Pilot experiment to provide evidence supporting the feasibility of using Azure Service Fabric

Thanh Hao Nguyen

2020 Laurea

Laurea University of Applied Sciences

Pilot experiment to provide evidence supporting the feasibility of using Azure Service Fabric

Thanh Hao Nguyen
Business Information Technology
Bachelor's Thesis
May 2020

Laurea University of Applied Sciences
Business Information Technology
Bachelor's Degree

Thanh Hao Nguyen

Abstract

Pilot experiment to provide evidence supporting the feasibility of using Azure Service Fabric

Year 2020 Pages 38

The purpose of this thesis project is to provide evidence to support feasibility of using Service Fabric in development for the client company - Nord Pool. The company had issues with some of its services processing data slower than allowed. With some theoretical research, developers of the company believe that their problem can be addressed using the product Microsoft Azure Service Fabric. The challenge at the time was finding positive evidence to support the feasibility of using said solution in development.

To accomplish the project's goal, the author of this report had created a pilot experiment. This experiment's objectives were to replicate some needed functions and applications using solely Service Fabric. The expected outcomes were small instances of the company's services running on a Service Fabric cluster. Additionally, the development process was required to follow the Service Fabric framework. If this requirement is fulfilled, it suggests that the platform can assist software development.

In this project, the author had utilized the "waterfall project management framework" and "software development life cycle". These two methods were used to draw up plans to conduct research into development requirements, make designs, build the needed instances and test the final product. Several tests were created which included functionality testing, high workload simulation (load test) and chaotic production environment scenarios simulation (chaos test). The sample services and their underlying infrastructure managed to pass all these tests which support the resiliency of applications built upon Service Fabric framework.

With this project's result, Nord Pool can have initial knowledge on the requirement and process of developing applications on Service Fabric. This report will act as initial documentation for future development. Within this document, technical requirements and knowledge needed to build application on Service Fabric was introduced. Following these findings, developers can be more prepared when working with the framework.

Keywords: Microsoft Azure Service Fabric, pilot experimenting, infrastructure, cloud computing, waterfall project management framework.

Table of Contents

1 Introduction

1.1 About the company

Electricity had become a vital part of human's daily activity. Energy is needed for every appliance in daily lives, such as smartphones, streetlights, servers, etc. According to the International Energy Agency (IEA), in 2019, the demand for energy is on the rise with a 4% increase in 2018. With roughly 8 billion people in demand for electricity and hundreds of suppliers around the world, it is impossible to get energy to where it is needed without proper market operation. Nord Pool is one of the world's leading power market operators. The company ensures that power supply and demand are met through its wide range of services. The company's main role is to ensure that electricity is bought and sold at equilibrium prices every single day. Additionally, there should not be any surplus of electricity from sellers or deficient from buyers. To achieve their goals, Nord Pool provide many services which plays a vital role in the functionality of the power market. The Day-ahead service provides an auction system where companies can post the expected selling electricity price and expected buying price. Once every company has posted, the system will match the selling price to its corresponding buying price. In case that there some special case that does not yield a match, they will be passed to the Intraday system. This service allows market attendees to engage in live auction of their electricity. Once the transaction has been settled, every transaction that is passed through the market must be approved and archived. This is achieved through the Clearing and Settlement Service (CASS). Additionally, both buyer and seller will be notified of the completed transaction. However, there are constant changes to order electricity amounts and prices. Market participants must be informed of such changes to take appropriate actions. Such notifications are provided by the REMIT UMM service (Urgent Market Messaging). This webpage notifies the market of all the unexpected changes in the consumption and transmission of electricity.

Currently, Nord Pool is a Nominated Electricity Market Operator (NEMO) in Europe's market with 380 companies from 20 countries trade on the markets in the Nordic and Baltic regions, in Germany and in the UK. The largest challenge for the company is having every transaction settled with the correct price at the correct time and with complete security. However, with such a large market, it is difficult for the current infrastructure to cope with such a large volume of data. Additionally, the system was not initially designed for such amount of work. The company has steadily increased their services and expand their markets without major updates to the infrastructure.

Nord Pool recently joined the Central West Europe's market. This is an enormous step forward for the company as their market is expanding exponentially. However, this also means the number of companies using Nord Pool's services could be quadrupled. Therefore, an

infrastructure change is needed. To stay ahead of the competition, product's performance, availability, and scalability are key components. Thus, alternatives scaling approaches should be considered. Service Fabric is one of such alternatives with numerous advantages. After consideration, the company has chosen Microsoft Azure Service Fabric to be evaluated as a feasible solution. However, clear evidence in the feasibility of Service Fabric is needed. Therefore, this experiment is commenced.

As mentioned, Nord Pool offers a variety of services whilst managing almost 400 companies in their market. In addition to processing data, the company must also maintain the data security and its Service Level Agreements (SLAs) with its customer. This requires the company's infrastructure to cope with an enormous amount of work.

Currently, within the company's system implementation, each service is considered by itself a process. Each of these processes will take the output of others as its own input. This implementation helps the data processing to be more effective as it will undergo layers of handling. Additionally, this operation efficiently keeps track of the data flow within the organization. For Nord Pool, this information is invaluable to improving their services.

Although the current operation implementation helps process data efficiently, this present a critical overhead upon the system. With the output of one service being the input of the next, every service is heavily dependent on one another. This dependency creates numerous disadvantages for development work such as maintenance, improvement, multiple points of failure. Additionally, within this system design, message processing is delayed as the messages are passed through a single instance of the service. This leads to many messages in queues and the service becomes "bottlenecked".

The current operation implementation also negatively impacts the company open SLA. With such dependencies between systems, multiple points of failure are present. In the scenario that one system fails, many others can be negatively affected if not shutting down altogether. This scenario can be caused not only by disaster (security breach, system malfunction) but also by introduction of "breaking changes". Any change to the system that prevents the compatibility with previous versions is considered a "breaking change" (Xavier et al 2017, 139). With software development where new features are constantly added, it is difficult to avoid "breaking changes". Therefore, in the case of Nord Pool software development operation, when one system presents a "breaking change", others system can be critically affected as they can be rendered cripple due to the incompatibility between versions. This causes a large overhead invested into just maintaining the system with every version update and "breaking change" presented. To keep up with their SLAs, it is evidenced that Nord Pool requires an improvement in their infrastructure and operation as soon as possible.

1.2 About the project

As Nord Pool is constantly expanding its services to more international customers, the infrastructure needs constant improvement to not only meet demand from new customer but also current consumer. In addition, with the range of services Nord Pool provides, the amount of strain placed on the current infrastructure has proved to be too much. If the company is to continue its services and expansion, a drastic improve is much needed. This project was commenced with the goal to provide Nord Pool with evidence supporting a solution: 'Microsoft Azure Service Fabric'.

As mentioned before, the decision to choose 'Microsoft Azure Service Fabric' or 'Service Fabric' was established prior to the commence of this project. Therefore, detailed explanation on the reason this solution was chosen will not be discussed in this thesis report. However, an important fact which contributed to 'Service Fabric' being chosen should be mentioned. The company's infrastructure is tightly integrated with Microsoft services. Therefore, within this project, other Microsoft's solution is utilized to collect data for analysis.

For 'Azure Service Fabric' to be considered a viable solution, the sample product and its underlying infrastructure must meet certain criteria set by the company. The sample application must carry out basic functionality of the chosen service. This requirement is crucial as it indicates that the current service can be migrated. Secondly, the application must be available during different hazardous scenarios. These situations should reflect threats to the application in production environment such as high workload, networking failure. Additionally, the application must be maintainable and easy to implement new updates to it.

During this project, multiple members from different teams were involved. This project requires different components which only different team members had the authority to provide. The separation of authority was made since the company followed 'least-privilege' security principle (See Appendix). Although multiple members were involved, the request for each component was made through providing tickets to teams. The names of the members of the teams responsible for providing the components will not be mentioned in this project report. This decision was made to protect the identity of the team members along with securing the corporate structure of Nord Pool. However, the team that provided certain components will be briefly mentioned to keep the integrity of this report.

Although initial research into the Service Fabric platform yields positive evidence supporting the product. Before making clear steps onto changing the software or the infrastructure underneath, the company needs to ensure that the current software can be re-developed and compatible with Service Fabric as this platform has specific requirements on how application is built. Therefore, this project object is to build a Minimum Viable Product (MVP) instance of one of the current running products. If the MVP can be successfully built, it will indicate that

Microsoft Azure Service Fabric is a profound platform not only migrating the current products but also future development.

2 Research and development methodology

According to the Project Management Institute in 2017, every project is a unique timeline created to either solve a problem or create a product. To achieve such goals, in every project, the participants must fully grasp the specific requirements for the unique problem at hand. Therefore, it is essential that every project must establish a profound and robust knowledge base of the research and development methodology. This methodology will be utilized throughout the project, acting as the guideline to ensure that the project participants completely understand what needs to be done and what does not.

There are numerous advantages that a solid research methodology brings forth. The webpage 'bbamantra.com', which specializes in collecting quality documentation and papers on project management, has pointed some keys advantages of a well-established research methodology. For researchers, the research method usually assists in developing logical thinking and organization throughout the project. Additionally, within various situations and stages, the research method could act as a guideline for decision making and keeping track of the progress.

For this project, a well-defined research methodology was established as the initial step. This ensure that the researcher had full understanding of the project scope and on which basis should the project be executed.

2.1 Project hypothesis and scope

At the beginning of every project, it is imperative that a project purpose and scope are established. According to Rouse in 2018, a project scope or scope statement is the documentation which defines the boundaries the project, establishes the duties of each members involved within the project and provides the procedure for which the project will be carried out. By having a clearly defined project scope, numerous advantages are achieved. Project participants would fully understand their roles and level of participation. Requirements for the final product are established. Additionally, the resource needed for the project to complete can be estimated. For complex project, a clear project scope can also prevent unnecessary expansion further than established vision.

However, a key distinction between project scope and product should be noted as these concepts can be easily misinterpreted for one another. The product scope refers to set of condition which the final product must meet (Villanova University 2020). The characteristic

regarding how a product should function or a how a process should be carried out are defined in the product scope. Contrast to product scope, the project scope primarily focuses on the procedure needed to achieve such product. By understanding the difference between these two concepts, project participants have clear awareness on what the project aim to achieve and what needs to be executed.

This project will preserve the core intention of its purpose to act as a proof of concept supporting the feasibility of using Service Fabric in development. This project will also keep to the core of the research strategy which it is based on – pilot experiment. Therefore, the research scope should be kept small. This project does not aim to actively improve the client company's infrastructure. However, this thesis does aim to provide evidence supporting a solution to such challenge.

In this project, there were 2 main parties involved which are the Clearance and Settlement Service team and the author of this report. However, the roles of each team and the team members were not equal. Since the CASS (acronym for Clearing and Settlement Service) was chosen to be used as a model to build the sample instance, the team responding for the service advised on designing and developing the sample. The main responsibility of the project was carried out by the author of this paper. The author acted as the main developer of the project with the responsibility of researching the requirements, gathering needed information, designing, and developing the sample applications, testing the application functions and the underlying infrastructure.

2.2 Research Strategy – Pilot Experiment

The research strategy could be considered the backbone of the research methodology. As mention above, during a project, the research methodology is the general guideline to keep researchers on track and the project manageable. At the core, the research strategy acts as the systematic instruction on how the research should be carried out along with the framework in which the research should follow and be conducted. On March 2014, in an article written for Mackenzie Corp on Marketing Research, Dinnen J wrote: "A Research Strategy is a step-by-step plan of action that gives direction to your thoughts and efforts, enabling you to conduct research systematically and on schedule to produce quality results and detailed reporting."

