CHALMERS

MICROSERVICES

SCALABLE • PORTABLE • FAST • RECOVERY

Encouraging automated tests for code-to-container delivery

Bachelor's thesis in Computer Science and Engineering

LUKAS CARLING
JOHAN FRIDLUND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY,
GOTHENBURG UNIVERSITY,
Sweden 2021

DEGREE PROJECT REPORT

Encouraging automated tests for code-to-container delivery

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY,
GOTHENBURG UNIVERSITY,
Sweden 2021

Encouraging automated tests for code-to-container delivery
LUKAS CARLING
JOHAN FRIDLUND

Supervisor: Jonas Duregård
Examiner: Arne Linde

Department of Computer Science and Engineering
Chalmers University of Technology / University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover: Illustration of microservices

Department of Computer Science and Engineering
Gothenburg 2021

Abstract

This thesis work is being performed in collaboration with Yolean, a company developing solutions for project and site management, using Kubernetes and microservices. Automation testing is popular and widely used in microservices development to ease the work for developers. However, Yolean has found automation testing less attractive due to the amount of noise it generates when tests are performed, for example, on microservices with partial health. The aim of this project is to further develop the test suit to ease test-management and ensure that tests produce relevant data.

In this project, the Kubernetes structure and the development environment was built upon Yoleans public platform Ystack. Through the use of this platform as well as regular communication with Yolean about the user experience within Ystack issues were being presented and prioritised. The approach to solve these issues was to research, find and develop improvements to them, which were then presented to developers at Yolean. Through user experience and their feedback it was decided to either keep the solution as it is, further improve the solution, or remove it.

Our findings suggest that there are further strong improvements to be made in order to encourage automated tests. The resulting solutions developed in this project was a timeout to kill tests that are running indefinitely to prevent database clogging, stop tests after they have passed a set amount of times, and add a delay to starting testing to reduce startup failing test data.

Keywords: Kubernetes, microservices, automation testing, jest, k8s

Sammanfattning

Detta examensarbetet sker på uppdrag från Yolean som jobbar med utveckling av lösningar för projekt- och platsledning och använder sig av verktyg som Kubernetes för utveckling av mikrotjänster. När det kommer till utveckling av mikrotjänster är det populärt med automationstester för att underlätta arbetsprocessen för utvecklarna. I Yoleans fall har de funnit att användandet av automationstester i vissa fall är kontraproduktivt då det kan generera överflödig data vid testning av exempelvis mikrotjänster som inte är helt stabila. Målet med detta projektet är att vidareutveckla testmiljön på Yolean för att underlätta användandet av automationstester och säkerställa att relevant data produceras av tester.

I projektet är utvecklingsmiljön byggd på Yoleans allmänt tillgängliga plattform Ystack. Genom användandet av plattformen och deras testmiljö, samt regelbunden kommunikation med företaget om hur det är att använda och utveckla på deras plattform kom vi fram till problem som behövde åtgärdas. Problemen prioriterades ofta hos Yolean och förslag på tillvägagångssätt gavs för att få flera ingångsvinklar på lösningarna. Dessa presenterades sedan till utvecklare på Yolean och genom kommentarer och åsikter efter användning bestämdes om lösningarna behövdes utvecklas vidare på, om de var färdiga, eller om de inte bidrog och kunde avskaffas.

Våra resultat visar på att det finns flera starka förbättringar som kan göras för att främja automatiserade tester. De resulterande lösningarna som utvecklats i detta projekt var ett avbrott vilket dödade tester som körde obegränsat, att stanna tester som passerat ett antal gånger, samt en fördröjning som reducerar felaktig testdata som skapas vid start av ett kluster.

Keywords: Kubernetes, microservices, automation testing, jest, k8s

Preface

We want to begin this report by thanking Jonas Duregård for becoming our supervisor. We also want to thank Yolean and Staffan Olsson for assistance in the project which has been of great help in analyzing the improvements we have created.

Terminology

A lot of the terminology will be explained in the theory chapter and is therefore not included in this chapter.

Node Package Manager (NPM) - An online repository for publishing open-source node.js projects. Including functionality is a command-line utility for interacting with said repository.

OS - Operating System

CI - Continuous Integrations is the practice of automating the integration of code changes from multiple contributors into a single software project.

CD - Continuous Delivery is a software engineering approach in which teams produce software in short cycles.

AWS - Abbreviation for Amazon Web Services. A cloud service.

Microservices - *Microservices* is a specialization of an implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software.

Contents

1. Introduction

This chapter will present the background and employer of this thesis work. Furthermore, it will also contain a description of the purpose, goal, and delimitations of the project.

1.1. Background

What defines a microservice varies depending on who receives the question, but in general terms it is an architectural approach in software development which utilizes a collection of services to form a fully functional application **[2][3].** The standard approach aside from this is a monolithic application architecture where all services and components are together in a singular program **[2]**. This difference can be seen in Figure 1 more clearly.

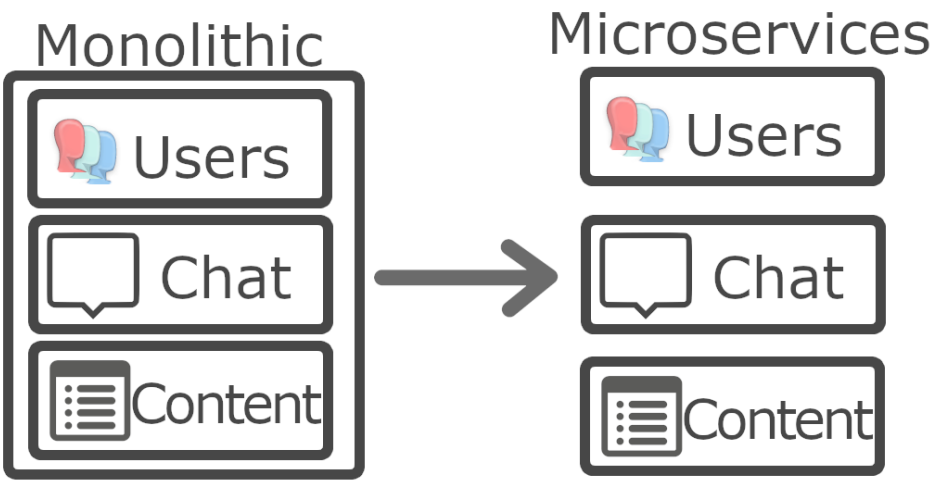

***Figure 1**, illustrates each service's independence in a microservice structure in contrast to a monolithic application structure where services are gathered in the same program.*

During development of a microservices product there is a need to ensure that components work before release in order to avoid issues. A common way to find deficiencies in a product is to test the different parts of it. For example, in a Kubernetes cluster it would be useful to see if pod X is up and running. One way to see if pod X is running is to manually test it. In large scale projects this is not efficient since tests for multiple pods might have to be performed. Instead test automation is a solution which could be implemented in order to ease workload **[4]**.

Microservices is a subject which has increased in popularity significantly within the past years among the software development and engineering industry. The trend on google search shows this quite clearly as more and more people are searching for information on the topic, as can be seen in Figure 2.

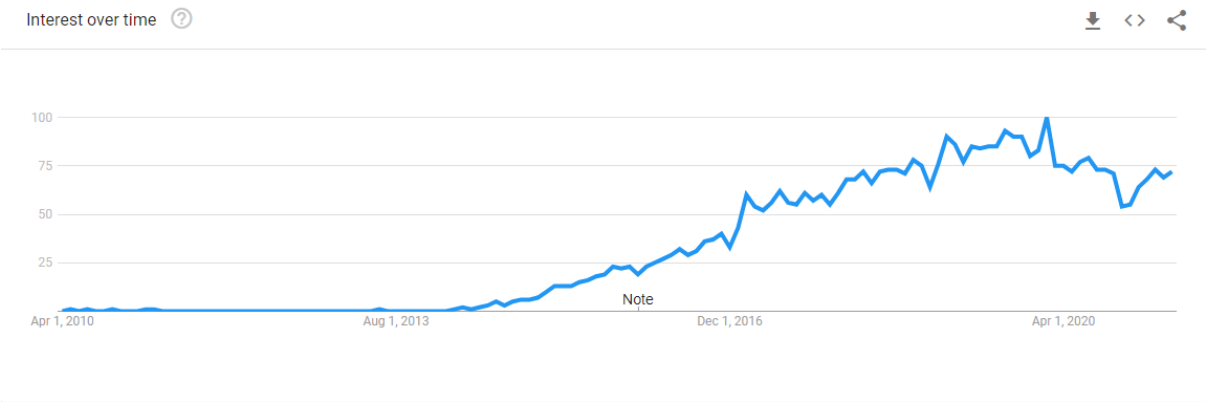

Figure 2, *Google trend chart for worldwide searches on "microservices"* ***[1]****.*

Automated tests cannot always provide useful information. The problem lies in how tests are used. One example of a test which could be useful in a production environment is the one mentioned earlier. What makes the test present bad data is for example if the cluster is in the starting state where the pod X has been started but is not running yet, but will run later. If the test claims there is an error with pod X a false positive has occurred.

