

Rapport de stage

2^{me} Master - finalité électronique

Lieu : CETIC asbl

Période : 17/09/2018 - 26/10/2017

Maitre de stage : Deru Laurent

Superviseur : Combéfis Sébastien

Tom Selleslagh

Table des matières

Note d'activités	3
Lieu de stage	4
1.1 Présentation de l'entreprise	4
1.2 Organisations	5
1.2.1 Pouvoir décisionnel	5
1.2.2 Organigramme	6
1.2.3 Départements	6
1.3 Apport de l'entreprise à la société	7
1.4 Motivation	9
Objectifs du stage	10
Phase d'apprentissage	11
3.1 6LBR	11
3.1.1 6LoWPAN	11
3.1.2 Principes généraux	12
3.1.3 Configuration	12
3.1.4 Mode de routage	13
3.2 Tayga	13
3.2.1 Principe	13
3.2.2 Mappage d'IPv6 en IPv4	13
3.2.3 Mappage d'IPv4 en IPv6	14
3.3 Conteneurisation d'application	14
3.3.1 Principe	14
3.3.2 Stratégie	15
3.4 Orchestration	16
Suivi du projet	17
4.1 Conteneurisation de l'application 6LBR	17
4.1.1 Choix de l'image de départ	18
4.1.2 Scripts d'installation et d'initialisation	18
4.1.3 En Pratique	18
4.2 Server LWM2M	19
4.2.1 Leshan Eclipse	19
4.2.2 EMQX	20
4.3 Conteneurisation de l'application tayga	20
4.3.1 En Pratique	21
4.4 Orchestration	21
4.4.1 Docker swarm	21
4.4.2 Kubernetes	22
Conclusion	23

Credits

25

Annexes

26

Maitre de stage :	Deru	Laurent	laurent.deru@cetic.be	+32488240726
Superviseur :	Combéfis	Sébastien	cbf@ecam.be	+32495111359
Etudiant :	Selleslagh	Tom	14161@ecam.be	+32479262071

Entreprise	CETIC Asbl	Aéropole, 28 Avenue J. Mermoz, 6041 Charleroi
Période de stage	du 17/09/2018 au 26/10/2018	

NOTE D' ACTIVITÉS

Objectifs généraux	Objectifs spécifiques
S'intégrer	<ul style="list-style-type: none"> • Comprendre le fonctionnement d'un centre de recherche • S'intégrer à la dynamique d'un département
Faciliter l'utilisation de 6LBR	<ul style="list-style-type: none"> • Comprendre le principe de fonctionnement de 6LBR • Comprendre le concept de conteneurisation d'application • Conteneuriser 6LBR • Utilisation d'un server LWM2M
Orchestrer des applications conteneurisées	<ul style="list-style-type: none"> • Comprendre le principe de l'orchestration • Tester plusieurs orchestrateurs • Créer un cluster dans lequel se trouve l'application 6LBR

Signature de l'étudiant

Signature du Maitre de Stage

Lieu de stage



FIGURE 1.1: Bâtiment du CETIC situé dans L'Aéropole de Gosselies

On notera que ce chapitre est largement inspiré des informations reprises par le CETIC sur son site internet. Beaucoup de passages n'ont quasiment pas été modifiés pour éviter d'altérer son message.

1.1 Présentation de l'entreprise

C'est en 2001, que le CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication) a été créé grâce à la collaboration de trois universités francophones : l'Université de Namur (UNamur) , l'Université Catholique de Louvain (UCL) et l'Université de Mons (UMons) ¹.

Le CETIC se définit comme un centre de recherche appliquée et de transfert de technologie au service des entreprises. Il entend être reconnu en tant qu'entreprise, partenaire, fiable, efficace et professionnel. Renforçant ses liens avec l'industrie tout en conservant des contacts privilégiés avec les universités. Le CETIC a élaboré son portefeuille de services et son expertise dans le domaine des technologies de l'information et de la communication (TIC).

Étant donné qu'il s'agit principalement d'un centre de recherche, il garde de bonnes relations avec les différents laboratoires universitaires, ce qui permet de maintenir ses équipes au centre du progrès. Financé dès sa création par le fonds FEDER1 (Phasing Out Objectif 1), il a très vite agrandi ses horizons en participant à plusieurs projets, notamment les programmes-cadre européens de R& D, le Plan Marshall (Pôle de compétitivités) et a développé ses collaborations avec les entreprises tant au niveau régional qu'eupéen. Le CETIC est agréé en Wallonie depuis 2004 comme un centre collectif de recherche.

1. à cette époque l'UNamur et l'UMons n'avaient pas encore été rebaptisées et s'appelaient respectivement la Faculté Notre-Dame de la Paix et la Faculté Polytechnique de Mons

1.2 Organisations

1.2.1 Pouvoir décisionnel

conseil d'administration

Le Conseil d'administration(CA) est composé de représentants désignés par les universités, et de personnes physiques nommées par l'assemblée générale de sorte que le CA comprenne une majorité de représentants d'entreprises exerçant des fonctions de recherche et/ou de direction issue des secteurs visés par le CETIC. Sa mission première est de s'assurer de la pérennité de l'entreprise.

Le Conseil d'Administration du CETIC se compose comme suit :

- Monsieur Bernard Bolle, CIN - NIC
- Monsieur Serge Boucher, Université de Mons
- Monsieur Patrick Donnay, Haulogy
- Monsieur Patrick Heymans, Université de Namur
- Monsieur Didier Lefebvre, Océ Software Laboratories
- Monsieur Jean-Didier Legat, UCLouvain (Vice-président)
- Monsieur Arnaud Ligot, CBlue (Vice-président)
- Monsieur Pierre Manneback, Université de Mons (Trésorier et Secrétaire)
- Madame Carine Michiels, Université de Namur
- Monsieur Quentin Poncelet, COMPUTERLAND
- Monsieur William Poos, NRB
- Monsieur Jean-Christophe Renauld, UCLouvain
- Monsieur Michel Rousseau, Alstom
- Monsieur Bruno Schröder, Microsoft (Président)
- Monsieur Nicolas Sottiaux, IGRETEC
- Madame Valérie Viatour, Chiveo

comité de direction

Son mandat est de s'occuper de la gestion courante de l'entreprise.

Le comité de direction est constitué comme suit :

- Damien Hubaux (Directeur général)
- Jean-Christophe Deprez (Directeur Positionnement Entreprises-Recherche)
- Rosario Maggio (Directeur administratif et financier)
- Stéphane Mouton (Directeur des Opérations)

1.2.2 Organigramme

Ci-dessous, l'organigramme du CETIC issu du dernier conseil d'administration.

