Politecnico di Torino

Publish a Website with Kubernetes

Marco Iorio

December 23, 2021

License

Acknowledgments

The author would like to thank all the people who contributed to this document, particularly Paola Belmonte who created an early version of this laboratory.

Contents

1 Introduction

The goal of this laboratory is to practice with the complete workflow required to **create, package and deploy a website on a Kubernetes-based cluster**. Overall, this aims to better highlight how the different building blocks learned during the previous laboratories (e.g., containers creation, applications deployment and exposition, ...), can be combined to achieve a fully functional service.

The very same approach, then, could be generalized, to account for more complex scenarios composed of multiple microservices which cooperate together and implement complex functionalities. An example of such complex setup is provided by the Online Boutique demo application[1] made available by Google.

To show the successful completion of the laboratory, **students will be requested at the end to send** on the `#lab` channel of the course's Slack workspace **the link to access their website**.

1.1 Physical setup

During this laboratory, we will leverage a different setup with respect to the previous exercises. Specifically, we will use:

1. A **management workstation**, to prepare the website, package it as a Docker image and interact with the target Kubernetes cluster (i.e., issue commands through `kubectl`). To this end, you can start the appropriate instance available in CrownLabs or, if you prefer, use your own laptop.
2. A dedicated **slice of the CrownLabs Kubernetes cluster**, to host the deployed website and expose it to the external world. In particular, your CrownLabs account has been associated with what we call a *sandbox namespace*, that is a Kubernetes namespace you can freely access and use to deploy your own workloads. Additional information about how to configure your management workstation to interact with the sandbox namespace is available on the CrownLabs website[2]. **Warning:** given you are using a shared infrastructure, your account is associated with limited permissions. In particular, you cannot access resources outside of your namespace, or cluster-wide resources (e.g., nodes). Nonetheless, be respectful of others: do not attempt to perform operations which might damage other users or the CrownLabs infrastructure itself.
3. A **public docker registry**, to publish the docker image containing the resulting website. You can leverage Docker Hub[3] (you need to either have or create a free account), or Harbor, a registry made available by CrownLabs[4]. To access the Harbor dashboard, you need to use your CrownLabs credentials; then, click on your username (top right of the page), select the *User Profile* tab and copy the CLI secret: that is the password to use to login with the `docker login` command. To push images on Harbor, their tag shall be composed of the entire registry url, the `cloud-sandbox` repository and the student's ID as image name prefix. For instance, a student with ID `s123098` could tag an image as `harbor.crownlabs.polito.it/cloud-sandbox/s123098/website:v0.1`

2 Creating the website

To create the website published during this laboratory we will leverage *Hugo*[5]. Hugo is an open-source static site generator, which simplifies the creation of static websites, such as personal pages,

[1] `https://github.com/GoogleCloudPlatform/microservices-demo`
[2] `https://crownlabs.polito.it/resources/sandbox/`
[3] `https://hub.docker.com/`
[4] `https://harbor.crownlabs.polito.it/`
[5] `https://gohugo.io/`

blogs, documentation resources, company or product pages. It features a wide variety of predefined **templates** (a.k.a. themes), while **content can be expressed in markdown**, which is then automatically converted by Hugo into the actual *html* pages.

2.1 Scaffolding the website structure

In order to create a new Hugo website, it is possible to leverage the `hugo new site` command, which initializes a new project and scaffolds the appropriate directory structure:

```
hugo new site path/to/website
```

Looking at the content of the directory, the `config.toml` file states various configuration directives (e.g., the base URL where the website will be published, the theme to use, ...). The `content` directory contains the markdown files about the different website sections, `static` stores all static content, including images, CSS, javascript, ..., and `themes` embeds the files about the selected theme (once downloaded).

To prevent issues when building the final website (e.g., missing CSS files), let explicitly configure the website base URL to be relative:

```
sed -i 's|baseURL = "http://example.org/"|baseURL = "/"|' path/to/website/config.toml
```

Note: it is suggested to initialize the resulting directory as a Git repository, in order to keep track of the different modifications and simplify the download of the selected theme. The Git repository can be initialized as follows (to be issued from `/path/to/website`):

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

2.2 Configuring a theme

With the initial structure in place, it is now time to select a theme for our website. Available themes can be explored on the official page[6]: you can freely choose the one you prefer. The following description assumes you selected the *Learn* theme, but the steps are definitely similar regardless of the choice (additional information is provided by each theme's subpage).

Once selected, it is necessary add the theme to the website directory structure. While it is possible to download it from the repository as a zip file and extract the content in the `themes` directory, we suggest to leverage **git submodules**. Git submodules allow to store a git repository (e.g., the one of the theme), as a subdirectory of another git repository (e.g., our website), though maintaining separate version control information. Overall, this allows to easily update the theme files whenever a new version is available upstream, while decoupling them from the main website repository.

