Research Report

RESEARCH TO ENTERPRISE ARCHITECTURE
GROUP 2 S6-RB03

Preface

Having concluded that no-code works counterproductive when designing and building an enterprise application, we have decided to start a second phase research plan. This report focuses mainly on non-functional requirements, architecture, and implementation for the group project. The goal is to eventually build an enterprise reservation system which can connect to an external system. We use the DOT-framework when conducting research. The chosen strategies can be found in the research plan. The names of who conducted the research can be found underneath every sub question.

Table of contents

How do we build a reservation system which is GDPR compliant, usable in most restaurants, flexible, performant, transferable, observable, easily deployable and secure?

1. How can data be handled safely and according to GDPR regulations

By: Youri van der Loo

Because we are working on a project that could be rolled out in the EU, we have to make sure we comply with privacy legislation. That's why we first want to research GDPR.

There are only a couple instances in which it's legal to process person data. Most of these instances are not really relevant, so most times you will have to get consent from the data subject. This consent must be freely given, specific, informed and unambiguous. The request for consent must be clearly distinguishable from other matters and presented in clear and plain language. Furthermore data subjects should be able to withdraw their consent at any time. It is also important for children under 13 to have permission from their parents before giving consent. Important is to keep documentary evidence of consent, otherwise the consent given is no use to you.

If you process data, you have to do so according to seven protection and accountability principles:

1. **Lawfulness, fairness and transparency** — Processing must be lawful, fair, and transparent to the data subject.
2. **Purpose limitation** — You must process data for the legitimate purposes specified explicitly to the data subject when you collected it.
3. **Data minimization** — You should collect and process only as much data as absolutely necessary for the purposes specified.
4. **Accuracy** — You must keep personal data accurate and up to date.
5. **Storage limitation** — You may only store personally identifying data for as long as necessary for the specified purpose.
6. **Integrity and confidentiality** — Processing must be done in such a way as to ensure appropriate security, integrity, and confidentiality (e.g. by using encryption).
7. **Accountability** — In order to be GDPR compliant, you need to be able to demonstrate GDPR compliance with these principles. The data controller is responsible for this.

It is required to handle data securely by implementing "appropriate technical and organizational measures."

Technical measures mean anything from requiring your employees to use two-factor authentication on accounts where personal data are stored to contracting with cloud providers that use end-to-end encryption.

Organizational measures are things like staff trainings, adding a data privacy policy to your employee handbook, or limiting access to personal data to only those employees in your organization who need it.

If a data breach occurs, the data subjects have to be notified within 72 hours or face penalties. (may be waived if technological safeguards are used, such as encryption, to render data useless to an attacker)

Hiring a data protection officer is not always mandatory. It is only mandatory when you are a public authority, when you need to monitor people systematically and regularly on a large scale or when you need to process a special categories of data on a large scale. One of the categories that is listed is data concerning health. In our system we are going to store dietary requirements, however this is not our core activity and not on a large scale. So we do not have to appoint a data protection officer, however there are some benefits to appointing one anyway.

The GDPR recognizes a litany of new privacy rights for data subjects, which aim to give individuals more control over the data they loan to organizations. Most of these rights often fall under the guise of transparency, below is a rundown of data subjects' privacy rights:

1. The right to be informed
2. The right of access
3. The right to rectification
4. The right to erasure
5. The right to restrict processing
6. The right to data portability
7. The right to object
8. Rights in relation to automated decision making and profiling.

All the mentioned data protection principles must be followed and also considered in the design of future products or activities.

2. What are commonly used enterprise architectures and their advantages/disadvantages?

By: Youri van der Loo

In order to build a software system, we first need to plan out an architecture. For this semester it is important that we can justify our different technical choices. That's why we are researching common enterprise architectures and their advantages and disadvantages.

While searching for commonly used enterprise architectures there were certain architectures that came up most of the times. Of these we have found out its characteristics, how it works, an example of the architecture, the advantages and disadvantages and some specific cases when you should probably use it.

