**Tài liệu triển khai GitLabs**

**Mục lục**

[Phần 1: Tổng quan về GitLab 3](#_Toc499888169)

[1) GitLab là gì? 3](#_Toc499888170)

[2) Đặc điểm của GitLab 3](#_Toc499888171)

[3) Sơ đồ kiến trúc: 4](#_Toc499888172)

[Phần 2: triển khai Gitlab 5](#_Toc499888173)

[1) Sơ đồ triển khai 5](#_Toc499888174)

[2) Yêu cầu cấu hình 5](#_Toc499888175)

[3) Hướng dẫn cài đặt Gitlabs 5](#_Toc499888176)

[4) Hướng dẫn cầu hình Keep-Alive 6](#_Toc499888177)

[5) Cấu hình Rsync 10](#_Toc499888178)

[6) Cấu hình ssh bỏ qua pass giữa active passive 12](#_Toc499888179)

[7) Cấu hình cơ chế đồng bộ, chịu lỗi 13](#_Toc499888180)

[8) Cấu hình Cron 21](#_Toc499888181)

[9) Qui trình khởi động GitLabs 21](#_Toc499888182)

[10) Kịch bản chịu lỗi: 22](#_Toc499888183)

# Tổng quan về GitLab

## GitLab là gì?

- GitLab khá nổi tiếng và là một mã nguồn mở của máy chủ Git được thực hiện bởi hơn 50.000 tổ chức. Trong vài năm gần đây Gitlad đã phát triển mạnh mẽ với sự hỗ trợ của cộng đồng mạng, hàng nghìn người sử dụng trên một máy chủ duy nhất hoặc một số máy chủ hoạt động tương tự. Nếu bạn cần thiết lập một máy chủ Git, thì GitLab cung cấp cho bạn một giải pháp hoàn hảo.

- Gitlab là một hệ thống self-hosted để quản lý mã nguồn của bạn. Bản đầu tiên được phát hành vào tháng 10/2011 và được cập nhật vào ngày 22 hàng tháng. Gitlab được phát hành theo tiêu chuẩn của MIT.

- Gitlab được thành lập bởi Dmitriy Zaporozhets năm 2013. Dự án bao gồm hai nhóm chính: một bên là “open source core team” và một bên là “GitLab B.V. team” (chi nhánh của công ty Gitlab).

- GitLab được sử dụng để lư trữ trên Github, nhưng với sự nỗ lực của Dmitriy Zaporozhets suốt một năm làm việc tại GitLab, kể từ tháng 1/2014 mã nguồn được lưu trữ trên sever chính của gitlab là gitlab.com. Các nhánh của GitLab được lưu trữ trên github, sẽ hoạt động như một source, nơi bạn có thể pull, push và merge các yêu cầu.

