Rapport Technique Administration Réseaux - 2TL2 - Groupe 3

Louis Arys Geoffrey Brogniet Martin Perdaens
Jean-Michaël Tang

11 mai 2020

Table des matières

Responsable de mission et bilan (11 mai 2020)

Louis Arys

Bilan : Pour le moment, tout se déroule plutôt bien, nous n'avons pas rencontré de soucis lors de la prise de contact (c'est la première fois que nous travaillons tous ensemble). Au niveau de l'implémentation du projet, nous arrivons à bien nous répartir le travail, et en cas de soucis, on peut compter sur les autres pour nous donner un coup de main. Nous nous complétons bien au niveau de nos forces et nos faiblesses. Donc pour moi le bilan est très positif !

1 Introduction

Vous trouverez ci-après les différentes spécifications techniques des différentes technologies que nous utiliserons dans ce projet. Elles sont suceptibles d'évoluer ou d'être étoffées selon l'évolution de celui-ci.

2 Méthodologie

Nous utilisons plusieurs outils pour organiser le groupe :

— `Trello :` permet de nous répartir le travail, de mettre les ressources importantes et les liens de manière claire dans un tableau de ressources, ainsi qu'une version synthétisée des consignes

— `Github :` permet de centraliser le code, d'éviter les conflits lorsque nous travaillons ensemble sur le code. Nous l'utilisons aussi pour faire le wiki de notre groupe.

— `Overleaf :` pour éditer des rapports à plusieurs en LaTeX

Pour le développement des différents conteneurs, nous suivont une méthodologie assez simple :

— ***`Répartition du travail :`*** *Nous répartissons la responsabilité de créer les conteneurs entre les différents membres du groupe. Chacun doit veiller à la bonne réalisation du son/ses conteneurs, mais peut demander à d'autres de l'aide en cas de besoin.*

— ***`Développement d'un nouveau conteneur :`*** *Nous écrivons le code permettant de faire fonctionner le logiciel sans Docker. Une fois que le code est fonctionnel, nous écrivons un* ***Dockerfile****, permettant d'installer/copier les fichiers de configuration nécessaire dans une nouvelle image Docker. Nous testons ensuite le bon fonctionnement du conteneur, et écrivons des scripts permettant de le déployer automatiquement. Nous faisons des tests sur le conteneur pour s'assurer qu'il fonctionne correctement. Une fois un conteneur fonctionnel, nous* ***`pushons`*** *l'image de celui-ci sur notre* ***Docker Hub****.*

— ***`Déployement d'un nouveau conteneur :`*** *Pour déployer un conteneur sur un VPS, nous* ***`pullons`*** *l'image de notre* ***Docker Hub****(ou nous la construisons localement via notre projet Github), puis nous le déployons à l'aide de scripts écrit précédemment, et permettant un déployement automatique. Nous testons à nouveau le bon fonctionnement du conteneur, puis nous inspectons notre structure Docker pour vérifier que tout fonctionne correctement.*

— *`Update d'un conteneur` : Nous faisons les changements en local, en manipulant les fichiers de configuration de l'image. Une fois les changements effectuer, nous testons le conteneur sur un réseau local. Si tout est correct, nous pushons la nouvelle image sur notre* ***Docker Hub****.*

3 DNS

Nous avons décidé d'utiliser un container docker dans lequel tournera un serveur DNS Bind9. La raison de ce choix est la suivante : Bind9 est le serveur DNS le plus répandu sous Linux. Il y aura donc moyen d'accéder facilement à la documentation, et en cas de soucis, nous pourrons plus facilement trouver des solutions à nos problèmes. De plus, le fonctionnement de ce serveur est relativement simple à prendre en main.

Nous avons implémenté deux conteneurs Docker différents. Le premier permet d'héberger un SOA externe, permettant de faire la convertion en addresse IP des noms de domaine utile pour Woody-Toys. Le second permet de faire de la résolution DNS pour les postes internes de Woody-Toys. Il contient aussi les Ressources Records des différents postes et services seulement utilisable à l'intérieur du réseau interne de Woddy-Toys, et des serveurs ne devant pas être visible de l'extérieur.

Niveau d'avancement : A l'heure actuelle, nous avons implémenté un serveur Bind9 très basique et nous l'avons testé sur un réseau docker local avec 2 autres containers Ubuntu.

Le DNS a été configuré pour lier les hôtes et un script de déployement automatique a été mise en place pour faciliter l'utilisateur au niveau des manipulations de commandes.

Niveau d'avancement : *Les deux conteneurs ont été implémenté, et sont déployés sur nos VPS. Il ne manque que le Ressource Record nécessaire à la VoIP. Il sera rajouté quand celle-ci sera opérationnelle.*

4 Database

Nous avons décidé d'utiliser une database de type mysql car nous avons tous un minimum de connaissance dans les bases de données relationnelles. De plus, une base de données relationnelle correspondra exactement aux besoins d'une entreprise comme WoodyToys, nécessitant de pouvoir gérer des clients, des stocks, ...

