

Un repo de montée d'une infra autour de gitlab, dans un contexte docker-compose

Edit

Manage topics

📄 284 commits

🌿 1 branch

📦 3 releases

👤 1 contributor

📄 GPL-3.0

Branch: master ▾


New pull request

Create new file

Upload files

Find file

























Clone or download ▾



Jean-Baptiste-Lasselle

Add files via upload

Latest commit 09435d4 just now

 coquelicot	série de tests pour joindre un hôte réseau soudeur, à un hôte réseau ...	a day ago
 documentation/images	Add files via upload	just now
 gitlab/runner	Commit initial Coquelicot	a month ago
 hubot-init-rocketcha/construction	Update initialiser-rocketchat-pour-hubot	6 hours ago
 hubot	Commit initial Coquelicot	a month ago
 img	Commit initial Coquelicot	a month ago
 mongo-init-replica/construction	série de tests pour joindre un hôte réseau soudeur, à un hôte réseau ...	a day ago
 mongodb/construction	série de tests pour joindre un hôte réseau soudeur, à un hôte réseau ...	a day ago
 nginx	Commit initial Coquelicot	a month ago
 rocketchat	Update Dockerfile	6 hours ago
 tests/sondereseau	Commit initial Coquelicot	a month ago
 .env	Update .env	5 hours ago
 .gitignore	Commit initial Coquelicot	a month ago
 LICENSE	Commit initial Coquelicot	a month ago
 README.md	Update README.md	an hour ago
 coquelicot.sh	Commit initial Coquelicot	a month ago
 docker-compose.avec-tout-rocket-...	Commit initial Coquelicot	a month ago
 docker-compose.yml	Update docker-compose.yml	5 hours ago
 fichier-temp-marguerite	Commit initial Coquelicot	a month ago
 fichierr	Commit initial Coquelicot	a month ago
 gitlab-ci.yml.example.yml	Commit initial Coquelicot	a month ago
 initialisation-iaac-cible-deploiement...	série de tests pour joindre un hôte réseau soudeur, à un hôte réseau ...	a day ago
 operations-verbose.sh	Update operations-verbose.sh	a day ago
 operations.sh	Update operations.sh	a day ago

 README.md

A ranger

Comment récupérer l'adresse IP interne d'un conteneur

```
[jibl@pc-100 coquelicot]$ docker exec -it mongo bash -c "hostname --ip-address"
```

172.24.0.5

## Reprise

Donc l'utilisateur initial est bien créé par les var. d'environnement de `ROCKETCHAT`, mais alors l'utilisateur initial est "inactif".... :

```
Ceci est la 1-ère exécution du soudeur de rocketchat
alors l'utilisateur initial ROCKETCHAT est inactif { "active" : false, } , il ne m'est pas possible de créer .
```

çaaaaa y est ! enfin j'ai cramé l'erreur sur l'utilisateur initial :

```
#
# docker logs hubot-rocketchat-initializer -f | grep success
#
100 103 0 66 100 37 104 58 ---:---:---:---:---:---: 104
{"success":false,"error":"&quot;jibjib&quot; is not a valid username, use only letters, numbers, dots, hyphens
{"success":false,"error":"Email already exists. [403]","errorType":403}
{"success":false,"error":"Email already exists. [403]","errorType":403}
```

Oh ouil ::: !!!! Quand je mets la valeur en dur ...:

```
Ceci est la 12-ième exécution du soudeur de rocketchat

+ TESTS AVANT TOUT CURL
+ test ping "rocketchat"
ping: "rocketchat": Name or service not known
+ (encore, mais avec nom réseau rocketchat en dur) test ping "rocketchat"
PING rocketchat (172.18.0.6) 56(84) bytes of data.
64 bytes from rocketchat.coquelicot_devops (172.18.0.6): icmp_seq=1 ttl=64 time=0.097 ms
64 bytes from rocketchat.coquelicot_devops (172.18.0.6): icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from rocketchat.coquelicot_devops (172.18.0.6): icmp_seq=3 ttl=64 time=0.105 ms
64 bytes from rocketchat.coquelicot_devops (172.18.0.6): icmp_seq=4 ttl=64 time=0.101 ms

--- rocketchat ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.090/0.098/0.105/0.008 ms
+ test ping mongo
PING mongo (172.18.0.3) 56(84) bytes of data.
64 bytes from mongo.coquelicot_devops (172.18.0.3): icmp_seq=1 ttl=64 time=0.102 ms
64 bytes from mongo.coquelicot_devops (172.18.0.3): icmp_seq=2 ttl=64 time=0.086 ms
64 bytes from mongo.coquelicot_devops (172.18.0.3): icmp_seq=3 ttl=64 time=0.078 ms
64 bytes from mongo.coquelicot_devops (172.18.0.3): icmp_seq=4 ttl=64 time=0.086 ms

--- mongo ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.078/0.088/0.102/0.008 ms

+ test affichage adresse ip dans le réseau docker
172.18.0.7
```

Ok, j'avais donc un problème certain: l'injoignabilité de l'hôte réseau rocketchat, depuis le conteneur soudeur. Et j'ai fait toute une batterie de tests, pour vérifier, que certes, à son démarrage, le soudeur ne trouve pas l'hôte réseau docker du plongeur, mais quand le plongeur (rocketchat), sera up n running (tout en étant unhealthy, parce qu'il ne sera healthy que lorsque le soudeur aura terminé son travail avec succès), alors le soudeur pourra joindre l'hôte réseau plongeur :

```
[jibl@pc-100 ~]$ docker exec -it sondereseau bash -c "hostname -i "
172.18.0.2
[jibl@pc-100 ~]$ docker exec -it hubot-rocketchat-initializer bash -c "hostname -i "
```

- tester mes changements dans [hubot-init-rocketcha/construction/initiaiser-rocketchat-pour-hubot]

OUIIIIIIIIIII ça y est , j'arrive à faire un user register :

- Références du fix RocketChat : [le fix sur git.agiri.ninja](#), la [doc officielle](#) indiquant d'utiliser la variable `ADMIN_USER`, la [doc officielle](#) endpoint `users.register` indiquant une commande qui ne marche pas, avec payload JSON au lieu de form data, et le fix `users.register` fait par le CEO lui-même (Gabriel Engel) :



- 9/17/18, 7:56 PM

```
rocketchat@69c762fb5897:/app/bundle$ curl http://localhost:3000/api/v1/users.register -d '{ "username": "jibjil"
{"success":false,"error":"Match error: Missing key 'username'"}
```

- Pour tester l'existence d'un utilisateur avec les logs du healthcheck rocketchat :

```
docker exec -it sondereseau bash -c "curl http://rocketchat:3000/api/v1/login -d 'username=jibjib&password=sup"
```

- Pour inspecter les logs de HEALTHCHECK :

```
docker inspect --format "{{json .State.Health }}" rocketchat
```

Le healthcheck Rocketchat, ainsi que son soudeur, ne sont pas encore opérationnels. L'erreur obtenue sur le HEALTHCHECK :

```
[jib1@pc-100 coquelicot]$ docker inspect --format "{{json .State.Health }}" rocketchat
{"Status":"starting","FailingStreak":0,"Log":[]}
[jib1@pc-100 coquelicot]$ docker inspect --format "{{json .State.Health }}" rocketchat
{"Status":"starting","FailingStreak":2,"Log":[{"Start":"2018-08-17T05:34:33.71663102+02:00","End":"2018-08-17T
[jib1@pc-100 coquelicot]$ docker inspect --format "{{json .State.Health }}" rocketchat
{"Status":"starting","FailingStreak":9,"Log":[{"Start":"2018-08-17T05:34:50.30459341+02:00","End":"2018-08-17T
```

Ci-dessus, on voit que l'on passe de "connection refused", à "Unauthorized" Quant au soudeur, il n'est même pas encore dans le docker-compose.yml.

## L'utilisateur initial RocketChat, et l'utilisateur RocketChat utilisé par HUBOT

Grâce à <https://rocket.chat/docs/administrator-guides/create-the-first-admin/>, cette reette crée un premier utilisateur Administrateur RocketChat, qui sera utilisé pour ensuite créer l'utilisateur qui sera utilisé par le HUBOT.

La création d'utilisateur est faite avec : <https://rocket.chat/docs/developer-guides/rest-api/users/create/>

```
UTILISATEUR_ROCKETCHAT_HUBOT=jbl
UTILISATEUR_ROCKETCHAT_HUBOT_MDP=jbl
CHATROOM_ROCKETCHAT_HUBOT=receptionyooma
# L'utilisateur initial, n'est plus l'utilisateur RocketChat que HUBOT utilise: il crée l'utilisateur utilisé
# cf. [https://rocket.chat/docs/administrator-guides/create-the-first-admin/]
USERNAME_UTILISATEUR_ADMIN_INITIAL=jbl
MDP_UTILISATEUR_ADMIN_INITIAL=jbl
EMAIL_UTILISATEUR_ADMIN_INITIAL=jbl@jbl.io
```

Vraiment merci Github: un parfait exemple pour illustrer les design pattern Scalable dans un Kubernetes, avec NodeJS. Excitement ce que je devrai trouver dans mon bootiemachin... Expliquer notamment mon design pattern avec le HEALTHCHECK sur les replicaSet MongoDB, pour la scalabilité, se référer à un replicaSet, et non à un hôte réseau particulier, ou un nom de base de données particulière, du cluster mongo....

Oh :

```
Docker supports the following restart policies:
Policy      Result
no          Do not automatically restart the container when it exits. This is the default.
on-failure[:max-retries] Restart only if the container exits with a non-zero exit status. Optionally, 1
always      Always restart the container regardless of the exit status. When you specify a
unless-stopped Always restart the container regardless of the exit status, including on daemon
startup, except if the container was put into a stopped state before the Docker
```

```
daemon was stopped.
```

Donc, ce qu'il faut que je fasse, pour mon conteneur `mongo-init-replica` :

```
mongo-init-replica:
# + Et comme cela, mon conteneur ne re-démarrera que si son exécution a terminé avec un code
# + d'erreur: si le replicaSet n'a pas été correctement créé.
# + Quand le conteneur aura créé le replicaSet
restart: on-failure[:max-retries]
```

Alors, si je prends cette approche, le healthcheck qui vérifiera l'existence, et l'état du replicaSet, doit être placé dans le conteneur de la BDD MongoDB de RocketChat, et non dans le conteneur `mongo-init-replica`. Ainsi, de plus, on déclarera une seule dépendance, au lieu de deux, du conteneur RocketChat, vers le conteneur MongoDB. On supprime la dépendance du conteneur `rocketchat`, pour le conteneur `mongo-init-replica`. Oui/ Non, il y a un super design pattern à résoudre là : parce que le conteneur qui crée le replicaSet, lui, doit attendre non pas que le replicaSet soit existant et dans le statut PRIMARY, mais simplement que le serveur soit UP N RUNNING... Alors? Alors une possibilité est de retirer la dépendance du conteneur `mongo-init-replica`, pour le conteneur `mongo`, et mettre (dans le `./docker-compose.yml`) le conteneur `mongo-init-replica` en mode :

```
mongo-init-replica:
# + Et comme cela, mon conteneur ne re-démarrera que si son exécution a terminé avec un code
# + d'erreur: si le replicaSet n'a pas été correctement créé.
# + Quand le conteneur aura créé le replicaSet, il terminera son exécution, et ne re-démarrera pas.
- restart: on-failure[:max-retries]
```

Et en plus, le conteneur serveur de base de données `mongo` expose un statut Healthy cohérent avec ce qu'attend, dans le principe, RocketChat : un replicaSet MongoDB UP N RUNNING.

Ce qui serait encore plus intéressant, serait de développer un healthcheck, qui permette d'attester l'existence d'un utilisateur spécifié comme ci-dessous :

```
- ROCKETCHAT_ROOM=canal-de-test-jb1
- ROCKETCHAT_USER=jb1
- ROCKETCHAT_PASSWORD=jb1
- ROCKETCHAT_AUTH=password
- BOT_NAME=jb1
```

Le nouveau HEALTHCHECK pourrait vérifier :

- Avec le même module NodeJS / Meteor utilisé par HUBOT pour se logger à RocketChat, qu'on arrive bel et bien à se connecter avec ce user, et ce mot de passe. Donc en mode SOFT, pour tester l'infrastructure de manière agnostique, et non en allant chercher un contenu de Base de données, pour l'interpréter comme attestant de la création d'un utilisateur et de la validité de son mot de passe.
- Que la chatroom `$ROCKETCHAT_ROOM` a bien été créée dans RocketChat. Pour cela, on peut réutiliser les modules HUBOT qui permettent d'entrer et/ou de sortir d'une chatroom, avec le HUBOT. Et utiliser ce module pour vérifier l'existence de la chatroom.
- Enfin, il faudra automatiser la création de l'utilisateur initial RocketChat, et la création de l'utilisateur RocketChat que le HUBOT va utiliser. Cela pourrait aussi être un HEALTHCHECK du conteneur HUBOT directement, qui ainsi, s'il est "HEALTHY", on sait qu'il arrive à se connecter et à se logger. Dans le même goût, on pourra vérifier non seulement que la chatroom existe, mais aussi que le Bot a les autorisations attendues pour lire et/ou écrire dans la chatroom.
- UNE IDÉE DE PATTERN : je fais un conteneur hyper léger, qui ne va faire que continuellement essayer de créer le user et la CHATROOM rocketchat, pour le fonctionnement du HUBOT. Ce conteneur va faire un peu comme j'ai fait avec mon conteneur `mongo-init-replica`, il va continuellement tenter d'initialiser ce qu'il y a à initialiser (la CHATROOM, et le user que HUBOT va utiliser dans RocketChat), jusqu'à réussir. Pour obtenir ce comportement, on utilise la config :

```

hubot-init-rocketchat:
# + Et comme cela, mon conteneur ne re-démarrera que si son exécution a terminé avec un code
# + d'erreur: si le user ou la chatroom n'ont pas été correctement créés dans RocketChat.
# + Quand le conteneur aura créé le user et la chatroom, il terminera son exécution avec succès, et
# + ne re-démarrera pas. Pour terminer, le HUBOT est lui en "--restart=always", et sera HEALTHY, lorsque s
# + healthcheck en attestera (il faut donc un healthcheck aussi pour le conteneur HUBOT)
restart: on-failure:10
depends_on:
# + Logique: le conteneur a besoin que rocketChat soit UP'N RUNNING, pour pouvoir créer le user et la chat
- rocketchat

```

Reste un problème: Comment le conteneur HUBOT est-il notifié que le conteneur `hubot-init-rocketchat` a bien terminé son travail avec succès (que le user et la chatroom RocketChat ont bien été créés)?

