Voorjaar 2023

Plasmeijer,Sem S.

Fontys hogeschool ICT

Sprint Research

Semester Enterprise Software

Contents

[1. Enterprise architecture & software platform 3](#_Toc132703095)

[1.1 Architecture 3](#_Toc132703096)

[1.1.1 Why 3](#_Toc132703097)

[1.2 How to start 3](#_Toc132703098)

[1.2.1 Stakeholders and ideas 3](#_Toc132703099)

[1.2.2 Requirements 4](#_Toc132703100)

[1.2.3 Describing initial architecture 4](#_Toc132703101)

[1.3 software platforms 5](#_Toc132703102)

[2. Architecture & quality 6](#_Toc132703103)

[2.1 Architecture styles 6](#_Toc132703104)

[2.1.1 Non-FUNCTIONAL requirements NFR 6](#_Toc132703105)

[2.1.2 Trends and styles 6](#_Toc132703106)

[2.2 What is an api gateway 8](#_Toc132703107)

[2.3 Quality and how to manage it? 8](#_Toc132703108)

[2.3.1 WHY? 8](#_Toc132703109)

[2.3.2 What? 8](#_Toc132703110)

[Security by design 9](#_Toc132703111)

[Messaging 9](#_Toc132703112)

[DevSecOps 10](#_Toc132703113)

[Deployment 10](#_Toc132703114)

[Kubernetes and container orchestration 10](#_Toc132703115)

[Cloud services 11](#_Toc132703116)

[Scalibity and Elasticity 11](#_Toc132703117)

[Virtualization 11](#_Toc132703118)

[Mangement 11](#_Toc132703119)

[12-factor 11](#_Toc132703120)

[Vendor lock-in 11](#_Toc132703121)

[services 11](#_Toc132703122)

[cost 11](#_Toc132703123)

# Enterprise architecture & software platform

Enterprise software is software created for the purpose of managing data and applications of an organisation. This is not to be confused with typical software solutions which have as a goal to satisfy the end-user.

Examples of enterprise applications are:

* Billing systems
* CRM: customer relation management systems
* ERP: enterprise resource planning systems
* E-commerce environments

## Architecture

### Why

When combining the typical applications with its counterpart the enterprise applications you get an enterprise architecture, a model which defines all important applications and their relations to each other. In martin fowler’s keynote he explains that this definition can be vague, what is important, who decides which is important but more importantly he explains why it is important.   
According to fowler’s hypothesis by maintaining, refactoring, and cleaning your codebase will allow you to create more functionality in the long-term allowing your software solution to be more competitive over a longer period even if this means a loss in the short-term.

## How to start

### Stakeholders and ideas

So how does one create a quality software architecture which is maintainable in the longer-term. In the start of a project, you will need to figure out what the core requirements and functionalities are which are only derivable from your stakeholders’ expectations. In Paul Rayner’s talk about event storming, he starts off with an explanation on how to create engaging ways to find user stories. The problem with for example user story mapping, a system where stakeholders start drawing out various parts of a system to create requirements, is that even though they are highly engaging they only focus on user interaction and often ignore the background systems which can be complicated.

In comes event storming, event storming starts with placing domain events on a timeline, domain events are processes which are a fact. They can happen in a step-by-step process; they can happen on a schedule, or they happen as the result of another event. All these events are part of a domain your experts might care about. To create a timeline, you simply write events on sticky notes and put them in chronological order. Because you do not know how big your timeline might be, create enough space which will not limit creativity. If during the process a question or discussion might arise that does not involve the current topic or is not resolved within a certain time allow stakeholders to simply put a note next to the event with their questioning. This allows to see in one visual where most of the problems might arise. To get the best results get different domain experts, if everybody is a software developer your result will eventually mimic a technical event timeline.

### Requirements

User story mapping is the process of creating requirements through the e yes of the end user. These stories can than later be used in agile teams to create a sprint backlog the scale and size of each story is figured out by how far into a project you are. A single user story can later be divided into a larger set of smaller stories, the larger story will than become an epic.

### Describing initial architecture

How to describe your architecture will come down to which stakeholder you are describing it to. A non-technical stakeholder might not know how or why you are using a certain language to compile your codebase. In Simon Brown’s guide to C4 software architecture models he explains how diverse levels of model might help in describing your project to a stakeholder.

Starting with C1 level this type of diagram puts personas and users central describing the distinct parts of your project into broad strokes and how different actors and systems might interact with each other during the process.

C2 is the level where application will be visible a banking system in C1 will now be separated into a front-end webpage, a mobile app, a back-end API which connects to a database model. Furthermore, each container of the application will start being labelled with the types of codebases it will use.

In C3 the user will completely be removed from the diagram and the different containers will be dissected into separate parts. The original back-end API will now be described into different sections like the REST controller, the security and email component and an account management system. The C3 and C4 models are meant for developers and architects to describe to each other how the next phase of a project might shape out.

Lastly the C4 model, in this version various parts of a container in C3 will be described. What kind of methods and classes will be called, where will errors be logged, which kind of abstractions might be used, how do the different models interact with each other. These types of models are useful if documentation will be longer lived, and new development teams or groups will be later added or replacing the current team.

