# Distributed Systems, project 1
# The Casino - Classic high-low game

**Innopolis University, Fall semester**

**Lev Svalov, Dmitry Podpryatov**
B18-DS-02.

September 2020

# Contents

# 1 Introduction

Hello!
As an intro to the report, we wanted to mention the goals and some details on what we are going to do in this project:

- The project itself is a simple high-low game - where you win when your decision about the next card (whether it is higher than the current or it is lower) is right. If it is, the streak continues. We considered this idea as quite entertaining, interesting and not challenging in terms of implementation.
- One of the primary goals during doing the project is to get acquainted with Docker Swarm and extensively understand all processes in it.
- To have fun. Also is important one, since we believe this is one of the key components of successful completion of the project.

The code part of the project can be found in the github repository.
Let's start!

# 2 Understanding Docker-machine and Docker Swarm deployment

## 2.1 What is Docker-machine and what is it used for?

**Docker-machine** is a tool for creating a remote virtual machine and managing the containers. It allows the user to control the docker engine of a VM created remotely.
The main reason you would use docker-machine is when you want to create a deployment environment for your application and manage all the micro-services running on it.

## 2.2 What is Docker Swarm, what is it used for and why is it important in Containers Orchestration?

**Docker Swarm** is a container orchestration tool.
What is container orchestration tool? They allow us to manage, scale, and maintain containerized applications. Docker Swarm consists of managers and workers in the swarm mode and it performs management and load balancing tasks.
Swarm is an important concept in Containers Orchestration because it allows the user to manage multiple containers deployed across multiple host machines.

## 2.3 Install Docker-machine based on the virtualization platform, create a Machine (named Master), and collect some relevant information

- 1. To install docker-machine, we need to complete steps from the following instruction.
- 2. To create the docker-machine with a name Master, we proceed the command:

```
docker-machine create --driver virtualbox Master
```

We used Virtual Box as the driver.

Note: after removing and recreating, the docker-machine can change IP address. That is why you can see different IPs on the screenshots. Docker-machines obtain their IPs in the order they were created. Thus from the IPs 192.168.99.108, 192.168.99.109, and 192.168.99.110 the first would go to Master, the second to Worker-1, and the last one to Worker-2.

- 3. Docker-machine drivers belong to several tools that can be used with docker-machine. The driver determines where the virtual machine is created.
  The list of some supported drivers:
  - Amazon Web Services
  - Microsoft Azure
  - DigitalOcean
  - Google Compute Engine
  - Microsoft Hyper-V
  - VMware Fusion
  - And others...
- 4. With docker-machine provisioning, you are able to supply a cloud hosts.
- 5. Experiment with some docker-machine commands:
  - Start docker-machine:

    ```
    docker-machine start
    ```

  - Stop docker-machine:

    ```
    docker-machine stop
    ```

  - Remove docker-machine:

    ```
    docker-machine rm
    ```

– List docker-machines:

```
docker-machine ls
```

We can apply most of the commands for several docker-machines as well as for the single one.

```
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine create --driver virtualbox Master
Running pre-create checks...
Creating machine...
(Master) Copying /home/dpodpryatov/.docker/machine/cache/boot2docker.iso to /home/dpodpryatov/.docker/machine/machines/Master/boot2docker.iso...
(Master) Creating VirtualBox VM...
(Master) Creating SSH key...
(Master) Starting the VM...
(Master) Check network to re-create if needed...
(Master) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env Master
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine ls
NAME     ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER      ERRORS
Master   -        virtualbox   Running   tcp://192.168.99.107:2376           v19.03.12
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine stop Master
Stopping "Master"...
Machine "Master" was stopped.
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine start Master
Starting "Master"...
(Master) Check network to re-create if needed...
(Master) Waiting for an IP...
Machine "Master" was started.
Waiting for SSH to be available...
Detecting the provisioner...
Started machines may have new IP addresses. You may need to re-run the `docker-machine env` command.
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine rm Master
About to remove Master
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed Master
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine ls
NAME   ACTIVE   DRIVER   STATE   URL   SWARM   DOCKER   ERRORS
dpodpryatov@dp:~/IU/DS/Project 1$
```

.

## 2.4 Create 2 workers

Use the same code as for the Master. Only need to change the name.

```
docker-machine create --driver virtualbox Worker-1
docker-machine create --driver virtualbox Worker-2
```

```
dpodpryatov@dp:~/IU/DS/Project 1$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER      ERRORS
Master     -        virtualbox   Running   tcp://192.168.99.108:2376           v19.03.12
Worker-1   -        virtualbox   Running   tcp://192.168.99.109:2376           v19.03.12
Worker-2   -        virtualbox   Running   tcp://192.168.99.110:2376           v19.03.12
dpodpryatov@dp:~/IU/DS/Project 1$
```

# 3 Container Docker Cluster farm deployment

## 3.1 Docker Swarm setup

- To initialize the swarm we need to ssh to Master node, initialize the swarm and after that, join 2 worker nodes to the swarm.

