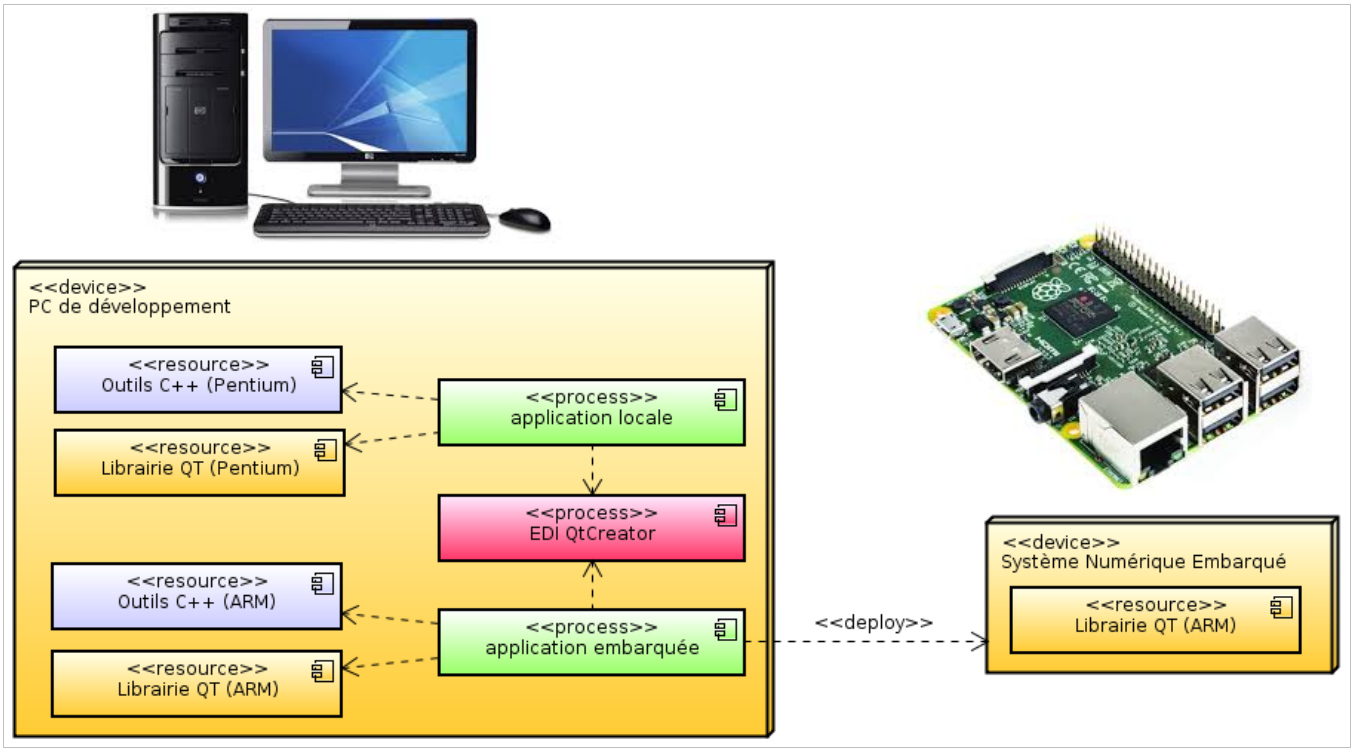
Ce document va nous permettre de comprendre comment installer et configurer l'ensemble des outils nécessaires pour réaliser à la fois du développement sur un poste classique dans un environnement Linux, mais aussi sur un système numérique embarqué à l'aide de la technique de compilation croisée appelée aussi « **cross-compiltion**».



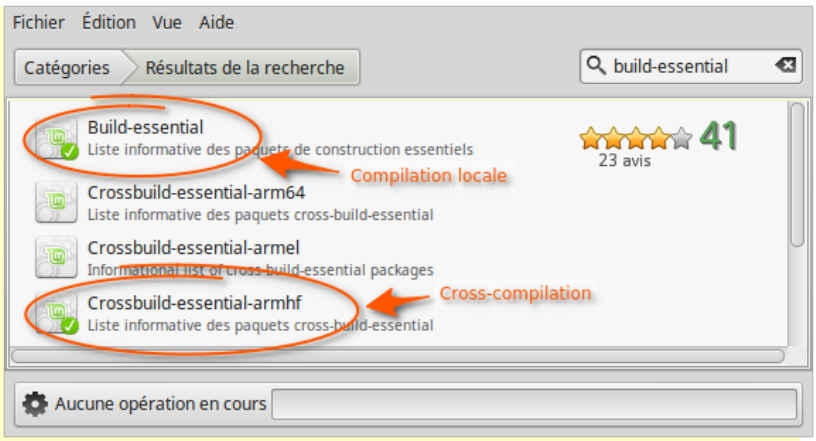
Installation des compilateurs GCC

Pour commencer, nous allons installer tous les outils de compilation nécessaires pour le poste Linux, d'une part le compilateur pour produire des logiciels sur le poste local et d'autre part le compilateur pour générer des programmes qui seront exécutés sur un système embarqué, comme la Raspberry**.** Sur le système embarqué lui-même, vous n'avez pas besoin d'installer de compilateur puisque la compilation se fera systématiquement sur le PC de développement.

Tout les outils de développement se trouvent dans le paquets « build-essential », mais en réalité, nous avons besoin de deux types de paquet pour les deux types de compilateurs.

Lorsque nous utilisons le « Gestionnaire de logiciels », nous voyons apparaître les deux types de paquets à installer. D'une part, le paquet « Build-essential », qui permet de compiler des programmes pour l'ordinateur hôte, d'autre part « Crossbuild-essential-armfh » qui va nous servir pour compiler les programmes pour l'informatique embarquée Raspberry.

Si vous possédez la Raspberry 1, choisissez plutôt le paquet « Crossbuild-essential-armel ».



Installation de la librairie Qt et de l'EDI QtCreator pour l'ordinateur hôte

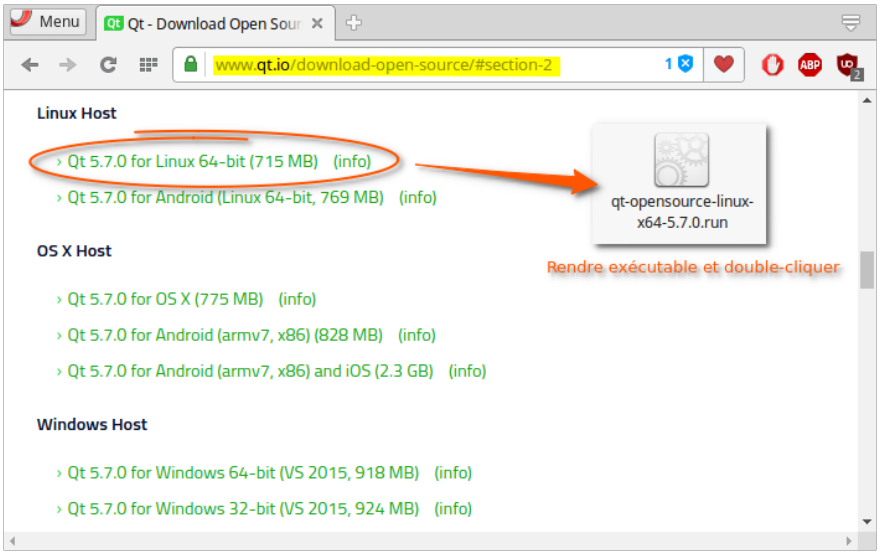
Nous pouvons procéder de deux façons différentes pour installer la librairie Qt avec l'environnement de développement intégré QtCreator (EDI).

Soit, nous récupérons l'ensemble du paquetage sur le site dédié dans sa dernière version et il suffira alors de double-cliquer sur l'installateur et suivre ce qui est proposé.

La première démarche paraît séduisante, surtout dans le cas où vous travaillez sur un système d'exploitation autre que Linux. L'inconvénient majeur toutefois, c'est que vous devez installer l'ensemble de l'outil (librairie Qt + QtCreator) exactement dans le même répertoire que sur le poste de développement, pour tous les postes qui vont utiliser les exécutables que vous aurez conçus (les postes cibles n'ont normalement pas besoin d'utiliser QtCreator, seule la librairie Qt est indispensable).

Également, si vous faites des mises à jour, vous devez tout réinstaller sur tous les postes qui vont bénéficier des nouveaux exécutables. Vous avez, page suivante, le site correspondant aux logiciels d'installation de la librairie Qt et de l'EDI QtCreator :

ATTENTION : Dans ce cas de figure, il faut également installer, en annexe, le paquet correspondant à la conception d'IHM avec la librairie graphique OpenGL. Pour cela tapez la commande suivante : `sudo apt-get install libgl1-mesa-dev`



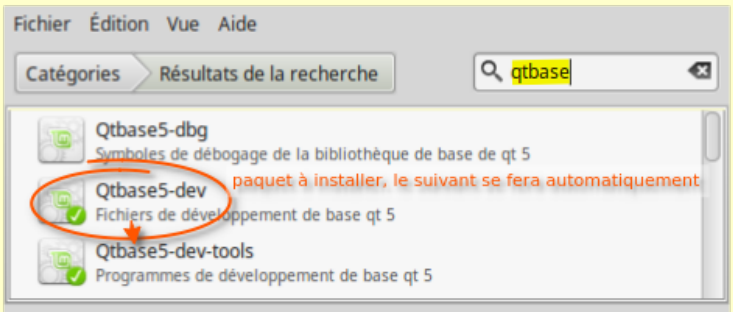
*Soit, dans le cas de Linux, nous pouvons proposer une installation ciblée (plusieurs paquets sont nécessaires pour que l'ensemble de l'outil soit intégralement opérationnel) en utilisant encore une fois le « **Gestionnaire de logiciels** ».*

La deuxième démarche consiste à utiliser le « gestionnaire de logiciels » (il est bien sûr possible de le faire en ligne de commande `sudo apt-get install`).

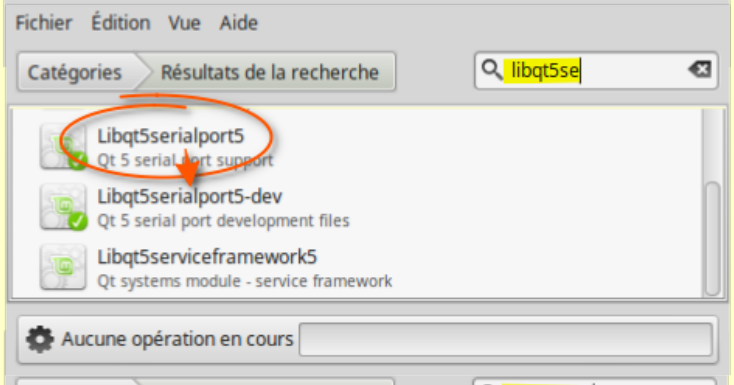
Cette deuxième démarche est plus adaptée à une installation sur l'ensemble d'un parc informatique, puisque nous pouvons installer uniquement la librairie Qt sur tous les postes, sans nous préoccuper de l'EDI QtCreator.

Ce dernier sera installé uniquement sur le poste de développement dédié à la génération des exécutables. Pour cette version de Linux, la librairie Qt actuelle est la version 5.5.1.

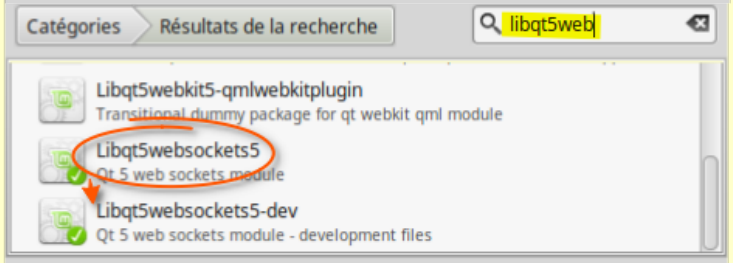
Pour la librairie Qt de base, voici le paquet à installer sur tous les postes cibles, ainsi bien entendu pour le poste de développement.



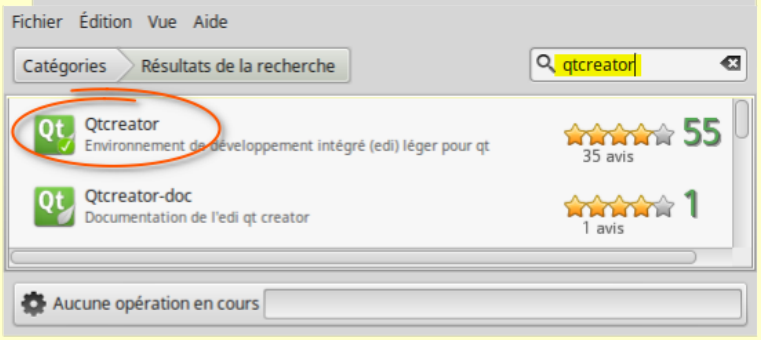
Si vous souhaitez intégrer du développement pour la communication série, comme avec les ports USB, intégrez également le paquet suivant :



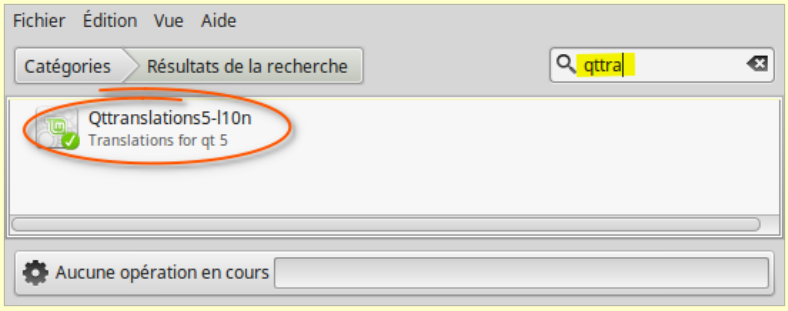
Si vous souhaitez faire de la communication réseau avec la technologie des « websockets », voici le paquet à installer :



l'EDI QtCreator devra être installé, uniquement sur le poste de développement dédié à la création des logiciels :



Si vous désirez l'avoir en français :



Compilation de la librairie Qt avec les sources pour une informatique embarquée (ARM-HF)

La librairie Qt est maintenant prête pour accepter des logiciels prévus pour le système hôte. Toutefois, il est nécessaire de créer une librairie **Qt** adaptée à une informatique embarquée de type Raspberry (avec un processeur **ARM-HF**). Cette librairie sera également présente sur l'ordinateur de développement afin que nous puissions réaliser de la compilation croisée « **cross-développement** ». Ainsi, les programmes créés sur l'ordinateur hôte pourront être auto-déployer sur les cibles de type Raspberry. Pour cela, nous avons besoins des sources de l'ensemble de la librairie pour les compiler en ARM-HF.

