

# phone-books

Lusianne and Martial

April 2020

## 1 Introduction

Afin de répondre à la problématique posée par pozos qui consiste à mettre en place une chaîne d'intégration et de déploiement continu d'une de leurs applications tout en réalisant le scan des images docker et le test d'une attaque de type xss de l'application avant sa mise en ligne.

En occurrence nous allons utiliser l'application : **crud-application-using-flask-and-mysql** que nous avons dénommée : **crud\_appli**. Ce tutoriel explique toutes les démarches à suivre en passant de l'infra, à la configuration et la démonstration.

## 2 Infrastructure

Un schéma décrivant notre infrastructure.

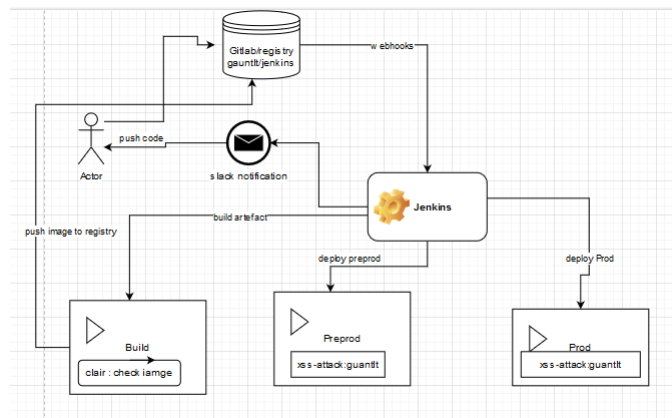


Figure 1: Schema infrastructure

Les outils :

- **Environnement de travail** : AWS en utilisant une stack pour déployer nos instances : **cloudFormation** notre provider.
- Instances :
  1. **server gitlabjenkins**: qui hébergera gitlab et jenkins chacun écoutant sur des ports différents que nous verrons dans la partie configuration.
  2. **serverbuild**: qui hébergera nos artefacts.
  3. **serverPreprod** va servir de preprod avant déploiement en prod.
  4. **serverprod** déploiement en production une fois preprod validé.
- **Outils**: Docker, Docker-compose, Ansible, git, debian, centos ,docker hub, MobaXterm, clair,guantlt ,jenkins, gitlab.

Après la description de notre Infrastructure passons a l'étape de configuration de chaque instance.

### 3 Configuration Instance

- Debian: sevrer Gitlab-Jenkins
- Centos: sever Build,Preprod, Prod

NB : Une fois notre satch déployée veuillez patienter un peu.

### 4 server gitlabjenkins

Une fois que notre server *gitlabjenkins* est ready , nous allons établir une connexion ssh via *mobaXterm* comme suit.

1. mettre url du server
2. cocher cette case et mettre comme username: **admin**
3. cliquer la dessus et aller sur 4 pour définir la clé privé.(on souhaite établir une connexion sécurisée entre notre machine local et notre server distant *gitlabjenkins*).
4. cocher cette case et insérer donc la clé privé qui nous a été générée lors de la création de la stack sur aws. Par défaut elle est downloadée sur votre machine local dans le dossier /download et porte l'extension **.pem** ou **ppk**.
5. cliquer la dessus et dans *Session name* mettre le nom qui vous permettra de distinguer vos instances, puis cliquer sur **OK**.

The image shows the mabaxterm configuration interface. The 'Basic SSH settings' tab is active, showing 'Remote host \*' with a red '1' next to it, 'Specify username' with a red '2' next to it, and 'Port' set to 22. Below this, the 'Advanced SSH settings' tab is highlighted with a red '3'. In the 'Advanced SSH settings' section, 'X11-Forwarding' and 'Compression' are checked, 'Remote environment' is set to 'Interactive shell', 'Execute command' is empty, 'SSH-browser type' is 'SFTP protocol', 'Use private key' is checked with a red '4' next to it, and 'Execute macro at session start' is set to '<none>'. The 'Bookmark settings' tab is also visible with a red '5' next to it. A yellow key icon is present on the right side of the 'Advanced SSH settings' section.

