Automatization of Scaling

Horizontal Insights

Teun van der Kleij (s1188351), Bas Plat (s1184827), Ties Greve (s1128147) en Jeroen Terpstra (s1189144)

Abstract:

In this research paper, the automatization of the scaling process within Kubernetes clusters at Vultr is researched. The hosting provider, Vultr, was chosen based on the requirements set by the startup company of Ernst Bolt for which this research is developed. The goal of this paper is to find a way to develop a methodology that can forehanded and efficiently automatically scale the nodes and nodepools, so applications deployed on the cluster can perform consistently without excessive use of resources. This research identifies the measurement data needed for scaling, the model, like Prophet and Neuralprophet, that can be used to predict the future cluster load and tests the creation process of nodes and nodepools in a confined environment with the use of prototyping.

The results show that the prediction model Prophet offers the most viable balance between the accuracy and speed. The data needed for the Prophet model to predict, is found to be CPU and RAM usage of all the different nodes. Furthermore, the average startup time for a node has been established to be around ten minutes, which forms the basis for the scaling strategies. Moreover, the limitations of the testing environment and availability of real-world data is discussed. The research emphasizes the importance of robust data systems and advises further refinements of the choice in prediction models, like LSTM-networks, once more prediction data is available.

1 Introduction

In this research paper, the automation of the scaling within Kubernetes clusters of nodes and nodepools at Vultr will be discussed. The hosting provider, Vultr, was chosen based on the requirements set by the startup company of Ernst Bolt for which this research is developed. There will be mostly looked at how to be able to start this process in a timely manner.

The goal of this paper is to find a way to develop a methodology that can forehanded and efficiently automatically scale the nodes and nodepools, so applications deployed on the cluster can perform consistently without excessive use of resources. This research is of great importance, since there is currently no way to scale nodepools automatically at Vultr. The missing component with the current scaling possibilities, is that it excepts all nodes to have the same specifications.

To achieve this goal, these previously mentioned aspects will be researched. This process starts with gathering a global image of what data needs to be measured to be able to make predictions. Thereafter, there will be investigated how, with this data, the predictions can be made for how much scaling needs to happen soon.

This research is of primary importance to the product owner of the startup company of Ernst Bolt. Secondarily, it can provide insights for other individuals and companies, who may face the same issue.

This research will be completed in a timespan of two weeks.

1.1 Research questions

1.1.1 Primairy question

The primary question for this research is as follows:

How can the scaling process of nodes on nodepools with Kubernetes be automated for Vultr?

1.1.2 Sub-questions

The sub questions to be able to answer the primary question are as follows:

Which data needs to be measured to determine what is needed for the scaling process?

- How often does the data need to be measured?

Which method(s) need to be used to be able to predict the scaling?

How can the scaling process be started in a timely manner?

- *How long does it take for nodes and nodepools to be created?*

How will the scaling process be implemented?

2 Methods

2.1 Literature research

In this research paper, there will be made use of literature research. The goal of this literature research is to research what information the hosting provider, currently used by the startup company, provides regarding the possibilities of implementing a custom scaling service. Furthermore, various methods to make predictions based on data analyzed over time will be explored.

For this literature research, the main type of sources that are used are trustworthy online sources. By trustworthy online sources, scientific research articles are meant. Each source is checked by looking at the objectivity of the source. Moreover, it is important that other research papers are referencing the source to confirm the information is deemed trustworthy by other researchers as well. Lastly, the authors and publication date will be examined.

Furthermore, a source used in this research paper can also be a technical source, like GitHub repositories and documentation of verified companies and applications.

2.2 Interview

As second research method, there will be made use of an interview. An interview has been planned with Mischa Mol, lecturer and researcher of Applied Data Science & AI. He can provide us with information about what are best practices and best type of methods to use for the predictions.

2.2.1 Interview questions

The follow questions are drafted for the interview with Mischa Mol. These questions are a guideline for the interview. Based on the given answers, there can be follow-up questions.

- Which methods can be used to ensure the new predicted capacity will be turned on in time?

