UNIVERSIDADE LUSÓFONA

IT Infrastructure and Network Administration

Exercises

João Caldeira

September, 2024

IT Infrastructure and Networks Administration

João Caldeira, Universidade Lusófona, Lisboa, Portugal

16.12.2025

Índice

Lista de Figuras

Lista de Tabelas

Prefácio

Este documento é um guia prático, fruto da compilação de uma longa lista de exercícios com feramentas de gestão e administração de infraestrutura e redes IT ministradas nas aulas da mesma unidade curricular na **Universidade Lusófona**. Cobre aspectos sobre a instalação de ambientes de **Virtualização, Containers e Cloud**. Agrega desde exercícios introdutórios até tarefas mais complexas e demoradas, incluindo **criação de containers, configuração de rede, firewalls, CI/CD** e muito mais. É voltado principalmente para alunos que estão a começar a aprender a gerir infraestruturas IT, porém também pode ser utilizado por indivíduos com níveis mais elevados de experiência. Cada exercício terá a solução correspondente e eventualmente algumas alternativas para realizar a mesma tarefa.

Alguns exercícios podem começar por listar os objetivos de aprendizagem e os materiais de leitura necessários, com referências adicionais, caso se justifique. A inclusão de notas ajuda a apresentar os métodos e dar alguns exemplos, mas são menos detalhadas do que os materiais de leitura. Algumas notas são mensagens importantes para concluir as tarefas corretamente.

Espera-se que o público alvo seja não iniciante em linguagens de programação e que tenha algum conhecimento em algoritmos e experiência em codificação genérica.

Autor

João Caldeira. Professor Auxiliar na Universidade Lusófona e Investigador Associado na COPELABS em Lisboa, Portugal.

Licença

Part I.

Tools Setup

1. Docker

1.1. Download

O Docker pode ser instalado em formato desktop (apenas Windows ou macOS) ou em formato de linha de comando (Windows, macOS e Linux). Aceda à página do **Docker** e faça o **download** da versão correspondente ao seu sistema operativo, tal como demonstrado na Figura 1.1.

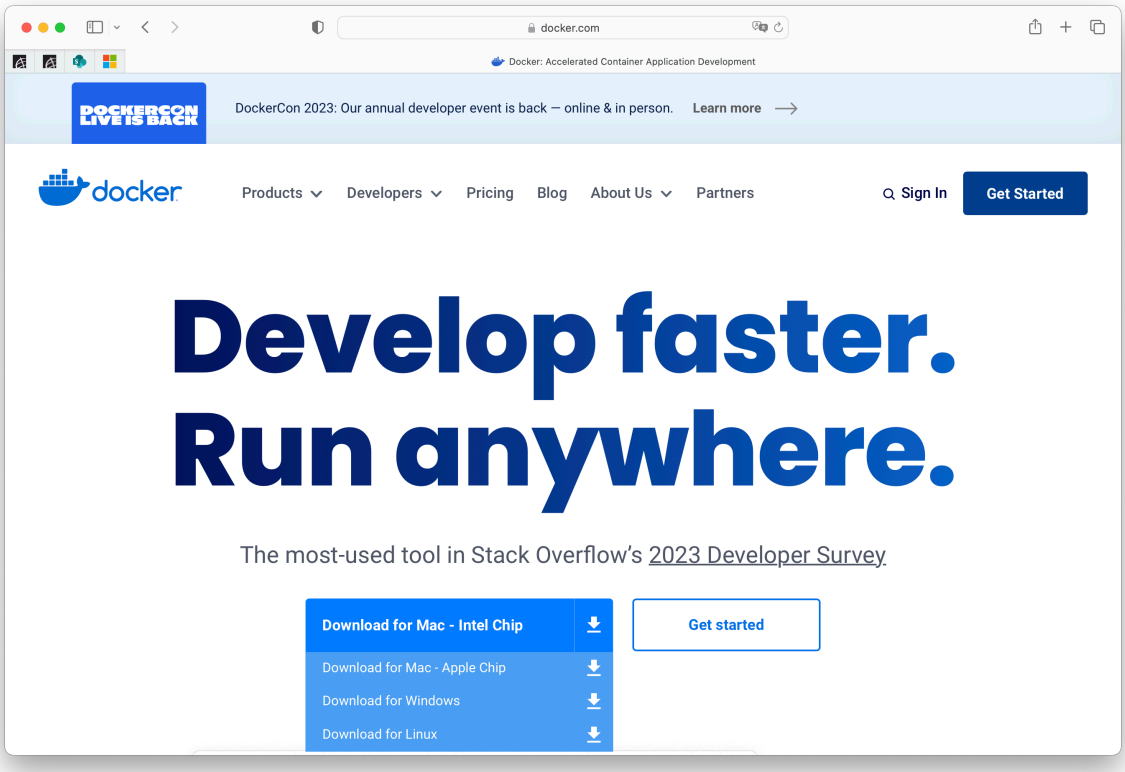

Figura 1.1.: Download de Docker

1.2. Instalação

1.2.1. Windows

> **Nota**
>
> **Instruções válidas apenas para Windows 8 ou posterior**

Obter o instalador da versão atual para Windows do Docker Desktop.

A instalação desta versão irá disponibilizar o cliente de linha de comandos e powershell.

> **Nota**
>
> **A instalação de Docker em ambiente Windows requer a activação da componente Windows System for Linux versão 2 (WSL2).**

Em caso de falha no WSL, consultar o Troubleshooting do Windows Subsystem for Linux | Microsoft Learn

1.2.2. macOS

> **Nota**
>
> **Instruções válidas apenas para MacOs**

Obter o instalador da versão atual para macOS do Docker Desktop.

Em macOS, também se pode instalar directamente em linha de comando utilizando semelhantes procedimentos que se indicam para Linux.

Os utilizadores de Docker em macOS devem ativar a seguinte opção nas configurações do Docker Desktop, tal como apresentado na Figura 1.2:

- Use Rosetta for x86/amd64 emulation on Apple Silicon
- Ativar e fazer Restart

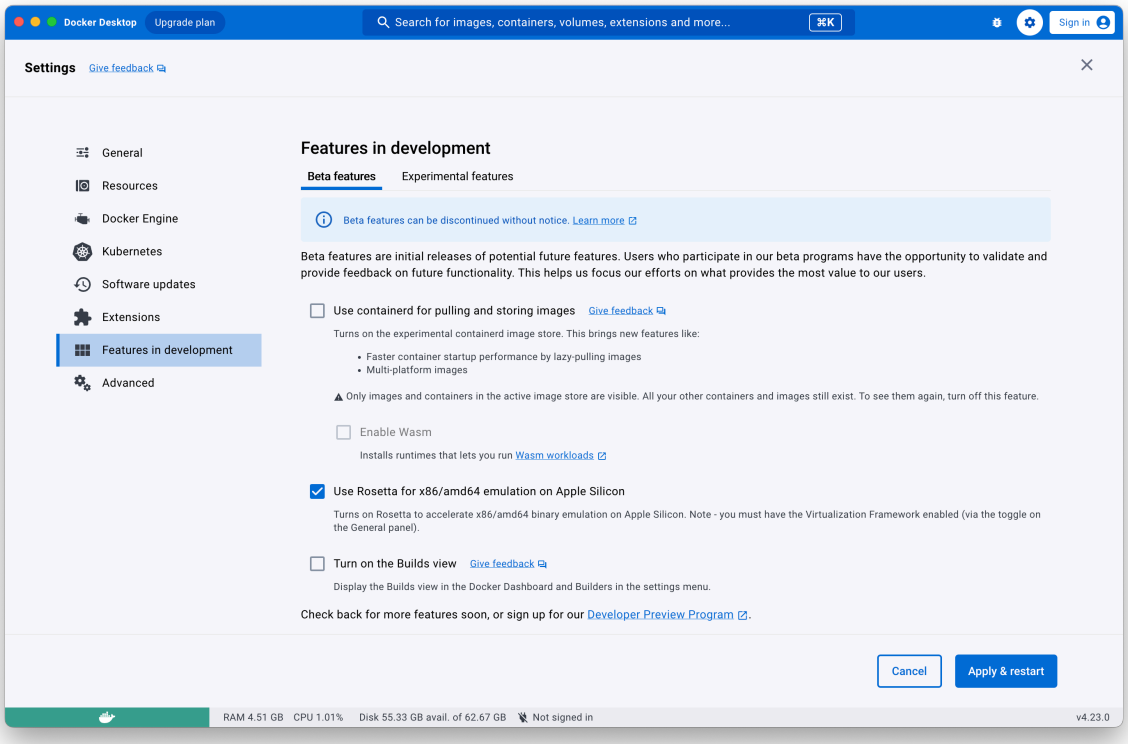

Figura 1.2.: Configuração Docker Desktop em macOS

1.2.3. Linux

> **Nota**
>
> **As indicações referem-se a instalação em Amazon Linux 2 (AWS)**

O processo pode variar em função da implementação Linux utilizada

- Ex.: Ubuntu.
- Actualizar pacotes, garantindo que o sistema operativo está actualizado (opcional)

```
sudo yum update -y
```

- Instalar o Docker Engine

```
sudo amazon-linux-extras install docker
```

- Iniciar o Docker

```
sudo service docker start
```

- Verificar o estado do serviço e versão

```
docker info
```

1.3. Docker Hub

1.3.1. Registo (opcional)

As instalações podem ser feitas de forma anónima ou em formato autenticado, via Docker Hub.

É recomendado que cada utilizador esteja registado.

