Hybrid IoT Platform

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:
Anant - Bir Singh
Markus Sundberg

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2017

Sammanfattning

Apparater kopplade till ett nätverk via Internet, så kallat Internet of Things (IoT), blir allt vanligare både privat i hushållet eller på företag. Allt från belysningen till kaffebryggare och tvättmaskiner, till bilarna som står i garaget och låset på dörrarna kan numera utrustas med sensorer och kopplas via Internet till en applikation hos användare. Combitech har utvecklat en Internet of Things (IoT) plattform som kan hantera inkommande data från olika sorters apparater från olika företag för att behandlas och skickas vidare till tredje part. Plattformen är utvecklad för Microsofts molntjänst Azure i ramverken .NET och Service Fabric. Problemet behövde lösas var att alla företag inte är redo att ständigt vara uppkopplade mot en molntjänst medan andra inte kan av säkerhetsskäl.

Uppgiften för examensarbetet var en konceptvalidering där plattformen skulle flyttas i sin helhet så intakt som möjligt till en hybridlösning där mikrotjänster kunde köras lokalt på en Windows Server eller mot molnet. Målet var att samma kodbas kan användas både lokalt eller mot molnet enbart med hjälp av konfiguration. När plattformen flyttades från molnet ersatte examensarbetet tre grundläggande komponenter som användes i Azures molntjänst IoT-hub, Service Bus och objektdatabas.

Examensarbetet identifierade RabbitMQ en kraftfull, välanvänd open-source message broker som valdes att ersätta IoT-hub och Service Bus. MongoDB ersatte objektdatabasen då den är en populär, välanvänd objektdatabas som har en väl fungerande community edition som Microsoft Azure har stöd för.

Examensarbetet utvecklade därefter tre mikrotjänster i .NET och Service Fabric som med hjälp av RabbitMQ och MongoDB kunde ersätta de tre befintliga komponenterna och dess funktionaliteter när plattformen kördes lokalt. Plattformen kördes sedan med framgång som hybridlösning där vissa delar kördes mot de nya komponenterna installerade lokalt samtidigt som andra kördes mot molnet, vilket validerade examensarbetets huvuduppgift.

Examensarbetet undersökte också möjligheten att använda samma kodbas för plattformen på en Linux Server. Det gjordes ett försök att ersätta befintlig .NET-bibliotek som ej är Linux-kompatibla med .NET Core-bibliotek. Det visade sig omöjligt då Service Fabric vid tidpunkten för försöket ej hade fullt .NET Core stöd.

Nyckelord: IoT, Molntjänst, Hybrid, Konceptvalidering, RabbitMQ, MongoDB

Abstract

Devices connected to an Internet-based network, known as the Internet of Things (IoT), are becoming more common both privately in the household or on businesses. Everything from the lighting to coffee makers and washing machines, to the cars in the garage and the door locks can now be equipped with sensors and connected via the Internet to an application by users. Combitech has developed an Internet of Things (IoT) platform that can handle incoming data from different types of devices from different companies for processing and forwarding to third parties. The platform is developed for Microsoft Cloud Service Azure in the .NET and Service Fabric framework. The problem needed to be solved where all companies are not ready to be permanently connected to a cloud service, while others can not for security reasons.

The assignment for the degree project was a conceptual validation where the platform would be moved in its entirety as intact as possible to a hybrid solution where micro services could be run locally on a Windows Server or against the cloud. The goal was that the same code base could be used locally or against the cloud only by means of configuration. When the platform was moved from the cloud, the thesis replaced three basic components used in the Azure cloud service IoT hub, Service Bus and object database.

The thesis work identified RabbitMQ a powerful, well-used open-source message broker that was chosen to replace the IoT hub and Service Bus. MongoDB replaced the object database as it is a popular, well-used object database that has a well-functioning community edition that Microsoft Azure supports.

The thesis then developed three micro services in .NET and Service Fabric that, using RabbitMQ and MongoDB, could replace the three existing components and its functionalities when the platform was run locally. The platform was then successfully run as hybrid solution, where some parts were run against the new components installed locally while others were running towards the cloud, which validated the master's thesis work.

The thesis also examined the possibility of using the same code base for the platform on a Linux server. An attempt was made to replace existing .NET libraries that are not Linux compatible with .NET Core libraries. It proved impossible when Service Fabric at the time of the trial was not fully .NET Core support.

Keywords: IoT, Cloud Service, Hybrid, Proof Of Concept, RabbitMQ, MongoDB

Förord

Vi vill tacka Combitech AB, Jakob Blomberg och Simon Gustafsson för möjligheten att arbeta med examensarbetet, all hjälp och handledning.

Vi vill även tacka Christin Lindholm och Christian Nyberg för all hjälp, kommentarer och kritik som examinator och handledare.

Innehållsförteckning

1 Inledning

Detta kapitel är en introduktion till examensarbetet som utförts hos Combitech AB. En beskrivning på vad examensarbetet ämnar att utföra, syfte, målformulering, problemformulering, avgränsning, motivering av arbetet och vilka resurser som använts.

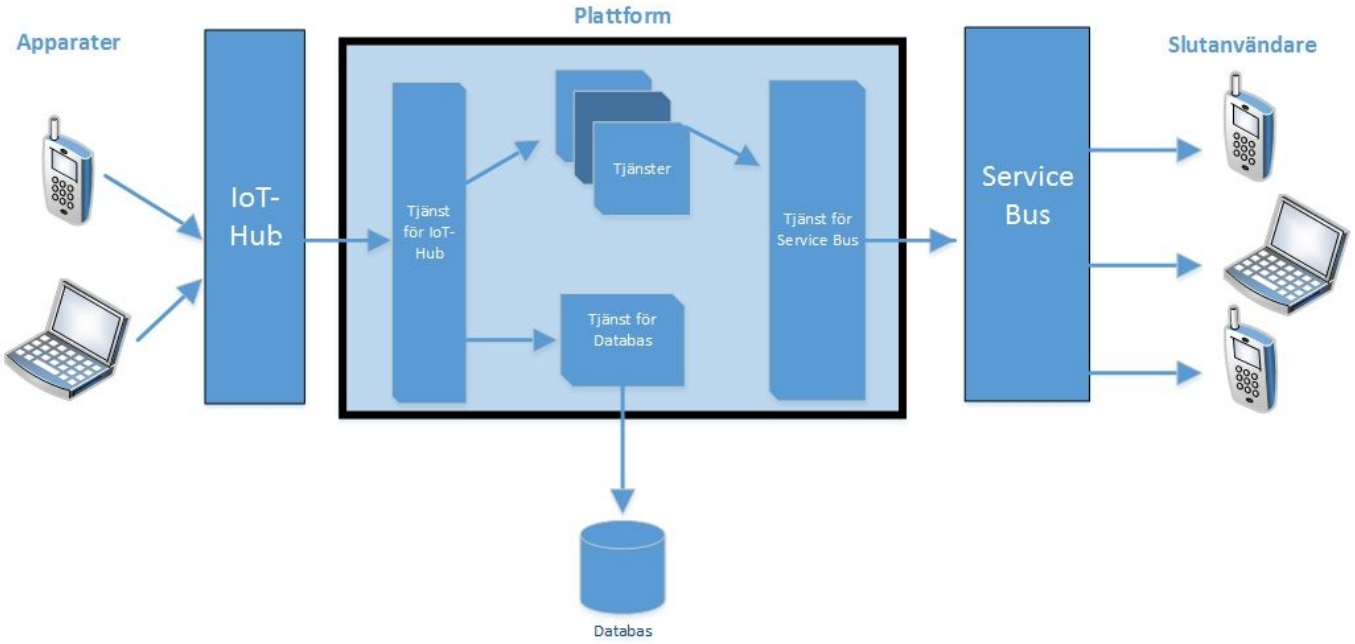
1.1 Bakgrund

Examensarbetet har utförts i samarbete med Combitech AB som är ett teknikkonsultbolag som har en bas i Norden men som även arbetar med internationella kunder[1]. Apparater som i den här rapporten refererar till något som är kopplat till ett nätverk via Internet, så kallat Internet of Things (IoT), blir allt vanligare både privat i hushållet eller på företag. Allt från belysningen till kaffebryggare och tvättmaskiner, till bilarna som står i garaget och låset på dörrarna kan numera utrustas med sensorer och kopplas via Internet till en applikation hos användare. Traditionellt har varje företag sina egna IoT-system med serverlösning, applikation och annat. Combitech har en tjänst som kan hantera apparater från många olika företag via en plattform som finns i Microsofts molntjänst Azure. Datan hämtas in i molnet och behandlas av olika tjänster på plattformen och kan sedan skickas vidare till en enda applikation hos slutanvändare. Plattformen är utvecklad i ramverket Service Fabric.

Problemet är att alla företag inte är redo att alltid vara uppkopplade mot molnet och andra kan inte av säkerhetsskäl. Den befintliga molntjänsten för IoT som Combitech marknadsför är uppbyggd av en plattform med tjänster, en IoT-hub som tar emot meddelande från apparater och dirigerar dem vidare till rätt tjänst i plattformen, en objektdatabas för lagring av rådata från de olika apparater och en Service Bus som distribuerar den av plattformen behandlade datan via ett publish/subscribe-mönster till slutapplikationer.

Examensarbetet innebär att undersöka och testa hur man på bästa sätt kan återanvända plattformen så intakt som möjligt när man går från att köra mot molnet till nya lokala lösningar. Arbetet omfattar en hybridlösning där befintliga tjänster körs mot molnet medan de nyutvecklade tjänsterna körs mot de valda ersättande komponenterna som är installerade lokalt. Målsättningen är att konfiguration ska kunna styra vilka tjänster som körs mot molnet och vilka som körs lokalt.

Inom ramen för examensarbetet ska likvärdiga alternativ för att ersätta komponenterna objektdatabas, IoT-hub och Service Bus identifieras, samt utveckla tillhörande tjänster i IoT-plattformen i ramverket Service Fabric som ska kommunicera med de nya komponenterna, ta emot och skicka vidare data i rätt format till rätt tjänster i plattformen eller till slutanvändare (se Figur 1).



Figur 1: Ersättande Komponenter och Arkitektur

1.2 Syfte

Syftet med examensarbetet är att undersöka, konceptvalidera samt testa hur det går att återanvända den existerande plattform så intakt som möjligt utanför molnet. Utifrån undersökningen identifiera och ersätta de komponenter och mikrotjänster som krävs för att plattformen ska fungera likvärdigt mot moln, som hybrid eller helt lokalt med enbart konfiguration som skiljer dem åt. Detta system kan komma de företag till gagn som inte vill eller kan ha sin IoT-struktur uppkopplad mot molnet och Internet.

1.3 Målformulering

Målet är att flytta plattformen till en Windows server så intakt som möjligt. Identifiera en ersättning till de komponenter som inte kan flyttas från molnet (objektdatabas, IoT-hub, Service Bus).

Arbetet omfattar en hybridlösning där befintliga mikrotjänster körs mot molnet och de identifierade komponenter som ersätter molntjänster är installerade lokalt som de nyutvecklade mikrotjänsterna körs mot.

Mikrotjänsterna i Service Fabric ska kommunicera med de identifierade komponenterna och distribuera data till de andra befintliga tjänsterna i plattformen. Konfigurera och/eller integrera de identifierade komponenterna med befintlig plattform och få det att fungera likvärdigt lokalt på en Windows Server.

Om tid finns testa samma koncept för Linux Server.

1.4 Problemformulering

Följande problem kommer examensarbetet att besvara. Alla frågor angående Linux besvaras i mån av tid! (1aii,1bii,1cii,4 ,5)

1. Vilka likvärdiga lösningar kan ersätta de komponenter i Microsoft Azure som ej kan flyttas till lokal server?

    a. Vilken tjänst kan ersätta objektdatabasen?

        i. För Windows Server

        ii. För Linux Server

    b. Vilken tjänst kan ersätta IoT-huben?

        i. För Windows Server

        ii. För Linux Server

    c. Vilken tjänst kan ersätta Service Bus?

        i. För Windows Server

        ii. För Linux Server

2. Kan man göra en hybridlösning där vissa delar av plattformen körs lokalt och andra delar i molnet?

3. Hur utvecklar man tjänsten i Service Fabric som kopplar ihop de nya komponenterna med befintlig plattform så den fungerar på ett likvärdigt sätt?

    a. När man får likvärdig data från IoT-hub

    b. När man sänder likvärdig data från Service Bus

    c. När man lagrar likvärdig data i objektdatabasen

4. Behövs det ytterligare komponenter för Linux eller kan samma komponenter som är identifierade för Windows användas?

5. Vad krävs för att plattformen ska fungera för användning på Linux?

1.5 Motivering av examensarbetet

Vi valde det här examensarbetet för att Combitech är ett bra företag som vi fick bra kontakt med. Examensarbetet är inom ett område som är aktuellt och de slutsatser vi drar kan vara användbara för alla som inte är redo eller inte kan koppla upp sig mot molnet.

