DATA2410 - Networking and cloud computing

Portfolio 2: Showtime

Task 2: A multiplayer game

Group 16
Andreas Torres Hansen (s338851)
Anders Hagen Ottersland (s341883)
Uy Quoc Nguyen (s341864)

Deadline:
May 23th 2021

Abstract

This document describes the details of our implementation on this project. This project is one of two projects given on the course DATA2410 - Networking and cloud computing that counts towards our final grade. In this project, we were supposed to implement a solution for one of two tasks, a webshop using a REST API or a multiplayer game with gRPC. Our team chose the latter due to our enthusiasm with games. Both tasks had requirements in terms of a list of user stories and some stretch goals. This document will go through the given tasks and how our implementation solves these goals in details. Before, we start off with the details we will go through the most exciting part first, namely how to run the implementation, our snake game, in the first section. Furthermore, this project can also be found on Github.

Contents

1 How to run

In this section, we will give a detailed step for step on how to start up our snake game. This demonstration is performed on WSL 2 with Windows powershell distro on a Windows 10 computer due to the project has been developed using Windows 10 with PyCharm. Thus, since WSL 2 with Ubuntu does not have a GUI we will be using Windows PowerShell distro on WSL 2 instead. We suspect that if the program is ran on a UNIX system it should be possible to run the game with similar commands, by just replacing `py` with `python3`. Moreover, the python version installed on this computer is 3.9.4 with pip 2.1. In addition, it would be required to have Docker and Docker Compose are installed on the computer following this demonstration.

1.1 The game

Firstly, we need to start the server! Per required from the task, there is a Dockerfile included in the project folder. Start off by building a Docker image from the Dockerfile with

```
> docker build -t name:tag .
```

the `name:tag` could be any name you want to give the image. In our case, we chose to simply call it "snake-service". Hence, our command is therefore

```
> docker build -t snake-service .
```

Now that the image has been built:

```
> docker image ls
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
snake-service   latest    3f3c839e8012   2 minutes ago    938MB
```

We run our game-server container with

```
> docker run --rm -it -p 50051:50051 --name snake-service snake-service
Server is listening...
```

The server is blocking. Which means that you cannot execute additional commands on this terminal. Therefore, open up a new terminal and start the game from the project folder with

```
> py client.py
```

After executing the command above you should have the following window open.

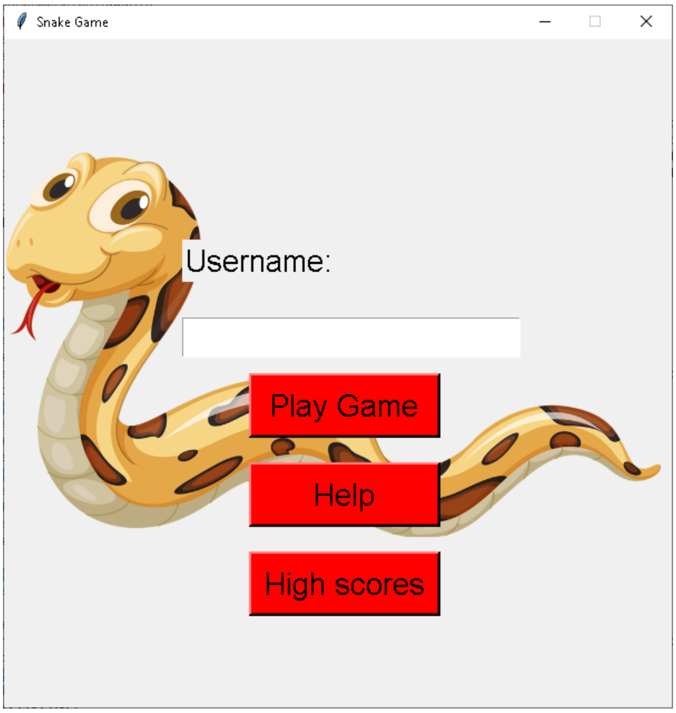


Figure 1: Index page of the snake-game developed on this project.

