Cost and Load Balancing

**PREPARED FOR**

Erik van der Schriek

Niko Kuijpers

Gertjan Schouten

**PREPARED BY**

Vincent Andersen

# 

# Versiebeheer

# 

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Changes** |
| 19/5/2021 | Vincent Andersen | Document formatting |
| 20/5/2021 | Vincent Andersen | Initial start |
| 25/5/2021 | Vincent Andersen | Added conclusion |
| 26/5/2021 | Vincent Andersen | Added calculation and conclusion |
|  |  |  |

# 

# Table of Contents

[**Versiebeheer**](#_lh5w11na753n) **1**

[**Table of Contents**](#_lqmmw99b7pfu) **2**

[**Introduction**](#_k6jx82pctwds) **3**

[**Research Question 1**](#_zc6h8lveooxw) **4**

[Sub-questions](#_505gm0evad4t) 4

[Methods](#_o0btrvpmesm3) 4

[1.1 How is the cost of a microservice application hosted on Azure determined?](#_8g59hr54302) 5

[1.1.1 How are hosting costs measured?](#_fjvfvfp9p4ht) 5

[1.3 Conclusion](#_4rj0uh4s2sh9) 6

[**Research Question 2**](#_vro83kh52c07) **7**

[Sub-questions](#_x9f5pacnhxrb) 7

[Methods](#_2qh5ytgj9ndf) 7

[2.1 Which services in our architecture are expected to have the most load and how can we test it?](#_x58yxsox3js3) 8

[2.1.1 Our architecture choices](#_l9i04oltsyil) 8

[2.1.2 What tools exist to test load?](#_mk2v3p3ifug0) 10

[**2.2 Which parts of our architecture can be improved or changed to ensure reliability in scaling?**](#_fhu9ujcgz3a) **11**

[2.3 Conclusion](#_rtg09jwlzne) 12

[**Sources**](#_qwibiw7iqp2l) **13**

# Introduction

This document aims to give proper insight into the usage of cloud hosting, load balancing and the cost of using such techniques. In this research document you will find that we try to rectify our architecture choices and provide examples of other existing solutions that we could’ve chosen. In order to gain more understanding of the costs and load structure of our application, this research document was made.

Currently we have a microservice based architecture running in docker containers that will be deployed on Azure as our cloud service of choice. We use Java in combination with spring boot as our primary framework, as well as RabbitMQ as our message broker. In order to receive relevant energy data we make use of mostly dutch made REST API’s that we call from our own restful API’s in the backend. These API’s publish their data onto the message exchange periodically while other related services listen to these messages. In terms of our frontend we have chosen React. We currently do not use any serverless techniques in our project.

The main goal of our given project is to make a fully functional microservice based application that is running in the cloud, which we can change and monitor more easily as opposed to a monolithic application as developers. In order to gain more understanding of how each service interacts within the context of a very large user base, 100 million as our standard, we have provided the following main research questions that are accompanied by sub-questions that are viewable per question.

1. What are the costs of running a microservice application in Azure and which costs apply to our project?
2. Which services can expect the most load given a large number of users and how can we improve or ensure reliability on scaling on these services?

# Research Question 1

How is the cost of running a microservice application in Azure calculated and which variables do we need to take note of?

## Sub-questions

* How is the cost of a microservice application hosted on Azure determined?
* What metrics can be used to calculate expected hosting costs?

## 

## Methods

|  |  |  |
| --- | --- | --- |
| **Method Group** | **Method** | **Implementation** |
| Library | Available product analysis | Find out how other applications or companies determine their costs. |
| Library | Community research | Find out how others have kept their cloud hosting costs as low as possible. |
| Field | Document analysis | Go through the documentation of Microsoft Azure to see what they mention about hosting costs. |
| Lab | Data analytics | Try to find metrics within our application in order to measure the expected hosting costs |

### 1.1 How is the cost of a microservice application hosted on Azure determined?

#### 1.1.1 How are hosting costs measured?

Because we use Microsoft Azure we will be looking at how they determine pricing when using their service. Azure provides a very extensive [pricing calculator](https://azure.microsoft.com/en-us/pricing/calculator/) for their users which one can use in order to add different hosting scenarios and get an estimated cost prediction based on the settings chosen by the user. Since the costs of hosting are abided by the costs that Microsoft Azure provides, this source will be the most correct calculator in that regard.

Cloud services such as Microsoft Azure determine their pricing based on the expense of maintaining the network. They have hardware, maintenance and labor costs which all affect the end costs per setting of the user. Within the settings options there are multiple cost tiers as well as the more powerful the hardware service you’re asking for, the more you as a company have to pay. Usually these metrics come in cost per GB of virtual RAM, virtual Disk, usage-based subscription costs, backups costs and CPU speed to name a few.

