PRACTICA 3

Przemysław Łent (ple991@gmail.com,usu1)
Ceyda Oktem(ceydaoktem93@gmail.com,usu2)
Irem Uzunbaz(iremuzunbaz@gmail.com,usu3)
Rifat Cakir(rfatcakr@gmail.com,usu4)

Contents

1. Part 1

This part contains preparing images of components created in lab2

First big problem was to install all required components on Ubuntu system, as it uses command *node.js* instead of *node*. It was solved by installing few additional packets.

After understanding the idea of using docker we had to change original source code from lab2, to use config files instead of running arguments. For example:

```
var configu = require('config');
backend_port = configu.provides.backendPort;//process.argv[ 3 ];
verbose = configu.parameter.verbose;//( process.argv[ 5 ] == "true" );
```

And changing the configuration files, default.js to make it work:

```
module.exports = {
    provides : {
        frontendPort : "8000",
        backendPort : "8001",
        configPort : "8100"
    },
    parameter: {
        verbose: true
    }
};
```

The change was also needed in assigning the ID to components. Instead of random from previous part, now they use:

```
requester.identity = (require('config').instanceId +'');// random.get();
```

Other changes were simpler, according to renaming the components and configuration in order to be consistent with assigned convention. After that components were successfully build.

2. Part 2

This part is about building the service and its deployment

Service definition needed correct location of each components and their image name in index.js, which was possible to do in easier way with using helper method *require.resolve* e.g.:

```
broker: {
                location: require.resolve("../../Components/lab2broker"),
                image: "tsir/lab2broker"
        },
```

And correct definitions of each used link e.g. :

```
lab2worker: {
        endpoint: ["lab2broker", "backendPort"]
    }
```

After successful creation of definition of service, the deployment definition is needed. According to requirements it was necessary to give correct name of previously created service and number of each component instances in index.js file:

```
module.exports = {
servicePath: require.resolve("../../Services/lab2Service"),
counts: {
    broker: 1,
    client: 3,
    worker: 6
}
};
```

The last part is about to finally make a deployment. With all previously created files it was possible to do it with provided deployers. But the task required to get clients and workers count as parameters. The cleverest solution for this problem is that we came up with was to edit provided deployer.js. The service path instead of being required as argument wasn't really needed as the counts of components were accessed from arguments, and given to deployment definition object were created straight in the code instead of loading previously created one from file.

```
deployer.deploy(
module.exports = {
servicePath: require.resolve("./Services/lab2Service"),
counts: {
        broker: 1,
        client: clntNum,
        worker: wrkrNum
}
}
);
```

Now it is possible to deploy by using command e.g.:

sudo nodejs deployerlab2.js 2 3

To see everything we need to use:

sudo docker ps –a

And we can stop it, to for example deploy again (while it wasn't possible to do without it) with some changes (if we want) by:

sudo docker rm –f $(sudo docker ps -a -q)

3. Part 3

4. Global comments

Creating deployment is very time consuming process and requires knowledge of each component. In our laboratories it was made simpler by using deploying programs and sample base configurations. Our experience shows that although we're using the virtualization by Docker (and already prepared image with its configuration), if we want to test code of component on our machine before real creation of "docker-component" image we need to fight with lots of Linux commands and packets, which might take as much time as real defining an creating the deployment.

What is worth mentioning, usually we need to modify code of each component (previously meant to run straight on local machine) before creating, previously brought up, final docker-components.

Docker is very powerful tool, but it's not meant to use by simple users, the configuration of every part requires lots of knowledge. The biggest help during actual creation of the solution were examples that provided in lab3 allow us to see, modify and test working example to better understand the mechanisms.

5. Bibliography

- TSR lectures and laboratory script
- http://stackoverflow.com/
- http://coderwall.com/
- https://docs.docker.com/ - Docker documentation