۸ استورلت

هدف پروژه استورلت اضافه کردن قابلیت پردازش به پروژه سوییفت جهت پردازش اطلاعات به صورت ایزوله و در مجاورت داده‌ها است. استورلت در واقع کد قابل اجرا است که می‌تواند همانند یک شی در روی سامانه ذخیره سازی سویفت قرار گرفته و روی اشیایی (داده‌ها) در سویفت اجرا شوند. اجرای استورلت ها روی اشیاء کاملا به صورت ایزوله خواهد بود. به این معنا که یک استورلت تنها می تواند به داده هایی دسترسی داشته باشد که مجاز به دسترسی آنها شده است. در ضمن کدهای استورلت دسترسی مستقیم به دیسک، شبکه و سایر منابع سوییفت ندارند.

برای راه‌اندازی استورلت لازم است ابتدا هیولا و کیستون[۲۳] نصب شوند. نصب هیولا را پیش از این در فصل ۶ بررسی کردیم. اما در اینجا لازم است که کی‌ستون ابتدا نصب شود و تشخیص هویت در استورلت به صورت پیش فرض با استفاده از کیستون صورت گیرد. پس ابتدا به سراغ کی‌ستون می‌رویم و در ادامه به سراغ استورلت باز خواهیم گشت.

۸.۱ راه‌اندازی کیستون

کیستون یک سرویس احراز هویت اوپن‌استک می‌باشد که بانک اطلاعاتی کاربر، کاتالوگ سرویس‌های اپن‌استک (ناحیه (region)، سرویس و نقاط پایانی) و API نقاط پایانی را مدیریت می‌کند. کیستون از مکانیسم‌های تایید هویت چندگانه همانند username-password ، Token-base systemو AWS-style Login پشتیبانی می‌کند و تقریبا کاری شبیه به کار اکتیو دایرکتوری در سرویس‌های مایکروسافتی انجام می‌دهد.

در اپن‌استک هر سرویس می‌تواند یک یا چند نوع نقطه پایانی داشته باشد. هر نقطه پایانی می‌تواند مدیر، داخلی و عمومی باشد. با این نقاط پایانی که شامل یکسری url می‌باشد، به سرویس‌ها متصل می‌شویم.

در Keystone چند نوع داده وجود دارد که عبارتند از:

- کاربر: هر کاربر به یک یا چند پروژه و یا دامنه مرتبط است.
- گروه: مجموعه‌ای از کاربران که به یک یا چند پروژه یا دامنه مرتبط است.
- پروژه: واحد مالکیت در اوپن‌استک شامل یک یا چند کاربر است.
- دامنه: واحد مالکیت در اوپن‌استک شامل کاربران، گروه‌ها و پروژه‌ها است.
- نقش : مشخص کننده نقش کاربر در یک پروژه است.

---

[۲۳] keystone

- توکن: شناسایی اعتبار مرتبط با کاربر یا کاربر و پروژه

علاوه بر موارد بالا، شامل نقاط‌پایانی، سرویس‌ها و ناحیه هم می‌باشد. به طور خلاصه در Keystone می‌توان کاربر، پروژه و نقش به صورت جداگانه تعریف نمود و در ادامه هر کاربر را با یک نقش خاص به پروژه متصل می‌کنیم.
روند کلی مدیریت احراز هویت درKeystone به طور خلاصه بدین شکل است که کاربر با یک درخواست که در آن اطلاعات کاربری و پروژه وجود دارد، توکن و لیست کاتالوگ را که شامل نقاط پایانی هم است می‌گیرد و درخواست اصلی خود را به نقطه پایانی مورد نظر سرویس خود می‌فرستد و آن سرویس با Keystone ارتباط برقرار می‌کند تا بررسی کند که آیا توکن موردنظر اجازه دسترسی درخواست شده را دارد یا خیر و اگر توکن اجازه دسترسی داشت، سرویس اوپن‌استکی مورد نظر، درخواست کاربر را انجام می‌دهد و نتیجه را به کاربر برمی‌گرداند.

۸.۱.۱ نصب کیستون

به منظور نصب و راه اندازی Keystone از داکر استفاده شده است و فرض بر این است که داکر روی سیستم نصب می‌باشد. بنابراین به منظور نصب و راه اندازی دستورات زیر اجرا شود:

```
wget https://opengit.ir/AliAkbarBadri/storlet-config/-
/raw/main/keystone.tgz?inline=false

sudo docker load -i keystone.tgz

docker network create monsternet

sudo docker run --name key -p 5000:5000 -h key --network monsternet -d
registry.zdrive.ir/keystone
```

سپس وارد کانتینر هیولا می‌شویم و key را پینگ می‌کنیم که مطمئن شویم همدیگر را می‌بینند.

```
sudo docker exec -it bmonster bash

ping key
```

پس از راه اندازی کانتینر هیولا و ورود به آن، نوبت به پیکربندی سرور پراکسی می‌رسد. چون هیولا نصب شده است، در ادامه میان افزار authtoken و keystoneauth را به پروکسی سرور اضافه می‌کنیم. Key نام کانتینر Keystoneاست که پورت ۳۵۳۵۷ برای دسترسی مدیر و پورت ۵۰۰۰ برای دسترسی عموم به سرویس‌های Keystoneمورد استفاده قرار می‌گیرد. برای اینکار ابتدا با ادیتور nano فایل کانفیگ پروکسی را باز می‌کنیم.

```
sudo docker exec -it bmonster bash

nano /etc/swift/proxy-server.conf
```

و این فایل را با محتویات زیر را جایگزین می‌کنیم:

```
[DEFAULT]

bind_port = 8080

workers = 8

user = swift

log_facility = LOG_LOCAL1

eventlet_debug = true


[pipeline:main]

#Yes, proxy-logging appears twice. This is so that

#middleware-originated requests get logged too.

#pipeline = catch_errors gatekeeper healthcheck proxy-logging cache bulk
tempurl slo dlo ratelimit crossdomain tempauth staticweb container-quotas
account-quotas proxy-logging proxy-server

pipeline = catch_errors gatekeeper healthcheck proxy-logging cache bulk
tempurl slo dlo ratelimit crossdomain authtoken keystoneauth staticweb
container-quotas account-quotas proxy-logging proxy-server


[filter:authtoken]

paste.filter_factory = keystonemiddleware.auth_token:filter_factory

www_authenticate_uri = http://key:5000/

auth_url = http://key:35357/

auth_plugin = password

project_domain_id = default
```

```
user_domain_id = default
project_name = admin
username = admin
password = ADMIN_PASS
cache = swift.cache
include_service_catalog = False
delay_auth_decision = True

[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = admin, swiftoperator

[filter:catch_error]
use = egg:swift#catch_errors

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:proxy-logging]
use = egg:swift#proxy_logging

[filter:bulk]
use = egg:swift#bulk

[filter:ratelimit]
use = egg:swift#ratelimit

[filter:crossdomain]
```

```
use = egg:swift#crossdomain


[filter:dlo]
use = egg:swift#dlo


[filter:slo]
use = egg:swift#slo


[filter:tempurl]
use = egg:swift#tempurl


[filter:tempauth]
#storage_url_scheme = default
#use = egg:swift#tempauth
#user_admin_admin = admin .admin .reseller_admin
#user_test_tester = testing .admin
[filter:staticweb]
use = egg:swift#staticweb


[filter:account-quotas]
use = egg:swift#account_quotas


[filter:container-quotas]
use = egg:swift#container_quotas


[filter:cache]
use = egg:swift#memcache
```

```
[filter:gatekeeper]

use = egg:swift#gatekeeper


[app:proxy-server]

use = egg:swift#proxy

allow_account_management = true

account_autocreate = true
```

مواردی که ما در این فایل تغییر دادیم شامل موارد زیر است:
اول. خطوط زیر در فایل به فایل قبلی اضافه شدند.

```
pipeline = catch_errors gatekeeper healthcheck proxy-logging cache bulk
tempurl slo dlo ratelimit crossdomain authtoken keystoneauth staticweb
container-quotas account-quotas proxy-logging proxy-server

.

.

.

[filter:authtoken]

paste.filter_factory = keystonemiddleware.auth_token:filter_factory

www_authenticate_uri = http://key:5000/

auth_url = http://key:35357/

auth_plugin = password

project_domain_id = default

user_domain_id = default

project_name = admin

username = admin

password = ADMIN_PASS

cache = swift.cache

include_service_catalog = False
```

```
delay_auth_decision = True


[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = admin, swiftoperator
#auth_url = http://key:35357/
.
.
.
```

دوم. با توجه به اینکه در اینجا از میان افزار کیستون برای اهراز هویت استفاده می‌شود، میتوان تنظیمات فیلتر tempAuth را کامنت یا حذف کرد که خطوط زیر است:

```
#[filter:tempauth]
#storage_url_scheme = default
#use = egg:swift#tempauth
#user_admin_admin = admin .admin .reseller_admin
#user_test_tester = testing .admin
#user_test2_tester2 = testing2 .admin
#user_test_tester3 = testing3
```

در فایل پیکربندی بالا operator_roles برای مشخص کردن مدیرهای سرویس هیولا است و به این صورت است که هر کاربر که در پروژه هیولا آن نقش‌ها را داشته باشد مدیر سرویس هیولا خواهد بود. باید دقت کرد که در خط `password = ADMIN_PASS` هر پسوردی نمی‌توان گذاشت و باید حداقل پیچیدگی را داشته باشد وگرنه در ادامه با ارور The request you have made requires authentication مواجه خواهد شد. پس باید به این نکته دقت کرد. همچنین بایستی بعد از این دستورات از نانو خارج شد و دستور زیر را زد تا تغییرات در هیولا اعمال شود:

```
swift-init main restart
```

در ادامه باید با دستور `exit` از کانتینر هیولا خارج شد.