## software platforms

At the start of a project a few choices will need to be made: one of the main choices is whether a project will be relying on a framework to help with development. Choosing between frameworks comes down to a set of different trade-offs: (Stukalov, 2018)

* Range of Applicability
* Development Speed
* Manageability & Flexibility

Diagram, timeline

Description automatically generatedAccording to Stukalov you can classify frameworks into 5 categories. Low code, Full-stack, Aggregators, Narrow Focussed and No Frameworks. With each there up- and downsides, it all comes down to the early 3 trade-offs. If a project has a limited timeframe with a clear scope which doesn’t need much customizability a full-stack framework will allow you to speed up your work process and create a product at a faster speed than a pure codebase.

Figure 1 5 levels of framework

To determine all the trade-offs you can create a matrix with all the different technologies weight against each other depending on what criterium one might find important.

# Architecture & quality

## Architecture styles

### Non-FUNCTIONAL requirements NFR

First off what are functional requirements. Functional requirements describe what a system must do. Simply said they are facts: send a message, create a log, update password. Non-functional requirements don’t describe a process but describe the parameters around a requirement. The message needs to be sent within 10seconds, the password must be secured with SHA, the request needs to be 99% accurate.

### Trends and styles

There are many ways to manage your architecture, in this chapter I will compare different architectures and how they might benefit which type of application. In the conclusion of this chapter, I will defend why I might choose a certain type of architecture over something else.

#### N-layer

Graphical user interface, application

Description automatically generatedN-layer or layered architectures are architectures where physical tier of the architecture is separated with its own logical layers. These layers manage their own dependencies where a higher layer cannot be called by a lower layer. The tiers are physically separated are hosted on their own machines. Separating the tiers allows for easier scalability and makes the system more resilient but might add latency due to distance and network communication.

Reasoning to use layered architecture:

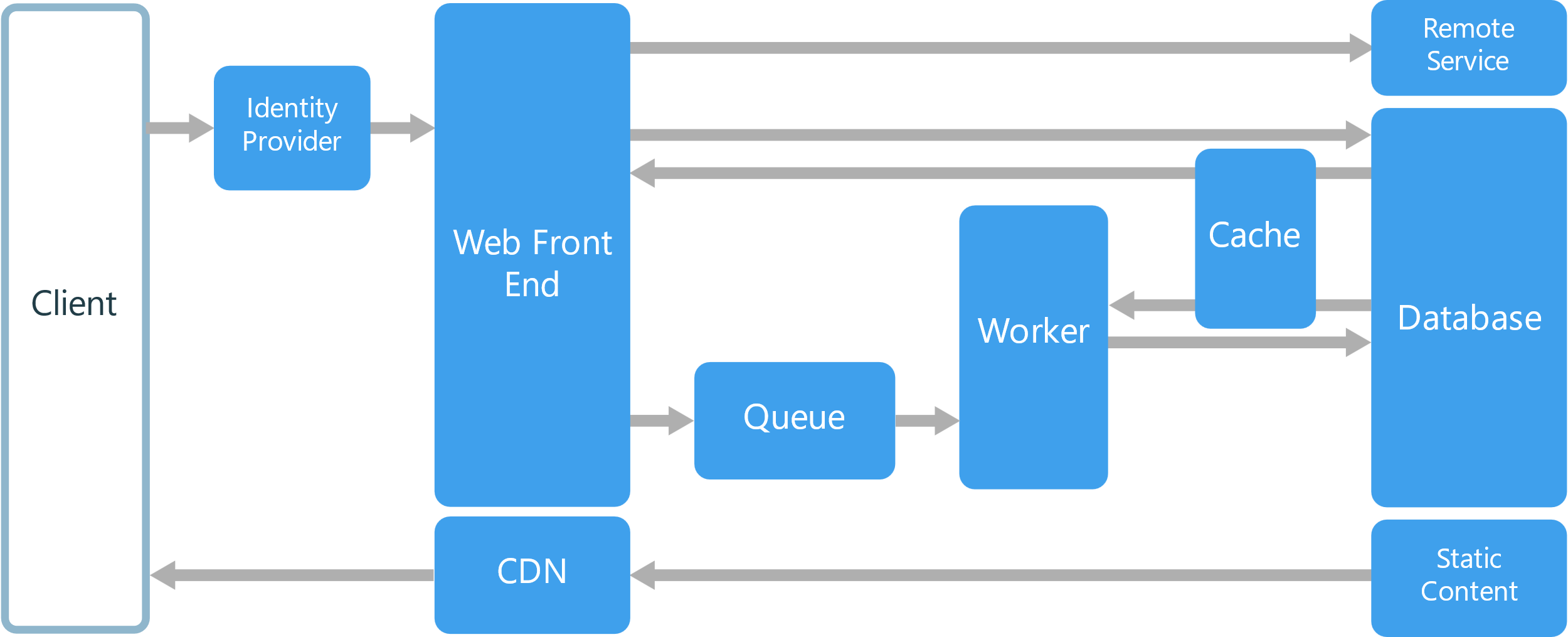
* Ease of use
* Easier to integrate with cloud platform.
* Not environment or OS bound
* Typically used in IAAS

Down sides:

* Useless middle tiers
* Incompatible with monolithic design
* Network security

#### Web-queue-worker

This architecture consists like many out of a web front end and a database, unlike comparable architectures the server requests are handled of by a worker which receives long intensive tasks or large batch jobs. Trademarks of this type of architecture are multiple databases, a quick reading cache and is often paired with a identity provider and email service. The batches and jobs the worker receive are handled asynchronously from the web front end and other workers.



