

GROUPE

Gaétan MORLET

Maxence GRIAS

Maxence PELIGRY

Paul RIVIERE

2021

4PROJ – IoT and the new consumption modes of the future

*Rendu technique sur la partie infrastructure mise en place
dans le cadre d'un POC.*

RENDU AVANT LE : 04/07/2021

SUPINFO TOURS

Site retenu : brilliant-market.com



Sommaire

1. Architecture	4
A. Schéma de l'architecture réseau	4
B. Schéma de l'architecture logique	5
C. Logiciel permettant de schématiser l'architecture	5
2. Environnement	6
A. Choix des solutions techniques	6
1. Hyperviseur	6
2. Pare-feu et routeur	6
3. OVH	6
4. CloudFlare	7
5. GitHub	7
6. Azure	7
7. Docker	7
8. Kubernetes	7
9. Terraform	8
10. Ansible	8
11. Grafana	8
B. Convention de nommage des équipements	8
C. Solution de sauvegarde	9
3. Configuration	10
A. Réseaux	10
1. Fortigate	10
2. VMware	12
B. Nom de domaine et protection web	13
1. OVH	13
2. CloudFlare	14
C. Infrastructure	14
1. Azure	14
2. VMware	15
D. Intégration continue	15
1. GitHub	15
2. Terraform	16
3. Ansible	18
E. Serveurs et services	20



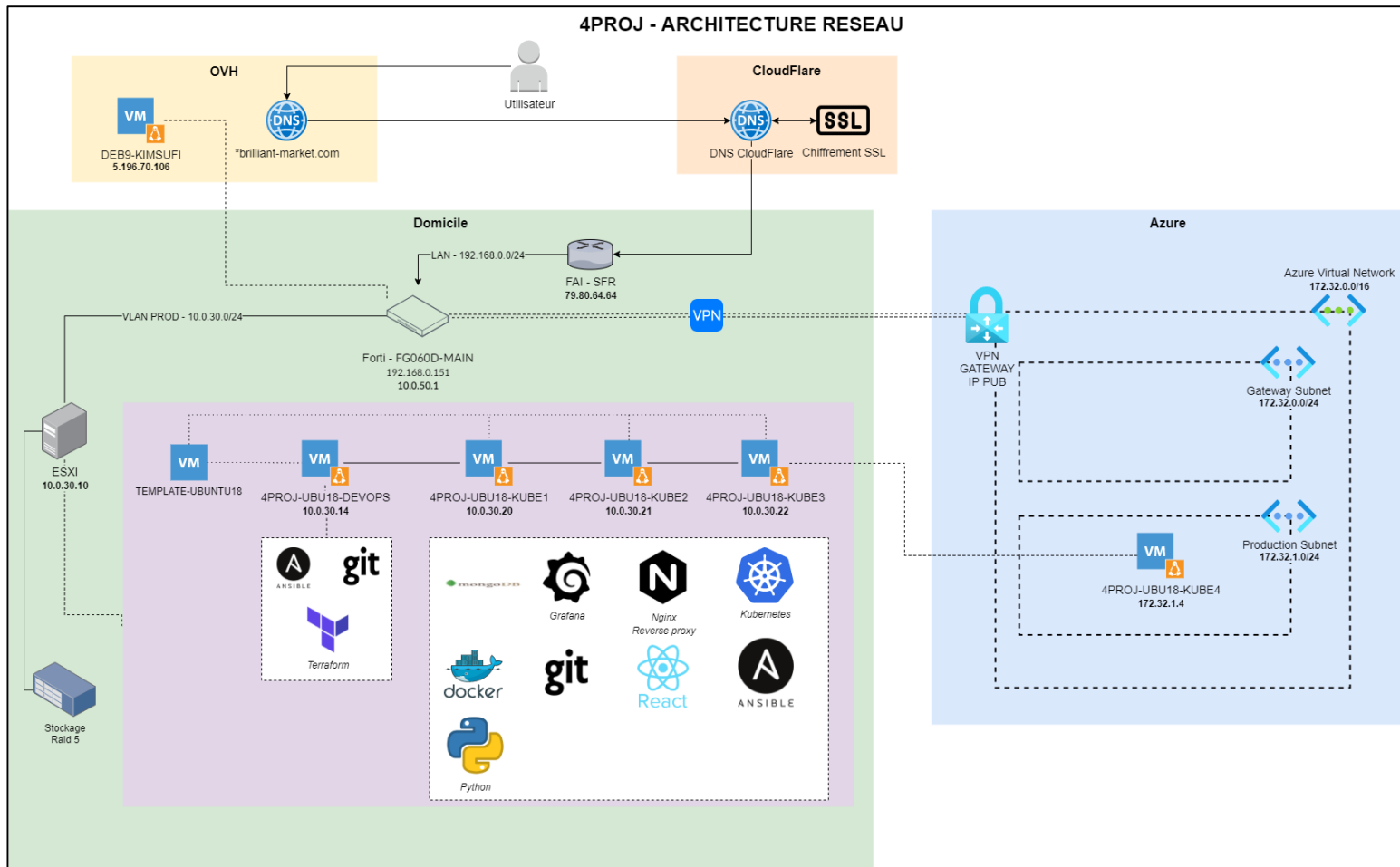
1. Template VMware	20
2. Serveurs et rôles	21
a. Kubernetes	21
b. Kubernetes - MongoDB	21
c. Kubernetes – WebApps et APIs.....	23
d. Kubernetes - Nginx	24
e. Kubernetes - Grafana.....	26
F. Sécurité et sauvegarde.....	27
1. Journaux d'évènements	28
2. Sauvegardes.....	28

Note : Pour accéder aux différentes machines virtuelles ou à l'infrastructure via le vpn merci de contacter Maxence Péligny sur TEAMS ou maxence.peligry@supinfo.com.



1. Architecture

A. Schéma de l'architecture réseau

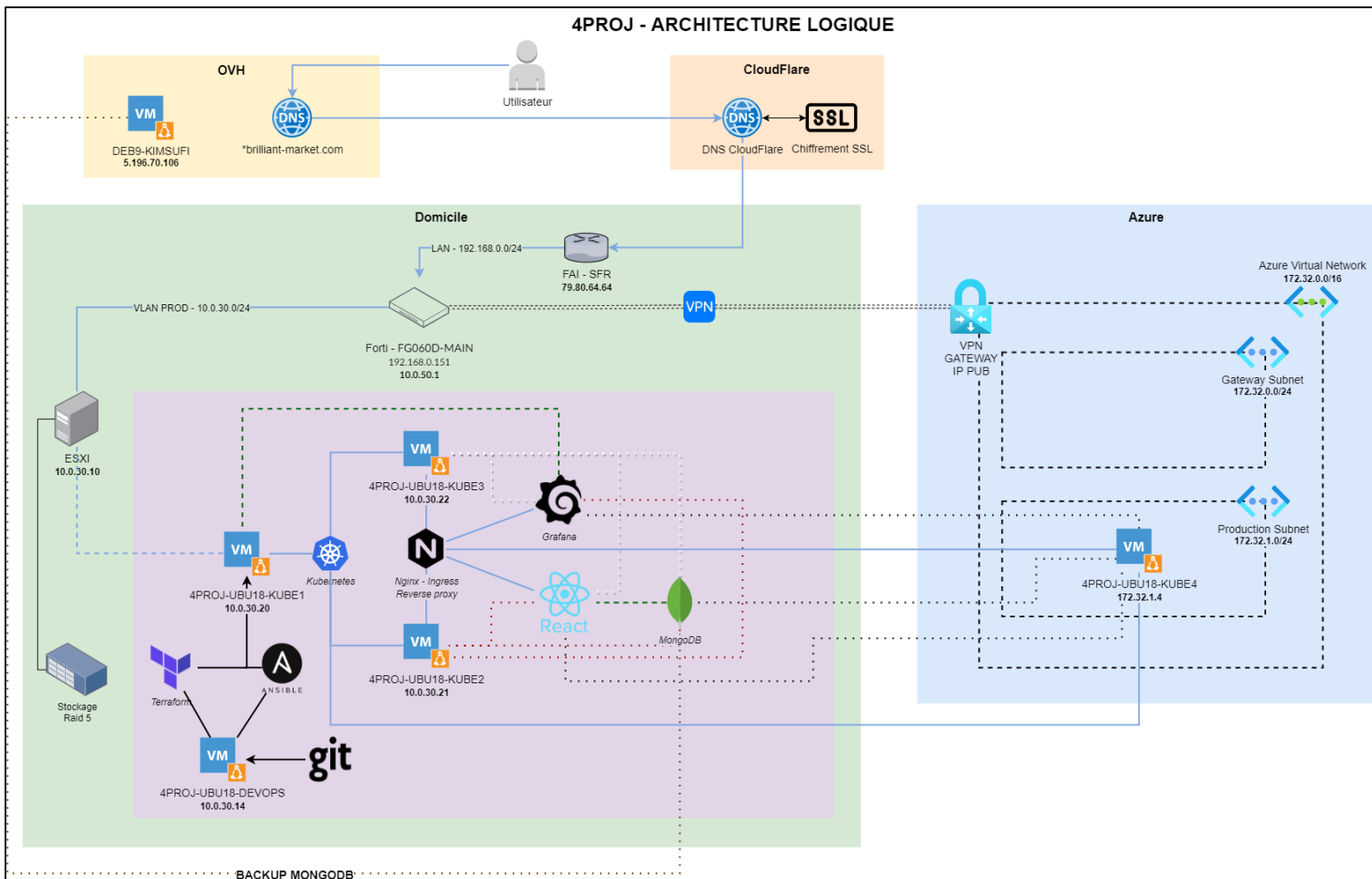


Note : Pour une meilleure clarté : l'ensemble des illustrations sont présentes dans le rendu .zip.