Je pense à la réponse suivante: le conteneur HUBOT va lui aussi continuellement redémarrer, mais en mode `restart=always`, et il va comprendre un HEALTHCHECK, qui testera :

- Que l'on arrive bel et bien à se connecter au serveur RocketChat, et à s'y authentifier avec les USER et PWD précisés par configuration du HUBOT. LE user existe donc et a bien été créé par le conteneur `hubot-init-rocketchat`
- Que le user, maintenant que l'on sait qu'il existe, et permet de s'authentifier, dispose bien des droits pour lire et/ou écrire comme souhaités dans la chatroom créée.
- Et le conteneur Gitlab, aura un `depends_on` sur le conteneur `hubot`, si bien que lorsque Gitlab démarrera, il sera assuré de pouvoir s'exprimer sur les chatops. Cette dépendance est tout de même assez forte, peut-être trop: on pourrait s'en passer et considérer que Gitlab peut commencer à travailler sans disposer de son petit bot rocketchat. D'ailleurs on pourrait penser à mettre en oeuvre une recette de déploiement qui "dépose" toute cette infra avec RocketChat, sur un Gitlab déjà existant et exploité depuis des mois. NOM DU DESIGN PATTERN : SOUDEUR / PLONGEUR Rq:

Un robot pourrait très bien avoir uniquement possibilité de lire, il pourrait envoyer un email, ou un message sur une autre chatroom, juste pour prévenir que quelqu'un a dit tel truc sur telle chatroom, en citant le nom du produit phare de l'entreprise dans la même phrase...

Reste à faire.

J'ai laissé le problème de l'initialisation Gitlab (code HTTP 402 au changement initial du mot de passe de l'utilisateur initial), de côté : je le traiterais en dernier, parce que je l'ai rencontré dans d'autres contextes, et le sais indépendant du présent contexte de travail.

## Une technique pour des variables d'environnement globales.

En dehors des variables d'environnement utilisées dans le `docker-compose.yml` et déclarées dans les Dockerfiles, Cette recette utilise un certain nombre de variables globales, au sens qu'elles sont utilisées, dans le `docker-compose.yml`, pour définir plusieurs services différents.

Par exemple, l'utilisateur ROCKETCHAT que le HUBOT utilise, est utilisé par le conteneur `rocketchat`, pour l'exécution de son healthcheck "plongeur", mais aussi pour la définition du conteneur `hubot`.

Voici la liste des variables d'environnements, et détail de leur utilisation :

## Référence doc. officielle Docker

cf. doc officielle [<https://docs.docker.com/compose/environment-variables/#the-env-file>]

### The ".env" file

You can set default values for any environment variables referenced in the Compose file, or used to configure Compose, in an environment file named `.env`:

```
$ cat .env
```

```

TAG=v1.5

$ cat docker-compose.yml
version: '3'
services:
  web:
    image: "webapp:${TAG}"

```

Attention, un piège existe quant à ces variables d'environnement, et leur interpolation :

Lorsque vous précisez une valeur de variable d'environnement dans le `./docker-compose.yml` (ou un `Dockerfile`), et que :

- Dans votre `docker-compose.yml`, vous avez le contenu :

```

rocketchat:
  container_name: "$NOM_CONTENEUR_ROCKETCHAT"
  image: coquelicot/rocket.chat:1.0.0
  build:
    context: ./rocketchat/construction/
    args:
      - UNEVARIABLE_ENV_ROCKETCHAT_PAR_EXEMPLE="$VALEUR_UNEVARIABLE"

```

- Dans votre `./env`, vous avez le contenu :

```
UNEVARIABLE=bernard
```

Alors, la valeur envoyée sera `"bernard"`, et non la valeur `bernard` !!! (ce qui m'a causé quelques soucis pour passer certaines valeurs, comme des mots de passes, et des logs RocketChat dont la gestion semble largement améliorable ! (ne serait qu'en pensant aux pauvres architectes / devops, qui font un `docker logs rockerchat .... 100`)

## La Rest API RocketChat, pour écrire le healthcheck RocketChat / HUBOT : auth

Petit extrait de la [doc officielle](#) Pour vérifier l'auth avec le user HUBOT

### Notes

-> You will need to provide the authToken and userId for any of the authenticated methods. -> If your user has two-factor(2FA) authentication enabled, you must send a request like this. -> If LDAP authentication is enabled, you must maintain the login in the same way as you normally do. Similarly if 2FA is enabled for an LDAP user. Everything stays the same.

- Example Call - As Form Data

```
curl http://localhost:3000/api/v1/login \
-d "username=myusername&password=mypassword"
```

```
curl http://localhost:3000/api/v1/login \
-d "user=myusername&password=mypassword"
```

```
curl http://localhost:3000/api/v1/login \
-d "user=my@email.com&password=mypassword"
```

- Example Call - As JSON

```
curl -H "Content-type:application/json" \
http://localhost:3000/api/v1/login \
-d '{ "username": "myusername", "password": "mypassword" }'
```

```
curl -H "Content-type:application/json" \  
http://localhost:3000/api/v1/login \  
-d '{"user": "myusername", "password": "mypassword"}'
```

```
curl -H "Content-type:application/json" \  
http://localhost:3000/api/v1/login \  
-d '{"user": "my@email.com", "password": "mypassword"}'
```

- Example Call - When two-factor(2FA) authentication is enabled

```
curl -H "Content-type:application/json" \  
http://localhost:3000/api/v1/login \  
-d '{"totp": {"login": {"user": {"email": "rocket.cat@rocket.chat"}, "password": "password"}, "code"'
```

```
curl -H "Content-type:application/json" \  
http://localhost:3000/api/v1/login \  
-d '{"totp": {"login": {"user": {"username": "rocket.cat"}, "password": "password"}, "code": "224610
```

#### Result

```
{  
  "status": "success",  
  "data": {  
    "authToken": "9HqLlyZOugoStsXCuFD_0YdwnNnunAJF8V47U3QHXSq",  
    "userId": "aobEdbYhXfu5hkeqG",  
    "me": {  
      "_id": "aYjNnig8BEAWeQzMh",  
      "name": "Rocket Cat",  
      "emails": [  
        {  
          "address": "rocket.cat@rocket.chat",  
          "verified": false  
        }  
      ],  
      "status": "offline",  
      "statusConnection": "offline",  
      "username": "rocket.cat",  
      "utcOffset": -3,  
      "active": true,  
      "roles": [  
        "admin"  
      ],  
      "settings": {  
        "preferences": {}  
      }  
    }  
  }  
}
```

### La Rest API RocketChat, pour écrire le healthcheck RocketChat / HUBOT : entrer dans une CHATROOM

Petit extrait de la [doc officielle](#) Pour vérifier que le HUBOT peut entrer dans la chatroom qui a été créée, avec le user qui lui a été attribué.

Je n'ai pour l'instant trouvé que le moyen de vérifier l'existence de la CHATROOM :

#### Channel Info



Retrieves the information about the channel. URL Requires Auth HTTP Method /api/v1/channels.info yes GET Query Parameters Argument Example Required Description roomId ByehQjC44FwMeiLbX Required (if no roomId) The channel's id roomId general Required (if no roomId) The channel's name

- Example Call

```
curl -H "X-Auth-Token: 9HqLLyZ0ugoStsXCuFD_0YdwnNnunAJF8V47U3QHXSq" \
-H "X-User-Id: aobEdbYhXfu5hkeqG" \
http://localhost:3000/api/v1/channels.info?roomId=ByehQjC44FwMeiLbX
```

- Example Result

```
{
  "channel": {
    "_id": "ByehQjC44FwMeiLbX",
    "ts": "2016-11-30T21:23:04.737Z",
    "t": "c",
    "name": "testing",
    "usernames": [
      "testing",
      "testing1",
      "testing2"
    ],
    "msgs": 1,
    "default": true,
    "_updatedAt": "2016-12-09T12:50:51.575Z",
    "lm": "2016-12-09T12:50:51.555Z"
  },
  "success": true
}
```

## La Rest API RocketChat, pour écrire le soudeur RocketChat / HUBOT : créer une CHATROOM

Petit extrait de la [doc officielle](#) Pour créer la CHATROOM que le HUBOT va utiliser pour s'exprimer

### Payload

Argument Example Required Description name channelname Required The name of the new channel members ["rocket.cat"] Optional Default: [] The users to add to the channel when it is created. readOnly true Optional Set if the channel is read only or not. [Default: false]

- Example Call

```
curl -H "X-Auth-Token: 9HqLLyZ0ugoStsXCuFD_0YdwnNnunAJF8V47U3QHXSq" \
-H "X-User-Id: aobEdbYhXfu5hkeqG" \
-H "Content-type: application/json" \
http://localhost:3000/api/v1/channels.create \
-d '{ "name": "channelname" }'
```

- Example Result

```
{
  "channel": {
    "_id": "ByehQjC44FwMeiLbX",
    "name": "channelname",
    "t": "c",
    "usernames": [
      "example"
    ],
    "msgs": 0,
  }
}
```

```
{
  "u": {
    "_id": "aobEdbYhXfu5hkeqG",
    "username": "example"
  },
  "ts": "2016-05-30T13:42:25.304Z"
},
"success": true
}
```

## Un petit i-robot, bombardé de fleurs françaises...

Ce repository Git vis à faire évoluer la recette du repo :

<https://github.com/RocketChat/Chat.Code.Ship>

J'ai pu vérifier que la dernière version distribuée par l'équipe RocketChat, est :

<https://github.com/RocketChat/Rocket.Chat>

D'autres instructions sont présentes [ici](#)

Encore plus intéressant, il apparaît que dans cette dernière version distribuée par RocketChat, [l'équipe Rocket Chat a décidé](#) d'intégrer le Hubot directement à l'intérieur de l'application RocketChat. Je cite :

Meteor doesn't interact really well with NPM and the NPM module loading mechanism which hubot uses for its scripts. So we've split out most of hubot's scripts into this separate module.

To add a new hubot script:

- If it is packaged in npm (probably via the hubot-scripts organization on github), just add it to the package.json, for example with:

```
$ npm install --save hubot-pugme
```

- If it is included in the old hubot-scripts repository, just add it to the admin settings:

```
redis-brain.coffee, shipit.coffee, whatis.coffee, <new-script-name>.coffee
```

- If it is a custom script, or a forked/tweaked version of a script, add it to the scripts/ directory.

J'ai pu entre autres vérifier qu'une upgrade majeure du hubot-rocketchat avait été publiée, ce qui pourrait expliquer les problématiques que j'ai rencontrées.

- Mon but ultime, réaliser une utilisation du [type suivant](#), possible avec [HUBOT](#) :

```
extension is per-room customizable, for example: one room for open source project Rocket.Chat developers via github integration, another for Minecraft server farms operators discussion and network monitoring, yet another for a drone delivery service's fleet monitoring and control (see screenshot below)
```

Et notamment, permettre d'incruster dans une chat room, de la géolocalisation avec openstreetmap. Je veux ainsi pouvoir Brancher mon gitlab public, mon Jenkins public, mon twitter, mon facebook, mon fil de news site internet corporate, sur mon RocketChat. Je veux aussi ajouter comme le coup de l'exemple des drones + la géoloc, un canal qui permet à des humains d'une organisation de discuter pour assurer un service, avec le canal associé de remontée qualité des clients (SAV). Le tout avec deux schemas, un BPMN l'autre ITIL, pour chaque paire (canal de service delivery + canal SAV associé au service).

-

de la faire évoluer, pour qu'elle permette un calcul des SLA, et un déploiement Kubernetes, le service mpongo DB étant automatiquement "scalé" par le scale-up Kubernetes, en cohérence avec le replicaset créé pour rocketchat, mentionné dans la configuration du service rocketchat.