برای آنکه Keystone، به کانتینر هیولای راه‌اندازی شده به عنوان یک سرویس ذخیره‌سازی اشاره کند، url مربوط به سرویس ذخیره سازی (object-store) را به سمت هیولا و url مربوط به شناسایی (identity) را به سمت Keystone تغییر می‌دهیم. برای این منظور ابتدا با دستور زیر وارد کانتینر key می‌شویم:

```
docker exec -it key bash
```

و دستورات زیر را در این کانتینر اجرا می‌کنیم:

```
export OS_IDENTITY_API_VERSION="3"

export OS_AUTH_URL="http://localhost:35357"

export OS_USER_DOMAIN_ID="default"

export OS_PROJECT_DOMAIN_ID="default"

export OS_PROJECT_NAME="admin"

export OS_USERNAME="admin"

export OS_PASSWORD="ADMIN_PASS"

openstack endpoint list
```

خروجی این دستور باید به شکل زیر باشد وگرنه لازم است تغییراتی ایجاد کنیم:

```
[root@key /]# openstack endpoint list
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------------------
-+
| ID                               | Region    | Service Name | Service Type | Enabled | Interface | URL
 |
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------------------
-+
| 5474194f49a1434393255f82744dcb4a | RegionOne | keystone     | identity     | True    | admin     | http://key:35357
 |
| 8c870baeea3645e681f5522a79bac21d | RegionOne | keystone     | identity     | True    | internal  | http://key:5000
 |
| fdc41a308c514627a103a41beddbfd91 | RegionOne | keystone     | identity     | True    | public    | http://key:5000
 |
| 321ad5f4fc184bdf87a55693fc62017d | RegionOne | openio-swift | object-store | True    | admin     | http://bmonster:8080/v1/
 |
| ee5ba230d7ed4473ad1bb657e83e3678 | RegionOne | openio-swift | object-store | True    | public    | http://bmonster:8080/v1/AUTH_%(tenant_id)s
 |
| 275e850fbc8e4ce9a5073a7e18ecf5ff | RegionOne | openio-swift | object-store | True    | internal  | http://bmonster:8080/v1/AUTH_%(tenant_id)s
 |
+----------------------------------+-----------+--------------+--------------+---------+-----------+----------------------------------------------
```

ستون ID یک نشانگر یکتا است و می‌تواند متفاوت باشد. در اجرای کیستون ما ستون‌های URL و Interface با این شکل متفاوت بودند. برای تغییر ستون URL مانند دستورات زیر عمل می‌کنیم:

```
openstack endpoint set --url http://key:35357 <id>

openstack endpoint set --url http://key:5000  <id>

openstack endpoint set --url http://key:5000  <id>

openstack endpoint set --url http://monster01:8080/v1/  <id>

openstack endpoint set --url "http://bmonster:8080/v1/AUTH_%(tenant_id)s"
<id>

openstack endpoint set --url "http://bmonster:8080/v1/AUTH_%(tenant_id)s"
<id>

openstack endpoint list
```

برای ستون Interface نیز کافیست به جای url-- از interface-- استفاده کنیم و مقادیر را با استفاده از جدول مرجعی که تصویرش پیش از این آمده بود تغییر دهیم. و به جای <id> از id مربوط به آن خط استفاده کنیم. پس از این باید وارد هیولا شویم و دوباره آن را راه‌اندازی کنیم:

```
docker exec -it bmonster  bash

swift-init all restart

exit

docker restart bmonster
```

۸.۱.۲ ایجاد پروژه، نقش و کاربر در Keystone

نام پروژه همان نام حساب در هیولا می‌باشد و شناسه پروژه همان شناسه (نام) حساب در هیولا می‌باشد. در این بخش یک پروژه به نام test به همراه دو کاربر tester و tester_member ایجاد می‌کنیم و به هر کدام از این کاربران یک نقش می‌دهیم تا بتوانیم با این کاربرها به پروژه متصل شویم. برای این منظور ابتدا وارد کانتینر key می‌شویم:

```
sudo docker exec -it key bash
```

سپس متغیرهای محیطی زیر را تعریف می‌کنیم:

```
export OS_IDENTITY_API_VERSION="3"
export OS_AUTH_URL="http://localhost:35357"
export OS_USER_DOMAIN_ID="default"
export OS_PROJECT_DOMAIN_ID="default"
export OS_PROJECT_NAME="admin"
export OS_USERNAME="admin"
export OS_PASSWORD="ADMIN_PASS"
```

در مرحله بعدی دستورات openstack زیر را برای تعریف پروژه، کاربر، نقش کاربر و همچنین وصل کردن یک کاربر به پروژه اجرا می‌کنیم:

```
openstack project create test
openstack role create test_member
openstack user create --project test --password testing tester
openstack role add --user tester --project test admin
openstack user create --project test --password member tester_member
openstack role add --user tester_member --project test test_member
```

در نهایت از کانتیر با دستور زیر خارج می‌شویم:

```
exit
```

پس از ایجاد پروژه در Keystone نوبت به بررسی ایجاد حساب متناظر در هیولا می‌رسد، به عبارت دیگر هدف بررسی این موضوع است که آیا با ساخت پروژه در `Keystone`، حساب متناظر آن در هیولا ایجاد گردیده است یا خیر. برای این منظور ابتدا باید وارد کانتینر `bmonster` شویم:

```
docker exec -it bmonster  bash
```

سپس دستورات زیر را اجرا می‌کنیم:

```
export OS_USERNAME=tester

export OS_PASSWORD=testing

export OS_TENANT_NAME=test

export OS_IDENTITY_API_VERSION="3"

export OS_AUTH_URL="http://key:5000"

export OS_PROJECT_NAME="test"

swift stat
```

خروجی به صورت زیر خواهد بود:

```
Account: AUTH_ac86adb18e0d416fbc7cc6fc43a1f4bd

                     Containers: 0

                        Objects: 0

                          Bytes: 0

Containers in policy "policy-0": 0

   Objects in policy "policy-0": 0

     Bytes in policy "policy-0": 0

           Meta Storlet-Enabled: True

    X-Account-Project-Domain-Id: default

         X-Openstack-Request-Id: txdd98d57c399444cfb6dce-005f9e7605

                   X-Timestamp: 1604134759.83055

                    X-Trans-Id: txdd98d57c399444cfb6dce-005f9e7605

                  Content-Type: application/json; charset=utf-8

                 Accept-Ranges: bytes
```

در نهایت دستور زیر وارد شود:

```
exit
```

۸.۱.۳ گرفتن توکن برای یک پروژه

به منظور گرفتن توکن برای یک پروژه همانند مثال زیر که در آن با کاربر tester برای پروژه test درخواست توکن داده شده است، عمل می‌نماییم:

```
curl -i --location --request POST 'http://localhost:5000/v3/auth/tokens'
\

--header 'Content-Type: application/json' \

--data-raw '{ "auth": {

    "identity": {

      "methods": ["password"],

      "password": {

        "user": {

          "name": "tester",

          "domain": { "id": "default" },

          "password": "testing"

        }

      }

    },

    "scope": {

      "project": {

        "name": "test",

        "domain": { "id": "default" }

      }

    }

  }

}'
```

خروجی این دستور به شکل زیر است:

```
curl -i --location --request POST 'http://localhost:5000/v3/auth/tokens'
\

--header 'Content-Type: application/json' \

--data-raw '{ "auth": {

    "identity": {

      "methods": ["password"],

      "password": {

        "user": {

          "name": "tester",

          "domain": { "id": "default" },

          "password": "testing"

        }

      }

    },

    "scope": {

      "project": {

        "name": "test",

        "domain": { "id": "default" }

      }

    }

  }

}'
```