Examensarbetet som sådant berör mycket nytt som inte provats på direkt under utbildningen som, C#, .NET, Windows Server, Service Fabric, Linux, Linux server, .NET Core, message broker med mera. Det ger stor möjlighet att lära sig något nytt samt att få erfarenhet av konfigurering och integrering vilket är en stor del av mycket arbete inom datateknik. Utveckling i C#, .NET, .NET Core och Service Fabric är också en bra merit samt eftertraktat på arbetsmarknaden.

1.6 Avgränsningar

Examensarbetet är avgränsat till att fungera på Windows Server 2016 Standard. Utvecklat med Visual Studio Enterprise 2015 och Service Fabric 2.5.216.

I mån av tid testas konceptet i Linuxmiljö. Mac och IOS stöds inte.

2 Teknisk Bakgrund

I det här kapitlet ges en teknisk bakgrund till de protokoll, koncept och verktyg som examensarbete har använt. De komponenter som examensarbetet har som uppgift att ersätta beskrivs, samt den mjukvara och API använts att ersätta dem.

2.1 Microsoft Azure

Microsoft Azure tidigare känt som Windows Azure är en molntjänst som innehåller en kollektion av alla .NET tjänster som Microsoft erbjuder[2]. Combitech har skapat en IoT-plattform med hjälp av dess verktyg för IoT. Microsoft erbjuder en mängd olika tjänster inom ramen för Azure men examensarbetet fokuserar på de tre som ska ersättas IoT-hub, en objektdatabas och en Service Bus.

2.1.1 IoT-hub

Microsofts IoT-hubs huvudsyfte är att upprätthålla anslutning mellan flera IoT-enheter och molnet[3]. IoT-huben kan ha miljontals enheter anslutna mot molnet, därför är skalbarhet och flödet från enheter upp i molnet en viktig aspekt. IoT-enheterna som är uppkopplade kan kommunicera med IoT-huben genom att använda sig av MQTT, AMQP som beskriv i delkapitel 2.2 och HTTP. När meddelande mottas från olika enheter är IoT-hubens uppgift att dirigera vidare dem till andra tjänster i molnet.

2.1.2 Objektdatabas

Azure använder sig av Cosmos DB tidigare känt som DocumentDB som är en NoSQL databas som sätter fokus på prestanda och skalbarhet[4][5]. Cosmos DB är en objektdatabas vilket innebär att den till skillnad från en vanlig relationsdatabas är anpassad att lagra objekt utvecklade enligt objektorienterade principer. Istället för att lagra rådata mappas objektets beståndsdelar i JSON format som lagras i objektdatabasen[5][6].

2.1.3 Service Bus

När en mikrotjänst eller applikation som körs i Azure behöver interagera med andra tjänster och applikationer utanför molnet[7] används Microsoft Azure Service Bus, som är en asynkron tjänst som sköter kommunikationen utåt. Azure Relay är en del av Service Bus som har en rad olika uppkopplingssätt och protokoll tillgängliga men examensarbetet har bara anledning att beakta delen Service Bus Messaging som Combitech använder. Service Bus Messaging fungerar som en message broker med publish/subscribe-mönster[8] vilket förklaras närmare i delkapitel 2.2.3.

2.2 Protokoll och koncept

IoT-plattformen utvecklad av Combitech och molntjänsten utvecklad av Microsoft har stöd för protokoll AMQP och MQTT som beskrivs i det här delkapitlet. RabbitMQ är implementerat på AMQP 0-9-1 som förklaras mer ingående via RabbitMQ beskrivningen i delkapitel 2.4.

2.2.1 AMQP - Advanced Message Queuing Protocol

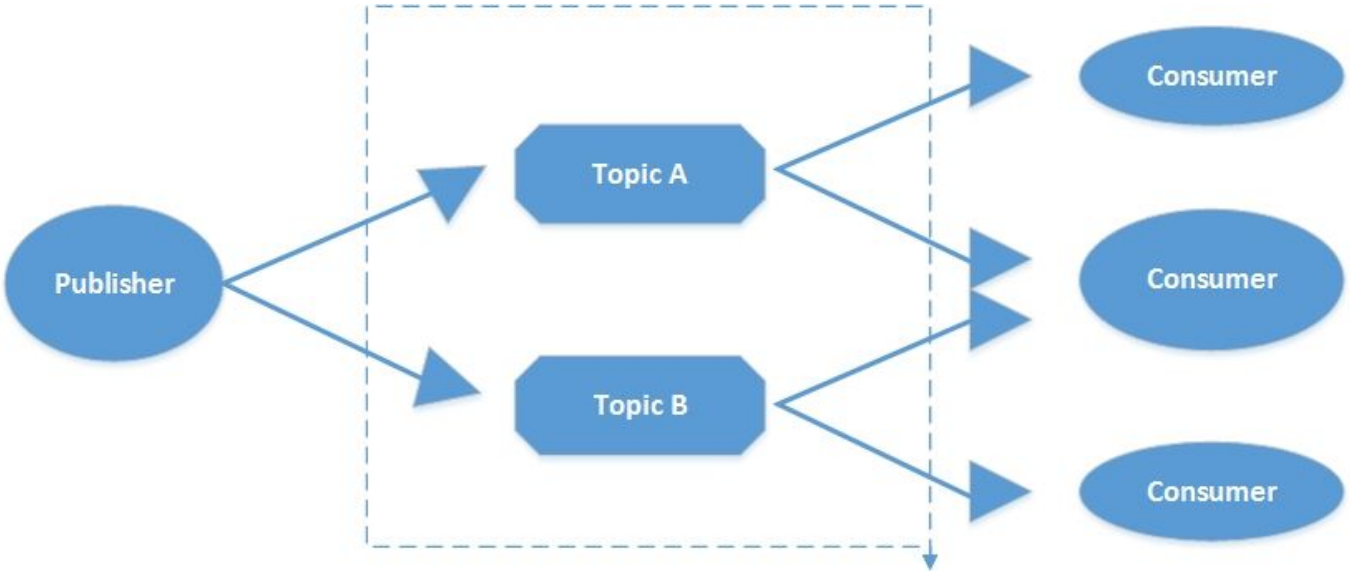
Advanced Message Queuing Protocol förkortat AMQP är ett av de vanligaste protokollen för kommunikation mellan programvara. Syftet med protokollet är att frigöra klienter från varandra och göra dem kompatibla att kommunicera med varandra oberoende av hur de är implementerade.[9]. AMQP och dess koncept förklaras mer ingående under kapitel 2.4 eftersom RabbitMQ implementerar AMQP 0-9-1.

2.2.2 MQTT - Message Queue Telemetry Transport

MQTT är ett protokoll designat att vara enkelt men ändå erbjuda hög pålitlighet. Utvecklat att användas av apparater med begränsad hårdvara, vid låg bandbredd och opålitliga nätverk med hög latens. Idén är att ta så lite resurser som möjligt i anspråk hos apparat och nätverk och det har gjort att MQTT är det ledande Machine to Machine-protokollet som används för mobila apparater och inom IoT[10].

2.2.3 Message Broker

Message broker är mellanprogramvara som används för att hantera meddelanden i kommunikation mellan olika applikationer. Hanteringen av meddelanden innebär att validera och/eller transformera meddelanden och sedan dirigera dem vidare i enlighet med en uppsättning regler[11]. Designmönstret publish/subscribe används för att frigöra olika applikationer från varandra och möjliggör att de behöver veta så lite som möjligt om varandra men ändå kunna utbyta meddelanden. Publisher, ibland kallad producent, publicerar meddelande till en message broker och subscriber, ibland kallad konsument, prenumererar på olika meddelanden. Frikopplingen av klienter med en mellanliggande message broker (se Figur 2) medför sändares/mottagares adress inte behöver vara känd av den andre, klienter behöver inte vara uppkopplade samtidigt och kommunikationen måste inte synkroniseras[12].



Figur 2. Publish/Subscribe-Mönster

2.2.4 Node-RED
Node-RED är en MQTT message broker för IoT med en editerare som kan nås via standard port och en webbläsaren som stödjer Javascript. I editorn kopplas olika noder ihop i olika flöden mellan applikationer som kan skicka och ta emot meddelanden[13].

2.3 MongoDB
I delkapitlet beskrivs MongoDB, dess koncept och verktyg samt formaten JSON och BSON.

2.3.1 Objektdatabas
MongoDB är en objektdatabas med stöd för Window, Mac OS och Linux[71] som använder sig av dokumentbaserad datastruktur. Skillnaden från till exempel Azure CosmosDB är att lagring av data sker med objekt mappade till BSON-dokument[14].

2.3.2 MongoDB Driver API .NET
MongoDB stödjer utveckling av klienter i flera språk varav C#/.NET är ett. MongoDB Driver API är ett objektorienterat ramverk som kan koppla upp/ner mot MongoDB, sätta upp/ta bort/lista databaser och kollektioner, göra queries efter olika filter, sätta in/ta ut/räkna document och en rad andra standardoperationer som förväntas av en objektdatabas[15].

2.3.3 JSON
JavaScript Object Notation eller JSON är ett format som används för att strukturera data. JSON är skapat för det ska vara enkelt för människor att läsa och skriva, samtidigt som det är lätt för maskiner att exekvera och generera[16]. Objektdatabaser brukar vanligtvis lagra objekt med formatet JSON[17]. JSON är ett textformat strukturerad med par bestående av ett namn och ett värde som är arrangerade i en lista (se Figur 3). Paret brukar kallas objekt och listan en array. JSON är språkoberoende[16].

```
"object": {
  "a": "b",
  "c": "d",
  "e": "f"
}
```

Figur 3: Exempel JSON format.

2.3.4 BSON
MongoDB använder sig av BSON-formatet som är JSON i binär form[18]. BSON har tillägg som inte JSON-formatet har som till exempel datum och tidsstämplar när meddelande skickats och mottagits[19][20]. MongoDB var det första större projektet som använder formatet, senare har BSON börjat implementeras så att det kan köras fristående från MongoDB[72].

2.3.5 Mongo Shell
Mongo Shell är ett gränssnitt för JavaScript som kommer förinstallerat med MongoDB[21]. Genom Mongo Shell kan man skriva kommandon som utför alla administrativa operationer samt standardoperationer på databasen[21].

2.3.6 Mongo Client

Mongo Client är ett API utformat som ett GUI som visualiserar en del av de administrativa operationerna och standardoperationer på databaser så de är överskådliga i realtid. Man kan direkt se databaser, kollektioner, dokument, hantera användare och roller. Det finns möjligheter att ställa in autentiseringsmekanismer för användare och konfigurera olika typer av säkerhetsprotokoll som TLS/SSL[20]. En Mongo Shell editerare är implementerad i Mongo Client och kommandon kan skrivas direkt i användargränssnittet, det medför att allt som kan göras i Mongo Shell kan också göras med Mongo Client[20].

2.4 RabbitMQ

I delkapitlet beskrivs RabbitMQ, dess grundläggande koncept. RabbitMQ implementerar AMQP 0-9-1 till fullo och delkapitlet är därmed också en beskrivning av AMQP 0-9-1.

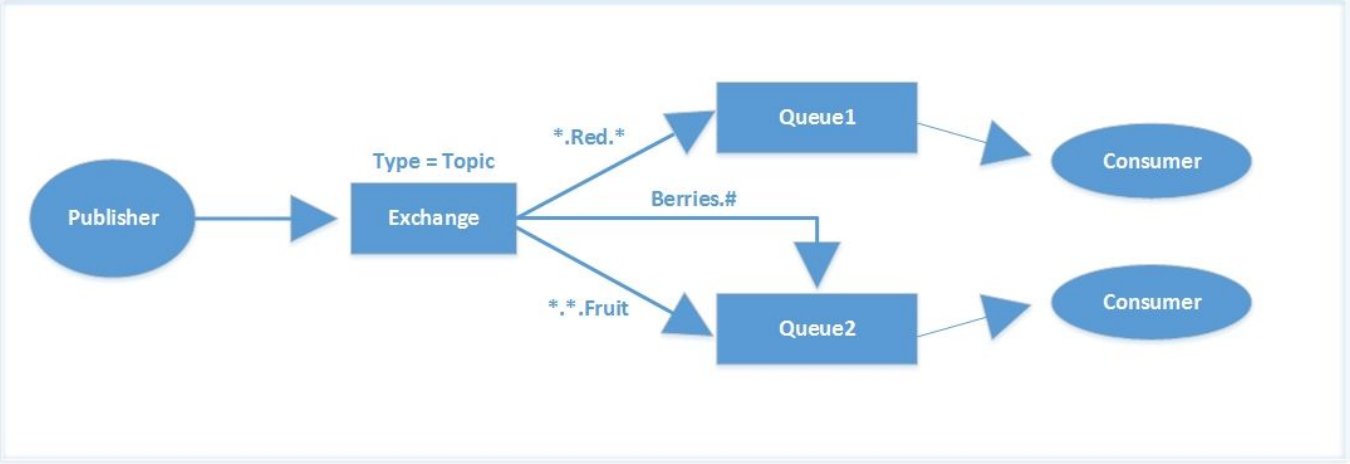
När funktioner beskrivs som inte är del en av standarden för AMQP 0-9-1, utan är tillägg i RabbitMQ nämns det uttryckligen.

