

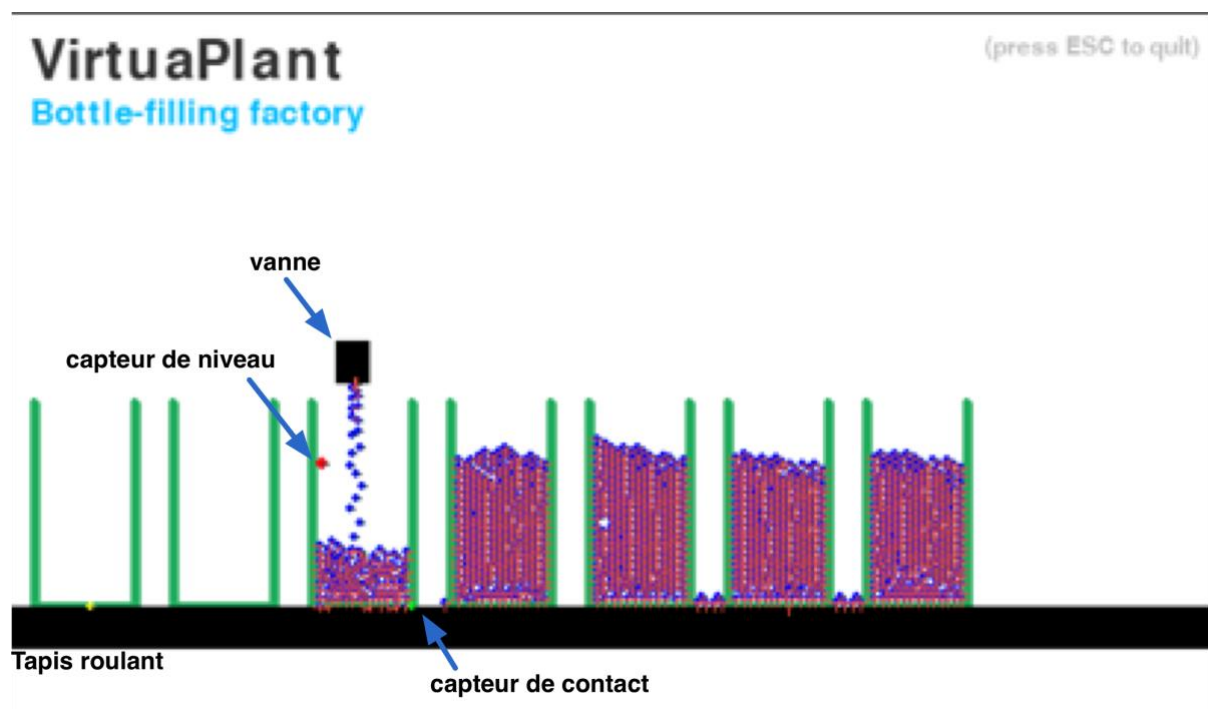
Travaux pratiques : Cyber-sécurité des systèmes industriels (8H)

Les travaux pratiques sont découpés en une série des exercices pour étudier la cyber-sécurité d'un système industriel de mise en bouteille. Dans une première partie, vous allez mettre en œuvre cette application industrielle par simulation. Dans la deuxième partie, vous allez programmer la partie commande du système sur une automate programmable virtuel. Dans la troisième partie, vous allez analyser les échanges réseaux du système et mener des attaques sur le système. Dans la dernière partie, vous allez sécuriser le système pour éviter ces attaques.

Pour la partie organisationnelle, vous travaillerez en trinôme ou binôme.

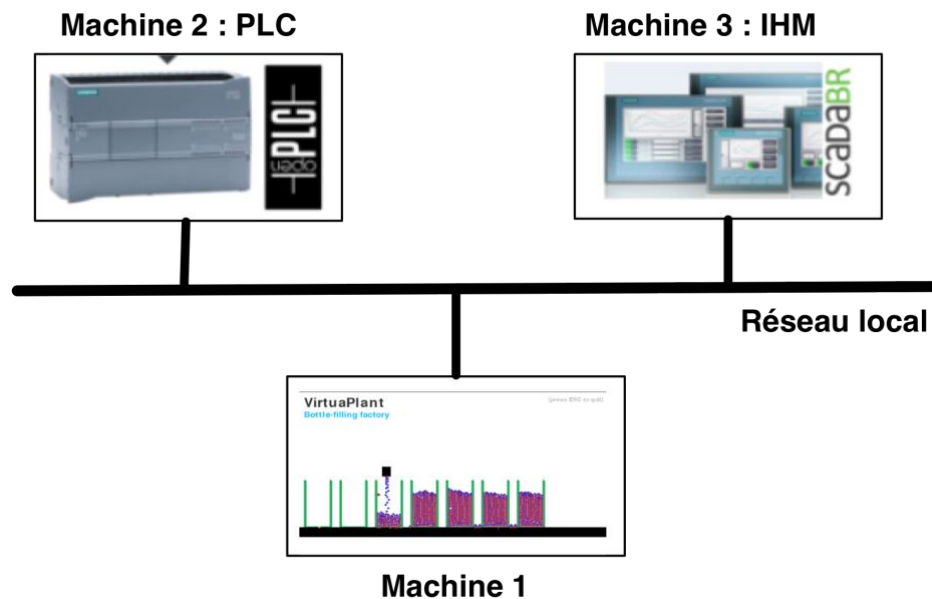
Partie 1 : Système de contrôle commande d'une chaîne de mise en bouteille (8 points)

