Usługi i aplikacje Internetu rzeczy (PBL5)

Serwer aplikacji IoT

Aleksander Pruszkowski

Instytut Telekomunikacji Politechniki Warszawskiej

IoTStack – Wykorzystanie doceker'a w Internecie Rzeczy

- IoTStack
  - Zaawansowany Server IoT
    - Dedykowany dla Raspberry PI

      - Autor oznajmia: nie działa na: Raspberry PI zero – ze względu na CPU (?)
      - Sprawdzono – działa także na PC jako maszynach wirtualnych pod VirtualBox
      - Nie wszystkie kontenery działają dla architektury X86_64
    - Bazuje na
      - Infrastruktura konteneryzacji:
        - Docker
      - Infrastruktura wspierająca:
        - DDNS (duckdns), Pi-hole
      - Infrastruktura archiwizacji:
        - Dropbox
      - Infrastruktura IoT:
        - Mosquitto (Mqtt), Node-RED, InfluxDB, Grafana
      - Infrastruktura zarządzania systemem:
        - Portainer
      - Pozostałe elementy
        - Adminer, openHAB, Home Assistant, zigbee2mqtt, TasmoAdmin, Plex media server, Telegraf, RTL_433, EspruinoHub, MotionEye, MariaDB, Plex, Homebridge

Obrazy i ikony pobrane ze stron projektów

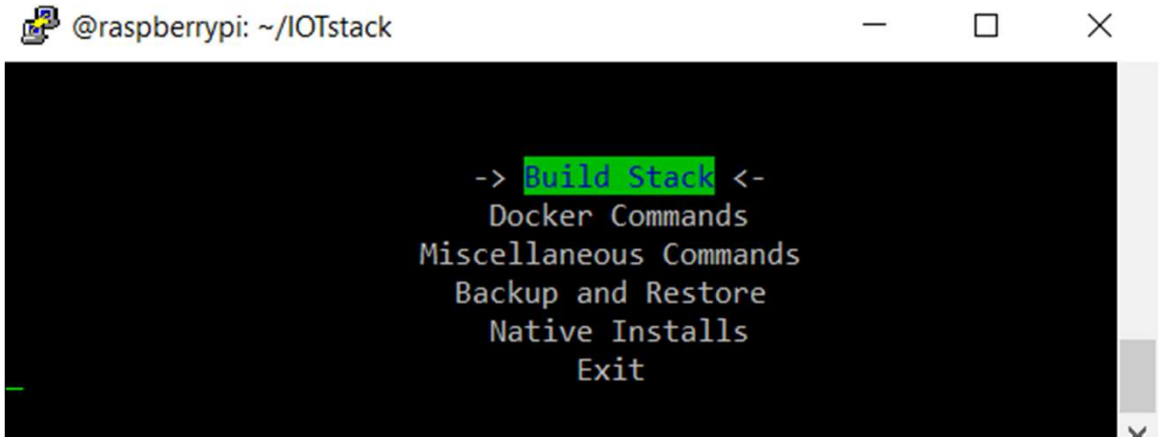
- IoTStack – po co?
  - Otrzymujemy jednolity system uruchamiania wielu różnych projektów dla systemu Internetu Rzeczy bez konieczności zgłębiania ich architektury/budowy
    - Co ważniejsze bez konfliktów np. związanych z zależnościami
      - Aplikacja A chce libssl w wersji X a aplikacja B w wersji Y
  - Mamy możliwość separacji uruchamianych systemów Internetu Rzeczy
    - W lekki sposób zwiększamy bezpieczeństwo całego systemu
  - System zapewnia podstawową konfiguracje dla typowego użytkownika
    - Broker MQTT + Node-Red + baza danych + system wizualizacji danych + narzędzia wspierające (ddns, dropbox, …) + bramki/całe systemy dla IoT
    - Podejście podobne do stosowanego w domowym sprzęcie dostępu do Internetu: włącz i używaj

- IoTStack – proces instalacji środowiska
  - Przygotowanie – linia poleceń
    - zależności

```
sudo apt-get update ; sudo apt install -y git
```

    - właściwa instalacja

```
git clone https://github.com/SensorsIot/IOTstack.git ~/IOTstack
cd ~/IOTstack
./menu.sh
```



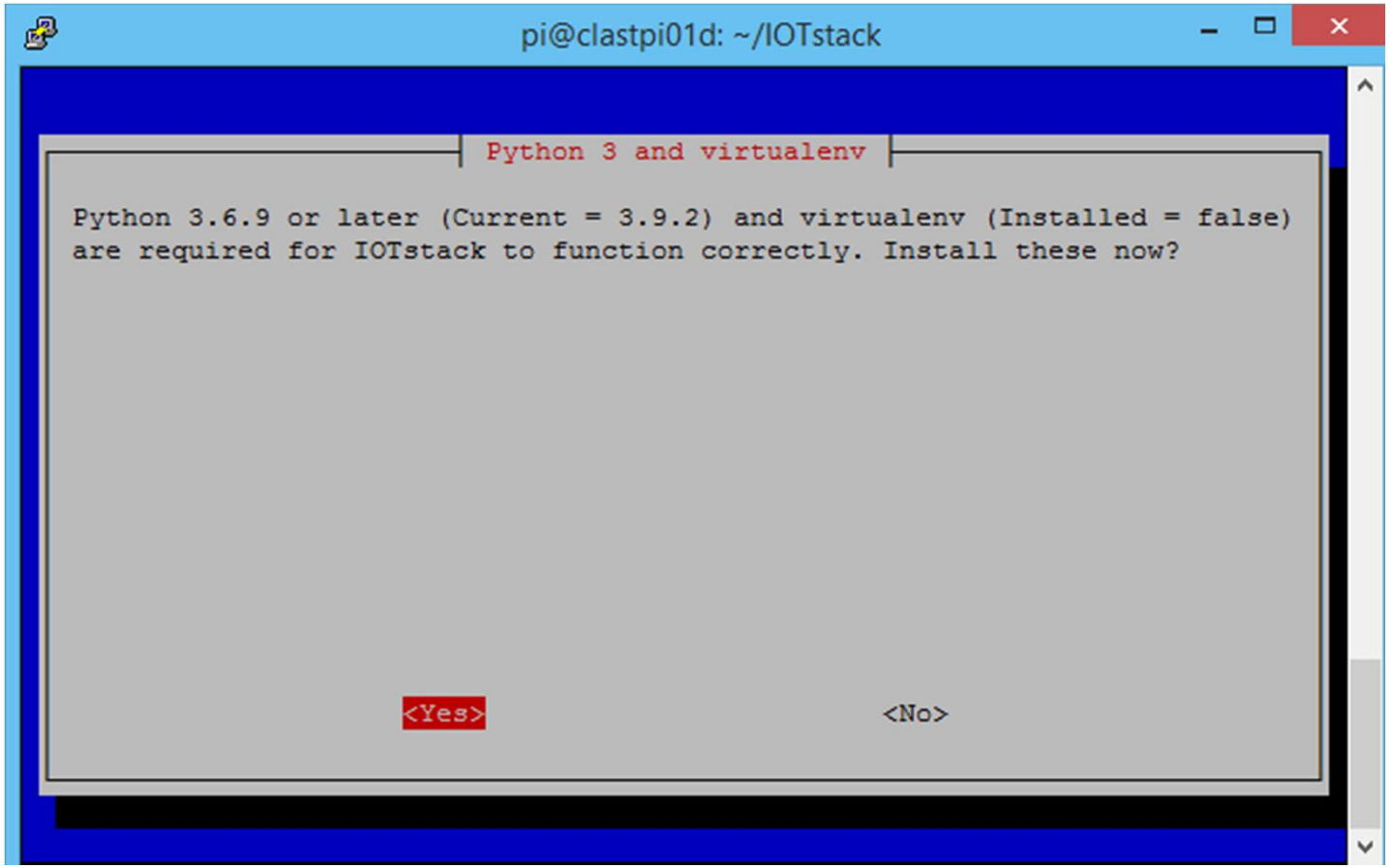
Warto przed powyższymi krokami sprawdzić czy nasze konto dołączono do grupy docker:

```
id `whoami` | grep docker
```

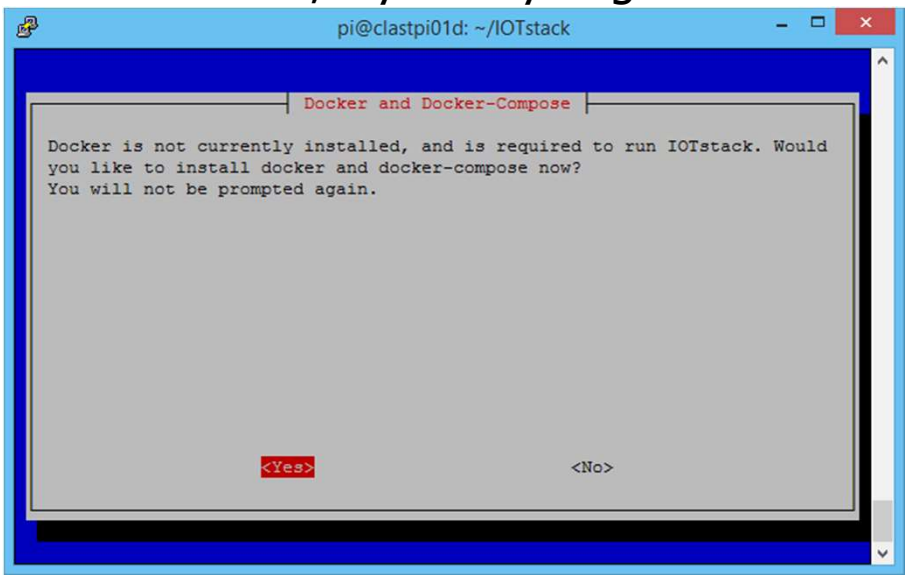
Aby się dołączyć wykonaj:

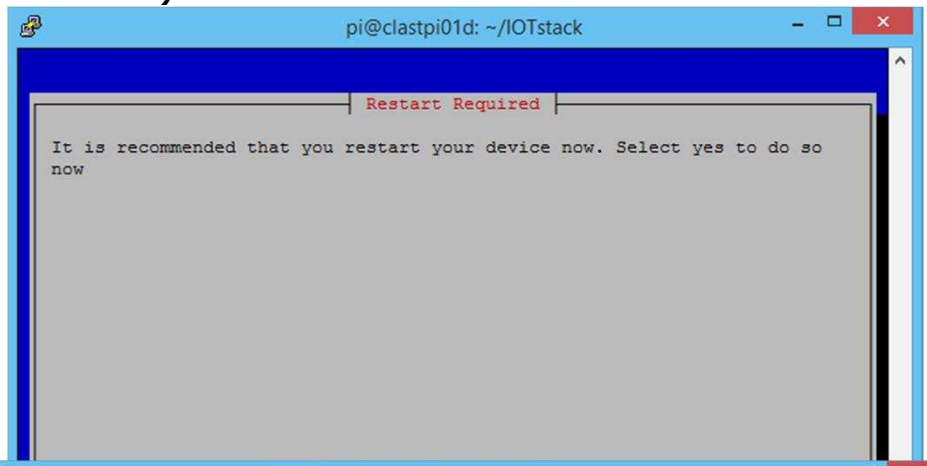
```
sudo usermod -a -G docker `whoami`
sudo reboot
```

- IoTStack – proces instalacji środowiska, cd.
  - Przygotowanie – wymagane modyfikacje środowiska – instalacja języka Python w wymaganej wersji oraz pakietu Virtualenv (pomaga pracować równocześnie z wieloma wersjami pakietu Python)



- IoTStack – proces instalacji środowiska, cd.
  - Przygotowanie – wymagane modyfikacje środowiska – instalacja pakietu docker, wybór wymaga restartu środowiska)