Figure 2: config mabaxterm

Nous venons ainsi d'établir une connexion ssh entre machine local et notre server distant *gitlabjenkins*. Nous allons reproduire la même chose pour nos instances:

- serverbuild
- serverPreprod
- serverprod

NB: **username:** centos

## 4.1 Gitlab

### 4.1.1 création user

Depuis votre navigateur favori tapez l'url de votre server *gitlabjenkins*, ce dernier s'ouvre et vous êtes amenés à définir un mot de passe. Ensuite votre mot de passe créer , vous logger en tant que **root**, puis vous saisissez votre mot de passe. Une fois logger en tant que root nous allons définir un user, pour cela cliquez sur le bouton encadrer en rouge (voir image ci-dessous), puis **new user** et renseigner les champs obligatoires puis valider, ensuite cliquez sur **edit** (voir image ) pour définir le mot de passe de notre user que l'on vient de créer, puis validez . Se déconnecter compte root et se loguer notre compte user que nous venons de créer.

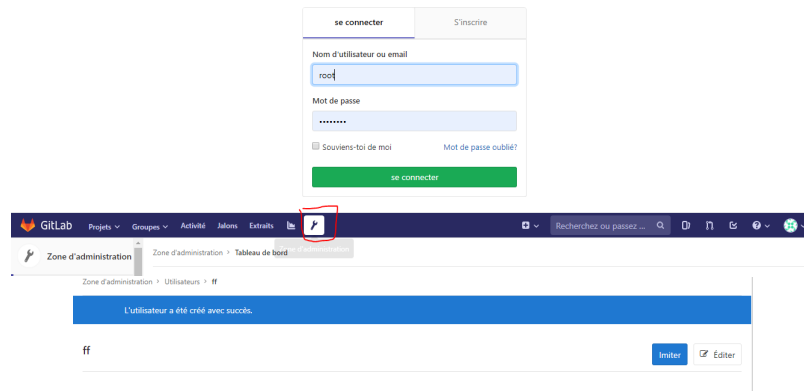


Figure 3: Création user

#### 4.1.2 Création Repo

Une fois logué avec le nouvel utilisateur nous allons créer nos différents Repo pour pouvoir réaliser cette chaîne CI/CD complète de déploiement de notre application.

- Repo crud\_appli: qui contient le code source de l'appli
- Repo jenkins-shared-library : Repo au quel nous ferions appel pour vérifier la syntaxe de la partie Dev
- Repo CI\_crud : qui va nous servir à checker le code Dev.
- Repo CD\_crud: qui contient le code des Ops.

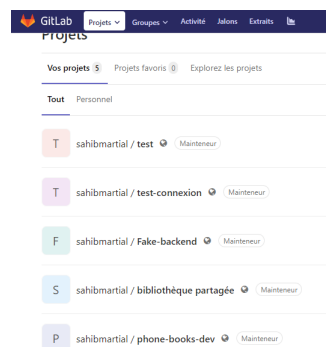


Figure 4: Création Repo

### 4.1.3 Push repo

Nous allons copier notre clé publique comme suit dans le compte de notre user gitlab afin de réaliser nos push simplement sans taper de login.

NB: Mettre notre cle à l'extension `.pem` sous le nom **id\_rsa** et notre clef pub sous l'appellation **id\_rsa.pub**.

NB: cette étape est délicate car si c'est mal fait on ne pourra pas réaliser des pushes en ssh.

- Faire juste un cp `authorized_keys` vers `id_rsa.pub`
- Faire juste un cp `xxx.pem` vers `id_rsa`

A la fin doit avoir ça dans notre dossier `.ssh`

```
centos@ip-172-31-84-201:~$ ls  
authorized_keys id_rsa id_rsa.pub private
```

Figure 5: Copy des clefs pub et private



Figure 6: Insertion publique key

Se connecter via mobaXterm sur notre server gitlabjenkins, puis rapatrier nos codes sur nos différents repo gitlab respectifs:

- crud\_appli :
- ci\_crud :
- cd\_crud :
- jenkins-shared-library:

### 4.1.4 Token for jenkins

Une fois les différents push terminés nous allons générer un token nécessaire à jenkins pour avoir communiquer avec gitlab comme ceci. Allez sur l'utilisateur , puis sur réglages ensuite choisir accès token et renseigner les champs et cocher API le Champ 2 détermine sur la figure détermine la durée de vie de votre token

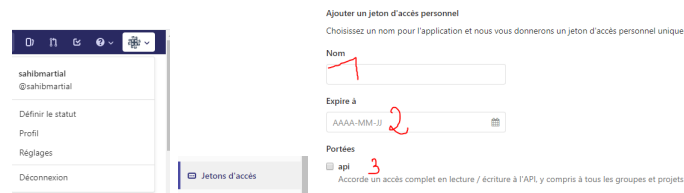


Figure 7: Création Token

NB: Après avoir renseigné les champs, un Token vous ai généré à conserver soigneusement car très utile pour **Jenkins**. Après la création du Token for jenkins , on passe ensuite au paramétrage de Jenkins.