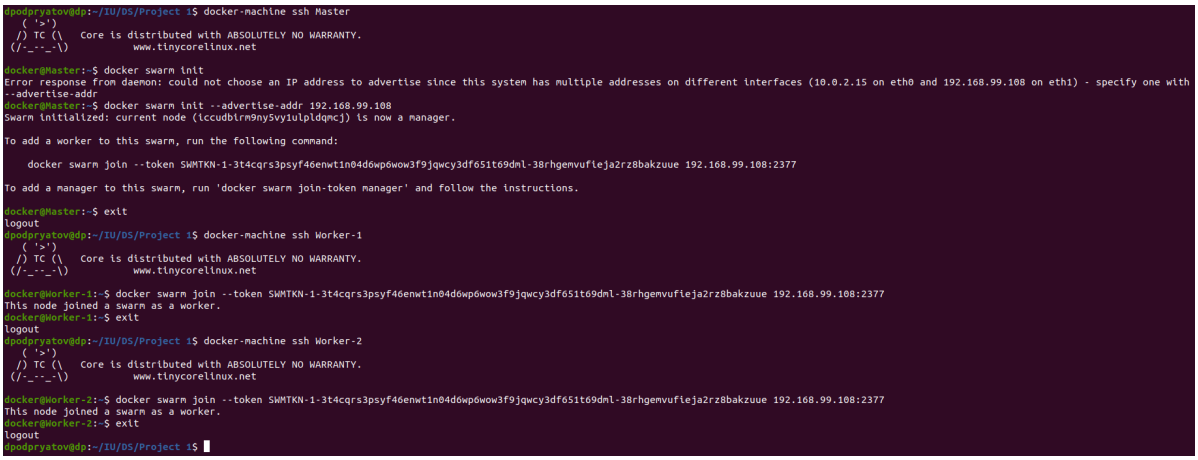
Initialize the swarm:

```
docker-machine ssh Master
docker swarm init --advertise-addr <vm ip>
```

Then *logout* or *exit* from Master and connect to workers (or connect from another shell). Join both workers to the swarm:

```
docker-machine ssh Worker-1
docker swarm join --token <token> <vm ip>:2377"

docker-machine ssh Worker-2
docker swarm join --token <token> <vm ip>:2377"
```

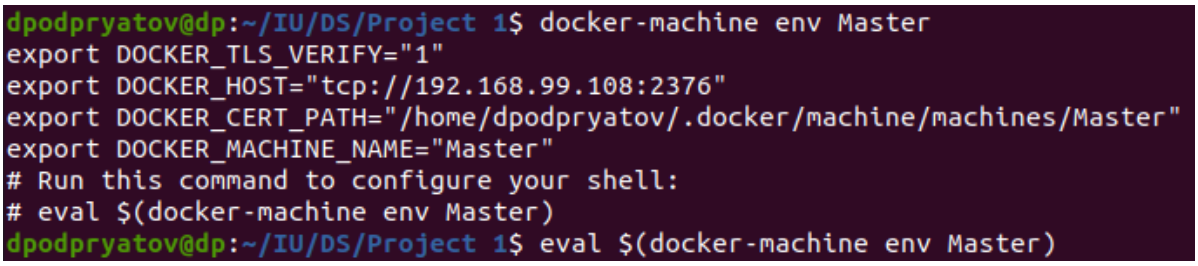



- Set terminal to talk to the master node:

  ```
  docker-machine env Master
  ```

  Which gives another command to execute:

  ```
  eval $(docker-machine env Master)
  ```




- Get information about the swarm (status active):

  ```
  docker info
  ```

```
Swarm: active
 NodeID: iccudbirm9ny5vy1ulpldqmcj
 Is Manager: true
 ClusterID: 3gea796chdnp5fzzy0d2h42yv
 Managers: 1
 Nodes: 3
 Default Address Pool: 10.0.0.0/8
 SubnetSize: 24
 Data Path Port: 4789
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 10
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 192.168.99.108
 Manager Addresses:
  192.168.99.108:2377
```

- Show the nodes in the swarm

  `docker node ls`

  Master node is marked with an asterisk, and also there are two active workers in the swarm.

```
dpodpryatov@dp:~/IU/DS/Project 1$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
iccudbirm9ny5vy1ulpldqmcj *   Master              Ready               Active              Leader              19.03.12
jjl8j1vu4qbsthr0icmo6i25o     Worker-1            Ready               Active                                  19.03.12
g82010bx6f07401kvcy56xju3     Worker-2            Ready               Active                                  19.03.12
dpodpryatov@dp:~/IU/DS/Project 1$
```

## 3.2 How can a Worker be promoted to Master and vice versa?

- To promote Worker to Master, we need to execute command:

  `docker node promote <node name>`

- To demote Master to Worker, we need to execute command:

  `docker node demote <node name>`

```
dpodpryatov@dp:~/IU/DS/Project 1$ docker node promote Worker-1 Worker-2
Node Worker-1 promoted to a manager in the swarm.
Node Worker-2 promoted to a manager in the swarm.
dpodpryatov@dp:~/IU/DS/Project 1$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
iccudbirm9ny5vy1ulpldqmcj *   Master              Ready               Active              Leader              19.03.12
jjl8j1vu4qbsthr0icmo6i25o     Worker-1            Ready               Active              Reachable           19.03.12
g82010bx6f07401kvcy56xju3     Worker-2            Ready               Active              Reachable           19.03.12
dpodpryatov@dp:~/IU/DS/Project 1$ docker node demote Worker-1 Worker-2
Manager Worker-1 demoted in the swarm.
Manager Worker-2 demoted in the swarm.
dpodpryatov@dp:~/IU/DS/Project 1$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
iccudbirm9ny5vy1ulpldqmcj *   Master              Ready               Active              Leader              19.03.12
jjl8j1vu4qbsthr0icmo6i25o     Worker-1            Ready               Active                                  19.03.12
g82010bx6f07401kvcy56xju3     Worker-2            Ready               Active                                  19.03.12
dpodpryatov@dp:~/IU/DS/Project 1$
```

In order to explain the process and reasons of promotion of some node to Manager node

(or in opposite - to demote), let's see what the main role of each node is:
**Manager node** handles cluster management tasks - maintaining cluster state, scheduling services, serving swarm mode. Whereas a **worker node** is only responsible for executing docker containers, it has no any other work to do.

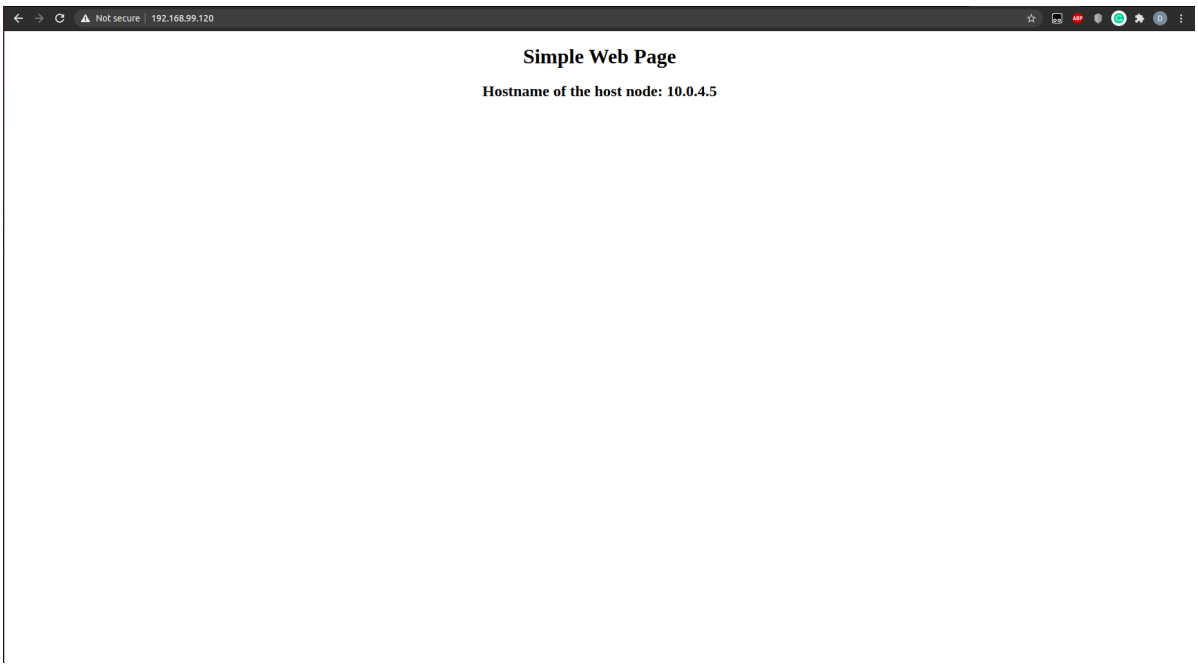
Documentation says that there are three statuses (and empty one) for managing control. Leader has full control, **Reachable** means that a node can be elected as a leader when it goes down, **Unavailable** node cannot participate in the management, and worker nodes have empty status and do not participate in the management.

From here, the swarm should maintain the "quorum of managers", e.g. a majority of managers. It is recommended to have an odd number of managers, and if the swarm looses the quorum of managers, the swarm keeps running but cannot add, update, remove and perform other management tasks.

So, sometimes you may need to change the role of some particular node, for instance - when the manager node is offline for maintenance, another worker node can be promoted to the manager.