A. Layered Architecture

Operation: data enters top layer and works its way down each layer until it reaches the bottom, usually a database

Example: MVC structure

Pros:

- Separation of concerns makes it maintainable, testable, easy to assign separate roles and easy to update and enhance layers separately

Cons:

- Source code can get really messy
- The larger the application the more resource-intensive
- Layer isolation can make it hard to understand the whole without understanding each module
- Coders can skip past layers to create tight coupling and produce a logical mess full of complex interdependencies
- Monolithic deployment is often unavoidable, so if you update a layer, you have to reinstall everything all over again
- Not scalable due to the coupling between the layers

Specific use cases:

- When you need to build a new application quickly
- When your team is inexperienced in terms of other architectures
- When you need to mirror traditional IT departments or processes
- When your application requires strict maintainability and testability standards
-

B. Event-Driven Architecture

Operation: Central unit that accepts all data and delegates it to separate modules that handle the particular type

Example: -

Pros:

- Easily adaptable to complex environments
- Easily scalable
- Better response time
- Easily extendable

Cons:

- Testing can be complex if the modules can affect each other, interactions can only be tested in a fully functioning system
- Error handling can be difficult to structure
- When modules fail, the central unit must have a backup plan
- A burst of messages can slow down processing speed
- Developing a systemwide data structure for events can be complex
- Maintaining a transaction-based mechanism for consistency is difficult because the modules are so decoupled and independent

Specific use cases:

- When you have an asynchronous system with an asynchronous data flow
- When the individual blocks in your application interact only with a few of many modules
- When you want to build a UI
- When your application leverages instant data communication that scales on-demand

C. Microkernel Architecture

Operation: A microkernel with most standard features that can have different plugins layered on top

Example: A good example of a microkernel architecture is Eclipse IDE. At first it is just a fancy editor. However when you start adding plugins it becomes a highly customizable and useful product.

Pros:

- Highly flexible
- Maintaining high-performance applications is nice, since the software can be customized to include only the features that are needed the most

Cons:

- Deciding what belongs in a kernel is difficult
- The plugins require quite some handshaking code, to make aware the plug-in is installed
- Modifying the microkernel can be difficult to nearly impossible, since other plugins depend on it
- Choosing the right granularity for the kernel function is already difficult, but changing it later is even more difficult
- Architecture is naturally suited to apps that are smaller in size

Specific use cases:

- When you develop tools used by a wide variety of people
- When your application has a clear division between routines and higher order rules
- When your application has a fixed set of core routines and a dynamic set of rules that must be updated frequently

D. Microservices Architecture

Operation: In a microservices architecture you split up an application into multiple microservices with each its own responsibility, they can each communicate in its own way.

Example: A microservice architecture is like a herd of guinea pigs, each micro service being a guinea pig. Expanding the herd will still leave you with nice small guinea pigs.

Pros:

- The whole software won't collapse if a microservice fails
- You can pick a suitable technology stack for each separate micro service
- Development is smoother since different teams can work parallel on services
- Easily maintainable
- Very flexible
- Easily scalable
- Easily expandable

- Services can be integrated into other applications

Cons:

- Services must be largely independent or else interaction can cause the cloud to become imbalanced
- An application can't always be split into independent units
- Performance can suffer when tasks are spread out between different microservices. The communication costs can be significant
- Too many microservices can confuse users as parts of the web page appear much later than others
- It is important to define a standard protocol for all services to adhere to.

Specific use cases:

- When you develop a website with small components
- When your business/web application is rapidly developing
- When your team is spread out (across the globe)

E. Client-server architecture

Operation: A client makes a request to the server. The server then responds to the client requests appropriately by sending over the requested resources.

Example: Examples could be a file or web server

Pros:

- Highly flexible
- Supports multiple clients
- Data is well protected
- Upgrading the server while the client remains unaffected is easy

Cons:

- Prone to a single point of failure
- Maintenance can be a complex
- Incompatible server capacity can affect the performance

Specific use cases:

- applications that focus on real-time services
- applications that require controlled access and offer multiple services for a large number of distributed clients

3. What can you do to make a project more transferable?

By: Luka Spaninks

For our group the project duration will only be approximately 17 weeks. However, the project will be continued afterwards by other programmers. This means that we have to structure the project in such a way that it is convenient for others to pick up where we left off.

First of all documentation is incredibly important. Without it, it becomes really difficult to get to know the project. Documentation should include among other things: the thought process behind choices and ideas, the software architecture and other design related items.

