Cloud Computing Applications and Services

University of Minho

Guide 2
Containers, Provisioning and Deployment

2024

Imagine now that you are asked to deploy *Swap* across all Universities in Portugal. Manually repeating the steps of *Guide 1* is going to be a time-consuming and tedious process... Do not worry, in this guide we will be looking into the Docker technology (Containers) and Ansible (automation tool) for making your life easier. Side note: virtually all Cloud providers are now offering Containers as a service, with Docker being the most used solution.
In this guide, we will go through the steps of configuring and deploying *Swap* on top of the Docker platform. For this, as the Cloud Provider ([☁]), you will setup and serve a Docker service in a Virtual Machine. Then, as the Application Developer ([</>]), you will deploy a containerized *Swap* application.

Recall these symbols from previous guides:

- [☁] Tasks to be performed by the *cloud provider*;
- [</>] Tasks to be performed by the *application developer*;

Part I

Docker

Docker is the most widely-known container technology. Containers provide loosely isolated and lightweight environments (when compared to Virtual Machines) to run applications and services. Take a look at the slides accompanying this guide and stay tuned for the theoretical classes to know more about containers!

Docker documentation is available at: `https://docs.docker.com`.

I-1 Setup [☁]

1. Create 1 VM (*node1*). Use the Vagrantfile from *Guide 0*.
2. On *node1*, Install Docker (`https://docs.docker.com/engine/install/ubuntu/`).
3. Add your user to the *Docker* group to avoid using *sudo* all the time (`https://docs.docker.com/engine/install/linux-postinstall/`).

I-2 Tasks [</>]

I-2.1 Sample application

In order to get acquainted with Docker, let us begin by setting up a Sample Application. Take note of the different steps required to get the application running.

1. Configure the sample application to run in a container. For that, follow the tutorial at `https://docs.docker.com/get-started/workshop/02_our_app/`. Make sure you read the tutorial's notes to understand every command you execute.
   Note: When running the container, replace the `127.0.0.1` IP by the *node1*'s IP (i.e., `192.168.56.101`).
2. Access the application from your browser (`http://<NODE1_IP>:3000`).
3. Understand the commands `docker {image, ps, exec, stop, start, kill, rm, logs}`.
   E.g., understand and explore the command: `docker exec -ti <CONTAINER_ID> /bin/sh`
4. Add some items to your list and then stop the container. Now start again the container and access the application. What happened to your items?
5. Now stop, remove, and run again the container. What happened to your items?

I-2.2 Docker network

Contrary to the Sample Application, Swap has multiple components (e.g., database and web app). For each, we will create a separate container. In order for these containers to communicate with each other, you will need to configure a Docker Network.

1. Create a network using the following command: `docker network create <NETWORK_NAME>`

   Explore the type of networks and how they work in Docker: `https://docs.docker.com/engine/network/drivers/`
2. Use the `docker network list` and `docker network inspect <NETWORK_NAME>` commands to check details about the new network.

Note: Docker networks provide a DNS-like service for containers. This allows using *hostnames* to refer to each container instead of explicit IP addresses, thus simplifying your setup.

Question: Why should one create separate images and containers for SWAP's database and Web application server?

I-2.3 Database server container

Let us start by preparing a MySQL container. Fortunately, for MySQL, there are already public Docker images one can use:

1. Pull the official MySQL Docker image (`mysql:latest`).
   - Explore the `docker image pull <IMAGE_NAME>` command to pull the image.
   - Check your local image inventory with the `docker image ls` command.
   - Explore the image's documentation at `https://hub.docker.com/_/mysql`.
2. Create a Docker container with a MySQL installation:

```
docker run --name <DB_CONTAINER_NAME> --net <NETWORK_NAME> -p 3306:3306 -d \
    -e MYSQL_USER=<DB_USERNAME> -e MYSQL_PASSWORD=<DB_PASSWORD> \
    -e MYSQL_DATABASE=<DB_DATABASE> -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
    -v mysql:/var/lib/mysql \
    mysql:latest
```

   Note: The option `-v mysql:/var/lib/mysql` creates a Docker volume named `mysql`, which will be mounted into the container at `/var/lib/mysql` (where MySQL stores data).
3. Check that MySQL service is running and well configured:
   - Use the command `docker volume ls` to check if the MySQL volume was created.
   - Check MySQL logs: `docker logs <DB_CONTAINER_NAME>`

- Verify if the Swap database was successfully created:

  ```
  docker exec -i <DB_CONTAINER_NAME> \
     mysql -u<DB_USERNAME> -p<DB_PASSWORD> <<< "SHOW DATABASES;"
  ```

I-2.4 Web application server container

Now let us setup a container for the Swap web server application, which will connect to the MySQL container. Please refer to Guide 1 and your previous Swap installation experience for the next tasks.

