Site Reliability Engineer – Case Study

Welcome

As a Site Reliability Engineer in our Infrastructure Operations team, you will work on our cloud-based infrastructure serving millions of requests every day.

This does not mean we are looking for someone whose aim is to avoid downtime at all costs only, but also to bring up their ideas on how to shape our architecture in the future and who is motivated to support our application engineers implementing new features and services.

In trivago, we have a culture of better trying and failing than never trying something new. This means we're also doing a lot of testing. In the sense of determining the best solution for our product, Split or A/B testing is an essential piece of this.

This leads to the challenge below:

Strength (tags)

containers, orchestration, traffic management, architecture, infrastructure, monitoring

THE CHALLENGE

You will find two applications: A Golang-based and a Java-based application. Both need to be containerized according to industry best practices. The Golang application needs to be compiled from source, while the Java application is delivered as a pre-built Jar file, runnable using Java 11.

Both are providing an HTTP service, binding to all interfaces on port 8080, with the same endpoints:

Route	Description
/	A static site. Should *not* appear in the final setup
/hotels	JSON object containing hotel search results
/health	Exposes the health status of the application
/ready	Readiness probe
/metrics	Exposes Prometheus metrics of the application

Your challenge will be to provide a load balancer setup like the following:

```
                            +------------------>Java app
                            |
                            |
                            |
                      30% of traffic
                            |
                            |
User +-------> load balancer+
                            |
                            |
                      70% of traffic
                            |
                            |
                            |
                            +------------------>Go app
```

The traffic distribution should be as follows: 70% of the requests are going to the application written in Golang, 30% of the requests are going to the application written in Java. Also, each HTTP response needs to carry a custom header, called `x-trv-heritage` which indicates the application that responded.

Your implementation must be runnable on a machine using x86_64 CPU architecture and must be built on top of Kubernetes.

One should be able to at least see that the traffic distribution works as expected in some form. As a bonus, you can try to show other metrics, like CPU usage, memory utilization, and latency as well to compare the two services.

Your implementation should:

- Build both container images locally
- Find a solution to make them available to a Kubernetes cluster
- Do not push them to a public registry on the internet!
- Setup an ingress solution of your choice
- Deploy both workloads
- Wait for the readiness of the system
- Run 100 requests against / of the applications under test

Do not share the case study and/or the results with third parties. Do not upload the case study and/or results to social platforms or other public spaces.

Please document your architecture decisions and any necessary steps to get your implementation running locally.

Here are a few tips and hints:

- k3d is a neat tool to run Kubernetes locally; it comes with a local container registry and can also import container images
- pick any ingress controller you like; we're heavily invested in the Istio service mesh and its ingress gateway
- there are great open-source monitoring solutions; we're actively using Prometheus and Grafana
- our Kubernetes manifests are usually packaged as Helm charts
- We are going to replicate and run your setup on Mac OS and Linux machines with x86_64 CPU architecture, Bash 4+ and Docker installed - keep that in mind when using `date`, `sed` & co
- No project ships without a README
- No project ships without a justfile
- "If it needs to be reproducible - automate it, if it can't be automated - document it"

This challenge is designed to give you a glimpse of the activities you would be tackling in your potential future team. The tasks in this case study are inspired by and very similar to those you need to deal with every day. This means that if you have fun solving this case study you are most likely on the right track!

As you will notice, not every single detail necessary to come up with a decision is included in the tasks. We are mostly interested in understanding your process of thought, so make your own assumptions when you feel it is necessary and do not forget to write them down in your answer. We're looking forward to getting your response and hope you can derive some learnings and joy from doing this assignment.

SUBMISSION DEADLINE: You should not need more than 3-4 hours to solve this case study. Please make sure that you submit your results within 7 business days after receiving. We would like to ask you to also grand us the same time to evaluate your results after submission.

DELIVERY FORMAT: Please submit your results via the link provided in the email from your recruiter. There you can upload your files in a zip archive

Please feel free to reach out to your recruiter if you have any questions!