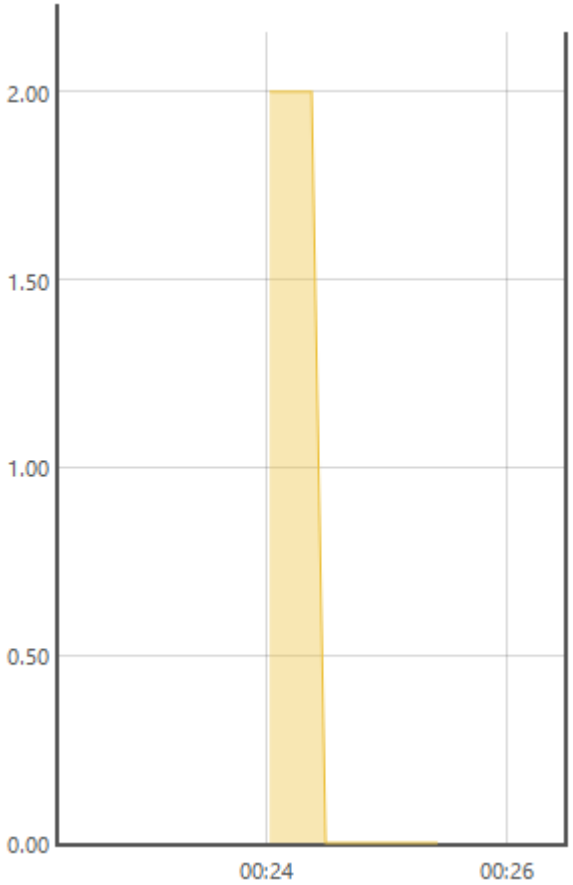

Figure 3*, An example of what a typical start of a cluster can look like. It starts with two assertions_failed which quickly turn into zero.*

This leads into another problem, what if during development there is no need to see if pod X is up and running since that is insignificant. The test still gives test data which complicates development since there is no need to see that data and it might obstruct significant test data to be seen.

The company this thesis work is being performed in collaboration with is Yolean, a company at the front of development when it comes to solutions for project and site management. They have previously worked with other big companies such as Volvo and NCC.

1.2. Purpose

The purpose of this project is to gain an improved understanding on how Kubernetes, microservices and automation testing interact with each other. Furthermore, to find improvements for the testing suite with which Yolean operates.

1.3. Goal

The goal for this project is to create a solution which will ease test-management and ensure that the tests will produce relevant data. This means that the end-product is supposed to have an effortless way of configuring the dev-loop currently present at Yolean to the developers' needs. Furthermore, automated tests in the project are also supposed to produce meaningful data, meaning results given by the solution should be relevant to the test. The main goals for this project is to reduce the amount of noise a testing environment produces and also allow for a more configurable environment.

1.4. Delimitation

No self-made clusters will be used, instead the project will focus testing on existing projects and adapt the test environment accordingly. The solutions will also be based on our local equipment.

2. Theory

To facilitate the reading experience of this report the following chapter will include explanations of several tools and components which are used in this project.

2.1. Kubernetes

Kubernetes is an open-source orchestration platform for managing containers in an automated manner **[5][6]**. It provides a convenient way to scale applications, to automatically deploy and update containers, and orchestrate complex container structures **[6]**. In Kubernetes, several Kubernetes objects are present which each have a unique role in the cluster and are described below. Knowledge about these objects is central in understanding the Kubernetes structure.

2.1.1 Usage of containers

Containers provide a way of running applications in an isolated environment, by packaging them together with everything needed to run them. **[7]** They are often compared to Virtual Machines, but provide additional benefits and are more lightweight; Running a Virtual Machine requires an entire Operating System to be booted, whereas containers run directly on top of the Operating System kernel. **[7]** This saves both time and resources. Running containers instead of normal applications is good for many different reasons. By isolating the application by running it as a container, outside factors such as other applications are eliminated. Thereby, providing a consistent and predictable environment. **[7]** When an application is containerized, it can be deployed on any Operating System, making it highly portable. **[7][8]**

2.1.2 Pod the smallest unit of Kubernetes

A pod is the smallest deployable unit of computing which you can create and manage in Kubernetes. The pod consists of one or a group of containers which has shared storage and network resources. It also has a specification on how to manage the different containers **[9]**.

2.1.3 Replica-set and its correlation to pods

The purpose of a replica-set is to ensure that a stable set of identical pods are running at any given time. Therefore it is often used to guarantee the availability of a specified number of identical pods. The replica-set is defined with fields, these fields include a selector which specifies how pods can be identified and acquired. A number which indicates the amount of replicas the given set should have. A pod template which describes the data new pods should create in order to meet the replica criteria **[10]**.

2.1.4 Services and their functions

A service in Kubernetes is an abstraction for a group of pods in a cluster. What a service does is to enable the group of pods to be assigned a name and a unique IP address **[11]**. A descriptive image of how a service works can be seen in Figure 4.

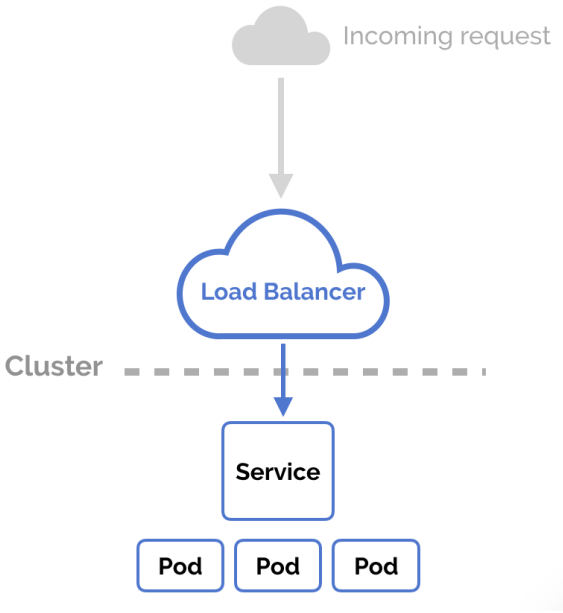

Figure 4; *Illustration which shows how a service acts as an interface between pods and network* **[12]**.

2.1.5 Deployment

Deployments are objects in Kubernetes which helps in the process of updating pods and replica-sets. Manually updating a resource to a new version can introduce a number of inconveniences such as long downtimes and human errors. A deployment automatically pushes updates to these containerized applications and enables roll backs if necessary **[13]**.

2.1.6 The use of namespaces

A namespace is a virtual sub-cluster which everything in Kubernetes is placed into, with the exception of nodes and persistent storage volumes which are shared between them **[14][15]**. Kubernetes exists with a default namespace which new components are placed into unless specified otherwise **[14]**. However, if more organization is needed the ability to create new namespaces exists. This can be useful when for example several teams are working on the same cluster, or there is a need to divide cluster resources between users **[15]**.

2.2. The Ystack tool

Ystack could be seen as the base for this project, and it is something called a micro-PaaS which is an abbreviation for Platform as a Service. PaaS works as a way for developers to avoid the complexities behind creating, configuring and managing servers. It can be seen as an abstraction on top of the underlying host, in this project that host will be the local setup of a cluster **[16][17]**. Ystack's goal as a PaaS is to allow developers to experiment with architecture of the cluster, make monitoring and alerts a first class tool in coding and support event-driven microservice patterns **[18]**.

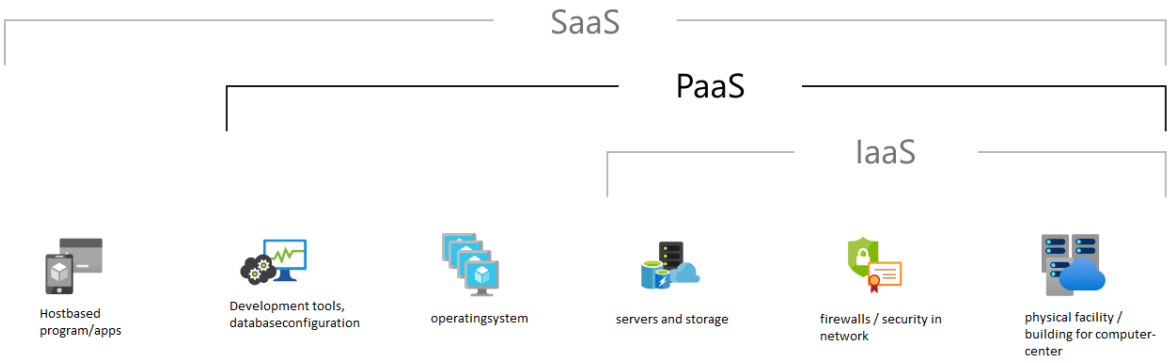

Figure 5; *An example of what PaaS covers and a comparison to Iaas and SaaS* ***[16].***

Ystack offers a lot of tooling where some are in the form of bash files in the bin folder of ystack. One of these tools commonly used in this project is y-skaffold for building images. The command *y-skaffold* is an extension of Skaffold's command which gives additional features which are developed by Yolean. These features include for example custom flags that can be set and used, change of local registry and ability to choose working namespace.