2.4.1 Routing Key

Routing key är nyckeln som används i hela kommunikationskedjan för att dirigera meddelanden från publisher till konsument. Nyckeln används olika beroende på typ av dirigering. När header dirigering används kan nyckeln vara flera attribut. Vid direkt dirigering är nyckeln samma som namnet på destinationskön. Topic-nyckel innebär att meddelandet skickas till alla som har bundit sig till den nyckeln. Fanout ignorerar routing key helt och alla meddelanden till alla som är bundna till den nyckeln[22].

Nyckeln sätts av publishern när den publicerar meddelandet till en exchange. En exchange dirigerar vidare meddelande till köer som bundit sig till den exchangen med nyckeln. Konsumenter som konsumerar från en kö på en nyckeln får alla meddelande med den nyckeln[22].

Routing key är 255 bytes och är bestående av ord skilda av punktnotation[25]. Med tecknet ´*´ kan man sedan maska av ett ord av nyckeln och med tecknet ´#´ ett eller flera ord. Publisher kan använda sig av avancerade nycklar och konsumenter via köer binda sig på vissa delar av nyckeln (se Figur 4).



Figur 4: Exempel på filtrering med Routing Key

2.4.2 Bindings
Bindningar är regler som en exchange använder för att dirigera meddelanden. En kö kan binda sig till en exchange med en routing key och bara få meddelanden med den nyckeln. Till skillnad från en kö måste inte en exchange ha minst en bindning för att existera. Meddelande som skickas till en exchange utan bindningar kastats eller skickas tillbaka till avsändaren beroende på egenskaperna som ställts in på från publisher[22].

Som tillägg till det AMQP 0-9-1 protokollet kräver har RabbitMQ lagt till möjligheten att binda exchange till exchange vilket medför möjligheten till en mer komplex nätverkstopologi. [23]

2.4.3 Exchange
När ett meddelande skickas till RabbitMQ hanteras det första av en exchange. Den dirigerar meddelandet vidare till en kö enligt regler beroende av vad det är för typ av exchange[22]. I RabbitMQ finns en uppsättning förimplementerade exchanges av de olika typerna som hanterar meddelanden om ingen exchange instansierats av användaren.
Exchange typer[22][Appendix F]:

- Default: Är en Direct exchange som är bunden till alla existerande köer via namn. Används i enklare applikationer där routing key och könamn är samma.
- Direct: Unicast routing med routing key. Används när ett meddelande har en mottagare.
- Fanout: Skickar till alla köer som är bundna till den, ignorerar routing key. Används vid broadcast routing där alla meddelanden skickas till alla mottagare samtidigt.
- Topic: Multicast routing med routing key. Används när ett meddelande har flera mottagare
- Headers: Multicast routing med headern som routing key. Används när ett meddelande har flera mottagare, dock mer komplex routing än topic routing då header kan bestå av flera attribut.

Det finns även en log[73] exchange för loggning av vad som händer i RabbitMQ-servern och en trace[74] exchange för debugging och spårning av meddelanden som standard i RabbitMQ, som AMQP 0-9-1 protokollet inte kräver.

När användaren skapar en exchange kan en uppsättning egenskaper ställas in efter behov. Följande egenskaper finns[59]:

- Exchange-namn
- Durability: Är flaggan satt sparas exchange vid omstart av RabbitMQ.
- Auto-delete: Är flaggan satt tas exchange bort när sista kön är klar.
- Arguments: Utrymme för specialinställningar/extrafunktioner för den specifika serverimplementationen typ RabbitMQ som implementerar AMQP 0-9-1.

2.4.4 Queue
En kö i RabbitMQ funkar som köer allmänt och agerar som en buffert för meddelande. En kö måste vara bunden till minst en exchange[45] och bestäms den inte av användaren binds kön till en förimplementerade exchange beroende på vilken typ av routing som efterfrågas. När en kö skapas och binds till en exchange kan det göras med en uppsättning egenskaper efter behov. En kö kan ha följande egenskaper finns[45]:

- Namn på kön.
- Durability: Är flaggan satt sparas kön vid omstart av RabbitMQ.
- Exclusive: Är flaggan satt är kön exklusiv för en uppkoppling och kön tas bort vid nedkoppling.
- Auto-delete: Är flaggan satt tas kön bort när sista konsumenten kopplar ner.
- Arguments: Utrymme för specialinställningar/extrafunktioner för den specifika server implementationen typ RabbitMQ som implementerar AMQP 0-9-1.

2.4.5 Connection och Channel
Uppkopplingar till och från RabbitMQ sker med AMQP-kanaler på applikationslagret som använder sig av TCP på transportlagret. En och samma TCP-uppkoppling kan ha flera AMQP-kanaler, men en AMQP-kanal kan inte delas mellan olika trådar i en applikation[22]. Säkra uppkopplingar kan åstadkommas med TLS (SSL)[29].

2.4.6 Publisher
Den som publicerar meddelanden till en exchange i RabbitMQ gör det med metoden *publish()*. När man publicerar meddelanden så kan man sätta vissa egenskaper som RabbitMQ-servern måste ta hänsyn till. Egenskaper hos en publisher[27]:

- Exchange: Namn på exchangen som man publicerar meddelandet till.
- Routing Key: Nyckel som meddelandet ska dirigeras med, användning är beroende av exchange-typ.
- Mandatory: Är flaggan satt skickas meddelande som inte kan dirigeras vidare tillbaka till publisher. Är flaggan inte satt kastas meddelandet utan notifikation.
- Immediate: Är flaggan satt skickas meddelande som inte kan konsumeras direkt av en konsument i kön tillbaka. Är flaggan inte satt köas meddelandet.

2.4.7 Consumer
Den som konsumerar meddelanden från en kö i RabbitMQ gör det med metoden *consume()*. När man konsumerar meddelanden så kan man sätta vissa egenskaper som RabbitMQ-servern kan ta hänsyn till. Egenskaperna är[24]:

- Queue-name: Namn på den kö man ska konsumera ifrån.
- Consumer-tag: Identifierar för konsumenten, måste var unik för en AMQP-kanal och autogenereras om den ej sätts.
- No-local: Är flaggan satt kommer servern inte att skicka meddelande tillbaka till publisher.
- No-ack: Är flaggan satt väntar inte servern på ett ack-meddelande utan tar bort meddelandet ur kön direkt.
- Exclusive: Förfrågan om exklusivt att få använda kön, kan nekas om där redan finns konsumenter.
- No-wait: Om flaggan är satt skickar servern inget svar på om metoden har genomförts eller ej utan genererar istället ett fel.
- Arguments: Utrymme för specialinställningar/extrafunktioner för den specifika server implementationen typ RabbitMQ som implementerar AMQP 0-9-1.

2.4.8 Virtual Host
Virtual hosts i RabbitMQ skapas via terminal[30] eller management plugin[34], de är en logisk gruppering av komponenter ej fysisk. När man kopplar upp sig mot en virtual host görs auktoriseringen där användarnamn och lösenord verifieras[26].

2.4.9 Användare och Accessrättigheter
Användare, lösenord och accessrättigheter tillhör en virtual host. Rättigheter som kan ges till en användare är kombinationer av läs-, skriv- och konfigurationsrättigheter[28]. Rättigheter och användare skapas via terminal[30] eller via management plugin[34].

2.4.10 Plugin
RabbitMQ har en rad plugin med utvecklade funktioner som går utanför AMQP 0-9-1. Det finns plugin som är utvecklade och stödjs av RabbitMQ[31] samt community plugin[32]. De som stödjs av RabbitMQ installeras med RabbitMQ-servern och kan aktiveras via terminalen[30]. Två plugin är en förutsättning för examensarbetet och beskrivs närmare.

2.4.10.1 MQTT Adapter
Pluginet för MQTT verkar på *publish()*-metoden när ett MQTT-meddelande publiceras till RabbitMQ-servern. Här översätter pluginet alla MQTT-attribut till motsvarande AMQP-attribut och gör om meddelandet till AMQP innan det dirigeras vidare. MQTT Adapter kommer förinstallerat och stödjs av RabbitMQ[33].

2.4.10.2 Management Plugin
RabbitMQ Management HTTP API är ett management plugin som möjliggör att sköta hanteringen av RabbitMQ-servern och dess komponenter från ett GUI i webbläsaren via en standard port. Utan detta måste alla komponenter skapas via terminalkommando.

Med Management plugin har man kontroll att ta bort och lägga till komponenter och användare, ställa in olika egenskaperna, ge accessrättigheter och partitionera via olika virtual host. Management plugin kommer förinstallerat och stödjs av RabbitMQ.[34]

2.4.11 RabbitMQ Client API .NET
RabbitMQ stödjer utveckling av klient i en rad språk varav C# och .NET är ett.
RabbitMQ Client API för .NET är ett objektorienterat ramverk för utveckling av RabbitMQ-klienter. Ramverket stödjer implementering av det som behövs för att koppla upp mot och kommunicera med RabbitMQ-servern, publicera/konsumera meddelande med egenskaper och attribut, SSL med mera. Man kan även initiera och radera exchanges och köer[35].

2.4.12 EasyNetQ Management Client API
Ramverket EasyNetQ Management Client API är objektorienterat och verkar mot standard porten på RabbitMQ Management plugin. Med EasyNetQ kan därmed alla management operationer utföras dynamiskt via en klient implementerad i C#/.NET.[36]

2.5 Raspberry Pi

Raspberry Pi är en minidator med många användningsområden och är ett bra verktyg för utvecklare. Den har stöd för operativsystemen Linux och en avskalad version av Windows 10, Windows 10 IoT Core. Raspberry Pi har ett eget operativsystem Raspbian som påminner om Linux i sin miljö.[37]

2.6 Service Fabric

Service Fabric är ett ramverk utvecklat av Microsoft för att hanterar mikrotjänster som i sin tur används till att utveckla molntjänster. Med mikrotjänster menas att man delar upp funktioner i olika mindre tjänster som tillsammans bildar applikationer. En applikation bestående av mikrotjänster laddas upp och sprids över ett kluster bestående av noder. Det innebär att funktioner blir skalbara och det höjer pålitligheten då en funktion kan spridas över flera noder och skulle det hända något med en mikrotjänst så tar en annan nod vid. Service Fabric kan användas i molnet eller lokalt[38].

Service Fabric använder två typer av mikrotjänster. Mikrojänsterna kan vara antingen Stateful eller Stateless. Om mikrotjänsten är i Stateful innebär det att tjänsten använder ett ramverk som håller reda på vilket tillstånd den befinner sig i och skulle något hända med mikrotjänsten kan tillståndet återskapas. Stateless använder inte sig av ett sådant ramverk[54].

3 Metod

I detta kapitel beskrivs metoden som använts för utföra examensarbetet, indelat i en beskrivning av utvecklingsprocess med dokumentation och en beskrivning av arbetsprocessen för varje iteration.

3.1 Utvecklingsprocess

Kanban[41] användes under examensarbetet som utvecklingsprocess då det en av de mest agila metoderna och anpassningsbar efter projektets behov. Kanban använder sig av en Kanban Board som struktureras i ett antal kategorier. Som Kanban Board för att hålla koll på flödet, framsteg och vad som skall göras framåt användes Visual Studios Team Services. Team Services är ett verktyg för en ökad planering och överblick av projektet som Combitech själva använder[39].

Kategorierna i Team Service är New, Open, Active och Closed som har använts under examensarbetet för att beskriva status för en task eller spike. New när de skapas, Open när de tas med på Kanban Board, Active när de är aktiva och Closed när de är klara. Work in Progress (WIP) bestämdes till fyra tasks eller spikes igång samtidigt[41].

Kanban valdes därför att nya uppgifter och krav från handledare kunde komma in löpande då examensarbetet hade många okända faktorer som behövde studeras och testas för att senare ta nytt grepp. Examensarbetarna hade själva utrymme för val och avvägande som kunde leda till snabba förändringar som krävde ett flexibelt arbetssätt.

Arbetet utfördes i tre iterationer med två faser som blev user stories. Varje user story delades in i en rad spikes för studier och tasks för utveckling. Varje fas började med ett möte med handledarna där examensarbetarna först redovisade framsteg och resultat från föregående fas. Handledarna gav sedan information om hur nuvarande systemet fungerar, samt instruktioner och tankar om hur man skulle gå tillväga för att implementera nästa steg på ett likvärdigt sätt.

En annan teknik som användes var parprogrammering, då endast en dator var uppsatt som utvecklingsmiljö satte det ett praktiskt hinder. Men även teoretiskt kändes det lämpligt då helt nya koncept studerades och testades från grunden. Då kunde diskussion i par vara fördelaktigt då det kan vara svårt att ta in allt nytt själv vilket leder till att risken för missuppfattningar och fel ökar.

Arbetet utfördes på Combitechs lokaler i Helsingborg vilket medförde effektiv kommunikation examensarbetarna emellan och med handledare som kunde ge kontinuerlig feedback. Ett vanligt arbetspass varade fem till sex timmar varje vardag och utöver timmarna på Combitech spenderades även tid med rapporten. All relevant information om utvecklingen dokumenterades för Combitech med hjälp av programmet Confluence[40].

