Projekt B

Erhvervsakademi Dania

Vejleder: Danny Anthonimuthu

Projekt B

IT-Teknolog

Af

Kristoffer B. Steffensen

Sami Dalgaard Tsomidis

Alexander Lee Bradshaw

Sebastian Weisel Kjærby

Anslag: 34.363

03/06/2022

Projekt B

Erhvervsakademi Dania

Vejleder: Danny Anthonimuthu

1 Resumé

Denne rapport undersøger hvordan en mulig løsning til at forbedre Domain Name System information til et system som servicerer kundernes hjemmesider og mail. Den gennemgår hvordan Domain Name System fungerer samt dens styrker og svagheder. Hvordan Domain Name System, bliver opretholdt og sikret ved hjælp af forskellige standarder og sikkerhedsprotokoller. Rapporten kigger også på hvordan en microservice løsning bliver opbygget ved brug af database og software som kalder til relevante Domain Name System server via Application Programming Interfaces.

2 Indholdsfortegnelse

Indhold

3 Figurliste

4 Introduktion

Virksomheden Wulff-production er i gang med at udvikle en større SaaS-løsning. I denne sammenhæng var der et ønske om at få lavet en microservice til håndtering af DNS-registrering via tredjeparts DNS-udbydere. Ligeledes skal microservicen være ansvarlig for at propagate DNS-informationer til andre services i SaaS-løsningen, f.eks. Nginx, Mail server etc.

5 Problemformulering

Hvordan kan man håndtere DNS-propagation fra et centraliseret system til flere andre services, der er afhængige af informationer i DNS-zoner?

- Hvordan kan der implementeres en API baseret interface til intern kommunikation?
- Hvordan kan der implementeres en system-agnostic løsning til DNS-registrering?
- Ligeledes, hvordan kan en skalerbar database-løsning implementeres?

6 Afgrænsninger

Vi som gruppe følte ikke der har været nogle eksterne begrænsninger til vores projekt. Alt information og værktøj til løsningsforslag var frit tilgængeligt.

Begrebet Software as a Service (SaaS) benyttes i rapporten, men vi går ikke i dybden med det, da det er et større teoretisk område og for vores microservice er det ikke nødvendigvis påkrævet viden.

7 Analyse

7.1 DNS

Domain Name System er et service som giver en almindelige bruger et navn i stedet for dens real IP-adresse. Som mange os i dag husker ikke alle folks telefonnummer husker vi hellere ikke hjemmesidens numerisk adresse på internettet. Derfor lever DNS en adressebog til din computer så det er mere brugervenligt og nemmere for mennesker.

Når man tænker telefonbog så tænker man, “simpelt og ligetil”. Du har et navn som skal referere til at tal og DNS er faktisk bare det med mange flere muligheder inden for sikkerhed og filtrering af trafik på internettet og ens egen netværk.

Men sådan fungerede det ikke i internets tidligst dage. Der havde hver en enhed på internettet en host file hvor der stod alle navne som var muligt tilgængelige for brugeren. I princippet var det en god løsning, et decentraliseret system som ikke skabte ekstra trafik på netværket. Men som internettet blev større blev host file også større og dermed et problem med hardware for den enkelte bruger og når alle host files, skulle opdateres hvor den så negeret fordelene ved at have en decentralized domain name system.

“The ARPA Internet illustrates the size-related problems; it is a large system and is likely to grow much larger. The need to have a mapping between host names (e.g., USC-ISIF) and ARPA Internet addresses (e.g., 10.2.0.52) is beginning to stress the existing mechanisms. Currently hosts in the ARPA Internet are registered with the Network Information Center (NIC) and listed in a global table (available as the file <NETINFO>HOSTS.TXT on the SRI-NIC host) [1]. The size of this table, and especially the frequency of updates to the table are near the limit of manageability. What is needed is a distributed database that performs the same function, and hence avoids the problems caused by a centralized database.” (Mockapetris, 1987)

Vi har stadig i dag en host file på netværk enheder men den er som regel tom hvor man som bruger eller administrator kan bruge den til at filtre nogle domains man ikke har lyst til at folk skal besøge. På grund af host files svagheder var man derfor nødt til at strukturere en system som ikke var centralized men distributed så den kunne vokse med internettets vækst og ikke bare i USA men verden over.

“The proposals varied, but a common thread was the idea of a hierarchical name space, with the hierarchy roughly corresponding to organizational structure, and names using "." as the character to mark the boundary between hierarchy levels.” (Mockapetris, 1987)

Domain Name System (DNS). Er et struktureret hierarki der inkorporerer forskellige administrative områder også kaldet ”zones”. I toppen af hierarkiet findes Rootserveren, som er en DNS-Nameserver der operer i Rootzonen.

Et trin under DNS-Rootserver findes TLD-serverne i DNS-hierarkiet. Serverne behandler direkte anmodninger eller forespørgsler ”Queries”, hvor de sendes til den relevante Top Level Domain (TLD)-server. Hvorefter forespørgslen går et trin længer ned i DNS-hierarkiet, hvor individuelle domains og subdomains findes.