2.3. Testing

Since this project is strongly based on the use of testing and its correlation to Kubernetes this chapter is an important part in understanding the different tools used. Due to the amount of independent components in a cluster it is common to test the health of microservices before performing tests on actual functionality.

2.3.1. Jest

Jest is a javascript library which allows creating, running and structuring tests and it ships as a NPM package. Jest aims to be simplistic, fast, safe and does this by working without a lot of configuration and allows tests to run asynchronously **[19]**.

2.3.1.1. Jest reporter

A jest test reporter allows for developers to hook into different lifecycle methods for tests, such as onRunStart() and onRunResult() **[20][21]**. This means that once tests are run, relevant code can be executed at that time, or as results from tests come in. Jest normally has a default reported for managing events such as those mentioned above. For this project a manual reporter has been implemented since there has been a need for logic which is not handled by the default reporter.

2.3.2. Assertion testing

Assertion based testing can be defined as a boolean expression where at a specific point in a program this expression will be true unless an error or a bug has occurred. The use of assertion testing makes it possible to detect errors by writing an expression to validate the given value from a process. For example an expression could compare a status-code from a web server with “200” to see if it is responding. If the values are equal it returns true, if they differ it returns false **[22]**.

2.4. Docker

Docker is a platform for developing, shipping and running applications. It enables an abstraction of your application which separates it from your infrastructure. This makes the deliverment of software faster. You can also manage the infrastructure the same way as you manage your applications. The purpose of Docker is to reduce the delay between writing code and taking it into production **[23]**. This can be done using Docker via the tool integrated in Docker called docker-compose. With a single command services can be created and started from a configuration.

2.5. Skaffold

Skaffold is an open source CI/CD-project command line tool, providing easy and repeatable Kubernetes deployment. It is an environment flexible tool focused on providing rapid feedback to the developers to encourage developers to write code instead of dealing with administrative tasks **[24][25]**. It builds code into container images, automatically pushes these to the Docker repository, deploys them onto the Kubernetes cluster and validates the images with container-structure-tests **[25]**.

2.5.1. Skaffold dev-loop

To further facilitate the development process for developers, Skaffold has a dev-loop tool with additional features which can be run through Skaffold dev. When run, Skaffold will build images or artifacts as mentioned above, but will then also watch these files for changes. As changes occur, Skaffold will rebuild these artifacts and redeploy them to the Kubernetes cluster **[26]**. This provides an easy way for CD on local applications and enables rapid feedback as changes are deployed and put into use.

2.6. Working with K3D

K3D is a lightweight wrapper which is used to run K3S in Docker. This is useful for local developments of Kubernetes clusters since memory might be a limit **[27]**. K3S allows for a installation of Kubernetes which supposedly uses half the memory consumption. Due to its low memory consumption K3S allows Kubernetes to be run on machines with at least 512MB ram. The reason why it is significantly smaller is because K3S has removed several of the drivers found in the base distribution of Kubernetes, with the idea that the missing drivers can be replaced with add-ons **[28]**.

2.7. Working with Minikube

Similar to K3D, Minikube is a tool which makes it possible to run Kubernetes locally. Minikube is supposed to quickly create a local cluster and allow the user to interact with it. The VM used to run Minikube uses approximately 5GB of ram and supports different hypervisors such as VirtualBox which means it can be run on both linux and windows **[29][30]**.

2.8. Monitoring with Prometheus

Prometheus is a system for monitoring and collecting data from components in a Kubernetes cluster. It is also an alerting tool if anything in the cluster goes wrong. This tool was of great help during the project to visualize how the results from tests look like. To be able to collect data from an application, the application needs to set up a http-endpoint which Prometheus collects information from **[31]**. Using Prometheus to monitor tests in a cluster can look

something like in Figure 6 where a test suite has been run four times after each other. Tests are failing in the beginning and after a while they all pass. It is also possible to filter data in Prometheus as can be seen in the picture as well where a filter has been selected to only show “assertions_failed”.

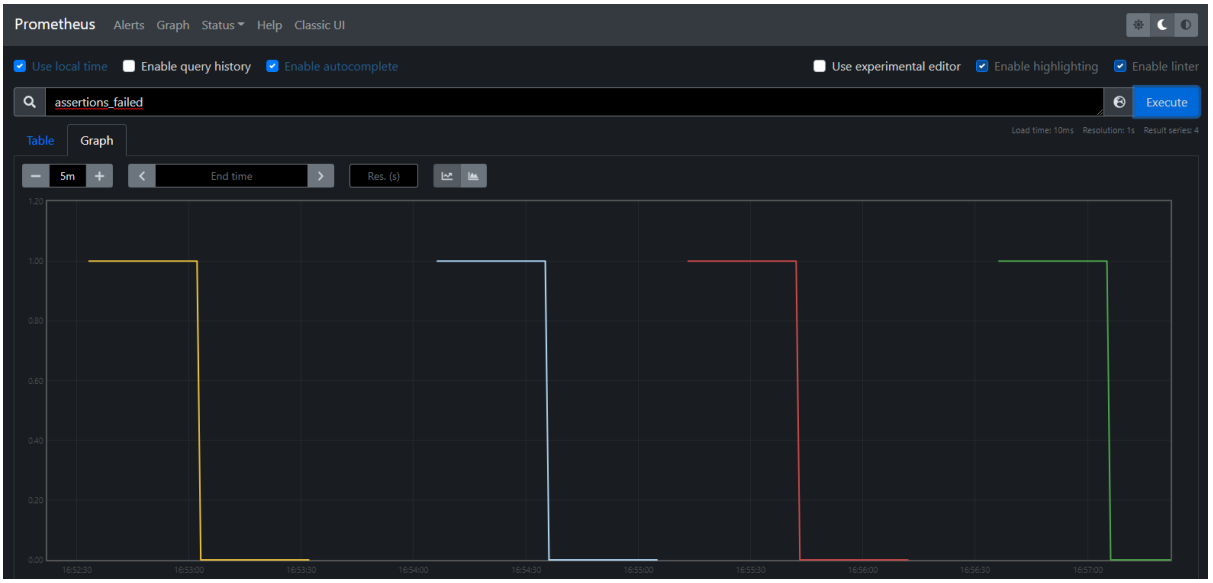

Figure 6; *Prometheus monitoring numbers of assertions_failed reported from tests.*

Prometheus consists of several components. The most important ones are the Prometheus server which scrapes and stores data in time tables as seen above, and the alertmanager which handles alerts **[31]**.

3. Method

A sprint-like working process with sprints of two weeks each was implemented and followed throughout the project. For regular feedback and communication, weekly meetings with a supervisor at Yolean were also decided upon. A timeplan with an assisting gantt diagram was created to give an overview of how long each part in the project would take. Finally the solutions were planned to be presented at the end to developers at Yolean to give user feedback and suggestions for further development.

The first sprints were dedicated to creating a Kubernetes cluster. When a cluster had been created, focus could be shifted on to the testing suite. A testing environment was set up with a tool called Jest. With a cluster up and running together with the option for creating and managing tests, research was started in order to find improvements. When something was assumed to be an improvement according to companies, sources or the background of this project, this was implemented and questions were asked to Yolean and their developers who responded with feedback. Based on the result of the questions the implementation was either kept, removed or improved.

Kubernetes was chosen as a tool since it had been worked with before and was therefore seen as an attractive tool due to the fact that most theory behind it was already known. The other tools were requested to be used by yolean considering they needed to be compatible with the infrastructure at yolean.

To find a solution to the given problems by yolean there was a need to look over the documentation for the tools used in the project. When a possible solution was found, it was implemented if it seemed attractive according to the prerequisites. The requirements were based off yolean’s idea on what their developers would benefit the most from, if it could be implemented.