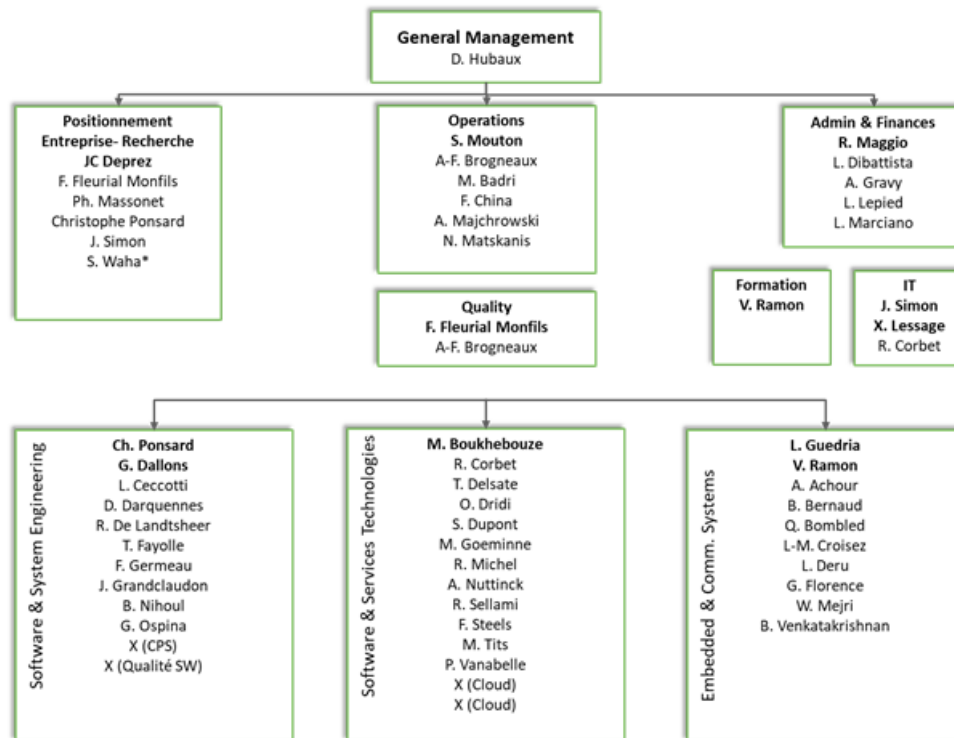


FIGURE 1.2: dernière version de l'organigramme du CETIC

1.2.3 Départements

Le CETIC s'articule actuellement² autour de trois départements :



Le département Software and System Engineering (SSE) aide les entreprises à concevoir des produits et services de grande qualité en termes de fiabilité, de sécurité, de maintenabilité dans le respect des normes internationale. Il apporte, également, un soutien méthodologique outillé sur tout le cycle de vie.



Le département Software Services Technologies (SST) aide les entreprises à exploiter les technologies émergentes pour concevoir des systèmes distribués sur base de nouvelles architectures informatiques capables de s'adapter à la charge. Ce travail est réalisé en exploitant les technologies d'automatisation de gestion d'environnements informatiques en appliquant les méthodes et outils de gestion de données.



Le département Embedded and Communicating Systems (ECS) aide les entreprises à exploiter les technologies émergentes pour concevoir des systèmes embarqués avec plus d'intelligence intégrée. Pour ce faire, les membres de ce département établissent une connectivité optimisée au contexte d'utilisation tout en exploitant les capacités des architectures matérielles complexes lors de la conception des objets intelligents.

2. Le CETIC est dans une phase de réflexion qui devrait aboutir à la création de nouveaux départements

1.3 Apport de l'entreprise à la société

Le CETIC offre son expertise aux entreprises informatiques qui cherchent un soutien pour intégrer les dernières avancées technologiques dans leurs produits et processus. Il vise à soutenir la croissance des entreprises en aidant celles-ci à réduire la prise de risque et à accélérer la mise sur le marché de leurs nouveaux concepts. Pour remplir ces différentes missions, le CETIC propose différents services aux entreprises et PME :

État de l'art

Exploitant les résultats des projets de recherche, le CETIC réalise des états de l'art adapté au contexte des acteurs avec lesquels il travaille.

Audit

Les audits permettent d'avoir un état des lieux à un instant donné. À l'issue de l'analyse, un ensemble de recommandations sont produites pour améliorer l'efficacité, en particulier au niveau des outils informatiques. En aval, un cahier de charges pour l'évolution de l'informatisation des processus peut également être produit.

Ces audits peuvent prendre plusieurs formes :

- aider à l'identification des besoins, esquisser les choix qui s'offrent en termes de technologies, faire le lien entre donneur d'ordre et fournisseur dans le cadre de l'exécution d'un contrat, l'estimation du coût d'un développement informatique...
- fournir des recommandations sur l'adoption des meilleures pratiques, conseiller sur l'évolution de l'architecture de solutions informatiques, la reprise, la maintenance et l'évolution d'un code source...
- apport du support concernant l'adoption de normes et de standards liés au logiciel afin de préparer à d'éventuelles certifications.

Conseils et accompagnements

La démarche d'accompagnement est organisée dans une optique d'amélioration de processus : sur base de la réalisation d'un état des pratiques, des recommandations sont produites. Un plan d'action peut ensuite être proposé ainsi qu'un accompagnement pour la mise en œuvre de ce dernier. Le CETIC fournit une assistance à la maîtrise de projet informatique dans des phases spécifiques du développement telles que : l'analyse des besoins, l'architecture logicielle, la mise en place de tests, amélioration es processus de développement...

Faisabilité technologique

Dans le cadre du développement de nouveaux produits informatiques, ou la mise à jour d'applications existantes, il est nécessaire de mettre en œuvre de nouveaux concepts, de nouvelles technologies. Le CETIC va *déminer* le terrain en étroite collaboration avec l'entreprise afin d'intégrer ceux-ci.

transfert d'expertises

Investi dans les secteurs de pointe tels l'eSanté, le cloud computing, l'open source, la sécurité, le transport/logistique, le sans-fil et la sémantique. Le CETIC a établi des accords de partenariat stratégique avec des leaders technologiques ainsi que des industriels, tant au niveau régional qu'europpéen, accélérant le transfert d'expertise au profit des entreprises belges.

Escrow

Le CETIC réalise des missions d'entiercement (Escrow). L'Escrow consiste généralement à déposer chez un tiers (notaire, société spécialisée) les sources, les plans, les procédures permettant de reconstituer un produit logiciel et/ou matériel. Ces pratiques sont courantes pour les produits logiciels et matériels d'une grande technicité et à longue durée de vie. Les sociétés d'Escrow, les notaires conservent les documents de l'Escrow en donnant aux deux parties toutes les garanties quant aux possibilités d'accès prévues par le contrat.

Cette mission consiste à auditer les documents de design et de réalisation, les plans de fabrication et d'installation (électroniques et informatiques) ainsi que les logiciels mis en œuvre. Il s'agit de vérifier que ceux-ci sont bien de nature à permettre au client de confier la fabrication de ces produits à une autre entreprise en cas de défaillance de son fournisseur.

En tant que centre de recherche indépendant et société sans but lucratif, le CETIC est reconnu comme expert technique, mais également comme tiers de confiance tant par les clients que par les fournisseurs ; ce qui est primordial dans ces matières critiques en termes de propriété intellectuelle.

1.4 Motivation

Le CETIC est une entreprise inspirante pour des personnes voulant évoluer dans le milieu de la recherche appliquée. Son statut de PME privée sans but lucratif lui permet de collaborer avec les entreprises en toute indépendance, dans le cadre de relations de confiance, dans un esprit de coopération et hors de toute compétition.

Le CETIC contribue activement au développement régional de la Wallonie, en appuyant et stimulant l'innovation dans le tissu économique local, et notamment dans les PME. Plus de 90% des entreprises contractant avec le CETIC ont leur siège en Wallonie. Le profil de ses clients est très diversifié :

- de grandes entreprises
- des PME matures investissant dans leur R& D
- des entrepreneurs en phase de définition de leur produit
- des entreprises étrangères (sur certaines technologies pointues)

Les équipes du CETIC font preuve d'initiative, de créativité et de curiosité dans les sujets de recherche abordés et font appel à leur complémentarité afin de garantir un résultat innovant et de qualité. Les liens étroits qu'elles entretiennent avec les industriels (notamment ceux du comité technique permanent) leur garantissent la pertinence des projets de recherche.

Cette entreprise met en avant un travail collaboratif efficace aussi bien au sein d'une même équipe qu'inter équipes et offre à ses employés un environnement de travail motivant alliant convivialité, flexibilité et autonomie. Cette ambiance de travail offre à chacun la possibilité de s'épanouir, de s'exprimer librement et d'entraîner sa créativité. En outre, il offre également à chacun la liberté d'améliorer continuellement ses compétences notamment grâce à la diversité des expertises représentées au sein du CETIC.

En conclusion, de par son statut d'ASBL, Le CETIC est un centre de recherche dont la motivation première n'est pas le profit. Une de ses valeurs clés est sa propension à ne pas vouloir réinventer la roue en facilitant les échanges inter département. Toutes ces caractéristiques en on fait un candidat de choix lors de ma prospection d'un endroit de stage.

