Deadlines

Each sprint has a hard deadline. Sprints can be handed in earlier. If a deadline is missed the points are not awarded and the solution is uploaded to GitHub so the next sprint can be accomplished.

Furthermore, if you run out of budget on your Exoscale account **a -15% penalty** will be incurred. Make sure you monitor your Exoscale account. (Please contact us if you require more budget.)

Sprint 1: Instance Pools

Deadline: 15th of October 2020 23:59

In this sprint you must demonstrate your ability to set up an instance pool and a network load balancer.

Pass criteria

MANUAL SOLUTION (5%)

You must demonstrate on a call how you set up your Exoscale environment in detail. Otherwise, the requirements are identical as with the automated solution.

AUTOMATED SOLUTION (15%)

You must upload your Terraform to GitHub or an alternative Git service and submit your link in Moodle. Your code must accept two input variables, `exoscale_key` and `exoscale_secret`. The automated checking system will run your code against a completely empty Exoscale account from a standard Ubuntu 20.04 environment. The automated system will check the following:

- The Terraform code must execute successfully,
- The presence of a single NLB with a single service listening on port 80,
- The presence of a single instance pool,
- That all backends on the NLB are green within 5 minutes of running Terraform,
- When deleting all instances from the instance pool, the instance pool must provision the new instances and the health check must become green within 5 minutes,
- When accessing the IP address of the NLB with the `/health` URL, that URL should respond with "OK" and a HTTP status code 200. (The easiest way to achieve this is to run the http load generator.)

Please make sure your your code is in the **main/master branch** of your Git repository.

Sprint 2: Monitoring

Deadline: 15th of November 2020 23:59

In this sprint you must demonstrate your ability to monitor a varying number of instances set up on an instance pool in the previous sprint using Prometheus.

Pass criteria

Note: Grafana has been moved to Sprint 3.

MINIMUM REQUIREMENTS (5%)

For manual method only: You must demonstrate on a call that everything from Sprint 1 still works, as well as the following points.

For both manual and automated systems:

- You must create one extra instance with Prometheus running on it on port 9090, accessible from the internet. (Yes, this is not production-grade, but we need it for testing.)
- When querying Prometheus for the `up` metric, the metric must display the instances in the instance pool.
- When querying Prometheus with the following PromQL query, it must result in one line on the graph for each instance: `sum by (instance) (rate(node_cpu_seconds_total{mode!="idle"}[1m])) / sum by (instance) (rate(node_cpu_seconds_total[1m]))`
- When querying the `/load` endpoint of the NLB IP address on port 80 from Sprint 1 repeatedly, the CPU usage increase must be visible in Prometheus using the above query.
- After destroying all instances in the instance pool, the above criteria must be true within 5 minutes again.

AUTOMATED SOLUTION (+5%)

In addition to the criteria above, the entire setup must be fully automated using Terraform, and the Terraform code must be uploaded to a Git repository accessible by the lecturers. The following criteria must also be true:

- Your code *must* be on the `master` branch and in the main folder.
- `terraform init` must succeed.
- `terraform apply` must succeed with the `exoscale_key` and `exoscale_secret` variables passed to it.
- `terraform destroy` must succeed and must leave no resources behind.
- After applying the configuration, all resources from Sprint 1 must be present, plus one single monitoring instance.
- Prometheus must be accessesible using port 9090 on the public IP of the monitoring instance within 5 minutes.

SERVICE DISCOVERY IMPLEMENTATION (+5%)

To achieve this goal, you must implement the service discovery agent yourself. (Do not copy code from the Internet.) You can pick the programming language of your choice.

- Upload your code to the repository with the Terraform code in the `servicediscovery` folder.
- Your code must be containerized with a `Dockerfile`. The `docker build` command must execute successfully.
- Your container built from the Dockerfile must accept the `EXOSCALE_KEY`, `EXOSCALE_SECRET`, `EXOSCALE_ZONE` (e.g. `at-vie-1`), `EXOSCALE_ZONE_ID`, `EXOSCALE_INSTANCEPOOL_ID`, and `TARGET_PORT` environment variables as configuration.
- Your application must write a Prometheus-compatible service discovery file to the path `/srv/service-discovery/config.json` inside the container (will be mounted as a volume). (No labels need to be added, only the IP and port.)
- The service discovery agent must detect changes in the instance pool within 2 minutes and write the appropriate service discovery file.