### Info intéressante

Sur [cette page](#), j'ai trouvé une info intéressante :

```
LISTEN_ON_ALL_PUBLIC will work if you set ROCKET_CHATROOM= ... with nothing after the =.  
  
LISTEN_ON_ALL_PUBLIC also activates DM - so no need to set both. When doing DM, remember to address the bot  
- i.e. bot pug me instead of pug me.  
  
Updating the doc and closing this ticket.  
  
Please open another ticket if you want to change the default GENERAL channel.
```

### Construire mon propre HUBOT pour MEs Jenkins Pipeline fleuris

Sur [cette page](#), l'auteur décrit comment re-construire le hubot-rocketchat, à partir de [hubot](#) tout seul. Autrement dit, je refais le build from source, et je n'aurai qu'à :

- regarder le code produit par l'équipe RocketChat, pour créer son hubot,
- bien apprendre les patterns décrits dans <https://hubot.github.com/docs/scripting/> Ceci afin :
- pour mattermost: <https://github.com/renanvicente/hubot-mattermost>
- de créer mes propres hubot pour mes Jenkins Pipeline fleuris, que j'aime une peu, beaucoup, à la folie, passionnément...
- de créer mes propres hubot pour tous les composants fleuris... cf :
  - <https://dzone.com/articles/how-to-create-a-simple-bot-using-hubot>
  - <https://github.com/RocketChat/hubot-rocketchat-boilerplate>
  - <https://www.icicletech.com/blog/automate-your-development-activities-with-hubot> (excellent celui-là, explique le dev avec le framework HUBOT). Extending the functionality for Hubot is as simple as placing the code in the "scripts" folder. et le code qu'il doit y avoir dans chacun de ces fichiers de script, doit respecter les règles linguistiques décrites dans la doc officielle "scripting, de hubot: <https://hubot.github.com/docs/scripting/>, et notamment importer en tant que dépendance, le robot, avec la syntaxe `module.exports = (robot) -> /* puis la suite du code Javascript, avec les méthodes à implémenter HEAR etc ... */`

### Construire mon propre HUBOT, avec sa personnalité, basée sur de l'intelligence artificielle

<https://www.chatbots.org/chatbot/a.i.i.c.e/>

## Utilisation

Pour exécuter cette recette une première fois :

```
export PROVISIONING_HOME=$(pwd)/coquelicot  
mkdir -p $PROVISIONING_HOME  
cd $PROVISIONING_HOME  
git clone "https://github.com/Jean-Baptiste-Lasselle/coquelicot" .  
chmod +x ./operations.sh  
./operations.sh
```

Soit, en une seule ligne :

```
export PROVISIONING_HOME=$(pwd)/coquelicot && mkdir -p $PROVISIONING_HOME && cd $PROVISIONING_HOME && git clone
```

Lorsque vous aurez exécuté une première fois l'instruction en une ligne ci-dessus, vous pourrez faire un cycle IAAC, sans re-télécharger d'image extérieures, en reconstruisant toutes les images qui ne sont pas téléchargées, avec :

```
export PROVISIONING_HOME=$(pwd)/coquelicot && docker-compose down && cd .. && sudo rm -rf $PROVISIONING_HOME &&
```

La commande ci-dessus, modulo une première exécution de commande, est idempotente

Pour exécuter cette recette une première fois, en mode verbeux :

```
export PROVISIONING_HOME=$(pwd)/coquelicot && mkdir -p $PROVISIONING_HOME && cd $PROVISIONING_HOME && git clone
```

## Mode plus léger

L'ensemble de l'infrastructure commence à peser, et notamment, pour optimiser le cycle de tests, il est bon de remarquer que cette recette implique :

```
docker pull centos:7
docker pull mongo:latest
docker pull gitlab/gitlab-ce:latest
docker pull gitlab/gitlab-runner:latest
docker pull rocketchat/rocket.chat:latest
docker pull rocketchat/hubot-rocketchat:latest
docker pull nginx:latest
```

C'est la liste des images téléchargées de l'extérieur. Il serait bon, pour la performance de votre cycle de tests, de ne télécharger qu'une seule et unique fois, à la première exécution de la recette.

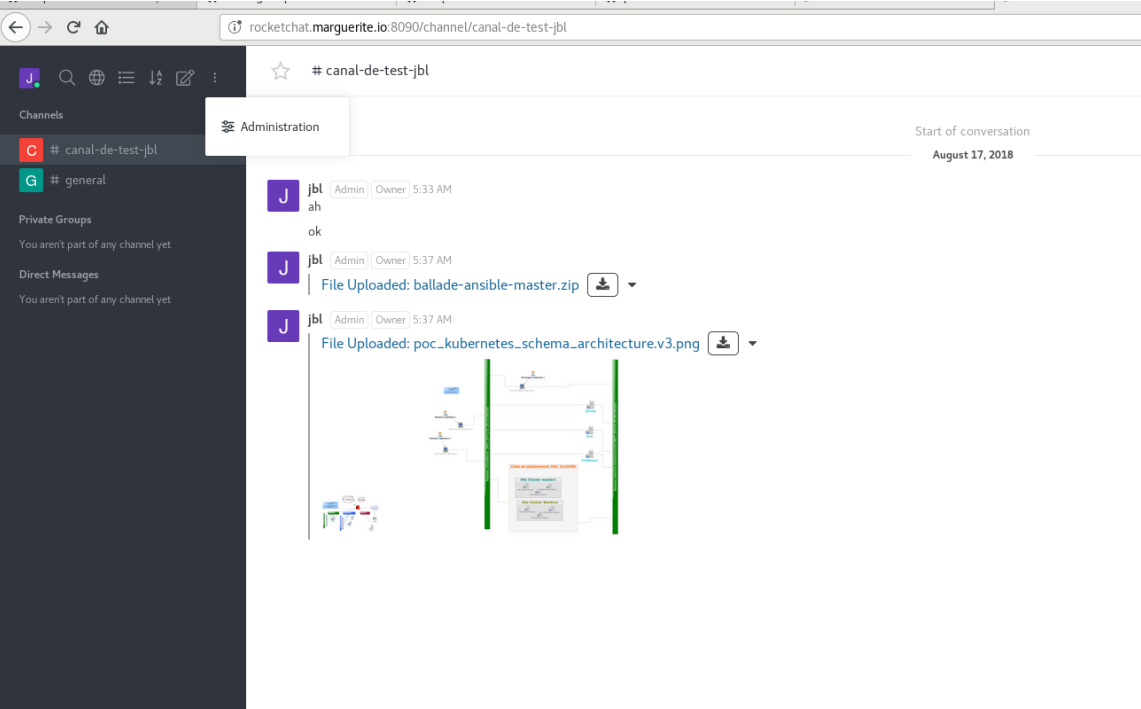
A chaque cycle, on pourra forcer la reconstruction build image docker locales, sans supprimer les images déjà prêtes et téléchargées :

```
docker-compose down && sudo rm -rf ./db && sudo rm -rf ./rocketchat/uploads/ && docker system prune -f && doc
```

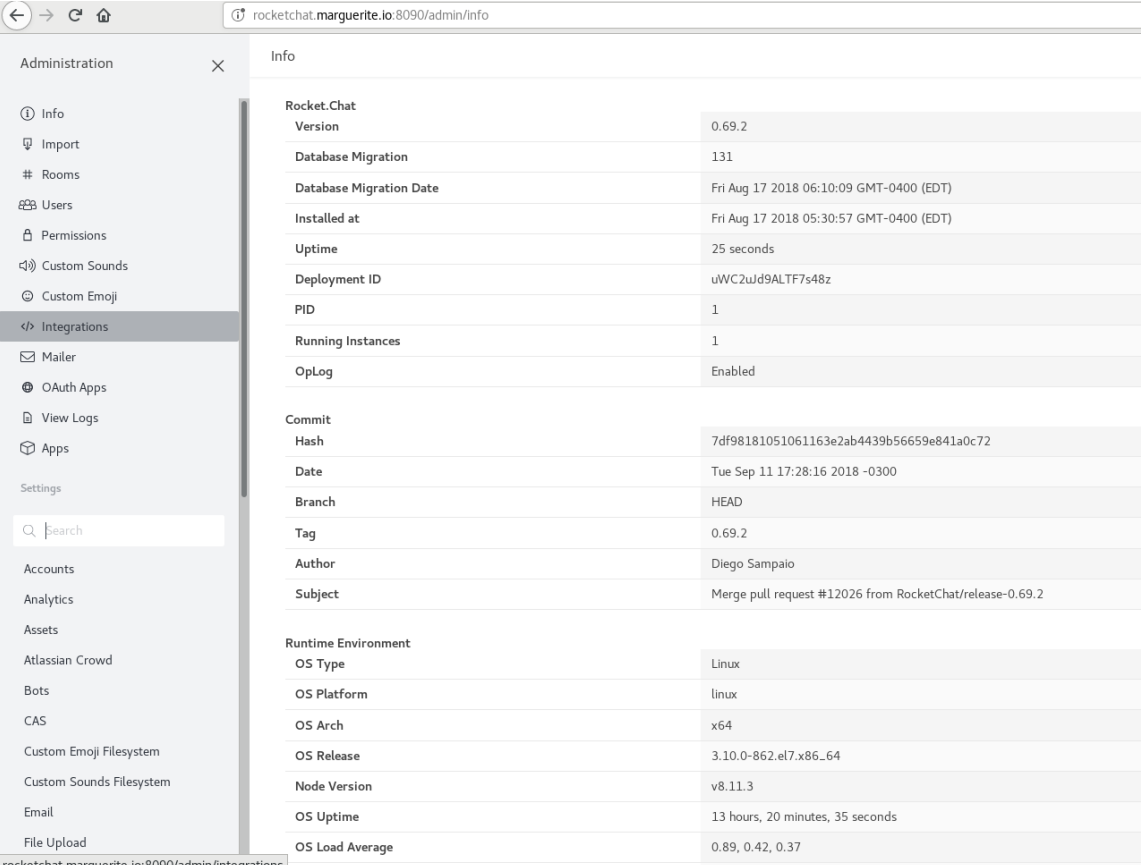
Lorsque vous exécuterez ces commandes, vous serez guidé interactivement :

- La recette s'exécutera
- Il vous sera demandé de créer un utilisateur rocketchat, qui devra correspondre à celui spécifié dans le `./docker-compose.yml`, avec les deux variables d'environnement `ROCKETCHAT_USER` et `ROCKETCHAT_PASSWORD` (cf. définition du conteneur `hubot`)
- Vous devrez de plus :
  - créer un fichier de script javascript qui contiendra le code Javascript de votre "Incoming WebHook", adapté à ma version de RocketChat, en vertu de la documentation officielle <https://rocket.chat/docs/administrator-guides/integrations/>,
  - Créer dans RocketChat un "Webhook" de type "Incoming" ( Menu Administration > Intégrations > Create New Integration (bouton en haut à droite de la page) :

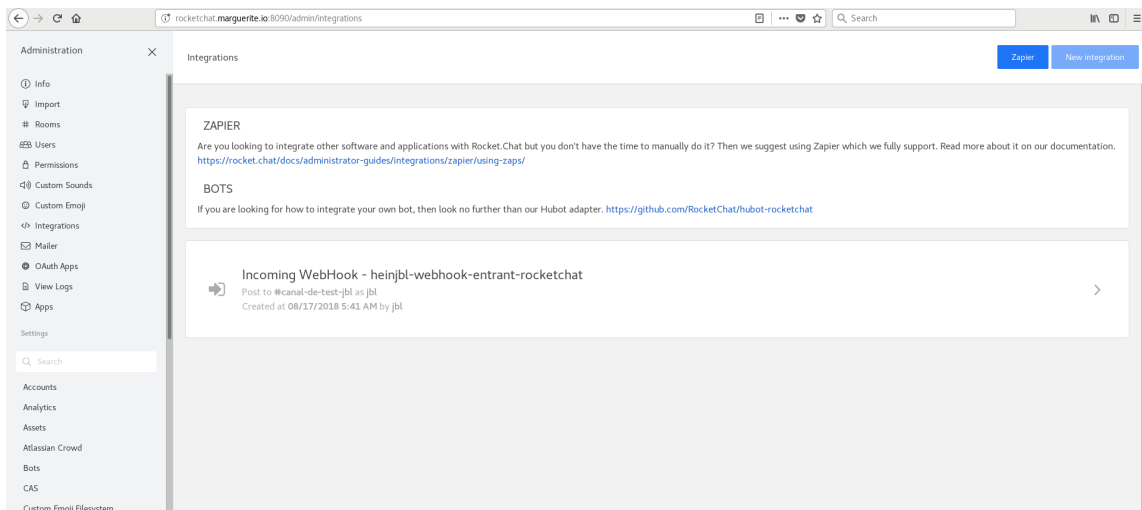
En haut à gauche, le menu "Administration" :



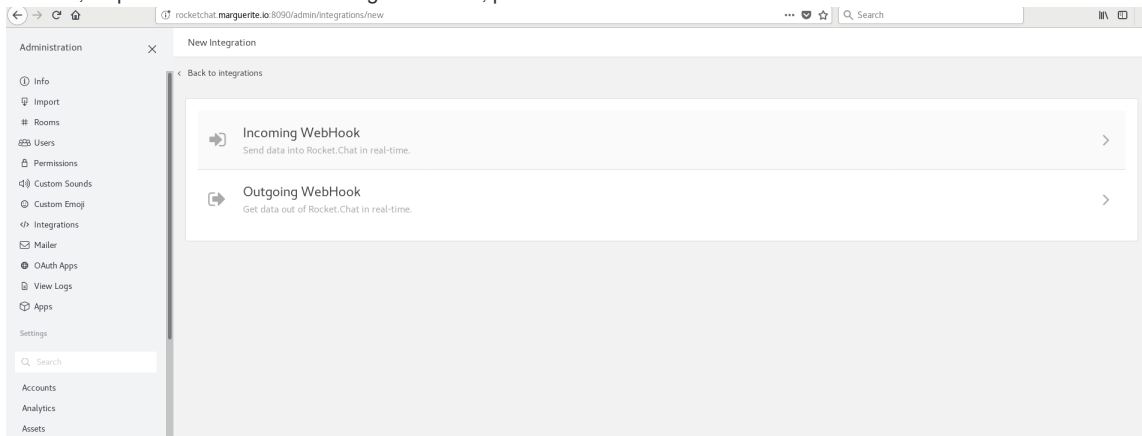
Sur le menu latéral gauche, le menu "Integrations" :



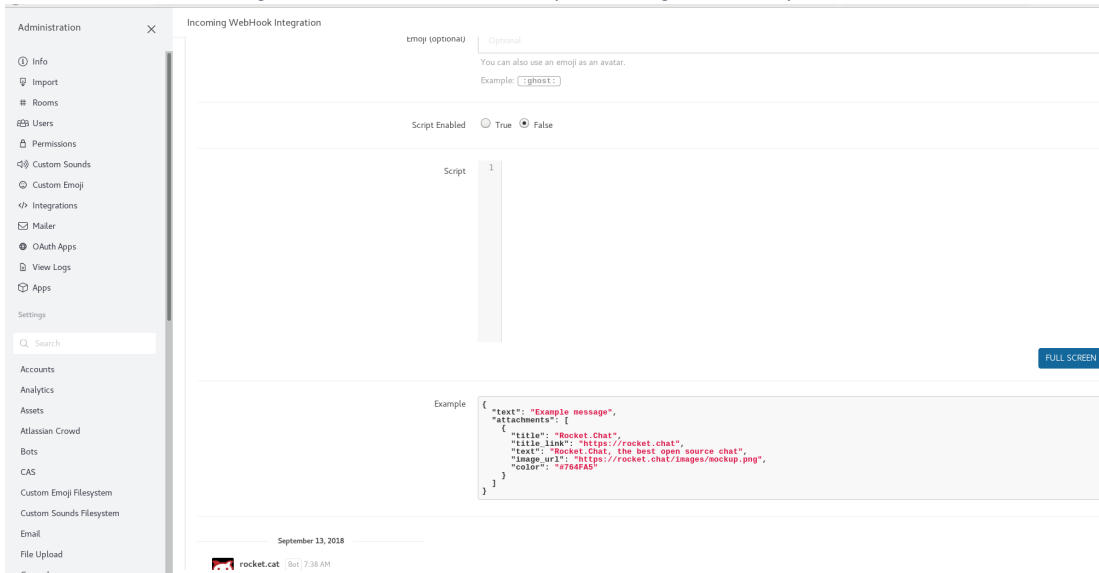
Le bouton bleu en haut à droite, "New Integration" :