Pour le choix de la base de données en elle-même, nous avons arrêté notre choix sur **MariaDB**. Ce choix découle de deux choses : d'une, mysql est une base de données facile à prendre en main et intuitive. De deux, MariaDB est un fork libre de mysql, et est actuellement quasiment pareil à celui-ci. Cela nous permet d'utiliser des outils *Open-Source*, ce qui nous tient à coeur.

Nous avons implémenté un conteneur avec une database MariaDB. Ce conteneur contient 2 tables : une pour l'intranet et l'autre pour le site b2b. Elle sera joignable par les deux sites directement mais ne sera pas accessible de l'extérieur.

Niveau d'avancement : Nous n'avons pas encore implémenté la database.

Niveau d'avancement : *l'image de la database a été complètement implémentée. Le travail est terminé, et le conteneur a été déployé.*

5 Websites

Nous avons décidé de partir sur une architecture très classique. Pour le **Front-end**, nous sommes partis sur du pur *HTML/JS/CSS*. Cela permettra de très rapidement obtenir quelques pages afin que le client puisse utiliser les fonctionnalités du Back-End.

*Pour le **Back-end**, nous nous sommes décidés d'implémenter un serveur Nodejs, pouvant être agrémenté si besoin est d'EJS pour faire du templating. Nous avons fait ce choix car nous avions de l'expérience dans l'implémentation de ce type de serveur. NodeJS est également efficace pour le routing et le déployement d'API.*

Niveau d'avancement : Nous avons actuellement implémenté un serveur très basique renvoyant un hello-world au navigateur. Ce serveur a, bien entendu, été containerisé et testé.

Niveau d'avancement : *Nous avons implémenté un serveur NodeJS reverse-proxy et deux serveur NodeJS faisant office de sites web. Pour terminer, il nous faut encore implémenter la connection HTTPS.*

6 Host

Nous avons implémenté différents conteneurs, permettant de tester les différents services de Woody-Toys. Nous implémenterons : un host ayant des outils DNS, un host ayant des outils pour tester les connections web, un host ayant des outils pour répurérer et envoyer des mails et un host pour tester la VoIP

Nous avons choisi d'implémenter un container très basique, contenant un noyau Ubuntu, et permettant, actuellement, d'effectuer des ping sur d'autres machines, afin de pouvoir tester le serveur DNS. Ce container sera étendu au besoin pour nous permettre de tester nos différents services. À terme, ce container servira aussi à émuler les machines se trouvant chez WoodyToys.

Niveau d'avancement : terminé pour les besoins actuels du projet.

Niveau d'avancement : *L'host DNS et l'host Web ont été implémentés et déployés. Il reste à développer l'host Mail et l'host VoIP.*

7 VoIP

Au terme de cette semaine de début de projet, nous ne pouvons nous exprimer sur le VoIP pour la raison simple que le cours abordant le sujet n'a pas encore eu lieu. Nous pouvons juste dire que nous le placerons sur un VPS à part car nous voulons séparer les différents services les uns des autres.

Nous n'avons pas encore commencé le développement de cette partie.

8 Mails

Le service mail a été configuré et est fonctionnel. Grâce à la bonne documentation de tvial, la mise en place a été assez simple une fois le SOA déployé.
Ce docker contient :