Notre architecture est composée de plusieurs briques. On peut y retrouver une brique Cloud avec Azure, une brique on premise avec un fournisseur d'accès internet, un pare-feu Fortigate et un serveur Dell faisant office de serveur ESXi. Pour l'hébergement de notre site nous avons acheté un nom de domaine chez OVH qui pointe vers les serveurs DNS de CloudFlare et ce dernier redirige le trafic vers notre infrastructure on-prem.



B. Schéma de l'architecture logique



Ce schéma présente d'un point de vue opérationnel l'ensemble des connexions et des liens entre les infrastructures, les applications, l'hébergement web, etc. Cela illustre le fonctionnement complet de notre solution.

C. Logiciel permettant de schématiser l'architecture

Draw.io est un outil gratuit permettant de dessiner (comme son nom l'indique) divers schémas et plus particulièrement des schémas réseaux avec des bibliothèques d'images. Dans notre projet nous avons utilisé la bibliothèque Citrix, Cisco, Azure, VMWare et le reste a été fait par nous-même.



2. Environnement

A. Choix des solutions techniques

1. Hyperviseur

Nous avons réfléchi sur l'hyperviseur qui était le plus adéquat aux besoins de la nouvelle infrastructure. Et c'est tout naturellement que notre choix s'est porté sur VMWare. Hyper-V ne correspondait pas à nos attentes notamment sur la partie template des machines virtuelles que nous avons créées et le déploiement via Terraform. De plus VMWare présente une très bonne documentation en ligne et une communauté importante. Le choix de VMWare et plus précisément la solution ESXi répondait à notre besoin avec des licences déjà acquises. De plus d'un point de vue compétence technique notre équipe avait déjà travaillé avec cet hyperviseur. Le choix de proxmox aurait pu être justifié, cependant le déploiement avec Terraform n'a pas été jugé satisfait par notre équipe et ce malgré son prix qui est gratuit. VMWare est donc le compromis parfait avec toutes nos attentes et c'est le produit qui répondra au mieux à la nouvelle infrastructure cliente.

Les spécificités concernant le serveur qui accueille l'hyperviseur ESXi est le suivant :

Serveur : PowerEdge T310 Server de Dell

Processeur : Intel Xeon 2.2GHz

Mémoires : 32GB RDIMM 2666MT/s Dual Rank

Disques : 1 To x 3

2. Pare-feu et routeur

La gestion des règles de par-feu, la segmentation du réseau et la robustesse étaient des éléments important à prendre en considération pour le choix de notre par-feu. Nous avons fait le choix d'utiliser un Fortigate 60-D. Ce par-feu constitue une gamme professionnelle chez son constructeur Fortigate. Il inspire robustesse et sécurité. De plus l'ensemble des documentations techniques y est disponible gratuitement et mis à jour régulièrement sur leurs sites internet. Ce dernier propose également des mises à jours régulières de son système d'exploitation et une externalisation de la journalisation des évènements sur leur cloud. Enfin nous possédions déjà l'équipement dans notre parc informatique.

3. OVH

Le nom de domaine utilisé pour ce projet et que nous avons acheté est brilliant-market.com, ce dernier a été acheté sur OVH. OVH est une société Française mondialement reconnue pour son sérieux et ses offres avantageuses. Ce web provider a été retenue. Son interface et sa documentation ont permis d'implémenter les configurations nécessaires aux bon déroulement de notre projet. Nous avons également utilisé OVH pour acheter un vps (virtual private server) pour y stocker les sauvegardes de mongodb (chiffrées). Le prix de ce vps constitue une force par rapport aux concurrents sur le marché actuel.



4. CloudFlare

Il n'est plus à démontrer que les serveurs dns (domain name system) les plus rapides au monde sont ceux de CloudFlare, sa protection contre les attaques web et ses performances sur la mise en cache de leurs différents sites web hébergés montre leurs forces sur ce marché.

Nous avons utilisé cette solution pour la partie transfert des zones DNS. De plus ne nous a rien coûté étant donné que nous utilisons la version gratuite qui ne nous limite pas par rapport à nos besoins.

5. GitHub

L'ensemble des ressources qui ont permis de créer ce projet sont hébergées sur GitHub. Nous avons choisi GitHub car c'est une solution gratuite, sécurisée et connue. Nous avons créé une organisation privée sur laquelle l'ensemble des acteurs de ce projet ont pu héberger leurs ressources. Nous n'avons pas fait le choix d'utiliser GitLab ou un GitLab hébergé on-premise pour des raisons de performance. GitHub présente l'avantage d'être connecté via un connecteur au docker hub (nous en reparlerons en détail dans la partie sur Docker).

6. Azure

Azure était la solution cloud qui répondait à nos besoins. Nous avons utilisé Azure en plus de notre infrastructure on-premise pour avoir un cloud hybride. Ce choix s'est porté suite car certains membres du projet étaient certifiés Azure. Les compétences sur Azure des acteurs ont permis de choisir Azure plutôt que AWS. Les prix utilisés sur Azure ont été sources de discussion et d'accord pour préférer ce cloud public plutôt qu'à un autre.

7. Docker

L'ensemble de nos sites web ou applications sont conteneurisés pour permettre de les faire fonctionner sur tous les environnements possibles. En terme de ressources et de performance docker répond parfaitement au besoin. Le maintien et les montées de versions répondaient aux exigences de notre équipe. Nous avons utilisé le docker hub pour pousser nos images docker et pour pouvoir les récupérer sur d'autres environnements. Nous avons utilisé la version gratuite de docker même si nous aurions préféré la version payante pour éviter les limitations d'image privée. Le connecteur vers GitHub y est parfaitement bien intégré, c'est aussi un point avantageux qui nous a poussé à utiliser le docker hub.

8. Kubernetes

En lien avec docker nous avons utilisé Kubernetes pour héberger notre infrastructure. Kubernetes est l'orchestrateur de container le plus connu avec une documentation en ligne. Développé par Google il est aujourd'hui au cœur de beaucoup d'entreprises. Les cloud providers ont même des services



dédiés à Kubernetes. Notre ingénieur infrastructure a été formée sur cette technologie plutôt que docker swarm c'est également pour cela que nous avons gardé Kubernetes.

Nous avons utilisé Kubernetes on-premise avec la stack kubeadm plutôt que des cloud providers qui proposent des services comme ACS ou AKS pour des raisons tarifaires.

9. Terraform

L'outil Terraform nous a permis de déployer rapidement un ensemble de ressources (réseaux, machines virtuelles, adresse ip publique, ect.) sur Azure et VMware. De cette manière nous pouvons redéployer un environnement de pré-production, de test ou reconstruite depuis le début sur notre infrastructure grâce à cet outil. Pour maintenir l'infrastructure et ajouter des nouvelles ressources Terraform est très performant.

10. Ansible

Ansible permet le lancement de tâche sur différentes machines virtuelles. Cela permet notamment de lancer des tâches en parallèle sur des environnements linux. Il complète Terraform et permet de créer des utilisateurs, installer des paquets, configurer des paquets, etc. Ce processus d'automation nous a fait gagner un temps précieux. Comme pour Terraform cela s'avéra utile pour reconstruire ou construit un nouvel environnement.

11. Grafana

Grafana permet de surveiller notre infrastructure actuelle et d'afficher sous formes de graphiques, de diagrammes ou de chiffres les données reçues avec l'aide de Prometheus. Cette solution est open-source et gratuite c'est pourquoi nous avons choisi de l'utiliser. Nous aurions pu choisir Splunk mais son coût fut un frein, nous n'avons pas retenue cette solution.

B. Convention de nommage des équipements

La convention de nom est très importante dans l'informatique, que ce soit dans les programmes ou dans l'infrastructure les noms permettent de se repérer pour nous mais également pour les autres. Il était primordial d'avoir une convention de nom pour nos équipements.

Nous avons choisi la convention suivante : **XXXXX-XXXXX**

XXXXX : Le vert représente le système d'exploitation installé ainsi que la version du système d'exploitation. En l'occurrence **UBU18** pour Ubuntu 18.04.