4. System design

An important part of the thesis work was to deploy an infrastructure with containerized applications and tests running on these applications. The system was built on WSL using the tool docker and K3D. When the system was running the two main components ystack and Kubernetes-assert could be set up. Ystack is the cluster which tests will be performed on, and Kubernetes-assert is an image which handles the tests. Therefore changes on tests will be applied to Kubernetes-assert where the tool jest is implemented. The solutions relating to logical test functions are in the Kubernetes-assert cluster while the solution for passing data is in the ystack.

This chapter will go more into detail and explain how the infrastructure was built and what measures were taken in order to deploy it.

4.1. Operating System

Since Linux OS's are popular when working with Kubernetes clusters, it is worth mentioning that a Windows OS was used in the project. However, it was required to use the windows subsystem for linux as the main terminal or in combination with a windows terminal. This was due to several bash files which needed to be run. The solution has been tested on a Linux OS by Yolean and is confirmed to run as expected there, but no other operating systems have been tested.

4.2. Kubernetes cluster

Several options were available when deciding how to create a Kubernetes cluster. The most common choices include setting up a cluster with the help of a cloud platform like AWS or Azure, but there were also local options like Minikube and K3D. The cloud options are beneficial as they use remote resources instead of personal hardware resources and have built in tools for cluster management. However, for the small lightweight clusters used in this project, using local resources would be feasible, and for non-complex clusters assisting tooling optional. Therefore, a decision was made to deploy the cluster locally.

4.2.1. Setup

The main component for a setup of a local cluster is the tool which allows for a cluster to be created. In the project two different approaches were performed with the tools K3D and Minikube. Both tools had a way of creating a cluster where Minikube was running on a virtual machine and K3D was depending on Docker. The problem was that due to other parts of the project there was a need for the tools to interact well with Docker. Minikube did not work well with Docker in this scenario and caused problems. Therefore K3D was the choice in this project.

Docker for windows worked on startup after a few changes had been made to the configuration. Since Docker was running on windows it was important that the setting for integration with windows subsystem for linux was on.

When the Docker engine and K3D had been properly setup and was running, K3D allowed for a cluster to be created by inputting; $k3d\ cluster\ create$ [$name$] into the terminal. An example of what a cluster can look like after it has been set up with K3D can be seen in the Figure below.

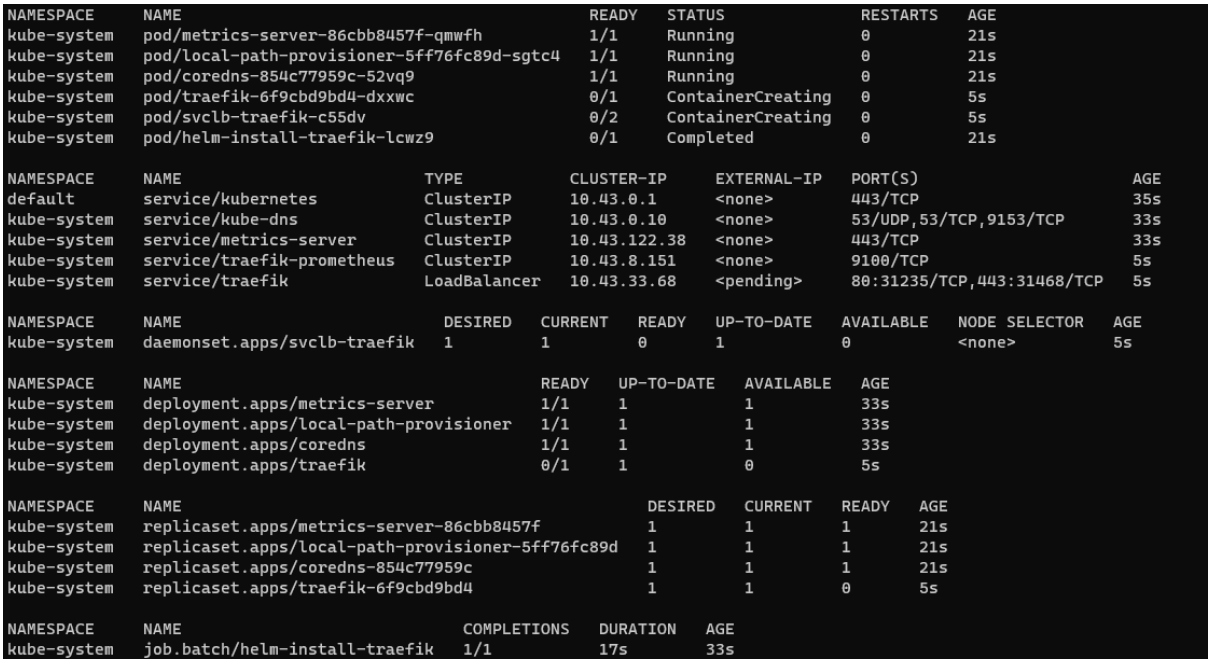

Figure 7; *An example of what a basic K3D cluster can look like.*

4.2.2. Ystack

Using a combination of Docker and Skaffold allowed for the Ystack packet to be built and shipped into the cluster as a container. Normally this would have been done with a command called $docker - compose\,[file]\,up$ but this task has been automated and instead the command $y - cluster - provision - k3s - docker$ was run. The script's task is to ease setup of the application where it runs multiple predefined docker-compose in one line. It also has a configuration for the application which removes the repetition of writing code whenever it is set up. An example of what the Ystack cluster could look like, can be seen in Figure 8. Note the namespaces that are called either Ystack or ystack-specs. It is under those namespaces the testing will be applied to. Ystack is set up in order for tests to be run since the cluster has an already configured testing suite and allows for direct research on Yolean's testing environment.

```
NAMESPACE     NAME                                       READY   STATUS        RESTARTS   AGE
kube-system   pod/local-path-provisioner-7ff9579c6-d6lsl   1/1     Running       0          3m44s
kube-system   pod/metrics-server-7b4f8b595-876sz          1/1     Running       0          3m44s
kube-system   pod/coredns-66c464876b-mn8w5                1/1     Running       0          3m44s
kube-system   pod/helm-install-traefik-mr64t              0/1     Completed     0          3m45s
kube-system   pod/traefik-5dd496474-xxwlt                 1/1     Running       0          3m27s
ystack        pod/minio-0                                 1/1     Running       0          3m44s
ystack        pod/bucket-create-ystack-builds-nrhjq       0/1     Completed     0          2m23s
ystack        pod/monitoring-proxy-f75fcf58f-8vgp7        1/1     Running       0          2m23s
ystack        pod/registry-5fbd584954-fc89p               1/1     Running       0          2m23s
ystack        pod/buildkitd-0                             1/1     Running       0          2m23s
default       pod/prometheus-operator-79cd654746-v7cdm    1/1     Running       0          100s
monitoring    pod/kube-state-metrics-c4cf4574b-sd4ln      1/1     Running       0          95s
monitoring    pod/node-exporter-fkp4d                     1/1     Running       0          95s
monitoring    pod/alertmanager-main-0                     2/2     Running       0          94s
monitoring    pod/prometheus-now-0                        2/2     Running       1          94s
ystack-specs  pod/ystack-test-7f45bbc4c-mjnrw             1/1     Terminating   0          70s

NAMESPACE     NAME                            TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                       AGE
default       service/kubernetes              ClusterIP      10.43.0.1       <none>        443/TCP                       3m58s
kube-system   service/kube-dns                ClusterIP      10.43.0.10      <none>        53/UDP,53/TCP,9153/TCP        3m56s
kube-system   service/metrics-server          ClusterIP      10.43.183.87    <none>        443/TCP                       3m56s
ystack        service/blobs-minio             ClusterIP      10.43.243.40    <none>        80/TCP                        3m44s
ystack        service/minio-hl-svc            ClusterIP      None            <none>        9000/TCP                      3m44s
kube-system   service/traefik-prometheus      ClusterIP      10.43.169.249   <none>        9100/TCP                      3m27s
kube-system   service/traefik                 LoadBalancer   10.43.194.227   <pending>     80:31715/TCP,443:31714/TCP    3m27s
ystack        service/builds-registry         NodePort       10.43.0.50      <none>        80:31710/TCP                  2m23s
ystack        service/prod-registry           ClusterIP      10.43.0.51      <none>        80/TCP                        2m23s
ystack        service/monitoring-nodeport     NodePort       10.43.225.134   <none>        9090:31718/TCP,9093:31712/TCP 2m23s
ystack        service/monitoring              ClusterIP      10.43.114.10    <none>        9090/TCP,9093/TCP             2m23s
ystack        service/buildkitd-nodeport      NodePort       10.43.132.184   <none>        8547:31713/TCP                2m23s
ystack        service/buildkitd               ClusterIP      10.43.21.184    <none>        8547/TCP                      2m23s
default       service/prometheus-operator     ClusterIP      None            <none>        8080/TCP                      101s
kube-system   service/kubelet                 ClusterIP      None            <none>        10250/TCP,10255/TCP,4194/TCP  96s
monitoring    service/alertmanager-main       ClusterIP      10.43.189.236   <none>        9093/TCP                      95s
monitoring    service/prometheus-now          ClusterIP      10.43.131.4     <none>        9090/TCP                      95s
monitoring    service/alertmanager-operated   ClusterIP      None            <none>        9093/TCP,9094/TCP,9094/UDP    94s
monitoring    service/prometheus-operated     ClusterIP      None            <none>        9090/TCP                      94s

NAMESPACE    NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
monitoring   daemonset.apps/node-exporter   1         1         1       1            1           kubernetes.io/os=linux   95s

NAMESPACE     NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   deployment.apps/local-path-provisioner   1/1     1            1           3m56s
kube-system   deployment.apps/metrics-server           1/1     1            1           3m56s
kube-system   deployment.apps/coredns                  1/1     1            1           3m56s
kube-system   deployment.apps/traefik                  1/1     1            1           3m27s
ystack        deployment.apps/monitoring-proxy         1/1     1            1           2m23s
ystack        deployment.apps/registry                 1/1     1            1           2m23s
default       deployment.apps/prometheus-operator      1/1     1            1           101s
monitoring    deployment.apps/kube-state-metrics       1/1     1            1           95s
ystack-specs  deployment.apps/ystack-test              0/0     0            0           70s

NAMESPACE     NAME                                              DESIRED   CURRENT   READY   AGE
kube-system   replicaset.apps/local-path-provisioner-7ff9579c6   1         1         1       3m45s
kube-system   replicaset.apps/metrics-server-7b4f8b595          1         1         1       3m45s
kube-system   replicaset.apps/coredns-66c464876b                1         1         1       3m45s
kube-system   replicaset.apps/traefik-5dd496474                 1         1         1       3m27s
ystack        replicaset.apps/monitoring-proxy-f75fcf58f        1         1         1       2m23s
ystack        replicaset.apps/registry-5fbd584954               1         1         1       2m23s
default       replicaset.apps/prometheus-operator-79cd654746    1         1         1       101s
monitoring    replicaset.apps/kube-state-metrics-c4cf4574b      1         1         1       95s
ystack-specs  replicaset.apps/ystack-test-7f45bbc4c            0         0         0       70s

NAMESPACE    NAME                                READY   AGE
ystack       statefulset.apps/minio              1/1     3m44s
ystack       statefulset.apps/buildkitd          1/1     2m23s
monitoring   statefulset.apps/alertmanager-main  1/1     94s
monitoring   statefulset.apps/prometheus-now     1/1     94s

NAMESPACE     NAME                                  COMPLETIONS   DURATION   AGE
kube-system   job.batch/helm-install-traefik        1/1           19s        3m56s
ystack        job.batch/bucket-create-ystack-builds 1/1           6s         2m23s
```