خروجی این دستور به شکل زیر است:

```
HTTP/1.0 201 Created

Date: Sat,

22 Oct 2022 13: 04: 22 GMT
```

```
Server: WSGIServer/0.1 Python/2.7.5

X-Subject-Token: gAAAAABjXNjVIwCUtbWet-hHJ9ehdrrxDsw7j3i2Wwz1kKVEIYH-
FS7jSFgfvglqw3dGzWGH6r6MqWiT5JjoKdrCs967xKWTaxbxuVZcgZ_XySjeJdS10BgNono5Z
CEitiUTaXGeNbTahaTBiXuWvEfQcNNaqwRDaH7C8bAZgdgyM6jrwl8wWs4

Vary: X-Auth-Token

Content-Type: application/json

Content-Length: 1932

x-openstack-request-id: req-be5a3177-cad8-46bf-9ff6-8fdba8d284e5



{

    "token": {

        "is_domain": false,

        "methods": [

            "password"

        ],

        "roles": [

            {

                "id": "b0977872cddf4752ac013e2b369cce07",

                "name": "reader"

            },

            {

                "id": "ab44717741fd44848a7d7585c46d6e27",

                "name": "admin"

            },

            {

                "id": "e9ffced359fa4031ba89ef8cadc6665f",

                "name": "member"

            }

        ],
```

```
        "expires_at": "2022-10-22T14:04:22.000000Z",
        "project": {
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "55cc0a649486434a9de4103c739a3f5c",
            "name": "test"
        },
        "catalog": [
            {
                "endpoints": [
                    {
                        "region_id": "RegionOne",
                        "url": "http://key:35357",
                        "region": "RegionOne",
                        "interface": "admin",
                        "id": "5474194f49a1434393255f82744dcb4a"
                    },
                    {
                        "region_id": "RegionOne",
                        "url": "http://key:5000",
                        "region": "RegionOne",
                        "interface": "internal",
                        "id": "8c870baeea3645e681f5522a79bac21d"
                    },
                    {
                        "region_id": "RegionOne",
```

```
                        "url": "http://key:5000",
                        "region": "RegionOne",
                        "interface": "public",
                        "id": "fdc41a308c514627a103a41beddbfd91"
                    }
                ],
                "type": "identity",
                "id": "bb8289be87de44c29e78da6643a0ad6b",
                "name": "keystone"
            },
            {
                "endpoints": [
                    {
                        "region_id": "RegionOne",
                        "url": "http://bmonster:8080/v1/",
                        "region": "RegionOne",
                        "interface": "admin",
                        "id": "321ad5f4fc184bdf87a55693fc62017d"
                    },
                    {
                        "region_id": "RegionOne",
                        "url":
"http://bmonster:8080/v1/AUTH_55cc0a649486434a9de4103c739a3f5c",
                        "region": "RegionOne",
                        "interface": "public",
                        "id": "ee5ba230d7ed4473ad1bb657e83e3678"
                    },
                    {
```

```
                        "region_id": "RegionOne",
                        "url":
"http://bmonster:8080/v1/AUTH_55cc0a649486434a9de4103c739a3f5c",
                        "region": "RegionOne",
                        "interface": "internal",
                        "id": "275e850fbc8e4ce9a5073a7e18ecf5ff"
                    }
                ],
                "type": "object-store",
                "id": "d11c717f6ac74ba192b403ef24a44a29",
                "name": "openio-swift"
            },
            {
                "endpoints": [],
                "type": "object-store",
                "id": "bf2e2bf10d4843bf9c1b5b46f206fbfe",
                "name": "openio-swift"
            }
        ],
        "user": {
            "password_expires_at": null,
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "7e3ba1467b624c999ee314b0a4a0dc08",
            "name": "tester"
        },
```

```
        "audit_ids": [
            "YxtfzUdmQZ2kDQT9xcX85A"
        ],
        "issued_at": "2022-10-22T13:04:22.000000Z"
    }
}
```

در خروجی بالا `X-Subject-Token` توکنی است که برای این کاربر تولید شده است و ۲۴ ساعت حداکثر معتبر است و هر توکن جدیدی نیز تولید شود توکن قبلی منقضی می‌شود. اما در بخش catalog[1]["endpoints"][2]["url"] یک endpoint مشخص شده است که برای این کاربر ثابت است و ما به آن curl می‌زنیم.

پس در خروجی دستور قبل، توکن و نقطه پایانی پروژه برای دسترسی و ارسال درخواست بازگردانده می‌شود. به منظور ساخت کیسه و شی در حساب (پروژه) مد نظر بعد از گرفتن توکن، همانند مثال زیر که برای حساب test انجام شده است، عمل می‌نماییم:

```
curl --location --request PUT 'http://loocalhost:8080/v1/
AUTH_55cc0a649486434a9de4103c739a3f5c/con1/' --header 'Content-Type:
application/json' --header 'X-Auth-Token: gAAAAABjXNjVIwCUtbWet-
hHJ9ehdrrxDsw7j3i2Wwz1kKVEIYH-
FS7jSFgfvglqw3dGzWGH6r6MqWiT5JjoKdrCs967xKWTaxbxuVZcgZ_XySjeJdS10BgNono5Z
CEitiUTaXGeNbTahaTBiXuWvEfQcNNaqwRDaH7C8bAZgdgyM6jrwl8wWs4'



curl --location --request PUT 'http://loocalhost:8080/v1/
AUTH_55cc0a649486434a9de4103c739a3f5c/con1/test' --header 'Content-Type:
application/json' --header 'X-Auth-Token: gAAAAABjU-pWfYllyYOAtSwyLz-
SncQzqTivyPELAQsRrPe38xsCivrT7wKGp8p78NJ2ginI3fCO3madZGdPVHxLv-2sQR09gXW-
dJavMDwChv2O-
7Gy388nNQIOEzey_eJ_XW2b6uEzbndNqgd_vCegZ4fJ6wFtEuJUCIlpVFa8TYSaq43A3uU' -
-data-raw 'test-body-test'
```

سپس با دستورات زیر می‌توانیم این کیسه یا شی را GET کنیم:

```
curl --location --request GET
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/' --
header 'Content-Type: application/json' --header 'X-Auth-Token:
gAAAAABjXNjVIwCUtbWet-hHJ9ehdrrxDsw7j3i2Wwz1kKVEIYH-
```

```
FS7jSFgfvglqw3dGzWGH6r6MqWiT5JjoKdrCs967xKWTaxbxuVZcgZ_XySjeJdS10BgNono5Z
CEitiUTaXGeNbTahaTBiXuWvEfQcNNaqwRDaH7C8bAZgdgyM6jrwl8wWs4'

curl --location --request GET
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/con1' --
header 'Content-Type: application/json' --header 'X-Auth-Token:
gAAAAABjU-pWfYllyYOAtSwyLz-
SncQzqTivyPELAQsRrPe38xsCivrT7wKGp8p78NJ2ginI3fCO3madZGdPVHxLv-2sQR09gXW-
dJavMDwChv2O-
7Gy388nNQIOEzey_eJ_XW2b6uEzbndNqgd_vCegZ4fJ6wFtEuJUCIlpVFa8TYSaq43A3uU'

curl --location --request GET
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/con1/test
' --header 'Content-Type: application/json' --header 'X-Auth-Token:
gAAAAABjU-pWfYllyYOAtSwyLz-
SncQzqTivyPELAQsRrPe38xsCivrT7wKGp8p78NJ2ginI3fCO3madZGdPVHxLv-2sQR09gXW-
dJavMDwChv2O-
7Gy388nNQIOEzey_eJ_XW2b6uEzbndNqgd_vCegZ4fJ6wFtEuJUCIlpVFa8TYSaq43A3uU'
```

که خروجی همچون تصویر زیر خواهد داشت:

۸.۲ راه‌اندازی استورلت

هدف پروژه استورلت اضافه کردن قابلیت پردازش به پروژه سوئیفت جهت پردازش اطلاعات به صورت ایزوله و در مجاورت داده‌ها است. استورلت در واقع کد قابل اجرا است که می‌تواند همانند یک شی در روی سامانه ذخیره‌سازی سوئیفت قرار گرفته و روی اشیائی (داده‌ها) در سوئیفت اجرا شود. اجرای استورلت‌ها روی اشیاء کاملا به صورت ایزوله خواهد بود. به این معنا که یک استورلت تنها می‌تواند به داده‌هایی دسترسی داشته باشد که مجاز به دسترسی آنها شده است. در ضمن کدهای استورلت دسترسی مستقیم به دیسک، شبکه و سایر منابع سوئیفت ندارند. یک کد استورلت شامل پیاده‌سازی یک اینترفیس به نام invoke است که آرگومان‌های آن شامل یک جریان (stream) ورودی، یک جریان خروجی و یک لاگر است. فرض بر این است که استورلت داده‌های مورد نیاز خود را از جریان ورودی می‌خواند و نتیجه آن را روی جریان خروجی می‌نویسد.