Nous pouvons récupérer les sources directement sur le site de Qt. Actuellement, la version est la 5.7, toutefois, nous avons choisi la version correspondant à celle installée sur tous les autres ordinateurs cibles, savoir la version 5.5.1 :

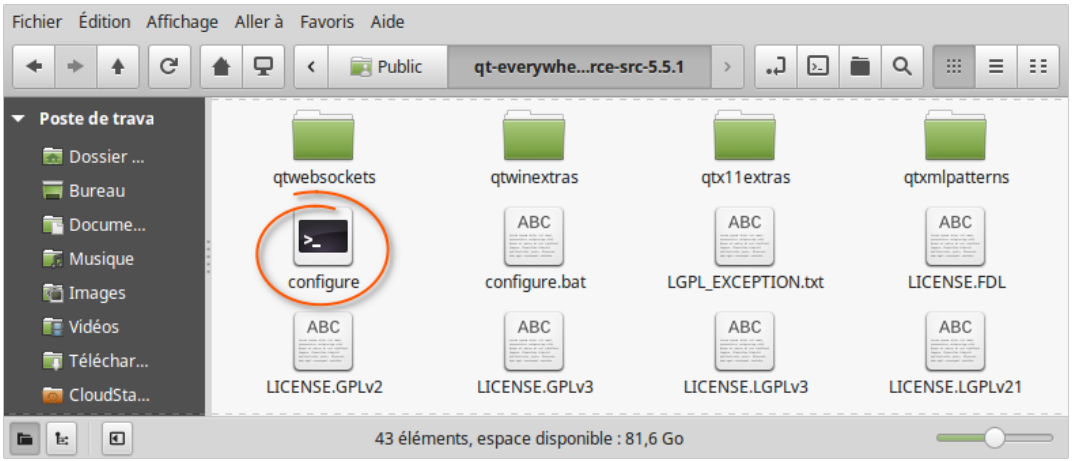
Trois phases sont nécessaires pour l'installation définitive : comme à chaque fois que vous devez installer un logiciel quelconque sur Linux à partir de ses sources :

La phase de configuration au travers de la commande : `$ ./configure`

La phase de construction à l'aide de l'utilitaire : `$ make`

La phase d'installation, en utilisant toujours l'utilitaire make : `$ sudo make install`

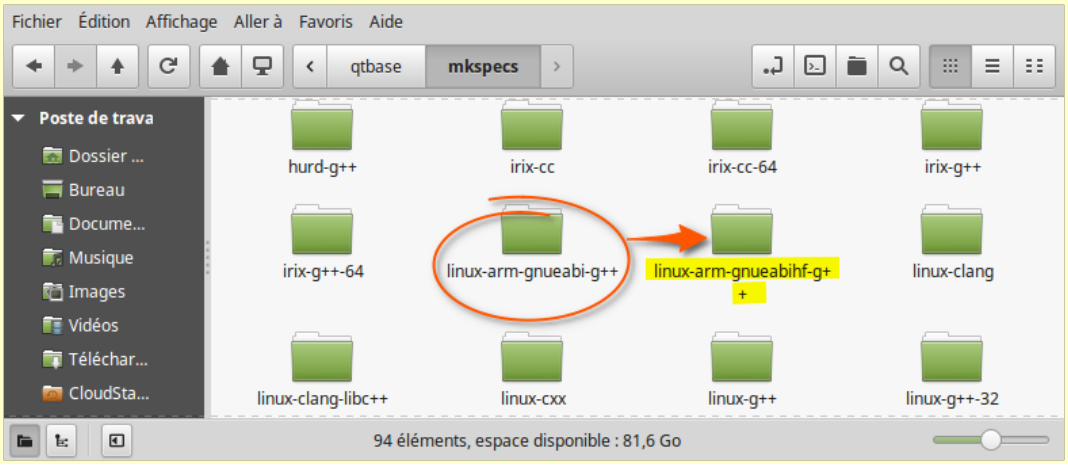
Décompacter l'archive sur l'ordinateur hôte : avant de réaliser ces trois phases, décompacter l'archive des sources de **Qt** dans un dossier qui vous sera accessible ultérieurement. Vous pouvez remarquer la présence de nombreux répertoires et du fichier de configuration exécutable (voir ci-dessous).



Configuration pour le compilateur ARM : Pour que l'exécutable « **configure** » sache quoi faire, il est indispensable au préalable de bien spécifier quel type de cible nous allons choisir.

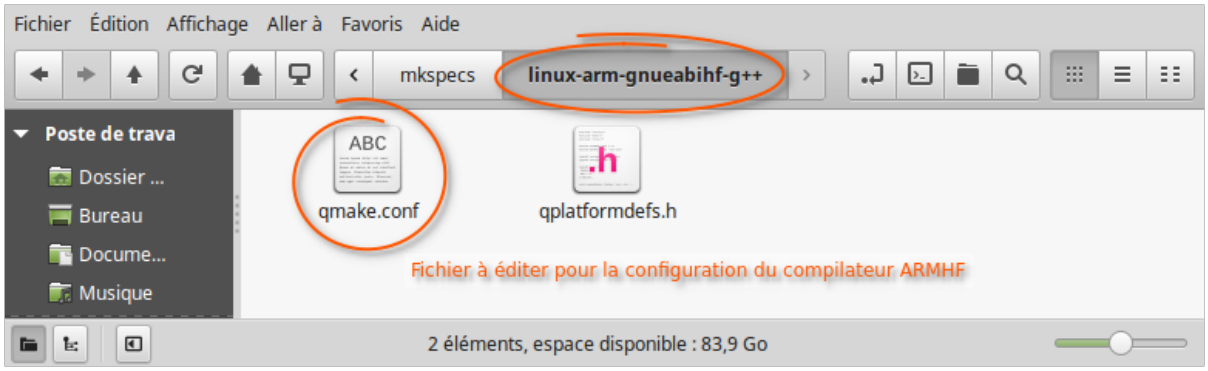
Pour cela, aller dans le répertoire « qtbase ». À l'intérieur de ce répertoire il existe un autre répertoire très important, nommé « mkspecs » qui, comme son nom l'indique, permet de choisir la ou les cibles à prendre en compte.

À l'intérieur de ce dernier, vous pouvez remarquer la présence de beaucoup de dossiers correspondants aux différents types de compilateur. Vous avez vraiment le choix. Nous découvrons justement le compilateur qui nous intéresse, avec un petit bémol tout de même, ce n'est à priori pas la bonne cible, savoir ***gnueabihf*** *(par contre, il convient très bien pour la Raspberry 1)*



Ce n'est pas très grave, il est possible de faire une copie de ce répertoire et de l'intituler avec la bonne cible.

Paramétrer le fichier qmake.conf de la cible gnueabihf : rentrer dans le dossier correspondant à la cible choisie et vous allez éditer le fichier de configuration avec le compilateur adapté, savoir « qmake.conf » :



*Ce fichier de configuration est pratiquement bien écrit, si ce n'est que nous devons à chaque fois bien préciser les bons outils de compilation de type « **gnueabihf** » (par défaut c'est « **gnueabi** »).*

qmake.conf

```
#
# qmake configuration for building with arm-linux-gnueabihf-g++
#

MAKEFILE_GENERATOR      = UNIX
CONFIG                 += incremental
QMAKE_INCREMENTAL_STYLE = sublib

include(../common/linux.conf)
include(../common/gcc-base-unix.conf)
include(../common/g++-unix.conf)

# modifications to g++.conf
QMAKE_CC                = arm-linux-gnueabihf-gcc
QMAKE_CXX               = arm-linux-gnueabihf-g++
QMAKE_LINK              = arm-linux-gnueabihf-g++
QMAKE_LINK_SHLIB        = arm-linux-gnueabihf-g++

# modifications to linux.conf
QMAKE_AR                = arm-linux-gnueabihf-ar cqs
QMAKE_OBJCOPY           = arm-linux-gnueabihf-objcopy
QMAKE_NM                = arm-linux-gnueabihf-nm -P
QMAKE_STRIP             = arm-linux-gnueabihf-strip
load(qt_config)
```

Lancer l'utilitaire de configuration : une fois que ce paramétrage est effectué, vous pouvez lancer la phase de configuration en spécifiant bien la cible (**xplatform**) pour que la génération se fasse pour les processeurs de type **ARM-HF**. Pour cela, placez-vous dans le répertoire de base de vos sources (là où est **configure**) et tapez la commande suivante :

```
$ ./configure -xplatform linux-arm-gnueabihf-g++ -nomake tests
```

-xplatform : spécifie le répertoire de configuration correspondant au type de processeur, ici linux-arm-gnueabihf-g++.

-nomake : empêche d'effectuer la phase finale des tests.

-prefix : (en option) spécifie le répertoire définitif de la librairie Qt (par exemple : -prefix /usr/local/Qt-5.5.1-ARMHF).

Au démarrage, on vous demande de choisir entre la version commerciale et la version open source, et d'autre part, comme d'habitude, vous devez accepter les conditions d'utilisation de la licence. Le temps de configuration est relativement long.

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@manu-N550JV ~/Documents/qt-everywhere-opensource-src-5.5.1 $ ./configure -xplatform
linux-arm-gnueabihf-g++ -nomake tests
+ cd qtbase
+ /home/manu/Documents/qt-everywhere-opensource-src-5.5.1/qtbase/configure -top-level -xp
latform linux-arm-gnueabihf-g++ -nomake tests
Which edition of Qt do you want to use ?

Type 'c' if you want to use the Commercial Edition.
Type 'o' if you want to use the Open Source Edition.

o


This is the Qt Open Source Edition.

You are licensed to use this software under the terms of
the Lesser GNU General Public License (LGPL) versions 2.1.
You are also licensed to use this software under the terms of
the GNU Lesser General Public License (LGPL) versions 3.

Type '3' to view the GNU Lesser General Public License version 3.
Type '2' to view the Lesser GNU General Public License version 2.1.
Type 'yes' to accept this license offer.
Type 'no' to decline this license offer.

Do you accept the terms of either license? yes
```

ATTENTION : tout cela ne peut fonctionner qu'à la condition d'avoir bien installé le paquetage du compilateur associé au processeur ARM (`sudo apt-get install crossbuild-essential-armhf`), ce que nous avons effectivement fait au tout départ.

Si tout s'est bien passé, vous devez obtenir les messages si dessous :

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
NOTE: Qt is using double for qreal on this system. This is binary incompatible against Qt 5.1.
Configure with '-qreal float' to create a build that is binary compatible with 5.1.
Info: creating super cache file /home/manu/Documents/qt-everywhere-opensource-src-5.5.1/.qmake.super

Qt is now configured for building. Just run 'make'.
Once everything is built, you must run 'make install'.
Qt will be installed into /usr/local/Qt-5.5.1

Prior to reconfiguration, make sure you remove any leftover files from
the previous build.

manu@manu-N550JV ~/Documents/qt-everywhere-opensource-src-5.5.1 $ make
```

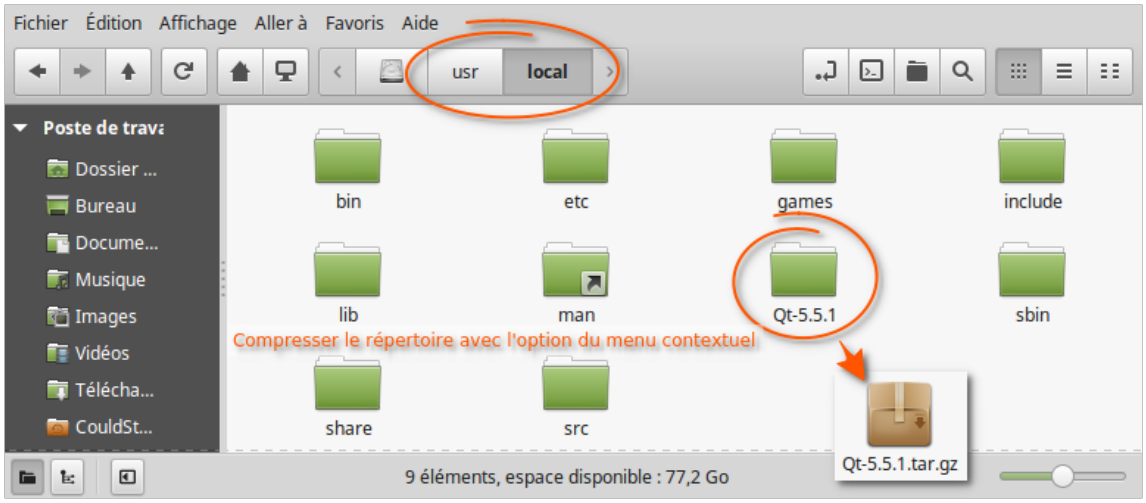
***Remarquez bien qu'à la fin de cette phase de configuration, on nous indique la procédure à suivre, c'est-à-dire de réaliser d'abord la phase de construction (création de tous les fichiers binaires et de toutes les librairies dynamiques) à l'aide de la commande* make *et ensuite la phase d'installation (placement de tous les fichiers binaires dans le répertoire définitif) avec la même commande, mais cette fois-ci avec l'option* install.**

Une fois que cette phase est (très très longue) terminée, nous pouvons exécuter l'installation qui cette fois-ci est beaucoup plus rapide puisque les librairies sont déjà construites.

