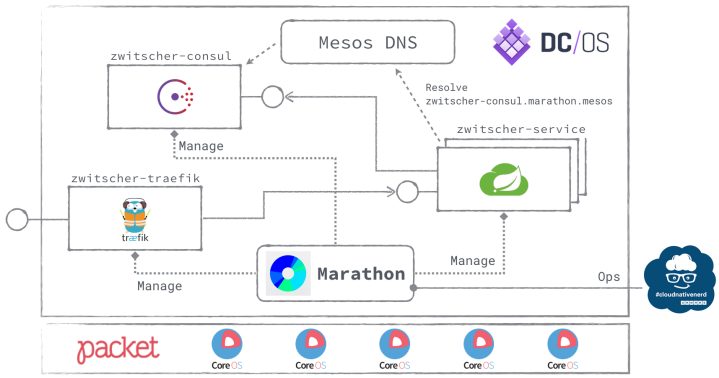
Exercise 4: Orchestration



Step 01: Writing a Marathon specification

To orchestrate our showcase we need to write the appropriate Marathon JSON specifications. Implement the JSON specification for your Zwitscher service and include the following:

- Make sure to give your application a unique ID, such as `zwitscher-service-<username>`
- Start 1 instance, with 0.5 CPUs and 256 of memory
- Specify your Docker container using BRIDGE networking mode with required port mappings
- Define health checks for the `/admin/health` endpoint
- Define `zwitscher-consul` as a dependency for your application
- Declare the `CONSUL_HOST` env variable and set the value to the Mesos DNS name of the zwitscher-consul application

The resulting JSON file might look something like the following:

```
{
  "id": "zwitscher-service-hitchhikersguide",

  "instances": 1,
  "cpus": 0.5,
  "mem": 256,

  "cmd": "java -Xmx128m -Dserver.port=$PORT0 -Dspring.cloud.consul.discovery.ip-address=$HOST -jar /app/

  "container": {
    "type": "DOCKER",
```

```
    "docker": {
      "image": "hitchhikersguide/zwitscher-service:1.0.1",
      "network": "BRIDGE",
      "portMappings": [
        {
          "containerPort": 0,
          "hostPort": 0,
          "protocol": "tcp"
        }
      ]
    }
  },

  "healthChecks": [
    {
      "protocol": "HTTP",
      "path": "/admin/health",
      "intervalSeconds": 10,
      "portIndex": 0,
      "timeoutSeconds": 10,
      "maxConsecutiveFailures": 3
    }
  ],

  "dependencies": [
    "zwitscher-consul"
  ],

  "env": {
    "CONSUL_HOST": "zwitscher-consul.marathon.mesos"
  },

  "labels": {
    "LAUNCHPAD_ENABLE": "true",
    "LAUNCHPAD_COLOR": "LIGHT_GREEN",
    "LAUNCHPAD_ROW": "1"
  }
}
```

The tricky parts are the port mapping and passing the details used during the service registration. To be able to scale the application easily the ports need to be assigned by DC/OS. The automatically assigned port is available via the `$PORT0` environment variable, which can then be used to set the `server.port` system property on startup. We also need to manually assign the external IP of the DC/OS node the service is running on, otherwise it would use the Docker internal IP instead which won't work.

The Marathon specifications for Consul and Traefik are already implemented. Make sure you study the details of those two specs as well.

Step 02: Deploy to DC/OS cluster

Deployment time!!! In this step we will learn how to manage the DC/OS cluster and marathon as well as deploy applications using the CLI and optionally Gradle.

Make sure the `dcos` CLI tool is available on the PATH. If not, browse the DC/OS demo cluster URL and follow the installation instructions. Once the CLI is installed, you first need to login to the demo cluster using the following command:

```
$ dcos auth login
```

This will display a URL in the shell. Open this URL in a browser window, authenticate and then copy the displayed access token into the shell. You should see a successful login.

Now it is time to explore the DC/OS cluster. You an always use the `--help` flag to get a list of the available commands and

options, e.g.

```
$ dcos
$ dcos node
$ dcos marathon
$ dcos marathon app list
```

Next we will deploy the Marathon specification for our Zwitscher service we created in the previous step. The Consul and Traefik specs should have already been deployed.

```
$ dcos marathon app add marathon-zwitscher-service.json
$ dcos marathon app show zwitscher-service
```

Open the DC/OS Web UI and follow the deployment of the service. Browser the service and have a look at the details and the log outputs. Once the service is marked as healthy, have a look at the Consol UI. You should now see a new instance of the service appear. Also have a look at the Traefik UI, there should be a new instance available as well.

If everything is OK it is time to scale the service. Either use the **Scale** button in the Web UI, or issue an update command for our Zwitscher service and increase the instance count to the desired value. Please don't go mad here, it's a shared cluster!

```
$ dcos marathon app update zwitscher-service instances=2
```

Finally, we will be using the Gradle Cloud Deployer plugin to control the deployment of the Zwitscher service using our build tool. Add the plugin to your `build.gradle` file and add the required configuration for Marathon.

```
deployer {
    marathon {
        id = "marathon-zwitscher-service"
        baseUrl = "http://147.75.100.195"
        strategy = "REPLACE"
        auth {
            token = dcosAuthToken()
        }
        files = [file("../marathon-zwitscher-service.json")]
    }
}
```

References

- http://mesosphere.github.io/marathon/docs/
- https://mesosphere.github.io/marathon/docs/application-basics.html
- https://mesosphere.github.io/marathon/docs/native-docker.html
- https://mesosphere.github.io/marathon/docs/ports.html
- https://mesosphere.github.io/marathon/docs/health-checks.html
- https://github.com/qaware/gradle-cloud-deployer