API integration

Cloud providers offer a wide range of tools to interact with their API's. Most importantly, every cloud provider offers a command line tool. With larger cloud providers this is often the better choice compared to their web interface. The cloud provider for the project work, Exoscale, offers a CLI for all major platforms.

> **Tip**
>
> The source code for this exercise is available on GitHub.

Using the CLI

When basic scripting is required the CLI may entirely be enough for the task. For example, you can query the list of instances in an instace pool as follows:

```
exo instancepool show INSTANCE-POOL-NAME --output-format json
```

You can use this command to generate your service discovery JSON file. Similarly, you could change the instance pool size:

```
exo instancepool update test --size 5
```

Using Go

When it comes to creating a web server the CLI solution usually goes out the window because it is quite hard to create a webserver that calls a shell script. (It involves creating a CGI script, which is an easy way to also create a security hole.) Go being a popular language for the cloud world many cloud providers offer an SDK for this language. It also has a built-in library for creating web servers which makes it ideal to receive webhooks and react on them.

As a first step for this exercise let's create a file called `go.mod` to enable Go module support. (If you are unfamiliar with Go you really, really don't want to write Go code without this.)

```
module github.com/yourname/go-example

go 1.14

require github.com/exoscale/egoscale v0.27.0
```

As you can see, we already pulled in the `egoscale` library. This is the SDK for Exoscale in Go.

Let's create a simple `main.go` that increases the size of an instance pool:

```
package main

import (
    "context"
    "flag"
```

```
    "log"

    "github.com/exoscale/egoscale"
)

func main() {
    zoneId, err := egoscale.ParseUUID("zone-id-here")
    if err != nil {
        log.Fatalf("invalid zone ID (%v)", err)
    }
    poolId, err := egoscale.ParseUUID("instance-pool-id-here")
    if err != nil {
        log.Fatalf("invalid pool ID (%v)", err)
    }

    // Create a new client
    client := egoscale.NewClient("https://api.exoscale.ch/v1/", "api-key-here", "secret-here")
    ctx := context.Background()

    //Request the current size of the instance pool
    resp, err := client.RequestWithContext(ctx, egoscale.GetInstancePool{
        ZoneID: zoneId,
        ID:     poolId,
    })
    response := resp.(egoscale.GetInstancePoolResponse)
    if len(response.InstancePools) == 0 {
        log.Fatalf("instance pool not found")
    } else if len(response.InstancePools) > 1 {
        //This should never happen
        log.Fatalf("more than one instance pool returned")
    }
    instancePool := response.InstancePools[0]

    //Resize the instance pool
    _, err = client.RequestWithContext(ctx, egoscale.ScaleInstancePool{
        ZoneID: zoneId,
        ID:     poolId,
        Size:   instancePool.Size + 1,
    })
    if err != nil {
        log.Fatalf("Failed to increase instance pool size (%v)", err)
    }
}
```

This little program will increase the instance pool size by 1. You can run it with `go run main.go`.

Python

A second popular language for cloud programming is Python. Exoscale also has an SDK for Python.

> **Note**
>
> The Python library for Exoscale currently does not work on Windows.

First, let's create a `requirements.txt`:

```
exoscale
```

Then we can run `pip install -r requirements.txt` and then implement this simple program:

```
import exoscale

if __name__ == "__main__":
    exo = exoscale.Exoscale(api_key="api-key-here", api_secret="api-secret-here",
config_file="")

    zone = exo.compute.get_zone("zone-name-here")

    ip = exo.compute.get_instance_pool("instance-pool-id-here", zone=zone)
    ip.scale(ip.size + 1)
```

That's it!