Writing documentations comes with a few inconveniences unfortunately. Examples are that it becomes outdated really fast and takes a lot of time to write. Precautions to minimize this are:

- Follow standards & conventions
- Write easy to understand code
- Use inline commenting
- Refer to external documentation (e.g. chosen language docs)

For this project we are using an online code repository (GitHub). This repository is open source, meaning that anyone can fork it or create pull requests. There are some things we can do to make this repository more accessible and organized. It is standard for most online repositories to have a README file which contains essential information about the project. The README is displayed on the page of the repository and has several responsibilities:

- Explaining what the project is/does
- Show imagery & examples of the project
- Explain how to install & set up the project
- Link to important resources (documentation, code coverage etc.)

Another important addition is choosing a license, this lets people know what they can and cannot do with your repository. We are working on an open source project so for example an MIT license perfectly fits our needs.

Adding a CONTRIBUTING file to the repository informs others on how to file a bug report, request a feature and create a pull request. This is also a standard in open source projects.

Using the release functionality of the online code repository is also recommended. Creating clear release notes and changelogs informs people on the current state of the product. This includes reporting what bugs got fixed and what new features were implemented.

Other important methods other than documentation include: Q&A sessions, mentoring, pair programming and code reviews. These are all really useful but don't really apply to our situation. We will probably not be available to tutor the students who continue the project. It would however be smart to set something up like a discord server or slack channel. This way communication can be established in an efficient way.

4. How do you make an application performant & horizontal scalable?

By: Luka Spaninks

Performance and horizontal scalability are very important non-functional requirements in an enterprise software system. Keeping response times to a minimum adds to the reliability and user experience. Horizontal scalability divides workloads among different nodes and makes sure the system can handle a large userbase. Another benefit is having less downtime and it is generally less expensive than vertical scaling. The downside is that it does increase the complexity of a system.

Adding automated tests early on in development can really help with maintaining a good performance. Unit & integrations tests can show you measurements of how long it takes to pass a piece of code. This helps developers easily show you where bottlenecks occur. Implementing logging & monitoring can help with finding what code is the main cause of slow performance in a staging or production environment.

During the development phase of software there are a few things you can do to improve quality and performance:

- Keeping client-side requests to a minimum helps with improving network latency.
- Choosing technologies that are low level and scale both horizontally and vertically help.
- Using abstract data types and keeping algorithms to certain standards reduce the chance of bottlenecks
- Removing unnecessary (development) modules when building the application can also improve performance

Server optimizations:

- Using content delivery networks in order to provide clients with static files faster
- Bundling files together and using minification (html, js & css) can help with performance
- Images can be optimized in a variety of ways to make load times faster
- Removing duplication reduces file size
- Use lightweight data structures & protocols where possible

Optimization can become counterproductive to readability and has the potential to add to the complexity of the project. That is why it might be smart to postpone optimization to the end of the development stage.

Having listed some best practices regarding increasing software performance, the questions remains how to make sure an application is horizontally scalable. Below we have listed some important do's that help us achieve this.

- Create stateless applications because a load balancer won't always assign a user to the same instance
- Use a load balancer
- Split business logic into different services to divide the workload
- Perform scalability tests
- Choose a database with availability over consistency (Cassandra & MongoDB)
- Monitor and log the different services
- Use asynchronous communication

Having looked at both performance and horizontal scalability they often go hand in hand. Best practices like using a load balancer, using asynchronous communication and logging can be applied to improve both.

5. What are best practices when efficiently deploying & testing an application while also using automation

By: Koen Rasters

A great strength of working with agile is automatic deployment. Some organizations still use manually deployment, which takes a lot of time. Because each phase of the release will be manually handed from one team to the other. With the help of DevOps, especially continuous integration, and continuous deployment, this can be more efficiently and faster.

A good deployment model is the 3-tier model. This model uses a development tier, staging tier, and a production tier. Sometimes there is also a fourth tier, the testing tier, but this one is most of the times integrated in the staging tier. An advantage of this model is that the code can be reviewed and tested by multiple people before it is deployed on the live environment.

This 3-tier model has led to the two concepts that most of the developers know today: Continuous Integration (CI) and Continuous Deployment (CD). Continuous Integration is responsible for the code to be integrated in the Testing phase and Continuous Deployment is responsible for the code to be deployed to production. When using CI/CD in your application you can reduce this life cycle of development time. A lot of processes will be run automatically which will speed up development. Continuous Integration can automatically run the unit tests in the application and can test the code coverage and code smells. Continuous Deployment can serve the code that is reviewed by the CI to a staging environment and eventually to a production environment.