Cette phase consiste juste à placer l'ensemble des bibliothèques à l'endroit par défaut avec comme nom de répertoire la version installée, savoir dans notre cas « /usr/local/QT-5.5.1 ». Vu sa localisation, vous devez exécuter la commande en mode administrateur :

```
$ sudo make install
```

Une fois que l'installation s'est terminée correctement, vous pouvez contrôler le résultat dans le système de fichier. L'idéal maintenant est de fabriquer une archive qui vous servira à la déployer sur l'ensemble des postes de développement, mais également sur l'ensemble des cartes **Raspberry**. Attention, vous devez décompacter l'archive exactement au même endroit que sa fabrication, savoir dans le dossier « **/usr/local** ».

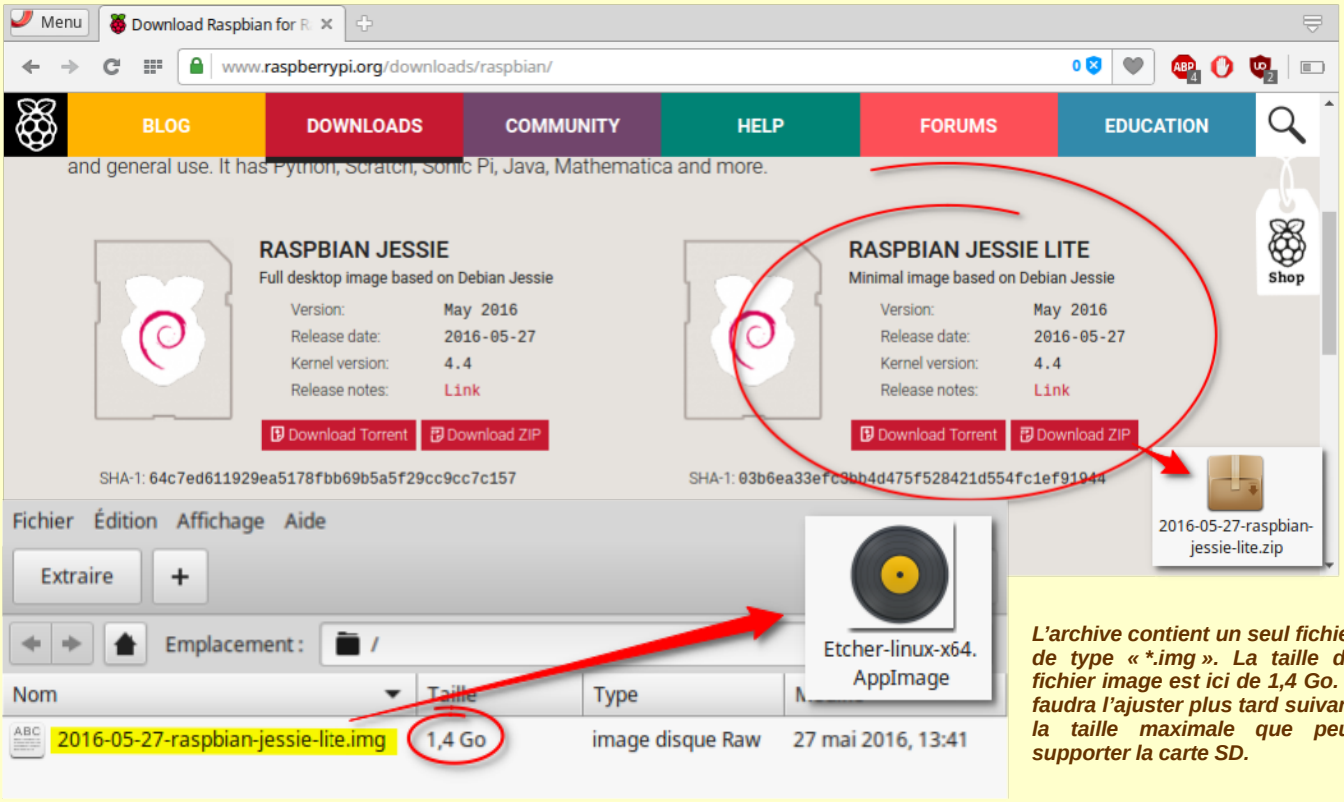


Création de la carte SD pour l'informatique embarquée de type Raspberry

Nous allons maintenant nous intéresser au système embarqué **Raspberry** qui servira de cible pour nos applications et qui doit intégrer à terme la librairie **Qt** que nous venons tout juste de construire. Nous en profitons pour montrer toutes les phases nécessaires, depuis l'installation du système d'exploitation, jusqu'à la forme définitive d'utilisation pour un système de « cross-développement ».

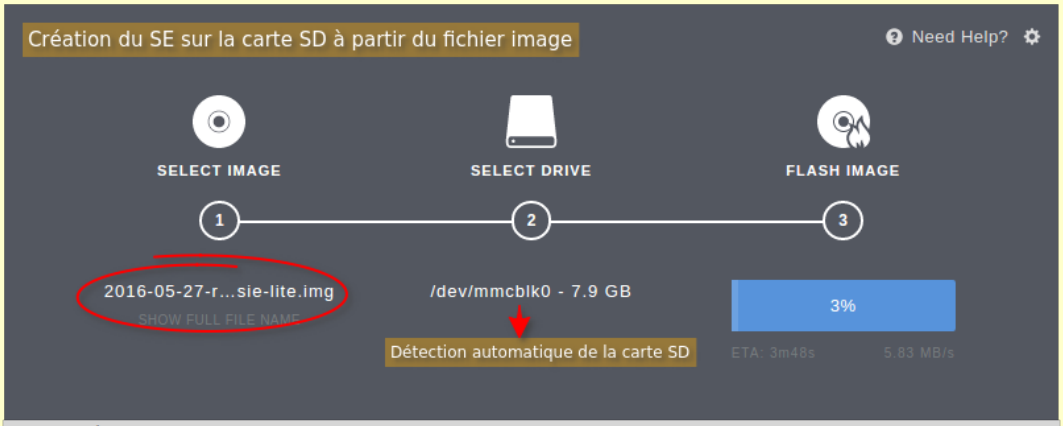
La première phase consiste à bien choisir le système d'exploitation. Il me semble évident, puisqu'il s'agit d'une informatique embarqué, de prendre une version sans bureau. Le système est alors beaucoup plus performant, il prend moins de place, et la plupart du temps, sur ces systèmes là, nous n'avons pas besoin de clavier ni d'écran. Le seul soucis sera au préalable de bien régler la communication réseau afin de les administrer à distance par « ssh ». Le système qui me parait le plus adapté est la « Raspbian Jessie Lite ».

Pour la dernière version, prenez le site officiel ci-dessous, sinon celui-ci : http://domoticx.com/raspberry-pi-sd-image-raspbian-linux-os/

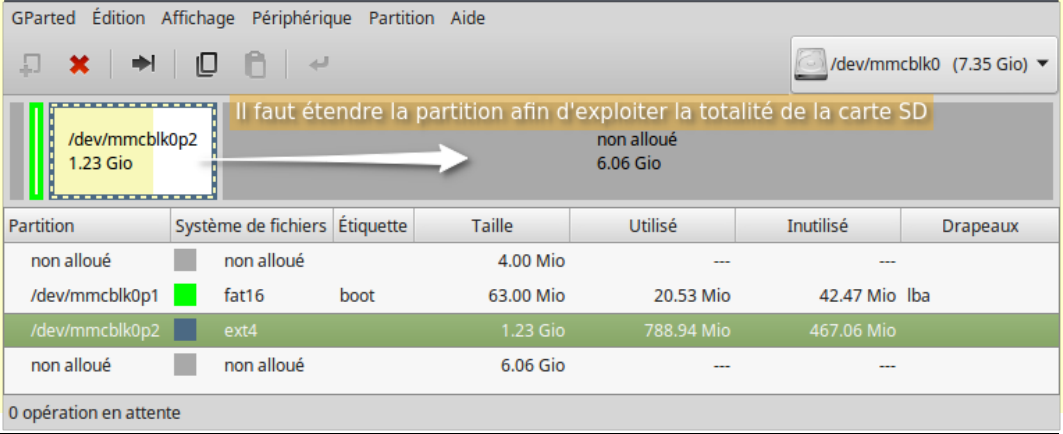


L'archive contient un seul fichier de type « *.img ». La taille du fichier image est ici de 1,4 Go. Il faudra l'ajuster plus tard suivant la taille maximale que peut supporter la carte SD.

Grâce à ce fichier et à l'utilitaire Etcher, nous pouvons l'installer directement sur une carte SD et nous avons ainsi un système d'exploitation de type Debian (Raspbian Lite) prêt à l'emploi.



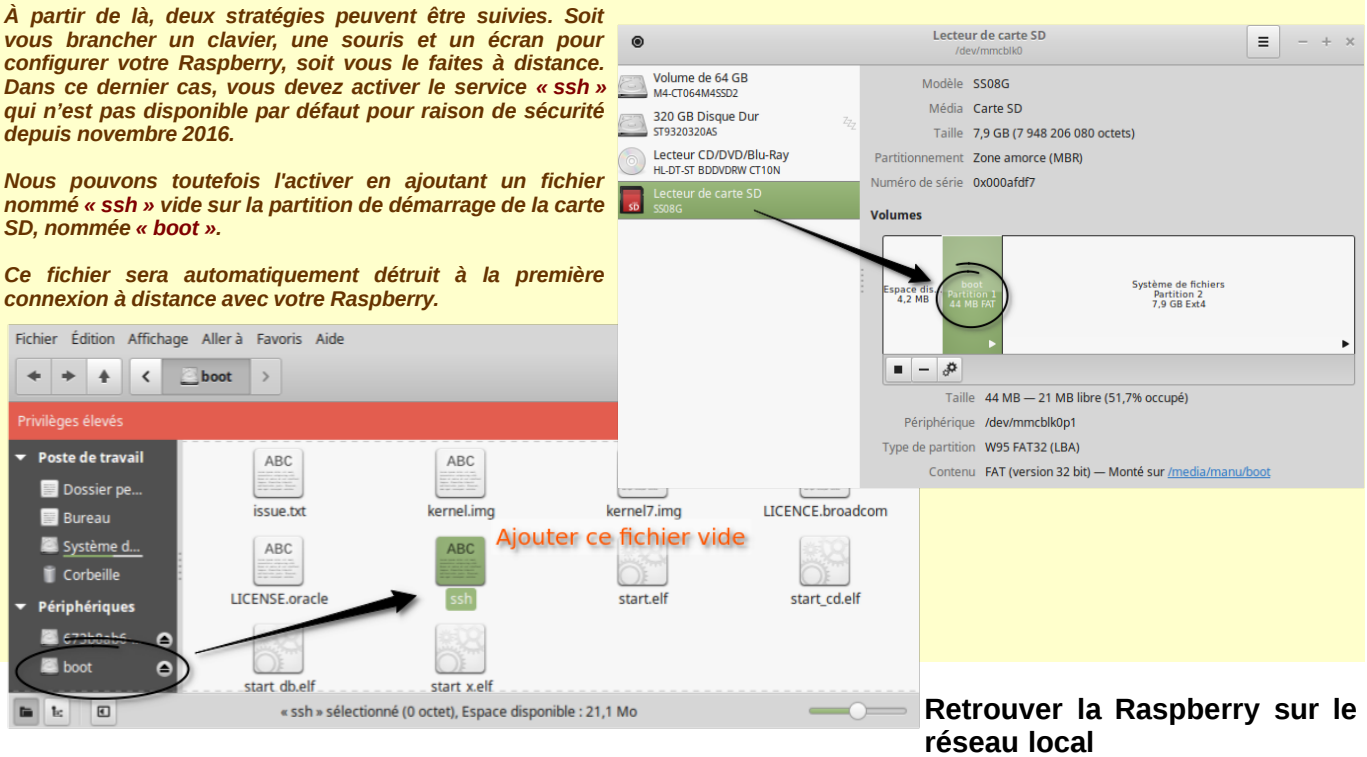
Avant de placer cette carte sur votre Raspberry, une chose à laquelle vous devez penser est de faire en sorte que la totalité de la carte SD soit exploitable. Seul 1,4 Go est opérationnel, alors que la carte SD possède certainement une capacité bien supérieure. Aidez-vous alors de l'outil GParted pour ajuster correctement cette partition.



À partir de là, deux stratégies peuvent être suivies. Soit vous brancher un clavier, une souris et un écran pour configurer votre Raspberry, soit vous le faites à distance. Dans ce dernier cas, vous devez activer le service « ssh » qui n'est pas disponible par défaut pour raison de sécurité depuis novembre 2016.

Nous pouvons toutefois l'activer en ajoutant un fichier nommé « ssh » vide sur la partition de démarrage de la carte SD, nommée « boot ».