Figure 8; *an example of what a ystack cluster can look like.*

4.2.3. Kubernetes-assert

Kubernetes-assert is a package that Yolean provides for testing in clusters. This setup uses jest together with configuration files as well as Docker and Skaffold to define the behaviour of each test run. In particular the jest reporter file in the runtime-nodejs folder is what determines how tests are being executed and what behaviour the program should have. This image is built into the cluster by locating the folder and inserting *y-skaffold build* into the terminal. A rebuild of the image is required after modifications have been made in order to include them into the cluster. Originally this file comes with reruns of tests with an interval, but was later configured to have several different behavioural configuration options which are discussed more closely in the following section.

4.3. System improvements

Throughout the project improvements to the system Yolean provides were performed to improve the developer experience and reduce noise generation from tests. These solutions are presented in this section.

4.3.1. Configuring reruns

A couple of options were available when configuring how the rerun of tests work within Yolean's package. Firstly, Skaffold offers a solution for a trigger which gives the user an option to only run tests when the related test file is updated. This is done through setting the flagg *trigger* to *polling* when running y-skaffold as such: *y-skaffold dev --trigger=polling*. Secondly, the interval between reruns could be manually set by the developers by setting RERUN_TIME: [number] in an env.json file. Both of these solutions give more freedom to configure the reruns of tests within the system for developers.

4.3.2. Delaying tests

Since all the tests are asynchronous, the solution for adding a delay was a bit more complicated than a normal if-statement. Tests were executed the first time no matter the rerun-time since they were called asynchronously on start. In order to reduce or remove the noise these first-time tests created, a variable for controlling when tests would be allowed to produce data was created. As seen in Figure 9 below, when the testing environment was created a function call with a timeout of *delay* seconds was called. This means that after *delay* seconds, the allowMod() function will be called and tests can now generate data. This delay was decided by a variable in an env.json file which sets the delay time.

```
let allow_modification = false;

// In the tracker class
if(allow_modification) {
    if (delta > 0) assertions_failed.inc(delta);
    if (delta < 0) assertions_failed.dec(-delta);
}

allowMod() {
    allow_modification = true;
    console.log("Allow modifications: ", allow_modification);
}

// In the constructor
setTimeout(() => {
    tracker.allowMod();
}, delay);

```

Figure 9; *The logical code for the delay function.*

In order to decide for a delay-variable there was a need to understand how long it took for a computer to properly set up the cluster. To retrieve an appropriate variable it was therefore decided that it will be based on the time it took for the computers used in this project to reach zero assertions failed. This test was done with two different computers and performed five times each to get a good estimate.

4.3.3. Exit tests condition

Having tests run continuously without ever stopping generates heaps of perhaps unintended meaningless data. By having an exit condition, the system can cancel the tests at an appropriate time and keep the reported data relevant. The first solution for this was to run tests until there were no more failed tests and then the system would stop and wait until a test had failed again to start reporting data. A problem with this approach was that all tests would rerun if one fails, creating unnecessary data. Furthermore, to know if a test fails again it would have to keep running the tests in the background and use resources.

Therefore, a second approach was attempted where tests have to pass three consecutive times before exiting. This intends to remove tests which have passed three times, therefore solving the issue where all tests need to rerun for the sake of a failing one. Furthermore, exiting the tests completely when the condition is met means that no more resources would be spent on monitoring tests in the background. This approach also makes sure that services being passed by tests are stable as they have to pass three consecutive times. A solution to a common problem when developing microservices. A more harsh condition with a higher number or consecutive passes could be set to ensure this further, but would cost more resources and time. Therefore three passes were decided to be a good middle ground. The solution builds upon a counter for each path, or test, which increases if the test did not fail and is set to zero again if ít does as can seen in the code below.

```
this._pathCounter = {};
this._prevFailTests = {};

// Save the current amount of failed tests for the path
this._pathsSeen[path].numFailingTests = numFailingTests;

// Compare current failed tests with previous amount of failed tests
if (this._prevFailTests[path] == this._pathsSeen[path].numFailingTests) {
    // Failed tests for path has not increased
    // Increase consecutively passed tests.
    this._pathCounter[path]++;
} else {
    this._pathCounter[path] = 0;
}

// Save failed tests for next run
this._prevFailTests[path] = this._pathsSeen[path].numFailingTests

// If tests has passed 3 or more times, stop the test from running
if(this._pathCounter[path] >= 3) {
    console.log("Test has passed more than 3 times:", path);
    // Remove test from reruns
}
```

Figure 10; *The logical code for the exit function.*

4.3.4. Timeout for tests

When running tests on a build overnight there is a risk of an enormous amount of data being produced by these tests if left running. This is a problem as it can fill databases with noise and cost both resources and time. The solution above partly attempts to solve this issue by stopping tests when they have passed three consecutive times. However, in many cases with large test suites the vast majority of the tests passes, but one or two tests persists on failing due to perhaps recent changes in just that part of the code. This would mean that all the tests will keep running overnight even with the solution presented above. This timeout approach for tests attempts to solve this issue by introducing a specified timeout allowing tests to run only during that time and then exiting.

Similar to the solution with a delay, a variable could be passed in the env.json which sets a timeout variable. The variable's function was to keep track on how long tests would be allowed to produce data before terminating. This was done through calling a method with a delay of *timeout* seconds which after the timeout, told the user the tests were stopped and exited. An example of what this code could look like can be seen in the Figure 11 below.

```
this._doRerun = true;

timeoutReached() {
    console.log("Stopping reruns, timeout has been reached.");
    console.log("Will not rerun until file modification");
    this._doRerun = false;
    // EXIT IMAGE OF DOCKER ASWELL?
}

if(this._timeout != 0) {
    setTimeout(() => {
        this.timeoutReached();
    }, this._timeout);
}
```

Figure 11; *The logical code for the timeout function.*

Different from the delay variable, a timeout variable will depend more on how long the user believes it is adequate to run tests. For example if a user wants to run tests after work hours a timeout can be set to 3 hours instead of filling the database with error logs for the entire duration the user is away.