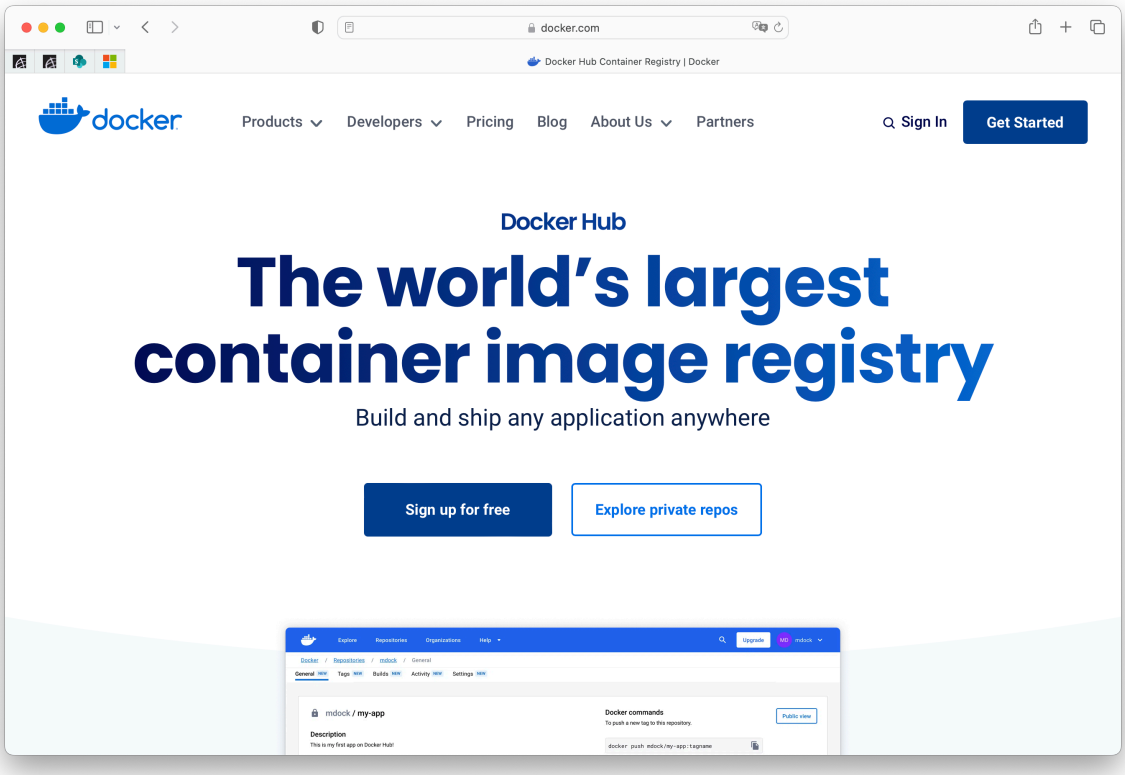

Figura 1.3.: Docker Hub

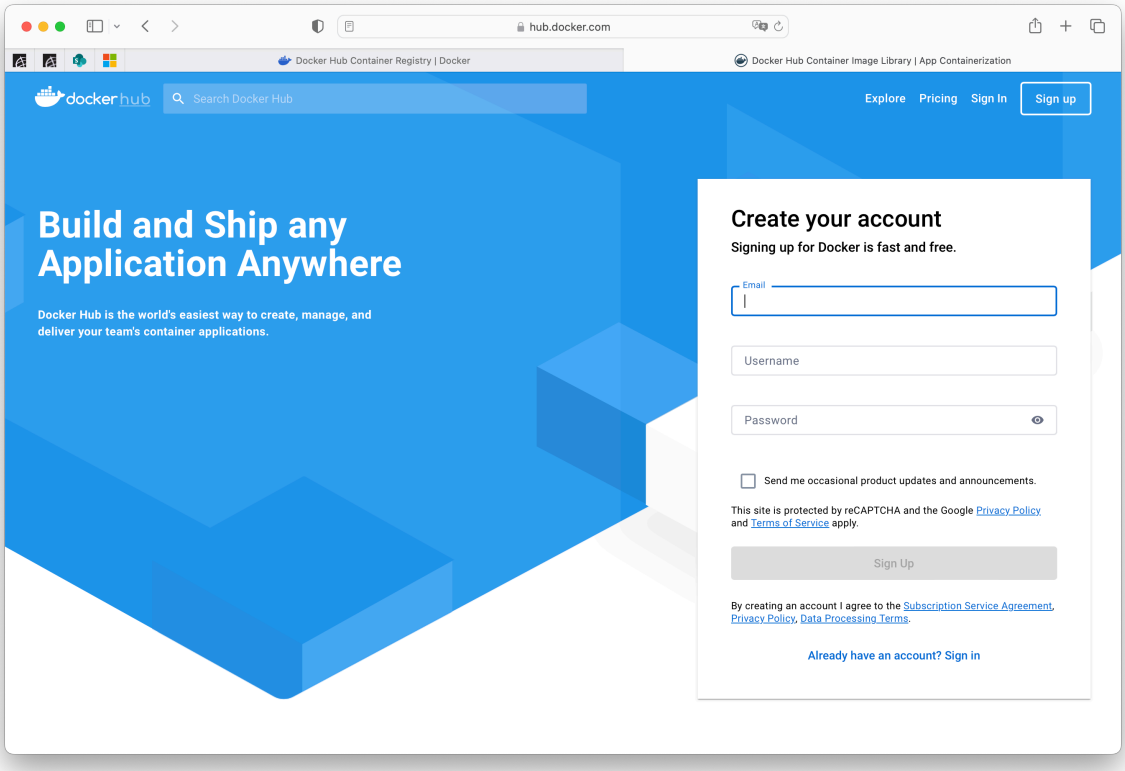

Figura 1.4.: Registo no Docker Hub

2. Portainer

2.1. Download

Para gestão de contentores iremos utilizar a solução **Portainer**, tal como identificado na Figura 2.1.

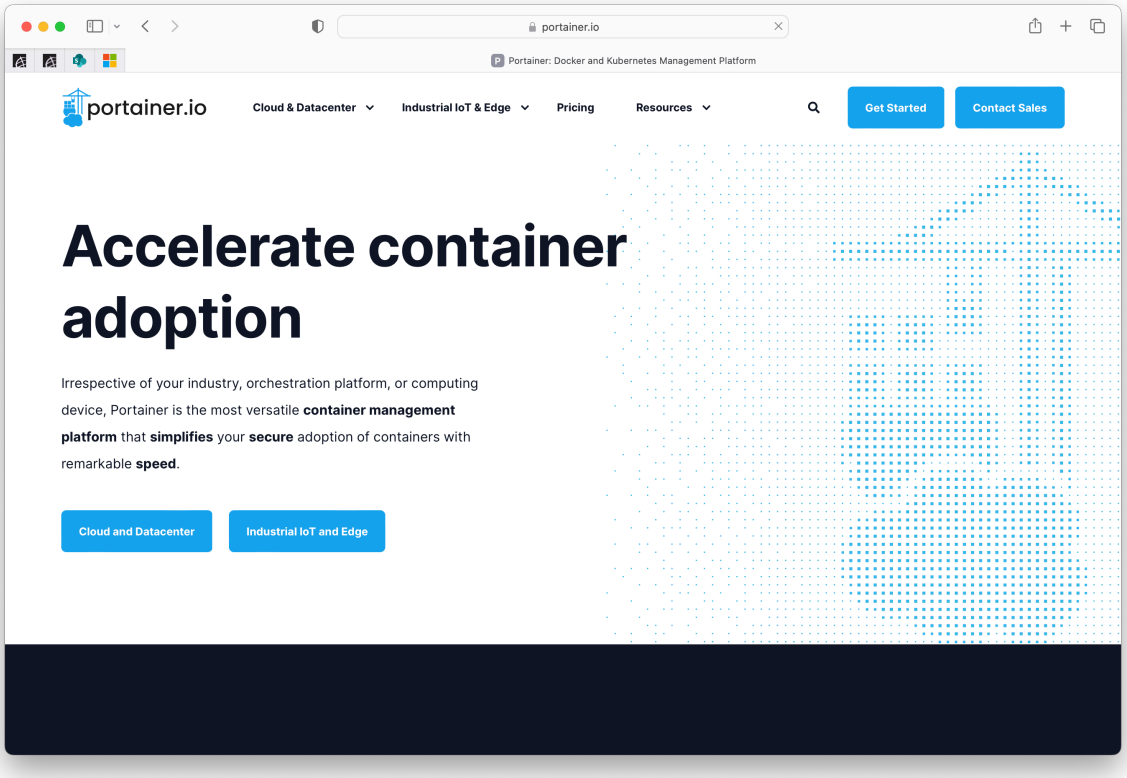

Figura 2.1.: Instalação de Portainer

2.2. Instalação

Basta instalar a versão **Community** e seguir os seguintes passos:

- Criar um volume para os dados persistentes

```
docker volume create portainer_data
```

- Instalar a **Community Edition**

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v
 ↪  /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data
 ↪  portainer/portainer-ce:latest
```

- Aceder à gestão de contentores no seguinte endereço.
  - Portainer Dashboard.

2.3. Configuração

- Fazer a configuração inicial do **Portainer**, criando um utilizador de gestão.

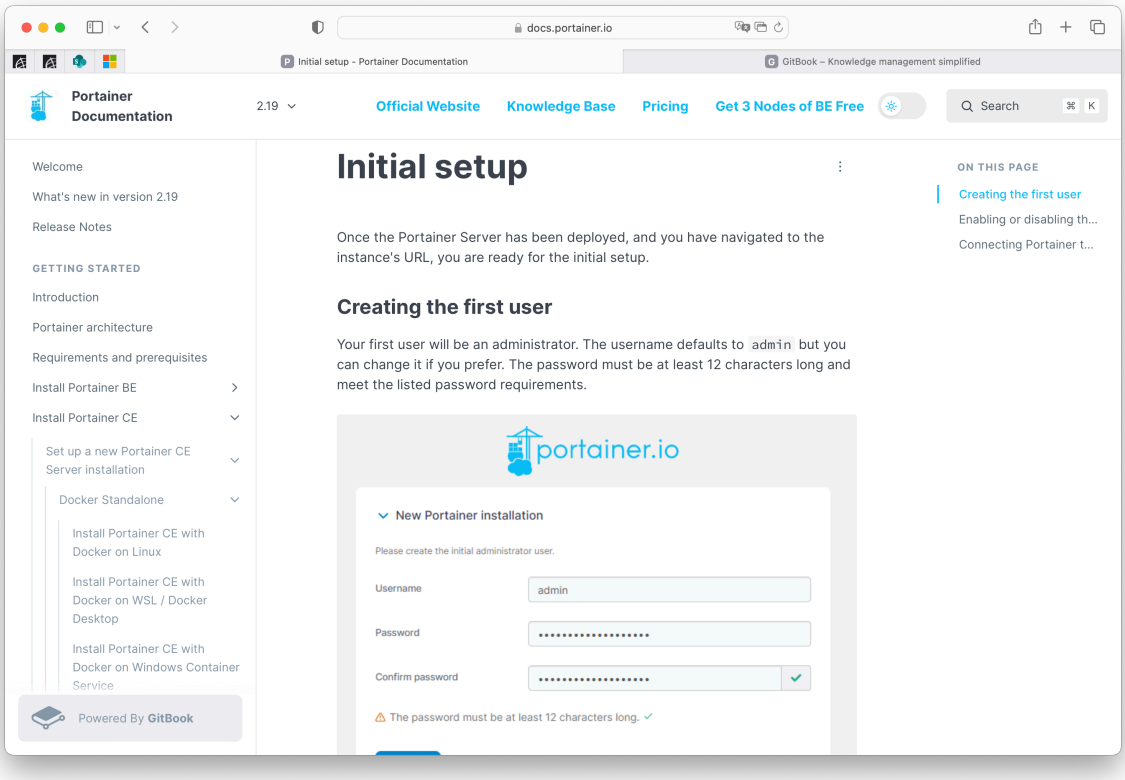

Figura 2.2.: Instalação de Portainer

- Escolher a opção **Get Started**

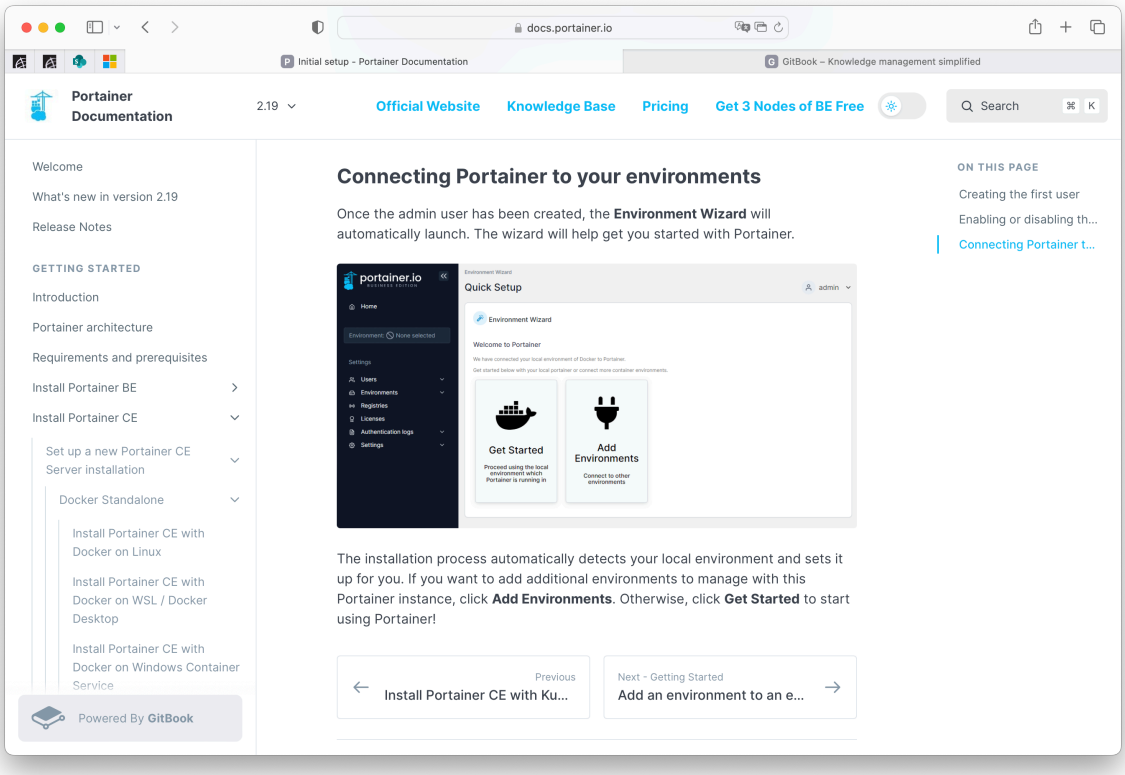

Figura 2.3.: Instalação de Portainer

- No menu do lado esquerdo seleccionar a opção Containers

3. Terraform

Here are the guidelines to install Terraform on **Windows**, **macOS**, and **Linux**.

3.0.1. 1. Install Terraform on Windows

3.0.1.1. Using Windows Installer (Recommended)

1. **Download Terraform**:
   - Visit the Terraform downloads page.
   - Select the Windows version (32-bit or 64-bit) and download the `.zip` file.
2. **Extract the Terraform Zip File**:
   - Extract the `.zip` file to a directory (e.g., `C:\terraform`).
3. **Add Terraform to System PATH**:
   - Open the **Start Menu**, search for `Environment Variables`, and click `Edit the system environment variables`.
   - In the `System Properties` window, click the `Environment Variables` button.
   - In `System Variables`, find `Path`, select it, and click `Edit`.
   - Add the path to the directory where `terraform.exe` is located (e.g., `C:\terraform`).
   - Click `OK` to close all dialogs.
4. **Verify Installation**:
   - Open `Command Prompt` or `PowerShell`.
   - Run:

```
terraform -v
```

3.0.2. 2. Install Terraform on macOS

3.0.2.1. Using Homebrew (Recommended)

1. Open **Terminal**.
2. Run:

```
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
```

3. **Verify Installation**:

```
terraform -v
```

3.0.2.2. Manual Installation (Alternative)

1. **Download Terraform**:
   - Visit the Terraform downloads page.
   - Select the macOS version and download the `.zip` file.
2. **Extract and Move Terraform**:
   - Extract the `.zip` file.
   - Move `terraform` binary to `/usr/local/bin`:

     ```
     sudo mv terraform /usr/local/bin/
     ```
3. **Verify Installation**:

   ```
   terraform -v
   ```

3.0.3. 3. Install Terraform on Linux

3.0.3.1. Using Package Repositories (Debian/Ubuntu)

1. **Add HashiCorp GPG Key and Repository**:

   ```
   curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
   sudo apt-add-repository "deb [arch=amd64]
    ↪  https://apt.releases.hashicorp.com $(lsb_release -cs) main"
   ```
2. **Install Terraform**:

   ```
   sudo apt-get update && sudo apt-get install terraform
   ```
3. **Verify Installation**:

   ```
   terraform -v
   ```

3.0.3.2. Using Package Repositories (RHEL/CentOS/Fedora)

1. **Add HashiCorp Repository**:

   ```
   sudo dnf config-manager --add-repo
    ↪  https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
   ```
2. **Install Terraform**:

   ```
   sudo dnf install terraform
   ```
3. **Verify Installation**:

   ```
   terraform -v
   ```

3.0.3.3. Manual Installation (For Other Distributions)

1. **Download Terraform**:
   - Visit the Terraform downloads page.
   - Select the Linux version and download the `.zip` file.
2. **Extract and Move Terraform**:

```
unzip terraform_<version>_linux_amd64.zip
sudo mv terraform /usr/local/bin/
```

3. **Verify Installation**:

```
terraform -v
```

These steps should enable you to install Terraform on Windows, macOS, and Linux systems easily.

4. Traefik

To install Traefik directly using Docker commands without Docker Compose, follow the steps below:

4.0.1. Step 1: Create a Traefik Network

First, create a Docker network that Traefik and other containers will use:

```
docker network create traefik_network
```

4.0.2. Step 2: Run Traefik Using Docker

- On your hard-drive, create a directory called `/traefik`. On `Windows` use `C:/traefik`. Use any other location if you wish.
- In that directory create the file `traefik.yml` with the following contents:

```
## traefik.yml

global:
  checkNewVersion: true
  sendAnonymousUsage: false

log:
  level: DEBUG

accessLog: {}

#tracing:
#  elastic: {}

metrics:
  prometheus: {}

ping: {}

# API and dashboard configuration
api:
  dashboard: true
  insecure: true

# Docker entrypoints backend
entryPoints:
```

```
  web:
    address: :80

  web-secure:
    address: :443

# Docker configuration backend
providers:
  docker:
    defaultRule: 'Host(`{{ if index .Labels "com.docker.compose.service" }}{{
 ↪  index .Labels "com.docker.compose.service" }}.aritlab.com{{ else }}{{
 ↪  trimPrefix `/` .Name }}.aritlab.com{{ end }}`)'
    endpoint: unix:///var/run/docker.sock
    # For Windows
    # endpoint: "npipe:////./pipe/docker_engine"
    watch: true
    exposedByDefault: true
```

- Use the following Docker command to install and run Traefik:

```
docker run -d --restart always -p 80:80 -p 8080:8080 -v
 ↪  /var/run/docker.sock:/var/run/docker.sock -v /traefik:/etc/traefik
 ↪  --network traefik_network --name traefik traefik:latest
```

Explanation:

- **`-d`**: Runs the container in detached mode (in the background).
- **`--restart always`**: Ensures Traefik restarts automatically if it stops.
- **`--network traefik_network`**: Connects Traefik to the created Docker network.
- **`-p`**: Maps host ports to container ports for HTTP (80), HTTPS (443), and the Dashboard (8080).
- **`-v /var/run/docker.sock:/var/run/docker.sock:ro`**: Allows Traefik to interact with Docker and detect running containers.
- **`-v /traefik:/etc/traefik`**: Allows Traefik to read the configuration file `traefik.yml` from local directory `/traefik`.

4.0.3. Step 3: Access the Traefik Dashboard

Open a web browser and navigate to `http://your-server-ip:8080`. You should see the Traefik dashboard showing active services and configurations.

4.0.4. Notes:

- Make sure your DNS records are correctly set up to point to your server IP for the domain you specify.
- For production, consider configuring HTTPS with Let's Encrypt and securing the Traefik dashboard.
- You can also customize Traefik further by creating a `traefik.toml` file and mapping it with the `-v` option.

This approach provides a quick way to set up Traefik using Docker without relying on Docker Compose.

Part II.

Classes

5. Week 2 (09-Oct-2025)

5.1. Docker Introductory Exercises

5.1.1. 1. Install Docker

- **Exercise**: Install Docker on your local machine.
- **Solution**: Follow the official installation instructions for your operating system:
  - **Windows & macOS**: Use Docker Desktop - Docker Installation Guide
  - **Linux**: Use `apt`, `yum`, or `dnf` to install Docker.

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

- **Verification**: Run the command:

```
docker --version
```

- **Expected Output**: Displays the installed Docker version.

5.1.2. 2. Run Your First Container

- **Exercise**: Run a simple container using the `hello-world` image.
- **Solution**:

```
docker run hello-world
```

- **Expected Output**: A message indicating that Docker is successfully installed.

5.1.3. 3. List Docker Images

- **Exercise**: List all Docker images on your system.
- **Solution**:

```
docker images
```

- **Expected Output**: A table listing all images, including `hello-world`.

5.1.4. 4. Pull a Docker Image

- **Exercise**: Pull the `nginx` image from Docker Hub.
- **Solution**:

```
docker pull nginx
```

- **Expected Output**: Confirmation of the download and extraction of the `nginx` image.

5.1.5. 5. Run a Web Server in a Container

- **Exercise**: Run the `nginx` container and map port 80 on the container to port 8080 on the host.
- **Solution**:

```
docker run -d -p 8080:80 nginx
```

- **Expected Output**: A unique container ID indicating that the container is running.
- **Verification**: Open `http://localhost:8080` in a web browser to see the default Nginx page.

5.1.6. 6. List Running Containers

- **Exercise**: List all running containers.
- **Solution**:

```
docker ps
```

- **Expected Output**: A table listing the running `nginx` container.

5.1.7. 7. Stop a Running Container

- **Exercise**: Stop the running `nginx` container.
- **Solution**:

```
docker stop <container_id>
```

- **Note**: Replace `<container_id>` with the actual ID from the `docker ps` command.
- **Expected Output**: The container ID indicating it was stopped successfully.

5.1.8. 8. Remove a Container

- **Exercise**: Remove the stopped `nginx` container.
- **Solution**:

```
docker rm <container_id>
```

- **Note**: Replace `<container_id>` with the actual ID of the container.
- **Expected Output**: The container ID indicating it was removed.

These exercises introduce basic Docker concepts, from running containers to creating images and using volumes. Completing them will give you a solid foundation in Docker usage.

6. Week 3 (16-Oct-2025)

6.1. Docker Introductory Exercises

6.1.1. 9. Build a Docker Image from a Dockerfile

- **Exercise**: Create a simple `Dockerfile` that uses the `nginx` base image and copies an `index.html` file into the container.
- **Solution**:

  Create a directory for your project:

  ```
  mkdir my-nginx
  cd my-nginx
  ```

  1. Create a file named `Dockerfile`:

     ```
     FROM nginx
     COPY index.html /usr/share/nginx/html/index.html
     # Expose the port
     EXPOSE 80
     ```

  2. Create a simple `index.html` with some content in the same directory.
  3. Build the Docker image:

     ```
     docker build -t my-nginx .
     ```

- **Expected Output**: Successful build message with a new image tagged as `my-nginx`.

6.1.2. 10. Run a Container with a Volume

- **Exercise**: Run the `nginx` container and mount a host directory to the container.
- **Solution**:

```
docker run -d -p 8080:80 -v /path/to/local/directory:/usr/share/nginx/html
 ↪  nginx
```

- **Confirmation**: Now put some files in directory `/path/to/local/directory` and check that they appear in the container in folder `/usr/share/nginx/html` .

- **Note**: Replace `/path/to/local/directory` with the path to your directory containing HTML files.
- **Expected Output**: A unique container ID indicating that the container is running.
- **Verification**: Open `http://localhost:8080` in a web browser to see the content of the mounted directory.

6.1.3. 11. Connect to a running Container

To connect to a running container called `nginx`, you can use the `docker exec` command. This allows you to run commands inside the container and access its shell. Here's how you can do it:

```
docker exec -it nginx /bin/bash
```

6.1.4. Explanation:

- `docker exec`: This command runs a command in a running container.
- `-it`: This flag allows you to interact with the container (interactive mode and pseudo-TTY).
- `nginx`: The name or ID of the running container you want to connect to.
- `/bin/bash`: The shell you want to use. If your container doesn't have `bash`, you can try `/bin/sh` instead.

After running this command, you will be inside the `nginx` container's shell, where you can navigate and execute commands as needed.

Pushing an image to Docker Hub involves a few straightforward steps. Here's a complete example of how to do it:

6.1.5. 12. Log In to Docker Hub

- **Registration**: Open Docker Hub Registration (https://hub.docker.com) in a web browser to register in Docker Hub.

Before you can push an image to Docker Hub, you need to log in:

```
docker login
```

You will be prompted to enter your Docker Hub username and password.

- According with the username you created on Docker Hub, build your image with that username. (for example: if your username is `jcppc`).
- Build the Docker image: `docker build -t jcppc/my-nginx .`

```
        docker build -t yourusername/my-nginx .
```

6.1.6. 13. Push the Image to Docker Hub

Now you can push your image to Docker Hub:

```
docker push yourusername/my-nginx:latest
```

6.1.7. 14. Verify the Image on Docker Hub

After a successful push, you can verify that your image has been uploaded by visiting your Docker Hub profile here and looking for `yourusername/my-nginx` in your repositories.

6.1.8. Conclusion

You have successfully created a Docker image for a simple Website application, built it, and pushed it to Docker Hub. You can now share this image with others or deploy it on various platforms that support Docker.

These exercises introduce basic Docker concepts, from running containers to creating images and using volumes. Completing them will give you a solid foundation in Docker usage.

7. Week 4/5 (23/30-Out-2025)

7.1. Portainer Introductory Exercises

7.1.1. Getting Started with Portainer

1. **Question**: What is Portainer, and why would you use it?
   - **Solution**: Portainer is a management tool that simplifies the use of Docker containers. It provides a graphical interface to manage Docker environments, making it easier to deploy, monitor, and manage containers.
2. **Question**: How do you install Portainer on a Docker-enabled system?
   - **Solution**: Use the following command:

```
docker volume create portainer_data

docker run -d -p 8000:8000 -p 9443:9443 --name portainer
 ↪  --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v
 ↪  portainer_data:/data portainer/portainer-ce:latest
```

3. **Question**: How can you access the Portainer web interface after installation?
   - **Solution**: Open a web browser and navigate to `https://<server-ip>:9443`.

7.1.2. Basic Container Management

4. **Question**: How do you deploy a new container using Portainer?
   - **Solution**: Go to "Containers" > "Add Container," specify the container name, image, ports, and other settings, then click "Deploy the container."
5. **Question**: How do you start, stop, and restart a container in Portainer?
   - **Solution**: Click on the container name, and use the "Start," "Stop," or "Restart" buttons.
6. **Question**: How can you view the logs of a container in Portainer?
   - **Solution**: Select the container, then click on the "Logs" tab.
7. **Question**: How do you execute a command inside a running container using Portainer?
   - **Solution**: Go to the container and click on "Console," then choose "bash" or "sh" and start the session.
8. **Question**: What are the steps to remove a container in Portainer?
   - **Solution**: Select the container, stop it if running, and click the "Remove" button.

7.1.3. Image Management

9. **Question**: How do you pull a new image from Docker Hub using Portainer?
    - **Solution**: Go to "Images" > "Pull Image," enter the name of the image, and click "Pull."
10. **Question**: How do you see all the Docker images stored on your system in Portainer?
    - **Solution**: Go to "Images" to view the list of images.
11. **Question**: How can you remove unused images in Portainer?
    - **Solution**: Select the image and click "Remove."

7.1.4. Network Management

12. **Question**: How do you create a new network in Portainer?
    - **Solution**: Go to "Networks" > "Add Network," specify the network details, and click "Create Network."
13. **Question**: How can you attach a running container to a different network in Portainer?
    - **Solution**: Go to the container's details, click on "Networks," and add the container to a new network.
14. **Question**: How do you view details about a specific network in Portainer?
    - **Solution**: Go to "Networks" and click on the network name to view its details.

7.1.5. Volume Management

15. **Question**: How do you create a new volume in Portainer?
    - **Solution**: Go to "Volumes" > "Add Volume," specify the volume name, and click "Create Volume."
16. **Question**: How can you mount a volume to a container in Portainer?
    - **Solution**: While deploying or editing a container, go to the "Volumes" section and add the volume.
17. **Question**: How do you remove a volume that is no longer in use?
    - **Solution**: Go to "Volumes," select the volume, and click "Remove."

7.1.6. Stack Management

18. **Question**: What is a stack in Portainer, and how do you deploy one?
    - **Solution**: A stack is a group of interrelated services defined by a Docker Compose file. Go to "Stacks" > "Add Stack," upload or paste your Compose file, and click "Deploy the stack."
19. **Question**: How can you view the status of a stack in Portainer?

- **Solution**: Go to "Stacks" and click on the stack name to see the details and status of each service.

20. **Question**: How do you remove a stack in Portainer?

- **Solution**: Select the stack, and click the "Remove" button.

7.1.6.1. Add a Stack (one container)

To add an Nginx container using Docker Compose in Portainer, you can follow these steps:

20.1. **Access Portainer:** - Log into your Portainer dashboard.

20.2. **Navigate to Stacks:** - Go to the menu on the left and click on **Stacks**. - Click on + **Add Stack**.

20.3. **Name the Stack:** - Give your stack a name, e.g., `nginx-webserver`.

20.4. **Add Docker Compose Configuration:** - In the **Web editor** section, paste the following Docker Compose YAML configuration:

```
    services:
      nginx:
        image: nginx:latest
        container_name: nginx_container
        ports:
          - "80:80"  # Exposing port 80 on the host to port 80 on the
↪  container
        volumes:
          - ./nginx/html:/usr/share/nginx/html  # Mounting local HTML
↪  directory to Nginx container
        restart: always
```

- **Explanation:**
  - `image`: Defines the Nginx image that will be pulled from Docker Hub.
  - `ports`: Maps port 80 on the host to port 80 on the container, making Nginx accessible via `http://your-server-ip:80`.
  - `volumes`: Maps local directories to the Nginx container for HTML files and custom Nginx configuration.
  - `restart`: Ensures that the container always restarts if it stops or fails.

20.5. **Deploy the Stack:** - Click on **Deploy the stack**. - Portainer will pull the Nginx image and start the container based on the configuration.

20.6. **Verify:** - After deployment, navigate to **Containers** or **Stacks** to see your running Nginx container. - Open your browser and visit `http://your-server-ip` to verify that Nginx is running.

You can adjust the Docker Compose configuration as needed (e.g., change port mappings, add more volumes, or set environment variables).

7.1.6.2. Add a Stack (one container)

To add a MySQL container using Docker Compose in Portainer, follow these steps:

20.7. **Access Portainer:** - Log into your Portainer dashboard.

20.8. **Navigate to Stacks:** - Go to the menu on the left and click on **Stacks**. - Click on + **Add Stack**.

20.9. **Name the Stack:** - Give your stack a name, e.g., `mysql-database`.

20.10. **Add Docker Compose Configuration:** - In the **Web editor** section, paste the following Docker Compose YAML configuration:

```
    services:
      mysql:
        image: mysql:latest
        container_name: mysql_container
        environment:
          - MYSQL_ROOT_PASSWORD=rootpassword  # Set the root password
          - MYSQL_DATABASE=mydatabase         # Optional: create a database
          - MYSQL_USER=myuser                 # Optional: create a user
          - MYSQL_PASSWORD=mypassword         # Optional: set password for the
↪  user
        ports:
          - "3306:3306"  # Expose MySQL port 3306
        volumes:
          - mysql_data:/var/lib/mysql  # Persist MySQL data
        restart: always

    volumes:
      mysql_data:
        driver: local
```

- **Explanation:**
  - `image`: Defines the MySQL image that will be pulled from Docker Hub.
  - `environment`: Set environment variables to configure MySQL (e.g., root password, database name, user, password).
  - `ports`: Maps port 3306 on the host to port 3306 on the container, making MySQL accessible.
  - `volumes`: Ensures data is persisted even if the container is stopped or removed.
  - `restart`: Automatically restarts the container if it stops or fails.

20.11. **Deploy the Stack:** - Click on **Deploy the stack**. - Portainer will pull the MySQL image and start the container based on the configuration.

20.12. **Verify:** - After deployment, navigate to **Containers** or **Stacks** to see your running MySQL container. - You can connect to the MySQL server using a MySQL client or management tool like `mysql` CLI, `MySQL Workbench`, or another application by specifying `your-server-ip:3306`.

Adjust the configuration (e.g., change environment variables, port mappings, or volume settings) as needed.

7.1.6.3. Add a Stack (with two containers)

To add both a MySQL container and an Nginx container in the same stack and network using Portainer, follow these steps:

20.13. **Access Portainer:** - Log into your Portainer dashboard.

20.14. **Navigate to Stacks:** - Go to the menu on the left and click on **Stacks**. - Click on **+ Add Stack**.

20.15. **Name the Stack:** - Give your stack a name, e.g., `nginx-mysql-stack`.

20.16. **Add Docker Compose Configuration:** - In the **Web editor** section, paste the following Docker Compose YAML configuration:

```
    services:
      nginx:
        image: nginx:latest
        container_name: nginx_container
        ports:
          - "80:80"  # Expose Nginx on port 80
        volumes:
          - ./nginx/html:/usr/share/nginx/html  # Mount HTML files
        networks:
          - web_network
        restart: always

      mysql:
        image: mysql:latest
        container_name: mysql_container
        environment:
          - MYSQL_ROOT_PASSWORD=rootpassword  # Set the root password
          - MYSQL_DATABASE=mydatabase         # Optional: create a database
          - MYSQL_USER=myuser                 # Optional: create a user
          - MYSQL_PASSWORD=mypassword         # Optional: set password for the
↪  user
        ports:
          - "3306:3306"  # Expose MySQL on port 3306
        volumes:
          - mysql_data:/var/lib/mysql  # Persist MySQL data
        networks:
          - web_network
        restart: always

    volumes:
      mysql_data:
        driver: local

    networks:
      web_network:
        driver: bridge
```

- **Explanation:**
  - **`services`:** Defines both the Nginx and MySQL containers.
  - **`networks`:** Both containers are connected to the same `web_network`, allowing them to communicate internally.
  - **`volumes`:** Persist data for MySQL and mount HTML/config files for Nginx.
  - **`restart`:** Ensures both containers restart automatically if they stop or fail.

20.17. **Deploy the Stack:** - Click on **Deploy the stack**. - Portainer will pull the necessary images and start the containers based on the configuration.

20.18. **Verify:** - After deployment, navigate to **Containers** or **Stacks** to see your running Nginx and MySQL containers. - Nginx will be accessible via `http://your-server-ip:80`, and MySQL can be accessed internally within the same network or externally via `your-server-ip:3306`. - You can connect to the MySQL server using a MySQL client or management tool like `mysql` CLI, `MySQL Workbench`, or another application by specifying `your-server-ip:3306`.

You can modify the configuration, such as changing environment variables, volume paths, or adding other services to the same network.

7.1.6.4. Installing MySQL Workbench

To install MySQL Workbench, follow the appropriate steps for your operating system:

7.1.7. 1. Windows:

20.19. **Download the Installer:** - Go to the MySQL Workbench download page. - Select your operating system and click on the **Download** button. - You may be asked to log in or sign up for an Oracle account, but you can click on **No thanks, just start my download** to skip this.

20.20. **Run the Installer:** - Open the downloaded `.msi` file and run the installer. - Follow the setup wizard steps, choosing the **Complete** installation type to install all necessary components.

20.21. **Finish the Installation:** - Once the installation is complete, click **Finish**. - You can now open MySQL Workbench from the Start Menu.

7.1.8. 2. macOS:

20.22. **Download the Installer:** - Go to the MySQL Workbench download page. - Select **macOS** and download the `.dmg` file.

20.23. **Install the Application:** - Open the downloaded `.dmg` file. - Drag the **MySQL Workbench** icon to the **Applications** folder.

20.24. **Open MySQL Workbench:** - You can now open MySQL Workbench from the **Applications** folder or through **Spotlight Search**.

7.1.9. 3. Linux (Ubuntu/Debian):

20.25. **Open Terminal and Update Package Lists:**

```
sudo apt update
```

20.26. **Install MySQL Workbench:**

```
sudo apt install mysql-workbench
```

20.27. **Launch MySQL Workbench:** - After installation, you can open it by searching for **MySQL Workbench** in your application menu or by running:

```
mysql-workbench
```

7.1.10. 4. Linux (CentOS/RHEL/Fedora):

20.28. **Enable the MySQL Repository (if not already installed):**

```
sudo yum install
 ↪ https://dev.mysql.com/get/mysql80-community-release-el7-3.noarch.rpm
```

20.29. **Install MySQL Workbench:**

```
sudo yum install mysql-workbench
```

20.30. **Launch MySQL Workbench:** - Open MySQL Workbench from the application menu or by running:

```
mysql-workbench
```

After installation, you can use MySQL Workbench to connect to your MySQL server and manage your databases.

7.1.11. Managing Users and Teams

21. **Question**: How do you add a new user in Portainer?
    - **Solution**: Go to "Users" > "Add User," enter the username and password, and click "Create user."
22. **Question**: How can you create a new team in Portainer?
    - **Solution**: Go to "Teams" > "Add Team," enter the team name, and assign users to the team.
23. **Question**: How do you assign a team to a specific environment?
    - **Solution**: Go to "Environments," select the environment, and assign a team under "Access Control."

7.1.12. Environment Management

24. **Question**: How do you connect Portainer to a remote Docker environment?
    - **Solution**: Go to "Environments" > "Add Environment," select "Docker" and "Agent," and provide the necessary details.
25. **Question**: How can you configure multiple Docker environments in Portainer?
    - **Solution**: Go to "Environments" and add as many environments as needed, configuring each individually.
26. **Question**: What steps are involved in enabling TLS for a Docker environment in Portainer?
    - **Solution**: While adding or editing an environment, check the "TLS" option and upload the required certificates.

7.1.13. Resource Management

27. **Question**: How do you limit the CPU and memory usage for a container in Portainer?
    - **Solution**: While deploying or editing a container, specify the CPU and memory limits in the "Resource control" section.
28. **Question**: How can you configure environment variables for a container?
    - **Solution**: During container setup, go to the "Environment variables" section and add the key-value pairs.
29. **Question**: How do you set up port mappings for a container in Portainer?
    - **Solution**: In the "Network" section during container deployment, specify the host and container ports.

7.1.14. Monitoring and Maintenance

30. **Question**: How can you view the resource usage (CPU, memory, etc.) of a container in Portainer?
    - **Solution**: Select the container and go to the "Stats" tab.
31. **Question**: How do you back up Portainer data?
    - **Solution**: Backup the `portainer_data` volume using `docker run` commands or manually copy the data to another location.
32. **Question**: How can you update Portainer to a newer version?
    - **Solution**: Pull the new image, stop and remove the current Portainer container, and re-deploy with the new version.

7.1.15. Security and Access Control

33. **Question**: How do you enable authentication in Portainer?
    - **Solution**: Go to "Settings," enable authentication, and configure users under "Users."
34. **Question**: How can you create and manage roles for users in Portainer?
    - **Solution**: Use the "Roles" section to define specific access permissions for users.
35. **Question**: How do you set up OAuth or LDAP authentication in Portainer?
    - **Solution**: Go to "Settings" > "Authentication," and configure the external authentication provider.

7.1.16. Advanced Topics

36. **Question**: How do you deploy a service with replicas using Portainer?
    - **Solution**: Go to "Stacks" and use a Docker Compose file to define the service with the `replicas` parameter.
37. **Question**: How can you set up automatic updates for containers using Watchtower in Portainer?
    - **Solution**: Deploy the Watchtower container through Portainer, specifying the `--cleanup` and `--interval` flags.
38. **Question**: How do you deploy a custom Docker image from a private registry in Portainer?
    - **Solution**: Add the registry under "Registries," then specify the image during container deployment.
39. **Question**: What is the process to configure Docker Swarm services in Portainer?
    - **Solution**: Go to "Stacks," and use Docker Compose files to define and deploy Swarm services.
40. **Question**: How do you use Portainer to manage Kubernetes clusters?
    - **Solution**: Connect to a Kubernetes environment in "Environments" and use the "Applications" feature to manage pods, services, and deployments.

These exercises cover fundamental and advanced aspects of using Portainer, helping professionals to understand both basic container management and more complex setups like multi-environment management, stacks, and security.

8. Week 6 (06-Nov-2025)

8.1. Terraform Introductory Exercises

8.1.1. Terraform with Local Docker - Code Example

Title: Terraform for Local Docker Containers

- This code uses Terraform to define and run a Docker container locally. It requires the Docker provider, pulls an image, and runs a container with specific configuration.
- Create a directory called `example1`.
- Create a file inside that directory called `nginx.tf`
- Include the following contents in this file.

Example Code:

```
# Install the Docker provider (if needed)
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "3.0.2"
    }
  }
}

# Define the Docker provider
provider "docker" {
  host = "unix:///var/run/docker.sock"
  # Use this below configuration instead if Docker is running on Windows
  # host = "npipe:////./pipe/docker_engine"
}

# Pull the latest Nginx image
resource "docker_image" "nginx_image" {
  name = "nginx:latest"
}

# Run a Docker container using the pulled Nginx image
resource "docker_container" "nginx_container" {
```

```
  image = docker_image.nginx_image.image_id
  name  = "nginx_server"
  ports {
    internal = 80
    external = 8080
  }
}
```

Explanation:

1. **Provider**: Connects to Docker using the local Docker socket.
2. **Image Resource**: Downloads the Nginx Docker image.
3. **Container Resource**: Creates a container from the Nginx image, exposing it on port `8080` locally.

- In the `example1` directory run the follwing commands one by one and check the results.

Commands to Run:

- `terraform init`: Initializes the Docker provider.
- `terraform plan`: To validate the configuration without really executing it.
- `terraform apply`: Provisions the Nginx container.

8.1.2. Terraform with Local Docker using variables

- Based on the class slides, convert the above exercise to use a variables file and apply it to this scenario.
- The parameter variables are the `name` of the container and the `external` port.

Example Code:

File: main.tf

```
# main.tf
# Install the Docker provider (if needed)
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "3.0.2"
    }
  }
}

# Define the Docker provider
provider "docker" {
  host = "unix:///var/run/docker.sock"
  # Use this below configuration instead if Docker is running on Windows
  # host = "npipe:////./pipe/docker_engine"
```

```
}

# Pull the latest Nginx image
resource "docker_image" "nginx_image" {
  name = "nginx:latest"
}

# Run a Docker container using the pulled Nginx image
resource "docker_container" "nginx_container" {
  image = docker_image.nginx_image.image_id
  name  = var.container_name
  ports {
    internal = 80
    external = var.external_port
  }
}
```

- Create a file inside that directory called `variables.tf`
- Include the following contents in this file.

File: variables.tf

```
# variables.tf
variable "container_name" {
  description = "Name of the Docker container"
  type        = string
}

variable "external_port" {
  description = "External port for the Docker container"
  type        = number
}
```

- Create a file inside that directory called `terraform.tfvars`
- Include the following contents in this file.

File: terraform.tfvars

```
# terraform.tfvars
container_name = "my_nginx_container"
external_port  = 8080
```

Commands to Run:

- `terraform init`: Initializes the Docker provider.
- `terraform validate`: It verifies that the configuration files (.tf and/or .tf.json) are syntactically correct.
- `terraform plan`: To validate the configuration without really executing it.
- `terraform apply`: Provisions the Nginx container.

8.1.3. Deploying MySQL to Local Docker with Terraform

- Using the same approach, deploy a MySQL container to docker.
- The parameter variables are the `name` of the container, the `external` port, the `database name` and the root `password` .
- You may search `Google` and/or `ChatGPT` to implement this exercise.
- This code uses Terraform to define and run a MySQL Docker container locally. It requires the Docker provider, pulls an image, and runs a container with specific configuration.
- Create a directory called `example2`.
- Create a file inside that directory called `main.tf`
- Include the following contents in this file.

File: main.tf

```
# main.tf
# Install the Docker provider (if needed)
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "3.0.2"
    }
  }
}

# Define the Docker provider
provider "docker" {
  host = "unix:///var/run/docker.sock"
  # Use this below configuration instead if Docker is running on Windows
  # host = "npipe:////./pipe/docker_engine"
}

resource "docker_image" "mysql" {
  name = "mysql:latest"
}

resource "docker_container" "mysql" {
  image = docker_image.mysql.image_id
  name  = var.container_name

  ports {
    internal = 3306
    external = var.external_port
  }

  env = [
    "MYSQL_DATABASE=${var.database_name}",
    "MYSQL_ROOT_PASSWORD=${var.root_password}"
```

```
  ]
}
```

- Create a file inside that directory called `variables.tf`
- Include the following contents in this file.

File: variables.tf

```
variable "container_name" {
  description = "Name of the Docker container"
  type        = string
}

variable "external_port" {
  description = "External port for the MySQL container"
  type        = number
}

variable "database_name" {
  description = "Name of the MySQL database to be created"
  type        = string
}

variable "root_password" {
  description = "Root password for the MySQL instance"
  type        = string
  sensitive   = true
}
```

- Create a file inside that directory called `terraform.tfvars`
- Include the following contents in this file.

File: terraform.tfvars

```
# terraform.tfvars
container_name = "mysql_container"
external_port  = 3306
database_name  = "my_database"
root_password  = "strongpassword123"
```

Commands to Run:

- `terraform init`: Initializes the Docker provider.
- `terraform validate`: It verifies that the configuration files (.tf and/or .tf.json) are syntactically correct.
- `terraform plan`: To validate the configuration without really executing it.
- `terraform apply`: Provisions the Nginx container.

9. Week 7/8 (20/27-Nov-2025)

9.1. Create a Nginx Cluster

- Using the contents explained in section Seção 6.1.2 create two different containers of nginx, called `nginx1`and `nginx2`, each, using a different external port.
- Make sure each container map a volume to different directories and containing a different `index.html` file. Change the contents of this file to identify the `nginx` node. This is needed to clearly distinguish the contents of each `nginx` node.
- **Verification**: Open `http://localhost:port` in a web browser to see the content of the mounted directory on each cluster node, where the `port` parameter is the value of each node.
- Since `nginx1` and `nginx2` are supposed to have the same application contents, how can we distribute the traffic between the two nodes ?
  - The answer is: Using a Reverse Proxy to perform load balancing. A Reverse Proxy like `Traefik`.

9.2. Traefik Installaltion

- Please refer to section Capítulo 4 to install Traefik.

9.3. Load Balancing (using Traefik)

- Implement a load balancing mechanism using `Traefik` to distribute the traffic between each `nginx` cluster nodes.
- On your hard-drive, create a directory called `dynamic` inside `/traefik`. On `Windows` use `C:/traefik`. Use any other location if you wish.
- Change the `traefik.yml` file to include the following contents:

```
## traefik.yml

global:
  checkNewVersion: true
  sendAnonymousUsage: false

log:
```

```
  level: DEBUG

accessLog: {}

#tracing:
#  elastic: {}

metrics:
  prometheus: {}

ping: {}

# API and dashboard configuration
api:
  dashboard: true
  insecure: true

# Docker entrypoints backend
entryPoints:
  web:
    address: :80

  web-secure:
    address: :443

# Docker configuration backend
providers:
  docker:
    defaultRule: 'Host(`{{ if index .Labels "com.docker.compose.service" }}{{
↪  index .Labels "com.docker.compose.service" }}.aritlab.com{{ else }}{{
↪  trimPrefix `/` .Name }}.aritlab.com{{ end }}`)'
    endpoint: unix:///var/run/docker.sock
    # For Windows
    # endpoint: "npipe:////./pipe/docker_engine"
    watch: true
    exposedByDefault: true

  file:
    directory: "/etc/traefik/dynamic"
    watch: true
```

- Inside directory `/traefik/dynamic` create the file `dynamic.yml` with the following contents:
- From Docker, get the ip addresses from each `nginx`cluster node.

```
## DYNAMIC CONFIGURATION
http:
  routers:
    labcluster:
      rule: "Host(`www.aritlab.com`)"
```

```
      service: labcluster

  services:
    labcluster:
      loadBalancer:
        servers:
            - url: "http://ip_address_nginx1:port1/"
            - url: "http://ip_address_nginx2:port2/"
```

- Use again the following Docker command to install and run Traefik:

```
docker run -d --restart always -p 80:80 -p 8080:8080 -v
 ↪  /var/run/docker.sock:/var/run/docker.sock -v /traefik:/etc/traefik
 ↪  --network traefik_network --name traefik traefik:latest
```

- In your hosts file, map the name `www.aritlab.com` to your own host hostname/ip address.

Open a web browser and navigate to `http://www.aritlab.com`. You should see now that `Traefik` is redirecting you either to `nginx1` or `nginx2`.

10. Week 9 (27-Nov-2025)

10.1. Create a Large Language Model Cluster

10.1.1. Create the LLM Engine Container

- We are going to use the Ollama software from here.
- Create a network called `llm_network`

```
docker network create llm_network
```

- Create a volume called `ollama-data`

```
docker volume create ollama-data
```

- Create the docker container with the `ollama` LLM engine.

```
docker run -d -v ollama-data:/root/.ollama -p 11434:11434 --network llm_network
 ↪  --name ollama ollama/ollama
```

10.1.2. Create the LLM Web UI

- Create a volume called `ollama-webui`

```
docker volume create ollama-webui
```

- Create the docker container with the `ollama` web graphical user interface.

```
docker run -d -p 3000:8080 --add-host=host.docker.internal:host-gateway -v
 ↪  ollama-webui:/app/backend/data --network llm_network --name llmui --restart
 ↪  always ghcr.io/open-webui/open-webui:main
```

10.1.3. Update hosts file

- Update your hosts file to point `llmui.mylab.com` to your own machine IP address.
  - ex: 192.168.1.100 llmui.mylab.com
- Check that you can access `llmui.mylab.com` using your browser.
- `Traefik` should now route your traffic to the `llmui` container.

10.1.4. Download your first LLM model

- Connect to your `ollama` container with this command:

```
docker exec -it ollama /bin/bash
```

- Run the following command to download the LLM model `llama3.2`.

```
ollama pull llama3.2
```

10.1.5. Ask Questions with the Web Interface

- Without `Traefik`, you can access your web interface in http://localhost:3000.
- If you have `Traefik` running, access your web interface in http://llmui.mylab.com.

10.1.6. How to download new LLM models on multiple hosts ?

- What can you use ?
  - Yes, a good method is by using our "friend" …… `Ansible`.
- We will do this in the next class.

11. Week 10 (4-Dez-2025)

11.1. Use Ansible to Modify an `index.html` file in Nginx Containers

11.1.1. Prepare the New index.html

- Create the file to be copied later into the containers.
- File /path/to/new_index.html

```
<html>
<head><title>Updated Nginx Page</title></head>
<body>
    <h1>Hello from Ansible!</h1>
</body>
</html>
```

11.1.2. Build a Docker Image from a Dockerfile

- **Exercise**: Create a simple `Dockerfile` that uses the `nginx` base image and include the `ssh server` and `python` into the container.
- **Solution**: Create a directory for your project.

```
mkdir nginx-ssh-container
cd nginx-ssh-container
```

- Create a file named `Dockerfile`.

```
# Use the official Nginx image as the base
FROM nginx:latest

# Update package list and install OpenSSH server
RUN apt-get update && \
    apt-get install -y openssh-server && \
    mkdir /var/run/sshd

# Install Python
RUN apt-get update && \
    apt-get install -y python3 python3-pip
```

```
# Set a password for the root user (replace 'password' with a secure password)
RUN echo 'root:password' | chpasswd

# Allow root login via SSH
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/'
 ↪  /etc/ssh/sshd_config

# Expose ports for Nginx and SSH
EXPOSE 80 22

# Start both SSH and Nginx services
CMD service ssh start && nginx -g 'daemon off;'
```

- Build the Docker image.

```
docker build -t my-nginx-with-ssh .
```

- **Expected Output**: Successful build message with a new image tagged as `my-nginx-with-ssh`.

11.1.3. Setup Two Nginx Containers

- Deploy two Nginx containers based on the created Docker image.

