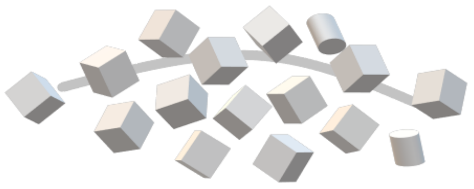
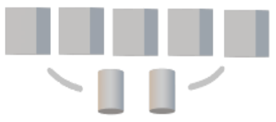


SOFTWARE TESTING

An analysis of agile best practices within cloud-based microservice architecture



Author: Carita Persson

EC Utbildning Malmö

Software Testing

Degree Project 20 p

Term: VT22

Abstract

Agile teams and cloud-based systems has become the norm of today´s organizational and system architecture. With more complex systems including cloud-based, microservices and container environment the testing architecture also becomes more complex.

Going from Waterfall and monolithic architecture there is a confidence that we still can use the same testing approach in this architecture.

Agile team’s independency gives us the ability to take faster decisions within the teams, decide preferable tools and work in parallel with the rest of the organization. Is the focus on faster deployment and faster performance making us down prioritize testing of the system?

With a raised usage of test automation, the discussion if test automation should be seen as a replacement of other testing approaches or if it should be seen as a tool has become a common topic in the development world.

The purpose of this thesis is an investigation of problem areas, best approaches, and test engineers’ opinions about the subject of agile teams working in cloud-based microservices systems.

The analysis is made by reaching out to experienced test engineers who have or are working as test leads in the mentioned team and system environment. The discussions were made mostly in written conversations online, with a semi-structured interview approach.

Alongside with the discussions I also researched written material about agile teams, microservices, different test methods, techniques, and levels. I did some experimental research by testing software and test techniques. Here I also analyzed PaaS, SaaS, and monitoring tools to gain a deeper knowledge.

The research shows a somewhat unexpected analysis result that leads the thesis into the theme of organizational structure alongside with system structure. The conclusion of the research is that we need to solve the problem with flaky end-to-end test and lack of monitoring with a focus on testing early and low-level. Also, that test engineering needs to get more prioritization in the development process alongside with communication. The research also became an inspiration with several ideas of further research within the themes found.

Keywords
Agile, Cloud based, Test Automation, Contract Testing, Monitoring, Observability.

Table of Contents

1. Abbreviations

GCP	Google Cloud Platform
AWS	Amazon Web Services
CI	Continuous Integration
CD	Continuous Deployment
TDD	Test Driven Development
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a service
FaaS	Function as a Service
SOA	Service-Oriented-Architecture
TDD	Test-Driven Development
CDC	Consumer-Driven Contract Testing

Thermology

In the book xUnit Test Patterns Gerard Meszaros sets the term *“Test Doubles”* as a generic name for Mock, Stub, Fake, dummy data. In this writing I will use that term when referring to replacement of production object/data for testing purposes. (Meszaros, 2007)

2. Introduction

2.1 Background

The methodology of waterfall is done in a linear sequential lifecycle model where all development activities are divided into separate phases. One phase needs to be reviewed and verified before moving to the next phase, which means that testing carries out only after development completion. In this test engineers are not involved from start in the development process. Testing cannot be done in parallel; each level of testing is done after each other. It also includes pre-determined requirements that needs to be followed and no changes can be made along the process. Long lead times, bottlenecks causing delays, heavy workload of documents to follow, no insight in other phases causing frustrations in the business. Alongside with this the monolithic architecture was used. Monolithic means a system is built as one indivisible unit. Usually built with one large codebase, where an update means that we need to modify the entire software and minor changes affects the entire application.

With the technology changing and evolving, demands on flexibility, scalability and better performance made many to migrate from monolithic/SOA to microservices. These demands also included the organizational structure, which have resulting in organizations changing architecture from waterfall to agile or a hybrid model between the two. The architecture changes meant a major change in how people were used to work.

With teams now working independently, systems including a large number of services and containers, the continuous integration and deployment could be made in parallels. Teams working on individual modules, deploying more frequent, testing can be done alongside the development process. It has solved many of the earlier frustrations and disadvantages seen in waterfall and monolithic. But there are also areas causing various problems with testing these systems. Testing monoliths we can use traditional testing approaches, microservices are too complex to do the same. We have independent teams to coordinate, components outside of the service, like databases, communication between each service and testing-levels. Microservices have a complex architecture and testing them are an architecture of its own.

Today many have migrated their systems to the cloud, made the application container based, using a variety of tools and resources like PaaS, IaaS, SaaS. We have test automation integrated in the system, different application platforms, frameworks, and languages.

All changes in organization and systems also means that we need to find new approached for testing, make sure to keep a high test-coverage and be able to trace issues inside the system.

2.2 Purpose

The purpose was to make an analysis how we can keep up with the growth of the usage of an agile approach of work, cloud-based environment, microservices, CI/CD and demand of faster performance.

2.3 Problems

The research main questions are:

1. What is the main problems companies are facing today within microservice testing when working agile in a cloud-based environment?
2. What could be a best practice approach to reach most effective testability and test coverage for a microservice environment?
3. What are the impressions of the priority of testing microservices/container applications among test leads?

2.4 Limitations

The focus has been on specific areas of microservice environment, and how approach testing within the system. The importance was that the project was cloud-based, and the project is done in an agile work approach. No testers who only work within waterfall models or not have any experience with test automation was contacted. It was not a requirement that they be working with test automation, only aware of what can be done with automation. The people I was contacting are active test engineers globally, within different fields. The questions have not included any directions towards other architectures than container based microservices, since there are so many different project architectures, and it could give a misunderstanding in the focus of research topic. Only time discussion went outside of the scope was when the person referred to earlier experiences or personal routines within test strategies and test plans.

The discussions are only done online, most of it written text. This was a choice upon brief time to write this degree project and to reach people globally in different time zones. There is no limitation in which field they have their expertise, but all are experienced test engineers. It is also limited to the abstract subject of microservices. What type of platforms, cloud host, specific companies’ software is not included in the study, it is limited to mentioned in some discussions as an example to clarify and visualize the point. The focus is not towards code, developer work, it is towards the testing role. I will not present any full project or code; this was something I used to build for testing different approaches in my research to gain a more detailed understanding only.

3. Theoretical background

3.1 What is cloud-based environment?

Simple explained cloud computing or cloud-based means that we use hosting for our systems and applications instead of storing them on a hard drive or an on-site server. So, cloud in this context means the internet, we store off-site. There are several types of cloud computing, where different services are being provided. We have the three main types: IaaS, SaaS, PaaS, and there is also FaaS and serverless. The definitions of which services that are included in what type and what is an overlap between them, can just as the word cloud-based be a bit abstract. This because people use the words differently and it includes several different systems and services.

3.1.1 IaaS, Infrastructure as a Service

With IaaS we have a service that will manage our infrastructure. This includes servers, storage, networking firewalls and virtualization. All the fundamentals are provided by a pay as you go basis. This gives a flexibility to scale resources as needed and increase the reliability of the infrastructure. We are still responsible for any data, applications, and operating system.

3.1.2 PaaS, Platform as a Service

With PaaS we still let the provider host the services included in IaaS, all the infrastructure. This model also includes everything we need to build our application, middleware, development tools, operating system, runtime, code libraries and more. Without the need to think about back-end features such as: maintenance and updates, we now can focus on hosting, build, and test.

3.1.3 SaaS, Software as a Service

This is the most known service; it is a user-ready software applications for specific functions. The service, updates and maintenance are managed by the provider. The user connects with the service over the internet and there is no need for installation on the user’s machine. Example of well-known SaaS: Gmail, Slack, Dropbox.

3.1.4 Serverless and FaaS, Function as a Service

Serverless and FaaS are other concepts used, they are many times said to be the same thing but there are some differences. With serverless the provider manages the configuration of the server, serverless databases, pipelines etc. basically all infrastructure. FaaS provides even more abstraction, it focuses on the services themselves and manages the server setup and installation of the application.

Then we also have different deployment models, Cloud, Hybrid and On-premises. I won´t go into any details about it here but worth to mention that it exists.

3.2 What is microservices?

Most applications used to be built with a monolithic architecture, a single application where everything is linked. Usually it has one codebase, including server-side, client-side, and database. All these components depend upon each other, this means that any change in in the code might affect the whole application. Since it only has one codebase the application might grow too big and make the development time-consuming and exceedingly difficult to scale.

