certs=ROOT_CERT,
↪ certfile=CLIENT_CERT, \
                        keyfile=CLIENT_KEY, cert_reqs=ssl.CERT_NONE,
↪ tls_version=ssl.PROTOCOL_TLSv1_2)
                    # Attention: It is needed to specify the port when
↪ secure!
                    # connect to secure broker
                    self.client.connect(self.broker, SECURE_MQTT_PORT)
                    logging.debug('Subscribed to secure broker')
                else:
                    self.client.connect(self.broker) #connect to broker
                    logging.debug('Subscribed to no secure broker')

                # subscribe on the broker on topic
↪ application_name/element_name
                logging.debug('Subscribe topic:'+  self.application_name + '
↪ /' + DISTANCE_ELEMENT_NAME)
                self.client.subscribe(self.application_name + "/" +
↪ DISTANCE_ELEMENT_NAME)

                # stay connected
                logging.debug('Waiting messages..')
                self.client.loop_forever()
                time.sleep(40) # wait
                raise
            else:
                logging.error("No broker found!")
                raise

        except:
            logging.error("Oops! ", sys.exc_info()[0], " occurred.")
```

```
            raise

# Star running
trigger = trigger_gpio()
trigger.run()
```

B List of Figures

C List of Listings

D Glossary

API Application Programming Interface. 6

CA Certification Authority, a trusted entity which is represented by a certificate that is used to verify the signature on a certificate issued by that authority (trust anchor). 10, 12, 30

DNS-SD DNS Service Discovery [1] is a way of using standard DNS programming interfaces, servers and packet formats to browse the network for services. 7

HTTP Hypertext Transfer Protocol is an application-layer protocol for transmitting hypermedia documents, such as HTML. 6, 7

IIoT Industrial Internet of Things. 4, 6, 9, 44

IPv4 Internet Protocol Version 4, a communication protocol. 7

IPv6 Internet Protocol Version 6 [29], a communication protocol. 7

mDNS multicast Domain Name Service [2], a protocol that implements a local distributed name resolving mechanism. 6, 7, 44

MQTT Message Queuing Telemetry Transport is a lightweight, publish-subscribe based network protocol that transports messages between devices. 5–7, 26–28, 44

mTLS Mutual TLS extends the TLS protocol by requiring clients to pass certificates, allowing to provide authorization mechanisms of Application services. 7, 12, 25

NTP Network Time Protocol. 48

PKI Public Key Infrastructure. 10, 49

PKI2go PKI2go is a patent-pending security service provided by Perinet. 10–13, 18, 22, 23, 25, 30, 31, 46, 59

SSH Secure Shell Protocol. 46, 47

TLS Transport Layer Security Protocol. Used by Application Protocols like MQTT or HTTP to allow secure data transfer. 7

UI User Interface. 6–9, 45

E References

[1] S. Cheshire and M. Krochmal. *DNS-Based Service Discovery*. RFC 6763. http://www.rfc-editor.org/rfc/rfc6763.txt. RFC Editor, Feb. 2013. URL: http://www.rfc-editor.org/rfc/rfc6763.txt.

[2] S. Cheshire and M. Krochmal. *Multicast DNS*. RFC 6762. http://www.rfc-editor.org/rfc/rfc6762.txt. RFC Editor, Feb. 2013. URL: http://www.rfc-editor.org/rfc/rfc6762.txt.

[3] Perinet GmbH. Dashboard Container User Guide. PRN.100.461. https://docs.perinet.io/.

[4] Perinet GmbH. periCORE Datasheet. PRN.100.375. https://docs.perinet.io/.

[5] Perinet GmbH. periCORE Development Kit Product Summary. PRN.100.546. https://docs.perinet.io/.

[6] Perinet GmbH. periCORE Development Kit Setup Application Note. PRN.100.376. https://docs.perinet.io/.

[7] Perinet GmbH. periCORE Development Kit User Guide. PRN.100.378. https://docs.perinet.io/.

[8] Perinet GmbH. periCORE Product Summary. PRN.100.301. https://docs.perinet.io/.

[9] Perinet GmbH. PeriLINE Product Summary. PRN.100.386. https://docs.perinet.io/.

[10] Perinet GmbH. periMICA Debian Container User Guide. PRN.100.462. https://docs.perinet.io/.

[11] Perinet GmbH. *periMICA Product ReBranding Application Note*. PRN.100.583. https://docs.perinet.io/.

[12] Perinet GmbH. periMICA Product Summary. PRN.100.389. https://docs.perinet.io/.

[13] Perinet GmbH. periMICA Quick Start Guide. PRN.100.391. https://docs.perinet.io/.

[14] Perinet GmbH. periMICA User Guide. PRN.100.392. https://docs.perinet.io/.

[15] Perinet GmbH. periNODE 0-10V Product Summary. PRN.100.380. https://docs.perinet.io/.

[16] Perinet GmbH. periNODE GPIO Product Summary. PRN.100.382. https://docs.perinet.io/.

[17] Perinet GmbH. periNODE Pt100 Product Summary. PRN.100.381. https://docs.perinet.io/.

[18] Perinet GmbH. periSTART Standard Product Summary. PRN.100.383. https://docs.perinet.io/.

[19] Perinet GmbH. periSWITCH 3-port Product Summary. PRN.100.385. https://docs.perinet.io/.

[20] Perinet GmbH. PKI2go Container User Guide. PRN.100.465. https://docs.perinet.io/.

[21] Perinet GmbH. Security Certificates Installation Guide. PRN.100.447. https://docs.perinet.io/.

[22] Perinet GmbH. Smart Components Datasheet. PRN.100.387. https://docs.perinet.io/.

[23] Perinet GmbH. Starter Kit Plus Product Summary. PRN.100.568. https://docs.perinet.io/.
[24] Perinet GmbH. Starter Kit Plus Quick Start Guide. PRN.100.394. https://docs.perinet.io/.
[25] Perinet GmbH. Starter Kit Plus Security Certificates Installation Guide. PRN.100.397. https://docs.perinet.io/.
[26] Perinet GmbH. Starter Kit Plus User Guide. PRN.100.548. https://docs.perinet.io/.
[27] Perinet GmbH. Starter Kit Product Summary. PRN.100.567. https://docs.perinet.io/.
[28] Perinet GmbH. Starter Kit Quick Start Guide. PRN.100.569. https://docs.perinet.io/.
[29] R. Hinden and S. Deering. *IP Version 6 Addressing Architecture*. RFC 4291. http://www.rfc-editor.org/rfc/rfc4291.txt. RFC Editor, Feb. 2006. URL: http://www.rfc-editor.org/rfc/rfc4291.txt.

F Revision History

Revision	Date	Author(s)	Description
1	April 6, 2022	dheu	Initial Release
2	2022-09-30	asch , dheu , shoe	Update DistanceControl with MQTT configuration, use variable application_admin_user, add Section 6