According to the Open University, there are 6 main research strategy with different characteristics. The 'Case Study' research strategy focuses on an in-depth investigation of a single case or a small number of cases. In the end, the investigation of such cases should yield significant understanding of the scenario and detect pattern for future cases. One of the most used methods for data gathering is 'Qualitative Interviews'. With this method, researchers conduct conversation with interviewees, within a focus group, to find the overall quality of

the product in the opinion of participants. For business research, another widely used method is ‘Quantitative survey’. These surveys are usually carried out on a high number of participants with question targeting to deprive detailed insights from respondents. The end goal of this method is to gather numerical data to determine statistical results. For organization with intention to implement new changes, an action-oriented research method would be required. This refers to practical business research which is directed towards a change or the production of recommendations for change.

For this project, the most suitable research strategy to be utilized as the guideline would be an action-oriented research. More specifically, the ‘pilot experiment’ is the most suitable strategy. According to Wikipedia contributors, A pilot study, pilot project, pilot test, or pilot experiment is a small-scale preliminary study conducted in order to evaluate feasibility, duration, cost, adverse events, and improve upon the study design prior to performance of a full-scale research project. Pilot experiments are frequently carried out before large-scale quantitative research, to avoid time and investment being used on an inadequately designed project. A pilot study is usually carried out on members of the relevant population. A pilot study is often used to test the design of the full-scale experiment which then can be adjusted. It is a potentially valuable insight and, should anything be missing in the pilot study, it can be added to the full-scale experiment to improve the chances of a clear outcome.

2.3 Project management framework – Waterfall

Once a research strategy has been established to act as the backbone and general guide for development, the next step is to draw of a specific timeline to follow.

Simply put, waterfall project management is a sequential, linear process of project management. It consists of several discrete phases. No phase begins until the prior phase is complete, and each phase’s completion is terminal—waterfall management does not allow you to return to a previous phase. Each step of this model acts as the prerequisites for the succeeding phase. Therefore, each phase is carefully designed and carried out. The Waterfall framework has numerous advantages, with its straight forwardness being a major factor in the decision to utilize it for this project. Although, in 2016, Powell-Morse had pointed out a critical disadvantage of this method which is its inability to allow developers to return to a previous stage without starting from the initial stage. This flaw put strain on the importance of careful planning and execution of every stage, particularly the design stage. However, this framework is the perfect match for the project at hand. At its core, this pilot experiment had the end goal of building a minimum vial product which meet basics necessities. The final product needed does not have any heavy requirements on its functionality. With such a narrow scope and minimum product specification, it is easy to create a set of requirements for the product and its implementation steps. Therefore, with the waterfall framework, developers can focus

on gather information and building the product as this framework discourage adding further changes to the requirements.

2.4 Development framework

Having a framework to follow can aid a developer significantly. With a framework, clear steps and goals can be defined and executed. Since this project final product is a software, the most suitable framework would be the Software Development Life Cycle (SDLC). According to the website Stackify (2020), SDLC is considered process which produces software with the highest quality and lowest cost in the shortest timespan. The website also stated that this framework has specific predefined steps and phases for developers to execute.

Firstly, the requirements for the final product need to be established. The end goal of this initial stage is for researchers to have a chance to analyse and written down the specification of the desired final product. This stage also acts as a foundation for all further research and development within the project. As later steps aim to fulfil the requirements already set in this phase.

The second stage is the ‘Analysis’ stage. Researcher can now gather information about the current system, resources, and personnel along with missing resources to complete the project. With these elements evaluated, project participants can clearly list missing elements and act upon such situation and eliminating any unnecessary hindrance with following stages.

In the ‘Design’ phase, outlines of the actual procedure needed to complete the product are drawn up. Typical topics of concern in this step include technical aspect such as the programming language to use, securities measures, data layers, etc. However, sometimes the overall design or blueprint of the product is drawn up in this stage. The blueprint could also act as the guide and target for developers to build their applications.

Utilizing the outlines from the previous steps, developers will start building the final products. In this ‘Implementation’ stage, no changes in the design should be made, developers should only focus on applying established logic and procedure. Any changes that are mandatory will force the project to return the initial stage. Therefore, only in case a major design flaw was found should changes be made.

After the product is developed, the ‘Test’ phase can begin. The final product would go under testing according to business requirement within this step. For every product, tests will be developed depending on the number of features, functionalities, and its overall requirements.

Finally, after the product has passed all the necessary testing, it is packaged and deployed into the live environment where customers can gain access to its services. During this 'Operation' stage, the product will be maintained until the end of its life cycle.

Using the Waterfall project management framework along with SDLC, a project timeline with executable phases could be drawn up. This timeline, as illustrated in figure 1, helped the keep the development progress focused.

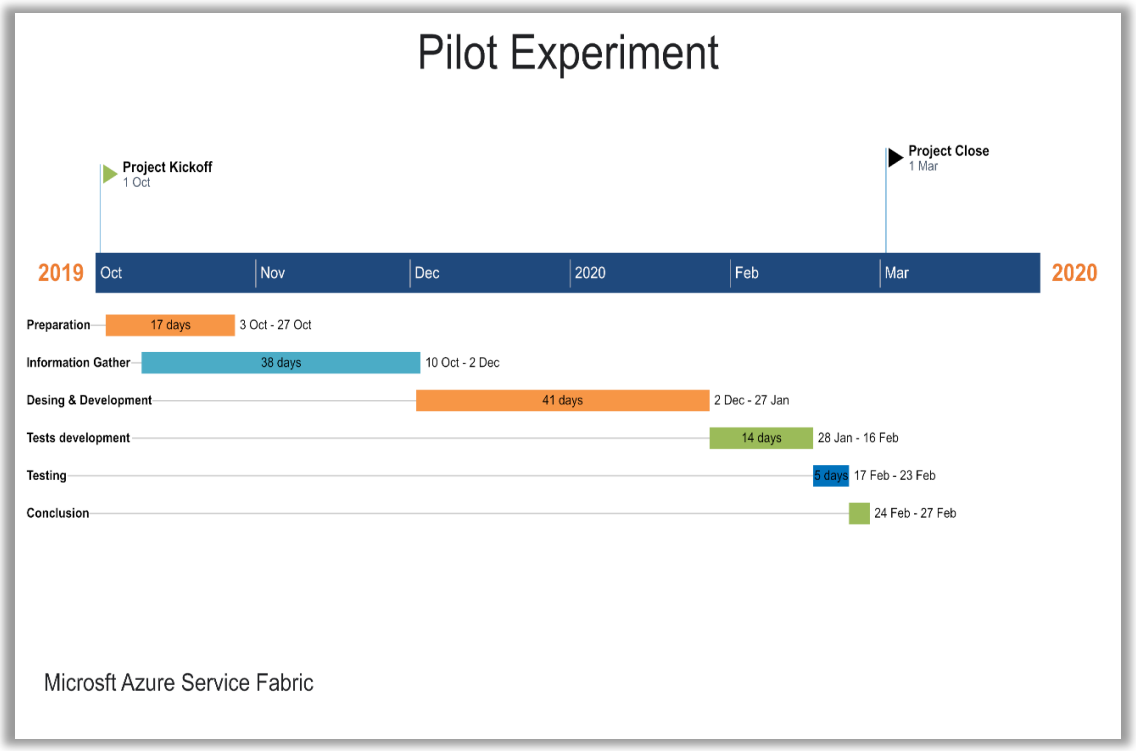


Figure 1: Project Timeline Gantt chart

The Gantt chart (See Appendix), in figure 1, have helped visualized the steps to which the project is executed. The Gantt chart created for this project which helps developers keeps track of how long their progress was taking. As the final product of this project is only a Minimum Viable Product (MVP), it is not deployed into the live environment for customers. Therefore, its life cycle will only exist within the time of the project and the Operation stage – where the product is maintained – will be omitted.

3 Theoretical frameworks

Within the actual implementation of the project, this section is the result of the design stage, where information about the product requirement and technical knowledge is acquired.

However, this section had been separated from the implementation for report purpose. Therefore, it will not be repeated in the implementation stage.

3.1 Software/Product Requirement

To meet the project requirement, the final product would be a working Minimum Viable Product instance of one of the running products. According to Wikipedia contributors, a MVP is a version of a product with just enough features to satisfy early customers and provide feedback for future product development. This MVP should only contain the bare-bones functionalities of the large-scale product without any added features.

Although the final product in this project is simple and contain little functionality, it can provide numerous positive points towards the hypothesis. Firstly, if a simple product can be built using the Service Fabric platform, it indicates that the complex product of the company can be migrated. Secondly, there could be almost zero downtime will the services are migrated as the application was developed independent of any current infrastructure support. Additionally, the final product and its development process will act as the foundation for future study into migration research of current application and infrastructure.

3.2 Microsoft Azure – Service Fabric

The main purpose of this study is to provide concrete evidence that a software can be built on the Microsoft Service Fabric framework to replace the current running application. However, before developing any product, it is imperative that a thorough understanding of the tool is established. By gathering such knowledge, fatal pitfalls can be avoided such as misdirection in development, lost of valuable research time, etc.

3.2.1 Cloud computing

To keep a sharp competitive edge, businesses actively seek out new ways of bringing better product to the customer with less and less time to market. One mean of achieving this goal is through utilizing the available functionalities of ‘cloud computing’.

An initial search on Google with the phrase: ‘Cloud computing’ will yield numerous resources on all aspects of the term. A further search into the topic, through the search engine ‘Google Scholar’, will also yield an abundant of research covering many aspects of the field. The result from Google Scholar also dates to 2008. A good example is the paper: “What cloud computing really means.” written by Knorr E and Gruman G in 2008. The authors had explained cloud in the term “cloud computing” is just an expression to represent the internet. This vague expression has led to many different definitions, from many vendors, of what the term means.

According to Microsoft, 'cloud computing' refers to the readily available computing services which company can utilized according to their needs. These resources include any computing functionality that a company would need to deploy their application such as: processing power, networking, storage, data security, web servers, etc. Additionally, large cloud providers, such as Microsoft, has servers and infrastructure expanding across the world. This allows business to build and deploy their application, products, and services to every desirable region. As Nord Pool market covers a large portion of the Baltic and Nordic region along with further development into the Central Europe market, it is only reasonable for the company to utilize this aspect of cloud computing.

3.2.2 Containerized application

Traditionally, a single web application would require the numerous components as illustrated in figure 2. First, a physical machine – typically a server – would be required to host the application. An appropriate Operating System is installed in the host. As different application has different requirement, the OS can vary from Linux to Unix to Windows. Depending on the application, a database could also be required. Thirdly, a web/app server is also needed to be installed in the host machine. This server ensure that the application can be connect through the internet. Finally, the application is built and installed upon all the above components.

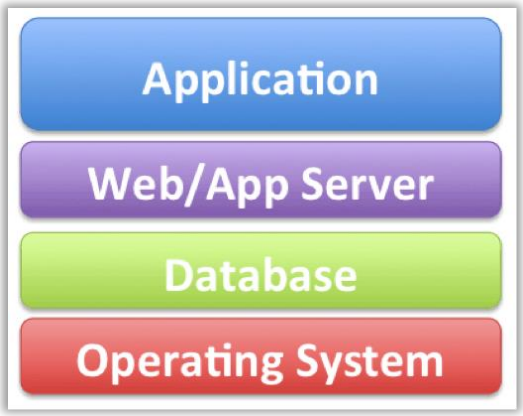


Figure 2 Traditional application topology

Although this method is effective for building single application, this approach is no longer effective in building complex application. This approach brings fort numerus disadvantage. The most significant flaw of this topology is its single point of failure. Since, the application is hosted on a single machine, it is susceptible to any threats towards the physical host. If the host fail, the only running instance of the application fails as well. If the instance fails for any reason beside host failure, the application is inaccessible for customers. Additionally, errors arising from the application could be from multiple sources including hardware failure, OS

malfunction, incompatibility of application and OS. With such numbers of sources of failure, it is difficult to investigate and find the exact failure whenever such situation comes.

