GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Assessable Task 2nd Term

Computer Systems 23/24

Desarrollo de Aplicaciones Web

Aarón Martín Bermejo
Francisco Lifante
Álvaro Maceda

Cicles Formatius

License

TABLE OF CONTENTS

Instructions to deliver

Follow the next instructions to deliver your solution to this assessable activity:

- The file to deliver must be in **PDF format**
- The delivered file must be in english.
- The name of it should follow the next format: “Surname1_Surname2_Name_CS_2ndTerm.pdf” where Surname1 is your first surname, Surname2 is your second surname and Name Is your name.
- The delivered file must have a cover with, at least, your name, surnames and as title it should say “Computer Systems 2nd Term Assessable Activity”.
- Inside the delivered file remember specifying to which exercise and section you are answering.
- Read thoroughly the statement of each exercise.
- You must write all the commands **as text**. All the **screenshots** attached **will be removed** before grading the exercise.

As a reminder on the general instructions on assessable activities that are specified on the student’s guide:

- The delivered solution must be individual and original.
- Any copies or frauds detected will be graded with a 0.
- If there’s a suspicion about copies or frauds, the student can be summoned to a revision meeting where the teacher will ask questions to verify the authenticity of the solution.

Exercise 1

Currently you are working as a System Administrator at a small business which is growingly rapdily. At the beginning, we were working hard on the hardware side, mounting, installing and managing computers and devices of the company to support the growth of employees that HR was pushing.

However, after that stressful step, you've realized that after installing all the devices there are huge security issues inside the network.

- First, there were some IoT devices that were meant to be used internally only that were infected and were acting as zombies for a huge botnet.
- After solving that, a colleague without not too much computer knowledge opened an email containing an attachment called "Bills January 23-24.exe" and infected its computer. Luckily, only two more computers with windows were switched on and the infection could be contained easily.

After those two jumpscares, management approved time to split our current network into different subnets to make it more secure and also to support further staff increasements.

Currently our network hast the IP address block `192.168.10.0/24` and our current needs are:

- We have 7 IoT devices counting the 3 security cameras, 3 door sensors and 1 3D printer. All of them need an IP each.
- We have 12 people on the economical side (accountants, market researchers, sellers…). All of them need an IP each.
- We have 8 software developers and 2 system administrators. The software developers **need two IPs each** because they are working heavily on networking and the system administrators (us) need only one IP each.

Based on our current needs, management told us that:

- We will have 9 IoT devices as much.
- We will have 15 persons on the economical side tops.
- We will have 10 software developers and 2 system administrators.

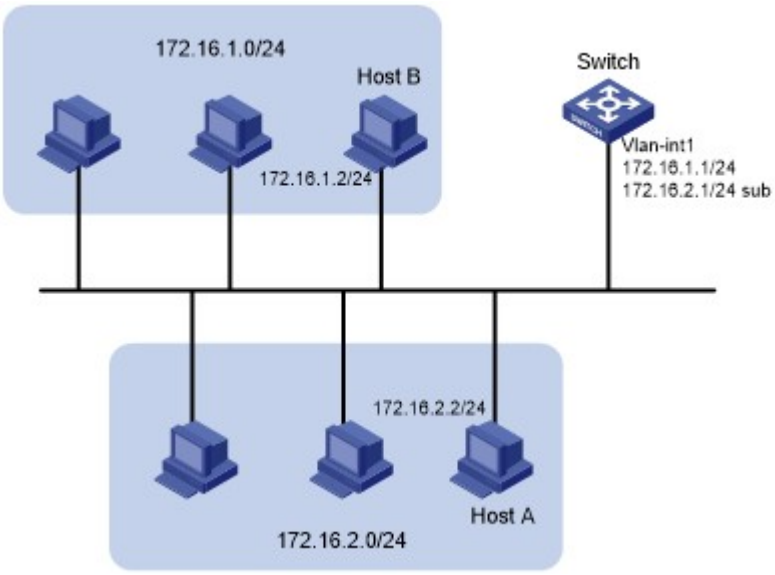
Taking that into account, we've decided to create 3 subnetworks.

Management asked us for the next things to approve the plan to divide the network.

Section a

Draw a diagram where the different networks are shown and the IPs are assigned for our **current needs**. They don't want to see more devices, although there should be space to accommodate the expansions.

You can create the network diagram with some tool like draw.io, packet tracer or something similar. Here's an example of what should be created:



The diagram should not be exactly this way, but in the diagram you need to specify the IPs of the devices, masks of the networks and the networks IPs.

Section b

Fill the next tables with the network information with the current IP needs.

Subnet	Network Address	Broadcast Address	Mask	Number of usable hosts

And for **each** network:

Network X

Device	IP
Device XXX	
…	
Router	
Empty IPs range	

Exercise 2

In this exercise, you will launch a development environment to test a Flask application which makes use of a Redis server.

- Flask is a Python framework for developing web applications.
- Redis is a server that stores data on memory and is often used for caching data.

You will launch the two services separately and, once you are sure they are working, you will launch both of them together.