3.2 Arbetsprocess

Arbetsprocessen delades in i tre iterationer där varje iteration delades upp i två faser. Varje fas är en user story som sedan delas upp i spikes och tasks som representeras av en punkt i punktlistan. Arbetet avslutas med en extra undersökningsfas för uppgiften som var i mån av tid.

3.2.1 Iteration 1

Den inledande iterationen innehåller uppstarten av projektet som omfattade inlärning av grundkoncept, installation av laborationsmiljö samt en översiktlig undersökning av ersättande komponenter. Laboration och studier av möjliga lösningar och utveckling av mikrotjänsten för IoT-hub.

3.2.1.1 Inlärningsfas 1

Den första inlärningsfasen består av inlärning av grundkoncept, installation av laborationsmiljö, en översiktlig studie och val av ersättande komponenter, samt förberedande studier och laborationer.

- Det första som görs är att installera laborationsmiljön som består av två datorer och en Raspberry Pi sammankopplade med en switch till ett lokalt nätverk. En dator görs till utvecklingsdator med Windows 10, Visual Studio 2015 med .NET och Service Fabric. Den andra datorn till en server med Windows Server 2016 installerat. Raspberry Pi installeras med Raspbian och där finns Node-RED förinstallerat.

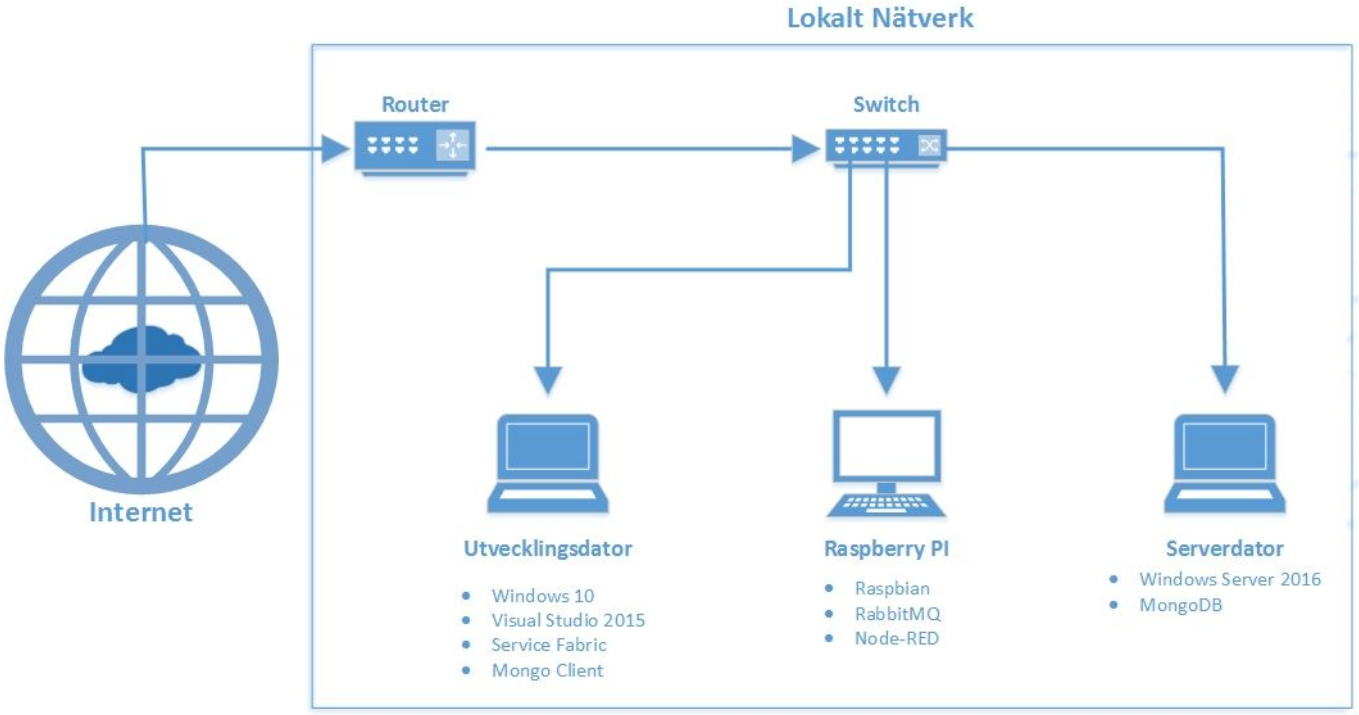
- En översiktlig studie görs för att undersöka vilken programvara som kan vara lämpliga att ersätta komponenterna från Azure. Studien undersöker först om det räcker med MQTT-broker eller generell message broker som RabbitMQ med MQTT-stöd är att föredra. I nästa steg eftersöks en message broker som kan vara mer lämpad än RabbitMQ för uppgiften och AMQP. När RabbitMQ och MongoDB väljs installeras RabbitMQ med plugin på Raspberry Pi och MongoDB på serverdatorn (se Figur 5).

- En studie görs på Combitechs begäran vad det finns för möjligheter till säker kommunikation för framtida användning. Auktorisering, autentisering och kryptering undersöks och dokumenteras.

- Grundkoncept för RabbitMQ och Service Fabric studeras då de är verktygen för den första mikrotjänsten som ska utvecklas.

- En studie görs om mikrotjänsterna som ska utvecklas behöver vara stateful eller stateless och vad det finns för möjligheter att garanterar att RabbitMQ bara levererar ett meddelande en gång.

- Fasen avslutas med studie och laboration av hur service fabric ramverket och RabbitMQ Client API fungerar praktiskt. Resultaten av övriga studier testas i praktiken för användning i nästa fas.



Figur 5: Lokalt nätverk med installerad mjukvara

3.2.1.2 Utvecklingsfas 1

Den första utvecklingsfasen består av utveckling av topologi för IoT-hub, mikrotjänst för IoT-hub samt test av mikrotjänsten i slutgiltig miljö.

- Utvecklingsfasen är en naturlig fortsättning på den avslutande laborationen under föregående fas. Först utvecklas uppsättningen av server-topologin via mikrotjänsten.

- Den delen av mikrotjänsten som hanterar mottagandet av MQTT-meddelande från olika apparater utvecklas i nästa steg.

- Slutligen testas mikrotjänsten att den fungerar med hjälp av Node-RED. Applikationen som tjänsten ingår i laddas upp till Service Fabric klustret på servern och testat igen i slutgiltig miljö. Den nya mikrotjänsten körs lokalt och övriga mikrotjänster i plattformen mot molnet i hybrid-konfiguration.

3.2.2 Iteration 2
I den andra iterationen studeras MongoDB och dess API samt BSON, resultatet av studierna testas vid laboration. Mikrotjänsten för kommunikation med databasen utvecklas i andra fasen av iterationen.

3.2.2.1 Inlärningsfas 2
Den andra inlärningsfasen består av studie av MongoDB, BSON och mikrotjänster i befintlig plattform. Laboration med MongoDB Driver API och Service fabric utförs där resultatet kontrolleras med Mongo Client.

- Inlärningsfasen börjar med att studera MongoDB, dess struktur och grundkoncept.
- BSON formatet och MongoDB Driver API för .NET studeras härefter.
- En liknande mikrotjänst i befintlig plattform studeras samt hur kommunikationen mellan olika mikortjänster inom Service Fabric klustret via proxy fungerar.
- Mongo Client identifieras och installeras som ett visuellt hjälpmedel på utvecklingsdatorn.
- Fasen avslutas med laboration med MongoDB Driver API, MongoDB-servern samt Service Fabric ramverket för att praktiskt testa resultat av studier samt förbereda inför implementeringen.

3.2.2.2 Utvecklingsfas 2
Den andra utvecklingsfasen består av utveckling av databasstrukturen och mikrotjänst för kommunikation med databas. Fasen avslutas med test av mikrotjänsten i slutgiltig miljö.

- Först utvecklas den del av mikrotjänsten som sätter upp objektdatabasen och dess struktur med hjälp av kunskap från föregående studier och laboration.
- Metoden som tar emot meddelanden via proxy från IoT-huben utvecklas sedan och proxy kopplingen i mikrotjänsten för IoT-huben utvecklas så att den skickar allt den får in till mikrotjänsten för objektdatabasen.
- Microsoft har ett kodexempel på hur man utvecklar för kommunikation till MongoDB. Metoder tas därifrån och anpassas till Combitechs struktur.
- Mikrotjänsten för IoT-huben och mikrotjänsten för objektdatabasen testas med hjälp av Node-RED och kontrolleras med Mongo Client API. Applikationerna där mikrotjänsterna ingår laddas upp till Service Fabric klustret och flödet mellan dem testats igen i slutgiltig miljö. De nya mikrotjänsterna körs lokalt och övriga mikrotjänster i plattformen mot molnet i hybrid-konfiguration.

3.2.3 Iteration 3
Den tredje iterationen består av inledande inlärning och studier av RabbitMQ som testas via laboration. Utvecklingen av mikrotjänsten för Service Bus sker i andra fasen.

3.2.3.1 Inlärningsfas 3
Den tredje inlärningsfasen består förberedelse av utveckling av Service Bus, samt studie och laboration av partitionering i RabbitMQ där EasyNetQ Management Client identifieras.

- Combitech har gett uppgiften att kontrollera hur man kan sätta upp nya användare, registrera nya apparater och tenants från en applikation. EasyNetQ Management API identifieras för management operationer via klient.

- En studie inför utvecklingen av Service Bus görs av partitionering i RabbitMQ-server och möjligheten att använda samma instans för både IoT-hub och Service Bus.

- En liknande mikrotjänst i befintlig plattform studeras samt hur kommunikationen mellan olika mikortjänster inom Service Fabric klustret via proxy fungerar.

- Fasen avslutas med laboration av RabbitMQ Client API och EaseNetQ Manager Client API där partitionering och andra management operationer testat.

3.2.3.2 Utvecklingsfas 3
Den tredje utvecklingsfasen består av utveckling av topologi för Service Bus, mikrotjänst för Service Bus och en enkel konsolkonsument. Slutligen testat alla utvecklade mikrotjänster tillsammans i slutgiltig miljö.

- Med resultatet från avslutande laboration i föregående fas utvecklas först den del som sätter upp partitioneringen och topologin för komponenterna i Service Bus på RabbitMQ-servern, så en serverinstans kan användas för båda tjänsterna.

- Vidare utvecklas metoden som tar emot meddelanden från mikrotjänsten för IoT-hub. Mikrotjänsten för IoT-hub kompletteras med proxy koppling så den skickar all data den får in till Service Bus.

- En enkel konsolapplikation utvecklas för att konsumera meddelanden från Service Bus samt testa ytterligare funktioner i EasyNetQ Management Client för framtida användning.

- Mikrotjänsten för Service Bus testas ihop med mikrotjänsten för IoT-hub med hjälp av Node-RED. Avslutningsvis laddas applikationerna där samtliga tre mikrotjänster laddas upp till Service Fabric klustret och samspelet mellan de tre nyutvecklade mikrotjänster testats i slutgiltig miljö. De nya mikrotjänsterna körs lokalt och övriga mikrotjänster i plattformen mot molnet.

3.2.4 Avslutningsfas

I den avslutande fasen studeras möjligheten att göra kodbasen kompatibel med Linux.

- Först studeras .Net Core och Service Fabric.
- Mikrotjänsten för IoT-hub testas i ett .NET Core projekt och beroenden av bibliotek utreds.
- Beroende av bibliotek går ej att rätta till och ersätta med nya. Fortsatta studier visar att problemet är att Service Fabric inte har fullt .NET Core stöd än, uppgiften visar sig omöjlig.

3.3 Källkritik

Majoriteten av källorna som används för informationsinsamling är valda programvarors och protokolls skapare och officiella hemsidor. Här görs en genomgång av samtliga källor.

- Combitech:
  Information om Combitechs verksamhet har hämtats till bakgrundsbeskrivningen. Källan anses pålitlig då vi har varit på företaget och arbetat. Källa som berörs är [1].

- Microsoft:
  Microsoft har utvecklat Service Fabric och Azure plattformen som använts under examensarbetet. Många källor har använts med information om en rad olika funktioner och strukturer inom Service Fabric. Mycket av informationen har testats praktiskt under implementation och Microsoft får anses som ett pålitligt företag med reservation att alla företag säljer produkter.

  Information om Azure har handlat om den grundläggande strukturen som ska ersättas och IoT-hub, objektdatabas och service bus är koncept kända från andra sammanhang och även här får Microsoft anses pålitliga. Källor som berörs är [2]-[5], [7], [8], [38], [44], [50], [54], [63]- [65], [69], [70], [76], [78]-[80], [Figur 2].

- JSON
  Här har övergripande information om JSON-formatet och dess funktioner hämtats. Källan anses vara pålitlig då det är den officiella hemsidan. Källa som berörs är[16]

- BSON
  Här har övergripande information om BSON-formatet och dess funktioner hämtats. Källan anses vara pålitlig då det är den officiella hemsidan. Källor som berörs är [18], [72].