7.1.1 DNS Hierarchy

Når der indtastes en hjemmeside i en browser, og det er en ”None-Cached” DNS-forespørgsel, sker der et DNS-Lookup og alle DNS-Lookup starter i Rootzone. Når en forespørgelse rammer Rootzone, vil forespørgslen derefter bevæge sig ned i hierarkiet af DNS-servere og først ramme TLD-serveren. Derefter vil forespørgslen fortsætte ned til serveren for specifikke Domains, Subdomains, indtil det til sidst rammer den authoritative nameserver for det korrekte domæne, som indeholder den IP-adresse på den hjemmeside, som forespørgslen er på. Denne IP-adresse returneres derefter til klienten. Selvom at denne proces virker langsom og med mange forskellige trin, er processen hurtig og mærkes sjældent hos brugeren.

Hvert enkelt af de 13 Rootzone har flere servere, som bruger Anycast-routing til at distribuere anmodninger baseret på belastning og beliggenhed. De 13 Rootservere er administreret og håndteret efter ICANN standarder, samt 2 af de 13 servere bliver også serviceret af ICANN dem selv. Der findes over 600 forskellige DNS-Rootservere fordelt på kloden. “Anycast has in recent years become increasingly popular for adding redundancy to DNS servers to complement the redundancy that the DNS architecture itself already provides. Several root DNS server operators have distributed their servers widely around the Internet, and both resolver and authority servers are commonly distributed within the networks of service providers.”(Abley, 2006).

7.1.2 DNS propagation

Når der opdateres en Nameserver for et domæne, er der en tidsperiode før at opdatering træder i kraft, det kaldes dns propagation. “When you change your domain Name Servers from your domain registrar, it may take up from 24 to 72 hours for the change to take effect and be reflected on every part of the world. This is what we usually call 'DNS Propagation' or 'DNS Propagation Time'.”(DNS Propagation, Global DNS Checker - Whatsmydns?, 2022)

Årsagen til at der er en tidsperiode før det træder i kraft, omhandler den Internet Service Provider (ISP) der bruges. Da, deres egen cache hvor der gemmes DNS information, ikke bliver opdateret 1:1 med opdateringsforespørgslen på nameserver. Det skyldes at der oftest sker mange ændringer på samme tid. Hvorefter der er mange ISP-Noder der skal tjekkes samt din egen lokalcache.

Når der åbnes et domæne i en browser, sendes forespørgsel ikke direkte til hosting serveren, den skal først passere flere ISP-noder. Så computeren vil først starte med at tjekke dens egen lokale DNS-cache, hvorefter anmodningen sendes til den lokale ISP evt. Viborg. Derefter går anmodningen til udbyderen et andet sted, hvorefter den forbinder til internetudbyderen et helt tredje sted.

Hver af ISP-noderne tjekker sin egen cache for at se, om den indeholder DNS-oplysningerne for domænet. Hvis den ikke er der, slår den op og gemmer den for at fremskynde processen næste gang og for at reducere trafikken. Det er grunden til, at de nye informationer omkring nameserveren ikke opdateres med det samme, ISP har forskellige intervaller for opdatering af cache, så nogle af dem vil stadig have den gamle DNS-information i hukommelsen.

7.1.3 AXFR / IXFR

Eller kaldet DNS zone transfer protokol er hvor hele eller dele af Zone filen bliver kopieret over til en anden DNS server. Oftest er det mellem to Name Servers for at holde redundans i DNS service. En som er Master DNS og de andre som er Secondary DNS. De to DNS server skulle gerne have ens information for at vedligeholde integritet.

“The most common scenario is for an AXFR client to open a TCP connection to the AXFR server, send an AXFR query, receive the AXFR response, and then close the connection. But variations of that most simple scenario are legitimate and likely: in particular, sending a query for the zone's SOA resource record first over the same TCP connection, and reusing an existing TCP connection for other queries.” (Lewis, 2010)

7.1.4 DNS Records

En DNS Server fungerer som et database for alle de domains den skal holde styr på. Men det er ikke kun Domain navne en DNS server skal holder styr på. “DNS servers store records. When a DNS query is sent by a device, that query gets a response from those records with the help of DNS servers and resolvers.”(Taylor, 2020) Der findes mange DNS records og de har deres forskellige egenskaber om det er adresse løsning, mail eller generelt information om DNS serveren. I alt findes der otteogfyrre aktive DNS record typer og følgende er nogle de mest brugte.

Den første og mest populær er DNS record A som indeholder IPv4 adresser. Til IPv6 har vi AAAA som fungere på samme måde.

CNAME laver en alias. I stedet for at den peger til en IP-adresse så peger den til et andet domain name. Man kunne selvfølgelig bare lave en A record mere men på den her måde skal du kun ændre record et sted for at opdatere begge domains.

PTR laver det omvendte af A og AAAA. Den forsyner dig med at domænenavn fra en IP-adresse kendt som reverse DNS.

MX er den service som sørger for emails bliver leveret til de mailservers. Som administrator af en mail server, skal man være bevidst om prioriterings hierarki for at sørge for modtagelse af emails.

SOA er en af de vigtigere record typer. State Of Authority som den står for forklarer lidt om hvad den information den lever. Den fortæller dig hvem er autoritets navneserver og serial tal for den DNS Zone. Det er vigtigt information at vide for at andre navneserver i at de skal verificer deres egen DNS Server imod autoritets navneserver.

