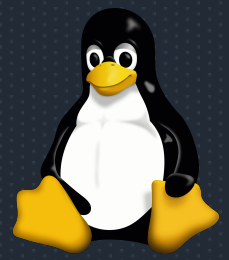


NSY103

- Linux : principes et programmation -

BOUDAOUD Samir

le cnam

2020/2021 Premier semestre

Mr ROUSSEL Olivier

Le projet

Gestion d'une file d'attente

Le script permet de gérer une liste de tâches en attente, celles-ci sont enregistrées dans une base de données, certaines sont programmées pour être exécutées à un certaine moment, d'autres directement. Une fois exécutée, elle retournera un résultat, nous mettrons à jour notre base de données en indiquant celui-ci afin de garder une trace.

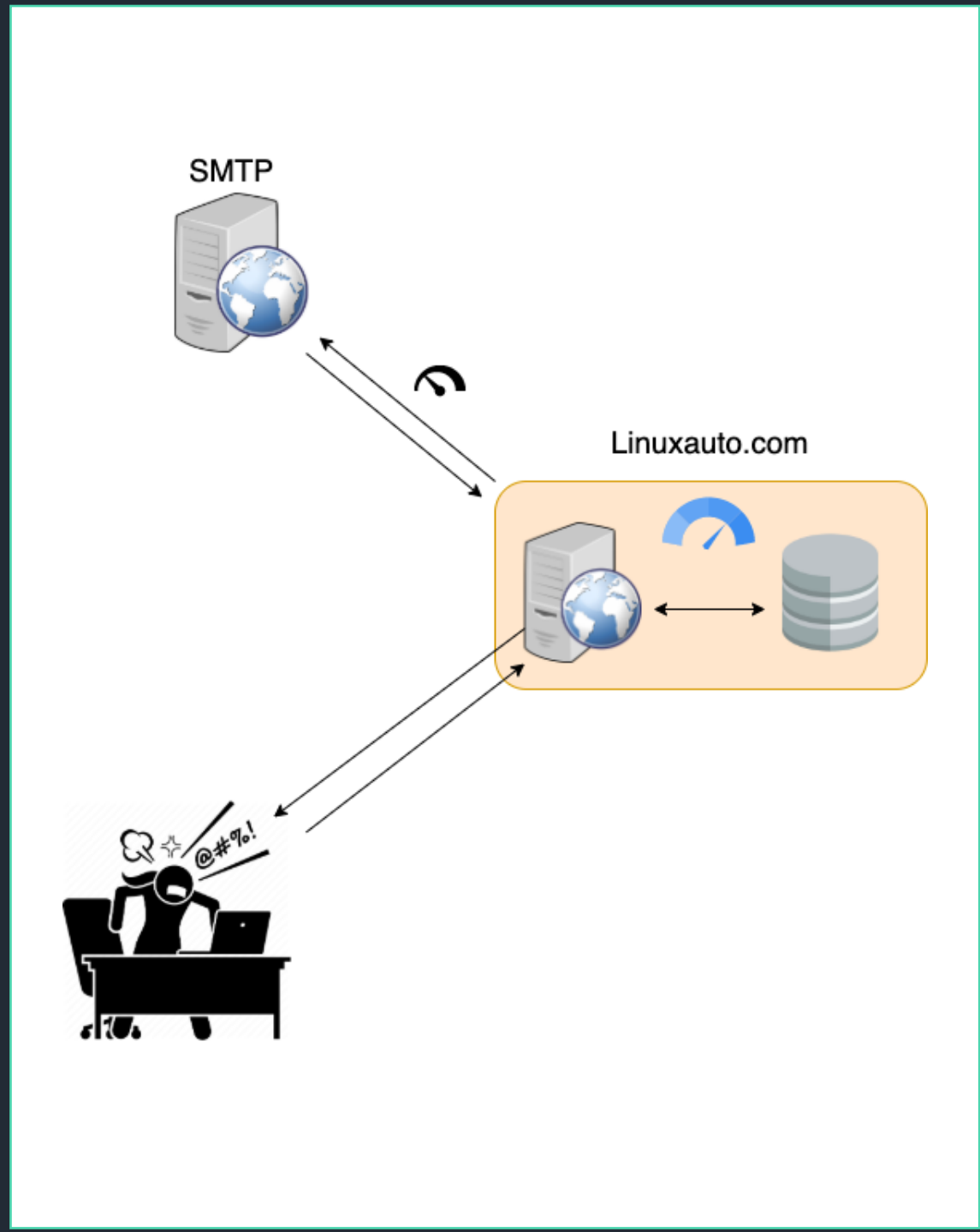
Notre script répond à un problème que nous allons tenter de résoudre: réduire au maximum le temps de traitement d'un script php.