Objectifs du stage

Le stage sera avant tout le préambule à un travail de fin d'études dont le sujet portera sur les micro-services et orchestration d'applications conteneurisées.

L'ambition du TFE est de réaliser une architecture distribuée dont l'orchestration sera **autonome**. L'orchestrateur aura connaissance des **spécificités des nœuds** (architecture, mémoire vive ...) et saura aussi si un container peut-être nomade ou non c'est-à-dire s'il est lié à une interface physique ou pas. Le déploiement sera testé sur une **architecture IoT** présentant plusieurs passerelles de communication différentes (6LowPan, Lora, Zwave...). Il va sans dire qu'une des spécificités de ce cluster sera de maintenir une continuité de services, et ce, même pendant les phases de mise à jour.

Le but du stage sera de réaliser un cluster avec au moins deux nœuds d'architecture différente sur lequel tournera l'application conteneurisée de 6LBR et un serveur LWM2M.

Ce projet amènera les connaissances de base dans les domaines de la conteneurisation et de l'orchestration. Comme il s'agit d'une première approche, une seule passerelle IoT sera déployée : le border router 6LowPAN développé par le CETIC.

Phase d'apprentissage

Le dessein de ce chapitre est de résumer les notions théoriques qui ont été assimilées pendant le stage et qui relèvent d'un caractère essentiel dans la compréhension du projet.

3.1 6LBR

Cette section n'a pas vocation à ré expliquer dans les détails le principe des réseaux 6LoWPAN mais bien de donner les bases nécessaires à la compréhension et la configuration de 6LBR.

3.1.1 6LoWPAN

6LoWPAN est l'acronyme de *IPv6 Low power Wireless Personal Area Networks* et est une couche d'adaptation pour IPv6 sur réseau limité, comme 802.14.5. En 802.15.4, la taille maximale du PSDU³ Est de 127 octets. On notera que sur ces 127 octets seulement 21 représentent des données utiles lorsque le protocole de transport utilisé est TCP. On peut faire grimper ce chiffre à 33 en utilisant UDP comme protocole de transport (voir fig 3.3). Dans ces conditions on ne respecte pas les spécifications d'IPv6 qui imposent un MTU minimal de 1280 octets.

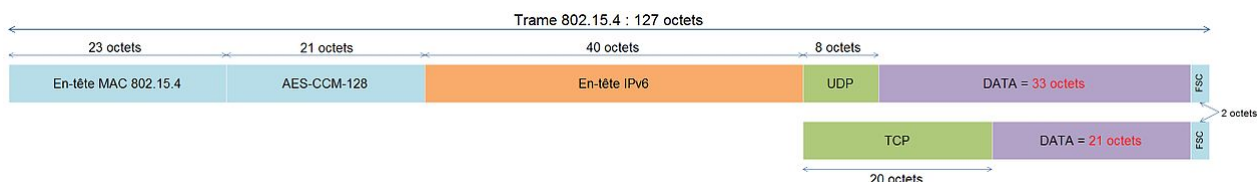


FIGURE 3.3: Datagramme 802.15.4

Pour s'adapter au support 802.15.4, l'équipement doit fragmenter un paquet IPv6 en plusieurs trames 802.15.4 et compresser l'entête IPv6, et l'équipement distant doit réassembler toutes les trames 802.15.4 reçues pour régénérer le paquet IPv6 d'origine. La couche d'adaptation de 6LoWPAN se situe entre la couche réseau et la couche liaison du modèle OSI, on pourrait la qualifier de couche 2,5 . Elle reçoit de la couche réseau des paquets IPv6 de 1280 octets et les envoie à son équivalent sur l'équipement distant dans des trames 802.15.4.

Le schéma de routage de 6LoWPAN peut être réalisé selon deux manières différentes :

- soit au niveau de la couche adaptation, la décision de routage se fait au niveau 6LoWPAN et donc seulement avec les fragments du paquet IPv6. Cette méthode est appelée **mesh-under** et permet d'avoir un délai de transmission plus court puisque le paquet IPv6 n'est reconstitué que sur l'équipement destinataire.
- soit au niveau de la couche réseau, le paquet IPv6 est reconstitué sur chaque équipement intermédiaire afin de prendre la décision de routage. Cette méthode, appelée **route-over**, est plus efficace dans des conditions dégradées (perte de paquets).

3. Physical layer Service Data Unit

3.1.2 Principes généraux

La CETIC a développé une solution de *border router* 6LoWPAN/RPL⁴ : 6LBR. Il peut fonctionner en tant que routeur autonome sur du matériel intégré ou sur un hôte Linux. 6LBR est conçu pour la flexibilité, il peut être configuré pour prendre en charge diverses topologies de réseau tout en interconnectant intelligemment les WSN⁵ avec le monde IP.

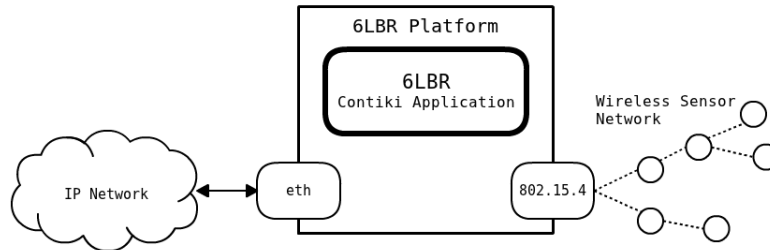


FIGURE 3.4: Schéma de principe de l'application 6LBR.

3.1.3 Configuration

configuration des interfaces

Il existe 3 modes de configuration des interfaces au sein de l'application 6LBR

schéma de principes	description	6lbr.conf
	<p>Le mode RAW-ETHERNET est le plus facile à mettre en œuvre mais est aussi le plus limitant. 6LBR utilise directement l'interface Ethernet spécifiée dans le fichier de configuration pour envoyer et recevoir les paquets.</p>	<pre>RAW_ETH=1 BRIDGE=0 DEV_ETH=eth0</pre>
	<p>Le mode BRIDGE crée une interface virtuelle qui devra être reliée au moyen d'un bridge à l'interface Ethernet. Il est cependant nécessaire que le module bridge soit présent dans le <i>kernel</i> du système.</p>	<pre>RAW_ETH=0 BRIDGE=1 CREATE_BRIDGE=1 DEV_ETH=eth0 DEV_BRIDGE=br0 DEV_TAP=tap0</pre>
	<p>Le mode ROUTING est le plus puissant mais le plus complexe. Comme pour le mode BRIDGE, une interface virtuelle est créée mais le routage devra intégralement être réalisé par l'utilisateur en fonction des besoins.</p>	<pre>RAW_ETH=0 BRIDGE=0 DEV_ETH=eth0 DEV_TAP=tap0</pre>

4. Routing Protocol for Low power and Lossy Networks est un protocole de routage de type route-over

5. Wireless Sensor Network

3.1.4 Mode de routage

Il existe plusieurs modes de routage disponible, mais nous avons durant le stage essentiellement travaillé avec le mode ROUTER. Les autres modes ne seront donc pas détaillés.⁶.

Dans ce mode, 6LBR agit comme un routeur IPv6 à part entière, interconnectant deux sous-réseaux IPv6. Le sous-réseau WSN est géré par le protocole RPL (cf. 3.1.1) et le sous-réseau Ethernet par IPv6 NDP⁷. ce mode fonctionne comme une passerelle entre Ethernet et 6Lowpan RPL.

3.2 Tayga

3.2.1 Principe

Comme expliqué au point 3.1.4 6LBR existe dans un monde uniquement IPV6. En outre, à l'heure actuelle un grand nombre de solutions d'orchestration et d'applications reste cantonnée dans le monde IPV4. Pourtant L'IPv6 se profile comme le successeur du protocole réseau classique IPv4 depuis l'essor des objets connectés. Cette nouvelle version offre de nombreux avantages, dont la prise en charge d'un plus grand nombre d'adresses (340 sextillions contre 4,3milliards pour l'IPV4).