5. Results

During the development of the product multiple results have been achieved. The results can be divided into two categories; results which were given during development and results received from Yolean's user experience. This chapter will go further into depth of each category.

5.1. Overview of results

The purpose of the thesis was to encourage the use of automated tests for microservices by facilitating the development experience and reducing the amount of noise data generated from tests. Several independent, but combinable, solutions were developed and presented to experienced developers at Yolean. These developers gave positive feedback about the solutions and can be considered to have fulfilled the purpose of the thesis.

A solution to decrease the amount of noise and focusing further on relevant data has been achieved. Instead of rerunning tests continuously without a basis on whether the test has failed or not, tests will run until they pass and then stop.

There were three solutions for which the results will be presented below. The first solution with a delay timer for the tests will be introduced together with the time measurement data which the delay was based on. Then comes the results for the exit condition solution. Following that, the results from the timeout solution will be presented and finally user experience results will be presented.

5.1. Delay data

The default delay for this solution was based on measurements by two separate computers for how long it took for all tests to pass during a normal test run. This measurement was made five times with each computer to get a good estimated value of the average time. The computers in table 1 were used in order to retrieve the test results in table 2. For Computer 1 there was a slight variation in the tests which ranged from 27-30 seconds, while Computer 2 seemed to approach 30 seconds. With the delay approach there was therefore no need to have a delay less than 30 seconds in this scenario since during that time, tests will most likely still fail.

Computer ID	CPU	RAM	GPU	DISK W/R
Computer 1	3,6 GHz	16 GB 2133 Mhz	RTX 2070	Write: 445 MB/s Read: 370 MB/s
Computer 2	3,5 GHz	12 GB 1600 Mhz	GTX 1080	Write: 469 MB/s Read: 515 MB/s

Table 1; *Computer specifications.*

Computer ID	Test 1	Test 2	Test 3	Test 4	Test 5
Computer 1	27s	29s	30s	29s	27s
Computer 2	29s	30s	29s	30s	29s

Table 2; *The time it takes for each computer to have all the tests pass on the ystack-cluster.*

As seen in Figure 12 and 13 there is a way to include a delay which avoids insignificant results to be displayed at the start of a cluster. In the figures it can be seen that without a delay, it is immediately reported that errors occur. With a delay less than 30 seconds, some of the errors are ignored, until the point where the delay stops, then the errors rise again. This can be seen in figure 13 where tests start at 0 and then rise after the delay. When the delay was larger than 30 seconds, no errors were reported.

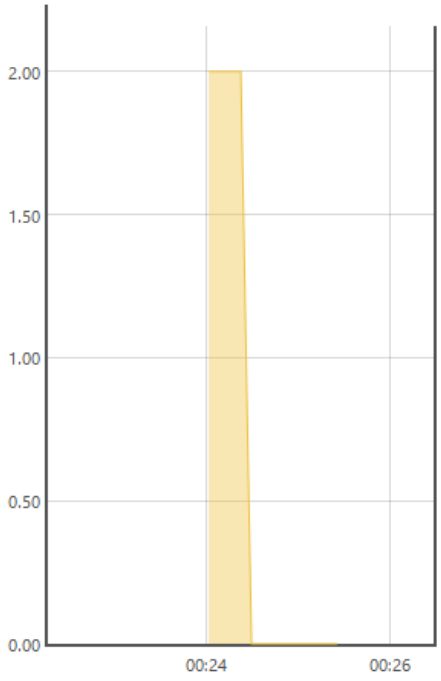

Figure 12, *Failed tests without a delay.*

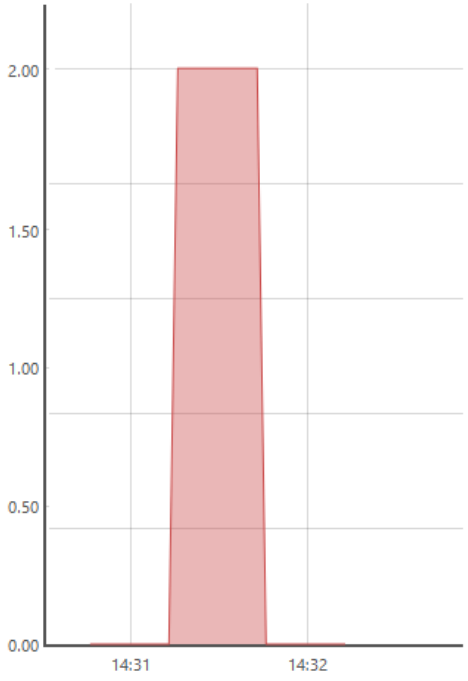

Figure 13*, Failed tests with a delay less than 30.*

5.2. Exit test condition

In order to remove tests which have passed more than three times in a row an exit condition has been made. As can be seen in Figure 14 tests react differently on reruns after the condition has been met. When the tests have passed three or more times they will not be run on reruns and therefore reduce the output noise.

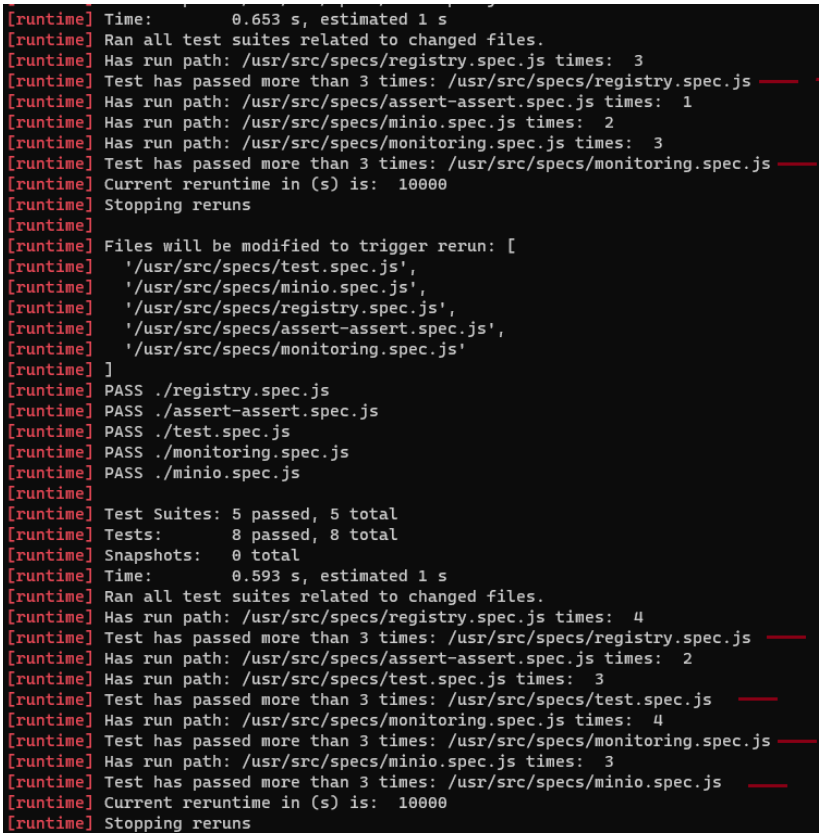

Figure 14; *Terminal output of when tests pass more than 3 times. It can be seen that before the rerun only two have passed more than 3 times, after the rerun four have passed more than 3 times.*

5.3. Timeout

The test timeout results in less database cluttering as it prevents unintended long tests to be run overnight or over a weekend. It makes a more configurable test run for developers. After the timeout reaches zero it will tell the developer that all tests will be stopped as can be seen in the figure below.

