

# Assignment 1

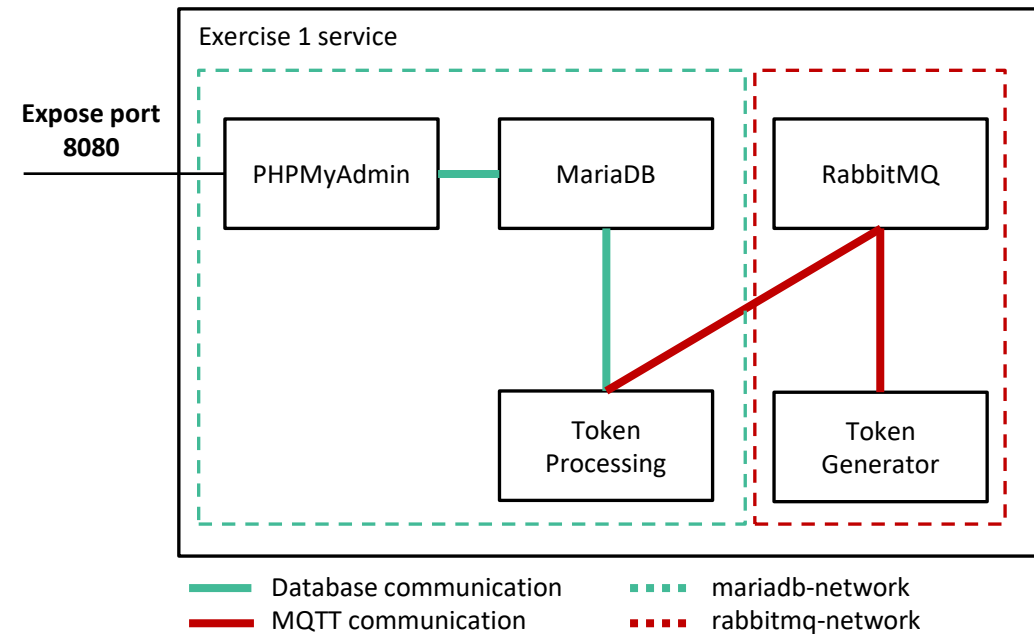
---

COMPUTING SYSTEMS AND INFRASTRUCTURES

*(SISTEMAS E INFRAESTRUTURAS DE COMPUTAÇÃO)*

# Assignment 1

- The objective of this exercise is to implement a proof of concept for an application that will generate secret tokens for authentication, similar to the well-known app "chave movil"
- The secret token will be generated by the Token Generator (app-pub.py) and published in a message broker (RabbitMQ). An application, Token Processing (app-sub.py), will consume the secret token and storage in a DB for auditing purposes
- The database can be browsed through a web browser through PHPMyAdmin
- Create a **Docker compose file** that is able to instantiate the required goals by instantiating and interconnecting the required services:
  - Message broker (RabbitMQ)
  - Database (MariaDB)
  - PHPMySQL
  - Token Generator – app-pub.py
  - Token Processing – app-sub.py



# Token Generator

---

- The token generator application has to be implemented in Python and fulfill the following requirements:
  - Generate a hexadecimal token using the library **secrets**
  - The length of the tokens is determined by an environmental variable in the container (**LENGTHTOKEN**)
  - The number of tokens that should generated is also determined by an environmental variable in the container (**NUMTOKENS**)
  - The tokens generated must be published in the channel **/sic/tokens**

# Token Processing

---

- The token processing application has to be implemented in Python and fulfill the following requirements:
  - Connect to the MariaDB and use the DB “**sic**”. This DB must be created when the MariaDB container is run
  - Create, if required, the table “**sic**” with the structure:
    - id (int, auto increment)
    - date (timestamp)
    - channel (varchar)
    - value (varchar)
  - Subscribe to the MQTT channel **/sic/#** (subscribe to /sic and all child channels)
  - Insert into the database a new record with the current date and time, the channel name and the token received