Présentation du problème

L'experience utilisateur avant tout

La société LINUXAUTO.COM commercialise des pièces auto en ligne, lorsqu'un nouveau client effectue une commande, de multiples tâches sont effectuées au même moment (creation compte client, edition des commandes, envoi des notifications etc.). L'entreprise s'est rendu compte que le temps moyen de traitement à cet instant était très long, ce qui n'était pas envisageable, elle charge alors ses développeurs de trouver une solution.

L'équipe de développeurs effectue des tests et se rend compte que l'envoi d'emails depuis un serveur SMTP externe ralentit drastiquement le traitement, l'équipe présente alors sa solution.



La solution

My name is Bash

Après avoir réalisé des tests, il s'avère qu'un envoi d'email prend 20x plus de temps que l'enregistrement d'une entrée en base de données.

L'équipe décide alors d'enregistrer les emails en base afin de les envoyer de manière asynchrone et de répondre plus rapidement au client.

L'important pour le client à cet instant est d'avoir un message de confirmation, l'email viendra ensuite.

Elle fait des tests de benchmarking afin de présenter les résultats au PDG

SMTP vs Database

Envoi d'emails à travers SMTP

Nombre d'emails à envoyer

2 Envoyer

2 messages envoyés avec succes
Temps de traitement: 1.0421829223633 ms

Envoi d'emails à travers SMTP

Nombre d'emails à envoyer

10 Envoyer

10 messages envoyés avec succes
Temps de traitement: 1.9559099674225 ms

Envoi d'emails à travers SMTP

Nombre d'emails à envoyer

20 Envoyer

20 messages envoyés avec succes
Temps de traitement: 4.6515591144562 ms

Enregistrement en BDD

Nombre d'entrées à sauvegarder en BDD

2 Envoyer

2 entrées sauvegardées avec succes
Temps de traitement: 0.011853933334351 ms

Enregistrement en BDD

Nombre d'entrées à sauvegarder en BDD

50 Envoyer

50 entrées sauvegardées avec succes
Temps de traitement: 0.021512031555176 ms

Enregistrement en BDD

Nombre d'entrées à sauvegarder en BDD

200 Envoyer

200 entrées sauvegardées avec succes
Temps de traitement: 0.11746692657471 ms

SMTP vs Database

Les développeurs présentent les résultats au PDG ainsi que les avantages:

Temps de réponse quasi-instantané

Experience utilisateur plus agréable

Pouvoir sur la charge du serveur

Le PDG valide

Détail du script

BASH & Crontask

Les tâches sont divisées en 2:

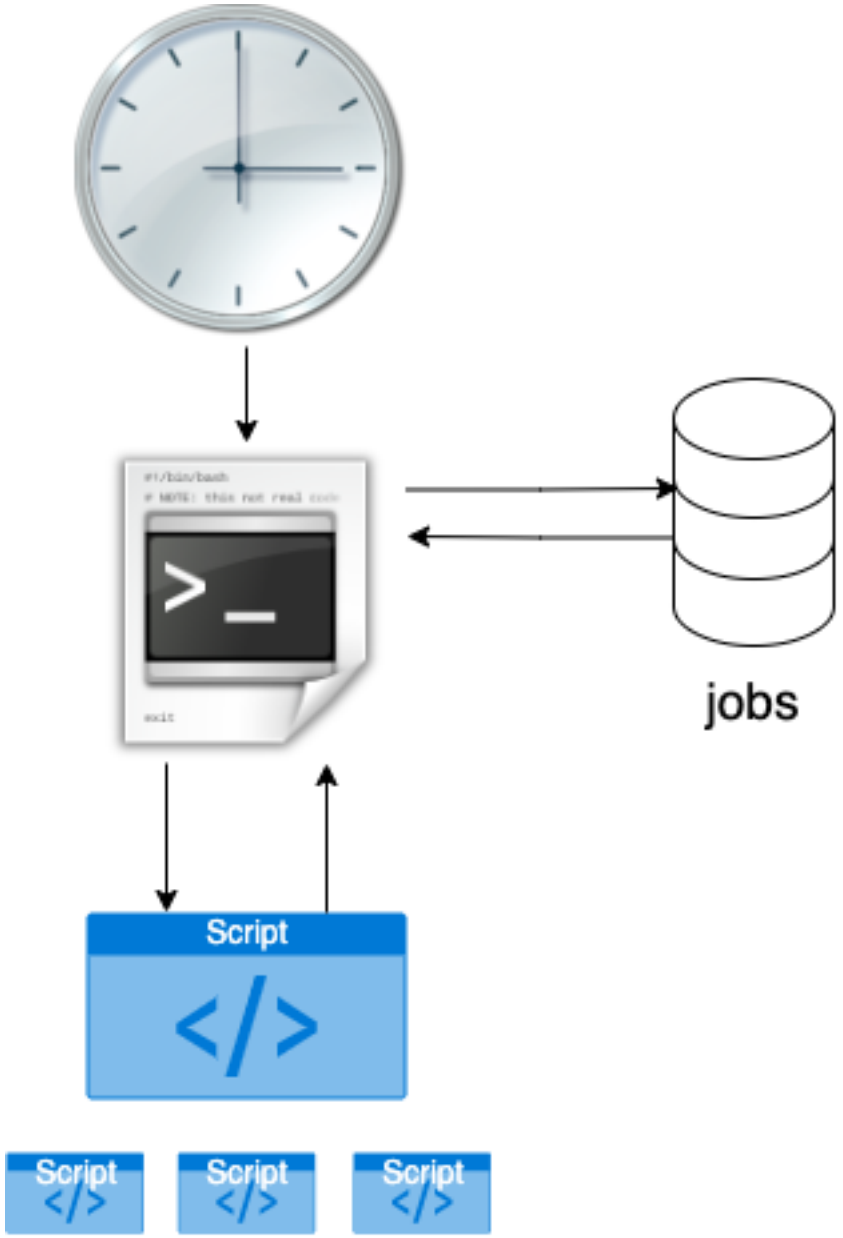
1. « SCHEDULED » : tâches planifiées à une date précise.
2. « TODO » : tâches à lancer dès que possible.

Le plan est simple: enregistrer les taches dans une table « jobs » et créer deux tâches CRON qui ont chacune un but bien précis:

1. Lancer les 10 tâches suivantes « a faire tout de suite »
2. Débloquer les tâches « planifiées » lorsque le temps de leur execution est venu

Jobs

type	date_completion	state	job
SCHEDULED	2021-02-16 15:34:54	pending	jobs/job2.php
TODO	2021-02-10 13:16:46	pending	jobs/job1.php
TODO	2021-02-03 08:56:20	pending	jobs/job3.php
TODO	2021-01-11 21:13:24	pending	jobs/job2.php
SCHEDULED	2020-12-25 08:44:53	pending	jobs/job2.php
SCHEDULED	2021-01-13 12:29:23	pending	jobs/job3.php
SCHEDULED	2021-02-14 04:48:16	pending	jobs/job1.php
TODO	2021-01-20 20:57:59	pending	jobs/job2.php
SCHEDULED	2021-01-14 00:03:52	pending	jobs/job3.php
SCHEDULED	2020-12-28 11:38:23	pending	jobs/job1.php
TODO	2021-01-26 22:52:34	pending	jobs/job1.php
SCHEDULED	2021-01-30 15:36:55	pending	jobs/job1.php
SCHEDULED	2021-02-09 19:37:26	pending	jobs/job1.php
SCHEDULED	2020-12-24 22:04:28	pending	jobs/job1.php
SCHEDULED	2021-02-11 21:24:00	pending	jobs/job3.php
TODO	2021-01-29 04:47:03	pending	jobs/job1.php