Dans l'attente d'une avancée dans ce domaine, une solution temporaire peut être envisagée : l'utilisation de l'application **Tayga**, un NAT64 sans état.

La plupart des gens connaissent le NAT avec état, qui est le plus souvent utilisé pour traduire les sessions de plusieurs hôtes internes (numérotés avec des adresses privées) sur un seul adresse globale sur l'interface externe du périphérique NAT.

Le NAT sans état effectue simplement une substitution des adresses IP à l'aide d'une table de mappage fourni par l'administrateur du réseau. Par exemple, une organisation dont l'allocation d'adresse globale était 198.51.100.0/24 mais dont les hôtes utilisaient les adresses dans le réseau 192.0.2.0/24 pourraient utiliser un NAT sans état pour réécrire 192.0.2.1 en 198.51.100.1, 192.0.2.35 à 198.51.100.35, etc., dans la direction sortante, et l'inverse dans la direction entrante. Tayga fonctionne de cette manière.

Quand on traduit des paquets d'IPv4 à IPv6 (ou IPv6 à IPv4), l'adresse de la source et la destination sont substituées dans les en-têtes de paquet en utilisant un mappage un pour un. Ce qui signifie que, pour échanger des paquets sur le NAT64, chaque hôte IPv4 doit être représenté par une adresse IPv6 unique et chaque hôte IPv6 doit être représenté par une adresse IPv4 unique.

3.2.2 Mappage d'IPv6 en IPv4

Comme expliqué au point précédent, en tant que NAT sans état, Tayga nécessite l'attribution d'une adresse IPv4 unique pour chaque hôte IPv6. Cette affectation peut être faite statiquement par l'administrateur réseau ou dynamiquement par Tayga à partir d'un pool d'adresses IPv4.

Le mappage dynamique permet à Tayga d'attribuer des adresses IPv4 aux hôtes IPv6. Par défaut, il est garanti que ces assignations restent utilisables endéans les deux heures après le dernier paquet vu, mais elles sont conservées jusqu'à deux semaines tant que le pool d'adresses n'est pas vide⁸.

6. La description détaillée des autres modes est disponible sur la page <https://github.com/cetic/6lbr/wiki/6LBR-Modes>

7. Neighbor Discovery Protocol (NDP) est un protocole utilisé par IPv6. Il opère en couche 3 et est responsable de la découverte des autres hôtes sur le même lien, de la détermination de leur adresse et de l'identification des routeurs présents.

8. Les assignations sont écrites sur le disque, elles sont donc persistantes lors du redémarrage du démon. Ce qui permet aux sessions TCP et UDP existantes de ne pas être interrompues.

3.2.3 Mappage d'IPv4 en IPv6

Tayga mappe les adresses IPv4 vers le réseau IPv6 conformément à la norme RFC 6052 qui stipule qu'une adresse IPv4 32 bits doit être ajoutée à un préfixe IPv6, appelé le préfixe NAT64. Cette adresse IPv6 résultante peut être utilisée pour contacter l'hôte IPv4 via le NAT64.

Le préfixe NAT64 doit être attribué dans le range d'allocation d'adresse IPv6 globale d'un site. Par exemple, si un site est alloué en `2001:db8:1::/48`, le préfixe `2001:db8:1:ffff::/96` peut être réservé pour le NAT64⁹. L'hôte IPv4 `198.51.100.10` est alors accessible via le NAT64 à l'aide de l'adresse `2001:db8:1:ffff::c633:640a`. Dans la pratique, il est possible d'utiliser la syntaxe `2001:db8:1:ffff::198.51.100.10` à la place.

NAT64 fournit par l'application 6LBR

Un NAT64 est déjà implémenté dans l'application 6LBR. La méthode utilisée est cependant différente, il s'agit d'un NAT avec état. L'application attribue un port d'une adresse IPv4 fixée préalablement aux *endpoints* du WSN. Ceci n'est pas compatible avec les principes de conteneurisation.

3.3 Conteneurisation d'application

3.3.1 Principe

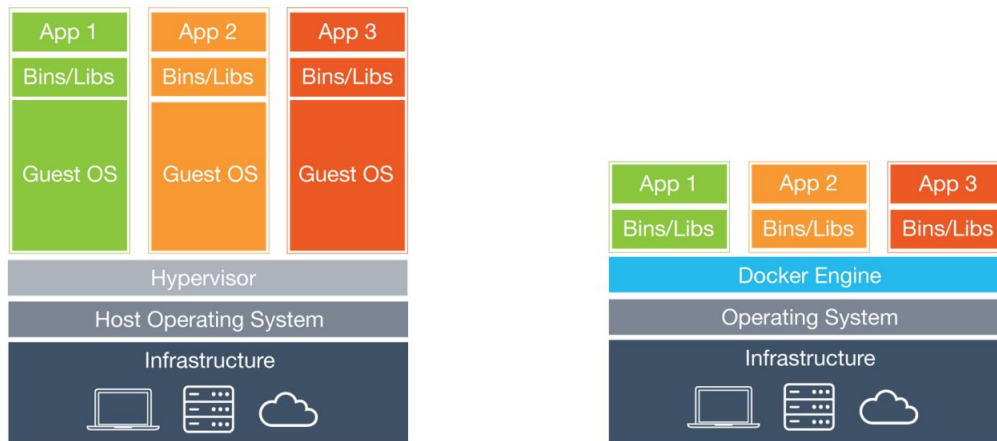


FIGURE 3.5: Comparaison entre utilisation de machines virtuelles et de conteneur

Comme le montre la figure 3.5, la virtualisation implique de créer des instances virtuelles spécifiques d'une infrastructure matérielle. A contrario, la conteneurisation crée simplement des conteneurs isolés au niveau du système d'exploitation qui seront gérés par une application tierce appelée *container engine*. Ainsi, l'OS est partagé par différents conteneurs plutôt que d'être cloné pour chaque machine virtuelle.

Un conteneur comprend un environnement d'exécution complet : une application accompagnée de toutes ses dépendances (bibliothèques, binaires, et fichiers de configuration) rassemblées en un seul package, appelé *image de conteneur*.

La taille d'une image de conteneur ne sera que de quelques dizaines de mégabytes, là où pour une machine virtuelle avec son système d'exploitation complet embarqué, on pourrait monter jusqu'à plusieurs gigabytes. De plus, le déploiement d'un conteneur s'effectuant sur un OS déjà opérationnel, le

9. Il y a plusieurs options pour la longueur du préfixe NAT64, mais un /96 est recommandé.

démarrage des applications conteneurisées est quasiment instantané.

Dans ce domaine, la solution proposée par **Docker** tend à s'imposer comme leader. De plus, il fournit un dépôt d'applications déjà conteneurisées (<http://hub.docker.com>).

3.3.2 Stratégie

Choix de la plateforme de travail

Pour chaque type de plateforme, il existe une version de Docker. Sur les versions *macos* et *windows*, la création d'un lien entre un device USB connecté à la machine et le conteneur n'est pas évidente. Il faut utiliser l'outil *docker-machine* qui permet de contrôler une machine virtuelle sur laquelle sera installé **Docker Engine**. Cette machine virtuelle supplantera l'application **Docker** tournant en avant-plan. En fonctionnant de cette manière, on perd du coup tout l'avantage de l'utilisation des conteneurs.

Tous les tests de conteneurs ont donc été réalisés sur des machines dont le système d'exploitation était une distribution Linux.

Standardisation des Dockerfile

Le mécanisme de Docker pour créer une image peut alourdir considérablement la taille finale de celle-ci. En effet, pour chaque commande **RUN** Docker crée un calque. C'est l'addition de tous ces calques qui donnera l'image finale. Il vient directement comme observation que plus il y a de calques plus l'image sera volumineuse.