Notre système industriel à étudier est une chaîne de mise en bouteille¹. Des bouteilles avancent sur un tapis roulant, commandé par un seul moteur. Un capteur de contact détecte lorsqu'elles sont en position, arrête le tapis roulant et actionne une vanne faisant couler le liquide. Un autre capteur de niveau permet de détecter lorsque la bouteille est pleine. La vanne de remplissage se referme, et le tapis roulant redémarre.



Notre système sera plutôt simulé avec un programme Python fourni. Pour commander le système, nous utilisons un PLC logiciel (OpenPLC) qui communiquera avec le système en utilisant le protocole ModBus TCP. La partie IHM est à programmer avec l'outil ScadaBR. L'architecture globale de notre installation industrielle est présentée dans la figure ci-dessous.

¹ <https://electrical-engineering-portal.com/plc-implementation-of-the-bottle-filling-application>



Exercice 1 : installation des machines virtuelles

Dans cette série de TPs, notre installation est construite grâce à un ensemble de machines virtuelles docker. Chaque machine virtuelle représente un composant de l'installation : PLC, IHM et système industrielle (*factory*). Pour construire cette installation vous allez récupérer depuis Arche, l'archive *scripts.zip* et l'extraire dans un répertoire.

Dans une console Linux, accédez à ce répertoire et lancer la commande suivante :

```
sudo apt-get install docker.io
sudo su
source buildsystem.sh
source runsystem.sh
```

L'adresse IP du PLC est 192.168.1.2, l'adresse IP de l'IHM est 192.168.1.3 et l'adresse IP de procédé physique (*factory*) est 192.168.1.4

Exercice 2 : Programmation du PLC

Dans cet exercice, l'objectif est de programmer le PLC pour commander notre système industriel. Nous utilisons un PLC logiciel qui est OpenPLC disponible sous ce lien (<https://www.openplcproject.com/getting-started/>). Nous avons besoin du OpenPLC runtime pour simuler un PLC et OpenPLC Editor pour créer des programmes Ladder.

Dans une première étape, nous installons sur la machine 2, ces deux outils.

Étape 1 : Installation de OpenPLC Editor

OpenPLC Editor est un outil pour créer des programmes en langage LADDER² (ou autres) à exécuter par OpenPLC runtime.

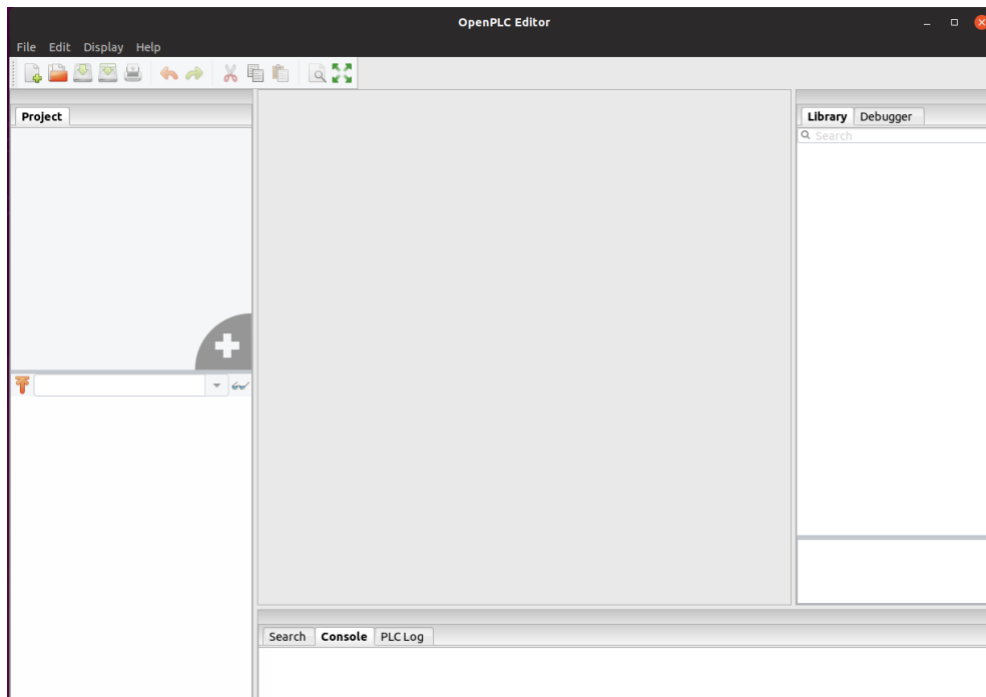
Pour installer cet outil, taper cette séquence des commandes dans un terminal Linux :

```
sudo apt-get install git
git clone https://github.com/thiagoralves/OpenPLC_Editor
cd OpenPLC_Editor
./install.sh
```

² https://en.wikipedia.org/wiki/Ladder_logic

Ensuite, dans le répertoire OpenPLC_Editor, taper la commande suivante pour lancer l'éditeur :

`./openplc_editor.sh`



Pour se familiariser avec l'outil, vous pouvez suivre le tutoriel disponible ici :

<https://openplcproject.com/docs/3-2-creating-your-first-project-on-openplc-editor/>

pour créer un premier programme qui commande une LED avec 2 boutons poussoirs.