## 3.3 Deploy a simple Web page

As it was mentioned, the code is available in the github repository. ***Master* branch represents the code for task 15 (full website) while branch *task07* contains the code for the task 7 (simple website).**
The whitest screenshots of simple web page with node hostname are depicted below. There are three images for each docker-machine (note IP addresses in the url). And, since we specified three replicas in the *docker-compose.yml*, there are 3 hostnames for each node. These hostnames can change after reloading the page, so, it is possible to access all nodes from each docker-machine.


Not secure | 192.168.99.120
Simple Web Page
Hostname of the host node: 10.0.4.5


Not secure | 192.168.99.121
Simple Web Page
Hostname of the host node: 10.0.4.4


## 3.4 How to scale instances in the Docker Swarm?

In the Docker Swarm, the user is able to scale the number of containers in the service using the command (containers running in a service are called tasks):

`docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>`

Nevertheless, there are no an automatic way to scale the services in the Docker Swarm. But, there exists a solution how auto-scaling can be achieved:
The main idea of the solution is to make a docker-machine in order to create machines (with docker) and link these machines to the existing Swarm cluster.
And, the goal is to monitor the cluster for utilisation of something (e.g. CPU, Memory, etc). When it becomes higher than threshold , the trigger for a docker-machine call raises and it forces the docker-machine to scale up the cluster.

# 4 Application Distribution

## 4.1 Validate that when a node goes down a new instance is launched

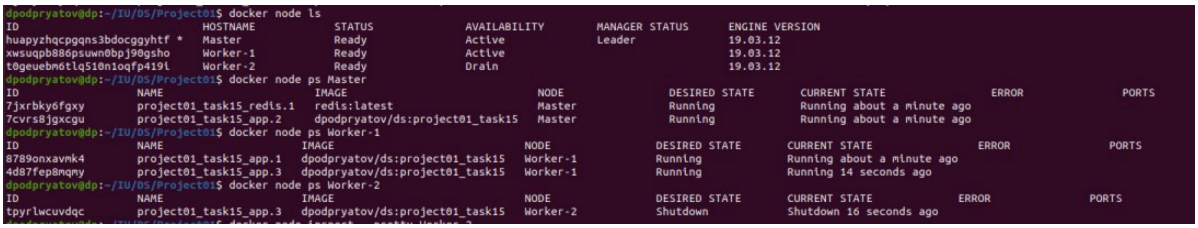
Here you can see the listing of working nodes in the Docker Swarm and services that run on the corresponding node:

```
dpodpryatov@dp:~/IU/DS/Project01$ docker node ps Master
ID                  NAME                    IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
7jxrbky6fgxy        project01_task15_redis.1   redis:latest                         Master    Running         Running 47 seconds ago
7cvrs8jgxcgu        project01_task15_app.2     dpodpryatov/ds:project01_task15      Master    Running         Running 49 seconds ago
dpodpryatov@dp:~/IU/DS/Project01$ docker node ps Worker-1
ID                  NAME                  IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
8789onxavmk4        project01_task15_app.1   dpodpryatov/ds:project01_task15      Worker-1  Running         Running 55 seconds ago
dpodpryatov@dp:~/IU/DS/Project01$ docker node ps Worker-2
ID                  NAME                  IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
tpyrlwcuvdqc        project01_task15_app.3   dpodpryatov/ds:project01_task15      Worker-2  Running         Running 56 seconds ago
dpodpryatov@dp:~/IU/DS/Project01$
```

We have decided to shut down the node **Worker-2** by changing its availability state using the command:

```
docker node update --availability drain <node name>
```

We apply the command and see what is happening:



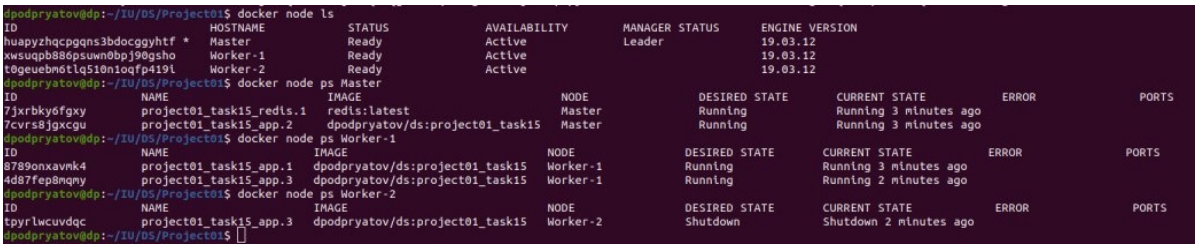

As you can see, the service that was assigned to the node **Worker-2**, after shutting down the **Worker-2**, is reassigned to the active node **Worker-1**.
Thus, the service is continuing its work, just on another node.