## 4.2 Jenkins

NB: Pour lancer jenkins il va falloir réaliser cette petite configuration vue que nous faisons tourner gitlab et jenkins sur la même instance. Sur notre instance server gitlabjenkins depuis mobaxterm faire:

- cd / pour aller a la racine puis
- nano cursus-devops/jenkins/docker-compose.yml
- modifier port:
  1. 8080:8080
  2. 8443:8443
- exécuter la commande :**docker-compose stop** , puis celle-ci **docker-compose up -d**.

Après avoir réalise ce paramétrage, dans votre navigateur taper l'url:8080 comme ceci <http://ec2-54-227-204-73.compute-1.amazonaws.com:8080> les logins pour se connecter sont:


- username :user
- mot de passe : bitnami

### 4.2.1 Install plugin

Une fois connecter allez dans **administrer jenkins** puis **Gestion plugin** puis dans **disponible**

et rechercher puis cocher les plugins suivant :

1. gitlab : pour synchroniser jenkins et gitlab
2. Slack Notification : pour notifier les users du statut des jobs



**Welcome to Jenkins!**

Username

Mot de passe

Sign in

☐ Keep me signed in

Figure 8: Création user

Administrer Jenkins

**Gestion des plugins**  
Ajouter, supprimer, activer ou désactiver des plugins qui peuvent étendre les fonctionnalités de Jenkins.  
▲ mises à jour disponibles

Mises à jour Disponibles Installés Avancé

Figure 9: Download plugin

### 3. Embeddable Build Status : pour le badge

Mettre un système de log pour checker la connexion entre gitlab et jenkins, dans *administer jenkins*, puis *log system* et définir un nouveau.

Nom gitlablogs

Enregistreurs

Enregistreur com.dabsquared.gitlabjenkins Niveau de log LE PLUS FIN Supprimer

Ajouter

Liste des loggers et des niveaux de log à enregistrer

Enregistrer

Figure 10: System log jenkins-gitlab

#### 4.2.2 Credentials











Nous allons définir nos credentials(4).

- vaultkey: choisir comme type *secret text* puis entrer le text et *id*
- devopskey : clé privé ssh pour établir les connexions ssh choisir comme type *secret file*, puis *id* de le cle.

- gitlabapi: choisir gitlab Api puis renseigner le token récupéré sur gitlab et donner son id .
  - Credentials gitlab :
  - slack :le type choisi *secret text* récupérer le token généré par slack pour jenkins et le copier puis définir l'id .
1. login : de notre user gitlab
  2. passwd: mot de passe de notre user

### Identifiants globaux (illimité)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Nom	Type	Description
 <a href="#">vault-key</a>	secret	Secret text	secret 
 <a href="#">devopskey</a>	devops.pem (sshkey)	Secret file	sshkey 
 <a href="#">logingitlab</a>	sahibmartial/***** (gitlablogin)	Nom d'utilisateur et mot de passe	gitlablogin 
 <a href="#">gitlabsahibmartial</a>	GitLab API token (gitlabsahibmartial)	GitLab API token	gitlabsahibmartial 
 <a href="#">slacknotification</a>	notificationslack	Secret text	notificationslack 

Icône: [S](#) [M](#) [L](#)

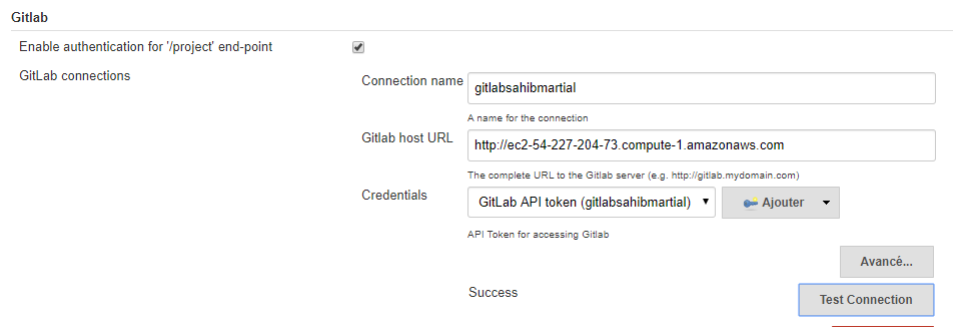
Figure 11: Credentials



### 4.2.3 Config system:Gitlab

On retourne sur Jenkins pour configurer le système en allant sur administrer , puis config system .