7.1.5 DNS Sikkerhed

DNS-Sikkerhed er en praksis hvor man beskytter DNS-infrastrukturen mod diverse cyberangreb for at kunne opretholde Quality of Service, QoS. En effektiv DNS-sikkerhedsstrategi bruger en række sikkerhedsforanstaltninger, herunder etablering af redundante DNS-servere, samt anvendelse af sikkerhedsprotokoller som DNSSEC samt strenge krav om DNS-LOGGIN

7.1.5.1 DNSSEC

DNSSEC fungerer som en af de sikkerhedsforanstaltninger der bruges i praksis for at sikre sig mod angreb. DNSSEC anvender en teknologi der signerer data'en digitalt for at sikre gyldigheden, data'en bliver underskrevet med en unik signatur, som ikke kan efterlignes. Denne signerings process sker ved hver DNS-forespørgelse. Det er sådan at man sikre sig den data man arbejder med stadig er gyldig og ikke er blevet manipuleret.

Med mange af internets tidlig protokoller var de ikke tænkt med sikkerhed i minde. DNS er en dem og er derfor åben for angreb på forskellige måder. DNS tunneling tager fordel over at en DNS query bliver sendt over UDP. "DNS uses Port 53 which is nearly always open on systems, firewalls, and clients to transmit DNS queries. Rather than the more familiar Transmission Control Protocol (TCP) these queries use User Datagram Protocol (UDP) because of its low-latency, bandwidth and resource usage compared TCP-equivalent queries. UDP has no error or flow-control capabilities, nor does it have any integrity checking to ensure the data arrived intact." (Hinchliffe, 2019) Fordi at UDP ikke får en response fra serveren at beskeden er modtaget kan der blive sendt mange DNS UDP pakker ud hvis den første efterspørgelse ikke bliver løst. Det giver en mulighed for en fjendtlig at sende brugeren forkert oplysning og sende dem den forkerte vej.

Spoofing eller hijacking er også en mulig sikkerhedsrisiko. Her handler det om at enten ændre i DNS resolvers cache eller udgør sig som en DNS Server for at sende dig til et korrupt hjemmeside hvor de for det meste ser ud til at være legitimt.

Som mange andre enheder på internettet oplever DNS også DDoS angreb. Ofte er det hvor de sender queries om subdomains som ikke eksisterer men til en domain som er legitim.

7.1.5.2 DNS-Firewall

DNS-Firewall, fungere som et værktøj der arbejder med en række forskellige sikkerheds- og internettjenester. DNS-Firewall opererer mellem brugeren og rekursive resolver samt den authoritative nameserver på de tjenester eller hjemmesider der forsøges at opnå. Hastighedsbegrænsning, er en af de måder at firewall'en beskytter på hvis et angreb skulle ske på servere, ved at prøve at overvælde servere. Hvis ovenstående skulle lykkes og serveren bliver overvældet, eller der skulle ske en anden form for nedbrydelse af serveren. Vil DNS-firewall'en være i stand til at opretholde forbindelse til operatører samt hjemmesider og tjenester som stadig vil være fungerende siden at der hentes informationer for den lokale lagringshukommelse.

7.2 Database

En database er en organiseret samling af struktureret information, eller data, hyppigt lagret elektronisk i et computersystem. Typisk har databaser en eller flere forskellige API'er til at oprette, få adgang til, administrere, søge og replikere de data, den indeholder. Generelt er datatyperne i en database begrænset til String data types (CHAR, VARCHAR, TEXT, ENUM), Numeric data types (BIT, BOOL, INT) og Date/Time, men der er også andre slags data som kan bruges, såsom filer på filsystemet eller store hashtabeller i hukommelsen, men datahentning og skrivning ville ikke være så hurtigt og let med den type systemer. I dag bruger vi mest relationelle databasestyringssystemer (RDBMS), hvoraf de mest populære er Oracle og MySQL, et open-source RDBMS.

7.2.1 RDBMS (Relational Database Management System)

RDBMS er en avanceret version af DBMS. Et RDBMS er en samling af programmer og funktioner, der gør it-teams og andre i stand til at oprette, opdatere, administrere og på anden måde interagere med en relationel database. Relationelle databasestyringssystemer gemmer data i tabeller, hvor de fleste kommercielle relationelle databasestyringssystemer bruger SQL til at tilgå databasen. Da SQL blev opfundet *efter* udviklingen af relationsmodellen, er det ikke nødvendigt for brug af RDBMS.

RDBMS er det mest populære databasesystem for organisationer over hele verden, MySQL, Oracle Database, Microsoft SQL Server og IBM DB2 er alle RDBMS.

Definition af gængse database relaterede termer:

- **Table** – En tabel er en matrix med data. En tabel i en database fremstår som et simpelt regneark.
- **Column –** En kolonne (dataelement) indeholder data af en og samme art, for eksempel kolonnens postnummer**.**
- **Row –** En række (= tuple, entry eller record) er en gruppe af relaterede data, for eksempel data for et abonnement.
- **Redundancy –** Lagring af data to gange.
- **Primary Key –** En primær nøgle er unik. En nøgleværdi kan ikke forekomme to gange i én tabel. Med en unik nøgle kan du kun finde én række.
- **Foreign Key –** En fremmednøgle er forbindelsesstiften mellem to tabeller.
- **Compound Key –** En sammensat nøgle er en nøgle, der består af flere kolonner, fordi én kolonne ikke er tilstrækkelig unik.
- **Index -** Et indeks i en database ligner et indeks bagerst i en bog.
- **Referential Integrity -** Referenceintegritet sørger for, at en fremmednøgleværdi altid peger på en eksisterende række.