The problems faced with monolithic applications combined with more applications deployed to the cloud led to the interest of microservice architecture.

3.2.1 History

There are some different claims of the origin of the term microservices, one is refereeing to Peter Rodgers presentation at the Web Service Edge Conference 2005, where he mentioned *"software components are micro-web-services"* (CloudComputingExpo, 2015),

Greg Young made a presentation where he explains the core ideas that has led to the microservices architectures as of today. He states that this goes back to the 1970s, and that SOA already had the aspects of microservices. (Young, 2016).

But for focus of microservices as we know them today in 2014 James Lewis and Martin Fowler decided to publish a detailed definition of microservices. This is a part of the definition they wrote:

"In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies." (Lewis, Fowler, 2014).

If we break this definition down in keywords we get:

- Small services, running in its own process
- Communicating, often an HTTP resource API
- Build around business capabilities
- Independently deployable
- Can be written in different languages
- Can use different data storage technologies

The advantage of this architectural approach is eliminating some of the problems the monolithic architecture is facing.

Since the service runs their own process independently, we get the possibility to develop and run them differently. Each team can decide their own approach, select their own code language, make code changes, build new features without too much concern about other services. Deployment is made isolated and automation tests can be set up to run with every deployment. With this independency all changes and behavior is connected to that specific service, in case of failure the rest of the application can still be up running.

3.3 Containerization

Simple explained containers are an encapsulated package of a software and its dependencies.

To explain the difference between a microservice and a container Ev Kontsevoy wrote this: *"A container is something DevOps people get excited about. A microservice is a software design pattern. It's something developers get excited about." (*Kontsevoy, 2021)

While microservices is created as single units they are mostly sharing dependencies like libraries, system tools and might need other services to run. Then we can use a container to store all we need to run the software. A container is portable and includes the application and all dependencies and can be deployed to a host operating system, which it shares with other containers. The data packed together inside of a container is called a container image and the instance of the image is the container.

3.3.1 CaaS, Container as a Service

Is a cloud-service for container-based software, it helps to automate the deploy and run process of the application in a more scalable and secure environment. It uses container virtualization, APIs, or web portal to connect with the application or containers and is many times compared as a subset of IaaS. CaaS does not have dependencies at the application level, which reduces the bottlenecks and resources and makes it have portability. One of the most well-known CaaS is Docker. From a testing perspective CaaS gives the service of providing logs, monitoring and performance checks to easier analyze the application data.

3.3.2 K8, Kubernetes

When using CaaS and the most common one is a s mentioned Docker, it might be that the application is installed on different operation systems. CaaS is mostly made to run a single node per host, to be able to manage all these different nodes we can use Kubernetes. Or as it is called a Kubernetes cluster, this container orchestration platform helps to simplify and automate the management of containers.

4. Methodology

4.1 Data Collection

To find the best approach for what to include in this research and which areas that needed to be a part of the analysis to make it reliable, I first contacted several different Test Engineers within different fields of testing. I wanted to form a better understanding in if they come across this topic and how they experience it affecting their work. It was also to get a feel of the level of knowledge if there was a high or low level of knowledge. These questions and conversations were only used to get a better understand of which tests that are focused on and to use that in my decision how to select and limit the research topics. I found it important to have this information from the perspective of test engineers, even more so with my background as a system developer. To not fall into the track of development and coding and down-prioritize the importance of the testing role. The answers were collected and used as a guiding point and not as a part of the actual analysis.

Different methods were used to collect data for the final analysis:

4.1.1 Discussions and questions

After the first contact with the test engineers, I decided to continue the same approach in the analysis. For importance of empiric data and to reliability in the analysis the discussions was made with the questions. The discussions were as mentioned above made in written form for the reason of time zone differences and time limit of the project. But this was a conscious choice to get an international perspective and see if the work strategies could vary depending upon location. The topic varies in the conversations depending upon the interviewed persons background and experience. For this reason, I found it important to let the person answer from their perspective but to use a semi-structured method in the follow-up questions. To keep this method in all conversations made it possible to keep the data comparable and dependable. I used a professional method that I have experience in from former education and used for several of years in earlier professions.

4.1.2 Online Research

After the first discussions and my experience in system development I researched the different areas of cloud-based, microservices, test planning and test strategies. At first, I made an activity diagram of an application, to have a visual overview of what parts we have included in an enterprise sized application. This way I could go thru the different sections and research the common test approaches done within that area. This gave me a more clear and specific way to narrow down the test strategy. Here it was of importance to use reliable sources, I tried to avoid personal blogposts and information, to keep the data fact-based and not opinion-based. Some physical books were also used, these was selected for the reason of their focus on specific topics within my research area.

4.1.3 Experiential Testing

To form a better understanding of best practices and to see the reach of different testing methods I built a small application, the application itself is not functional only built to get the tests to run. This was only made for the purpose of trying the methods and in some cases see the impact in performance. This does not include larger testing as end-to-end testing, since I found that it would be too time-consuming to develop the size of application to make the result relevant. During the time of the research, I also had access to different software and services that I analyzed and, in some cases, tried different test scenarios. This also gave me the possibility to access their documents and other educational material.

4.2 Ethics while research

The first conversations and questions asked on this topic was asked without this research in mind. The discussion started with my own interest in this topic, the questions asked was out of curiosity what other opinions was on the topic. It was this that lead up to the decision of this degree project, to deepen my own knowledge. For that reason, the people I contacted was not aware of the later purpose of our earlier conversations. All people involved was contacted again and asked permission to use their data in this research. And the purpose of the follow up conversations was clearly explained and once again asked permission to be used in the study. It was significant important that the participants felt comfortable to discuss freely therefore all was given the option to be anonymous and to specify if there were any specifics that they didn´t want me to use. Only a few wanted to be anonymous, and most was positive to share their names and profession. When starting to analyze the data I made the decision to remove all names and only where it could be relevant for clarification mention their area of profession. This conclusion was made upon both respect of privacy and readability. Also as mentioned above, the information evaluation of the research material was a high priority.

4.3 Analysis method

The get an overview of the data from the discussions I followed Boyatzis explanation of structured thematic analysis (Boyatzis, 2009). First, I used the research questions to identify various categories, these categories were later studied from the perspective of the main questions. This thematic analysis method gave themes with relevant and reliable data that could be used for further analytics. It was taken into consideration that the method had weak parts and can be viewed as it has been directed towards certain results based upon the interviewed and interviewers' subjective perspectives. This made it of significant importance to use the data based upon the theme of the questions. Also, the further research of the themes identified was made with careful consideration of information evaluation and keep it neutrally relevant to the themes.

5. Results

In this section a summary of the interviewed persons will be presented. It is chosen to call the discussions for interviews and the persons for respondents to reach better readability. It will also include a presentation of the practical research made.

5.1 Interviews with Test Engineers

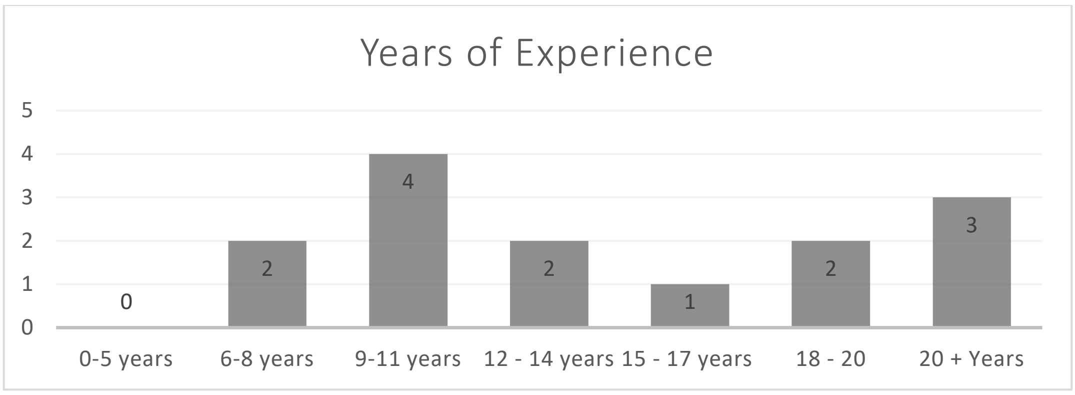
For the interviews there was 14 different people answering the questions. The responses varied in both in topic direction and length. As mentioned in the Ethics while research section some conversations was started on the topic before the research started, some of the subjects discussed is outside of this research and will therefore not be a part of the results.