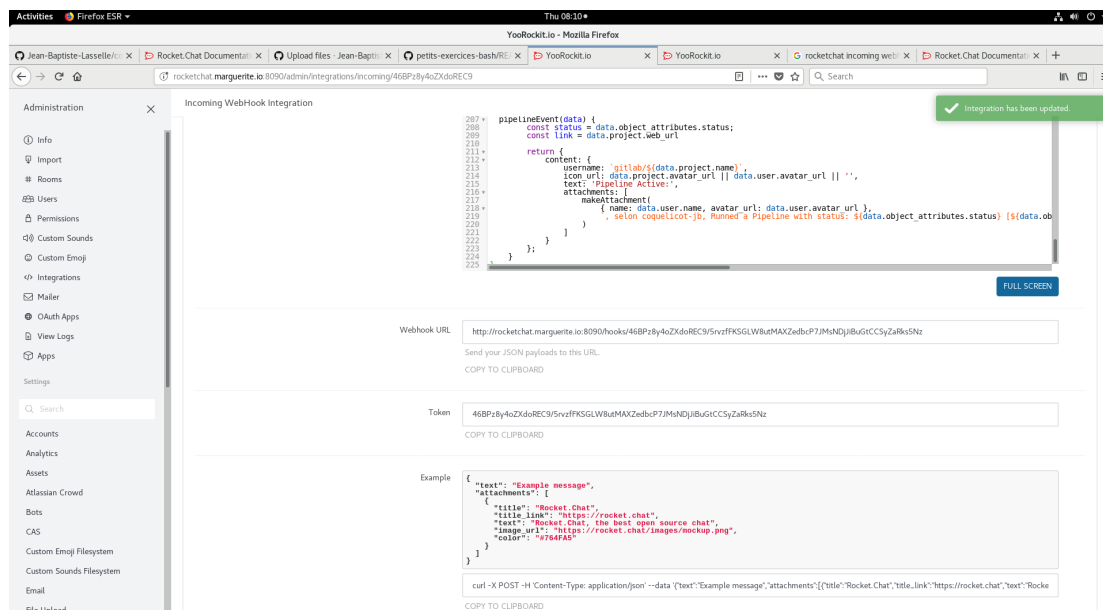


Et enfin, cliquer sur le bouton "Incoming Webhook", pour créer un webhook entrant :



- o Ce webhook entrant devra être configuré Pour l'utilisateur RocketChat que le HUBOT va utiliser, avec le formulaire qui s'affiche
- o Ce webhook doit être configuré de manière à utiliser le script "Incoming Webhook" que vous avez écrits :





- Maintenant RocketChat correctmeent démarré, et le Webhook entrant créé et configuré, la recette puet se terminer : ELLE va tout simplement redémarrer le `hubot` (cf. `./docker-compose.yml`). Il vous suffit de presser la touche entrée, pour laisser la recette se terminer.
- La recette se termine, et vous pourrez constater la sortie log suivante (pour le conteneur `hubot`), et attestant du succès de la connexion du HUBOT dans le serveur RocketChat :

```
$ docker logs -f hubot
```

```
npm info install hubot-rocketchat@1.0.11
npm info postinstall hubot-rocketchat@1.0.11
npm info install rocketbot@0.0.0
npm info postinstall rocketbot@0.0.0
npm info prepublish rocketbot@0.0.0
npm info ok
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Starting Rocketchat adapter version 1.0.11...
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Once connected to rooms I will respond to the name: jbllocks
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO I will also respond to my Rocket.Chat username as an alias: jbl
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Connecting To: rocketchat:3000
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Successfully connected!
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Logging In
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Successfully Logged In
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO rid: []
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO All rooms joined.
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Preparing Meteor Subscriptions..
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Subscribing to Room: __my_messages__
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Successfully subscribed to messages
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] INFO Setting up reactive message list...
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] WARNING Expected /home/hubot/scripts/package to assign a function to
[Fri Aug 17 2018 12:10:56 GMT+0000 (UTC)] WARNING Loading scripts from hubot-scripts.json is deprecated and wi.

Your hubot-scripts.json is empty, so you just need to remove it.
```

Manques de ce repo Git :

- Il reste à appliquer les instructions en fin de cette page, pour customiser les webhooks Gitlab. Pour ce faire, je vais donc utiliser l'interface graphique web de rocketchat, pour aller à l'admin et créer un "incoming webhook", et ce en utilisant le script suivant :

```

/* eslint no-console:0, max-len:0 */
// see https://gitlab.com/help/web_hooks/web_hooks for full json posted by GitLab
const NOTIF_COLOR = '#6498CC';
const refParser = (ref) => ref.replace(/^refs\/(?:tags|heads)\/(.+)$/, '$1');
const displayName = (name) => name.toLowerCase().replace(/\s+/g, '.');
const atName = (user) => (user && user.name ? '@' + displayName(user.name) : '');
const makeAttachment = (author, text) => {
  return {
    author_name: author ? displayName(author.name) : '',
    author_icon: author ? author.avatar_url : '',
    text,
    color: NOTIF_COLOR
  };
};

const pushUniq = (array, val) => ~array.indexOf(val) || array.push(val); // eslint-disable-line

class Script { // eslint-disable-line
  process_incoming_request({ request }) {
    try {
      let result = null;
      const channel = request.url.query.channel;
      switch (request.headers['x-gitlab-event']) {
        case 'Push Hook':
          result = this.pushEvent(request.content);
          break;
        case 'Merge Request Hook':
          result = this.mergeRequestEvent(request.content);
          break;
        case 'Note Hook':
          result = this.commentEvent(request.content);
          break;
        case 'Issue Hook':
          result = this.issueEvent(request.content);
          break;
        case 'Tag Push Hook':
          result = this.tagEvent(request.content);
          break;
        case 'Pipeline Hook':
          result = this.pipelineEvent(request.content);
          break;
      }
      if (result && result.content && channel) {
        result.content.channel = '#' + channel;
      }
      return result;
    } catch (e) {
      console.log('gitlabevent error', e);
      return {
        error: {
          success: false,
          message: e.message || e
        }
      };
    }
  }

  issueEvent(data) {
    return {
      content: {
        username: 'gitlab/' + data.project.name,
        icon_url: data.project.avatar_url || data.user.avatar_url || '',
        text: (data.assignee && data.assignee.name !== data.user.name) ? atName(data.a
        attachments: [
          makeAttachment(
            data.user,
            ` , selon coquelicot-jbl, ${data.object_attributes.state} an is

```



```

*Description:* ${data.object_attributes.description}.
See: ${data.object_attributes.url}`
    )
  ]
}
};
}

commentEvent(data) {
  const comment = data.object_attributes;
  const user = data.user;
  const at = [];
  let text;
  if (data.merge_request) {
    const mr = data.merge_request;
    const lastCommitAuthor = mr.last_commit && mr.last_commit.author;
    if (mr.assignee && mr.assignee.name !== user.name) {
      at.push(atName(mr.assignee));
    }
    if (lastCommitAuthor && lastCommitAuthor.name !== user.name) {
      pushUniq(at, atName(lastCommitAuthor));
    }
    text = `coquelicot-jbl: commented on MR [`${mr.id}`] [`${mr.title}`]](${comment.url})`;
  } else if (data.commit) {
    const commit = data.commit;
    const message = commit.message.replace(/\n[^\s\S]+/, '...').replace(/\n$/, '');
    if (commit.author && commit.author.name !== user.name) {
      at.push(atName(commit.author));
    }
    text = `coquelicot-jbl: commented on commit [`${commit.id.slice(0, 8)}`] [`${message}`]](${comment.url})`;
  } else if (data.issue) {
    const issue = data.issue;
    text = `coquelicot-jbl: commented on issue [`${issue.id}`] [`${issue.title}`]](${comment.url})`;
  } else if (data.snippet) {
    const snippet = data.snippet;
    text = `coquelicot-jbl: commented on code snippet [`${snippet.id}`] [`${snippet.title}`]](${comment.url})`;
  }
  return {
    content: {
      username: 'gitlab/' + data.project.name,
      icon_url: data.project.avatar_url || user.avatar_url || '',
      text: at.join(' '),
      attachments: [
        makeAttachment(user, `${text}\n${comment.note}`)
      ]
    }
  };
}

mergeRequestEvent(data) {
  const user = data.user;
  const mr = data.object_attributes;
  const assignee = mr.assignee;
  let at = [];

  if (mr.action === 'open' && assignee) {
    at = '\n' + atName(assignee);
  } else if (mr.action === 'merge') {
    const lastCommitAuthor = mr.last_commit && mr.last_commit.author;
    if (assignee && assignee.name !== user.name) {
      at.push(atName(assignee));
    }
    if (lastCommitAuthor && lastCommitAuthor.name !== user.name) {
      pushUniq(at, atName(lastCommitAuthor));
    }
  }
  return {

```

```

        content: {
          username: `gitlab/${mr.target.name}`,
          icon_url: mr.target.avatar_url || mr.source.avatar_url || user.avatar_url || '',
          text: at.join(' '),
          attachments: [
            makeAttachment(user, `${mr.action} MR [#${mr.id}] ${mr.title}](${mr.url})`
          ]
        }
      };
    }
  }

  pushEvent(data) {
    const project = data.project;
    const user = {
      name: data.user_name,
      avatar_url: data.user_avatar
    };
    // branch removal
    if (data.checkout_sha === null && !data.commits.length) {
      return {
        content: {
          username: `gitlab/${project.name}`,
          icon_url: project.avatar_url || data.user_avatar || '',
          attachments: [
            makeAttachment(user, `removed branch ${refParser(data.ref)} from ${project.name}`)
          ]
        }
      };
    }
    // new branch
    if (data.before == 0) { // eslint-disable-line
      return {
        content: {
          username: `gitlab/${project.name}`,
          icon_url: project.avatar_url || data.user_avatar || '',
          attachments: [
            makeAttachment(user, `pushed new branch [${refParser(data.ref)}] to ${project.name}`)
          ]
        }
      };
    }
    return {
      content: {
        username: `gitlab/${project.name}`,
        icon_url: project.avatar_url || data.user_avatar || '',
        attachments: [
          makeAttachment(user, `selon coquelicot-jbl, a poussé (pushed) ${data.ref} à la version ${data.checkout_sha} (sha)`, {
            text: data.commits.map((commit) => ` - ${new Date(commit.time).toLocaleDateString()} ${commit.message}`)
            color: NOTIF_COLOR
          })
        ]
      }
    };
  }

  tagEvent(data) {
    const tag = refParser(data.ref);
    return {
      content: {
        username: `gitlab/${data.project.name}`,
        icon_url: data.project.avatar_url || data.user_avatar || '',
        text: '@all',
        attachments: [
          makeAttachment(
            { name: data.user_name, avatar_url: data.user_avatar },
            `push tag [${tag}] ${data.checkout_sha.slice(0, 8)}](${data.project.url})`
          )
        ]
      }
    };
  }
}

```

```

    }
  };
}

pipelineEvent(data) {
  const status = data.object_attributes.status;
  const link = data.project.web_url

  return {
    content: {
      username: `gitlab/${data.project.name}`,
      icon_url: data.project.avatar_url || data.user.avatar_url || '',
      text: 'Pipeline Active:',
      attachments: [
        makeAttachment(
          { name: data.user.name, avatar_url: data.user.avatar_url },
          ` , selon coquelicot-jb, Runned a Pipeline with status: ${data..
        )
      ]
    }
  };
}
}

```

Qui est une simple petite modification du script donné en fin de cette documentation. Cette modification permet de définir des messages particuliers qui seront rapidement reconnaissables, quand postés par le `hubot`, dans RocketChat, suite à un évènement Gitlab.