Étape 2 : créer le programme Ladder pour commander notre système

Dans OpenPLC Editor créer un projet pour coder en LADDER la logique de commande de notre système de mise en bouteille.

Le système comporte les composants suivants :



- un bouton de démarrage qu'on lui associe la variable START_BUTTON
- un capteur de niveau qu'on lui associe la variable LEVEL_SENSOR
- un capteur de contact qu'on lui associe la variable CONTACT_SENSOR
- un moteur du tapis roulant qu'on lui associe la variable MOTOR
- une vanne de remplissage qu'on lui associe la variable NOZZLE

Une variable auxiliaire nommée RUN est utilisée pour sauvegarder l'état du démarrage de système. La logique de commande du système est la suivante :

- START_BUTTON démarre ou arrête le système. Vous pouvez utiliser RUN pour sauvegarder cet état
- Si CONTACT_SENSOR est désactivé ou LEVEL_SESNOR est activé alors MOTOR démarre et NOZZLE s'arrête
- Si CONTACT_SENSOR est activé et LEVEL_SENSOR est désactivé alors NOZZLE démarre et MOTOR s'arrête

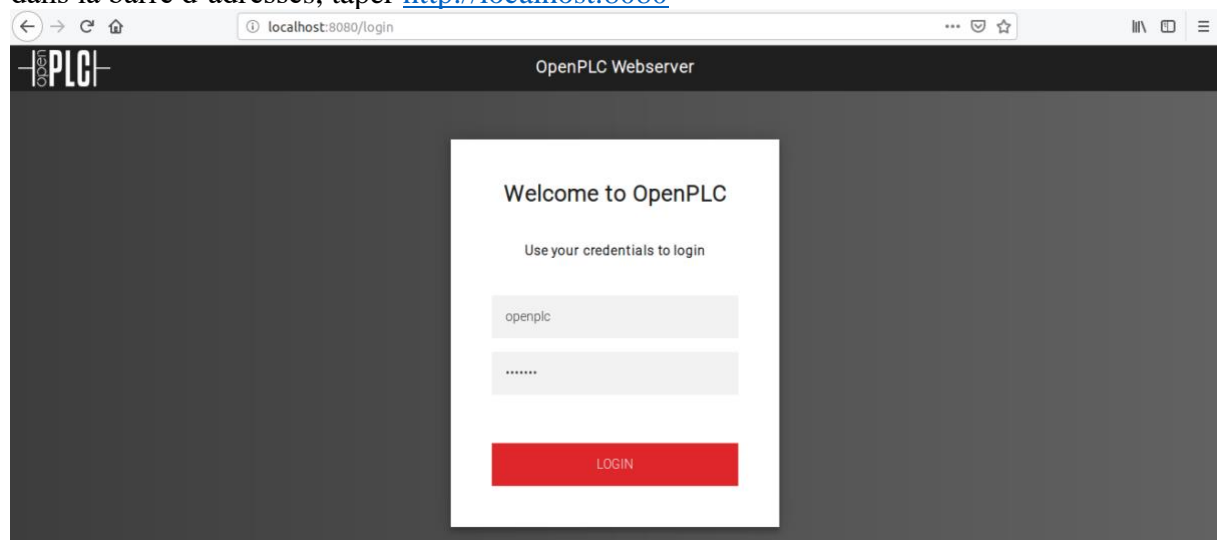
Q1. Créer un programme LADDER pour construire ces commandes du système. La table de correspondance entre les variables et leurs localisations dans la mémoire du PLC est la suivante :

Description:		Class Filter: All					
#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	START_BUTTON	Local	BOOL	%QX0.0	0		
2	LEVEL_SENSOR	Local	BOOL	%QX0.1	0		
3	CONTACT_SENSOR	Local	BOOL	%QX0.2	0		
4	MOTOR	Local	BOOL	%QX0.3	0		
5	NOZZLE	Local	BOOL	%QX0.4	0		
6	RUN	Local	BOOL	%QX0.5			

Q2. Compiler le programme  dans un fichier *factory.st*, et simuler son fonctionnement avec OpenPLC Editor 