L'objectif est donc de créer un *Dockerfile*¹⁰ avec le minimum de commandes. Pour standardiser tous les *Dockerfile*, un template a été créé. Ce modèle se construit comme suit :

1. L'image de départ
2. L'ajout d'un dossier avec deux scripts Shell
3. Le lancement d'un script d'installation avec la commande RUN
4. Le lancement d'un script d'initialisation avec la commande CMD

Le script d'installation aura pour missions :

- l'installation des dépendances
- l'installation et/ou compilation de l'application
- la suppression de tout ce qui n'est pas essentiel

On placera dans le script d'initialisation toutes les tâches à réaliser lors du lancement du conteneur.

On pourra, par exemple, attribuer à ce script les tâches suivantes :

- le lancement de l'application
- la configuration de l'application
- les règles de routages
- ...

Voici le template utiliser pour la création des images de conteneurs :

```
FROM debian
ADD needed_files ./needed_files
RUN chmod -R +x /needed_files && needed_files/Install.sh
CMD needed_files/OnBoot.sh
```

10. nom du fichier servant à la construction d'images de conteneur

3.4 Orchestration

Le point 3.3 montre bien l'utilité des conteneurs dans une architecture constituée de micro-services. La puissance de cette technologie apparaît lorsque plusieurs de ces services sont mis en réseau dans un *cluster*. L'orchestration désigne le processus d'organisation du *travail* de ces conteneurs. Les outils fournis par l'orchestrateur permettent de gérer le déploiement de conteneurs, d'automatiser leurs mises à jour et de contrôler leur état.

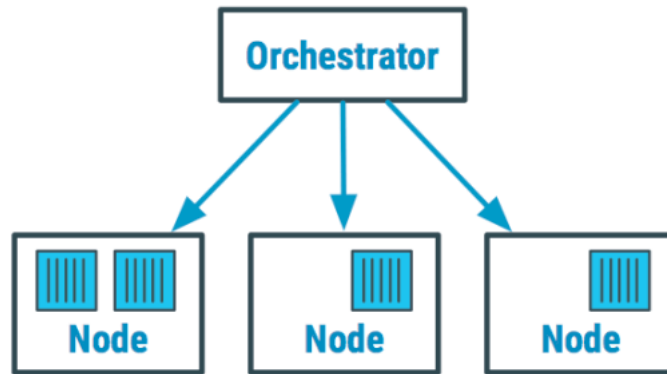


FIGURE 3.6: Schéma de principe de l'orchestration

Suivi du projet

4.1 Conteneurisation de l'application 6LBR

On aurait pu imaginer que le mode *RAW_ETHERNET* serait choisi pour sa facilité de mise en œuvre. Cependant lors des premiers tests un problème de communication a été rencontré : la page du webserver fourni par 6LBR n'était pas joignable au moyen de la commande *curl*.

Les logs de 6lbr montrent que le *processing* du paquet ne se passe pas de la manière attendue. Comme le montre la figure 4.7, l'application reçoit bien une demande, mais ne répond pas.

```
2018-09-19 12:31:52.872910: PACKET: PFE: eth_input: Processing frame
2018-09-19 12:31:52.873034: PACKET: PFE: bridge_output: Sending packet to 02:42:0c:ff:ff:aa:8d:cb
2018-09-19 12:31:52.873394: PACKET: PFE: dest prefix eth : bbbb::
2018-09-19 12:31:52.873513: PACKET: PFE: dest eth : bbbb::7147:b885:16f1:dcd8
2018-09-19 12:31:52.873754: PACKET: PFE: eth_output: 02:42:0c:ff:ff:aa:8d:cb
2018-09-19 12:31:52.874111: PACKET: PFE: eth_output: Sending packet to Ethernet
2018-09-19 12:31:52.874190: PACKET: ETH: write: 118
2018-09-19 12:31:52.874514: PACKET: TAP: write: 118
2018-09-19 12:32:03.514186: PACKET: TAP: read: 94
2018-09-19 12:32:03.514365: PACKET: ETH: read: 94
2018-09-19 12:32:03.514461: PACKET: PFE: eth_input: Processing frame
2018-09-19 12:32:04.578122: PACKET: TAP: read: 94
2018-09-19 12:32:04.578265: PACKET: ETH: read: 94
2018-09-19 12:32:04.578340: PACKET: PFE: eth_input: Processing frame
2018-09-19 12:32:06.658229: PACKET: TAP: read: 94
2018-09-19 12:32:06.658406: PACKET: ETH: read: 94
2018-09-19 12:32:06.658506: PACKET: PFE: eth_input: Processing frame
2018-09-19 12:32:10.738119: PACKET: TAP: read: 94
2018-09-19 12:32:10.738245: PACKET: ETH: read: 94
2018-09-19 12:32:10.738289: PACKET: PFE: eth_input: Processing frame
```

FIGURE 4.7: Résultats de l'affichage des logs de l'application 6LBR

Des essais ont été réalisés avec la configuration des interfaces en mode *BRIDGE*. La fonction *curl* renvoie bien les informations du web serveur :

```
pi@raspberrypi:~$ curl http://[fd00::212:4b00:616:fd9]
<html><head><link rel="stylesheet" type="text/css" href="6lbr_layout.css" /><title>Info - 6LBR</title></head>
<body class="page_rubrique"><div id="container"><div id="banner"><h1>6LBR</h1><h2>6Lowpan Border Router</h2><
div class="nb"><div class="gm"><span>System</span></div><div class="gm"><a href="sensors.html">Sensors</a></d
iv><div class="gm"><a href="network.html">Status</a></div><div class="gm"><a href="config.html">Configuration
</a></div><div class="gm"><a href="statistics.html">Statistics</a></div><div class="gm"><a href="admin.html">
Administration</a></div></div></div><div id="intro_home"><h1>Info</h1></div><div id="left_home"><h2>Info</h2>
Hostname : 5f1a38fd2af9<br />Version : 1.5.x (Contiki-contiki-6lbr-1.5.0-2489-gb85ef16a3)<br />Mode : SmartBr
idge<br />Uptime : 0h 7m 40s<br /><br /><h2>WSN</h2>Multicast: None<br />MAC: CSMA<br />RDC: br-rdc (0 Hz)<br
/>Security: nullsec<br />HW address 0 : 0:12:4b:0:6:16:f:d9<br />Address : fd00::212:4b00:616:fd9<br />Local
address : fe80::212:4b00:616:fd9<br /><br /><h2>Ethernet</h2>HW address : 2:0:6:16:f:d9<br /><div id="footer
">6LBR By CETIC (<a href="http://cetic.github.com/6lbr">documentation</a></div></i></div></i>
```

FIGURE 4.8: Résultats attendu de la commande *curl*

Le Mode *ROUTING* sera préféré au mode *BRIDGE* car le bridge n'est pas créé lors du lancement du conteneur. Les causes de ce problème n'ont pas été explorées durant le stage. Pour les utilisateurs qui désirent cependant utiliser un bridge pour connecter les interfaces, il est possible de le faire manuellement au moyen des outils fournis par le package Linux *bridge-utils*. En outre, sur les premiers essais d'orchestration, le mode *BRIDGE* posait problème.

4.1.1 Choix de l'image de départ

Le choix de l'image de départ pour la construction d'une image de conteneur peut être un facteur déterminant dans la taille finale de l'application. Souvent le choix se porte sur **Alpine**, une distribution Linux ultra légère (l'image Docker ne fait que 4Mo). Cependant les premiers essais réalisés avec **Alpine** ne sont pas concluants.

L'application 6LBR utilise des fonctions de contextualisation obsolète issues de la librairie *GNU c*, **Alpine** n'utilise pas cette librairie et lui préfère *musl* dans laquelle toutes ces fonctions utilisant la notion de contexte ne sont pas présentes. Les essais sont donc réalisés, sur base de l'image **Debian Jessie**(58Mo) en attendant la mise à jour de 6LBR.