Since each instance of the application would require a physical machine to run, business would need to invest in not only the physical hardware but also the overhead of their maintenance services. Another flaw of this method is its service latency. Since the application is hosted as specific geological location, customers will have a much longer latency when accessing the service from further geological locations. To combat this issue, companies can deploy multiple sites which host servers for their application. However, this method will raise more cost as much hardware and maintenance is needed.

To mitigate the overhead cost of the traditional method of application deployment, virtual machines are use. Multiple instances of the application can be running on a single machine by partitioning its resources with multiple virtual machine (See Appendix). However, in this method, the overhead is reduced but still withstand. To permanently fix this issue, companies can utilize application containerization technology. Unlike previous methods of application deployment, application containerization provides multiple advantages over its predecessors. Rouse M, in 2019, has explained that this method of deployment differs by the fact that application require neither physical machine nor virtual machine to be built upon. Containerized application only utilizes the minimum amount resources to which it needs to run. Application containerization is fundamentally different from that of virtualization, especially in that it does not require a hypervisor. Containers also do not run their own individual instances of the operating system. A container houses the application code along with all its dependencies (bins, libraries, etc.). A container orchestration software tool (e.g. Docker Engine, Microsoft Azure Service Fabric) sits between the containers and the host operating system, and each container on the machine accesses a shared host kernel instead of running its own operating system as virtual machines do. Illustrated in figure 4, containerized application takes up less resources, compared to virtual machines, while running more applications.

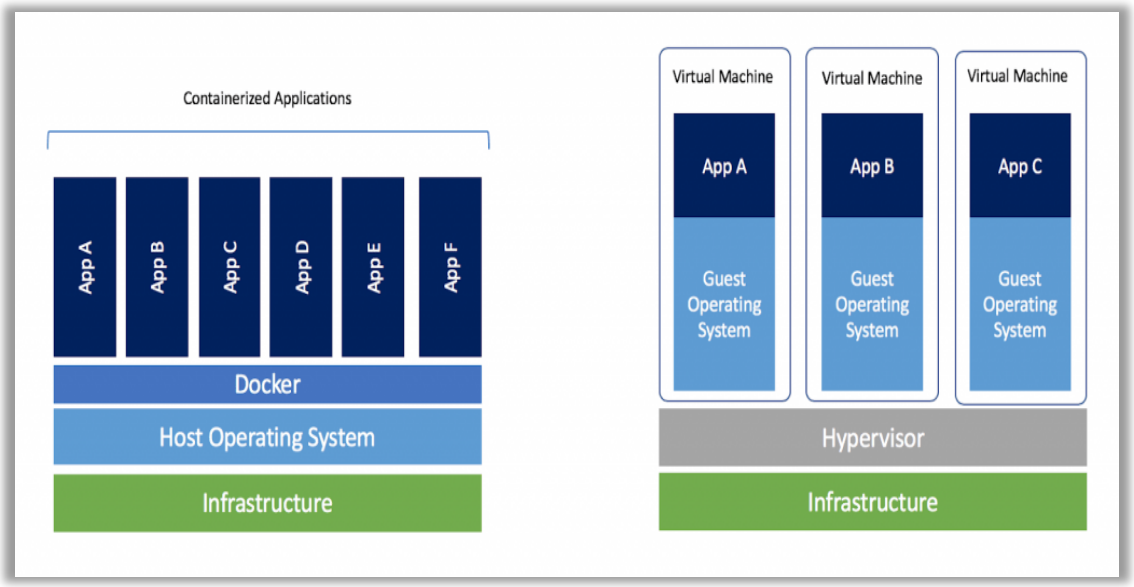


Figure 3: Virtual machine architecture and container architecture comparison

After an application has been containerized, they can be deployed on specific nodes. These nodes are typically a physical server partitioned from a cloud vendor in desired location across the world. By utilizing the readily available resource of cloud computing, web servers and storage, along with application containerization, Nord Pool can build their products and deploy them to any market in any geological location.

3.2.3 Container Orchestration – Microsoft Azure Service Fabric

According to Avi Networks in 2020, 'Container orchestration' is the automation of all aspects of coordinating and managing containers. Container orchestration is focused on managing the life cycle of containers and their dynamic environments. Container orchestration works with tools like Kubernetes and Docker Swarm. Configurations files tell the container orchestration tool how to network between containers and where to store logs. The orchestration tool also schedules deployment of containers into clusters and determines the best host for the container. After a host is decided, the orchestration tool manages the lifecycle of the container based on predetermined specifications. Container orchestration tools work in any environment that runs containers. Although, Microsoft Azure Service Fabric can act as an Orchestration tools, how application interact with this tool differs from Kubernetes and Docker Swarm.

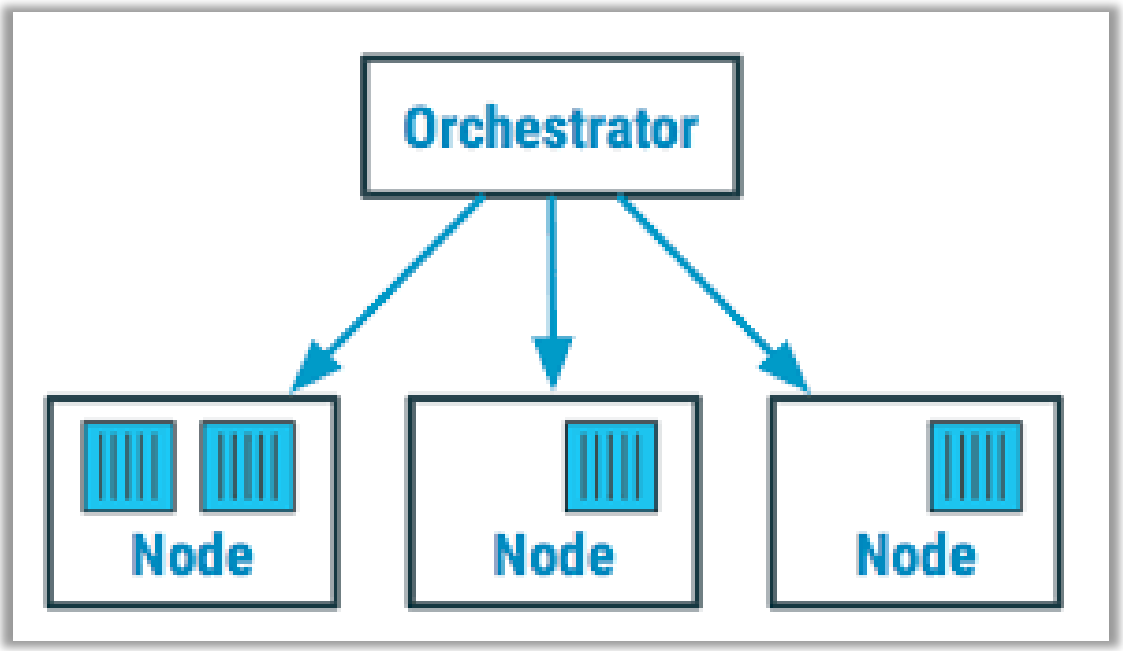


Figure 3 Simple cluster orchestrator topology

Illustrated in figure 3, showed the simple hierarchy of Orchestration and nodes within containerized application topology. Container orchestration plays a vital role in the application life cycle. The orchestrator is responsible for ensuring the availability of the application. Depending on the metrics set on resource utilized by each node, the orchestrator (e.g. Kubernetes, Service Fabric) will create or terminate instances of the running application. With chaotic situation such as nodes failure or network failure, the orchestrator is also responsible for stabilizing the cluster add or removing nodes and re-routing traffic to the correct instance.

3.3 Web framework - ASP.NET CORE

Acting as a development platform, Service Fabric has certain requirements for how applications are developed. One critical requirement the use of ASP.NET CORE web applications framework. These frameworks provide a standard way to build and deploy applications while reducing overhead in development. According to Mircosoft, web applications built on Service Fabric platform is required to use ASP.NET Core web framework. This framework provides all the necessary libraries for application development.

Stated by GoodFirms in 2020, a web application framework is a code library that makes web development quicker and easier by giving basic patterns for building reliable, scalable, and maintainable web applications. Web frameworks exist to make it easier for the developer to make a web application. These are shortcuts that can prevent otherwise overwhelming and repetitive code. In example, developers may need a function code to handle data validation

for the web application. To prevent having to re-write that from scratch each time developers create a web service; the framework will have it already available.

ASP.NET Core is the new version of the ASP.NET web application framework mainly targeted to run on .NET Core platform. ASP.NET Core is a free, open-source, and cross-platform framework for building cloud-based applications, such as web apps, IoT apps, and mobile backends. It is designed to run on the cloud as well as on-premises. Same as .NET Core, it was architected modular with little overhead, and then other more advanced features can be added as NuGet packages as per application requirement. This results in high performance, require less memory, less deployment size, and easy to maintain.

Depending on the purpose of an application or service, the method which it uses to handle data differs from each other. According to Sturtevant J in 2016, there are currently two main service types which can be built using Service Fabric which are Stateful service and Stateless service. However, another recent service type has emerged as an effective framework for application which is Actor service.

Stateful service stores its state - data representing the application - inside of Service Fabric cluster. The state is shared by all nodes in a partition. State changes are replicated from the primary node to secondary nodes automatically in a transactional manner. Stateless service (formerly Stateless reliable services) stores its state in an external source. Almost every service has its state. It could be a list of customers or a result of some computation. In case of stateless service, the state is stored in SQL Database, Azure Storage or DocumentDB. Within these two models, the Orchestrator will be handling creating and terminating nodes once certain metrics are met. The orchestrator will also be responsible for handling backup and recovery in the situation of cluster failure.

Actor Service, based on stateful service, is a service used in the actor model. In this model, an actor is the smallest unit of computation which can be the application or the nodes. Actors will only send and receive messages from other actors. Once a message is received, the actor responses in multiple ways such as make internal decision, create more actors, determine e the response, etc. In contrast with the other models, the orchestrator is not responsible for actors (nodes) creation or deletion. Since actors only communicate with each other, they will determine the numbers of running nodes. The orchestrator, however, will still be responsible for cluster recovery.

3.4 Message Broker (Publish – Subscribe Model)

In 2018, Geernick V had explained the functionality of a message broker in a simple definition. He stated that such service is “a program that translates a message to a formal messaging protocol of the sender, to the formal messaging protocol of the receiver” as illustrated in

figure 6. This service is required whenever a request is made to a system. A simple example for this scenario is when a user request data from certain website. A request is made through and API sending a message to the Broker which will find the correct service that has the corresponding response. Without a broker, messages and request may not be received by the correct service.

According to Wikipedia contributors, Publish–Subscribe is a sibling of the message queue paradigm and is typically one part of a larger message-oriented middleware system. Most messaging systems support both the pub/sub and message queue models in their API, e.g. Java Message Service (JMS). Although this pattern provides greater network scalability and a more dynamic network topology, with a resulting decreased flexibility to modify the publisher and the structure of the published data.

