وابستگی میان پکیجی

تهمینه دیکشنری‌ای شامل لیست پکیج‌ها به همراه پیش‌نیازهای آن‌ها را از مدیر خود دریافت کرد. قرار بود تا *تهمینه* هر زمان که از او پکیجی خواسته شد او پیش‌نیازهای آن پکیج را اعلام کند. دیکشنری به صورت زیر بود:

```
{'pkg1': ['pkg3'], 'pkg2':['pkg3'], 'pkg3': [], 'pkg4':['pkg1', 'pkg2']}
```

برای دانلود پروژه اولیه روی این لینک کلیک کنید.

حالا *تهمینه* نیاز دارد تا تابعی با نام `sort_dependencies` را پیاده‌سازی کند که در این تابع دیکشنری لیست تمامی پکیج‌ها و نام یک پکیج را به عنوان ورودی دریافت می‌کند و در صورتی که پکیج مورد نظر پیش‌نیاز داشت فقط لیست پیش‌نیازهای آن پکیج را برگرداند و در صورتی که پیش‌نیازی نداشت یک لیست خالی برگرداند.

برای مثال اگر به تابع `sort_dependencies` نام پکیج `pkg4` را بدهند ابتدا باید بررسی کند که آیا پکیج شماره چهار پیش‌نیازی دارد یا خیر. در این مثال پیش‌نیازهای این پکیج برابر است با:

```
pkg4:[pkg1, pkg2]
```

حال باید بررسی شود آیا این پیش‌نیازها، خودشان پیش‌نیاز دارند یا خیر. بعد از بررسی، پاسخ برابر است با:

```
[pkg3,pkg1,pkg2]
```

البته پاسخ زیر نیز قابل قبول است، در کل هر خروجی قابل نصبی از تابع `sort_dependencies` قابل قبول است:

```
[pkg3,pkg2,pkg1]
```

حالت بالا بدون هیچ مشکلی نصب می‌شود زیرا پکیج شماره یک و دو نیاز به نصب پکیج شماره سه پیش از خود دارد و در هر دو خروجی بالا این مورد به درستی رعایت شده‌است. اما چهار خروجی زیر اشتباه است:

```
[pkg2,pkg3,pkg1]
```

```
[pkg1,pkg3,pkg2]
```

```
[pkg2,pkg1,pkg3]
```

```
[pkg1,pkg2,pkg3]
```

زیرا در این خروجی‌ها پکیج‌ها بدون ارور نصب نمی‌شوند و پیش نیاز پکیج یک به درستی رعایت نشده است.

نکات تکمیلی

- فانکشن نباید پکیج تکراری در خروجی داشته باشد.
- ترتیب پکیج ها به همان ترتیبی باشد که می بایست نصب شوند. **توجه کنید** هر خروجی که بتوان با آن بدون مشکل پکیج ها را نصب کرد، قابل قبول است و مسئله به ازای هر ورودی چندین جواب صحیح دارد.
- نام تابع `sort_dependencies` در فایل `solution.py` به هیچ عنوان نباید تغییر کند.
- خروجی می بایست لیستی از نوع رشته باشد.

نحوه ارسال جواب

در این تمرین شما تنها مجاز به تغییر محتوای فایل `solution.py` هستید. تغییرات خود را روی این فایل اعمال کنید و پس از تکمیل آن فایل کامل شده `solution.py` را ارسال کنید.

خطای 503 و اسکریپت Lua

رودابه برای دیپلوی وب‌سرور خود، یک *Helm Chart* نوشته است که یک *deployment* با ایمیج *Nginx* ایجاد می‌کند. در *Helm Chart* مورد نظر برای دسترسی به وب سرور از *Ingress* استفاده شده است.

