# Utveckling av en webbapplikation för att visualisera och analysera användaraktivitet

Mattis Cedercreutz

Examensarbete
Informationsteknik
2021

| | |
|---|---|
| EXAMENSARBETE | |
| Arcada | |
| | |
| Utbildningsprogram: | Informationsteknik |
| | |
| Identifikationsnummer: | 18775 |
| Författare: | Mattis Cedercreutz |
| Arbetets namn: | Utveckling av en webbapplikation för att visualisera och analysera användaraktivitet |
| Handledare (Arcada): | Fredrik Welander |
| | |
| Uppdragsgivare: | ProdLib |
| | |

Sammandrag:
ProdLib är ett företag som utvecklar och delar 2- och 3D modeller via en lättanvänd applikation. Från applikationen kan olika tillverkares bibliotek laddas ner med deras modeller som kan sedan öppnas i design- och arkitektprogram. Företaget har ett internt verktyg som följer med vilka modeller som används och hur mycket. Eftersom företaget växer med ökande hastighet har verktyget blivit långsamt och otillräckligt . Utvecklingsarbetet var att skapa en ny webbapplikation som klarar av den ökande tillväxten, har optimerade sökningar med möjlighet för ytterliga. Syftet med arbetet var att bygga en webbapplikation för att visualisera och analysera användaraktiviteten och händelser i ProdLib applikationen. Arbetet beskriver hur utvecklingsmiljön sätts upp med användning av Docker. Arbetet går också ytligt igenom semantiken av grafdatabaser och vad de är. Slutligen beskriver arbetet vilka AWS verktyg är nödvändiga för att lansera applikationen till Amazons molnnätverk och hur de är konfigurerade med användningen av programmeringsramverket Pulumi.

Som resultat fick ProdLib en ny webbsida som använder gränssnittet Grafana med en personlig programvara kopplad för att visualisera grafer och tabeller. ProdLib fick också en grafdatabas med en API för att vidare analysera informationen. Till sist lanserades hela webbapplikationen också till AWS molnet.

Till framtidsplanerna hör skapandet av en personlig sida för varje tillverkare som visualiserar aktiviteten av deras modeller och dokument. Applikationen kan också utvecklas vidare genom att sätta upp ett månatligt rapportsystem, som skickar progressrapporter med information om hur företaget växer månatligen.


| | |
|---|---|
| Nyckelord: | ProdLib, RDF, Grafana, AWS |
| Sidantal: | 30 |
| Språk: | Svenska |
| Datum för godkännande: | |

| DEGREE THESIS | |
|---|---|
| Arcada | |
| | |
| Degree Programme: | Informationsteknik |
| | |
| Identification number: | 18775 |
| Author: | Mattis Cedercreutz |
| Title: | Developing a web application to visualize and analyze user activity |
| Supervisor (Arcada): | Fredrik Welander |
| | |
| Commissioned by: | ProdLib |
| | |

Abstract:
ProdLib is a company that creates and distributes 2- and 3D models through an easy-to-use application. Users can download manufacturer libraries containing models of their products, which can then be opened from the application to design or architectural software's, like Revit or AutoCAD. The company has an internal tool to follow and visualize the usage of models. The rapid growth of the company has caused the tool to be slow and lacking necessary properties. The development work was to create a new better web application that can handle the increasing growth of the company, has optimized querying of data with possibilities to further develop new properties. The purpose of this thesis was to develop and build a web application to visualize and analyze user activity. The thesis work describes how to set up the development environment using Docker, studies the semantics of graph databases and what they are. And lastly describes the configuration of necessary services to publish the application to AWS cloud using Pulumi.

As a result, ProdLib got a new web application that uses Grafana with a private data source plugin to visualize graphs and tables. They also got a graph database with an API to further analyze the data. Lastly as a result the web application was published in AWS.

Future development would include creating personalized dashboards for each manufacturer that visualizes usage of their models and documents. The application can also be further developed by adding a monthly reporting system, that sends progress reports to follow company growth.



| Keywords: | ProdLib, RDF, Grafana, AWS |
|---|---|
| Number of pages: | 30 |
| Language: | Swedish |
| Date of acceptance: | |

| | |
|---|---|
| OPINNÄYTE | |
| Arcada | |
| | |
| Koulutusohjelma: | Tietotekniikka |
| | |
| Tunnistenumero: | 18775 |
| Tekijä: | Mattis Cedercreutz |
| Työn nimi: | Verkkosovelluksen kehitys visualisoimaan ja analysoimaan käyttäjätilastoja |
| Työn ohjaaja (Arcada): | Fredrik Welander |
| | |
| Toimeksiantaja: | ProdLib |
| | |


Tiivistelmä:

ProdLib on yritys, joka luo ja jakaa 2- ja 3D malleja helppokäyttöisestä sovelluksesta. Käyttäjät voivat ladata sovelluksen kautta kirjastoja valmistajien tuotemalleista. Näitä malleja voi avata suoraan mallinnus- ja arkkitehtisovelluksista, kuten Revit tai AutoCAD. Yrityksellä on sisäinen työkalu, jolla seurata ja visualisoida mallien käyttömääriä, mutta yrityksen nopean kasvun myötä työkalusta on tullut hidas ja siitä puuttuu tarpeellisia toiminnallisuuksia.

Kehitystyön tarkoitus oli luoda uusi, parempi työkalu, joka hallitsee yrityskasvun, sisältää optimoituja tiedonhakukutsuja ja mahdollistaa uusien toiminnallisuuksien kehityksen. Tämän lopputyön tarkoitus oli suunnitella ja rakentaa verkkosovellus visualisoimaan ja analysoimaan mallien käyttömääriä sekä sovelluksen käyttäjämääriä. Työ kuvailee miten kehitysympäristö on rakennettu käyttäen Dockeria, tutkii pinnallisesti graafitietokantojen semantiikkaa sekä esittää niiden merkityksen. Viimeiseksi työ käy myös läpi, miten nettisivun toiminnallisuuteen vaikuttavat AWS palvelut määritellään käyttäen Pulumia.