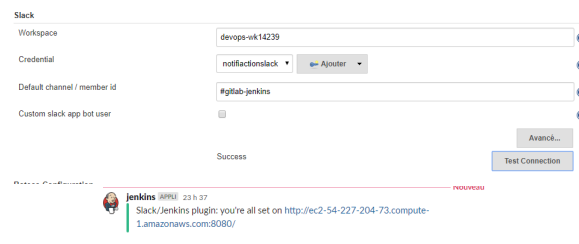
En premier nous allons configurer gitlab comme ceci



The screenshot shows the 'GitLab' configuration page in Jenkins. It includes a checkbox for 'Enable authentication for "/>

Figure 12: Config gitlab

### 4.2.4 Config system:notification slack



The screenshot shows the 'Slack' configuration page in Jenkins. It includes fields for 'Workspace' (devops-wk14239), 'Credential' (notificationslack), 'Default channel / member id' (#gitlab-jenkins), and 'Custom slack app bot user'. There are 'Avancé...' and 'Test Connection' buttons. Below the form, a 'Recent Configuration' section shows a message from Jenkins to the Slack channel.

Figure 13: Config slack notification

### 4.2.5 Config system:shared library

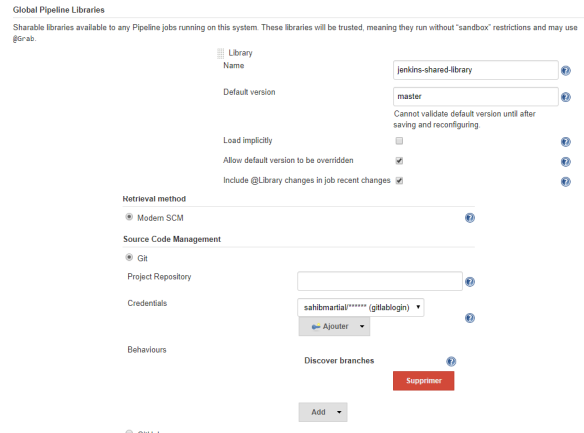


Figure 14: Config slack notification

## 4.3 Configuration des jobs

### 4.3.1 Pipeline CD\_crud

clicquez sur **Nouveau Item** pour créer le pipeline *CD-crud* ainsi la fenêtre de configuration du pipeline s'ouvre et nous allons paramétrer certains champs.



Figure 15: Synchrono gitlab-jenkins

### 4.3.2 Créer des déclencheurs( Build Triggers)

- cocher ce champ : Générez quand un changement est poussé vers Git-Lab.puis cliquez sur advanced pour générer le token,

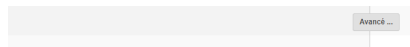


Figure 16: Génération token Webhook

puis allez sur gitlab sur le repo concerné, dans **réglages** ensuite sur **intégrations** et renseignez les champs suivants:

- 1 : url fournit par jenkins comme sur l'image ci-dessous
- 2 : le token récupéré

On valide et fait un test on doit avoir le résultat suivant :

**Intégrations**

Les webhooks peuvent être utilisés pour lier des événements lorsque quelque chose se passe dans le projet.

URL

Jeton secret

Utilisez ce jeton pour valider les charges utiles reçues. Il sera envoyé avec la demande dans l'en-tête HTTP X-Gitlab-Token.

Figure 17: Activation webhook gitlab

Webhooks (1)

Événements push

Vérification SSL: activé

Hook exécuté avec succès: HTTP 200

Figure 18: Activation webhook gitlab

### 4.3.3 Pipeline

Renseignez les champs comme sur la figure en précisant l'url du repo concerné et mettre les credentials et on enregistre On reprend les procédures pour le

Repository URI:

Credentials:

**Pipeline**

Definition

Pipeline script from SCM

SCM:

Figure 19: Config Pipeline

prochain job : **ci-dev**.

#### 4.3.4 synchronisation des jobs

Ici nous allons synchroniser le job ci-dev de sorte du au success il déclenche le lancement du job **CD-crud**. Donc on se rend sur administrer du job **Cd-crud** et faire: Une fois toutes les configurations terminées on procède a la

### Build Triggers

☒ Construire après le build sur d'autres projets

Projet à surveiller

☒ Déclencher que si la construction est stable

Figure 20: Synchronisation Jobs

démonstration.