Because we have implemented a backend to store high scores it would be necessary to provide a username you want to play as. After you have provided the GUI with a username you can either hit Enter or click on “Play Game” button to start playing our snake game.

There are additional buttons presents on the window above as well. The “Help” button will give you a basic overview on what the game is above and provide you with instructions on how to play the game. Additionally, the “High scores” button will give you a list of high scores of different players that has played this game. High scores are stored on a SQL server running on google cloud. Hence, the list contains some of our high scores.

1.2 The game with monitoring

Now, the project also includes a docker-compose.yml file that also starts up containers pulled from the web as well as building the.

```
> docker-compose up
```

You will build and run Prometheus, cAdvisor and Grafana containers as well as the game-server. The three formerly mentioned containers are used for monitoring the resource usage as specified by one of the Stretch goals of the task. Prometheus is used for scraping metrics from cAdvisor, wheras cAdvisor is getting resource and network traffics from all the running containers. Grafana is used for visualisation. All of these containers can be accessed with http://localhost:9090/, http://localhost:8080/ and http://localhost:3000/ on a web browser respectively, after the containers are up and running.

We tend to use Grafana to monitor our resource usage. Head to http://localhost:3000/ on your favorite browser and you will be met with the page shown below:

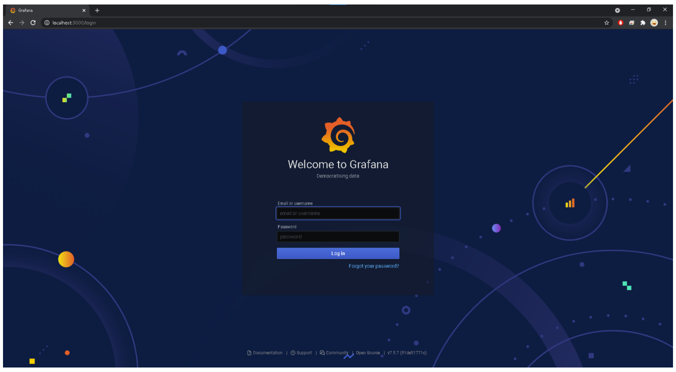
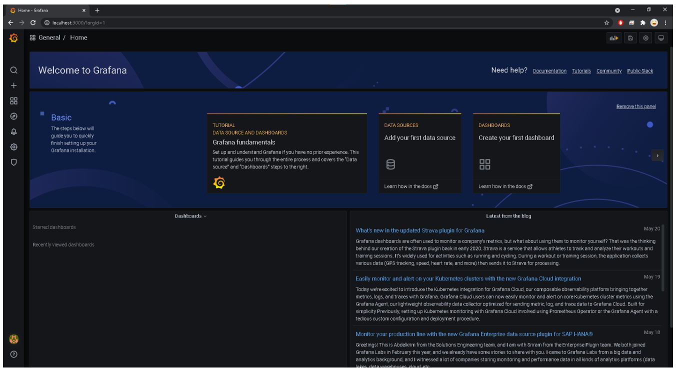


Figure 2: User interface of Grafana after running its container.

Sign in with `username:` *admin* and `password:` *admin* then skip creating a new password. You should be seeing the following



The steps below will guide you to quickly finish setting up your Grafana installation.

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

What's new in the updated Strava plugin for Grafana

Grafana dashboards are often used to monitor a company's metrics, but what about using them to monitor yourself? That was the thinking behind our creation of the Strava plugin back in early 2020. Strava is a service that allows athletes to track and analyze their workouts and training sessions. It's widely used for activities such as running and cycling. During a workout or training session, the application collects various data (GPS tracking, speed, heart rate, and more) then sends it to Strava for processing.