برای دانلود پروژه اولیه روی این لینک کلیک کنید. ساختار پروژه اولیه به این صورت است:

```
helm-pg
├── Chart.yaml
├── lower.lua
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── answer.yaml
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

رودابه در ابتدا اقدام به نصب وب‌سرور با کمک اجرای دستور زیر می‌کند و وب‌سرور او با موفقیت نصب می‌شود.

terminal

```
helm install nginx -f values.yaml ./
```

دقت داشته باشید *رودابه* برای دسترسی به وب‌سرور با آدرس `chart1-example.local` ، در *Helm Chart* خود *Ingress* تعریف کرده است. حال او با دو مشکل مواجه است که در ادامه به شرح دقیق‌تر هرکدام می‌پردازیم.

مشکل اوّل

رودابه با وارد کردن آدرس `chart1-example.local` در مرورگر خود وارد با خطای *503* مواجه می‌شود. به او کمک کنید تا این مشکل را حل کند.

مشکل دوّم

رودابه بعد از برطرف کردن خطای *503* حال نیازمندی تازه‌ای از سمت تیم *SEO* به سوی او آمده تا تمامی کاراکترهای بزرگ در *URL* را به کاراکترهای کوچک تبدیل کند. برای حل این نیازمندی تابعی با زبان *Lua* بنویسید که تمامی کاراکترهای بزرگ در *URL* را به کاراکترهای کوچک تبدیل کند.

برای مثال اگر *URL* ورودی مانند زیر باشد:

```
/PRODUCT/bar-21-DEEP-bass/
```

تابع شما باید *URL* زیر را برگرداند:

```
/product/bar-21-deep-bass/
```

نکات تکمیلی

- تابع نوشته شده را در فایل `lower.lua` در بلوک `set_by_lua_block` قرار دهید.
- فایل `lower.lua` را در مسیر `/template/ingress.yaml` در *annotation* زیر *import* کنید.

terminal

```
1 nginx.ingress.kubernetes.io/server-snippet
```

- جهت استفاده از *kubernetes* روی سیستم خود می‌توانید از *Minikube* استفاده نمایید.
- برای نصب *Ingress* بر روی *Minikube* روی سیستم خود می توانید از دستور زیر استفاده نمایید.

terminal

```
minikube addons enable ingress
```

- جهت باز نمودن آدرس `chart-example.local` نیاز به تعریف رکورد *DNS* در سیستم شما می باشد. رکورد مورد نظر می بایست به *IP* آدرس کلاستر کوبر اشاره کند
- جهت دریافت *Ingress IP* برای تعریف رکورد *DNS* می‌توانید از دستور زیر استفاده نمایید.

terminal

```
kubectl get ingress
```

نحوه‌ی ارسال جواب

تغییرات مورد نظر برای **مشکل اوّل** را بر روی *Helm Chart* مربوطه در فایل `/template/answer.yaml` اعمال کنید.

تابع *Lua* نوشته شده برای **مشکل دوّم** را در فایل `lower.lua` قرار دهید.

در نهایت پوشه `helm-pg` را *zip* کرده و ارسال کنید. توجه کنید که پس از *extract* کردن فایل *zip* شما، باید فایل `lower.lua` را ببینیم.

```
helm-pg
├── Chart.yaml
├── lower.lua
├── templates
│   └── ...
└── ...
```

نیازمندی های تیم معماری

صبح امروز جلسه‌ای در تیم معماری اپلیکیشن (متشکل از تیم دوآپس و توسعه نرم افزار) برگزار شد و در این جلسه نیاز به پیاده‌سازی تغییراتی بر روی سرویس زیر درخواست شد:

manifest.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: python
spec:
  containers:
  - name: python
    image: python:3.9.10
    args:
    - sleep
    - "3600"
    volumeMounts:
    - name: auth
      mountPath: /tmp
      subPath: auth
  volumes:
    - name: auth
      secret:
        secretName: auth
```

برای دانلود پروژه اولیه روی این لینک کلیک کنید.

لیست تغییرات

۱. خواسته شده تا فایل `auth` به دو فایل `user.txt` از نوع *configmap* و `pass.txt` از نوع *secret* تغییر پیدا کند و در فولدر `/tmp` قرار بگیرد.

۲. قبل از در دسترس قرار گرفتن اپلیکشن نیاز است هربار فایلی از مسیر زیر:

`https://raw.githubusercontent.com/Digiexercise/simple-socket/main/sample-socket.py`

دانلود شده و در دایرکتوری `/python` جایگذاری شود. سپس با دستور زیر در اپلیکیشن اجرا شود:

terminal

```
python3 /python/sample-socket.py
```

توجه داشته باشید که محتوای دایرکتوری `/python` **نباید** `persist` باشد و در هر *restart* باید مجدد فایل جدید دانلود شود.