# About the dockers (1/2)

---

- Rely only on official and latest images to develop this exercise (**RabbitMQ, MariaDB, Python, PHPMyAdmin**)
- RabbitMQ requires the MQTT plugin to be enabled
- Python requires the Paho MQTT (**PIP install to support the MQTT message broker**)
- Python MySQL Connector Python (**PIP install to support MySQL connections**)
- Create the required Dockerfile to build custom images
- Evaluate the usage of volumes and networks
- Address the dependencies requirements of the containers
- The compose file must be self-contained → **Do not depend on existing custom images, volumes, or networks**

# About the dockers (2/2)

---

- The DB must be persistent by using a volume **mariadb-data-sic**
- The implementation of your solution must consider network isolation, for this consider two different networks for your containers
  - **rabbitmq-network**
  - **mariadb-network**
- A container in your solution must be executed if and only if its dependencies are already satisfied
- Be careful with the syntax in your files and remember to follow the instructions described here
- All the debug information generated by your applications must be published in the channel **/sic/log**

# Example of python MQTT connection (publish & subscribe)

---

```
import paho.mqtt.client as mqtt

def on_disconnect(client, userdata, rc):
    print("Disconnected with result code "+str(rc))

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

def on_publish(client, userdata, result):
    print("Message published: "+str(result))

if __name__ == '__main__':
    client= mqtt.Client("sic-pub")
    client.on_publish = on_publish
    client.on_connect = on_connect
    client.on_disconnect = on_disconnect

    client.connect("MQTT_SERVER",1883)

    client.publish("/sic", "MSG")
```

```
import paho.mqtt.client as mqtt

def on_disconnect(client, userdata, rc):
    print("Disconnected with result code "+str(rc))

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("/sic/#")

def on_subscribe(client, userdata, mid, granted_qos):
    print("Subscribed")

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

if __name__ == '__main__':
    client= mqtt.Client("sic-sub")
    client.on_message = on_message
    client.on_subscribe = on_subscribe
    client.on_connect = on_connect
    client.on_disconnect = on_disconnect

    client.connect("MQTT_SERVER",1883)

    client.loop_forever()
```

# Example of python MySQL connection

---

```
import mysql.connector
from mysql.connector import errorcode

if __name__ == '__main__':
    try:
        cnx = mysql.connector.connect(user='root',
password='my-secret-pw', host=MARIADB_SERVER',
database='sic')
        print("Connected to database")

        cursor = cnx.cursor()
    except mysql.connector.Error as err:
        print("Error connecting to database")
        exit(0)
```

```
try:
    TABLE = (
        "CREATE TABLE `sic` ("
        "  `id` int(11) NOT NULL AUTO_INCREMENT,"
        "  `date` TIMESTAMP NOT NULL,"
        "  `client` varchar(64) NOT NULL,"
        "  `value` int(11) NOT NULL,"
        "  PRIMARY KEY (`id`)"
        ") ENGINE=InnoDB")
    cursor.execute(TABLE)
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
        print("Table already exists")
    else:
        print("Error creating table")
```