4.1.2 Scripts d'installation et d'initialisation

Comme décrit au point 3.3.2, Les *Dockerfile* seront tous réalisés de la même manière.

Le script d'installation suit scrupuleusement les instructions d'installation disponible dans la documentation de 6LBR¹¹.

Le script d'initialisation, présenté ci-dessous, a quant à lui, plusieurs rôles : la configuration de l'application, le lancement de celle-ci, le routage¹² et une dernière étape consistant à maintenir le conteneur *en vie* en utilisant une commande d'avant plan qui ne se termine pas, ici un *tail -f* des logs de 6lbr.

Script 6lbr_Docker_OnBoot.sh :

```
#6LBR Configuration
nvmtool --update --dft-router bbbb::1 /etc/6lbr/nvm.dat
echo "address=$LWM2MSERVER" >> /etc/6lbr/nvm.conf
sleep 2
#Launch application
service 6lbr start
sleep 10
#Routing and IPV6 Configuration
echo 0 > /proc/sys/net/ipv6/conf/tap0/disable_ipv6
echo 1 > /proc/sys/net/ipv6/conf/tap0/forwarding
echo 1 > /proc/sys/net/ipv6/conf/eth0/forwarding
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
echo 0 > /proc/sys/net/ipv6/conf/all/disable_ipv6
ip addr add bbbb::1/64 dev tap0
ip addr add cccc::1/64 dev tap0
#Keep Container alive
cat /var/log/6lbr.ip
tail -f /var/log/6lbr.log
```

4.1.3 En Pratique

Une image de conteneur déjà construite à été poussée sur le hub de docker sous le tag **selltom/6lbr**. De cette manière, le lancement de l'application 6LBR est maintenant facilité. Il suffit d'utiliser la commande suivante :

```
docker run -t --privileged --device=/dev/bus/usb/001/005 selltom/6lbr:amd64
```

11. <https://github.com/cetic/6lbr/wiki/Other-Linux-Software-Configuration>

12. obligatoire puisque le mode choisi est ROUTER, comme mentionné au point 3.1.4

le paramètre `-device` correspond à l'endroit où est connecté la passerelle sur le nœud. Le premier nombre correspond au numéro de bus et le second au numéro du device sur ce bus. Ces informations peuvent être récupérées via la commande `lsusb`

```
pi@raspberrypi:~$ lsusb
Bus 001 Device 005: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
Bus 001 Device 004: ID 04f2:0408 Chicony Electronics Co., Ltd
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

FIGURE 4.9: Résultats de la commande `lsusb`

Le paramètre `-privileged` donne **tous** les droits à un container, il est donc à utiliser avec prudence.

Pour les personnes voulant utiliser l'application avec un configuration différente, les fichiers nécessaires à la construction de l'image de 6LBR sont disponibles dans le dépôt github suivant :

https://github.com/Sellto/6LBR_Docker.git

4.2 Server LWM2M

L'application 6LBR est fournie avec deux *plugins* : un **serveur web** permettant de configurer l'application et un client **LWM2M** permettant d'utiliser le protocole de communication CoAP :

CoAP (Constrained Application Protocol) est un protocole de transfert Web optimisé pour les périphériques et réseaux contraints utilisés dans les réseaux de capteurs sans fil pour former l'Internet des objets. Basé sur le style architectural REST, il permet de manipuler au travers d'un modèle d'interaction client-serveur les ressources des objets communicants et capteurs identifiées par des URI en s'appuyant sur l'échange de requêtes-réponses et méthodes similaires au protocole HTTP.

L'application jouera le rôle de server CoAP. Le standard de communication utilisé sera LWM2M qui ajoute une couche d'abstraction permettant de gérer un certain nombre de problématiques de l'Internet des Objets.

Le standard ne sera pas décrit dans ce rapport car, durant le stage, le server LWM2M ne servait que de vérification à la bonne communication avec le border router 6LoWPAN. il fera cependant l'objet d'un chapitre du travail de fin d'études.

Deux serveurs LWM2M ont été étudié durant le stage : Leshan et EMQX.

4.2.1 Leshan Eclipse

Leshan est avant tout une libraires mise a disposition des développeurs pour les aider à concevoir leur propre client et serveur LWM2M. Une server demo est cependant disponible permettant de tester la communication avec le client LWM2M. L'image de conteneur **gebart/leshan** utilisée lors des tests provient du hub Docker.

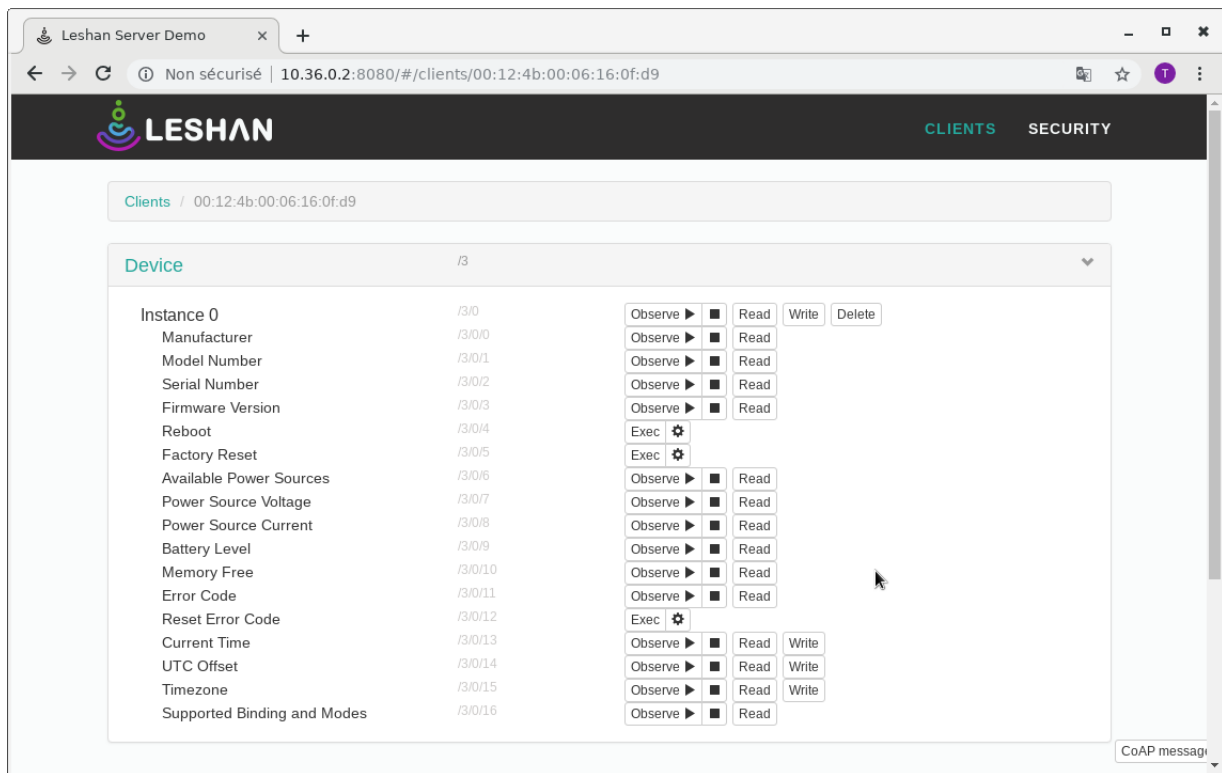


FIGURE 4.10: 6LBR connecté au server LWM2M démo Leshan

4.2.2 EMQX

EMQX est un *broker* MQTT possédant un *plugin* faisant office de passerelle entre MQTT et LWM2M. Ce qui semble être un outils très intéressant pour les ambitions futures du projet. En outre un version conteneurisée officielle est disponible sur le hub Docker.

Cependant la configuration du plugin LWM2M à été réduite au strict minimum , et seul le port peut être configuré. Ce port point directement vers une adresse ipv4. Il n'a pas l'air possible de pouvoir configuré ce plugin pour qu'il point vers une adresse IPv6. Deux solutions sont envisagées :

1. une *issue*¹³ a été soumise sur le GitHub du plugin. A ce jour, toujours aucune réponse des développeurs.
2. Utiliser un NAT64 pour *convertir* les adresses IPv6 du WSN en adresses IPv4. Cette solution sera utilisée par l'intermédiaire de Tayga (cf 3.2).

4.3 Conteneurisation de l'application tayga

Le *Dockerfile* utilisé pour la construction de l'image de notre nat64 suit également le template du point 3.3.2 avec comme point de départ la distribution linux alpine.

Le script d'installation se base sur le fichier *tar* de la version 0.9.2 Tayga¹⁴.

L'un des objectifs principaux dans la conteneurisation de Tayga est d'obtenir un conteneur qui s'auto-configue. A cette effet un exécutable a été créé : **netinfo**¹⁵. Cette application se base sur les librairies *arpa/inet.h* et *ifaddrs.h* pour extraire les informations utiles à la configuration de Tayga. Comme le montre la figure 4.11, plusieurs outils sont disponibles.

13. <https://github.com/emqx/emqx-lwm2m/issues/4>

14. <http://www.litech.org/tayga/>

15. Code source disponible sur la page <https://github.com/Sellto/netinfo>