۳. اپلیکیشن مورد نظر باید سه *Replica* داشته باشد و اطمینان حاصل شود که هر سه پاد در حال سرویس‌دهی به کاربران می‌باشند.
۴. شناسه یکتا پاد را در مسیر `/root/pod_id.yml` ذخیره شود.

توجه داشته باشید برای انجام اینکار از قابلیت `lifecycle` استفاده شود. **همچنین** شناسه ذخیره شده در `pod_id.yml` باید یک عدد یکتا، به ترتیب و طبیعی باشد و با *restart* پاد تغییر نکند.

نکات تکمیلی

- برای اجرای کوبرنتیز می‌توانید از *Minikube* روی سیستم خود استفاده نمایید.
- فقط `manifest.yml` را تغییر داده و ارسال نمایید.
- فایل `manifest.yml` باید دارای یک kind باشد.

نحوه‌ی ارسال جواب

شما فقط می‌توانید محتوای فایل `manifest.yml` را تغییر دهید. تغییرات خودتان را بر روی `manifest.yml` اعمال کنید و فایل اصلاح شده را ارسال نمایید.

پاکسازی سرورهای خراب

حامد مسئول تعدادی سرور پایتونی است که با یک *haproxy* درخواست‌های کاربرها را بین تمامی سرورها تقسیم می‌کند. از بین این سرورها تعدادی به طور *تصادفی* مشکل پیدا می‌کنند. مشکل به این صورت است که این سرورها به جای *statusCode* از نوع `200` ، جواب با *statusCode* از نوع `500` می‌دهند. حال *حامد* می‌خواهد با یک *Ansible Playbook* سرورهایی که مشکل دارند را از مدار خارج کند. به حامد در این امر کمک کنید. شما موظف هستید که یک فایل `tasks.yml` را بسازید که سرورهایی که مشکل دارند را پیدا کند و آن‌ها را از سرورهای پشت *haproxy* خارج کند.

برای دانلود پروژه اولیه روی این لینک کلیک کنید.

توضیح سرور پایتونی

سرور پایتونی فقط دارای یک مسیر (*route*) `/` می‌باشد. که با پول کردن ایمیج داکر زیر می‌توانید آن را تست کنید.

```
registry.gitlab.com/qio/custom/135803/ansibleflaskapi
```

این مسیر یا همان *route* در واقع اسم سرور را به ما پاسخ می‌دهد. برای مثال اگر اسم سرور `server32` باشد، جواب `"server32"` خواهد بود.

هر یک از سرورها دارای یک `Enviroment Variable` به نام `STATUS` میباشد که به طور پیش فرض روی `200` گذاشته شده‌اند. شما می‌توانید با تغییر این متغیر به عدد `500` این مشکل اپ رو تکرار کنید.

برای آشنایی بیشتر با این سرور، فایل `docker-compose` را مشاهده و بررسی کنید.

نکات تکمیلی

- اجرای فایل *ansible* باید روی `localhost` باشد.
- نام فایل *playbook* باید `tasks.yml` باشد.

- تعداد سرورهایی که داخل کانفیگ `haproxy` هستند و همینطور تعداد سرورهایی که مشکل دارند تصادفی است. پس `playbook` مورد نظر باید کاملا *dynamic* باشد.
- **در صورت نیاز!** بدانید پروژه حین داوری در فولدر `/home/project` قرار دارد. ممکن است پاسخ شما نیازی به این نکته نداشته باشد.

- فقط `tasks.yml` را تغییر داده و ارسال نمایید.
- در صورتی که سروری، خراب تشخیص داده شد، باید آن از کانفیگ haproxy خارج شود.