7.2.2 MySQL

MySQL er en Oracle-backed open source RDBMS baseret på SQL. MySQL kører på de fleste platforme, Windows, Linux, Unix. MySQL er oftest forbundet med web apps og onlinepublicering. MySQL er baseret på klient-server model. MySQL bruger en MySQL server, som håndterer alle databaseinstruktionerne (commands). Serveren er tilgængelig som et separat program for brug i et Klient-Server netværksmiljø og som et library der kan indlejres eller linkes ind i seperate applikationer.

MySQL opererer med flere hjælpeprogrammer, der understøtter administration af MySQL databaser. Instruktioner er sendt til MySQL server via MySQL klient, som er installeret på en computer.

7.2.3 Client-Server

Klient-Server Arkitektur er en computing model, hvor server hosts leverer og administrerer de fleste af de ressourcer og tjenester, der skal udnyttes af klienten. I Klient-Server arkitektur, har man en eller flere klientcomputere forbundet til en central server gennem et netværk eller via internetforbindelse. I dette system deles computerressourcer. Klient-Server arkitektur er også kendt som en networking/computing model eller klient server netværk, fordi all anmodninger og tjenester leveres over et netværk.

Klient-Server arkitektur er en arkitektur af et computernetværk, hvor mange klienter (fjernprocessorer) anmoder om og modtager service fra en centraliseret server (værtscomputer). Klientcomputere forsyner en grænseflade, der giver en computerbruger

mulighed for at anmode om tjenester fra serveren og vise de resultater, serveren returnerer. Servere venter på, at der kommer anmodninger fra klienter og svarer derefter på dem. Ideelt set leverer en server en standardiseret gennemsigtig grænseflade til klienter, så klienter ikke behøver at være opmærksomme på det specifikke system (dvs. hardwaren og softwaren), der leverer tjenesten. Klienter er ofte opsat på arbejdsstationer eller på personlige computere, mens servere er opsat andre steder på netværket, normalt på mere kraftfulde maskiner. Denne computermodel er især effektiv, når klienter og serveren hver har forskellige opgaver, som de rutinemæssigt udfører. I hospitalsdatabehandling kan en klientcomputer for eksempel køre et applikationsprogram til indtastning af patientoplysninger, mens servercomputeren kører et andet program, der administrerer databasen, hvori informationen er permanent lagret. Mange klienter kan få adgang til serverens informationer samtidigt, og samtidig kan en klientcomputer udføre andre opgaver, såsom at sende e-mail. Fordi både klient- og servercomputere betragtes som intelligente enheder, er klient/server-modellen helt anderledes end den gamle "mainframe"-model, hvor en centraliseret mainframe-computer udførte alle opgaverne for sine tilknyttede "dumb" terminaler.

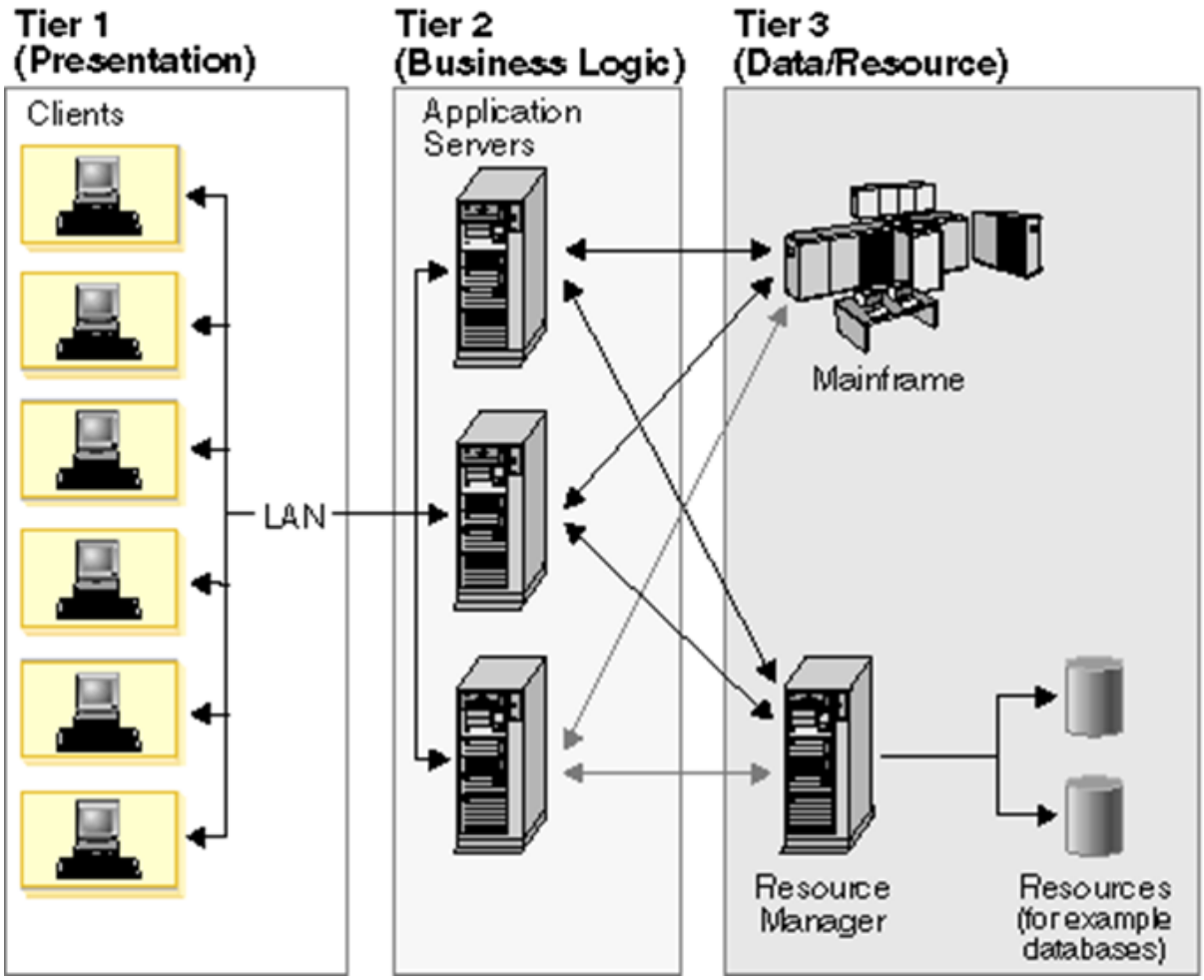
Karakteristika for en klient-server arkitektur:

- Klient- og servermaskiner har brug for forskellige mængde hardware- og softwareressourcer.
- Klient- og servermaskiner kan tilhøre forskellige leverandører.
- Horisontal skalerbarhed (forøgelse af klientmaskinerne) og vertikal skalerbarhed (migrering til en mere kraftfuld server eller til en multiserverløsning).
- En klient- eller serverapplikation interagerer direkte med en transportlagsprotokol for at etablere kommunikation og sende eller modtage information.
- Transportprotokollen bruger derefter lavere lags protokoller til at sende eller modtage individuelle beskeder. En computer har således brug for en komplet stak af protokoller for at køre enten en klient eller en server.
- En enkelt server-klasse computer kan tilbyde flere tjenester på samme tid; Der kræveset separat serverprogram for hver tjeneste.

7.2.3.1 3-tier klient server arkitektur

Den traditionelle klient server arkitektur involverer 2 niveauer, et klientniveau og et serverniveau. Et andet almindeligt design af klient/server systemer bruger 3 niveauer:

- En klient, der interagerer med brugeren.
- En applikationsserver, der indeholder applikationens buisness logic.
- En ressourcemanager, der gemmer data.



Figur 1. Three-tiered client/server architecture (ref. no. 10)

7.2.4 SQLAlchemy

SQLAlchemy er et SQL toolkit og Object-Relational Mapper, ORM, til Python. Formålet med det er at kunne oversætte Python classes til SQL-tabeller, samt at konverterer funktionskald til SQL-statements. Ved at bruge SQLAlchemy er det også let senere at skifte til f.eks. PostgreSQL i stedet for MySQL hvis det ønskes.

7.3 API

Application Programming Interface, API, anvendes til kommunikation mellem en eller flere applikationer, i denne sammenhæng et vilkårligt stykke software med klart defineret funktionalitet, mellem disse applikationer er der en fast defineret grænseflade der specificerer hvordan requests og responses håndteres.

7.3.1 Typer

7.3.1.1 SOAP

Simple Object Access Protocol benytter en standardiseret besked struktur, denne består af følgende:

- Envelope – Denne omslutter beskeden med ”tags” der begynder & konkluderer beskeden, deraf navnet.
- Header – Definerer de specifikke krav for beskeden, bl.a. autorisation. Headeren er ikke påkrævet.
- Body – Indeholder enten request eller response.
- Fault – Viser info omkring fejl der opstår under request eller respons. Fault er ikke påkrævet.

7.3.1.2 RPC

Remote Procedure Call tillader klienter at eksekverer en procedure på serveren og serveren at sende et output tilbage til klienten. Dette fungerer ved at API’en definerer nogle offentligt tilgængelige metoder, disse kan så kaldes med nødvendige argumenter.

7.3.1.3 Websocket

Websocket benytter tovejskommunikation i modsætning til HTTP’s ensrettede kommunikation. Derfor benytter Websocket heller ikke et prædefineret beskedsmønster, klienten eller serveren kan sende beskeder til modparten. Dette betyder altså, at serveren og klienten kan snakke uafhængigt af hinanden.

Modsat HTTP, åbner en Websocket kun en enkelt TCP-forbindelse til hele kommunikationscyklus, dette eliminerer en masse af det overhead der er ved HTTP som er nødt til at åbne en ny TCP-forbindelse for per request/response.

7.3.1.4 REST

REpresentational State Transfer, REST, definerer et set funktioner som klienter kan benytte til at tilgå ressourcer på den givne server. Disse metoder bruges bl.a. til CRUD arbejde:

- GET - Read.
- POST - Create.
- PUT - Update.
- DELETE - Delete.

I bund og grund kan REST ses som et regelsæt, bestående af seks begrænsninger – Den sjette er dog valgfri, der skal overholdes for at kunne skabe en reel RESTful API.

1. Uniform interface – Resurser er identificerbare ud fra en enkelt URI.
2. Client-Server – Serveren stiller blot resurser til rådighed, det er op til klienten at request specifikke services.
3. Stateless - Serveren opbevarer ikke nogen klient specifik information – Dette vil derfor sige at individuelle kald er uafhængige af hinanden. Så, når klienter laver en request skal alle påkrævede informationer indgå i den enkelte request, dette inkluderer også autorisationsnøgle.
4. Cacheable – Hvis en tilpas ny kopi af en request er i en cache vil denne blive benyttet i stedet for at initierer unødvendigt Client-Server trafik.
5. Layered system – Arkitekturen kan være spredt over flere server lag.
6. Code on demand – Valgfri – Dette tillader udvidet funktionalitet ved at klienter kan hente applets eller scripts.