XXXXX : Le bleu représente le rôle installé sur le serveur ainsi que son identifiant unique qui le différencie des autres serveurs s'il y'en a plusieurs.



Par exemple :

UBU18-	DEVOPS
UBU18-	KUBE1
UBU18-	KUBE2
UBU18-	KUBE3

C. Solution de sauvegarde

L'ensemble des ressources, codes et documents sont stockés sur GitHub. Il est donc tout à fait facile de redéploier l'infrastructure en cas de perte des ressources ou des applications. Toutefois les données ne doivent pas être perdus. Pour cela il faut sauvegarder régulièrement les documents mongodb. Les dump mongodb sont externalisés sur un serveur distant en plus du serveur local pour plus de sûreté.



3. Configuration

A. Réseaux

1. Fortigate

Le pare-feu physique Fortigate-60D est derrière une box internet SFR. La box internet SFR laisse passer l'ensemble des ports entrant sur l'IP 192.168.0.151 (Fortinet-60D sur le réseau domestique) à travers la DMZ. De cette manière c'est le Fortigate qui fait le rôle de routeur.

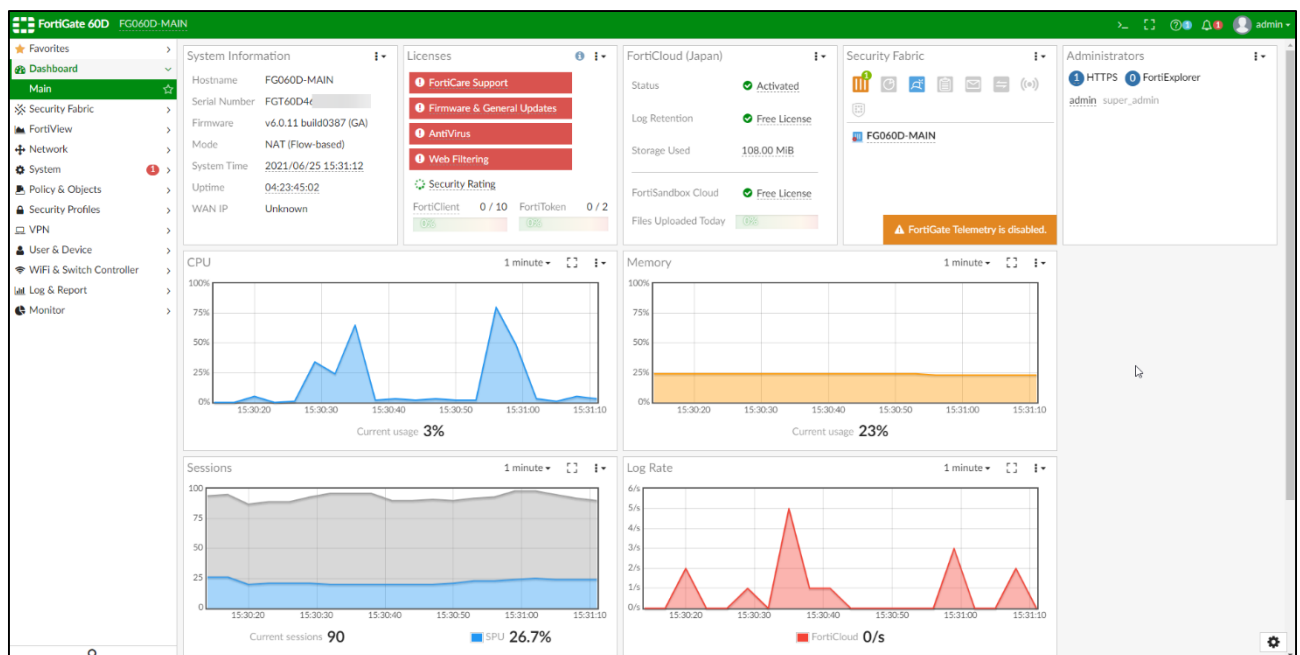
RÉSEAU / PARAMÈTRES AVANCÉS / DMZ

Répondre à un ping sur le port WAN ☒

Adresse DMZ : 192.168.0.151

Appliquer

Le fortigate est administrable sur l'url <https://10.0.50.1/ng/system/dashboard/1>.








Il y'a trois interfaces sur le Fortigate, une WAN relié directement à la box SFR, une interface1 relié à l'ESXi et interface2 relié à un NAS Synology.

FortiGate 60D				
INTERNAL 1 2 3 4 5 6 7 DMZ WAN1 WAN2				
+ Create New Edit Delete				
Status	Name	Members	IP/Netmask	Type
Physical (8)				
+	internal1 (PROD)		0.0.0.0 0.0.0.0	Physical Interface
-	VLAN-MANAGEMENT		10.0.50.1 255.255.255.0	VLAN
-	VLAN-PREPROD		10.0.40.1 255.255.255.0	VLAN
-	VLAN-PRODUCTION		10.0.30.1 255.255.255.0	VLAN
+	internal2 (SYNOLOGY)		10.0.20.1 255.255.255.0	Physical Interface
+	wan1 (SFR)		192.168.0.151 255.255.255.0	Physical Interface
-	IPSEC-AZURE		0.0.0.0 0.0.0.0	Tunnel Interface
-	IPSEC-FREEBOX		0.0.0.0 0.0.0.0	Tunnel Interface

Voici les différentes routes statiques configurés pour accéder au vpn site to site Azure ou à la box SFR.

<div><div>+ Create New</div><div><div>Edit</div><div>Clone</div><div>Delete</div></div></div>			
Destination	Gateway	Comment	Interface
0.0.0.0/0	192.168.0.1		 SFR (wan1)
172.16.0.0/24	0.0.0.0		 IPSEC-FREEBOX
172.32.0.0/16	0.0.0.0		 IPSEC-AZURE

Les différentes règles du pare-feu qui permettent de protéger, restreindre et autoriser les équipements de communiquer entre eux et vers internet. Ici les règles de pare-feu avec l'ID 34 et 35 permettent de rediriger le trafic entrant vers le cluster Kubernetes. Cela a pour objectif de renvoyer les utilisateurs accédant au nom de domaine brilliant-market de pointer vers l'ip publique de la box SFR et d'être aiguillé vers le cluster K8S où un reverse proxy nginx (avec un ingress) est mit en place afin de déterminer la bonne route en fonction du sous domaine utiliser.

ID	Name	From	To	Source	Destination	Schedule	Service	Action	NAT
WAN SFR 3/8									
34		SFR (wan1)	VLAN-PRODUCTION	all	VIP-BRILLIANT_MARKET.COM-HTTPS	always	HTTPS	ACCEPT	Enabled
35		SFR (wan1)	VLAN-PRODUCTION	all	VIP-BRILLIANT_MARKET.COM-HTTP	always	HTTP	ACCEPT	Enabled
22		SFR (wan1)	VLAN-MANAGEMENT	IP-EXT-VPS-KIMSUFU	VIP-FORTI-MANAGER	always	SSH	ACCEPT	Enabled
PRODUCTION 2/8									
1	PROD vers WAN	VLAN-PRODUCTION	SFR (wan1)	all	all	always	PING DNS HTTP HTTPS NTP	ACCEPT	Enabled
26		VLAN-PRODUCTION	VLAN-PREPROD	ADDR-PROD-ESXI	ADDR-PROD-DEVOPS	always	ALL	ACCEPT	Enabled
PRE-PRODUCTION 1/3									
32		VLAN-PREPROD	SFR (wan1)	ADDR-PP-DOCKER	IP-EXT-VPS-KIMSUFU	always	SSH-2152	ACCEPT	Enabled
VPN SSL 1/2									
36		SSL-VPN tunnel interface (ssl.root)	PROD (internal1) VLAN-PRODUCTION	SSLVPN_TUNNEL_ADDR1 VPN-GROUP-SUPINFO	ADDR-PROD-KUBE1 ADDR-PROD-KUBE2 ADDR-PROD-KUBE3	always	ALL	ACCEPT	Disabled
VPN IPSEC 2/4									
29		IPSEC-AZURE	VLAN-PRODUCTION	SUBNET-AZURE	SUBNET-PRODUCTION	always	ALL	ACCEPT	Enabled
30		VLAN-PRODUCTION	IPSEC-AZURE	SUBNET-PRODUCTION	SUBNET-AZURE	always	ALL	ACCEPT	Disabled

Enfin voici la configuration du VPN IPsec site to site qui permet de connecter notre infrastructure on-premise à Azure. Ce VPN site to site permet d'établir une connexion entre les réseaux azure privées où se trouve UBU18-KUBE4 et le réseau on-premise de production où se trouve les machines virtuelles UBU18-KUBE1, UBU18-KUBE2 et UBU18-KUBE3.