    - Do we need to consider the resources and time used by these methods in the calculation?
- What can be done to ensure the predictions are trustworthy?
    - For example, is historic data needed for a larger trust in the prediction? If so, which?
- What is an efficient way to test eventual prototypes?
    - What is a proper source for test data in case this is necessary?
- What would need to change once the real data comes in the methods?
    - Is it possible to keep this in consideration beforehand?
- Is it useful to predict multiple values at once, e.g. CPU and memory?

2.3 Available product analysis

As a third research method, available product analysis is used. The goal of this research method is to find existing, comparable systems, examine which data is used by other providers for their scaling process, and how those providers handle their predictions.

2.4 Prototyping

The final research method that will be used in this research is prototyping. With the use of testing, data can be gathered about the startup times of nodes and nodepools. Based on these results, conclusions can be made about the time needed to start the scaling process in a timely manner.

3 Results

3.1 Which data needs to be measured to determine what is needed for the scaling process?

Calculation an effective way of managing Kubernetes Clusters require a thought through approach to determine when the nodes and nodepools need to be scaled. By making use of preventive scaling strategies, unnecessary resources and underperforming applications are prevented. The data that needs to be measured, how this data can be used in scaling decisions and which fallback-strategies can be implemented in unforeseen situations will be discussed the coming chapters.

3.1.1 Measuring data needed for scaling decisions

To plan the scaling of Kubernetes nodes accurately, the following data is essential (Ju, 2021):

- **CPU-usage:** Measuring the CPU-usage of the nodes provides an insight in the load on the cluster. High and long-lasting CPU-load can be an indication that more capacity is needed. Furthermore, it can show that too many have been created when the CPU-load is load for a significant amount of time. It is important to take the reserved CPU in mind. It is possible that CPU is not being used but is reserved.
- **Memory usage**: Just like CPU-load, it is important to monitor the memory usage. If the memory usage is consistently too high, applications may fail due to memory exhaustion. This shows the need for scaling. When memory exhaustion occurs, Kubernetes will not be able to schedule more pods.
- **Requests**: The number of incoming requests to the applications on the cluster provides a direct indication of the workload. A rising trend in requests can anticipate future need of resources (Coralogix, 2024).

3.1.2 Model output

The following parameters can be predicted when modeling scaling decisions:

- **Nodes**: The expected number of nodes needed for the optimal cluster performance.
- **Total number of virtual CPU's (vCPU's)**: The number of vCPU's needed to manage the expected cluster load without performance loss.
- **Total amount of RAM**: The predicted amount of memory needed for the optimal performance of the cluster.
- **Requests**: The expected increase or decrease of incoming requests, allowing resources to be adjusted accordingly

3.1.3 Fallback strategies

In case of unforeseen circumstances of inaccurate predictions, the following indicators can be used to initiate scaling actions (DavidW, 2024):

- **Response time**: An increase in response time of the applications may indicate overload, requiring immediate scaling
- **CPU-usage**: When CPU-usages suddenly rises sharply and sustains the high level, it can be a trigger to create more nodes.
- **Memory usage**: High memory usage with direct predictions may be a cause for scaling to prevent crashing. The reserved memory needs to be taken into consideration. Memory might not be used but may be reserved.
- **Unscheduled pods**: Unscheduled pods might not be scheduled due to resource limits. This may be an indication that more capacity is required (Rabih, 2022).

3.2 Which method(s) need to be used to be able to predict the scaling?

Two commonly used models for time-based predictions are ARIMA en SARIMA (Brownlee, 2020). These models were brought up during the interview with Mischa Mol (see Appendix 1: Elaboration interview Mischa mol). There are more accurate , but heavier models like Random Forest Regression (AnalytixLabs, 2023), LSTM (Alhamid, 2021) and Neuralprophet (Laptev, Triebe, & Hewamalage, 2021). Furthermore, there are alternatives like Prophet (Brownlee, 2020), AutoTS (AutoTS, 2024), Darts (Time Series Made Easy in Python, 2024) and simpler models like naive forecasts.

During the interview (Appendix 1: Elaboration interview Mischa mol), Mischa pointed out that the choice should be between Prophet and Neuralprophet. He recommended to put specialized version aside, since there is no real data available to compare the models.

