NOHAR

Zeste de savoir

Un bot xkcd en Go - épisode 2

12 novembre 2020

Table des matières

Ceci est le deuxième volet d'une trilogie sur la réalisation d'un petit projet non-trivial en Go, et de son déploiement dans un cluster Kubernetes. Dans l'épisode précédent , nous sommes partis d'une idée formulée sur le serveur Discord de ZdS et avons implémenté, testé et déployé notre bot Discord dans le cloud.

Dans ce billet, nous allons ajouter à ce bot sa *killer feature*, c'est-à-dire **LA** fonctionnalité qui va servir à le différencier de tous les autres projets similaires. Autrement dit, nous allons donner à nos utilisateurs une bonne raison d'inviter **notre** bot xkcd plutôt qu'un autre sur leur serveur!


1. Ce que nous allons réaliser

Si l'on fait une recherche sur top.gg pour trouver un bot qui interagit avec xkcd, nous pouvons remarquer que la plupart des projets qui remontent implémentent pas mal de features similaires les uns par rapport aux autres.

— Tous ces projets (ou presque) permettent de récupérer un xkcd à partir de son numéro, ou bien de récupérer le dernier paru.
— Certains rajoutent une fonctionnalité `random`, permettant d'afficher un xkcd au hasard.
— Certains permettent de faire des recherches, mais de façon plus ou moins laborieuse.

Cela dit, lorsque l'on y regarde de plus près, les bots les plus complets sont aussi les plus complexes à utiliser. Si nous voulons nous différencier, il faut que nous soignions **l'expérience utilisateur** et la rendions la plus intuitive possible. Sur Discord, de nombreux bots utilisent une syntaxe avec un préfixe arbitrairement complexe. C'est d'ailleurs ce que nous avons implémenté jusque là, mais est-ce *vraiment* la façon la plus simple d'interagir avec un bot?

En ce qui me concerne, je suis partisan d'adopter une logique simple: pour interagir avec notre bot, il **faut** que ce soit aussi simple que `@xkcd <ce que je veux>`. Typiquement, ce que nous allons viser dans ce billet:

— `@xkcd 386` doit retourner le xkcd #386,
— `@xkcd latest` doit retourner le tout dernier xkcd,
— `@xkcd random` doit retourner un xkcd au hasard,
— `@xkcd help` doit bien sûr retourner un message d'aide.

Pour ce qui est de rechercher un xkcd, il faut que nous soyons capables de répondre de façon intuitive au besoin le plus courant: lorsque quelqu'un se souvient de la *punchline* ou d'un détail de l'une de ces BD (donc dans les dialogues, le titres ou à la rigueur le texte alternatif), il doit pouvoir le retrouver en décrivant ces mots-clés.

Par exemple si je me souviens de cette fameuse BD à propos des injections SQL , où la mère répond qu'elle surnomme son fils "Bobby tables", je devrais pouvoir retrouver celle-ci en tapant `@xkcd bobby tables`. Contrairement à certain des bots sus-cités, il est *hors de question* de confronter l'utilisateur à une liste de résultats qu'il doit parcourir: notre bot doit répondre avec la BD qui correspond le mieux à la recherche, et donner un moyen à l'utilisateur de préciser sa demande si ce n'est pas ce qu'il voulait. En somme, nous avons besoin de gérer la syntaxe classique des moteurs de recherche: `@xkcd video games -20` ou `@xkcd video games -ferret` doit permettre de retrouver ce xkcd à propos des jeux vidéos dont on se souvient, et qui n'est pas le #20 où il est question d'un furet , et où l'un des personnages dit à un moment "*Let's go play video games*".

Enfin, par soucis du détail, lorsque le bot ne sait pas quoi répondre, cela ne veut plus dire la même chose qu'avant: ce n'est plus une erreur, mais simplement qu'il n'a rien trouvé de satisfaisant. Pour cette raison, plutôt que de réagir à la commande de l'utilisateur avec un (sous-entendant que c'est lui qui a fait du caca), il convient de lui indiquer que l'on ne sait pas quoi répondre: .