## 5 Scan

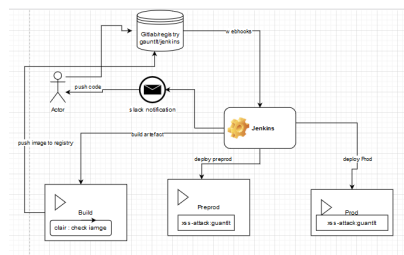


Figure 21: archi scan and security

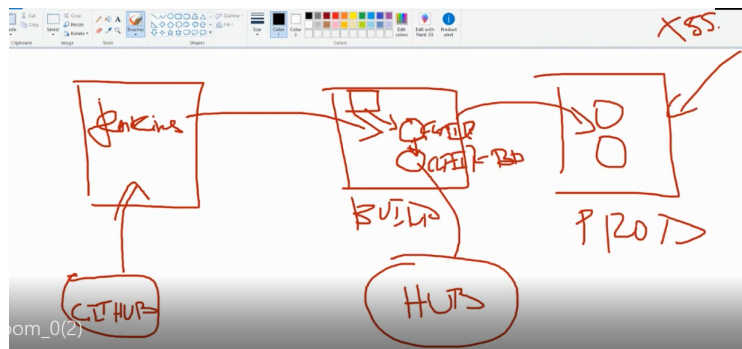


Figure 22: Description du scan and securite

Ici nous allons checker la version des images docker et leurs vulnérabilités via l'outil clair. Les 2 images à scanner sont :

- Frontend : python:3.5-alpine

- Backend: mysql:5.5

## 5.1 Frontend

Le résultat et analyse du scan cette image que l'on retrouve sur la machine build dans le dossier: /tmp/cd.frontend. Comme peut le voir pas de vulnérabilités

```
[centos@ip-172-31-81-150 tmp]$ cat cd_frontend-scan-report.json
{
  "image": "ec2-54-198-58-209.compute-1.amazonaws.com/sahibmartial/cd-crud/image_app:image_frontend",
  "unaproved": [],
  "vulnerabilities": []
}[centos@ip-172-31-81-150 tmp]$
```

Figure 23: resultat scan image frontend

détectées dans notre image .

## 5.2 Backend

Le résultat et analyse du scan de cette image que l'on retrouve sur la machine build dans le dossier /tmp/cd.backend. Nous avons les CVE(Common Vulnerabilities and Exposures ou CVE est un dictionnaire des informations publiques relatives aux vulnérabilités de sécurité.), des CVE vulnérables ont été détectés:

```
[centos@ip-172-31-81-150 tmp]$ cat cd_backend-scan-report.json
{
  "image": "ec2-54-198-58-209.compute-1.amazonaws.com/sahibmartial/cd-crud/image_mysql:image_mysql",
  "unaproved": [
    "CVE-2020-12243",
    "CVE-2015-3276",
    "CVE-2019-13565",
    "CVE-2017-17740",
    "CVE-2017-16350",
    "CVE-2019-13057",
    "CVE-2018-12886",
    "CVE-2019-51887",
    "CVE-2018-20804",
    "CVE-2005-2541",
    "CVE-2018-20482",
    "CVE-2019-9023",
    "CVE-2017-11164",
    "CVE-2017-7245",
    "CVE-2017-16231",
    "CVE-2017-7246",
    "CVE-2016-2779",
    "CVE-2018-1000054",
    "CVE-2011-3374",
    "CVE-2018-1000058",
    "CVE-2018-9234",
    "CVE-2019-14853",
    "CVE-2019-19909",
    "CVE-2019-12909",
    "CVE-2019-13027",
    "CVE-2019-12904",
    "CVE-2018-6829",
    "CVE-2011-4816",
    "CVE-2011-3389"
  ],
  "vulnerabilities": [
    {
      "vulnerability": "CVE-2020-12243",
      "namespace": "debian:9",
      "severity": "Unknown"
    },
    {
      "vulnerability": "CVE-2015-3276",
      "namespace": "debian:9",
      "severity": "Negligible"
    },
    {
      "vulnerability": "CVE-2019-13565",
      "namespace": "debian:9",
      "severity": "Low"
    },
    {
      "vulnerability": "CVE-2017-17740",
      "namespace": "debian:9",
      "severity": "Negligible"
    },
    {
      "vulnerability": "CVE-2017-14159",
      "namespace": "debian:9",
      "severity": "Negligible"
    },
    {
      "vulnerability": "CVE-2019-13057",
      "namespace": "debian:9",
      "severity": "Low"
    },
    {
      "vulnerability": "CVE-2018-12886",
      "namespace": "debian:9",
      "severity": "Unknown"
    }
  ]
}
```