Étape 3 : Installation de OpenPLC runtime

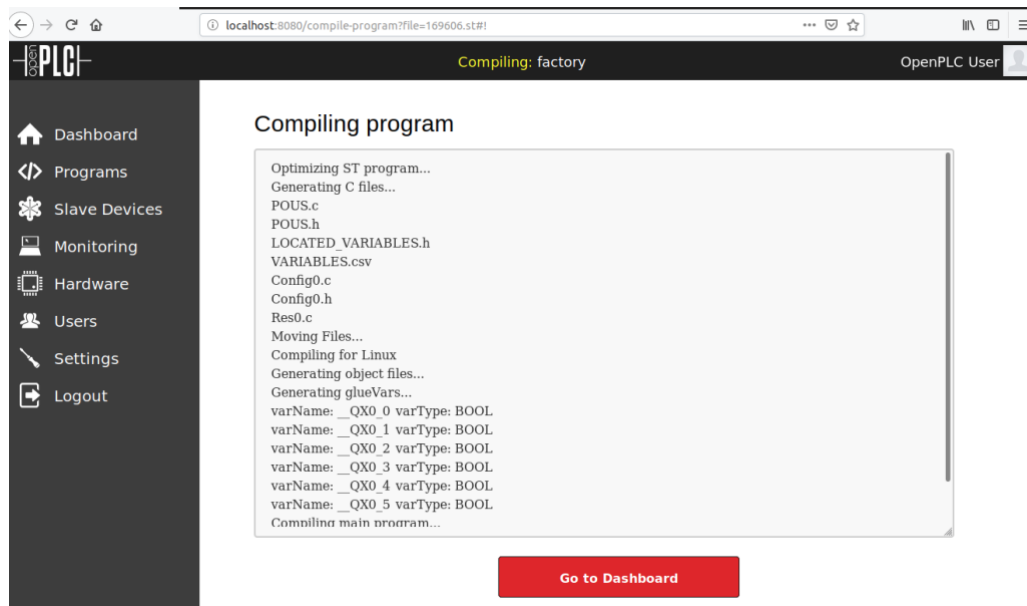
Pour accéder au PLC lancé dans une machine virtuelle docker, ouvrez un navigateur web et dans la barre d'adresses, taper <http://localhost:8080>



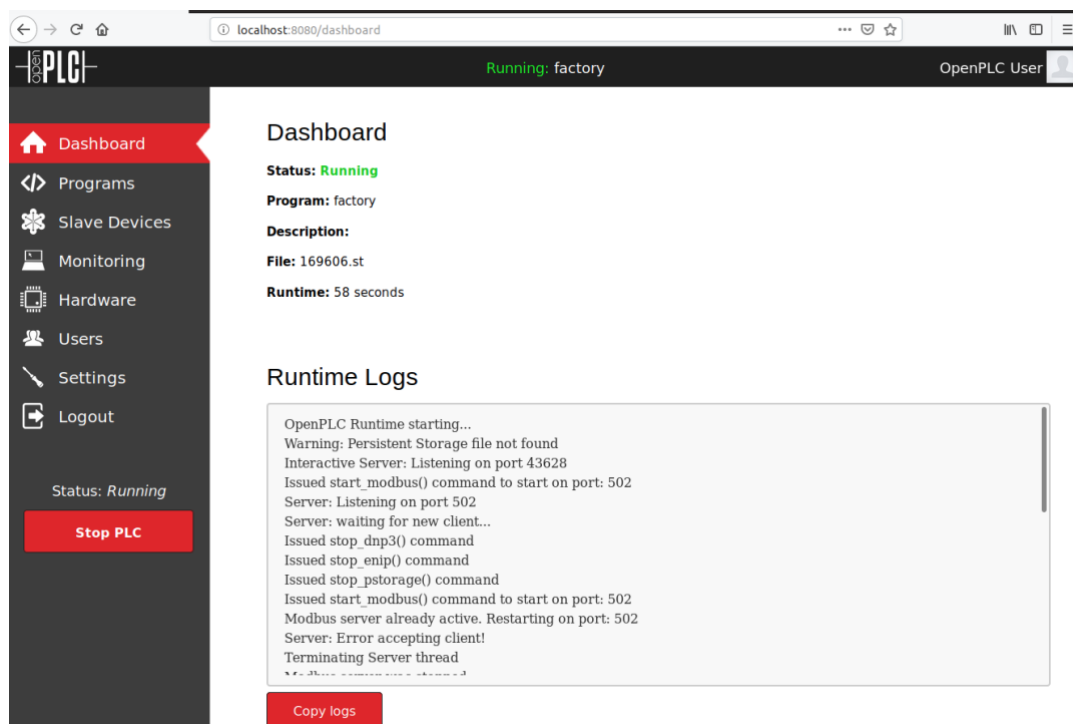
Pour accéder à l'interface web du PLC, le nom d'utilisateur est **openplc** et le mot de passe est **openplc**.