How the service discovery is tested: the service discovery agent is tested separately outside of the cluster provisioned by Terraform. The hand in automation builds a container image from the `Dockerfile` located in the `servicediscovery` folder and runs it with the correct environment variables, then observes if the service discovery file (`/srv/service-discovery/config.json`) inside the container contains the correct details as instances are stopped and started.

We recommend creating an account on the Docker hub to `docker push` your image. You can then pull the image from your `userdata` instead of uploading the code directly.

Sprint 3: Autoscaling

Deadline: 15th of January 2021 23:59 ~~15th of December 2020 23:59~~

In this sprint you must demonstrate your ability to receive a webhook from Grafana and adjust the number of instances in the instance pool from the previous sprint under load.

Your service will be hit with a large number of requests and it must scale up as outlined in the project work document.

Pass criteria

MINIMUM REQUIREMENTS (5%)

Note: If you only fulfill your minimum requirements please contact us for a review appointment.

- All items from Sprint 1 and 2 must still work.
- You must install Grafana on your monitoring server such that it is publicly available on port 3000 of your monitoring server.
- You must add your Prometheus server as a datasource to Grafana.
- You must create a dashboard in Grafana with two graphs that both show the average CPU usage across all the instances in the instance pool from Sprint 1. The graph must not include the CPU usage of the monitoring server.
- One of the graphs must have an alert set to trigger when the average CPU usage is over 80% over 1 minute after 1 minute of "pending".
- The second graph must have an alert set to trigger when the average CPU usage is below 20% over 1 minute after 1 minute of "pending".
- The 80% alert must trigger send a webhook to the autoscaler of the monitoring host on the `/up` URL. If the CPU usage is still high after 5 minutes the webhook must be repeated. It MUST NOT send a webhook for resolution (when the CPU usage drops below 80%).
- The 20% alert must trigger a webhook to the `/down` URL of the autoscaler in the same manner as described above.
- When the NLB IP is hit on the `/load` endpoint by a client and the average CPU usage climbs over 80% it must start new instances periodically until the CPU usage drops below 80%.

AUTOMATED SOLUTION (+5%)

In addition to the criteria above, the entire setup must be fully automated using Terraform, and the Terraform code must be uploaded to a Git repository accessible by the lecturers. The following criteria must also be true:

- Your code *must* be on the `master` branch and in the main folder.
- `terraform init` must succeed.
- `terraform apply` must succeed with the `exoscale_key` and `exoscale_secret` variables passed to it.

> - `terraform destroy` must succeed and must leave no resources behind.
>
> #### AUTOSCALER IMPLEMENTATION (+5%)
>
> To achieve this goal, you must implement the autoscaler yourself. (Do not copy code from the Internet or from fellow students.) You can pick the programming language of your choice.
>
> - Upload your code to the repository with the Terraform code in the `autoscaler` folder.
> - Your code must be containerized with a `Dockerfile`. The `docker build` command must execute successfully.
> - Your container built from the Dockerfile must accept the `EXOSCALE_KEY`, `EXOSCALE_SECRET`, `EXOSCALE_ZONE` (e.g. `at-vie-1`), `EXOSCALE_ZONE_ID`, `EXOSCALE_INSTANCEPOOL_ID`, and `LISTEN_PORT` environment variables as configuration.
> - When an HTTP request is sent to the `LISTEN_PORT` on the `/up` endpoint your autoscaler must increase the size of the instance pool given in `EXOSCALE_INSTANCEPOOL_ID` by 1 and respond with a HTTP 200 status code.
> - When an HTTP request is sent to the `LISTEN_PORT` on the `/down` endpoint your autoscaler must decrease the size of the instance pool given in `EXOSCALE_INSTANCEPOOL_ID` by 1 and respond with a HTTP 200 status code.
> - If the instance pool size is 1 and scaling down is requested the autoscaler must ignore the request and respond with an HTTP 200 status code.
> - Your autoscaler must stop gracefully with an exit code of 0 when sent a TERM signal in less than 30 seconds.