**1.2 What metrics can be used to calculate expected hosting costs?**

Using Docker hub we can get the following data regarding docker images:

|  |  |  |
| --- | --- | --- |
| **Service** | **Size Compressed (mb)** | **CPU %** |
| Sunservice | 120 | *Varying* |
| GasService | 119 | *Varying* |
| WindService | 120 | *Varying* |
| MeteoserverService | 122 | *Varying* |
| Registry | 121 | *Varying* |
| Userservice | 128 | *Varying* |
| Gateway | 170 | *Varying* |
| Dashboard | 265 | *Varying* |
|  |  |  |
| MySQL (x2) | 154 (x2 = 308) | *Varying* |
| RabbitMQ | 83 | *Varying* |
| Redis | 37 | *Varying* |
| Prometheus | 67 | *Varying* |
| **Total CPU:** | | ~1.80GHZ idle load |

### 

### 1.3 Conclusion

When trying to calculate hosting costs, a lot of variables come into play for which you must take note of. Every cloud host provider has their own cost calculator which you can use to input the variables needed to get an estimated result, which can of course differ from reality. A lot of data can be found in one place such as prometheus or kubernetes which give them all neatly in one screen for the developer to see.

# Research Question 2

Which services can expect the most load given a large number of users and how can we improve or ensure reliability on scaling on these services?

## Sub-questions

* Which services in our architecture are expected to have the most load and how can we test it?
* Which parts of our architecture can be improved or changed to ensure reliability in scaling?

## Methods

|  |  |  |
| --- | --- | --- |
| **Method Group** | **Method** | **Implementation** |
| Library | Available product analysis | Find out how other applications or companies determine load in their application. |
| Library | Community research | Find out how others have used metrics or other variables in order to test load in their application. |
| Library | Literature study | Find out how to improve scalability within an application. |
| Field | Document analysis | Look at Microsoft Azure documentation to see what they mention about load balancing and load expectancy. |
| Lab | Data analytics | Try to find measurable metrics within our application for load expectancy. |
| Lab | Non-functional test | Try to find a way in order to test our application based on scalability. |