### 4.1.1 How the redistribution of the instances can happen when the dead node comes back alive.

We make the node **Worker-2** back alive using the same command, just with another state - "active":

```
docker node update --availability active <node name>
```

After it becomes active, we can see the distribution of services **does not** change:




It is explained by the fact that the node **Worker-1** is able to handle the work of services by its own and there is no need to reassign that **Worker-2**'s service back.
Therefore, the redistribution stays unchanged after making **Worker-1** active.

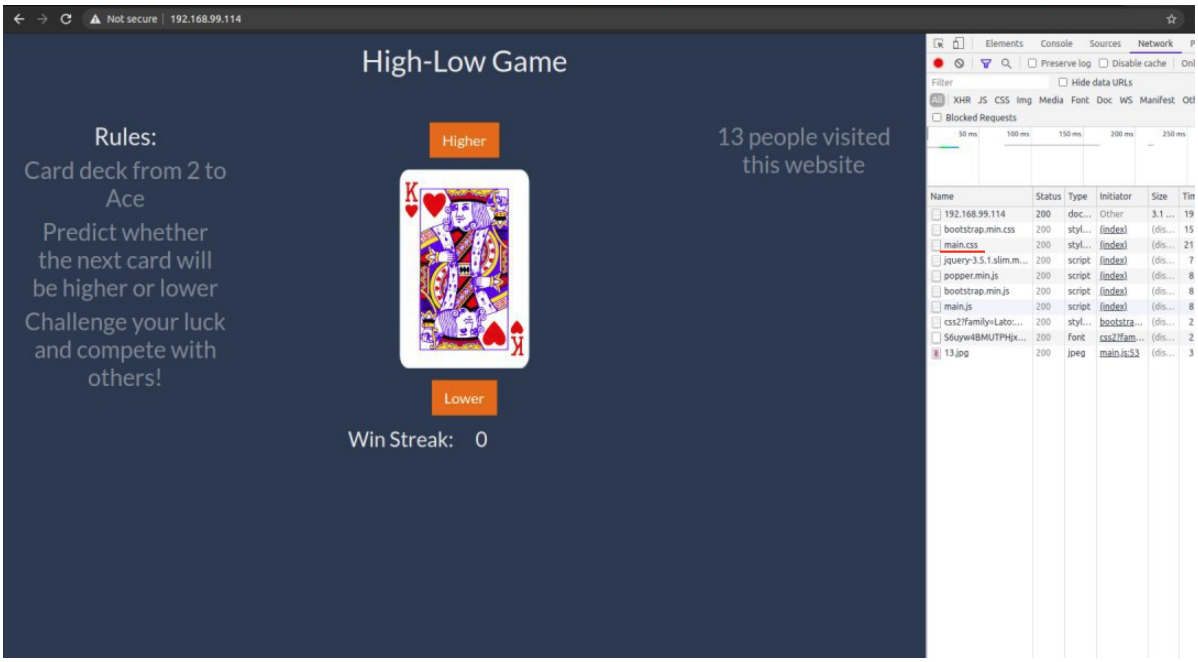
## 4.2 Changes in the servers farm after the application update

To check the changes in the servers farm, we have decided on tiny update in our application that was supposed to ruin the design of the app.

We renamed **.css** file inside of one of the running containers during the working of the services:

```
dpodpryatov@dp:~/IU/DS/Project01$ docker exec -it project01_task15_app.2.7cvrs8jgxcgu3g9159nmor5qq bash
root@1aae955c37d9:/app# ls
Dockerfile  app.py  docker-compose.yml  requirements.txt  restart.sh  start.sh  static  stop.sh  templates
root@1aae955c37d9:/app/static/css# mv main.css mainn.css
root@1aae955c37d9:/app/static/css# ls
bootstrap.min.css  mainn.css
root@1aae955c37d9:/app/static/css#
```

But there were no any impact of the current work of the services and from all nodes the name **.css** stayed unchanged and worked properly:
*(the .css filename is underlined in red)*




Well, we explain it with the following argument:
Basically, in the docker structure of our application, there is one image, and one container that has 3 replicas among all 3 nodes in docker swarm, and these replicas are created during the deployment of the services.
Therefore, the changes were made during the work of the application does not affect the current state of the application. They will appear only after redeployment.

## 4.3 Monitor performance and logs on the servers farm with the Docker Swarm

Docker Swarm uses the command-line interface (CLI).
Swarm has two primary log destinations: the daemon log (events generated by the Docker service), and container logs (events generated by containers). Swarm doesn't maintain separate logs, but appends its own data to existing logs (such as service names and replica numbers).
Swarm shows logs on a per-service basis using:

```
docker service logs <service name>
```

This aggregates and presents log data from all of the containers running in a single service. Swarm differentiates containers by adding an auto-generated container ID and instance ID to each entry.
To centralize the logs, each node in the swarm will need to be configured to forward both daemon and container logs to the destination.