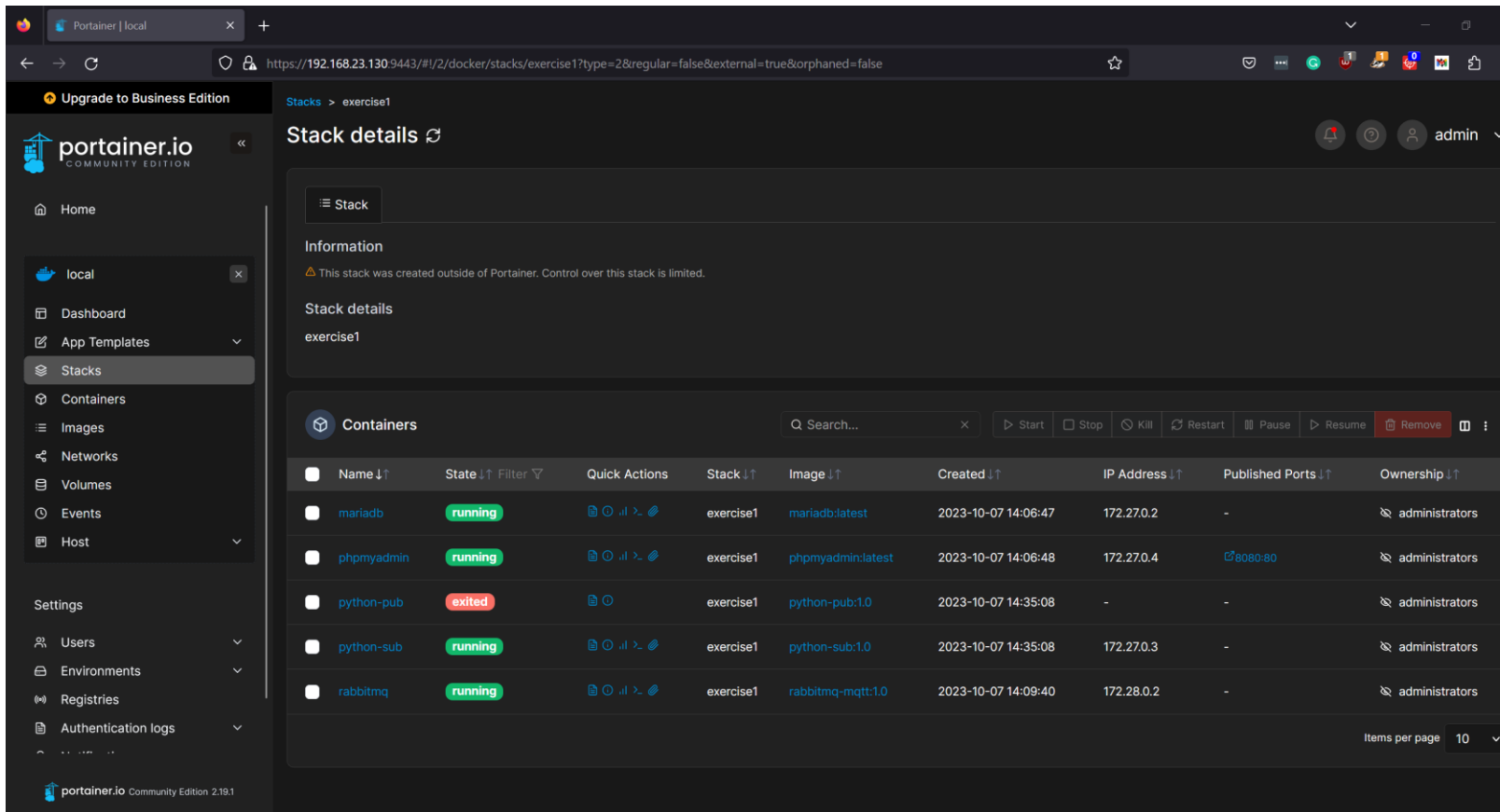


# Examples of the expected outcome (1/7)

---

- View of the directory to be submitted to the InforEstudante (.zip)
  - exercise1
    - compose.yaml
    - Dockerfile
  - pub
    - app-pub.py
    - Dockerfile
  - sub
    - app-sub.py
    - Dockerfile