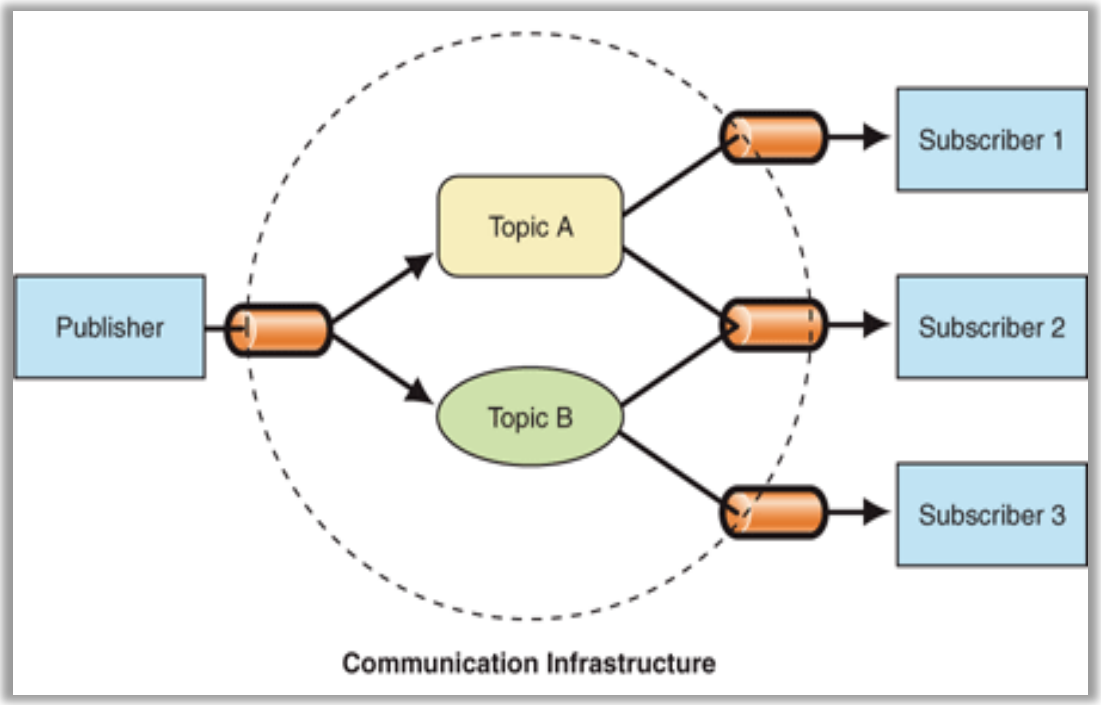


Figure 4 Message Broker topology

In figure 4, users are considered publishers as they send requests to the system. The message broker is represented by the dashed circle. Finally, services which have correct response to each type of message are considered subscribers.

3.5 SignalR

ASP.NET SignalR is a library for ASP.NET developers that simplifies the process of adding real-time web functionality to applications (Microsoft 2019). Real-time web functionality refers to the ability of having server code publish content to connected clients instantly. With this

tool, clients will constantly receive data without having to create requests. SignalR can be used to add "real-time" web functionality to applications and modify services to enhance user experience. Chat message applications are an example of this tool. Any time a user refreshes a web page to see new data, or the page implements long polling to retrieve new data, SignalR can be the most suitable tools for building such services. Examples include dashboards and monitoring applications, collaborative applications such as simultaneous editing of documents, job progress updates, and real-time forms. SignalR also enables modern types of web applications that require high frequency updates from the server, for example, real-time multi-player gaming.

3.6 Infrastructure Testing Tools

3.6.1 Load Testing – Nbomber testing tool

According to Wikipedia contributors, the term ‘Load testing’ generally refers to the practice of replicating the expected usage of a software program by simulating multiple users accessing the program concurrently. As such, this testing is most relevant for multi-user systems; often one built using a client/server model, such as web servers.

Load and performance testing analyses software intended for a multi-user audience by subjecting the software to large numbers of virtual and live users along with monitoring performance measurements under different intensity. Load and performance testing are usually conducted in a test environment identical to the production environment before the software system is permitted to be deployed.

NBomber is a modern and flexible load testing framework for Pull and Push scenarios, designed to test any system regardless a protocol (HTTP, WebSockets, AMQP, etc.) or a semantic model (Pull/Push). This tool can be easily designed to replicate how normal customers would interact with the application. Additionally, this tool was programmed to send a large amount of request to test the limit of the system. With this test, the cluster can be clearly seen responding to a replica of production environment. Any failure in this test will suggest that the underlying infrastructure is clearly not ready for the application to be deployed upon.

3.6.2 Chaos Test

The aim of this test is to ensure the application resilience in case of infrastructure failure. Even though the application could have gone through intensive test to ensure its functionality or availably, this test is still a vital part which the cluster must pass before the application can be deployed.

The chaos test aims at the scenario where the application has been running on a large enough number of machines. In this state, the cluster could be vulnerable to unexpected situation

regarding nodes failures such as nodes sudden termination or nodes networking errors. By actively testing these situations, engineers are more prepared for the application deployment to production environment.

4 Implementation

Using the knowledge gathered in the theoretical framework, the implementation phase was carried out. This section documents the procedure and designs that were executed.

4.1 Preparation

Before development can begin, the infrastructure hosting the application needs to be in place. This task was completed by setting up the local cluster which had enough partitioned resources to reflect a production cluster as shown in figure 5. There were 5 nodes for the application to be hosted. Each node had its own internal storage for stateful service development. Since stateless services required an external storage, one was also setup outside the cluster. Security measure against external threat must also be in place along with authentication method. Additionally, for testing purpose, monitoring of the application was also integrated with the application health Microsoft Azure Application Insight.

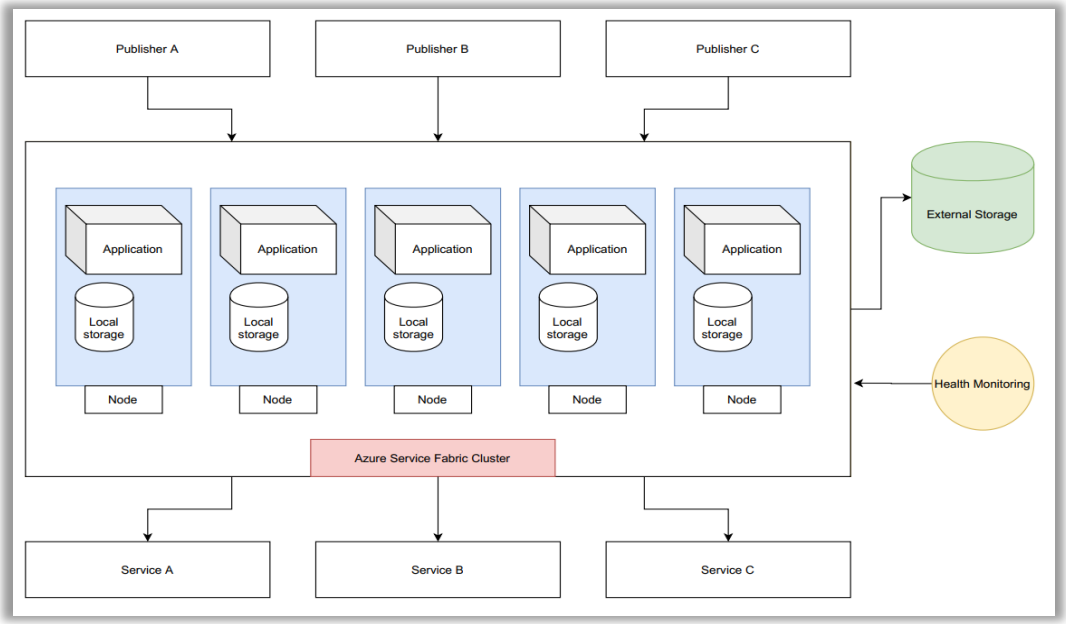


Figure 5 Application local cluster

4.2 Choosing the service to experiment

The CASS (acronym for Clearing and Settlement Service) was chosen for this project as it is an importance part of the business along with the fact that it is in urgent need of improvement. As mentioned above, this service handles every transaction that passes through the Nord Pool market. This includes transactions from over hundreds of companies across the Nordic and

Baltic region. This service plays a vital role in Nord Pool's role as market organizer. With this service, customer can view the entire lifecycle of a transaction from trade to invoice. These invoices can also be accessed through a web interface. Additionally, the CASS service helps customer be aware of their require collateral and make crucial adjustment to collateral cost.

Currently, the service is running on an infrastructure that complete the job, however, it has become a single point of entry for the transactions. This has caused the transaction to be queued up before being processed one by one. With the company extending its services to larger markets, the number of transactions will only increase in volume resulting in longer queue time. Since the CASS is a vital service, this issue must be fix or mitigated as soon as possible to retain customer's satisfaction and the company's SLAs.

4.3 Information gathering

4.3.1 Product requirement

The main functionality of the CASS system follows a close resemblance to the Message Broker (Publish-Subscribe model). Each transaction (a message), from different companies (publishers), is passed to the service to be sorted. Once the transaction is processed, it is sent to the correct services (subscriber) for further processing. Which service the message needs to send to depends on the information in the message itself. One major requirement for Nord Pool products is that the information published must reflect real-time condition of the market. Any delayed information update can result in decision being made based on false data. This could cause customers millions and jeopardizing Nord Pool's open SLA. It is vital that this component is included in the MVP as without it, the company product will not meet customer's need. With such requirements, the MVP software needs to have some main functions of the current application. The MVP need to receive transactions (messages) from users. Afterwards, the messages need to be sorted and send to the corresponding services. Finally, the status of the transaction is required to be reported in real time.

4.3.2 Development on Microsoft Azure Service Fabric

With the other orchestrator, developers can choose to develop application using templates and library of their choice. Application can be developed in any fashion from different teams and passed to DevOps to be deployed. However, this is not the case with Azure Service Fabric. Any application developed on this platform must specify that it is a 'Service Fabric Application' through Visual Studio IDE (See Appendix) graphical user interface and use one of platform templates as shown in figure 8. The web applications built on the platform are required to be built using the ASP.NET Core framework. This will require developers to understand how to build products on such framework along with familiarizing with the templates. However, by following these procedures, the application will be ensured to have compatibility with the cluster and the orchestrator. Additionally, since Service Fabric will handle both containerizing

and orchestrating the cluster, it reduces complication significantly compared to managing separate tools for each task. As a cluster orchestrator, Azure Service Fabric will also be responsible for managing the cluster according to the metrics set. Therefore, developers will only need to focus on building products and the DevOps team will only need to focus on infrastructure maintenance while Service Fabric handles application health and availability.

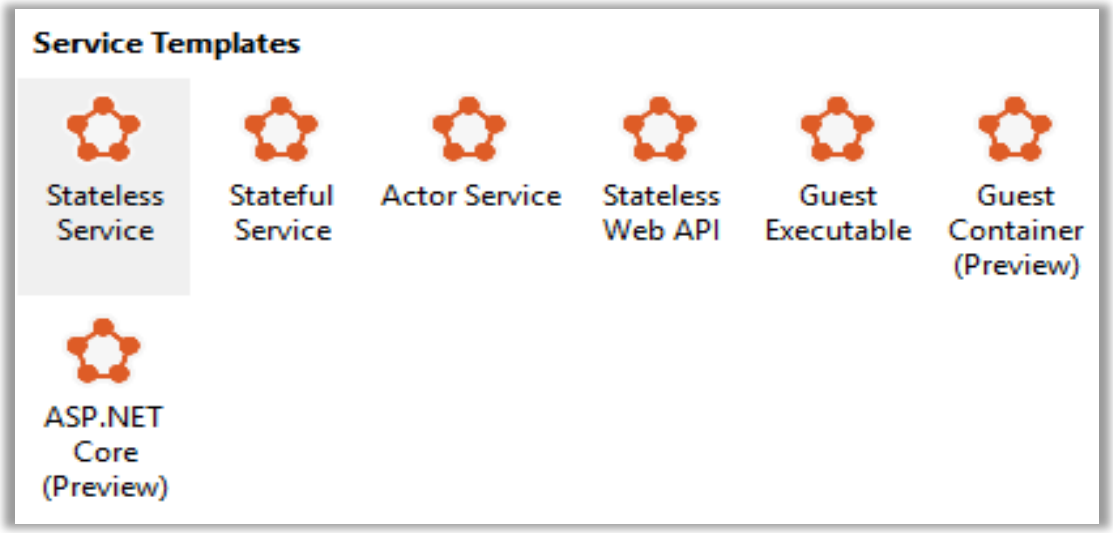


Figure 6 Microsoft Azure Service Fabric service templates

4.3.3 Testing requirement

Within this project, Azure Service Fabric was not only tested for its functionality as a platform for development but also a cluster orchestrator.