Ce fichier sera automatiquement détruit à la première connexion à distance avec votre Raspberry.



Retrouver la Raspberry sur le réseau local

Votre Raspberry est maintenant prête à être exploitée. Vous pouvez placer votre carte SD dans le slot prévu à cet effet avec le système minimum que nous venons de construire. Brancher votre Raspberry. Pour l'exploiter, nous devons localiser votre système embarqué sur le réseau afin que nous puissions l'atteindre et finaliser toutes les autres installations.

*Pour cela, le plus simple est peut-être d'utiliser la commande « **nmap** » qui possède pas mal d'options. Je vais juste vous montrer deux exemples d'écriture qui vont résoudre ce qui nous intéresse ici. Enfin, pour avoir toutes les informations, il vaut mieux se placer en mode administrateur. Si vous ne l'avez pas sur votre système. Il suffit de l'installer :* `$ sudo apt-get install nmap`

`$ sudo nmap -sP 172.16.20.0-255` *// scanne une plage d'adresses en récupérant uniquement les ordinateurs connectés sans d'autres spécifications supplémentaires. Si vous ne placez cette option -sP, pour chaque ordinateur découvert, vous aurez beaucoup d'informations rajoutées qui peuvent faire perdre du temps si vous possédez beaucoup d'ordinateurs sur cette plage là.*

```
Fichier Édition Affichage Rechercher Terminal Aide
MAC Address: 8C:DC:D4:50:88:6A (Hewlett Packard)
Nmap scan report for 172.16.20.64
Host is up (-0.100s latency).
MAC Address: 00:19:B9:2C:41:A6 (Dell)
Nmap scan report for 172.16.20.66
Host is up (-0.098s latency).
MAC Address: B8:27:EB:78:6E:CD (Raspberry Pi Foundation)
Nmap scan report for 172.16.20.1
Host is up.
Nmap scan report for 172.16.20.31
Host is up.
Nmap done: 256 IP addresses (18 hosts up) scanned in 1.94 seconds
manu@manu-N550JV ~ $ sudo nmap 172.16.20.66

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-27 09:53 CEST
Nmap scan report for 172.16.20.66
Host is up (0.00071s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: B8:27:EB:78:6E:CD (Raspberry Pi Foundation)
```

Le service ssh est bien opérationnel

*Voici ce que nous obtenons à l'issue de cette commande. 18 systèmes numériques connectés ont été découverts. Nous trouvons la Raspberry à l'adresse **172.16.20.66**. Maintenant que nous connaissons son adresse, il est possible de connaître toutes les informations associées à cette Raspberry en relançant la commande uniquement sur cette adresse et sans option supplémentaire. Vous découvrez alors que le service **ssh** est bien en activité.*

Configuration de base de la Raspberry

Vu que nous risquons d'avoir plusieurs Raspberry sur le même réseau, il serait judicieux de changer son nom d'hôte. Pour cela, il suffit de modifier les fichiers de configuration « ***/etc/hostname*** » et « ***/etc/hosts*** ».

*Pour lancer cette configuration, il faut d'abord se connecter à la raspberry par « ssh », sachant que l'identifiant s'appelle « **pi** » par défaut et que son mot de passe est « **raspberry** » .*

`$ ssh pi@172.16.20.66` *// la première fois tapez 'yes' et saisissez votre mot de passe.*

```
Fichier Édition Affichage Rechercher Terminal Aide
  GNU nano 2.2.6        File: /etc/hostname          Modified

raspdev

^G Get Hel^O WriteOu^R Read Fi^Y Prev Pa^K Cut Tex^C Cur Pos
^X Exit    ^J Justify^W Where I^V Next Pa^U UnCut T^T To Spell
```

Le nouveau nom de votre raspberry

Attention, ces changements ne peuvent s'effectuer qu'en mode administrateur.

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
  GNU nano 2.2.6        File: /etc/hosts              Modified

127.0.0.1       raspdev          <- Nouveau nom réseau en ipv4
::1             localhost ip6-localhost ip6-loopback

^G Get Hel^O WriteOu^R Read Fi^Y Prev Pa^K Cut Tex^C Cur Pos
^X Exit    ^J Justify^W Where I^V Next Pa^U UnCut T^T To Spell
```

Une fois que ces deux modifications ont été exécutés, je vous invite à « rebooter » votre système pour que ces changements soient pris en compte.

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@manu-HPE-120fr ~ $ ssh pi@raspdev
The authenticity of host 'raspdev (192.168.1.26)' can't be esta
blished.
ECDSA key fingerprint is SHA256:I+lIFQwXuCwx4n6QqjcXe804aTnjfvE
D1Fwgd24Qlgo.    <- Plus besoin de connaître l'adresse IP
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'raspdev' (ECDSA) to the list of kno
wn hosts.
pi@raspdev's password:

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the exte
nt
permitted by applicable law.
Last login: Tue Sep 27 13:29:17 2016 from manu-hpe-120fr.home
pi@raspdev:~ $
```

L'invite du système a également changé

Maintenant nous pouvons nous connecter par « ssh », non plus avec l'adresse IP, mais directement avec le nouveau nom de votre Raspberry.

Je vous suggère de créez un répertoire particulier, « projets » par exemple, dans lequel nous placerons tous nos différents développements.

Mise en place de l'archive Qt-5.5.1.tar.gz dans la Raspberry

Souvenez-vous que nous avons généré une archive de la librairie Qt exploitable pour tous les systèmes embarqués possédant un processeur de type **ARM-HF.** Attention, nous devons décompacter l'archive dans la Raspberry exactement au même endroit que nous l'avons fabriqué, savoir dans le dossier « **/usr/local** ».

*Dans un premier temps, nous devons déployer l'archive « **Qt-5.5.1.tar.gz** ». Nous pourrons ensuite la décompresser en mode administrateur vers le bon répertoire requis. Ce déploiement se fera par « ssh » au travers de la commande « scp ».*

*La commande **scp** permet de copier un fichier ou un répertoire **(-r)** du client vers le serveur ou du serveur vers le client. Le chemin du serveur peut être indiqué en absolu (**/home/btssnir/Public** par exemple) ou relatif à partir du répertoire de base. Pour utiliser **scp**, vous devez connaître l'arborescence exacte des répertoires de la machine distante.*

```
$ scp fichier login@serveur:chemin  // copie un fichier particulier sur une machine distante dans le dossier spécifié par chemin

$ scp -r répertoire login@serveur:chemin  // copie d'un répertoire entier avec les mêmes critères
```

Voici la commande que nous pouvons soumettre pour déployer cette archive dans le répertoire courant de notre nouvelle Raspberry :

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@manu-HPE-120fr ~/Documents $ scp Qt-5.5.1.tar.gz pi@raspdev:/home/pi
pi@raspdev's password:    <- Saisissez votre mot de passe
Qt-5.5.1.tar.gz                               100%   48MB  11.9MB/s   00:04
                                                  <- Progression en temps réel
manu@manu-HPE-120fr ~/Documents $
```

*Une fois que l'archive est bien dans la Raspberry, nous pouvons nous connecter par **ssh**, nous placer ensuite dans le répertoire où se situe l'archive et saisir la commande de décompression :*

```
$ sudo tar xzvf Qt-5.5.1.tar.gz -C /usr/local  // sudo est indispensable vu l'endroit où nous effectuons la décompression
```

La commande « tar » est un outil très puissant pour la manipulation d'archive, dont voici quelques options :
- *c : crée l'archive*
- *x : extrait l'archive*
- *f : utilise le fichier donné en paramètre*
- *v : active le mode « verbeux »*
- *z : ajoute la compression Gzip « *.gz »*
- *J : utile pour la (dé)compression « *.xz »*
- *-C : spécifie le répertoire de localisation de la décompression*

Une fois que l'archive est bien décompressée, dans le bon répertoire, exactement au même endroit que l'ordinateur de développement, nous pouvons dès lors la supprimer. Elle ne nous sert plus à rien.

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@manu-HPE-120fr ~ $ ssh pi@raspdev
pi@raspdev's password:

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Sep 27 17:31:14 2016 from manu-hpe-120fr.home
pi@raspdev:~ $ cd projets/
pi@raspdev:~/projets $ ls
Qt-5.5.1.tar.gz
pi@raspdev:~/projets $ sudo tar xzf Qt-5.5.1.tar.gz -C /usr/local
pi@raspdev:~/projets $ ls /usr/local/
bin  etc  games  include  lib  man  Qt-5.5.1  sbin  share  src
pi@raspdev:~/projets $ rm Qt-5.5.1.tar.gz
pi@raspdev:~/projets $ ls
pi@raspdev:~/projets $
```

Commande à exécuter pour décompresser l'archive au bon endroit

La librairie Qt a bien été placée dans le bon répertoire

À l'issu de tous ces réglages et de toutes ces installations, notre Raspberry est maintenant prête à être utilisée depuis un poste de développement. Nous pouvons confectionner autant de programme que nous désirons, soit uniquement en écrivant avec du C++ basique, soit en utilisant en plus la bibliothèque Qt pour faire, par exemple de la communication réseau par « websocket ».

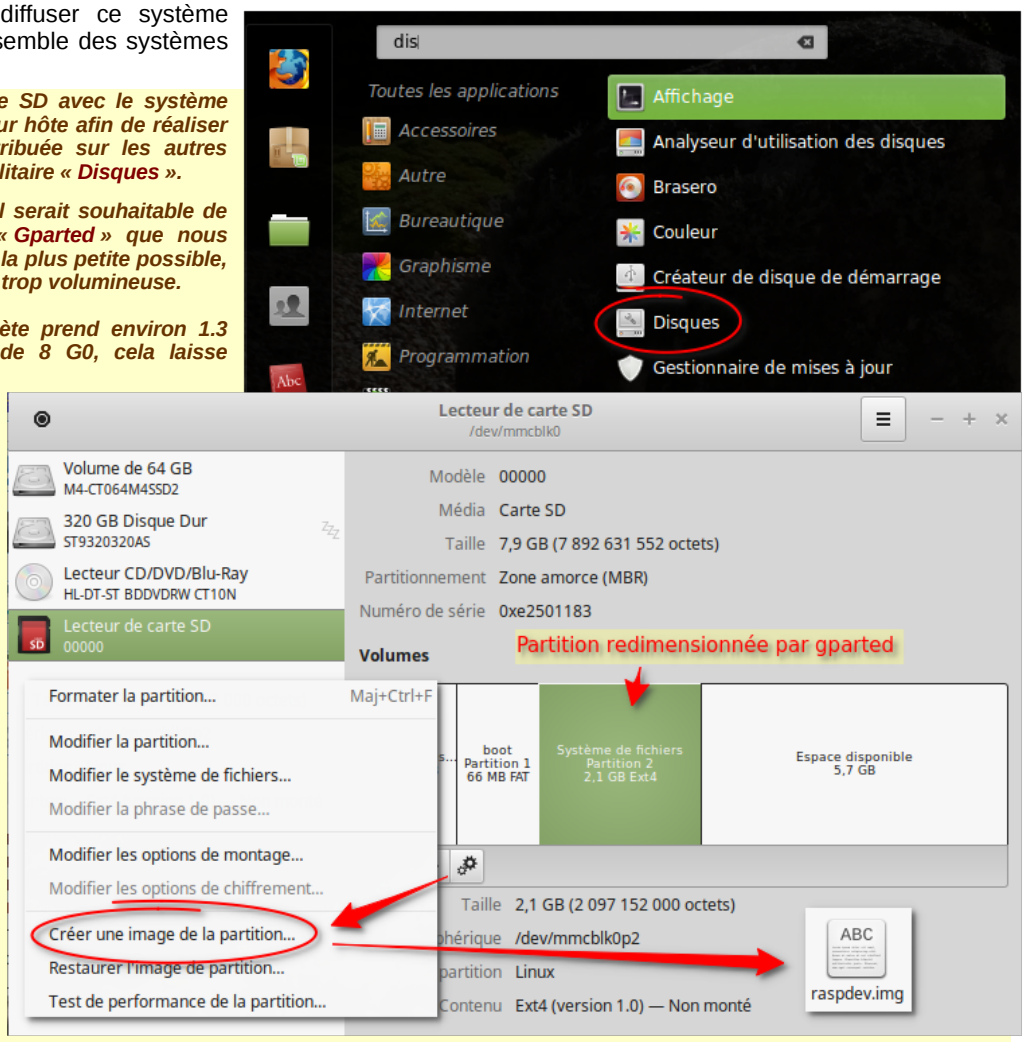
L'idéal, maintenant, est de faire une image de ce système d'exploitation parfaitement réglé, afin de le retrouver intact si jamais, au cours de nos différentes expériences, nous le détruisons par inadvertance. Surtout, cette image sera très utile pour toutes les Raspberry présentes sur le réseau. Avec cette technique, une seule installation complète avec les différents réglages sont suffisants pour diffuser ce système d'exploitation spécifique sur l'ensemble des systèmes embarqués de type Raspberry.

Dans ce cas, reprenez votre carte SD avec le système complet et placez la sur l'ordinateur hôte afin de réaliser cette image qui pourra être distribuée sur les autres Raspberry. Nous allons utiliser l'utilitaire « Disques ».

Attention, avant de faire l'image, il serait souhaitable de réduire la partition avec l'outil « Gparted » que nous avons déjà utilisé, pour quelle soit la plus petite possible, sinon notre image serait beaucoup trop volumineuse.

Effectivement, l'installation complète prend environ 1.3 Go et si vous avez une carte de 8 G0, cela laisse beaucoup d'espace libre. Donc, pensez bien à réduire cette partition en laissant toutefois une marge de 500 Mo pour que cela fonctionne, sinon la réduction ne se fait pas.

