Artículo
Ariel Arias · 22 nov, 2022 Lectura de 7 min

IAM & IRIS / IRIS Health desde un mismo archivo YML

Disclosure Statement: Sugerencias para relalizar pruebas en ambientes usados para demostraciones o desarrollos, no en ambientes productivos.

Caso de uso: teniendo IAM, lo ejecutamos desde un archivo YML, y necesitamos que se conecte a una Instancia IRIS en seguida, pero IRIS tiene deshabilitado el usuario IAM y la aplicación IAM.

Alternativa: Modificamos el archivo YAML para agregar la imagen IRIS / IRIS for Health que queremos utilizar, nos aseguramos que usen la misma red; ya sea que la hemos definido previamente y queremos acoplar estas nuevas imágenes a esta red, o creamos una nueva para aislar estas nuevas imágenes en un ambiente propio; y lo principal: ejecutamos un script para habilitar el usuario y la aplicación IAM de modo que al iniciar quede todo el ambiente operativo.

Requisito: Acceso a WRC para descarga de imagen de IAM (IRIS API Manager) Distribution page

PASO A PASO

Lo primero es comentar que la primera vez que quise usar el script que viene con el mismo instalador, me encontré con dos inconvenientes (para mi, es probable que quien tiene mayor conocimiento de Docker y Kubernetes no lo vea como inconveniente) y es que, por un lado, quería conectar este IAM con una instancia de IRIS que ya estaba en ejecución y con algunas API desarrolladas, entonces esperaba que fuera "ejecutar y conectar", y me percaté que:

- El usuario IAM y la aplicación IAM de mi instancia están deshabilitados.

- Al momento de ejecutar con "docker compose up -d", se crea una nueva red con el nombre "scriptsdefault", entonces si quiero que se comunique con mi instancia (otra imagen docker de IRIS en otra red llamada "irisenv-net" debo ejecutar unos pasos adicionales.

Una alternativa que encontré y me llevó a escribir este artículo, es, primero crear una red en docker que será utilizada en adelante por mis imagenes

```
docker network create -d bridge irisenv-net
```

Y luego modificar el docker-compose.yml para utilizar una red ya creada por docker y que usan otras imagenes en ejecución, para ello, agregué al archivo lo siguiente:

```
networks:
  irisenv-net:
    external: true
```

Con esto, las imagenes que usen la red "irisenv-net" se comunicarán entre ellas sin pasos adicionales. Entonces para cada imagen del archivo, agregué también:

```
networks:
      - irisenv-net
```

Me quedaba pendiente lo del usuario y aplicación IAM, que al ser una instancia ya en ejecución debía ejecutar manualmente tanto la habilitación como el establecer la contraseña que usaría mi usuario IAM, pues hay que pasarle ese parámetro al script.

Pero, ¿Si no está en ejecución?, ¿Puedo desde un mismo script levantar una imagen de IRIS, al mismo tiempo que levanto el IAM y que queden automáticamente conectadas y además el IAM tome la licencia existente en IRIS?: Mi respuesta fue SI. De la siguiente forma:

- Agregar en un mismo archivo (docker-iris.yml por ejemplo) la llamada a la imagen de IRIS que voy a utilizar, luego las que requiere IAM y las de IAM debo hacerlas depender de IRIS, entonces es un paso de copiar y pegar y modificar para dejar todo establecido.

Hasta acá tenemos un archivo que al ser ejecutado iniciará 4 imagenes: IRIS, IAM-MIGRATION, IAM y DB

Entonces, ¿Cómo resolvemos que al iniciar la instancia de IRIS, además de cargar la licencia, habilite el usuario IAM con una contraseña que conozco y puedo pasarle como parámetro al IAM, y además se habilite la aplicación IAM?. Respuesta: con un script que se ejecute al momento de iniciar la imagen de IRIS, es decir, tenemos un archivo docker llamado "irisdpfile" que contiene el detalle de la imagen que usaremos y agregamos la llamada al script.