2. Indexer et rechercher des documents avec bleve

2.1. Extraire les dialogues d'une BD

Pour indexer une BD, nous allons avoir besoin d'en isoler *les dialogues*. Souvenez-vous des méta-données que nous retourne xkcd:

```
{
  "month": "2",
  "num": 386,
```

```
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

Si l'on observe de plus près le format du *transcript*, on remarque que:

— Les lignes décrivant ce qui est dessiné sont encadrées par `[[` et `]]`;
— Le texte alternatif figure entre `{{` et `}}`;
— Les dialogues sont des lignes au format `Personnage: texte de la réplique`, c'est le texte de la réplique qui nous intéresse;
— Les autres lignes sont "de la voix off", elles aussi nous intéressent.

Écrivons une petite méthode pour extraire les répliques d'un `Comic`:

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

Bien, maintenant passons aux choses sérieuses…

2.2. La bibliothèque d'indexation qui va bien

Vous aurez compris, nous allons avoir besoin d'incorporer un moteur d'indexation à notre bot. Ça tombe bien, car il en existe un tout beau en Go, qui supporte l'indexation et la recherche de documents arbitraires avec la syntaxe classique que nous venons de décrite plus haut: bleve
.

Créons un petit package pour ajouter les fonctionnalités de `bleve` à notre programme:

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

Je vous vois venir d'ici:

> **Tu utilises des variables globales ?! Mais c'est mal!**

Dans les langages *sans namespace* comme C, certes, mais en Go c'est plutôt courant, voire idiomatique à en juger par le package net/http . Dans notre programme, nous avons seulement besoin d'indexer un unique type de documents, et nous n'aurons certainement jamais besoin de plus que ça, alors autant nous simplifier la vie, non?

Pour résumer ce code:

- `SetPath` indique où est sauvegardé l'index que nous voulons utiliser, sur le disque,
- `HasComic` retourne `true` si l'index contient déjà la BD dont le numéro est passé en argument,
- `AddComic` indexe toutes les méta-données d'une BD après en avoir extrait les dialogues,
- `Search` réalise une recherche et retourne une liste de résultats, chacun composé d'un numéro de BD et d'un score de recherche: la liste sera ordonnée avec les résultats les plus pertinents en premier (c'est le comportement par défaut de `bleve`).

Simple et efficace.

3. Indexer xkcd

Maintenant que nous avons incorporé un moteur d'indexation à notre bot, il est temps de lui faire manger des données. Réfléchissons à la façon la plus maligne de nous y prendre.

3.1. Sonder le site pour détecter les nouvelles parutions

Supposons que le bot possède déjà un index bien rempli. Pendant qu'il va tourner, de nouveaux contenus seront publiés. Dans ces conditions, il faut que le bot soit capable de détecter les nouvelles parutions sur xkcd. Pour ce faire, nous allons implémenter un "watcher", qui va récupérer, à intervalles régulier, le dernier xkcd paru, et regarder s'il ne le connait pas encore. Le cas échéant, lorsqu'une nouvelle BD est parue, celui-ci va appeler des callbacks en leur passant les méta-données de la BD.

De la même façon que pour l'indexation, nous allons implémenter ceci sous la forme d'un état privé qui sera global à tout le processus.

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

Notre état privé va donc être composé:

— du numéro du dernier xkcd,
— des *callbacks* à appeler en cas de nouvelle BD,
— de la période à laquelle nous allons sonder le site,
— et d'un mutex pour se protéger contre les accès concurrents.

Les méthodes implémentées ici n'ont rien de particulier, il s'agit de getters et de setters protégés par le mutex. Implémentons maintenant la mise à jour régulière:

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
```

```
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

La partie la plus *tricky*, ici, c'est celle que j'ai surligné. Lorsque l'on crée un *watcher*, on lance un appel périodique à la méthode `checkUpdates` dans une goroutine séparée. Vous remarquerez que je n'ai pas pris la peine de programmer l'arrêt de cette goroutine puisque nous avons uniquement besoin que celle-ci tourne *tant que le bot est en marche*. Si ce package `xkcd` avait été destiné à être distribué indépendamment du bot, il aurait fallu le faire, mais en ce qui nous concerne pour ce projet, il ne sert à rien de résoudre un problème qui n'est pas le nôtre.

Maintenant, il ne nous reste plus qu'à implémenter nos fonctions publiques:

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

Ainsi:

- `SetWatchPeriod` permet de modifier la période à laquelle on interroge xkcd,
- `OnNewComic` enregistre un callback qui sera appelé avec les dernières publications,
- `LatestNum` retourne le numéro du dernier xkcd paru, sans avoir besoin de faire un appel au site, ce qui va nous permettre d'économiser quelques appels superflus.

Ces trois fonctions démarrent le watcher si celui-ci n'existe pas encore (c'est à cela que servent les lignes `once.Do(initWatcher)`).

3.2. L'indexation initiale

Quand le bot va démarrer pour la première fois, il va falloir qu'il se mette à indexer tout le contenu existant. Quand il va (re-)démarrer (supposons après une longue coupure), il va

également falloir qu'il rattrape tout ce qu'il a raté depuis la dernière fois qu'il a fonctionné. Cela revient donc à:

— Récupérer le tout dernier xkcd au démarrage dont le numéro est `N`,
— Itérer de `1` à `N` et indexer toutes les BD *qui ne sont pas déjà présentes dans l'index.*

Lorsque cette boucle sera terminée, l'index sera à jour.

Il faut faire très attention ici: cette mise à jour risque de résulter en de très nombreux appels au site d'xkcd, particulièrement la première fois que le bot sera lancé. Afin de nous comporter en gens civilisés, il est bon de **ne pas faire d'appel au site pour récupérer des infos que nous possédons déjà** et de **mettre un délai entre deux appels au site**. Espacer nos appels d'une seconde est suffisant. Cela signifie que la toute première passe d'indexation va prendre 40 minutes à la louche (le site comporte 2383 *strips* à l'heure où sont écrites ces lignes).

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

L'indexation sera donc démarrée de la façon suivante:

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
```

```
}
```