REST er umiddelbart den mest populære tilgang når det kommer til moderne web API'er.

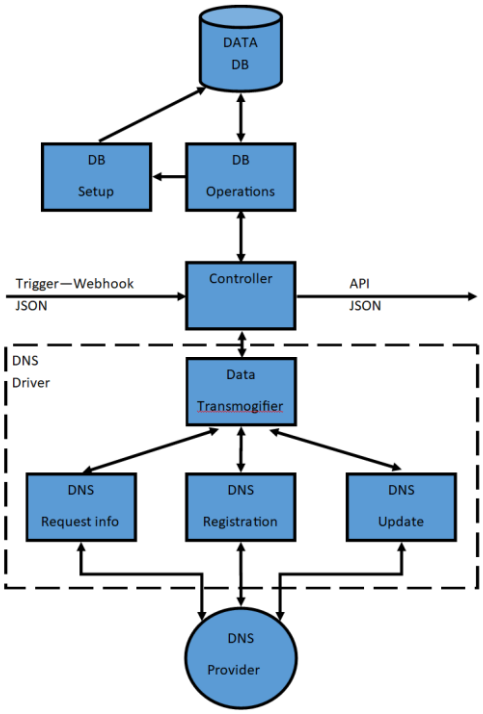
7.3.2 FastAPI

Dette er et Python Framework der gør det forholdsvis let & hurtigt at udvikle high-performance API'er. FastAPI er baseret på OpenAPI specifikationen og kommer med indbygget automatisk dokumentation, i form af en Swagger UI.

For at benytte FastAPI skal man dog bruge et ASGI server program såsom, Uvicorn. Dette er ansvarligt for at håndtere indkommende requests fra klienter.

8 Løsningsforslag

Den foreslåede løsning er i form af en microservice der består af en database, controller og DNS-driver, som kommer til at indgå i en større SaaS løsning Wulff-production arbejder på. Controlleren er ansvarlig for at kommunikerer med andre microservices via dens REST baserede API, samt at delegerer opgaver til hhv. database & DNS-driver alt afhængigt af de indkomne requests. Formålet med DNS-driveren er at kunne abstrahere fra hvilken DNS-udbyder der benyttes, så der, udefra set kun er en generel måde at foretage operationer ved DNS-udbyderen. DNS-driveren sørger så for at de givne informationer bliver parsed korrekt alt efter hvilken udbyder & operation der skal foretages. Database, controller & DNS-driver bør separeres ud i forskellige classes så de enkelte dele kun har ansvar for hver deres type opgaver.



Figur 2. Blokdiagram

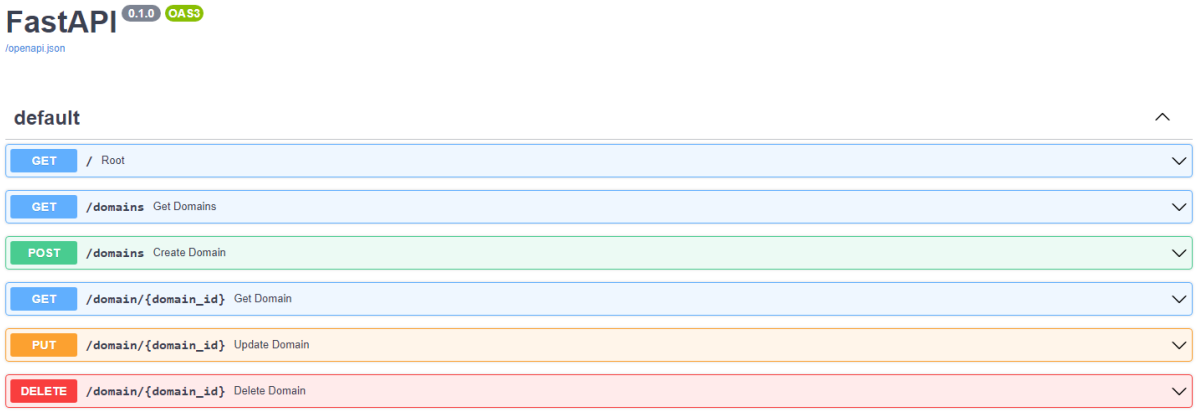
8.1 Controller

Controlleren fungerer som entry point til denne microservice. Andre services kan f.eks. opdatere, hente eller slette DNS-informationer via en API. Fordelen ved at implementerer det som en API er at servicen kan separeres fuldstændigt fra alle andre microservices og fremtidigt kan der nemt udvides, da der blot tilføjes flere endpoints.

8.2 API

Implementeringen af API'en kan gøres på flere måder, hvis den skal være "*True*" RESTful, skal alle resurser have et konkret endpoint, dette vil for eksempel sige at i vores tilfælde vil der skulle være et endpoint for hver type record, for at man kan GET dem individuelt. Eksempelvis:

- GET /domain/{domain_id}/AAAA – For at få alle AAAA records til det specifikke domæne.
- GET /domain/{domain_id}/AAAA/{record_id} – For at få en specifik AAAA record til det specifikke domæne.



Figur 3. FastAPI endpoint eksempler – Swagger doc

En anden tilgang er at lave en API der ikke er RESTful og i stedet benytte sig af query parametre, det kunne f.eks. se således ud:

- https://blah.xyz/domains?domain_id=123&record=AAAA – For at få alle AAAA records til det specifikke domæne.

