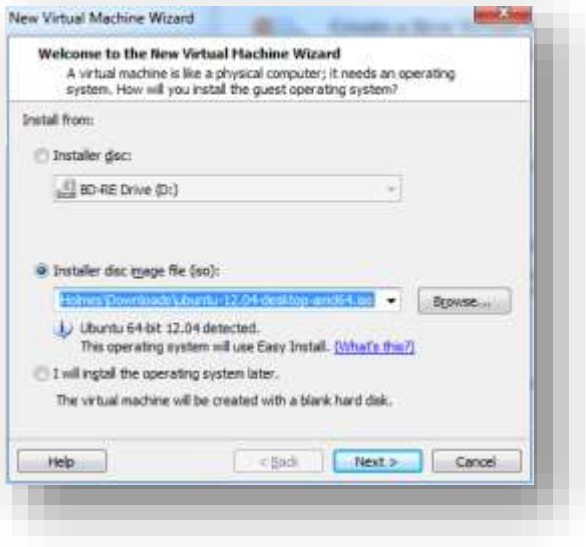
Tutorial: MQTT (Message Queuing Telemetry Transport)

1 Linux Ubuntu Computer

Boot your computer on Linux or use a VMware workstation

1.1 Installing Ubuntu in a VM on Windows (optional)

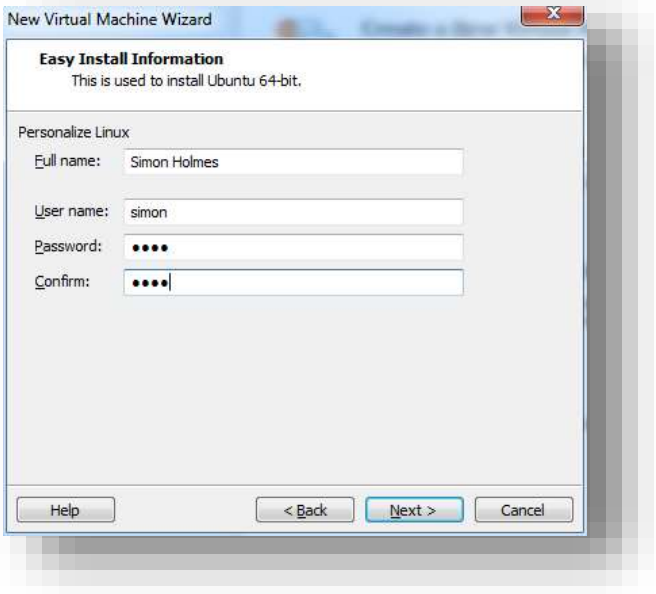
1. Download the Ubuntu iso (desktop not server) and the free VMware Player.
2. Install VMware Player and run it, you'll see something like this:



3. Select "Create a New Virtual Machine"
4. Select "Installer disc image file" and browse to the Ubuntu iso you downloaded.

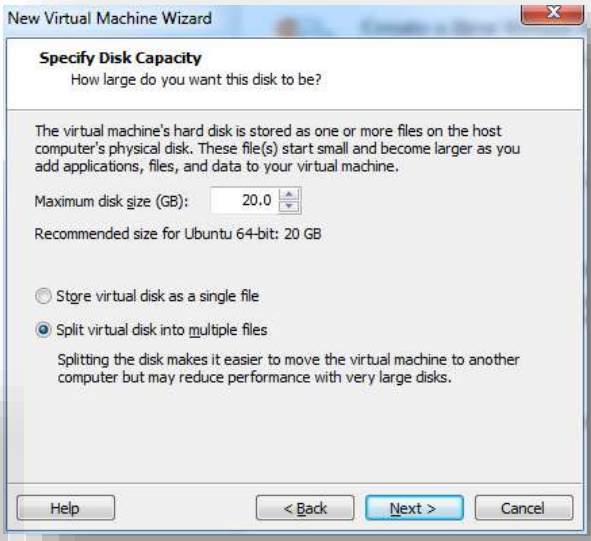
You should see that it will use Easy Install – this takes