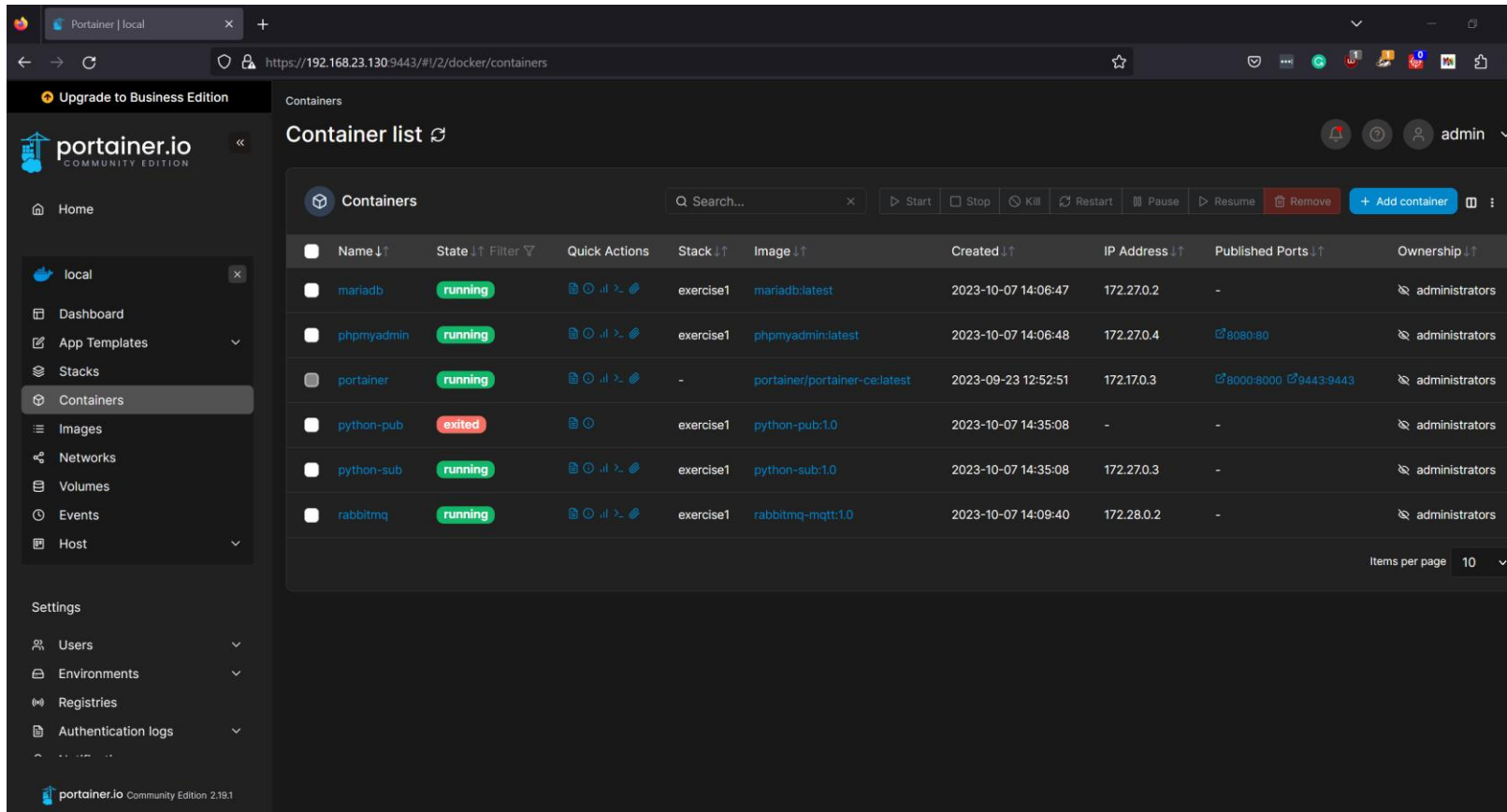
# Examples of the expected outcome (2/7)



The screenshot displays the Portainer.io interface for a stack named 'exercise1'. The left sidebar shows the navigation menu with 'Stacks' selected. The main panel shows the 'Stack details' for 'exercise1', which was created outside of Portainer. Below this, a table lists the containers in the stack:

Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
mariadb	running	[Icons]	exercise1	mariadb:latest	2023-10-07 14:06:47	172.27.0.2	-	administrators
phpmyadmin	running	[Icons]	exercise1	phpmyadmin:latest	2023-10-07 14:06:48	172.27.0.4	8080:80	administrators
python-pub	exited	[Icons]	exercise1	python-pub:1.0	2023-10-07 14:35:08	-	-	administrators
python-sub	running	[Icons]	exercise1	python-sub:1.0	2023-10-07 14:35:08	172.27.0.3	-	administrators
rabbitmq	running	[Icons]	exercise1	rabbitmq-mq:1.0	2023-10-07 14:09:40	172.28.0.2	-	administrators

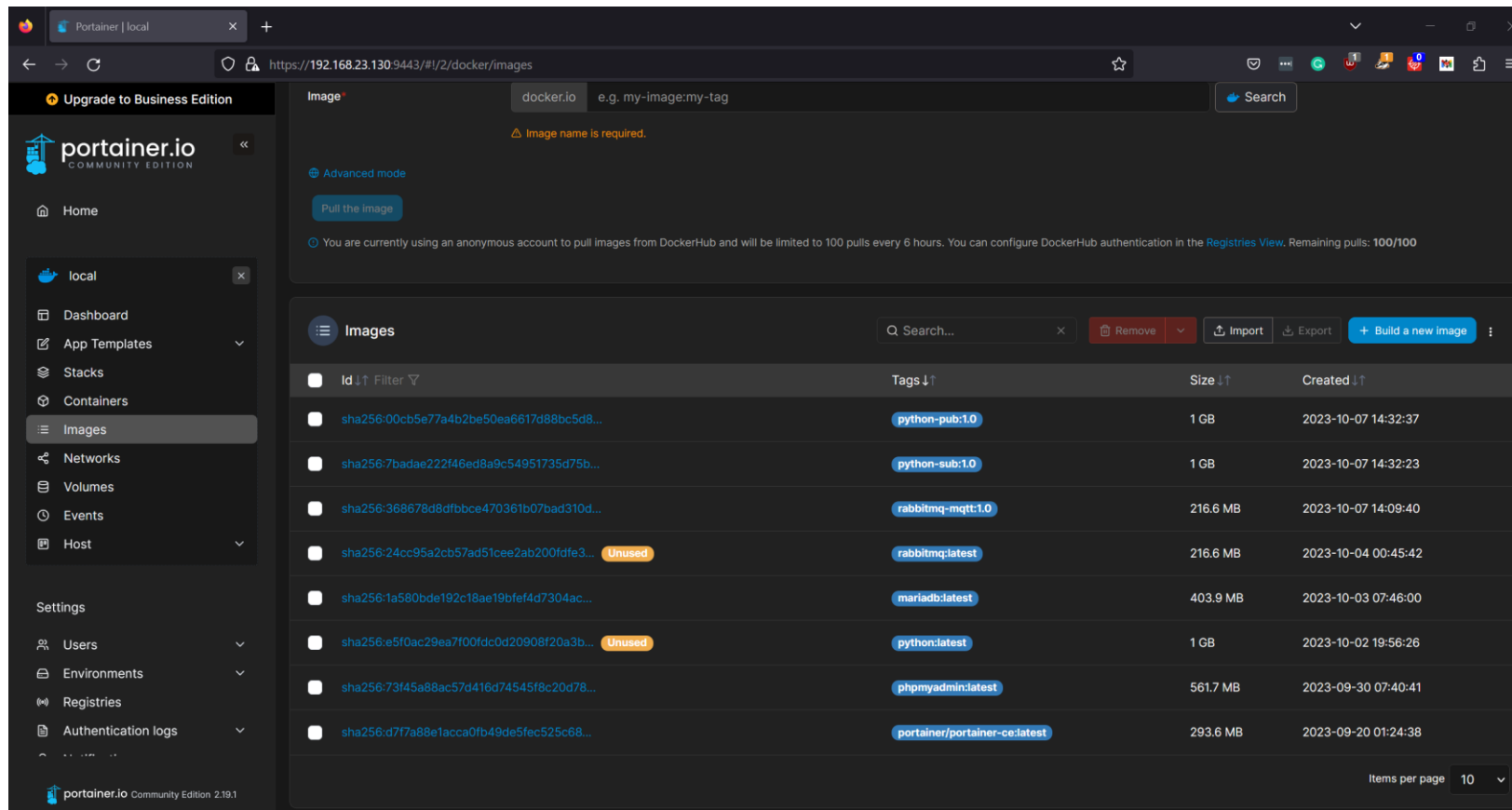
# Examples of the expected outcome (3/7)



