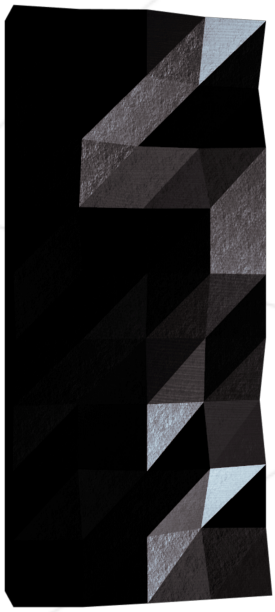
β

This project is currently under beta-test, until it is validated by a fair number of students in the 42 community. Please report any typo, incoherence, inconsistency, error, using the form https://tally.so/r/3lVDJo



IaC-1

Infrastructure as Code

Charles Garoux chgaroux@student.42lyon.fr

Summary: architect a Cloud infrastructure and deploy it automatically with modern tools

Version: 1.2

Contents

Chapter I

Foreword

The yak is a large species of domesticated ruminant with a long fleece, native to the Himalayas. The female yak is called *dri* or *drimo* by the Tibetans and *nak* by the Sherpas.
Source: Internet

In DevOps, we say "cattle, not pets." So in this image, we have a "pet" and "cattle." Hint: IaC is your buddy.

Chapter II

Introduction

This is the first in a series of projects designed to introduce the DevOps technique of Infrastructure as Code (IaC).

You are going to be both the architect and the magician for this project.
You will design the cloud infrastructure for a provided web application, and you will work your magic when your infrastructure automatically deploys, leaving you time to grab a coffee.

Chapter III

Scenario

You are working at a company as a developer, and you have a boss who has no sense of priorities.

One morning, the boss shows up in front of your office with a coffee in one hand and a USB stick in the other. A conversation begins:

- The boss starts: "My nephew coded something for his internship." He hands over the USB stick and continues: "Put it in the cloud, and it needs to be robust when there's traffic."
- You begin to reply: "But..."
- The boss cuts you off: "Perfect, you've got three months," and puts the stick on your desk.

No sooner have you looked at the USB stick than he disappears from the open space.

When you open the USB stick, you find a classic web application with basic but existing documentation.

Chapter IV

Platform Choice

The school does not provide the servers required for the project. You will have to provide your own infrastructure.

Fortunately, most cloud providers offer trials in the form of credits or services with free quotas. These trials can make it possible to complete the project at a lower cost. With sensible and smart usage, you can get by for just a few euros. As a reminder: "cattle, not pets"—shut down your servers when you're not working.

The author of the project has done so at a cost of less than five euros.

There are many different cloud platforms. Here is a non-exhaustive list of well-known platforms:

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure

The choice of cloud provider is yours, and this decision is part of the project. During the evaluation, you will explain your choice of platform.

This freedom is limited to solving the subject's problems without cheating.

Chapter V

Choice of IaC Technology

In addition to the platform, this project gives you the choice of deployment technology for Infrastructure as Code.

Different tools exist, some of which are proprietary to cloud platforms for deploying their services:

- CloudFormation by Amazon Web Services
- Azure Resource Manager by Microsoft Azure
- Cloud Deployment Manager by Google Cloud

Other tools can deploy across different providers:

- Terraform by HashiCorp
- Pulumi by Pulumi

The tools presented here are well-known examples, so it is up to you to choose the one you want to use. Make sure that the technology you choose matches the project's requirements.

`Terraform and Pulumi will most likely match the project's requirements.`

The choice of technology is yours, and this decision is part of the project. During the evaluation, you will explain your choice of technology.

`This freedom is limited to solving the project's problems without cheating.`

Chapter VI

Mandatory Part

You must be able to deploy a cloud infrastructure to host the application provided as a resource for the project.

The cloud infrastructure must be provisioned using an Infrastructure as Code (IaC) tool.

VI.1 Cloud

The application must run continuously on at least two separate servers, as far as the provider allows, preferably on different server farms.
Make it visible in the application which server is serving the current page (e.g., display the server's IP address).
It is permitted to modify the source code of the provided application to meet this requirement.

Set up a system to distribute the load evenly across servers.

The user should remain logged in for a normal session duration, even if they refresh the page.