نحوه‌ی ارسال جواب

شما فقط می‌توانید محتوای فایل `tasks.yml` را تغییر دهید. تغییرات خودتان را بر روی `tasks.yml` اعمال کنید و فایل اصلاح شده را ارسال نمایید. همچنین در نظر داشته باشید نحوه اجرای فایل `tasks.yml` شما به صورت زیر خواهد بود:

terminal

```
ansible-playbook tasks.yml
```

پیکربندی HAProxy

محمد در نظر دارد پیکربندی مربوط به *HAProxy* را برای وب‌سایتی که جدیدا در دست طراحی است انجام دهد. در معماری انجام شده، *HAProxy* به عنوان لودبالانسر/پراکسی، جلوی سرورهایی که برای وب سایت ایجاد شده‌اند قرار گرفته است در این پیاده‌سازی از چندین سرور (*api application* و *nginx webserver*) استفاده شده است و برای پاسخ به درخواست‌های وب‌سایت از این سرورها استفاده می‌شود.

در وب‌سایت مورد نظر یک اپلیکیشن با نام *api* که یک *flask api* است وظیفه جواب دادن به درخواست‌ها روی مسیرهای `/games` ، `/auth` و `/mobile` را دارد.

همچنین دو وب سرور *Nginx* وجود دارد که وظیفه جواب دادن به درخواست‌ها روی مسیر روت یا `/` را دارند که به صورت *Fail Over* یکدیگر کار می‌کنند.

برای دانلود پروژه اولیه روی این لینک کلیک کنید.

حال از محمد خواسته شده تا کارهای زیر را انجام دهد:

۱. در *HAProxy* نیاز به پیکربندی داریم که بتوانیم با توجه به بالا بودن هر دو وب‌سرور *nginx*، برای مسیر روت (یا همان `/`) ابتدا فقط ترافیک به وب‌سرور با نام `app_primary` ارسال شود و در صورت از دسترس خارج شدن سرور اول ترافیک، به سمت وب‌سرور دوم با نام `app_secondary` ارسال گردد و زمانی‌که مشکل وب‌سرور اول حل شد ترافیک مجدد فقط به وب سرور `app_primary` ارسال شود.

توجه داشته باشید:

- تنظیمات *HAProxy* باید به گونه‌ای باشد که همزمان ترافیک به هر دو وب‌سرور *nginx* **ارسال نگردد.**
- وب‌سرورهای *Nginx* روی پورت `80` کار می‌کنند.
- در تنظیمات *HAProxy* برای وب‌سرورهای *Nginx* یک *backend* با نام *app* تعریف شده‌است که در باکس زیر جزئیات آن را می‌بینید:

haproxy.cfg

```
backend app
    http-response add-header x-Server %b/%s
    default-server check inter 1s fall 1 rise 3
```

۲. در پیکربندی *HAProxy* مسیرهای زیر باید توسط اپلیکیشن مورد نظر(یعنی *api* یا *nginx*) جواب داده شود:

- درخواست‌های به `/games` ، `/mobile` و `/auth` باید حتما توسط اپلیکیشن *api* جواب داده شود.
- درخواست‌ها به مسیر روت باید توسط وب‌سرور *Nginx* جواب داده شود.

۳. تمامی درخواست‌هایی که توسط پلتفرم‌های همراه (یعنی موبایل، تبلت و ...) حتما بایستی به مسیر `/mobile` هدایت شوند. و در صورتی‌که درخواست مورد نظر از این پلتفرم‌ها نباشد بایستی به مسیر درخواستی خود هدایت شود.

توجه داشته باشید:

- در صورتی‌که درخواست مورد نظر از پلتفرم‌های همراه (موبایل، تبلت و ...) **نباشد** و مسیر مورد نظر در هیچ یک از *backendها وجود *نداشته باشد** *بایستی توسط *Nginx** پاسخ داده شود و به مسیر روت هدایت شود.
- اپلیکیشن *api* روی پورت `5000` کار می‌کند.

۴. در صورتی‌که درخواستی دارای کوئری باشد (به عنوان مثال `/digikala?test=foo`) بایستی به مسیر روت (یا همان *nginx*) هدایت شود.

۵. پیکربندی *HAproxy* باید به گونه‌ای باشد وقتی درخواستی با هدر `X-Secret: key` `BASE64DECODED` `base64` به *HAProxy* رسید قسمت *BASE64DECODED* از هدر درخواستی به صورت *base64* در هدری با نام `auth-hash` به اپلیکیشن *api* داده شود.