### 2.1 Which services in our architecture are expected to have the most load and how can we test it?

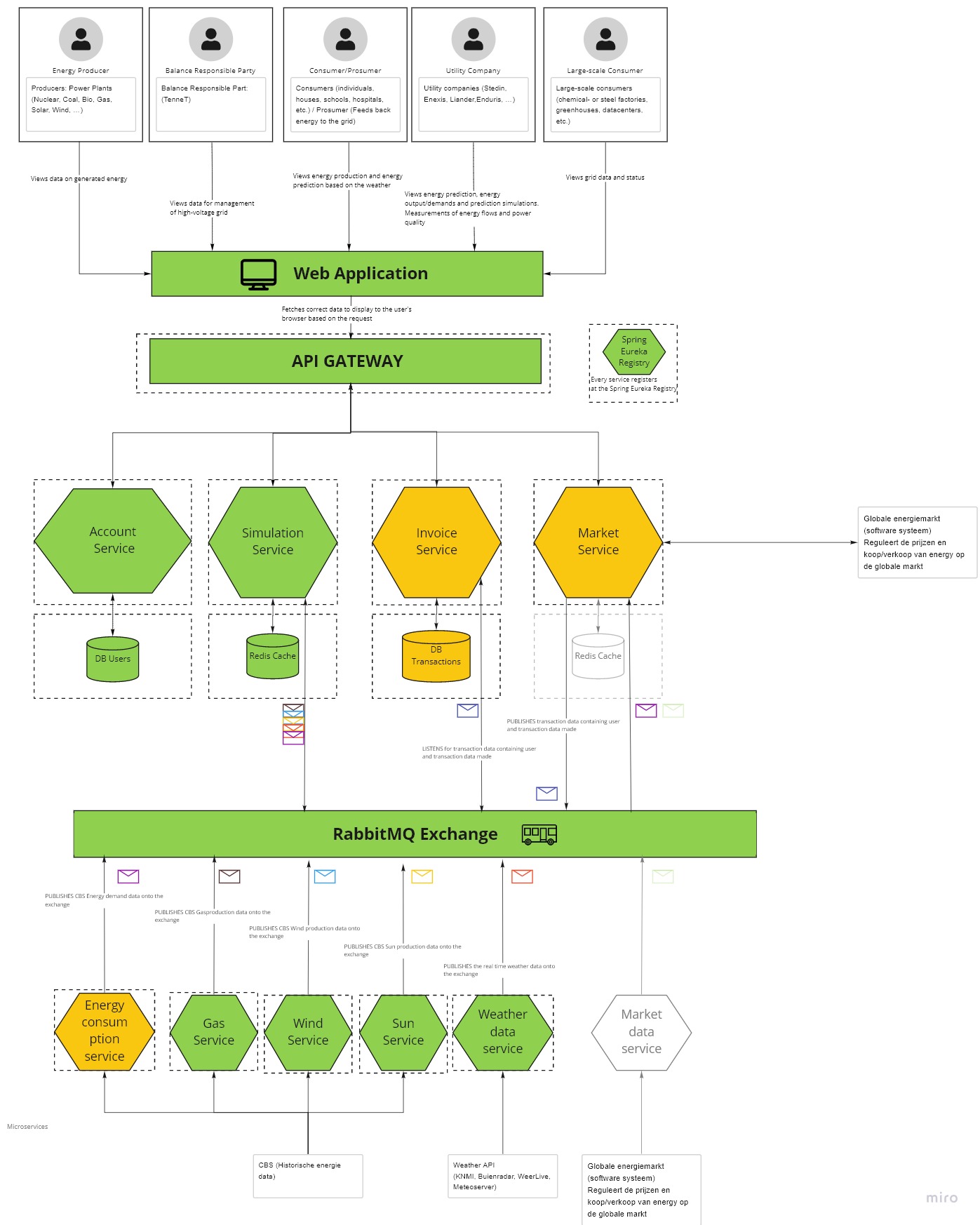
#### 2.1.1 Our architecture choices

This semester focussed on microservice based architecture and cloud solutions. For our group project we have made the following chases in order to increase scalability:

1. Usage of RabbitMQ’s publish/subscribe system which allows the different services to handle messages from the queue, even if a service goes down for a bit as the messages persist.
2. A redis cache on energy information requests as that one is mostly used for simulation and energy info purposes. As we anticipate that it is requested more than other services we have introduced a cache on the data for the most common requests. This reduces hops and requests between services and allows the simulation service to serve the information faster and more directly to every user.   
   If a lot of requests were to be made to the simulation service, then that would be the only service that needs scaling as the serving of the data is happening in this place only.
3. We separated the market and invoice service as we thought that invoicing is an action that a user is okay with receiving later, after the buy/sell of an order on the market. If the market service tends to be busy, we can independently scale the invoice and market service from each other based on the load.

With these decisions in mind we do expect the most load to come from the simulation and market service as those two services are both part of the main functionality of the application. The invoice service, while highly dependent on the market service, is okay with handling load at a slower pace so it doesn’t necessarily need to scale as fast as the market service.

The account service can scale depending on the amount of users active, but we can be prepared for a minimum amount of vertical scaling if we knew how many daily users were to always log in to use the other services. This could also be time dependent as workers enter office hours and exit them, so they need to log in and out of their systems.



*Fig 2.1 A -* [*C2 model*](https://miro.com/app/board/o9J_lSyGmWo=/) *of the backend architecture using Miro.*

#### 2.1.2 What tools exist to test load?

One thing to note is that Microsoft Azure had built-in load testing but is now deprecated as of March 31 2020, as for the importance of this mention, it is interesting to know their stated reasons as to why they removed these features:

* “Load testing is typically initiated for seasonal events such as…”
* “Load testing requires a certain level of expertise to ensure you have the confidence in the results. This ranges from understanding the application & deployment architecture, designing of load tests, authoring/executing of tests at scale and analyzing the results to identify performance and application bottlenecks.”
* “We’ve found that very few organizations rely on in-house expertise for this. Instead, most prefer to engage consultants to help them.”

They also stated that they were lacking in this area and other open-source alternatives were already ahead of them which gave them the position that their own tool is no longer needed.

As such they have recommended the following list of tools:

* [Soap UI](https://www.soapui.org/tools/soapui/)
* [Apache Jmeter](https://jmeter.apache.org/)
* [CloudTest Lite](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/soasta.cloudtest-lite?tab=overview)
* [Tricentis Flood](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/tricentis.flood_performance_load_testing?tab=overview)
* [Loadrunner Cloud](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/micro-focus.ms-stormrunner-load?tab=Overview)

The ones that we will take a look at are Soap UI and Apache Jmeter as open source alternatives as they are widely used and their set of features look easy to set up and use for our application. Another would be the usage of an npm loadtest package which is another easy to use API testing tool.