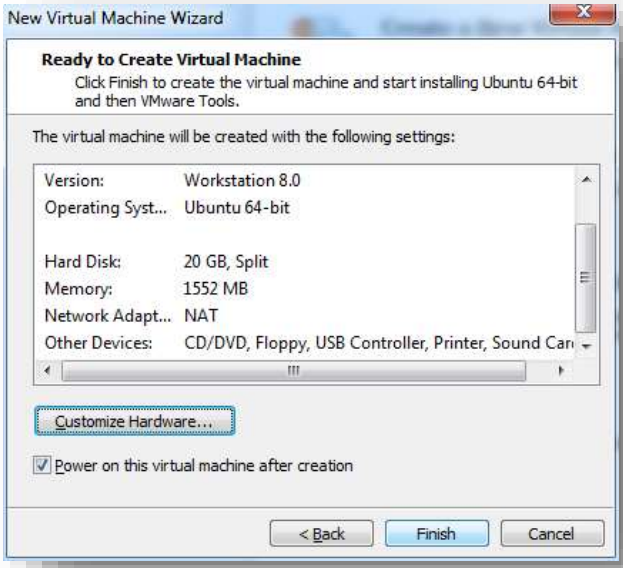
5. Enter your full name, username and password and hit next



Tutorial: MQTT (Message Queuing Telemetry Transport)

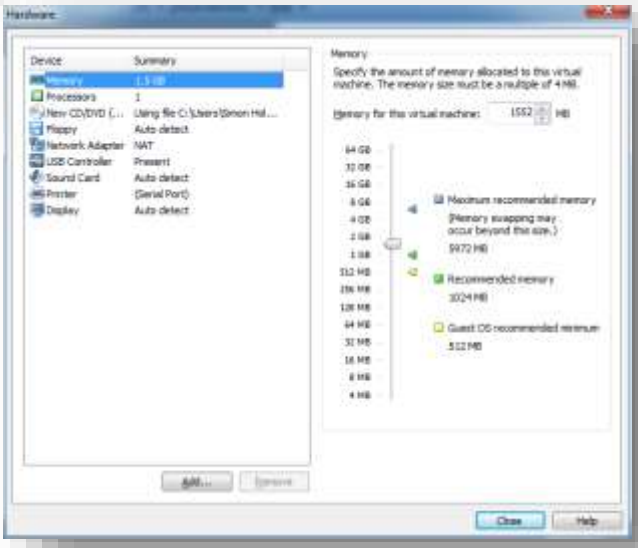
6. Select the maximum disk size and type. Unless you're planning on some really CPU intensive work inside the VM, select the "Split virtual disk into multiple files" option. Hit next when you're happy with the settings.





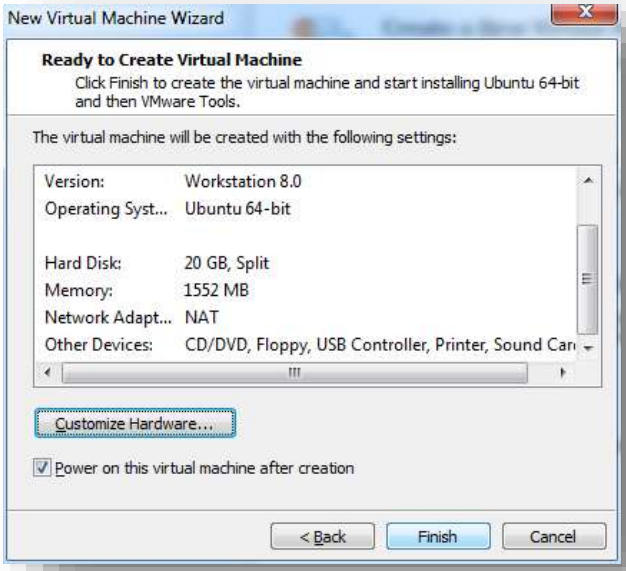
7. This brings you to the confirmation page. Click "Customize Hardware"

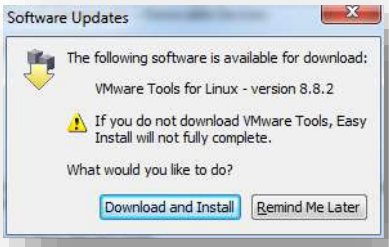
8. In the hardware options section select the amount of memory you want the VM to use. In this instance I've gone for 1.5GB out of the 8GB installed in my laptop. Leave everything else as it is and click Close.