```
docker run -d -p 8081:80 -p 2221:22 --name nginx1 my-nginx-with-ssh
docker run -d -p 8082:80 -p 2222:22 --name nginx2 my-nginx-with-ssh
```

- Verify the contents by accessing the Nginx containers in a browser:
  - For nginx1: http://localhost:8081
  - For nginx2: http://localhost:8082
- Confirm that you can also login to those nginx servers. Use the password defined in the `Dockerfile` above.

```
ssh root@localhost -p 2221
ssh root@localhost -p 2222
```

- Exit from both hosts.

11.1.4. Generate a SSH Key Pair

- Generate a Key Pair (Private & Public Keys). This will allow you to connect (over SSH) to any host without a password.
- When generating the keys, just press `Enter` on any inputs requests.

```
ssh-keygen -t rsa -b 4096 -C "ansible_control_machine"
```

- Copy your public key into the nginx servers to be able to login without a password. Enter the password when needed.

- After this step, you are now able to login into the nginx servers without password.

```
ssh-copy-id -i ~/.ssh/id_rsa.pub -p 2221 root@localhost
ssh-copy-id -i ~/.ssh/id_rsa.pub -p 2222 root@localhost
```

- Login again to those nginx servers and confirm that you don't need to enter the password anymore.

```
ssh root@localhost -p 2221
ssh root@localhost -p 2222
```

11.1.5. Create a New Ansible Playbook

- Write an Ansible playbook to modify the index.html file inside the containers.
- Create file `update_nginx_index.yml`

```
- name: Update index.html in Nginx containers
  hosts: nginx_containers
  tasks:
    - name: Ensure the new index.html exists inside the container
      copy:
        src: /path/to/new_index.html
        dest: /usr/share/nginx/html/index.html
        remote_src: no
```

- Create file `inventory.yml`

```
all:
  hosts:
    nginx1:
      ansible_host: localhost
      ansible_port: 2221
      ansible_user: root
      ansible_python_interpreter: /usr/bin/python3
    nginx2:
      ansible_host: localhost
      ansible_port: 2222
      ansible_user: root
      ansible_python_interpreter: /usr/bin/python3
  children:
    nginx_containers:
      hosts:
        nginx1:
        nginx2:
```

- Check that you can ping the hosts by executing the following command.

```
ansible -i inventory.yml -m ping
```

- If you have success, now you can run the playbook.

11.1.6. Run the Playbook

- Use the following command to execute the playbook and update the containers.

```
ansible-playbook -i inventory.yml update_nginx_index.yml
```

11.1.7. Expected Outcome

- The index.html file inside both nginx1 and nginx2 containers is replaced with the custom version.
- Restarting the containers reflects the updated content.
- Verify by accessing the Nginx containers in a browser:
  - For nginx1: http://localhost:8081
  - For nginx2: http://localhost:8082
- Both should display the updated message: “Hello from Ansible!”

12. Week 10/11 (4/11-Dec-2025)

12.1. Use Azure Cli to manage Azure Resources

12.1.1. Download Azure Cli

- Access the following site to download the tool for your operating system and install it.

12.1.2. Run the Azure client

- Check the client version

```
az version
```

- If you have already, just upgrade to the latest version.

```
az upgrade
```

- Make sure you afre able to login to Azure.
- Enter your username and password in the browser and return to the command line.

```
az login
```

- Once you are authenticated, you can execute all other commands.
- Try the following to list existing resources in your Account.

```
az resource list
```

- Try the following to list existing VMs in your Account.

```
az vm list
```

12.1.3. Create the first template (empty template) file.

- Open any code editor.
- Create a new file named `azuredeploy.json`.
- Add the following contents to it.

```
{
  "$schema":
 ↪  "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "resources": []
}
```

12.1.4. Deploy the template

- Make sure you are logged in to Azure.

```
az login
```

- If you have multiple Azure subscriptions, choose the subscription you want to use by running the following.

```
az account set --subscription SubscriptionName
```

- When you deploy a template, you can specify a resource group to contain the resources. Before running the deployment command, create the resource group in a specific location (use Western Europe).

```
az group create --name myResourceGroup --location 'westeurope'
```

- Deploy the template by using the resource group you just created. Give a name to the deployment so you can easily identify it in the deployment history.
- For convenience, also create a variable that stores the path to the template file.
- This variable makes it easier for you to run the deployment commands because you don't have to retype the path every time you deploy.
- Replace `{provide-the-path-to-the-template-file}` and the curly braces {} with the full path of your template fil created earlier.

```
templateFile="{provide-the-path-to-the-template-file}"
az deployment group create --name my-first-blanktemplate --resource-group
 ↪  myResourceGroup --template-file $templateFile
```

- The deployment command returns results. Look for ProvisioningState to see whether the deployment succeeded.
- Documentation for ARM templates can be found here.

12.1.5. Azure Quickstart Templates

- Azure ARM Quickstart templates can be found here.

12.2. Exercises

12.2.1. Deploy a VM

- Based on the Quickstart templates, deploy your first Linux VM to Azure.
- Do so, by adding contents only in the `"resources": []` section of your template file.

12.2.1.1. Solution

- The following command requests several pieces of input from the user. These include:
  - Name of the Resource Group (resourceGroupName)
  - Location of the Azure datacenter that hosts the VM (location)
  - A name for resources related to the VM (projectName)
  - Username for the administrator user (username)
  - A public SSH key for accessing the VM's terminal (key)

```
# You need to have generated a pair of public/private keys before in your
 ↪  machine.
# Execute the following in Linux or MacOS. For Windows, OpenSSH Client must be
 ↪  installed.
ssh-keygen -t rsa -b 2048
read -r pubkey < ~/.ssh/id_rsa.pub

# This example uses variables as an input for the commands
echo "Enter the Resource Group name:"
read resourceGroupName
echo "Enter the location (i.e. westeurope):"
read location
echo "Enter the project name (used for generating resource names):"
read projectName
echo "Enter the administrator username:"
read username

# This prints all the variables

echo $pubkey
echo ""
echo $resourceGroupName
echo $location
echo $projectName
echo $username

# Get the content from this file and put into `azuredeploy.json` file
#
 ↪  https://raw.githubusercontent.com/azure/azure-quickstart-templates/master/quickstarts/mi
# Change SKU to "Standard" and publicIPAllocationMethod to "Static"

# This example uses variables as an input for the commands but you can
 ↪  hard-code the values instead.
```

```
az group create --name $resourceGroupName --location "$location"

az deployment group create --resource-group $resourceGroupName --template-file
 ↪  ./azuredeploy.json --parameters projectName=$projectName
 ↪  adminUsername=$username adminPublicKey="$pubkey"

az vm show --resource-group $resourceGroupName --name "$projectName-vm"
 ↪  --show-details --query publicIps --output tsv
```

- Connect to the virtual machine.

```
ssh <adminUsername>@<ipAddress>
```

13. Week 12 (18-Dec-2025)

13.1. Use Azure Cli to manage Azure Resources

13.1.1. Download Azure Cli

- Access the following site to download the tool for your operating system and install it.

13.1.2. Run the Azure client

- Check the client version

```
az version
```

- If you have already, just upgrade to the latest version.

```
az upgrade
```

13.1.3. Download this exercise files

- Download the files needed to run this exercise from here.
- Expand the zip file.
- Move into the directory created by expanding the zip file.

13.1.4. Login to Azure

- Make sure you are able to login to Azure.
- Enter your username and password in the browser and return to the command line.

```
az login
```

13.1.5. Create a Resource Group

- You need to create a resource group (in this example named `ul-eventhubs-rg`) in a specific `location` to contain your hardware/software resources.
- If you don't know the possible locations execute: `az account list-locations -o table`and then pick the name you want.

```
az group create --name ul-eventhubs-rg --location westeurope # In this example
 ↪  we are using  westeurope
```

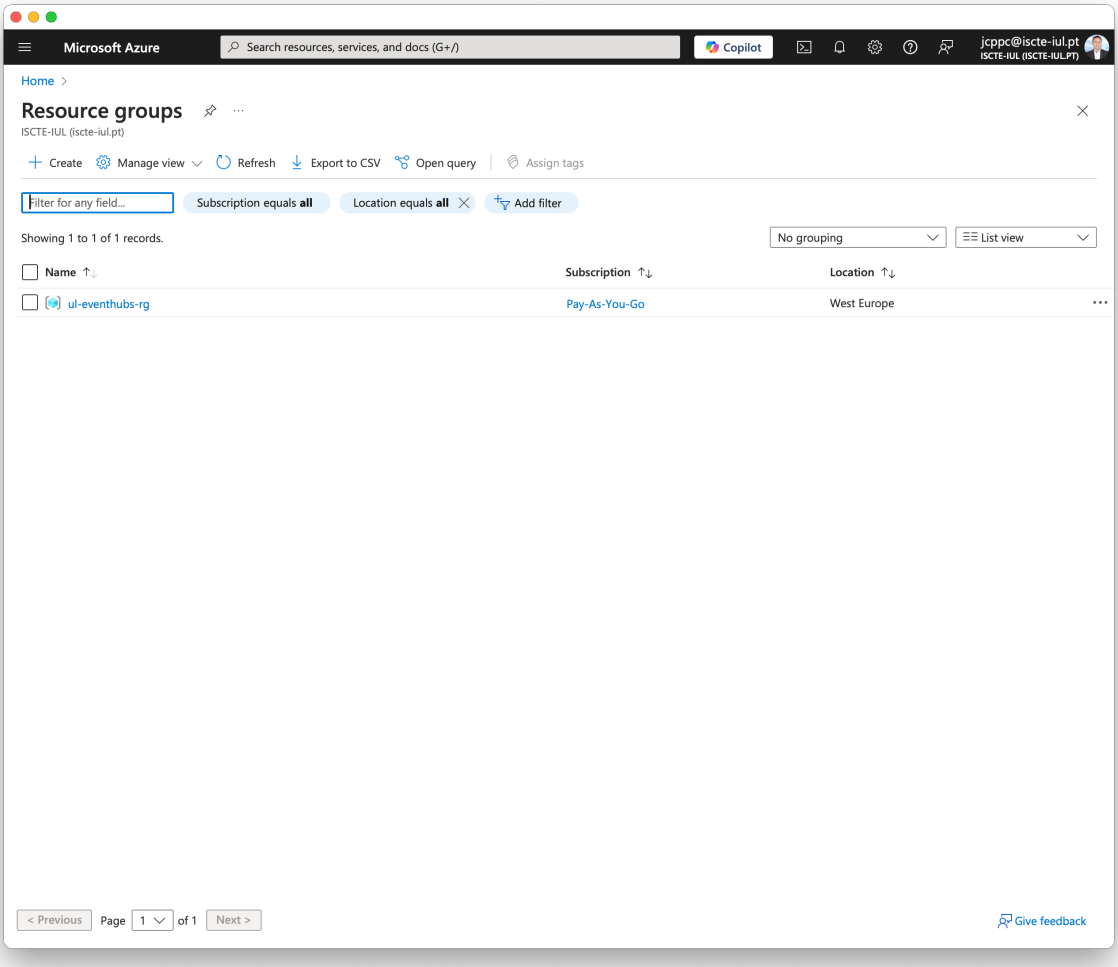

Figura 13.1.: Azure Portal - After Creating the Resource Group

13.1.6. Deploy and instantiate an Event Hub

- Define the name of your template (downloaded file `event-hub-template.json`) file in a variable. In this case, `templateFile`.
- In the same directory, run the following command, taking into account that the resource group name must be the one (`ul-eventhubs-rg`) that you created in the previous step.
- Answer the question for the project name (ex: `arit-eventhub1`), and check in the Azure Portal within the resource groups the result of this command.
- The execution of the command may take some minutes.