El script que vamos a utilizar:

```
zn "%SYS"
Do ##class(Security.Users).UnExpireUserPasswords("*")
set st=$SYSTEM.Security.ChangePassword("IAM","iampassword")
set iamuser = ##class(Security.Users).%OpenId("iam")
set iamuser.Enabled = 1
set iamuser.AccountNeverExpires = 1
Set st = iamuser.%Save()
Set iamapp = ##class(Security.Applications).%OpenId("/api/iam")
Set iamapp.Enabled = 1
Set st = iamapp.%Save()
zn "USER"
```

Si vemos el detalle, en este archivo, habilitamos los usuarios que puedan estar expirados (por ejemplo el "superuser"); luego establecemos una nueva password para el usuario IAM y finalmente habilitamos la aplicación "/api/iam".

Este archivo, lo guardé en el mismo directorio desde donde estoy ejecutando el "docker-iris.yml" y lo nombré: "enableirisuser.script"

De esta forma, desde el dockerfile de IRIS (irisdpfile) agregué las líneas:

```
USER irisowner

COPY ./enableirisuser.script /tmp/enableirisuser.script

RUN iris start IRIS \
    && iris session IRIS < /tmp/enableirisuser.script \
    && iris stop IRIS quietly

USER irisowner
```

Nota: Deben quedar en una misma línea de ejecución el iniciar la instancia y llamada al script, me pasó que quise

hacer todo por separado y no me dio resultado.

Así, logré desde un mismo archivo, levantar una instanacia de IRIS y de IAM, donde IAM lograba comunicarse con IRIS sin problema y dejar en seguida habilitada la licencia para poder crear workspaces, ambiente de desarrolladores, y todo lo que sea necesario desde IAM.

```
docker-compose -f docker-iris.yml up -d
```