Tutorial: MQTT (Message Queuing Telemetry Transport)

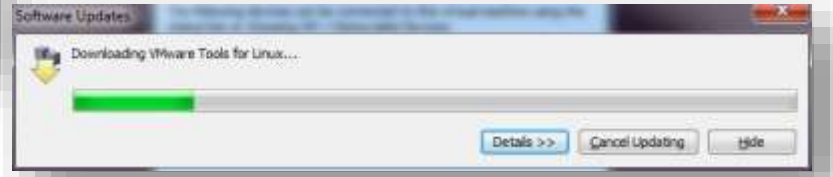
9. This brings you back to the confirmation page. Click Finish this time





10. You will probably be prompted to download VMware Tools for Linux. Click "Download and Install" to continue

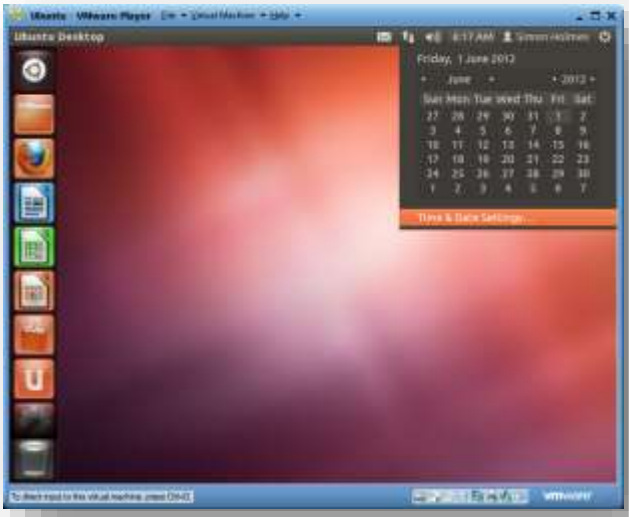
11. Wait for it to install



12. Ubuntu will then start to install, so keep waiting (or do what I did and go to bed!)

Tutorial: MQTT (Message Queuing Telemetry Transport)

13. When all is done you'll be presented with the Ubuntu login screen. So enter your password and you're on your way.



14. Click the clock in the top right to set your time and date settings

15. Once you've set that up, you're up and running with Ubuntu in VMware Player on your Windows machine. Congratulations and enjoy!

2 MQTT introduction :

MQTT is a lightweight publish/subscribe messaging protocol. It is useful for use with low power sensors, but is applicable to many scenarios.

2.1 Publish/Subscribe

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, common interface for everything to connect to. This means that you if you have clients that dump subscribed messages to a database, to Twitter, Cosm or even a simple text file, then it becomes very simple to add new sensors or other data input to a database, Twitter or so on.

2.2 Topics/Subscriptions

Tutorial: MQTT (Message Queuing Telemetry Transport)

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem. For example, multiple computers may all publish their hard drive temperature information on the following topic, with their own computer and hard drive name being replaced as appropriate:

sensors/COMPUTER_NAME/temperature/HARDDRIVE_NAME

Clients can receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards. Two wildcards are available, + or #.