To research these models, multiple prototypes have been developed. The data used for prototyping originates from the Kennedy Space Center (Dumoulin, 1996). This data has been modified to train the models (ARIMA, SARIMA, Prophet and Neuralprophet) to our specific needs. Before, each record in the dataset contained the time and request sent over a two-month period. The data has been modified to a table with time intervals of five minutes and the number of requests per interval. For the periods with a low number of requests, a second version of the dataset has been created. In this second dataset, the time intervals with missing data have been filled up by interpolation. Each model can now be tested.

The dataset has been split from the 16th of august onwards. Thirty percent of the data has not used in the training period, but that part has been used in the testing period. This date has been put in the middle of the working week (Wednesday), so accurate patterns of a week can be created.

During the testing phase, the time the model needed for training and predicting has been measured. These tests have been executed on the same machine with a time slot of five minutes for the most accurate data.

After the testing phase, the models have been compared based on accuracy. In the figures below (1 t/m 4), the blue lines show historical data, the green line shows the predicted data, and the red lines shows the actual data.

Figure 1 shows a graph with the predicted data using the ARIMA-model. This figure shows that the model does not take variations in the days into account, causing the model to predict a horizontal line.

Figure 1: Graph of predicted data ARIMA

Figure 2 shows the graph with the predicted data using the SARIMA-model. This model was heavier to train. Therefore, the choice was made to train this model with intervals of an hour instead of five minutes.

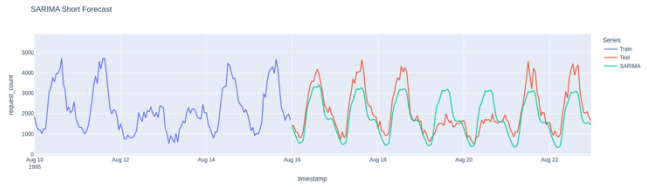

Figure 2: Graph of predicted data SARIMA

Figure 3 shows the Prophet model and Figure 4 shows the Neuralprophet model. These two models have comparable predictions with a difference of one percent in their accuracy of their prediction. However, Neuralprophet took longer to train.

Figure 3: Graph of predicted data Prophet

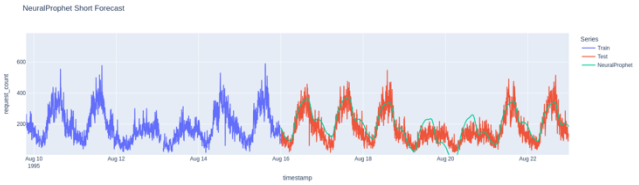

Figure 4: Graph of predicted data Neuralprophet

The results of the prototyping can be found in Table 1: Results of the prototyping testing. This table contains the previously tests models. These models are compared against each other based on training time and accuracy of the prediction.

Model name	ARIMA	SARIMA	Prophet	Neuralprophet

Training time (seconds)	5.1	37.1	2.9	19.4[1]
Prediction time – 1 week (seconds)	0.0	0.0	2.7	0.0
Prediction time – 16 days (seconds)	0.1	0.0	2.1	0.0
MAPE[2] – 1 week	53%	31%[3]	44%	43%
MAPE[2] – 16 days	50%	40%[3]	45%	44%

Table 1: Results of the prototyping testing

The results show that ARIMA is the fastest model, while Neuralprophet is the most accurate. Prophet is less accurate but faster, making it the preferred choice for the prediction model.

3.3 How can the scaling process be started in a timely manner?

The discussed scaling will need to start in a timely manner. If scaling is delayed until each node is fully utilized, there will be a period in which the cluster cannot handle the incoming requests. This is not a desirable situation.

By scaling proactively, load on the nodes and nodepools will be spread evenly and overloading will be prevented.

3.3.1 How long does it take for nodes and nodepools to be created?

To gain a better image about the amount of time needed to create nodes and nodepool, tests will be executed. The plan used for these tests will be discussed in the coming chapter.

3.3.1.1 Goal

The goal of testing the startup times of nodes and nodepools within Vultr is to determine what the average creation time is. With the results of the tests an estimation can be made about how early the scaling process needs to start.