فراخوانی و اجرای استورلت‌ها توسط API های استاندارد سوئیفت صورت می‌گیرد و نیاز به استفاده از ابزار یا روش خاصی نیست، فقط باید هدرهای لازم برای اجرای استورلت تنظیم شده باشد. استورلت‌ها به صورت همگام اجرا

می‌شوند، به این صورت که کد استورلت در یک کانتینر داکر قرار گرفته و اجرا می‌شود. استورلت فعلا از دو زبان پایتون و جاوا پشتیبانی می‌کند.
اجرای استورلت می‌تواند در سه حالت اجرا شود:

۱. در زمان آپلود شی به سوئیفت: در این حالت استورلت شیٔای که آپلود شده است را تغییر داده و بعد در سوئیفت ذخیره می‌کند. به عنوان مثال فایلی به صورت معمولی آپلود شده و استورلت می‌تواند آن را به صورت رمز شده در آورده و بعد رد سوئیفت ذخیره کند.
۲. در هنگام دانلود شی از سوئیفت: در این حالت هنگام دانلود شی استورلت اجرا شده و فایل تغییر یافته در اختیار کاربر قرار خواهد گرفت. مثلاً فایلی که به صورت رمزشده ذخیره شده بود از رمز خارج شده و در اختیار کاربر قرار می‌گیرد. یا به جای تحویل تصویر با اندازه واقعی به کاربر عکس کوچک شده دانلود خواهد شد.
۳. در هنگام کپی شی: اجرای استورلت در هنگام کپی کردن یک شی از یک محل در سوئیفت به محلی دیگر (از یک کیسه به کیسه دیگر) صورت می‌گیرد.

برای نصب استورلت لازم است که ابتدا سرویس‌های هیولا و کیستون به صورت داکری نصب شوند. تشخیص هویت در استورلت به صورت پیش فرض با استفاده از کیستون صورت می‌گیرد. لذا قبل از نصب استورلت باید از کارکرد سرویس مطمئن شویم. همانطور که پیش از این دیدیم؛ نام کانتینر هیولا bmonster و نام کانتینر کیستون را key گذاشته شد. ابتدا لازم است تغییراتی در فایل docker-compose هیولا به شکل زیر ایجاد شود. بایستی چهار مورد به قسمت volumes اضافه شود (در فایل زیر برجسته شدند) که این موارد مربوط به استورلت هستند:

```
# ~/hayoola/SN-docker-compose.yaml
version: "3.8"
services:
  monster1:
    image: mymonster:latest
    restart: always
    container_name: bmonster
    hostname: bmonster
    ports:
      - 8088:8080
      - 6224:6200
```

```
      - 6225:6201
      - 6226:6202
      - 881:873
      # - 29999:19999
    volumes:
      - ./rings:/rings
      - /mnt/sdc1:/srv/node/sdc1
      - /etc/localtime:/etc/localtime:ro
      - ./swift:/etc/swift
      - ./10-swift.conf:/etc/rsyslog.d/10-swift.conf
      - ./01-json-template.conf:/etc/rsyslog.d/01-json-template.conf
      - ./50-default.conf:/etc/rsyslog.d/50-default.conf
      - ./rsyslog.conf:/etc/rsyslog.conf
      - ./rsyslog:/etc/logrotate.d/rsyslog
      - /var/run/docker.sock:/var/run/docker.sock
      - ./docker_device:/home/docker_device
      - /usr/local/lib/storlets:/usr/local/lib/storlets
      - /usr/local/libexec/storlets:/usr/local/libexec/storlets
    networks:
      - monsternet
    healthcheck:
      test: ["CMD", "curl", "-i", "-X GET" ,
"http://bmonster:8080/healthcheck"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - monsternet
```

```
networks:

  monsternet:

    name: monsternet
```

سپس لازم است یک بار این کانتینر restart شود.

فایل‌های مربوط به نصب استورلت را دانلود کرده و در آدرس root/ کانتینر هیولا قرار می‌دهیم:

```
wget https://opengit.ir/AliAkbarBadri/storlet-config/-/raw/main/storlet-package.zip

unzip storlet-package.zip

cd storlet-package

sudo docker cp storlet.tgz bmonster:/root/
```

سپس وارد هیولا می‌شویم:

```
docker exec -it monster bash

cd /root/

tar -xf storlet.tgz

mv storlet/storlets .

mv storlet/devstack/ .
```

سپس دستورات زیر را وارد می‌کنیم تا فولدرهای مورد نیاز ساخته شوند، داکر درون هیولا نصب شود و ایمیج اوبونتو ۲۰.۰۴ گرفته شود.

```
mkdir -p /home/docker_device/cache/scopes

mkdir -p /home/docker_device/logs/scopes

mkdir -p /home/docker_device/scripts

mkdir -p /home/docker_device/storlets/scopes

mkdir -p /home/docker_device/pipes/scopes

mkdir -p /opt/stack/requirements/
```

```
apt update

apt install sudo

apt-get install docker.io

docker pull ubuntu:20.04
```

در ادامه یک فایل به نام `opt/stack/requirements/upper-constraints.txt` می‌سازیم و محتویاتی که درون فایلی به همین نام در آدرس `storlet-package/storlet/upper-constraints.txt` روی هاست قرار دارد در این فایل کپی می‌کنیم.
سپس اسکریپت زیر را اجرا می‌کنیم:

```
cd /root/storlets/

bash -x  s2aio.sh install
```

پس از اجرا شدن فرایند نصب که نسبتا طولانی نیز است، لازم است تغییراتی در فایل‌های کانفیگ داده شود. ابتدا با دستور زیر فایل `proxy-server.conf` را باز می‌کنیم:

```
nano /etc/swift/proxy-server.conf
```

و خط زیر را تغییر می‌دهیم:

```
pipeline = catch_errors gatekeeper healthcheck proxy-logging cache
bulk tempurl storlet_handler  slo dlo ratelimit crossdomain authtoken
keystoneauth staticweb container-quotas account-quotas proxy-logging
proxy-server



[filter:storlet_handler]

use = egg:storlets#storlet_handler

storlet_container = storlet

storlet_dependency = dependency

storlet_gateway_module = docker
```

```
storlet_gateway_conf = /etc/swift/storlet_docker_gateway.conf
storlet_execute_on_proxy_only = false
execution_server = proxy
```

سپس با دستور زیر فایل `object-server.conf` را باز می‌کنیم:

```
nano /etc/swift/object-server.conf
```

و خطوط زیر را تغییر می‌دهیم:

```
pipeline = healthcheck recon storlet_handler object-server


[filter:storlet_handler]
use = egg:storlets#storlet_handler
storlet_container = storlet
storlet_dependency = dependency
storlet_gateway_module = docker
storlet_gateway_conf = /etc/swift/storlet_docker_gateway.conf
storlet_execute_on_proxy_only = false
execution_server = object
```

سپس با دستورات زیر فایل `storlet-proxy-server.conf` را با کپی کردن فایل `proxy-server.conf` ایجاد می‌کنیم:

```
cp /etc/swift/proxy-server.conf /etc/swift/storlet-proxy-server.conf
nano /etc/swift/storlet-proxy-server.conf
```

و خط زیر را تغییر می‌دهیم:

```
pipeline = proxy-logging cache storlet_handler slo proxy-logging
proxy-server
```

سپس با دستور زیر فایل `storlet_docker_gateway.conf` را ایجاد می‌کنیم:

```
nano /etc/swift/storlet_docker_gateway.conf
```

و محتویات زیر را وارد می‌کنیم:

```
[DEFAULT]

storlet_logcontainer = storletlog

host_root = /home/docker_device

cache_dir = /home/docker_device/cache/scopes

log_dir = /home/docker_device/logs/scopes

script_dir = /home/docker_device/scripts

storlets_dir = /home/docker_device/storlets/scopes

pipes_dir = /home/docker_device/pipes/scopes

storlet_timeout = 40

docker_repo =

restart_linux_container_timeout = 3
```

در نهایت یک بار سوئیفت را restart می‌کنیم:

```
swift-init restart all
```

ممکن است در انتهای خروجی مواردی همچون مورد زیر ببینید که اشکالی ندارند:

```
The option “workers” is not known to keystonemiddleware
```