+ can be used as a wildcard for a single level of hierarchy. It could be used with the topic above to get information on all computers and hard drives as follows:

sensors/+/temperature/+

As another example, for a topic of "a/b/c/d", the following example subscriptions will match:

a/b/c/d +/b/c/d a/+/c/d a/+/+/d +/+/+/+

The following subscriptions will not match:

a/b/c b/+/c/d +/+/+

can be used as a wildcard for all remaining levels of hierarchy. This means that it must be the final character in a subscription. With a topic of "a/b/c/d", the following example subscriptions will match:

a/b/c/d # a/# a/b/# a/b/c/# +/b/c/#

Zero length topic levels are valid, which can lead to some slightly non-obvious behaviour. For example, a topic of "a//topic" would correctly match against a subscription of "a/+/topic". Likewise, zero length topic levels can exist at both the beginning and the end of a topic string, so "/a/topic" would match against a subscription of "+/a/topic", "#" or "/#", and a topic "a/topic/" would match against a subscription of "a/topic/+" or "a/topic/#".

2.3 Quality of Service

MQTT defines three levels of Quality of Service (QoS). The QoS defines how hard the broker/client will try to ensure that a message is received. Messages may be sent at any QoS level, and clients may attempt to subscribe to topics at any QoS level. This means that the client chooses the maximum QoS it will receive. For example, if a message is published at QoS 2 and a client is subscribed with QoS 0, the message will be delivered to that client with QoS 0. If a second client is also subscribed to the same topic, but with QoS 2, then it will receive the same message but with QoS 2. For a second example, if a client is subscribed with QoS 2 and a message is published on QoS 0, the client will receive it on QoS 0.

Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

0: The broker/client will deliver the message once, with no confirmation.

1: The broker/client will deliver the message at least once, with confirmation required.

2: The broker/client will deliver the message exactly once by using a four step handshake.

Tutorial: MQTT (Message Queuing Telemetry Transport)

3 Mosquitto

Mosquitto is an open source (BSD licensed) message broker that implements the MQ Telemetry Transport protocol version 3.1. MQTT provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for “machine to machine” messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers like the Arduino.

3.1 Installing Mosquitto

As of version 11.10 Oneiric Ocelot, mosquitto will be in the Ubuntu repositories so you can install as with any other package. If you are on an earlier version of Ubuntu or want a more recent version of mosquitto, add the mosquitto-dev PPA to your repositories list – see the link for details.

Exercice 1 : Install mosquitto from your package manager.

```
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
sudo apt-get update
```

If the command “apt-add-repository” is not recognized, it can be installed with:

```
sudo apt-get install python-software-properties
```

4 Test Mosquitto with test.mosquitto.org (a MQTT server/broker)

This is test.mosquitto.org. It hosts a publicly available Mosquitto MQTT server/broker. MQTT is a very lightweight protocol that uses a publish/subscribe model. This makes it suitable for "machine to machine" messaging such as with low power sensors or mobile devices.

4.1 The server

The server listens on the following ports:

1883 : MQTT, unencrypted

8883 : MQTT, encrypted

8884 : MQTT, encrypted, client certificate required

8080 : MQTT over WebSockets, unencrypted

8081 : MQTT over WebSockets, encrypted

The encrypted ports support TLS v1.2, v1.1 or v1.0 with x509 certificates and require client support to connect. In all cases you should use the certificate authority file mosquitto.org.crt to verify the server connection. Port 8884 requires clients to provide a certificate to authenticate their connection. If you wish to obtain a client certificate, please get it touch.

You are free to use it for any application, but please do not abuse or rely upon it for anything of importance. You should also build your client to cope with the broker restarting.

Please don't publish anything sensitive, anybody could be listening.

Tutorial: MQTT (Message Queuing Telemetry Transport)

4.2 Caveats

This server is provided as a service for the community to do testing, but it is also extremely useful for testing the server. This means that it will often be running unreleased or experimental code and may not be as stable as you might hope. It may also be. Finally, not all of the features may be available all of the time, depending on what testing is being done. In particular, websockets and TLS support are the most likely to be unavailable.

In general you can expect the server to be up and to be stable though.

4.3 Test with a remote Mosquitto MQTT server/broker (http://test.mosquitto.org)

4.3.1 D3 MQTT topic tree visualizer

http://test.mosquitto.org/sys/ allows to visualize the $SYS tree of the broker. See how the tree change dynamically.