## Architecture et orchestration des opérations de déploiement & exploit

### Rappels Docker Compose , depends\_on, HEALTHCHECKs

À rédiger

### Dépendances à l'exécution

Disponibilité du serveur `mongo` => dépendance du conteneur `mongo-init-replica` Disponibilité du serveur `mongo` => dépendance du conteneur `mongo-init-replica` Disponibilité du serveur `mongo` => dépendance du conteneur `mongo-init-replica`

### Mongo init replicaSet : doit attendre la dispo serveur Mongo

Je dois faire un HEALTHCHECK pour que le conteneur `mongo-init-replica` :

- Tente de créer le replicaSet, avec sa commande d'exécution ( `CMD ["/bin/bash"]` ...):

```
mongo mongo/rocketchat --eval "rs.initiate({ _id: 'rs0', members: [ { _id: 0, host: 'mongo:27017' } ]})"
```

- et vérifie que ce replicaSet existe, et "est en bonne santé" avec un HEALTHCHECK exécutant les commandes :

```
# - 1 - On commence par récupérer le statut du replicaSet : un objet JSON sera renvoyé.
mongo mongo/rocketchat --eval "rs.status()"
```

```
# - 2 - Ensuite, on vérifie que dans le JSON, on trouve bien mention de l'ID du replicaSet, et on vérifie son :
# tout en "parsant" le JSON. Pour cela, on doit utiliser la structure de l'output de cette commande, pré
# par la documentation officielle : https://docs.mongodb.com/manual/reference/command/replSetGetStatus/#
```

```
# -- d'après la doc officielle, il y a une section "members", qui liste tous les replciaSet: je suis quasi sûr
# qu'il est possible de parcourir l'arbre JSON à l'aide de commandes mongoDB, du genre du find()
# ou encore (à tester) :
# mongo mongo/rocketchat --eval "rs.status({ _id: 'rs0', members: [ { _id: 0, host: 'mongo:27017' } ]})"

# Donc, si :
# mongo mongo/rocketchat --eval "rs.status({ _id: 'rs0', members: [ { _id: 0, host: 'mongo:27017' } ]})"
# retourne une valeur, c'est que le replicaSet "rs0" existe.
# D'après [https://docs.mongodb.com/manual/reference/replica-states/], si le replmicaSet existe, il
# doit être dans l'état 1 "PRIMARY", pour que le replicaSet, formé d'une seule réplique, soit prêt à l'emploi
# RocketChat :
# -----
# Number      Name      State Description
# 0           STARTUP      Not yet an active member of any set. All members start up in this state. The
#                               mongod parses the replica # set configuration document while in STARTUP.
# 1           PRIMARY      The member in state primary is the only member that can accept write operations. Eligible
# -----
# Pour résumer, on attend que la commande retourne une valeur, et même la valeur "1". dans tous les autres cas,
# un "exit 1"
export RESULTAT_REQUETE_MONGO=$(mongo mongo/rocketchat --eval "rs.status({ _id: 'rs0', members: [ { _id: 0, host: 'mongo:27017' } ]})" |
if [ "$RESULTAT_REQUETE_MONGO" -eq "" ] then;
echo " DEBUG - [RESULTAT_REQUETE_MONGO=$RESULTAT_REQUETE_MONGO] "
echo " Ok, le replicaSet [rs0] a bien été créé dans l'hôte MongoDB [mongo:27017] "
exit 0
else
echo " Ok, le replicaSet [rs0] a bien été créé dans l'hôte MongoDB [mongo:27017] "
exit 0
fi
```

- Avec un tel HEALTHCHECK, et configuré avec "restart=always", le conteneur `mongo-init-replica` re-démarrera tant qu'il n'aura pas créé avec succès le réplicaSet attendu par le conteneur RocketChat.
- C'est enfin un pattern que je peux appliquer pour tous les développements d'applications scalable (qui supporte le scale up Kubernetes) NodeJS / Meteor / Mongoose / MongoDB.

## RocketChat : doit attendre la disponibilité du replicaSet rs0 dans le serveur Mongo

### Tests IAAC

Voici un exemple de résultat de test que j'ai mené, après avoir ainsi manuellement créé un canal RocketChat, et un webhook entrant, pour ensuite envoyer la requête CURL suivante (qui simule une émission d'évènement d'une instance GITLAB) :

```
curl -X POST -H "x-gitlab-event: Pipeline Hook" --data-urlencode 'payload={ "object_kind": "pipeline", "object": { "success": false, "error": "Invalid integration id or token provided." } }
```

Etant donné l'erreur qui m'est donnée en retour par le `hubot`, je cherche donc à modifier le token envoyé dans la requête.

Je prends la valeur mentionnée dans le `./docker-compose.yml`, pour configurer le `hubot`, avec les variables d'environnement `GITLAB_TOKEN` / `GITLAB_API_KEY`, et j'exécute ma requête modifiée :

```
curl -X POST -H "x-gitlab-event: Pipeline Hook" --data-urlencode 'payload={ "object_kind": "pipeline", "object": { "success": false, "error": "Invalid integration id or token provided." } }
```

et là, j'obtiens une réponse tout à fait différente du serveur RocketChat, une page HTML ! :

```
<!DOCTYPE html>
<html>
<head>
<meta name="referrer" content="origin-when-crossorigin">
<script>/* eslint-disable */

'use strict';
```

```

(function() {
  var debounce = function debounce(func, wait, immediate) {
    var timeout = void 0;
    return function () {
      var _this = this;

      for (var _len = arguments.length, args = Array(_len), _key = 0; _key < _len; _key++) {
        args[_key] = arguments[_key];
      }

      var later = function later() {
        timeout = null;
        !immediate && func.apply(_this, args);
      };

      var callNow = immediate && !timeout;
      clearTimeout(timeout);
      timeout = setTimeout(later, wait);
      callNow && func.apply(this, args);
    };
  };

  var cssVarPoly = {
    test: function test() {
      return window.CSS && window.CSS.supports && window.CSS.supports('--foo: red');
    },
    init: function init() {
      if (this.test()) {
        return;
      }

      console.time('cssVarPoly');
      cssVarPoly.ratifiedVars = {};
      cssVarPoly.varsByBlock = [];
      cssVarPoly.oldCSS = [];
      cssVarPoly.findCSS();
      cssVarPoly.updateCSS();
      console.timeEnd('cssVarPoly');
    },
    findCSS: function findCSS() {
      var styleBlocks = Array.prototype.concat.apply([], document.querySelectorAll('#css-var-'));
      var counter = 1;
      styleBlocks.map(function (block) {
        if (block.nodeName === 'STYLE') {
          var theCSS = block.innerHTML;
          cssVarPoly.findSetters(theCSS, counter);
          cssVarPoly.oldCSS[counter++] = theCSS;
        } else if (block.nodeName === 'LINK') {
          var url = block.getAttribute('href');
          cssVarPoly.oldCSS[counter] = '';
          cssVarPoly.getLink(url, counter, function (counter, request) {
            cssVarPoly.findSetters(request.responseText, counter);
            cssVarPoly.oldCSS[counter++] = request.responseText;
            cssVarPoly.updateCSS();
          });
        }
      });
    },
    findSetters: function findSetters(theCSS, counter) {
      cssVarPoly.varsByBlock[counter] = theCSS.match(/(--[^\s;]+:[^\s;]*)/g);
    },
    updateCSS: debounce(function () {
      cssVarPoly.ratifySetters(cssVarPoly.varsByBlock);
      cssVarPoly.oldCSS.filter(function (e) {
        return e;
      });
    });
  };
}());

```

```

    }).forEach(function (css, id) {
      var newCSS = cssVarPoly.replaceGetters(css, cssVarPoly.ratifiedVars);
      var el = document.querySelector('#inserted' + id);

      if (el) {
        el.innerHTML = newCSS;
      } else {
        var style = document.createElement('style');
        style.type = 'text/css';
        style.innerHTML = newCSS;
        style.classList.add('inserted');
        style.id = 'inserted' + id;
        document.getElementsByTagName('head')[0].appendChild(style);
      }
    });
  }, 100),

  replaceGetters: function replaceGetters(oldCSS, varList) {
    return oldCSS.replace(/var\(--.*?\)/gm, function (all, variable) {
      return varList[variable];
    });
  },

  ratifySetters: function ratifySetters(varList) {
    varList.filter(function (curVars) {
      return curVars;
    }).forEach(function (curVars) {
      curVars.forEach(function (theVar) {
        var matches = theVar.split(/:\s*/);
        cssVarPoly.ratifiedVars[matches[0]] = matches[1].replace(/;/, '');
      });
    });

    Object.keys(cssVarPoly.ratifiedVars).filter(function (key) {
      return cssVarPoly.ratifiedVars[key].indexOf('var') > -1;
    }).forEach(function (key) {
      cssVarPoly.ratifiedVars[key] = cssVarPoly.ratifiedVars[key].replace(/var\(--.*?/g, '');
      return cssVarPoly.ratifiedVars[key];
    });
  },

  getLink: function getLink(url, counter, success) {
    var request = new XMLHttpRequest();
    request.open('GET', url, true);
    request.overrideMimeType('text/css;');

    request.onload = function () {
      if (request.status >= 200 && request.status < 400) {
        if (typeof success === 'function') {
          success(counter, request);
        }
      } else {
        console.warn('an error was returned from:', url);
      }
    };

    request.onerror = function () {
      console.warn('we could not get anything from:', url);
    };

    request.send();
  }
};

var stateCheck = setInterval(function () {
  if (document.readyState === 'complete' && typeof Meteor !== 'undefined') {
    clearInterval(stateCheck);
    cssVarPoly.init();
  }
}, 100);

```

```

var DynamicCss = {};

DynamicCss.test = function () {
    return window.CSS && window.CSS.supports && window.CSS.supports('--foo: red');
};

DynamicCss.run = debounce(function () {
    var replace = arguments.length > 0 && arguments[0] !== undefined ? arguments[0] : false;

    if (replace) {
        var colors = RocketChat.settings.collection.find({
            _id: /theme-color-rc/i
        }, {
            fields: {
                value: 1,
                editor: 1
            }
        }).fetch().filter(function (color) {
            return color && color.value;
        });

        if (!colors) {
            return;
        }

        var css = colors.map(function (_ref) {
            var _id = _ref._id,
                value = _ref.value,
                editor = _ref.editor;

            if (editor === 'expression') {
                return '--' + _id.replace('theme-color-', '') + ': var(--' + value + ' '
            }

            return '--' + _id.replace('theme-color-', '') + ': ' + value + ' ';
        }).join('\n');
        document.querySelector('#css-variables').innerHTML = ':root {' + css + ' ';
    }

    cssVarPoly.init();
}, 1000);
})();
</script>

<link rel="icon" sizes="16x16" type="image/png" href="assets/favicon_16.png" />
<link rel="icon" sizes="32x32" type="image/png" href="assets/favicon_32.png" />
<link rel="icon" sizes="any" type="image/svg+xml" href="assets/favicon.svg" />
<title>YooRockit.io</title><meta name="application-name" content="YooRockit.io"><meta name="apple-mobile-web-a
<meta http-equiv="content-language" content=""><meta name="language" content="">
<meta name="robots" content="INDEX, FOLLOW">
<meta name="msvalidate.01" content="">
<meta name="google-site-verification" content="">
<meta property="fb:app_id" content="">

<base href="/">

<link rel="stylesheet" type="text/css" class="__meteor-css__" href="/9b704c621adc8a1dcae59307e0023894dbfd241:
<link rel="stylesheet" type="text/css" class="__meteor-css__" href="/theme.css?1807b422d007328786a00d34e6c63:
<meta charset="utf-8" />
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="expires" content="-1" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="fragment" content="!" />
<meta name="distribution" content="global" />
<meta name="rating" content="general" />
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"

```

```

<meta name="mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="msapplication-TileImage" content="assets/tile_144.png" />
<meta name="msapplication-config" content="images/browserconfig.xml" />
<meta property="og:image" content="assets/favicon_512.png">
<meta property="twitter:image" content="assets/favicon_512.png">
<link rel="manifest" href="images/manifest.json" />
<link rel="chrome-webstore-item" href="https://chrome.google.com/webstore/detail/nocfbnmjnnkbpkabodi
<link rel="mask-icon" href="assets/safari_pinned.svg" color="#04436a">
<link rel="apple-touch-icon" sizes="180x180" href="assets/touchicon_180.png" />
<link rel="apple-touch-icon-precomposed" href="assets/touchicon_180_pre.png">

<script src="/packages/es5-shim/es5-shim-sham.min.js"></script>
</head>
<body>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" style="display: none">
  <symbol viewBox="0 0 20 20" id="icon-add-reaction">
    <g fill="none" fill-rule="evenodd">
      <g transform="translate(3 3)">
        <circle fill="currentColor" cx="9" cy="5" r="1" />
        <circle fill="currentColor" cx="5" cy="5" r="1" />
        <path d="M7 0a7 7 0 1 0 14 7 7 7 0 0 7-7M4.172 9.328a4 4 0 0 0 5.656 0" stroke="currentColor"
      </g>
      <path d="M16.2 1.2v5.2m-2.6-2.6h5.2" stroke="currentColor" stroke-width="1.5" stroke-linecap="squa
    </g>
  </symbol>
  <symbol viewBox="0 0 512 512" id="icon-at">
    <path d="M256 8C118.941 8 8 118.919 8 256c0 137.058 110.919 248 248 248 52.925 0 104.68-17.078 147.092
  </symbol>
</svg>

[Je vous passe le reste du HTML...]
</body>
</html>

```

OK, il faut ré-implémenter ce test sur son principe : Il faut pouvoir créer des requêtes CURL, qui correspondraient exactement à des requêtes qu'enverrait mon Gitlab, avec des webhooks etc.. Il me manque pour l'instant l'intégration Gitlab, et je dois donc la mocker, pour tester l'intégration hubot RocketChat, indépendamment de l'intégration Gitlab HUBOT . C'est du test d'intégration infra typique.

Finalement, Les Jenkins pipelines utiliseront aussi leur propre HUBOT pour poster des messages aux développeurs par exemple.

Notez :

- J'ai ajouté deux HEALTHCHECK élémentaire pour les conteneurs MongoDB et RocketChat :
  - Pour MongoDB, Le conteneur `mongo-init-replica` doit attendre que le serveur MongoDB, soit disponible, pour faire son travail : créer le replicaSet que RocketChat va Utiliser.
  - De plus, Le serveur MongoDB, doit
  - Pour RocketChat, on doit attendre que le service soit disponible, avant de redémarrer le service Hubot (pilote par Gitlab).
- Trouver le moyen de faire fonctionner les depends\_on avec les HEALTHCHECK de chaque conteneur. Y compris le conteneur qui initialise le replicaset : Quand il a terminé son travail, alors seulement l'instance applicative RocketChat peut démarrer.