Easily monitor and alert on your Kubernetes clusters with the new Grafana Cloud integration

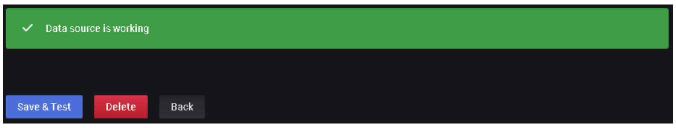
Today we're excited to introduce the Kubernetes integration for Grafana Cloud, our composable observability platform bringing together metrics, logs, and traces with Grafana. Grafana Cloud users can now easily monitor and alert on core Kubernetes cluster metrics using the Grafana Agent, our lightweight observability data collector optimized for sending metric, log, and trace data to Grafana Cloud. Built for simplicity Previously, setting up Kubernetes monitoring with Grafana Cloud involved using Prometheus Operator or the Grafana Agent with a tedious custom configuration and deployment procedure.

Monitor your production line with the new Grafana Enterprise data source plugin for SAP HANA®

Greetings! This is Abdelkrim from the Solutions Engineering team, and I am with Sriram from the Enterprise Plugin team. We both joined Grafana Labs in February this year, and we already have some stories to share with you. I came to Grafana Labs from a big data and analytics background, and I witnessed a lot of companies storing monitoring and performance data in all kinds of analytics platforms (data lakes, data warehouses, cloud, etc

Figure 3: Grafana after signing in.

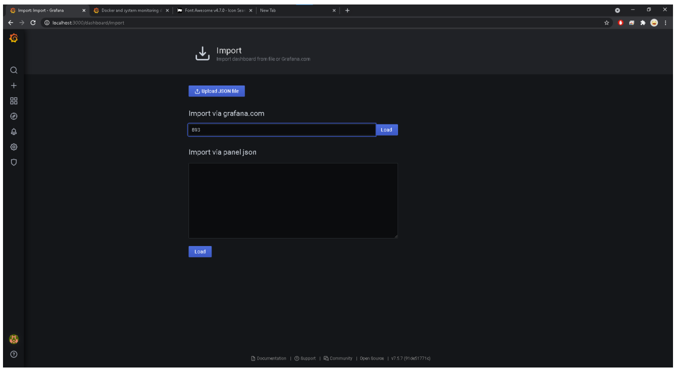
From here click on ⚙ on the veritcal navigation bar on the right and choose 🛢 Data Sources, then 🛢 Add data source. Choose the Time series database prometheus and change the "URL" to "prometheus:9090" then go ahead and click "Save & Test" at the bottom of that page. If everything is going well, this should pop up:



✓ Data source is working

Figure 4: After following the steps above you should get this message after clicking "Save & Test".

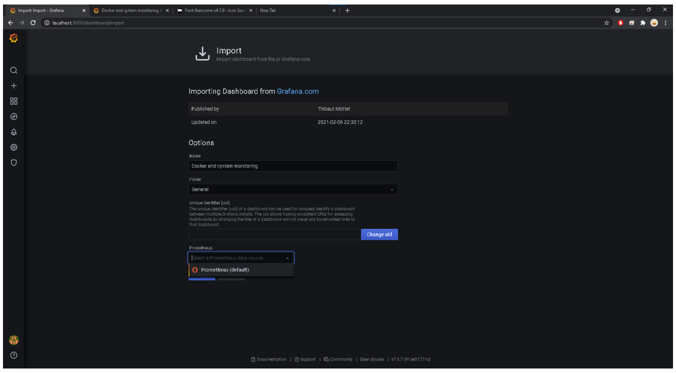
Then, click on the ✚ icon on the vertical navigation bar and choose 📥 import. From here you can choose "Import via Grafana.com" or "Import via panel json". The easiest is to head over to https://grafana.com/grafana/dashboards/ and find a dashboard you like. We used https://grafana.com/grafana/dashboards/893 when tracking our resource usage. So we enter "893" into the "Import via Grafana.com" field and hit "load" .