4.3.2 Demo and manipulation on temperature gauge

http://test.mosquitto.org/gauge/ is an HTML5 canvas gauge for temperature obtained from an MQTT subscribe.

A local process runs every 15 seconds to update the value by adding a random value in the range +/-2 degrees.

Exercice 2 : Publish to the "temp/random" topic to change the gauge and to test it :

```
mosquitto_pub -h test.mosquitto.org -t temp/random -m 23.0
```

Exercice 3 : Subscribe to the "temp/random" and see what happen as soon temp/random changes :

```
mosquitto_sub -h test.mosquitto.org -t temp/random -v
```

Exercice 4 : Write a temperature profile in a file and publish it with time stamps. Trace in an other file, the change of the temperature through the client/subscribe.

Exercice 5 : Display and compare both files wth gnuplot (install gnuplot if necessary).

4.4

4.4.1 MQTT Man pages

For more information on MQTT, see http://mqtt.org/ or the Mosquitto MQTT man page: http://mosquitto.org/man/

4.4.2 mosquitto — an MQTT broker

```
mosquitto [-c config file] [ -d | --daemon ] [-p port number] [-v]
```

4.4.3 mosquitto_pub

Tutorial: MQTT (Message Queuing Telemetry Transport)


```
mosquitto_pub [-A bind_address] [-d] [-h hostname] [-i client_id] [-I client id
prefix] [-k keepalive time] [-p port number] [-q message QoS] [--quiet] [-r] [-S] {
-f file | -l | -m message | -n | -s } [ [-u username] [-P password] ] [ --will-topic
topic [--will-payload payload] [--will-qos qos] [--will-retain] ] [[ { --cafile
file | --capath dir } [--cert file] [--key file] [--ciphers ciphers] [--tls-
version version] [--insecure] ] | [ --psk hex-key --psk-identity identity [--
ciphers ciphers] [--tls-version version] ]] [--proxy socks-url] [-V protocol-
version] -t message-topic
```

Exercice 6 : Test these examples

Publish temperature information to localhost with QoS 1:

- mosquitto_pub -t sensors/temperature -m 32 -q 1

Publish timestamp and temperature information to a remote host on a non-standard port and QoS 0:

- mosquitto_pub -h 192.168.1.1 -p 1885 -t sensors/temperature -m "1266193804 32"

Publish light switch status. Message is set to retain because there may be a long period of time between light switch events:

- mosquitto_pub -r -t switches/kitchen_lights/status -m "on"

Send the contents of a file in two ways:

- mosquitto_pub -t my/topic -f ./data
- mosquitto_pub -t my/topic -s < ./data

Send parsed electricity usage data from a Current Cost meter, reading from stdin with one line/reading as one message:

- read_cc128.pl | mosquitto_pub -t sensors/cc128 –l

4.4.4 mosquitto_sub

```
mosquitto_sub [-A bind_address] [-c] [-C msg count] [-d] [-h hostname] [-
i client_id] [-I client id prefix] [-k keepalive time] [-p port number]
[-q message QoS] [-R] [-S] [-N] [--quiet] [-v] [ [-u username] [-P
password] ] [ --will-topic topic [--will-payload payload] [--will-qos
qos] [--will-retain] ] [[ { --cafile file | --capath dir } [--cert file]
[--key file] [--tls-version version] [--insecure] ] | [ --psk hex-key --
psk-identity identity [--tls-version version] ]] [--proxy socks-url] [-V
protocol-version] [-T filter-out...] -t message-topic...
```

Tutorial: MQTT (Message Queuing Telemetry Transport)

Exercice 7 : Test these examples

Subscribe to temperature information on localhost with QoS 1:

- mosquitto_sub -t sensors/temperature -q 1

Subscribe to hard drive temperature updates on multiple machines/hard drives. This expects each machine to be publishing its hard drive temperature to sensors/machines/HOSTNAME/temperature/HD_NAME.