Firstly, the MVP needed to be tested to ensure it has basic functions of the base product. This would require a sample operation of sending messages, receiving messages. Afterwards, the message needed to be sorted and delivered to the correct service. Finally, the status of the message must be reported in real-time. If this test is successful, its result will indicate that Service Fabric could act as a development platform for future migration.

Secondly, the cluster infrastructure is tested against the Chaos test and the Load test. With the first case, Microsoft offers its own framework and built-in Service Fabric tool for developing such test. Developers would only need to specify the endpoint of the service and the intensity of the ‘chaos’ within a PowerShell script. More specifically, this script can carry out multiple disastrous scenarios include single nodes failure, multiple nodes failure, nodes networking error. Throughout this test, the orchestrator must response by keeping the service available and accessible. The application health can be monitored through the web interface of the monitoring tool ‘Microsoft Azure Application Insight’. If the application maintains

available throughout the test, it will provide positives evidence supporting the orchestration ability of Service Fabric.

Finally, the application must endure a heavy workload with a high number of requests. The NBomber tool can be configured complete such task. Like the Chaos test tool, NBomber only requires the service endpoint as the target. However, within the NBomber test, specific scenario, steps, and type of request are needed. The result of this test can be view through the built-in graphical user interface. To be considered successful, the percentage of received request must be substantial to the number of requests made. The percentage needed to range from 80% to 100% for the test to be successful. If the percentage falls under 60% then the infrastructure has fallen short of expectation and yields negative results against Service Fabric as an orchestrator.

4.4 Design & Product development

The diagram in figure 8 illustrate the workflow of the messages along with the main functions of the MVP message broker. The message broker main functions are situated in the middle square of the diagram. Firstly, the TransactionReceiver was needed. This component managed checking every incoming transaction. It checks the transaction origin, format, and time. A transaction can be rejected if it does not have the correct format or does not contain valid information about the transaction. If a transaction is rejected, the component will update the status of the transaction and pass that status to TransactionStatus which will then send the status to the original publisher. Once, a transaction has passed all the checks, only necessary information is passed to the TransactionClassifier.

Next, the received messages need to be sorted using a TransactionClassifier. This component will receive only need information such as the purpose of the transaction, the collateral amount, etc. This will use this information to parse the transaction to the suitable service. Once a transaction has been moved to the next stage, the status is then parse to the TransactionStatus for publishers to be notify of their transaction status. This component includes a function to receive message, store in message in database, send to message and send status.

Finally, the TransactionStatus need to use SignalR framework. This component frequently sends the status of the publishers' transaction throughout every stage of the process. This includes updating the status of the transaction when it is being processed by the service.

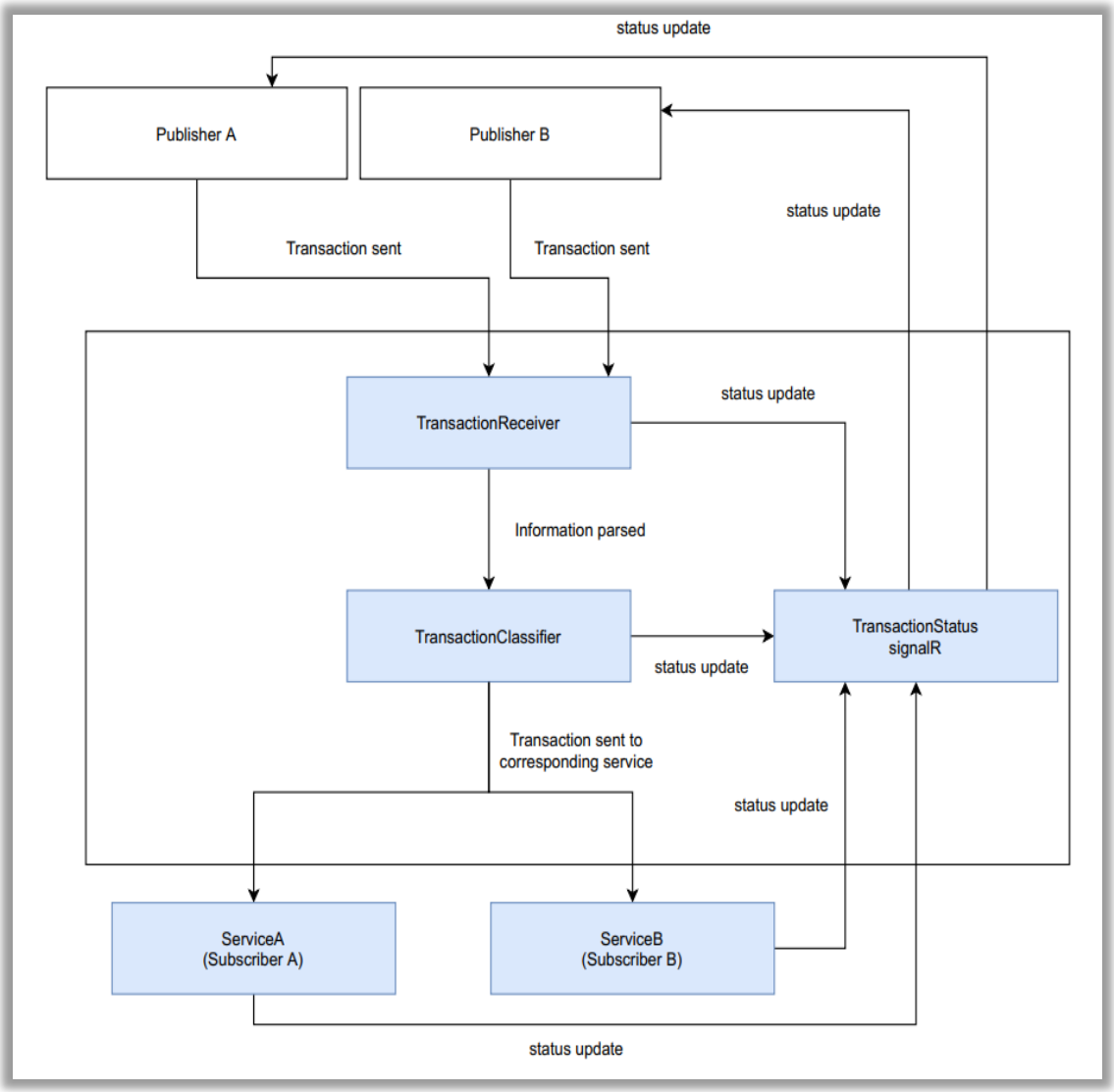


Figure 7 Transaction flow

As part of researching into functionality of Service Fabric, applications was developed following the different service types include stateful, stateless and actor service. Since each service type had different template and infrastructure characteristic, the CASS team took this change in the project to see if the product is compatible with these frameworks. Although the application was developed multiple times, the main components remained unchanged.

Firstly, three sample application needed to be built following the three different services types. Using Service Fabric templates, the applications automatically generate three main components which are the public interface, the listener, and the program. The stateful service is name ServiceA. The stateless service is ServiceB and the actor model was used to create Actor Model Service. For this first phase, only the listener within each service needs modification to receive request. Next the service controllers are needed to act as clients to send

request. These controllers are placed within the API EntryPoints. The simple application topology within this phase is shown in figure 9. Since the stateful/stateless service has different entry points from Actor Model Service, two controllers were needed. These controllers generate and send requests to different entry points.

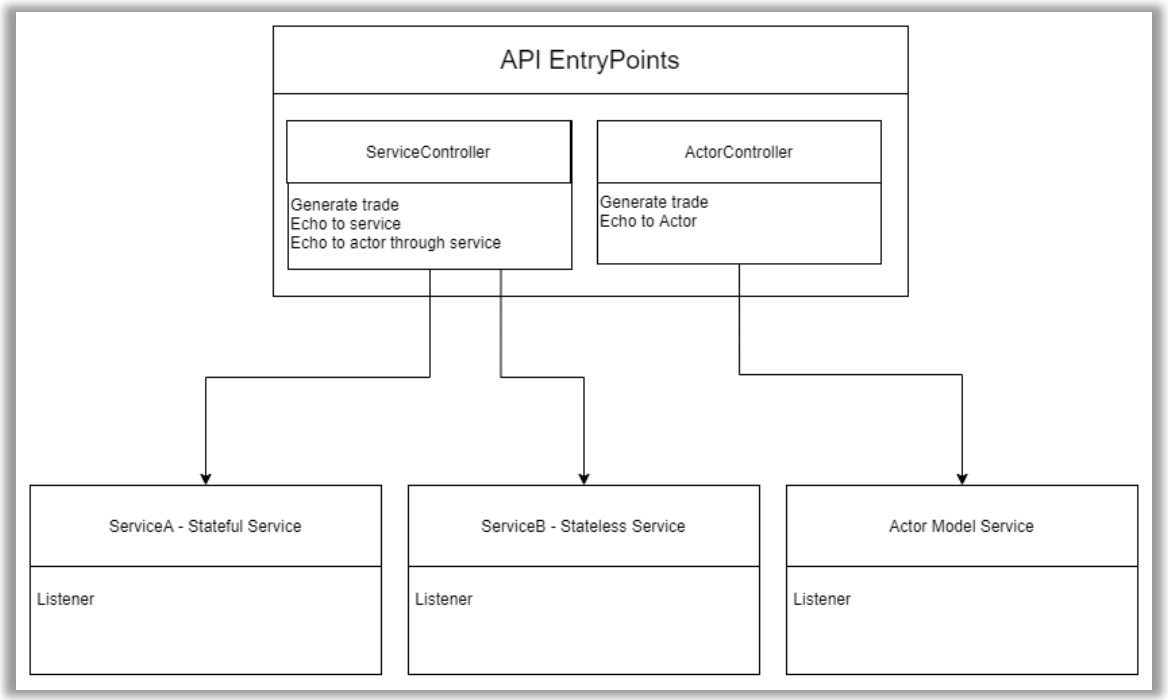


Figure 8 Simple application topology

Once the simple services are made, it is time to create the service will full functionality represented which satisfies requirements and have all components represented in figure 7. ServiceA was chosen to be modified with the complete system illustrated in figure 9.

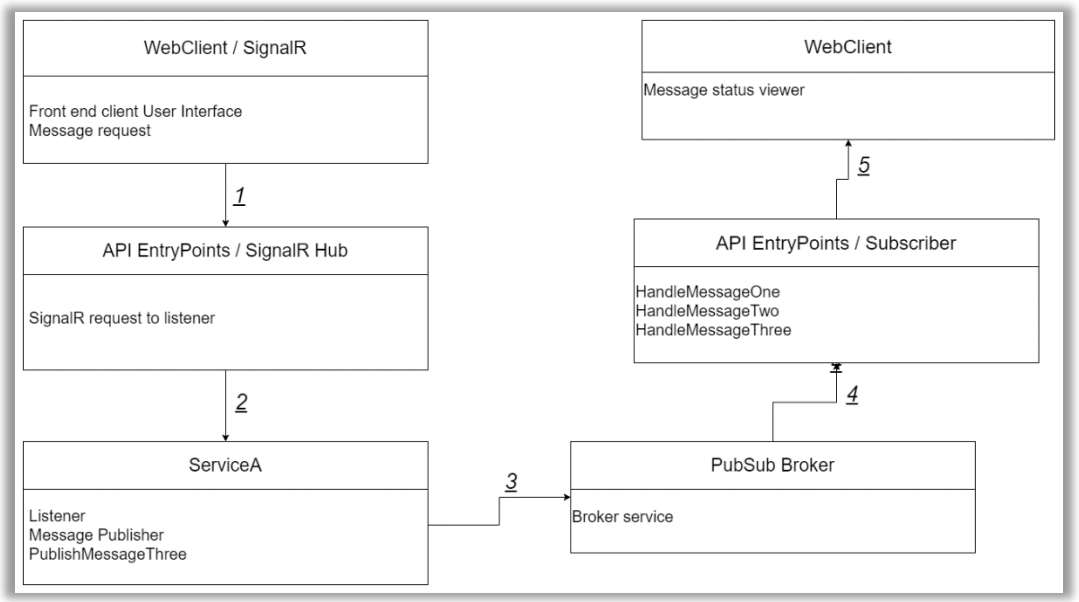


Figure 9 Full functionality service topology

First, a simple WebClient was developed using SignalR web UI framework. This WebClient needed to have a UI which allow 2 type of request to be made. A message needed to be sent to the service and receive the response that the message is processed. This function is called through the “Send Message” button. A message ID needed to be generated from the service and sent back to the UI. The “Save Trade Even” trigures this function. Furthermore, ServiceA will have a new function to publish a message every 5 seconds and the WebClient must show that information.