Import: Import - Grafana
Docker and system monitoring
Font Awesome v4.7.0 - Icon Sear
New Tab
localhost:3000/dashboard/import
Import
Import dashboard from file or Grafana.com
Upload JSON file
Import via grafana.com
Load
Import via panel json
Load
Documentation | Support | Community | Open Source | v7.5.7 (91de51771c)

Import via grafana.com: 893

Figure 5: Loading dashboard https://grafana.com/grafana/dashboards/893 in Grafana

After hitting "load" you should have this page on your browser:


Import: Import - Grafana
Docker and system monitoring
Font Awesome v4.7.0 - Icon Sear
New Tab
localhost:3000/dashboard/import
Import
Import dashboard from file or Grafana.com
Change uid
Documentation | Support | Community | Open Source | v7.5.7 (91de51771c)

Importing Dashboard from Grafana.com

Published by	Thibaut Mottet
Updated on	2021-02-06 22:30:12

Options

Name: Docker and system monitoring

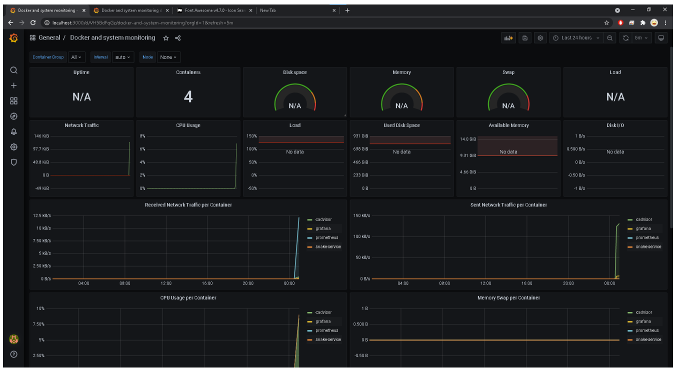
Folder: General

Unique identifier (uid)
The unique identifier (uid) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The uid allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

Prometheus: Select a Prometheus data source
Prometheus (default)

Figure 6: Page for importing dashboard in Grafana.

Select “Prometheus” then click on “import”. The result should look like this:



localhost:3000/d/VH5BdFqGz/docker-and-system-monitoring?orgId=1&refresh=5m

Container Group: All

Interval: auto

Node: None

Figure 7: Monitoring dashboard in Grafana.

Set the appropriate time range and interval, then it should be easy to monitor the resource usage, network traffic of the game-server container. To make it easier to monitor the traffic, we implemented bots to play the game. To start a bot execute the command

```
> py client.py --bot
```

This will skip the page given in Figure 1 and you will see the bot is playing by itself. There is also a script included in the project folder that can concurrently start n amount of bots playing. Execute

```
> ./start_bots.cmd n
```

in Windows PowerShell to start n (n is an integer number) bots to play our snake game. This concludes this section. In the next section we will give a rundown on the implemented code on both the client and server side of this game.