- mosquitto_sub -t sensors/machines/+/temperature/+

Subscribe to all broker status messages:

- mosquitto_sub -v -t \$SYS/#

5 The Seven Best MQTT Client Tools

Everybody from MQTT beginner to expert needs a handy tool to try out stuff or for debugging. The site (http://www.hivemq.com/blog/seven-best-mqtt-client-tools) gives a brief overview of the best MQTT client tools for different platforms and highlight special features.

Exercice 8 : Install MQTT.fx on Windows and configure it to connect the remote Mosquitto MQTT broker that you installed. Use the graphical user interface to publish and subscribe events.

This tool will be useful for the following.

6 MQTT Client Programming

6.1 MQTT Client in C# programming language

First we need to install M2Mqtt for .Net https://m2mqtt.wordpress.com/.
The better way is to use the "package manager console" in Visual Studio to download M2Mqtt as a nugget package. See https://www.nuget.org/packages/M2Mqtt/.

After reading http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-m2mqtt on M2qtt API, implement a Visual Studio Project : "Visual C#", "Application Console Win32" (empty project) to test the API.

Exercice 9 : Implement an event publisher and test it on your Mosquitto MQTT broker
Exercice 10 : Implement an event subscriber and test it on your Mosquitto MQTT broker

* MQTT Client in C# programming language (optional)

For the student that are not used with Java or C# programming, here you can find sample POSIX / C code to test MQTT API in the Paho Eclipse project (https://projects.eclipse.org/projects/technology.paho)

See for libraries :

https://www.eclipse.org/downloads/download.php?file=/paho/1.0/m2mqtt/M2Mqtt_4.0.0.0_bins.zip

See for sample codes :

Tutorial: MQTT (Message Queuing Telemetry Transport)

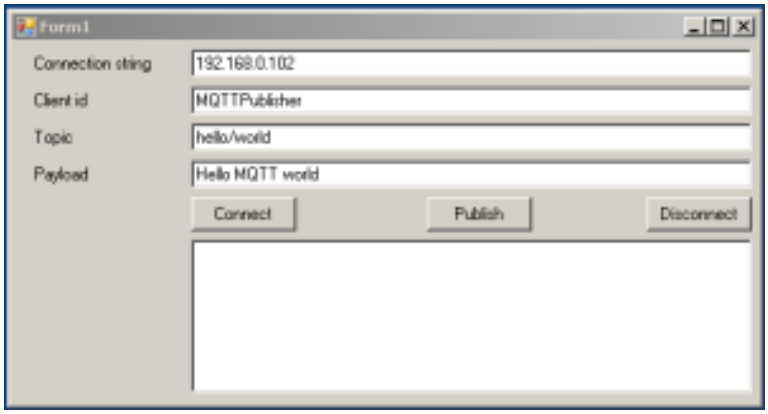
https://www.eclipse.org/downloads/download.php?file=/paho/1.1/eclipse-paho-mqtt-c-windows-1.0.3.zip

and explanations at https://eclipse.org/paho/clients/c/

6.2 Graphical Client in C# programming language (optional)

Exercice 11 : Build a graphical publish/subscribe MQTT client on Windows

1. Build such a Graphical interface using a "Visual C# / Application Windows Forms" Project



   First design the graphical interface layout using the toolbox to drag and drop widgets (graphical components) : 4 Labels, 5 Texboxes, 3 Button
   - Labels are : "Connection string", "Client id", "Topic", "Payload"
   - Buttons are : "Connect", "Publish", "Disconnect"
2. Implement all the corresponding handlers to events coming from the widgets and MQTT. They are :
   - MQTT connection
   - MQTT disconnection
   - MQTT Published event
   - MQTT Subscribed event

6.3 MQTT and Complex event processing (CEP)

Event processing is a method of tracking and analyzing (processing) streams of information (data) about things that happen (events) and deriving a conclusion from them. Complex event processing, or CEP, is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible.

Exercice 12 : Design an CEP architecture using MQTT, and implement a sample of it using mosquito.