Une fois que l'image est faite, vous pouvez la reproduire pour toutes les cartes SD des autres Raspberry. Après coup, penser encore une fois à redimensionner la partition pour quelle soit la plus grande possible pour chacune des cartes.

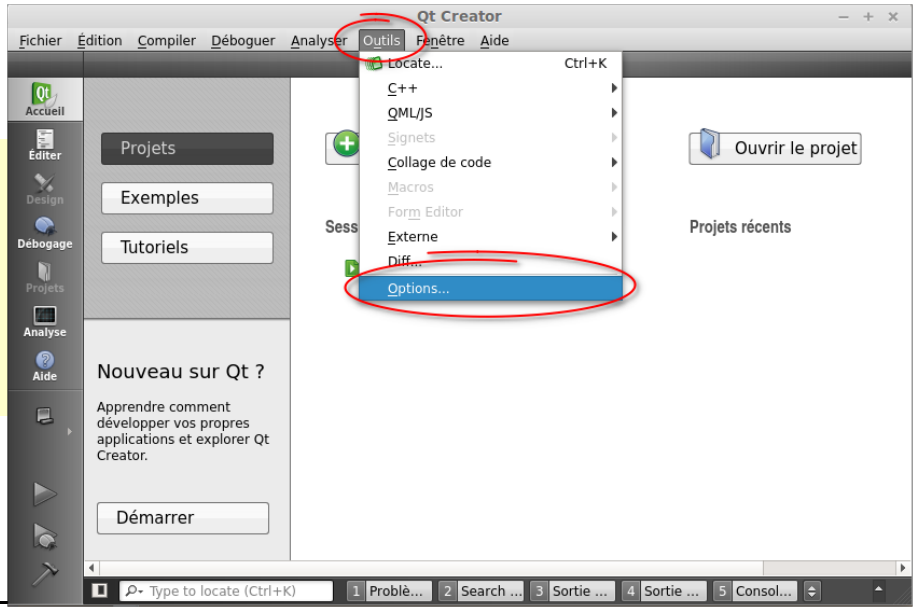


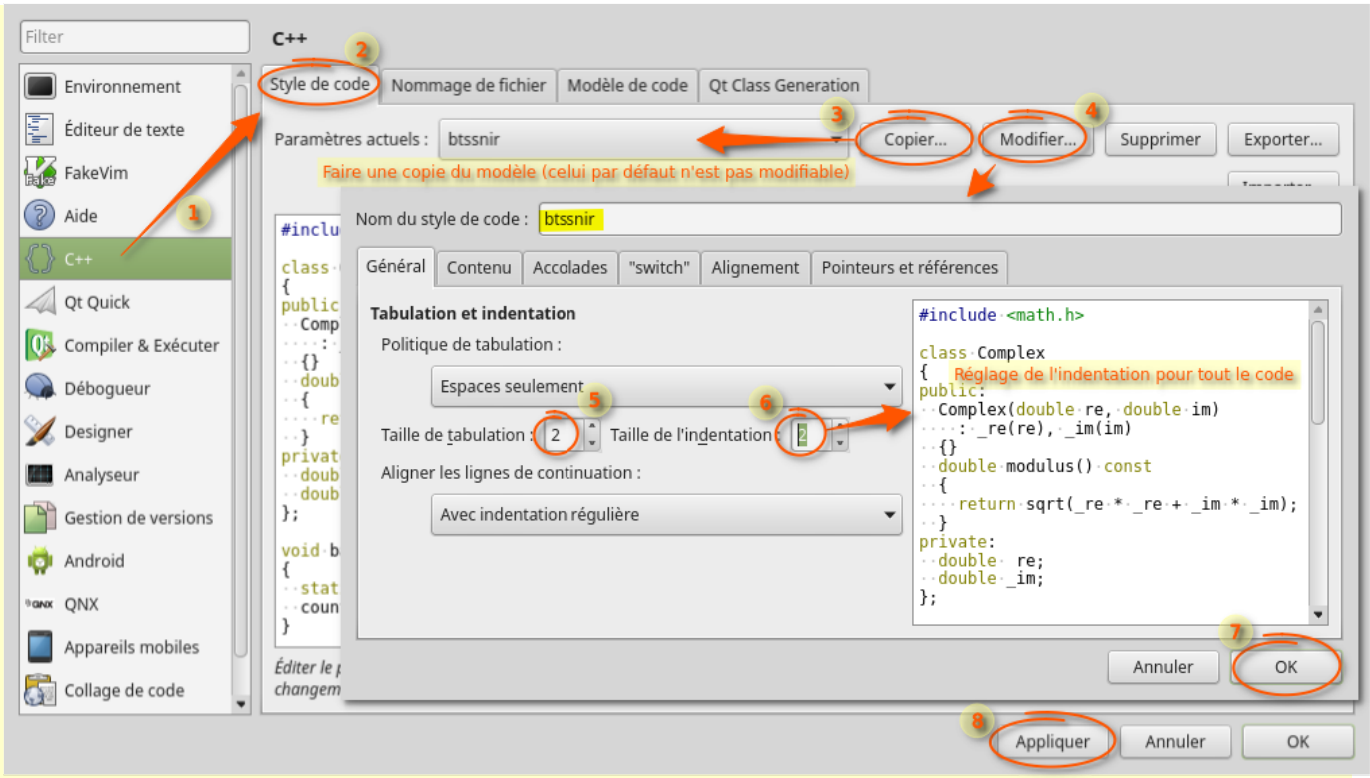
Configuration de Qt Creator pour réaliser le cross-developpement

Nous disposons maintenant de tous les éléments nécessaires qui vont nous permettre de réaliser des programmes, avec ou sans la librairie Qt, depuis notre poste de développement pour être exécuté sur une cible de type informatique embarquée, c'est ce qui s'appelle faire du « cross-développement ». QtCreator permet de le faire simplement, vous éditer votre source et le déploiement ainsi que l'exécution se fera automatiquement sur la cible choisie. Ce chapitre nous montre comment régler QtCreator pour atteindre cet objectif.

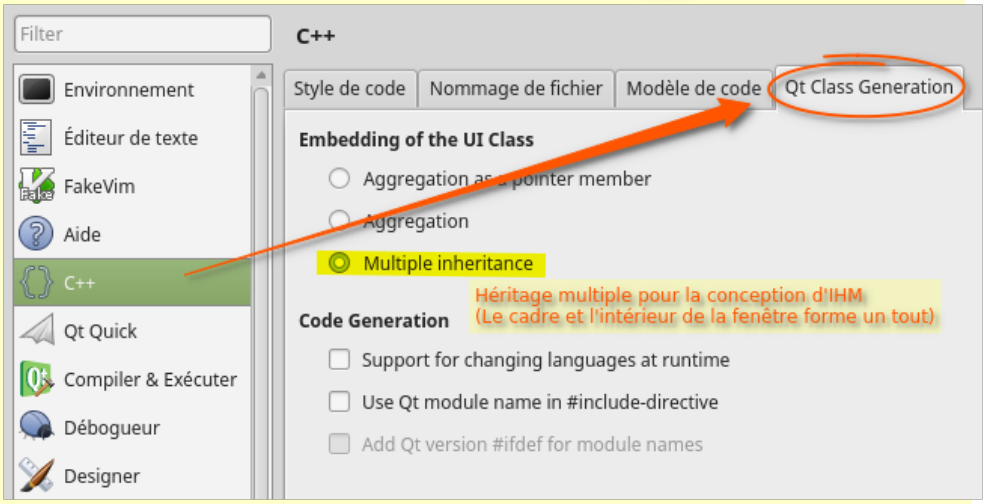
Tout se passe au travers de la rubrique « Options » du menu « Outils » .

Avant de se focaliser sur les réglages pour la compilation distante, je vous propose dans un premier temps de faire des réglages plus généraux, d'une part pour régler l'indentation automatique, ensuite pour avoir l'héritage multiple lors de la conception d'IHM, et enfin pour faire en sorte que les fichiers compilés soient dans le même répertoire que les sources .

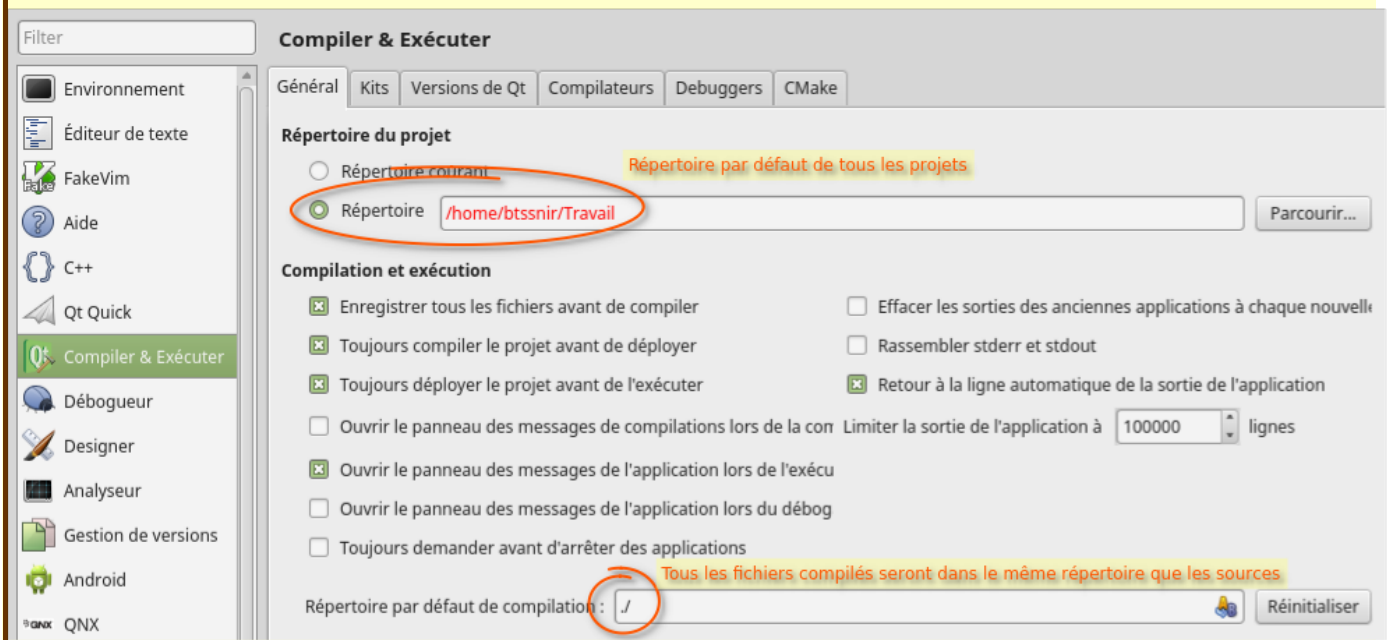




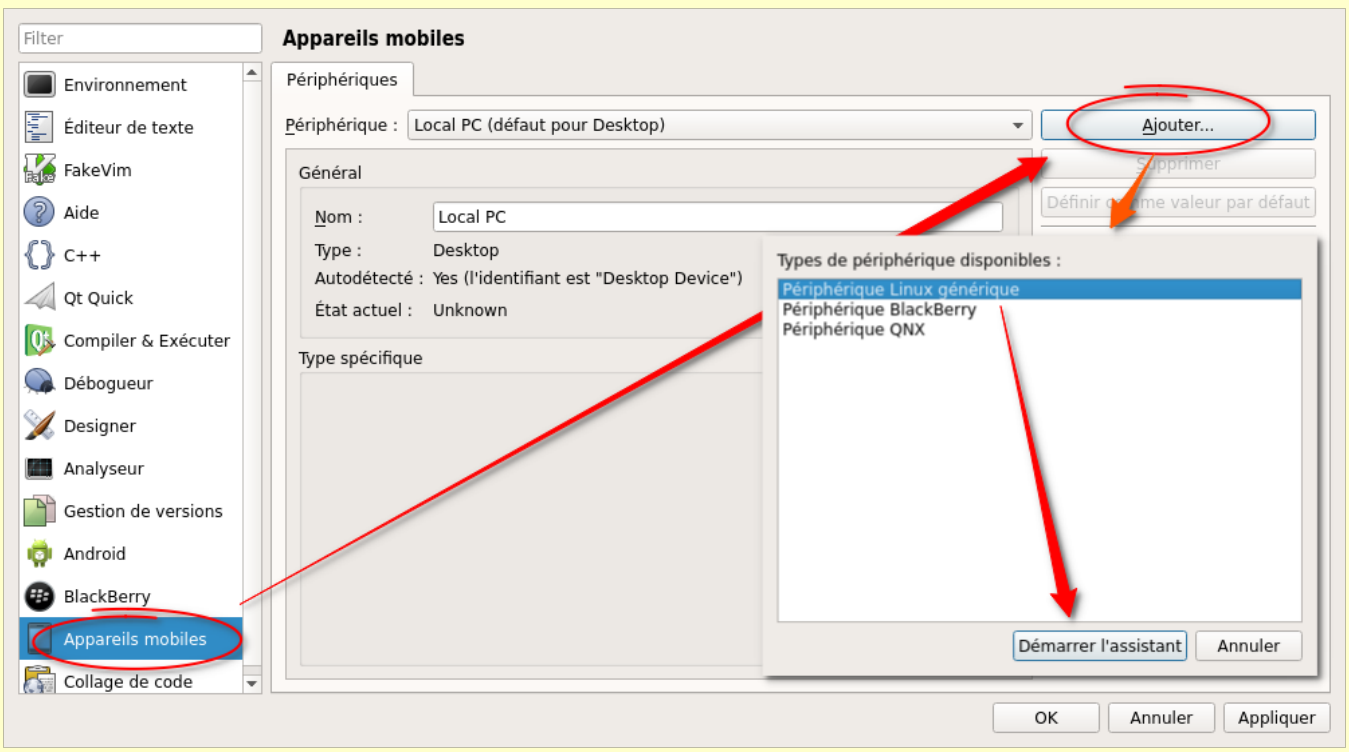
Les premiers réglages se situent dans la rubrique « C++ ».