۸.۳ تست کردن استورلت

در این قسمت نحوه‌ی تست کردن استورلت را توضیح می‌دهیم تا هم نحوه استفاده از آن بدانیم و هم مشکلات احتمالی را بشناسیم. کدهای تستی نسبتا متنوعی به زبان‌های جاوا و پایتون در ریپو گیتهاب خود استورلت وجود دارد. برای شروع از مثالی با نام simple شروع می‌کنیم که ساده‌ترین مثال پایتونی موجود در این ریپو است.
برای اجرای یک استورلت لازم است ابتدا دو کیسه با نام‌های storlet و dependency بسازیم:

```
curl --location --request PUT
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/storlet/'
--header 'Content-Type: application/json' --header 'X-Auth-Token:
gAAAAABjeJwZrplPduIvsHidj8mt2ceTEO-
7FV5GLvcBJRdx5f8I8xsVCbmIGL7pjeQdxwhoXz7a6QsQZIq-
rgR7mBEkUSVaAmgsKWSeZg_VasFROvALvp_FYrEE2WOHlWWFT77ffPjnQSDH4SULOGx3H9Kel
X09HKw9_ZoGj-KhYoKAkESdGnk'



curl --location --request PUT
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/dependenc
y/' --header 'Content-Type: application/json' --header 'X-Auth-Token:
gAAAAABjeJwZrplPduIvsHidj8mt2ceTEO-
7FV5GLvcBJRdx5f8I8xsVCbmIGL7pjeQdxwhoXz7a6QsQZIq-
rgR7mBEkUSVaAmgsKWSeZg_VasFROvALvp_FYrEE2WOHlWWFT77ffPjnQSDH4SULOGx3H9Kel
X09HKw9_ZoGj-KhYoKAkESdGnk'
```

سپس بایستی استورلت را درون کیسه storlet آپلود کنیم:

```
curl --location --request PUT --upload-file simple.py
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/storlet/s
imple.py\'

--header 'X-Object-Meta-Storlet-Language: Python' \

--header 'X-Object-Meta-Storlet-Interface-Version: 1.0' \

--header 'X-Object-Meta-Storlet-Object-Metadata: no' \

--header 'X-Object-Meta-Storlet-Main: simple.SimpleStorlet' \

--header 'X-Auth-Token:
gAAAAABjeK48QXGuYnrpBSZZ6nPNSHJUG5d_wHvrzEPELAL4jxnMhXB1y23THI7aVdncx8CZH
TKc0uPY4n_IUbe9sYI_IygeJdJE88gLok9usrw5mIJiFZXlqohbed050fsGgXMOR_zljNgy53
69SoSWl5Qpy5nGzN78G3tmifVY2mKqG_gXsbs'
```

در اینجا چهار هدر جدید اضافه شده است:

- X-Object-Meta-Storlet-Language: زبان استورلت را مشخص می‌کند و می‌تواند Java یا Python باشد
- X-Object-Meta-Storlet-Interface-Version: ورژن استورلت
- X-Object-Meta-Storlet-Main: نام ماژول و کلاس استورلت
- X-Object-Meta-Storlet-Object-Metadata

با دستور زیر نیز می‌توان لیست استورلت‌های موجود را گرفت:

```
curl --location --request GET
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/storlet/'
--header 'Content-Type: application/json' --header 'X-Auth-Token:
gAAAAABjeMRsSc4kQbzs90y3Rs0lk67sk07A1RChMm9JlRXzY96JH92PANuwgG2B76m0MXC4y
CaZNNSrqdDEMmnP2CBjPp2x0AwAL7VY58iq-
iQehTZ1cwsKiVBf6zcYIXpg3ilSHxiJmpslzT113DsUhvwb_x_5JvZALlUvjFh47Au2_diOi5
k'
```

که باید حداقل simple.py که پیش از این آپلود کردیم نشان دهد.

سپس می‌توان این استورلت را روی یک شی دلخواه اجرا کرد. در اینجا ما یک فایل متنی با نام source.txt ایجاد کردیم و داخل آن abcdefghijklmonp ریخته‌ایم. حال می‌خواهیم استورلت simple.py را روی شی source.txt اجرا کنیم:

```
curl --location --request GET
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/con1/sour
ce.txt' --header 'Content-Type: application/json' --header 'X-Run-
Storlet: simple.py' --header 'X-Auth-Token:
gAAAAABjeMRsSc4kQbzs90y3Rs0lk67sk07A1RChMm9JlRXzY96JH92PANuwgG2B76m0MXC4y
CaZNNSrqdDEMmnP2CBjPp2x0AwAL7VY58iq-
iQehTZ1cwsKiVBf6zcYIXpg3ilSHxiJmpslzT113DsUhvwb_x_5JvZALlUvjFh47Au2_diOi5
k'
```

این دستور همانند GET کردن یک شی است با این تفاوت که یک هدر 'X-Run-Storlet: simple.py' اضافه شده است که شامل استورلتی است که می‌خواهیم روی شی اجرا شود.

در صورتی که این دستور با خطای زیر مواجه می‌شود، احتمالا مشکل از میزان دسترسی استورلت به فولدر /home درون کانتینر هیولا است.

```
badri@burna-7:~$ curl --location --request GET 'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/con1/source.txt' --header 'Conte
nt-Type: application/json' --header 'X-Auth-Token: gAAAAABjeNZEVbS5DwfTwdDAatWlBLz_5DizbXu6_ZTcUJsPBeW0m9XyMo-GsgEg--N8-8XwGpiLU2Q7P7ehquAFbtNK
X3q8fb8g4W_jC1G5z8cNPDF5isfbbZnsJqMbWmQO0do9i--7GWH2tcpACelsF2pBCO0s8d7Ero4HhFCXRINMs4NQlZ0' --header 'X-Run-Storlet: simple.py'
<html><h1>Service Unavailable</h1><p>The server is currently unavailable. Please try again at a later time.</p></html>badri@burna-7:~$
```

پس با دستور زیر این مشکل حل می‌شود:

```
docker exec -it monster bash

chmod -R 777 /home
```

بدین شکل یک استورلت را آپلود کردیم و این استورلت را هنگام دانلود (GET) کردن یک شی روی آن اجرا کردیم و نتیجه بدست ما می‌رسد اما شی اصلی تغییر نمی‌کند. حال اگر بخواهیم در هنگام آپلود کردن یک شی، استورلتی روی آن اجرا شود و نتیجه این اجرا ذخیره شود، می‌توانیم از دستور زیر استفاده کنیم که همانند آپلود کردن یک شی است و فقط هدر به آن اضافه شده است. زیر پس هر گاه فایل source2.txt را GET کنیم شکل تغییریافته آن را مشاهده خواهیم کرد:

```
curl --location --request PUT
'http://localhost:8088/v1/AUTH_55cc0a649486434a9de4103c739a3f5c/con1/sour
ce2.txt' --header 'Content-Type: application/json' --header 'X-Run-
Storlet: simple.py' --header 'X-Auth-Token:
gAAAAABjgeUAKwYgjuWYw6tPNJoGeaUrvgFOL0d3Rvzlp4xiPsX6uxC6D1pX3JG1oCkcW8bMx
KEYR7S2177d88j3F9Cck4W3zCiJfDZXZ3LzK1OBmRHZa8mYK2ddOonT8XQ6IET1yIK7XjGQxI
QL38rD2yhQ4Xg7h1r-OdrFOABVWNzenymqwOs' --data-raw 'abcdefghijklmonp '
```

می‌توان دستورات بالا را به شکل کد پایتونی درآورد که به راحتی قسمت گرفتن توکن و ران کردن اجرا شود و سرعت و انعطاف بالاتر رود. این کدها در یک ریپو گیتلب با نام کاهو ذخیره شده‌اند که تا اینجا، بخش گرفتن یک شی به صورت با و بدون اجرای استورلت، آپلود شی با و بدون اجرای استورلت و آپلود استورلت انجام شده‌اند و به درستی کار می‌کنند.

ابتدا کتابخانه‌ی زیر را برای ارسال درخواست‌های HTTP نصب می‌کنیم.

```
pip install requests
```

برای استفاده از این ریپو می‌توانید از دستورات زیر استفاده کنید:

```
get object with runinng storlet: python3 get-object.py con1 source3.txt
simple.py

get object: python3 get-object.py con1 source3.txt
```

```
get list of storlets: python3 get-object.py storlet

upload object: python3 upload-object.py source4.txt

upload object with running storlet: python3 upload-object.py source4.txt
simple.py



upload storlet: python3 upload-storlet.py same.py same.py
same.SameStorlet
```

به مرور این ریپو تکمیل می‌شود.

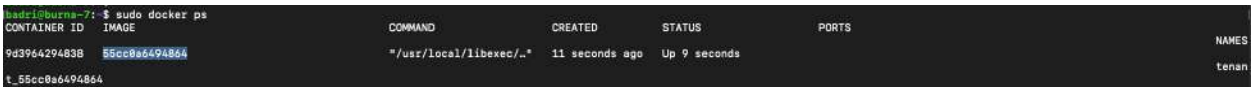
۸.۴ اجرای ocr در استورلت

این بخش شامل کدها، دستورات و نتایج اجرای اپلیکیشن OCR روی استورلت است. یک سند از OCR روی کاتب می‌توان مشاهده کرد. در این سند نحوه‌ی اجرا و بخش‌های مختلف آن توضیح داده شده است. بنابراین ما به سراغ نحوه‌ی اجرای آن روی استورلت می‌رویم.