5.2 Experiential Research

Here various categories within testing are researched, some categories were researched only with document and fact analysis others included hands-on testing. The methods and software used was selected upon varied reasons that will be clarified in their sections that follows. Not all experiential research will be presented in detail or with visual content.

5.3 Respondents data

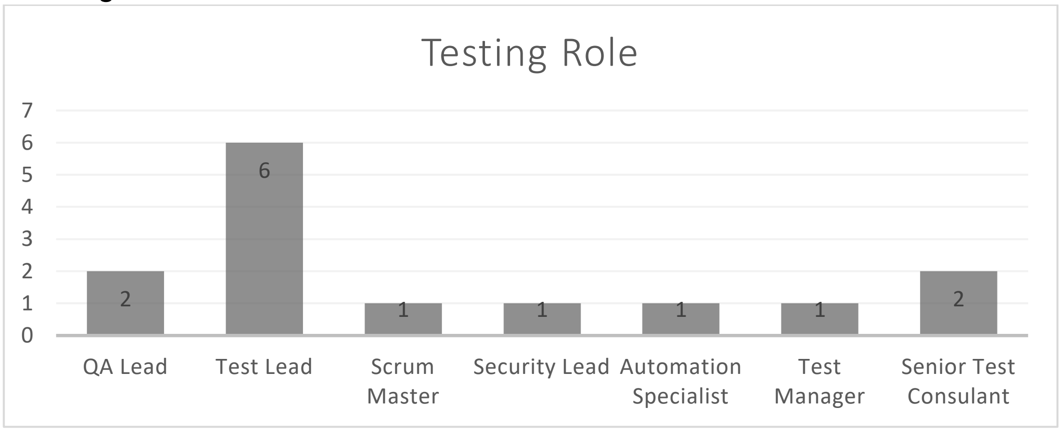
5.3.1 Years of experience



(Chart showing columns with numbers of people and their years of experience)

This chart shows the years of experience the test engineers participating in the interviews. 2 people have 6-8 years, 4 people 9-11 years, 2 people 12-14 years, 1 person 15-17 years, 2 people 18 – 20 years and 3 people +20 years.

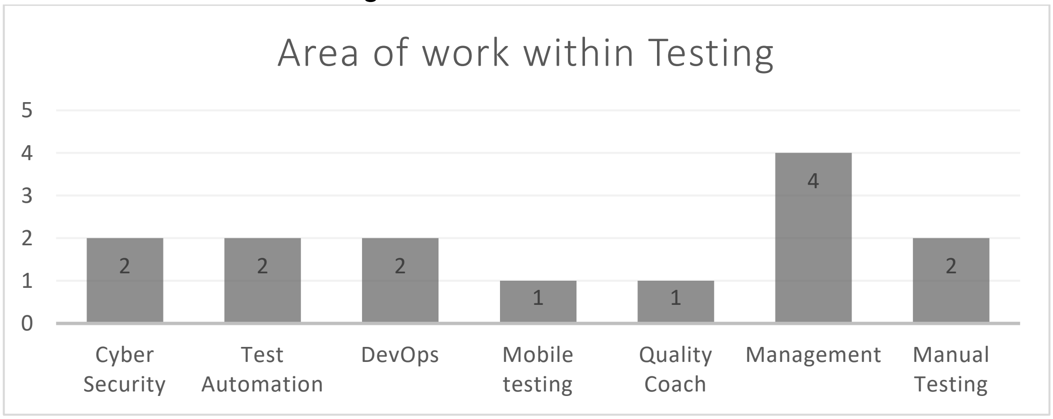
5.3.2 Testing Role



(Chart showing columns with numbers of testing role titles)

This chart shows testing role titles, where the person works as of today. 2 people QA Lead, 6 people Test Lead, 1 person Scrum master, 1 person Security Lead, 1 person Automation specialist, 1 person Test manager, 2 people Senior consultant.

5.3.3 Area of work within testing



(Chart with columns of Area of work within testing)

This chart is showing Area of work within testing. 2 people Cyber security, 2 people Test Automation, 2 people DevOps, 1-person Mobile testing, 1 person Quality Coach, 4 people Management, 2 people Manual testing.

5.4 Themes Identified

Based upon the answers from respondents I will in this part list the themes found in the interviews. Since the respondents was given the option to answer freely from their own perspectives the answers varied a lot. Therefor I will list the themes I found relevant to the specific topic.

1. Below are the themes identified on the topic of problems test engineers are facing. Here I asked about their thoughts of over-all problems and based on their answer some follow up questions were made for clarity. Also, to assure that the answer was from a test role perspective and not developer or organization based.

5.4.1 Production, Staging and testing environment differences.

One topic mentioned to be a problem and something that could cause more problem later was the overconfidence in staging environment. Explained with the example that there still is differences and the environments is not close enough to rely upon. If testing is made in a test or stage environment, we can´t know for sure that the application will behave the same in production.

5.4.2 Access

This was explained to be a problem working with microservice in any environment not just an agile work approach. That the testing is handed over to testers without reliable and accurate documentation, a tester can´t know all the ways this specific application and its services works without getting access to it.

5.4.3 Knowledge

This is a topic being repeated in almost every discussion, the difficulty to find the right person responsible. This includes to find the person who can explain certain features or architecture, to get access to software needed. Also, knowledge between different teams, the teams might work only on one part of an application or feature. If tests are being done over the team structure it might cause a bottleneck and delay testing.

5.4.4 End-to-End Testing

There are several problems mentioned under the topic of end-to-end testing. End-to-end testing is a method used in most projects and is seen to be the ultimate answer of the applications health. The problem mentioned is that end-to-end testing do not tell us about the interaction between containers, not between services or databases. It is also known to be flaky, to fail to produce reliable data.

5.4.5 Tool Sprawl

From the topic of knowledge this is one theme that I decided is important enough to set in its own theme. With agile the teams decide tools, frameworks etc. within the team, of course within certain limits. Different teams have different priorities and preferences, and this easily cause the summary of tools used to raise in numbers. It becomes difficult to overview, to find the correct software used and, in the end, to integrate data from a high amount of software.

2. In this section the topic was about best practices. First, I asked this question in an abstract direction to let the participants answer based on their own knowledge or what they would wish for. I realized my mistake here as the answers went in a lot of different directions and not focused on test coverage. This made me go back with a follow up question that involved type of testing and priority of those.

5.4.6 Performance Testing, Stress

Stress testing were one of the test techniques mentioned often as a best approach. The reasons for why we should do stress testing varied in the answers, but theme identified were to stress test the overall system. The approach mentioned was based on if we have available data from load testing. If we have access, we should automatically generate test data from that model and set about 10% greater. If we don´t have data available, the approach should be focused on the interconnected systems individually. Create a diagram of the system, find if we have critical areas in the service call chain, and then do high level stress tests on each individually. One important note was that we need to make sure there is a deploy available to manage potential downtime.

5.4.7 Test Automation, Contract Testing

To resolve the end-to-end testing problem mentioned above contract testing is mentioned. The arguments of why this is a best practice theme were not only about solving the end-to-end problems. It was mentioned as a method to gain control of the interaction between services, by isolating each service and with test doubles check that requests and responses share the same understanding documented in their contract. Mostly a provider-driven approach was discussed, an argument for this was that it covered evaluating the backend functionality and was seen as a more stable approach than consumer-driven contract testing. If an application was tested with contract testing it ensured stability earlier in the testing process and limited the need for end-to-end testing.

5.4.8 Logging

To reach full testability and observability logging were a theme that was included in the topic. Although, even with the importance of logging this topic also highlighted that most applications today are setup with good throughout logging data. It was more so the lack of knowledge how to collect the right data from unorganized logging systems.

5.4.9 Monitoring, Distributed Tracing

In the theme of logging, monitoring was another topic mentioned by mostly all participants. The type of monitoring that was seen as a best approach depended upon the participants background and specialization. The people with a profession within cyber security it was clear that this was a priority and both distributed tracing and service mesh. The importance of having abstracting layers (service-mesh) to take care of service-to-service communications, and to follow the request flow with distributed tracing. Other professions also mentioned monitoring and observability as one of the top best practice themes. That with monitoring we gain control to see issues in real-time and can easily detect where in application the problem occurs.

3.This question was asked to get an understanding how test engineers experience level of priority when it comes to testing in the development lifecycle. To see if there could be differences depending upon their role, company and perspective based upon the other topics discussed.