# 5 Playing with Memory

## 5.1 What is Out Of Memory Exception (OOME)?

**Out Of Memory Exception (OOME)** is raised in the situation when the service attempts to use more memory than the swarm node has available.

- When the exception is experienced, a container, or the Docker daemon, might be killed by the kernel killer.
- For avoiding this issue, ensure that your application is configured in such a way that hosts on which the app is run have satisfactory amount of memory.

## 5.2 Deploy a docker container with at least 15% of CPU every second for memory efficiency.

We need to add the following code in the *docker-compose.yml* for the app service. This tells managers to **always** allocate that service (app in our case) at least the specified fraction of the CPU time. In our case, app would have 20% of CPU time reserved to it.

```
resources:
  reservations:
    cpus: '0.2' # Should be >= 0.15. 0.2 >= 0.15, so it's ok
```

# 6 Compression

## 6.1 Verify the size of the Docker images that you're working with. Can this size be reduced and how can we achieve this?

In our application, we are working with only 1 image that is replicated among all 3 nodes (1 Manager and 2 Workers), and here you can see info about the image:

```
dpodpryatov@dp:~/IU/DS/Project01$ docker stack deploy -c docker-compose.yml project01_task07
Creating network project01_task07_default
Creating service project01_task07_app
dpodpryatov@dp:~/IU/DS/Project01$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
dpodpryatov/ds      <none>              6066758c69d1        6 minutes ago       126MB
dpodpryatov@dp:~/IU/DS/Project01$
```

So, summing up, we have 378 mb as final size of Docker images of the application.

## 6.2 Can this size be reduced and how can we achieve this?

We are able to reduce the size of the one common image in the project by proceeding the following procedures:

- **Smaller Image Base**. There is a helpful alternative that is lightweight and has a minimal POSIX environment – **Alpine**. Compared to other OS images, Alpine is much smaller in size. The most downloaded OS image, Ubuntu, is 188 MB, while Alpine is only 5 MB.
- **.dockerignore**. Excluding certain files that aren't necessary for the image helps in reducing the image size.
- **Multi-Stage Builds Feature**. It allows users to divide the Dockerfile into multiple stages. Since the process only transfers the necessary components of the artifact, there is no need to clean up manually after every instruction. Therefore, you avoid adding unnecessary layers, which has a considerable impact on the overall image size.