2 Implementation

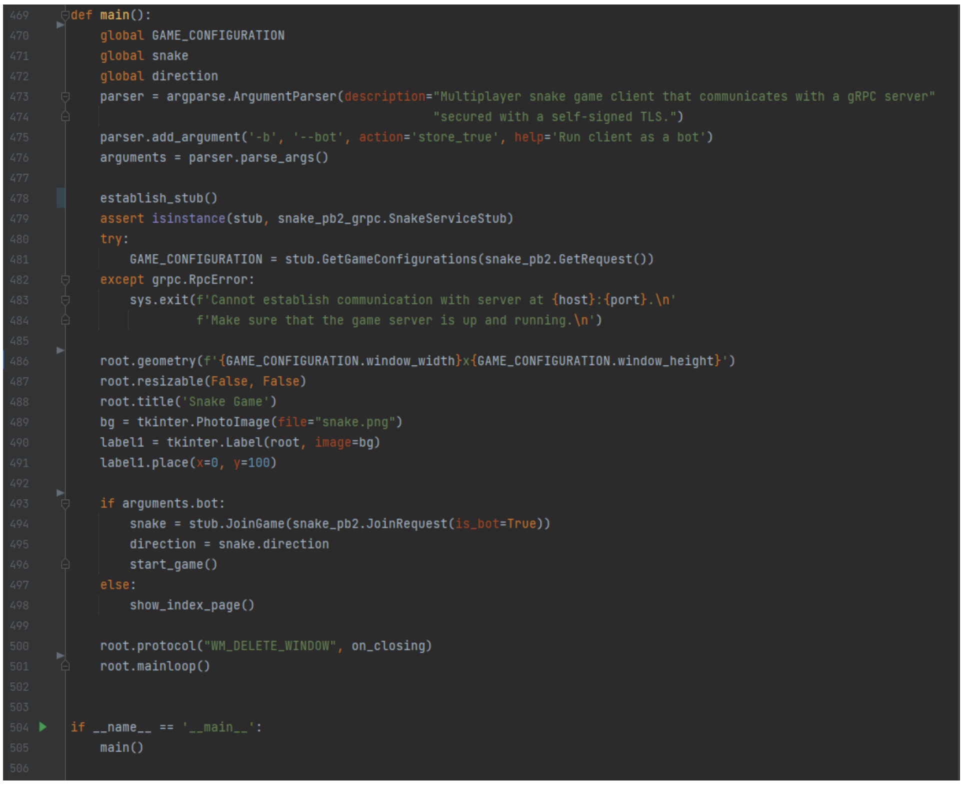
In this section, we will walk through the code we have written to make all this magic possible. Before we dive in let's take a look at the project folder:

```
Data2410-snake
|
+-- service
|   |
|   +-- protobufs
|   |   |
|   |   +-- snake.proto
|   +-- snake-server
|       |
|       +-- bot-names.json
|       +-- crt.pem
|       +-- key.pem
|       +-- requirement.txt
|       +-- server.py
|       +-- snake_pb2.py
|       +-- snake_pb2_grpc.py
|       +-- tkinter-colors.json
+-- client.py
+-- crt.pem
+-- docker-compose.yml
+-- Dockerfile
+-- prometheus.yml
+-- README.md
+-- snake.png
+-- snake_pb2.py
+-- snake_pb2_grpc.py
+-- start_bots.cmd
```

As we can see there are some redundancy of files. For instance snake_pb2.py and snake_pb2_grpc.py is both present in root and ./service/snake-server folders. They are both output when compiling snake.proto. The same goes for crt.pem. The reason for this is that it makes it easier for us to run the program from PyCharm without getting file- and module not found errors. The README.md is there because we wanted to have a nice readme file for our Github.

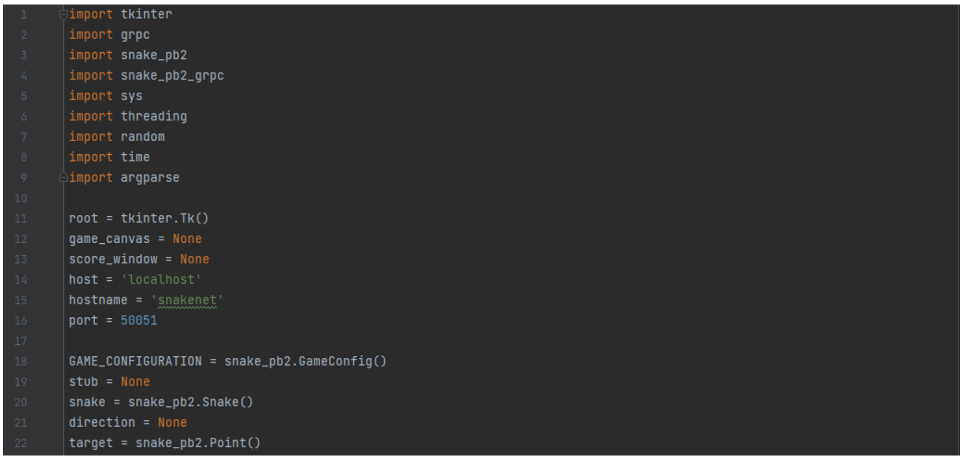
2.1 Entrypoint main() function of client.py.