- MongoDB:
  Från MongoDB har information hämtats om dess konceptet, MongoDB Driver API, JSON samt BSON. All använd information om MongoDB Driver API har testats praktiskt och får anses pålitlig. JSON och BSON stämmer med allmän kunskap om vad koncepten innebär och anses som en pålitliga. Källor som berörs är [6], [14], [15], [17], [19], [21], [46], [47], [51], [61], [62], [71], [81].

- DigitalOcean:
  DigitalOcean är ett API för en molntjänst som har en community sida där en kortare beskrivning av AMQP-protokollet har hämtats. Pålitligheten är lägre när man hämtar något från en hemsida som är ett community och på en hemsida från ett företag man inte känner till närmare sen innan. Men beskrivningen innehåller inga konstigheter och är i linje med vad som lästs om AMQP på andra håll till exempel RabbitMQ och källan har mest använts för att det är en bra sammanfattning. Källa som berörs är [9].

- MQTT.org:
  Från MQTT.org har beskrivningen om vad MQTT är för något och en jämförelse mellan olika MQTT-brokers på deras GitHub-sida använts. När det gäller vad MQTT är för något anses källan väldigt pålitlig då det är protokollets officiella hemsida. Källan till jämförelsen är också pålitlig då MQTT.org GitHub-sida länkad från MQTT.org. MQTT-funktionen för RabbitMQ MQTT Adapter har också testats praktisk med gott resultat. Källa som berörs är [10], [43].

- Techopedia:
  Här har en generell beskrivning av vad en message broker är hämtats som stämmer bra överens med vad som finns att läsa de andra länkarna t ex HiveMQ och RabbitMQ. En message broker RabbitMQ och publish/subscribe-mönstret har utförligt testats praktiskt under examensarbetet med gott resultat vilket också verifierar beskrivningen. Källa som berörs är [11].

- HiveMQ:
  HiveMQ är ett företag som står bakom sin egen MQTT-broker. Informationen som hämtas här är om publish/subscribe-mönstret. Informationen anses vara pålitlig då det stämmer överens med andra källor där mönstret beskrivs som RabbitMQ och Techopedia. Källa som berörs är[12].

- Node-RED:
  En generell beskrivning av vad Node-RED är för något har hämtats här och informationen får anses pålitlig då det är den officiella hemsidan samt att Node-RED har använts i praktiken under examensarbetet. Källa som berörs är [13].

- RabbitMQ:
  På RabbitMQ officiella hemsida och GitHub-sida har information om RabbitMQ och AMQP 0-9-1 koncept hämtats. Information om plugin och RabbitMQ Client API har också hämtats här. RabbitMQ är gratis med öppen källkod men det ligger ett företag bakom som heter Pivotal som bland annat också utvecklat JAVA Spring.
  Pivotal får anses pålitliga och dokumentationen för RabbitMQ är utförlig. Det mesta av koncepten som skrivits om har testats i praktiken vid laboration och implementering. Källor som berörs är [22] - [35], [45], [48], [49], [52], [53], [55] - [60], [68], [73] - [75], [77], [82], [Figur 4].

- EasyNetQ:
  EasyNetQ är projekt med öppen källkod som har utvecklat en rad funktioner för RabbitMQ men examensarbetet har bara använt Manager Client API. Det har testats utförligt under laboration och implementering vilket har styrkt att det håller vad det lovar och kan göra allt som RabbitMQ Manager plugin också kan. Källan anses därmed pålitlig. Källor som berörs är [36], [67].

- Raspberry PI:
  Från Raspberry PI hämtades grundläggande information om vad Raspberry PI är.
  Då det är den officiella hemsidan anses källan pålitlig. Källa som berörs är [37].

- Visual studio Team Service:
  En enklare beskrivning av vad Team Service har hämtas härifrån som får anses pålitlig då examensarbetet har övervakat sin utvecklingsprocess med verktyget och att Microsoft som står bakom anses vara ett pålitligt företag. Källa som berörs är [39].

- Atlassian:
  En kortfattad beskrivning på vad Confluence är och har för funktion
  hämtades här samt allmän information om Kanban. Då det är den officiella hemsida för Confluence och examensarbetet har dokumenterat med verktyget får källan anses som pålitlig. Källa som berörs är [40], [41].

- Mosquitto:
  En beskrivning av vad Mosquitto är har hämtats på den officiella hemsidan och källan anses därmed som pålitlig. Källa som berörs är [42].

- DB-Engines:
  DB-Engines är en hemsida som är publicerad av ett österrikiskt företag
  vid namnet Solid-IT. Hemsidan tillhandahåller information över majoriteten av existerande databaser samt rankar databaserna efter popularitet månadsvis. Källan har använts att verifiera att MongoDB är den mest använda objektdatabasen något som även Microsoft skriver. Källan anses därmed pålitlig. Källa som berörs är [66].

- Mongo Client:
  All information som hämtats från Mongo Client har verifierats via användning av Mongo Client GUI API. Mongo Client har utvecklats som ett projekt på Massachusetts Institute of Technology. Källan anses därmed pålitlig. Källa som berörs är [20].

- Lostechies:
  Källan beskriver sig själv som ett publikt forum där tekniska ideér och tankar diskuteras. Här finns flera olika skribenter som skriver om flera områden inom Informationsteknik. Skribenten Derek Greer är konsult som arbetar inom IT har gjort källorna som inspirerat figurerna. Pålitligheten för en okänd skribent på ett publikt forum är inte särskilt hög om man inte vet något om skribenten eller forumet. Källorna anses ändå vara pålitliga då RabbitMQ källor[22][25] bekräftar att de är korrekta och källorna har enbart använts för de ger en bra överblick.
  Källor som berörs är [Figur 7] - [Figur 10].

4 Analys och Resultat

Det här kapitlet beskriver övervägningar, val och resultat som kommit av utvecklingen av de tre mikrotjänsterna, studierna och användningen av RabbitMQ, MongoDB och dess API. Beskrivningen följer arbetsprocessen i metoden där val, övervägningar och resultat redovisas kronologiskt.

Kapitlet avslutas med en beskrivning av strukturen för Service Fabric och Combitechs plattform som använts under examensarbetet men ej varit förmål för övervägningar eller val.

4.1 Iteration 1

Här beskrivs analys och resultat av förberedelse och implementation av mikrotjänsten för IoT-hub.

4.1.1 Inlärningsfas 1

Den första inlärningsfasen består av studier och val av programvara samt laboration för att verifiera resultat och förbereda inför nästa fas.

4.1.1.1 Val av programvara & verktyg

Många saker var redan givna och det mesta i den inledande installationsfasen var inte föremål för val eller överväganden. Den existerande plattformen är utvecklad med Service Fabric och C#/.NET som också skulle användas till de nya mikrotjänsterna. Det lokala nätverket med två datorer och en Raspberry Pi var vad Combitech tillhandahöll och några andra uppsättningar var aldrig aktuella.

Det som har utvärderats och valts är de programvaror, med API för utveckling, plugins och verktyg som ska ersätta befintliga komponenter i Azure. Kravet var att göra en översiktlig studie då Combitech ville att den mesta tiden skulle ägnas åt huvuduppgiften själva konceptvalideringen.

Combitech kände redan till RabbitMQ och därför kom studien att handla om att försöka hitta ett bättre alternativ. Först undersöktes om det kunde vara tillräcklig att använda en MQTT-broker som IoT-hub. En välanvänd MQTT-broker med öppen källkod är Mosquitto[42] som Combitech redan hade erfarenhet av. När Mosquitto jämförs med RabbitMQ och dess MQTT-plugin fanns att Mosquitto har fler funktioner för MQTT men RabbitMQ med plugin är likvärdig på alla generella förväntade funktioner[43]. Med det som stöd bestäms i samråd med Combitech att det är bättre att använda en mer generell message broker som RabbitMQ. Möjligheten till fler protokoll, fler routing möjligheter och funktioner är värt mer än fler MQTT-funktionaliteter.

Vidare eftersöks en generell message broker med öppen källkod utifrån ett AMQP-perspektiv. AMQP är viktigt eftersom det är ett standardiserat protokoll och det som används i Azure förutom MQTT och HTTP[65].

Det finns många generella message brokers men ingen kan hittas vid en översiktlig studie som är AMQP-centrerad, open-source och mer mogen sin uppgift än RabbitMQ. RabbitMQ har utvecklats i 10 år, är väl utbredd och klarar av den mesta routingen med ett brett utbud av policies och plugins[75].

Att välja en objektdatabas var enkelt då MongoDB är en av de mest använda och dess gratis community edition täcker behoven för vad som krävs av vår mikrotjänst[66]. Endast enkla databas-operationer används och inom ramen för examensarbete kommer enbart insättning att ske. Den största faktorn att MongoDB var det enda alternativet var att Microsoft Azure redan har stöd för MongoDB utöver sin egen objektdatabas[44]. Man kan därmed utveckla en tjänst som lätt kan användas både mot MongoDB i molnet eller lokalt.

Combitech har behov av att köra systemet med säkerhet oavsett om det är i molnet, lokalt eller hybrid. En förutsättning för de valda programmen är stöd för säker kommunikation och en uppgiften var att kontrollera att tillräckligt stöd finns och dokumentera det. Examensarbetet validerar sitt koncept på lokalt nätverk utan kryptering men dokumentationen är viktig för fortsatt utveckling.

MongoDB och RabbitMQ studeras och båda har inbyggt stöd för kryptering med TLS (SSL) som kan aktiveras[29][51]. Båda har auktorisation via lösenord och användarnamn, vilket examensarbetet använder de i de olika implementationerna[28][46]. MongoDB och RabbitMQ har också stöd för autentisering med LDAP och SASL med hjälp av konfigurering och plugin[47][48][49].

4.1.1.2 Studie och Laboration

När valet av programvara är gjort görs en förberedelse inför utvecklingen av mikrotjänsten för IoT-hub. RabbitMQ har ett officiellt API för utveckling av klient till RabbitMQ-servern som identifieras. RabbitMQ Client API blir ett naturligt val att utveckla de delar i mikrotjänsten som sätter upp topologin i och kommunicerar med RabbitMQ-servern. RabbitMQ Client API har också stöd för .NET Core vilket är en förutsättning för kompatibilitet med Linux[68].

I Combitechs plattform används stateful services för liknande tjänster vilka garanterar att meddelanden inte tappas bort och en indexeringsmarkör hanterar att samma meddelanden inte skickas igen om något oförväntat händer. Combitech ger i uppgift att kontrollera om det räcker med en stateless service som har en enklare struktur om RabbitMQ kan uppnå tillräckligt hög pålitlighet.

En studie görs och det visar sig att det finns åtskillig funktionalitet i RabbitMQ för att åstadkomma hög pålitlighet. Man kan göra köer och exchanges durable[59][45] vilket innebär att om fel skulle uppstå på servern kan tidigare tillstånd återskapas, vilket medför att inga meddelanden försvinner. Att ett meddelande bara ska levereras en gång kan lösas med ACK-meddelanden, vilket fungerar så att när konsumenten har tagit emot ett meddelande skickas ett ACK-meddelande och först när servern får ACK-meddelandet tar den bort det konsumerade meddelandet ur kön[53]. Det finns risk att något kraschar i den lilla tidsluckan mellan att meddelandet tagits emot och ACK-meddelandet har hunnit skickas men det ansågs i samråd med Combitech att vara tillräckligt pålitligt då det är låg sannolikhet att det inträffar.

När en apparat skickar ett MQTT-meddelande i nuvarande system skickas tidsstämpeln för avsändandet med i meddelandets payload då MQTT inte har stöd för tidsstämpel som egenskap. Alla apparater har dock inte möjligheten att skicka med tidsstämpel ens i payload på grund av dess begränsningar och därför sätter Azure IoT-hub en tidsstämpel[50] när meddelandet köas i molnet och på så vis får man ändå en viss vetskap om hur aktuell datan är.

Combitech gav uppgiften att undersöka hur det skulle fungera att göra i RabbitMQ. Det visade sig att RabbitMQ inte har stöd för att sätta en tidsstämpel när ett meddelande kommer till servern, men att servern har en intern klocka och det finns ett community plugin som har löst problemet[55]. Vid laboration testades funktionaliteten och det konstaterades att det av någon anledning inte fungerade. Fortsatta undersökningar av källkod konstaterade att när pluginet RabbitMQ MQTT Adapter gör om ett MQTT-meddelande till AMQP skapar den samma egenskaper som MQTT-meddelandet har stöd för[57]. Där finns ingen tidsstämpel i MQTT-meddelandet och därför skapas ingen egenskap för tidsstämpel i AMQP-formatet heller, vilket leder till att pluginet för tidsstämpel inte har någon egenskap att sätta till tiden för serverns interna klocka[56].

Avslutningsvis laboreras det med Service Fabric och RabbitMQ Client API. Test hur man startar ett Service Fabric kluster på serverdatorn genomförs och hur man publicerar en enklare mikrotjänst till klustret. Exchanges och köer testas från mikrotjänsten och MQTT-pluginet testas att det fungerar som förväntat med hjälp av Node-RED.