- *Postfix qui est un serveur mail utilisant sendmail permettant les échanges d'emails.*
- *Dovecot qui est un serveur IMAP sécurisé et open-source.*
- *saslauthd avec ldap auth qui est un processus daemon qui gère les authentification par texte et a comme rôle d'être un serveur isolant les codes des administrateurs en processus unique, et de permettre une authentification de services proxy facile pour les clients qui ne connaissent pas SASL.*
- *Amavis qui est une interface optimisé entre le MTA, le scanneur de contenu (comme les virus) et également SpamAssassin (qui une plateforme anti-spam open-source permettant aux administrateurs de filtrer des emails et bloquer les spams).*
- *ClamAV qui est un anti-virus open-source détectant les chevaux de Troie, les virus et les logiciels malveillant.*
- *OpenDKIM qui est un "Domain Key Identified Mail" open-source permettant de fournir ce service aux sendmail MTA malgré le filtre du protocole milter.*
- *OpenDMARC qui est un projet communautaire permettant de fournir des packages open-source DMARC.*
- *fail2ban qui est une application qui réduit le nombre de tentative de connexion.*
- *Fetchmail qui permet d'extraire des mails sur des serveurs de messageries distant vers le système de livraison de la machine locale. Fetchmail peut être exécuté en daemon pour sonder un ou plusieurs systèmes selon une intervalle spécifiée. Les courriers peuvent être récupéré par tout serveur utilisant POP2, POP3, IMAP2bis, IMAP4, and IMAP4rev1. Normalement, fetchmail utilise des liens TCP/IP pour la sécurité, mais peut aussi utiliser SMTP si des sites refusent pour des raisons de sécurités qui leur est propre, ce qui rend fetchmail plus flexible à ce niveau là.*
- *Postscreen qui est une défense multi-couche permettant entre autre de protéger le serveur contre la surcharge en laissant passer les mails de clients légitimes et en filtrant les spambots. Dans un déploiement typique, postscreen gère le service MX en TCP sur le port 25 pendant que la messagerie client servira sur le port 587 qui nécessite une authentification. Postscreen met temporairement les clients authentiques dans une liste blanche ceux qui passent le test d'authentification. En faisant cela, le traffic devient moins lourd sur le serveur puisqu'ils passent directement.*
- *Postgrey qui utilise une méthode de greylisting afin de limiter les spams. Pour cela, chaque requête reçu par Postfix via SMTP sera vérifié par le tercet CLIENT_IP / SENDER / RECIPIENT. Quand le premier tercet a été vu pour la première fois ou il y a moins de 5min, le mail sera rejeté avec une erreur temporaire. Le greylisting est une méthode permettant de bloquer une grosse quantité de spam au niveau du serveur mail sans avoir recours à une analyse exhaustive sur le serveur ou une analyse heuristique qui alourdit le serveur. Le greylisting se base sur le fait que la plupart des spams ne se comportent pas de manière "normale" dans les systèmes mail. Relativement efficace, il est tout de même préférable qu'il soit combiné avec d'autres logiciels anti-spam et anti-virus.*

— *Sieve qui est un support permettant de filtrer tout les emails et autorise la réception ou l'envoi de mails selon le titre du mail en question.*
— *LetsEncrypt qui permet d'obtenir des certificats de confiance pour les navigateurs, ce qui rends les serveur HTTPS disponible, sans avoir recours à une intervention humaine.*
— *Un script setup.sh qui permet de faciliter la configuration du serveur mail.*
— *Des données persistantes qui permet de faire des backups en cas de problèmes.*
— *Des tests d'intégration pour vérifier l'intégrité du serveur.*

9 Firewall

Actuellement nous n'avons pas encore commencé à faire un conteneur firewall.

Au terme de cette semaine, nous ne pouvons pas encore nous prononcer sur le firewall vu que la sécurité n'est pas encore de mise.

10 Problèmes rencontrés

10.1 Pour le DNS

Il a fallu comprendre comment marchait Bind9 ainsi que la manière de le containeriser proprement. Mais pour le moment tous les problèmes rencontrés ont été résolus.

Un autre soucis a été de faire en sorte que le DNS externe soit bien joignable par l'extérieur du VPS, ainsi que le choix de l'architecture. Le nombre de d'architecture possible pour un réseau DNS est élevée. Il nous a donc fallu faire un choix

`Solution` : faire des recherches, réfléchir à toutes les possibilités d'architectures.

10.2 Pour le serveur NodeJS

Il a falu implémenter un conteneur NodeJS faisant office de reverse-proxy, ce qui n'avait jamais été fait par les membres du groups.

`Solution` : faire des recherches. Nous avons trouvé un module NodeJs permettant de configurer un reverse-proxy.

10.3 Pour le serveur mail

La documentation étant relativement complète, il n'y a pas eu beaucoup de soucis. Le seul moment où il y a eu quelques petits soucis mineurs était de bien configurer Thunderbird pour que le mail fonctionne, mais cela a été très vite réglé en changeant le nom d'utilisateur qui faussait les données de l'email.

10.4 Pour les outils d'organisation

Il a fallu un petit temps pour les prendre en mains pour ceux qui ne les avaient pas encore utilisés. Cela a été vrai surtout pour Overleaf(LaTeX) qui demande un petit temps pour le prendre en main.

11 Architecture

Nous avons réparti l'architecture de notre réseau sur les trois des 4 VPS que nous possédons. Vous pourrez retrouver la description de nos choix ci-dessous.