Tuleviin kehitys suunnitelmiin sisältyy valmistajien tuotetilastotietojen esitys omilla sivuilla jolloin jokaisen valmistajan mallien käyttömääriä voidaan erikseen seurata. Sovellukseen voidaan lisätä myös kuukausittainen raportointi systeemi, mikä lähettää tilastotietoja yrityskasvun seuraamista varten.



| | |
|---|---|
| Avainsanat: | ProdLib, RDF, Grafana, AWS |
| Sivumäärä: | 30 |
| Kieli: | Ruotsi |
| Hyväksymispäivämäärä: | |

# INNEHÅLL

## FIGURER



## Tabeller

# 1 INLEDNING

I dagens läge har det skapats, kopierats och lagrats över 97 zettabyte med information i världen. Det är likvärdigt med 97 triljoner gigabyte. Enligt förutsägelsen antas det stiga till 181 ZB vid året 2025 (Statista, 2022). Digital information är så förankrad i varje aspekt av vårt liv och samhälle, att det värker vara omöjligt att stoppa växten av informationsproduktion. Varje dag skickas det över 500 tusen nya inlägg i Twitter, 4 petabyte av data genereras av Facebook och vid året 2024 ökar mängden e-post skickat per dag till 361 miljarder (Bulao, 2021). All den information som skickas och sparas har ändå stor nytta för företag. Analysering av produkträckvidd och statistik hjälper företagen följa med deras framgång och tillväxt. Statistiken hjälper företagen att göra beslut och ändringar för att nå önskade resultat. Jag anser att sådan personlig information som sparats av användaren inte borde få säljas vidare till andra företag, utan informationen borde enbart vara ett verktyg för företag att följa med sin utveckling.

## 1.1 Bakgrund

ProdLib är ett företag som utvecklar och delar BIM och CAD-modeller. BIM (Building information model) är digitala representationer av byggda artefakt med syftet att bilda en pålitlig bas för beslut och att främja design, bygg- och operationsprocesser. ProdLib fungerar som en kanal till arkitekter och designers via en applikation som installeras och sedan körs från modellerings- och arkitektprogram. När användaren öppnar modeller eller dokument från ProdLib-applikationen sparas anmärkningar av inlägget och händelsen loggas in till en databas. Med den här informationen kan företaget följa med hur tillverkarens produkter används. Eftersom ProdLib växer med ökande hastighet har den nuvarande applikationen, som utvecklades för flera år sedan, blivit långsam och applikationen önskas ha mer egenskaper för att följa med företagets utveckling. Till exempel statistik om nya användarregistreringar och applikationens installationsmängder.

## 1.2 Syfte och mål

Syftet med den praktiska delen av examensarbetet har varit att producera en webbapplikation för företaget ProdLib. Applikationens avsedda användning är att analysera och visualisera användaraktivitet. Informationen som sparas när modeller laddas ner och öppnas i modellerings program kan analyseras för att följa med hur tillverkarens produkter används och var. Det finns flera olika webbverktyg för att visualisera informationen och Google Analytics är en av dom mest använda verktygen för analysvisualisering. ProdLib valde ändå att gå med verktyget Grafana, ett liknande webb-analytics verktyg som är totalt gratis med öppen källkod. Grafana möjliggör också programmering av programvara för egna datakällor, vilket var nödvändigt för ProdLib att visualisera statistiken av användaraktivitet. Grafanas gränssnitt är också visuellt trevligare än Google Analytics. Applikationens syfte är att vara ett verktyg för att följa med hur företaget växer och med informationen hur modeller laddas kan ProdLib marknadsföra vidare till nya företag.

Målsättningen med examensarbetets teoretiska del är att beskriva hur utvecklingsmiljön är strukturerad och hur AWS-molninfrastrukturen konfigureras. Utöver det här ges även en översikt hur grafdatabasstrukturen och sökningsmetoder ger möjlighet att undersöka relationer mellan information mycket djupare.

## 1.3 Metoder och avgränsningar

Jag planerade utgångspunkterna till projektet tillsammans med min kollega Paul Villavicencio och vi beslöt att använda Grafana-ramverket för att visualisera aktivitetsinformationen. För att analysera informationen och skicka den vidare till Grafana frontend behövs en API. Jag beslöt att programmera API:t med NodeJS och Express, eftersom NodeJS är mycket populärt programmeringsramverk som är designad för att bygga webb-applikationer och Express underlättar programmering av API strukturen med hjälp av MVC (Model-View-Controller) standarden. Grafana kan inte direkt visualisera den råa information som sparats i databaser utan det kräver en programvara, en så kallad ”datasource plugin”, för att formatera data. Vi beslöt också att sätta upp en graf-

databas för strukturinformationen av ProdLib-applikationens bibliotek. Dessa bibliotek har rekursiva kategorier, artiklar, listor, egenskaper, tabeller och annat som innehåller relationer och förbindelsen med varan och därför ger en grafdatabas möjlighet att analysera strukturen och innehållen på ett djupare sätt. Jag valde att börja utvecklingsprocessen enbart lokalt med användning av Docker och sedan överföra applikationsmiljön steg för steg till företagets AWS moln.

Eftersom Typescript transkompileras (när ett programmeringsspråk läses in och ändras till ett annat) till Javascript så går jag inte djupare in på språkets för och nackdelar i arbetet. Av säkerhetsskäl för företaget kommer jag att inte beskriva strukturen av Grafanas programvara som formaterar informationen från databasen och självaste strukturen av API:t. De är byggda på ett standard sätt och angår för det mesta enbart företaget.

## 1.4 Struktur

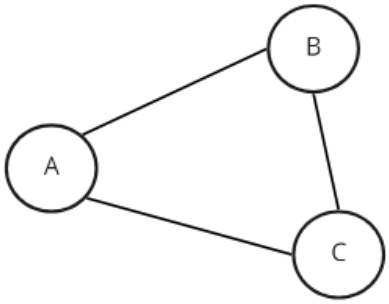
I nästa kapitel beskriver arbetet ytligt grafdatabaser och vad grafteori är. Kapitel 3 riktar läsaren till applikationens utvecklingsmiljö och hur den sätts upp med *docker-compose*. Sedan i den sista delen går arbetet igenom hur man lanserar alla de olika nödvändiga system i AWS. Av säkerhetsskäl döljer jag känslig information och beskriver mer ytligt hur de olika system är utvecklade.

# 2 GRAFDATABAS

En grafdatabas använder grafteori för att lagra, kartlägga och fråga relationer. Det är huvudsakligen en samling av kanter och noder, där varje nod representerar en enhet såsom en person eller en organisation och varje kant representerar en koppling eller relation mellan två noder. En nod definieras av en unik identifierare och har flera kanter kopplade till sig, oavsett om den är inkommande eller utgående, och den har egenskaper som uttrycks som en kollektion av nyckel-värde par. En kant definieras också av en unik identifierare, och den har en start- och en slut nod, såväl som en kollektion av egenskaper. En grafdatabas kan användas för analys av relationer och förbindelser mellan data, därför används den mycket i socialmediadata, för data med dynamiska scheman, och i försäljning, där det används för att analysera kopplingar mellan kunders handlingar. (Techopedia, 2021)