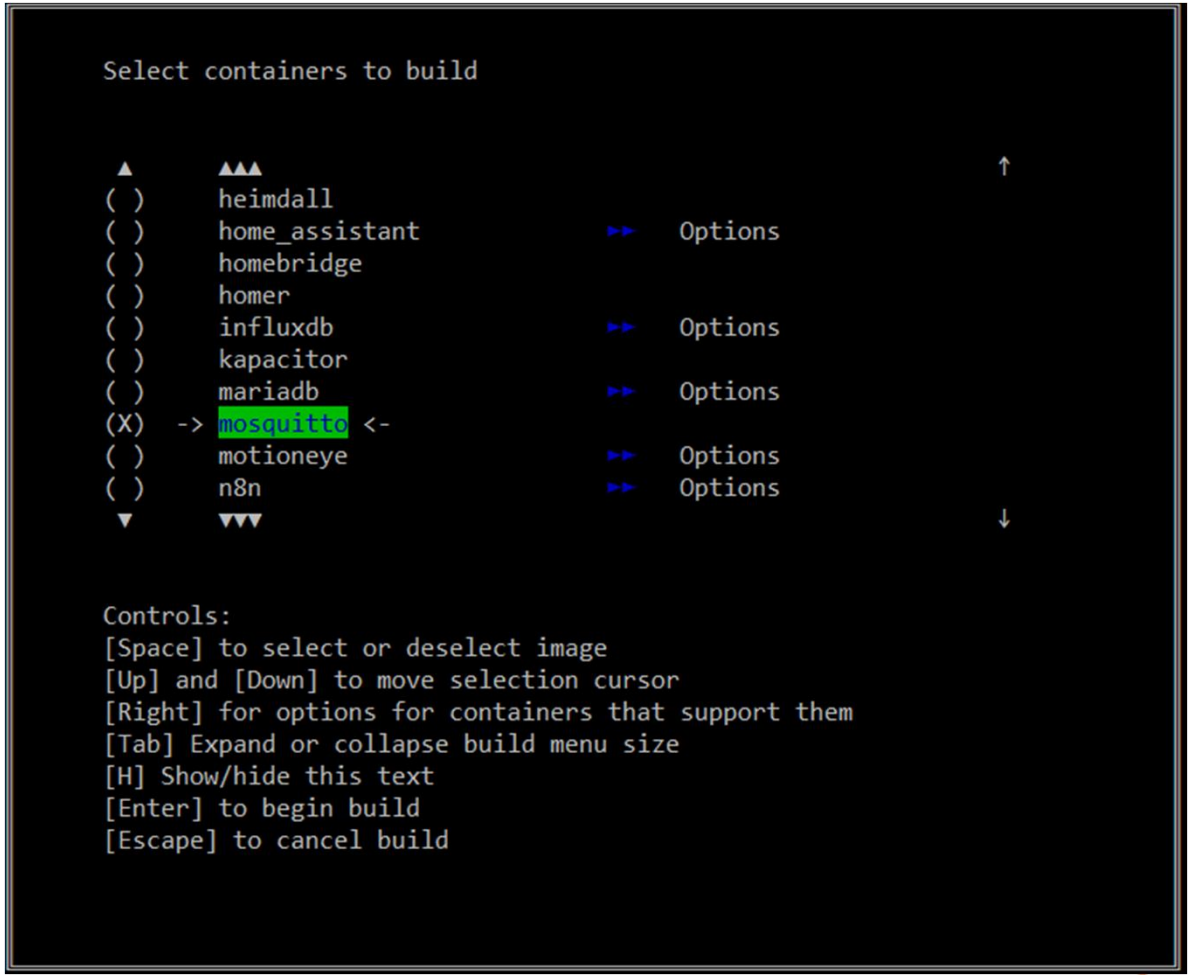




- Gdy pojawiają się problemy

można inaczej wywołać ./menu.sh

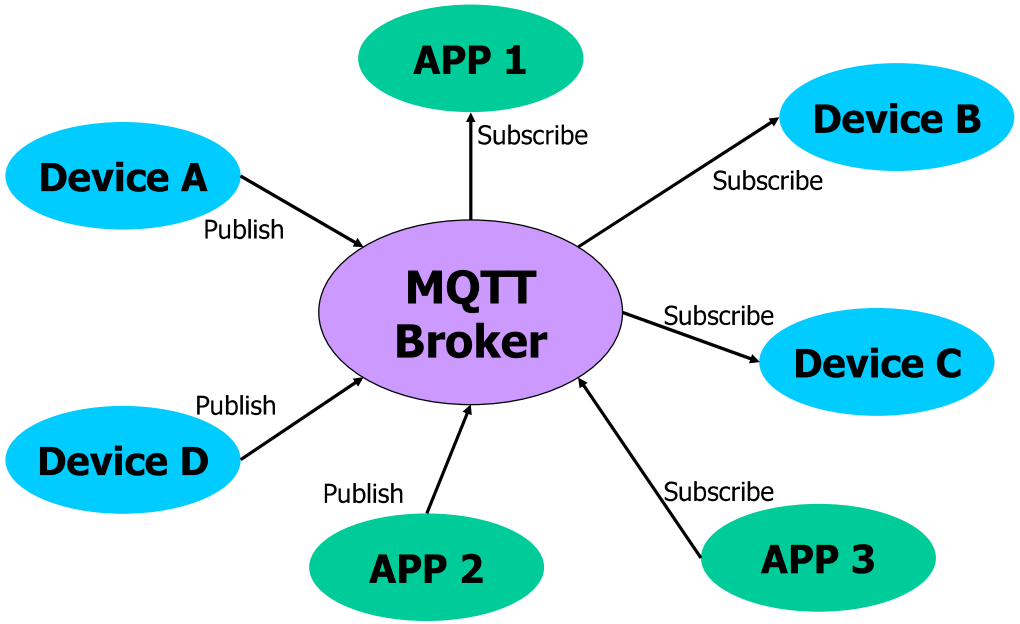
- IoTStack – proces instalacji środowiska, cd.
  - Przygotowanie – wybór i konfiguracja kontenerów
    - Mosquitto [https://mosquitto.org/]



- IoTStack – proces instalacji środowiska, cd.
  - Mosquitto - broker protokołu MQTT
    - Podstawy protokołu MQTT to model współpracy „publish - subscribe"
      - [https://mqtt.org/]
    - Komunikacja opiera się na wirtualnych kanałach identyfikowanych za pomocą tematów (ang. Topic)
      - Strona publikująca (ang. publisher) rozgłasza wiadomość określając jej temat
        - Nie ma ścisłych reguł relacji temat-wiadomość – temat nie musi opisywać treści wiadomości
      - Strony subskrybujące (ang. subscriber) rejestrują się na wiadomości o określonych tematach
      - Gdy broker – element centralny - odbierze od publikatorów wiadomość o określonym temacie - zostanie ona rozesłana do zainteresowanych subskrybentów

- IoTStack – proces instalacji środowiska, cd.
  - Protokół MQTT to…
    - Lekki protokół komunikacji urządzeń IoT o nazwie „**MQ T**elemetry **T**ransport"
      - Bazuje na łączności z wykorzystaniem protokołu TCP
        - Wyjątkiem jest odmiana zwana MQTT-SN - tzw. MQTT dla NO-TCP devices
        - TCP to ciężki protokół - dla węzłów o małych zasobach trudny w implementacji
      - Schemat komunikacji oparty jest o wzorzec publikacja/subskrypcja
        - Nie jest wymagana znajomość adresów poszczególnych elementów systemu a tylko węzła centralnego
    - Zaleta
      - Brak ścisłego związania strony publikującej z subskrybującą
        - Wielu publikatorów może przesyłać wiadomości do jednego subskrybenta
        - Podobnie jeden publikator może rozsyłać wiadomość wielu subskrybentom
    - Wady
      - Subskrybent nie wie od kogo otrzymał wiadomość

- IoTStack – proces instalacji środowiska, cd.
  - Protokół MQTT to…
    - Węzeł centralny (Broker) pośredniczy w inteligentnym dostarczaniu wiadomości
      - Węzły łączą się tylko z brokerem



    - Broker Mosquitto pozwala także łączyć się z innymi brokerami tworząc z nimi mosty
      - Rozszerzenie zwiększa skalowalność i bezpieczeństwo

- IoTStack – proces instalacji środowiska, cd.
  - Protokół MQTT to…
    - Tematy mogą tworzyć hierarchię, elementy rozdzielamy znakiem '/'
      - np.: pokój w mieszkaniu numer 12 …

      `Budynek/mieszkanie/12/pokoj`
    - Tematy wiadomości mogą opisywać ich znaczenie
      - np.: temperatura w pokoju 12

      `Budynek/mieszkanie/12/pokoj/temperatura`
    - Subskrybenci mogą oczekiwać wiadomości wieloznacznych (ang. wildcard)
      - `#` - interesuje mnie każda wiadomość
      - `Budynek/mieszkanie/12/#` - czyli interesuje mnie wszystko co publikują węzły w mieszkaniu numer 12
        - Znak # zastępuje całe poddrzewo wiadomości
      - `Budynek/mieszkanie/+/+/temperatura` – wszystkie odczyty temperatury we wszelkich mieszkaniach i ich pokojach
        - Znak + zastępuje dowolną nazwę na jednym poziomie

- IoTStack – proces instalacji środowiska, cd.
  - Protokół MQTT to…
    - System nie definiuje tematów – tu ma dowolność twórca węzłów
      - Pamiętajmy o ograniczeniach
        - kodowanie UTF-8
        - faktyczna długość tematu nie może być większa niż 65 536B
    - System nie definiuje treści wiadomości – tu także twórca ma dowolność
      - Ograniczeniem jest długość wiadomości 268 435 456B
        - Uwaga odbiorcy musza być wstanie odebrać wiadomość i muszą ją zrozumieć – publikujący nie może zawęzić grupy odbiorców poza ustaleniem właściwego tematu
    - Odbiorca wiadomości nie wie kto jest jej publikatorem
      - o ile nie dodamy takiej informacji w jej treści lub inteligentnie nie zdefiniujemy tematu

- IoTStack – proces instalacji środowiska, cd.
  - Protokół MQTT to…
    - Istnieje duża liczba bibliotek dla wielu języków programowania – np. Paho (Eclipse) [https://www.eclipse.org/paho/]

Client	MQTT 3.1	MQTT 3.1.1	MQTT 5.0	LWT	SSL / TLS	Automatic Reconnect	Offline Buffering	Message Persistence	WebSocket Support	Standard MQTT Support	Blocking API	Non-Blocking API	High Availability
Java	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Python	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔	✖
JavaScript	✔	✔	✖	✔	✔	✔	✔	✔	✔	✖	✖	✔	✔
GoLang	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
C	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
C++	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Rust	✔	✔	✖	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔
.Net (C#)	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✖	✔	✖
Android Service	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
Embedded C/C++	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✔	✔	✖

    - Używane są implementacje dla węzłów niewspieranych przez popularne systemy operacyjne (tzw. Bare Metal): Arduino, ESP8266, ESP32, mBED, …

Źródło: http://www.eclipse.org/paho

- IoTStack – proces instalacji środowiska, cd.
  - UWAGA! Mosquitto – dostarczane z IoTStack zawiera dość uproszczoną konfigurację
    - Jak ją odkryć – analizując plik ~/IOTstack/docker-compose.yml

```
...
mosquitto:
 volumes:
 - ./volumes/mosquitto/config:/mosquitto/config
...
```

Miejsce składowania plików konfiguracyjnych Na maszynie „hostującej"

Gdzie kontener będzie „widział" pliki

Zatem konfiguracja jest zapisana w ./volumes/mosquitto/config/**mosquitto.conf**

```
listener 1883
persistence true
persistence_location /mosquitto/data
log_dest stdout
log_timestamp_format %Y-%m-%dT%H:%M:%S
connection_messages false
allow_anonymous true
```

- Powyższa konfiguracja „ma problemy" z bezpieczeństwem(!)
  - „Wpuści" każdego użytkownika (opcja: allow_anonymous ustawiona na: true)
- Przypomnienie - zmiana konfiguracji „z poza" kontenera wymaga tutaj
  - 1)wyłączenia kontenera
  - 2)zmian pliku na maszynie hostującej
  - 3)ponowne uruchomienie kontenera

- IoTStack – proces instalacji środowiska, cd.
  - UWAGA Mosquitto – dostarczane z IoTStack zawiera dość uproszczoną konfigurację, cd.
    - Jak przeprowadzić proces z linii poleceń – jedna z metod:
      - Odczytanie Id kontenera

```
>docker ps
CONTAINER ID IMAGE              COMMAND                  CREATED      STATUS                  PORTS                    NAMES
0d95f1beef8a iotstack_mosquitto "/docker-entrypoint...." 4 weeks ago  Up 3 hours (healthy)    0.0.0.0:1883->1883/tcp   mosquitto
...
```

      - Zatrzymanie kontenera o określonym ID

>docker stop 0d95f1beef8a lub >docker kill 0d95f1beef8a – różnica to sposób unicestwienia procesu

      - Edycja pliku konfiguracyjnego

```
>nano ./volumes/mosquitto/config/mosquitto.config
```