```
Available commands:
--ipv4name          IPV4 interface name
--ipv4addr          IPV4 address
--ipv4router        IPV4 router address

--ipv6addr          IPV6 address
--ipv6name          IPV6 interface name
--ipv6natsubnet     IPV6 nat subnet
--ipv6router        IPV6 router address
--ipv6net           IPV6 prefix

--taygaconf [POOL_NETWORK][MASK] auto configuration of tayga
```

FIGURE 4.11: Outils fournis par l'application netinfo

Grâce à **netinfo**, le script d'initialisation est réduit au strict minimum en utilisant la commande :

```
netinfo --taygaconf $TAYGA_POOL_ADDR $TAYGA_POOL_NET
```

`TAYGA_POOL_ADDR` et `TAYGA_POOL_NET` sont des variables d'environnements transmises au lancement du conteneur.

4.3.1 En Pratique

Une image de conteneur déjà construite a été poussée sur le hub de docker sous le tag **selltom/nat64**. De cette manière, le lancement de l'application Tayga est maintenant facilité. Il suffit d'utiliser la commande suivante :

```
docker run -t --privileged selltom/nat64:amd64
```

Pour les personnes voulant utiliser l'application avec une configuration différente, les fichiers nécessaires à la construction de l'image de Tayga sont disponibles dans le dépôt github suivant : https://github.com/Sellto/docker_nat64.git

4.4 Orchestration

4.4.1 Docker swarm

La documentation de **Docker Swarm** annonce une compatibilité avec le monde IPv6. Pourtant la réalité est tout autre.

Docker Swarm déploie les conteneurs sur un *mode* de réseau particulier appelé *ingress*¹⁶. À sa création le réseau déploie un conteneur caché qui permet de faire le *load-balancing* des différents conteneurs entre les différents nœuds.

Le réseau *ingress* par défaut est *IPv4 only*, pour créer un réseau *dual stack* IPv4 - IPv6 sous **Docker Swarm**, il faut supprimer le réseau déjà prévu par défaut pour en recréer un autre. Pour exemple, la commande utilisée pour créer un réseau dual stack de type *ingress* :

```
docker network create --ipv6 --subnet=2001::/64 --subnet=172.30.0.0/16 --ingress myNet
```

Le problème c'est que le load-balancer n'a pas d'adresse IPv6. De ce fait, docker désactive l'IPv6. C'est un genre de *bug* présent dans une version qui n'est pas en bêta nous conduit à ne pas utiliser **Docker Swarm** pour la suite du projet.

16. Ce réseau est créé avec le driver overlay

4.4.2 Kubernetes

A l'heure actuelle, kubernetes ne permet pas de créer cluster d'applications avec des adresses ipv6. Cependant la version 1.13 devrait solutionner ce problème (version bêta prévue pour le 6 novembre). Pour réaliser les tests, un conteneur spécial englobant 6LBR et le nat64 a été crée. L' image de conteneur déjà construite à été poussée sur le hub de docker sous le tag **selltom/4LBR**. En outre, les fichiers nécessaire à la construction de cette image sont disponible sur la branche **4lbr** du dépôt github https://github.com/Sellto/6LBR_Docker.git

Création d'un cluster

Peu importe l'architecture des nœuds et le rôle qui leur sera attribué, deux *packages* devront obligatoirement être installés sur tous les nœuds :

- **docker** permettant de faire tourner les *pods* sur le nœuds
- **kubeadm** permettant d'amorcer le *cluster* ou le rejoindre.

Pour initialiser le cluster, on lancera sur le nœud sélectionné pour devenir le *master* la commande :

```
kubeadm init
```

Cette action aura pour conséquence le déploiement de 4 *pods* sur le master :

- **kube-apiserver** composant qui expose l'API de kubernetes
- **etcd** stockage sous la forme clé-valeur utilisé pour la sauvegarde de l'état du *cluster*.
- **kube-scheduler** composant qui attribue et sélectionne un nœud sur lequel sera exécuté un *pods* nouvellement créé.
- **kube-controller-manager** composant qui fait tourner le contrôleur qui surveille l'état du *cluster* via l'*apiserver* et effectue des modifications en tentant de déplacer l'état actuel vers l'état souhaité.

Pour connaître la commande à exécuter sur les autres nœuds qui souhaitent rejoindre le *cluster*, il suffit d'utiliser la commande suivante sur le *master* :

```
kubeadm token create --print-join-command
```

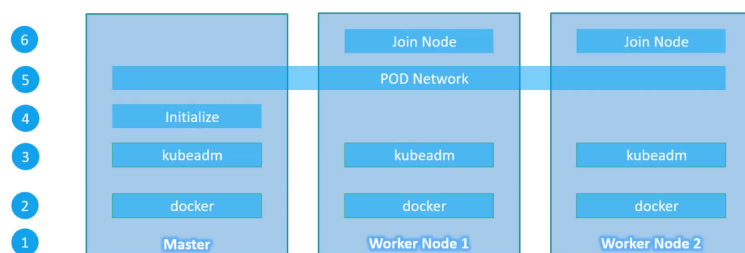
Mise en réseau

Pour que les *pods* puissent communiquer entre eux, il est nécessaire d'installer un *add-on*. Il en existe beaucoup, cependant seulement 3 garantissent un fonctionnement sur une cluster de nœuds à architecture hétérogène : Cilium, Flannel et Weave.

Cilium n'ayant jamais réussi à être déployé en raison de sa complexité d'installation et Flannel présentant des instabilités sur les nœuds à processeur ARM, Il a été décidé de travailler avec **Weave**.

Notre *cluster* est maintenant prêt a faire tourner des applications.

Steps



Conclusion

L'objectif premier du stage était de réaliser un *cluster* composé de nœuds à architecture hétérogène. Comme on peut l'observer sur la figure 5.12, dans son état actuel, deux nœuds sont à présent, dans le *cluster* :

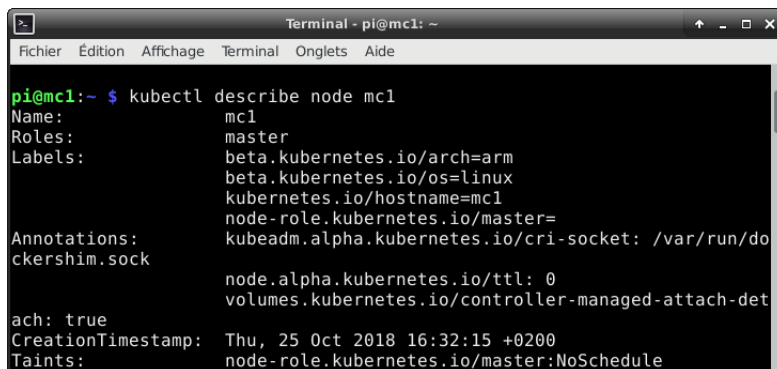
- un MacBook pro utilisant Linux dont le nom d'hôte est *masterone*
- un raspberry pi v3 dont le nom d'hôte est *mc1*