3.3.1.2 Scope

The tests will consist of three different types of nodetypes from Vultr (Regular performance, AMD High performance and Intel High performance). Per nodetype, two nodeplans will be selected for the tests (2CPU with 2 GB RAM and 4 CPU with 8 GB RAM).

Three different periods have been selected to run the tests on. The choice for these times have been made based on the common start- and end times for a workday (tests starts from 9.00 AM and 4.00 PM) and another moment in the middle of the day (tests starts from 1.00 PM)[4].

3.3.1.3 Testing environment

On the Vultr environment of the startup of Ernst Bolt, three clusters have been used to run the tests.

3.3.1.4 Test execution

The tests are executed at the previously described periods. Furthermore, the tests will be run across two days: Friday 13 December 2024, and Sunday 15 December 2024.

The tests will be run on (a) cluster(s) with the starting situation on the cluster being one node with one nodepool.

On both days the tests will be run by creation one nodepool with one, two and three nodes. These tests will be executed in each period.

The tests are executed by making an API-call to the API of Vultr.

3.3.1.5 Test results

The test results are included in Appendix 2: Test results of startup time testing. The test was not fully conducted due to external factors, explained in **Error! Reference source not found.** Testing. In agreement with the product owner, it is determined that a conclusion can still be made from these results.

3.3.1.6 Conclusion

From these results, the following ranking has been made:

- 1: The fastest
- 2: The average
- 3: The slowest

[1] Trained on a machine which had a GPU available

[2] MAPE stands for *Mean Absolute Percentage Error.* This is a benchmark metric used for determining the accuracy of the models. A low MAPE shows a more accurate model.

[3] MAPE is lower, since this model uses intervals of one hour which means there is less noise relative to the other models.

[4] The tests are executed in the time zone of CET (+1 GMT)

	Intel High performance	Regular performance	AMD High performance
Nodepool creation time (2cpu2gb)	1	3	2
Node creation time (2cpu2gb)	3	1	2
Nodepool+Node creation time (2cpu2gb)	1	3	2
Nodepool creation time (4cpu8gb)	2	1	3
Node creation time (4cpu8gb)	1	3	2

Table 2: Ranking of nodetypes

Table 2: Ranking of nodetypes shows that "Intel High performance" comes out of the test as the best. A point to be made is the differences between the nodetypes are minimal.

The test results reveal that there is no significant difference between days during the workweek and the weekend for both the creation of nodes and nodepools. Furthermore, the difference in startup times of nodepools with different number of nodes resulted in an inconsistent factor and is not taken into consideration in the conclusion.

The data did show the startup times of nodes was so close that the conclusion can be made that the creation process is an asynchronous process.

Moreover, the entire startup time (nodes and nodepools together) did not reveal any significant difference with different number of nodes. The biggest difference found was a difference of eight seconds on a time of 8:48 minutes and 8:40 minutes.

The startup times of nodes does differ per nodetype and nodeplan. This data will not result in any difference, since the startup time of the nodes and nodepools together do not show a significant difference.

The average time for the creation process of (a) node(s) with one nodepool was 8:43 minutes.

3.4 How will the scaling process be implemented?

The prediction model will be part of a larger module responsible for the entire scaling process. The architecture and setup of this module is not part of the scope of this research. This research will only discuss the implementation of the prediction model within this module. For the model, the languages Python and R are the most convenient to use. This is due to the large number of available packages and the extensive community support for statistics (Wainaina, 2023).

The result of the prediction model can be expressed into a new unit, which we will call the Node Ranking Index (NRI). This unit is a ratio based on the power of a node. This NRI is visible in Table 3: Node Ranking Index (NRI)

Index	Plan	vCPUs	Memory	Hourly Price
2	vhp-2c-2gb-intel	2	2GB	$0.027 /hr.
4	vhp-4c-8gb-intel	4	8GB	$0.071 /hr.
8	vhp-8c-16gb-intel	8	16GB	$0.143 /hr.

Table 3: Node Ranking Index (NRI)

4 Discussion

4.1 Conclusion

To be able to scale nodes and nodepools automatically, difference parts are needed. The first part is the prediction model. The choice of model has come down to Prophet. The prediction model will be developed in Python, since Python provides packages needed for the statistical methods and models for prediction, and the developers for this project have experience with Python unlike R. With the limited realization phase for this model, the choice has come down to the language where the developers have the most experience.