این کد از پکیج pytesserct استفاده کرده است که کتابخانه‌ای پایتونی برای استفاده از کتابخانه‌ی قدرتمند tesseract-ocr است. اما از آن‌جایی که هیچ کتابخانه‌ی خارجی در ایمیج پایه‌ی استورلت نصب نیست، لازم است که ایمیج دیگری از روی ایمیج پایه ایجاد کنیم و این کتابخانه‌ها را درون آن نصب کنیم وگرنه هیچ استورلتی که کتابخانه‌ی خارجی از آن استفاده شده است قابل اجرا نخواهد بود.

برای اینکار بعد از اجرای هر استورلت اگر دقت کنیم می‌بینیم که یک داکر کانتینر از روی یک ایمیج ساخته می‌شود که بعد از پایان اجرا از بین می‌رود. این ایمیج همان ایمیج پایه‌ای است که باید تغییر دهیم. در تصویر زیر می‌توانیم آن را مشاهده کنیم که نام این ایمیج `55cc0a6494864` است.

```
sudo docker ps
```



حال بایستی یک کپی از این ایمیج بگیریم تا در آتی استفاده کنیم:

```
sudo docker tag 55cc0a6494864:latest 55cc0a6494864_main:latest
```

حال باید این ایمیج `55cc0a6494864_main` را به عنوان ایمیج پایه بگیریم و موارد لازم را روی آن نصب کنیم. محتویات داکرفایل را در ادامه مشاهده می‌کنیم:

```
FROM 55cc0a6494864_main


RUN apt-get -y update; apt-get -y install tesseract-ocr-fas; apt-get -y
install python3-pip


RUN pip3 install pillow \

    && pip3 install pytesseract \

    && pip3 install pdf2image \

    && pip3 install requests
```

در خط اول این فایل ما ایمیج پایه را مشخص کردیم. سپس پکیج `tesseract-ocr-fas` را نصب کردیم که همان پکیج `tesseract-ocr` است که از زبان فارسی نیز پشتیبانی می‌کند. کتابخانه‌ی `pytesseract` در حقیقت از این پکیج که با زبان C++ نوشته شده است استفاده می‌کند و خود تنها یک واسطه است. سپس `pip` را نصب کردیم تا بتوانیم پکیج‌های پایتونی را نصب کنیم. در ادامه نیز چهار کتابخانه‌ی پایتونی را نصب کردیم. از `pillow` هم داخل `pytesseract` استفاده شده است و هم اینکه برای خواندن و نوشتن و پیش‌پردازش و پس‌پرداش تصاویر کتابخانه‌ای لازم است. دو کتابخانه‌ی `pdf2image` و `requests` نیز برای استفاده‌های آتی نصب شده‌اند که در آتی به آن‌ها می‌پردازیم.
با دستور زیر یک ایمیج با نام `55cc0a6494864_temp` را ایجاد می‌کنیم:

```
sudo docker build -t 55cc0a6494864_temp .
```

سپس ایمیج پایه قبلی را پاک کرده و این ایمیج جدید را جایگزین آن می‌کنیم:

```
sudo docker image rm 55cc0a6494864

docker image tag  55cc0a6494864_temp 55cc0a6494864
```

حال هر استورلتی که اجرا شود سراغ این ایمیج جدید می‌رود و کتابخانه‌های لازم نیز نصب هستند.

حال به سراغ کد استورلت ocr می‌رویم. این کد کاملا کامنت‌گذاری شده است. این کد در این لینک نیز موجود است.

```
class OCRStorlet(object):

    def __init__(self, logger):

        self.logger = logger

    def __call__(self, in_files, out_files, params):

        """

        The function called for storlet invocation

        :param in_files: a list of StorletInputFile

        :param out_files: a list of StorletOutputFile

        :param params: a dict of request parameters

        """


        self.logger.debug('START\n')

        self.logger.debug(' check if packages are installed?\n')

        # check the packages

        try:

            from PIL import Image

            from io import BytesIO

            import pytesseract

            self.logger.debug('      Yes!\n')

        except ModuleNotFoundError as e:

            self.logger.debug('      '+ str(e) + '\n')

        # metadata

        metadata = in_files[0].get_metadata()

        metadata['test'] = 'ocr'

        out_files[0].set_metadata(metadata)
```

```
        self.logger.debug(' Reading the image!\n')
        # read block by block and pretend to be an image
        image = b""
        while True:
            buf = in_files[0].read(512)
            if not buf:
                break
            # concat image together
            image += buf
        self.logger.debug(' OCRing!\n')
        # read blocks as an image
        image = Image.open(BytesIO(image))


        # pass the image to pytesseract
        custom_oem_psm_config = r'--oem 3 --psm 6'
        ocr_text = pytesseract.image_to_string(image, lang='eng+fas',
config=custom_oem_psm_config)
        self.logger.debug(' Start to return result:\n')


        # send back the result as byte
        try:
            self.logger.debug('     Writting back...\n')
            out_files[0].write(ocr_text.encode('utf-8'))
        except Exception as e:
            self.logger.debug('     '+ str(e) + '\n')
        self.logger.debug('END\n\n')
        in_files[0].close()
```

```
        out_files[0].close()
```

در این کد یک تابع __init__ داریم که یک logger را در اختیار ما قرار می‌دهد تا بتوانیم در مراحل مختلف از آن برای لاگ کردن استفاده کنیم که در استورلت به عنوان دیباگر نیز به ما کمک می‌کند. این لاگ‌ها در فایل زیر قابل مشاهده است:

```
/home/docker_device/logs/scopes/55cc0a6494864/ocr.OCRStorlet/storlet_invo
ke.log
```

سپس در تابع __call__ ابتدا چک می‌کنیم که آیا پکیج‌ها نصب هستند یا خیر. اگر نصب نباشند یک لاگ می‌اندازیم تا ببینیم چه پکیجی نصب نیست. همانطور که می‌بینید ما از این لاگر به عنوان دیباگر نیز استفاده می‌کنیم. سپس متادیتاهای لازم را ست می‌کنیم.
مرحله‌ی بعدی نوبت خواندن تصویر به صورت بلاک بلاک است تا بعد از چسباندن آن‌ها به همدیگر با استفاده از دو کتابخانه‌ی io و Image این تصویر را به صورت یک تصویر دربیاوریم تا بتوانیم با آن هر کاری لازم است انجام دهیم.
در ادامه نیز این تصویر به کتابخانه‌ی pytesseract می‌دهیم و نتیجه‌ی آن را تبدیل به byte array کرده و برمی‌گردانیم.
حالا که کد این استورلت را فهمیدیم لازم است تا از کتابخانه‌ی کاهو استفاده کنیم تا استورلت خود را اپلود کنیم همانطورکه پیش از این استورلت simple را آپلود کردیم. با استفاده از دستور زیر این کار امکان پذیر است:

```
python3 upload-storlet.py ocr.py ocr.py ocr.OCRStorlet
```

این دستور در ورودی اول فایل مورد نظر را دریافت می‌کند، ورودی دوم نام مدنظر ماست و ورودی آخر نام کلاس اصلی ما در ماژول مورد نظر است. بعد از این دستور باید `<Response [201] >` را در ترمینال خود همچون تصویر زیر ببینیم.

با دستور زیر می‌توانیم ببینیم چه استورلت‌هایی داریم:

```
python3 get-object.py storlet
```

که همچون تصویر را باید مشاهده کنیم:

```
badri@burna-7:~/kahu$ python3 get-object.py storlet
input and token time: 1.0593314170837402
storlet time: 0.8396267890930176
all time: 1.8989582061767578
-------------------------
ocr.py
same.py
simple.py
test.py
thumbnail.py
```

سپس بایستی یک تصویر را آپلود کنیم که تصویر تستی زیر است:

با این دستور آپلود عکس را می‌توان انجام داد:

```
python3 upload-file.py pic.png pic.png con1
```

در نهایت با دستور زیر می‌توان استورلت ocr را روی این تصویر اجرا کرد:

```
python3 get-object.py con1 pic.png ocr.py
```

که خروجی زیر را دریافت خواهیم کرد:

همانطور که مشخص است حدودا ۷ الی ۹ ثانیه برای اجرای این استورلت روی تصویر ما زمان لازم بود. کمتر از یک ثانیه‌ی آن مربوط ورودی دادن و توکن احراز هویت گرفتن بود و باقی آن مربوط به اجرای استورلت بود.