## Dernière erreur obtenue dans ma pile de travail

### Déboguer un HEALTH\_CHECK s'exécutant dans un conteneur

Je cherche à faire marcher mon HEALTHCHECK MONGODB Pour visualiser les logs de l'exécution du healthcheck :



```
docker inspect --format "{{json .State.Health }}" mongo
```

J'ai aussi le test suivant :

```
# -
# Le fichier exécutable "replicaset-health-check.silencieux" constitue le healthcheck, et
# est déjà présent dans le conteneur, dans le répertoire [/usr/local/bin]
# Le fichier exécutable "replicaset-health-check" est exactement identique à son pendant silencieux, sauf qu'il
# comprend des instructions echo, pour afficher des valeurs, en vue de leur vérification pour test. Il est déjà
# présent dans le conteneur, dans le répertoire "/testlive" :
# -
docker exec -it mongo bash -c "./testlive/replicaset-health-check"
```

## Secondaire

C'est mon Gitlab qui me pose problème, notamment lorsque je dois changer le mot de passe de l'utilisateur initial, j'obtiens une erreur 402, et les logs suivants sur le serveur :

```
==> /var/log/gitlab/gitlab-rails/production.log <==
Started PUT "/users/password" for 192.168.80.9 at 2018-08-17 10:27:01 +0000
Processing by PasswordsController#update as HTML
  Parameters: {"utf8"=>"✓", "authenticity_token"=>"[FILTERED]", "user"={"reset_password_token"=>"[FILTERED]"},
Can't verify CSRF token authenticity
Completed 422 Unprocessable Entity in 4ms (ActiveRecord: 0.0ms)

==> /var/log/gitlab/gitlab-rails/production_json.log <==
{"method":"PUT","path":"/users/password","format":"html","controller":"PasswordsController","action":"update",

==> /var/log/gitlab/gitlab-rails/production.log <==

ActionController::InvalidAuthenticityToken (ActionController::InvalidAuthenticityToken):
  lib/gitlab/middleware/multipart.rb:97:in `call'
  lib/gitlab/request_profiler/middleware.rb:14:in `call'
  lib/gitlab/middleware/go.rb:17:in `call'
  lib/gitlab/etag_caching/middleware.rb:11:in `call'
  lib/gitlab/middleware/rails_queue_duration.rb:22:in `call'
  lib/gitlab/metrics/rack_middleware.rb:15:in `block in call'
  lib/gitlab/metrics/transaction.rb:53:in `run'
  lib/gitlab/metrics/rack_middleware.rb:15:in `call'
  lib/gitlab/middleware/read_only/controller.rb:38:in `call'
  lib/gitlab/middleware/read_only.rb:16:in `call'
  lib/gitlab/middleware/basic_health_check.rb:25:in `call'
  lib/gitlab/request_context.rb:18:in `call'
  lib/gitlab/metrics/requests_rack_middleware.rb:27:in `call'
  lib/gitlab/middleware/release_env.rb:10:in `call'
```

## Piste

J'ai trouvé [ici](#) sur le oueb :

```
Pablo @pablolarrimbe · 1 year ago

@tyranron Thanks for the response. As you can see, without setting that parameter, most of my images still dis

The problem that I have if I change the external URL is that then my reverse proxy can't talk with the nginx s
Tyranron
Tyranron @tyranron · 1 year ago

@pablolarrimbe what kind of HTTP conflict appears? Would you be so kind to provide some logs with corresponden

I have similar setup in Kubernetes cluster where Nginx acts as reverse-proxy. Following strait-forward config
```

```
external_url 'https://gitlab.reverse-proxy.domain.com' # corrected according to your domains
nginx['listen_port'] = 80
nginx['listen_https'] = false
nginx['proxy_set_headers'] = {
  'X-Forwarded-Proto' => 'https',
  'X-Forwarded-Ssl' => 'on'
}
```

In my case SSL termination happens on reverse-proxy and traffic between reverse-proxy and Gitlab is not secure

## Plus en détails

Pour tout détruire et nettoyer :

```
docker-compose down --rmi all && docker system prune -f
```

Pour déboguer cette recette :

```
export PROVISIONING_HOME=$(pwd)/coquelicot && mkdir -p $PROVISIONING_HOME && cd $PROVISIONING_HOME && git clone
```

Ou :

```
docker-compose down --rmi all && docker system prune -f && docker-compose --verbose build && docker-compose --
```

Pour inspecter les logs d'exécution de chaque conteneur :

```
export NOM_DU_CONTENEUR=gitlab
export NOM_DU_CONTENEUR=mongo
export NOM_DU_CONTENEUR=mongo-init-replica
export NOM_DU_CONTENEUR=rocketchat
export NOM_DU_CONTENEUR=hubot
```

```
docker logs $NOM_DU_CONTENEUR
```

Pour tester la connectivité entre deux conteneurs, cette recette met à disposition un service 'sondereseau', que l'on peut utiliser de la manière suivante :

```
# - lien entre les conteneurs 'gitlab' et 'rocketchat'
export NOM_DU_CONTENEUR1=gitlab
export NOM_DU_CONTENEUR2=rocketchat
# - lien entre les conteneurs 'mongo-init-replica' et 'mongo'
export NOM_DU_CONTENEUR1=mongo-init-replica
export NOM_DU_CONTENEUR2=mongo
# - lien entre les conteneurs 'rocketchat' et 'mongo'
export NOM_DU_CONTENEUR1=rocketchat
export NOM_DU_CONTENEUR2=mongo
# - lien entre les conteneurs 'rocketchat' et 'hubot'
export NOM_DU_CONTENEUR2=rocketchat
export NOM_DU_CONTENEUR1=hubot

# - on installe l'utilitaire linux 'ping' dans les conteneurs à tester :

# - Vérifier les OS dans chaque conteneur ... :
docker exec -it $NOM_DU_CONTENEUR1 bash -c "apt-get update -y && apt-get install -y iputils-ping && ping -c 4 localhost"
docker exec -it $NOM_DU_CONTENEUR1 bash -c "yum update -y && yum install -y iputils && ping -c 4 localhost"
docker exec -it $NOM_DU_CONTENEUR1 bash -c "apk update -y && apk add -y iputils-ping && ping -c 4 localhost"
```

```

docker exec -it $NOM_DU_CONTENEUR2 bash -c "apt-get update -y && apt-get install -y iputils-ping && ping -c 4 ."
docker exec -it $NOM_DU_CONTENEUR2 bash -c "yum update -y && yum install -y iputils && ping -c 4 localhost"
docker exec -it $NOM_DU_CONTENEUR2 bash -c "apk update -y && apk add -y iputils-ping && ping -c 4 localhost"

# -- on utilise la propriété de transitivité du ping réseau, vu en tant que morphisme (le morphisme existe, si
export COMMANDE="ping -c 4 $NOM_DU_CONTENEUR1"

echo " sonde réseau : ping du conteneur [NOM_DU_CONTENEUR1] vers le conteneur [sondereseau] : "
docker exec -it $NOM_DU_CONTENEUR1 bash -c "ping -c 4 sondereseau"
echo " sonde réseau : ping du conteneur [sondereseau] vers le conteneur [NOM_DU_CONTENEUR2] : "
docker exec -it sondereseau bash -c "ping -c 4 $NOM_DU_CONTENEUR2"
# - composée :
echo " sonde réseau : ping du conteneur [NOM_DU_CONTENEUR1] vers le conteneur [NOM_DU_CONTENEUR2] : "
docker exec -it $NOM_DU_CONTENEUR1 bash -c "ping -c 4 $NOM_DU_CONTENEUR2"

```

Ok, les tests préliminaires m'amènent à conclure, qu'il faut "réparer" la configuration réseau docker-compose, afin de pouvoir utiliser la recette. Une configuration plus fine, devrait être orchestrée par un Ansible / Terraform.

Il faudra corriger pour que les conteneurs ne fasse mention réciproque que de leur nom de domaine nginx, dans leurs configurations respectives. Et l'ensemble de l'intégration dépendra de la configuration de la résolution de noms de domaines, dans la pile réseau docker.

## Les fichiers qui mentionnent des noms de domaine

Ces fichiers devront donc faire l'objet d'une "templatisation Ansible Role".

```

./gitlab/runner/config.toml
./gitlab/runner/servers.conf
./nginx/chatops.conf
./nginx/hosts

```

Pour Hubot et Rocketcat, je n'ai trouvé aucune mention notable de nom de domaine ou adresse IP pour désigner un hôte réseau, dans une configuration.

## Tests réseaux

```

rm -rf ./fichier-temp-marguerite
echo "$(pwd)" >> ./fichier-temp-marguerite
cat ./fichier-temp-marguerite
export NOM_REPERTOIRE_COURANT=$(awk -F / '{print $4}' ./fichier-temp-marguerite)
echo " NOM_REPERTOIRE_COURANT=$NOM_REPERTOIRE_COURANT"
echo "-----"

export NOM_DU_RESEAU_A_ANALYSER="$NOM_REPERTOIRE_COURANT"_devops
echo " NOM_DU_RESEAU_A_ANALYSER=$NOM_DU_RESEAU_A_ANALYSER"
echo "-----"

export NOM_CONTENEUR_SONDE_RESEAU=sonde-reseau-marguerite

docker run -it --name $NOM_CONTENEUR_SONDE_RESEAU --network='$NOM_DU_RESEAU_A_ANALYSER' -d centos:7
docker exec -it conteneur-sonde-reseau bash -c "yum update -y && yum install -y iputils"

```

Puis, pour exécuter un test de connexion réseau IP :

```

export NOM_DU_CONTENEUR=rocketchat
export NOM_DHOTE_RESEAU_IP=$NOM_CONTENEUR
export NOM_CONTENEUR_SONDE_RESEAU=sonde-reseau-marguerite
docker exec -it $NOM_CONTENEUR_SONDE_RESEAU bash -c "ping -c 4 $NOM_DHOTE_RESEAU_IP"

```

## sondereseau

La présente recette provisionne (cf. `./docker-compose.yml`) un conteneur dont le nom est `sondereseau`, qui peut être utilisé pour réaliser des tests réseaux, notamment pour tester les réseaux par lesquels les conteneurs échangent entre eux.

### Nota Bene

Le petit script :

```
rm -rf ./fichier-temp-marguerite
echo "$(pwd)" >> ./fichier-temp-marguerite
cat ./fichier-temp-marguerite
export NOM_REPERTOIRE_COURANT=$(awk -F / '{print $4}' ./fichier-temp-marguerite)
echo " NOM_REPERTOIRE_COURANT=$NOM_REPERTOIRE_COURANT"
echo "-----"
```

N'est qu'une astuce Bash, pour permettre de "calculer" le nom du réseau créé par la configuration `docker-compose.yml`. J'y ait eut recours, parce que je n'ai pas encore trouvé de manière de forcer le nommage du réseau ainsi créé.

## Dernières erreurs

Tous les conteneurs démarrent correctement, sauf le conteneur `rocketchat`. En examinant les logs du conteneur, je constate que c'est la connexion, du conteneur `rocketchat`, vers le conteneur `mongo`, qui pose problème, mais que l'hôte réseau IP est joignable :

```
$ docker ps -a
CONTAINER ID        IMAGE                                     COMMAND                  CREATED            STATUS
56ed881530b9        mongo:latest                          "docker-entrypoint.s..." 5 minutes ago      Exited (
2b867f1182de        rocketchat/rocket.chat:latest         "node main.js"           5 minutes ago      Exited (
945fb625f1f3        marguerite/sonde-reseau:0.0.1         "/bin/bash"              5 minutes ago      Up 5 min
fc6a83c17edc        marguerite/mongo:1.0.0                "docker-entrypoint.s..." 5 minutes ago      Up 5 min
8430360cda16        jbl/gitlab-runner:latest              "/usr/bin/dumb-init ..." 5 minutes ago      Up 5 min
13cc988fa8e0        nginx                                  "nginx -g 'daemon of..." 5 minutes ago      Exited (
8f7f30b60402        gitlab/gitlab-ce:latest               "/assets/wrapper"        5 minutes ago      Up 5 min
7fb1baeac47f        rocketchat/hubot-rocketchat:latest     "/bin/sh -c 'node -e..." 5 minutes ago      Up 5 min
520a7b41e9a6        nginx:latest                          "nginx -g 'daemon of..." 12 hours ago       Up 12 ho
f6abad0ffcdb        jenkins:2.60.3                       "/bin/tini -- /usr/l..." 12 hours ago       Up 12 ho
b8546132b2be        centos:7                              "/bin/bash"              13 hours ago       Up 13 ho

$ docker exec -it sondereseau bash -c "ping -c 4 mongo"
PING mongo (172.28.0.7) 56(84) bytes of data.
64 bytes from mongo.gitlab-rocketeer_devops (172.28.0.7): icmp_seq=1 ttl=64 time=0.196 ms
64 bytes from mongo.gitlab-rocketeer_devops (172.28.0.7): icmp_seq=2 ttl=64 time=0.127 ms
^C
--- mongo ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.127/0.161/0.196/0.036 ms
$ docker logs rocketchat

/app/bundle/programs/server/node_modules/fibers/future.js:313
    throw(ex);
    ^
MongoError: failed to connect to server [mongo:27017] on first connect [MongoError: connect ECONNREFUSED 172.2
  at Pool.<anonymous> (/app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mongodb-co
  at emitOne (events.js:116:13)
  at Pool.emit (events.js:211:7)
  at Connection.<anonymous> (/app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mong
  at Object.onceWrapper (events.js:317:30)
  at emitTwo (events.js:126:13)
  at Connection.emit (events.js:214:7)
  at Socket.<anonymous> (/app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mongodb-
  at Object.onceWrapper (events.js:315:30)
  at emitOne (events.js:116:13)
```

```

at Socket.emit (events.js:211:7)
at emitErrorNT (internal/streams/destroy.js:64:8)
at _combinedTickCallback (internal/process/next_tick.js:138:11)
at process._tickCallback (internal/process/next_tick.js:180:9)

```

Avec plus de tests réseaux, j'ai déterminé que le conteneur ROPketchat cessait son exécution, car il n'arrivait pas à se connecter à la base de données dans le conteneur `mongo` : Le temps que la base de données MongoDB soit "Up n running", le conteneur rocketchat a déjà échoué à la connexion BDD.

J'ai d'abord ajouté un Healthcheck à ma définition de service mongodb, avec un dockerfile, cf. `mongodb/construction/dockerfile` : Aucun effet, malgré la directive `depends_on` ( `./docker-compose.yml` ) qui déclare une dépendance explicite du conteneur `rocketchat` , pour le conteneur `mongo` .

J'ai ensuite ajouté une directive `restart: always` , pour que le conteneur rocketcaht, re-démarré à chaque fois qu'il stoppe son exécution: ainsi, il re-démarré, jusqu'à ce que la BDD MongoDB soit disponible.

Cette solution ne me satisfait pas à terme, car elle ne tire pas profit de la définition des HEALTHCHECK, et de leur implication dans la définition et le paramétrage des SLAs.