Étape 4 : Chargement du programme dans OpenPLC runtime

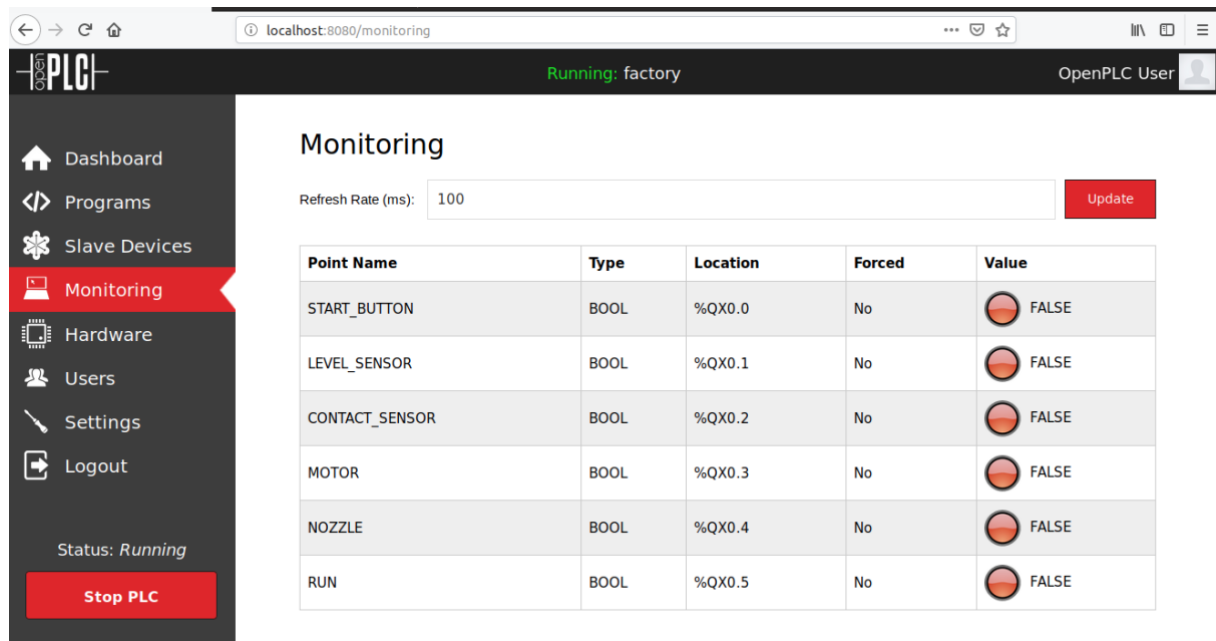
Ensuite, vous allez charger le fichier de votre programme dans le PLC. Cliquer dans le menu à gauche sur Programs. Dans la nouvelle fenêtre, cliquer sur *Browse* pour choisir votre fichier *factory.st* et cliquer ensuite sur le bouton *Upload Program*. Donnez le nom *factory* à votre programme et cliquer sur la nouvelle bouton Upload program.



Dans le menu **Settings**, désactiver les options **Enable DNP3 Server** et **Enable EtherNet/IP Server**. Garder seulement **Enable ModbusServer**.
 Pour démarrer le PLC, cliquer sur le bouton **Start PLC** situé dans le menu à gauche.



Cliquer maintenant dans le menu de gauche sur **Monitoring** pour visualiser l'état des variables de votre programme.



Si vous voulez arrêter le PLC, vous cliquez sur le bouton **Stop PLC**.

Exercice 3 : Démarrage de la chaîne de mise en bouteille (Machine 1)

Maintenant que notre PLC est programmé pour commander notre système. Vous allez lancer le simulateur de la chaîne de mise en bouteille depuis la machine 1.

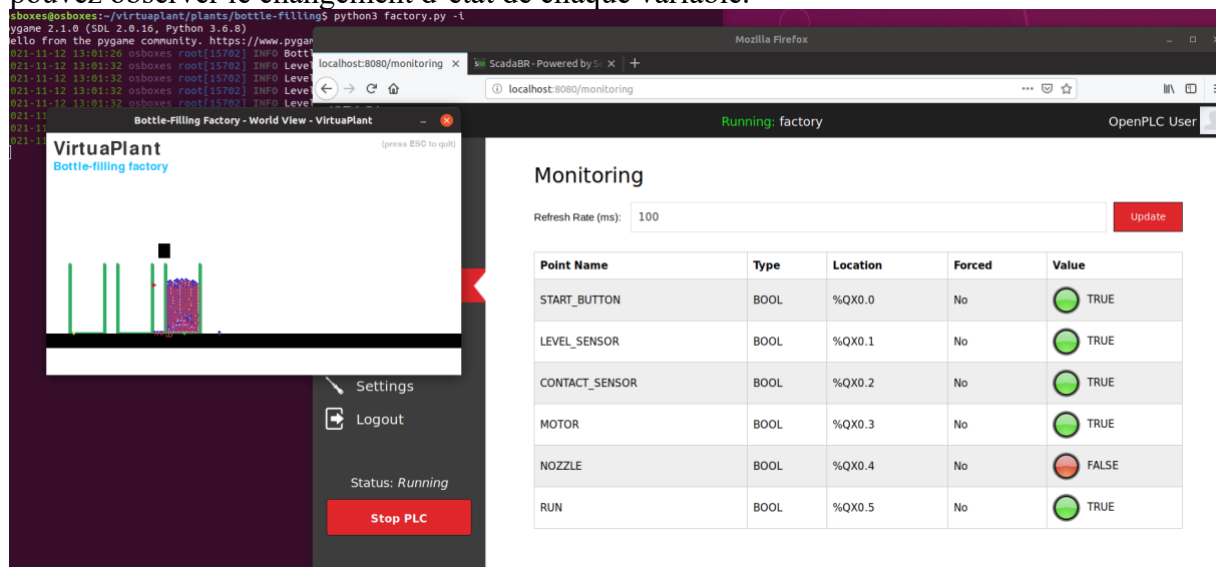
Dans un terminal Linux, tapez cette commande :

```
docker exec -ti factory_host bash
```