5.4.10 Test Automation

Test Automation was mentioned to be seen as a priority. To implement automated tests instead of manual testing. Here unit testing was discussed, developers might implement these tests and see them as enough to verify the application as tested.

5.4.11 Push to Prod

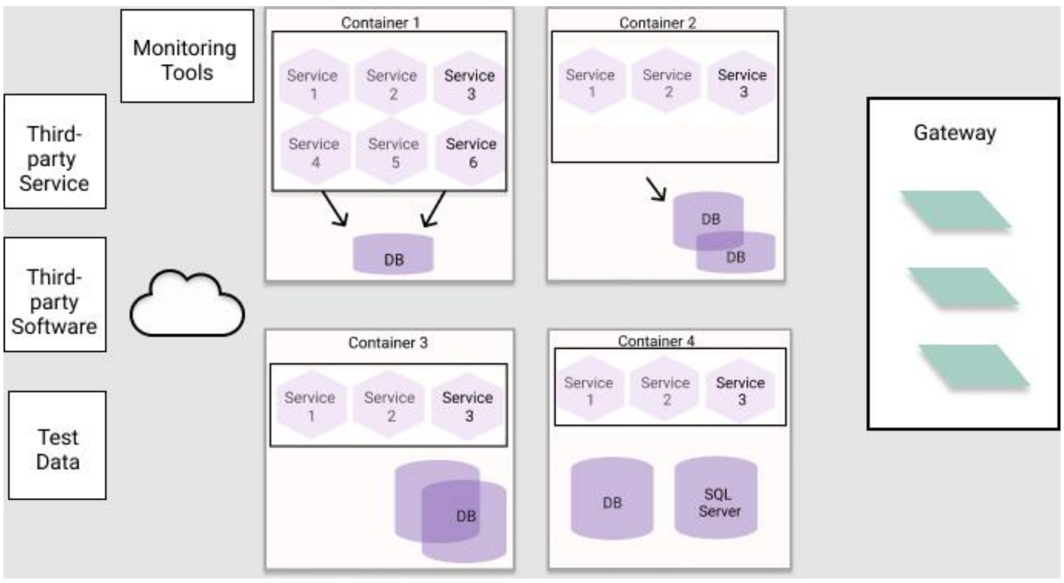
One participant said it like this: *“easy to fall into the trap of push to prod”*. The impression that if there is good monitoring in production things should go fast and easiest solution is to push directly to prod without testing. In the same category it is mentioned that with minor changes it is easy to justify skipping testing, since it is in fact just a minor change.

Developer perspective was another term used, said in the context of CD, testing being overlooked in favor of fast deployment.

5.5 Application Coverage

In this section I have taken the themes identified above and studied the different topics closer. I also continued my study of the thesis topic to make sure I cover most parts of what type of testing can be done within microservices.

I used this architecture diagram as a reference to see paths to cover and what could be important.



(Diagram with a simplified microservice architecture)

In this we can investigate some type of testing and follow if it covers all parts of the application. A note here that I will not investigate TDD or any of that type of testing, this is a decision based upon two reasons. The time limit of this thesis and the fact that the research I have done do not show test driven development as a common solution to implement to an already existing application.

5.5.1 Unit Testing

To start testing on a bottom level to ensure that each function within a service works correct, we use unit testing. Here we write the tests to observe the behavior of modules and can easily see changes in their state and find problems in the very first stage of implementation. It only focuses and gives us the result of that specific function. Even if unit tests mostly are done by developers and not handed over to a test engineer, I find it important to include it in this thesis.

5.5.2 Integration Test

In my research I came across Integration testing in most discussions and studies of microservices and agile work approach, therefore I looked closer at the topic.

After unit tests has confirmed the functionality of separate units it is needed to check the interaction of different units. On this second level of testing more of the application needs to be up running to determine that the interaction is correct. There are many parts and ways to test the integration, we have:

- Integration between different services
- Connection to databases
- Connection to other external sub-systems
- Higher to lower or Lower to higher level testing

- API testing

5.5.3 Synthetic Monitoring

To identify issues in the application before it reaches the end-user synthetic monitoring can be used. There are other terms used for this technique, in this thesis I have decided to only use the term synthetic monitoring with the opinion that it is a part of the testing process.

Automated, scriptable tools are used to perform synthetic monitoring. It is possible to cover most of an application's different areas with this monitoring method. I investigated some of the methods mentioned the most in my study:

- Ping test – check reachability of the application, ensure it is available to users.
- API tests – To monitor the applications API endpoints in both single and multiple-step endpoint requests.
- Step or Simulation tests – Here we use test doubles to simulate the steps taken by a user in the process of using our application and different functions.
- SSL Certificate monitoring - Ensure the security of the application by monitoring the validity, encrypted connection and up-to-date of the certificate.

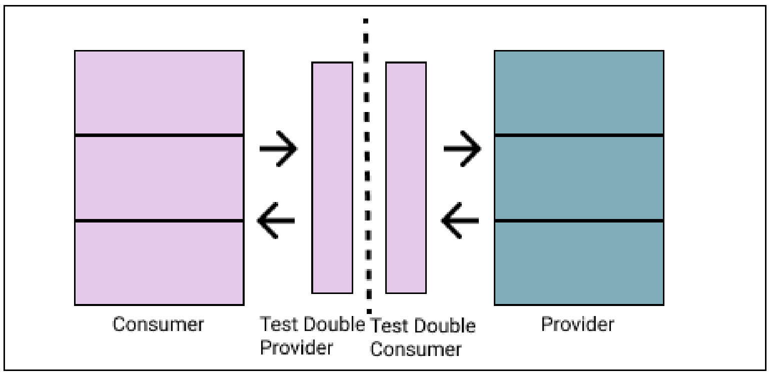
5.5.4 CDC, Consumer-Driven Contract Testing

In the section 5.4.7 Test Automation, Contract Testing above it is mentioned that the discussions mostly were directed towards Provider-Driven Contract testing. To see the functionality from the consumer requests and to ensure the provider understands the requests coming there is consumer-driven contract testing.

Here I decided to look further into the concept and write a small application to get a better understanding of coverage possibilities. In my analysis of the contract testing topic, I came across a team of testers and developers that was looking into this topic. The discussion was as mentioned directed towards provider-driven and backend, and it was done with the software Pact. Therefore, I decided to use the very same in my experiential research of CDC testing.

Pact is a code-first tool for contract testing. It is used for both Provider and Consumer-driven Contract testing. There is also Pactflow, which integrate with Pact but is used in the deploy pipeline as an automated contact test.

To explain contract testing this diagram shows the flow:



(Visual diagram over the flow in Contract testing)

As mentioned, I focused on the left part, where the Consumer sends requests, as seen in the diagram the request is sent to a provider contract with test doubles.

In my application I decided to build an Android application using Kotlin, the application is simply just comparing a To-do list. The test written is focusing on the function communication and not user interface behavior. The test is directed towards a test double API. Since it is possible to write these tests on a unit level it is a direct communication between consumer and provider as shown in the example above. I compared this with how we would need to set up integration test to make sure to cover similar field. This diagram shows the flow for integration test:

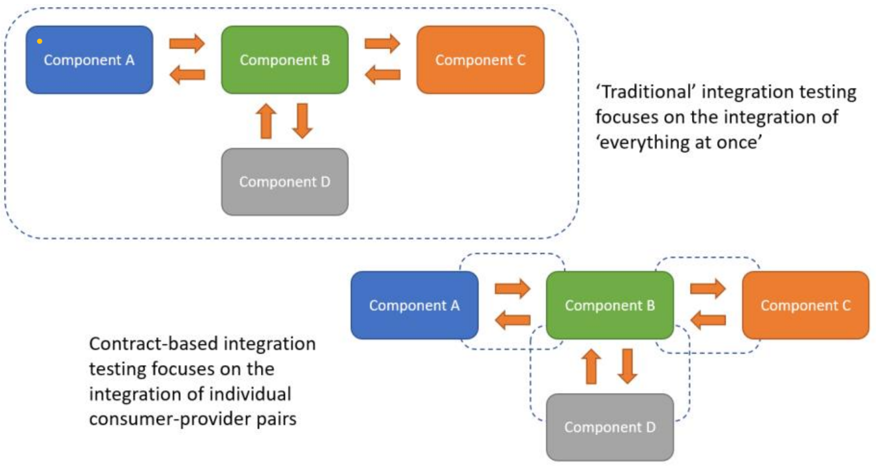

Component A
Component B
Component C
Component D
'Traditional' integration testing focuses on the integration of 'everything at once'
Component A
Component B
Component C
Component D
Contract-based integration testing focuses on the integration of individual consumer-provider pairs

Diagram from the article: An introduction to contract testing - part 2 - introducing contract testing

By Bas Dijkstra

As seen in the diagram there is requests sent and received with focus on the full service instead of just one service at the time.