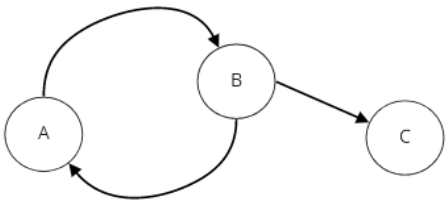
## 2.1 Grafteori

Grafteori är studiet av matematiska objekt strukturerade av linjer och punkter, *grafer*. Punkterna i grafen kallas noder och linjer kallas kanter. Varje kant har en uppsättning av en eller två noder kopplade till sig (Jonathan L. Gross, 2020). Noder och kanter kan ha egenskaper som påverkar grafens funktionalitet och struktur. De olika egenskaper innebär bland annat vikt, direktion eller riktning. Beroende på vad grafen representerar och uttrycker har noderna och kanterna olika egenskaper. En standardgraf definieras som en serie av noder och kanter med ända egenskapen att de är antingen kopplade med varan eller inte (se figur 1). (Flovikx, 2020)



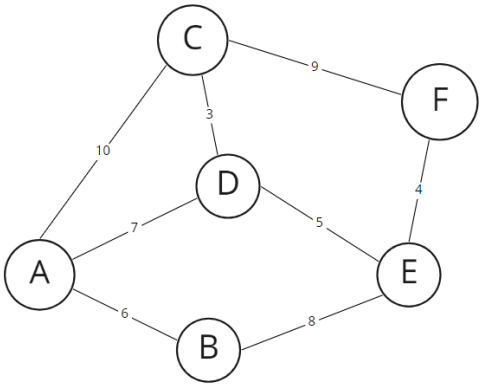

*Figur 1 Exempel av en graf*

När en graf definieras med riktade kanter från en punkt till en annan kallas det en *digraf*. Det betyder att om en kant är riktad från B till C så kan man röra sig endast från B till C (se figur 2). (Flovikx, 2020)




*Figur 2 Exempel av en simpel digraf*

Om längden av kanten har betydelse kallas den en *viktad graf* och den kan ha riktade eller oriktade kanter. Vikten i grafen kan representera olika saker beroende på vad grafen representerar. Om grafen nedan (se figur 3) representerar ett tågnätverk, så kantens vikt kan representera till exempel kostnaden av att resa från nod till nod. (Flovikx, 2020)



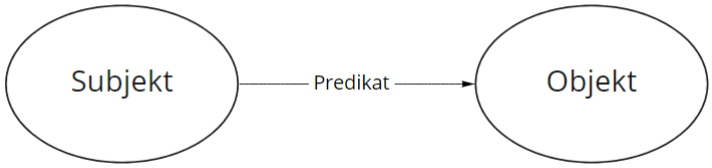

*Figur 3 Exempel av en viktad graf*

## 2.2 RDF

RDF (Resource Description Framework) är ett riktat, markerat grafdataformat som representerar information i nätet. RDF används ofta för att representera personlig information, sociala nätverk, metainformation angående digitala artefakt samt för att förse integration av separata källor med information. (Semantics, 2014) Följande kapitel definierar syntaxen och semantiken av SPARQL söknings språket för RDF.

### 2.2.1 Definition

Den underliggande strukturen av varje uttryck i RDF är ett set av så kallade *RDF-tripplar*, vilka konsisterar av ett subjekt, en predikat och ett objekt. En serie av RDF-tripplar kallas en RDF graf och en serie av noder i grafen är serien av subjekt och objekt. (W3C, 2014)




*Figur 4 Strukturen av en trippel pattern*

Varje trippel representerar ett uttryck av en relation mellan element betecknade av noder. Riktningen av predikaten är viktig för att den alltid riktar mot objektet.

### 2.2.2 Syntax

RDF-tripplar skrivs som ett blanksteg separerad lista av ett subjekt, predikant och objekt. Med SPARQL söknings-programmeringsspråket kan RDF-tripplar sökas enligt exemplet (se figur 5), där subjekten är *:book1*. Predikatet är *dc:title* och *$title* är variabelnamnet för objektet som returneras från sökningen. (W3C, 2014)

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
PREFIX  : <http://example.org/book/>