Dans le nouveau terminal, tapez la commande suivante :

```
cd security-ics  
python3 factory.py -s 192.168.1.2 -i
```

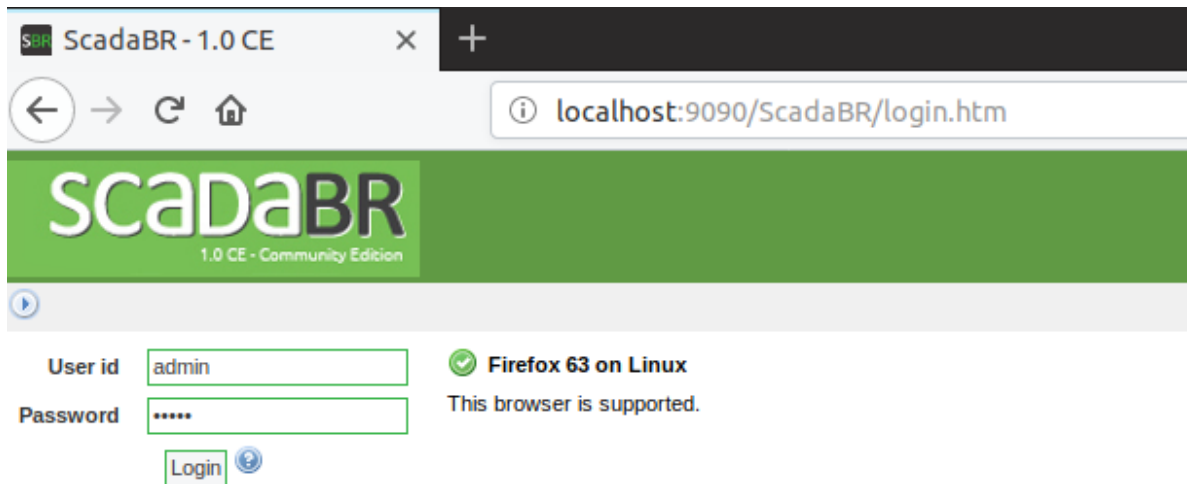
Votre chaîne de mise en bouteille démarre et dans la rubrique Monitoring du PLC, vous pouvez observer le changement d'état de chaque variable.





Si vous constatez que le comportement de votre système ne correspond pas à sa logique de commande prévue, il faut revenir dans OpenPLC Editor pour modifier le programme LADDER, générer le fichier *factory.st* et le charger dans OpenPLC runtime.


Exercice 4 : construisons une IHM pour notre système (Machine 3)

Pour construire l'IHM de notre système, nous utilisons l'outil ScadaBR³. Dans un navigateur web, taper l'adresse <http://localhost:9090/ScadaBR> pour accéder à l'interface principale de l'outil. Le login est admin et le mot de passe est également admin.



Dans l'interface graphique de ScadaBR, vous allez configurer les sources de données, c'est à dire l'adresse IP de la machine PLC. Dans le menu principal cliquer sur , ensuite dans la page affichée, sélectionnez comme Data Sources ModbusIP et cliquer sur .



Configurez la source de données avec les valeurs suivantes. Indiquer dans le champ *Host*, l'adresse IP de votre PLC (192.168.1.2). Dans la rubrique Points, vous allez créer les E/S de votre système c'est à dire le CONTACT_SENSOR, LEVEL_SENSOR, MOTOR, NOZZLE, START_BUTTON. Pour chacun d'eux, indiquer son offset, c'est à dire sa position dans les E/S de l'automate. Par exemple, le CONTACT_SENSOR possède un offset de 2 (%QX.02). Activer la valeur Settable pour le START_BUTTON, puisqu'elle sera activée depuis l'IHM. Il faut également penser à activer les Points, en cliquant sur .