Name: IPSEC-AZURE

Comments:

Network Edit

Remote Gateway : Static IP Address (13.74.157.194) , Interface : wan1

Authentication Edit

Authentication Method : Pre-shared Key

IKE Version : 2

Phase 1 Proposal Edit

Algorithms : AES256-SHA1, 3DES-SHA1, AES256-SHA256

Diffie-Hellman Group : 2

Phase 2 Selectors

Name	Local Address	Remote Address
IPSEC-AZURE	0.0.0.0/0.0.0.0	0.0.0.0/0.0.0.0

Edit Phase 2 ✓ ↺

Name: IPSEC-AZURE

Comments:

Local Address

Subnet

Remote Address

Subnet

Advanced...

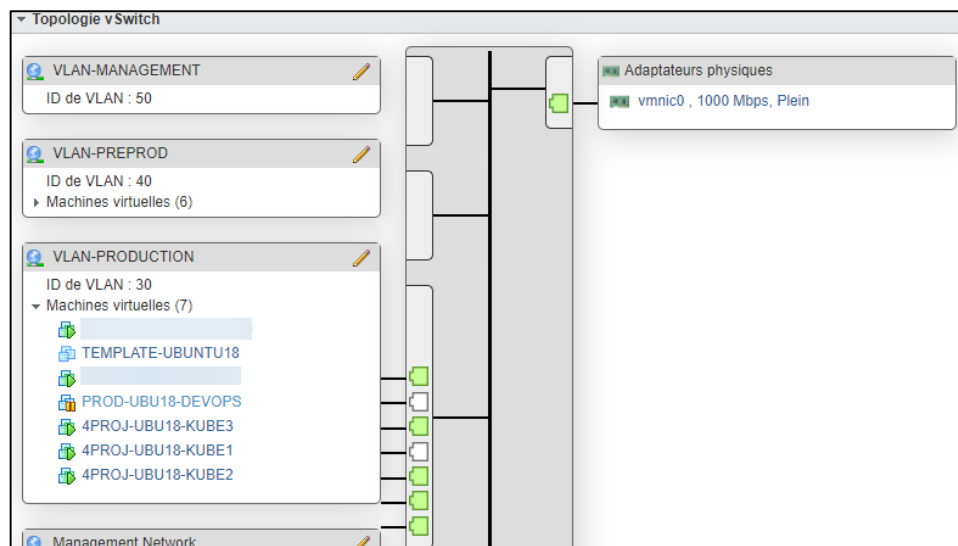
2. VMware

L'interface d'administration de VMware est <https://10.0.30.10/ui/#/login>.

The screenshot displays the VMware vSphere Client interface for a virtual machine named 'nc-ass-vip.sdv.fr'. The interface is divided into several sections:

- Summary:** Shows the VM's name, version (6.5.0), state (Normal), and uptime (116.89 jours).
- Hardware:** Lists the VM's configuration, including:
 - Fabricant: Dell Inc.
 - Modèle: PowerEdge T320
 - CPU: 6 CPUs x Intel(R) Xeon(R) CPU E5-2420 v2 @ 2.20GHz
 - Mémoire: 31.96 Go
 - Mémoire persistante: 0.0
 - Virtual Flash: 0.0 utilisée, 0.0 capacité
 - Mise en réseau:
 - Nom d'hôte: nc-ass-vip.sdv.fr
 - Adresses IP: 1. vmk0: 10.0.30.10, 2. vmk0: fe80::2247:47ff:fe97:83ac
 - Serveurs DNS: 1. 89.2.0.2, 2. 89.2.0.1
 - Passerelle par défaut: 10.0.30.1
 - IPv6 activé: Oui
 - Adaptateurs hôtes: 2
 - Réseaux: Nom: VLAN-MANAGEMENT, VM: 0
- Configuration:** Shows the VM's profile (ESXi-6.5.0-20170404001-standard), vMotion status (Pris en charge), and vSphere HA status (Non configuré).
- Informations sur le système:** Displays system information such as the date of installation (Samedi, 02 Mai 2020, 08:51:35 UTC), BIOS version (2.2.0), and BIOS publication date (Jeudi, 06 Février 2014, 01:00:00 +0100).
- Résumé de performances de la dernière heure:** A graph showing CPU and memory usage over the last hour.

La gestion des VLAN s'effectue au niveau des groupes de port sur le vSwitch. Nous avons utilisé la tag 30 pour utiliser le VLAN de production.



B. Nom de domaine et protection web

1. OVH

Le nom de domaine est brilliant-market.com, le renouvellement automatique a été désactivé volontairement pour transférer le projet sur un autre nom de domaine au besoin et ne pas devoir payer cet hébergement-là.

Voici la configuration de la zone DNS de brilliant-market.com :

Domaines / brilliant-market.com / Zone DNS

brilliant-market.com

Expire le 15 mars 2022 [Renouveler](#)

Informations générales **Zone DNS** Serveurs DNS Redirection DynHost GLUE DS Records

ⓘ Simplifiez-vous la vie et optez pour le renouvellement automatique de votre domaine ! Finies les relances et les suppressions inopportunes. [Activer le renouvellement automatique](#)

✓ Les entrées ont bien été supprimées de votre zone DNS.

✗ Nos systèmes ont détecté des erreurs pendant la vérification de votre zone. **Vos dernières modifications n'ont donc pas été prises en compte.** Afin d'y remédier, veuillez corriger les problèmes suivants :

1. zone brilliant-market.com/IN: has no NS records

Vous pouvez voir ici la configuration des diverses entrées de votre domaine.

Vous avez également la possibilité de configurer ces entrées pour relier votre domaine à vos différents services (bouton « ajouter une entrée »).

[Supprimer](#) Tous

<input type="checkbox"/> Domaine	TTL	Type	Cible	
<input type="checkbox"/> brilliant-market.com.	0	A	79.80.64.64	⋮
<input type="checkbox"/> api.brilliant-market.com.	0	A	79.80.64.64	⋮
<input type="checkbox"/> apiml.brilliant-market.com.	0	A	79.80.64.64	⋮
<input type="checkbox"/> monitor.brilliant-market.com.	0	A	79.80.64.64	⋮
<input type="checkbox"/> simulation.brilliant-market.com.	0	A	79.80.64.64	⋮



Et voici les serveurs DNS de CloudFare que nous avons configuré pour faire la bascule :

📁 Serveur DNS	IP associée	Statut
gloria.ns.cloudflare.com	-	Actif
kirk.ns.cloudflare.com	-	Actif

2. CloudFlare

CloudFare permet de récupérer le nom de domaine de chez OVH et de renvoyer vers l'ip publique de la box internet SFR où se trouve le cluster K8S. De plus le chiffrement SSL/TLS y est activé, ce qui permet d'établir une connexion chiffrée.

Gestion DNS pour brilliant-market.com					
+ Ajouter un enregistrement		🔍 Rechercher des enregistrements DNS			⚙️ Avancé
Type	Nom	Contenu	Durée TTL	État du proxy	
A	api	79.80.64.64	Automatique	☁️ Proxied	Modifier ▶
A	apiml	79.80.64.64	Automatique	☁️ Proxied	Modifier ▶
A	brilliant-market.com	79.80.64.64	Automatique	☁️ Proxied	Modifier ▶
A	monitor	79.80.64.64	Automatique	☁️ Proxied	Modifier ▶
A	simulation	79.80.64.64	Automatique	☁️ Proxied	Modifier ▶
A	www	79.80.64.64	Automatique	☁️ Proxied	Modifier ▶

✔️ **Votre mode de chiffrement SSL/TLS est Complet**

Ce paramètre a été changé pour la dernière fois le il y a 3 mois.

☐ Désactivé (non sécurisé) ⓘ
Pas de chiffrement appliqué

☐ Flexible
Chiffre le trafic entre le navigateur et Cloudflare.

☒ **Complet**
Procède à un chiffrement de bout en bout à l'aide d'un certificat auto-signé du serveur.

☐ Complet (strict)
Procède à un chiffrement de bout en bout, mais requiert un certificat d'une autorité de certification approuvée ou signé par une autorité de certification d'origine de Cloudflare sur le serveur.

C. Infrastructure

1. Azure

L'ensemble de la configuration Azure se passe sur Terraform et Ansible. Ces parties seront détaillées plus tard.



2. VMware

Nous avons utilisé l'image officiel de VMware ESXi pour l'installer sur le serveur Dell. Une fois l'installation terminée nous lui avons attribué une ipv4 statique ainsi que sa passerelle et les dns.

Pour la partie stockage nous avons utilisé un datastore composé de 3 disques de 1To sous forme de RAID5. Et un datastore avec un disque SSD Flash. Le raid 5 a été fait sur le matériel Dell et non sous VMware.

Nom	Type de lecteur	Capacité
datastore	Non-SSD	2,72 To
DS-SSD	Non-SSD	232,25 Go

D. Intégration continue

1. GitHub

L'ensemble de nos machines virtuelles ont git d'installé. Pour l'installer sur nos environnements nous utilisons Ansible qui installe le paquet git.

```
---
- name: "Installation des paquets"
  package:
    name: "{{ item }}"
    state: "present"
  with_items:
    - "htop"
    - "apt-transport-https"
    - "ca-certificates"
    - "curl"
    - "gnupg-agent"
    - "software-properties-common"
    - "git"
    - "gcc"
    - "shc"
```

Nous avons créé une organisation privée sur git avec 5 répertoire. Chaque répertoire a une fonction unique. Pour la partie infrastructure nous utilisons le répertoire « infrastructure ».