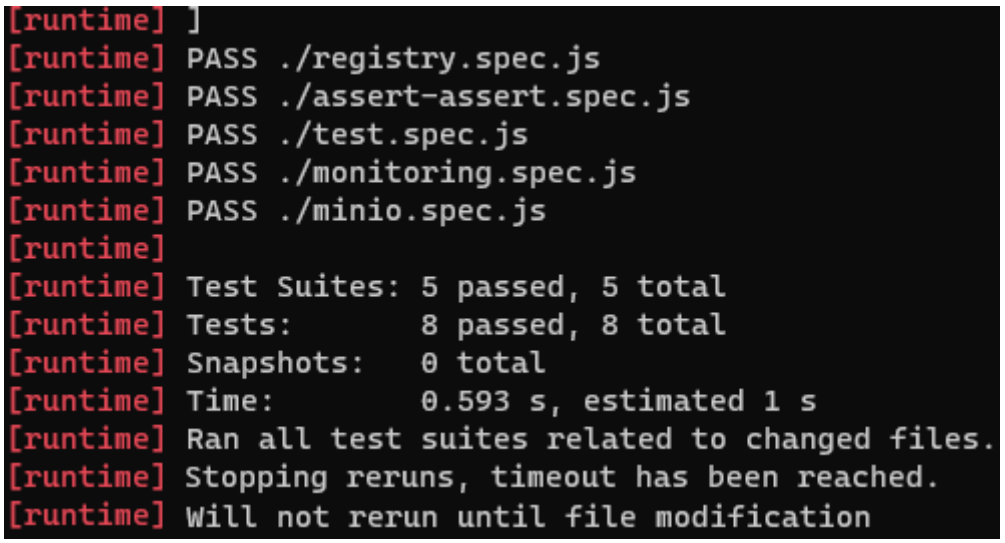

Figure 15; *Terminal output when timeout reaches zero.*

5.4. User experience

Before any iteration of user experience, the first problem to solve was a configurable rerun time. When this had been implemented users told us that they were unable to make the solution work, but an examination of the logical code showed promise that the code seemingly should work. After a second iteration users were able to use the solution. Feedback given was that the implementation helped users a lot since they were able to change reruns and runs were stopped after tests passed. Suggested improvement was to create a user friendly interface which showed configuration decisions via terminal feedback. For the next iteration users commented the following:

1. *Focus on a CI/CD environment where we want the asserted backends to keep running.*
2. *The specs should exit after X consecutive passes of all specs.*
3. *An initial wait of Y seconds before starting specs could be useful to reduce test noise.*
4. *A configurable interval between reruns would also be useful to reduce noise.*

Third iteration focused on point 2 and 3 where the delay solution was implemented in order to resolve issue 3 and the exit condition to resolve issue 2. Feedback given to iteration three can be seen in the following list:

1. *It's a clear improvement that specs stop running upon success.*
    a. *Specs that produce test data no longer fill our database.*
    b. *Logs get easier to follow.*
2. *Actually it's also common that both dev and CI/CD results in failed specs that keep running, at least until the next working day. There is no improvement for that scenario.*
3. *We tricked ourselves, based on ystack example specs, specs and app should have separate skaffold yamls.*
    a. *But they should have the same so that developers:*
        i. *Get one stream of logs to follow.*

    ii. *Don't need two terminals.*
    iii. *Are always encouraged to be test driven during skaffold dev.*

Point 1 of the user experience feedback was positive and stated that their database no longer gets filled with test data and logs are easier to follow. Therefore this solution is seen as a noticeable improvement. Users still believe that noise has not been reduced enough when it comes to longer runs of tests since they may run overnight. The fourth iteration focused on Point 2 to solve this issue with the timeout solution. Feedback given on iteration four and general comments about further development can be seen below:

1. *An initial wait of Y seconds before starting specs could be useful to reduce test noise*
    a. *Relatively low priority.*
2. *A configurable interval between reruns would also be useful to reduce noise*
    a. *Relatively low priority. 10 s is a very good default.*
3. *But the specs should exit 0 after X consecutive passes of all specs*
    a. *... it's also common that both dev and CI/CD results in failed specs that keep running*
    b. *Can we find criterias and runtime changes so that spec containers always exit?*
    c. *Should lead to a Completed status in kubectl get pods.*
4. *Evaluate a max time for the specs container, after which it exits non-zero if there are still test failures*
    a. *Can this be combined with restart policy so that the pod doesn't crash loop but instead we get an Error status.*
    b. *How do we prevent exit when someone is actually developing?*
        i. *Max time countdown should be reset on actual file watch change*
        ii. *We can define "actually developing" as editing the specs at least once within the max time interval.*
5. *We tricked ourselves, based on ystack example specs that specs and app should have separate skaffold yamls*
6. *The examples we run should dev both backend and specs in the same skaffold dev command.*
7. *If we succeed with the above, the y-assert hack in ystack could be replaced with something that basically does kubectl get pods with some selector on specs' labels and checks the pod status.*

The fifth iteration focused on point 1 and 3 and are addressed in the delay solution and exit condition solution.

6. Discussion

This chapter will discuss the results in the report and if our question at issue could be answered. The ethics behind developing such a product and its use in society will also be discussed together with the sustainability. The last thing this chapter will cover is the improvements we believe can be done to this product.

6.1. Improving the development loop

At first we believed that Skaffold allowed for parameters to change the rerun-time. The issue was that if parameters were passed to an already built image, there were no found tools for passing the parameters to be used in the image. Hence the reason to have a json file with different variables that could be changed, due to yolean using pre-built images in their cluster. The first iteration of this solution was believed to be too time consuming. After realising that the variables could be changed easily by manually adjusting the json file or using a script, the users stated this was a clear improvement to running the tests.

The reason why having a json file worked is because of the way the testing image is configured. When *skaffold dev* is executed, the image tag for Kubernetes-assert is executed in the build command. This meant that the test image could be retrieved from anywhere since there was no need to rebuild it and therefore less time-consuming. Unfortunately as stated above, configuration changes of the image were not possible unless you wanted to rebuild it. The thing is that since the image is already built, it will copy all the tests from the working directory and run all of them. That is where we had the idea to pass a file with configuration information in it. What this meant is that the config file, triggered whenever it was updated, was copied into the working directory of the image. Therefore we could use the information in the json and since the file was not very large, we thought this was an appropriate solution.

6.2. Reducing the noise

Reducing the amount of insignificant test data had many different solutions. For example the delay method would almost completely remove the amount of false positive errors at the start of a cluster. The problem with this method is that for a large cluster, waiting for 60 seconds or more is a reasonable time to not look at the error-logs. Although it was seen as an improvement to have this function, the only real issue it solved was to avoid filling the database with irrelevant logs.

In chapter 4.3.3 Exit test condition, Delaying tests, above it is mentioned that tests are exited after three consecutive passes and the reasoning behind how more could be cost and time consuming while less could not ensure that the pod is functioning properly. To further elaborate on this; normally a pod or a component of a Kubernetes cluster goes up for 5-10 seconds before failing and exiting. Another common issue is that a pod is stuck in restarting or startup and therefore is unable to handle whatever functionality it should.

The first scenario will be caught with the tests since if a rerun of the tests is set to 10 or more seconds 5-10 of errors will be caught in the second rerun since they will already have failed. The other scenario was handled with our timeout solution where if tests did not pass for longer than an amount of hours, they would stop.

6.3. User experience

We believe that a good part about the user results was the regular communication with Yolean since a good amount of feedback from the developers behind the product was received this way. This made it easier to know if the solutions were on the right track or not and get suggestions for other improvements.

A small issue with the communication that was noticed late in the project was the fact that our supervisor was the only one presenting their experiences of our progress. Effectively meaning that communication to developers occurred through a middleman. This made it hard to know exactly what the developers said about the solutions and there was no option to ask questions regarding these opinions as well.

It can be noted that there were several points in the feedback which were not addressed. The amount of suggestions for further improvement from the developing team was much appreciated, but both time, resources and priorities made it hard to develop all these suggestions.

6.4. Deciding between K3D and Minikube

One of the first problems we encountered was when we tried to create our local cluster. Normally at a company you use a cloud provider such as AWS or Azure but for our development we needed to work on our own machines. To do that, a virtual machine such as K3D or Minikube can be used in order to run the cluster. Since we had been working with Minikube before we decided to try that first. The problem was that Minikube was able to create a cluster, but in our project Docker was needed as well. Minikube did not properly integrate Docker which gave us errors which had about zero documentation. Instead we were recommended to use K3D which integrated Docker better.

6.5. Ethics

The maybe most obvious ethical problem with a solution like this or any similar solution which involves automation of a task is that people might lose their jobs. If this means that companies will no longer need as many application testers that might mean people will lose their jobs or that less jobs will exist in that area. Due to the obviousness of this ethical problem it will not be gone into more detail **[32]**.

Another issue with automated tests is that although it might be cheaper and less time-consuming there is no way to tell if they always provide good performance. The tests might be automated with the idea that they will return errors if something goes wrong, but how can the bugs be known before they have been found. Manual testing will maybe cost more, but instead might provide better results since the manual tester can with enough time find unknown bugs **[33]**.

The question is; if testing for applications becomes automated and thus lowering the quality, is that the right thing to do in order to save money? In this case we believe that automation can be useful since developers will not be able to put all that time into testing their application during development. Before release it could be good if some manual testing was implemented as well in order to find the bugs not found during production.