³ <https://www.openplcproject.com/reference/scadabr/>

Modbus IP properties

Name

Export ID (XID)

Update period minutes(s)

Quantize ☐

Timeout (ms)

Retries

Contiguous batches only ☐

Create slave monitor points ☐

Max read bit count

Max read register count

Max write register count

Transport type

Host

Port

Encapsulated ☐

Event alarm levels

Data source exception

Point read exception

Point write exception

Modbus node scan

Nodes found

Modbus read data

Slave id

Register range

Offset (0-based)

Number of registers

Point locator test

Slave id

Register range

Modbus data type

Offset (0-based)

Bit

Number of registers

Character encoding

Points

Name	Data type	Status	Slave	Range	Offset (0-based)
CONTACT_SENSOR	Binary		1	Coil status	2
LEVEL_SENSOR	Binary		1	Coil status	1
MOTOR	Binary		1	Coil status	3
NOZZLE	Binary		1	Coil status	4
START_BUTTON	Binary		1	Coil status	0

Point details

Name

Export ID (XID)

Slave id

Register range

Modbus data type

Offset (0-based)

Bit

Number of registers

Character encoding

Settable ☐

Multiplier

Additive

Show Applications

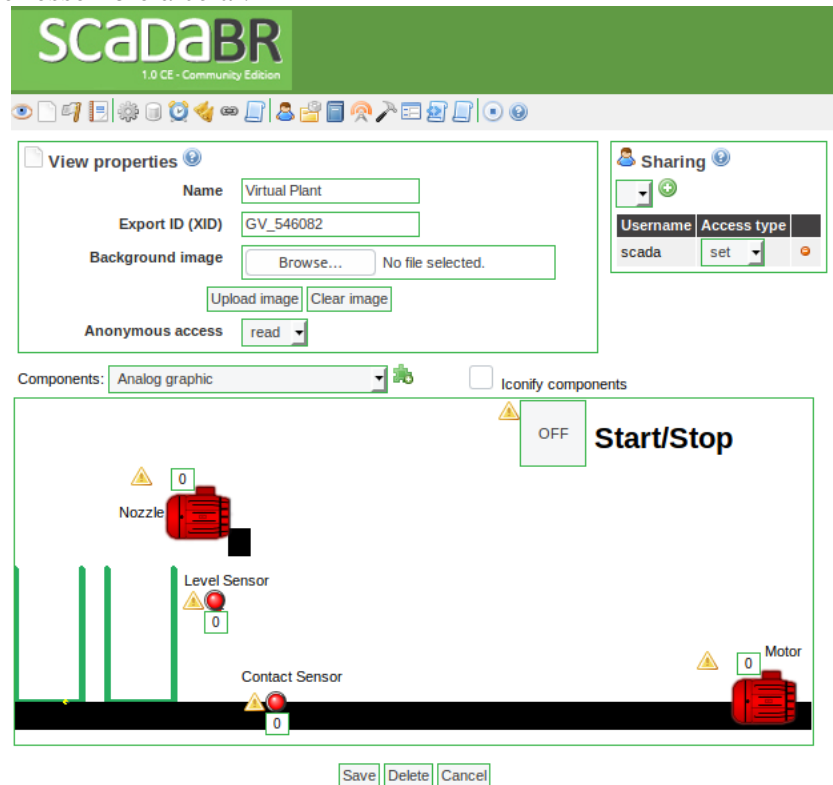
Dans le menu principal (la barre d'outil principal), cliquer sur le watch list pour afficher les E/S de votre système.

Points	Watch list
openPLC - CONTACT_SENSOR	openPLC - START_BUTTON 0 Nov 22 06:25
openPLC - LEVEL_SENSOR	Point value may not be reliable
openPLC - MOTOR	openPLC - NOZZLE 0 Nov 15 06:25
openPLC - NOZZLE	Point value may not be reliable
openPLC - MOTOR	openPLC - MOTOR 0 Nov 22 06:25
Point value may not be reliable	openPLC - LEVEL_SENSOR 0 Nov 22 06:15
Point value may not be reliable	openPLC - CONTACT_SENSOR 0 Nov 22 06:15

Vous allez maintenant, construire une simple interface graphique de votre IHM pour visualiser l'état du système. Cliquer sur le bouton Graphical Views . Dans la page affichée, cliquer sur pour créer une nouvelle interface. Comme image de fond (Background image), utiliser l'image (*background.png*) disponible sur Arche, et cliquer sur Upload image pour l'associer à votre interface. Pour rajouter les différents composants (les deux capteurs, le nozzle et le moteur), on utilisera les composants de type Binary graphic. Pour chaque élément, vous avez

besoin de lui associer son *Point* (son E/S associé au PLC) et un aspect graphique. Pour le bouton ON/OFF, vous pouvez utiliser le component *Button (write)*.

Votre interface ressemble à cela :



Pour tester votre IHM, il faut que le PLC soit en cours d'exécution et que vous lancez le programme Python `factory.py` du système physique sans l'option `-i` :

```
python3 factory -s 192.168.1.2
```

A cette étape, vous avez un système SCADA opérationnel.

Partie 2 : Analyse des paquets Modbus TCP (4 points)

Dans cette deuxième partie, vous allez analyser les paquets Modbus échangés entre les différents équipements de votre installation industrielle. L'objectif est d'étudier ces échanges pour identifier les messages échangés et les valeurs des différents champs.

Exercice 1 : Messages Modbus entre PLC et le système.