3.3. Mettre tout bout à bout

Notre bot a maintenant besoin de prendre 3 paramètres de configuration:

— Son token,
— Le chemin de son index dans le système de fichiers,
— La période à laquelle nous voulons détecter les nouvelles publications.

La façon idiomatique de nous y prendre, c'est de créer une structure `Config` qui contient ces valeurs, et que l'on passera en argument de la fonction `Run`. Voici comment celle-ci est désormais adaptée:

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

Aucune difficulté particulière ici.

3.4. Adapter le démarrage du bot

Mettons enfin à jour notre `main` pour prendre en compte ces nouvelles valeurs de configuration. Le bot va maintenant lire les variables d'environnement suivantes:

— `BOT_TOKEN` : le token Discord,
— `WATCH_PERIOD` : la période de raffraichissement (1 minute par défaut),
— `INDEX_PATH` : le chemin de l'index.

Je le place dans une balise *spoiler*.

app/main.go

4. Réagir aux mentions

La dernière chose qu'il nous reste à faire avant de déployer le bot, c'est de le rendre capable de réagir lorsque l'on s'adresse directement à lui. En soi, ce n'est pas bien difficile, mais il y a des choses qui ne s'inventent pas.

Il faut savoir que lorsque l'on mentionne quelqu'un sur Discord, cela peut prendre l'une des deux formes suivantes (où `1234567` est à remplacer par l'ID de la personne) :

— `<@1234567>`
— `<@!1234567>`

Ensuite, et nous pourrons le vérifier en testant le bot, à partir du moment où les documents sont indexés avec le numéro du xkcd comme identifiant unique, il est bon de savoir qu'*une recherche sur le numéro d'un xkcd remontera forcément le document correspondant en première position.* Cela veut dire que nous n'avons pas besoin de créer un cas particulier pour chercher un xkcd en fonction de son ID. Cool, non? 🍊

En dehors de cela, aucune difficulté. Vous remarquerez que le code suivant rajoute également, par rapport au précédent billet, la commande `random` pour récupérer un xkcd au hasard (ce qui nous impose d'initialiser le générateur de nombres aléatoires):

```
{
  "month": "2",
  "num": 386,
  "link": "",
  "year": "2008",
  "news": "",
  "safe_title": "Duty Calls",
  "transcript":
      "[[A stick man is behind computer]]\nVoice outside frame: Are you coming
  "alt":
      "What do you want me to do?  LEAVE?  Then they'll keep being wrong!",
  "img": "https://imgs.xkcd.com/comics/duty_calls.png",
  "title": "Duty Calls",
  "day": "20"
}
```

5. Persister les données avec Kubernetes

5.1. Mettre à jour l'image Docker

Nous n'avons rien à changer au niveau des `Dockerfile` du précédent billet. Poussons simplement la nouvelle image:

```
$ make image
$ make publish
```

5.2. Mettre à jour le déploiement

Au niveau du déploiement, notre bot est maintenant un petit peu plus compliqué à gérer proprement. 🍊

En effet, si nous le déployons de la même façon que la dernière fois, cela ne posera *a priori* aucun problème et, une fois passées les 40 minutes d'indexation, celui-ci fonctionnera tout à fait normalement. Cependant, lorsque nous le redémarrerons, toutes les données qu'il avait indexées seront perdues, et il faudra qu'il aspire à nouveau les métadonnées sur le site d'xkcd… Pour éviter cela, nous avons besoin que les données écrites par notre programme sur son système de fichiers soient **persistantes**. C'est là qu'intervient une nouvelle abstraction de Kubernetes: le *Persistent Volume Claim*.

Concrètement, lorsqu'un *pod* a besoin d'écrire ou de lire des données qui doivent lui survivre, il est possible de demander à Kubernetes: «Réserve-moi un espace de telle taille sur un disque». Celui-ci va alors se débrouiller pour nous fournir un "espace disque", ou plutôt, un *volume* avec les propriétés que nous lui demandons, que nous pouvons *monter* dans les conteneurs de manière à ce que ceux-ci puissent lire ou écrire dedans.