Secondly, the SignalR Hub was added to the API EntryPoints. In the framework, this Hub is considered the central service for status to be updated. Within this Hub, each button from the WebClient UI triggers a certain request sent to ServiceA listener.

New function to publish the message to the broker was added to ServiceA. Although, in figure 4, the message broker appears to be a stand-alone component, the PubSub model requires publishing and subscribing message function to be implemented within the services. The new publish functions in ServiceA can be seen in the screen shots in figure 9 and 10. Next, the broker was generated by using Service Fabric built in package. However, some modification was needed for it to send the request to its correct subscriber. The subscribers were added to the API EntryPoints as shown in figure 11. Within these functions, SignalR is utilized to send real-time update of the message status to the WebClient.

```
2 references | 1 author, 1 change | 0 exceptions
public async Task<TradeEvent> PublishEchoMessage(TradeEvent tradeEvent)
{
    var savedTradeEvent = await EchoFromService(tradeEvent);
    var brokerClient = new BrokerClient();
    await brokerClient.PublishMessageAsync(new PublishedEchoMessage { Content = $"Echo from Service: {tradeEvent.TradeId}" });

    return savedTradeEvent;
}

2 references | 1 author, 1 change | 0 exceptions
public async Task<TradeEvent> PublishTradeEventMessage(TradeEvent tradeEvent)
{
    var savedTradeEvent = await SaveTradeEvent(tradeEvent);
    var brokerClient = new BrokerClient();
    await brokerClient.PublishMessageAsync(new PublishedTradeEventMessage { Content = $"Saved trade event: {tradeEvent.TradeId}" });

    return savedTradeEvent;
}
```

Figure 10 Publish message function (1)

```
1 reference | 1 author, 1 change | 0 exceptions
protected override async Task RunAsync(CancellationToken cancellationToken)
{
    this._cancellationToken = cancellationToken;
    while (true)
    {
        await Task.Factory.StartNew(PublishMessageThree, this._cancellationToken);
        await Task.Delay(TimeSpan.FromSeconds(5), cancellationToken);
    }
}

1 reference | 1 author, 2 changes | 0 exceptions
private static async Task PublishMessageThree()
{
    var brokerClient = new BrokerClient();
    await brokerClient.PublishMessageAsync(new PublishedAutoMessage { Content = $"Service A is publishing Sample Event One at {DateTime.Now}" });
}
}
```

Figure 11 Publish message function (2)

```
[Subscribe]
0 references | 1 author, 1 change | 0 exceptions
private Task HandleMessageOne(PublishedMessageOne message)
{
    ServiceEventSource.Current.ServiceMessage(Context, message: $"Processing PublishedMessageOne: {message.Content}");

    var tradeHubContext = Startup.ServiceProvider.GetRequiredService<IHubContext<TradeHub>>();
    tradeHubContext.Clients.All.SendAsync(method: "ReceiveMessage", arg1: $"Processed PublishedMessageOne: {message.Content}");

    return Task.CompletedTask;
}

[Subscribe]
0 references | 1 author, 1 change | 0 exceptions
private Task HandleTradeEventMessage(PublishedTradeEventMessage tradeEventMessage)
{
    ServiceEventSource.Current.ServiceMessage(Context, message: $"Processing PublishedTradeEventMessage: {tradeEventMessage.Content}");

    var tradeHubContext = Startup.ServiceProvider.GetRequiredService<IHubContext<TradeHub>>();
    tradeHubContext.Clients.All.SendAsync(method: "ReceiveTradeEventMessage", arg1: $"Processed PublishedMessageTwo: {tradeEventMessage.Content}");

    return Task.CompletedTask;
}

[Subscribe]
0 references | 1 author, 1 change | 0 exceptions
private Task HandleAutoMessage(PublishedAutoMessage autoMessage)
{
    ServiceEventSource.Current.ServiceMessage(Context, message: $"Processing PublishedAutoMessage: {autoMessage.Content}");

    var tradeHubContext = Startup.ServiceProvider.GetRequiredService<IHubContext<TradeHub>>();
    tradeHubContext.Clients.All.SendAsync(method: "ReceiveAutoMessage", arg1: $"Processed PublishedMessageThree: {autoMessage.Content}");

    return Task.CompletedTask;
}
```

Figure 12 Subscriber functions

5 Testing

5.1 Tests development

Since the example product had many aspects, different tests were created to test all those elements. The software functionality was tested along with the infrastructure supporting the application.

5.1.1 Application Test

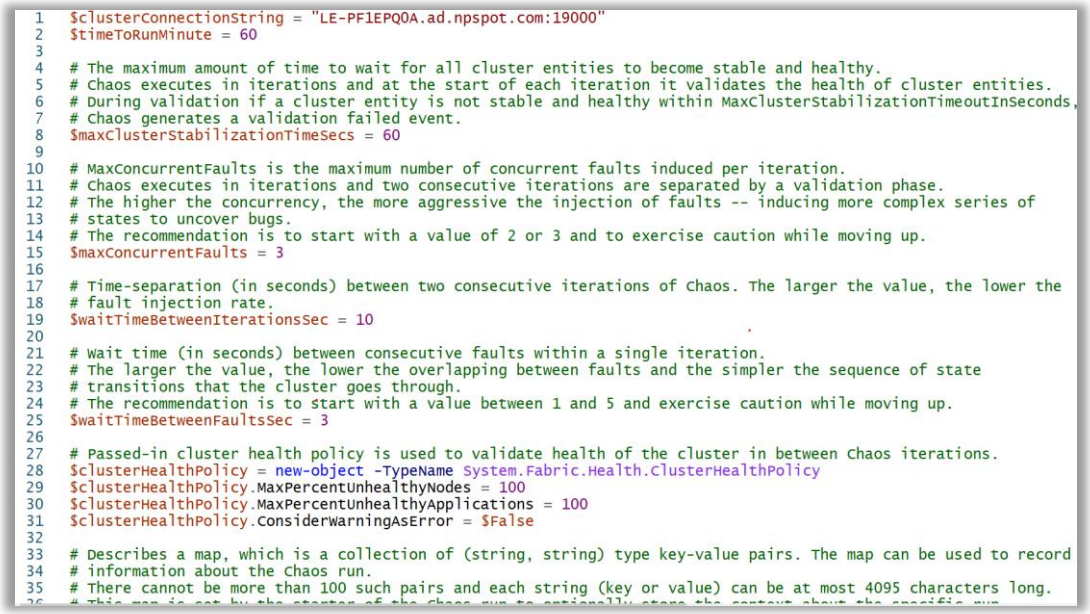
Firstly, the three simple services needed to be tested by sending echo request through the Controllers within API EntryPoints. The services' response can be monitored through Microsoft Azure Application Insight

Secondly, manual testing through the WebClient UI is needed to see the response from different types of requests. The response can be view throw the web page as well.

5.1.2 Infrastructure Test

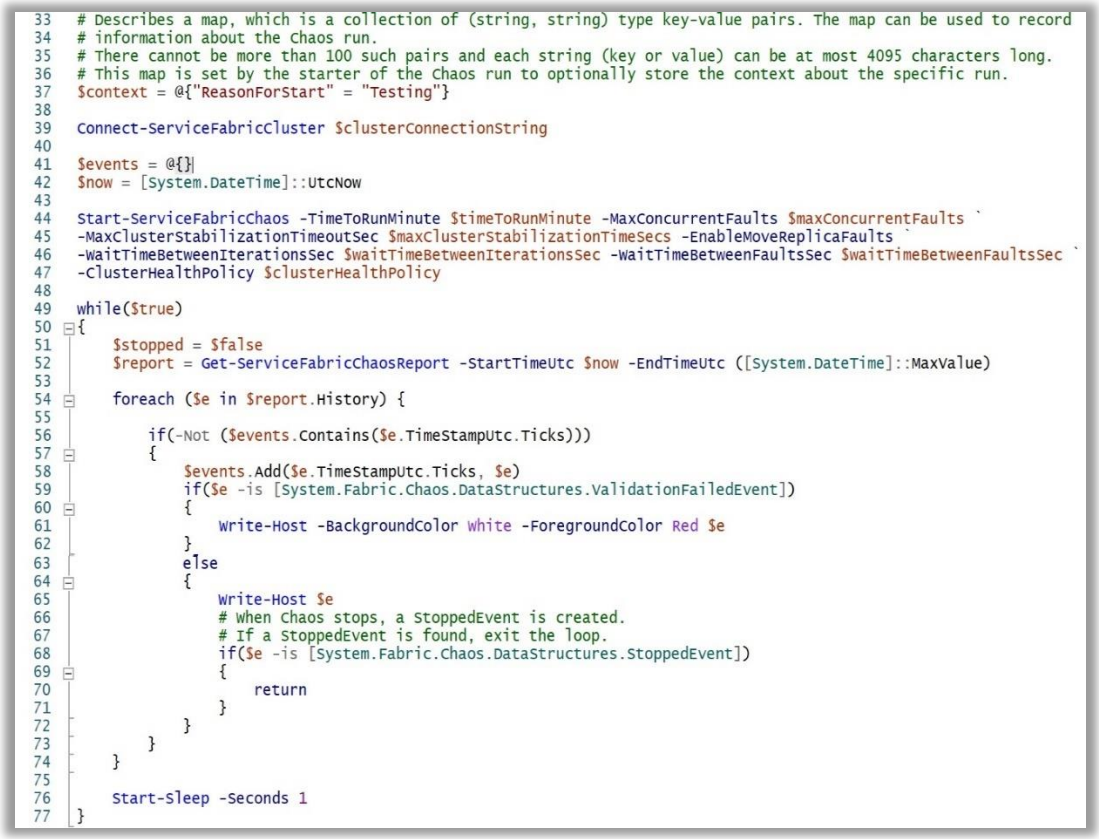
Firstly, within the NBomber Load test, different scenarios were created as the application was developed in following multiple service types. With the actor service types, the service endpoint is the actor's endpoint which is the node's endpoint. On the other hand, service developed following stateful or stateless type takes the cluster endpoint as the service endpoint. Since NBomber script require the targeted endpoint, different scenarios were created with similar steps. Within the test script, the tool sends a large amount of http request until the service cluster reaches its maximum capacity.