Avec l'outil Wireshark, vous allez capturer les messages Modbus échangés entre le PLC (machine 2) et le système industriel (machine 1). Sur votre machine, et dans un terminal, lancer Wireshark. Utiliser le filtre Modbus pour afficher seulement les messages Modbus. Lancer ensuite les différents composants de votre installation (PLC, système, IHM). Depuis l'IHM activer le système avec le bouton ON/OFF. Arrêter le système et la capture au bout de 30 secondes.

Q1. Quels sont les types de messages Modbus utilisés ?

Q2. Quel est le Function Code du message Modbus utilisé pour lire un coil ?

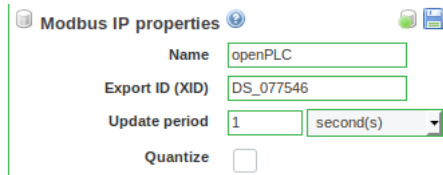
Q3. Que signifie le champ Reference Number dans un message pour lire un coil ?

Q4. Comment ils sont identifiés les paires de messages query et response ?

Q5. Quel est le Function code du message ModBus utilisé pour écrire un coil ?

Exercice 1 : Messages Modbus entre PLC et l'IHM.

Dans ScadaBR, configurer le data Source Modbus IP pour interroger le PLC toutes les secondes.



Lancer wireshark sur la machine 3 (IHM) pour capturer les messages modbus. Lancer ensuite votre système. Arrêter le système et la capture au bout de 30 secondes.

Q1. Quels sont les types de messages ModBus utilisés ?

Q2. Quel est l'intervalle du temps entre deux messages Query successifs ?

Q3. Quel est le Function Code du message ModBus utilisé pour le lire un coil ?

Q4. Combien de coils sont lus par ce message ?

Q5. Quel est le Function code du message ModBus utilisé pour écrire un coil ?

Partie 3 : Notre installation est under Attack ! (4 points)

Dans cette troisième partie, vous allez développer et tester des attaques sur l'installation industrielle. Nous allons tester des attaques pour forcer les valeurs des capteurs ou actionneurs,.

Vous allez utiliser un programme Python utilisant le protocole modbus pour forcer certaines valeurs d'E/S sur le PLC. Depuis l'archive attacks.zip sur Arche récupérer le fichier attacks.py

Installer les bibliothèques suivantes :

```
pip3 install coloredlogs pymodbus
```

Ce programme python contient un squelette de code pour réaliser 4 types d'attaques. Une première est déjà codé dans la fonction *never_stop*.

Q1. Tester cette attaque sur votre système en lançant le programme python avec l'option -c never_stop.

Q2. Coder et tester maintenant 3 autres attaques :

- *never_stop* : l'objectif de l'attaquant est que le convoyeur des bouteilles ne s'arrête jamais, c'est à dire qu'aucune bouteille ne soit rempli.
- *stop_all* : l'objectif de l'attaquant est de forcer l'arrêt complet du système.
- *stop_and_fill* : l'objectif de l'attaquant est de forcer l'ouverture sans arrêt du Nozzle pour déborder le remplissage d'une bouteille.

Partie 4 : Protéger et sécuriser votre installation (4 points)

Dans la partie 3, vous avez pu tester et constater que notre installation est la cible des attaques pour forcer les valeurs de capteurs et actionneurs. Nous allons dans dernière étape, sécuriser notre installation avec un pare-feu pour limiter les communications entre les machines.

Dans cet exercice, vous allez revoir l'architecture réseau du système industriel pour créer deux zones différentes séparées par un firewall : la première zone contient la station de l'IHM, la machine PLC et le système industriel, la deuxième zone (représente Internet) contient la machine d'attaque.

Vous allez configurer les machines système (192.168.1.4), PLC (192.168.1.2) et IHM (192.168.1.3) avec la pare-feu UFW (Uncomplicated Firewall)⁴ pour interdire toutes les connexions sortantes et entrantes sur ces 3 machines, à l'exception de :

- La machine 1 (Système) communique seulement avec la machine 2 (PLC) sur le port 502 en sortie
 - La machine 3 (IHM) communique seulement avec la machine 2 (PLC) sur le port 502 en sortie
 - La machine 2 (PLC) accepte des connexions entrantes sur le port 502 depuis les adresses IP de la machine 1 ou la machine 3
- Par exemple cette règle de firewall sur le PLC autorise les connexions entrantes sur port 502 depuis la machine 192.168.1.4 (*factory*) :

Pour accéder au terminal du docker de OpenPLC , utiliser la commande suivante :

```
docker exec -ti openplc_host bash
```

```
ufw enable
```

```
ufw allow from 192.168.1.4 to any port 502
```

Des exemples pour configurer UFW sont disponibles ici :

<https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-18-04>

Pour accéder au terminal du docker de SCADABR , utiliser la commande suivante :

```
docker exec -ti scadabr_host bash
```

Pour accéder au terminal du docker de Factory , utiliser la commande suivante :

```
docker exec -ti factory_host bash
```

A modifier sur les docker si vous avez une erreur ipatble6 :

```
Adjust /etc/default/ufw to have "IPV6=no"
```

⁴ <https://doc.ubuntu-fr.org/ufw>