Les comparto cómo quedó mi archivo yml finalmente:

```
version: "3.7"

services:
  irishost:
    depends_on:
      - irishealth
    build:
      context: .
      dockerfile: irisdpfile
    container_name: irishost
    volumes:
    - type: bind
      source: ./Licencias/
      target: /licencias
    - type: bind
      source: ../sharingweb/T2017/
      target: /common_shared
    - type: bind
      source: ./wwwshared/
      target: /wwwfiles
    command: --key /licencias/iris.key  --check-caps false
    environment:
      IRIS_MASTER_HOST: APPTNOTSrv # DNS based on the name of the service!
      IRIS_MASTER_PORT: 51773
      IRIS_MASTER_USERNAME: SuperUser
      IRIS_MASTER_PASSWORD: SYS
      IRIS_MASTER_NAMESPACE: APPINT
      IRIS_USERNAME: SuperUser
      IRIS_PASSWORD: sys
      WEBGW_HOST: webgateway2022
      IRIS4H_HOST: irishealth
    ports:
    - "3071:1972" # 51773 is the superserver default port
    - "3072:52773" # 52773 is the webserver/management portal port
    - "3171:53773" # 53773 is the JDBC Gateway port
    tty: true
    links:
      - irishealth:irishealth
    networks:
      - irisenv-net
  iam-migrations:
    depends_on:
      - db
    image: intersystems/iam:2.8.1.0-3
    container_name: iam-migrations
    hostname: iam-migrations
    command: kong migrations bootstrap
```

```
    environment:
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-iam}
      KONG_PG_HOST: db
      KONG_PG_PASSWORD: ${KONG_PG_PASSWORD:-iam}
      KONG_PG_USER: ${KONG_PG_USER:-iam}
      KONG_CASSANDRA_CONTACT_POINTS: db
      ISC_IRIS_URL: 'IAM:iampassword@irishost:52773/api/iam/license'
      ISC_CA_CERT: ${ISC_CA_CERT}
    restart: on-failure
    links:
      - db:db
    restart: on-failure
    networks:
      - irisenv-net
  iam:
    depends_on:
      - irishost
      - db
    image: intersystems/iam:2.8.1.0-3
    container_name: iam
    hostname: iam
    environment:
      KONG_ADMIN_ACCESS_LOG: /dev/stdout
      KONG_ADMIN_ERROR_LOG: /dev/stderr
      KONG_ADMIN_LISTEN: '0.0.0.0:8001'
      KONG_ANONYMOUS_REPORTS: 'off'
      KONG_CASSANDRA_CONTACT_POINTS: db
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-iam}
      KONG_PG_HOST: db
      KONG_PG_PASSWORD: ${KONG_PG_PASSWORD:-iam}
      KONG_PG_USER: ${KONG_PG_USER:-iam}
      KONG_PROXY_ACCESS_LOG: /dev/stdout
      KONG_PROXY_ERROR_LOG: /dev/stderr
      KONG_PORTAL: on
      KONG_VITALS: on
      KONG_PORTAL_GUI_PROTOCOL: http
      KONG_PORTAL_GUI_HOST: 'localhost:8003'
      KONG_ADMIN_GUI_URL: 'http://localhost:8002'
      ISC_IRIS_URL: 'IAM:iampassword@irishost:52773/api/iam/license'
      ISC_CA_CERT: ${ISC_CA_CERT}
      irishost: '172.26.0.2'
    ports:
      - target: 8000
        published: 8000
        protocol: tcp
      - target: 8001
        published: 8001
        protocol: tcp
      - target: 8002
        published: 8002
        protocol: tcp
      - target: 8003
        published: 8003
        protocol: tcp
      - target: 8004
        published: 8004
        protocol: tcp
```

```
      - target: 8443
        published: 8443
        protocol: tcp
      - target: 8444
        published: 8444
        protocol: tcp
      - target: 8445
        published: 8445
        protocol: tcp
    restart: on-failure
    networks:
      - irisenv-net
    links:
      - db:db
  db:
    image: postgres:9.6
    container_name: db
    hostname: db
    environment:
      POSTGRES_DB: ${KONG_PG_DATABASE:-iam}
      POSTGRES_PASSWORD: ${KONG_PG_PASSWORD:-iam}
      POSTGRES_USER: ${KONG_PG_USER:-iam}
    volumes:
      - 'pgdata:/var/lib/postgresql/data'
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "${KONG_PG_USER:-iam}"]
      interval: 30s
      timeout: 30s
      retries: 3
    restart: on-failure
    networks:
      - irisenv-net
    depends_on:
      - irishost
    stdin_open: true
volumes:
  pgdata:
    external: true
networks:
  irisenv-net:
    external: true
```

Y el "irisdpfile":

```
FROM intersystems/iris:2022.1.0.209.0
LABEL maintainer="Ariel Arias <ariel.arias@intersystems.com>"

USER root

RUN apt-get -y update

# Install some aditional software may be needed later
RUN apt-get install -y vim nano net-tools unzip wget sudo iputils-ping links

RUN echo "FROM IRISDPFILE"

USER irisowner
```

```
COPY ./Licencias/iris4h2023.key /usr/irissys/mgr/iris.key

COPY ./enableirisuser.script /tmp/enableirisuser.script

RUN iris start IRIS \
    && iris session IRIS < /tmp/enableirisuser.script \
    && iris stop IRIS quietly

USER irisowner

# add default health check
HEALTHCHECK --interval=1m --timeout=10s --start-period=1m --retries=3 \
  CMD /irisHealth.sh || exit 1

USER irisowner

EXPOSE 1972
EXPOSE 52773
EXPOSE 53773
```

#Administración del sistema #Consejos y trucos #Despliegue #Docker #InterSystems API Manager (IAM) #Portal de ideas de InterSystems #InterSystems IRIS #InterSystems IRIS for Health