The data that will be used in the predictions can best be saved in a separate database. This database will receive information about the cluster multiple times per minute. This data will consist of the number of nodes and information per node, like CPU- and RAM-

usage and the number of requests. With this data an accurate image can be created.

Since the model has influence on the future data needed for training, it is necessary to take precautions to prevent error accumulation. To prevent the model affecting the future decisions, a combination of fallback-mechanisms and downward adjustment will be applied. This will verify if there are unscheduled pods, and the CPU-usage will be taken into consideration.

By correcting under-predictions with a fallback-method and over-predictions with a downward-adjustment, the model will not fall into a feedback loop. This security mechanism will ensure a more stable and trustworthy model.

The tests results of the startup times let us to draw several conclusions. The creation of nodes is an asynchronous process and has no relations with the creation time of the nodepools. Furthermore, the nodetype “Intel High performance” has come best from the test for all nodeplans. The results showed that the average creation time was 8:43 minutes. To ensure the process starts early enough, the prediction model will look a minimal of ten minutes into the future to scale if necessary.

4.2 Discussion

4.2.1 Testing

The tests about the startup times contain limited data. These limitations are caused by both Vultr as time.

During the second test day, the original plan was to test during all three time periods. However, errors from Vultr were given about limits with creation and removal of nodes and nodepools. As a result, no more calls could be made through the API of Vultr until the number of nodes and nodepools got below the limit again. Together with the product owner, the decision was made to not continue the tests and draw a conclusion based on the data already available to us. Originally, there would have been two more day of tests, however these had to be taken out of the test plan.

Furthermore, the tests were limited by time as well. For greater quality of data, the testing should be conducted over more days with multiple nodepools to be able to compare more and gain a more objective conclusion.

Due to the limits previously discussed, tests with just creating a node on an existing nodepool have not been conducted either.

As a result of these limits, the tests were executed on one day during the workweek and partially during one day in the weekend. On the day during the workweek, it was possible to obtain an entire day of data. This was not the case for the day during the weekend. The collected data is in Appendix 2: Test results of startup time testing.

4.2.2 Researched models

The number of models that were tested for sub question 2 is limited. With more time, more models could have been tested. This could lead to different and better data. Still, those tests would not provide a completely accurate image, since no actual data was available to test the models against.

4.2.2.1 Little expertise

A second interview with another Machine-Learning expert could provide more insights. With this second interview more feedback and confirmation of conclusions could be gathered.

4.3 Advises

We advise to save more data in the database than mentioned so far. This would be useful for future expansions of the prediction model to be trained quickly. Gathering this information proactive, it will be easier to switch to a more complex and advance model in the future.

Moreover, we advise to change to a neural network in the future, like LSTM. LSTM-networks are most suitable for processing and analyzing time-depending data and sequences, like response times, planning timings and trends in requests. With the use of a LSTM-model, complex patterns and dependencies in the data will be better identified and predictions will be refined.

4.4 Follow-up research

In follow-up research, the tests of the models can be rerun with the real data available of the cluster load. With this data, more advanced and complex models can get tested, like neural networks. These models provide the opportunity to make more accurate predictions and support understanding the cluster.

Furthermore, larger tests could be made for the startup times of nodes and nodepools. We advise to expand these tests with more test days, different nodeplans for

different nodetypes and include tests for just adding nodes to an existing nodepool. This last test would be useful to see the difference between adding nodes on an existing nodepool and a newly created nodepool. This could be beneficial for the predictions.