- *51.178.41.115 : Nous avons regroupé sur ce VPS tous les conteneurs en rapport avec l'architecture interne de Woody-Toys. Cela comprend : les postes utilisateurs, le DNS (résolveur interne), l'intranet, le SOA externe et la database.*
- *51.178.41.16 : Nous avons regroupé sur ce VPS tous les conteneurs en rapport avec les sites web de Woody-Toys. Cela comprend : le site public www.wt2-3.ephec-ti.be, le site pour les revendeurs b2b.wt2-3.ephec-ti.be et un serveur faisant reverse-proxy.*
- *51.178.41.7 : Nous avons regroupé sur ce VPS tous les conteneurs en rapport avec les services nécessaire à Woody-Toys. Cela comprend : le serveur mail et le serveur VoIP.*

Nous avons choisi cette architecture pour plusieurs raisons :

- *Premièrement, cela permettra d'isoler les postes internes et la base de données de Woody-toys de l'extérieur.*
- *Deuxièmement, nous avons réunis ensemble, dès que possible, les conteneurs du même type afin de facilité les déployements, et la maintenance.*

La DMZ, présente sur tous les schémas est en réalité un réseau Docker Swarm reliant tous les VPS, sur lequel tourne un réseau Overlay, reliant les différents éléments devant être dans la DMZ.

11.1 VPS 51.178.41.115

Nous avons réunis tous les éléments du réseau interne de Woody-Toys pour simuler l'isolation du réseau. Ils sont sur un réseau Docker avec aucunes liaisons avec l'extérieur. Cela permet 2 choses : premièrement, tous les postes et le DNS interne sont insolés de l'extérieur. Deuxièmement, l'intranet sera bien juste joignable par les postes internes de Woody-Toys.

En plus des éléments internes nous avons rajouté 2 éléments de la DMZ : le DNS externe et la base de données.

- *Pour le DNS externe, c'est surtout pour une raison de facilité : en mettant les 2 DNS sur le même VPS, cela facilite le déployement des conteneurs.*
- *Pour la base de données, c'était une nécessité. La mettre sur ce VPS en particulier permet de la relier à la fois au réseau Ovelay de la DMZ, et au réseau Docker interne. Ainsi, l'intranet a accès à ladite database sans devoir relier celle-ci à l'extérieur du VPS. Elle sera donc isolée.*

Nous avons relié le port 53 du conteneur SOA externe avec le port 53 du VPS pour permettre celui-ci de fonctionner correctement.

11.2 VPS 51.178.41.16

Nous avons réunis les deux conteneurs Webs. La raison de ce choix est pûrement empirique, nous voulons avoir les mêmes types de conteneurs ensemble. Comme il n'était pas possible de lier en même temps les deux conteneurs au port 80 (ou 443 pour la connection HTTPS), nous avons dû rajouter un troisème conteneur qui ferait office de reverse-proxy.

Sections	Subnet	Masque
Service informatique interne	172.20.0.0 /24	255.255.255.0
Direction	172.20.10.0 /24	255.255.255.0
Secrétariat	172.20.11.0 /24	255.255.255.0
Commerciaux	172.20.12.0 /24	255.255.255.0
Comptabilité	172.20.13.0 /24	255.255.255.0
Hangar	172.20.14.0 /24	255.255.255.0
Atelier	172.20.15.0 /24	255.255.255.0
DMZ	10.0.0.0 /16	255.255.0.0

TABLE 1 – **Table d'adressage**

Ainsi, le port 80 du conteneur proxy est relié au port 80 du VPS. Et les requêtes pour les sites sont redirigées par le proxy vers les bons conteneurs. Les conteneurs web écoutent quant à eux leur port 8080, et ne sont pas directement accessible de l'extérieur.

Les 3 conteneurs tournent sur le réseau Overlay de la DMZ. Cela leur permet d'accéder à la base de données tournant sur le VPS 51.178.41.115, vu qu'un réseau Docker Overlay permet de réunir sur un même réseau docker plusieurs hôtes Docker distant.

11.3 VPS 51.178.41.7

Nous avons réunis les différents conteneurs de services. De nouveau, la raison de ce choix est empirique car nous désirons rassembler ensemble les mêmes type de conteneur. Les deux conteneurs tournent sur le réseau Docker Overlay de la DMZ.

Dans le cas du conteneur mail, les ports 25, 143, 587 et 993 qui sont respectivement reliés aux ports 25, 143, 537, 993 du VPS.

12 Monitoring

Pour surveiller le bon fonctionnement du réseau, il y a plusieurs manières. Docker propose différents outils permettant de faire cela.

- *Docker permet de lister les réseaux présent sur un Docker host à l'aide de la commande* `docker network ps`
- *Docker permet de voir l'état des réseaux présent sur un Docker host grâce à la commande* `docker network inspect <nom_du_réseau>`
- *Docker permet aussi de lister les conteneurs tournant sur un Docker host à l'aide de la commande* `docker ps`
- *Docker permet de voir l'état d'un conteneur docker grâce à la commande* `docker inspect <nom_du_conteneur>`
- *Pour ce qui est de vérifier le SOA, on peut vérifier les configuration à l'aide de* ***dig****.*
- *Pour vérifier le DNS interne, il faudra utiliser le conteneur dnsutils, qui contient l'utilitaire* ***dig*** *et* ***dnslookup****.*

