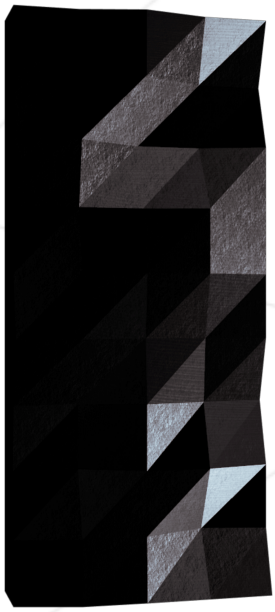

dontpanic_live

It's impossible that the Magrathean whale is alive, somehow we will do it.

Summary: Fantastic slice for any project out there. Let's plug the game into the real world, where humans live.

Version: 1.0

Contents

Chapter I

Foreword

“The price of reliability is the pursuit of the utmost simplicity.” C.A.R. Hoare, Turing Award lecture

Software systems are inherently dynamic and unstable. A software system can only be perfectly stable if it exists in a vacuum. If we stop changing the codebase, we stop introducing bugs. If the underlying hardware or libraries never change, neither of these components will introduce bugs. If we freeze the current user base, we’ll never have to scale the system. In fact, a good summary of the SRE approach to managing systems is: "At the end of the day, our job is to keep agility and stability in balance in the system."

Site Reliability Engineering: How Google Runs Production Systems, https://sre.google/sre-book/table-of-contents/

Chapter II

Introduction

So far, you've been testing and running your game in your local environment. This is a good thing, we can directly implement and test the code, doing it several times and seeing what happens. When you are working with more people and at the same time you need to ensure the consistency of your program's operation, we can no longer rely on believing that the program can run, we must guarantee that it will actually work.

We're talking about serving your application to more players – or as we say, "putting it into production" –, with that, some problems arise. For example, unreliable systems can quickly erode users' confidence, so we want to reduce the chance of system failure. For that, in this project, you will get to know the wonderful world of putting your application into production in an automatic way. So, you will interact and configure a server that will serve your application to the world.

We recommend that you read these documents, to start to understand better about the subject: 1 and 2.

Chapter III

General instructions

- You won't have to modify other parts of your game, but add the necessary part to carry out the deployment process. However, you may want to change your Docker images.
- This is probably a new thing for you, we encourage you to read a lot of documentation on the subjet.
- You will have to use the server provided by 42 São Paulo to implement your application.
- There are several tools that will help you in the process, but we will focus on using the ones that GitHub provides, due to the ease and documentation available. Therefore, you will always have to use it.
- Be creative and use your brain.

Chapter IV

Mandatory part

IV.1 Deployment features

- You must implement a CI[1] using GitHub Actions[2].
- This CI should run at least the tests you've been building along your game, in the previous documents.
- The name of the file that describes your CI workflow must have a meaningful name.
- All your workflows must run using 'ubuntu-latest'.
- You can just use "uses" directive in your workflow with "actions/checkout*"
- You need to host your own runners on the server provided by 42 São Paulo and make sure that the workflow runs it.
- You must ensure that everything going to the default branch (normally main) of your repository is working properly. That means you shouldn't commit to the default branch directly, you should make a Pull Request which will be analyzed by CI and just merge the code into the default branch if everything is ok. Yes, this will be checked in the evaluation.

- You will implement a CD[3], which will deploy your code on 42 São Paulo's servers.
- When we say "deploy" we mean that the code that will pass through the CD should automatically be applied and reflected to the world.
- You can use any tool, but we recommend that you KISS and use Shell Script for this process.
- The CD will only run when you create a lightweight Git tag. After that, you should not touch in anything on the server to properly apply your modifications.
- You should also use the Git tag to version your code in an organized way. Your code versions should follow the following pattern:
    - minor releases: Includes some new features and functions.
      v1.0 -> v1.1

[1] https://www.redhat.com/topics/devops/what-is-ci-cd
[2] https://github.com/features/actions
[3] https://www.redhat.com/topics/devops/what-is-ci-cd

- major releases: Major new features and/or large architectural changes. v1.1 -> v2.0

`The application should be up and running for the evaluation.`

Chapter V

Bonus part

For the bonus to be valid, the mandatory part must be perfect.

Cool, you managed to implement a CI and CD and now your game is available to the world. Amazing! Whenever we want to implement new things, we can get them up and running quickly. Fortunately our users love our game and every now and then when we need to perform a deployment our service drops for a few seconds and maybe minutes and our support gets full, also users get mad. Some of them even still are using the previous cached failed request, which makes them out for a bit more time than actually they should be.

For this bonus, you will have to use Docker Swarm[1], with the containers that already exist in your project, but scaled to two replicas each. Now, whenever we create a Git tag and upload a new version, the service is still available because Swarm will balance the load and ensure that there is always something to access. This means that you will not only keep one container of your program running, but at least two to ensure high availability and that when a deployment is performed, you always have an active process that can be accessed around the world.

To wrap it up, use Docker Swarm to have at least two replicas of each of your containers and ensure that when a deployment happens, your service is not unavailable. You need to send tests to assure the evaluator of this behavior.

[1] https://docs.docker.com/engine/swarm/

Chapter VI

Turn-in and peer-evaluation

VI.1 Turn-in

Turn in your work on your repo Git. Only the work included on your repo will be reviewed during the evaluation.

VI.2 Peer-evaluation

Another group will evaluate your game and your code and will positively welcome the quality of it.