SELECT  $title
WHERE   { :book1  dc:title  $title }
```

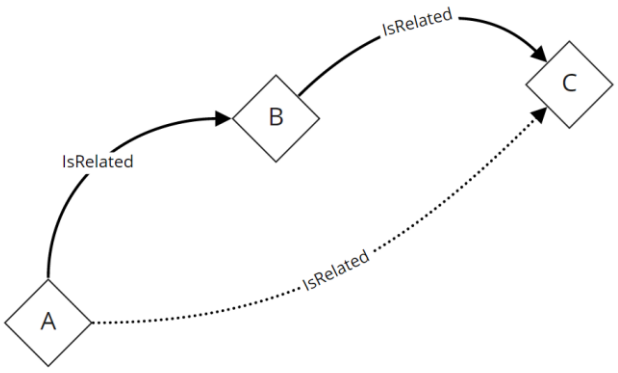
*Figur 5 Exempel hur RDF-tripplar söks med SPARQL*

## 2.3 Ontologin

Med enkla ord kan man definiera ontologi som studiet av vad som finns, men en annan del av definitionen som uttrycker mer mening i denna kontext, är att ontologi omfattar egenskaper och relation hos de entiteter som existerar. När man pratar om entiteter och deras relationer, refereras ontologi som *formell ontologi.* Dessa är teorier som försöker ge exakta matematiska formuleringar av egenskaper och relationer i vissa entiteter. Sådana teorier föreslår vanligtvis axiom om dessa enheter, skrivna på något formellt språk baserat på något system av formell logik. (Vázquez, 2018)

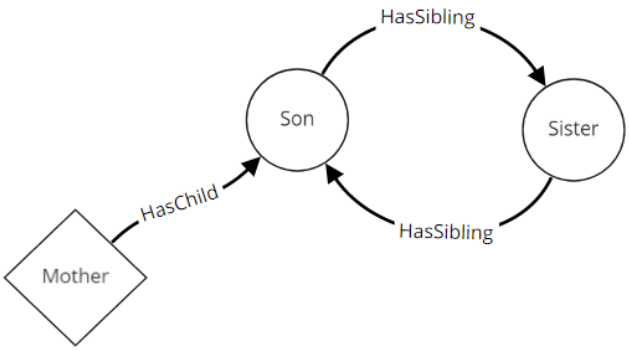
OWL (Web Ontology Language) är designad att användas för applikationer som bearbetar informationen i stället för att enbart presentera informationen till användaren. OWL är ett beräkningsmässigt logikbaserat programmeringsspråk med ändamålet att information uttryckt i OWL kan läsas av datorprogram, websidor och databasinstrument. OWL dokument kallas ontologier, som kan publiceras till internet och sedan refereras av andra ontologier. (Web, 2012) En ontologi beskriver djupare om grafdatabasens struktur och nodernas relationer. OWL ontologin erbjuder klasser, individer, attribut och referenser med vilka grafdatabasens egenskaper kan beskrivas.

Som exempel kan en graf konfigureras med attributen transitiv. Transitivt betyder att om subjekt A är besläktad till objekt B med predikat *IsRelated* och subjektet B har relation till objekt C med samma predikat *IsRelated*, då transiterar egenskapen *IsRelated* likmässigt från A till C (se figur 6).




*Figur 6 Exempel av transitiv predikat i grafen*

Grafens noder kan också specificeras som reflexiva. Då reflekterar objektet inriktade predikaten tillbaka till subjektet (se figur 7).




*Figur 7 Exempel av refexiv attribut i grafen*

Relationer mellan noder kan beskrivas noggrant med mycket flera relations egenskaper och det möjliggör hög komplexitet i databasstrukturen.

### 2.3.1 Varför behövs en ontologi?

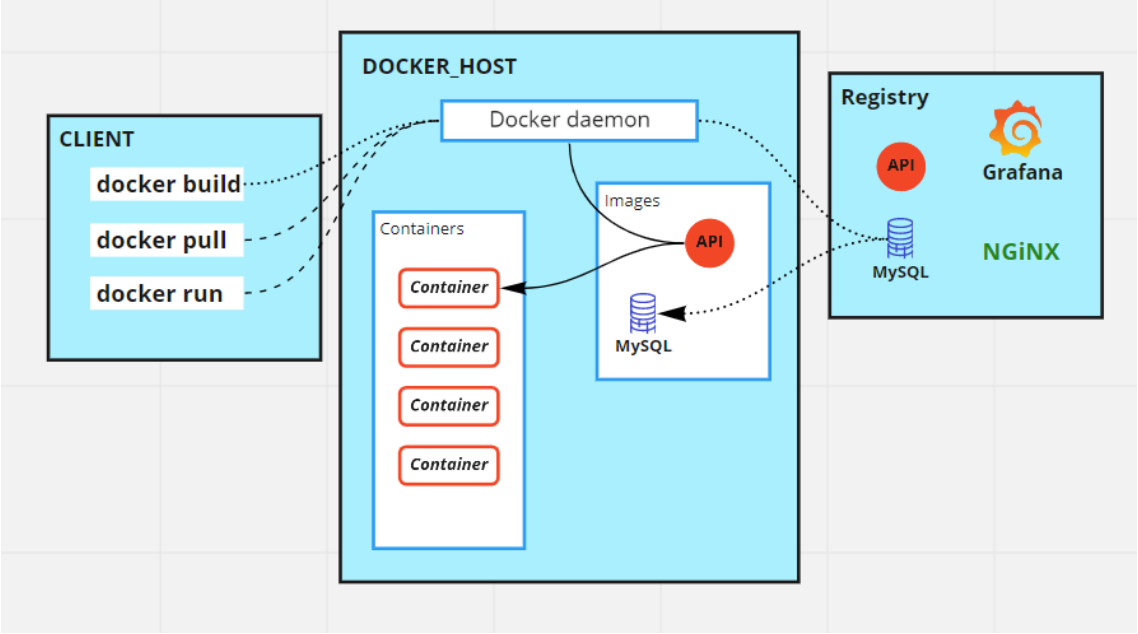
Ontologin är inte någonting som är nödvändigt för att visualisera process och användar-händelser i applikationen. ProdLibs datastruktur för tillverkarens bibliotek möjliggör användning av komplexa relationer mellan subjekt och objekt och det orsakar att in-formationen inte lönas lagras enbart med RDF-tripplar utan en ontologi skapas för att bearbeta och beskriva förbindelsen och relationer i datastrukturen.

# 3 UTVECKLINGSMILJÖN

För applikationens utveckling skapas en miljö lokalt som innehåller alla element nödvändiga för att köra applikationen. Olika enheter som utvecklas och sätts upp är: Grafana med privat programvara, API, grafdatabas och MySQL databas.

## 3.1 Docker-infrastrukturen

Docker använder en client-server arkitektursprincip. Docker-klienten pratar med Docker-daemon enheten, vilket styr hur Docker-containers byggs, körs och distribueras. Docker klienten och daemon kan antingen köras i ett system eller med förbindelse till en avlägsen Docker-daemon. Docker-klienten och daemon-enheten kommunicerar med användning av en REST API över UNIX-uttag eller nätverksgränssnitt. En annan Docker klient är Docker-Compose, som underlättar arbete med applikationer som innehåller flera Docker-containers.




*Figur 8 Docker infrastrukturen*

Följande Docker-begrepp citerade från källan (Docker, 2021)

**Docker Daemon** lyssnar på Docker-API begäran och hanterar Docker-bilderna, containers, nätverk och volymer. En Docker Daemon kan kommunicera med andra daemonenheter.

**Docker Client** är det primära sättet användaren interagerar med Docker. När kommandot *docker run* körs, skickar klienten kommandot vidare till *dockerd* enheten, som utför det. Docker-klienten kan kommunicera med mer än en daemon.

**Docker Desktop** är en applikation som hjälper bygga, dela och köra applikationer lätt från Mac eller Windows operativsystem. Docker desktop hanterar den komplexa systeminställningen och låter en fokusera mer på självaste programmeringen.

**Docker-register** förvarar Docker-bilder. Docker hub är ett allmänt register vilket innehåller bilder som vem som helst får använda och Docker söker som standard bilder från hubben, men om den hittar inte så söker den bilden från användarens privata register.

**Docker-bild** är en skrivskyddad enhet med instruktioner för byggande av en Docker container. Oftast är en bild baserad av en annan bild, med tilläggs konfigureringar. Som exempel kan en bild byggas baserad av en Ubuntu-bild, men den installerar Apache webbservern, din webbapplikation och såväl som konfigureringsdetaljer som behövs för att köra applikationen. För att skapa och konfigurera en egen bild, skapas en *Dockerfile* som innehåller simpel syntax för att definiera de steg som krävs att bygga och köra bilden. Varje instruktion i Docker-filen skapar ett lager i bilden. När man modifierar Docker-filen och bygger om bilden, ändras endast de lager som har blivit modifierade. De orsakar att bilderna är små, lätta och snabba jämfört med andra virtualiseringstekniker.

**Docker container** är en körbar instans av en bild. En container kan skapas, startas, stoppas, flyttas eller tas bort med Docker-API. Man kan ansluta en container till en eller mera nätverk, koppla lagring till den eller skapa en ny bild baserat på dess nuvarande läge. Som standard är containers relativt isolerade från andra containers. Man kan defi-

niera hur isolerad containers nätverk, lagring eller andra subsystem är från de andra containers. En Docker container är definierad av dess bild samt med eventuella konfigurationsalternativ definierade när bilden byggs eller startas.

### 3.1.1 docker-compose

Compose är ett verktyg för att definiera och köra multi-container Docker-applikationer. Multi-container betyder att man kan köra flera isolerade Docker containers i en samma enhet. För docker-compose används YAML-filer för att konfigurera applikations egenskaper. Med filen kan sedan en docker container startas med alla applikationer samtidigt.
Det finns tre viktiga steg för att använda compose:

1. Definiera applikations miljön med en Dockerfile fil, så att den kan återanvändas var som helst.
2. Definiera omgivningen och applikations egenskaper i en docker-compose.yml fil.
3. Kör kommandot docker-compose up och Docker startar hela applikations miljön.

## 3.2 Docker installation

Följande kapitel beskriver vilka Docker-bild konfigureras i den lokala utvecklingsmiljön med användning av Docker compose och Docker-filar. De olika applikationer som skapas till bilder beskrivs och listas i tabell 1.

*Tabell 1*

| | |
|---|---|
| Grafana | Användargränssnit för att visualisera och analysera grafer och tabeller |
| API | Applikations protokoll som analyserar informationen från databasen och skickar vidare till Grafana. |
| Grafdatabas | Databas för att förvara information av ProdLib applikationen. |
| MySQL databas | Databas för att förvara information av användaraktivitet samt inlägg från ProdLib applikationen. |

För varje verktyg definieras applikations principer i docker-compose filen. Dessa konfigureringar definierar applikationens struktur och egenskaper. Konfigureringen kan bland annat innehålla information av lagringen, öppna specifika portar för anslutningar och definiera miljövariabler. Compose kan sedan starta upp instanserna med kopplingar till varandra i en container. Relationsdatabasen MySQL konfigureras med att definiera användarinformation för databasen samt en volym för att spara data. Docker volymen är en lagringsenhet dit information sparas av instansen, även om docker containern stängs eller raderas så stannar informationen i volymen. (se bilaga 1)

För utvecklingsmiljöns grafdatabas används Blazegraph. Blazegraph är en öppen källkods databassystem som kan köras i en fristående server som stöder grafdatabasstrukturen RDF med söknings språket SPARQL.

För API:t skapas en *Dockerfile* fil som innehåller konfigureringar hur API-bilden skapas. Det kräver en samling av attribut, miljövariabler och kommandon som körs vid byggskede av bilden (se figur 9).