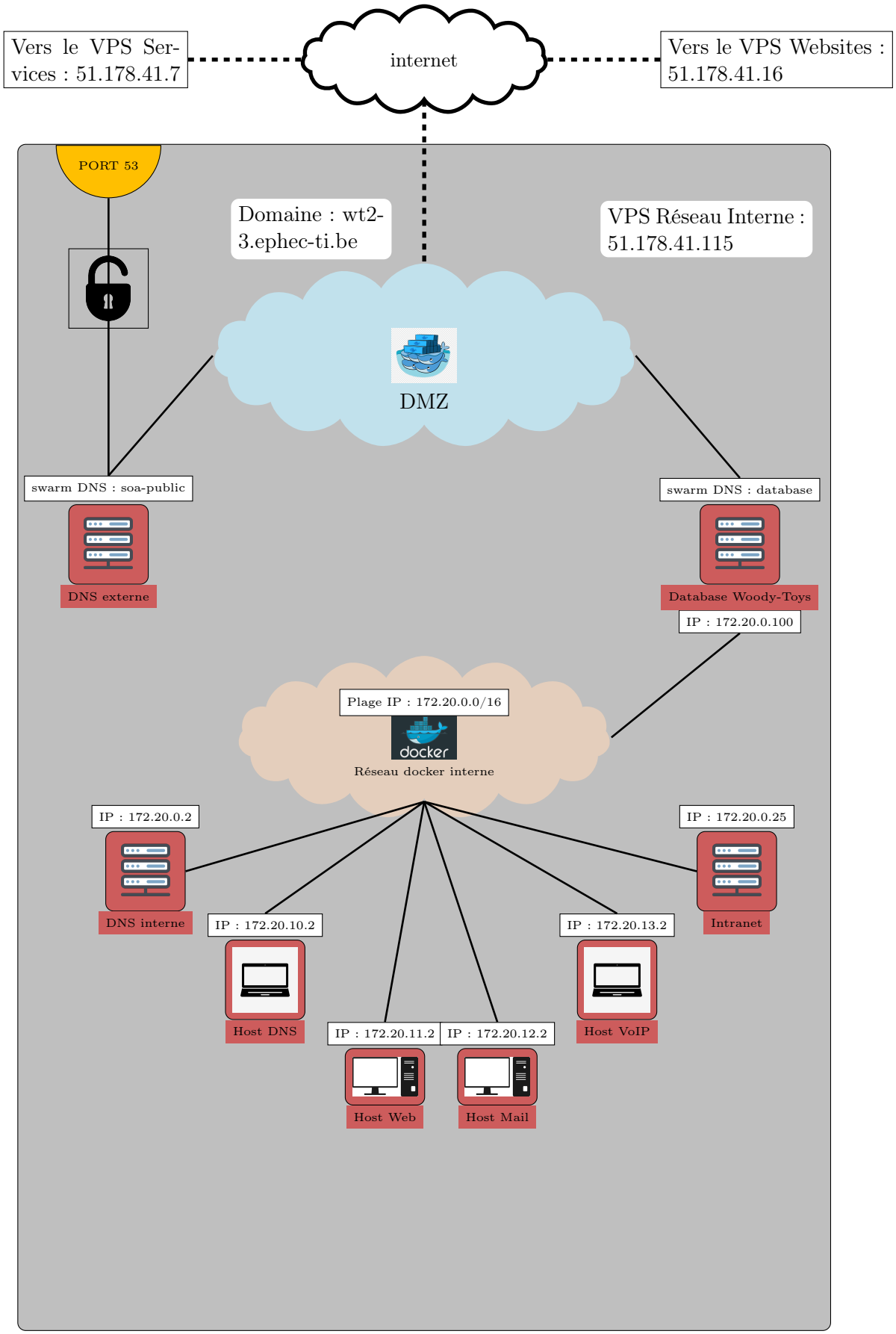
13 Plan d'adressage

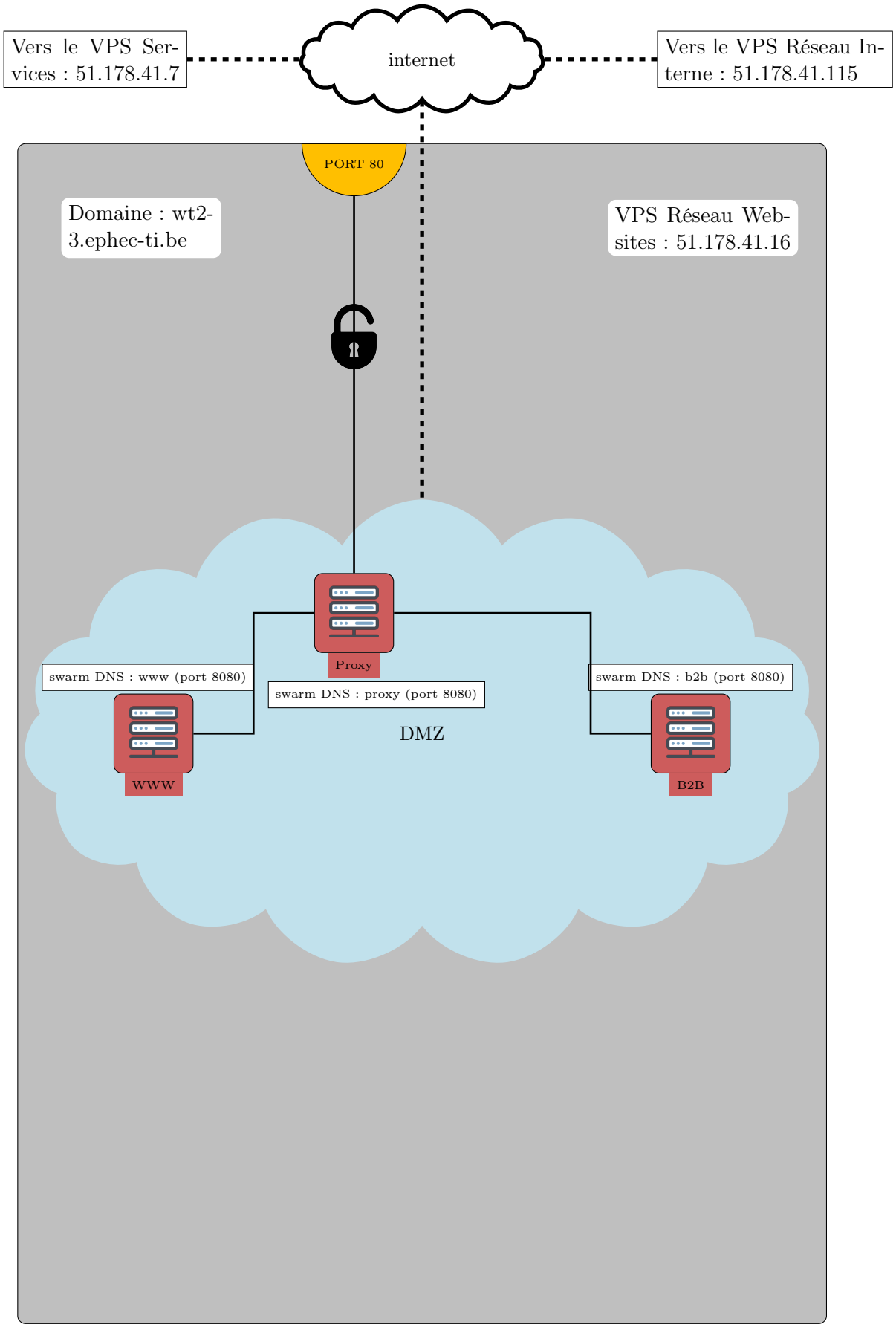
Le réseau a été divisé en différents sous-réseaux. Nous avons fait un sous-réseau par section de l'entreprise Woody-Toys. Cette subdivision permettra de plus facilement gérer les permissions et les communications entre les différentes entités composant l'entreprise. Nous avons aussi attribuer un sous-réseau pour la gestion des services informatiques internes.

Schéma du prototype

Légende pour les 3 schémas prototypes :

— **En rouge :** ce sont les machines terminales du réseau. Cela prend en compte les serveurs, les portables et les ordinateurs fixes.
— **En jaune :** ce sont les ports du VPS.
— **En bleu :** le réseau swarm qui relie les différents VPS.
— **En blanc :** les différentes informations techniques, et les éléments extérieurs au VPS.
— **En orange :** les réseaux internes Docker.


Vers le VPS Services : 51.178.41.7
internet
Vers le VPS Websites : 51.178.41.16
PORT 53
Domaine : wt2-3.ephec-ti.be
VPS Réseau Interne : 51.178.41.115
DMZ
swarm DNS : soa-public
DNS externe
swarm DNS : database
Database Woody-Toys
IP : 172.20.0.100
Plage IP : 172.20.0.0/16
docker
Réseau docker interne
IP : 172.20.0.2
DNS interne
IP : 172.20.10.2
Host DNS
IP : 172.20.11.2
Host Web
IP : 172.20.12.2
Host Mail
IP : 172.20.13.2
Host VoIP
IP : 172.20.0.25
Intranet


Vers le VPS Services : 51.178.41.7
internet
Vers le VPS Réseau Interne : 51.178.41.115
PORT 80
Domaine : wt2-3.ephec-ti.be
VPS Réseau Websites : 51.178.41.16
Proxy
swarm DNS : www (port 8080)
swarm DNS : proxy (port 8080)
swarm DNS : b2b (port 8080)
DMZ
WWW
B2B