      - Ponowne uruchomienie kontenera

```
>docker run -it --rm -d -p 1883:1883 -v ./volumes/mosquitto/config:/mosquitto/config iotstack_mosquitto
```

Lub użyć narzędzie `docker-compose` wywołując:

```
>docker-compose up –d
```

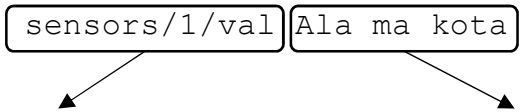
- IoTStack – proces instalacji środowiska, cd.
  - Testowanie/używanie systemu MQTT
    - Po co? – testowanie czy system zadziała podczas dalszych prac
      - Na Raspberry PI, może wymagać instalacji: `sudo apt-get install mosquitto-clients`
      - Narzędzia linii poleceń (testowanie lokalne)
        - mosquitto_sub – subskrypcja określonych wiadomości, dla przykładu subskrybujmy wszelkie wiadomości:

```
mosquitto_sub -h 10.0.126.196 -p 1883 -t "#" -v
```

        - mosquitto_pub – publikacja wiadomości:

```
mosquitto_pub -h 10.0.126.196 -p 1883 -t sensors/1/val -m "Ala ma kota"
```

Dla powyższego przykładu otrzymamy (w terminalu wywołaniu subskrybenta):



W jakim temacie jest odebrana wiadomość (# - czyli subskrybowano wszelkie wiadomości), ta część wyniku uzyskiwana jest poprzez użycie opcji -v

Odebrana wiadomość

Podczas pracy z kontenerami proszę pamiętać iż używanie interfejsu loopback (127.0.0.1) może być zawodne – wtedy lepiej używać numeru IP karty sieciowej istniejącej i działającej na tym komputerze

- IoTStack – proces instalacji środowiska, cd.
  - Testowanie/używanie systemu MQTT
    - Realne testy na Raspberry PI – pomiar temperatury CPU
      - Okno I – cykliczna (2sek.) publikacja:

```
watch "/usr/bin/vcgencmd measure_temp | mosquitto_pub -h 10.0.126.196 -t \"t1\" -l"
```

      - Okno II (sztuczne wymuszenie zmian obciążenia CPU dla wywołania zmiany temperatury, może wymagać instalacji pakietu: `sudo apt-get install stress`) :

```
stress -cpu 3
```

      - Obserwacja publikowanych wiadomości

```
mosquitto_sub -h 10.0.126.196 -t "t1" -v
```

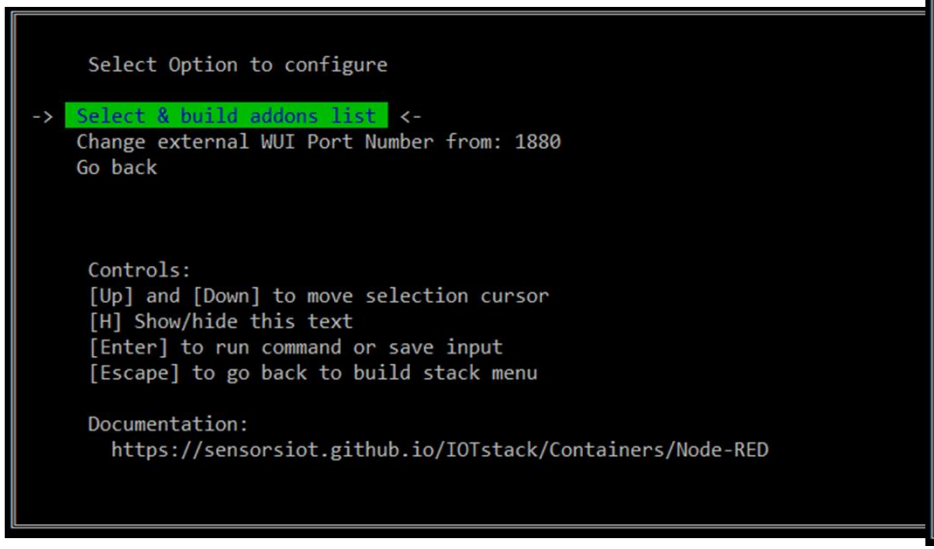
      - Dając przykładowy wynik:

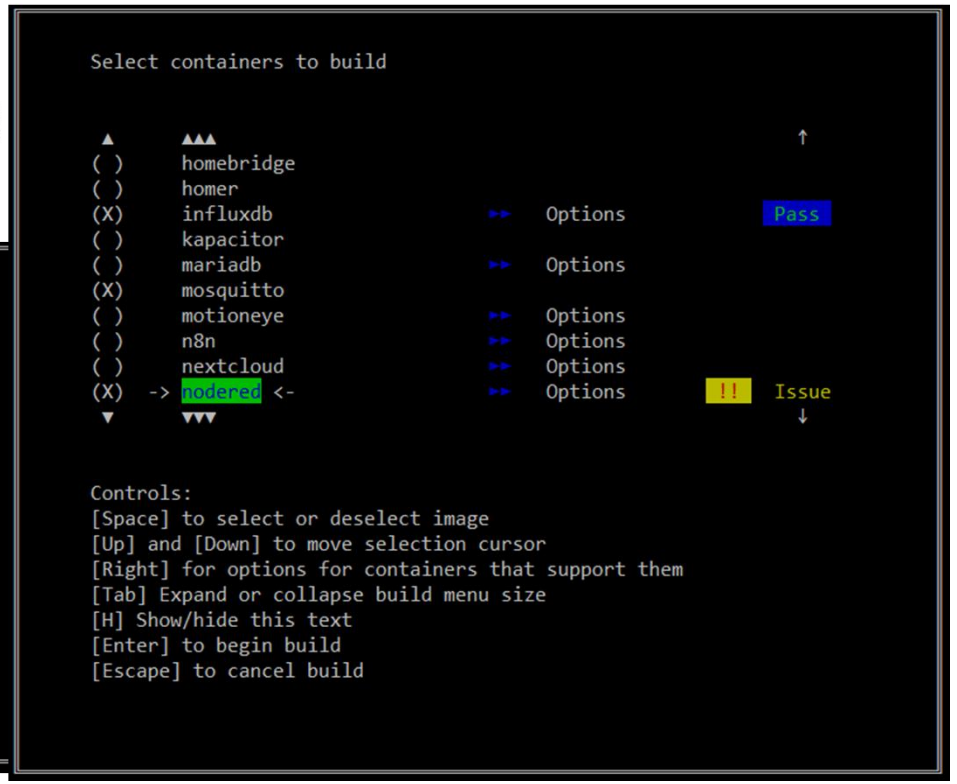
```
t1 temp=33.6'C
t1 temp=33.1'C
t1 temp=34.0'C
...
```

W przykładzie powyżej pominięto informacje o numerze portu, zakładając ze kontener mosquitto działa na domyślnym 1883

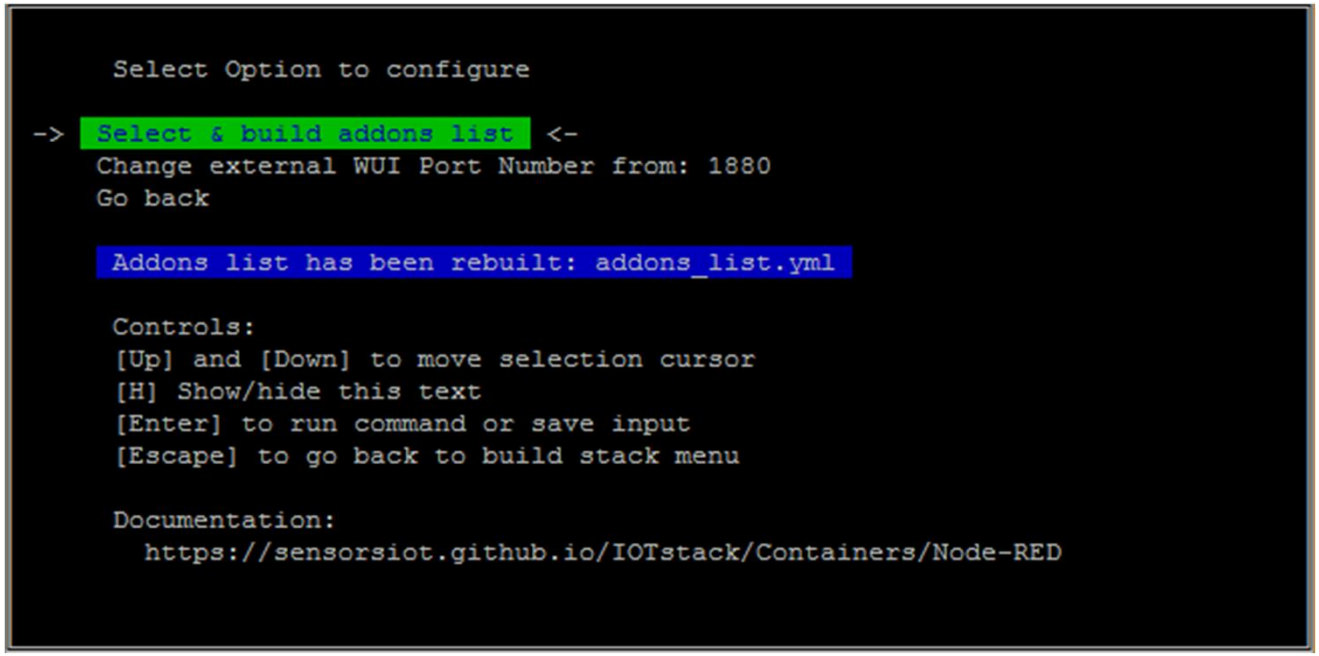
- IoTStack – proces instalacji środowiska, cd.
  - Przygotowanie – wybór i konfiguracja kontenerów, cd.
    - Node-RED

    + wstępna konfiguracja (numer portu)

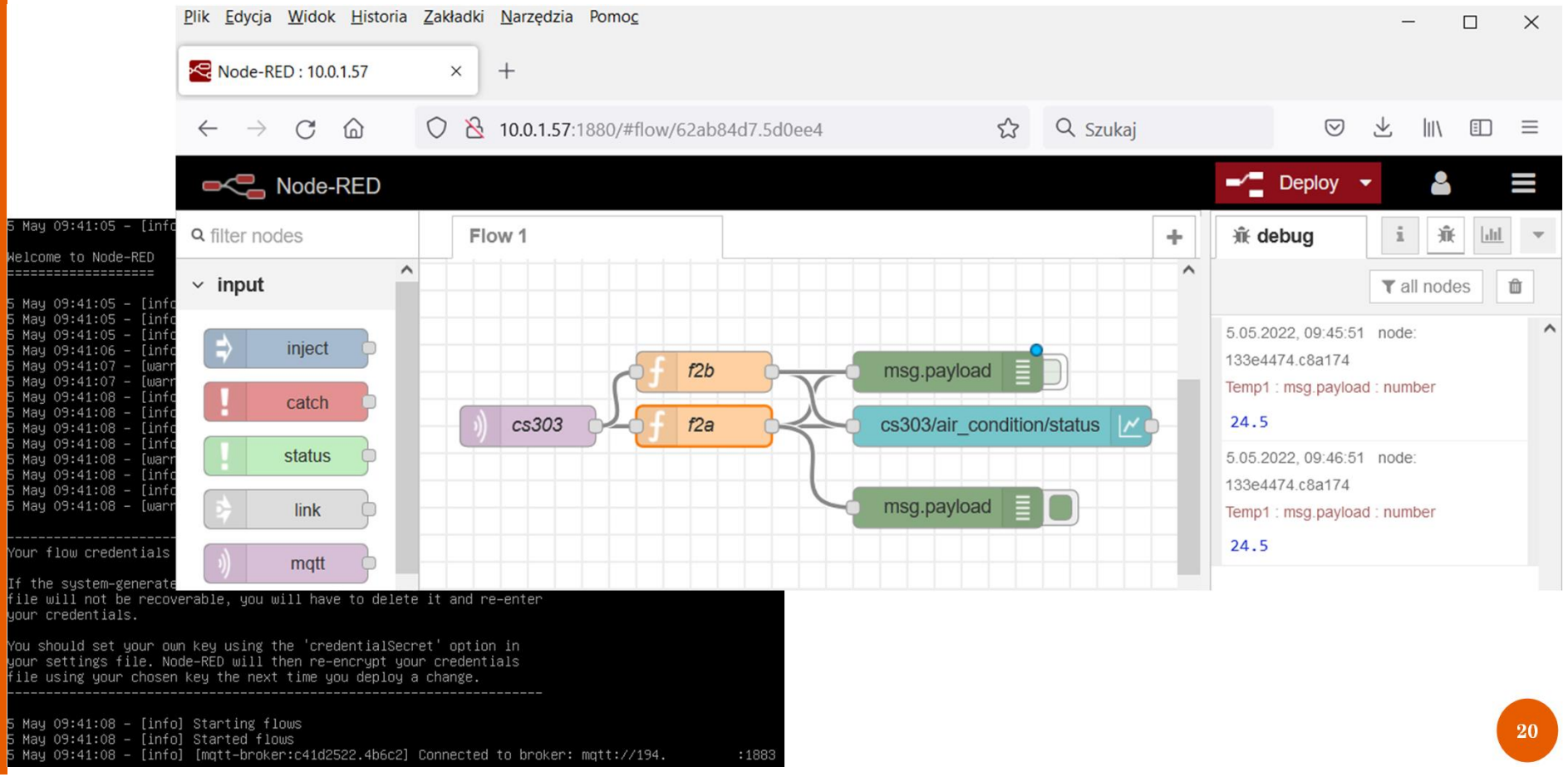




- IoTStack – proces instalacji środowiska, cd.
  - Node-RED [https://nodered.org/]
    - Uwaga w najnowszej wersji IoTStack brak pliku addons_list.yml owocuje błędem tworzenia kontenerów – aby temu zapobiec trzeba wejść w opcje nodered a potem wybrać „Select & build addons list" tam wybrać co nas interesuje (lub zostawić domyślne dodatki) i klawiszem Enter zatwierdzić wybór – co zapisze wspomniany plik

