4. 보안

읽기 전용 파일시스템

- 참고 문서 : https://github.com/openfaas/workshop/blob/master/lab4.md
- OpenFaaS에서는 실행 환경의 루트 파일 시스템을 읽기 전용으로 만들어 보안 취약점을 줄일 수 있음
- 새로운 함수 만들기

```
$ faas-cli new --lang python3 ingest-file --prefix="127.0.0.1:5000"
```

- 함수 작성하기

```
# ingest-file/handler.py

import os
import time

def handle(req):
    # Read the path or a default from environment variable
    path = os.getenv("save_path", "/home/app/")

    # generate a name using the current timestamp
    t = time.time()
    file_name = path + str(t)

    # write a file
    with open(file_name, "w") as f:
        f.write(req)
        f.close()

    return file_name
```

- 빌드하기

```
$ faas-cli build -f ./ingest-file.yml
```

- registry push

```
$ faas-cli push -f ./ingest-file.yml
```

- deploy

```
$ faas-cli deploy -f ./ingest-file.yml --gateway http://127.0.0.1:9888
```

- 확인

```
$ minikube kubectl -- get pods -n openfaas-fn
```

- 테스트

```
$ echo "Hello function" | faas-cli invoke -f ingest-file.yml ingest-file --gateway http://
127.0.0.1:9888
```

- `readonly_root_filesystem: true` 설정

```
# ingest-file.yml

...
functions:
  ingest-file:
    lang: python3
    handler: ./ingest-file
    image: alexellis2/ingest-file:latest
    readonly_root_filesystem: true
```

- 다시 빌드, 푸시, 배포
- 다시 테스트하면 에러 발생

```
Traceback (most recent call last):
  File "/home/app/index.py", line 19, in <module>
    ret = handler.handle(st)
  File "/home/app/function/handler.py", line 13, in handle
```

```
    with open(file_name, "w") as f:
OSError: [Errno 30] Read-only file system: '/home/app/1642580501.7571223'
```

Timeout 설정

- 참고 문서 : https://github.com/openfaas/workshop/blob/master/lab8.md

- API Gateway의 기본 timeout 설정은 20초
- timeout 종류

> `read_timeout` - HTTP의 request를 읽는 데에 허용되는 시간
> `write_timeout` - HTTP의 response를 작성하는 데에 허용되는 시간
> `exec_timeout` - 함수가 실행될 수 있는 최대 시간

▶️ timeout을 짧게 바꿔보기

- 함수 만들기

```
$ faas-cli new --lang python3 sleep-for --prefix="127.0.0.1:5000"
```

- 함수 작성하기

```
import time
import os

def handle(req):
    sleep_duration = int(os.getenv("sleep_duration", "10"))
    preSleep = "Starting to sleep for %d" % sleep_duration
    time.sleep(sleep_duration)  # Sleep for a number of seconds
    postSleep = "Finished the sleep"
    return preSleep + "\n" + postSleep
```

- yml 수정하기

```
...
functions:
  sleep-for:
    ...
    environment:
      sleep_duration: 10
      read_timeout: "5s"
      write_timeout: "5s"
      exec_timeout: "5s"
```

- 빌드, push, 배포, 확인

```
$ faas-cli build -f ./sleep-for.yml
$ faas-cli push -f ./sleep-for.yml
$ faas-cli deploy -f ./sleep-for.yml --gateway http://127.0.0.1:9888
$ minikube kubectl -- get pods -n openfaas-fn
```

- invoke해보기

```
$ echo | faas-cli invoke sleep-for --gateway http://127.0.0.1:9888
```

`sleep_duration` 이 timeout보다 길기 때문에 아무런 결과도 출력하지 않고 에러와 함께 종료됨

- yml의 sleep_duration 환경변수를 2로 변경해보기

```
# sleep-for.yml

...
functions:
  sleep-for:
    ...
    environment:
      sleep_duration: 2
```

```
$ faas-cli build -f ./sleep-for.yml
$ faas-cli push -f ./sleep-for.yml
$ faas-cli deploy -f ./sleep-for.yml --gateway http://127.0.0.1:9888
$ minikube kubectl -- get pods -n openfaas-fn
```

```
$ echo | faas-cli invoke sleep-for --gateway http://127.0.0.1:9888
```

`sleep_duration` 이 timeout보다 짧아 sleep 후 결과가 출력됨

환경변수와 HTTP header를 이용한 auth 설정

- 참고 문서 : https://github.com/openfaas/workshop/blob/master/lab5.md
- 함수 만들기

```
$ mkdir env-auth && cd env-auth
$ faas-cli new --lang python3 env-auth --prefix="127.0.0.1:5000"
```

- auth 랜덤생성하기

```
$ tr -dc A-Za-z0-9 </dev/urandom | head -c 32
# jAUxOetVwz9qZKlMCOeTh6UuzrAoYFIi
```

- yml 수정하기

```
# env-auth.yml

...
functions:
  env-auth:
    ...
    environment:
      write_debug: true
      auth_token: #생성된 토큰
```

- 함수 작성하기

```
# env-auth/handler.py

import os

def handle(req):
```

```
    auth_token = os.getenv("auth_token")
    user_auth_token = os.getenv("Http_X_Auth_Token")

    string = ""
    if auth_token == user_auth_token:
        string = "Authorized! Welcome."
    else:
        string = "Unautorized! Bye."

    return string
```