```
templateFile="event-hub-template.json"
az deployment group create --name ul-eventhub-template --resource-group
 ↪  ul-eventhubs-rg --template-file $templateFile
```

```
# As an alternative, you can also run the above command without defining a
 ↪  variable like this:
az deployment group create --name ul-eventhub-template --resource-group
 ↪  ul-eventhubs-rg --template-file event-hub-template.json
```

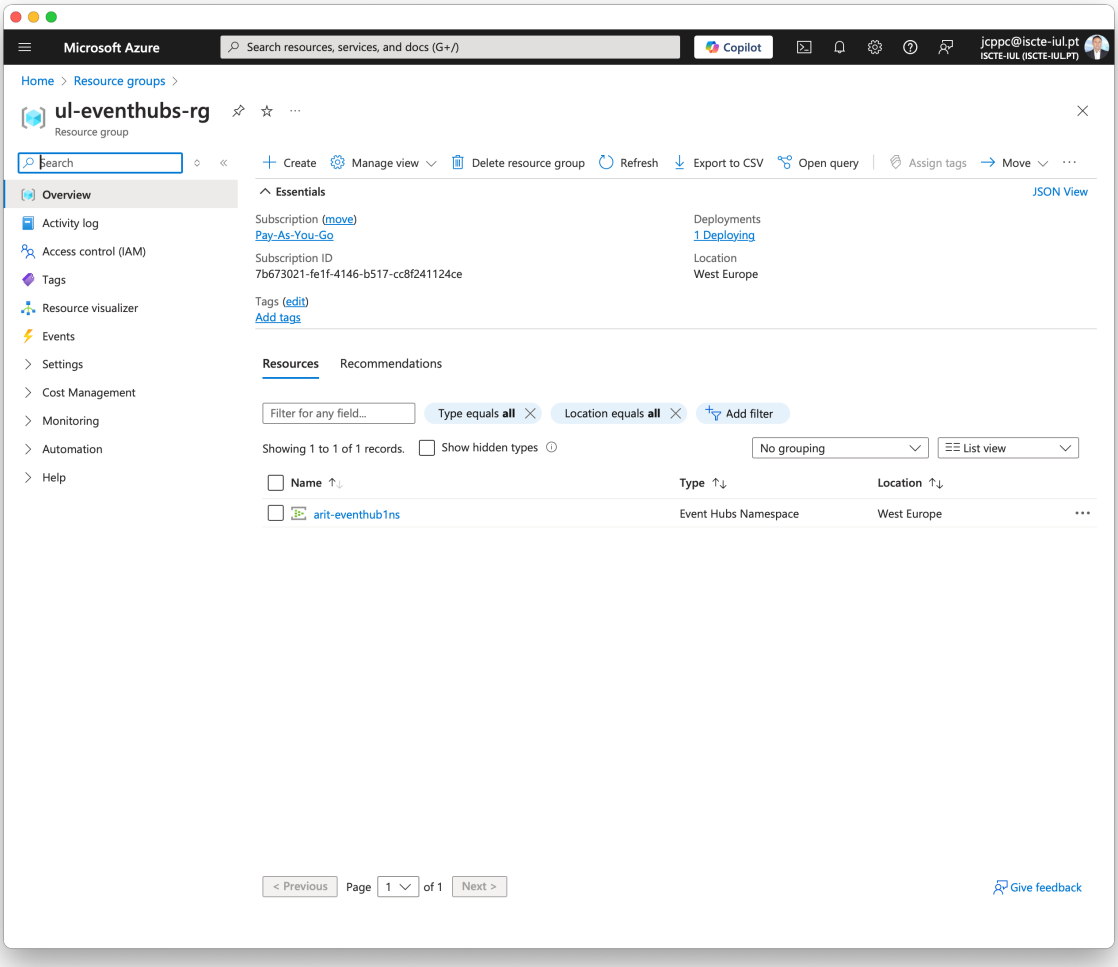

Figura 13.2.: Azure Portal - After Creating the Event Hub

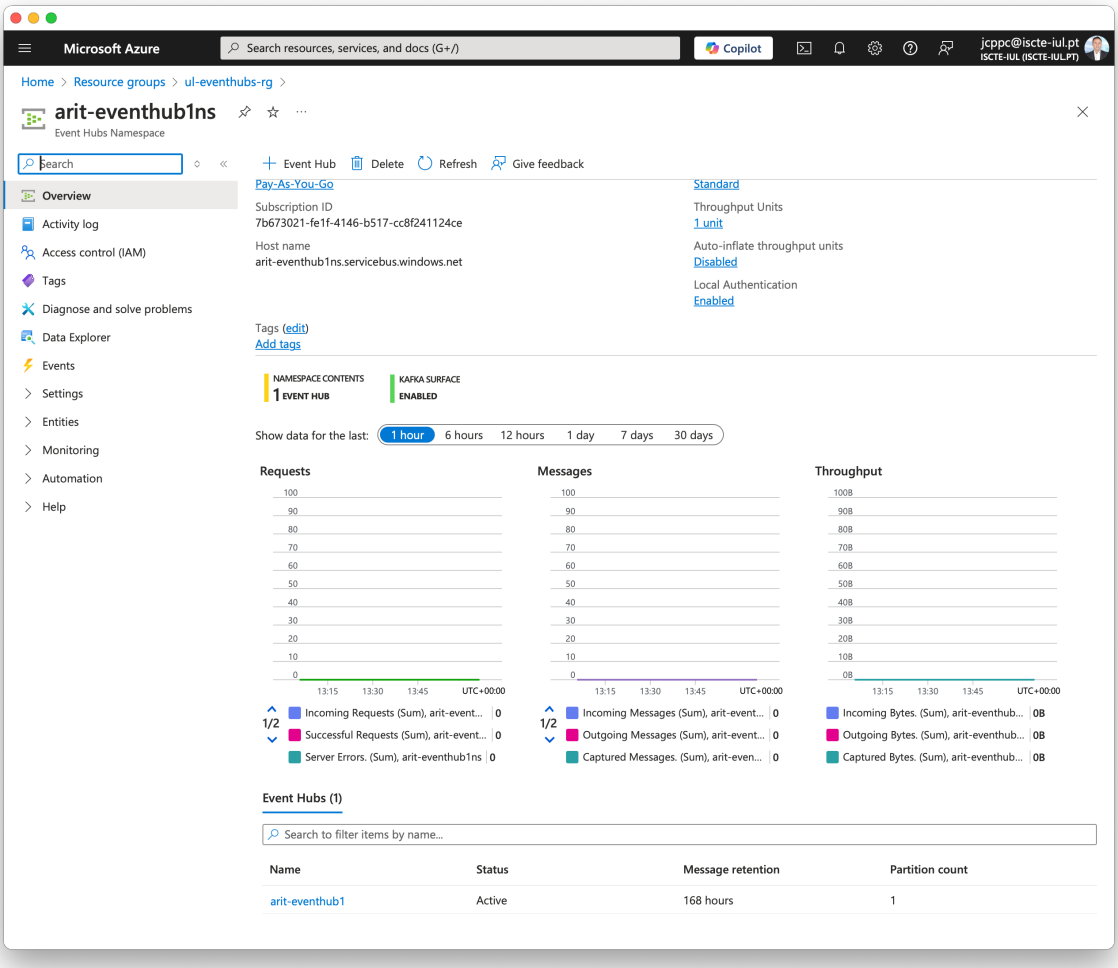

Figura 13.3.: Azure Portal - After Creating the Event Hub

13.2. Generating Data for the Azure Event Hub

13.2.1. Download and Install NodeJS

- Download NodeJS for your OS here.
- Install NodeJS.
- After installation, test it with this command.

```
node -v
```

- Positioned in the directory containing the files for this exercise, run the following command.