Ensuite dans la rubrique « Compiler et Exécuter ».

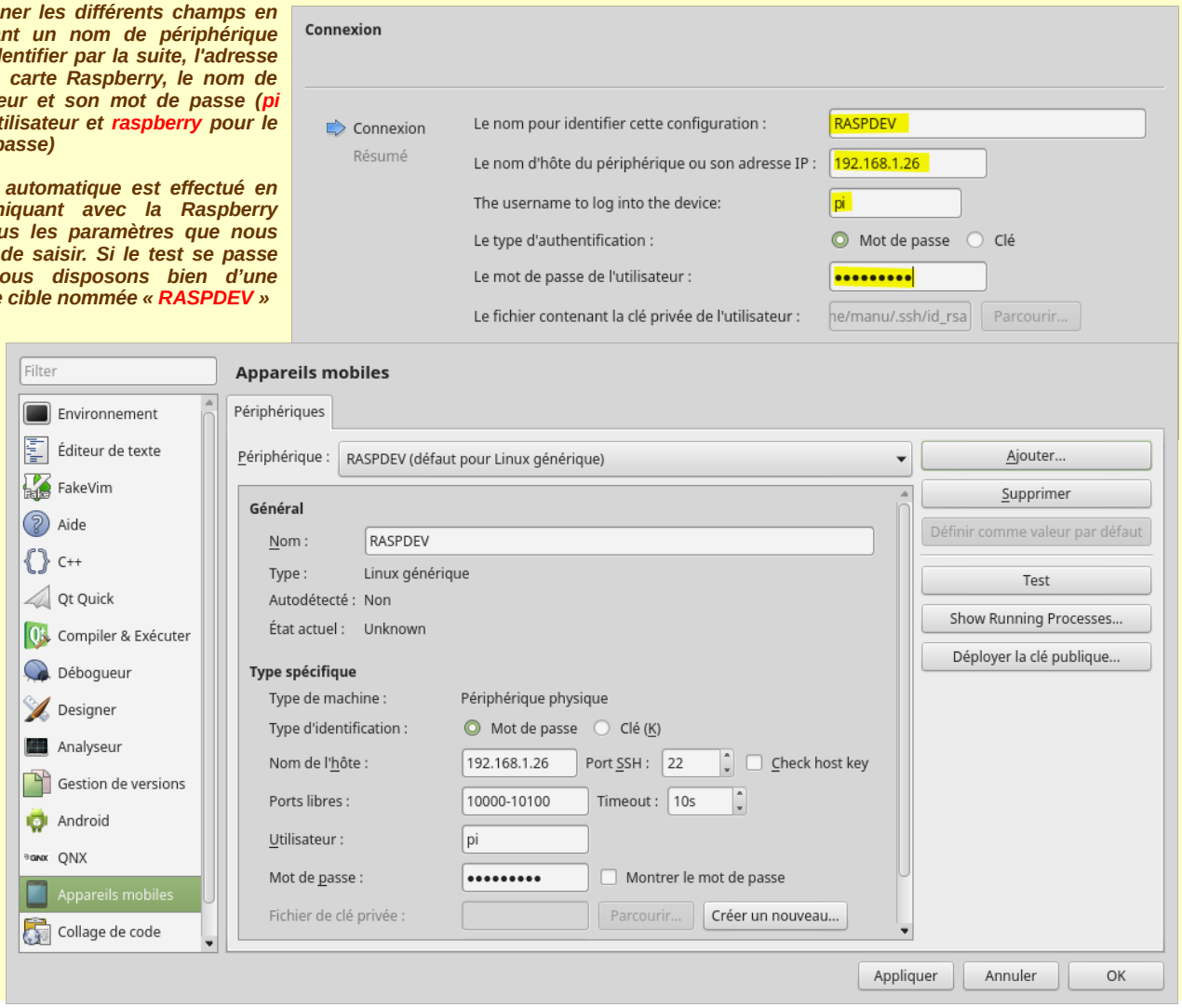


Tous les réglages suivants se consacrent maintenant pour la mise en œuvre de « cross-compilation ». La première phase consiste à créer ses différentes cibles, si vous avez plusieurs cartes Raspberry. Pour cela, placez-vous dans la rubrique « **Appareils mobiles** » et cliquez sur le bouton « **Ajouter...** ». On vous demande alors de choisir votre type de périphérique. Choisissez la première option (la Raspberry est un système Linux) et démarrer l'assistant.



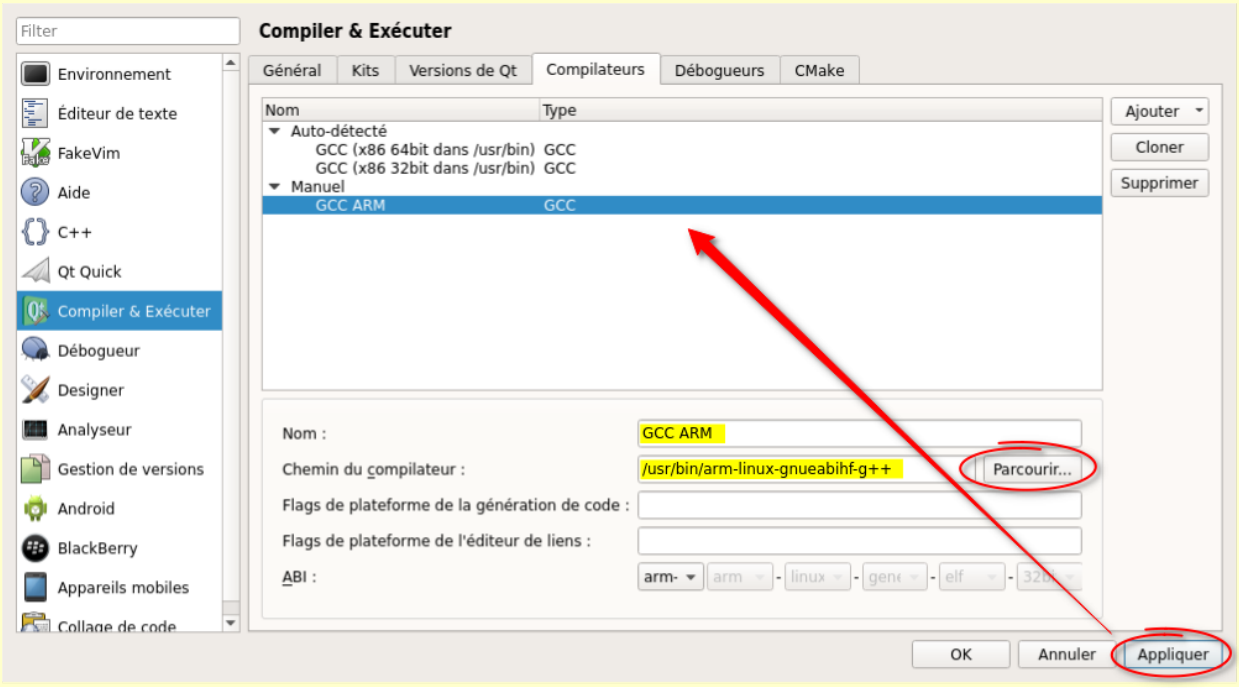
Renseigner les différents champs en proposant un nom de périphérique pour l'identifier par la suite, l'adresse IP de la carte Raspberry, le nom de l'utilisateur et son mot de passe (pi pour l'utilisateur et raspberry pour le mot de passe)

Un test automatique est effectué en communiquant avec la Raspberry avec tous les paramètres que nous venons de saisir. Si le test se passe bien, nous disposons bien d'une nouvelle cible nommée « RASPDEV »



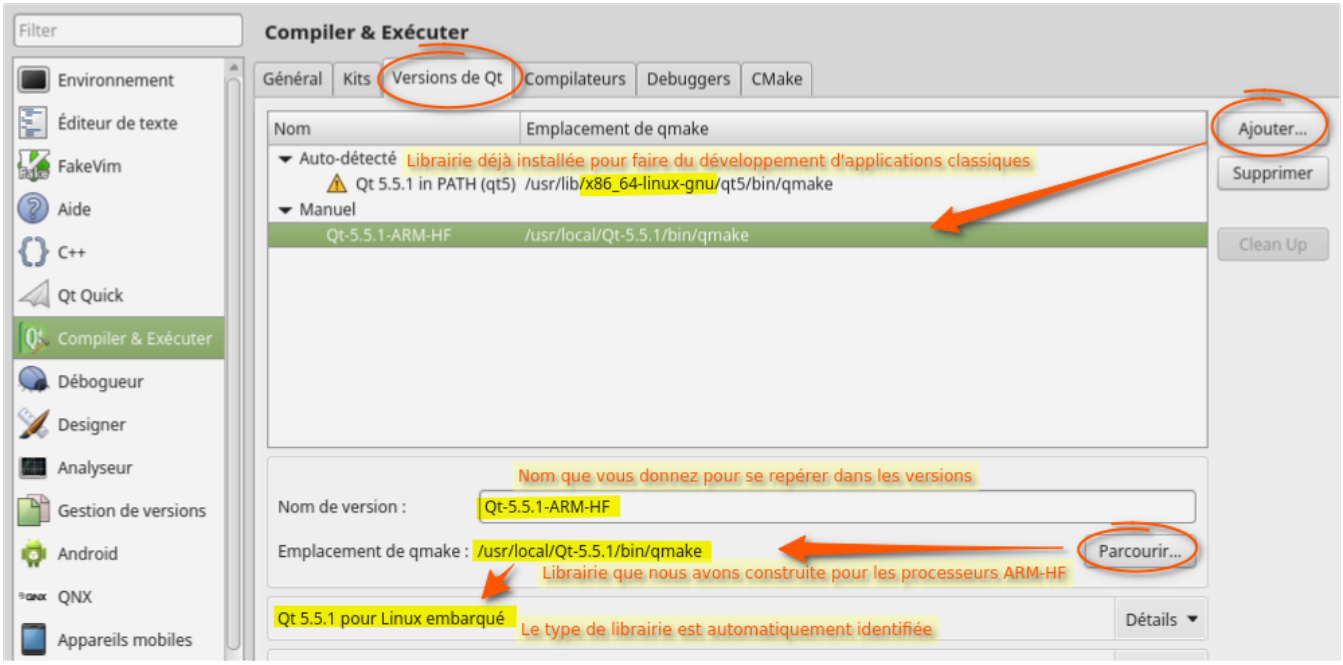
Compilateur ARM dans QtCreator. Pour réaliser notre « cross-développement », il est bien entendu nécessaire que QtCreator prenne en compte le compilateur ARM, sinon cela ne sert à rien. Pour cela, nous devons nous placer dans la rubrique « Compiler & Exécuter », cliquer ensuite sur le bouton « Ajouter » et choisir un compilateur de type « **GCC** » ce qui est le cas pour notre compilateur ARM.

Vous pouvez dès lors compléter les deux champs principaux, d'abord donner un nom à votre compilateur et le localiser à l'aide du bouton « Parcourir... » . Là aussi, pour que votre nouveau compilateur soit pris en compte, validez avec le bouton « Appliquer ».



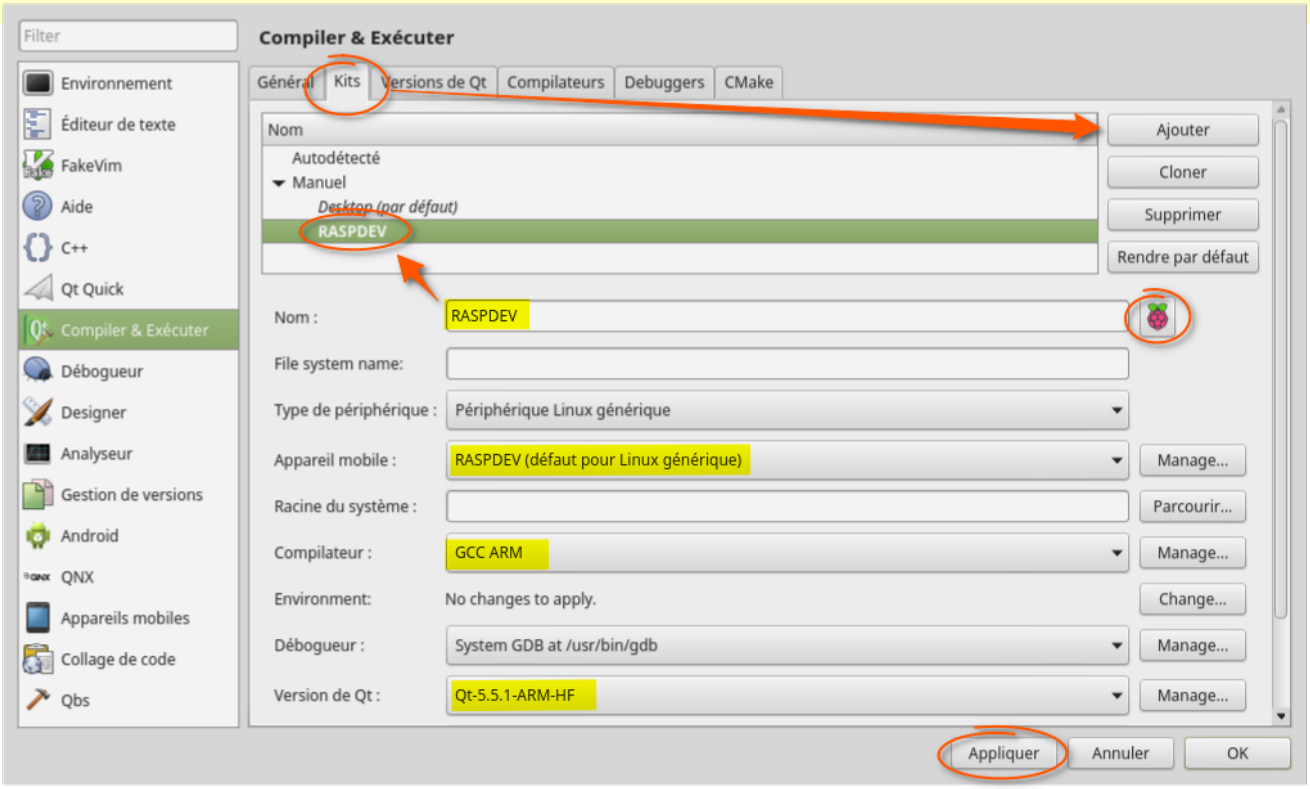
Version de Qt pour processeur ARM : La version de Qt est importante lorsque vous devez développer des applications avec le cœur de Qt et certains modules supplémentaires comme par exemple la gestion du réseau. Vous devez donc prendre une version de Qt compatible pour les processeurs ARM, celle là même que nous avons construite précédemment.