## Đặc điểm của GitLab

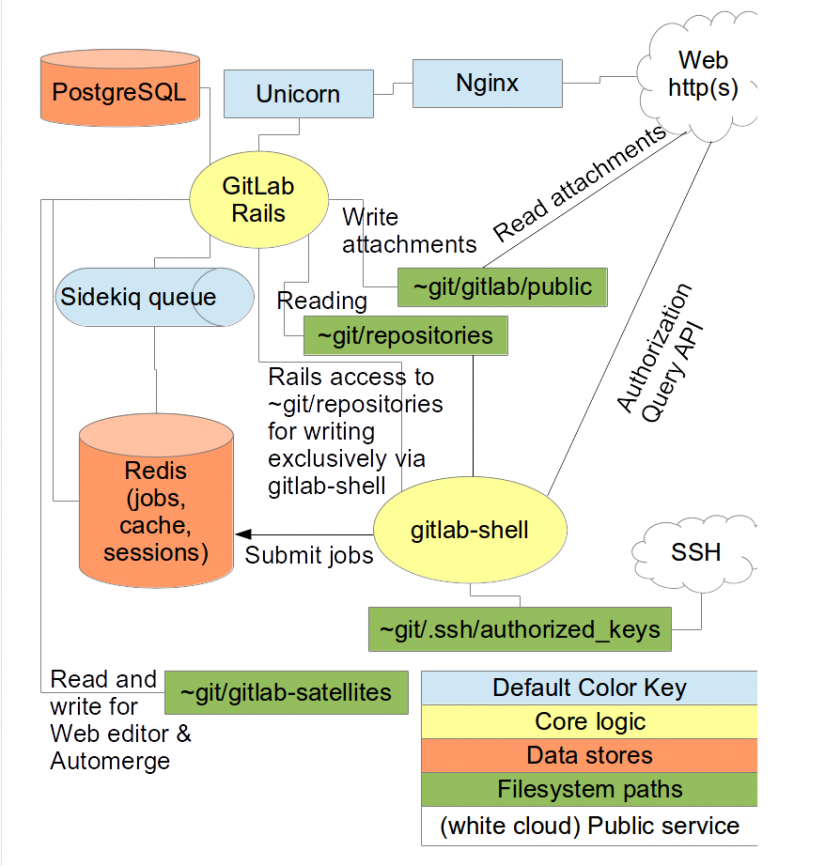
**GitLab hỗ trợ ba phiên bản:**

- Gitlab community editon (CE) - Gitlab phiên bản cộng đồng: là phiên bản mã nguồn mở. Được cung cấp qua Git từ kho lưu trữ chứa gitlab. Phiên bản mới nhất của gitlab được các nhà phát triển release tại các nhánh stable và nhánh master.

- Gitlad enterprise edition (EE) - Gitlab phiên bản doanh nghiệp: là phiên bản có sẵn không lâu sau khi phát hành bản CE, được cung cấp từ kho lưu trữ của gitlab.com. Một doanh nghiệp đăng lý GitLab được sự support của GitLad BV những khó khăn khi cài đặt.

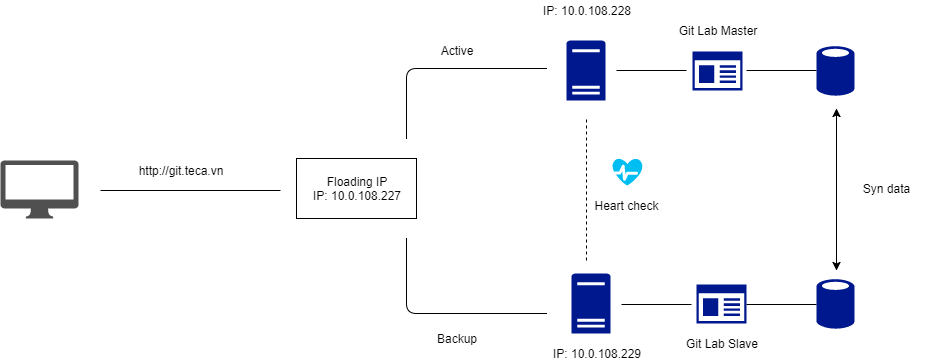
- Gitlab continuous intergration (CI): là một giải pháp tích hợp được thực hiện bở nhóm phát triển Gitlab

## Sơ đồ kiến trúc:



# triển khai Gitlab

## Sơ đồ triển khai



## Yêu cầu cấu hình

|  |  |  |
| --- | --- | --- |
| Server | Cấu hình | Mô tả |
| CentOS 7 | CPU: 4 core  RAM: 16GB  HDD: 500GB  Ip tĩnh: 10.200.200.228 | - Sử dụng làm Active server |
| CentOS 7 | CPU: 4 core  RAM: 16GB  HDD: 500GB  Ip tĩnh: 10.200.200.229 | - Sử dụng làm Passive server  - Làm server backup, chịu lỗi khi Active xảy ra vấn đề |

Cài đặt

## Hướng dẫn cài đặt Gitlabs

**Yêu cầu:**

- Cài đặt Gitlabs trên cả Active và Passive

**Bắt đầu:**

**Bước 1:**

- Trên CentOS, mở port HTTP và SSH vào system firewalld

$ sudo yum install -y curl policycoreutils-python openssh-server

$ sudo systemctl enable sshd

$ sudo systemctl start sshd

$ sudo firewall-cmd --permanent --add-service=http

$ sudo systemctl reload firewalld

**Bước 2:**

- Cài đặt Postfix để gửi thông báo qua Emails. Nếu sử dụng giải pháp khác cho vấn đề gửi Email, bỏ qua bước này.