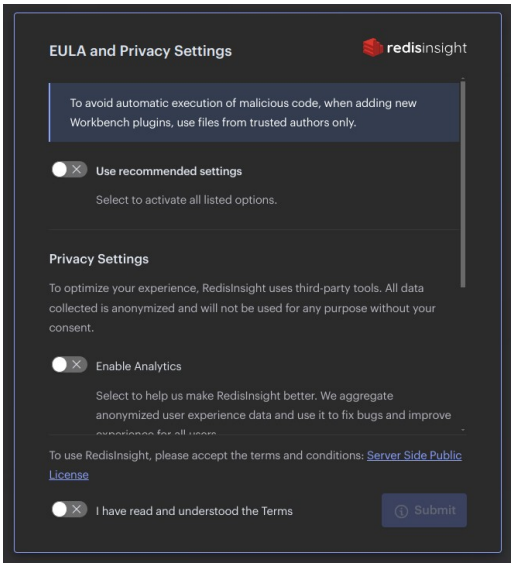
For the evaluation you only need to provide the commands and files requested in the last section. The other sections are meant to help you with the practice

Section a: Launch the Redis server

You can launch a Redis server locally using the `redis-stack` image: https://hub.docker.com/r/redis/redis-stack

If you launch a container with that image you will get two services: the Redis server, needed by the Flask app, and RedisInsight. The last one provides a web service for inspecting the contents of the Redis server.

- Launch a container based on the `redis-stack` image. The container should be deleted once stopped and map port `8001` to port `8001` of your machine. It must not map any other ports.
- Test that the container is working. To do so, access your `localhost` on port 8001: `http://localhost:8001`. You should see a screen like this one:



Section b: Run the Flask app

To test the Flask app, you will need to create a custom image based on `python:3.8-slim`. But first, you need to test the commands needed for running the Flask app in a standard container:

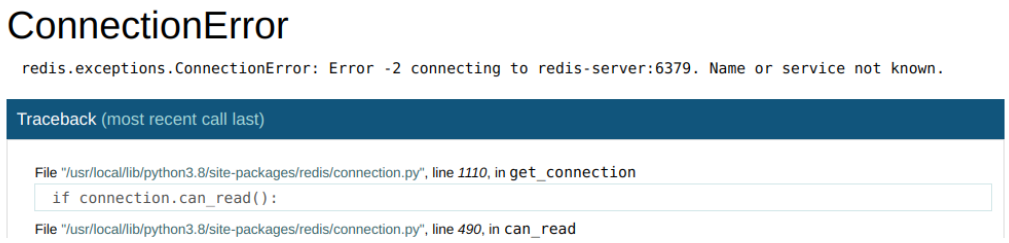
1) Copy `requirements.txt` and the `app` directory to a directory in your machine (this will be considered the base directory for the practice)
2) Run a container which binds your base directory to `/python` directory in the container. The container must map port `5000` (where the Flask app runs) to your machine's port `5000`. Launch the container in interactive mode running the `/bin/bash` command. This way, you will have a shell inside that container.
3) Prepare the machine to run the app. To do that, change to the `/python` directory inside the container and install the dependencies with the command: `pip install -r requirements.txt`
4) Change to `/python/app` directory and run the app with the command `flask run –host:0.0.0.0 --debug`. You should see the output of the Flask server:

```
 * Debug mode: on
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-211-379
 * Debug mode: on
```

After that, you should be able to access the Flask app `/hello` endpoint in your local machine: http://localhost:5000/hello:

Hello, World!

If you try to access the root endpoint (http://localhost:5000) you will get an error because the app can't connect to the Redis server:



Section c: Create Flask image

To avoid having to repeat all of the above each time we launch the Flask server you are going to create your own image.

1) Create a Dockerfile in the base directory. The Dockerfile must be based on the `python:3.8-slim` image and do the following:
   a) Copy the `requirements.txt` file to the container's root directory
   b) Run the command to install the dependencies:

      ```
      pip install --no-cache-dir -r /requirements.txt
      ```

   c) Set the environment variables. You can do that using `ENV` in the Dockerfile:

      ```
      ENV FLASK_APP=app.py
      ENV FLASK_RUN_HOST=0.0.0.0,
      ENV FLASK_ENV=development
      ENV REDIS_SERVER=FILL-LATER
      ```

   d) Run the command `flask` with parameters `run` and `–debug` using the exec notation.
2) Build an image using that Dockerfile
3) Run a container based on the created image which binds your base directory to `/python` directory in the container. The container must map the port `5000` (where the Flask app runs) to your machine's port `5000`
4) Test that the container is working, by accessing the `/hello` endpoint: http://localhost:5000/hello
5) Modify the source code on your machine (change `Hello, World!` on `app.py` line 30 for whatever you want) and test that the new content is returned by the app refreshing the browser.

Section d: Launch the two containers together

For the practice, you must launch the two containers **in a private network**.

1) Run the Redis container in detached mode, mapping the port `8001` to the port `8001` in your machine. It must not map any other ports. The container should be removed when stopped.
2) Modify the Dockerfile and change `REDIS_SERVER`. The value of this variable must be the hostname of the Redis server.
3) Build the image of the Flask server
4) Launch the flask server binding your `/app` directory to the `/app` directory in the container. The container must run in detached mode and map port `5000` to port `5000` in your machine.

After that, you should be able to access the root endpoint and see an incrementing counter: http://localhost:5000

Deliverables

- Explain and enumerate the commands used in this section
- Write the final contents of your Dockerfile.