Här upptäcks att det skapas temporära köer när MQTT-meddelanden skickas till RabbitMQ-servern och att de är auto-delete som förinställning[77]. Det passar väldigt bra då det är att föredra att köerna tas bort när en apparat kopplar ned. Om många olika apparater kopplar upp och ned sig mot RabbitMQ-servern fylls den snabbt upp med inaktiva köer. Vid laboration med MQTT-pluginet upptäcks också att eftersom MQTT bara stödjer topic dirigeras meddelanden via förinstallerade exchanges då meddelandet inte kan ha någon annan typ av routing.

4.1.2 Implementation av IoT-hub

När den avslutande laborationen hade testat verktyg, verifierat studier och svarat på frågor påbörjades implementationen av mikrotjänsten för IoT-hub under den första utvecklingsfasen. Service Fabric och RabbitMQ Client API användes för utvecklingen.

RabbitMQ Client API är implementerat så att exchanges och köer är idempotenta vilket innebär att man kan göra samma operation många gånger utan att förändra usprungsresultatet. Därför görs valet att sätta upp topologin för IoT-huben på RabbitMQ-servern dynamiskt när tjänsten startas då topologin inte kommer ändras om tjänsten startas om[58].

Tjänsten kopplar upp sig mot RabbitMQ-servern skapar en durable[59] *MasterExchange* som binds till alla förimplementerade exchanges[22]. Detta görs för att fånga upp alla sorters meddelanden som kommer in till IoT-huben oavsett vad det är för typ av routing som efterfrågas, IoT-hubens uppgift är att hämta in all information och skicka vidare den i plattformen.

Som föregående laboration visade för MQTT så skickas meddelande med topic och skulle inte *MasterExchange* vara bunden till förimplementerade exchange *amq.topic* hade meddelanden inte kunnat skickas vidare via *MasterExchange.* Man skulle kunna använda den förimplementerande exchange *amq.topic*[22] som exchange för IoT-hub men då binder man sig till ett protokoll och då AMQP har andra routing möjligheter kan meddelande missas. Valet blir att göra en ny *MasterExchange* som binds till alla förimplementerad exchanges i RabbitMQ se figur 6 [22].

Enligt AMQP-konceptet måste en konsument binda sig till en kö och en kö måste vara bunden till en exchange för att existera[45]. Eftersom IoT-huben ska samla in all data som kommer till plattformen är mikrotjänsten för IoT-hub den enda konsumenten av data.
Med det som grund skapas en durable[59] kö *MasterQueue* som binds till *MasterExchange (s*e Figur 6) med routing key ´#´ som betyder att kön prenumererar på alla meddelande oavsett topic[25]. Att det bara finns en kö och en konsument ligger också till grund att *MasterExchange* implementeras av typ fanout så den skickar vidare allt till de köer som är bundna till exchangen utan att verifiera routing key, då det ej behövs[22].

Vidare skapas en konsument, RabbitMQ Client API har stöd för några olika konsumenter och valet faller på den event-baserade[52]. Det passar bra för IoT-huben ska lyssna på allt men bara vara aktiv när något meddelande finns att hämta samt att konsumentens metod *on-deliver()* kan köras asynkront vilket överensstämmer med Service Fabric strukturen och metoden som inhämtningen av data sker i.

När den event-triggade metoden hämtar in data skickas först ett ACK-meddelande tillbaka och RabbitMQ-servern som får ACK-meddelande tar bort meddelandet från kön, alltså kan mottagna meddelanden inte skickas igen som tidigare studie och laboration visat[53].
I nästa steg görs en rad verifieringar som är i linje med Combitechs befintliga struktur som beskrivs i delkapitel 4.6.

Meddelandet valideras och tidsstämpel läggs till, då servertid inte kunde sättas i RabbitMQ-servern vilket föregående studier och laboration visade sätts en tid när meddelandet kommer till mikrotjänsten istället. Ett kvarstående problem blir då att man inte kan veta om meddelandet har köats länge på servern och blivit inaktuellt, men med Combitechs samråd bestäms att det är tillräckligt för examensarbetet som främst inriktar sig på konceptvalideringen.

Figur 6: Topologi för IoT-hub.

4.2 Iteration 2

Här beskrivs analys och resultat av förberedelse och implementation av mikrotjänsten för lagring i objektdatabasen.

4.2.1 Inlärningsfas 2

Under inlärningsfas 2 studeras MongoDB och valet att använda dess officiella MongoDB Driver API för utveckling är givet. API är utvecklat av MongoDB och är det API som Microsoft kräver för utveckling mot både MongoDB i Azure och lokalt[15]. MongoDB Driver API har också stöd för .NET Core vilket är en förutsättning för kompatibilitet med Linux[61]. BSON studeras och det identifieras hur man ska omvandla JSON-meddelande till BSON-dokument[62].

Slutligen utförs en laboration där funktioner i MongoDB Driver API testas med en enklare mikrotjänst i Service Fabric och kommunikation upprättas med MongoDB-servern. Inga avancerade databasoperationer ska användas och möjligheterna är inte så många att utforska. Vid laboration upptäcks dock att det är väldigt svårt att överblicka resultat via terminalen då det måste göras i flera steg när man ska kontrollerar att data lagras korrekt. MongoDB har Mongo Shell där man kan scripta på ett enklare sätt men det innebär att lära sig många av Mongos script-kommando. Istället identifieras ett GUI API som heter Mongo Client[20] som testades och fungerade väl.

4.2.2 Implementation av Databas

Efter laboration med Mongo Driver API grundfunktioner inleds utvecklingen av mikrotjänsten för lagring i databas. Först implementeras den asynkrona post-metoden som IoT-huben via proxy skickar meddelande till. Tjänsten kopplar upp mot MongoDB-servern och hämtar rätt databas och rätt kollektion i databasen[15]. Kollektionen har samma namn som den tenant i Combitechs system som meddelandet kommer ifrån. Meddelandet görs sedan om från en byte-array till JSON och från JSON till BSON-dokument med hjälp av klasser i MongoDB Driver API[62].

Microsoft tillhandahåller en klass med metoder som exempel för utveckling till MongoDB och Azure[76]. I samråd med Combitech bestäms att klassen ska omarbetas eftersom den är väl utformad och använder sig av SSL-säkerhet. Metoderna i exemplet modifieras till Combitechs struktur och dess meddelandeklass MessageRoot.

4.3 Iteration 3

Här beskrivs analys och resultat av förberedelse och implementation av mikrotjänsten för Service Bus.

4.3.1 Inlärningsfas 3

Lägga till nya tenants, apparater och användare görs i det befintliga systemet från en applikation. Därför gav Combitech uppgiften att kontrollera om detta var möjligt även med RabbitMQ. Efter att ha studerat och laborerat med RabbitMQ Client API[35] och Management pluginet[34] var det känt att bara köer och exchanges kunde skapas och tas bort via klienten, alla andra management operationer måste göras via Management plugin och dess HTTP GUI via webbläsare. Vid närmare undersökning identifieras EasyNetQ Management Client API[36] som verkar mot standard porten för Management plugin. EasyNetQ använder alltså sig av befintligt Management plugin för att utföra management operationer från klient. Eftersom de olika management operationerna ska ske från en utomstående applikation och att EasyNetQ verkar mot Management plugin som är kompatibelt med Linux spelar det ingen roll att EasyNetQ Management Client inte är fullt kompatibel med .NET Core då koden inte ska installera i Linux miljö[67].

Eftersom RabbitMQ används som både IoT-hub och Service Bus gavs uppgiften av Combitech att undersöka hur detta görs på bästa sätt. RabbitMQ studerades för att om möjligt hitta ett sätt att partitionera en serverinstans och hur man minimerar risken att meddelande skickas till fel del av servern där man i värsta fall får rundgång mellan Service Bus och IoT-hub. Det visar sig att auktorisering sker per virtual host[26] och det blir då lämpligt att särskilja IoT-hub och Service Bus åt på virtual host nivå.

I Service Busen ska en konsument kunna prenumerera på data från en viss tenant och frågan är hur man minskar risken att meddelanden kommer till någon obehörig. I RabbitMQ kan kommunikation mellan olika virtual host inte ske, så att använda partitionering i en partition går inte. Till exempel att ha en Service Bus virtual host som skickar vidare till olika tenant virtual hosts är uteslutet. Lösningen blir att skapa olika Service Bus virtual host för varje tenant som IoT-huben skickar direkt till beroende på destination för meddelandet.

Under studien identifieras dock lösningar på virtual host kommunikation som implementerats via två olika plugin som ej används eller utforskas närmare av examensarbete. Plugin Shovel[60] och Federation[61] används till klustring och kommunikation mellan olika virtual host på olika servrar eller inom samma instans. I den avslutade laborationen testats studierna för partitionering i praktiken och EasyNetQ Management Client API testats med Service Fabrice och RabbitMQ Client API.

4.3.2 Implementation av Service Bus

Resultatet av föregående studier och laboration leder till att EasyNetQ Manager Client API också används till att utveckla mikrotjänsten för Service Bus förutom RabbitMQ Client API och Service Fabric.

Implementationen börjar med en uppsättning av partitioneringen och dess topologi med hjälp av EasyNetQ Manager Client API. En manager-klient skapas som hämtar administratören, en användare med fulla läs- skriv- och konfigureringsrättigheter[36]. Med administratören skapas två virtual hosts med tillhörande topologi och rättigheter i tur och ordning. När en virtual host är skapad måste adminstratören först ge sig själv fulla läs- skriv- och konfigureringsrättigheter i den nyskapade virtual hosten. Administratören kan sedan fortsätta och skapa en exchange[36].

Virtual host med tillhörande exchange skapas utifrån ett tenant-namn eftersom det är så befintligt system fungerar, konsumenter prenumererar på data från en tenant. Exchange skapas av topic typ med topic '#' vilket ger konsumenten alla topics från den tenant-exchange[25]. Valen för typ av exchange och topic görs utifrån att Combitechs befintliga system har en default-topic för varje tenant som går att prenumerera på.

Vidare implementeras den asynkrona post-metoden som IoT-huben skickar meddelande till via proxy. Mikrotjänsten kopplar upp mot den virtual host med samma namn som den tenant meddelandet kommer ifrån. Sedan publiceras meddelandet till tenant-exchange med routin key '#'. Eftersom data ska vara aktuell implementeras inga köer och då lagras inga meddelande när det inte finns några konsumenter uppkopplade mot exchange[45].

4.3.3 Implementation av Konsol Konsument

För att kunna prenumerera på meddelanden i testsyfte och att testa EasyNetQ Manager Client API ytterligare görs en enkel konsol-applikation som med en manager-klient kollar upp vilka virtual hosts som existerar[36]. Låter sedan användaren av applikationen skapa eller logga in som en användare för den valda virtual host[36].

Därefter skapas en event-baserad konsument[52] och en temporär kö[77] som är auto-delete[45]. Konsumentens metod *consume()* görs till en tråd som ligger och väntar på att meddelanden ska komma in till exchange i vald virtual host.

Temporär kö med auto-delete väljs för att man ska kunna ha många olika konsumenter som kopplar upp och ner sig. När uppkoppling sker från en användare skapas en temporär kö dynamiskt och det behöver inte sättas upp någon kö för den användaren i förväg och servern behöver inte veta vilka eller hur många användare som kommer att vara aktiva samtidigt. När nedkoppling sker tas kön bort enligt auto-delete-flaggan och lämnar inte kvar en massa inaktiva köer på servern som måste rensas upp[45][77].

Köer är inte implementerade i förväg i Service Bus som beskrevs i föregående sektion det håller datan aktuell då exchangen kastar meddelanden som kommer in när ingen kö är bunden till den. När en konsument sedan kopplar upp med sin kö får den alla meddelande från det ögonblicket och framåt. Kön är durable ifall något oförutsett skulle hända och ett tidigare tillstånd behöver återskapas för att inte tappa bort någon data.

4.3.4 Sluttest

När alla delar är implementerade görs ett sluttest av den obligatoriska delen av examensarbetet. Applikationerna som de tre nyutvecklade mikrotjänsterna är en del av publiceras till Service Fabric klustret[Appendix D]. Två stycken konsolkonsumenter startas och loggas in till de två av Service Bus uppsatta virtual host för Tenant 1 och Tenant 2[Appendix C]. Att exchanges och köer sätts upp korrekt kontrolleras i via webläsaren och RabbitMQ Management plugin, där kan man också se en översikt på belastningen för hela servern[Appendix B].

Meddelande skickas på två olika topic med Node-RED[Appendix A] och meddelande tas emot i Konsol Konsumenterna[Appendix C]. Lagring av meddelande i objektdatabasen kontrolleras med Mongo Client[Appendix E].

4.4 Avslutningsfas

I den avslutande fasen undersöks Service Fabric och .NET Core med mål att kunna använda samma kodbas i båda systemen enbart med skillnad i konfiguration. Först undersöks hur Service Fabric och dess kompatibilitet med .NET Core har utvecklats. Det identifieras att inkapsling via ASP .NET Core tjänster stöds i Service Fabric, men det är strukturerat enligt Model-View-Controller också känt som MVC mönster och inte full kompabilitet med .NET Core då det är en genväg via inkapsling[64].