Secondly, following Microsoft's documentation, a chaos test was written using PowerShell as the test is a built-in component of Azure Service Fabric. Screen shots from the Chaos Test PowerShell script are represented by figures 10 and 11. The scripts specified that there would be a chaos scenario running every 10 seconds. The chaos test maximum concurrent fault was set to 3. This indicates that there would be 3 cluster failure scenarios occurring. The cluster only have 60 seconds to stabilize and keep the application running. If the service is unavailable longer than that period, this test will be considered a failure. Additionally, the health of the cluster and any failed test will be constantly updated.



```powershell
$clusterConnectionString = "LE-PF1EPQ0A.ad.npspot.com:19000"
$timeToRunMinute = 60

# The maximum amount of time to wait for all cluster entities to become stable and healthy.
# Chaos executes in iterations and at the start of each iteration it validates the health of cluster entities.
# During validation if a cluster entity is not stable and healthy within MaxClusterStabilizationTimeoutInSeconds,
# Chaos generates a validation failed event.
$maxClusterStabilizationTimeSecs = 60

# MaxConcurrentFaults is the maximum number of concurrent faults induced per iteration.
# Chaos executes in iterations and two consecutive iterations are separated by a validation phase.
# The higher the concurrency, the more aggressive the injection of faults -- inducing more complex series of
# states to uncover bugs.
# The recommendation is to start with a value of 2 or 3 and to exercise caution while moving up.
$maxConcurrentFaults = 3

# Time-separation (in seconds) between two consecutive iterations of Chaos. The larger the value, the lower the
# fault injection rate.
$waitTimeBetweenIterationsSec = 10

# Wait time (in seconds) between consecutive faults within a single iteration.
# The larger the value, the lower the overlapping between faults and the simpler the sequence of state
# transitions that the cluster goes through.
# The recommendation is to start with a value between 1 and 5 and exercise caution while moving up.
$waitTimeBetweenFaultsSec = 3

# Passed-in cluster health policy is used to validate health of the cluster in between Chaos iterations.
$clusterHealthPolicy = new-object -TypeName System.Fabric.Health.ClusterHealthPolicy
$clusterHealthPolicy.MaxPercentUnhealthyNodes = 100
$clusterHealthPolicy.MaxPercentUnhealthyApplications = 100
$clusterHealthPolicy.ConsiderWarningAsError = $False

# Describes a map, which is a collection of (string, string) type key-value pairs. The map can be used to record
# information about the Chaos run.
# There cannot be more than 100 such pairs and each string (key or value) can be at most 4095 characters long.
```

Figure 13: Chaos test PowerShell script (1)



```powershell
# Describes a map, which is a collection of (string, string) type key-value pairs. The map can be used to record
# information about the Chaos run.
# There cannot be more than 100 such pairs and each string (key or value) can be at most 4095 characters long.
# This map is set by the starter of the Chaos run to optionally store the context about the specific run.
$context = @{"ReasonForStart" = "Testing"}

Connect-ServiceFabricCluster $clusterConnectionString

$events = @{}
$now = [System.DateTime]::UtcNow

Start-ServiceFabricChaos -TimeToRunMinute $timeToRunMinute -MaxConcurrentFaults $maxConcurrentFaults `
-MaxClusterStabilizationTimeoutSec $maxClusterStabilizationTimeSecs -EnableMoveReplicaFaults `
-WaitTimeBetweenIterationsSec $waitTimeBetweenIterationsSec -WaitTimeBetweenFaultsSec $waitTimeBetweenFaultsSec `
-ClusterHealthPolicy $clusterHealthPolicy

while($true)
{
    $stopped = $false
    $report = Get-ServiceFabricChaosReport -StartTimeUtc $now -EndTimeUtc ([System.DateTime]::MaxValue)

    foreach ($e in $report.History) {

        if(-Not ($events.Contains($e.TimeStampUtc.Ticks)))
        {
            $events.Add($e.TimeStampUtc.Ticks, $e)
            if($e -is [System.Fabric.Chaos.DataStructures.ValidationFailedEvent])
            {
                Write-Host -BackgroundColor White -ForegroundColor Red $e
            }
            else
            {
                Write-Host $e
                # When Chaos stops, a StoppedEvent is created.
                # If a StoppedEvent is found, exit the loop.
                if($e -is [System.Fabric.Chaos.DataStructures.StoppedEvent])
                {
                    return
                }
            }
        }
    }

    Start-Sleep -Seconds 1
}
```

Figure 14: Chaos test PowerShell script (2)

5.2 Test results

5.2.1 Application Test

Firstly, the Controllers in the API EntryPoints were used to send echo request to all services. The response was view from the monitoring tool. The result of the test is shown in the screen shot of the tool's web UI in figure 15.

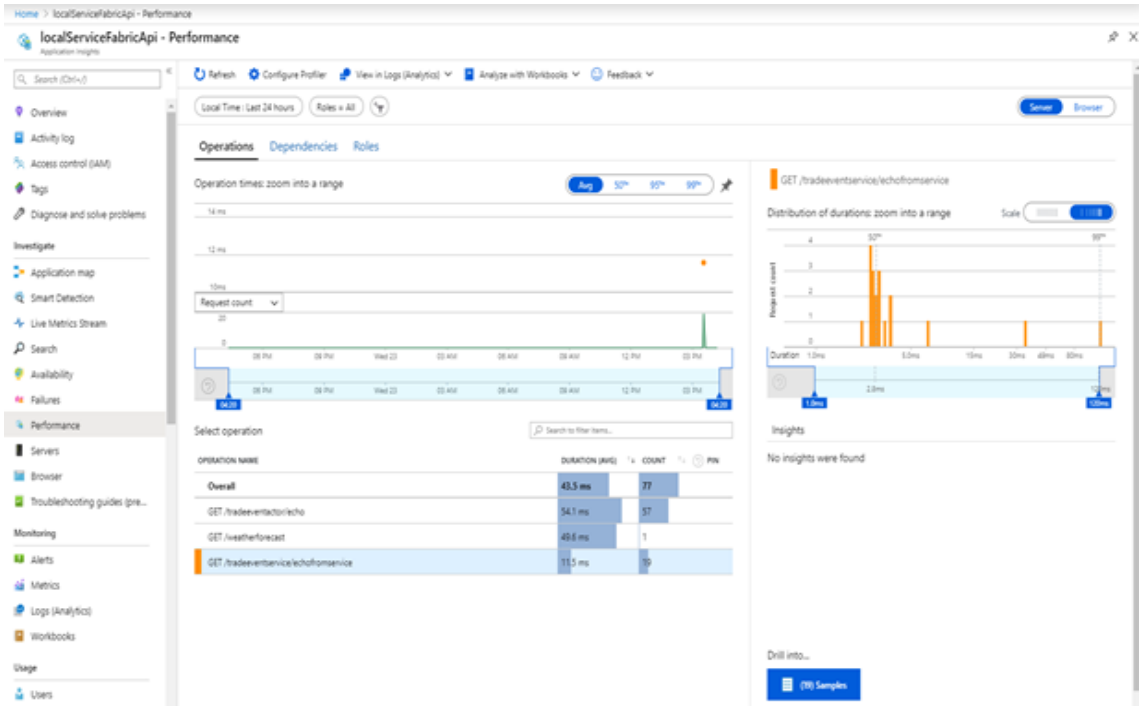


Figure 15 API test result

From the figure, the results can be seen in the bottom middle column under "Operation" section. The result showed that every service responded in a certain amount of time without failure.

Next, the full functionality application is tested. After sending multiple messages to the service, results can be seen in the web browser in figure 15. The three columns, from left to right, display the status of the message.

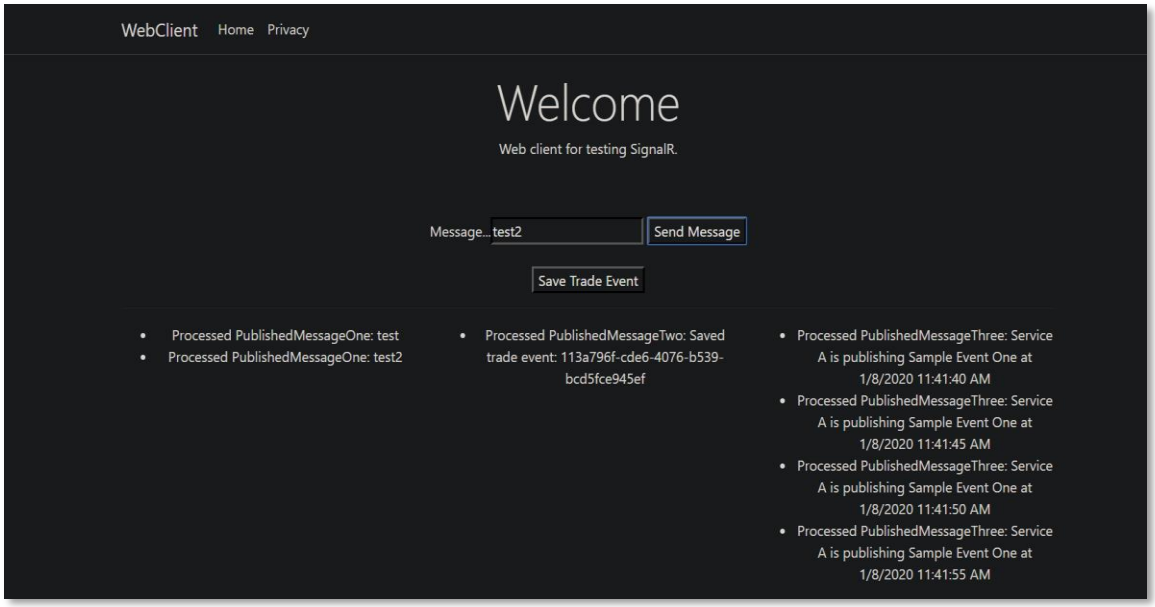


Figure 16: Application manual testing result

Firstly, by adding simple test and triggering the “Send Message” button, the first type of request was made to the service. Its corresponding response can be view from the bottom left column. Both times the function was triggered, the response was the same: “Processed PublishedMessageOne”. This showed that the correct response was received when the request was made.

Next, the function to generate a message ID was called with the button “Save Trade Event”. From the middle column, a response “Processed PublishMessageTwo” can be seen. This also indicated that the correct subscriber was called. Additionally, a random message ID was shown. This response showed that the subscriber function was triggered correctly.

Finally, from the bottom right column, the response “Processed PublishMessageThree” is shown every 5 seconds. This showed that not only does the ServiceA is publishing message accordingly, but also the broker is sending its message to the right subscriber and SignalR is updating the process in real-time.

5.2.2 Infrastructure Test

Load test result

The result from the Load test are gathered from the NBomber graphical user interface. Both tests had yield positive results, as shown in Figure 13 and Figure 14, with every request having passed. For the scenario which the actor service type was developed, the number of requests was 4701. The number of requests that was receive is 4701. This resulted in the percentage of received quest at 100%. With the actor service type, the application responded

very well under high workload. The result was similar with the stateful/stateless type service. Some useful findings were also found within this test. Although the test ran for same amount of time, some advantages appear from using the actor service type over the other type. Firstly, the numbers of request processed from the actor type was greater than the others with 4701 requests received compared to 3534. This indicates that application built using actor service had better response time. Additionally, the indicators also showed that almost all of actor type service's response are less than 800ms. On the other hand, application built from other service types only have about 80% of responses under 800ms. Although this comparison was not part of testing criteria, this finding will act as a foundation for future development when service designing in involved.