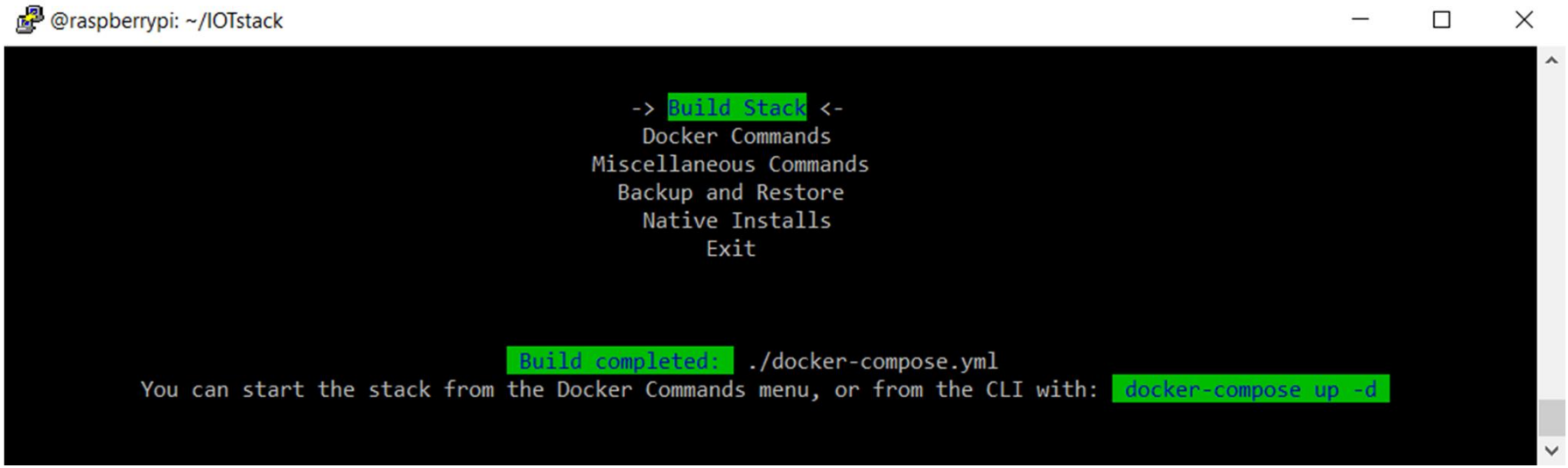


- IoTStack – proces instalacji środowiska, cd.
  - Node-RED [https://nodered.org/]
    - Środowisko tworzenia i uruchamiania prostych aplikacji dla urządzeń Internetu Rzeczy
      - Aplikacja ≡ flow



IoTStack – Wykorzystanie doceker'a w Internecie Rzeczy

- IoTStack – użytkowanie (instalacja)
  - Finalizacja tworzenia i instalacji kontenerów



- IoTStack – użytkowanie (instalacja)
  - Finalizacja tworzenia i instalacji kontenerów, cd.
    - Budowanie kontenerów rozpoczyna polecenie:

      `>docker-compose up -d`

      - Budowa jest zgodna z zawartością pliku `docker-compose.yml` - fragment:

```
version: '3.6'
services:
  mosquitto:
    container_name: mosquitto
    build:
      context: ./.templates/mosquitto/.
      args:
      - MOSQUITTO_BASE=eclipse-mosquitto:latest
    restart: unless-stopped
    environment:
    - TZ=Etc/UTC
    ports:
    - "1883:1883"
    volumes:
    - ./volumes/mosquitto/config:/mosquitto/config
    ...
 nodered:
    container_name: nodered
    build: ./services/nodered/.
    restart: unless-stopped
    user: "0"
    environment:
    - TZ=Etc/UTC
    ports:
    - "1880:1880"
    ...
    devices:
    - "/dev/gpiomem:/dev/gpiomem"
...
```

Uwaga! Wybranie w menu.sh Node-Red może wiązać się z komunikatem o błędzie w pliku tzw. „adds on".
Bez rozwiązania tego problemu IoTStack może nie chcieć poprawnie zbudować wybranych kontenerów.

- IoTStack – użytkowanie (instalacja)
  - Finalizacja tworzenia i instalacji kontenerów, cd.
    - Proces pobierania obrazów i budowania kontenerów realizowany jest automatycznie, zajmuje sporo czasu(!)

```
ap@raspberrypi:~/IOTstack $ docker-compose up -d
WARNING: Some networks were defined but are not used by any service: nextcloud
Creating network "iotstack_default" with driver "bridge"
Pulling grafana (grafana/grafana:)...
latest: Pulling from grafana/grafana
57fb4b5f1a47: Pull complete
eec1deac55e4: Pull complete
9d627d22c017: Extracting [=>                                                 ]  32.77kB/1.362MB
44402a81b422: Download complete
17d2ae62c7ac: Download complete
9ac7996e04d3: Downloading [===>                                              ]   4.84MB/75.85MB
8595927c6fb8: Download complete
80551ad5d37f: Download complete
926e17151ee8: Download complete
```

    - Zatrzymywanie działających a wykreowanych z `docker-compose.yaml` obrazów najprościej wykonać poprzez

```
>docker-compose down
```

Zadanie:
Zainstalować niezbędne dla działania klientów komponenty MQTT (zarówno publikujących jak subskrybujących) wybierając broker Mosquitto, zainstalować także Node-Red – etap I

Napisać prostą aplikacje (flow) dla tego środowiska (opis znajduje się w pliku Node-Red_pomoc.pdf umieszczonym na serwerze studia) filtrującą wiadomości i wykreślającą tendencje zmian odebranych danych numerycznych w czasie (dane dla brokera generować za pomocą mosquitto_pub) - etap II

- IoTStack – proces instalacji środowiska, cd.
  - UWAGA! Mosquitto – dostarczane z IoTStack zawiera dość uproszczoną konfigurację, cd.
    - Co można zrobić by zwiększyć bezpieczeństwo brokera
      - Dodać w konfiguracji

```
allow_anonymous false
password_file ./my_passwd
```

      - Kreacja dla pierwszego użytkownika w my_passwd

```
mosquitto_passwd -c –b ./my_passwd tester TesterPassPBL5zima
```

      - Kreowanie następnych użytkowników – pomijamy opcję „-c"
      - Uwaga – powyższe wprowadza tylko fazę obowiązkowego uwierzytelnienia nie szyfrowania danych (!)

- IoTStack – proces instalacji środowiska, cd.
  - UWAGA Mosquitto – dostarczane z IoTStack zawiera dość uproszczoną konfigurację, cd.
    - Co zrobić gdy coś po stronie brokera nie działa – zaglądamy w logi, tu dla domyślnej instalacji kontenerów niezbędna jest zmiana konfiguracji
    - Zatrzymujemy kontener i edytujemy plik `./IOTstack/volumes/mosquitto/config/mosquitto.conf`
    - Linie odpowiedzialne za logowanie pracy brokera to te z opcjami `log_xxx`:

```
log_dest file /mosquitto/log/mosquitto.log     #gdzie plik z logami ma się znaleźć
log_dest stdout                          #logowanie na STDOUT – z kontenerami
                                         #działającymi w trybie HEADLESS mało przydatne
log_timestamp_format %Y-%m-%dT%H:%M:%S         #format nagłówka logowania (opcja działa
                                               #z nowszymi brokerami mosquitto)
log_type all       #co logować: tu wszystko (uwaga na „zajechanie" pamięci SD)
                   #inne opcje: debug, error, warning, notice, information, subscribe,
                   #unsubscribe, websockets, none
```

    - I uruchamiamy ponownie kontener

- IoTStack – proces instalacji środowiska, cd.
  - Testowanie/używanie systemu MQTT
    - Kod w języku Python – subskrypcja z uwierzytelnieniem
      - Zakłada się że biblioteka PAHO MQTT jest zainstalowana w systemie

```
import paho.mqtt.client as mqtt

def on_connect(mqttc, obj, flags, rc):            #gdy zestawiono/niezestawio polaczenie
    print("rc: "+str(rc))

def on_message(mqttc, obj, msg):                          #gdy odebrano wiadomosc
    print(msg.topic+" "+str(msg.qos)+" "+str(msg.payload))

def on_subscribe(mqttc, obj, mid, granted_qos):           #gdy zestawiono subskrypcje
    print("Subscribed: "+str(mid)+" "+str(granted_qos))

mqttc = mqtt.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.on_subscribe = on_subscribe

mqttc.username_pw_set("tester", "TesterPassPBL5zima")
mqttc.connect("10.0.126.196", 1883, 60)
mqttc.subscribe("t1", 0)

mqttc.loop_forever(timeout=1.0, max_packets=5, retry_first_connection=False)
```

- IoTStack – proces instalacji środowiska, cd.
  - Testowanie/używanie systemu MQTT
    - Kod w języku Python – publikacja z uwierzytelnieniem
      - Zakłada się że biblioteka PAHO MQTT jest zainstalowana w systemie

```
import paho.mqtt.publish as publish

message="Wiadomosc xxxxxx"
my_auth={'username': "tester", 'password': "TesterPassPBL5zima"}

publish.single("t1", message, hostname="10.0.126.196", port=1883, auth=my_auth)
```

Zadanie:
Dodać do systemu IoT - posiadającego broker MQTT akceptującego wyłącznie połączenia uwierzytelnione - klienta publikującego temperaturę CPU (emulowane za pomocą losowych wartości z zakresu 0…100) oraz klienta subskrybującego. Niech każdy z klientów ma własne uwierzytelnienia a ich implementacja będzie zapisana w języku Python.

Publikuj różne typy danych w osobnych tematach, zaproponuj przemyślną ich hierarchę i nazwy (typami danych mogą być temperatura CPU, zajętość dysku, ilość odebranych i wysłanych danych poprzez kartę sieciową, itd.)