```
FROM node:14

# Definiera mapp för applikationen
WORKDIR /app

# Spara miljövariablar
ENV PORT=7000

ENV MYSQL_DB_HOST="http://db:3306/"
ENV MYSQL_DB_USER="user"
ENV MYSQL_DB_PWD="password"
ENV MYSQL_DB_NAME="database"
ENV MYSQL_DB_PORT="3306"

ENV BLAZE_DB_URL="http://blazegraph:9999/blazegraph/sparql"
ENV BLAZE_DB_PREFIX="PREFIX prodlib: <http://www.example.com/Prodlib_ontology#>"
ENV BLAZE_DB_BASEIRI="http://www.example.com/Prodlib_ontology#"

# Kopiera applikations konfigurerings filar
COPY package*.json ./
COPY tsconfig.json ./

# Kopiera källkod till /app/src mappen
COPY src /app/src

# Installera node paket
RUN npm install
# Kompilera applikationen
RUN npm run build

# Rikta applikationen till en port
EXPOSE 7000
```

*Figur 9 Dockerfile specificerar attributer och miljövariabler till bilden*

Läsaren lägger märke också hur i konfigureringen (se figur 9) applikationen refererar de andra instansen i Docker containern. I stället för att referera den lokala enheten med rutten localhost, så refereras både MySQL databasen och Blazegraph databasen med namnet som specificeras i docker-compose filen. (se bilaga 1)

Efter det konfigureras självaste visualiserings gränssnittet Grafana. Bilden för grafana finns på den allmänna Docker hub registret och i compose filen konfigureras lagrings egenskaper samt applikationsporten för bilden. I konfigureringen definieras också en volym som refererar till en lokal mapp där privatutvecklade programtillägg ligger. Vid behov kan allmänna programtillägg tilläggas med att definiera miljövariabeln *GF_INSTALL_PLUGINS* med en lista av allmänna programvara. Detta gör ändring till grafanas konfigurerings fil *grafana.ini* när bilden byggs upp och installerar listan av programvara till applikationen. (se bilaga 1)

# 4 AWS

Följande kapitel beskriver hur webbapplikationen sätts upp i AWS (Amazon Webb Services) molnet. AWS är den största leverantören av molntjänster i världen, med över 200 produkter och servicen i molnteknologi. Molntjänster kan kategoriseras i tre huvudkategorier. Varje typ av molntjänst tillhandahåller olika nivåer av kontroll, flexibilitet och förvaltning.

1. Infrastrcture as a Service (IaaS)
    a. IaaS innehåller bas byggsten för en moln infrastruktur. Vanligtvis erbjuder det tillgång till olika nätverkstjänster, datorer och datalagrings plats. IaaS ger den högsta nivån av flexibilitet och hantering av IT resurser.
2. Platform as a Service (Paas)
    a. PaaS tar bort behovet att hantera den underliggande infrastrukturen och tillåter en att fokusera på utvecklingen och mer av hantering av applikationer.
3. Software as a Service (Saas)
    a. SaaS erbjuder en fullständig produkt som körs och hanteras av tjänsteleverantören. I de flesta fallen när man pratar om SaaS så refererar man slutanvändarens applikationer, som webbaserad e-post. Med SaaS behöver man inte tänka på hur tjänsten eller infrastrukturen hanteras utan enbart på applikationens funktionalitet.

(Amazon Cloud Computing 2020)

AWS verktyg sätts upp med användning av deras kontrollpanel eller sedan med ett programmerings ramverk kallad Pulumi. Med Pulumi kan hela infrastrukturen beskrivas med ett programmeringsspråk som Typescript.

## 4.1 Vad är Pulumi?

Pulumi är ett IaC-verktyg med öppen källkod för att designa, distribuera och hantera resurser i molninfrastruktur. Pulumi används för att skapa infrastrukturelement som virtuella maskiner, nätverk och databaser. Verktyget används också för att designa moderna molnkomponenter, kluster och serverlösa funktioner. Till skillnad från Terraform så använder Pulumi riktiga programmeringsspråk för utveckling. Med Pulumi tvingas man inte lära ett nytt programmeringsspråk bara för att hantera infrastruktur, utan konfigurationsfiler kan skrivas i Python, JavaScript eller TypeScript. (Danicic, 2020)

```
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";

const bucket = new aws.s3.bucket("b", {
	acl: "private",
	tags: {
		Environment: "Dev",
		Name: "My bucket"
	}
})
```

*Figur 10 Exempel hur en S3 enhet definieras med Pulumi i Typescript.*

Terraform är ett deklarerande IaC-verktyg. Användare skriver konfigurationsfiler för att beskriva de nödvändiga komponenterna till Terraform. Verktyget genererar sedan en plan som beskriver de steg som krävs för att nå önskat tillstånd. Om användaren accepterar dispositionen, utför Terraform konfigurationen och bygger den önskade infrastrukturen. (Velimirovic, 2020)