Bibliography

Alhamid, M. (2021, Juni 26). *LSTM and Bidirectional LSTM for Regression*. Retrieved from Medium: https://towardsdatascience.com/lstm-and-bidirectional-lstm-for-regression-4fddf910c655

AnalytixLabs. (2023, December 26). *Random forest regression*. Retrieved from Medium: https://medium.com/@byanalytixlabs/random-forest-regression-how-it-helps-in-predictive-analytics-01c31897c1d4

AutoTS. (2024, December 9). Retrieved from ouro: https://ouro.foundation/posts/chronos/018fcfc1-e07e-7cbb-bbf8-26a166429bda

Brownlee, J. (2020, Mei 8). *Time Series Forecasting With Prophet in Python*. Retrieved from Machine learning mastery: https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/

Coralogix. (2024, December 17). *Top 6 Kubernetes Metrics & How to Monitor Them*. Retrieved from Coralogix: https://coralogix.com/guides/kubernetes-monitoring/kubernetes-metrics/)

DavidW. (2024, Maart 14). *Mastering Predictive Scaling in Kubernetes*. Retrieved from Medium: https://overcast.blog/mastering-predictive-scaling-in-kubernetes-6e09501afbec

Dumoulin, J. (1996, May). *NASA-HTTP.* Retrieved from Traces In The Internet Traffic Archive: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

Greve, T., & van der Kleij, T. (2025, Januari 14). *Research-Scaling-Automatization*. Retrieved from https://github.com/Scaling-Insights/Research-Scaling-Automatization

Greve, T., & van der Kleij, T. (2025, Januari 14). *Research-Scaling-Automatization*. Retrieved from https://github.com/Scaling-Insights/Research-Scaling-Automatization

Ju, L. (2021). *Proactive auto-scaling for edge computing systems with Kubernetes.* New York: Association for Computing Machinery.

Laptev, N., Triebe, O., & Hewamalage, H. (2021, November 30). *NeuralProphet: The neural evolution of Meta's Prophet*. Retrieved from Meta: https://ai.meta.com/blog/neuralprophet-the-neural-evolution-of-facebooks-prophet/

Rabih, E. (2022, Maart 22). *6 Metrics to Watch for on Your Kubernetes Cluster*. Retrieved from Komodor: https://komodor.com/learn/6-metrics-to-watch-for-on-your-kubernetes-cluster/

Time Series Made Easy in Python. (2024, December 9). Retrieved from unit8co: https://unit8co.github.io/darts/

Wainaina, P. (2023, Oktober 24). *The Complete Guide to Time Series Forecasting Models*. Retrieved from Medium: https://medium.com/@wainaina.pierre/the-complete-guide-to-time-series-forecasting-models-ef9c8cd40037

During the preparation of this work, parts of sentences have been reformulated and summarized with the support of ChatGPT.

Appendix 1: Elaboration interview Mischa mol

- What would need to change once the real data comes in the methods?
    - Is it possible to keep this in consideration beforehand?
- Is it useful to predict multiple values at once, e.g. CPU and memory?

Which methods can be used to ensure the new predicted capacity will be turned on in time? Do we need to consider the resources and time used by these methods in the calculation?

It sounds like a time series. It will most likely not go from zero to one hundred.

First thought, it is about a time series, thus SARIMA. SARIMA has seasonality. Otherwise ARIMA, a model with ore accuracy than prophet. A step further is neuralprophet, however this requires a GPU, and a neural network is required for this as well. Try these three methods first. SARIMA will not perform well, so look at prophet and neuralprophet.

There are a few extremely specific

There are also several highly specialized models based on these techniques, but without training data, this is very difficult. If prophet or neuralprophet would be better, I cannot say.

What can be done to ensure the predictions are trustworthy? For example, is historic data needed for a larger trust in the prediction? If so, which?

Try real data and data like real data. Furthermore, there is not much more you can do besides more training data.

What is an efficient way to test eventual prototypes? What is a proper source for test data in case this is necessary?

You can search on Kaggle for a dataset similar to the data you are doing to work with.

What would need to change once the real data comes in the methods? Is it possible to keep this in consideration beforehand?

It is most convenient to look at a dataset like yours. This is most useful when real-life data needs to be used.

You can see that with each data point, it becomes more trustworthy. It might be a good idea to first gather data from e.g. a month before the model can be used properly.

Is it useful to predict multiple values at once, e.g. CPU and memory?

Possibly, do look at the complexity. The more values that are added, the more complex it becomes. This would mean you come closer to a neural network as well. For now, I would stick with the few outputs.

How can the model train quickly and with what kind of data?

Is a bit complicated. How many training points are needed for the current data?