IoTStack – Wykorzystanie doceker'a w Internecie Rzeczy

Dodatek A:

W języku Python możliwe jest wywołanie poleceń systemowych czy wręcz innych aplikacji – pobieranie temperatury:

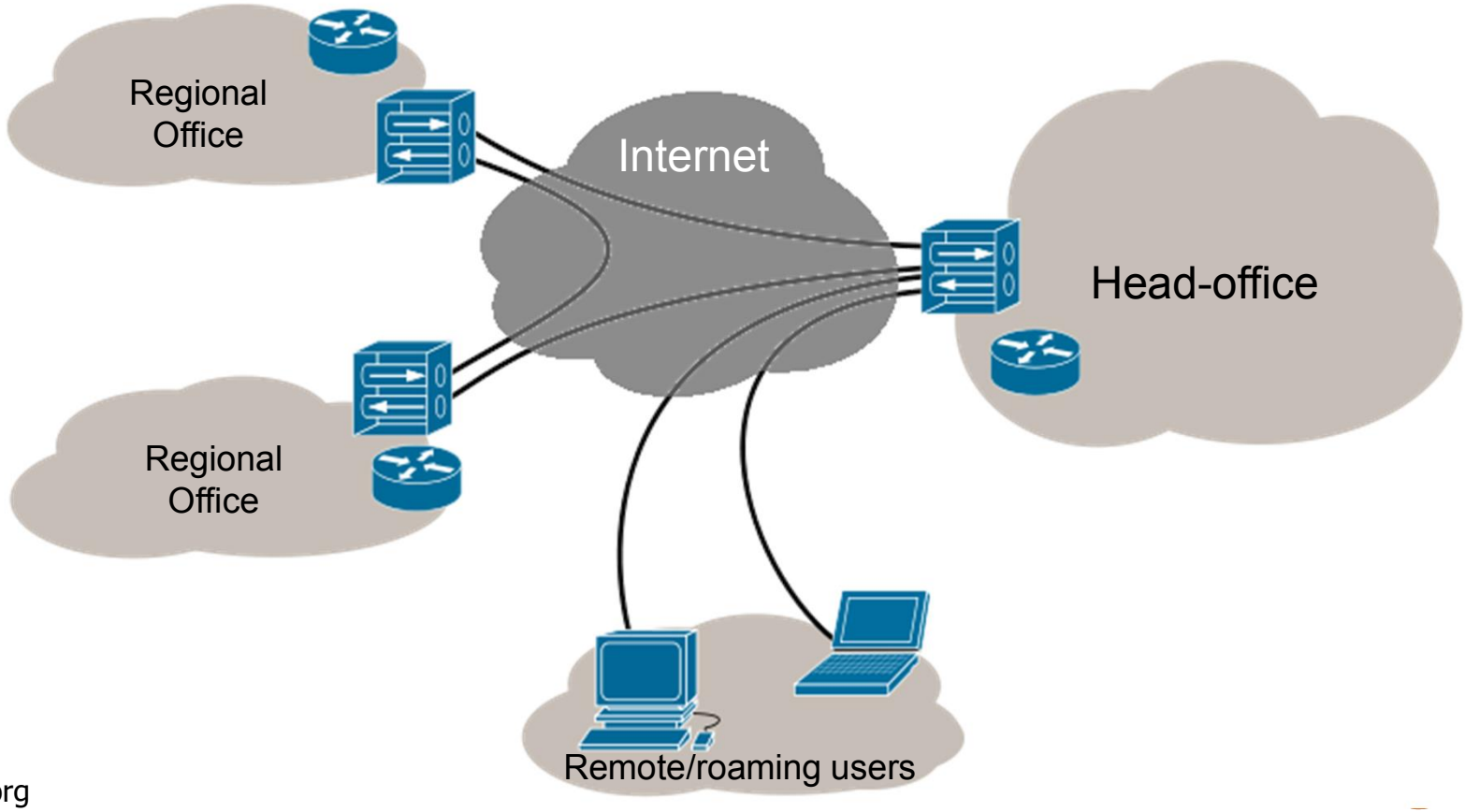
```
import subprocess
invo="/usr/bin/vcgencmd measure_temp"
p=subprocess.run(invo.split(), capture_output=True, text=True, input="hello").stdout
print(p)
```

lub pobieranie wielkości głównego systemu plików:

```
from subprocess import PIPE, Popen
process = Popen(args="df -h | grep \"/\$\"", stdout = PIPE,shell = True)
p=process.communicate()[0]
print(p)
```

VPN - co to takiego?

- Problem dostępu do serwera z aplikacjami (OpenVPN)
  - VPN - co to takiego?
    - Definicja: VPN - Virtual Private Network



Żródło: en.wikipedia.org

- VPN - co to takiego?
  - Klasyfikacja VPN
    - protokół użyty do budowy tunelu
    - miejsce zakończenia tuneli (np.: klient, dostawca sieci - niemal transparenty dla użytkowników)
    - wspierana topologa (np.: site-to-site, network-to-network, ...)
    - poziom dostarczanych zabezpieczeń/usług np.:
      - tylko tunelowanie ruchu
      - tunelowanie + szyfrowanie
      - tunelowanie + szyfrowanie + kompresja)
    - warstwa OSI w jakiej działa VPN (typowo: 2 lub 3)
    - liczba jednocześnie gwarantowanych połączeń (np.: tylko jedno lub wiele)

- VPN - co to takiego?
  - Implementacje VPN
    - SSH (Secure Shell)
      - OpenSSH (OpenBSD Secure Shell) - niemal domyślnie instalowany w wielu dystrybucjach Linux
      - SSHuttle - VPN bez konfigurowania serwera
    - PPTP (Point to Point Tunneling Protocol)
      - Dobrze znany i powszechnie używany protokół tunelowania ruchu
      - Zawiera dużą liczbę błędów - jest nie zalecany dla bezpiecznych wdrożeń
      - Protokół opracowany przez firmę Microsoft - standardowo obecny w systemach operacyjnych tej firmy (od wersji Windows 98 i NT)
    - L2TP (Layer 2 Tunneling Protocol)
      - Działa w warstwie 2
      - Nie zapewnia poufności i mocnego uwierzytelniania
        - zapewnione może to być np.: poprzez wykorzystanie protokołu IPsec (Internet Protocol Security, IP Security)

- VPN - co to takiego?
  - Implementacje VPN, cd.
    - Windows SSTP (Secure Socket Tunneling Protocol)
      - Tuneluje ruchu PPP (point-to-point protocol) poprzez połączenia SSL/TLS
      - Używa port 443 TCP
        - może być instalowany w niesprzyjających środowiskach - np.: polityka bezpieczeństwa dopuszcza otwarcie na routerach dostępowych jedynie określonych portów
      - Brak znanych i otwartych implementacji (poza systemem Windows)
    - OpenVPN
      - Pakiet zapewniający tworzenie bezpiecznych połączeń punkt-punkt lub strona-strona
      - Może przenosić ruch w warstwach 3 lub 2
      - Tworzony tunel może przenosić ruch wykorzystując protokoły UDP lub TCP (mniej zalecane - problem TCP over TCP)
      - Otwierany na serwerze port „nasłuchujący klientów” może być dowolny (!)
      - Do szyfrowania używa biblioteki OpenSSL oraz protokoły SSLv3/TLSv1
      - Dostępny dla wielu platform (GNU/Linux, Android, OpenBSD/FreeBSD/NetBSD, Mac OS X, Windows XP/Vista/7/10)

- VPN - co to takiego?
  - OpenVPN, cd.
    - Utworzony jako „Open source” przez James Yonan
    - Zapewnia wielość konfiguracji i opcji uruchamiania
    - Wspiera dynamicznie zmieniające się numery IP urządzeń (DHCP, DDNS)
    - Zapewnia wysoką skalowalność (setki i tysiące użytkowników)
    - Jest lekki a funkcjonalnie odpowiada funkcjom wspieranym przez IPSec

Konfiguracja OpenVPN

- OpenVPN - przykładowa realizacja VPN
  - Uruchomienie manualne
    - Co uruchamiamy na serwerze

      ```
      openvpn --config myvpn_server.conf
      ```

    - Co uruchamiamy na komputerze klienckim

      ```
      openvpn myvpn_client.conf
      ```

      - Brak opcji --config podczas wywołania - jest tu akceptowany
  - Co różni pliki myvpn_server.conf i myvpn_client.conf - bardzo wiele!
  - Różnice konfiguracji - serwer
    - myvpn_server.conf to ustawienia: (pamiętajmy o równoległości ustawień!)
    - listen [adress]
      - adres - z jakim adresem klienci mają się łączyć (uwaga NAT!)
      - np.: listen 0.0.0.0 lub listen 192.168.1.1 (np.: gdy mamy wiele interfejsów sieciowych i tylko jeden z nich ma być użyty)

- OpenVPN - przykładowa realizacja VPN
  - Różnice konfiguracji - serwer, cd.
    - topology [typ_topologi]
      - typ_topologi - określa jak serwer udostępnia sieć klientom, możliwe typy topologii:
        - subnet - w tym trybie alokuje się pojedynczy numer IP każdemu klientowi
        - net30 - topologia point-to-point wymuszająca alokacje jednej podsieci klasy „/30” dla każdego klienta
        - p2p - topologia point-to-point gdzie interfejsy klienckie (tun) odpowiadają interfejsom serwerowym (tun), ten tryb alokuje jedno IP dla każdego klienta (w prze
    - server [adres maska]
      - adres - numer IP pod jakim zakończy się tunel, np.: 10.8.1.0
      - maska - maska sieci zakończenia tunelu, np.: 255.255.255.0
      - uwaga! serwer w tworzonej sieci VPN otrzymuje dla powyższych przykładów jako pierwszy - adres: 10.8.1.1 natomiast klienci otrzymują kolejne, zaczynając od 10.8.1.2 a kończąc zależnie od maski - tu ostatni to 10.8.1.254

- OpenVPN - przykładowa realizacja VPN
  - Różnice konfiguracji - serwer, cd.
    - push [opcja]
      - umożliwia „wrzucenie” określonych opcji konfiguracyjnych łączącemu się klientowi, np.: zmiana tablicy routingu, informacje o DNS’ach
      - opcje wstawiamy w cudzysłowach (”....”)
      - uwaga - jest po stronie klienckiej możliwość ustawienia opcją „pull-filter” zgody lub odrzucenia „wrzucanej” opcji klienckiej np.: odrzucenie wszelkich opcji zmiany trasy nie pasujących do wzorca „192.168.1."

```
--pull-filter accept "route 192.168.1."
--pull-filter ignore "route ”
```

- OpenVPN - przykładowa realizacja VPN
  - Różnice konfiguracji - klient
    - myvpn_client.conf to ustawienia:
    - remote [adres port]
      - adres - pod jakim znajdziemy serwer np.: myvpn.example.com lub 192.168.1.1
      - port - numer portu na jakim nasłuchuje (lub jest dostępny) serwer
      - w pliku konfiguracji klienta można podać wielokrotnie opcje remote tworząc listę serwerów VPN z jakimi klient może próbować się połączyć
        - nie zdolność do połączenia z jednym z wymienionych na liście serwerów, przerzuca klienta na następny serwer z podanej tak listy
        - przydatna jest także opcja „remote-random” wybierająca losowo serwer z listy

- OpenVPN - przykładowa realizacja VPN
  - Zmiana ścieżek routingu
    - Mechanizm niebezpieczny - serwer wymusza na klientach zmianę ich tras
      - konsekwencje: podczas gdy mamy zestawione połączenia (np.: poprzez klienta SSH) z jakimś serwerem, rozpoczęcie zestawiania połączenia z serwerem VPN który wymusza zmianę ścieżek może spowodować zerwanie zestawionego połączenia
    - Realizacja - dodanie ścieżki do pojedynczego węzła (tu o IP: 192.168.1.2)

```
push "route 192.168.1.2 255.255.255.255”
```

      - Dodawać można także całe podsieci:

```
push "route 192.168.1.0 255.255.255.0”
```

    - Przydatna opcja: `push "redirect-gateway def1”`
      - opcja wymusza na klientach zmianę wszelkich tras, tak aby cały ruch przechodził przez tunel
      - przy kierowaniu całego ruchu warto propagować klientom informacje o lokalizacji serwerów DNS dostępnych w sieci VPN (inaczej stracą dostęp do usługi DNS)
      - np.: push "dhcp-option DNS 10.8.1.1"

- OpenVPN - przykładowa realizacja VPN
  - Zmiana ścieżek routingu, cd.
    - Uwaga! powyższe operacje zmiany tras nie zadziałają poprawnie bez przekierowania ruchu między interfejsami sieciowymi serwera VPN
      - objawia się to nieco enigmatycznie: usługi oferowane przez serwer VPN będą poprzez tunel dostępne ale te oferowane na innych komputerach w sieci VPN już nie
    - Minimalna zmiana konfiguracji serwera - pomoże poinstruowanie jądra systemu gdzie działa serwer VPN o zmianie domyślnej polityki przekierowywania ruchu (niezbędne niemal zawsze, a wystarczające tylko w najprostszych przypadkach architektur sieciowych)

`sysctl net.ipv4.ip_forward=1` (lub: `echo 1 > /proc/sys/net/ipv4/ip_forward`)

`sysctl net.ipv6.conf.all.forwarding=1`

- OpenVPN - przykładowa realizacja VPN
  - Zmiana ścieżek routingu, cd.
    - Ustawienie reguł firewall z tworzeniem NAT - gdy zamiana polityk nie zadziałała
      - zakładamy: serwer VPN wpuszcza swoich klientów poprzez TUN0 (tu o IP: 10.8.1.0) a ETH0 zapewnia łączność z siecią urządzeń do których mają klienci mieć dostęp - klienci widzą tę sieć będąc za „NAT”

```
iptables -A FORWARD -i eth0 -o tun0 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -s 10.8.1.0/24 -o eth0 -j ACCEPT
iptables -t nat -A POSTROUTING -s 10.8.1.0/24 -o eth0 -j MASQUERADE
```