6. Analysis

The analysis made from the empiric data is categorized based upon the different research questions. They are listed in sub-categories to have a controlled overview of the data in each theme and how to easier show how the data relate to the main research question.

6.1 Research question 1, Difficulties

What is the main problems companies are facing today within microservice testing when working agile in a cloud-based environment?

6.1.1 Perspective & Focus about problem areas

Here I divided the topic in two sections to easier define a pattern in the analysis. This first section is an analysis of the participants focus and perspective of problem areas.

Secondly the opinions they had about problems faced when working agile with cloud-based microservices.

Focus & Perspectives:

In this topic most answers were focused on non-technical subjects. This study was made with the decision to contact people who is working or have worked as test leads. What type of test lead or test field was not of the same importance, only that the experience was within agile and microservices. This had influence in their answers, a somewhat clear difference. Participant who mostly works within manual testing answered only within the area of the team's work environment. One example in the answers is a participant who clearly states this: *"the problem in any environment is not about access or running tests, it is about access to the information"*.

In these answers there are also a focus on the test planning more than test execution, mentioned was documentation, goal with the testing. Some mentioned short-staffed teams with too high workload.

One respondent has worked with in both larger companies and as an educator on the topic of Cyber Security testing. They had an opinion on testing that understanding was the most important focus. They stated it like this

"To take time to go thru every step of the system, gain all knowledge needed before going into the topic of testing is the way we can make an application safe".

The participants that have a more technical profession and some a background in system development mentioned more technical focus areas in the discussions. More about framework, parts of development pipeline, type of applications was discussed.

Tools within different areas was another focus, testing, monitoring and management. The participants in all roles and background had sometime during the discussion a focus on this topic.

Problem areas:

This question is one of the main questions for this thesis along with best practices, therefore it was important to make the conversations as in-depth as possible. This also gave results that varied on various levels. But two themes were repeated and stood out, overview and communication. These two will be mentioned in more dept in the next coming sections below. End-to-end testing were pointed out as a problem in every single discussion. In my own research end-to-end testing were also a repeated find to be a complex problem when testing microservices. In the research where monolithic applications were presented end-to-end seemed to be of a more reliable solution. On this topic I made follow up question of why end-to-end testing were used and if the reliability were questioned in their development process. The answers were basically the same, convenient and a "do as we always done" mindset in management.

There were also answers that went in details of security testing. I will not include those answers in this thesis with the argument that security testing is a topic of its own. It is a complex area and I felt that it wouldn´t do the topic justice to research within the time limit.

One participant mentioned that the one thing they thought where the most significant problem were test and production environmental differences. Clarification here were that we can´t predict all services behavior until they are up running in production.

6.1.2 Knowledge outside of the team

This topic goes as a connected thread in all discussions, no matter knowledge or profession. But the participants give different viewpoints of how and why. These are some I could find in my analysis of their answers:

- Agile teams decide tools, frameworks etc. within the team and do not communicate their decisions outside of the team.
- Developer team do not have testers in the team, this brings late communication when and what to test (for example: what has been updated, is regression testing needed)
- Documentation is non-existing.
- Goals and priorities might not be clear for all teams and causing unmotivating work environment.
- Developers and testers do not share same perspective of what is important, fast deployment or reliability tested.
- The difficulty for test leads to get the full picture of the application. To collect all information needed for tracing and observability, monitoring.

6.1.3 System Complexity

It is known fact that large applications with microservices, containers, IaaS, PaaS, FaaS, native development and more is a complex architecture. The answers mentioned this in a couple of different ways, I tried to identify and connect the ones that highlighted the same problem area.

To know that we covered the most critical areas and not missed anything significant was something mentioned in both end-to-end testing discussions and in discussion of how much different test techniques we need. Here the opinions differ a bit, in my analysis I can´t pinpoint exactly why since there is no clear pattern connecting these opinions. This will be explained in section *6.2 Research question 2, Approaches* below. Both in my own research and in answers from participant there are arguments that too little test automation is done, without automation we cannot do an approved test coverage, and argument that too much test automation is done, that automation is seen as the ultimate answer of cloud and service testing and not as a tool.

Testing too much with no clear test strategy was the answer from one participant, explanation of the statement was that test strategies tended to be high-level. Causing it to be interpret that as many different test methods and techniques as possible are needed to cover the testing areas.

Knowledge of frameworks, coding and cloud architecture among testers causing problems to understand where in the development process to start testing and to decide for correct type and technique for specific areas was mentioned by a participant with a role within DevOps and Scrum.

Performance and priorities, here I will combine several topics that was mentioned as my analysis says that all these answers were in the end pointed to the same things. There was the risk of performance issues if too much testing is included, both before deploying and in production. Too prioritize setting up a full monitoring architecture to reach full observability, instead of the logging sprawl. Logging sprawl is definition of logging included in the code without any documentation. (Sidenote here that I make assumption that by documentation it refers to diagram, application map or architecture connection.)

6.2 Research question 2, Approaches

What could be a best practice approach to reach most effective testability and test coverage for a microservice environment?

6.2.1 Shift Left

Shift left practice is the approach mentioned the most as best practice, agreement that we need to test early in the development process. One participant mentioned this as the best practice to avoid bottlenecks in both test process and in bug fixes. Testing early gives an overview of critical areas and makes it easier to set up a low-level test plan. Also mentioned in the topic of shifting from waterfall to agile, that it is a benefit to implement a shift left testing to easier get the teams to work side by side in the development pipeline. In the perspective of security one participant mention shift left and integrate with the DevOps team. To drop tools into the pipeline for a better test flow.

6.2.2 Low-level Test

To test often and focus on services functions is another approach that is mentioned by both participants and in most research materials I came across. There is a difference in the how depending upon when the material was written. With the change of new architecture ideas, more knowledge, and added resources the opinions have changed.

In my analysis I followed up two well-known companies that I am familiar with and looked on their test approaches from around 2014-2016 and today. Company 1, 2014. First, I want to mention that this company has testing as a high priority and in my analysis the conclusion is that their discussions of development process have more focus on testing than coding in most material I read. They had a structure of, Unit test, Integration test and end-to-end integrated test. This combined with logging and monitoring. Today they do use unit test but with caution, explained that unit tests limit code changes and are time-consuming. End-to-end test is something they trying to avoid completely. The approach they have today is:

- Cross-functional teams, with test expertise areas
- Unit tests but not seen as testing, it´s quality checking.

- Implementation testing only on isolated complex parts of the application
- Integration testing
- Contract testing, consumer-driven
- Test in production, Internal testing
- Monitoring

Company 2, 2013. When company 1, had testing as a high priority company 2 is the opposite. They leave a lot to the developers, put the developers responsible for code quality and reliability. Therefore, they also put a lot of focus on unit testing and secondly automation test. Test automation is seen as testing and most used. From the automation they rely upon logs to trace issues. And for today they have the approach:

- Developer code-ownership
- Unit tests
- Test Automation, automate all them tests.
- Test in production, dummy data/ testing doubles
- Beta testing

6.2.3 Test in production

From shifting left we have a test coverage of functions and communication between services, but we still do not know how our system will behave in production. Here there is a clear opinion among the participants and my research, where I couldn´t find many differences. To test in production is high priority, my analysis shows that it is mentioned just as much as Test automation in deployment pipeline as it is in discussions of test in test environment.

A bit of a side theme including the three sections above, felt worth to be mentioned in the context of best practice instead of or together with End-to-end testing. As results and analysis shows end-to-end testing to be flaky and a problem, load testing is something mentioned as often. Load testing is mentioned as an obvious test needed, both in integration testing and end-to-end. Both load and stress testing will give us answers of the application’s overall behavior and critical areas both within the services and communication between.

6.2.3 Monitoring, Observability

In this section I will collect all types of monitoring, logging, synthetic monitoring, tracing for easier overview.

Monitoring of some sort is mentioned in every discussion, every research material on microservice architecture. The least mentioned is logging, I tried to find a theme of why but couldn´t find any specifics. If we look at some answers from the participants logging is mentioned, one to use logging data for testing. Another is for measure data to find potential issues. Therefore, I connect this with monitoring more than basic logging. Reason for this is one participants’ answer that they do have lots of logs but no real structure to monitor what and why. And that the data is difficult to use for more than specific already appeared issues and then mostly serves the developers.