$ sudo yum install postfix

$ sudo systemctl enable postfix

$ sudo systemctl start postfix

**Bước 3:**

- Add gói GitLab repo và cài đặt gói

$ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.rpm.sh | sudo bash

- Cấu hình URL cho thực thi GitLab

$ sudo EXTERNAL\_URL="**http://gitlab.example.com**" yum install -y gitlab-ee

**Bước 4: Truy cập giao diện chính**

- Truy cập thông qua URL (Khi setup domain)

**http://gitlab.example.com**

- Truy cập thông qua IP

**http://IP**

- Tại lần đầu tiên truy cập vào GitLab, ta sẽ phải cấu hình user “root” của hệ thống. Đây là user quản trị toàn bộ hệ thống GitLab.

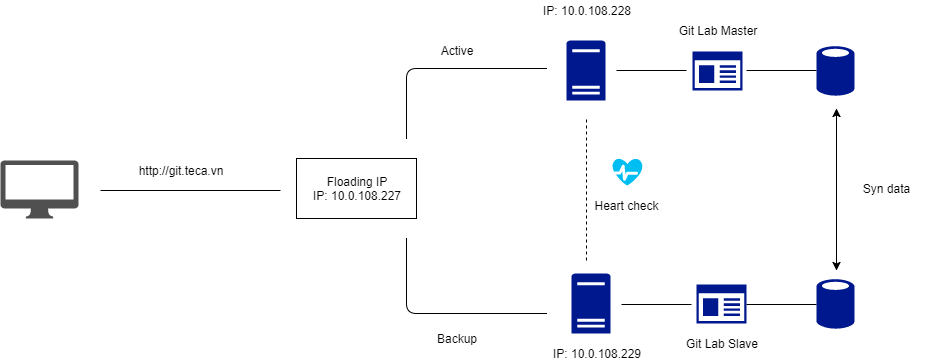
**Tùy chỉnh gitlabs:**

## Hướng dẫn cầu hình Keep-Alive

**Giới thiệu**

- Keepalived là gì : hiểu đơn giản keepalived là phần mền để tạo ra 1 VIP ( Virtual IP - IP ảo ). VIP này đại diện cho 2 hay nhiều IP thật trên các máy chủ hay các thiết bị. Khi người dùng truy cập vào website hay các ứng dụng sử dụng VIP này. Keepalived sẽ sử dụng các thuật toán để ánh xạ VIP vào những IP thật của các máy chủ, thiết bị

**Mô hình:**



**Tại Gitlabs active:**

**Bước 1: Cài đặt gói**

# yum -y install keepalived

**Bước 2: Cấu hình**

# vi /etc/keepalived/keepalived.conf

###

vrrp\_script check\_state {

script “/etc/gitlab/check\_state.sh” # verify the pid existance

interval 2 # check every 2 seconds

weight 4 # add 2 points of prio if OK

}

vrrp\_instance VI\_1 {

interface ens37 # interface to monitor

state MASTER

virtual\_router\_id 51 # Assign one ID for this route

priority 101 # 101 on master, 100 on backup

virtual\_ipaddress {

10.200.200.227 dev ens37 # the virtual IP

}

track\_script {

check\_state

}

}

**Bước 3: Script check trạng thái service Gitlabs**

# vi /etc/gitlab/check\_state.sh

###

#!/bin/bash

### Server Active check

WEBSERVER\_CHECK=127.0.0.1

STATUS\_CODE=`curl -sL -w "%{http\_code}\\n" "$WEBSERVER\_CHECK" -o /dev/null`

echo $STATUS\_CODE

if [ $STATUS\_CODE == 200 ]

then

echo "true"

exit 0

elif [ $STATUS\_CODE == 000 ]

then

echo "Code fail!!"