      - Powyższe ustawienia firewall’a muszą być aktywne po rozpoczęciu pracy serwera OpenVPN (np.: należy zachować odporność na restart maszyny czy brak interfejsów ETH0 i TUN0)

Konfiguracja OpenVPN

- OpenVPN - przykładowa realizacja VPN
  - Podobieństwa konfiguracji serwera i klienta
    - proto [protokół]
      - protokół - UDP lub TCP użyte do przenoszenia tunelowanego ruchu
        - proszę pamiętać o problemach wydajności i opóźnień związanych z realizacją TCP-over-TCP!
    - port [port]
      - port - określa tzw. wspólny mianownik łączący serwer i klientów, np.: typowo wstawia się wartość „1194”
      - gdy serwer OpenVPN dla klientów jest umiejscowiony „za routerem”, konieczne jest otwarcie odpowiedniego portu czy przekierowania w routerze

- OpenVPN - przykładowa realizacja VPN
  - Podobieństwa konfiguracji serwera i klienta, cd.
    - compress [alg]
      - alg - włącza i określa typ kompresji transferowanych informacji
      - alg może być: "lzo", "lz4" lub można pozostawić pole puste (zdecyduje automat)
      - LZ4 oferuje najlepszą wydajność przy najniższym wykorzystaniu procesora
      - dla zachowania wstecznej zgodności zalecane jest używanie „lzo" co jest równoważne z użyciem opcji „comp-lzo yes"
        - uwaga! obecnie użycie opcji „comp-lzo" jest nie zalecane
      - proszę mieć w pamięci, że kompresja używana w tunelowaniu ruchu może zaburzyć wyniki podawane przez narzędzia testujące szybkość sieci (np.: „iperf") - często preparują one do przesyłania wiadomości w stylu „ABC...Z" które kompresują się dość dobrze
    - keepalive [interwał] [timeout]
      - interwał - co ile przesyłać specjalistyczny pakiet ping
      - timeout - po jakim czasie braku odpowiedzi od strony przeciwnej połączenie ma być restartowane
      - wspiera utrzymanie połączenia nawet gdy nic nie jest przesyłane

- OpenVPN - przykładowa realizacja VPN
  - Podobieństwa konfiguracji serwera i klienta, cd.
    - dev [interfejs]
      - interfejs - określenie która z wirtualnych kart sieciowych ma być użyta
      - podajemy typ: tun czy tap oraz numer interfejsu np.: tun0
        - użycie „tun” bez podania numeru - oznacza, że system ma dynamicznie użyć któregoś z interesów TUN
      - interfejs TUN przenosi wyłącznie ruch IPv4 lub IPv6 i pracuje w warstwie 3 OSI
      - interfejs TAP przenosi ruch Ethernet 802.3 i pracuje w warstwie 2 OSI
        - ten interfejs wydaje się bardziej uniwersalny, ma jednak wadę w postaci bardziej zawiłej konfiguracji oraz taki model komunikacji sprawia, że przenoszony jest cały ruchu poprzez tunel w tym np.: pakiety rozgłoszeniowe warstwy 2
        - uwaga! stosowanie interfejsu TAP i praca w warstwie 2 może wymagać odpowiednich ustawień serwera DHCP w lokalnej sieci LAN
      - pod Linux utworzenie urządzenia TUN wykonuje się poprzez

        ```
        mknod /dev/net/tun c 10 200 && modprobe tun
        ```

      - dla Windows pakiet instalacyjny OpenVPN udostępnia specjalne narzędzie ("C:\Program Files\TAP-Windows\bin\addtap.bat")

- OpenVPN - przykładowa realizacja VPN
  - Podobieństwa konfiguracji serwera i klienta, cd.
    - verb [poziom]
      - poziom - ustala poziom raportowanych informacji
        - domyślnie 1, każdy poziom pokazuje oprócz swoich raportów także raporty poziomów niższych
        - poziom 3 jest zalecany aby mieć pełną wiedzę co dzieje się z serwerem
        - poziom 0 nie raportuje nic poza błędami krytycznymi (przydatne dla twórców kodu źródłowego serwera)
      - uwaga! podczas początkowych zmagań zaleca się uruchamianie serwera OpenVPN z linii poleceń i obserwację generowanych w konsoli logów
        - niektóre konfiguracje domyślnie w np.: Debianie początkującym mogą sprawiać wiele problemów z odnalezieniem właściwych logów
    - user [nazwa_użytkownika] i group [nazwa_grupy]
      - sprawia że OpenVPN uruchamia się z prawami wybranego użytkownika i w wybranej grupy - należy jednak zwrócić uwagę na prawa dostępu do urządzeń i plików(!)
      - przydatne aby ograniczyć przywileje i zabezpieczyć się przed przejęciem serwera przez osoby nie upoważnione

Konfiguracja OpenVPN

- OpenVPN - przykładowa realizacja VPN
  - Konfiguracja ze statycznym kluczem
    - Generacja klucza - wystarczy utworzyć jeden klucz

```
openvpn --genkey --secret static_key_for_may_server.key
```

    - Konfiguracja na serwerze - przykład

```
dev tun
ifconfig 10.8.1.1 10.8.1.2
secret static_key_for_may_server.key
```

    - Konfiguracja klienta - przykład

```
remote myvpn.example.com
dev tun
ifconfig 10.8.1.2 10.8.1.1
secret static_key_for_may_server.key
```

      - powyżej myvpn.example.com to adres serwera

```
static_key_for_may_server.key:
-----BEGIN OpenVPN Static key  -----
e5e4d6af392a9d53
171ecc237a8f996a
97743d146161405e
c724d5913cff0a0c
30a48252dfbeceb6
e2e7bd4a8357df78
4609fe35bbe99c32
bdf974952ade8fb9
71c224aaf4f256ba
eeda72ed4822ff98
fd66da2efa9bf8c5
e70996323e0f96a9
c94c9d9afb17637b
283da25cc99b37bf
6f7e15b38aedc3e8
e6adb4ccca5c5463
-----END OpenVPN Static key  -----
```

- OpenVPN - przykładowa realizacja VPN
  - Konfiguracja z statycznym kluczem, cd.
    - Zalety
      - Prosta i szybka w tworzeniu konfiguracja
      - Brak problemów z generowaniem właściwych kluczy i zarządzaniem PKI („Public Key Infrastructure”)
    - Wady
      - Podejścia nie można skalować - tylko jeden klient komunikuje się z tylko jednym serwerem VPN
      - Utrata czy ujawnienie klucza powoduje konieczność jego wymiany
      - Klucz musi być przechowywany jawnym tekstem w obu elementach systemu
      - NAJWAŻNIEJSZE: klucz trzeba wymieniać poufnym kanałem!!!

Zadanie:
Wykorzystując metodę pracy ze statycznym kluczem skonfigurować OpenVPN dla potrzeb swojej grupy zajęć PBL5 (serwerem niech będzie VM a klientem laptop)

Konfiguracja OpenVPN

- OpenVPN - przykładowa realizacja VPN
  - Konfiguracja oparta o mechanizmy RSA - rozwiązuje wady podejścia ze statycznym kluczem
  - Zamiany w konfiguracji obu stron
    - `ca ca.crt` - certyfikat wspólnej jednostki autoryzacji (CA)
    - `cipher AES-256-CBC` - szyfrowanie używane dla właściwej transmisji danych
  - Jakie OpenVPN wspiera metody szyfrowania?

    `openvpn --show-ciphers`

```
DES-CFB 64 bit default key (fixed) (TLS client/server mode)
DES-CBC 64 bit default key (fixed)
...
AES-128-CBC 128 bit default key (fixed)
AES-128-OFB 128 bit default key (fixed) (TLS client/server mode)
AES-128-CFB 128 bit default key (fixed) (TLS client/server mode)
AES-192-CBC 192 bit default key (fixed)
AES-192-OFB 192 bit default key (fixed) (TLS client/server mode)
AES-192-CFB 192 bit default key (fixed) (TLS client/server mode)
AES-256-CBC 256 bit default key (fixed)
AES-256-OFB 256 bit default key (fixed) (TLS client/server mode)
AES-256-CFB 256 bit default key (fixed) (TLS client/server mode)
AES-128-CFB1 128 bit default key (fixed) (TLS client/server mode)
AES-192-CFB1 192 bit default key (fixed) (TLS client/server mode)
AES-256-CFB1 256 bit default key (fixed) (TLS client/server mode)
AES-128-CFB8 128 bit default key (fixed) (TLS client/server mode)
AES-192-CFB8 192 bit default key (fixed) (TLS client/server mode)
AES-256-CFB8 256 bit default key (fixed) (TLS client/server mode)
```

```
...
CAMELLIA-128-CFB 128 bit default key (fixed) (TLS client/server mode)
CAMELLIA-192-CFB 192 bit default key (fixed) (TLS client/server mode)
CAMELLIA-256-CFB 256 bit default key (fixed) (TLS client/server mode)
...
CAMELLIA-128-CFB8 128 bit default key (fixed) (TLS client/server mode)
CAMELLIA-192-CFB8 192 bit default key (fixed) (TLS client/server mode)
CAMELLIA-256-CFB8 256 bit default key (fixed) (TLS client/server mode)
...
SEED-CBC 128 bit default key (fixed)
SEED-OFB 128 bit default key (fixed) (TLS client/server mode)
SEED-CFB 128 bit default key (fixed) (TLS client/server mode)
```

- OpenVPN - przykładowa realizacja VPN
  - Konfiguracja oparta o mechanizmy RSA - konfiguracja serwera
    - dh [DH_file]
      - DH_file - nazwa pliku z parametrami Diffie Hellman np.: dh2048.pem
      - niezbędne dla wymienia kluczy
      - generowane poprzez: `openssl dhparam -out dh2048.pem 2048`
      - dh2048.pem może być publiczny
    - cert [cert_file]
      - cert_file - certyfikat serwera podpisany przez CA np.: myserver.crt
    - key [key_file]
      - key_file - klucz prywatny dla serwera np.: myserver.key
      - pamiętaj że to poufny plik, aby go chronić wykonaj: `chmod go-rwx *.key`
    - username-as-common-name
      - opcja sprawdzania przez serwer czy nazwa użytkownika podana przez klienta jest zgodna z certyfikatem tego użytkownika (wskazanym przez opcję cert)

Konfiguracja OpenVPN

- OpenVPN - przykładowa realizacja VPN
  - Konfiguracja oparta o mechanizmy RSA - konfiguracja u klienta
    - cert [cert_kliencki]
      - cert_kliencki - certyfikat klienta podpisany przez wspólne CA, np.: nazwa_klienta_cn.crt
    - key [klucz_kliencki]
      - klucz_kliencki - prywatny klucz klienta np.: nazwa_klienta_cn.key
    - remote-cert-tls server
      - wskazanie jakiej metody kryptograficznej należy używać w łączności z serwerem
    - Pliki z certyfikatami i kluczami można połączyć z plikiem konfiguracyjnym tworząc jeden łatwy w przenoszeniu i bardziej bezpieczny plik OVPN
      - format treści związany z certyfikatami i kluczami:

```
<ca> ... tresc pliku ca.crt ... </ca>

<cert> ... tresc pliku nazwa_klienta_cn.crt ... </cert>

<key> ... tresc pliku nazwa_klienta_cn.key ... </key>
```

Uwaga opis dla starej wersji Easy-RSA

- OpenVPN - przykładowa realizacja VPN
  - Jak wygenerować pliki: ca.crt, dh2048.pem, myserver.{crt|key}, nazwa_klienta_cn.{crt|key}?
    - Pomocna paczka Easy-RSA, zakłada że sami dla siebie będziemy CA!
      - procedurę generacji certyfikatów i kluczy można wykonać na niemal dowolnym komputerze - warunek „wysoka entropia” (czyli maszyny wirtualizowane słabo spełniają ten warunek)
    - Tworzymy/modyfikujemy plik „vars” o przykładowej treści (zawiera stałe niezbędne dla generacji kluczy):

```
export CA_EXPIRE=3650     - liczba dni ważności certyfikatu „common authority”
export KEY_EXPIRE=365     - liczba dni ważności klucza
export KEY_COUNTRY="PL”   - kraj generowanego klucza
export KEY_PROVINCE="Mazovia” - obszar/województwo/...
export KEY_CITY="Warsaw”  - miasto
export KEY_ORG="ITPW”     - organizacja dla której będzie tworzony klucz
export KEY_EMAIL=”....”   - kontakt
export KEY_OU=”...”       - nazwa jednostki organizacyjnej
export KEY_NAME=”myserver” - nazwa systemu
export KEY_CN=”myserver”  - wspólna nazwa (common name)
```

Uwaga opis dla starej wersji Easy-RSA

- OpenVPN - przykładowa realizacja VPN
  - Jak wygenerować pliki: ca.crt, dh2048.pem, myserver.{crt|key}, nazwa_klienta_cn.{crt|key}? cd.
    - Wykonujemy szereg czynności - budujemy certyfikat CA („Common authority”)

```
source ./vars     - ustawienie zmiennych środowiskowych
./clean-all       - wyczyszczenie „śmieci” z poprzednich akcji
./build-ca        - właściwa budowa CA
```

      - tu otrzymamy m.in.: ca.crt, ca.key (bardzo poufny i cenny!), index.txt, serial
    - Tworzymy certyfikat i klucz dla serwera (koniecznie w tym samym oknie terminala lub ponawiamy wywołanie „source ./vars”)

```
./build-key-server myserver
```

    - Tworzymy certyfikat i klucz dla klientów (można wykonać to polecenie wielokrotnie dla każdego klienta któremu chcemy dać dostęp, pamiętaj o „source ./vars”):

```
./build-key nazwa_klienta_cn
```

    - Powyższa nazwa „nazwa_klienta_cn” będzie wpisana w certyfikat i dla każdego klienta musi być unikatowa - o ile stosujemy opcję „username-as-common-name” (wysoce zalecana!)