The screenshot displays the Portainer web interface, specifically the 'Containers' section. The left sidebar shows the navigation menu with options like Home, local, Dashboard, App Templates, Stacks, Containers (selected), Images, Networks, Volumes, Events, Host, and Settings. The main area shows a 'Container list' table with columns for Name, State, Quick Actions, Stack, Image, Created, IP Address, Published Ports, and Ownership. The table lists several containers, including mariadb, phpmyadmin, portainer, python-pub, python-sub, and rabbitmq. The portainer container is highlighted with a blue background.

Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
mariadb	running	[Icons]	exercise1	mariadb:latest	2023-10-07 14:06:47	172.27.0.2	-	administrators
phpmyadmin	running	[Icons]	exercise1	phpmyadmin:latest	2023-10-07 14:06:48	172.27.0.4	8080:80	administrators
portainer	running	[Icons]	-	portainer/portainer-ce:latest	2023-09-23 12:52:51	172.17.0.3	8000:8000 9443:9443	administrators
python-pub	exited	[Icons]	exercise1	python-pub:1.0	2023-10-07 14:35:08	-	-	administrators
python-sub	running	[Icons]	exercise1	python-sub:1.0	2023-10-07 14:35:08	172.27.0.3	-	administrators
rabbitmq	running	[Icons]	exercise1	rabbitmq-mq:1.0	2023-10-07 14:09:40	172.28.0.2	-	administrators