exit 1

else

echo "not known"

exit 1

fi

echo "fail all"

exit 1

**Bước 4: Khởi động service Keep alived**

# systemctl start keepalived

# systemctl enable keepalived

# systemctl status keepalived

**Tại Passive Gitlabs**

**Bước 1: Cài đặt gói**

# yum -y install keepalived

**Bước 2: Cấu hình**

# vi /etc/keepalived/keepalived.conf

###

vrrp\_script check\_state {

script “/etc/gitlab/check\_state.sh” # verify the pid existance

interval 2 # check every 2 seconds

weight 2 # add 2 points of prio if OK

}

vrrp\_instance VI\_1 {

interface ens37 # interface to monitor

state MASTER

virtual\_router\_id 51 # Assign one ID for this route

priority 101 # 101 on master, 100 on backup

virtual\_ipaddress {

**10.200.200.227** dev ens37 # the virtual IP

}

track\_script {

check\_state

}

}

**Bước 3: Script check trạng thái service Gitlabs**

# vi /etc/gitlab/check\_state.sh

###

#!/bin/bash

### Server Active check

WEBSERVER\_CHECK=127.0.0.1

STATUS\_CODE=`curl -sL -w "%{http\_code}\\n" "$WEBSERVER\_CHECK" -o /dev/null`

echo $STATUS\_CODE

if [ $STATUS\_CODE == 200 ]

then

echo "true"

exit 0

elif [ $STATUS\_CODE == 000 ]

then

echo "Code fail!!"

exit 1

else

echo "not known"

exit 1

fi

echo "fail all"

exit 1

**Bước 4: Khởi động service Keep alived**

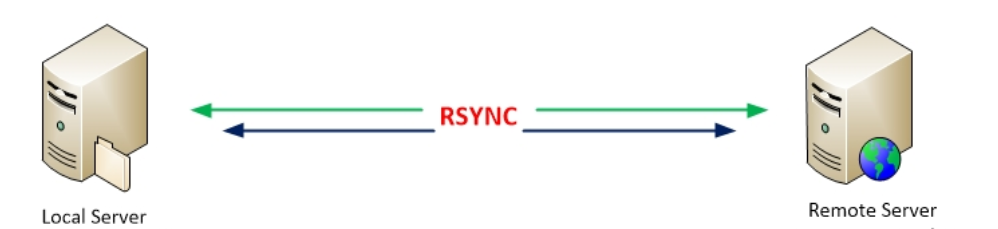
systemctl start keepalived

systemctl enable keepalived

systemctl status keepalived

## Cấu hình Rsync

**Giới thiệu**

- (Remote Sync) là một công cụ dùng để sao chép và đồng bộ file/thư mục được sử dụng rộng rãi trong môi trường Linux. Với sự trợ giúp của rsync, bạn có thể đồng bộ dữ liệu trên local hoặc giữa các server với nhau một cách dễ dàng.

**Lưu ý:**

- Câu lệnh đồng bộ dữ liệu nằm trong script đồng bộ

**Cấu hình**

**Tại Master:**

**Bước 1: Cài đặt gói**

# yum -y install rsync

**Bước 2: Cấu hình file main config**

# vi /etc/rsyncd.conf

###

[backup]

path = /var/opt/gitlab

hosts allow = 10.200.200.229

hosts deny = \*

list = true

uid = root

gid = root

read only = false

**Bước 3: Khởi động**

# systemctl restart rsyncd

# systemctl enable rsyncd

**Tại slave**

**Bước 1: Cài đặt gói**

# yum -y install rsync

**Bước 2: Cấu hình file main config**

[backup]

path = /var/opt/gitlab

hosts allow = 10.200.200.228

hosts deny = \*

list = true

uid = root

gid = root

read only = false

**Bước 3: Khởi động**

# systemctl restart rsyncd

# systemctl enable rsyncd

## Cấu hình ssh bỏ qua pass giữa active passive

**Giới thiệu**

- Hỗ trợ cho việc đồng bộ giữa liệu quan script

**Bắt đầu:**

**Tại cả Master và Agent**

**Bước 1: Tạo key**

# ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/home/jsmith/.ssh/id\_rsa):[Enter key]

Enter passphrase (empty for no passphrase): [Press enter key]

Enter same passphrase again: [Pess enter key]

Your identification has been saved in /home/xxx/.ssh/id\_rsa.