Après un test en local, j'ai pu déterminer qu'un index sur tous les xkcd depuis la création du site nécessite 40Mio. Je vais donc ici réclamer un volume avec une petite marge (100Mio) à Kubernetes:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: xkcd-index
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

Cette configuration lui dit "débrouille-toi pour me donner le volume que tu veux, mais je veux qu'il fasse au moins 100Mio, et je te garantis *qu'un seul pod à la fois* cherchera à y accéder en lecture et en écriture".

```
$ kubectl apply -f deploy/volume.yml
persistentvolumeclaim/xkcd-index created
```

Maintenant, adaptons le fichier de déploiement, de manière à ce que notre bot stocke son index dans ce volume (et configurons-le, au passage, pour qu'il sonde le site toutes les heures en quête d'une nouvelle publication):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: xkcd-bot
spec:
  replicas: 1
  selector:
    matchLabels:
      app: xkcd-bot
  template:
    metadata:
      labels:
        app: xkcd-bot
    spec:
      containers:
      - name: bot
        image: neuware/xkcd-bot
        env:
        - name: BOT_TOKEN
          valueFrom:
            secretKeyRef:
              name: bot-secret
              key: token
        - name: INDEX_PATH
          value: /var/lib/xkcd.index
        - name: WATCH_PERIOD
          value: 1h
        volumeMounts:
          - mountPath: /var/lib/xkcd.index
            name: xkcd-index-storage
            readOnly: false
      volumes:
      - name: xkcd-index-storage
        persistentVolumeClaim:
          claimName: xkcd-index
```

Appliquons cette configuration:

```
$ kubectl apply -f deploy/deployment.yml
deployment.apps/xkcd-bot configured
$ kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
xkcd-bot-5fbd9cc49c-6xm59   1/1     Running   0          19s
```

Et voilà! Notre déploiement est mis à jour, le pod a été automatiquement recréé avec sa nouvelle image, sa nouvelle configuration, et son volume de données. Jetons un oeil à ses logs:

```
$ kubectl logs -f xkcd-bot-5fbd9cc49c-6xm59
2020/11/11 13:59:47 Up and running. Press Ctrl-C to exit.
2020/11/11 13:59:47 updating the index
2020/11/11 13:59:57 indexed comic #1
2020/11/11 13:59:58 indexed comic #2
2020/11/11 13:59:59 indexed comic #3
...
```

Bien, allons prendre un café (et écrivons ce billet) le temps qu'il finisse de créer son index, avant de le tester.

6. Testons !

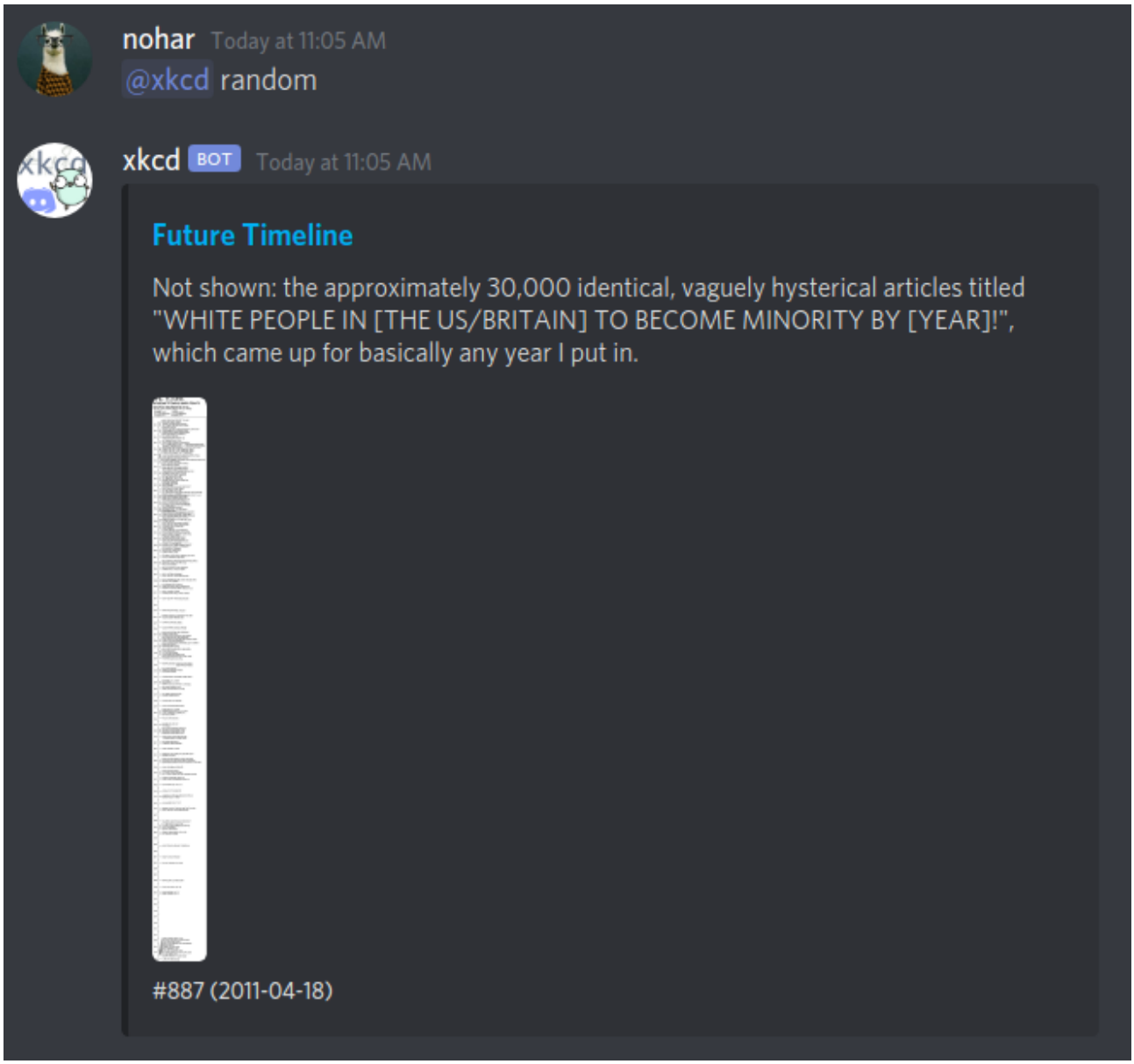
Allez, l'heure du grand test a sonné!

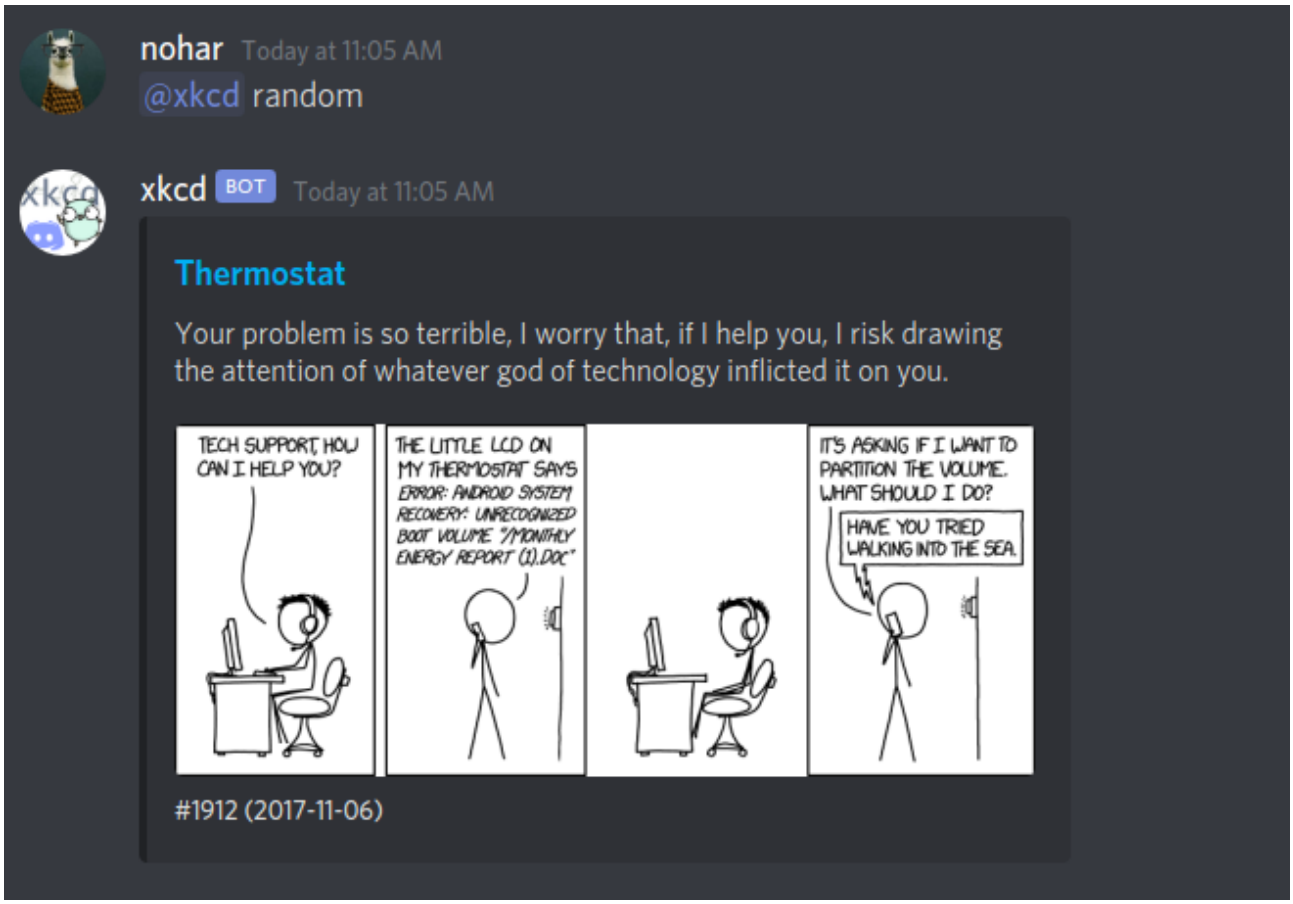
Pour rappel, vous pouvez vous-même tester mon instance du bot en l'invitant sur votre serveur .

6.1. Afficher un xkcd aléatoire

