KPMG Applcation Migration Workshop

These are the instructions for the workshop to migrate a Python application to a container environment. Most of the exercises will we walk through first and then run at your own pace.

The application requires several external services. Once of these will require mock implementations. You can duplicate the work done during the workshop to build a local Docker based test environment.

The using a browser to tes the code it is advisable to open the developer console from the view developer menu to see any issues.

Workshop Setup

These steps have already been done for the DaDesktop environment.

Download and install Docker Engine.

The the student user access to Docker by adding the docker group.

```
sudo usermod -a -G docker student
```

Log out and log in again for the change to take effect.

Download and install Visual Studio Code.

Download the application `TiCaR.zip` into `~/Downloads`.

Clone the course materials.

`git clone https://github.com/dr-phill-edwards/KPMG`.

End of setup.

Exercise 1: Set up GitHub replica for the application

Make sure that the course material is up to date.

```
cd ~/KPMG
git pull
```

Create an SSH key pair. Hit enter twice to have no passcode.

```
ssh-keygen -t rsa
ls -la ~/.ssh
```

Create a Docker volume for GitHub and a network for the workshop.

```
docker volume create github
docker network create workshop
```

Go to the course directory and copy the SSH public key.

```
cd ~/KPMG
cp ~/.ssh/id_rsa.pub GitHub
```

Look at the Docker build file `GitHub/Dockerfile`.

```
FROM alpine:latest
RUN apk update && apk add git openssh
RUN ssh-keygen -A
RUN adduser -D git && passwd -u git
RUN ssh-keygen -A
WORKDIR /home/git
RUN mkdir -m 700 .ssh
COPY id_rsa.pub .ssh/authorized_keys
RUN chown -R git:git .ssh
USER git
RUN git config --global init.defaultBranch master
RUN mkdir ticar && cd ticar && git init --bare
USER root
CMD ["/usr/sbin/sshd", "-D"]
```

Build the Docker image.

```
docker build -t github:latest GitHub
```

Look at the Docker Compose file `compose.yml`.

```
name: kpmg
services:
  github:
    image: github:latest
    ports:
      - "2222:22"
    volumes:
      - github:/home/git
    networks:
      - workshop
volumes:
  github:
    external: true
networks:
  workshop:
    external: true
```

Run the service and check that it is running.

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

SSH needs to be told how to connect the git repository. Create the file `~/.ssh/config`.

```
Host github.com
    User git
    Hostname localhost
    Port 2222
    PreferredAuthentications publickey
    PubKeyAuthentication yes
    IdentityFile ~/.ssh/id_rsa
```

Verify that that the password free SSH connection works. Use control-D to exit. The Git command should return nothing and no error.

```
ssh git@github.com
git ls-remote git@github.com:ticar
```

Unpack the TiCaR source code.

```
cd
unzip Downloads/TiCaR.zip
cd TiCar
```

It is a Git repository, but the files need to be committed checked in.

```
git add .
git commit -m "Initial checkin"
```

Change the remote origin to the new repository and push the code.

```
git remote remove origin
git remote add origin git@github.com:ticar
git push origin master
```

This repository can now be used to caompare changes in the code. Working changes can be comitted and pushed.

End of exercise 1.

Exercise 2: Set up MySQL

We are going to install MySQL in a Docker container.

First pull the Docker image from Docker Hub.

```
docker pull mysql:9
```

Create a data volume for MySQL.

```
docker volume create mysql
docker ps
```

A script `~/KPMG/bin/initdb` has been created to set the root password and create a user.

```
#! /bin/bash

docker run -d --name mysql \
    -e MYSQL_ROOT_PASSWORD=rootpw \
    -e MYSQL_USER=ticar -e MYSQL_PASSWORD=ticarpw \
    --network kpmgworkshop_default -h mysql \
    -p 3306:3306 -v mysql:/var/lib/mysql mysql:9
```

Feel free to change the user name or the passwords. Run the script.

```
initdb
```

Connect to the MySQL container.

```
docker exec -it mysql sh
```

Run the MySQL client and give the user permission to create datadase and tables. Type in the root password.

```
mysql -h localhost -u root -p mysql
mysql> grant all privileges on *.* to 'ticar'@'%';
mysql> flush privileges;
mysql> control-D
```

Verify that you can connect as the user. Enter the user's password. Create the TiCaR database.

```
mysql -h localhost -u ticar -p mysql
```

```
mysql> create database ticar_db;
mysql> control-D
```

Exit using control-D.