```
Terminal - pi@mc1: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
pi@mc1:~ $ clear
pi@mc1:~ $ kubectl get node
NAME          STATUS    ROLES    AGE   VERSION
masterone     Ready    <none>   4m14s v1.12.1
mc1           Ready    master   12d   v1.12.1
pi@mc1:~ $
```

FIGURE 5.12: liste des nœuds à présent, dans le *cluster*

Lorsque l'on regarde la description de ces deux nœuds (cf. Fig 5.14 et 5.13), on peut voir que l'architecture est disponible sous le label **beta.kubernetes.io/arch**. Cette information est très intéressante pour une des aspirations du travail de fin d'études : le lancement automatique de conteneurs sur certains nœuds du cluster.



```
Terminal - pi@mc1: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
pi@mc1:~ $ kubectl describe node mc1
Name:          mc1
Roles:         master
Labels:        beta.kubernetes.io/arch=arm
               beta.kubernetes.io/os=linux
               kubernetes.io/hostname=mc1
               node-role.kubernetes.io/master=
Annotations:   kubeadm.alpha.kubernetes.io/cri-socket: /var/run/do
               ckershim.sock
               node.alpha.kubernetes.io/ttl: 0
               volumes.kubernetes.io/controller-managed-attach-det
ach: true
CreationTimestamp: Thu, 25 Oct 2018 16:32:15 +0200
Taints:        node-role.kubernetes.io/master:NoSchedule
```

FIGURE 5.13: Description du nœud mc1


```
Terminal - pi@mc1: ~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
pi@mc1:~$ kubectl describe node masterone
Name:          masterone
Roles:         <none>
Labels:        beta.kubernetes.io/arch=amd64
               beta.kubernetes.io/os=linux
               kubernetes.io/hostname=masterone
Annotations:   kubeadm.alpha.kubernetes.io/cri-socket: /var/run/do
               ckershim.sock
               node.alpha.kubernetes.io/ttl: 0
               volumes.kubernetes.io/controller-managed-attach-det
               ach: true
CreationTimestamp: Wed, 07 Nov 2018 14:50:41 +0100
Taints:        <none>
Unschedulable: false
Conditions:
```

FIGURE 5.14: Description du nœud masterone

Un autre des objectifs était de faire tourner 6LBR sur le cluster. Dans notre cas il s’agit de la version modifiée rebaptisée 4LBR. Un fichier yaml (présenté dans les annexes) a été créé. Il suffit maintenant d’utiliser l’outil en ligne de commande **kubectl** pour lancer notre application depuis le *master* :

```
kubectl create -f 4lbr.yaml
```

Pour vérifier que le border router fonctionne bien, un *pod*s contenant l’application **emqx** a été déployé (ce *pod*s se base sur le fichier emqx.yaml placé dans les annexes.)

La figure 5.15 montre qu’un topic MQTT est bien créé dans l’application emqx sur base d’un client LWM2M.

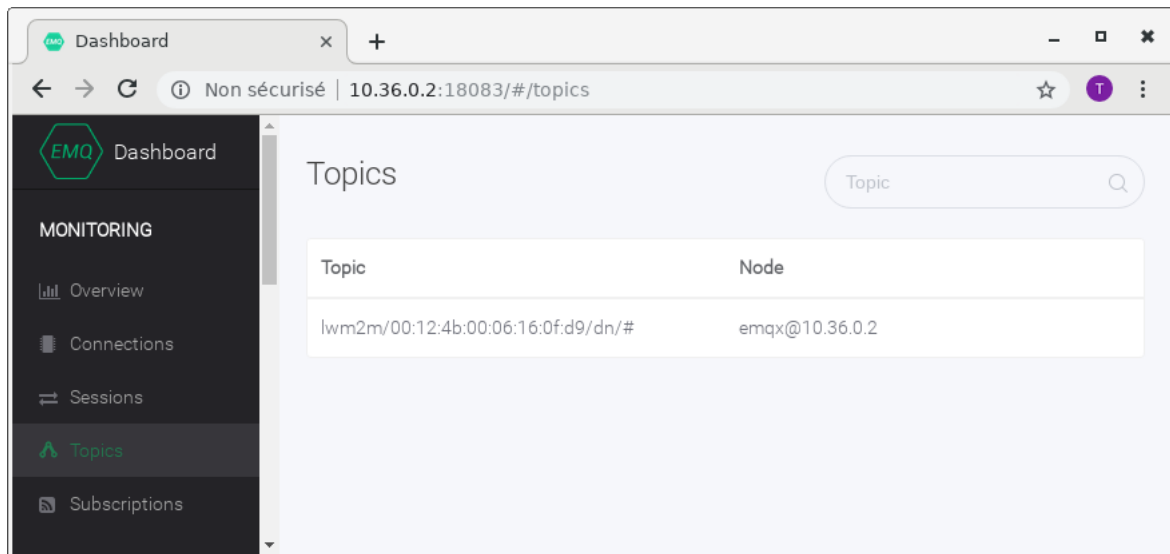


FIGURE 5.15: topic créé grâce au *plugin* LWM2M du broker EMQX

On peut donc s’accorder à dire que les objectifs fixés en début de stage sont tous atteints. Nous retrouvons bien un cluster avec au moins deux nœuds d’architecture différente sur lequel tourne l’application conteneurisée de 6LBR et un serveur LWM2M. Ce cluster représente une bonne base de départ pour le projet futur envisagée.

En outre, ces 6 semaines d’apprentissage dans le milieu de la recherche m’ont montré que les explorations réalisées sont toutes aussi importantes que le résultat final et qu’il était important de les documenter pour les personnes suivantes.

Credits

- <http://www.qualit-blog.com/introduction-a-la-conteneurisation/>
- <https://www.itpro.fr/wp-content/uploads/2015/12/04493437e0c5fab5497351217a3a07e0.jpg>
- <https://blog.alexis-hassler.com/2014/08/25/taille-images-docker.html>
- <https://github.com/cetic/6lbr/wiki>
- <i2.wp.com/blog.docker.com/wp-content/uploads/f753d4e8-9e22-4fe2-be9a-80661ef696a8-3.jpg>
- <https://gautric.github.io>
- <https://www.eclipse.org/leshan/>
- <https://www.emqx.io/>

Annexes

```
apiVersion: v1
kind: Pod
metadata:
  name: 4lbr
spec:
  containers:
  - name: 4lbr
    image: selltom/4lbr:amd64
    securityContext:
      privileged: true
    volumeMounts:
      - mountPath: /dev/ttyUSB0
        name: ttyusb0
    env:
      - name: LWM2MSERVER
        value: cccc::64:ad0e:303
    ports:
      - containerPort: 80
  nodeSelector:
    radio: 6lbr
    beta.kubernetes.io/arch : amd64
  volumes:
  - name: ttyusb0
    hostPath:
      path: /dev/ttyUSB0
```

4lbr.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: emqx
spec:
  containers:
  - name: emqx
    image: emqx/emqx:v3.0-rc.1
    ports:
      - containerPort: 8080
      - containerPort: 18083
      - containerPort: 11883
      - containerPort: 1883
      - containerPort: 8083
      - containerPort: 8084
      - containerPort: 8883
  nodeSelector:
    beta.kubernetes.io/arch : amd64
```

emqx.yaml