It is good to have your application horizontally scalable so it can be deployed in the cloud on a Kubernetes cluster. The Kubernetes cluster will handle the scaling of your application, when there is a need for more resources the cluster will deploy more pods or vice versa.

A good practice is to use containers and virtual machines instead of just running it on a machine. With containerizing your application, it always has the same isolated environment where the application is running on. Containerized applications are good horizontally scalable. Continuous Integration benefits from these containerized applications. When pushing the code to the repository a CI pipeline can automatically build and test the container before it will be deployed. If the code is not working correctly the pipeline will fail and will inform the developers. The containers will be the same for the testing environment, staging environment and production environment, so everything will be the same for each environment.

When using updates for your application it is best to use rolling updates. This will ensure the application will be updated without any down time. When using Kubernetes older pods will be replaced by newer pods one by one. This will cause the application to be updated without any notice for the client because the load balancer will only traffic the client to the running pods.

Backups are an important thing to have because when the deployment fails or something goes wrong, you need to have a backup so you can roll back the application. Notifications can be used to inform the developers when something goes wrong and can message them to roll back the application.

Load testing is import so you know what to expect for performance when deploying the application to the production environment. Because automated tests and unit tests cannot guarantee for an error-free performance in the production environment. This is why load testing need to be tested in a staging environment with data which is almost identical to the production environment.

6. How do we connect the reservation service to a messaging bus?

By: Luka Spaninks

In order to make our service work in an event-driven microservices architecture, we have to make sure some sort of messaging is possible. Within the context of the group project, we have chosen to use Apache Kafka. Apache Kafka is a widely used enterprise level service bus which has support for many languages. The main reason for choosing Kafka is the collaboration with another group. They have chosen Kafka which makes it more convenient for us to use it as well.

In our project we have an internal and external service bus. The internal service handles the communication between all the services within our system. The external service bus is used to integrate our system into a bigger microservices environment. For the sake of not repeating ourselves we will be talking only about the internal service bus.

In order to connect our application to Kafka we have to first choose a client library. As mentioned in our tech stack we use the Go programming language. Having looked on Google we see that the first library on GitHub for Go is the one from segmentio. Having done some community research this library seems to be the most user-friendly and contains enough features for regular users.

Now that the library is installed, we can import it and use it in our producers and consumers. In the environment file we define Kafka properties like brokers, topic prefix, and credentials. Furthermore, we create a producer interface and a consumer interface containing all the required methods. Next, we create a Kafka folder containing the implementations of the previously mentioned interfaces. Now we are able to consume and produce messages.

Within our project we have decided upon a topic naming convention based on standards. We use the <app-name>.<data-type>.<event-name> for our internal service bus. Furthermore, we stick to a few best practices in order to keep everything well organized and secure:

- Do not use fields that can change
- Do not tie topic names to services, consumers, or producers
- Leave metadata out of the name if it can be found elsewhere
- Use kebab case

For more information: https://devshawn.com/blog/apache-kafka-topic-naming-conventions/

```
<project>.<product>.<event-name>
<app-name>.<data-type>.<event-name>
<team-name>.<app-name>.<event-type>.<event-name>
```

Kafka naming conventions example

This library satisfied the needs for our internal project just fine, but unfortunately didn't work for the service bus provided to group1. This is why we had to go for the library: "confluent-kafka-go". In order to get this to work we had to install the TDM c compiler and therefor also make changes to the docker images.

7. How do we apply observability to the reservation service?

By: Luka Spaninks

Within an enterprise software system a lot of traffic is to be expected. Things can go wrong, response times can get low and inefficient processes can occur very easily. In order to prevent this we want to install monitoring software in our stack. This way we can have a good overview of system metrics and therefore quickly catch any problems.

For our monitoring needs we went for Prometheus and Grafana. Prometheus used to scrape metrics and Grafana is used to visualize this data in dashboards. We chose these because they are really flexible and work well in a Kubernetes cluster. Also, both are widely used so the community and thus support is relatively big.