در این دستور نام کیسه به کاهو ورودی داده می‌شود که در اینجا con1 است. همچنین می‌توان با دستور زیر کیسه‌ی دیگری ساخت تا آن را ورودی داد:

```
python3 create-container.py con-ocr
```

همچنین می‌توان هنگام آپلود کردن نیز از استورلت استفاده کرد. برای اینکار با استفاده دستور زیر می‌توان با استفاده از ابزار کاهو هنگام آپلود کردن یک تصویر فایل خروجی را نیز مشخص کرد که خروجی استورلت را درون آن بریزد. مثلا برای همان تصویر تستی قبلی خواهیم داشت:

```
python3 upload-file.py pic.png pic.txt con1 ocr.py
```

در این دستور نتیجه اجرای استورلت ocr روی فایل آپلودی pic.png درون فایل pic.txt در کیسه‌ی con1 ذخیره می‌شود.

```
badri@burna-7:~/kahu$ python3 upload-file.py pic.png pic.txt con1 ocr.py
input and token time: 1.0133862495422363
storlet time: 8.938635349273682
all time: 9.952021598815918
---------------------------
```

برای دیدن نتیجه آن نیز می‌توان فایل pic.txt را به شکل زیر دریافت کنیم:

```
python3 get-object.py con1 pic.txt
```

خروجی این دستور به شکل زیر خواهد بود:

```
badri@burna-7:~/kahu$ python3 get-object.py con1 pic.txt
input and token time: 1.0173954963684082
storlet time: 0.971898078918457
all time: 1.9892935752868652
---------------------------
عدیریت اشیای بزرگa وبژگ های پلتفرم داده ای صوشمند برنا
» امکان مدیریت اشیاء بزرگ با روش های ایسنا و پویاTS
امکان قطعه بندی فایل های بزرگ جبت بهینه سازی ذخیره و بازیابی + 8938
احنید · امکان ایجاد اشیای پویای ترکیبی با سایز متغیر برای مدیریت کلان داده ها
امکان ذخیره سازی داده ها بصورت رمزنگاری شده ®
= © پشتیبانی از روش های مختلف احراز هویت پردازش داده ها
کنترل دسترسی چند لایه (مدیریت نحوه دسترسی به داده ها) وی » توانایی پردازش داده ها در مقیاس بالا
* پشتیبانی از پردازش های بلادرنگ و داده های سری زمانیa
ذخیره سازی متوازن = » پشتیبانی از پردازش جریان داده ها و پردازش های آفلاین
+ توزیعمتوازن داده در حامنه های خرلبی مخحلف + » پشتیبانی از محاسبات بدون سرور (و0/۳۴۱ 00 Y(50۷۵/626
مدیریت توازن مخزن داده ها (قابلیت وزن دهی به دیسک ها) · پشتیبانی از روش های یادگیری ماشین و پردازش های هوش مصنوعی ·
تا » توازن خودکار داده ها در نودهای جدید سازگاری سخت افزاری 4
سازگار با انوا ها جبت راه انداز »
sp ling taoوب pepe senna emu: {TY
· (شناسایی خرابی داده ها) پیمایش خود کار مخازن داده
بشتبانی از ذخیره سازی داده ها در نودهای رزرو (در صورت قطع بودن نظارت ۳۶۰ درجه ·
بخشی از شبکه) » نظارت بر عملکرد سامانه با سطوح ریزدانگی مختلف (دیسک، سرور. کلاستر)
3 » (005) مدیریت و توزیع مطمئن اطلاعات پیکر بندی و توپولوژی بین نودها · نظارت جامع بر شاخص های سطح کیفیت سرویس
عدم وجود نقطه شکست در معماری سامانه ۷ راهکار جامع تجمیع لاگ و کشف نارهنجاری ها
Pre ® تأخیر در سرویس دهی در زمان بازیابی داده ۱ · سیستم هشدار برای مدیریت شرایط بحرانی
· امکان پردازش رخدادهای پیچیده
· امکان گزارش گیری به صورت گرافیکی با قابلیت دریافت نتیجه در قالب فایل
```

۹ ابزارهای هوش مصنوعی

این بخش شامل توضیحات مربوط به مدل‌های هوش مصنوعی است که با استفاده از storlet قرار است روی اشیاء هیولا اجرا شوند. در انتخاب این تسک‌ها سعی بر این شده است که مسائلی انتخاب شوند که برای کاربران جذاب و کارا باشد تا مطابق با اهداف هیولا در جهت گسترش بیزینسی و بازار خود باشد. مدل‌ها نیز با هدف اینکه در دمو استفاده شوند انتخاب شده‌اند و سعی شده است بهترین مدل موجود برای هر یک از مسائل انتخاب شود که بدون آموزش مجدد یا تنظیم دقیق قابل استفاده باشد.

۹.۱ ترجمه ماشینی

در این بخش نحوه استفاده و اجرای مدل انتخابی برای مسئله ترجمه ماشینی انگلیسی به فارسی و برعکس توضیح داده شده است. در اینجا از مدل mt5 استفاده شد. این مدل بر روی دادگان فارسی توسط گروه persiannlp تنظیم دقیق شده است و در سه سایز small، base و large منتشر شده است. کدهای مربوط به ترجمه انگلیسی به فارسی در ادامه آورده شده و کدها کامنت‌گذاری شده است. ابتدا ورودی متنی کاربر که از فایل خوانده می‌شود توکنایز و انکود می‌شود سپس ورودی که تبدیل به عدد شده است به مدل داده می‌شود تا خروجی تولید شود اما این خروجی به صورت token_idها است و بایستی دیکود شود تا به زبان فارسی دربیاید. در نهایت نیز این خروجی در فایل نوشته می‌شود.

```
from transformers import MT5ForConditionalGeneration, MT5Tokenizer
import sys
# size of model and model name
model_size = "small"
model_name = f"persiannlp/mt5-{model_size}-parsinlu-translation_en_fa"
# load the dokenizer
tokenizer = MT5Tokenizer.from_pretrained(model_name)
# load the model
model = MT5ForConditionalGeneration.from_pretrained(model_name)

def run_model(input_string, **generator_args):
```

```
    # tokenize and encode the input
    input_ids = tokenizer.encode(input_string, return_tensors="pt")
    # translate the input
    res = model.generate(input_ids, **generator_args, max_length = 100)
    # decode the token ids
    output = tokenizer.batch_decode(res, skip_special_tokens=True)
    # write on output
    f = open("output.txt", "w")
    f.write(output[0])
    f.close()
    print(output[0])
    return output
# check there is a input file
if len(sys.argv) < 2:
    sys.exit('No file input ERROR, give a file path!')

# open the input file
f = open(sys.argv[1], "r")
input_text = f.read()
f.close()

run_model(input_text)
```

برای اجرای این کد ابتدا یک virtualenv ایجاد می‌کنیم تا کتابخانه‌های لازم را درون آن نصب کنیم:

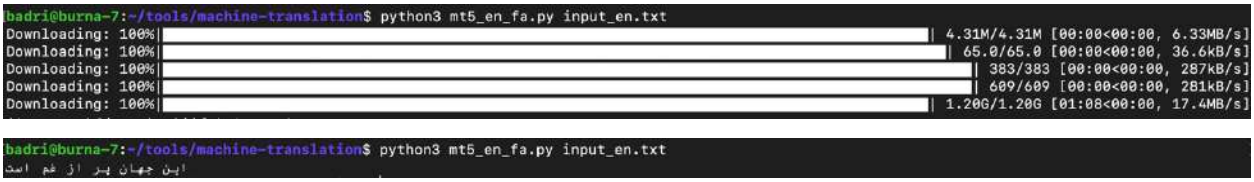
```
pip install virtualenv
virtualenv myenv
source myenv
```

برای اجرای این کد لازم است ابتدا کتابخانه زیر نصب شود:

```
pip install transformers[sentencepiece]
```

بعد از نصب این پکیج کافی است که با دستور زیر کد را اجرا کنیم. با اجرای کد ابتدا توکنایزر و مدل یک بار دانلود می‌شوند که حجمی حدود ۱.۲ گیگ دارند. سپس با اجرای برنامه و ورودی دادن فایل به آن ترجمه آن پرینت می‌شود و همچنین در خروجی نیز به نمایش درمیاید. برای مثال با نوشتن عبارت « this world is full of sadness» در فایل input_en.txt و اجرای برنامه زیر مشاهده می‌کنیم:

```
python3 mt5_en_fa.py input_en.txt
```