Stop the MySQL container now that the data volume has been initialised.

```
docker rm -f mysql
```

Edit the file `~/KPMG/compose.yml` and add entries for MySQL. The two space indentation is important.

```
name: kpmg
services:
  github:
    image: github:latest
    ports:
      - "2222:22"
    volumes:
      - github:/home/git
    networks:
      - workshop
  mysql:
    image: mysql:9
    ports:
      - 3306:3306
    volumes:
      - mysql:/var/lib/mysql
    networks:
      - workshop
volumes:
  github:
    external: true
  mysql:
    external: true
networks:
  workshop:
    external: true
```

Start the services.

```
cd ~/KPMG
docker compose down
docker compose up -d
docker ps
```

End of exercise 2.

Exercise 3: Configure Code

There are files in the `~/KPMG/TiCaR` directory to containerise the application. The file `docker.yaml` sets application environment settings. The database credentials should be obtained from a secrets manager.

```
DOCKER:
  SERVER_NAME: 'localhost:5000'
  DEBUG: True
  TESTING: False
  LOG_TO_ELK: False
  SQLALCHEMY_DATABASE_URI: "mysql+pymysql://ticar:ticarpw@mysql/ticar_db"
```

There is a `Dockerfile` for the Docker build.

```
FROM python:3-slim
RUN apt update && apt install -y python3-dev default-libmysqlclient-dev build-
essential pkg-config
RUN python -m pip install pip-tools my
WORKDIR /ticar
COPY . .
RUN pip-compile requirements.in
RUN mv requirements.txt req.txt && grep -v msgspec req.txt > requirements.txt
RUN FLASK_APP=ticar.app
ENV FLASK_ENV=docker
CMD ["flask","run","--host=0.0.0.0"]
```

Copy the files into the application directory. Some environment variables have been set to save typing.

```
cd ~/TiCaR
cp ~/KPMG/TiCaR/Dockerfile .
cp ~/KPMG/TiCaR/docker.yaml ticar/config
cp ~/KPMG/TiCaR/raw.html ticar/templates
cp ~/KPMG/TiCaR/static/* ticar/static
```

Delete the last two lines statrting `mysqlclient` from `requirements.in` and add an entry `crypography`.

Build the Docker image.

```
docker build -t ticar:latest .
```

Find the IP address of the MySQL server by examining the network. Look for the mysal container and its `ipnet`. It should be something like 10.89.0.3. Run the container interactively.

```
docker run -it --rm --network workshop --add-host mysql:10.89.0.3 ticar:latest sh
```

Create and migrate the database tables.

```
flask db upgrade head
control-D
```

Edit the file `~/KPMG/compose.yml` and add entries for TiCaR. The two space indentation is important.

```
name: kpmg
services:
  github:
    image: github:latest
    ports:
      - "2222:22"
    volumes:
      - github:/home/git
    networks:
      - workshop
  mysql:
    image: mysql:9
    ports:
      - 3306:3306
    volumes:
      - mysql:/var/lib/mysql
    networks:
      - workshop
  ticar:
    image: ticar:latest
    ports:
      - 5000:5000
    networks:
      - workshop
volumes:
  github:
    external: true
  mysql:
    external: true
networks:
  workshop:
    external: true
```

Start the services.

```
cd ~/KPMG
docker compose down
docker compose up -d
```

```
docker ps
```

Check the application logs.

```
podman compose logs ticar
```

See that it was unable to connect to ElasticAPM. Implementing. afake service is out of scope for today. We will diable the service.

Open Visual Studio Code and go to **File → Open Folder...**. Select the **TiCaR** folder and open it.

Open the file `ticar/app.py` and add modify the if statement on line 44 to fix the issue.

```
    if app.config["ENVIRONMENT"] not in ["testing", "jenkins", "docker"]:
        apm = ElasticAPM()
        apm.init_app(app)
        with app.app_context():
            app.ldap_cache = LDAPCache(app.app_context())
```

Rebuild the image.

```
cd ~/TiCar
docker build -t ticar:latest .
```

Restart the services.

```
cd ~/KPMG
docker compose down
docker compose up -d
docker ps
```

Check the application logs.

```
podman compose logs ticar
```

Verify that the web server is running.

```
curl -i http://localhost:5000/health
```

You should see the health JSON message.

End of exercise 3.

Exercise 4: Reverse Proxy

Set up a remote proxy to forward to the application. Build the Docker image.

```
cd ~/KPMG
docker build -t nginx:proxy Nginx
```

Add the Nginx service to `compose.yml`.

```
  nginx:
    image: nginx:proxy
    ports:
      - "8080:80"
    networks:
      - workshop
```

Run the services.

```
docker compose down
docker compose up -d
docker ps
```

All of the containers should be running.

End of exercise 4.

Exercise 5: Problem Solving

There are going to be issues getting the application to work. We will work through the steps to solve issues. Do each of the steps immediately rather than at the end of the walkthough.

Make a call to the application's health check.

```
curl -i http://localhost:8080/ticar/health
```

What happened?

Look at the logs for the application.

```
cd ~/KPMG
docker compose logs ticar
```

What is the issue?