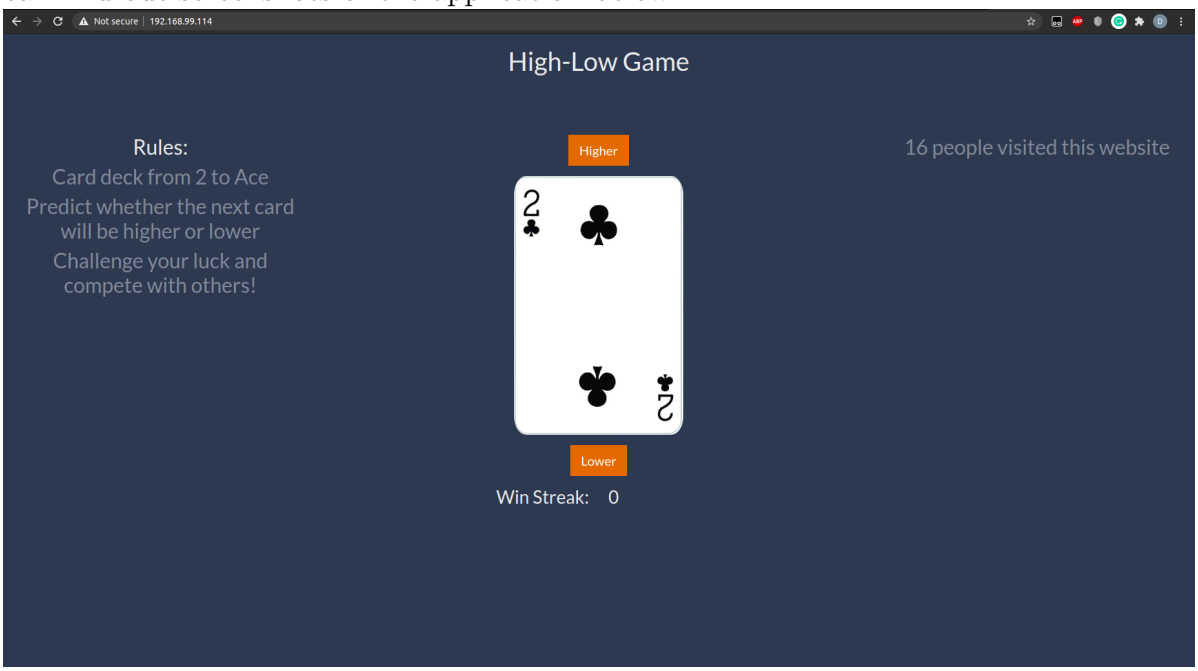
# 7 Web application

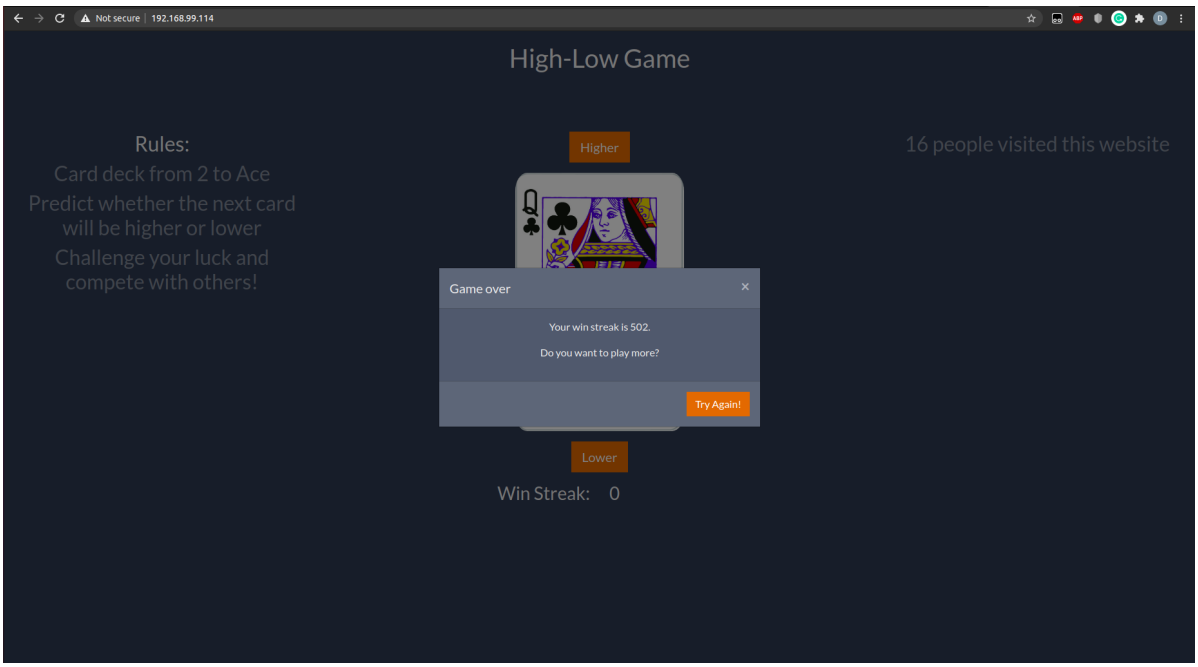
We decided to create a card mini-game known as High-Low or Hi-Lo. The rules are simple: a player makes a bet and is given a card (usually from 52 card deck). On their turn, they can either predict whether the next card from the deck will be higher or lower or take the prize. If a player makes a correct prediction, the prize increases, otherwise they loose their bet.

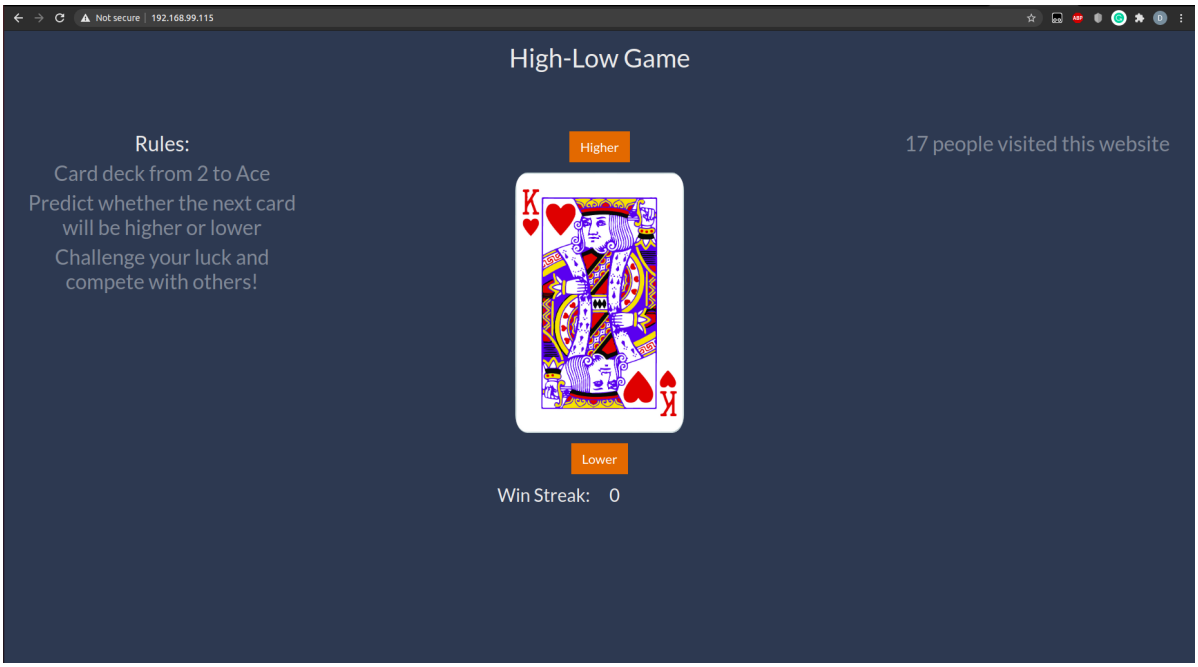
Since gambling is prohibited in the schools and universities, we omitted the part with bets, and so the game consists of only guessing the relative value of the next card. In the end, we got very simple yet fun and random game.