Your public key has been saved in /home/xxx/.ssh/id\_rsa.pub.

The key fingerprint is:

33:b3:fe:af:95:95:18:11:31:d5:de:96:2f:f2:35:f9 xxx@local-host

**Bước 2: Truy cập thư mục chứa key**

# ssh-copy-id -i ~/.ssh/id\_rsa.pub remote-host

xxx@remote-host's password:

Now try logging into the machine, with "ssh 'remote-host'", and check in:

.ssh/authorized\_keys

to make sure we haven't added extra keys that you weren't expecting.

**Bước 3: Test kết nối**

# ssh remote-host

## Cấu hình cơ chế đồng bộ, chịu lỗi

**Giới thiệu**

- Script đồng bộ cho phép dữ liệu tự động đồng bộ giữa Master – Slave và cơ chế chịu lỗi khi gặp sự cố

**Bắt đầu**

**Tại Master**

**Bước 1: Sử dụng script check-rsync.sh**

# vi /etc/gitlab/check-rsync.sh

###

#!/bin/bash

### VAR

WEBSERVER\_CHECK=10.200.108.229 #slave

STATUS\_CODE=`curl -sL -w "%{http\_code}\\n" "$WEBSERVER\_CHECK" -o /dev/null`

RSYNC\_CHECK=0 # 0 = not sync; 1 = sync; 3 = fail check web service; 4 = fail check service local

TIME=`date`

LOG=/var/log/script-gitlabs/master.log

MESSAGE\_LOG=""

STATE\_RSYNC=`ps -ef | grep -v grep | grep 'rsync ' | wc -l`

STATE\_SLAVE=0 # 0 = fail; 1 = true; 2 = error

STATE\_SERVICE=0 # 0 = fail; 1 = true;

STATE\_WEB\_SERVICE=0 # 0 = fail; 1 = true; 2 = error

CHECK\_PROCESS\_RSYNC=`ps -ef | grep -v grep | grep 'rsync -azvr --delete /var/opt/gitlab/' | wc -l`

### Function LOG

logger\_func () {

if [ ! -d /var/log/script-gitlabs/ ]; then

mkdir -p /var/log/script-gitlabs/

fi

echo "$TIME: $1" >> $LOG

}

### Start Check

Message="Start process sync data active-passive !!!!"

echo $Message

logger\_func "$Message"

### check service rsyncd

if (( $STATE\_RSYNC == "0" ))

then

echo "rsyncd not on"

fi

### Check web service slave

if [ $STATUS\_CODE != 000 ]

then

if [ $STATUS\_CODE == 200 ]

then

#echo "Check service Passive Gitlab On!!"

STATE\_SLAVE=1

else #Error service on gitlab Passive

#echo "Check status code Fail! - status code = $STATUS\_CODE "

STATE\_SLAVE=2

fi

else

# echo "Service gitlab on remote not on"

STATE\_SLAVE=0

fi

### Check web service master

LOCAL\_STATUS\_CODE=`curl -sL -w "%{http\_code}\\n" "127.0.0.1" -o /dev/null`

if [ $LOCAL\_STATUS\_CODE != 000 ]

then

if [ $LOCAL\_STATUS\_CODE == 200 ]

then

#echo "Check service Active Gitlab On!!"