4PROJ-Groupe1 / infrastructure Private

<> Code Issues Pull requests Actions Projects Security Insights Settings

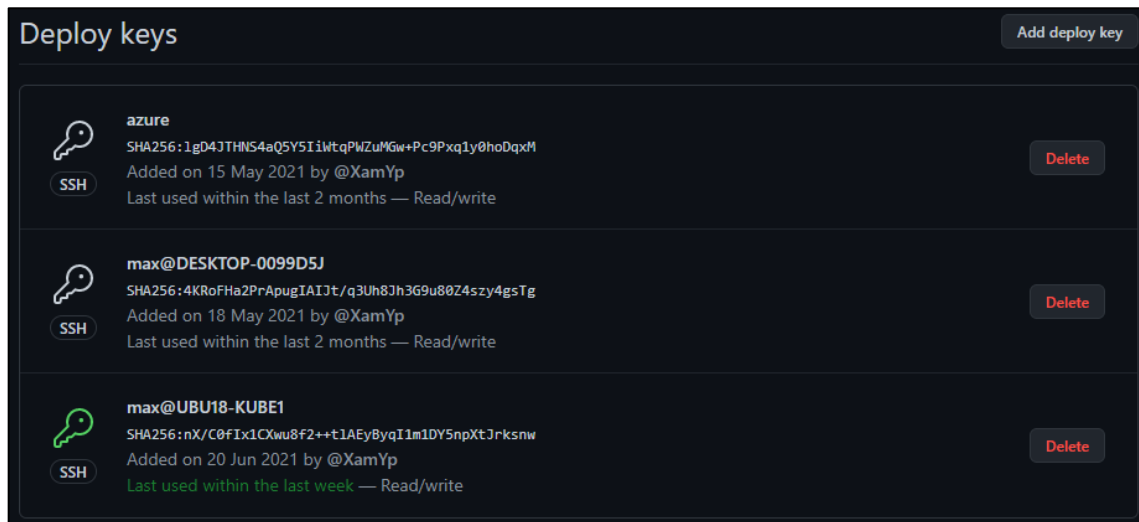
master 2 branches 0 tags Go to file Add file Code

XamYp Update 03-ingress.yml 86692ec 3 days ago 112 commits

ansible	Vérification sur un cluster AKS, all is fine	last month
kubernetes	Update 03-ingress.yml	3 days ago
sauvegarde	Mongo backup script done	last month
terraform	Change vcpu	3 months ago
LICENSE	Initial commit	5 months ago
README.md	Update README.md	4 months ago
architecture.drawio	Added architecture.drawio	5 months ago
cles-publiques.md	Fichier de clés publiques	5 months ago
schema_reseaux.md	Modificat du schema réseau	5 months ago



Il faut renseigner l'ensemble des clés ssh dans la partie « Settings » et « Deploy keys ».



2. Terraform

Terraform nécessite d'être installé sur un serveur uniquement, il n'y a pas comme Ansible d'agent à installer. Terraform a donc été installé sur la machine virtuelle UBU18-DVOPS. Il y'a deux fichiers de déploiement terraform (.tf) le premier pour Azure et le second pour VMware.

Pour Azure on y retrouve les ressources : *ressources groupes, virtual network, subnet, public ip, virtual network gateway, local network gateway, network gateway connection, network interface, linux virtual machine*. Ces ressources permettent de déployer des réseaux locaux dans notre ressources groupes avec une vm directement connectée, un vpn site to site avec une ip publique.

La partie VPN :

```
resource "azurerm_virtual_network_gateway" "AzureTerraformNetworkGateway" {
  name                = "VPNAzureToOnprem"
  location            = azurerm_resource_group.AzureTerraformGroup.location
  resource_group_name = azurerm_resource_group.AzureTerraformGroup.name

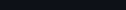
  type      = "Vpn"
  vpn_type  = "RouteBased"

  active_active = false
  enable_bgp    = false
  sku           = "VpnGw1"

  ip_configuration {
    name                = "vnetGatewayConfig"
    public_ip_address_id = azurerm_public_ip.AzureTerraformPubIP.id
    private_ip_address_allocation = "Dynamic"
    subnet_id           = azurerm_subnet.AzureTerraformSubnet.id
  }
}

resource "azurerm_local_network_gateway" "AzureTerraformLocalNetworkGateway" {
  name                = "LocalNetwork"
  resource_group_name = azurerm_resource_group.AzureTerraformGroup.name
  location            = azurerm_resource_group.AzureTerraformGroup.location
  gateway_address     = "79.80.64.64"
  address_space       = ["10.0.30.0/24"]
}
```



```
resource "azurerm_virtual_network_gateway_connection" "AzureTerraformNetworkGatewayConnection" {  
    name                = "AzureToForti"  
    location             = azurerm_resource_group.AzureTerraformGroup.location  
    resource_group_name = azurerm_resource_group.AzureTerraformGroup.name  
  
    type              = "IPsec"  
  
    virtual_network_gateway_id = azurerm_virtual_network_gateway.AzureTerraformNetworkGateway.id  
    local_network_gateway_id   = azurerm_local_network_gateway.AzureTerraformLocalNetworkGateway.id  
  
    shared_key = "
```









Pour la partie machine virtuelle :

```
resource "azurerm_linux_virtual_machine" "AzureTerraformLinuxVM" {
  name                        = "4PROJ-UBU18-KUBE4"
  resource_group_name       = azurerm_resource_group.AzureTerraformGroup.name
  location                  = azurerm_resource_group.AzureTerraformGroup.location
  size                      = "Standard_DS1_v2"
  admin_username            = "max"
  admin_password            = "1qaz!@WSX"
  disable_password_authentication = false
  network_interface_ids = [
    azurerm_network_interface.AzureTerraformNIC.id,
  ]

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }
}
```

Voici le ressource groupe « **4PROJ** »

<input type="checkbox"/> Name ↑↓	Type ↑↓
<input type="checkbox"/>  4PROJ-UBU18-KUBE4	Virtual machine
<input type="checkbox"/>  4PROJ-UBU18-KUBE4_disk1_28079fa1345a45f8bd7b315e1fea659f	Disk
<input type="checkbox"/>  4PROJ-VPN-IP	Public IP address
<input type="checkbox"/>  AzureToForti	Connection
<input type="checkbox"/>  LocalNetwork	Local network gateway
<input type="checkbox"/>  NIC-4PROJ-UBU18-KUBE4	Network interface
<input type="checkbox"/>  vNetAzure	Virtual network
<input type="checkbox"/>  VPNAzureToOnprem	Virtual network gateway



Pour VMware on utilise deux fichiers, le *main.tf* qui va décrire et déployer les éléments et un *variables.tf*. Le *variables.tf* contiendra le mot de passe de l'ESXi par mesure de sécurité et voici le *main.tf* :

```
provider "esxi" {
  esxi_hostname     = "10.0.30.10"
  esxi_hostport     = "22"
  esxi_hostssl      = "443"
  esxi_username     = "root"
  esxi_password     = var.esxi_password
}

resource "esxi_guest" "createVirtualMachine" {
  count = 3
  guest_name = "4PROJ-UBU18-KUBE${count.index + 1}"
  disk_store = "datastore"
  guestos    = "ubuntu-64"

  boot_disk_type = "thin"
  boot_disk_size = "50"

  memsize      = "4096"
  numvcpus     = "3"
  power        = "on"
  clone_from_vm = "TEMPLATE-UBUNTU18"

  network_interfaces {
    virtual_network = "VLAN-PRODUCTION"
    nic_type        = "vmxnet3"
  }
}
```

Pour initier le projet Terraform et installer les dépendances :

```
terraform init
```

Pour créer le plan d'exécution :

```
terraform plan
```

Application du plan :

```
terraform apply
```

Lien vers le dépôt : [infrastructure/terraform at master · 4PROJ-Groupe1/infrastructure \(github.com\)](https://github.com/4PROJ-Groupe1/infrastructure)

3. Ansible

Ansible nécessite d'être installé sur la vm qui va pousser le déploiement et d'être installé sur la vm qui reçoit le déploiement. Il est également important que la clé publique ssh de la vm de déploiement soit dans la vm qui va recevoir le déploiement. Pour se faire on utilise :

```
ssh-copy-id max@10.30.0.XXX
```



Ansible contient de dossier, le premier permet de déployer les utilisateurs sur les machines virtuelles et les configurations (paquets, désactivation de la swap, ect..). Le second dossier permet de créer l'environnement Kubernetes et de joindre automatiquement les nœuds au master.