1. Explore the Dockerfile reference documentation at
   `https://docs.docker.com/engine/reference/builder/`
2. Explore the Dockerfile instructions (e.g., `FROM`, `RUN`, `WORKDIR`, `COPY`, `EXPOSE`, `CMD`, `ARG`) and set up a Dockerfile to build a Docker image for the Swap's web application server.

   Use Ubuntu 24.04 as the base image for the Dockerfile (`FROM` instruction).

   Note 1: The tasks for migrating and seeding the database should be ignored in this phase.

   Note 2: Define the following argument (`ARG`) variables to set the timezone and noninteractive mode: `TZ=US` and `DEBIAN_FRONTEND=noninteractive`.
3. Use the command `docker build` to build the Swap Docker image.

   ```
   docker build -t <SWAP_IMAGE_NAME> .
   ```
4. Use the command `docker run` to run the Swap container.

   ```
   docker run --net <NETWORK_NAME> -p 8000:8000 \
      --name <SWAP_CONTAINER_NAME> <SWAP_IMAGE_NAME>
   ```

   Note: This is only an example, you should add any other additional container flags that your image may require.
5. Test your setup by accessing the app through a web browser (`http://<NODE1_IP>:8000`).
6. Do you remember the variables `<DB_HOST>`, `<DB_DATABASE>`, `<DB_USERNAME>`, and `<DB_PASSWORD>` defined at Swap's `.env` configuration file? Ensure that these can be dynamically set when running the Swap container and are not hard-coded at the Dockerfile.

Part II

Automation of processes

Docker greatly simplifies the repetitive process of deploying applications. But let us go back to your task, which is to install Swap in all Portuguese Universities. There are two main problems here:

- You still had to do some manual setups to configure and launch containers. If such is going to be repeated to deploy *Swap* across all Universities...
- Even worse, you are told that the Universities do not have docker installed at their infrastructures!

Do not worry, next we will learn how to provision systems and deploy services in an automatic and reproducible fashion.

Ansible

Ansible is an open-source automation tool used for configuration management, application deployment, intraservice orchestration, and provisioning. In this guide, you will use Ansible to automate the manual tasks done in Part I. Take a look at the slides accompanying this guide and remember the theoretical class on Provisioning!

Ansible documentation is available at:

- Ansible - `http://docs.ansible.com/ansible/`
- Ansible Getting Started - `https://docs.ansible.com/ansible/latest/getting_started/index.html#%20intro-getting-started`
- Example of Ansible playbook - `https://github.com/ansible/ansible-examples/tree/master/wordpress-nginx`

Ansible Projects

Throughout the remaining practical classes, you will be working with two Ansible projects:

- ***CloudProvider project***: will contain all the necessary Ansible playbooks to install relevant tools (e.g., Docker, Kubernetes) and services (e.g., monitoring) on your infrastructure to be used by users (e.g., App developers).
- ***AppDeveloper project***: will contain all the necessary Ansible playbooks to configure, install and deploy the Swap application resorting to different technologies (e.g., Docker, Kubernetes, etc).

Take a look at the respective projects' code provided along with this guide. For now, we will focus on using Ansible to automate the Docker installation on *node1* as part of the CloudProvider project. Next, we will work on the AppDeveloper project to automate (via Ansible) all the manual steps required to deploy Swap with Docker.

II-1 Setup [☁]

1. Create an additional VM (*myvm*). This VM will be used to run Ansible. The Swap application will be deployed on *node1*.

   ```
   vagrant up myvm node1
   ```

   Note 1: Destroy the previous *node1* VM to start with a clean VM: `vagrant destroy node1`
   Do not forget to save important files to your PC before destroying the VM!

   Note 2: Check again the Vagrantfile and notice how Ansible is being automatically installed on *myvm* by Vagrant through the `provision_myvm` script.

2. Explore the `scp` command and use it to copy the projects' files to *myvm*. For example, the following command copies the `CloudProvider` folder (and all its content) to *myvm*.

   ```
   scp -r CloudProvider vagrant@192.168.56.10:
   ```

II-2 Tasks

II-2.1 Automation of the Docker installation [☁]

Let us start by using Ansible to install and configure Docker on *node1*. Please refer to the theoretical class slides to recall Ansible-related topics.

1. Take a look at the structure of the *CloudProvider* Ansible Project. The project contains the following files/directories:

   (a) ***ansible.cfg***: This is the Ansible configuration file, which allows for adjusting certain settings. In this specific case, we are setting up the location of the inventory file, turning off the SSH key host checking, and configuring the remote user. Find more about the Ansible configuration settings at: `https://docs.ansible.com/ansible/latest/reference_appendices/config.html`

   (b) ***hosts***: This is the Ansible inventory file, which defines the groups of hosts upon which commands, modules, and tasks in a playbook operate. In this specific case, we are defining a group named *"nodes"* composed of two hosts (*node1* and *node2*). Find more about the Ansible inventory at `https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html`.