Pour terminer, J'en arrive à un point où une seule erreur subsiste, et c'est une nouvelle erreur pour le conteneur rocketchat :

```

$ docker logs rocketchat
/app/bundle/programs/server/node_modules/fibers/future.js:280
    throw(ex);
    ^

Error: $MONGO_OPLOG_URL must be set to the 'local' database of a Mongo replica set
    at OplogHandle._startTailing (packages/mongo/oplog_tailing.js:218:13)
    at new OplogHandle (packages/mongo/oplog_tailing.js:76:8)
    at new MongoConnection (packages/mongo/mongo_driver.js:214:25)
    at new MongoInternals.RemoteCollectionDriver (packages/mongo/remote_collection_driver.js:4:16)
    at Object.<anonymous> (packages/mongo/remote_collection_driver.js:38:10)
    at Object.defaultRemoteCollectionDriver (packages/underscore.js:784:19)
    at new Collection (packages/mongo/collection.js:97:40)
    at new AccountsCommon (packages/accounts-base/accounts_common.js:23:18)
    at new AccountsServer (packages/accounts-base/accounts_server.js:18:5)
    at server_main.js (packages/accounts-base/server_main.js:9:12)
    at fileEvaluate (packages/modules-runtime.js:343:9)
    at require (packages/modules-runtime.js:238:16)
    at /app/bundle/programs/server/packages/accounts-base.js:2012:15
    at /app/bundle/programs/server/packages/accounts-base.js:2019:3
    at /app/bundle/programs/server/boot.js:411:36
    at Array.forEach (<anonymous>)

```

J'ai encore creusé mon investigation, et cette fois il apparaît, que la valeur de configuration de la variable `MONGO_OPLOG_URL` , est bien correcte. Si cette variable d'environnement est précisée pour la configuration du conteneur `rocketchat` , c'est parce que le conteneur `rocketchat` doit pouvoir communiquer avec l'instance MongoDB, par un canal particulier. On trouvera rapidement avec une recherche google, que ce canal est mis à disposition par MongoDB, et les applications peuvent l'utiliser pour écouter les événements sur l'historique mongodb (celui lié à la notion "mongo replay"). D'autre part, si on fait un :

```
docker logs mongo-init-replica
```

On constate que ce conteneur a échoué dans sa tâche, et ce parce que la connexion lui a été refusée. La tâche accomplie par ce conteneur est d'initialiser le "replicaSet" MongoDB, donc le "replicaSet" n'est pas initialisé correctement, et la connexion utilisée par `rocketchat` , configurée avec la variable `MONGO_OPLOG_URL` , échoue, parce qu'elle nécessite l'initialisation du replicaSet.

J'ai encore approfondi l'étude, et ai découvert que la valeur de `MONGO_OPLOG_URL` , doit correspondre au nom du replicaSet MongoDB créé par le conteneur `mongo-init-replica` , avec le client mongo, à l'aide de la commande :

```
mongo mongo/rocketchat --eval "rs.initiate({ _id: 'rs0', members: [ { _id: 0, host: 'mongo:27017' } ]})"
```

Cette requête TCP a pour effet de créer le replicaSet de nom `rs0` dans MongoDB, et c'est la raison pour laquelle elle est utilisée pour définir la commande d'exécution du conteneur `mongo-init-replica` (précisée dans un Dockerfile, avec la syntaxe `CMD ["/chemin/quel/conque/fichier-executable"]`) :

```
version: '3'

services:
  gitlab:
    # - [... etc... J'ai abrégé la configuration]
  mongo-init-replica:
    # image: mongo:3.2
    image: mongo:latest
    container_name: 'mongo-init-replica'
    command: 'mongo mongo/rocketchat --eval "rs.initiate({ _id: 'rs0', members: [ { _id: 0, host: 'mongo:27017' } ]})"'
    networks:
      - devops
    depends_on:
      - mongo
  rocketchat:
    image: rocketchat/rocket.chat:latest
    container_name: 'rocketchat'
    volumes:
      - ./rocketchat/uploads:/app/uploads
    environment:
      - PORT=3000
      - ROOT_URL=http://rocketchat.marguerite.io:3000
      - MONGO_URL=mongodb://mongo:27017/rocketchat
      - MONGO_OPLOG_URL=mongodb://mongo:27017/local?replicaSet=rs0
      - MAIL_URL="smtp://smtp.google.com"
    ports:
      - 3000:3000
    expose:
      - "3000"
    depends_on:
      - mongo
    networks:
      - devops
    restart: always
```

Ainsi, ci-dessus, `rs0`, le nom du replicaSet créé avec le conteneur `mongo-init-replica`, doit être le nom du replicaSet mentionné par `MONGO_OPLOG_URL`.

Une fois relancé l'ensemble du docker-compose, on constate que de nouveaux problèmes se manifestent :

- D'abord, le conteneur `mongo-init-replica` échoue toujours à sa première tentative de création du replicaSet, parce que le conteneur `mongodb` n'est pas prêt. On peut donc relancer l'exécution du conteneur `mongo-init-replica`, afin de créer le replicaSet : `docker start mongo-ibit-replica`
- Une fois cela fait, on peut alors suivre les logs du conteneur `rocketchat`, qui démarre effectivement, mais tout en émettant des avertissements qu'il faudra traiter comme des erreurs. Sortie standard de ces erreurs :

```
/app/bundle/programs/server/node_modules/fibers/future.js:313
    throw(ex);
    ^
MongoError: no primary found in replicaset or invalid replica set name
    at /app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mongodb-core/lib/topologies/
    at Server.<anonymous> (/app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mongodb-
    at Object.onceWrapper (events.js:315:30)
    at emitOne (events.js:116:13)
    at Server.emit (events.js:211:7)
```

```

    at /app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mongodb-core/lib/topologies/
    at /app/bundle/programs/server/npm/node_modules/meteor/npm-mongo/node_modules/mongodb-core/lib/connection/
    at _combinedTickCallback (internal/process/next_tick.js:131:7)
    at process._tickCallback (internal/process/next_tick.js:180:9)
Updating process.env.MAIL_URL
Starting Email Interceptor...
Warning: connect.session() MemoryStore is not
designed for a production environment, as it will leak
memory, and will not scale past a single process.
Setting default file store to GridFS
LocalStore: store created at
LocalStore: store created at
LocalStore: store created at
Fri, 17 Aug 2018 10:52:39 GMT connect deprecated multipart: use parser (multipart, busboy, formidable) npm mo
Fri, 17 Aug 2018 10:52:39 GMT connect deprecated limit: Restrict request size at location of read at npm/node_
Updating process.env.MAIL_URL
Using GridFS for custom sounds storage
Using GridFS for custom emoji storage
ufs: temp directory created at "/tmp/ufs"
→ System → startup
→ +-----+
→ |                                     |
→ |                               SERVER RUNNING                               |
→ |                                     |
→ |                                     |
→ |                                     |
→ | Rocket.Chat Version: 0.69.2                                             |
→ |       NodeJS Version: 8.11.3 - x64                                       |
→ |       Platform: linux                                                    |
→ |       Process Port: 3000                                                  |
→ |       Site URL: http://rocketchat.marguerite.io:3000                    |
→ |       ReplicaSet OpLog: Enabled                                          |
→ |       Commit Hash: 7df9818105                                           |
→ |       Commit Branch: HEAD                                               |
→ |                                     |
→ +-----+

```

- Ensuite, on voit que le conteneur `hubot`, a arrêté son exécution. Si on le re-démarre, avec un `docker start hubot`, et que l'on inspecte les logs de son exécution, on voit que le hubot démarre correctement, puis stoppe, en logguant l'erreur suivante:

```

[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO Starting Rocketchat adapter version 1.0.11...
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO Once connected to rooms I will respond to the name:
Rocket.Cat
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO I will also respond to my Rocket.Chat username as an alias:
rocket.cat
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO Connecting To: rocketchat:3000
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO Successfully connected!
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] INFO Logging In
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] ERROR Unable to Login:
{"isClientSafe":true,"error":403,"reason":"User has no password set","message":"User has no password set
[403]","errorType":"Meteor.Error"} Reason: User has no password set
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] ERROR If joining GENERAL please make sure its using all caps.
[Fri Aug 17 2018 10:55:37 GMT+0000 (UTC)] ERROR If using LDAP, turn off LDAP, and turn on general user
registration with email verification off.

```

- Il faut donc maintenant créer l'utilisateur que le HUBOT réclame !

## REPRISE

Je pense qu'il va falloir complètement changer l'image de hubot que j'utilise et sa configuration, pour utiliser une image conforme à : <https://github.com/RocketChat/Rocket.Chat.Ops/tree/develop/hubots/hubot-gitsy> qui est le hubot "latest" distribué par l'équipe RocketChat.

# ChatOps with Rocket.Chat

## Inspired in Gitlab: "From Idea to Production"

We've all got pretty amazed with the Gitlabs Idea to Production demonstration, and we felt inspired by doing the same, so we took the challenge and prepared this tutorial, with a different stack, to take your ideas to production, with Gitlab, Rocket.Chat and Hubot, all packed in a nice Docker containers stack.

Maybe we can call it...

##Chat, code and ship ideas to production

Let's take a look to this stack first, so you understand what we will be running in the following services containers:

- Gitlab CE (latest)
- Rocket.Chat (latest)
- MongoDB (3.2)
- Hubot-RocketChat (latest)
- Gitlab-Runner (latest with Dockerfile modifications)
- Nginx (latest as a reverse proxy)

## How does it work?

First we need to setup our environments, and if it's your first time running it, you should follow this instructions carefully, so we can get everything connected.

I you've already done these steps, just go inside your directory, in terminal, and type:

```
docker-compose up -d
```

To stop all services:

```
docker-compose stop
```

## GITLAB CE

In our docker-compose.yml, adjust the following variables:

```
GITLAB_OMNIBUS_CONFIG: |
  external_url 'http://git.dorgam.it/'
  gitlab_rails['gitlab_shell_ssh_port'] = 22
  gitlab_rails['lfs_enabled'] = true
  nginx['listen_port'] = 8081
```

You should set your external domain url, and leave the others.

INFO: because we can't have more than one container lintening in the same port number, our services will be all listening in different ports, and we will let a NGINX reverse proxy take care of the rest.

Then set your volumes to make sure your data will be persisted:

```
volumes:
  - ./gitlab/config:/etc/gitlab
  - ./gitlab/logs:/var/log/gitlab
```



```
- ./gitlab/data:/var/opt/gitlab
```

If your docker installation accepts your working directory as a volume, you can use the relative path.

And then we create a common shared network so the containers can communicate to each other. We will set a static ipv4 address, so we can use in others containers hosts files:

```
networks:
  devops:
    ipv4_address: 172.20.0.4
```

Now, you should just enter in terminal, and type inside this directory:

```
docker-compose up -d gitlab

docker logs -f chatops_gitlab_1
```

Now you will see the containers logs, it takes a while, but you can make sure that gitlabs is running accessing <http://git.dorgam.it:8081> in your browser.

## Gitlab Runner

You will need a registered runner to work with your pipeline, so we got the gitlab/gitlab-runner docker image and created a Dockerfile in `./gitlab/runner` to install a nginx http server inside it and register it in your gitlab. This is actually the most trick part, we know we shouldn't put two services inside the same container, but remember this is just an example, in real life, you will have your own server with your own runners.

So, you will find in `docker-compose.yml`:

```
runner:
  build: ./gitlab/runner/
  hostname: "runner"
  restart: "always"
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  links:
    - gitlab
  environment:
    - GITLAB_HOST=gitlab:8081
  ports:
    - "8000:8000"
  expose:
    - "8000"
  networks:
    devops:
      ipv4_address: 172.20.0.5
```

We've set a env variable `GITLAB_HOST=gitlab:8081`, using the service name as url address, that only will work inside docker network, where containers can find each other by the service name.

Let's go to the terminal and build our runner:

```
docker-compose build runner
```

If everything goes well, just put it up:

```
docker-compose up -d runner
```

Once the runner's container is up, you need to register it in your gitlab. Go at the runners page of your gitlab project and copy the token to the `-r` option, then we will put your external url domain inside the `/etc/hosts` file so the runner knows where your git repository is, and then register your runner:

```
docker exec -it chatops_runner_1 /bin/bash -c "echo '172.20.0.10    git.dorgam.it' >> /etc/hosts"

docker exec -it chatops_runner_1 /usr/bin/gitlab-runner register -u http://gitlab:8081/ci -r BwU14yBJTbnJjX8 -
```

TIP: You can also set a volume for the `/etc/hosts` file, so it will be persisted in your host machine.

EXPLAIN: `docker exec -it [name-of-container] [command]`

You can get the name of your container using `docker ps`

A message like this should appear:

```
Registering runner... succeeded                  runner=8e641b0b
Runner registered successfully. Feel free to start it, but if it's running already the config should be
automatically reloaded!
```

You can check if your runner is communicating by going to the Gitalabs Runner's page.

### Specific Runners

#### How to setup a specific Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the [GitLab Runner](#) section for information on how to install it).
2. Specify the following URL during the Runner setup: `http://git.dorgam.it/ci`
3. Use the following registration token during setup: `BwU14yBJTbnJjX8`
4. Start the Runner!

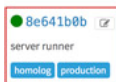
### Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

[Disable shared Runners](#) for this project

This GitLab server does not provide any shared Runners yet. Please use the specific Runners or ask your administrator to create one.

#### Runners activated for this project



[Remove Runner](#)

## MongoDB

First start mongodb container, then we need to start mongo-init-replica, so mongodb turns into a replica set primary server. In the terminal:

```
docker-compose up -d mongo
```

TIP: You can check the logs by using `docker logs -f [container_name]`, it's better than being attached to the container.

When it's done, run:

```
docker-compose up -d mongo-init-replica
```

This will initiate the replicaset configuration, and exit the container.