Any addition, modification, or deletion made in the application must be immediately reflected in the other instances. A delay of a few seconds is acceptable, but must be justified.

The infrastructure must be resilient to various failures, such as the loss of a server instance or high load (CPU/RAM), ensuring high availability of the application.
We recommend automating high-availability tests using a script, tool, or another appropriate solution.

The infrastructure must be able to scale by adjusting the number of server instances according to their capacity to handle the load.
We recommend automating scalability tests using a script, tool, or another appropriate solution.

Costs should be optimized as much as possible, bearing in mind that you will need to justify your choice of cloud services and options.
It is up to you to balance cost savings with infrastructure capacity.
It is advisable to provide a configuration option offering minimal cost, to limit development expenses.

Security measures must be implemented to prevent access to resources or services that should not be publicly accessible. Even in the cloud, standard security rules must be followed.
Secrets must be protected in the best possible way. There should be no secrets stored in the code repository, scripts, or unsecured files.

The administrator must be alerted in the event of malfunctions, such as application inaccessibility or unresponsive servers.

VI.2 Infrastructure as Code

The code should be as modular and reusable as possible.

The deployment region must be easy to select, for example, by being defined at the root of the configuration.
This selection should be possible with simplified naming, for example, using names like "EU" or "Paris" rather than identifiers like "eu-west-3".

The capacity of server and database instances must be easy to select, for example, by being defined at the root of the configuration.
It should be possible to make this selection using simplified naming, for example, by using terms such as "small", "medium", or "large" rather than technical identifiers such as "e2-standard-1".

Global static values must be modifiable by the end user to meet their needs. They should be accessible without requiring modifications to the deployment code, for example, for the selectable server type, the email address for alerts, etc.
The file format used to store them is free, as long as it is compatible with the technology used.

VI.3 Documentation & accessibility

One or more schemas must be created to describe the structure of the infrastructure.
The infrastructure may vary depending on the selections made during deployment, such as the number of server farms, for example. In this context, it is acceptable to provide a schema representing the default infrastructure.

The application and infrastructure should be deployable with as few actions and as little technical knowledge as possible.

This implies documenting the few steps necessary for deployment, such as the commands to execute, retrieving secrets, installing secrets, etc.

The application must be operational after deployment, with no further action required.

Chapter VII

Bonus

There are lots of possibilities, so have fun exploring and experimenting!

Here are some ideas:

- Deploy the application in multiple regions around the world, with content synchronization between regions.
- Integrate Chaos Engineering to test infrastructure resilience.
- Deploy a set of relevant alarms and metrics grouped together on a dashboard.
- Add a domain name and enable HTTPS protocol to access the application.
- Add log management functionality (the application already produces logs).
- Add an automatic database backup and restore system.
- Create careful documentation using a dedicated tool, such as Docusaurus, for example.

Chapter VIII

Elimination case

It is strictly prohibited to have one or more secrets (passwords, API keys, etc.) in plain text in the repository. Furthermore, it is strictly forbidden to include them in a script or file that would be accessible after the deployment is completed.

Chapter IX

Forbidden

Elements that hinder the pedagogical dynamics of the project are strictly forbidden. Here are some examples of platforms, services, and technologies that are prohibited:

- PaaS: Vercel, Google App Engine, Amazon Elastic Beanstalk, Azure App Service, ...
- Serverless: AWS Lambda, Google Cloud Functions, Azure Functions, ...
- Orchestrators (managed or not) managing server clusters: Kubernetes, Docker Swarm, Nomad, OpenShift, ...

Although orchestrators are prohibited, the use of Docker Compose is allowed, as it cannot manage server clusters.

The use of external modules to deploy infrastructure is prohibited. For example, using a Terraform module to deploy a complete network on AWS from the Terraform Registry or Github is prohibited.

Modifying the provided application with the intention of deceiving the evaluator by displaying false information is obviously prohibited.

Any attempt to defy these prohibitions will result in a -42 to the project.

If you feel that something compromises the pedagogical dynamic, it is probably forbidden. If in doubt, please contact the educational team and/or the project author.

Chapter X

Return and peer-evaluation

The assignment is entirely submitted via a Git repository which must contain:

- The code used for deployment
- The modified application
- At least basic documentation so that the evaluator can deploy it themselves
- At least one diagram describing the infrastructure
- Everything needed for the project and the correction