# 

### 2.2 Which parts of our architecture can be improved or changed to ensure reliability in scaling?

Our architecture which is shown in [fig. 2.1 A](#_l9i04oltsyil) is set up as a traditional microservice architecture. When looking at possible bottlenecks and user traffic we have already established that the simulation and market service will have the most varying and heavy hitting traffic while the account service will most likely have peak times at around work and closing times.

From here we have several options we can explore and one is moving some of the simulation functionality to the cloud. This is because of the following reasons:

1. The usage of a simulation service is highly dependent on the planning of e.g. construction of a new solar park and estimating energy production.
2. This makes the service rather unpredictable in terms of load and the timings of them as we do not know when companies are trying a lot of simulations at once. Wildly varying peak times are better off handled in the cloud than scaled up e.g. manually.

Moving the market service to the cloud could also be encouraged as the up-time of a market service is crucial. Having a cloud provider scale based on load would help rather than having to do it yourself.

Another option in addition to the one mentioned here is to split up the simulation service into different parts of energy simulations. As of right now a single microservice handles \_all\_ simulations while different companies could work on using different energy parts of the simulation which would unnecessarily scale up unused parts of the service. This also makes it more fail safe as only single parts of the simulation would then break instead of it together introducing more complexity, but more reliability in the service.

### 2.3 Conclusion

The parts of our architecture where we expect the most load seem to make sense and we have prepared some architectural preparations for it. While we now know we could’ve split up some more parts to make it run more efficiëntly for scaling purposes, we still have that future open for change due to the microservice structure that we have.

While searching for cloud testing tools we have come up with 2-3 good tools that we would like to use, as our primary testing would be api testing for which SoapUI, Apache Jmeter and Npm load test are candidates for future use as they also give some insights on the test results as well.

**Reflection**

In the end I didn’t get to do a lot of lab testing apart from watching some statistics in kubernetes during idle load. Although it does come in handy to get a glimpse of idle costs, we didn’t get that far into the project yet where we can load test yet which is where the real statistics come into play. Next time I could maybe spare some time by making a smaller side project in which to test load on when an api is being called for example to get a general idea of what the cpu % consumption might be like.

In terms of thinking about our architecture, while we already did it at the beginning of the project, it was nice to look at our implementation version and validate our choices as I went through each service and their use cases in order to figure out which service will get hit the most. Next time I would like to properly load test to see if I was right.

# Sources

|  |  |
| --- | --- |
| **Researcher** | **Source** |
| Vincent | Microsoft. (n.d.). Pricing Calculator. Retrieved May 20, 2021, from <https://azure.microsoft.com/en-us/pricing/calculator/> |
| Vincent | Palian, J. (2015, May 1). How the Cost of Cloud Computing is Calculated. Retrieved May 20, 2021, from <https://expedient.com/knowledgebase/blog/2015-05-01-how-the-cost-of-cloud-computing-is-calculated/#:%7E:text=When%20setting%20price%2C%20cloud%20providers,need%20for%20its%20IaaS%20cloud>. |
| Vincent | Paszkowski, B. L. (2021, March 15). Re: cloud aanvragen voor jullie projecten [Comment]. Retrieved from <https://fhict.instructure.com/courses/11008/discussion_topics/60576> |
| Vincent | Cool, J. (2021, April 20). Cloud-based load testing service end of life. Retrieved May 20, 2021, from <https://devblogs.microsoft.com/devops/cloud-based-load-testing-service-eol/> |
| Vincent | SmartBear. (n.d.). The World’s Most Popular API Testing Tool | SoapUI. Retrieved May 20, 2021, from <https://www.soapui.org/tools/soapui/> |
| Vincent | Apache Software Foundation. (n.d.). Apache JMeter - Apache JMeterTM. Retrieved May 20, 2021, from <https://jmeter.apache.org/> |
| Vincent | SOASTA. (n.d.). CloudTest Lite. Retrieved May 20, 2021, from <https://azuremarketplace.microsoft.com/en-us/marketplace/apps/soasta.cloudtest-lite?tab=overview> |
| Vincent | Tricentis. (n.d.). Tricentis Flood. Retrieved May 20, 2021, from <https://azuremarketplace.microsoft.com/en-us/marketplace/apps/tricentis.flood_performance_load_testing?tab=overview> |
| Vincent | Micro focus. (n.d.). Getting Started with LoadRunner Cloud. Retrieved May 20, 2021, from <https://azuremarketplace.microsoft.com/en-us/marketplace/apps/micro-focus.ms-stormrunner-load?tab=Overview> |
| Vincent | npm: loadtest. (2020, November 20). Retrieved May 25, 2021, from <https://www.npmjs.com/package/loadtest> |