CRON 1 - TODO

Lancer les tâches « dès que possible »

Le premier script sera lancé toutes les minutes, il se chargera d'aller vérifier en base de donnée si des tâches « TODO » existent.
Ces tâches sont en effet à executer dès que possible.

Il récupère 10 tâches à la fois pour ne pas surcharger le serveur, chacune de ces tâches comporte un script à executer, ce script renvoi un message d'erreur ou de succès.

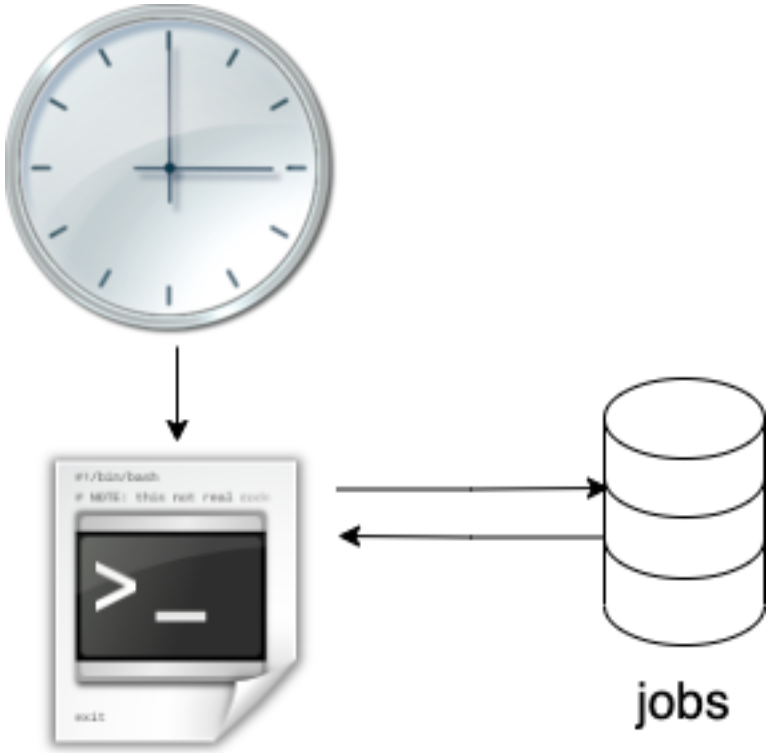
Il mettra enfin à jour chaque tâche en base afin d'indiquer son état.

Le retour du script est envoyé dans fichier .success.log pour les messages de sortie standards et .error.log pour les messages d'erreurs.

CRON 2 - SCHEDULED

Lancer les tâches « planifiées »

Le second script sera lancé à toutes les 5 minutes, il se chargera d'aller vérifier en base de données si des tâches « SCHEDULED » existent.
Si la tâche est entrée dans son temps, il la met à jour en modifiant son type vers « TODO », et c'est le CRON1 qui se chargera se son execution.




```
if [ "$type" == "SCHEDULED" ]
then
  # c'est un sheduled, on vérifie les dates
  NOW=$(date +%Y-%m-%d\ %H:%M:%S)

  if [ "$date_completion" \< "$NOW" ]; then
      # la date est passée, on le debloque afin qu'il soit traité
      sqlite3 $HERE/database.db "UPDATE jobs SET type='TODO' WHERE rowid = '$id'"
      ((nb_changes++))
  fi
fi
```