C'est une fonctionnalité que l'on n'avait pas implémentée dans le précédent billet, vérifions que ça fonctionne:

6. *Testons!*

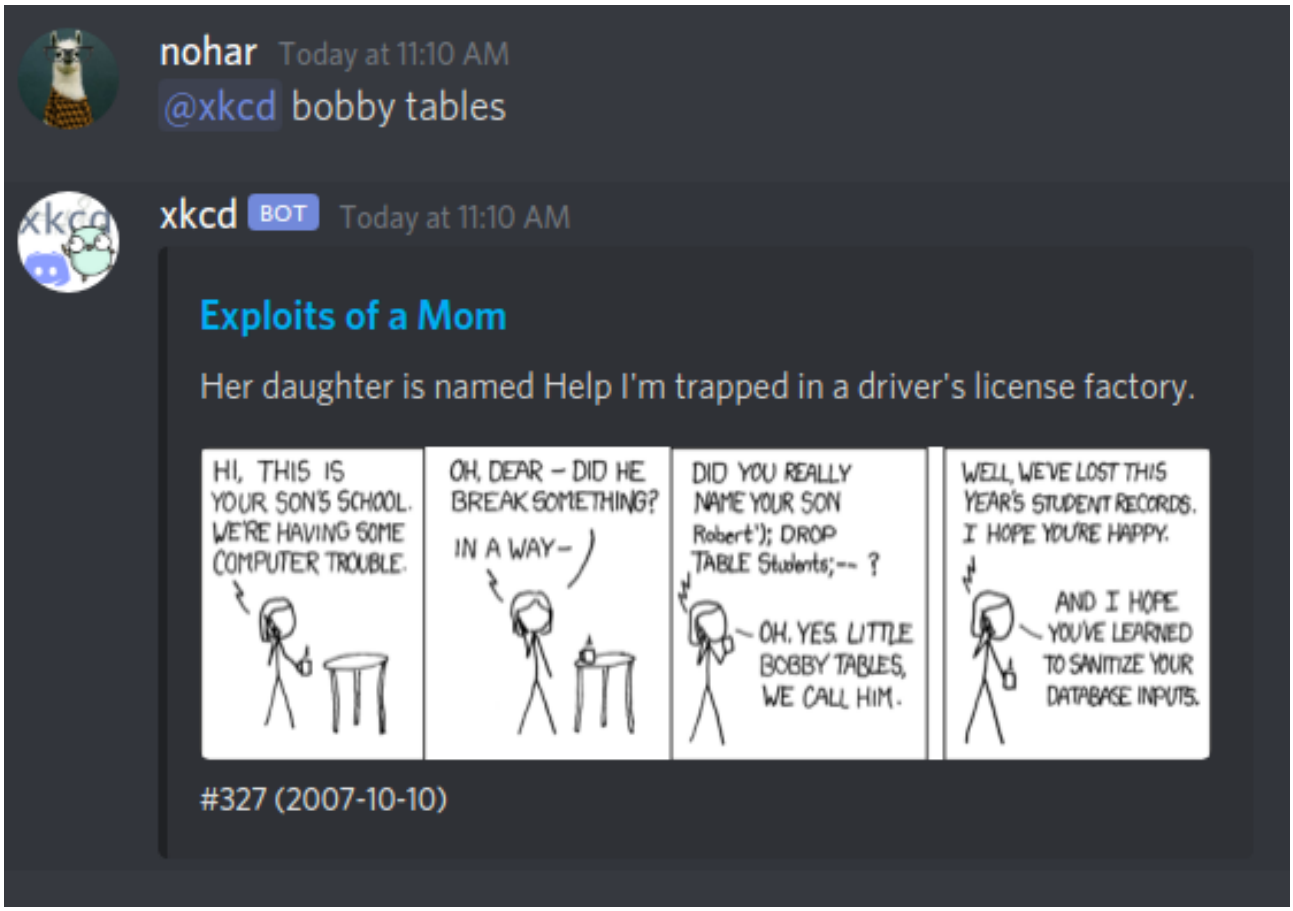

nohar Today at 11:05 AM
@xkcd random
xkcd BOT Today at 11:05 AM
Future Timeline
Not shown: the approximately 30,000 identical, vaguely hysterical articles titled "WHITE PEOPLE IN [THE US/BRITAIN] TO BECOME MINORITY BY [YEAR]!", which came up for basically any year I put in.
#887 (2011-04-18)



Bon… je ne vais pas spammer avec des captures d'écran, on voit bien ici que deux appels successifs retournent deux *strips* au hasard.

6.2. Et la recherche alors ?