Pour jouer le playbook du déploiement des utilisateurs :

```
ansible-playbook --user max --become -k -K -i inventories/production.ini
playbooks/users.yml
```

Pour jouer le playbook du déploiement des configurations :

```
ansible-playbook --user max --become -k -K -i inventories/production.ini
playbooks/installationDockerK8s.yml
```

On a ajouté un daemon.json avec des configurations spécifiques pour être poussés sur les environnements, cela permet notamment de stocker le mot de passe du docker hub de manière sécurisé dans Kubernetes.

```
- name: "Ajout de la clé de signature apt pour docker"
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present

- name: "Ajout du depot pour la version stable"
  apt_repository:
    repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial stable
    state: present

- name: "Installation de docker et ses dépendances"
  apt:
    name: "{{ packages }}"
    state: present
    update_cache: yes
  vars:
    packages:
      - docker-ce
      - docker-ce-cli
      - containerd.io
      - docker-compose

- name: "Ajout de les utilisateurs dans le groupe docker"
  user:
    name: "{{ item }}"
    group: docker
  with_items:
    - max
    - maxence
    - paul
    - gaetan

- name: "Ajout des utilisateurs dans le sudo"
  lineinfile:
    dest: /etc/sudoers
    line: '{{ item }} ALL=(ALL) NOPASSWD: ALL'
    validate: 'visudo -cf %s'
  with_items:
    - max
    - maxence
    - paul
    - gaetan

- name: "Déploiement du daemon.json"
  copy:
    src: files/daemon.json
    dest: /etc/docker/daemon.json
```



Il est import de peupler le fichier production.ini dans ansible/1-UtilisateursEtConfigurations/inventories/production.ini de sorte à indiquer les vms sur lesquelles déployer la tâche.

```
[production:children]
vmware
azure

[vmware]
4PROJ-UBU18-KUBE1 ansible_host=10.0.30.20
4PROJ-UBU18-KUBE2 ansible_host=10.0.30.21
4PROJ-UBU18-KUBE3 ansible_host=10.0.30.22

[azure]
4PROJ-UBU18-KUBE4 ansible_host=172.32.1.4
```

E. Serveurs et services

1. Template VMware

Pour déployer les machines virtuelles sur VMware nous avons créé une première machine virtuelle sur laquelle nous avons installé Ubuntu 18.04 car c'est une version stable de Ubuntu qui fonctionne avec Kubernetes et qui est aussi disponible sur Azure. Cela permet d'uniformiser l'ensemble de notre infrastructure. Nous avons créé un script qui sera exécuté sur chacune des machines une fois que la Template sera généré en machine virtuel via Terraform pour changer le nom de la machine ainsi que ses configurations réseaux (impossible de changer les configurations réseaux sur les machines virtuelles avec Ansible pour des raisons évidentes). Une fois la vm terminée nous l'avons converti en template.

Voici le script utilisé (change en fonction des vms) :

```
#!/bin/bash
NEW_HOSTNAME="UBU18-KUBE1"

if [ ! -n "$NEW_HOSTNAME" ] ; then
    echo 'Manque argument: new_hostname'
    exit 1
fi

if [ "$(id -u)" != "0" ] ; then
    echo "Désolé, vous devez être en root."
    exit 2
fi

CUR_HOSTNAME=$(cat /etc/hostname)

echo "Le nom actuel est $CUR_HOSTNAME"

hostnamectl set-hostname $NEW_HOSTNAME
hostname $NEW_HOSTNAME

sudo sed -i "s/$CUR_HOSTNAME/$NEW_HOSTNAME/g" /etc/hosts
sudo sed -i "s/$CUR_HOSTNAME/$NEW_HOSTNAME/g" /etc/hostname

echo "Le nouveau nom est $NEW_HOSTNAME"
cat /etc/netplan/00-installer-config.yaml
cp /home/max/script/kube1/00-installer-config.yaml /etc/netplan/00-installer-config.yaml
netplan apply
cat /etc/netplan/00-installer-config.yaml

echo "Le serveur va redémarrer"
sleep 3
shutdown -r now
```



2. Serveurs et rôles

Nom du serveur	Rôles	Description
TEMPLATE-UBUNTU18	Template pour déploiement	Permet de déployer des nouvelles machines.
UBU18-DEVOPS	Intégration continue	Git, Ansible et Terraform pour le déploiement continu.
UBU18-KUBE1	Maître Kubernetes	Gestion des nœuds et de l'état du cluster.
UBU18-KUBE2	Nœud Kubernetes	MongoDB, REACT, Nginx, Grafana, Prometheus.
UBU18-KUBE3	Nœud Kubernetes	MongoDB, REACT, Nginx, Grafana, Prometheus.
UBU18-KUBE4	Nœud Kubernetes	MongoDB, REACT, Nginx, Grafana, Prometheus.
UBU18-KIMESUFI	VPS de sauvegarde mongodb	Les dump mongodb sont envoyés sur ce serveur.

Les serveurs UBU18-KUBE1, KUBE2 ET KUBE3 sont déployés par Terraform et Ansible. Les utilisateurs des serveurs sont Gaétan, Maxence, Max, Paul (déployé via Ansible).

```
- name: Configuration des comptes utilisateurs
  hosts: all
  tasks:
    - name: Création des comptes
      user:
        name: "{{ item.username }}"
        groups: "{{ item.groups }}"
        shell: "{{ item.shell | default('/bin/bash') }}"
        password: "{{ item.password | password_hash('sha512') }}"
        comment: "{{ item.name | default(omit) }}"
        uid: "{{ item.uid | default(omit) }}"
        createhome: yes
        append: yes
      with_items: "{{ user4proj }}"
      become: yes
```

```
---
user4proj:
  - username: paul
    name: Paul RIVIERE
    password: 'XXXXXXXXXX'
    groups: adm,cdrom,dip,plugdev,lxd,sudo
  - username: maxence
    name: Maxence GRIAS
    password: 'XXXXXXXXXX'
    groups: adm,cdrom,dip,plugdev,lxd,sudo
  - username: gaetan
    name: Gaetan MORLET
    password: 'XXXXXXXXXX'
    groups: adm,cdrom,dip,plugdev,lxd,sudo
```

a. Kubernetes

Kubernetes est installé sur l'ensemble des environnements. Ils communiquent entre eux via le réseau de production ou via le vpn site to site pour la machine UBU18-KUBE4. Il s'agit d'un cluster on-premise, cela est important car nous avons dû implémenter un reverse proxy nginx avec un ingress pour pouvoir rendre nos services Kubernetes accessibles depuis l'extérieur. Nous avons également dû unset le http_proxy et le https_proxy pour gérer notre cluster kubernetes localement afin d'éviter une error de type « *Unable to connect to the server : net/http : TLS handshake timeout* ».

b. Kubernetes - MongoDB

La communauté MongoDB a créé un répertoire GitHub pour créer un replicaset sous Kubernetes. Nous l'avons utilisé et suivi les étapes indiquées par le readme. Toutefois nous avons dû faire un ajout sur le déploiement des ressources mongo. En effet, la communauté a pensé à des cluster K8S sous Azure ou AWS via les services ACS ou AKS. Là en l'occurrence il s'agit d'un cluster on-prem. La génération du stockage au déploiement des ressources ne s'y fait donc pas automatiquement nous avons dû ajouter des **PersistentVolume** et des **PersistentVolumeClaim**.



```

---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume-3
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/home/max/data2"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-volume-example-mongodb-0
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  volumeName: "pv-volume-1"

```

Une fois terminée l'ensemble du déploiement s'est bien déroulé. Voici les étapes que nous avons faites :

1. Clonage du repos git
git clone <https://github.com/mongodb/mongodb-kubernetes-operator.git>
2. Installation des ressources definitions
kubectl apply -f config/crd/bases/mongodbcommunity.mongodb.com_mongodbcommunity.yaml
3. Nous voulions que l'opérateur regarde le namespace mongo avec la commande
kubectl apply -k config/rbac --namespace mongo
4. Installation de l'opérateur
kubectl create -f config/manager/manager.yaml --namespace mongo
5. Déployer le replicaset
kubectl apply -f config/samples/mongodb.com_v1_mongodbcommunity_cr.yaml --namespace mongo

Voici le résultat

```

max@UBU18-KUBE1:~$ kubectl get all -n mongo
NAME                                     READY   STATUS    RESTARTS   AGE
pod/example-mongodb-0                  2/2     Running   0           94d
pod/example-mongodb-1                  2/2     Running   0           94d
pod/example-mongodb-2                  2/2     Running   0           94d
pod/mongodb-kubernetes-operator-7cddf7cbd4-smx6r  1/1     Running   1          100d

NAME                                     TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/example-mongodb-svc            ClusterIP      None          <none>         27017/TCP  100d

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongodb-kubernetes-operator  1/1     1             1           100d

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/mongodb-kubernetes-operator-7cddf7cbd4  1         1         1       100d

NAME                                     READY   AGE
statefulset.apps/example-mongodb        3/3     94d

```



c. Kubernetes - WebApps et APIs

Pour déployer notre solution web et les deux apis nous utilisons des containers qui ont été créés à partir d'un docker file par l'équipe de développement. Une fois la mise en place des images docker sur GitHub elles sont trigger par le docker hub qui se charge de construire l'image et de la rendre accessible.