Uwaga opis dla nowej wersji Easy-RSA

- OpenVPN - przykładowa realizacja VPN
  - Jak wygenerować pliki: ca.crt, dh2048.pem, myserver.{crt|key}, nazwa_klienta_cn.{crt|key}?
    - Pomocna paczka Easy-RSA, zakłada że sami dla siebie będziemy CA!
      - procedurę generacji certyfikatów i kluczy można wykonać na niemal dowolnym komputerze - warunek „wysoka entropia” (czyli maszyny wirtualizowane słabo spełniają ten warunek)
      - instalacja paczki `apt-get  install easy-rsa`
    - Kopiujemy treść do roboczego miejsca: `cp -a /usr/share/easy-rsa easy-rsa`
    - Tworzymy/modyfikujemy plik „vars” o przykładowej treści (zawiera stałe dla generacji kluczy):

```
export EASYRSA_DN="org"
export EASYRSA_REQ_COUNTRY="PL"
export EASYRSA_REQ_PROVINCE="MAZOVIA"
export EASYRSA_REQ_CITY="WARSAW"
export EASYRSA_REQ_ORG="ITPW"
export EASYRSA_REQ_EMAIL="pbl5@pl.pl"
export EASYRSA_KEY_SIZE=2048
export EASYRSA_REQ_CN="capbl5"
export EASYRSA_REQ_OU="IT"
export EASYRSA_CERT_EXPIRE=3650
```

Uwaga opis dla nowej wersji Easy-RSA

- OpenVPN - przykładowa realizacja VPN
  - Jak wygenerować pliki: ca.crt, dh2048.pem, myserver.{crt|key}, nazwa_klienta_cn.{crt|key}? cd.
    - Kopiujemy wzorcowy zestaw plików

```
mkdir test && cp -a /usr/share/easy-rsa test && cd test
```

    - Inicjujemy prace z narzędziem

```
source ./vars ; ./easyrsa init-pki ; openssl rand -out pki/.rnd -hex 256
```

    - Wykonujemy szereg czynności -budujemy certyfikat CA („Common authority”)

`./easyrsa build-ca nopass` - właściwa budowa CA („bez hasłowe”)

    - Powstaną m.in. pliki: ./pki/ca.crt, ./pki/private/ca.key (bardzo poufny i cenny!)
    - Tworzymy certyfikat i klucz dla serwera (koniecznie w tym samym oknie terminala lub ponawiamy wywołanie „source ./vars”)

`./easyrsa gen-req myserver nopass` - tworzone jest żądanie podpisania

`./easyrsa sign-req server myserver` - podpisanie przez CA

    - Powstaną pliki: ./pki/private/server_openvpn.key i ./pki/issued/server_openvpn.crt

Uwaga opis dla nowej wersji Easy-RSA

- OpenVPN - przykładowa realizacja VPN
  - Jak wygenerować pliki: ca.crt, dh2048.pem, myserver.{crt|key}, nazwa_klienta_cn.{crt|key}? cd.
    - Generacja pliku DH

```
./easyrsa gen-dh
```

    - Tworzymy certyfikaty dla klientów (koniecznie w tym samym oknie terminala lub ponawiamy wywołanie „source ./vars”)

`./easyrsa build-client-full` nazwa_klienta_cn `nopass`

    - Powstaną pliki: ./pki/private/nazwa_klienta_cn.key i ./pki/issued/nazwa_klienta_cn.crt
      - Powyższa nazwa `„nazwa_klienta_cn”` będzie wpisana w certyfikat i dla każdego klienta musi być unikatowa - o ile stosujemy opcję „username-as-common-name” (wysoce zalecana!)

- OpenVPN - przykładowa realizacja VPN
  - Uwierzytelnienie dodatkowe
    - Użyteczne gdy nie ufamy całkowicie certyfikatom i kluczom (sens?) lub gdy chcemy inaczej kontrolować dostęp do systemu (np.: dostęp czasowy)
    - auth-user-pass-verify [polecenie] [metoda]
      - polecenie - dowolny program/skrypt wykonywany dla ustalenia czy przekazana nazwa użytkownika i jego hasło są poprawne
      - metoda - wybór metody przekazania nazwy i hasła użytkownika do uwierzytelnienia (do wyboru: „via-env” lub „via-file”)
      - uwaga! wymaga ustawienia - uznawanej za mniej bezpiecznej - opcji ‘script-security 2’
    - script-security [poziom]
      - poziom - ustala stopień ochrony serwera (domyślnie 1, poziom 3 - nie bezpieczny)
    - client-cert-not-required
      - włączenie uwierzytelnienia wyłącznie poprzez nazwę i hasło
      - certyfikaty i klucze klienckie są zbędne(!)
      - opcja ta jest nie zalecana

- OpenVPN - przykładowa realizacja VPN
  - Jak odwołać certyfikaty i klucze przekazane klientom
    - Kiedy istnieje taka potrzebna?
      - gdy określonego klienta chcemy pozbawić dostępu do naszego serwera
      - gdy określony klucz zostanie wykradziony a nie chcemy zmieniać kluczy i certyfikatów serwerowych jak i klienckich
    - Odwołanie oznacza unieważnienie podpisanego wcześniej certyfikatu
    - Proces odwoływania - na serwerze gdzie był podpisywany certyfikat wywołujemy

      `source ./vars` lub prościej `. ./vars`

      `./revoke-full nazwa_klienta_cn`
    - Po odwołaniu certyfikatu powstaje plik `crl.pem` - którego zawartość trzeba umieścić w miejscu określonym przez konfiguracje serwera OpenVPN

- OpenVPN - przykładowa realizacja VPN
  - Jak odwołać certyfikaty i klucze przekazane klientom, cd.
    - Aby odwoływanie certyfikatów przez serwer OpenVPN funkcjonowało w konfiguracji należy dodać opcję
      - crl-verify [PEM]
      - PEM - plik np.: crl.pem zawiera treść wygenerowaną po wykonaniu „`revoke-full`”

      lub
      - crl-verify [path_to_dir] dir
      - path_to_dir - wskazanie katalogu w którym znajdują się pliki o nazwach będących numerami seryjnymi certyfikatów do wycofania - pliki mogą być puste
    - Uwaga wyciek certyfikatów głównych (CA) jest przesłanką do bezwzględnej wymiany wszystkich certyfikatów!!!

- OpenVPN - przykładowa realizacja VPN
  - Jak odwołać certyfikaty i klucze przekazane klientom, cd.
    - Gdzie znaleźć numer seryjny certyfikatu?
      - typowo w katalogu keys/ podczas tworzenia certyfikatów tworzone są także pliki *.pem/*.crt zawierające odpowiednie pola

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 241 (0xf1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=PL, ST=Mazovia, L=Warsaw,O=ITPW,OU=Zsut, CN=.../name=.../emailAddress=...
        Validity
            Not Before: Nov 12 01:18:04 2021 GMT
            Not After : Aug 19 01:18:04 2023 GMT
        Subject: C=PL, ST=Mazovia, L=Warsaw, O=Zsut, CN=.../emailAddress=...
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
...
```

Podgląd certyfikatu z linii poleceń

```
openssl x509 -in nazwa_klienta_cn.crt -text
```

Konfiguracja OpenVPN

- Utrzymanie OpenVPN
  - Jak uruchamiać automatycznie OpenVPN - strona serwerowa (dla Linux)
    - Ręczne uruchamianie - choć możliwe to nie praktyczne
    - Rozwiązanie poprzez wpis do „/etc/rc.local” - dystrybucje uciekają od tego podejścia
    - Własne skrypty uruchamiane przez Crontab - opcja @reboot nie jest wspierana przez wszystkie dystrybucje (Raspbian wspiera, Debian nie wspiera)
    - SystemD - następca InitD - bardziej wszechstronny i bardziej zawiły dla początkujących
  - Jak uruchomić z linii poleceń OpenVPN - strona kliencka
    - Metoda może być także zastosowana po stronie serwerowej

```
/usr/sbin/openvpn --config nazwa_klienta_cn.ovpn --auth-user-pass haslo.txt
```

    - Plik `haslo.txt` musi zawierać dwie linie: nazwa klienta i jego hasło klienta
      - jeżeli ustawiono w konfiguracji serwera „`username-as-common-name`” to nazwa klienta musi być zgodna z nazwą zapisaną w certyfikacie
    - Pliki `nazwa_klienta_cn.ovpn` i `haslo.txt` muszą mieć dobrze dobrane prawa dostępu - wyciek wbudowanego klucza czy samego hasła jest niebezpieczny!

- Utrzymanie OpenVPN
  - Gdy serwer już działa i automatyzacja procesów odciąża nas z doglądania systemu - może nastąpić katastrofa: włamanie, serwer może przestać obsługiwać klientów, ... - pomocne staną się logi
  - Konfigurowanie „logów”?
    - log [plik]
      - plik - nazwę pliku do którego będą wpisywane informacje przez OpenVPN
        - uwaga! plik jeżeli istnieje w czasie uruchamiania serwera będzie ucięty
    - log-append [plik]
      - jak wyżej ale nie ucina danych z poprzednich uruchomień serwera
    - status [plik] [czas]
      - zlecenie wpisywania statusu serwera do pliku co zadany czas
  - Uwaga niektóre dystrybucje w tym zakresie mają nieco nieprzewidywalne domyślne konfiguracje

- Utrzymanie OpenVPN, cd.
  - Ponad standardowe logi – możliwe jest wywołanie skryptu gdy nastąpi podłączenie lub odłączenie się klienta
    - client-connect [nazwa_skryptu_up]
      - nazwa_skryptu_up - nazwa skryptu wywoływanego gdy użytkownik zostanie połączony
      - informacje kto się loguje i z jakiego adresu są dostępne dla skryptu w jego zmiennych środowiskowych
    - client-disconnect [nazwa_skryptu_down]
      - jak wyżej tyle, że wywołanie nastąpi gdy klient się rozłączy
      - akcja nie musi być wykonana natychmiast!

Zadanie:
Wykorzystując RSA czyli PKI (Publik Key Infrastrukturę) skonfigurować OpenVPN, tak aby umożliwić niezależne podłączanie się wszystkich osób pracujących w grupie PBL5 (trzy/cztery osoby). Zapewnij właściwe logowanie akcji podejmowanych przez serwer OpenVPN.

O ile potrafisz zapewnij aby system odrzucił wielokrotne połączenia klientów stosujących te same dane uwierzytelniające

- Problem dostępu do brokera MQTT z zastosowaniem kryptografii
  - Procedura tworzenia kluczy jest podobna do użytej dla OpenVPN, choć realizowana „na piechotę", o kolejnych krokach:
    - Znajdź plik openssl.cnf i skopiuj go do swojego katalogu, za pomocą np.:

```
cp `find / 2>/dev/null | grep openssl.cnf | head -n 1` .
```

    - Wykreuj plik .RND