In VSCode open the file `ticar/config/docker.yaml`and fix the problem.

In a terminal window, rebuild the application and restart.

```
cd ~/TiCaR
docker build -t ticar:latest .
cd ~/KPMG
docker compose down
docker compose up -d
docker ps
```

Try the health check again.

```
curl -i http://localhost:8080/ticar/health
```

It should now be working.

Now try the application home page http://localhost:8080/ticar/.

What happened? Try using curl to see the response headers.

```
curl -i http://localhost:8080/ticar/
```

What happened? What is the problem?

Unfortunately, the application does a lot of redirects. This is going to require some code changes to fix.

In VSCode open `ticar/views/main.py`. Change line 18.

```
return redirect(os.environ.get("URI_PREFIX") + url_for("%s.landing" %
current_app.config["MODS"][0][0]))
```

Now try the application home page http://localhost:8080/ticar/ in a browser.

What happened?

Check the logs.

```
cd ~/KPMG
docker compose logs ticar
```

What is happening?

The security redirect URL is missing the prefix. We can fix this by adding a callback to the login manager to redirect correctly. In VSCode open `ticar/security.py`. After the `load_user()` function starting on line 44 add a new funtion.

```
    @login_manager.unauthorized_handler
    def unauthorized():
        prefix = app.config["URI_PREFIX"]
        return app.redirect(prefix + app.url_for('security.login', next=prefix
+'/dashboard/'))
```

Open `ticar/views/security.py`. Change the body of the login function starting on line 105 to add the next URI to the session.

```
    user_id = get_user_from_env()
    get_user(user_id)
    session["landing_url"] = request.args.get("next")
    landing_url = get_landing_url()
    return redirect(landing_url)
```

Look at the function `get_user_from_env()` starting on line 55. We need to add "docker" to the environment list to obtain a user name from the environment.

Make the changes, rebuild and restart.

Retry the URL is the browser. What happened? Check the application logs.

We have a redirect loop caused by an error!

Edit the file ticar/views/security.py`. Add `abort` to the Flask imports.

The `get_user()` function starting line 74 needs to return the user on line 93.

Change the body of the `login()` function starting on line 107.

```
    user_id = get_user_from_env()
    if not get_user(user_id):
        abort(403)
    session["landing_url"] = request.args.get("next")
    landing_url = get_landing_url()
    return redirect(landing_url)
```

Make the changes, rebuild and restart.

Retry the URL is the browser. What happened?

The redirect loop has been broken by returning a 403 error.

There is still a problem with LDAP being missing.

End of exercise 5.

Exercise 6: LDAP

We will walk through this and then do the exercise.

Build a Docker image containing an OpenLDAP server.

```
cd ~/KPMG
docker build -t ldap:latest OpenLDAP
```

Add an LDAP service to `compose.yml` before the application service entry.

```
  ldap:
    image: ldap:latest
    ports:
      - 389:389
    networks:
      - workshop
```

Restart compose. Check the logs for errors.

```
docker compose down
docker compose up -d
docker compose logs ldap
```

You should see LDAP entries have been created.

End of exercise 3.

Exercise 4: More Problem Solving

Do each of these steps immediately rather than at the end of the walkthough.

Enable LDAP in `ticar/app.py` by adding to the if statment on line 44.

```
    if app.config["ENVIRONMENT"] not in ["testing", "jenkins", "docker"]:
        apm = ElasticAPM()
        apm.init_app(app)
        with app.app_context():
            app.ldap_cache = LDAPCache(app.app_context())
    elif app.config["ENVIRONMENT"] == "docker":
        with app.app_context():
            app.ldap_cache = LDAPCache(app.app_context())
```

Open the file `ticar/utils/ldap_utils.py` and change the LDAP search filter string.

```
filter_string="(&(objectClass=person))",
```

Rebuild the application and restart. Try the URL in the browser. Check the application logs.

You should see that the can now login and the dashboard page gets displayed!

End of exercise 4.

If Time Allows

Fix the URLs in the dasboard landing pages `ticar/templates/landing.html` and `ticar/templates/dashboard/landing.html`. Prefix every `{{ url_for(⋯) }}` with `{{ config['URL_PREFIX'] }}`.

What else needs fixing?

Things To Do

- Web page links need URL prefixes.
- More url_for calls in Python need URI prefixes.
- Run the flask application in an Apache WSGI server.
- Some included CSS and JavaScript files are missing.
- Tickets go to a remote site o get rework template xlsx files.
- Login and logout need to be implemented instead of a fixed username.
- Elastic APM needs to be added.
- Implement JWT tokens and SSO.
- Implement getting xlsx files.
- Added users and groups to LDAP.
- Migrate another application to the mock framework.
- Migrate the application to a real staging environment.