The research for best approach comes to land on MELT,

- Metrics
- Events
- Logs
- Traces

6.2.4 Communication

This topic has become a critical part of this research, participants mention communication repeatedly. I also had the opportunity to do this research at a time where I had closer insight in a company were I both could study and ask questions about their internal architecture within both organization and system. This gave me data to study some areas closer and compare to other parts of my research. First, I want to mention that this company has a well-covered communication approach, it was noticeably clear that they put a lot of planning into this. The employees get onboarded with information how to communicate with all areas of the organization, using several different channels like Slack, Video meetings, daily, weekly, monthly updates within and outside of the teams. Test engineers work in the teams but also have their own meetups. In my analysis this gives a reliability to know that testing solves issues in the application and developers are fast informed of these. Although, communication is lacking in areas where it affects the test engineers. My conclusion for this is that all professions have different priorities, knowledge and understanding.

As one participant mentioned that in any environment not just agile, developers understanding, and data availability is not enough.

A find in this topic is that different test engineers have different perspectives, and this seems to cause a misunderstanding between testers. Testing is seen as a group of whole, working towards the same goals. This seems to depend on testers backgrounds and focus areas, but not to be discussed among testers therefore causing misunderstandings.

6.2.5 Organizational structure

I will start this section with two insights of one company´s problems and changes in their organizational structure:

"... fragmented ecosystem of developer tooling where the only way to find out how to do something was to ask your colleague. 'Rumor-driven development', we endearingly called it" (Spotify, 2020)

On the same topic within a new client-side architecture build, they made changes by parallel testing and development, adding more people to the test groups along when the feature they were working on became more stable. They went from 30 employees to 1000 in four months. And they state that this gave them control over the testing process. (Spotify, 2020.1)

The text above summarizes my analysis of this topic rater well. It shows that no matter how much tools, automation, expertise, IaaS, PaaS we use if there isn´t a well-working structure with enough resources it becomes an issue. I do a parallel to "put out fires" in development, where we tend to fix an issue without thinking ahead. In the end the codebase becomes fragile, costly, and time-consuming to fix.

From the participants answers there is a shortage of test engineers within the organizations that might cause critical areas in testing a complex system gets overlooked. A best practice is to invert the resources and make sure testing is a priority with good test plans, strategy, and knowledge coverage. From the themes above we can also see that expertise in different areas is needed. A test engineer team with organizational knowledge, development knowledge, manual, regression knowledge. And specific specialist areas as: Security, platform, accessibility and more.

6.3 Research question 3, Priority

What are the impressions of the priority of testing microservices/container applications among test leads?

Analysis of this question were following a somewhat connected thread thru all answers, the majority started the answer on this question with: *"No, ... "*. The clarifications followed were divided to the theme sections below for easier summarizing.

6.3.1 We and them

There is a clear opinion of a division between test engineers and developers, and test engineers and management. For developers and testers, the cause is mentioned to be that developers do not provide enough information and have too little knowledge, understanding for testing. There was also a difference depending on the test engineer's role in these answers, manual tester with a more theoretical background focused the answers on developers' ability to give too technical information instead of documentation. Opposite for the participants with roles and backgrounds in development, security, or test automation.

For the management and test engineers' opinions it was down prioritization of testing in favor of production that was mentioned. More importance of build, release new features than application performance.

6.3.2 Continuous Deployment

This theme continues from the theme above, we and them. I will use the participants statement from above for explanation: *""push to prod" trap".* This is said in other ways by other participants, that it is easy to overlook testing minor changes, since they are just slight changes. If test engineers are left out on these minor changes it will become a bottleneck in the end if issues are detected later in production. Here I also connect the managements decisions of new features and fast changes to be a part of the opinions that test engineers are not needed to be involved unless there are issues.

6.3.3 Test or Tool opinions

Two things are connected to this theme in my analysis: Test automation and unit testing. Unit testing is mostly done by developers and by the answers from the participants there is a problem with developers view on it. Developers see Unit tests as a good enough assurance for definition of done when test engineers do not. Another theme found in the analysis is the opinion if Test Automation is testing or just a tool in testing. From the participants answers most sees test automation as a tool, it can help the testing process but not replace manual testing. (Sidenote: that I use the word manual testing for clarity but am aware that some test engineers do not agree with the term, that it should be called just testing).

6.3.4 Resource shortage

In my own research I studied opinions about team structure with two main questions: 1. How many test engineers in each team/in the development. 2. How does it differ from number of developers for the same workload? The subject was not easy to find information about, as mentioned above most material about cloud-based and microservices is towards development, or management in agile. Testing and test engineers are mentioned in the sidelines or grouped together as test lead or test engineers in the descriptions, diagrams seen. This when developers are divided in DevOps, Backend, Front-End, Android, iOS and more. Management is mentioned in specific roles, like, Financial, Operation manager, Domain Expert, Senior management.

And for the main questions it was not possible for exact numbers since it varies depending on company, size of system, application. Although, there were a pattern in one test engineer per team/service. This did not seem to change with larger systems or with deployment frequency. It was less likelihood that more test engineers were added to a team than extra developers where there was a need for more resources. This I connect to the push to prod theme above.

7. Discussion

7.1 Findings

With a background as a developer, I have been working with Cloud-based systems and microservices. Agile teams are also a familiarity. Adding years in the profession of technology such as mobile devices, tv and other media, together with my educational background including assistive devices; Most topics I researched was familiar at some level. Also, the knowledge of the lack of this topic among developers was one of the things that got me to start research it before I decided to write this degree project. The empiric investigation and analysis were at first thought to be more towards a case study of lower-level performance testing in microservices. During the time I started the research the circumstances to be able to investigate this in detail changed and I also felt that the questions in this study were important to gain a deeper understanding from the perspective of test engineering.

Next sections include a discussion of findings in the analysis, thought, some opinions and conclusions about the findings.

7.1.1 Difficulties

What is the main problems companies are facing today within microservice testing when working agile in a cloud-based environment?

Not one participant gave just one area the reason to be the main problem, but most focus on non-technical areas. That the focus went in the direction of test automation or not test automation was not surprising, this is one of the most common discussions I can see among test engineers today. Although, that few participants didn´t even mention any type of technical test structure more than end-to-end testing as a problem directed me into research more about organizational structure than I had expected. The opposite could be said while researching reading and video material except from end-to-end testing. One question I get from this is if this only is opinion-based or knowledge-based to some degree? Test engineers that doesn´t come from a technical background and mostly focus on manual testing might not know what could be done with test automation. And could the focus on test automation be influenced from developers and CI/CD pressure. For readability I will continue this dividing theme in this section.

One problem that I see and would want to say is the main problem is communication. Both as said in the discussions, there is too little information developer/tester/management and between teams. When going from waterfall to agile we got a better parallel process and more independency. I do not question this, agile gives more benefits and facing less problems for the cloud-based system environment. But could the independency cause the teams to prioritize it more than teamwork outside of their own no matter chance for improvements. Test engineers is as I found in my research mostly just one in a team, it is easy to overlock that one person's "actual" work tasks. Again, the *"push to prod"* statement, get the release out and deal with potential problems later. Larger business tends to do test strategies on high-level which leaves room for great planning within the teams but also leaves room for ignorance and down-prioritization.

End-to-end testing is a clear problem found, the tests being flaky, takes long time to set up, to run. Gives the conclusion that the result is not worth the effort if we can cover it in other ways. As mentioned in 6.1.3 System complexity there are lots of areas within microservices that can be problematic. When starting this research, I thought it would be easy to focus on best practice for lower-level testing and see the performance within the service when they communicate with each other. That with unit test, API test and end-to-end test it would be easy enough. The insight and knowledge along the way shows a different result. But in my opinion, it is correct to focus on insight between services. The more services, containers, databases we add the more activity we need to maintain. One conversation I had was about the problem of the software outside of specific services/containers. In that example it was a database which caused several critical issues but did not show up in the logs or issue reports. They had unit tests, test environment, regression test and end-to-end test done but this shows the importance of testing integration. I will summarize this topic with a statement from J.B Rainsberger article series on integrated tests: *"You write integrated tests because you can't write perfect unit tests. You know this problem: all your unit tests pass, but someone finds a defect anyway. Sometimes you can explain this by finding an obvious unit test you simply missed, but sometimes you can't. In those cases, you decide you need to write an integrated test to make sure that all the production implementations you use in the broken code path now work correctly together."* (J.B Rainsberger, 2009/2021).