In order to install Prometheus and Grafana in our Kubernetes cluster we used the README provided by the GitHub repository Kube-Prometheus. After that we had to implement a metrics endpoint in all of our services using the Prometheus client for Golang. This metrics endpoint is scraped and saved by Prometheus every 5 second.

```
engine.GET("/metrics", gin.WrapH(promhttp.Handler()))
```

Then we port forward the Prometheus service using kubectl and access it from localhost on port 9090.

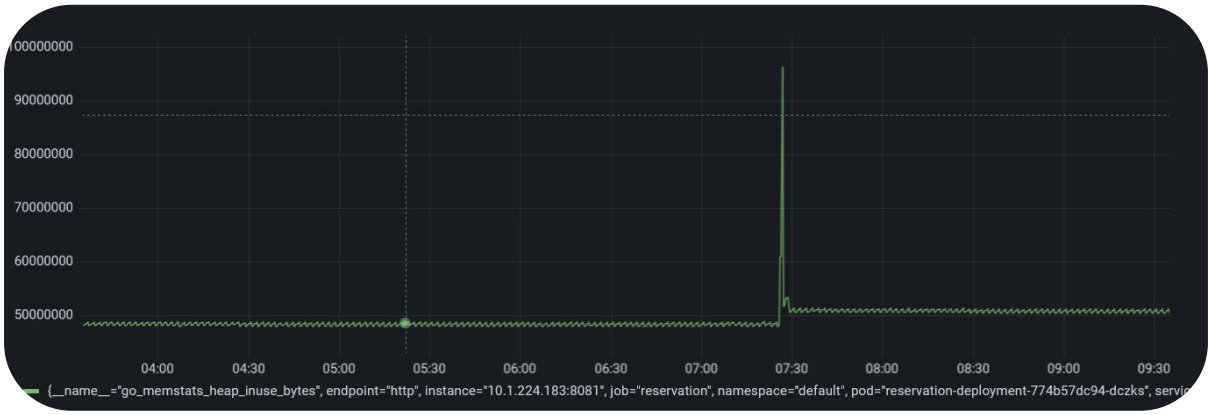
```
kubectl port-forward -n monitoring svc/prometheus-k8s 9090
```

In the image below you can see that 1 reservation service is up and running and the endpoint that is being scraped is: “/metrics”.

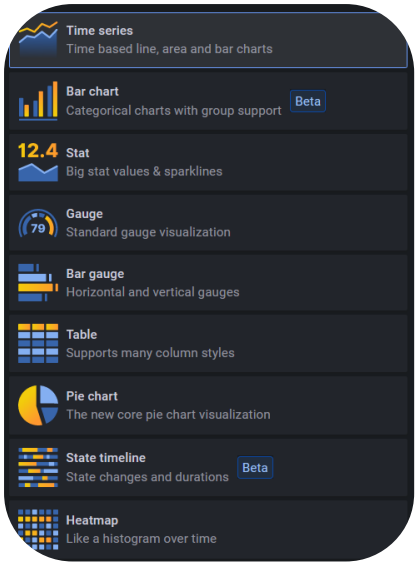
serviceMonitor/default/reservation-monitor/0 (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.1.224.183:8081/metrics	UP	endpoint="http" instance="10.1.224.183:8081" job="reservation" namespace="default" pod="reservation-deployment-774b57dc94-dczks" service="reservation"	12.138s ago	4.035ms	

Now we can port forward Grafana and add Prometheus as a datastore. Now we can visualize metrics in a flexible dashboard. Below you can see an example in the number of heap memory bytes are in use of the reservation service. The advantage of using Grafana is that it is highly customizable and can connect to many datastores. Prometheus is really popular and therefor has a large community backing it.



Below are just a few of the visualization options you can use with Grafana.