Assuming that the server is up and running. When starting client.pyit will call on the function `main()` shown below: As seen above, we have defined some global variables which all the later discussed functions is going to access.



```python
def main():
    global GAME_CONFIGURATION
    global snake
    global direction
    parser = argparse.ArgumentParser(description="Multiplayer snake game client that communicates with a gRPC server"
                                                 "secured with a self-signed TLS.")
    parser.add_argument('-b', '--bot', action='store_true', help='Run client as a bot')
    arguments = parser.parse_args()

    establish_stub()
    assert isinstance(stub, snake_pb2_grpc.SnakeServiceStub)
    try:
        GAME_CONFIGURATION = stub.GetGameConfigurations(snake_pb2.GetRequest())
    except grpc.RpcError:
        sys.exit(f'Cannot establish communication with server at {host}:{port}.\n'
                 f'Make sure that the game server is up and running.\n')

    root.geometry(f'{GAME_CONFIGURATION.window_width}x{GAME_CONFIGURATION.window_height}')
    root.resizable(False, False)
    root.title('Snake Game')
    bg = tkinter.PhotoImage(file="snake.png")
    label1 = tkinter.Label(root, image=bg)
    label1.place(x=0, y=100)

    if arguments.bot:
        snake = stub.JoinGame(snake_pb2.JoinRequest(is_bot=True))
        direction = snake.direction
        start_game()
    else:
        show_index_page()

    root.protocol("WM_DELETE_WINDOW", on_closing)
    root.mainloop()


if __name__ == '__main__':
    main()
```

Figure 8: Entrypoint of client.py.

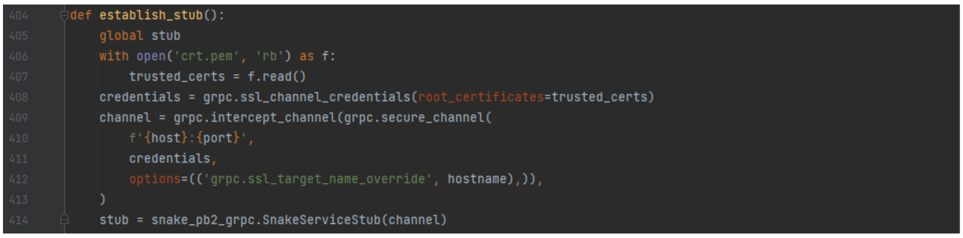


```python
import tkinter
import grpc
import snake_pb2
import snake_pb2_grpc
import sys
import threading
import random
import time
import argparse

root = tkinter.Tk()
game_canvas = None
score_window = None
host = 'localhost'
hostname = 'snakenet'
port = 50051

GAME_CONFIGURATION = snake_pb2.GameConfig()
stub = None
snake = snake_pb2.Snake()
direction = None
target = snake_pb2.Point()
```

Figure 9: Imported modules and global variables in client.py.

As we can see from main(), that it starts with the `argparse` module. This was added in the later stages so that bots can bypass the `show_index_page()` as we simply wanted the bots to start playing the game instead of having to enter a username and click on start playing on the page shown in Figure 1. This made it easier to create the script start_bots.cmd to make n number of bots play the game from PowerShell, which we used for monitoring later.

The first the code will execute is the establish_stub() function



```python
def establish_stub():
    global stub
    with open('crt.pem', 'rb') as f:
        trusted_certs = f.read()
    credentials = grpc.ssl_channel_credentials(root_certificates=trusted_certs)
    channel = grpc.intercept_channel(grpc.secure_channel(
        f'{host}:{port}',
        credentials,
        options=(('grpc.ssl_target_name_override', hostname),)),
    )
    stub = snake_pb2_grpc.SnakeServiceStub(channel)
```

Figure 10: establish_stub() function in client.py.