# Examples of the expected outcome (4/7)

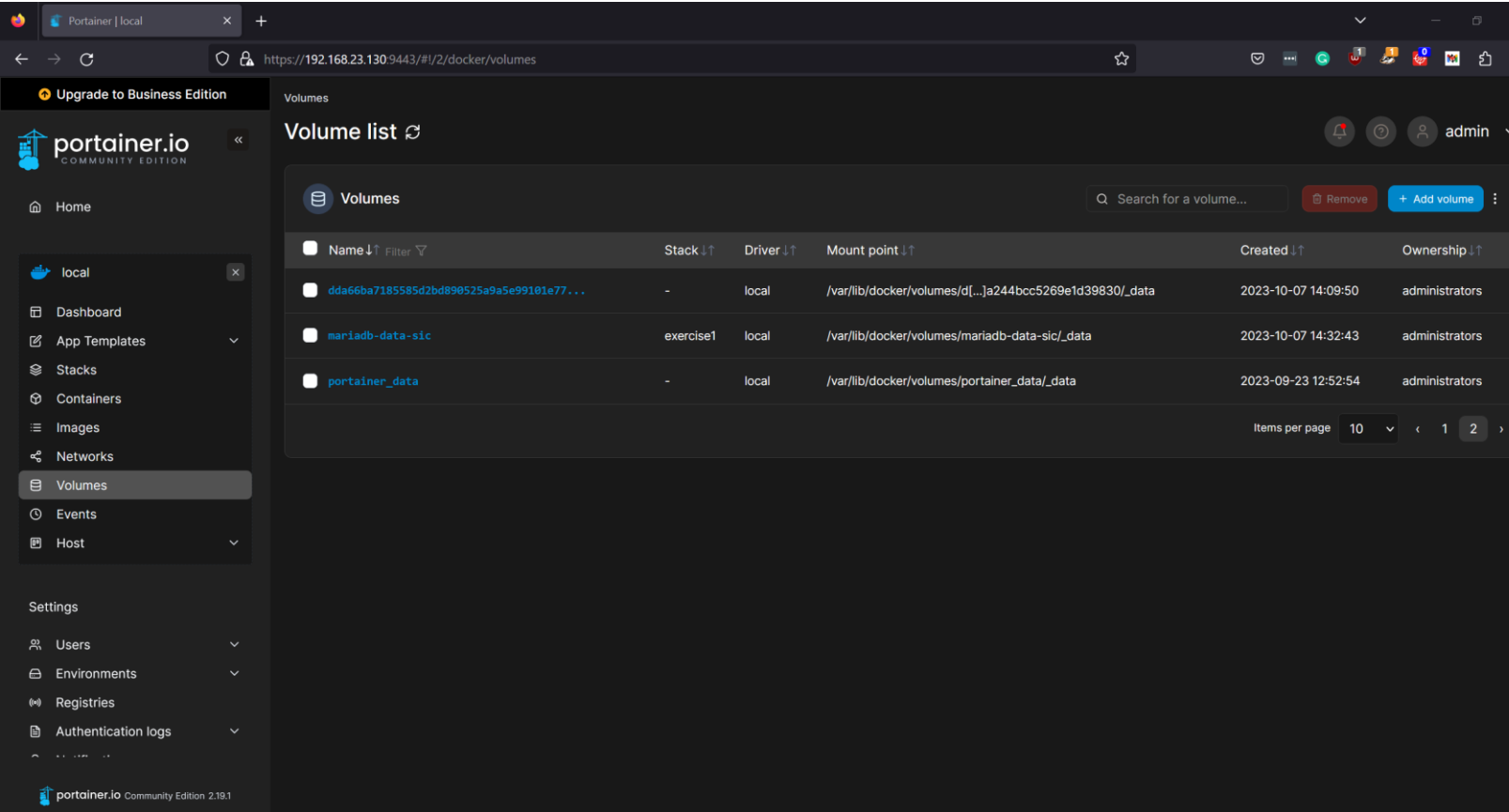


# Examples of the expected outcome (5/7)

The screenshot displays the Portainer.io Community Edition web interface. The left sidebar contains navigation links: Home, local (selected), Dashboard, App Templates, Stacks, Containers, Images, Networks (highlighted), Volumes, Events, Host, Settings, Users, Environments, Registries, and Authentication logs. The main content area is titled 'Network list' and shows a table of Docker networks. The table has columns for Name, Stack, Driver, Attachable, IPAM Driver, IPv4 IPAM Subnet, IPv4 IPAM Gateway, IPv6 IPAM Subnet, IPv6 IPAM Gateway, and Ownership. The networks listed are: bridge (System), host (System), mariadb-network, none (System), and rabbitmq-network. The 'bridge' and 'host' networks are marked as 'System' and have 'public' ownership. The 'mariadb-network' and 'rabbitmq-network' are user-defined and have 'administrators' ownership. The 'none' network is also marked as 'System' and has 'public' ownership. The interface includes a search bar, a 'Remove' button, and an 'Add network' button. The bottom right corner shows 'Items per page' set to 10.