8.3 Database

Database relaterede operationer bør håndteres fra et centralt sted, så der kun er et entry point til databasen for hele applikationen. Dette vil i praksis kunne opnås ved at implementerer en class som en singleton – Hvorved det kun er muligt at instansiere den en gang.

8.3.1 Datastrukturen

Baseret på hvordan en DNS-zone file er konstrueret og hvad den kan indeholde, kan man komme med et bud på hvordan en potentiel datastruktur kunne implementeres.

```
$ORIGIN example.com.     ; designates the start of this zone file in the namespace
$TTL 3600                ; default expiration time (in seconds) of all RRs without their own TTL value
example.com.  IN  SOA   ns.example.com. username.example.com. ( 2020091025 7200 3600 1209600 3600 )
example.com.  IN  NS    ns                    ; ns.example.com is a nameserver for example.com
example.com.  IN  NS    ns.somewhere.example. ; ns.somewhere.example is a backup nameserver for example.com
example.com.  IN  MX    10 mail.example.com.  ; mail.example.com is the mailserver for example.com
@             IN  MX    20 mail2.example.com. ; equivalent to above line, "@" represents zone origin
@             IN  MX    50 mail3              ; equivalent to above line, but using a relative host name
example.com.  IN  A     192.0.2.1             ; IPv4 address for example.com
              IN  AAAA  2001:db8:10::1        ; IPv6 address for example.com
ns            IN  A     192.0.2.2             ; IPv4 address for ns.example.com
              IN  AAAA  2001:db8:10::2        ; IPv6 address for ns.example.com
www           IN  CNAME example.com.          ; www.example.com is an alias for example.com
wwwtest       IN  CNAME www                   ; wwwtest.example.com is another alias for www.example.com
mail          IN  A     192.0.2.3             ; IPv4 address for mail.example.com
mail2         IN  A     192.0.2.4             ; IPv4 address for mail2.example.com
mail3         IN  A     192.0.2.5             ; IPv4 address for mail3.example.com
```

Figur 4. DNS-zone file eksempel

Fra ovenstående billede kan det ses at der kan for hvert domæne kan være flere A-record, AAAA-records etc. Dette betyder derfor, at der skal benyttes datastruktur der tillader at der kan tilføjes ekstra attributter til hver database entry. Dette kunne gøres ved at tilføje en ny kolonne med en ”*Alter*” statement, problemet med dette er at det er relativt dyrt at alter en tabel, da SQL laver en kopi af den eksisterende tabel, tilføjer den nye kolonne og dernæst omdøbes den nye tabel, så den kan erstatte den gamle og den gamle tabel droppes. Derudover låses der også for manipulering mens kopieringen foretages. Udover førnævnte problematik opstår der også en masse redundante felter for alle de entries der ikke benytter den/de ekstra kolonner der bliver tilføjet.

Derfor er en datastruktur hvor attributterne for en enhed kan defineres i en tabel for sig selv, for så derefter at blive associeret med de enheder hvor enkelte attributter er relevante. Til dette formål findes Entity-Attribute-Value (EAV) modellen.

8.3.2 EAV

Implementeringen af EAV involverer som minimum tre tabeller.

1. Entity – I vores tilfælde kunne det være domæner.
2. Attribute – Dette kunne være record types.
3. Value – Værdier for de forskellige record types.

Denne model tillader nu at man med "*Insert*" statements, kan tilføje en vilkårlig mængde attributter til en specifik entity uden at tilføje en masse redundante felter ved andre.

*Entity & Attribute id bør være foreign keys i værditabellen

Entity

id	domain_name	dns_provider
1	test.blah	r53

Attribute

id	label
1	A
2	AAAA

Value

id	attribute_id	entity_id	value
1	1	1	192.168.1.1
2	1	1	192.168.1.2
3	2	1	2001:0db8:85a3:0000:0000:8a2e:0370:7334

Figur 5. EAV-eksempel (Ref. No. 9)

Ovenstående eksempel kan udvides yderligere, værditabellen kan splittes ud i flere tabeller, med en tabel for hver data type f.eks. VARCHAR, INT. Den tilgang vil gøre data validering ligetil.

Det kan blive en nødvendighed at lave en ekstra tabel der samler alle de relevante informationer i et fladt format, så det gør aflæsning lettere.

8.4 DNS-driver

Principielt kunne de andre services i SaaS-løsningen lave requests direkte til DNS-udbyderen, for diverse information, zone file etc. Problemet ved dette kunne f.eks. være hvis man på et tidspunkt beslutter sig for at skifte DNS-udbyder, nu skal man ud og ændrer i ALLE services for at de henvender sig til den nye udbyder.

Et muligt alternativ kunne være at tvinge alt kommunikation gennem controlleren. Nu er det kun controlleren der har fat i DNS-udbyderen og alle andre services kommunikerer med controlleren. Så, hvis man nu ville skifte DNS-udbyder er det kun i controlleren der skal foretages ændringer.

Men, man kan tage det et skridt længere, alt DNS relateret kunne føres ud i en DNS-driver, denne vil have til formål at tage imod de nødvendige informationer fra controlleren, transmogriffe dem om til et format DNS-udbyderen kan forstå, sende requesten og så ligeledes fører det tilbage til det ønskede format igen når den får en response. Dette niveau af abstraction tillader også at man har flere DNS-udbydere tilgængelige og det eneste der ændrer sig, er hvordan data transmogriffes, samt hvordan requests håndteres.