Two months' worth of data with seasonality of 288.

Is data each five minutes necessary, why not ten or fifteen minutes? It is peculiar that it takes such a long time now. This is possibly due to the high seasonality. All ARIMA derivatives are good for understanding, however, do look quickly at other models, since these are more accurate.

Appendix 2: Test results of startup time testing

For each test, the values are recorded in seconds with millisecond precision. For indication of a decimal, the comma (,) is used.

Workweek day Tests

Intel High Performance

2cpu2gb	**9:00**	**13:00**	**16:00**
Nodepool	87,872	67,476	103,769
1 Node	522,095	520,988	516,212
Nodepool	62,309	28,439	139,388
2 Nodes	525,403	519,449	519,065
Nodepool	41,700	51,801	110,837
3 Nodes	535,620	522,479	528,669

4cpu8gb	**9:00**	**13:00**	**16:00**
Nodepool	43,792	78,094	134,021
1 Node	517,285	521,543	536,671
Nodepool	86,740	102,003	96,810
2 Nodes	521,988	521,622	531,142
Nodepool	58,020	126,682	126,004
3 Nodes	524,325	525,225	524,483

Regular Performance

2cpu2gb	**9:00**	**13:00**	**16:00**
Nodepool	125,197	90,815	99,143
1 Node	524,769	522,895	526,818
Nodepool	48,026	127,538	90,551
2 Nodes	525,261	525,887	524,672
Nodepool	59,680	123,413	86,078
3 Nodes	528,259	531,067	531,052

4cpu8gb	**9:00**	**13:00**	**16:00**
Nodepool	28,221	32,581	62,879
1 Node	518,673	518,278	527,489
Nodepool	103,083	61,628	94,233
2 Nodes	523,607	527,649	528,189
Nodepool	141,713	66,175	55,465
3 Nodes	528,573	532,870	531,851

AMD High Performance

2cpu2gb	**9:00**	**13:00**	**16:00**
Nodepool	41,718	73,201	46,612
1 Node	527,201	521,523	520,390
Nodepool	68,736	53,223	100,807
2 Nodes	519,123	523,422	522,170
Nodepool	62,454	83,890	89,964
3 Nodes	528,774	527,795	527,480

4cpu8gb	**9:00**	**13:00**	**16:00**
Nodepool	73,197	73,201	81,644
1 Node	524,441	521,523	524,279
Nodepool	36,831	53,223	139,158
2 Nodes	522,599	523,422	532,105
Nodepool	142,231	83,890	147,863
3 Nodes	533,421	527,785	555,875

Weekend day Tests

2cpu2gb	**Intel**	**Regular**	**AMD**
Nodepool	96,858	106,552	102,129
1 Node	520,905	530,938	527,507
Nodepool	76,889	108,531	103,298
2 Nodes	529,895	531,447	527,507
Nodepool	71,822	108,153	102,823
3 Nodes	525,702	533,631	532,546

Appendix 3: Explanation test project for startup tests

The test project used for executing the tests of the startup times is a .NET C# project. In this project, the necessary endpoints to the Vultr API have been included. This project can be found in the Scaling-Insights organization on GitHub in the repository called “Research-Scaling-Automatization”. The project name is “StartUpTimeTestingProject” (Greve & van der Kleij, Research-Scaling-Automatization, 2025).

In the upcoming text, methods are underlined, and italic (*method*) and properties are in italic (*property*).

Properties en methods

Different configurations can be done by changing the following properties:

- *vkeId*: This is the ID of the cluster.
- *token*: This is the API key of the Vultr account.
- *backendImage*: This is the link to the docker image of the backend on the container registery on the Vultr account.

Furthermore, three different types of nodeplan with two variations per nodeplan have been included as properties. Tests can be executed with one of these tests by setting the property *nodePlan* to one of those values.

To execute the tests, the static method *TestProcess* needs to be run. It is mandatory for both the wished amount of pool as nodes to have been filled in.

Inside the *TestProcess*, the duration of the creation and deletion of nodes and nodepools is logged to the console.

There is a method in the project that can create clusters. This method has been included to be able to run tests across different clusters async, without interference with tests on a different cluster (*CreateCluster*).