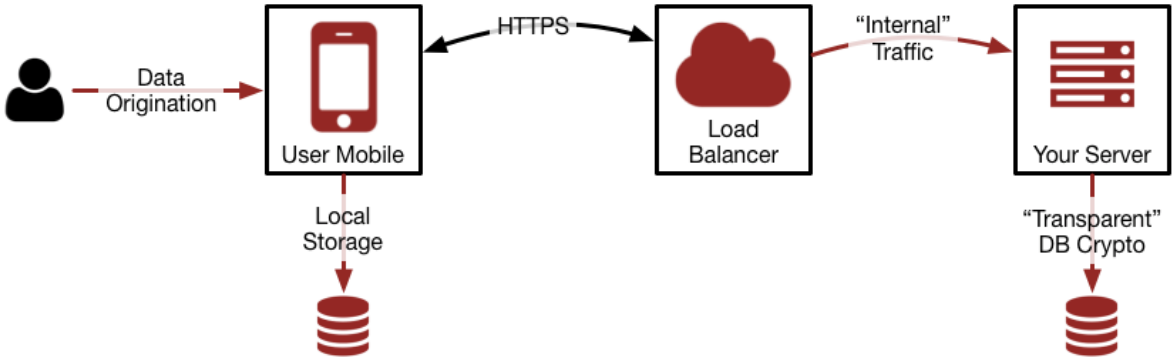


8. How can messaging within an enterprise system be done reliably and securely?

By: Youri van der Loo

In an enterprise system we value the reliability and the security of the system. In order to improve the system on this area there are multiple things you should look into.

First of all, security processes should be established to protect and encrypt data at all levels of your enterprise. All communications and data transmission should be protected with end-to-end encryption to prevent data theft. As you can see in the image below, HTTPS alone is not enough. Black is HTTPS encrypted, however red is still plaintext. Implementing end-to-end encryption makes sure the data gets encrypted at the start and only gets decrypted at the point of use. This way you are not vulnerable at multiple points throughout your system anymore since your data will be encrypted throughout the system.



Besides encrypting data, it is also smart to encrypt secrets that the microservices use for communication like API keys, client secrets, or credentials for basic authentication.
If you make use of any third-party app, your encryption should extend to that app as well. Also, you

should be aware of the vulnerabilities the libraries you are using are bringing with them, so frequently scan the source code, new code contributions, and deployment pipeline for vulnerable dependencies.

To further prevent other people from retrieving our data without permission it is important that authorization/authentication is included. There are a lot of different possible aspects to a microservices architecture, in order to deliver secure and effective authorization/authentication across the microservices it is important to put the right tools and protocols in place.

Besides thinking about all these ways to protect our system, it is also important to make sure your system works as intended. By including unit tests like SAST and DAST into your CI/CD pipeline you relieve the developers of manual security checks.

Attackers can always try to get authorized by brute forcing your authentication service, although this often takes some time you can slow this down even more. By implementing a rate limit, you limit them to only a few attempts per second. This way they might lose interest in this type of attack.

Also, it is important to give each role as limited amount permission as possible. This way you minimize the risk of unauthorized people gaining access to parts of the system you do not want them to.

Reliability can be improved by using an asynchronous messaging bus which stores messages in case a service is unavailable. You can also use event sourcing to add another layer of reliability on top of messaging. However, this does cause the system to become a lot more complex.

9. How to deploy our project with Kubernetes?

By: Youri van der Loo & Koen Rasters

In order to apply scalability to our project we decided to work with the widely used Kubernetes. Before we could implement Kubernetes we first had to discover how to deploy our project with it.

First we created 4 Virtual Machines on the Netlab environment, each of these VMs use the Ubuntu template on which we installed microk8s.

```
NAME      STATUS   ROLES    AGE     VERSION
kube100   Ready    <none>   10d     v1.24.0-2+59bbb3530b6769
kube200   Ready    <none>   10d     v1.24.0-2+59bbb3530b6769
kube175   Ready    <none>   6d23h   v1.24.0-2+59bbb3530b6769
kube150   Ready    <none>   10d     v1.24.0-2+59bbb3530b6769
```

With kubectl we can retrieve all the nodes and their statuses.

The kubes 100, 200 and 150 are master nodes. We chose for three master nodes, since this ensures a high availability cluster.

175 is a worker node, which is not running any kubectl activities.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
mongo	1/1	1	1	3d
config-deployment	1/1	1	1	3h49m
gateway-deployment	1/1	1	1	3h49m
reservation-deployment	1/1	1	1	3h49m
datetime-deployment	1/1	1	1	3h49m

For each of our component in our architecture we have created a Kubernetes deployment.

Mongo is the database deployment

Three respective microservice deployments and a gateway deployment

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.152.183.1	<none>	443/TCP	10d
config	ClusterIP	10.152.183.72	<none>	80/TCP	3h50m
datetime	ClusterIP	10.152.183.113	<none>	80/TCP	3h50m
reservation	ClusterIP	10.152.183.197	<none>	80/TCP	3h50m
gateway-service	LoadBalancer	10.152.183.118	192.168.168.51	8080:31131/TCP	3h50m
mongo-nodeport-svc	NodePort	10.152.183.165	<none>	27017:32000/TCP	3d

Furthermore we also created a service for each deployment.