```
resource "aws_s3_bucket" "b" {
	bucket = "my-tf-test-bucket"
	acl = "private"

	tags = {
		name = "My bucket"
		Environment = "Dev"
	}
}
```

*Figur 11 Exempel hur en S3 definieras med Terraform*

## 4.2 Installation av applikationsmiljön

Följande kapitel går igenom hur den utvecklade webbapplikationen lanseras i AWS molntjänsten. Webbapplikationen är en del av ProdLib AWS infrastruktur och för att säkerställa företagets säkerhet går arbetet enbart igenom de verktyg som är nödvändiga för applikationens funktionalitet.

De olika verktygen listas och beskrivs i tabell 2.

*Tabell 2*

| | |
|---|---|
| VPC | Ett virtuellt nätverk i AWS molnet där verktygen lanseras och kommunicerar med varandra |
| AutoScalingGroup | En samling av EC2 instanser som kan automatiskt skalas och hanteras |
| Neptune | Graf databastjänst i AWS målnvercket |
| S3 | Objekt lagringstjänst i AWS målnet |
| RDS | Relationsdatabas |
| Load balancer | Distribuerar inkommande programtrafik över EC2-instanser i tillgänglighetszoner |

Virtuella privata molnet (VPC) tillåter en att starta AWS-resurser i ett definierat virtuellt nätverk. VPC är som ett traditionellt datacenter med servrar och lagrings system, men fördelen att ha alla AWS verktyg till förfogande och en lätt skalbar infrastruktur. I konfigureringen definieras attributen *cidrBlock* med värdet ”10.0.0.0/16” vilket specificerar en räckvidd av IP-adresser till privata molnet (se figur 12).

```
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";

const main = new aws.ec2.Vpc("main", {
    cidrBlock: "10.0.0.0/16",
});
```

*Figur 12 Exempel hur VPC konfigureras*

Webbapplikationen kan köras i en EC2 instans med både Grafana visualiserings plattformen och API:t i samma maskin. För att sätta upp applikationerna i instansen behövs följande förberedelser.

### 4.2.1 Installation av Grafana

Grafana är en webbapplikation för att köra dataanalys, visualisera statistik och för att övervaka applikationer med hjälp av paneler. Eftersom Grafana är en öppen källkodslösning, kan privata programvara utvecklas från grunden för integration med flera olika datakällor. Grafana kan installeras på Windows, macOS, Ubuntu eller köras rakt från en Docker bild. EC2 instansen som används i arbetet är en Ubuntu maskin och Grafana installeras på följande sätt.

Ladda ner till installations paketet för senaste stabila version:

| |
|---|
| sudo apt-get install -y apt-transport-https |
| sudo apt-get install -y software-properties-common wget |
| wget -q -O - https://packages.grafana.com/gpg.key \| sudo apt-key add - |
| echo "deb https://packages.grafana.com/oss/deb stable main" |
| sudo tee -a /etc/apt/sources.list.d/grafana.list |
| |

Efter det installera Grafana:

| |
|---|
| sudo apt-get update |
| sudo apt-get install grafana |

Och sedan kan Grafana servern startas i standardporten *3000*.

```
sudo service grafana-server start
```

### 4.2.2 Installation av API

För API:t skapas en katalog i EC2 instansen dit källkoden kopieras från en S3 lagringsenhet. API:t byggs sedan upp enligt samma principer som krävs för att köra den lokalt. I detta fall användes inte Docker utan applikationen kan köras rakt med användning av *Node* eller *PM2* (PM2 är ett daemon process hanterare som kan starta och uppehålla applikationer 24/7).

```
mkdir ./backend-API
aws s3 cp s3://namnet_av_s3_bucket/mappen_till_källkoden ./backend-API
cd ./backend-API
npm install
tsc
pm2 start build/index.js
```

### 4.2.3 EC2 instanskontroll

AWS verktyget AutoScalingGroup möjliggör gruppering av EC2 instanser och underlättar arbetet när instansens storlek eller konfigureringar måstes ändras. Med automatiskt skalande grupper kan man specificera en AWS Launch template, lanserings principer som utförs då när instansen är skapad eller startas för första gången. Eftersom applikationen som körs i instansen kräver förberedelser för att starta, definieras en AutoScalingGroup med lanseringsprinciper i AWS molnet.  Följande figurer i arbetet beskriver hur systemet konfigureras.

```
const amiId = aws.ec2.getAmi({
	filters: [
	{
		name: "name",
		values: ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"],
	},
	{
		name: "virtualization-type",
		values: ["hvm"],
	},
	],
	mostRecent: true,
	owners: ["099720109477"],
});
```

*Figur 13 Exempel konfigurering av EC2 med Pulumi*

Hur lanseringsmallen med automatiskt skalande gruppen konfigureras (se figur 14).

```
const launchTemplate = new aws.ec2.LaunchTemplate('aws-ec2-launchTemplate',
	{
		imageId: amiId.id,
		instanceType: "t3.medium",
		keyName: "ec2AccessKeyPair",
		iamInstanceProfile: { arn: iamp.arn },
		vpcSecurityGroupIds: securityGroupIds,
		updateDefaultVersion: true,
		userData: usrData,
	});
const autoScalingGroup = new aws.autoscaling.Group('application-production-asg', {
	desiredCapacity: 1,
	maxSize: 2,
	minSize: 1,
	launchTemplate: {
		id: launchTemplate.id,
		version: '$Latest'
	},
	targetGroupArns: [targetGroup.arn],
	vpcZoneIdentifiers: subnetIds
});
```

*Figur 14 Exempel konfigurering av EC2 Launch template*

Läsaren lägger märke till argumentet *userData* i lanseringsmallen (se figur 14). Värdet till detta är ett objekt vilket specificerar de lanserings principer som skall utföras när EC2 instansen är startad. Föregående kapitlet beskrev vilka kommandon är nödvändiga för att installera och starta applikationerna. Följande figur (se figur 15) visar exempel av objektets struktur med installations argument för Grafana.

```
import { createUserData } from "pcloudinit";

const userData = createUserData(["install_grafana"],
{
  "install_grafana": {
    commands: {
      "00_install_transport_https": {
            command: "sudo apt-get install -y apt-transport-https"
            },
      "01_install_software_properties": {
            command: "sudo apt-get install -y software-properties-common wget"
      },
      ...
   }
}
```

*Figur 15 Lanserings principer för Launch template*

## 4.3 Neptune

Neptune är en grafdatabas service som underlättar körande och byggande av applikationer med högt koncentrerade databasstrukturer. Kärnan av Neptune är en kraftig högt presterande databasmotor, vilket stöder söknings programmeringsspråk som Gremlin och W3C SPARQL. (Neptune, 2021) Följande kapitel går djupare igenom viktiga egenskaper av installationen av Neptune-verktyget.

### 4.3.1 Installation av Neptune

För applikationen sätts upp en Neptune cluster vilket innehåller grafdatabas instansen. Viktiga nyckelkomponenter av servicen är:

- Primära databas instansen – stöder sök och skriv operationer, och utför alla datamodifikationer till Neptune cluster volymen.
- Neptune replica – Ansluter till samma lagrings volym som primära databas instansen och stöder enbart söknings operationer. Varje Neptune DB cluster innehåller upp till 15 repliks instanser därutöver den primära databas instansen.
- Clustervolymen – Lagrings system designad för stabilitet och hög tillgänglighet, dit Neptune data lagras. En clustervolym innehåller kopior av data över flera tillgänglighetszoner i en AWS region. Eftersom informationen automatiskt replikeras över tillgänglighetszoner är den mycket hållbar och det finns liten risk för dataförlust.

(Neptune, 2021)

ProdLibs biblioteks struktur förvandlas till en ontologi och den kan laddas upp till Neptune via S3 lagring. Informationen kan inte laddas upp till instansen direkt utan den måstes gå via en VPC endpoint. De tillåter en säker förbindelse med AWS Glue via privata nätverk till S3 instansen utan någon uppvisande till den allmänna nätverk.