توجه داشته باشید:

- اپلیکیشن *api* به این صورت کار می‌کند که اگر درخواستی با هدر `auth-hash` روی مسیر `/auth` برسد هدر مورد نظر را در خروجی چاپ می‌کند. در نظر داشته باشید که خروجی چاپ شده حتما بایستی مقدار *base64* کلید مورد نظر در هدر *X-Secret* باشد.
- درخواست‌های روی مسیر *auth* به صورت زیر ارسال می‌شود و دارای هدر *X-Secret* می‌باشند:

terminal

```
curl -H "X-Secret: key BASE64DECODED RANDOM_STRING" 127.0.0.1:8080/auth
```

به عنوان مثال برای کلید *BASE64DECODED* باید مقدار زیر چاپ شود:

```
QkFTRTY0REVDT0RFRAo=
```

- هیچ تغییری روی کد اپلیکیشن *api* **نباید** داده شود.
- عبارت بعد از *BASE64DECODED* یک رشته تصادفی می‌باشد.
- سرویس *HAProxy* در هیچ حالتی نباید خطای `5XX` بدهد و برای هر مسیری (موجود و یا ناموجود) باید ریسپانس کد `200` داشته باشد.
- در تنظیمات *HAProxy* برای اپلیکیشن *api* که یک *Flask api* می‌باشد *backend* با نام *api* تعریف شده است. در ادامه جزئیات *api* را مشاهده می‌کنید.

haproxy.cfg

```
backend api
    http-response add-header x-Server %b/%s
    default-server check inter 1s fall 1 rise 3
```

- تعداد بکندهای تعریف شده در تنظیمات *HAProxy* نباید بیشتر از دو مورد موجود (*api* و *app*) باشد.
- برای ایجاد پروژه اولیه از یک فایل `docker-compose` استفاده شده است که پورت `8080` برای *HAProxy* پابلیش شده است که درخواست های به پورت `8080` هاست (یا همان سیستم شما)، به *HAProxy frontend* روی پورت `80` ارسال می‌شوند. در ادامه جزئیات *frontend* را مشاهده می‌کنید:

haproxy.cfg

```
frontend http
    bind *:80
```

تنظیمات *HAProxy* به شرح زیر می‌باشد:

haproxy.cfg

```
# haproxy.cfg
global
    user root
    group root
    daemon
    stats timeout 30s

defaults
    log    global
    mode    http
    timeout connect 5000
    timeout client  50000
    timeout server  50000

frontend http
    bind *:80

backend api
    http-response add-header x-Server %b/%s
    default-server check inter 1s fall 1 rise 3

backend app
    http-response add-header x-Server %b/%s
    default-server check inter 1s fall 1 rise 3
```

نکته: در تعریف *backend*ها در *HAProxy* در نظر داشته باشید که اپلیکیشن *api* روی پورت `5000` کار می‌کند و وب‌سرورهای *nginx* روی پورت `80` کار می‌کنند.

اجرای پروژه اولیه

برای اجرای از دستور زیر استفاده کنید:

terminal

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

برای از دسترس خارج کردن *app* وب‌سرور *nginx* می‌توانید از دستور زیر استفاده نمایید:

terminal

```
docker-compose stop app_primary
```

برای انجام تست دسترسی به مسیرهای مورد نظر از مرورگر خود آدرس `127.0.0.1:8080` را باز نمایید به عنوان مثال:

```
http://127.0.0.1:8080/
http://127.0.0.1:8080/mobile
http://127.0.0.1:8080/games
http://127.0.0.1:8080/auth
```

برای تست مسیر `/auth` میتوانید از دستور زیر استفاده کنید:

terminal

```
curl -H "X-Secret: key BASE64DECODED RANDOM_STRING" 127.0.0.1:8080/auth
```

نحوه‌ی ارسال جواب

شما فقط می‌توانید محتوای فایل `haproxy.cfg` را تغییر دهید. تغییرات خودتان را بر روی `haproxy.cfg` اعمال کنید و این فایل را *Zip* کرده و ارسال کنید. توجه کنید که پس از *extract* نمودن

فایل زیپ، باید فایل `haproxy.cfg` مشاهده شود.