Three respective microservice services

A gateway with an external IP to be accessed on port 8080 in this case.

The gateway redirects the requests to each microservice via the service.

Here you can see that the gateway is running at the specified IP and port and that it manages to reach the responsible microservice.

Conclusion

In the previous research report: "**Research Report - Appsemble**", we have mainly conducted research towards the different parts of the low code platform Appsemble and other low code platforms. The main discoveries from this research were that Appsemble can only be sufficiently performant if used solely for the front-end. However, Appsemble cannot directly be connected to a messaging system, all the workarounds to connect to a messaging system are not ideal. Also, there are other low code platforms that are a better option then Appsemble, still we prefer to work with a custom tech stack that fits the project so that we can better fulfill the learning outcomes.

In this research report we conducted research towards architecture related matters, as well as just important stuff for projects in general.

In our research towards the **GDPR regulations** we discovered that in most cases, ours too, you need to ask for consent. Processing data needs to happen according to 7 principles: 1. Lawfulness, fairness and transparency, 2. Purpose limitation, 3. Data minimization, 4. Accuracy, 5. Storage limitation, 6. Integrity and confidentiality, 7. Accountability. Furthermore, we learned that the data subject has a couple of rights to gain more control over their data.

While researching the different types of **architectures** we can implement, we found that the most suitable architectural design patterns for our case would be the microservices, as well as the event-driven architecture. They allow for easy scalability, development, reliability, extensibility, and they are lightweight, because they handle less data at once in addition to asynchronous messaging.

In order to improve the **transferability** of the project it is obviously important to well-document all steps in the process you make. Since this documentation can become outdated quickly it is important to: follow standards & conventions, write easy to understand code, use inline commenting and to refer to external documentation. It is common for a git repository to contain a .README file where you explain the project, show imagery & examples, explain how to setup the project and where you link to important resources. Furthermore, it is nice to have clear release notes where you explain what bugs got fixed and what features got implemented

When it comes to **performance and horizontal scalability**, using automated testing during development can ensure the quality of the product is sufficient, and therefore the performance is good. Additionally, monitoring and logging can help identify errors and what parts of the software hinder performance. During development keeping client-side requests to a minimum, choosing technologies that are low level and scale, as well as removing unused modules and using abstract data types helps improve the performance of the software. For the server using CDNs, image optimizations, removing duplication and using lightweight data structures helps optimize the performance. Since optimization can become counterproductive to readability and increase the complexity of the software – a good practice is to keep a list of best practices to use while developing. Those can be things such as – using a load balancer, splitting business logic into different services to reduce load, performing scalability tests, etc.

While looking into **automated deployment** we found a well-known deployment model that we will be using - the 3-tier model. This model uses a development tier, staging tier, and a production tier. An advantage of this model is that the code can be reviewed and tested by multiple people before it is deployed on the live environment. This model also introduces CI/CD, using this in your application reduces the life cycle of development time, since a lot of processes will be run automatically.

Creating stateless microservices in combination with Kubernetes makes sure the stack becomes easily scalable and therefore allows for better performance. The Kubernetes cluster will handle the scaling of your application, when there is a need for more resources the cluster will deploy more pods.

Instead of running it on a machine it is a good practice to use containers and virtual machines, this way it always has the same isolated environment where the application is running on. Containerized applications are properly horizontally scalable.

The building and testing of the container can also be included in the CI pipeline. All containers will be the same for the testing environment, staging environment and production environment.

While looking for ways to implement **observability**, we managed to find a way to satisfy our monitoring needs by using Prometheus in combination with Grafana. Prometheus is used to scrape the metrics endpoints of different services and Grafana takes this data and allows for building flexible custom dashboards. We managed to get Prometheus and Grafana running in our Kubernetes deployment active on the Fontys VCenter environment.

Reliable and secure messaging

Messaging can be done securely by implementing encryption (TLS) end-to-end. Also, using role based authentication can help with making sure the right people/services have access to data. On top of that, make sure you apply testing to your code. Reliability can be achieved by using a message bus or event implement event sourcing.