Name	Stack	Driver	Attachable	IPAM Driver	IPv4 IPAM Subnet	IPv4 IPAM Gateway	IPv6 IPAM Subnet	IPv6 IPAM Gateway	Ownership
bridge	-	bridge	false	default	172.17.0.0/16	172.17.0.1	-	-	public
host	-	host	false	default	-	-	-	-	public
mariadb-network	exercise1	bridge	false	default	172.27.0.0/16	172.27.0.1	-	-	administrators
none	-	null	false	default	-	-	-	-	public
rabbitmq-network	exercise1	bridge	false	default	172.28.0.0/16	172.28.0.1	-	-	administrators

# Examples of the expected outcome (6/7)



The screenshot displays the Portainer.io web interface, specifically the 'Volumes' section. The left sidebar shows the navigation menu with 'Volumes' selected. The main content area, titled 'Volume list', contains a table of Docker volumes. The table has columns for Name, Stack, Driver, Mount point, Created, and Ownership. Three volumes are listed: 'dda66ba7185585d2bd898525a9a5e99101e77...', 'mariadb-data-sic', and 'portainer\_data'. Each volume is associated with a 'local' driver and a specific mount point. The 'Created' column shows timestamps, and the 'Ownership' column lists 'administrators'. At the bottom right of the table, there is a pagination control showing 'Items per page 10' and page numbers '1' and '2'.

Name	Stack	Driver	Mount point	Created	Ownership
dda66ba7185585d2bd898525a9a5e99101e77...	-	local	/var/lib/docker/volumes/d[...a244bcc5269e1d39830]/_data	2023-10-07 14:09:50	administrators
mariadb-data-sic	exercise1	local	/var/lib/docker/volumes/mariadb-data-sic/_data	2023-10-07 14:32:43	administrators
portainer_data	-	local	/var/lib/docker/volumes/portainer_data/_data	2023-09-23 12:52:54	administrators

# Examples of the expected outcome (7/7)

The screenshot displays the phpMyAdmin web interface. The left sidebar shows the database structure with a tree view containing 'information\_schema', 'mysql', 'performance\_schema', 'sic', and 'sys'. The 'sic' database is selected, and the 'Table: sic' is viewed. The main area shows a table with 24 rows. The table structure is defined by the query: `SELECT * FROM `sic``. The table has columns: `id`, `date`, `client`, and `value`. The data rows show a sequence of token generations, starting with 'Starting tokens generation!' and ending with 'Finishing the tokens generation!'. Each row includes a checkbox for selection, and links for Edit, Copy, and Delete. The table is sorted by `id` in ascending order.

	id	date	client	value
<input type="checkbox"/>	13	2023-10-07 13:34:11	/sic/log	Starting tokens generation!
<input type="checkbox"/>	14	2023-10-07 13:34:12	/sic/tokens	Token #0 was: 481a07
<input type="checkbox"/>	15	2023-10-07 13:34:13	/sic/tokens	Token #1 was: 9a92c6
<input type="checkbox"/>	16	2023-10-07 13:34:14	/sic/tokens	Token #2 was: 0d8015
<input type="checkbox"/>	17	2023-10-07 13:34:15	/sic/tokens	Token #3 was: 2a261f
<input type="checkbox"/>	18	2023-10-07 13:34:16	/sic/tokens	Token #4 was: f29608
<input type="checkbox"/>	19	2023-10-07 13:34:17	/sic/tokens	Token #5 was: e083b3
<input type="checkbox"/>	20	2023-10-07 13:34:18	/sic/tokens	Token #6 was: 60b13d
<input type="checkbox"/>	21	2023-10-07 13:34:19	/sic/tokens	Token #7 was: 88f79a
<input type="checkbox"/>	22	2023-10-07 13:34:20	/sic/tokens	Token #8 was: 147c34
<input type="checkbox"/>	23	2023-10-07 13:34:21	/sic/tokens	Token #9 was: f50fa2
<input type="checkbox"/>	24	2023-10-07 13:34:21	/sic/log	Finishing the tokens generation!

# Final remarks

---

- The assignment must be developed in groups of **2** students
- The assignment must be submitted in InforEstudante in a **.zip** file containing all required files
- Students must enroll in one of the available defense time slots available in InforEstudante
  
- Submission deadline: **28/10/2023**
- Defence: **30/10/2023**