Pour cela, toujours dans la rubrique « Compiler & Exécuter », cliquez sur l'onglet « Versions de Qt » et cliquez sur le bouton « Ajouter... » . La boîte de dialogue de sélecteur de fichier apparaît alors, il suffit d'explorer le système de fichier pour atteindre l'utilitaire « qmake » présent dans le répertoire « bin ». N'oubliez pas de valider en cliquant sur le bouton « Appliquer ».



Mise en place d'un nouveau Kit de développement associé à la Raspberry : Nous possédons maintenant tous les ingrédients pour la fabrication d'un kit de développement personnalisé, qui va nous permettre dès le départ de choisir un type de projet dont la cible est directement la Raspberry avec tout ce qu'il faut pour que la génération et le déploiement se fasse automatiquement.

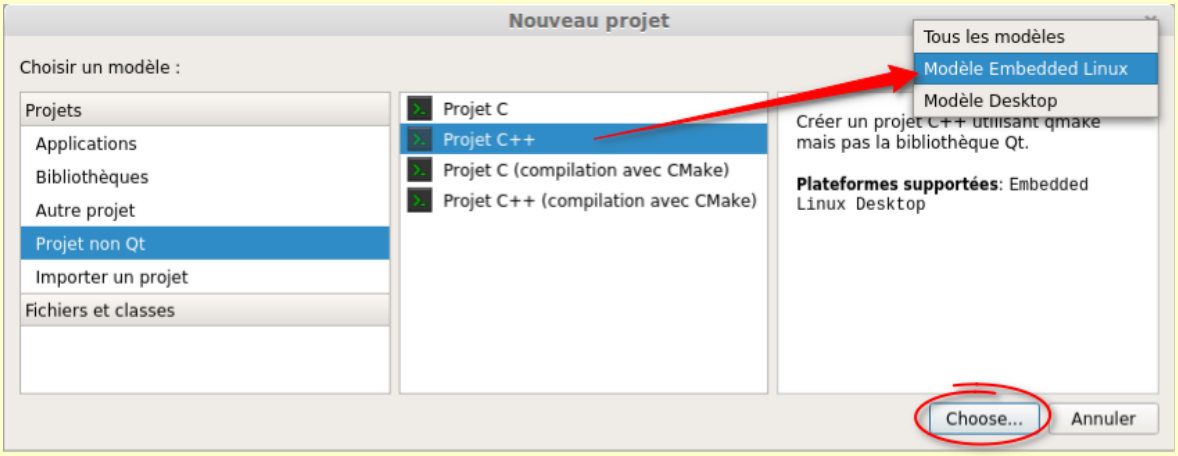
Pour cela, toujours dans la rubrique « Compiler & Exécuter », cliquez sur l'onglet « Kits » et cliquez sur le bouton « Ajouter ». Il suffit de sélectionner les différents éléments que nous venons de configurer : l'appareil mobile, le compilateur et la version de Qt. Donnez un nom à votre nouveau kit de développement. N'oubliez pas de valider en cliquant sur le bouton « Appliquer ».



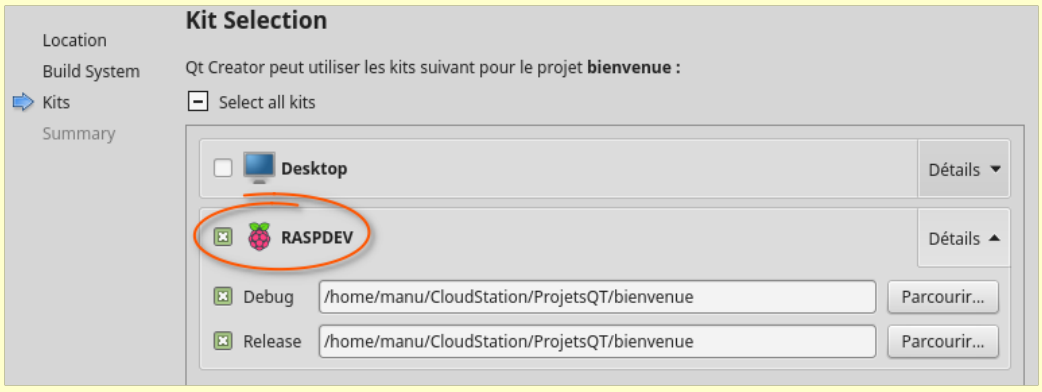
Premier projet pour la Raspberry par « cross-développement »

Tout est prêt. Nous pouvons réaliser notre premier projet en utilisant le kit de développement que nous venons d'élaborer. Ce premier projet est extrêmement simple, nous allons juste mettre en œuvre un programme qui souhaite la bienvenue. Dans ce cas de figure, nous n'utiliserons pas la bibliothèque Qt. Il s'agit simplement d'un programme C++ basique.

***Choix du type de projet** : Démarrer Qt Creator et demander à faire un nouveau projet. Choisissez donc un « Projet non Qt ». Par défaut, au départ les projets sont prévus pour réaliser du développement pour des PC de bureau « Modèle Desktop ». Ici, vu le kit que nous venons de mettre en place, nous devons choisir un autre modèle prévu pour les systèmes embarqués linux « Modèle Embedded Linux ».*

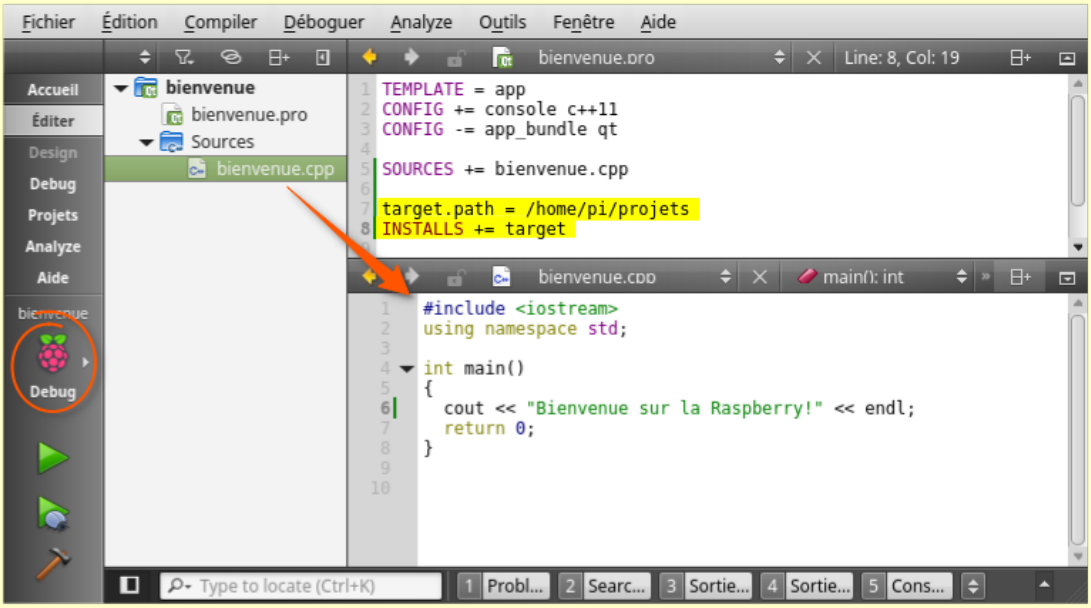


Dès que vous avez validé et donné un nom à votre projet, vous voyez apparaître votre kit de développement « RASPDEV »

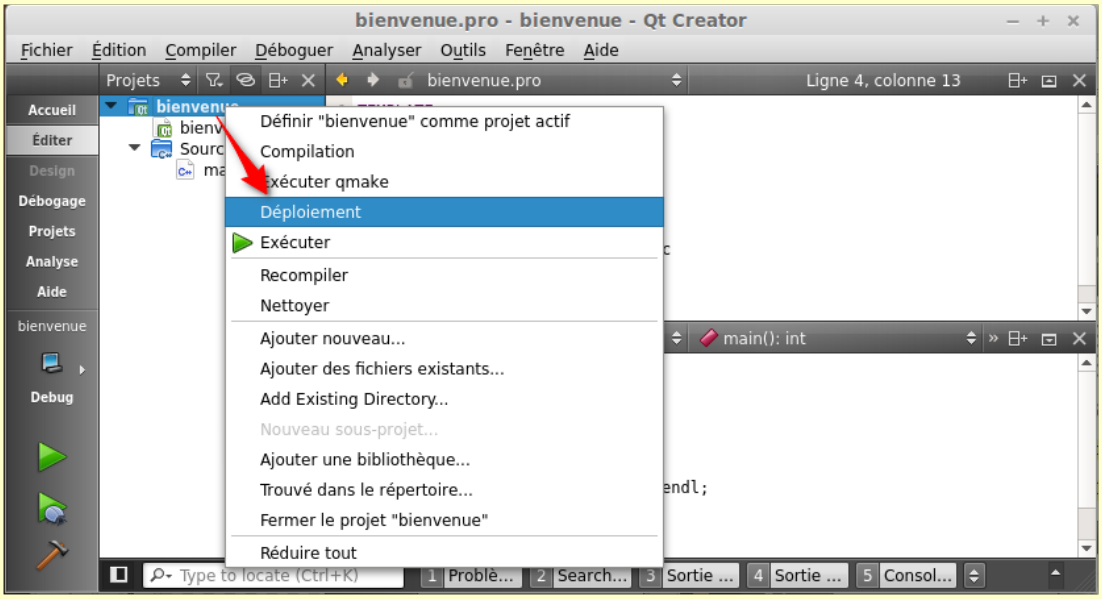


Édition, déploiement et exécution du projet : Une fois que vous avez validé le projet, vous vous retrouvez avec deux fichier, le fichier principal et le fichier de projet.

Pour que le déploiement s'effectue dans de bonne condition, il est nécessaire de préciser le répertoire d'accueil de votre exécutable, le dossier partagé de la Raspberry. Grâce aux deux lignes supplémentaires surlignées ci-dessous, nous indiquons la cible de l'exécutable dans le fichier de projet.

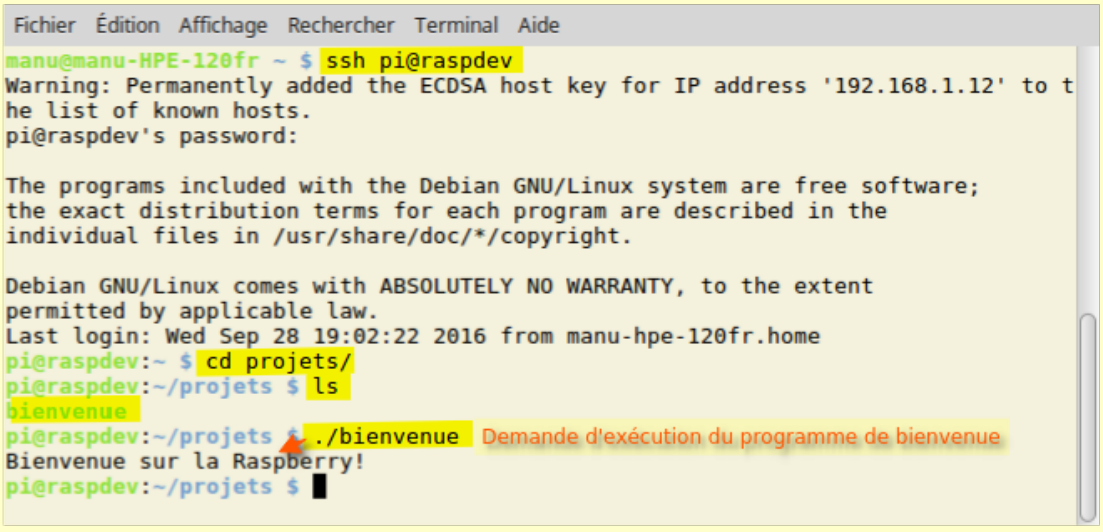


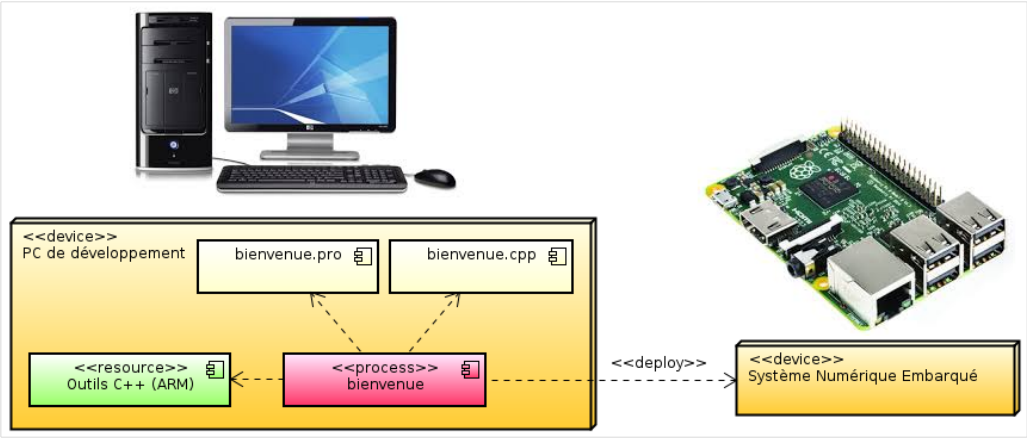
Il ne vous reste plus qu'à compiler votre programme et à déployer votre exécutable :



Si tout s'est bien passé, vous pouvez maintenant travailler sur la Raspberry à distance à l'aide d'une communication « ssh ».