Reflection

We have gained a better understanding of what it takes to build enterprise software. We have done a lot of practical research and made a plan for components we were not able to complete (such as authentication & authorization). We made the project more challenging for ourselves by using the go programming language. Unfortunately, we did not get to implement a lot of the features we had planned, but we get that the focus was more on the non-functional requirements this semester.

References

Wolford, B. (2019, February 13). What is GDPR, the EU's new data protection law? GDPR.Eu.

Retrieved 28 March 2022, from https://gdpr.eu/what-is-gdpr/

Wayner, P. (2021, June 19). How to choose the right software architecture: The top 5 patterns.

TechBeacon.

Retrieved 31 March 2022, from https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice

Appinventiv. (2022, January 28). What Are The Different Types of Enterprise Software

Architectures?

Retrieved 31 March 2022, from https://appinventiv.com/blog/choose-best-enterprise-architecture/

Richards, M. (2015, February). Software Architecture Patterns. O'Reilly Online Learning.

Retrieved 31 March 2022, from https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/

Starting an Open Source Project. (2022, March 28). Open Source Guides.

Retrieved 3 April 2022, from https://opensource.guide/starting-a-project/

4 Effective Knowledge Transfer Methods for Software Teams. (2021, July 1). Codingsans

Consulted on 3 April 2022, from https://codingsans.com/blog/knowledge-transfer-methods-for-software-teams

L. (2018, August 27). On Importance of Knowledge Transfer Between Software Developers. Medium.

Retrieved 3 April 2022, from https://medium.com/@logicify/logicify-best-practices-for-knowledge-transfer-between-developers-353ce86cc98a

Dagli, R. (2021, October 14). How to Start an Open Source Project on GitHub – Tips from Building My Trending Repo. freeCodeCamp.Org.

Retrieved 3 April 2022, from https://www.freecodecamp.org/news/how-to-start-an-open-source-project-on-github-tips-from-building-my-trending-repo/

T. (2019, August 11). *11 Ways To Improve Software Quality*. Testpoint.

Retrieved 4 April 2022, from https://testpoint.com.au/11-ways-to-improve-software-quality/

Odhiambo, D. (2018, November 2). *Performance Optimization in Software Development - The Andela Way*. Medium.

Retrieved 4 April 2022, from https://medium.com/the-andela-way/performance-optimization-in-software-development-ae7952ab885e

S. (2021, October 21). *Horizontal Scalability in Your Software: The What, Why, and How*. SentinelOne.

Retrieved 4 April 2022, from https://www.sentinelone.com/blog/horizontal-scalability-software/

Jegannathan, B. (2018, April 24). *10 tips for building "Purely" horizontal scalable Architectures*. Medium.

Retrieved 4 April 2022, from https://blog.francium.tech/10-tips-for-building-purely-horizontal-scalable-architectures-3e525b3667bb

***How to Build a Scalable Web Application for Your Project*. (2021, November 17). Gearheart.** Retrieved 4 April 2022, from https://gearheart.io/articles/how-build-scalable-web-applications/

Seymour, S. (2020, March 29). *Apache Kafka: Topic Naming Conventions*. Shawn Seymour. Retrieved 25 April 2022, from https://devshawn.com/blog/apache-kafka-topic-naming-conventions/

4 Best Practices to Ensure Smooth Deployments. loadninja.com. Retrieved 2 June 2022, from https://loadninja.com/articles/performance-testing-deployment-best-practices/

Best practices for deploying web apps have evolved. Platform.Sh. Retrieved 2 June 2022, from https://platform.sh/blog/2020/best-practices-in-deploying-web-apps-updated-for-2020/

Putano, B. (2020, April 28). 8 Best Practices for Agile Software Deployment. Stackify. Retrieved 2 June 2022, from https://stackify.com/deployment-best-practices/

Team, E. Y. (2022, March 18). Best Practices for Application Deployment. EngineYard. Retrieved 2 June 2022, from https://www.engineyard.com/blog/best-practices-for-application-deployment/

P. (2019). *GitHub - prometheus-operator/kube-prometheus: Use Prometheus to monitor Kubernetes and applications running on Kubernetes*. GitHub. Retrieved June 2, 2022, from https://github.com/prometheus-operator/kube-prometheus