   (c) ***roles***: This folder contains the project's roles. Roles provide a structured way to organize tasks, templates, files, and variables. In this specific case, we have a role named "docker" that defines the tasks for installing Docker and their variables. Find more about Ansible roles at `https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html`.

   (d) ***install_docker.yml***: This is an Ansible playbook for installing Docker. Playbooks are lists of tasks that automatically execute for your specified inventory or groups of hosts. In this specific case, we are configuring the role *"docker"* to execute on the hosts of the *"nodes"* group. Find more about Ansible playbooks at `https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html`.

2. Recall the manual steps taken for the Docker installation on the setup of Part I and compare them with the tasks defined in the role *"docker"*. Notice the use of Ansible modules for defining such tasks (*e.g.*, modules `apt`, `file`, `command`, `group`, `systemd`, *etc.*). Find more about Ansible modules at `https://docs.ansible.com/ansible/latest/plugins/module.html`.

3. Connect to *myVM* and run the *"install_docker.yml"* playbook to install Docker on *node1*:

   ```
   ansible-playbook install_docker.yml
   ```

   Note: Notice how the playbook will execute its tasks only for the hosts that are accessible (*i.e.*, since *node2* is not accessible, Docker is installed only on *node1*).

4. Connect to *node1* and verify if Docker was correctly installed (e.g., try to run the `docker` command).

II-2.2 Automation of the database server deployment [</>]

Now, as a Swap developer, let us move on to the *AppDeveloper* project and use Ansible to automate the deployment of the Swap application with Docker, starting with its database server component.

1. The *AppDeveloper* project follows the same structure as in the *CloudProvider* project, containing a configuration file (`ansible.cfg`), an inventory file (`hosts`), playbooks (e.g., `docker-swap-install.yml`), and the *roles* folder. In this case, the inventory file defines one host group (*"dockerswap"*) with a single host (*node1*), which is where we are deploying Swap.

2. Start by inspecting the `docker-swap-install.yml` playbook. Notice how it invokes two different roles (`docker-mysql` and `docker-swap`) to execute on the hosts of the *"dockerswap"* group. Also, notice how each role has a specific tag associated (e.g., role `docker-mysql` has the tag `mysql`).
3. The role `docker-mysql` for the MySQL deployment is already provided. Start by inspecting it.
   (a) Observe how the manual steps from sections *I-2.2* and *I-2.3* are being automated via Ansible.
   (b) Observe the different Ansible modules used to work with Docker (*e.g.*, `docker_network`, `docker_image_info`, `docker_image`, `docker_container`).
   (c) Notice the Ansible *variables* used in the role's tasks (*e.g.*, `db_host`, `db_image`, `db_name`, *etc.*) and how they are defined in the *group_vars/all.yml* file. The *group_vars* folder allows for organizing each group's variables more easily. Find more about Ansible group variables at `https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html#organizing-host-and-group`
   (d) Take a look at the last four tasks of this role. What is their main purpose?

   Note: Each file inside the *group_vars* folder should be named after one of the inventory host groups and contain its respective variables. However, when we want to define common variables across all groups, we can place these inside a file named *all.yml*. In our specific case, we could have defined all these variables inside a file named *"dockerswap.yml"*.
4. Run the playbook to deploy only MySQL, leveraging the tag attributed to its corresponding role:

```
ansible-playbook docker-swap-install.yml --tag mysql
```

II-2.3 Automation of the web application server deployment [</>]

Now, we need to deploy the Swap web application server.

1. Complete the playbook by filling in the `docker-swap` role.

   Hints:

   (a) Recall the necessary steps from section *I-2.4* for building and deploying Swap's web app server.
   (b) Notice the *files* folder inside the `docker-swap` role. This folder allows you to place files that need to be copied to a remote host. Place in this folder the Swap's `Dockerfile` developed in section *I-2.4*.
   (c) Explore the *copy* module to copy a file your local machine to the remote one.
   (d) Explore the *docker_image* module to build the Swap's image from the Dockerfile.
   (e) Explore the *docker_container* to run the Swap's container.

   Note 1: Ansible has a really nice official documentation, use it to explore new modules!
   Note 2: You are building scripts that do powerful commands (*i.e.*, install packages, change system configurations). With great power comes great responsibility, use the `become` directive (for privilege escalation) only when needed!
2. Run the playbook to deploy only Swap, leveraging the tag attributed to its corresponding role:

```
ansible-playbook docker-swap-install.yml --tag swap
```

Extras

1. Modify the Swap Docker image to include the database migration and seed tasks. Note that the database seed task should run optionally (i.e., only when the user asks).
2. Explore Ansible Vault to protect passwords and avoid having these in plaintext (*e.g.*, MySQL user's password).

Learning outcomes

Part I: Hands-on experience with software container technology (Docker). Deployment of an application on top of a container platform.

Part II: Experiment with systems provisioning, deployment, and configuration management workflows via Ansible. Experiment with playbooks that hold reproducible provisioning and deployment recipes. Understand the importance of task automation and self-documentation.