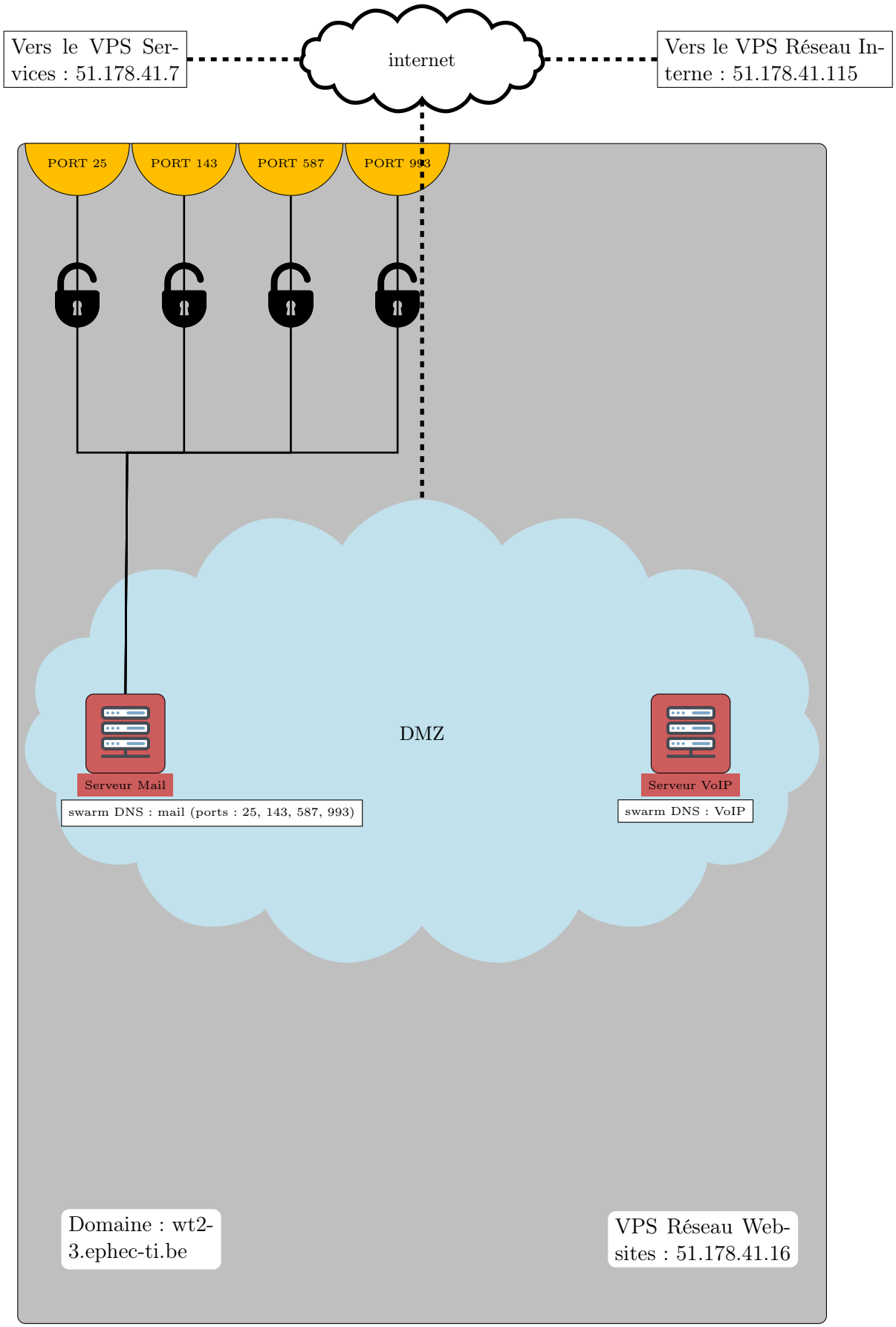
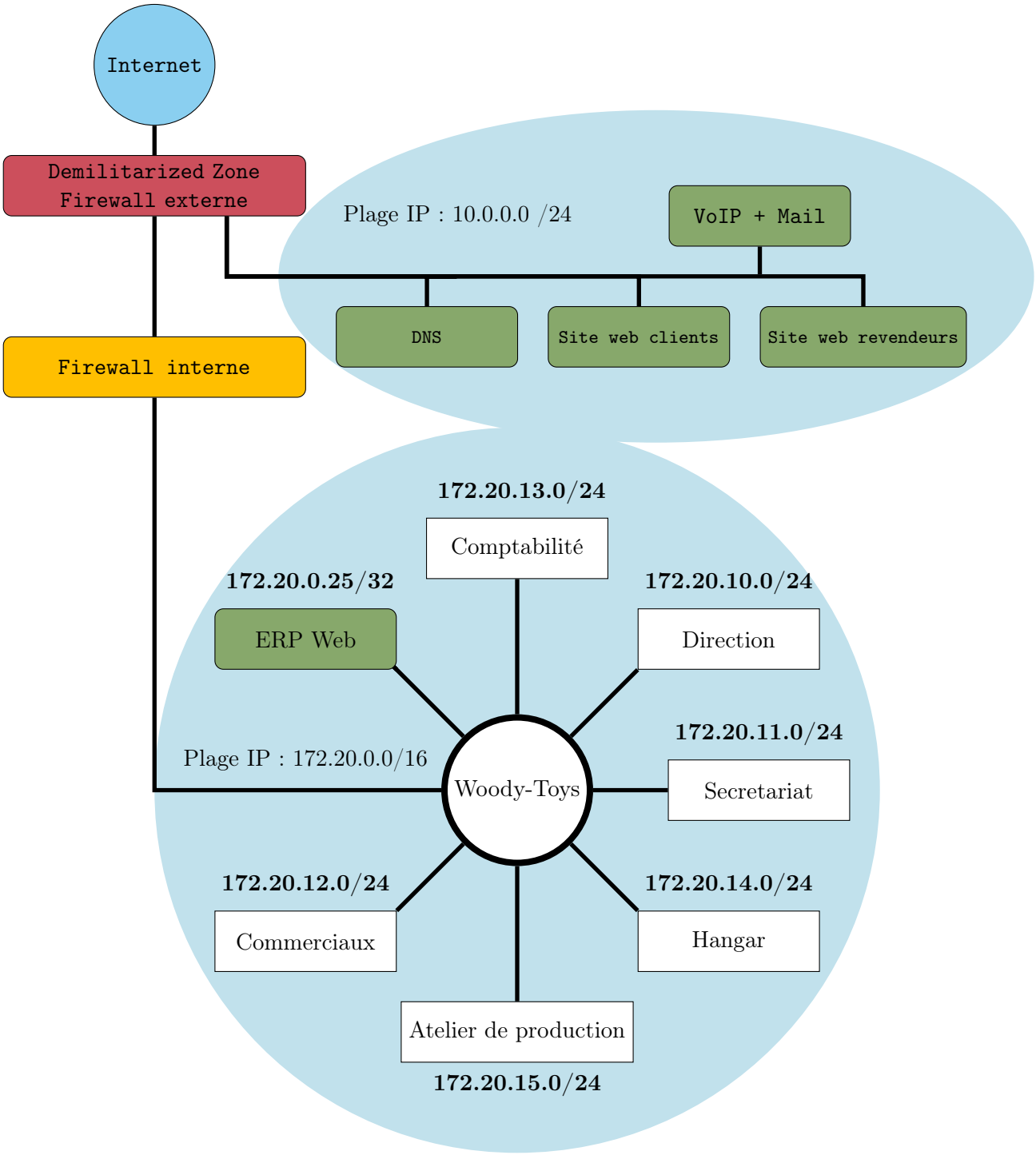