```
const neptuneCluster = new aws.neptune.Cluster("neptune-cluster", {
      clusterIdentifier: "neptune-cluster-demo",
      engine: "neptune",
      backupRetentionPeriod: 5,
      preferredBackupWindow: "07:00-09:00",
      skipFinalSnapshot: true,
      iamDatabaseAuthenticationEnabled: true,
      applyImmediately: true,
      vpcSecurityGroupIds: securityGroupIds,
      neptuneSubnetGroupName: subnetGroupName,
});

const neptuneClusterInstance = new aws.neptune.ClusterInstance("neptune-cluster-
instance", {
            clusterIdentifier: neptuneCluster.id,
            engine: "neptune",
            instanceClass: "db.r4.large",
            applyImmediately: true,
            neptuneSubnetGroupName: subnetGroupName
      }));
}

const s3Endpoint = new aws.ec2.VpcEndpoint("s3-endpoint", {
      vpcId: aws_vpc.main.id,
      serviceName: "com.amazonaws.us-west-2.s3",
      routeTableIds: routeIds
});
```

*Figur 16 Konfigurering av Neptune cluster med VPC endpoint*

Läsaren lägger märket hur konfigureringen av Neptune instansen innehåller referens till säkerhetsgrupper och VPC Endpoint refererar en rutttabell (se figur 16). Säkerhetsgruppen tillåter anslutningar från och till instansen via specificerade IP adresser och portar, medan rutt tabellen kontrollerar trafiken mellan verktyget och VPCn. Varje subnet associerad till rutt tabellen har tillgång till endpointen (Endpoint, 2021). De betyder att rutt tabellen som endpointen är associerad till måste innehålla subneter där Neptune och S3 instansen lanseras.

### 4.3.2 Hur data flyttas till Neptune

För att skicka data till Neptune instansen används instansens ”loader” verktyg. Informationen skickas med en http POST request till Neptune instansens som innehåller information varifrån datat hämtas, vilken format och IAM rollen som är kopplad till instansen. IAM rollen ger Neptune instansen tillåtelse att ladda filen från S3.

```
curl -X POST \
    -H 'Content-Type: application/json' \
    https://your-neptune-endpoint:port/loader -d '
    {
      "source" : "s3://bucket-name/object-key-name",
      "format" : "format",
      "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",
      "region" : "region",
      "failOnError" : "FALSE",
      "parallelism" : "MEDIUM",
      "updateSingleCardinalityProperties" : "FALSE",
      "queueRequest" : "TRUE"
    }'
```

*Figur 17 Exempel av POST request vilket laddar databas filen från S3 till Neptune*

## 4.4 Realtionsdatabas

RDS (Relational Database Service) är ett verktyg för att lansera och hantera relationsdatabaser med olika väl bekanta relationsdatabasmotorer. De olika databasmotorer är till exempel, MySQL, PostgreSQL, MariaDB, Oracle Database och SQL Server. ProdLib sparar informationen av användaraktivitet i en relationsdatabas med MySQL databasmotor som är redan konfigurerad i AWS nätverket. Webb applikationen söker därmed information av både grafdatabasen och MySQL databasen. Informationen som sparas av inlägg från applikationen är lineariskt utan komplexa förbindelser vilket är en optimal lösning för att spara och söka från en relationsdatabas.

## 4.5 Anslutning till instansen

För att ansluta till EC2 instansen kan man använda AWS Elastic verktyget, vilket möjliggör förbindelser med via en SSH-klient. För det behövs ändå tillstånds referenser till självaste nätverket där instansen är lanserad. En lastbalanserare distribuerar automatiskt inkommande trafik över flera instanser, liksom EC2 instanser, container instanser eller IP-addresser. Den övervakar statuset av registrerade instanser och dirigerar trafik enbart till de som fungerar. (balancer, 2021)

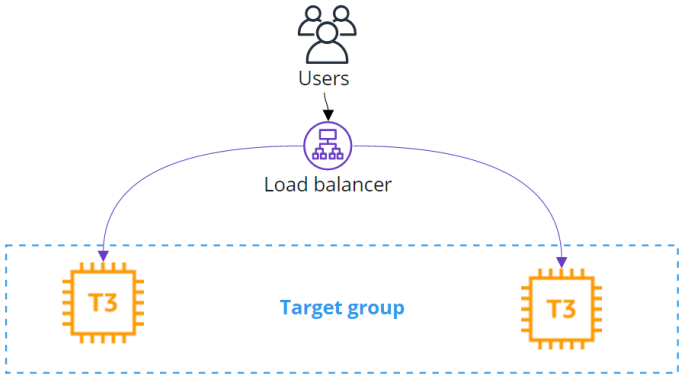
```
const loadBalancer = new aws.lb.LoadBalancer("loadBalancer", {
    internal: false,
    loadBalancerType: "application",
    securityGroups: [aws_security_group.lb_sg.id],
    subnets: aws_subnet["public"].map(__item => __item.id),
    enableDeletionProtection: true,
});

const targetGroup = new aws.lb.TargetGroup("targetGroup-ec2", {
    port: 80,
    protocol: "HTTP",
    vpcId: main.id,
});