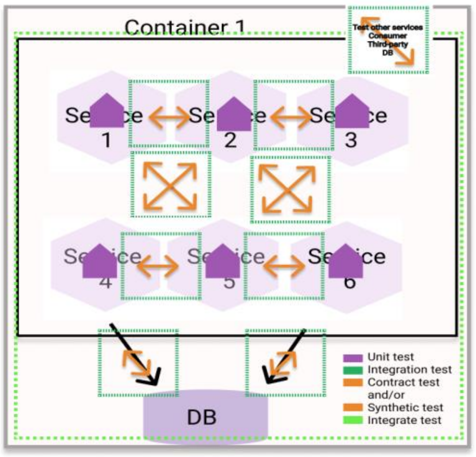
This leads to the problem with knowledge, testers that don´t have the knowledge of coding, microservice architecture could never set this as a priority in a test plan.

The two topics above connected in my analysis when I came to research the reason for tool sprawl and access. Today most uses PaaS of some kind, GCP, Azure, AWS. There was where I started the analysis, I investigated what type of monitoring could be done within GCP? Would it be possible to do the topics mentioned as best practice? Problems found are that both lack possibility for one or another monitoring feature. Some can be imported by a third-party tool, the problem remains, the usage of several tools for different features. And even integrate with the PaaS they can´t combine the data outcome. Teams decide themselves what tools to use, with probably the PaaS as common connection to the rest of the organization. And of course, teams have their own preferences what they like to work with. With these facts it is easy to see the cause of tool sprawl.

7.1.2 Approaches

What could be a best practice approach to reach most effective testability and test coverage for a microservice environment?

This part was the most complicated part to research and analyze, not only for the complexity. To keep the data dependable, neutral, and manageable there were times I had to make decisions in priority of research, ask myself if the decision were based upon analysis or my own past knowledge. As mentioned in section 5.5 Application coverage I used the diagram to follow the coverage. With this I took the analysis data and set into the diagram. It ended up something like this:



I do believe Unit tests are important, if not only to make sure the functions work as expected. Here we can also work on the topic of communication and knowledge. Developers who write unit tests will get more insight in testing and could gain more skills in write clean code. Clarification that I do not say that developers write bad code or lack skills, my point is that we learn and develop from our work processes. Although, Unit tests should not be implemented everywhere or be seen as a coverage for the entire service or application.

My conclusion is connected to my first topic for this research, more focus on the integrations between services. We have contract testing; it will be implemented early in development and prevent flaky testing. During the research I came upon a writing that I felt explains where my conclusion was going: *"Strong integration tests, consisting of collaboration tests (clients using test doubles in place of collaborating services) and contract tests (showing that service implementations correctly behave the way clients expect) can provide the same level of confidence as integrated tests at a lower total cost of maintenance"* (J.B Rainsberger, 2015)

Reading the text above it says *the way clients expect* and that is where we can gain stability, reliability. CDC, consumer-driven-contract testing. With mobile applications and other similar platforms, we need to focus on consumer experience. There are unit tests in backend, we add test automation in the deployment pipeline and add minor integration testing, there we have coverage of backend. With this there is ability to set up performance tests such as stress and load testing.

With CDC and API tests on consumer-side the need for test environments decreases, which means testing in production can be prioritized. Test engineers can put focus on manual testing when new releases go in production. Same for developers the risks for system crashes can be caught earlier in the process, and a release do not cause a bottleneck of bug fixes.

Alongside with a more organized shift left approach the analysis shows that monitoring is significant. This too to be implemented early in the development process by synthetic monitoring. We can cover most areas of an application with synthetic monitoring and identify issues before they reach the end-user. This is also an insight for the developers to see more of the actual behavior of the application and functions.

The problem highlighted in the research was tool sprawl and unorganized logs. During a conversation I had when analyzing the different tools and software used by a company the person said this:

"We don´t have to reinvent the wheel. We just have to find the best resources that lessen the scattering"

That statement pinpoints the exact problem with agile teams decide tools within the teams and having diverse needs and preferences. Together with the find above in section 7.1.1 Difficulties, most PaaS do not have the possibility to monitor all areas mentioned, not even with third-party extensions. In this research I tried to find a tool where we would be able to:

+ Integrate with PaaS

use MELT:

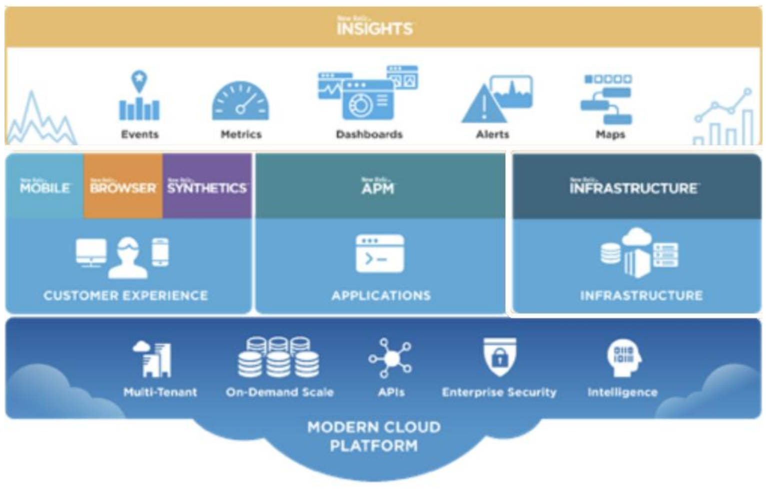
- Metrics
- Events
- Logs
- Traces

+ Synthetic monitoring

+ Separated API testing

Preferably without needing to use some sort of extension integration except from the PaaS, IaaS, and SaaS.

I would say that the result was low, few options have this wide inclusion in their tools. But New Relic was one who had all these features, during the time I did this research I had access to investigate the tool closer. Here we can see the infrastructure:



(Photo from New Relics website, blogs

New Relic 2021)

Conclusion of this is that the tool would be able to do all the things that today is scattered out in different tools by the teams. Which get us into the theme of communication again. Which will be my main final part in this discussion of best practice findings in this degree project.

Communication.

Above we have gone thru:

- ✓ Unit test
- ✓ Integration tests (Provider-driven)
- ✓ CDC, Consumer-driven-Contract test
- ✓ CD Automation test
- ✓ Synthetic monitoring
- ✓ Manual test
- ✓ Distributed tracing
- ✓ System Monitoring/ Observability

Most of these can and I would go as far as saying will need the teams, developers, just as tester to communicate, to work together learn and teach each other of their area of profession, over the team borders.

I did not have any thoughts that this research would give a straight-line answer for one architecture to be the ultimate one. It is obvious that everything depends upon individual factors, such as product, company, organizational structure, software´s used and so on. There will always be room for improvements alongside of new knowledge and development, these findings should be seen as a reference part of the whole. The direction the themes found led shows that we can´t focus or ignore areas within either organization or system architecture when it comes to software testing.

7.1.3 Priority

What are the impressions of the priority of testing microservices/container applications among test leads?

The conclusion on this is simply: The impression is that testing is not seen as a priority for microservices. All answers except one from discussions and interviews were started with no, the other was started with *"it depends"*. The second answer were also explaining that it is easy to ignore testing if the changes are small and the monitoring don´t show critical issues. This I connect with the same as a no and down prioritization of testing. One theme I found and been analyzing back and forth, test engineers seem to down talk their own importance. By this I mean that in discussions about testing and team organization there seem to be a pattern where testers meet the discussions with approving to release early, take on more tasks with the argument that it will be more work, but we will handle it. Opposite from my impression with developers there the answer would be something like, this is more workload, we need more developer resources. Is this a behavior of old times, do as we always have done? In a waterfall environment with a monolithic application a smaller team of test engineers would easier handle to have full overview in tests and monitoring. But with agile split up teams and a more complex architecture I can´t ignore the thought that testers need to speak up, the management to put testing in budget priority instead or alongside that new feature. This conclusion also comes from the research material and so much focus on test automation, writing code as the ultimate answer of solving issues of all kinds.

7.2 Method

The result from the methods used to investigate, select, and collect topics and data felt to have given a good result. Afterwards it felt to be good choices both for reliability and variety. The fact that I already had started conversations on the topic with different test engineers made it easy to continue the investigation with semi-structured questions. This method is something I have worked with before both in research and for years in previous professions.