* Simple architecture
* Easy to deploy and manage.
* Front end is stand alone and can be made as a SPA.
* The worker can be scaled separately.

Drawbacks:

* The front-end and workers can become too large without careful management.

#### MIcroservices

Microservice architecture is a form of architecture where each business case or part of an application a standalone separate service is. This allows for small maintainable code bases which each can be handled by a small team. A service within the architecture has no dependency on the other services within the project and thus might operate with their own models, database and codebase.

#### event-driven

A system in which events are delivered in real time, so when a consumer receives the event, it can act immediately. Due to consumers and producers being decoupled the system can scale with ease. But this can lead to problems when events must be processed in order or if the logic that’s being processed isn’t changing.

## What is an api gateway

An API gateway is a service usually used in microservices to manage, control, and monitor different microservices. The gateway acts as a facilitator between the different services allowing for it log the different use cases of API calls but also it can be used to record and analyse calls and responses both on a security level but also performance measurements.

## Quality and how to manage it?

### WHY?

The primary advantage of API monitoring is that it lets you know when your API is down or if performance has declined before your clients let you know. You can investigate and address the problems more quickly if you have the appropriate knowledge. You can identify problems before they reach production if your CI/CD pipeline includes API tracking. Without measurements you cannot improve the stability, security, or response time of an application.

### 2.3.2 What?

As discussed earlier one of the main jobs of the gateway is to monitor and analyse the different microservices around it. But which metrics should it monitor for each API. Starting with RPM or requests per minute is a metric that tracks the amount of calls an API receives per minute.

Latency is delay between the start of a call and the time of receival. Usually measured in MS the closer the number is to zero the better the measurement.

Failure rate is the percentage of calls that result into a failure or error. Based on whether the service is internal or external decisions can be made to either improve the failure rate or find alternative solutions in cases of failure.

The last three metrics are infrastructure metrics. These metrics will measure the uptime and performance of your service. API uptime is the calculations of server availability per hour or per minute. Time to First is a measurement which checks how long it takes for a service to perform the first transaction when a user first enters a webpage and lastly Memory and CPU usage of a service.

# Security by design

# Messaging

# DevSecOps

Due to the rise of AGILE working at the start of the century many companies have development teams that work more efficient, deliver faster and produce more features of higher quality than before. As a result, the more traditional way of delivering and operating the production of that code can’t keep up. To counteract this change in development speed DevOps was created and the development team became responsible for maintaining quality of production, testing and release.

The idea of integrating different parts of development into single development team where multiple developers are responsible for the whole product on its own, also fused with the security aspects of development. DevSecOps was created to make it so the whole team knew about security features and how to create security by design.

## Deployment

CI CD has become common place in most development teams where in the GIT environment automatic tests are run after a new merge request, after they have passed a new build is created which can be deployed to a server.

The deployment can happen in different steps DTAP is a common order where the first step is merging to development where most other developers can pull their features from and merge to. After they passed review, they can be deployed on a testing or test acceptance server where they can be tested by a dedicated team or customers. Lastly after all features have passed testing it can be deployed to production.

## Kubernetes and container orchestration

Due to how servers work when expanding or even creating a new deployment server it will have different configurations than another server. Containers exist to standardize these configurations and create an environment where deployment can be automized. To help with the management of different containers and their scalability you need a tool to manage them the two market leaders in this field are Kubernetes and Docker swarm.

# Cloud services

## Scalibity and Elasticity

## Virtualization

Virtualization is the act of running a single but mostly multiple virtual machines on a single hardware. This idea is the reason cloud computing works, the sharing of physical hardware. Each instance inside a machine can be its own OS, application or even a virtual infrastructure network.

## Mangement

Depending on how you want to manage the resource and security of your cloud service you want to investigate what kind of cloud services you want to use. Public cloud services like AWS manage, arrange, and provide the resources to your application, private cloud services are the opposite where you can control the data and resources yourself. This does come with a higher cost of maintenance. You can also opt into a hybrid model where you separate different parts of your application and host them on different cloud services which can communicate.

## 12-factor

## Vendor lock-in

Vendor lock-in is created when your software relies on a single service without abstraction or interfaces to handle your business and logic layers. This might lead to difficulty switching cloud services later down the line. A similar effect can happen when a part of the system already relies on a different product that’s not part of the cloud service but will make using a different service more difficult than necessary.

## services

* Data storage
* Data analysis
* Monitoring & Logging of your running applications and resource usage
* Identity & Access management
* Container Orchestration
* Virtual machines management
* Database solutions
* Autoscaling

## cost