## Rocket.Chat

To put Rocket.Chat up you just need to set the environment variables `PORT` and `ROOT_URL` and run it:

```
rocketchat:
  image: rocketchat/rocket.chat:latest
  hostname: 'rocketchat'
  volumes:
```

```

- ./rocketchat/uploads:/app/uploads
environment:
- PORT=3000
- ROOT_URL=http://chat.dorgam.it:3000
- MONGO_URL=mongodb://mongo:27017/rocketchat
- MONGO_OPLOG_URL=mongodb://mongo:27017/local
- MAIL_URL="smtp://smtp.google.com"
links:
- mongo:mongo
- gitlab:gitlab
ports:
- 3000:3000
expose:
- "3000"
depends_on:
- mongo
networks:
devops:
  ipv4_address: 172.20.0.8

```

You can set MongoDB address, if you're using another service, and `MAIL_URL` in case you have a internal smtp server.

Run:

```
docker-compose up -d rocketchat
```

Now go register your Rocket.Chat Admin user, by <http://chat.dorgam.it:3000/>, and **create a user and a channel for the bot**.

## Hubot

Hubot is our framework for building bots, my favorite actually, here you can set a lot of params, just keep in mind that most of hubots scripts crashes if they don't find their environment variables, so be carefull when configuring these:

```

hubot:
  image: rocketchat/hubot-rocketchat:latest
  hostname: "hubot"
  environment:
    - ROCKETCHAT_URL=rocketchat:3000
    - ROCKETCHAT_ROOM=devops
    - ROCKETCHAT_USER=rocket.cat
    - ROCKETCHAT_PASSWORD=bot
    - ROCKETCHAT_AUTH=password
    - BOT_NAME=Rocket.Cat
    - LISTEN_ON_ALL_PUBLIC=true
    - EXTERNAL_SCRIPTS=hubot-help, hubot-seen, hubot-links, hubot-diagnostics, hubot-gitsy, hubot-gitlab-agile
    - GITLAB_URL=http://gitlab/api/v3/
    - GITLAB_API_KEY="cNhskKLDNs1KDKiS"
    - GITLAB_TOKEN=cNhskKLDNs1KDKiS
    - GITLAB_RECORD_LIMIT=100

  links:
    - rocketchat:rocketchat
    - gitlab:gitlab
  volumes:
    - ./hubot/scripts:/home/hubot/scripts
# this is used to expose the hubot port for notifications on the host on port 3001, e.g. for hubot-jenkins-n
ports:
- 3001:3001
networks:
devops:
  ipv4_address: 172.20.0.9

```

First you need to change `ROCKETCHAT_ROOM` , `ROCKETCHAT_USER` , that will be the username that you created in rocket.chat, and `ROCKETCHAT_PASSWORD` in plain text. As you can see I'm using Rocket.Cat, a natural rocket.chat bot, that comes with the installation, but you can create another one at your own image.

In the `./hubot/scripts` folder we can persist hubots scripts, there is a lot of them in github, you will be amazed.

Save your changes and run:

```
docker-compose up -d hubot
```

Check the logs to see if everything went well and then go to your channel and ask for a "yoda quote", just for fun.

## NGINX Reverse Proxy

So as we've said before, docker containers can't connect to the same port simultaneously, that's why each service has its own port, but that's not cool for the a user friendly experience, so what we've done here is putting a NGINX Reverse Proxy in front of every service, listening to port 80 (and 443 if you like) and proxy\_passing the connections to the services on their own port.

The NGINX configuration file is persisted in `./nginx/chatops.conf` , and you should change the domain names if you want.

```
upstream chat{
    ip_hash;
    server rocketchat:3000;
}

server {
    listen 80;
    server_name chat.dorgam.it;
    error_log /var/log/nginx/rocketchat.error.log;

    location / {
        proxy_pass http://chat;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $http_host;

        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forward-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forward-Proto http;
        proxy_set_header X-Nginx-Proxy true;

        proxy_redirect off;
    }
}

server {
    listen 80;
    server_name git.dorgam.it;
    error_log /var/log/nginx/gitlab.error.log;

    location / {
        proxy_pass http://gitlab:8081;
        proxy_redirect off;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Protocol $scheme;
        proxy_set_header X-Url-Scheme $scheme;
    }
}
```

```
server {
    listen 80;
    server_name www.dorgam.it;

    location / {
        proxy_pass http://runner:8000;
        proxy_redirect off;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Protocol $scheme;
        proxy_set_header X-Url-Scheme $scheme;
    }
}

server {
    listen 80;
    server_name hom.dorgam.it;

    location / {
        proxy_pass http://runner:8000;
        proxy_redirect off;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Protocol $scheme;
        proxy_set_header X-Url-Scheme $scheme;
    }
}
```

Just save your changes and:

```
docker-compose up -d nginx
```

When you change these confs, remember to reload then into NGINX:

```
docker exec -it chatops_nginx_1 /bin/bash -c "service nginx reload"
```

Hosts file is also persisted, so you can add or remove anything.

## Rocket.Chat Webhook Integration Script

Although Hubot is a very powerfull tool for bot scripting, you might wanna add some webhooks integration to our channels in Rocket.Chat.

For that, there is a pretty simple script that you can change as you like, to read the gitlabs webhooks and throw some messages inside your project channel.

Here is what you gonna do:

**ajout jbl :**

la méthode décrite cidessous implique un fichier de script à modifier. Je veux voir si je peux psécifier ce script au boot du conteneur, avec la variable d'environnement `EXTERNAL_SCRIPTS` ci dessous :

```
docker run -it -e ROCKETCHAT_URL=<your rocketchat instance>:<port> \
-e ROCKETCHAT_ROOM='' \
```

```
-e LISTEN_ON_ALL_PUBLIC=true \
-e ROCKETCHAT_USER=bot \
-e ROCKETCHAT_PASSWORD=bot \
-e ROCKETCHAT_AUTH=password \
-e BOT_NAME=bot \
-e EXTERNAL_SCRIPTS=hubot-pugme, hubot-help \
rocketchat/hubot-rocketchat
```

## Create Rocket.Chat Incoming WebHook

Access your rocket.chat from your browser (<http://chat.dorgam.it>) and go to the top menu (the little arrow besides your name) and click in Administration > Integrations > New Integration > Incoming WebHook.

Fill the form with the name of the script, the #channel (with sharp signal) where the messages will appear, and on until script. You activate script, and paste this script inside the script box:

```
/* eslint no-console:0, max-len:0 */
// see https://gitlab.com/help/web_hooks/web_hooks for full json posted by GitLab
const NOTIF_COLOR = '#6498CC';
const refParser = (ref) => ref.replace(/^refs\/(?:tags|heads)\/(.+)$/, '$1');
const displayName = (name) => name.toLowerCase().replace(/\s+/g, '.');
const atName = (user) => (user && user.name ? '@' + displayName(user.name) : '');
const makeAttachment = (author, text) => {
  return {
    author_name: author ? displayName(author.name) : '',
    author_icon: author ? author.avatar_url : '',
    text,
    color: NOTIF_COLOR
  };
};

const pushUniq = (array, val) => -array.indexOf(val) || array.push(val); // eslint-disable-line

class Script { // eslint-disable-line
  process_incoming_request({ request }) {
    try {
      let result = null;
      const channel = request.url.query.channel;
      switch (request.headers['x-gitlab-event']) {
        case 'Push Hook':
          result = this.pushEvent(request.content);
          break;
        case 'Merge Request Hook':
          result = this.mergeRequestEvent(request.content);
          break;
        case 'Note Hook':
          result = this.commentEvent(request.content);
          break;
        case 'Issue Hook':
          result = this.issueEvent(request.content);
          break;
        case 'Tag Push Hook':
          result = this.tagEvent(request.content);
          break;
        case 'Pipeline Hook':
          result = this.pipelineEvent(request.content);
          break;
      }
      if (result && result.content && channel) {
        result.content.channel = '#' + channel;
      }
      return result;
    } catch (e) {
      console.log('gitlabevent error', e);
      return {

```

```

        error: {
          success: false,
          message: e.message || e
        }
      };
    }
  }

  issueEvent(data) {
    return {
      content: {
        username: 'gitlab/' + data.project.name,
        icon_url: data.project.avatar_url || data.user.avatar_url || '',
        text: (data.assignee && data.assignee.name !== data.user.name) ? atName(data.a
        attachments: [
          makeAttachment(
            data.user,
            `${data.object_attributes.state} an issue _${data.object_attri
*Description:* ${data.object_attributes.description}.
See: ${data.object_attributes.url}`
          )
        ]
      }
    };
  }

  commentEvent(data) {
    const comment = data.object_attributes;
    const user = data.user;
    const at = [];
    let text;
    if (data.merge_request) {
      const mr = data.merge_request;
      const lastCommitAuthor = mr.last_commit && mr.last_commit.author;
      if (mr.assignee && mr.assignee.name !== user.name) {
        at.push(atName(mr.assignee));
      }
      if (lastCommitAuthor && lastCommitAuthor.name !== user.name) {
        pushUniq(at, atName(lastCommitAuthor));
      }
      text = `commented on MR [${mr.id} ${mr.title}](${comment.url})`;
    } else if (data.commit) {
      const commit = data.commit;
      const message = commit.message.replace(/\n[^\s\S]+/, '...').replace(/\n$/, '');
      if (commit.author && commit.author.name !== user.name) {
        at.push(atName(commit.author));
      }
      text = `commented on commit [${commit.id.slice(0, 8)} ${message}](${comment.url})`;
    } else if (data.issue) {
      const issue = data.issue;
      text = `commented on issue [${issue.id} ${issue.title}](${comment.url})`;
    } else if (data.snippet) {
      const snippet = data.snippet;
      text = `commented on code snippet [${snippet.id} ${snippet.title}](${comment.url})`;
    }
    return {
      content: {
        username: 'gitlab/' + data.project.name,
        icon_url: data.project.avatar_url || user.avatar_url || '',
        text: at.join(' '),
        attachments: [
          makeAttachment(user, `${text}\n${comment.note}`)
        ]
      }
    };
  }
}

```

```

mergeRequestEvent(data) {
  const user = data.user;
  const mr = data.object_attributes;
  const assignee = mr.assignee;
  let at = [];

  if (mr.action === 'open' && assignee) {
    at = '\n' + atName(assignee);
  } else if (mr.action === 'merge') {
    const lastCommitAuthor = mr.last_commit && mr.last_commit.author;
    if (assignee && assignee.name !== user.name) {
      at.push(atName(assignee));
    }
    if (lastCommitAuthor && lastCommitAuthor.name !== user.name) {
      pushUniq(at, atName(lastCommitAuthor));
    }
  }
  return {
    content: {
      username: `gitlab/${mr.target.name}`,
      icon_url: mr.target.avatar_url || mr.source.avatar_url || user.avatar_url || '',
      text: at.join(' '),
      attachments: [
        makeAttachment(user, `${mr.action} MR [${mr.iid}] ${mr.title} (${mr.url})`),
      ]
    }
  };
}

pushEvent(data) {
  const project = data.project;
  const user = {
    name: data.user_name,
    avatar_url: data.user_avatar
  };
  // branch removal
  if (data.checkout_sha === null && !data.commits.length) {
    return {
      content: {
        username: `gitlab/${project.name}`,
        icon_url: project.avatar_url || data.user_avatar || '',
        attachments: [
          makeAttachment(user, `removed branch ${refParser(data.ref)} from ${project.name}`),
        ]
      }
    };
  }
  // new branch
  if (data.before === 0) { // eslint-disable-line
    return {
      content: {
        username: `gitlab/${project.name}`,
        icon_url: project.avatar_url || data.user_avatar || '',
        attachments: [
          makeAttachment(user, `pushed new branch [${refParser(data.ref)}] to ${project.name}`),
        ]
      }
    };
  }
  return {
    content: {
      username: `gitlab/${project.name}`,
      icon_url: project.avatar_url || data.user_avatar || '',
      attachments: [
        makeAttachment(user, `pushed ${data.total_commits_count} commits to branch ${project.name}`),
        {
          text: data.commits.map((commit) => ` - ${new Date(commit.time).toLocaleDateString()} ${commit.message}`),
        }
      ]
    }
  };
}

```



```

                                color: NOTIF_COLOR
                            }
                        ]
                    }
                };
            }

            tagEvent(data) {
                const tag = refParser(data.ref);
                return {
                    content: {
                        username: `gitlab/${data.project.name}`,
                        icon_url: data.project.avatar_url || data.user_avatar || '',
                        text: '@all',
                        attachments: [
                            makeAttachment(
                                { name: data.user_name, avatar_url: data.user_avatar },
                                `push tag [${tag} ${data.checkout_sha.slice(0, 8)}](${data.pro
                            )
                        ]
                    }
                };
            }

            pipelineEvent(data) {
                const status = data.object_attributes.status;
                const link = data.project.web_url

                return {
                    content: {
                        username: `gitlab/${data.project.name}`,
                        icon_url: data.project.avatar_url || data.user.avatar_url || '',
                        text: 'Pipeline Active:',
                        attachments: [
                            makeAttachment(
                                { name: data.user.name, avatar_url: data.user.avatar_url },
                                `Runned a Pipeline with status: ${data.object_attributes.statu
                            )
                        ]
                    }
                };
            }
        }
    }
}

```

As you can see, in Rocket.Chat even the integrations are full open sourced, you can change the messages inside the script if you like, by just changing the content prepared inside the event functions.

Save your integration and test it with curl, using some gitlab webhook json, like this:

```
curl -X POST -H "x-gitlab-event: Pipeline Hook" --data-urlencode 'payload={  "object_kind": "pipeline",  "ob
```

TIP: With outgoing and incoming webhooks you can **connect Rocket.Chat to whatever you want**. You can be monitoring all of your services with Zabbix, Rancher, Puppet, or even CloudStack, Heroku, Azure, and basically everything that has a API to alert you.

That is what makes Rocket.Chat the most powerfull ChatOps opensource tool in the world!

## That's All Folks!

Now you have your own chatops environment set with Gitlab, Rocket.Chat and Hubot!

You can try to make your own CI Pipeline and get started with your interactions.

Please, feel free to contribute to this tutorial, and also take a look to our links below.

Thanks to all the guys that made it possible:

- <https://hub.docker.com/r/rocketchat/hubot-rocketchat>
- <https://github.com/github/hubot-scripts>
- <https://gitlab.com/gitlab-org/omnibus-gitlab/>
- [Rocket.Chat Team](#)