A new submodule (for the *Learn* theme) can be initialized as follows (the content of the theme is downloaded in the `themes/hugo-learn` folder):

```
git submodule add https://github.com/matcornic/hugo-theme-learn.git themes/hugo-learn
```

Finally, it is necessary to configure the theme in the `config.toml` file, adding the appropriate line:

[6] `https://themes.gohugo.io/`

```
theme = "hugo-learn"
```

Tip: this might be a good moment to commit your changes.

2.3 Adding the site content

Now, it is possible to proceed adding the actual content to your website. Each section of the website is represented by a different folder inside the `content` directory, and it shall contain one or multiple markdown files corresponding to the different section pages. One page shall always be named `_index.md`, and it represents the root page of the given section. **Warning:** the final appearance of the website depends on the theme selected during the previous step. Different themes might also rely on different content structures: refer to their respective documentation for additional information.

To create the content files, you can leverage once more the appropriate `hugo` command, which takes care of configuring the necessary metadata information at the beginning of each one (based on pre-defined templates). Specifically, to create two sections (e.g., `basic` and `advanced`), each composed of two pages (e.g., `first` and `second`), it is possible to proceed as follows:

```
# Modify the template to avoid marking the pages as draft
sed -i 's/draft: true/draft: false/' archetypes/default.md

# Create the root "index" file
hugo new _index.md

# Create the "basic" section
hugo new basic/_index.md
hugo new basic/first.md
hugo new basic/second.md

# Create the "advanced" section
hugo new advanced/_index.md
hugo new advanced/first.md
hugo new advanced/second.md
```

With the website structure in place, you can add the actual content, editing the different files. As mentioned earlier, each file starts with a metadata section, which includes the title and possible additional information (e.g., a priority, to enforce a different sections order). The content can be written below the metadata part, leveraging the markdown syntax to express formatting options.

2.4 Building the website

While writing the different contents, it is often useful to preview the resulting website. Hugo offers the possibility to start a local webserver by means of the `hugo server` command (to be issued from the root directory of the website): the site can then be accessed at `http://localhost:1313`, and every change in the content is automatically reflected to the preview.

Differently, to generate the files (i.e., *html* pages, *CSS* stylesheets and *JavaScript* scripts) to be published, you can leverage the `hugo -v` command (once more from the root directory of the website). The outcome of the process is made available in the `public` subdirectory.

3 Packaging the website as a Docker image

It is now time to prepare the **Docker image** packaging the website created in the previous section, so that it can be later deployed on a Kubernetes cluster. To this end, it is necessary to create the appropriate `Dockerfile` (within the root directory of the website). Overall, the instructions in the Dockerfile shall perform two main actions:

1. **generate the website files** to be published (by means of the `hugo -v` command);
2. **serve the resulting content** (i.e., by means of a web server, such as *nginx* or *apache*).

The preparation of the Dockerfile with the appropriate instructions is left to the reader, which shall leverage the concepts learned during the previous laboratories.

Suggestion: in order to reduce the resulting image size, it is good practice to leverage the **multi-stage build** approach. In a nutshell, a first container including all the necessary software is in charge of all the preparatory steps (e.g., build the website), while the resulting artifacts are copied to a second image, including only the applications required at runtime (e.g., the webserver).

Once the Dockerfile is complete, it is possible to build the image and verify it works correctly by executing it as a local container. You should be able to navigate the resulting website.

If the website works properly, the last step involves **pushing** the image to a public repository, such as Docker Hub or Harbor, so that it can be lated retrieved by Kubernetes. **Note:** pay attention to tag the image correctly: in case it is pushed to Docker Hub, it shall be prepended with your username, while for Harbor you need to specify the entire registry url, the `cloud-sandbox` repository and prefix the image name with your PoliTO ID. Do always specify a version tag for your images, to ensure the correct version is used in production.

4 Deploying the website on the Kubernetes cluster

With the Docker image pushed on a public repository, it is finally possible to deploy the website on the student's **sandbox namespace** on the Kubernetes cluster. The definition of the appropriate manifests is left to the reader, which should determine the types of resources (and their configuration) required to **deploy the website and make it accessible** to external clients. To select the appropriate resource configuration, let consider the following requirements:

1. the resulting webpage should be accessible at `https://sxxxxxx.sandbox.crownlabs.polito.it` (where `sxxxxxx` represents the student's ID);
2. the certificate of the website should be valid, to avoid warning messages from the browser; to simplify this task, each sandbox namespace already contains the `crownlabs-sandbox-secret` secret, including a trusted wildcard certificate;
3. the Kubernetes nodes are characterized by private IP addresses, and LoadBalancer services cannot be used;
4. (bonus) the application should survive the failure of a Kubernetes worker node, without notice from possible clients.

To show the successful completion of the laboratory, **students are requested to send** on the `#lab` channel of the course's Slack workspace **the link to access their website**.