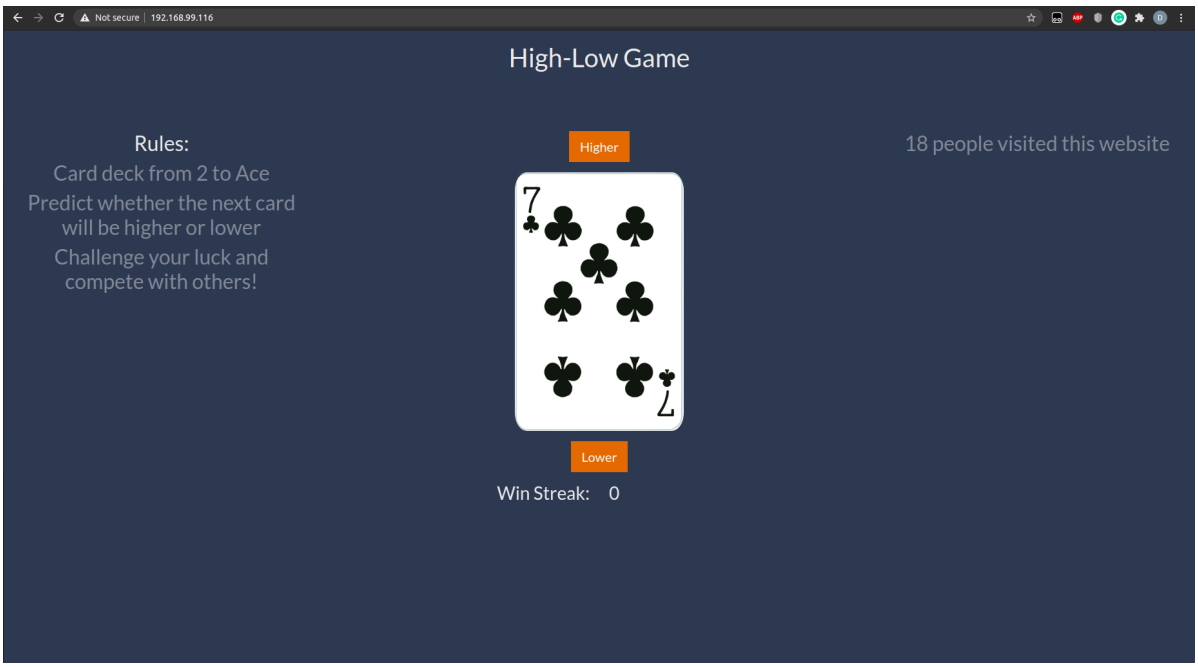
As it was mentioned, the code can be found in the github repository. We created Flask-based website with Redis for counting visitors. Bootstrap was used for css. Most of the coding were done via javascript, and, due to the lack of experience with Flask there is basically no server side of the game, and all calculations are done on the user side. Thus it is very easy to cheat (for example, by manually increasing the win streak or card value variables in browser console). Nevertheless, security was not the main goal of

this project. Because of the lack of experience there is no leader board, although initially it was planned. We estimated our time and decided not to over complicate things. You can find out screenshots of the application below.





Not secure | 192.168.99.114
High-Low Game
Rules:
Card deck from 2 to Ace
Predict whether the next card will be higher or lower
Challenge your luck and compete with others!
Higher
Game over
Your win streak is 502.
Do you want to play more?
Try Again!
Lower
Win Streak: 0
16 people visited this website


Not secure | 192.168.99.115
High-Low Game
Rules:
Card deck from 2 to Ace
Predict whether the next card will be higher or lower
Challenge your luck and compete with others!
Higher
Lower
Win Streak: 0
17 people visited this website




# 8 Conclusion

To sum everything up, we have learnt about the Container Orchestration and Docker Swarm in-depth, applied this knowledge, and tried to answer on questions clearly and in all details. We also have implemented the high-low game as a part of the project. Finally, we've enjoyed doing the project and, of course, playing the game.
Thank you!