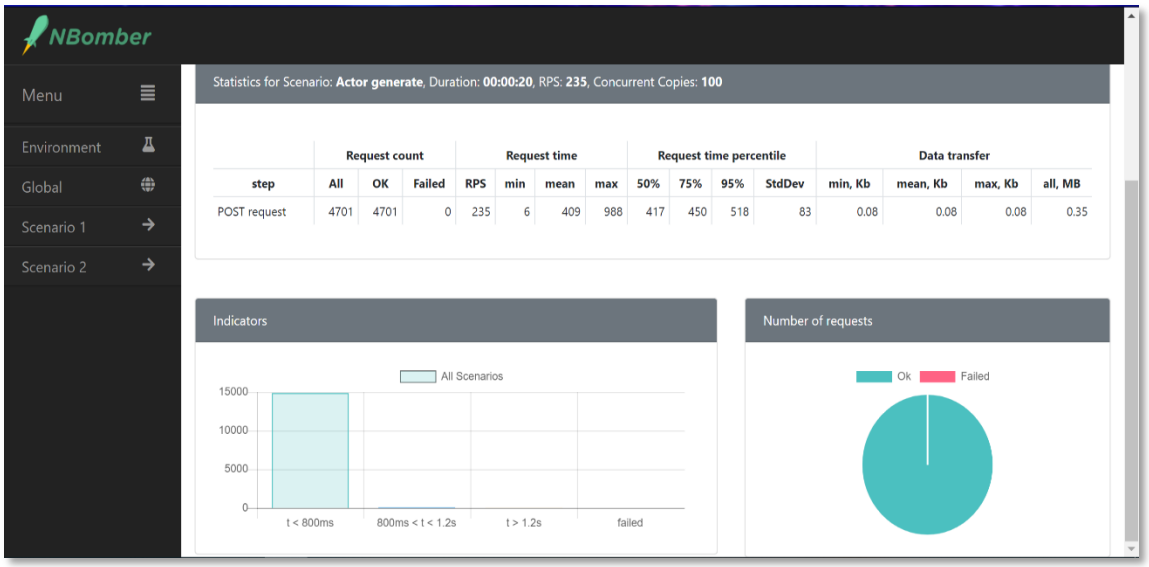


Figure 17: Application Load testing result (1)

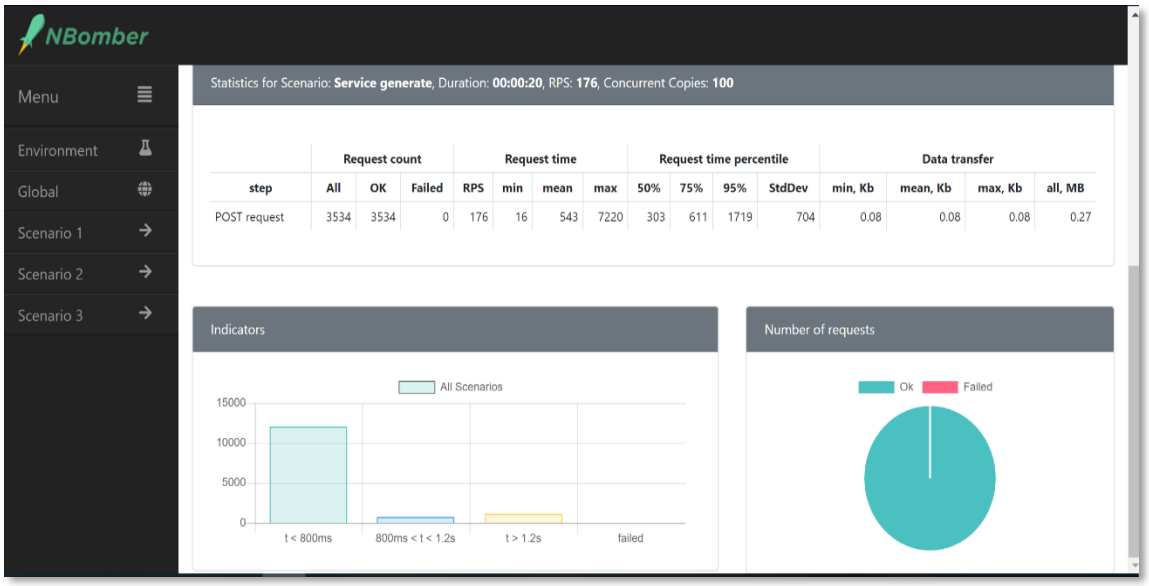


Figure 18: Application Load testing result (2)

Chaos Test result

After running the Chaos Test, the cluster health can be view through the Service Fabric monitor console. During the test, some nodes and the application health showed red which indicates its failure as shown in figure 19. However, within seconds, the nodes returned a green status to its health. The result was repeated multiple times for multiple nodes throughout the test. This indicates that the Orchestrator was able to response to certain disaster scenario and stabilize the cluster by itself.

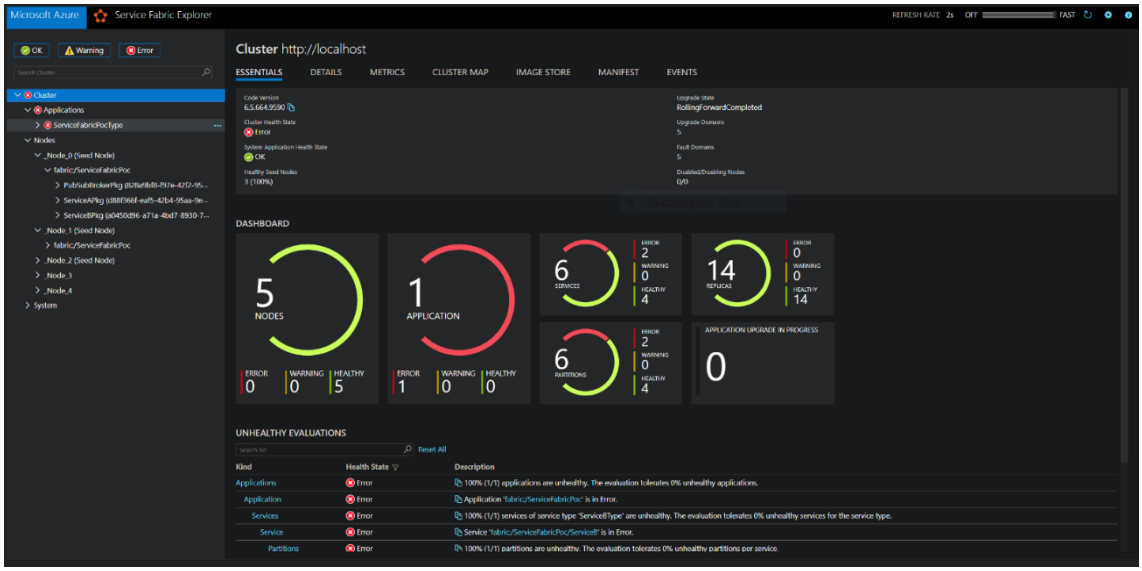


Figure 19 Service Fabric management console during Chaos test

6 Conclusion

The project had not only provided evidence to support usage of Service Fabric in development but also given the opportunity to learn for all parties involved. The author of the report, especially, had managed to improve significantly in understanding more about cloud application architecture. To complete the project, the author had to learn new concepts including containerized application, cluster orchestration, infrastructure testing and using Service Fabric as a development platform. The author was only able to complete this project by combining these new understanding with established software engineering skills.

With the gathered knowledge base along with utilizing the waterfall method and software development lifecycle, the author created replicas of some of Nord Pool services on the Service Fabric platform. This product, along with its infrastructure, followed development criteria that the platform has. For example, web applications need to be created with certain templates and use the ASP.NET Core framework. Once the replicas were created, their functionality and architecture were tested.

The positive test results provide some evidence that Azure Service Fabric has the capabilities to act as a platform for developers to build applications and services upon it which meet customers demand. Additionally, the sample application along with its infrastructure passed both the load test and chaos test. This indicates that Azure Service Fabric can provide both application high availability and cluster resilience. During the load testing phase, the author also found unexpected results. The load test was running for the same amount of time against

services built on the stateful service model and actor service model. However, the actor model managed to handle significantly more requests compared to its counterpart. This indicate that services following this model have better processing time. With this finding, developers can have better knowledge on how to design application architecture.

Although the test results were positive, the decision for this product to be integrated into the system depends on further research into other aspects. Some element which affect this choice include price, time needed for migration, cost of migration, etc. One of the biggest drawbacks from fully migrating to Service Fabric would be its high dependency on its own template and architecture. For example, since the ASP.NET Core framework needs to be followed, developers who are not familiar with this framework will need training. Furthermore, for every service to work smoothly together, the entire product line needs to be migrated and rebuilt following the platform requirement. With future research, perhaps the benefit of migration will outweigh its cost. As for this project, Nord Pool can use the knowledge gathered and the report as initial documentation for future development.

References

Electronic sources

Historical and Impact Analysis of API Breaking Changes: A Large-Scale Study. Xavier L, Brito A, Hora A, Valente M. 2017. Accessed 5th November 2019:
https://homepages.dcc.ufmg.br/~mtov/pub/2017-saner-breaking-apis.pdf

IEA organization. 2019. Global energy demand rose by 2.3% in 2018, its fastest pace in the last decade. Accessed 30 December 2019:
https://www.iea.org/news/global-energy-demand-rose-by-23-in-2018-its-fastest-pace-in-the-last-decade

Principles of Chaos Engineering. May 2018. Accessed 13 January 2020:
http://principlesofchaos.org/?lang=ENcontent

Service Fabric Testability Scenarios. Microsoft. 2019.Accessed 13 January 2020:
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-testability-scenarios

Stateless, Stateful or Actor service. Dajbych V. 2017. Accessed 29 November 2019:
https://www.dajbych.net/stateless-stateful-or-actor-service

Project Scope. Rouse M. April 2020. Accessed 5th November 2019:
https://searchcio.techtarget.com/definition/project-scope

Real-time ASP.NET with SignalR. Microsoft Corporation 2020. Accessed 23rd November 2019:
https://dotnet.microsoft.com/apps/aspnet/signalr

The Open University. Understanding different research perspectives. Accessed 14 November 2019:
https://www.open.edu/openlearn/money-management/understanding-different-research-perspectives/content-section-6

U.S Energy Information Administration. 2019. Electricity explained. Accessed 30 December 2019:
https://www.eia.gov/energyexplained/electricity/

What cloud computing really means. Gruman G, Knorr E. April 2008. Accessed 15 November 2019:
http://skysolutions.co.zw/docs/What_Cloud_Computing_Really_Means.pdf

What is an API? (Application Programming Interface). MuleSoft LLC. 2020. Accessed November 2019:
https://www.mulesoft.com/resources/api/what-is-an-api

What is Gantt-chart. microTool Ltd. 2020. Accessed 20th November 2019:
https://www.microtool.de/en/knowledge-base/what-is-a-gantt-chart/

What Is SDLC? Understand the Software Development Life Cycle. Stackify. 2020. Accesses May 2020:
https://stackify.com/what-is-sdlc/#wpautbox_about

Figures

Appendixes

Appendix 1: Application Programmable Interface

An application programming interface (API) is a computing interface which defines interactions between multiple software intermediaries (MuleSoft 2020). It defines the kinds of requests that two programs can be make between each other. Within the API, rules are set up for how to make the requests, the data formats that should be used, the conventions to follow. APIs are fully customizable; they also provide extension mechanisms to users for creating and customizing the API to fit their needs.

Appendix 2: Gantt Chart

According to 'microtool.de', a Gantt chart is a type of bar chart that illustrates a project schedule. This chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis. The width of the horizontal bars in the graph shows the duration of each activity. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project.

Appendix 3: Virtual Machine

In computing, a virtual machine (VM) is an emulation of a computer system (Rouse 2019). Virtual machines are based on computer architectures and provide functionality of a physical computer. They require a partitioned resource such RAM, CPU, Memory from their host machines. Their implementations involve specialized set of hardware andsoftware.

Appendix 4: Least-privilege security principle

The "least-privilege" security principle refers to the access right of individuals within an organization. It states that employees should only receive enough access to resource that enable them to finish their job without any further extension (Rouse 2017). This restrict access rights and reduce security flaws within cooperate systems.

Appendix 5: Visual Studio IDE

Integrated development environments (IDE) are software application which comprise of sets of tools which aid developers in building services (Microsoft 2020). Visual Studio is Microsoft's IDE to support developers building any type of applications. This IDE has unique tools with aids in integration with Microsoft Azure resources.