STATE\_WEB\_SERVICE=1

else #Error service on gitlab Passive

#echo "Check status code Fail! - status code = $STATUS\_CODE "

STATE\_WEB\_SERVICE=2

fi

else

# echo "Service gitlab active is off"

STATE\_WEB\_SERVICE=0

fi

### Check service gitlab on server

gitlab-ctl status | grep ^run

if [ $? == 0 ]

then

# echo "Service gitlab On"

STATE\_SERVICE=1

fi

### Condition RSYNC\_CHECK

if [ $STATE\_SLAVE == 0 ] && [ $STATE\_SERVICE == 1 ] && [ $STATE\_WEB\_SERVICE == 1 ] && [ $CHECK\_PROCESS\_RSYNC == 0 ]

then

RSYNC\_CHECK=1

else

case "$STATE\_SLAVE" in

0) Message="State gitlabs slave is off"

echo $Message

logger\_func "$Message"

;;

1) Message="State gitlabs slave is on - Dong bo nguoc Passive - Master"

echo $Message

logger\_func "$Message"

;;

2) Message="State gitlabs slave is error - http\_code = $STATUS\_CODE"

echo $Message

logger\_func "$Message"

;;

\*) Message="Loi kiem tra, check state service gitlab slave"

echo $Message

logger\_func "$Message"

;;

esac

case "$STATE\_SERVICE" in

0) Message="Service gitlabs is off - Kiem tra dong bo du lieu Master - Slave"

echo $Message

logger\_func "$Message"

;;

1) Message="Service gitlabs is on"

echo $Message

logger\_func "$Message"

case "$STATE\_WEB\_SERVICE" in

0) Message="Service Gitlab web is off"

echo $Message

logger\_func "$Message"

;;

1) Message="Service Gitlab web is on"

echo $Message

logger\_func "$Message"

;;

2) Message="Service Gitlab web - http\_code = $LOCAL\_STATUS\_CODE"

echo $Message

logger\_func "$Message"

;;

\*) Message="Loi kiem tra, check state service Gitlab web"

echo $Message

logger\_func "$Message"

;;

esac

case "$CHECK\_PROCESS\_RSYNC" in

0) Message="Process rsyncd is not run"

echo $Message

logger\_func "$Message"

;;

1) Message="Process rsyncd is running - Cant not rsyncd"

echo $Message

logger\_func "$Message"

;;

\*) Message="Loi kiem tra, check process rsyncd"

echo $Message

logger\_func "$Message"

;;

esac

;;

\*) Message="Loi kiem tra, check state service gitlab local"

echo $Message

logger\_func "$Message"

;;

esac

fi

## Syn data

if [ $RSYNC\_CHECK == 1 ]

then

rsync -azvr --delete /var/opt/gitlab/ root@$WEBSERVER\_CHECK:/var/opt/gitlab/

Message="Sync to Passive GitLabs - Check state true"

echo $Message

logger\_func "$Message"

#echo "$TIME: Sync to Passive GitLabs - Check state true" >> $LOG || echo "Something wrong, check state rsyncd" >> $LOG

else

Message="Can not sync data active-passive - Check log "

echo $Message

logger\_func "$Message"

fi

Message="End process sync data active-passive !!!!"

echo $Message

logger\_func "$Message"

**Tại Slave:**

Bước 1: Sử dụng script check\_fail\_over.sh

# vi /etc/gitlab/check\_fail\_over.sh

####

#!/bin/bash

### Server Active check

WEBSERVER\_CHECK=10.200.200.228 #master #truong xet 1 file main config

STATUS\_CODE=`curl -sL -w "%{http\_code}\\n" "$WEBSERVER\_CHECK" -o /dev/null`

TIME=`date`

LOG=/var/log/script-gitlabs/slave.log

MESSAGE\_LOG=""

STATE\_RSYNC=`ps -ef | grep -v grep | grep 'rsync ' | wc -l`

STATE\_MASTER=0 # 0 = fail; 1 = true; 2 = error

STATE\_SERVICE=0 # 0 = fail; 1 = true;

### Function LOG

logger\_func () {

if [ ! -d /var/log/script-gitlabs/ ]; then

mkdir -p /var/log/script-gitlabs/

fi

echo "$TIME: $1" >> $LOG

}

### Start Check

Message="Start process check fail over active-passive !!!!"

echo $Message

logger\_func "$Message"

### Check web service

if (( $STATE\_RSYNC == "0" ))

then

echo "rsyncd not on"

fi

### Check web service

if [ $STATUS\_CODE != 000 ]

then

if [ $STATUS\_CODE == 200 ]

then

#echo "Check service Active Gitlab On!!"