Nous avons créé un namespace « **web** » où sont regroupés l'ensemble des services et déploiements liés à notre site internet. On crée un fichier yaml qui contiendra :

- La création de notre namespace **web**
- La clé secrète de notre docker hub sous forme de secret
- Les différents déploiements pour chaque service applicatif
- Les services associés aux différents déploiements

Voici un exemple de configuration dans [infrastructure/kubernetes/web/deployment.yml](#) :

```
apiVersion: v1
kind: Namespace
metadata:
  name: web
---
# kubectl create secret generic regcred --from-file=.dockerconfigjson=.dockerconfigjson
# kubectl get secret regcred --output=jsonpath={.data.\.dockerconfigjson}
apiVersion: v1
data:
  .dockerconfigjson: ewoJImF1dGhzIjogewoJCSJodHRwciovL2luZGV4LmRvY2
kind: Secret
metadata:
  name: regcred
  namespace: web
type: kubernetes.io/dockerconfigjson
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  namespace: web
spec:
  selector:
    matchLabels:
      app: web-app
      tier: frontend
  replicas: 2
  template:
    metadata:
      labels:
        app: web-app
        tier: frontend
    spec:
      containers:
        - name: web-app
          image: "xamyp/site-4proj:latest"
          ports:
            - name: http
              containerPort: 3000
      imagePullSecrets:
        - name: regcred
---
apiVersion: v1
kind: Service
metadata:
  name: web-app
  namespace: web
spec:
  selector:
    app: web-app
    tier: frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: http
```



```
max@UBU18-KUBE1:~$ kubectl get all -n web
```

NAME	READY	STATUS	RESTARTS	AGE
pod/webapi-84d66f799f-knsvg	1/1	Running	0	11d
pod/webapi-84d66f799f-lvx6q	1/1	Running	0	16h
pod/webapiml-7cf7f7f7f5-k2m5s	1/1	Running	0	5d16h
pod/webapiml-7cf7f7f7f5-mh56c	1/1	Running	0	5d16h
pod/webapp-55bd47b968-c72pg	1/1	Running	0	11d
pod/webapp-55bd47b968-nkg56	1/1	Running	0	11d
pod/websimulation-66945db544-92jws	1/1	Running	0	16h
pod/websimulation-66945db544-9vww9	1/1	Running	0	16h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/web-api	ClusterIP	10.100.112.161	<none>	8082/TCP	86d
service/web-api-ml	ClusterIP	10.110.210.107	<none>	5000/TCP	5d16h
service/web-app	ClusterIP	10.100.132.95	<none>	3000/TCP	86d
service/web-simulation	ClusterIP	10.108.240.113	<none>	3000/TCP	3d16h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/webapi	2/2	2	2	86d
deployment.apps/webapiml	2/2	2	2	5d16h
deployment.apps/webapp	2/2	2	2	86d
deployment.apps/websimulation	2/2	2	2	3d16h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/webapi-84d66f799f	2	2	2	86d
replicaset.apps/webapiml-7cf7f7f7f5	2	2	2	5d16h
replicaset.apps/webapp-55bd47b968	2	2	2	86d
replicaset.apps/websimulation-66945db544	2	2	2	3d16h

Voici le résultat final.

d. Kubernetes - Nginx

Étant donné que nous travaillions sur un cluster K8S on-prem le travail de génération d'un external-ip n'est pas fonctionnel. Si nous avions travaillé sur des services cloud K8S il aurait été plus simple de faire pointer notre site web vers le dns de nos services k8s **web-api**, **web-api-ml**, **web-app** et **websimulation**. Lorsqu'il s'agit d'un cluster k8s baremetal cela n'est pas généré automatiquement. Il faut alors penser à un reverse-proxy. Nous avons implémenter Nginx comme reverse-proxy.

Avant de débiter il faut créer le namespace « **ingress-nginx** ». Puis il faut installer l'ensemble des ressources nginx nécessaires pour le bon fonctionnement avec K8S. Nous nous sommes inspiré du git officiel kubernetes/ingress-nginx.

Voici le premier fichier yaml : [infrastructure/01-deploy.yaml at master · 4PROJ-Groupe1/infrastructure \(github.com\)](#)

Début du fichier de configuration 01-deploy.yaml :

```
apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
---
# Source: ingress-nginx/templates/controller-serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.23.0
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.44.0
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
```




Dans un troisième temps nous devons créer les services sur le namespace *ingress-nginx* qui permettait d'accéder aux services *web-api*, *web-api-ml*, *web-app* et *websimulation* sur le namespace *web*. Cela est important pour la dernière partie sur les ingress controller.

Voici le début de la configuration ([infrastructure/02-externalname.yml at master · 4PROJ-Groupe1/infrastructure \(github.com\)](#)) :

```
kind: Service
apiVersion: v1
metadata:
  name: web-app
  namespace: ingress-nginx
spec:
  type: ExternalName
  externalName: web-app.web.svc.cluster.local
---
kind: Service
apiVersion: v1
metadata:
  name: web-api
  namespace: ingress-nginx
spec:
  type: ExternalName
  externalName: web-api.web.svc.cluster.local
```

Enfin nous devons créer les différents ingress controller pour traduire les URL provenant d'un client en requête interne dans le cluster K8S (via OVH → CloudFlare → SFR → Fortigate → Master K8S) pour atteindre le bon service. Le lien entre l'ingress et le service se fait via le **serviceName**.

Le lien pour la configuration des ingress-controller : [infrastructure/03-ingress.yml at master · 4PROJ-Groupe1/infrastructure \(github.com\)](#)

Exemple avec l'url <https://brilliant-market.com> :

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-webapp
  namespace: ingress-nginx
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  tls:
  - hosts:
    - brilliant-market.com
  rules:
  - host: brilliant-market.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web-app
          servicePort: 3000
```



e. Kubernetes - Grafana

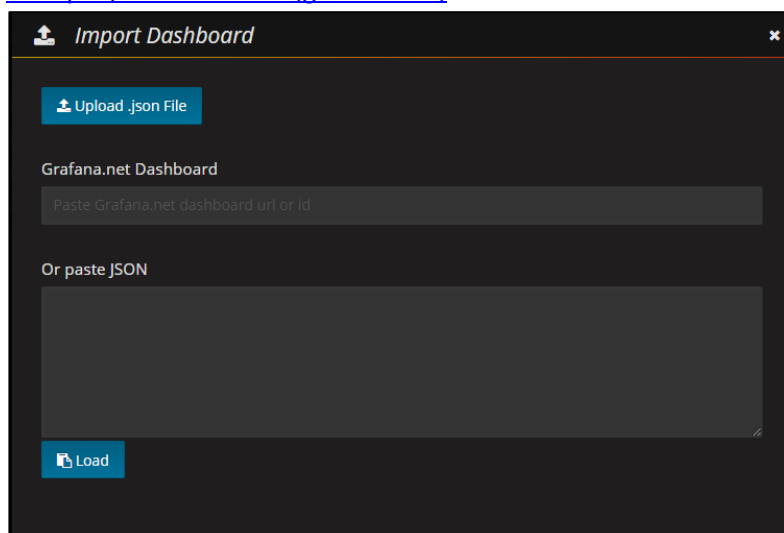
Grafana permet la création des dashboard afin d'observer l'état de notre cluster K8S et des différents états mémoires, réseaux, processeurs de notre infrastructure. Nous avons utilisé Prometheus pour récupérer les métriques sur nos différents équipements/applications.

Pour créer le namespace Grafana et toutes les différentes ressources nous avons utilisé le fichier yaml suivant : [infrastructure/manifests-all.yaml at master · 4PROJ-Groupe1/infrastructure \(github.com\)](#)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: monitoring
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus-k8s
  namespace: monitoring
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
```

Une fois le namespace et les ressources déployés, nous nous sommes connecté via le lien : [Grafana \(brilliant-market.com\)](#)