همچنین برای ترجمه فارسی به انگلیسی نیز کد زیر به شکل همان کد بالا اجرا می‌شود و تنها نام مدل تغییر می‌کند.

```
from transformers import MT5ForConditionalGeneration, MT5Tokenizer

import sys


model_size = "small"

model_name = f"persiannlp/mt5-{model_size}-parsinlu-opus-
translation_fa_en"

tokenizer = MT5Tokenizer.from_pretrained(model_name)

model = MT5ForConditionalGeneration.from_pretrained(model_name)


def run_model(input_string, **generator_args):

    input_ids = tokenizer.encode(input_string, return_tensors="pt")

    res = model.generate(input_ids, **generator_args, max_length = 100)
```

```
    output = tokenizer.batch_decode(res, skip_special_tokens=True)
    f = open("output.txt", "w")
    f.write(output[0])
    f.close()
    print(output[0])
    return output


if len(sys.argv) < 2:
    sys.exit('No file input ERROR, give a file path!')


f = open(sys.argv[1], "r")
input_text = f.read()
f.close()


run_model(input_text)
```

برای اجرای این برنامه کافی است که دستور زیر را وارد کنیم که ابتدا همچون مدل قبلی فایل‌های مدل برای یک بار دانلود شوند و سپس خروجی همچون تصویر زیر برای ورودی «این روزها می‌گذرند» دریافت کنیم.

```
python3 mt5_fa_en.py input_fa.txt
```

۹.۲ تشخیص اشیاء

در این بخش نحوه استفاده و اجرای مدل انتخابی برای مسئله تشخیص اشیا توضیح داده شده است. در این بخش از مدل yolov5s استفاده شده است. این ورژن از yolov5 از گیت خود این مدل دانلود شده و حجمی ۲۹ مگابایتی دارد و سبک است. کد کامنت‌گذاری شده استفاده از این مدل در ادامه آورده شده است. در این کد ابتدا مدل لود می‌شود. سپس عکس ورودی خوانده می‌شود، عکس به عکس مربعی ۶۴۰ در ۶۴۰ تبدیل می‌شود و سپس به مدل داده می‌شود تا پیش‌بینی آن را بگیرد. در نهایت برای هر پیش‌بینی که مدل بیش از ۰.۴ اطمینان

داشته باشد، نام کلاس، چهار گوشه‌ی کادر دور شی، میزان اطمینان ذخیره می‌شود تا در مرحله بعدی با استفاده از نام کلاس‌ها که از فایل خوانده شده روی عکس اعمال شوند و در فایل خروجی ذخیره شوند.

```
import numpy as np
import cv2
import sys


if len(sys.argv) < 2:
    sys.exit('No file input ERROR, give a file path!')


# step 1 - load the model


net = cv2.dnn.readNet('yolov5s.onnx')


# step 2 - feed a 640x640 image to get predictions


def format_yolov5(frame):
    row, col, _ = frame.shape
    _max = max(col, row)
    result = np.zeros((_max, _max, 3), np.uint8)
    result[0:row, 0:col] = frame
    return result



image = cv2.imread(sys.argv[1])
input_image = format_yolov5(image) # making the image square
blob = cv2.dnn.blobFromImage(input_image , 1/255.0, (640, 640),
swapRB=True)
```

```
net.setInput(blob)
predictions = net.forward()

# step 3 - unwrap the predictions to get the object detections

class_ids = []
confidences = []
boxes = []

output_data = predictions[0]

image_width, image_height, _ = input_image.shape
x_factor = image_width / 640
y_factor =  image_height / 640

for r in range(25200):
    row = output_data[r]
    confidence = row[4]
    if confidence >= 0.4:

        classes_scores = row[5:]
        _, _, _, max_indx = cv2.minMaxLoc(classes_scores)
        class_id = max_indx[1]
        if (classes_scores[class_id] > .25):

            confidences.append(confidence)

            class_ids.append(class_id)
```

```
            x, y, w, h = row[0].item(), row[1].item(), row[2].item(),
row[3].item()
            left = int((x - 0.5 * w) * x_factor)
            top = int((y - 0.5 * h) * y_factor)
            width = int(w * x_factor)
            height = int(h * y_factor)
            box = np.array([left, top, width, height])
            boxes.append(box)

class_list = []
with open("classes.txt", "r") as f:
    class_list = [cname.strip() for cname in f.readlines()]

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.25, 0.45)

result_class_ids = []
result_confidences = []
result_boxes = []

for i in indexes:
    result_confidences.append(confidences[i])
    result_class_ids.append(class_ids[i])
    result_boxes.append(boxes[i])

for i in range(len(result_class_ids)):

    box = result_boxes[i]
```

```
    class_id = result_class_ids[i]



    cv2.rectangle(image, box, (0, 255, 255), 2)

    cv2.rectangle(image, (box[0], box[1] - 20), (box[0] + box[2],
box[1]), (0, 255, 255), -1)

    cv2.putText(image, class_list[class_id], (box[0], box[1] - 10),
cv2.FONT_HERSHEY_SIMPLEX, .5, (0,0,0))

output = sys.argv[1].split('.')[0]+'_detected.'+sys.argv[1].split('.')[1]

cv2.imwrite(output, image)
```

برای اجرای این کد ابتدا یک virtualenv ایجاد می‌کنیم تا کتابخانه‌های لازم را درون آن نصب کنیم:

```
pip install virtualenv

virtualenv myenv

source myenv
```

ابتدا باید کتابخانه زیر نصب شود:

```
pip install opencv-python==4.5.5.62
```

سپس با دستور زیر روی هر عکسی می‌توان این کد را اجرا کرد تا نتایج روی تصویری با همان نام و پسوند _detected اعمال می‌شود:

```
python3 yolo5s.py image
```

برای مثال با اجرا روی عکس سمت چپ، عکس سمت راست تولید می‌شود که نوع شی مورد نظر روی کادر آن مشخص می‌شود:

۹.۳ تولید متن از تصویر

در این بخش نحوه استفاده و اجرای مدل انتخابی برای مسئله‌ی جذاب تولید متن توضیح از روی تصویر توضیح داده شده است. در این مسئله از مدل vit-gpt2-image-captioning استفاده شده است. این مدل ۹۴۰ مگابایت است و با توجه به مسئله‌ای که حل می‌کند حجم معقولی دارد. این مدل از دو مدل vit (vision transformer) و gpt2 استفاده شده است. مدل vit برای ایجاد یک بردار تعبیه (امبدینگ) از روی تصویر و مدل gpt2 برای تولید متن از روی بردار تعبیه استفاده می‌شود. هر دوی این مدل‌ها در بخش خود جزء قدرتمندترین مدل‌ها هستند و در زمان ارائه‌ی مقاله‌شان از پرچمداران بودند. در این مدل ابتدا مدل و توکنایزرش برای یک بار دانلود می‌شوند. سپس چک می‌شود که آیا جی‌پی‌یو موجود است یا خیر. سپس حداکثر تعداد توکن‌های کپشن تولیدی و همچنین تعداد beamها برای تولید متن مشخص می‌شوند. در ادامه برای تصویر ورودی ابتدا تبدیل به RGB انجام می‌شود و سپس فیچر ورودی استخراج شده و متن از روی آن تولید می‌شود. این متن که شامل token idها است به توکنایزر داده می‌شود تا از روی آن متن انگلیسی استخراج شود.

```
import sys

from transformers import VisionEncoderDecoderModel, ViTFeatureExtractor,
AutoTokenizer

import torch

from PIL import Image


# check if there is a input image

if len(sys.argv) < 2:

    sys.exit('No file input ERROR, give a file path!')
```

```
# load the model and tokenizer

model2 = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

feature_extractor = ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")


# check if there is a gpu

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model2.to(device)


max_length = 16

num_beams = 4

gen_kwargs = {"max_length": max_length, "num_beams": num_beams}

def predict_step(image_paths):

  # loop on multiple image inputs

  images = []

  for image_path in image_paths:

    i_image = Image.open(image_path)

    # convert to RGB if it is not.

    if i_image.mode != "RGB":

      i_image = i_image.convert(mode="RGB")


    images.append(i_image)


  # extract features with model

  pixel_values = feature_extractor(images=images, return_tensors="pt").pixel_values

  pixel_values = pixel_values.to(device)
```

```
  # generate the caption
  output_ids = model2.generate(pixel_values, **gen_kwargs)

  # decode the caption with tokenizer
  preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
  preds = [pred.strip() for pred in preds]
  return preds

# write the caption on file
output = predict_step([sys.argv[1]])
print(output[0])
f = open("output.txt", "w")
f.write(output[0])
f.close()
```

برای اجرای این کد ابتدا یک virtualenv ایجاد می‌کنیم تا کتابخانه‌های لازم را درون آن نصب کنیم:

```
pip install virtualenv
virtualenv myenv
source myenv
```

سپس کتابخانه‌های زیر باید نصب شود.

```
pip install transformers
pip install torch
pip install pillow
```

با اجرای برنامه با دستور زیر ابتدا فایل مدل برای یک دانلود می‌شود و سپس کد روی تصویر ورودی اجرا می‌شود.

```
python3 image_captioning.py image
```

مثلا برای تصویر زیر متن «a cat laying on top of a couch next to another cat» تولید می‌شود.