```
openssl rand -out .rnd -hex 10
```

    - Wykreuj certyfikaty CA

```
openssl req -batch -days 3650 -nodes -new -x509 -keyout ca_mqtt.key -out ca_mqtt.crt \
-config openssl.cnf -subj "/C=PL/ST=PL/L=WARSAW/O=PW/CN=ca1mqtt"
```

    - Klucz i żądanie podpisania certyfikatu dla Brokera o IP 10.0.126.196

```
openssl genrsa -out 10.0.126.196.key 2048
openssl req -batch -out 10.0.126.196.csr -key 10.0.126.196.key -new -config openssl.cnf \
-subj "/C=PL/ST=PL/L=WARSAW/O=PW/CN=10.0.126.196"
```

    - Dlaczego posługujemy się numer IP – broker i klient bazują na tym jako tzw. CN

- Problem dostępu do brokera MQTT z zastosowaniem kryptografii
  - Procedura tworzenia kluczy jest podobna do użytej dla OpenVPN, choć realizowana „na piechotę", o kolejnych krokach, cd.
    - Aby połączenia były możliwe, należy w odpowiedni sposób przekazać alternatywne nazwy DNS i IP tego brokera – najlepiej przez plik openssl-ext.cnf o przykładowej treści:

```
[ req_ext ]
subjectAltName = @alt_names
[alt_names]
DNS.1 = mqtt.zsut.tele.pw.edu.pl
DNS.2 = raspberrypi12.local
IP.1 = 194.29.169.4
IP.2 = 10.0.1.220
```

    - A następnie jako CA podpisać certyfikat:

```
openssl x509 -req -in 10.0.126.196.csr -CA ca_mqtt.crt -CAkey ca_mqtt.key \
-CAcreateserial -out 10.0.126.196.crt -days 12 -extfile openssl-ext.cnf -extensions req_ext
```

    - Ten etap kończymy z kluczowymi plikami: ca_mqtt.{key|crt}, 10.0.126.196.{key|crt}

- Problem dostępu do brokera MQTT z zastosowaniem kryptografii
  - Procedura tworzenia kluczy jest podobna do użytej dla OpenVPN, choć realizowana „na piechotę", o kolejnych krokach, cd.
    - Aby zbadać zawartość wygenerowane certyfikatu

```
openssl x509 -in 10.0.126.196.crt -text
```

    - Tu szczególnie należy zwrócić uwagę na pola tzw. aliasów
    - W następnym kroku tworzy PKI dla każdego z klientów niezależnie

```
openssl genrsa -out user1.key 2048
openssl req -batch -nodes -out user1.csr -key user1.key -new -config openssl.cnf \
-subj "/C=PL/ST=PL/L=WARSAW/O=PW/CN=user1"
openssl x509 -req -in user1.csr -CA ca_mqtt.crt -CAkey ca_mqtt.key -CAcreateserial \
-out user1.crt -days 3600
```

    - Tu wygenerowaliśmy PKI dla klienta o nazwie user1 – proszę pamiętać, że pozostali powinni mieć inne nazwy

- Problem dostępu do brokera MQTT z zastosowaniem kryptografii
  - Konfiguracja brokera
    - Dodajemy wpis

```
listener 8883
cafile ./ca_mqtt.crt
keyfile ./10.0.126.196.key
certfile ./10.0.126.196.crt
require_certificate true
use_identity_as_username true
```

Broker będzie nasłuchiwał na porcie 8883
Tu proszę zauważyć że brakuje pliku ca_mqtt.key – on nie uczestniczy w pracy brokera i zalecane jest aby był przechowywany poza tym urządzeniem
Proszę pamiętać o wykreowaniu pliku `my_passwd` dla użytkownika `user1` z hasłem `user1password`

i restartujemy broker

- Testowanie z użyciem mosquitto_sub (okno I):

```
mosquitto_sub -h 10.0.126.196 -p 8883 --cafile ca_mqtt.crt --cert user1.crt \
--key user1.key -u user1 -P user1password -t "#" -v
```

- Testowa publikacja (okno II)

```
mosquitto_pub -h 10.0.126.196 -p 8883 --cafile ca_mqtt.crt --cert user1.crt \
--key user1.key -u user1 -P user1password -t "test" –m "testowa wiadomosc"
```

- Problem dostępu do brokera MQTT z zastosowaniem kryptografii
  - Aplikacje użytkowe w języku Python

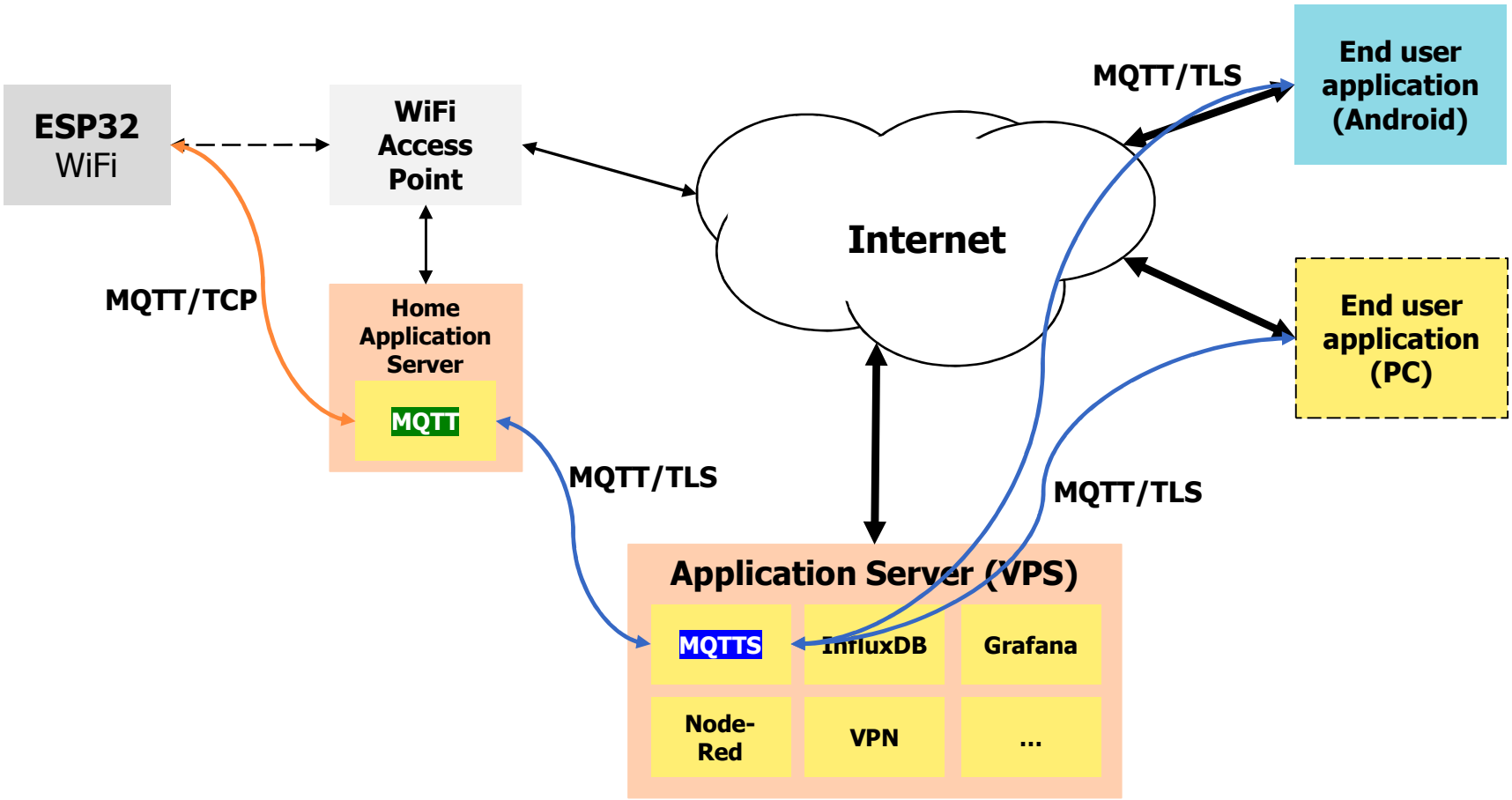
```
import time
import paho.mqtt.client as mqtt
import ssl
def load_ssl_context():
   context=ssl.create_default_context(cafile="ca_mqtt.crt")
   context=ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
   context.load_cert_chain("user1.crt", "user1.key")
   return context
ssl_ctx=load_ssl_context()
def on_message(mqttc, obj, msg):
   print(msg.topic+" "+str(msg.qos)+" "+str(msg.payload))
mqttc = mqtt.Client()
mqttc.on_message = on_message
mqttc.tls_set_context(ssl_ctx)
mqttc.username_pw_set("user1", "user1password")
mqttc.connect("10.0.126.196", 8883, 60)
mqttc.subscribe("#", 0)
mqttc.loop_start()
while True:
   time.sleep(1)
```

Fragmenty odpowiedzialne za połączenie z wykorzystaniem szyfrowania

Brak w systemie biblioteki paho-mqtt:
su –c "apt update –y ; apt update install python3-pip –y"
su –c "pip3 install paho-mqtt"

Zadanie:
Dodać do systemu IoT w brokerze MQTT szyfrowanie połączeń i akceptowanie wyłącznie połączeń uwierzytelnionych, utworzyć także klienta publikującego stan przestrzeni dyskowej i pod innym tematem wiadomości - stan interfejsów sieciowych, utworzyć także dwóch klientów subskrybujących po jednym temacie z wiadomości wysyłanych przez klienta publikującego.
Niech każdy z tych klientów ma własne uwierzytelnienia i pliki PKI.

- Problem kryptografii i małych urządzeń
  - Mechanizm „bridge" w Mosquitto jest rozwiązaniem problemu zbyt małych zasobów na urządzeniu IoT
    - Architektura



- Problem kryptografii i małych urządzeń
  - Mechanizm „bridge" w Mosquitto jest rozwiązaniem problemu zbyt małych zasobów na urządzeniu IoT, cd.
    - Konfiguracja – składnia

```
connection <name>                    #nazwa połączenia
```

    - Lista brokerów do jakich chcemy się podłączyć tym brokerem (jeżeli opcja round_robin jest ustawiona na false, to pierwszy adres jest traktowany jako adres głównego brokera i nawiązywane jest połączenie tylko z nim, jeżeli jako true – wtedy wszystkie połączenia są jednakowo traktowane)

```
address <host>[:<port>] [<host>[:<port>]]
```

    - Specyfikacja reguł przekazywania pakietów dla wszystkich (#) lub poszczególnych tematów wiadomości, prefiksy wprowadzają mapowanie tematów wiadomości

```
topic <topic> [[[out|in|both] qos-level] local-prefix remote-prefix]
```

- Problem kryptografii i małych urządzeń
  - Mechanizm „bridge” w Mosquitto jest rozwiązaniem problemu zbyt małych zasobów na urządzeniu IoT, cd.
    - Przypadki użycia

#Przekazuj wszystko bez mapowania

```
connection bridge_01
address 10.0.1.91:1883
topic # out 0
topic # in 0
```

#Przekazuj wszystko z mapowaniem

```
connection bridge_02
address 10.0.1.91:1883
topic # out 0 b2/ ""
topic # in 0 b1/ ""
```

#Co wyśle B1 to B2 doda "b2/..."

#Przekazuj z mapowaniem i uwierzytelnieniem

```
connection bridge_03
address 10.0.1.91:1883
username login_do_10_0_1_91
password haslo_do_10_0_1_91
topic # out 0 b2/ ""
topic # in 0 b1/ ""
topic b1/t1/# in 0 b1/t1/ b2/t2/ #przemapowane poddrzewa
```

Zadanie:
Zbuduj system z dwoma brokerami – jeden ma wpuszczać tylko ruch MQTTS (8883) a drugi tylko MQTT (1883) – dla testów przygotuj czterech klientów: publikującego i subskrybującego. Niech dwaj klienci łączą się z pierwszym brokerem i identyczną parę dwóch klientów którzy łączyć się będą z drugim brokerem. Tematy wiadomości używane przez klientów i mechanizm bridge dobierz tak aby dawały informację o drodze jaką pokonała wiadomość pomiędzy brokerami.
Dane do zbudowania treści wiadomości klientów publikujących niech pochodzą z narzędzi raportujących stan systemów operacyjnych maszyn na jakich oni pracują (liczba uruchomionych procesów, wielkość dostępnej pamięci, …).

Dziękuje za uwagę