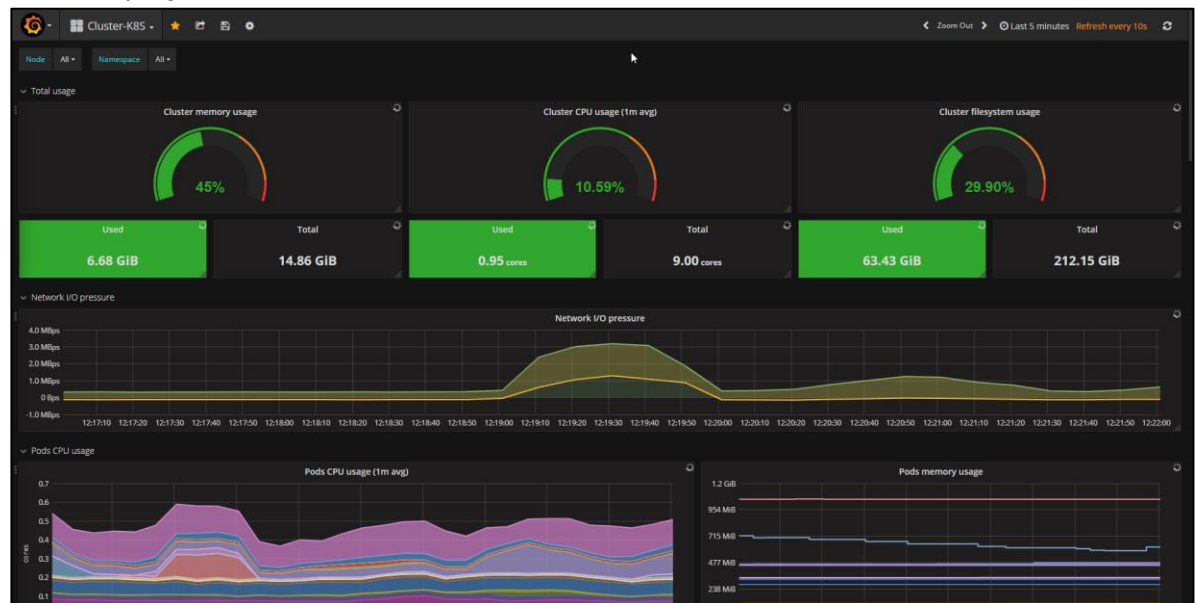
Puis nous avons importé le dashboard souhaité (disponible à l'adresse [infrastructure/Kubernetes-Dashboard-1617540593782.json at master · 4PROJ-Groupe1/infrastructure \(github.com\)](#)):





Et nous pouvons afficher les métriques sous forme de dashboards visuels avec la possibilité de filtrer les éléments à afficher par machine virtuelle ou par namespace.

Voici un aperçu du résultat final :



F. Sécurité et sauvegarde

1. Journaux d'évènements

Nous avons la possibilité de consulter les journaux d'évènements réseaux à travers notre pare-feu Fortigate ou à travers le forti cloud.

Traffic Logs

Last 7 Days

Export

Add Filter

Customize Columns

Log Files

#	Time	Level	Firewall Action	User	Source	Destination	Service	Sent/Received	Application
1	13:10:35	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
2	13:10:05	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
3	13:09:35	notice	close			1	HTTPS-44443	2.11 KB/3.62 KB	N/A
4	13:09:05	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
5	13:08:35	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
6	13:08:05	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
7	13:08:05	notice	close			1	HTTPS-44443	2.02 KB/2.74 KB	N/A
8	13:08:05	notice	close			1	HTTPS-44443	1.86 KB/3.58 KB	N/A
9	13:07:35	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
10	13:07:05	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
11	13:06:55	notice	close			1	HTTPS	1.15 KB/2.67 KB	N/A
12	13:06:35	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
13	13:06:05	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
14	13:05:35	notice	close			1	HTTPS-44443	2.1 KB/3.62 KB	N/A
15	13:05:05	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
16	13:04:35	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
17	13:04:05	notice	close			1	HTTPS-44443	2.11 KB/3.62 KB	N/A
18	13:04:03	notice	client-rst			1	HTTP	80 B/44 B	N/A
19	13:03:35	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
20	13:03:05	notice	close			1	HTTPS-44443	2.11 KB/3.62 KB	N/A
21	13:03:05	notice	close			1	HTTPS-44443	1.91 KB/2.63 KB	N/A
22	13:03:05	notice	close			1	HTTPS-44443	1.86 KB/3.58 KB	N/A
23	13:02:35	notice	close			1	HTTPS-44443	2.1 KB/3.67 KB	N/A
24	13:02:25	notice	client-rst			1	HTTPS	80 B/44 B	N/A
25	13:02:05	notice	close			1	HTTPS-44443	2.11 KB/3.62 KB	N/A

Ces journaux d'évènements permettent de connaître l'adresse IP source et l'adresse IP de destination avec un détail des filtres. Pour cet exemple nous avons mis en place des filtres explicites.



2. Sauvegardes

Les documents mongoDB sont essentiels à la production. Ils contiennent l'ensemble des utilisateurs, des produits et d'autres éléments qui sont vitales pour le site. Nous effectuons un dump des bases mongoDB chiffrées avec gpg. Une fois le dump effectué nous avons mis en place une tâche cron qui envoie le dump chiffré et compressé sur un serveur distant.

Voici le script (disponible à cette adresse [infrastructure/mongo-backup.sh at master · 4PROJ-Groupe1/infrastructure \(github.com\)](https://github.com/4PROJ-Groupe1/infrastructure/blob/master/mongo-backup.sh)):

```
#!/bin/bash
date_backup=$(date +"%Y_%m_%d_%I_%M_%p")
mongodump --uri "mongodb://my-user:Supinfo@example-mongodb-
svc.mongo.svc.cluster.local:27017/?replicaSet=example-mongodb" --
out=/home/max/backup/mongo/$date_backup
cd /home/max/backup/mongo/
tar -cvzf - $date_backup | gpg --batch -c --passphrase xxxxxxxxxxxx >
mongo_backup_$date_backup.tar.gz.gpg
rm -rf /home/max/backup/mongo/$date_backup
scp -i ~/.ssh/id_rsa -P 2152
/home/max/backup/mongo/mongo_backup_$date_backup.tar.gz.gpg
root@5.196.70.106:/root/mongo/.
```

Voici le résultat (il est tout à fait possible de changer la tâche cron pour que les dump puissent s'effectuaient à des intervalles plus court)

```
root@DEB9-KIMSUI:~/mongo# ls -l
total 164
-rw-r--r-- 1 root root 2172 mai 17 15:37 mongo_backup_2021_05_17_01_37_PM.tar.gz.gpg
-rw-r--r-- 1 root root 2176 mai 18 05:00 mongo_backup_2021_05_18_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2174 mai 19 05:00 mongo_backup_2021_05_19_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2172 mai 20 05:00 mongo_backup_2021_05_20_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2171 mai 21 05:00 mongo_backup_2021_05_21_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 mai 22 05:00 mongo_backup_2021_05_22_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2171 mai 23 05:00 mongo_backup_2021_05_23_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 mai 24 05:00 mongo_backup_2021_05_24_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 mai 25 05:00 mongo_backup_2021_05_25_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2172 mai 26 05:00 mongo_backup_2021_05_26_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2172 mai 27 05:00 mongo_backup_2021_05_27_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2171 mai 28 05:00 mongo_backup_2021_05_28_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 mai 29 05:00 mongo_backup_2021_05_29_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 mai 30 05:00 mongo_backup_2021_05_30_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 mai 31 05:00 mongo_backup_2021_05_31_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2170 juin 1 05:00 mongo_backup_2021_06_01_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2172 juin 2 05:00 mongo_backup_2021_06_02_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2170 juin 3 05:00 mongo_backup_2021_06_03_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2174 juin 4 05:00 mongo_backup_2021_06_04_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2176 juin 5 05:00 mongo_backup_2021_06_05_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2169 juin 6 05:00 mongo_backup_2021_06_06_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2174 juin 7 05:00 mongo_backup_2021_06_07_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2174 juin 8 05:00 mongo_backup_2021_06_08_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2171 juin 9 05:00 mongo_backup_2021_06_09_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2172 juin 10 05:00 mongo_backup_2021_06_10_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2171 juin 11 05:00 mongo_backup_2021_06_11_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2178 juin 12 05:00 mongo_backup_2021_06_12_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2172 juin 13 05:00 mongo_backup_2021_06_13_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2173 juin 14 05:00 mongo_backup_2021_06_14_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2634 juin 15 05:00 mongo_backup_2021_06_15_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2623 juin 16 05:00 mongo_backup_2021_06_16_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2624 juin 17 05:00 mongo_backup_2021_06_17_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2624 juin 18 05:00 mongo_backup_2021_06_18_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2628 juin 19 05:00 mongo_backup_2021_06_19_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2634 juin 20 05:00 mongo_backup_2021_06_20_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2624 juin 21 05:00 mongo_backup_2021_06_21_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2633 juin 22 05:00 mongo_backup_2021_06_22_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2627 juin 23 05:00 mongo_backup_2021_06_23_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2624 juin 24 05:00 mongo_backup_2021_06_24_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2631 juin 25 05:00 mongo_backup_2021_06_25_03_00_AM.tar.gz.gpg
-rw-r--r-- 1 root root 2623 juin 26 05:00 mongo_backup_2021_06_26_03_00_AM.tar.gz.gpg
```

La sauvegarde des ressources Kubernetes ne sont pas à prévoir étant donné qu'avec l'intégration continue et le code sur GitHub il est possible de redéployer une infrastructure rapidement.