The topic is also something I have an interest in since I started as a developer, this was something I had to remind myself about to not make it affect the analysis or my research approach. Therefore, I feel that the discussions with the test engineers were important and gave an insight in the industry that I wouldn´t have found in reading material.

Since my main thought in this research was different from start, I do feel that I could have structured the discussions differently if I had thought of this approach before.

The time limit made it necessary to make some decisions of how detailed my research would be, if I should try to get closer to the main thought and study some test techniques in a detailed case study with the other themes found as a more side-research. In review this approach with high-level research with a full overview of agile and cloud-based microservices felt like a good choice. This has given me a deeper knowledge of opinions in the industry, both in organizational structures and in thoughts of testing what and why. Deeper insight in test architectures and available software as well.

It was difficult to find research material that were directed towards testing only and based on facts and not opinions. The analysis was carefully done, and a lot of data found was deselected upon lack of reliability. Here I maybe could have widened the research area and taken more time finding material. Something I will keep in mind for continued research on this topic. Same goes for the experimental research, it was not possible to both do the discussions, research on the themes found in the analysis and to developer a project to try all techniques mentioned in the research. This is something to have in mind in further research.

8. Conclusion

Two approaches are used in this study, where problem areas, best approaches, and test engineer's role in the industry, is investigated. The research was done by

- Discussions and semi-structured interviews with test engineers about their knowledge, experience, and opinions.
- With investigating reading and video material and experimental research on some of the material found.

Below follows the conclusions from discussions and findings for each of the study´s three main questions.

8.1 What is the main problems companies are facing today within microservice testing when working agile in a cloud-based environment?

Most participants focused this topic on non-technical subjects.

Lack of documentation both of system structure to get an overview what and how to test and in data collection was a concern mentioned.

Too independent teams were reason to a variety of problems mentioned. Such as, testing being overlooked in favor of fast deployment, tool sprawl where it becomes difficult to know where to find everything needed to monitor and trace data, less knowledge sharing or collaboration for reliable testing coverage and architecture.

End-to-End or integrated tests are flaky, time-consuming and don´t give reliable results.

System complexity makes it easy to overlock critical areas or unseen problems of the application. To test for the expected and not the unexpected.

Test environments and test doubles are too different from production environment. One participant mention that there is an overconfidence in staging environment that cause unnecessary issues.

8.2 What could be a best practice approach to reach most effective testability and test coverage for a microservice environment?

Shift left, to test early gives us the possibility to avoid bottlenecks and bugs in production.

Focus on low-level testing, test within the services. Here participants mention unit test, synthetic monitoring.

Test automation in the deploy pipeline, check before going to production.

Communication between test engineers and developers, knowledge exchange and common priorities. This includes CDC, consumer-driven contract testing.

Minimize tool sprawl with software that can integrate with used PaaS and have the ability for connected dashboards with: Metrics, Events, Logs, Traces as well as synthetic monitoring.

Test in production, internal testing with real user data instead of test environment or test doubles gives an accurate view of system behavior in production.

Gain control over the testing process, as mentioned in the example in section 6.2.5 where the company changed to parallel testing and development, adding more people to the test groups instead of working with limited resources or too few test engineers in each team. . A test engineer team with organizational knowledge, development knowledge, manual, regression knowledge. And specific specialist areas as: Security, platform, accessibility and more.

Without effective communication and enough resources, it wouldn´t matter how much testing or how many techniques, or test levels we have. There will be bottlenecks and issues.

8.3 What are the impressions of the priority of testing microservices/container applications among test leads?

All answers from participants when asking if they thought testing was seen as a priority, started the same way, with a no or it depends.

For the management and test engineers' opinions it was down prioritization of testing in favor of production that was mentioned. More importance of build, release new features than application performance.

One participant used the statement *"Push to prod trap*" and other gave similar answers with explanation that it is easy to overlook testing minor changes in favor of fast deployment.

Test automation seen as a replacement of other types of testing is mentioned. In answers from participants most of them sees test automation as a tool, it can help the testing process but not replace manual testing. (See explanation of the use of manual testing in section 6.3.3)

There was a pattern in one test engineer per team/service. This did not seem to change with larger systems or with deployment frequency. It was less likelihood that more test engineers were added to a team than extra developers where there was a need for more resources.

9. References

Books

Aroraa, Kale, Manish (2017). *Build Microservices with .NET Core*. Pack Publishing, 2017

Boyatzis, R E. (2009) *Transforming qualitative information: thematic analysis and code*. Ca.: Sage Publications, 2009

Meszaros, Gerard (2007). *xUnit test Patterns: Refactoring Test Code*. Addison-Wesley, 2007

Websites

Cloud Computing Expo, Peter Rodgers speech:

https://web.archive.org/web/20180520124343/http://www.cloudcomputingexpo.com/node/80883 (read 05-01-2022)

Young, Greg Young speech, The long sad history of microservices:

https://skillsmatter.com/skillscasts/11143-event-sourcing-and-microservices (read 25-12-2021)

Martin Fowler and James Lewis, Microservices – a definition of this new architectural term

https://martinfowler.com/articles/microservices.html (read 25-12-2021)

Kontsevoy, Microservices, Containers and Kubernetes

https://goteleport.com/blog/microservices-containers-kubernetes/ (read 25-12-2021)

Microsoft, .NET Microservices: Architecture for Containerized .NET Applications

https://docs.microsoft.com/en-us/dotnet/architecture/microservices/ (read 25-12-2021)

Google cloud, testing overview

https://cloud.google.com/functions/docs/testing/test-overview (read 11-12-2021)

Google cloud, cloud monitoring

https://cloud.google.com/monitoring (read 11-12-2021)

Microsoft, what are microservices?

https://docs.microsoft.com/en-us/devops/deliver/what-are-microservices (read 25-12-2021)

Microsoft, microservices architecture style

https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (read 25-12-2022)

Microservices.io, What are microservices

https://microservices.io/index.html (read 26-12-2021)

Microservice.io, Pattern: Service Integration Contract Test

https://microservices.io/patterns/testing/service-integration-contract-test.html (read 26-12-2021)

Microservice.io, Pattern: Service Component Test

https://microservices.io/patterns/testing/service-component-test.html (read 26-12-2021)

IBM, SOA vs. Microservices: What’s the Difference?

https://www.ibm.com/cloud/blog/soa-vs-microservices (read 26-12-2021)

Aws, Automating your API testing with AWS CodeBuild, AWS CodePipeline, and Postman

https://aws.amazon.com/blogs/devops/automating-your-api-testing-with-aws-codebuild-aws-codepipeline-and-postman/ (read 26-12-2021)

New Relic, New Relic one

https://docs.newrelic.com/docs/new-relic-one/use-new-relic-one/get-started/introduction-new-relic-one (read 28-12-2021)

New Relic, Lookout

https://docs.newrelic.com/docs/new-relic-one/use-new-relic-one/core-concepts/new-relic-lookout-monitor-your-estate-glance (read 28-12-2021)

New Relic, Learn courses

https://learn.newrelic.com/ (Watched 28-11-2021)

Bas Dijkstra, An introduction to contract testing – part 2 – introduction contract testing

https://www.ontestautomation.com/an-introduction-to-contract-testing-part-2-introducing-contract-testing/ (read 30-12-2021)

Pact, what is contract testing

https://pactflow.io/blog/what-is-contract-testing/ (read 30-12-2021)

Pact, how pact works

https://docs.pact.io/getting_started/how_pact_works (read 30-12-2021)

Pact, pactSwift overview

https://docs.pact.io/implementation_guides/swift/ (read 31-12-2021)

Spotify, How We Use Golden Paths to Solve Fragmentation in Our Software Ecosystem

https://engineering.atspotify.com/2020/08/17/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem/ (read 29-12-2021)

Spotify, Spotify Modernizes Client-Side Architecture to Accelerate Service on All Devices

https://engineering.atspotify.com/2020/05/28/spotify-modernizes-client-side-architecture-to-accelerate-service-on-all-devices/ (read 10-01-2022)

J.B Rainsberger: Integrated Tests Are a Scam

https://blog.thecodewhisperer.com/permalink/integrated-tests-are-a-scam (read 11-01-2022)

J.B Rainsberger: Clearing up the integrated test scam
https://blog.thecodewhisperer.com/permalink/clearing-up-the-integrated-tests-scam (read 11-01-2022)