const applicationListener = new aws.lb.Listener("application-listener", {
    loadBalancerArn: loadBalancer.arn,
    port: "443",
    protocol: "HTTPS",
    sslPolicy: "ELBSecurityPolicy-2016-08",
    defaultActions: [{
        type: "forward",
        targetGroupArn: targetGroup.arn,
    }],
});
```

*Figur 18 Exempel konfigurering av en lastbalansern, målgruppen samt applikations lysnaren*

Beroende på hur lastbalanseraren är konfigurerar kan den tillåta anslutningar från allmänna källor. Då enligt en definierad lyssnare dirigerar balanseraren inkommande trafik till instansen. Lyssnaren är en process som söker efter anslutningsförfrågningar, med hjälp av konfigurerade protokoll och portar. De definierade reglerna för lyssnaren avgör hur lastbalanseraren dirigerar trafiken till sin registrerade målgrupp (Listener, 2021). Målgruppen berättar åt balanseraren vilken instans trafiken ska dirigeras till. En målgrupp kan innehålla EC2 instanser, fasta IP-adresser eller AWS Lambda (en serverlös, händelsestyrd datortjänst) funktioner (TargetGroup, 2021).




*Figur 19 Exempel hur Load balancer dirigerar anslutningar från användaren*

# 5 RESULTAT

Som slutresultat fick företaget ProdLib en webbapplikation för att visualisera och analysera användaraktivitet. Med hjälp av applikationen kan företaget följa med vilka modeller och dokument laddas ner. ProdLib kan också i framtiden skapa personliga gränssnitt åt tillverkaren för att följa med hur mycket deras produkter används. Utvecklingen av grafdatabasen ledde till vidare utvecklingsprojekt där den nuvarande lagringssystem av tillverkarens bibliotek och hela ProdLib applikations infrastrukturen lagras också i grafdatabaser. Grafdatabaser erbjuder en mycket djupare vision av biblioteksstrukturen och underlättar utvecklingen av biblioteken i framtiden. Applikationens förmåga att visualisera data på så många olika sätt, med värmekartor, tårtdiagrammer, histogrammer och stapeldiagrammer ger möjlighet att marknadsföra företaget till nya kunder och tillverkaren. Företaget kan också följa med hjälp av applikationen mycket noggrannare progress och användarmängder.

*Figur 20 Exempel av slutresultatet*

# 6 SLUTLEDNING

I detta arbete har det beskrivits hur webbapplikationens utvecklingsmiljö sätts upp med användning av Docker. Hur det lönar sig att använda docker-compose verktyget i fallet där man använder sig av flera olika delmoment som sammanfattar applikationen (front-end, back-end, databas bl.a.). Arbetet gick också ytligt igenom semantiken av grafdatabaser och beskrev definitionen av grafteorin. När informationen som sparas innebär komplexa relationer mellan subjekt och objekt lönar det sig att bygga databasen med en ontologi. Det ger möjlighet att specificera relationerna med olika egenskaper mellan noder. Objekt kan definieras att reflektera deras relation till subjektet tillbaka, för exempel om A känner B så betyder det att B känner A.

Slutligen beskrev arbetet hur applikationen lanseras till AWS nätverkstjänsten med användning av Pulumi. Nätverkstjänsten innehåller mycket med olika verktyg som måstes refereras och kopplas med säkerhetsgrupper, IAM roller, rutt tabeller med mera. Nätverket innehåller subnäter med verktyg som dirigerar kopplingar och tillåter anslutningar hit och dit. Hela helheten är en mångsidig och komplex infrastruktur som kräver mycket kunskap och erfarenhet för att sätta upp.

# 8 REFERENSER


balancer, L. (2021). Hämtad från https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/introduction.html

Bulao, J. (2021). *How Much Data Is Created Every Day in 2021?* Hämtad från https://techjury.net/blog/how-much-data-is-created-every-day/#gref

Burke, J. (n.d.). *https://www.techtarget.com/searchnetworking/definition/CIDR*.

Danicic, D. (2020, AUGUST 7). Hämtad från https://phoenixnap.com/blog/what-is-pulumi

Docker. (2021). Hämtad från https://docs.docker.com/get-started/overview/

Endpoint. (2021). Hämtad från https://docs.aws.amazon.com/vpc/latest/privatelink/vpce-gateway.html

Flovikx, V. (2020, Aug 12). Hämtad från https://towardsdatascience.com/what-is-graph-theory-and-why-should-you-care-28d6a715a5c2

Jonathan L. Gross, J. Y. (2020, Aug 12). In J. L. Gross, *Handbook of Graph theory* (pp. 2-7). Hämtad från https://towardsdatascience.com/what-is-graph-theory-and-why-should-you-care-28d6a715a5c2

Listener. (2021). Hämtad från https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-listeners.html

Neptune. (2021). Hämtad från https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

Neptune. (2021). Hämtad från https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

Semantics, R. 1. (2014, February 25). *RDF 1.1 Semantics*. Hämtad från https://www.w3.org/TR/rdf11-mt/

Statista. (2022). *worldwide-data-created*. Hämtad från Statista: https://www.statista.com/statistics/871513/worldwide-data-created/

TargetGroup. (2021). Hämtad från https://aws.amazon.com/blogs/aws/new-application-load-balancer-simplifies-deployment-with-weighted-target-groups/

Techopedia. (2021). *Graph Database, What Does Graph Database Mean?* Hämtad från https://www.techopedia.com/definition/30577/graph-database

Vázquez, F. (2018, Dec 8). Hämtad från https://towardsdatascience.com/ontology-and-data-science-45e916288cc5

Velimirovic, A. (2020, September 17). Hämtad från https://phoenixnap.com/blog/pulumi-vs-terraform

W3C. (2014, February 25). *RDF 1.1 Concepts and Abstract Syntax*. Hämtad från https://www.w3.org/TR/rdf11-concepts/

Web, W. S. (2012, 12 11). *Web Ontology Language (OWL)*. Hämtad från https://www.w3.org/OWL/

# 9 BILAGOR

1. Applikationens docker-compose.yml fil

```yaml
version: '3.3'
services:
    web:
        build:
            context: ./
            target: dev
        volumes:
            # Volymet för källkoden
            - .:/src
        command: npm run dev
        ports:
        # <Utsatta Port> : < Porten för instansen innanför container>
            - "7000:7000"
        depends_on:
            # Instans beroende faktorer
            - mysql
    db:
        image: mysql:5.7
        restart: always
        environment:
            MYSQL_DATABASE: 'db'
            MYSQL_USER: 'user'
            MYSQL_PASSWORD: 'password'
            MYSQL_ROOT_PASSWORD: 'password'
        ports:
            - '3306:3306'
        expose:
            - '3306'
        volumes:
            - my-db:/var/lib/mysql
    blazegraph:
        image: openkbs/blazegraph-docker
        container_name: blazegraph-docker
        ports:
            - 9999:9999
        volumes:
            # Blazegraph data lagring
            - type: volume
              source: blazegraph_data
              target: /var/lib/blazegraph/data
              read_only: false
            - $PWD/workspace:/home/developer/workspace
            - $PWD/data:/home/developer/data
            - $PWD/.java:/home/developer/.java
            - $PWD/.profile:/.profile
        restart: always

    grafana:
        image: grafana/grafana:latest
        container_name: grafana
        volumes:
            - ../../grafana-plugin:/var/lib/grafana/plugins
        environment:
            - GF_INSTALL_PLUGINS=grafana-piechart-panel
        ports:
            - "3000:3000"
volumes:
    my-db:
    blazegraph_data:
```