Essayons avec l'exemple dont j'ai parlé au début. Le bot arrive-t'il à retrouver le xkcd sur les injections SQL si je lui demande `bobby tables`?

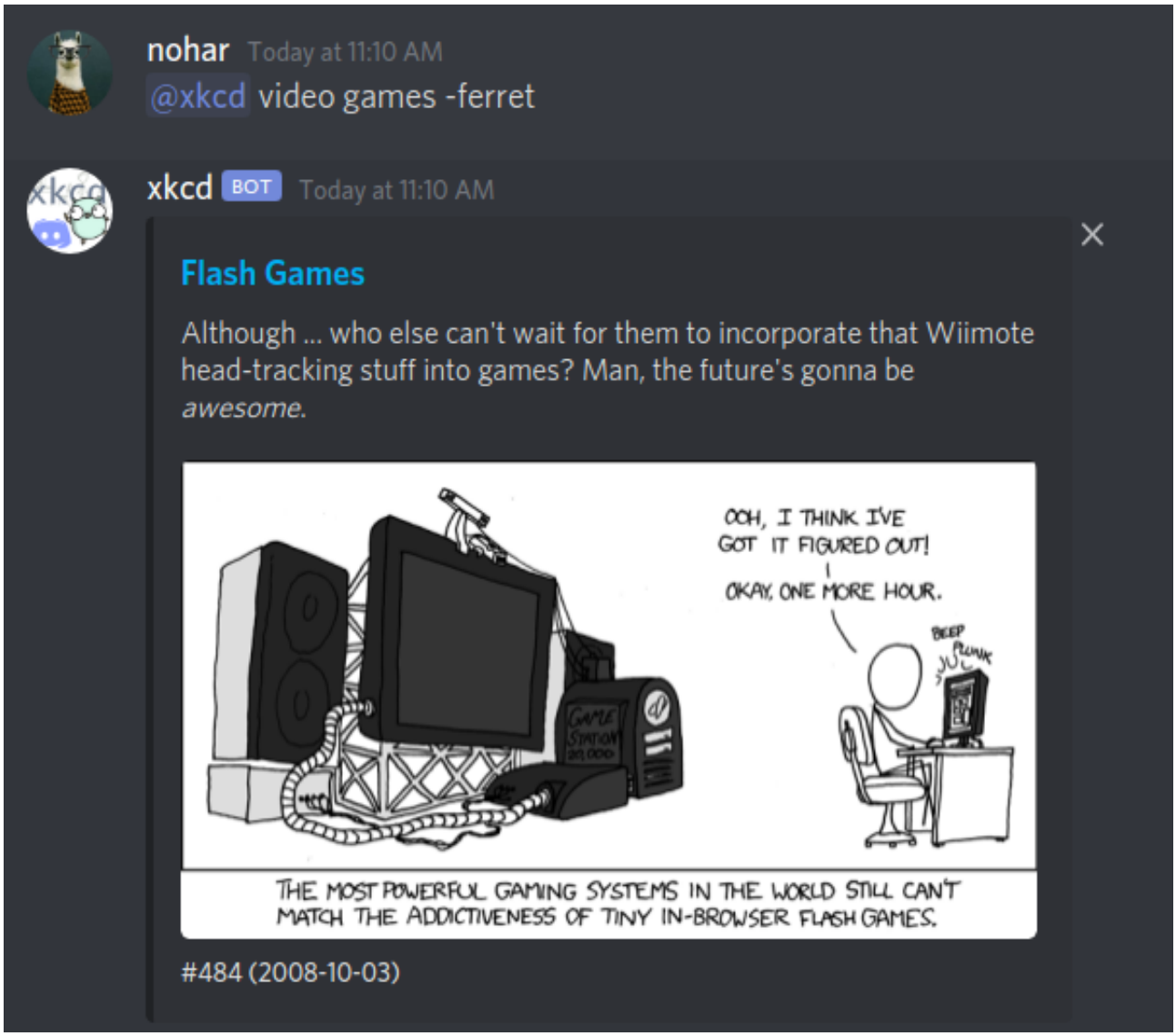


Victoire! \o/

Essayons maintenant une recherche plus compliquée. "Je veux le xkcd qui parle des jeux vidéo là"...

6. Testons!

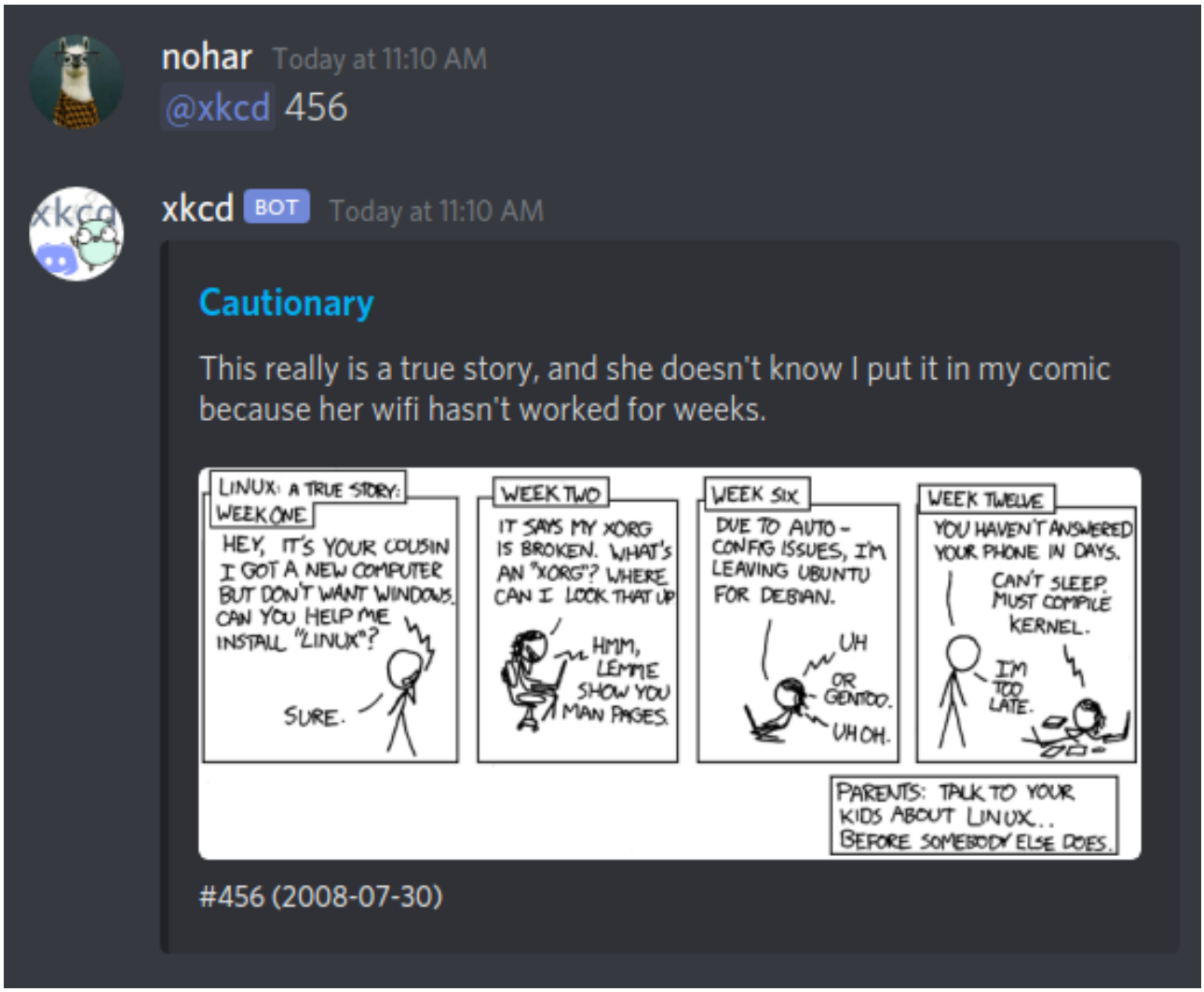
Non, pas celui sur le furet...



Voilà, celui-là.

6.3. Au fait, on peut toujours les rechercher par leur ID ?

Essayons, demandons-lui le numéro 456 pour voir.



Bon, je pense que l'on peut conclure que tout fonctionne comme on s'y attend.

Voilà qui conclut notre second épisode.

Dans le prochain billet, nous allons nous éloigner des fonctionnalités du bot, et nous mettrons à instrumenter celui-ci, de manière à pouvoir surveiller son fonctionnement comme il se doit sur tout projet un minimum sérieux.

Ce sera pour nous l'occasion de découvrir la notion de *service* dans Kubernetes, et d'acquérir quelques bons réflexes sur l'observabilité d'un système.

Contenu masqué

Contenu masqué n°1 :

Contenu masqué

app/main.go

```
$ kubectl logs -f xkcd-bot-5fbd9cc49c-6xm59
2020/11/11 13:59:47 Up and running. Press Ctrl-C to exit.
2020/11/11 13:59:47 updating the index
2020/11/11 13:59:57 indexed comic #1
2020/11/11 13:59:58 indexed comic #2
2020/11/11 13:59:59 indexed comic #3
...
```

Retourner au texte.