L'exécutable est un projet en mode console. Une fois que la communication est établie, vous pouvez lancer votre programme :



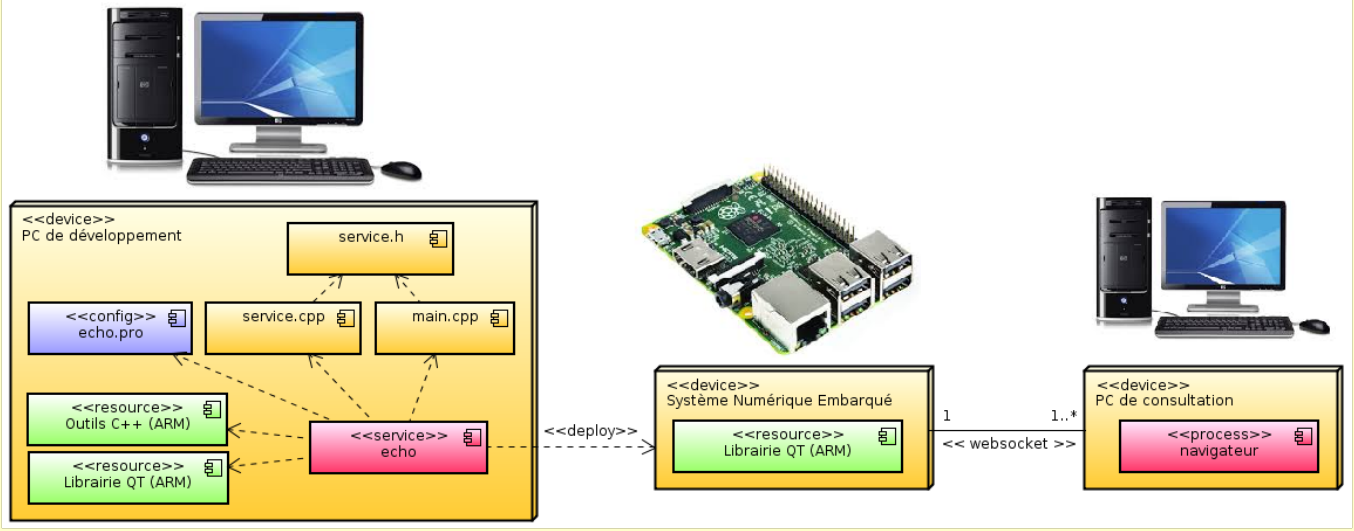


En résumé, voici ce que nous avons mis en œuvre avec les deux systèmes numériques.

Deuxième projet qui utilise la librairie Qt intégrée à la Raspberry

Le premier projet nous a permis de valider la conception d'un programme élaboré sur un ordinateur hôte pour le déployer et l'exécuter ensuite sur la Raspberry. Par contre, dans cet exemple, nous n'avons pas encore utilisé les compétences de la librairie Qt intégrée à la Raspberry. C'est ce que je propose maintenant de faire en créant un service qui utilise le protocole « **websocket** ».

L'objectif ici, n'est pas de comprendre ce protocole, mais de faire un programme relativement simple qui permet de communiquer entre différents ordinateurs. Le service est placé sur cette simple petite carte qui prend en compte les compétences de la librairie Qt intégrée. Comme tout service, plusieurs clients peuvent communiquer simultanément. Pour les tests, nous utiliserons un simple navigateur.



Le service que vous allez mettre en œuvre est très simple et consiste à renvoyer le message saisie par le client dans son navigateur. L'objectif ici, n'est pas de faire un algorithme compliqué, mais simplement de valider la communication réseau en créant un service simple au travers d'un protocole qui permet de communiquer par Internet, implémenté par la librairie Qt intégré à la Raspberry.

echo.pro

```
QT += core websockets
QT -= gui
TEMPLATE = app

CONFIG += console c++11
CONFIG -= app_bundle

SOURCES += main.cpp service.cpp
HEADERS += service.h

target.path = /home/pi/projets
INSTALLS += target
```

main.cpp

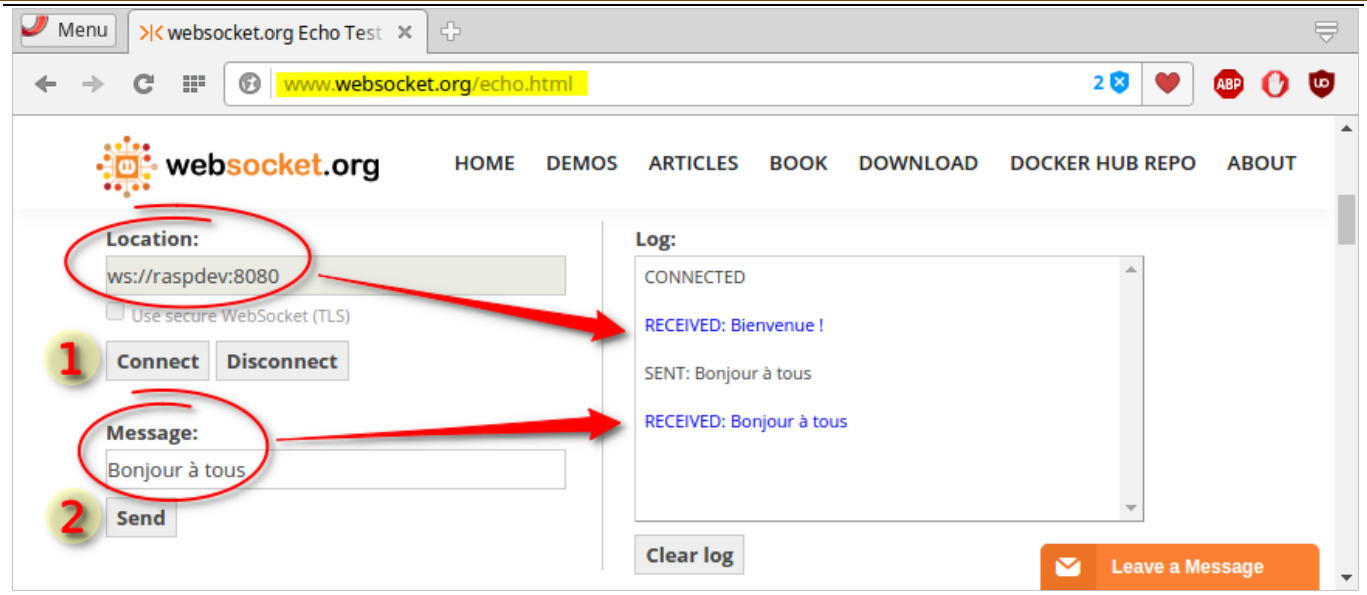
```
#include <QCoreApplication>
#include "service.h"

int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
  Service echo;
  return a.exec();
}
```

service.h

```
#ifndef SERVICE_H
#define SERVICE_H

#include <QObject>
#include <QWebSocketServer>
#include <QwebSocket>
using namespace std;

class Service : public QObject
{
  Q_OBJECT
public:
  explicit Service();
private:
  QWebSocketServer service;
public slots:
  void nouvelleConnexion();
  virtual void receptionMessage(const QString &message);
  void deconnexion();
};

#endif // SERVICE_H
```

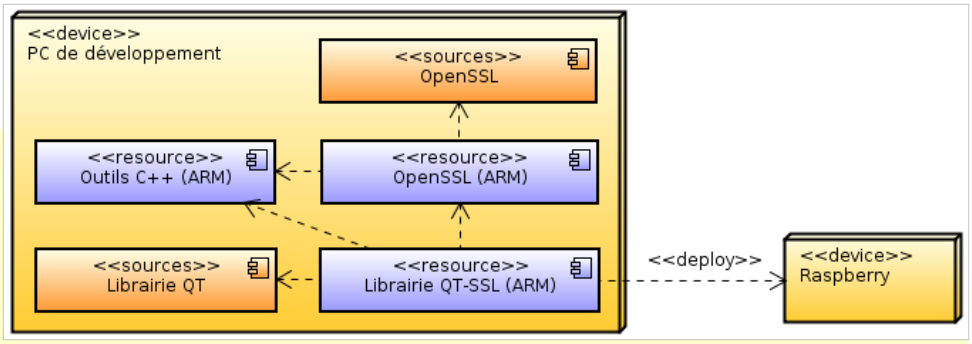
service.pp

```
#include "service.h"
#include <algorithm>

Service::Service() : service("echo", QWebSocketServer::NonSecureMode, this)
{
  if (service.listen(QHostAddress::Any, 8080))
    connect(&service, SIGNAL(newConnection()), this, SLOT(nouvelleConnexion()));
}

void Service::nouvelleConnexion()
{
  QWebSocket *client = service.nextPendingConnection();
  client->sendTextMessage("Bienvenue !");
  connect(client, SIGNAL(textMessageReceived(QString)), this, SLOT(receptionMessage(QString)));
  connect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
}

void Service::receptionMessage(const QString &message)
{
  QWebSocket *client = (QWebSocket *) sender();
  client->sendTextMessage(message);
}

void Service::deconnexion()
{
  QWebSocket *client = (QWebSocket *) sender();
  client->deleteLater();
  disconnect(client, SIGNAL(textMessageReceived(QString)), this, SLOT(receptionMessage(QString)));
  disconnect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
}
```



Compilation croisée avec OpenSSL

Tout ce que nous venons de voir tout au long de ces différents chapitres fonctionnent parfaitement bien. Il reste un petit bémol toutefois. Lorsque nous compilons la librairie Qt à partir des sources, tout les modules ne sont pas intégrer, notamment ce qui concerne la sécurisation des échanges par « **websocket** » avec « **openssl** ».



Pour cela, dans un premier temps, la seule solution, à l'image de la librairie Qt, est de réaliser une compilation croisée à partir des sources de openssl. La technique demeure la même, il faut se placer dans le répertoire de base des sources et faire appel à l'outil de configuration, avec la commande ci-dessous en mode console :

```
# Compilation croisée pour la génération de binaire OpenSSL
$ ./Configure linux-armv4 --cross-compile-prefix=arm-linux-gnueabihf- shared no-ssl3 –prefix=/usr/local/openssl
```

La première option indique quel est le type de cible (linux-armv4) suivi ensuite du compilateur adapté à la cible (arm-linux-gnueabihf) en spécifiant l'endroit où seront installés l'ensemble des binaires (/usr/local/openssl). Nous précisons entre temps les algorithmes qui ne doivent pas être pris en compte (no-ssl3) – dans ce dernier cas, il faut dire que Qt utilise plutôt l'algorithme SSL2. Nous pouvons également éliminer certains autres algorithmes comme par exemple (no-idea no-mdc2 no-rc5 no-zlib).

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@HPE-120fr ~/Public/openssl-1.0.2j $ ./Configure linux-armv4 --cross-compile-prefix=arm-linux-gnueabihf-
 shared no-idea no-mdc2 no-rc5 no-zlib no-ssl3 --prefix=/usr/local/openssl
```

Après cette phase de configuration, vous devez passer à la phase de construction, ceci en trois étapes :

```
$ make depend (prend en compte le choix des algorithmes).
$ make
$ sudo make install
```

Maintenant que nous avons la librairie OpenSSL prête pour une compilation croisée pour un processeur de type ARM, nous pouvons refaire notre librairie Qt qui va intégrer cette fois-ci ce module supplémentaire en proposant des options adaptées, notamment la localisation des fichiers inclus ainsi que l'endroit où se situent les librairies statiques et dynamiques :

```
# Compilation croisée pour la librairie Qt avec le module OpenSSL
$ ./configure -xplatform linux-arm-gnueabihf-g++ -openssl -I /usr/local/openssl/include -L /usr/local/openssl/lib -nomake tests
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
manu@HPE-120fr ~/Public/qt-everywhere-opensource-src-5.5.1 $ ./configure -opensource -confirm-license
 -xplatform linux-arm-gnueabihf-g++ -openssl -I /usr/local/openssl/include -L /usr/local/openssl/lib
-nomake tests
```

```
$ make
$ sudo make install
```

Paradoxalement, une fois que la librairie Qt est définitivement générée, nous pouvons nous passer maintenant de la librairie OpenSSL. Elle est complètement intégrée à la librairie Qt.