Figure 24: résultat scan CVE vulnérables

## 6 Security

Dans cette partie nous allons simuler une attaque de type xss attack, c'est a dire voir si notre application ne contient pas de faille pouvant attaquer les systèmes des utilisateurs en ligne via l'outil gauntlt. L'attaque xss est s'est déroulée sans trouver de faille dans notre application.

```

+ gauntlt nos.atack
[DEPRECATION] Ce joyau a été renommé optimist et ne sera plus pris en charge. Veuillez passer à optimist dès que possible.
#lent
Fonctionnalité: Recherchez les scripts intersites (xss) en utilisant arachni contre scanme.nmap.org

Scénario: en utilisant arachni, recherchez les scripts intersites et vérifiez qu'aucun problème n'est détecté # xss.atack: 4
Étant donné que "arachni" est installé # gauntlt-1.0.13 / lib / gauntlt / attack_adapters / arachni.rb: 1
Et le profil suit: # gauntlt-1.0.13 / lib / gauntlt / attack_adapters / gauntlt.rb: 9
  | nom | valeur |
  | url | https://s2-54-86-83-164.compute-1.amazonaws.com |
  | lorsque je lance une attaque "arachni" avec: # gauntlt-1.0.13 / lib / gauntlt / attack_adapters / arachni.rb: 5
  ==
  arachni --checks = xss --scope-directory-depth-limit = 1 {url}
  ==
  Ensuite, la sortie doit contenir «0 problème a été détecté». # aruba-0.7.4 / lib / aruba / cucumber.rb: 166

1 scénario (1 réussi)
4 étapes (4 réussies)
0m6.537s

```

Figure 25: résultat attack xss

## 7 Démonstration

les playbooks à surveiller :

- install\_crud.yml : nous buildons nos images.
- clair-scan.yml : nous scanons, nous pushons sur notre registry et nous supprimons .
- check\_deploy\_app.yml: nous checkons notre application est bien deployée , ensuite nous simulons une attack xss

## 8 Conclusion

Ce travail à été laborieux surtout au niveau du debugage des erreurs mais nous avons pu comptés sur l'aide des collègues pour donner nous donner un coup de main quand il le fallait. Faites attentions a la version Ansible(pour nous ansible 2.5) que vous utilisée car y'a certaines déclarations qui ne marcheront pas en fonction des versions dans vos playbooks. En une phrase Observer, Analyser et Appliquer "*sahibmartial*".

Amat Victoria Curam

## 9 Annexe

### 9.1 Configure Registry

Si vous souhaitez installer le registry de A à Z en partant d'une machine vierge nous vous conseillons ce tuto: <https://eazytraining.fr/informatique/devops/activation-du-container-registry-docker-sur-gitlab-ce/>.

Pour notre part nous avons utilisé une stack où le registry était déjà configuré , nous avons juste réalisé ses étapes suivantes :

#### 9.1.1 Activation du registry sur un repo

Se rendre sur le repo voulu pour notre part c'est **cd-crud** , ensuite dans réglables, puis sur générale puis visibilité ...(image 1), un menu se déroule,

activez la partie registry(image2) enfin validez. vous verrez sur votre gauche sur le repo enregistrement ou registry (image 3). Vous cliquez dessus(image 3) il vous explique comment stocker vos images la syntaxe a utiliser . NB: Lorsque

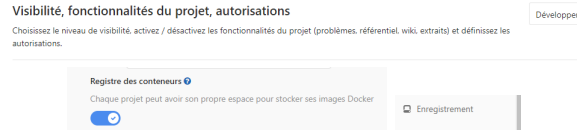


Figure 26: Activation du registry sur un repo

nous créons notre registry , nous le créons avec un certificat pour notre cas tout est réalisé dans la stack et visible sur ce chemin: Nous avons crée le certificat



Figure 27: Certificat registry

avec le nom de notre machine et nous avons le fichier *ca.crt*.  
Sur nos instances build et Prod il va falloir copier ce certificat

- Build: ce certificat lui permettra de faire le push des images sur le registry
- Prod :ce certificat permettra de récupérer les images sur le registry

Il nous faut exécuter sur chaque instance cette commande :

Création des dossiers: `mkdir -p /etc/docker/certs.d/ec2-54-198-58-209.compute-1.amazonaws.com/`

Copy certificat: copier le fichier **ca.crt** sur chaque instance dans le dossier d'accueil: `etc/docker/certs.d/ec2-54-198-58-209.compute-1.amazonaws.com/`