STATE\_MASTER=1

else #Error service on gitlab Active

#echo "Check status code Fail! - status code = $STATUS\_CODE "

STATE\_MASTER=2

fi

else

# echo "Service gitlab on remote not on"

STATE\_MASTER=0

fi

### Check service

gitlab-ctl status | grep ^run

if [ $? == 0 ]

then

STATE\_SERVICE=1

fi

## Fail over.

if [ $STATE\_MASTER == 0 ] || [ $STATE\_MASTER == 2 ] && [ $STATE\_SERVICE == 0 ]

then

gitlab-ctl start

Message="Fail Over true - Start gitlab service on local"

echo $Message

logger\_func "$Message"

else

case "$STATE\_MASTER" in

0) Message="State gitlabs active is off"

echo $Message

logger\_func "$Message"

;;

1) Message="State gitlabs active is on"

echo $Message

logger\_func "$Message"

;;

2) Message="State gitlabs active is error - http\_code = $STATUS\_CODE"

echo $Message

logger\_func "$Message"

;;

\*) Message="Loi kiem tra, check state service gitlab active"

echo $Message

logger\_func "$Message"

;;

esac

case "$STATE\_SERVICE" in

0) Message="Service gitlabs is off"

echo $Message

logger\_func "$Message"

;;

1) Message="Service gitlabs is on - Kiem tra dong bo du lieu Master - Slave"

echo $Message

logger\_func "$Message"

;;

\*) Message="Loi kiem tra, check state service gitlab local"

echo $Message

logger\_func "$Message"

;;

esac

fi

Message="End process check failover active-passive !!!!!"

echo $Message

logger\_func "$Message"

## Cấu hình Cron

**Giới thiệu**

- Cron là một tiện ích cho phép thực hiện các tác vụ một cách tự động theo định kỳ, ở chế độ nền của hệ thống. Crontab (CRON TABLE) là một file chứa đựng bảng biểu (schedule) của các entries được chạy.

**Một số lệnh thường dùng:**

crontab -e: tạo hoặc chỉnh sửa file crontab

crontab -l: hiển thị file crontab

crontab -r: xóa file crontab

**Cấu hình**

**Tại Master**

**Bước 1:**

# crontab -e

\*/5 \* \* \* \* bash /etc/gitlab/check\_fail\_over.sh

@reboot gitlab-ctl stop

**Bước 2:**

# systemctl status crond

**Tại Slave**

**Bước 1:**

# crontab -e

\*/5 \* \* \* \* bash /etc/gitlab/check-rsync.sh

@reboot gitlab-ctl stop

**Bước 2:**

# systemctl status crond

## Qui trình khởi động GitLabs

**Yêu cầu:**

- Khởi động đúng theo mô hình, active – service gitlab on, passive – service gitlab off

**Qui trình:**

Tại server Gitlabs:

- Kiểm tra trạng thái:

# gitlab-ctl status

- Nếu service off, để khởi động sử dụng cmd:

# gitlab-ctl start

- Để stop service

# gitlab-ctl stop

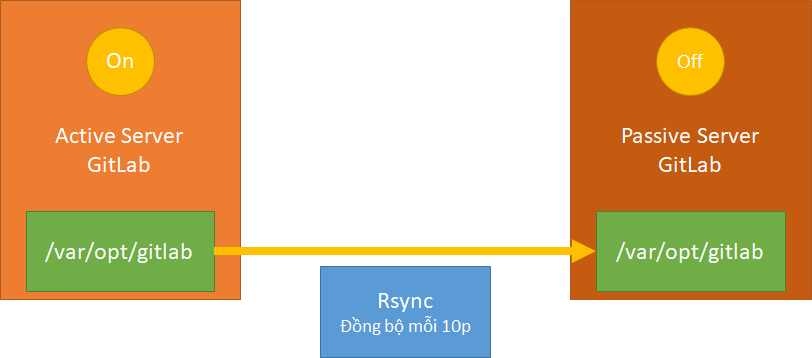
## Kịch bản chịu lỗi:

**Kịch bản 1:**

- Gitlab Active server khởi động và chạy, Passive server sẽ tắt service GitLabs

- Tất cả luồng xử lý sẽ tới GitLab Active Server để xử lý

- Đồng bộ dữ liệu thư mục “/var/opt/gitlab” từ Active đến Passive mỗi 10p

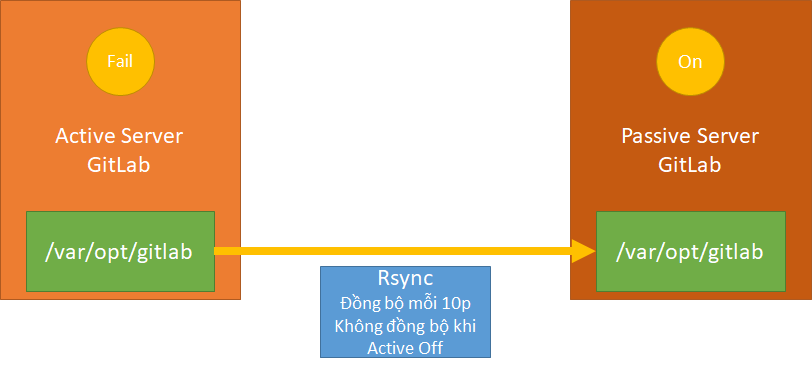


**Kịch bản 2:**

- Active server lỗi, bật Service GitLab Service tại Active sẽ ở trạng thái treo hoặc tắt

- Passive sẽ check trạng thái Web Active Server và tự bật theo chu kỳ check (chu kỳ check 5p/lần). Khi cơ chế chịu lỗi tại Passive được chạy, luồng dự liệu sẽ tới Passive thay vì Active. Cơ chề đồng bộ giữa Active – Passive sẽ không được chạy

- Cần đồng bộ ngược từ Passive tới Active = script “sync\_passive\_active.sh” để tránh mất mát dữ liệu.



**Kịch bản 3:**

- Active server sau khi sửa lỗi thành công, yêu cầu trở lại mô hình chạy bình thương

- Passive server đông bộ ngược lại cho Active server bằng script “sync\_passive\_active.sh” và phải thỏa mãn yêu cầu:

+ Tắt service trên Active server

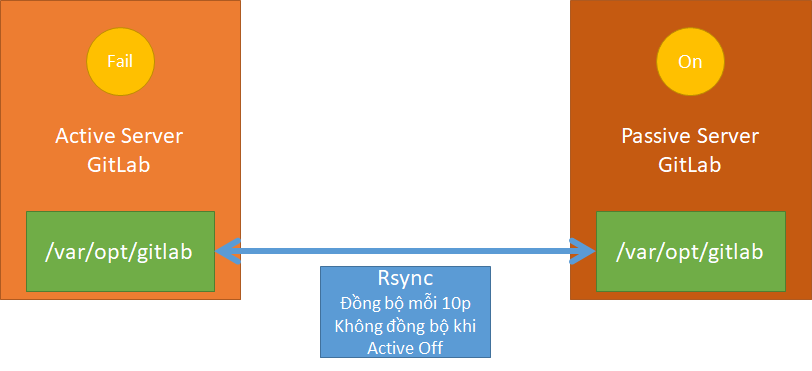
+ Service Gitlabs trên Active phải đang được chạy

- Sau khi đồng bộ dữ liệu bằng script:

+ Tắt service Gitlab tại Passive

+ Bật service Gitlab tại Active

- Mô hình trở lại bình thường



Note:

- Kiểm tra log của các Server tại:

/var/log/script-gitlabs/master.log

/var/log/script-gitlabs/slave.log

/var/log/script-gitlabs/sync-passive-active.log

/var/log/script-gitlabs/rsync.log

e