💥 HTTP header에 `X-[va-riable]` 이 입력되면 환경변수에 `Http_X_[va_riable]` 으로 인식됨

- 빌드, 푸시, 배포

```
$ faas-cli build -f ./env-auth.yml
$ faas-cli push -f ./env-auth.yml
$ faas-cli deploy -f ./env-auth.yml --gateway http://127.0.0.1:9888
$ minikube kubectl -- get pods -n openfaas-fn
```

- 테스트

```
$ curl http://10.204.24.143:9888/function/env-auth --header "X-Auth-Token: jAUxOetVwz9qZKl
MCOeTh6UuzrAoYFIi" -d ""
```

```
$ curl http://10.204.24.143:9888/function/env-auth --header "X-Auth-Token: " -d ""
```

💥 HTTP header에 `X-[va-riable]` 이 입력되면 환경변수에 `Http_X_[va_riable]` 으로 인식됨

Secrets 설정

- 참고문서 : https://github.com/openfaas/workshop/blob/master/lab10.md

🔥 환경변수에 auth token을 입력하는 것보다 더욱 secure한 방법

- auth token을 이용한 secret 생성

```
$ echo -n jAUxOetVwz9qZKlMCOeTh6UuzrAoYFIi | faas-cli secret create auth-token --gateway h
ttp://127.0.0.1:9888
```

- 생성된 secret 확인

```
$ faas-cli secret ls --gateway http://127.0.0.1:9888
$ minikube kubectl -- get secret auth-token -n openfaas-fn -o json

# if you want delete secrets
# minikube kubectl -- delete secret auth-token -n openfaas-fn
```

- yml 수정

```
# env-auth.yml

...
functions:
  env-auth:
    ...
    environment:
      write_debug: true
      auth_token: #생성된 토큰
    secrets:
      - auth-token
```

💥 auth_token값이 YAML에 직접 기록되어 저징되지 않으므로 더욱 안전합니다.

- 함수 수정

```
# env-auth/handler.py

import os

def handle(req):
    auth_token = os.getenv("auth_token")
    with open("/var/openfaas/secrets/auth-token","r") as authToken:
        auth_token = authToken.read()

    user_auth_token = os.getenv("Http_X_Auth_Token")

    string = ""
    if auth_token == user_auth_token:
        string = "Authorized! Welcome."
    else:
```

```
        string = "Unautorized! Bye."

    return string
```

- 빌드, 푸시, 배포

```
$ faas-cli build -f ./env-auth.yml
$ faas-cli push -f ./env-auth.yml
$ faas-cli deploy -f ./env-auth.yml --gateway http://127.0.0.1:9888
$ minikube kubectl -- get pods -n openfaas-fn
```

- 테스트

```
$ curl http://10.204.24.143:9888/function/env-auth --header "X-Auth-Token: jAUxOetVwz9qZKl
MCOeTh6UuzrAoYFIi" -d ""
$ curl http://10.204.24.143:9888/function/env-auth --header "X-Auth-Token: " -d ""
```

HMAC 이용

- 참고문서 : https://github.com/openfaas/workshop/blob/master/lab11.md
- HMAC (Hash-based Message Authentication Code)

> HMAC은 sender/receiver가 미리 공유하는 대칭 키를 사용. sender는 메시지를 전송하려고 할 때 해시를 생성하고 이 데이터는payload와 함께 전송됨. receiver는 공유 키로 payload에 서명하고 해시가 일치하면 payload가 보낸 사람의 것으로 인식됨.

- 함수 만들기

```
$ mkdir hmac-protected && cd hmac-protected
$ faas-cli new --lang python3 hmac-protected --prefix="127.0.0.1:5000"
```

- 시크릿 생성하기

```
$ tr -dc A-Za-z0-9 </dev/urandom | head -c 32
# urLtXvFo64L6ghAHDRZWgCRHJ70AroRS
$ echo -n urLtXvFo64L6ghAHDRZWgCRHJ70AroRS | faas-cli secret create payload-secret --gatew
ay http://127.0.0.1:9888

$ minikube kubectl -- get secret payload-secret -n openfaas-fn -o json
```

- yaml 파일 수정하기

```
# hmac-protected.yml

...
functions:
  hmac-protected:
    ...
    secrets:
      - payload-secret
```

- 함수 수정

```
import os, hmac, hashlib

def validateHMAC(message, secret, hash):
    receivedHash = getHash(hash)

    expectedMAC = hmac.new(secret.encode(), message.encode(), hashlib.sha1)
    createdHash = expectedMAC.hexdigest()

    return receivedHash == createdHash

def getHash(hash):
    if "sha1=" in hash:
        hash=hash[5:]
    return hash

def handle(req):
    messageMAC = os.getenv("Http_Hmac")
    with open("/var/openfaas/secrets/payload-secret","r") as secretContent:
        payloadSecret = secretContent.read()

    if validateHMAC(req, payloadSecret, messageMAC):
        return "Successfully validated: " + req
    return "HMAC validation failed."
```

- 빌드, 푸시, 배포

```
$ faas-cli up -f ./hmac-protected.yml --gateway http://127.0.0.1:9888
$ minikube kubectl -- get pods -n openfaas-fn
```

🔥 `up` 명령어로 빌드, 푸시, 배포를 한 번에 수행할 수 있습니다!

- 테스트

```
$ echo -n "Welcome!" | faas-cli invoke hmac-protected --sign hmac --key=urLtXvFo64L6ghAHDR
ZWgCRHJ70AroRS --gateway http://127.0.0.1:9888
```

```
$ echo -n "This is a message" | faas-cli invoke hmac-protected --sign hmac --key=abcde  --
gateway http://127.0.0.1:9888
```