```
# On Windows
npm install

# If your are on Linux run
sudo npm install
```

13.2.2. Configure a connection string

- To send/receive events to/from the Azure Event Hub you need to get a connection string from the Azure Portal.
- In the Azure Portal, within the resource group and event hub namespace section, expand `Settings` and then `Shared access policies`.
- Drill down in `RootManagedSharedAccessKey` and copy the value of `Connection string-primary key` variable.

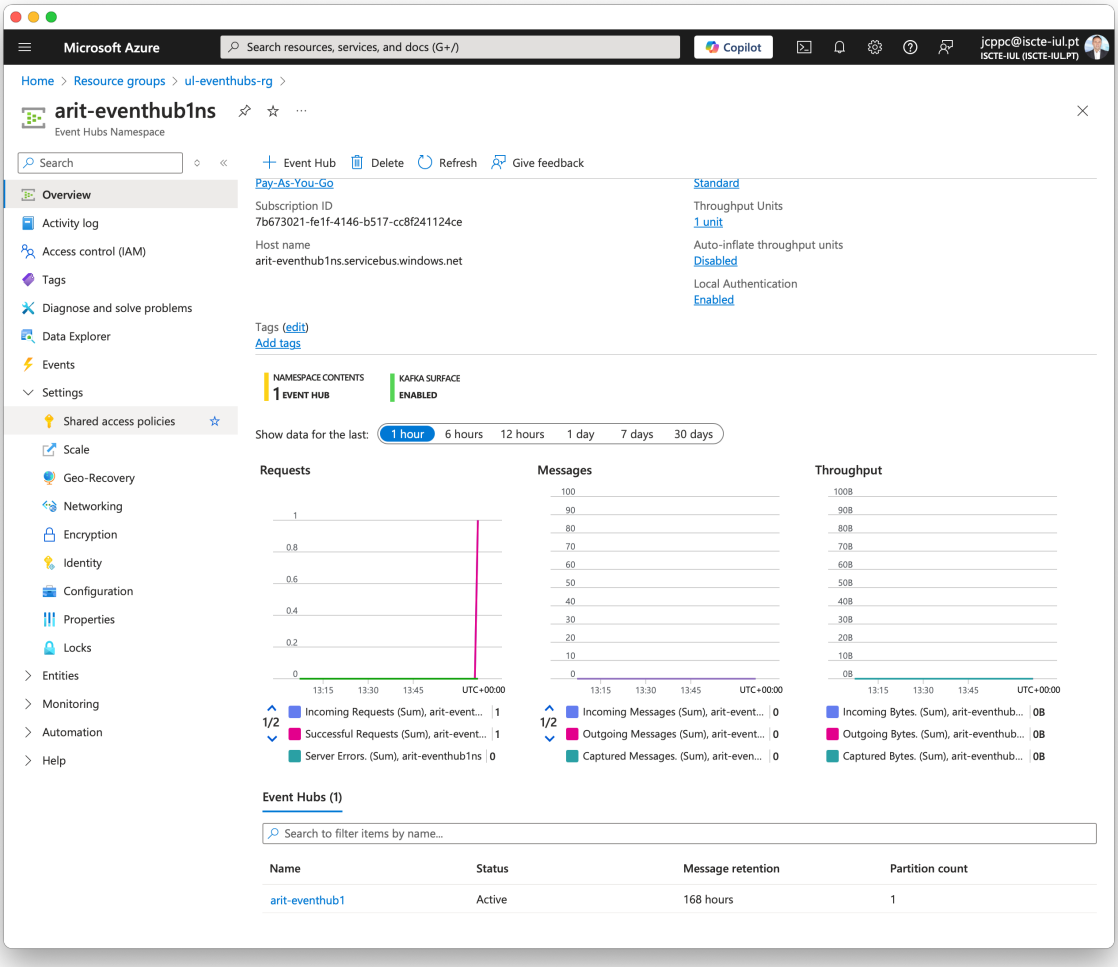

Figura 13.4.: Azure Portal - Getting the Event Hub Connection String

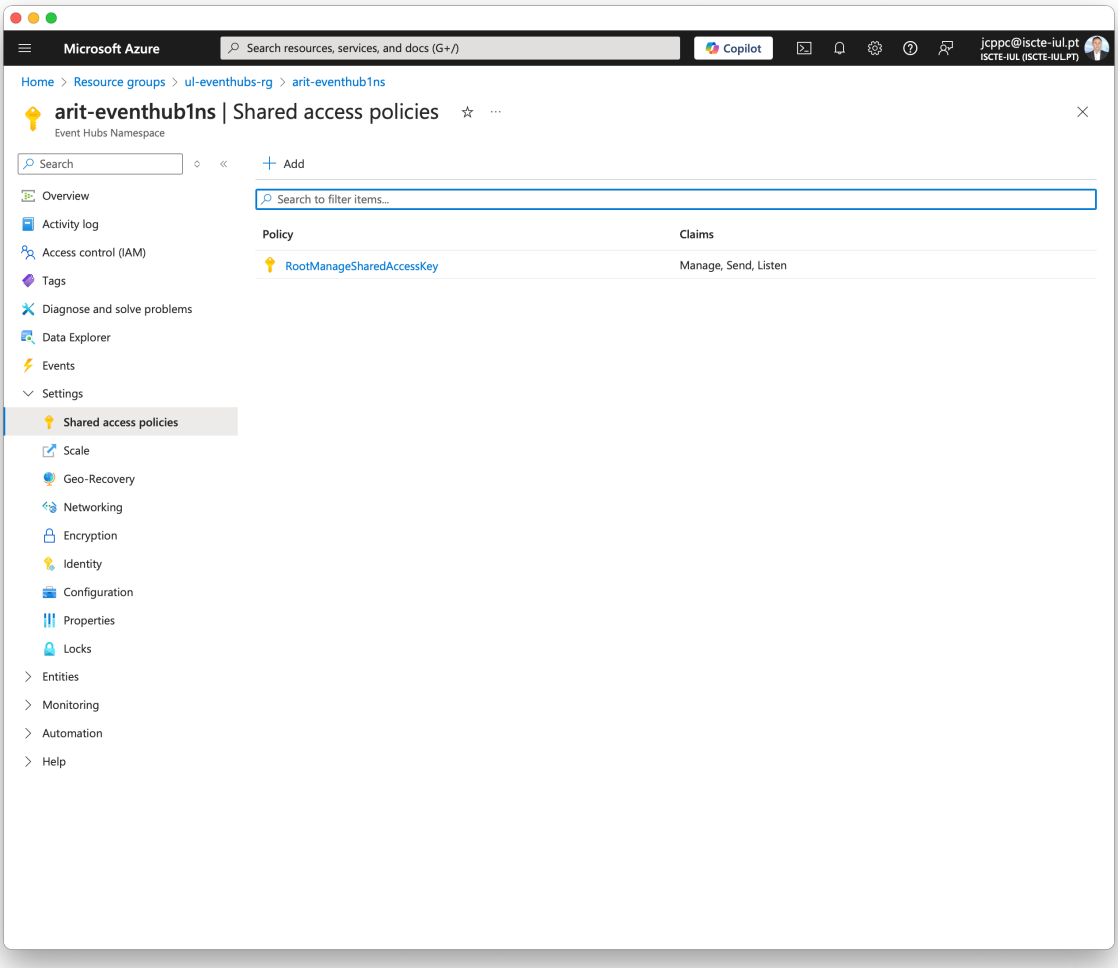

Figura 13.5.: Azure Portal - Getting the Event Hub Connection String

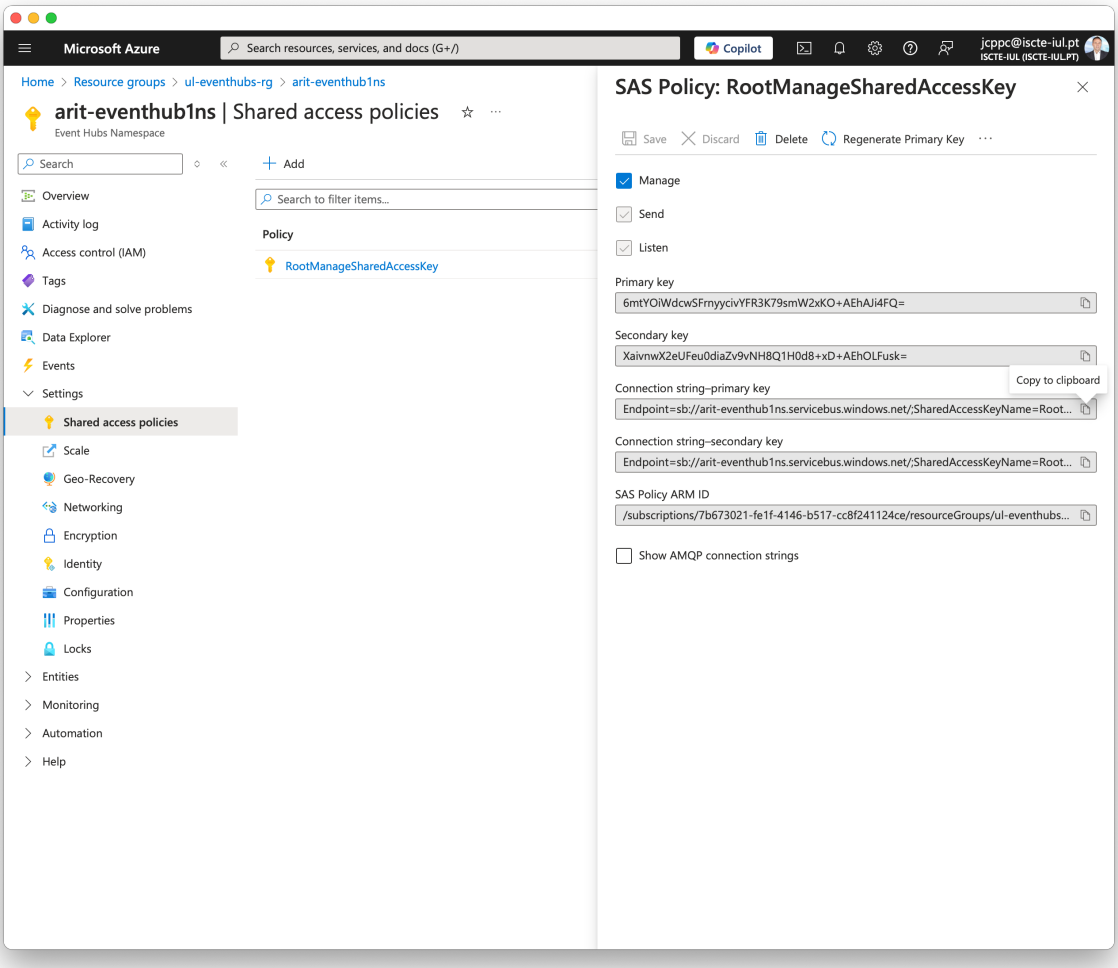

Figura 13.6.: Azure Portal - Getting the Event Hub Connection String

- Replace the connection string in the files `send-azure.js` and `receive-azure.js`. In variable `connectionString` put the value copied from the Portal in `Connection string-primary key`. Variable `eventHubName` should have the value you enter when running the template, which is the name of your Event hub.

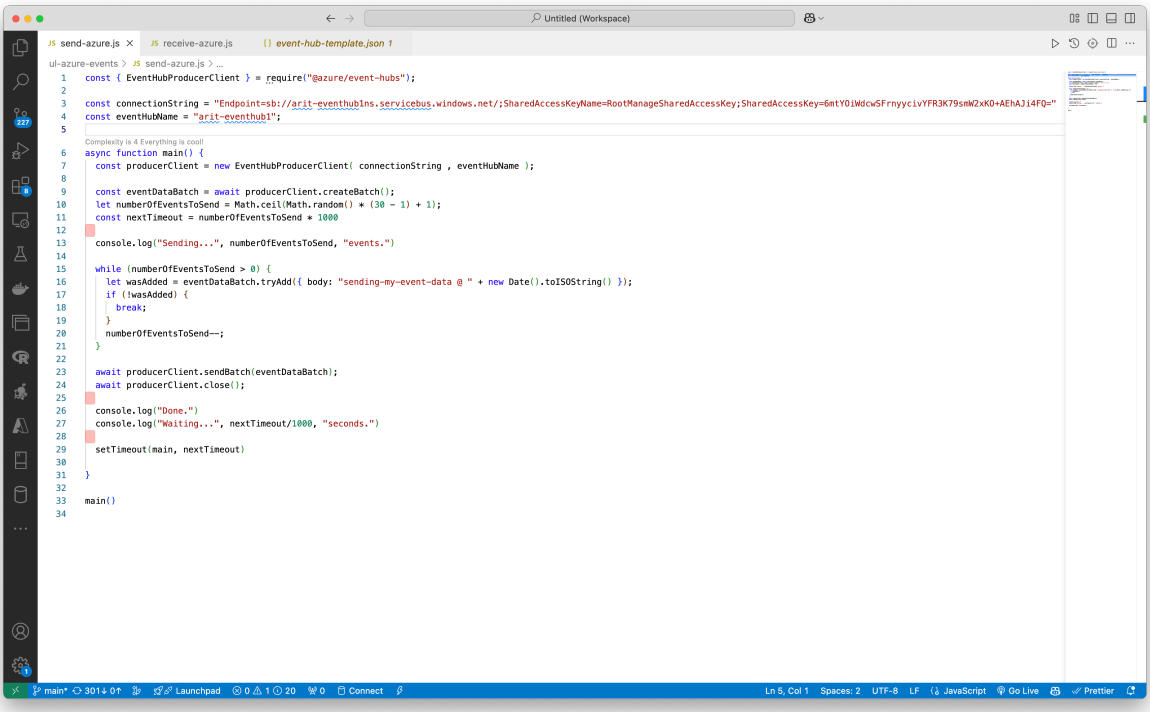

Figura 13.7.: Azure Portal - Getting the Event Hub Connection String

13.2.3. Send Events to Azure

- To send events (random data and random intervals) to Azure event Hub run the following.

```
node send-azure.js
```

- Check the results in the Azure Portal.

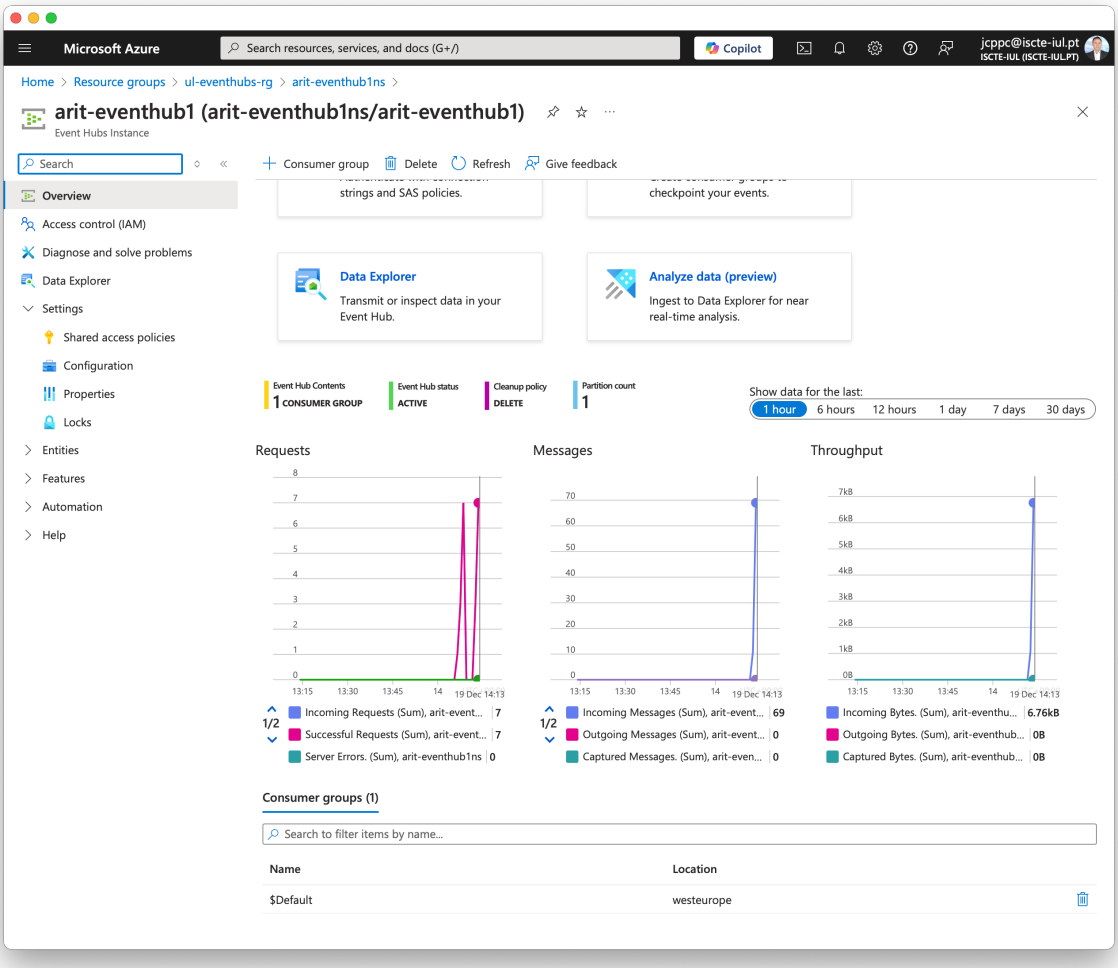

Figura 13.8.: Azure Portal - Getting the Event Hub Connection String

13.2.4. Receive Events from Azure

- To receive the same events from the Azure Event Hub run the following.

```
node receive-azure.js
```

13.3. Removing Resources

- If you want to delete everything, delete your resource group. Everything within will be removed. Use the following command.

```
az group delete --name ul-eventhubs-rg --yes
```

UNIVERSIDADE
LUSÓFONA