På længere sigt sikrer driver løsningen at man har mulighed for at skalerer systemet som ens kundebase vokser. Ligeledes, giver det mulighed for at kunder kan vælge mellem flere udbydere, dette kan have relevans baseret på geografisk lokation.

9 Diskussion

Implementeringsforslaget bestående af API baseret controller, MySQL Database & DNS-driver, mangler at et reelt implementeringsforsøg for at teste validiteten af løsningen som helhed. Der kan kun drages konklusioner ud fra de teoretiske aspekter. Udelukkende set fra et teoretisk synspunkt bør løsningsforslaget være opgaven fyldestgørende, men da det ikke kan påvises med en konkret implementering, kan der kun tages udgangspunkt i de konklusioner der kan foretages ud fra de udarbejdede analyser.

10 Konklusion

Udelukkende ud fra de teoretiske forslag og løsninger denne rapport påviser, kan der ikke drages nogen konkret konklusion til hvorvidt problemstillingen er tilfredsstillende løst. Da, der er mangel på en håndgribelig implementering som sammenligningsgrundlag for om kravene i Wulff-productions problemstilling er løst.

11 Perspektivering

Fremtidigt arbejde vil fokusere på at lave en proof-of-concept implementering til at teste de forskellige løsningsforslag der er givet i denne rapport.

12 Referencer

Abley, J., 2006. RFC 4786 - Operation of Anycast Services. [online] Datatracker.ietf.org. Available at: <https://datatracker.ietf.org/doc/html/rfc4786>. 1

Cloudflare, 2022. What is DNS security?. [online] www.cloudflare.com. Available at: <https://www.cloudflare.com/learning/dns/dns-security/> [Accessed 3 June 2022]. 2

Dnspropagation.net. 2022. DNS Propagation, Global DNS Checker - Whatsmydns?. [online] Available at: <https://dnspropagation.net/> [Accessed 2 June 2022]. 3

Hinchliffe, A., 2019. DNS Tunneling: how DNS can be (ab)used by malicious actors. [online] Unit42. Available at: <https://unit42.paloaltonetworks.com/dns-tunneling-how-dns-can-be-abused-by-malicious-actors/#:~:text=DNS%20uses%20Port%2053%20which,usage%20compared%20TCP%2Dequivalent%20queries> [Accessed 3 June 2022]. 4

Lewis, e., 2010. RFC 5936 - DNS Zone Transfer Protocol (AXFR). [online] Datatracker.ietf.org. Available at: <https://datatracker.ietf.org/doc/html/rfc5936#section-4> [Accessed 2 June 2022]. 5

Mockapetris, P., 1987. RFC 882 - Domain names: Concepts and facilities. [online] Datatracker.ietf.org. Available at: <https://datatracker.ietf.org/doc/html/rfc882>. 6

Mockepetris, P., 1987. RFC 1034 - Domain names - concepts and facilities. [online] Datatracker.ietf.org. Available at: <https://datatracker.ietf.org/doc/html/rfc1034>. 7

Taylor, R., 2020. Know the eight most common DNS records – BlueCat Networks. [online] BlueCat Networks. Available at: <https://bluecatnetworks.com/blog/know-the-eight-most-common-dns-records/> [Accessed 3 June 2022]. 8

En.wikipedia.org. 2022. Zone file - Wikipedia. [online] Available at: <https://en.wikipedia.org/wiki/Zone_file> [Accessed 2 June 2022]. 9

Ibm.com. 2022. IBM Docs. [online] Available at: <https://www.ibm.com/docs/en/txseries/9.1?topic=SSAL2T_9.1.0/com.ibm.cics.tx.doc/concepts/c_three_tierd_clnt_sevr_arch.html> [Accessed 3 June 2022]. 10

Oracle.com. 2022. What is a database?. [online] Available at: <https://www.oracle.com/database/what-is-database/> [Accessed 3 June 2022]. 11

SearchDataManagement. 2022. What is a RDBMS (Relational Database Management System)?. [online] Available at: <https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system#:~:text=Some%20examples%20of%20specific%20systems,MySQL%2C%20Microsoft%20SQLServer%20and%20PostgreSQL> [Accessed 3 June 2022]. 12

SearchOracle. 2022. What is MySQL? - Definition from WhatIs.com. [online] Available at: <https://www.techtarget.com/searchoracle/definition/MySQL> [Accessed 3 June 2022]. 13

Techopedia.com. 2022. What is Client/Server Architecture? - Definition from Techopedia. [online] Available at: <https://www.techopedia.com/definition/438/clientserver-architecture> [Accessed 3 June 2022]. 14

Encyclopedia Britannica. 2022. client-server architecture | Definition, Characteristics, & Advantages. [online] Available at: <https://www.britannica.com/technology/client-server-architecture> [Accessed 3 June 2022]. 15

Lia.deis.unibo.it. 2022. [online] Available at: <http://lia.deis.unibo.it/Courses/PMA4DS1112/materiale/9.client-server%20model.pdf> [Accessed 3 June 2022]. 16

13 Appendix

Se, ”*Wulff-productions_problemformulering.pdf*” under ekstramateriale i WiseFlow.

GitHub:

https://gitfront.io/r/BrinkPepega/YSpzSoSu4mDL/DNS-prop/