Mise en place des tâches CRON

Les 5 étoiles représentent respectivement

m : Minute h : Hour dom : Day Of Month mon : Month dow : Day Of Week (0 = dimanche)

Le premier script est sera lancé toutes les minutes, et le second toutes les 5 minutes.

crontab -l : liste toutes les taches cron de l'utilisateur en cours
crontab -e : ouvre la table cron afin d'éditer les tâches à executer

crontab

```
NSY103=/Users/samirboudaoud/Desktop/NSY103-Linux/projet_cron_jobs

* * * * * sh $NSY103/runJobs.sh >> $NSY103/runJobs.success.txt 2>> $NSY103/runJobs.error.txt
*/5 * * * * sh $NSY103/checkSchedules.sh >> $NSY103/checkSchedules.success.txt 2>> $NSY103/checkSchedules.error.txt
```

runJobs.sh

Lance les tâches à faire « dès que possible »

Le script s'occupe premièrement d'aller récupérer les tâches en base de données grâce à sqlite.

Ceci nous retourne une liste de résultats séparés par un « pipe », nous avons choisi de séparer chaque colonne par un point-virgule.

Pour parcourir ces résultats, il faut qu'ils soient « itérables », les tableaux nous aideront.

En utilisant la commande « read » il est possible de lire chacune des entrées et de les placer dans un tableau.

```
#!/bin/bash
#!/usr/bin/php

HERE="/Users/samirboudaoud/Desktop/NSY103-Linux/projet_cron_jobs/"
JOBS=$(sqlite3 -separator ";" -newline "|" $HERE/database.db
"SELECT rowid, type, date_completion, state, job FROM jobs WHERE
type='TODO' LIMIT 10")
```

Résultat

```
5;TODO;2020-12-25 08:44:53;pending;jobs/job2.php|6;TODO;2021-01-13
12:29:23;pending;jobs/job3.php|9;TODO;2021-01-14
00:03:52;pending;jobs/job3.php|10;TODO;2020-12-28
11:38:23;pending;jobs/job1.php|14;TODO;2020-12-24
22:04:28;pending;jobs/job1.php|22;TODO;2020-12-22
17:08:24;pending;jobs/job1.php|23;TODO;2021-01-10
23:42:04;pending;jobs/job3.php|25;TODO;2021-01-10
04:10:21;pending;jobs/job1.php|26;TODO;2021-01-14
07:27:55;pending;jobs/job1.php|31;TODO;2021-01-10
00:44:44;pending;jobs/job3.php|
```

runJobs.sh

Lance les tâches à faire « dès que possible »

```
# Separe chaque ligne et la place dans un tableau sous forme d'un string
IFS='|' read -r -a jobs_array <<< "$JOBS"

# [@] all values
for row in "${jobs_array[@]}"
do
  #...
done
```

1. On définit le séparateur pour la commande « read » grâce à IFS « Internal Field Separator »
2. On lit les données grâce à « read »
   -r: pour ne pas considérer le caractère d'échappement
   -a: pour placer les résultats de manière séquentielle dans un tableau
3. On peut itérer sur chacun de nos éléments

Extrait de « man read »

```
-a aname
          The words are assigned to sequential indices of the array variable aname, starting at 0
             aname is unset before any new values are assigned.  Other name arguments are ignored.

-r        Backslash  does not act as an escape character.  The backslash is considered to be part
             of the line. In particular, a backslash-newline pair can not be used as a line continuation.
```

runJobs.sh

Lance les tâches à faire « dès que possible »

Pour chaque entrée:

Distinguer chaque champ et le placer dans sa variable
Si la tâche est à executer tout de suite, la lancer

```bash
for row in "${jobs_array[@]}"
do
  id=$(echo "$row" | awk -F ";" '{print $1}');
  type=$(echo "$row" | awk -F ";" '{print $2}');
  job=$(echo "$row" | awk -F ";" '{print $5}');

  if [ "$type" == "TODO" ]
  then
    # on traite directement
    runJob $job $id
  fi
done
```

```bash
nb_success=0;
nb_error=0;
HERE="/Users/samirboudaoud/Desktop/NSY103-Linux/projet_cron_jobs/"
# params :
# $1 : le path du job ex: 'jobs/job1.php'
# $2 : id de la row
runJob() {
  result=$(php "$1")

  case $result in
  "success")
    # 👍 Le programme a été réalisé avec succès
    ((nb_success++))
    # MAJ de l'état en BDD
    sqlite3 $HERE/database.db "UPDATE jobs SET state='success', type='DONE' WHERE rowid = '$2'"
    ;;
  "erreur")
    # 💥 Le programme a retourné une erreur
    ((nb_error++))
    # MAJ de l'état en BDD + Log l'erreur avec la date
    NOW=$(date +%Y-%m-%d\ %H:%M:%S)
    sqlite3 $HERE/database.db "UPDATE jobs SET state='error', type='DONE' WHERE rowid = '$2'"
    echo "$NOW: L'email correspondant au job $2 executé depuis le script $1 a renvoyé une erreur" >> .logs
    ;;
  *)
    echo "Désolé, je ne comprends pas votre réponse ¯\_(ツ)_/¯"
    ;;
  esac
}
```

Extrait de « man awk »

```
FS      regular expression used to separate fields; also settable by option -Ffs.
```

checkSchedules.sh

Vérifie les tâches planifiées

Quand au script qui vérifie les tâches planifiées, il est très simple, il vérifie sur chacune d'entre elles la date d'execution, si elle est dépassée, il met à jour le statut de la tâche en « TODO »

```
#!/bin/bash
#!/usr/bin/php
HERE="/Users/samirboudaoud/Desktop/NSY103-Linux/projet_cron_jobs/"
JOBS_SCHEDULED=$(sqlite3 -separator ";" -newline "|" $HERE/database.db "SELECT rowid,
type, date_completion FROM jobs WHERE type='SCHEDULED' AND date_completion <
datetime('now')")

# Separe chaque ligne et la place dans un tableau
IFS='|' read -r -a jobs_array <<< "$JOBS_SCHEDULED"

nb_changes=0
for row in "${jobs_array[@]}"
do
  id=$(echo "$row" | awk -F ";" '{print $1}');
  type=$(echo "$row" | awk -F ";" '{print $2}');
  date_completion=$(echo "$row" | awk -F ";" '{print $3}');

  if [ "$type" == "SCHEDULED" ]
  then
    # c'est un sheduled, on vérifie les dates
    NOW=$(date +%Y-%m-%d\ %H:%M:%S)

    if [ "$date_completion" \< "$NOW" ]; then
        # la date est passée, on le debloque afin qu'il soit traité
        sqlite3 $HERE/database.db "UPDATE jobs SET type='TODO' WHERE rowid = '$id'"
        ((nb_changes++))
    fi
  fi
done

echo '########################################'
echo "$(date) : Script complété: $nb_changes jobs sont passé au statut: TODO"
```

Tout est en place

v1.0.0 ✅

Le script fonctionne à merveille, mais les développeurs ne s'arrêtent pas la, et notent une liste d'améliorations pour les versions futures du projet.

Parmi celles-ci, faire en sorte de pouvoir:

- Gérer les erreurs de manière plus adaptée, avec des notifications et des rapports.
- Relancer des tâches n fois en fonction de leur importance.
- Gérer des priorités.
- Passer des paramètres aux différentes tâches.
- Horodater la sortie d'erreurs.
- Rendre le script ouvert à d'autres types de tâches.
- Rendre les tâches « scalables » afin de pouvoir les lancer au moment opportun
  En fonction de l'affluence sur le serveur, le débit de la gestion des tâches évoluera

Conclusion

Un traitement plus rapide

Le traitement des tâche est devenu hyper rapide, les clients sont heureux de recevoir une réponse quasi-instantanée, le PDG de LINUXAUTO.com félicite son équipe de développeurs et décide désormais de fournir gratuitement avec chaque commande un masque à l'effigie de Linux.

Tout est bien qui finit bien.