In one of our solutions we have decided that after three consecutive passes the tests will assume the affected component will not fail and therefore tests will not be rerun. The issue with this solution is that although for some applications this may be an appropriate buffer to ensure that nothing will fail. The ethical problem we are asking ourselves is; what if something fails and who is responsible in cases where failures are dangerous. An example of such a case is if tests are run on an airplane to see if it is properly working, the test results claim it is safe to fly on the airplane but during flight the airplane breaks down and crashes. How much can we rely on the test data and who is responsible if we assume incorrectly.

6.6. Sustainability

As with many technological products, our solution to the problem at issue might be replaced tomorrow if a better solution can be found. What makes our product unique and sustainable is the fact that on the market, we are among the firsts to develop such solutions and hopefully future improvements by yolean or other companies might be based on what we have done. Skaffold is a tool still in development which means that things in our solution might become deprecated soon, but for Yolean it seems like we have achieved something they will use in the future until better solutions come. Since this solution is almost handcrafted for Yolean, even though the theory can be applied to almost any solution, it might take a while for the solution to become deprecated due to the special set up and tooling.

6.7. Further improvements

Currently the user needs to manually change the json file used in order to pass variables to the testing suite. Therefore one of the most obvious improvements would be to automate this process as well with the use of flags or something similar to increase time efficiency.

The delay variable in this project was calculated based on running the tests five times in a row on two different machines, taking the time values and calculating what we assumed was an appropriate time to wait before running tests. For a future project this could also be automated by running the tests once, and based on the time calculate an appropriate delay. Considering how for most projects the acceptable time before a cluster is expected to work can vary a lot there should be enough buffer in order to automate this process.

7. Conclusion

In conclusion a lot has been learnt through this project and the members of this project have gained greater insight in how companies work with microservices. This includes everything from the exciting tools Yolean is working with to the suggestions for improvement from developers which shows how much more can be worked on.

Given the results from user experiences it can be stated that the two most promising solutions were the exit test condition and the timeout. The reasoning behind this is that according to the users these solutions improved the testing suite significantly and the amount of noise was reduced. Since one of the goals of this report was to produce more relevant test data this is therefore seen as a path towards that goal.

A config of reruns was created in order for users to be able to change the interval they wanted tests to run in. This was one of the first things user experiences gave positive feedback to and was also the base to the timeout solution. Due to these facts this solution is one of the most important ones in the project since it opened up the possibility for further advancements. The usage of a json file containing variables was firstly implemented with the rerun solution and that is also a part in why this step was so important.

The delay solution has been concluded to be, although not seen as strongly as, an improvement to the reduction of noise. This is due to the fact that a delay function helps to reduce the amount of noise data in the start up of a cluster, but normally data for the first minute is overlooked because of the amount of failures which are seen as irrelevant. Therefore it helps reduce the amount of logs filling up the database but it does not improve the user experience as much as we hoped for.

As a final note, it can be concluded that this project has helped encourage test automation with microservices. It has also expanded our understanding on how microservices function and their use in the industry.

8. References

[1] "Microservices," *Google Trends*, [Online] Available: https://trends.google.com/trends/explore?date=2010-03-02%202021-05-21&q=microservices [Accessed May. 21, 2020]

[2] "What are Microservices?," *Amazon Web Services*, [Online] Available: https://aws.amazon.com/microservices/ [Accessed May. 21, 2020]

[3] "Microservice Architecture," *Microservices.io*, [Online] Available: https://microservices.io/ [Accessed May. 21, 2020]

[4] P. Vuollet, "What Is Test Automation? A Simple, Clear Introduction," *Testim*, Aug. 06, 2019. [Online] Available: https://www.testim.io/blog/what-is-test-automation/ [Accessed Apr. 07, 2020]

[5] "What is Kubernetes?," *Microsoft Azure*, [Online] Available: https://azure.microsoft.com/en-us/topic/what-is-kubernetes/ [Accessed May. 10, 2020]

[6] "What is Kubernetes?," *Red Hat*, [Online] Available: https://www.redhat.com/en/topics/containers/what-is-kubernetes [Accessed May. 10, 2020]

[7] "Containers at Google," *Google Cloud*, [Online] Available: https://cloud.google.com/containers/ [Accessed May. 10, 2020]

[8] R. Gibb, "What are Containers?," *Stackpath*, May. 30, 2019. https://blog.stackpath.com/containers/ [Accessed May. 10, 2020]

[9] "Pods," *Kubernetes*, [Online] Available: https://kubernetes.io/docs/concepts/workloads/pods/ [Accessed Apr. 19, 2020]

[10] "Replicaset," *Kubernetes*, [Online] Available: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/ [Accessed Apr. 19, 2020]

[11] "Kubernetes Services," Vmware, [Online] Available: https://www.vmware.com/topics/glossary/content/kubernetes-services [Accessed Apr. 19, 2020]

[12] M. Palmer, "Kubernetes Ingress with Nginx Example," *Matthew Palmer*, [Online], Available: https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html [Accessed May. 13, 2020]

[13] “What is a Kubernetes deployment?,” *Red Hat*, [Online] Available: https://www.redhat.com/en/topics/containers/what-is-kubernetes-deployment [Accessed May. 10, 2020]

[14] “What is a Kubernetes Namespace?,” *Vmware*, [Online] Available: https://www.vmware.com/topics/glossary/content/kubernetes-namespace [Accessed May. 15, 2020]

[15] “Namespaces,” *Kubernetes*, [Online] Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ [Accessed May. 15, 2020]

[16] “Vad är PaaS?,” *Microsoft Azure*, [Online] Available: https://azure.microsoft.com/sv-se/overview/what-is-paas/ [Accessed May. 16, 2020]

[17] S. Domino, “What is micro-PaaS and why it’s the future of app development,” *Hackernoon*, Feb. 21, 2017. [Online] Available: https://hackernoon.com/what-is-micro-paas-and-why-its-the-future-of-app-development-3aa30d086703 [Accessed May. 16, 2020]

[18] “Ystack,” *Yolean*, [Online] Available: https://github.com/Yolean/ystack [Accessed May. 16, 2020]

[19] *Jest*, [Online] Available: https://jestjs.io/ [Accessed Apr. 29, 2020]

[20] C. Wren, “Writing a Jest Test Reporter,” *Plain English*, Mar. 16, 2019. [Online] Available: [https://javascript.plainenglish.io/writing-a-jest-test-reporter-cb7c123ec211 [Accessed 2020-05-16]

[21] “Configuring Jest,” *Jest*, [Online] Available: https://jestjs.io/docs/configuration [Accessed May. 16, 2020]

[22] “Assertion Testing,” *Tutorialspoint*, [Online] Available: https://www.tutorialspoint.com/software_testing_dictionary/assertion_testing.htm [Accessed Apr. 12, 2020]

[23] “Docker overview,” *Docker*, [Online] Available: https://docs.docker.com/get-started/overview/ [Accessed Apr. 29, 2020]

[24] “Skaffold Documentation,” *Skaffold*, [Online] Available: https://skaffold.dev/docs/ [Accessed Apr. 28, 2020]

[25] A. Egorov, “Intro to Skaffold for easy Kubernetes development,” *Medium*, Nov. 20, 2019. [Online] Available: https://medium.com/flant-com/skaffold-kubernetes-development-tool-2897d6903e02 [Accessed Apr. 28, 2020]

[26] “skaffold dev,” *Skaffold*, [Online] Available: https://skaffold.dev/docs/workflows/dev/ [Accessed Apr. 30, 2020]

[27] “Overview,” *K3D*, [Online] Available: https://k3d.io/ [Accessed Apr. 17, 2020]

[28] A. Jeffries, “What’s the difference between k3s vs k8s,” *Civo*, Sep. 24, 2019. [Online] Available: https://www.civo.com/blog/k8s-vs-k3s [Accessed Apr. 17, 2020]

[29] “Install Tools,” *Kubernetes*, [Online] Available: https://kubernetes.io/docs/tasks/tools/ [Accessed May. 13, 2020]

[30] “Getting Started Using Minikube,” *Cilium*, [Online] Available: https://docs.cilium.io/en/v1.7/gettingstarted/Minikube/ [Accessed May. 13, 2020]

[31] “Overview,” *Prometheus*, [Online] Available: https://prometheus.io/docs/introduction/overview/ [Accessed May. 15, 2020]

[32] T. Mayor, “Ethics and automation: What to do when workers are displaced,” *MIT sloan*, Jul. 08, 2019. [Online] Available: https://mitsloan.mit.edu/ideas-made-to-matter/ethics-and-automation-what-to-do-when-workers-are-displaced [Accessed May. 17, 2020]

[33] A. Munir, “Man VS Machine in Software Testing,” *Linkedin*, Jun. 25, 2014. [Online] Available: https://www.linkedin.com/pulse/20140625070125-116595839-man-vs-machine-in-software-testing/ [Accessed May. 24, 2020]