Combitech ger i uppgift att göra ett praktiskt test med koden för mikrotjänsten för IoT-hub i en .NET Core applikation för att se vilka bibliotek som har kompabilitet och vilka som inte har det. Det visar sig att som väntat RabbitMQ och MongoDB API fungerar eftersom de bland annat är valda av den anledningen, men inget annat fungerar[68][69]. Fortsatta eftersökningar visar att Service Fabric är problemet då Microsoft har inte utvecklat full kompabilitet för .NET Core än utan kommer i en senare version[63]. Service Fabric har ett SDK för utveckling i Linux med Java eller .Net Core men detta precis som att kapsla in befintlig plattform i ASP .NET Core tjänster innebär omarbete av stora delar av koden och uppfyller inte en av grunderna till examensarbetet nämligen att kunna använda samma i stort sett samma kodbas.

Examensarbetet med utvecklade mikrotjänster, RabbitMQ och MongoDB med tillhörande API lyckas göra det möjligt att använda samma kodbas lokalt, i molnet eller hybrid med enbart konfiguration som skiljer, men vid användning i Linux tar det stopp och är omöjligt att åstadkomma, åtminstone tills vidare då Service Fabric inte är fullt kompatibelt med .NET Core[63].

4.5 Service Fabric Struktur

En del av implementation har inte varit föremål för val eller avvägningar utan har varit enligt strukturen som Service Fabric ramverk har satt upp.

4.5.1 Stateless service

De olika mikrotjänsterna implementerades som stateless service eftersom några tillstånd ej behövde sparas då det identifierades andra lösningar i RabbitMQ. Med en stateless service kommer en enklare struktur än en stateful service bestående av en konstruktor där en stateless service skapas och en asynkron tråd RunAsync som körs kontinuerligt och väntar på olika händelser[54]. Utöver detta kan man implementera vilka metoder man vill. I de olika mikrotjänsterna hämtas alla parametrar in i konstruktorn och metoder implementeras som askynkrona trådar enligt befintlig struktur[54].

4.5.2 Asynkrona Metoder

Eftersom tjänster kommunicerar med varandra i realtid är asynkrona metoder i form av trådar konvention så att ingen tjänst blockerar en annan någon längre tid. Som standard är *RunAsync()*[54] förimplementerad att användas för kontinuerlig uppdatering. I enlighet med det här mönstret implementeras samtliga metoder i alla tjänster, som post()-metoderna och de olika metoderna som opererar på databasen.

4.5.3 Proxy

Proxy fungerar som en pekare och Microsoft har implementerat en struktur där man med hjälp av ett interface IService och en proxy wapper kan nå och anropa metoder i andra tjänster. Det fungerar så att tjänsten som ska använda sig av proxy-pekaren ärver själv från Iservice. Samtidigt implementerar tjänsten som ska vara destination för proxy-pekaren ett nytt interface över de metoder som ska kunna användas från andra tjänster, interfacet ärver också från Iservice[78]. Sedan skapas en proxy till mikrotjänstens metoder med hjälp av en proxy wrapper och interfacet över tjänstens metoder samt en URI med adressen till var tjänsten finns i Service Fabric klustret[78]. Kommunikation via proxy sker från mikrotjänsten för IoT-hub till de andra två tjänsterna för lagring i databas och för Service Bus.

4.5.4 XML Konfiguration

Det finns flera olika XML-filer i en Service Fabric applikation som används för uppsättning och konfigurering. Det finns tre stycken filer local1node.xml, local5node.xml samt cloud.xml som styr beroende på vilken miljö applikationen används i. De bestämmer vad man ska ha för olika konfigureringar beroende på om man ska debugga på en lokal dator med 1 node, 5 noder eller konfigureringen när man publicerar applikationen till molnet eller en server med cloud.xml[79].

I de här filerna får de olika parametrarna sina värden. Parametrarna ska sedan skrivas in i ApplicationManifest.xml[80] där alla tjänster och dess parametrar för applikationen är mappade. Slutligen förs parametrarna in i settings.xml för varje enskild tjänst där enbart parametrar aktuella för den tjänsten finns med. Den här strukturen medför att parametrar bara behöver ändras i en XML-fil och ändringen gäller för hela applikationen ända ut till berörd tjänst. Parametrarna hämtas in i varje utvecklad mikrotjänst i examensarbetet från settings.xml.

4.6 Befintliga Datastrukturer

Den här delen av implementation har inte varit föremål för val eller avvägningar utan har varit enligt befintliga strukturer i Combitechs plattform. Strukturerna har ej varit nödvändiga för att bevisa konceptet men har använts och beskrivs för tydlighet.

4.6.1 Loggning

Combitech har en struktur på plats med tillhörande bibliotek som gör olika sorters loggning av vad som händer i mikrotjänsterna. Den här strukturen implementerats rakt av och parametrarna för loggning hämtas in i konstruktorn som initierar ett logger-objektet. I koden där fel fångas skrivs sedan felen med hjälp av loggern till en loggfil.

4.6.2 JSON Validator
När ett meddelande kommer in till IoT-huben görs en rad verifieringar som är i linje med Combitechs befintliga struktur. Meddelandet tas emot som en byte-array, görs till en JSON-sträng. JSON-strängen valideras att ha korrekt meddelandestruktur med en klass JSON Validator som Combitech har utvecklat.

4.6.3 MessageRoot
Ett verifierat meddelande görs om till ett objekt för meddelanden *MessageRoot* som Combitech har utvecklat. På objektet kan sedan en rad operationer utföras som till exempel lägga till en tidsstämpel som görs i mikrotjänsten för IoT-hub. Datastrukturen har använts som den är i enlighet med befintlig struktur men är ej nödvändig för konceptet.

5 Slutsats

Kapitlet innehåller det viktigaste sammanfattade resultatet som en allmän redogörelse av hur syfte och målformulering har uppnåtts och med svar på frågorna i problemformuleringen.

5.1 Allmänna Slutsatser

Syftet och målformuleringen med examensarbetet har uppfyllts på ett tillfredsställande sätt. De viktigaste aspekterna av examensarbetet har uppfyllts då en hybridlösning har utvecklats och testats med gott resultat. Med de nyutvecklade mikrotjänsterna finns nu också möjlighet att konfigurera plattformen mot lokala komponenter, en lokal IoT-hub och Service bus på en partitionerad RabbitMQ-server och en lokal databas på en MongoDB-server.

De nya utvecklade komponenterna tar in, hanterar och skickar ut data på ett likvärdigt sätt och befintlig kodbas kan med de nya tilläggen användas intakt både lokalt som mot molnet. Skillnaden nu är att när man kör mot lokal IoT-Hub och Service Bus konfigurerar man mot de nya mikrotjänsterna som kommunicerar med RabbitMQ och mot befintliga mikrotjänster när man kör mot samma komponenter i molnet. Mikrotjänsten för kommunikation med databas kan däremot användas i båda fallen då Azure har stöd för MongoDB.

RabbitMQ och MongoDB kan installeras på Linux och tillhörande API är .NET Core kompatibla, så valda komponenter och utvecklad kod för dem är redo att köras på Linux. Problemet som satte käppar i hjulet för att använda samma kodbas på Linux var att Service Fabric inte har fullt .NET Core stöd än. I mån av tid fann examensarbetet att extrauppgiften var omöjlig vid given tidpunkt.

5.2 Problemformulering

1. Vilka likvärdiga lösningar kan ersätta de komponenter i Microsoft Azure som ej kan flyttas till lokal server?

    a. Vilka tjänster kan ersätta objektdatabasen?

        i. För Windows Server

Likvärdig databas att använda med Windows Server är MongoDB. MongoDB är en väl använd objektdatabas som Azure redan har stöd för och kan därför anses likvärdig. Samma kodbas kan användas att köras mot MongoDB lokalt på Windows Server eller mot molnet.

        ii. För Linux Server

Likvärdig databas att använda med Linux Server är MongoDB. Ett önskemål för examensarbetet var att samma tjänst skulle kunna användas för både Windows och Linux. MongoDB har Linux stöd, dess API har .NET Core stöd och är därför ett givet likvärdigt val även för Linux då Azure redan har stöd för MongoDB. Samma kodbas kan användas att köras mot MongoDB lokalt på Linux Server eller mot molnet.

b. Vilka tjänster kan ersätta IoT-huben?

i. För Windows Server

Likvärdig tjänst att använda som IoT-hub med Windows Server är RabbitMQ. AMQP är det viktigaste protokollet för befintligt system förutom MQTT och RabbitMQ är det mest utbredda och kompetenta AMQP-centrerade message broker som examensarbetet har identifierat och kan ersätta samtliga nödvändiga funktioner i befintliga system.

ii. För Linux Server

Likvärdig tjänst att använda som IoT-hub med Linux Server är RabbitMQ. Ett önskemål för examensarbetet var att samma tjänst skulle kunna användas för både Windows och Linux. RabbitMQ har Linux stöd, dess API har .NET Core stöd och är därför ett givet likvärdigt val även för Linux.

c. Vilka tjänster kan ersätta Service Bus?

i. För Windows Server

Likvärdig tjänst att använda som Service Bus med Windows Server är RabbitMQ.
Publish/subscribe-mönstret är den använda strukturen för befintlig Service Bus och det är också mönstret bakom strukturen för en message broker. Därför är RabbitMQ ett logiskt likvärdigt val att också ersätta Service Bus då den implementerar olika varianter av publish/subscribe-mönstret, redan ersätter IoT-hub så kunskap kan återanvändas samt kan ersätta samtliga nödvändiga funktioner i befintliga system.

ii. För Linux Server

Likvärdig tjänst att använda som Service Bus med Linux Server är RabbitMQ. Ett önskemål för examensarbetet var att samma tjänst skulle kunna användas för både Windows och Linux. RabbitMQ har Linux stöd, dess API har .NET Core stöd och är därför ett givet likvärdigt val även för Linux.

2. Kan man göra en hybridlösning där vissa delar av plattformen körs lokalt och andra delar i molnet?

Ja. Examensarbetet har bevisat i tre steg att hybridlösning är möjlig då de av examensarbetet utvecklade mikrotjänsterna har körts mot de lokalt installerade RabbitMQ och MongoDB servrarna samtidigt som de övriga mikrotjänsterna i befintlig plattform har behållit sin konfiguration och körts mot molnet.

3. Hur utvecklar man tjänsten i Service Fabric som kopplar ihop de nya komponenterna med befintlig plattform så den fungerar på ett likvärdigt sätt?

   a. När man får likvärdig data från IoT-hub

Utvecklad mikrotjänst tillsammans med uppsatt topologi för IoT-hub i RabbitMQ-servern motsvarar likvärdig funktionalitet som Combitech använder sig av i Azure IoT-hub. I båda fallen skickar apparater MQTT-meddelanden till IoT-huben som tar emot meddelandet och skickar det till en server som ett AMQP-meddelande. Examensarbetet har löst detta genom RabbitMQ MQTT adapter, en exchange och en kö där mikrotjänsten prenumererar på alla inkommande meddelande. Meddelandet tas i båda fallen emot av en mikrotjänst som en byte-array som valideras och lägger till en tidsstämpel för att skickas vidare till andra mikrotjänster i plattformen. Samma data kommer in till IoT-huben och samma data kommer ut från mikrotjänsten i befintligt system och de av examensarbetet utvecklade komponenterna.

   b. När man sänder likvärdig data från Service Bus

Utvecklad mikrotjänst tillsammans med uppsatt topologi för Service Bus i RabbitMQ- servern motsvarar likvärdig funktionalitet som Combitech använder sig av i Azure Service Bus. I båda fallen kan en konsument via AMQP prenumerera på samtlig data från en tenant på en default topic. Examensarbetet har löst problemet genom att partitionera RabbitMQ-servern och skapa en virtual host för varje tenant med en exchange som konsumenten kan koppla upp sig emot för prenumeration.

   c. När man lagrar likvärdig data i objektdatabasen

Utvecklad mikrotjänst tillsammans med uppsatt databas och kollektioner på MongoDB-servern likvärdig funktionalitet som Combitech använder sig av i Azure databas. I båda fallen tar mikrotjänsten emot meddelandet från en annan mikrotjänst och sparar rådatan i kollektioner sorterade efter tenant. Examensarbetet har löst det genom meddelandet som tas emot i form av en byte-array först konverteras till JSON och sedan till ett BSON-dokument som lagras i databasen.

Eftersom Azure har stöd för MongoDB innebär det att kodbasen för examensarbetets mikrotjänst kan användas rakt av och köras mot MongoDB lokalt på server eller mot molnet.

4. Behövs det ytterligare komponenter för Linux eller kan samma komponenter som är identifierade för Windows användas?

Nej. Valet av komponenterna RabbitMQ och MongoDB stödjer båda operativsystemen Linux och Windows, samt deras API för utveckling stödjer .NET Core. Detta är en förutsättning för möjligheten att använda samma kodbas som grund för båda operativsystemen.