Vers le VPS Services : 51.178.41.7
internet
Vers le VPS Réseau Interne : 51.178.41.115
PORT 25
PORT 143
PORT 587
PORT 993
DMZ
Serveur Mail
swarm DNS : mail (ports : 25, 143, 587, 993)
Serveur VoIP
swarm DNS : VoIP
Domaine : wt2-3.ephec-ti.be
VPS Réseau Websites : 51.178.41.16

Schéma de Woody-Toys

— **Cercle bleu clair :** Zone réseau interne de Woody-Toys.
— **Ellipse bleue claire :** Zone réseau de la DMZ de Woody-Toys.
— **Rectangle vert :** Les différents serveurs contenant les services Woody-Toys.
— **Rectangle blanc :** Les différents subnets réseaux dans le réseau interne de l'entreprise Woody-Toys.



Notre architecture se base sur une architecture incluant deux firewalls et une DMZ(Demilitarised Zone). Le but de cette architecture est de séparer les ressources accessible de l'extérieur de celles ne devant être seulement disponible en interne. Cela permet de mettre deux couches de sécurité à notre architecture : un premier firewall permettant de se protéger des attaques de l'extérieur, mais laissant passer toutes les demandes/requêtes venant de l'internet. Un second, effectuant un filtrage stricte des requêtes allant vers l'intérieur, et empêchant ainsi n'importe qui de rentrer dans le réseau interne de l'entreprise. C'est aussi pourquoi nous avons mis le serveur `ERP Web` à l'intérieur, car il n'est pas nécessaire qu'il soit accessible de l'extérieur.