5. Vad krävs för att plattformen ska fungera för användning på Linux?

RabbitMQ och MongoDB kan installeras på Linux och tillhörande API är .NET Core kompatibla, så valda komponenter och utvecklad kod för dem är redo att köras på Linux.
När Service Fabric får fullt stöd för .NET Core återstår att identifiera och ersätta de bibliotek och kod som behöver bytas från .NET till .NET Core. Sedan kan samma kodbas användas mot Windows och Linux Server, lokalt eller mot molnet.

5.2 Reflektion över etiska aspekter

En möjlig följd av examensarbetets arbete är en ökad säkerhet för företag och organisationer i behov av sekretess och skyddande av konfidentiell information.

Att alltid vara uppkopplad mot Internet och molnet kan vara ett problem för många i en allt mer uppkopplad värld och finns många olika företag och organisation i många olika sammanhang som har ett behov av att ha hög säkerhet och kontroll över sin data.

Examensarbetet har berört ett utav de här sammanhangen där det gäller uppkopplingen av av apparater mot Internet. Internet of things är en smart och användbar innovation som kan behövas även för dem som har apparater att koppla upp som samlar in känslig data.

Därför kan det komma samhällsnytta utav möjligheten för Combitech att erbjuda en helt lokal lösning för ett övergripande känsligt system eller en hybridlösning som delvis kan köras lokalt för känsligare delar.

Nyttan kan även komma dem till gagn som inte helt litar på molnet av andra anledningar eller kanske rent av gammal vana som har en möjlighet att prova på fördelarna med ett sammankopplat IoT-system utan att behöva utsätta sig för risken att vara uppkopplad mot molnet och Internet.

5.3 Framtida utvecklingsmöjligheter

Examensarbetes utvecklingsprocess och resultat har lett till flera möjligheter för framtida vidareutveckling. Eftersom det är en konceptvalidering finns det väldigt många möjliga vägar att utveckla systemet både i det korta och långa perspektivet.

Uppskalning av konceptet kan vara värt att undersöka närmare då examensarbetet bara testat konceptet i minimal form. Service Fabric klustret stödjer skalning och man kan sätta upp fler noder med fler instanser av applikationer och mikrotjänster om så behövs. När RabbitMQ-servern används för IoT-hub och Service Bus lokalt så är skalningen inte lika självklar och hur det görs bäst när apparaterna och tenants blir många till antalet. RabbitMQ stödjer klustring via plugin och studier och test av hur man tillämpar det bäst för lastbalansering är en framtida möjlighet. Det samma gäller för MongoDB installerat lokalt, hur lastbalanserar man bäst lagringen av data när skalan blir större med fler tenants och apparater. MongoDB har ett system som kallas Sharding som kan distribuera data över flera instanser i ett kluster vid hög genomströmning av data som är värt att undersöka vidare[81].

I nuvarande system finns möjligheten för en konsument att prenumerera på samtlig data från en tenant via en default topic, med RabbitMQ routing key finns möjligheten att göra en mer specifik indelning. Man skulle till exempel kunna prenumerera på en typ av apparat från en tenant eller kanske bara en specifik apparat, möjligheterna är många. En möjlighet till vidareutveckling är stöd för att meddelanden skickas med fler protokoll. RabbitMQ stödjer förutom MQTT, STOMP, HTTP och AMQP 1.0[82].

Som beskrivs i Analys och Resultat uppstod en del problem när uppgiften att sätta en tidsstämpel när meddelandet köas på RabbitMQ-servern skulle lösas. Problemet vara att MQTT pluginet inte skapade egenskapen för tidsstämpel när den översatte MQTT till AMQP då egenskapen inte finns för MQTT. När plugin för tidsstämpel sedan ska sätta servertiden fanns ingen egenskap att sätta. Eftersom RabbitMQ med plugin har öppen källkod skulle detta kunna åtgärdas genom att modifiera ett av de två pluginen så att plugin för tidsstämpel eller MQTT pluginet skapar egenskapen. Detta kräver dock att sätta sig in i Erlang språket som RabbitMQ är skrivet i. På längre sikt kan man tänka sig att kunskap i Erlang öppnar upp väldigt omfattande möjligheter då man kan skräddarsy eller skapa helt nya plugin efter behov.

En självklar fortsättning är att fullfölja arbetet med att kunna använda samma kodbas i Linux som undersöktes i examensarbetets avslutande fas. När Service Fabric får stöd för .Net Core är möjligheterna att lyckas avsevärt mycket bättre.

6 Terminologi

.NET	En ramverk utvecklad i C# av Microsoft
API	Application Programming Interface
Broker	Mellanprogramvara för att sköta kommunikation mella två klienter
BSON	Binärt JSON-format
JSON	JavaScript Object Notation, Ett textformat för datastruktur
Molntjänst	En tjänst somsom är tillgängligt via Internet
MongoDB	En dokument-/objekt-databas
Multicast	Kommunikation till flera klienter
SDK	Software Development Kit
Service Fabric	Ramverk för hantering av mikrotjänster
SSL	Secure Sockets Layer (SSL)
Tenant	Innehavare av IoT-apparaten som sänder datan
TLS	Transport Layer Security (TLS)
Unicast	Kommunikation till en klient

7 Källförteckning

[1] Fakta om Combitech
http://www.combitech.se/om-oss
[2017-05-28]

[2] What is Azure?
https://azure.microsoft.com/en-us/overview/what-is-azure/
[2017-05-28]

[3] What is IoT-Hub?
https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-what-is-iot-hub
[2017-05-28]

[4] DocumentDB to CosmosDB
https://azure.microsoft.com/sv-se/blog/dear-documentdb-customers-welcome-to-azure-cosmos-db/
[2017-05-25]

[5] DocumentDB introduction
https://docs.microsoft.com/en-us/azure/documentdb/documentdb-introduction
[2017-05-25]

[6] JSON & BSON
https://www.mongodb.com/json-and-bson
[2017-05-25]

[7] Service Bus Fundamentals
https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-fundamentals-hybrid-solutions
[2017-05-25]

[8] Service Bus Overview
https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview
[2017-05-25]

[9] What is AMQP?
https://www.digitalocean.com/community/tutorials/an-advanced-message-queuing-protocol-amqp-walkthrough
[2017-05-26]

[10] What Is MQTT?
http://mqtt.org/faq
[2017-05-26]

[11] What is Message Broker?
https://www.techopedia.com/definition/16959/message-broker
[2017-05-26]

[12] MQTT Publish Subscribe Pattern
http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe
[2017-05-26]

[13] Node-RED Overview
https://nodered.org/
[2017-05-26]

[14] MongoDB Architecture
https://www.mongodb.com/mongodb-architecture
[2017-05-25]

[15] MongoDB C# drivers
http://mongodb.github.io/mongo-csharp-driver/2.4/getting_started/quick_tour/
[2017-05-26]

[16] What is JSON?
http://www.json.org/
[2017-05-26]

[17] What is JSON and BSON?
https://www.mongodb.com/json-and-bson
[2017-05-26]

[18] What is BSON?
http://bsonspec.org/
[2017-05-27]

[19] MongoDB JSON Data Date
https://docs.mongodb.com/manual/reference/mongodb-extended-json/#data_date
[2017-05-27]

[20] Mongo Client User Manual
https://www.mongoclient.com/docs/user_manual.html
[2017-05-26]

[21] Mongo Shell
https://docs.mongodb.com/manual/mongo/
[2017-05-27]

[22] AMQP Concepts
https://www.rabbitmq.com/tutorials/amqp-concepts.html
[2017-05-26]

[23] RabbitMQ Exchange Bindings (E2E)
https://www.rabbitmq.com/e2e.html
[2017-05-27]

[24] AMQP Consume Method
https://www.rabbitmq.com/amqp-0-9-1-reference.html#basic.consume
[2017-05-28]

[25] RabbitMQ Routing Key
https://www.rabbitmq.com/tutorials/tutorial-five-dotnet.html
[2017-05-28]

[26] RabbitMQ Virtual Host
https://www.rabbitmq.com/vhosts.html
[2017-05-28]

[27] AMQP Publish Method
https://www.rabbitmq.com/amqp-0-9-1-reference.html#basic.publish
[2017-05-28]

[28] RabbitMQ Access Control
https://www.rabbitmq.com/access-control.html
[2017-06-01]

[29] RabbitMQ TLS (SSL)
https://www.rabbitmq.com/ssl.html
[2017-06-06]

[30] RabbitMQ Terminal Admin Commands
https://www.rabbitmq.com/man/rabbitmqctl.1.man.html
[2017-06-01]

[31] RabbitMQ How To Enable Plugins
https://www.rabbitmq.com/plugins.html
[2017-06-01]

[32] RabbitMQ Community Plugins
https://www.rabbitmq.com/community-plugins.html
[2017-06-01]

[33] RabbitMQ MQTT-Plugin
https://www.rabbitmq.com/mqtt.html
[2017-06-01]

[34] RabbitMQ Management Plugin
https://www.rabbitmq.com/management.html
[2017-06-02]

[35] RabbitMQ Client API
https://www.rabbitmq.com/dotnet-api-guide.html
[2017-06-02]

[36] EasyNetQ Management API
https://github.com/EasyNetQ/EasyNetQ/wiki/Management-API-Introduction
[2017-06-02]

[37] What Is Raspberry Pi
https://www.raspberrypi.org/help/faqs/#introWhatIs
[2017-06-02]

[38] Service Fabric Overview
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview
[2017-06-02]

[39] Visual Studio Team Service
https://www.visualstudio.com/team-services/
[2017-06-03]

[40] What Is Confluence?
https://confluence.atlassian.com/doc/develop-technical-documentation-in-confluence-226166494.html
[2017-06-03]

[41] Kanban with Atlassian
https://www.atlassian.com/agile/kanban
[2017-06-03]

[42] What is Mosquitto?
http://mosquitto.org/
[2017-06-03]

[43] Broker Capabilities
https://github.com/mqtt/mqtt.github.io/wiki/server-support
[2017-06-03]

[44] MongoDB Introduction
https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb-introduction
[2017-06-03]

[45] AMPQ Queue Declare Method
https://www.rabbitmq.com/amqp-0-9-1-reference.html#queue.declare
[2017-06-03]

[46]MongoDB authorization
https://docs.mongodb.com/manual/core/authorization/
[2017-06-04]

[47]LDAP SASL MongoDB
https://docs.mongodb.com/manual/tutorial/configure-ldap-sasl-openldap/
[2017-06-04]

[48]SASL RabbitMQ
https://www.rabbitmq.com/authentication.html
[2017-06-04]

[49]Rabbit LDAP
https://www.rabbitmq.com/ldap.html
[2017-06-04]

[50] Azure Timestamp
https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-c2d
[2017-06-06]

[51]MongoDB SSL
https://docs.mongodb.com/manual/tutorial/configure-ssl/
[2017-06-06]

[52] EventingBasicConsumer
https://www.rabbitmq.com/releases/rabbitmq-dotnet-client/v3.6.10/rabbitmq-dotnet-client-3.6.10-client-htmldoc/html/type-RabbitMQ.Client.Events.EventingBasicConsumer.html
[2017-06-04]

[53] Ack and Different Types of Confirms RabbitMQ
https://www.rabbitmq.com/confirms.html
[2017-06-04]

[54] Stateful and Stateless Services
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-introduction#stateless-reliable-services
[2017-06-05]

[55] Timestamp Plugin
https://github.com/rabbitmq/rabbitmq-message-timestamp/blob/master/README.md
[2017-06-05]

[56]Timestamp källkod
https://github.com/rabbitmq/rabbitmq-message-timestamp/blob/master/src/rabbit_timestamp_interceptor.erl#L59
[2017-06-04]

[57]MQTT Adapter källkod
https://github.com/rabbitmq/rabbitmq-mqtt/blob/master/src/rabbit_mqtt_processor.erl#L163
[2017-06-05]

[58] RabbitMQ idempotent
https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html
[2017-06-04]

[59] Exchange
https://www.rabbitmq.com/amqp-0-9-1-reference.html#exchange.declare
[2017-06-04]

[60]Shovel Plugin:
https://www.rabbitmq.com/shovel.html
[2017-06-04]

[60]Federation Plugin:
https://www.rabbitmq.com/federation.html
[2017-06-04]

[61] MongoDB .NET Core support
https://mongodb.github.io/mongo-csharp-driver/
[2017-06-04]

[62] MongoDB JSON to BSON Serialization
http://mongodb.github.io/mongo-csharp-driver/2.4/reference/bson/serialization/
[2017-06-04]

[63] Service Fabric .NET Core Support Still Under Progress
https://github.com/Azure/service-fabric-issues/issues/255
[2017-06-04]

[64] Service Fabric First Application
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-create-your-first-application-in-visual-studio
[2017-06-04]

[65] Azure IoT-Hub Protocol
https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-protocol-gateway
[2017-06-04]

[66] Database rankings
https://db-engines.com/en/ranking/document+store
[2017-06-05]

[67] EasyNetQ and .NET Core
https://github.com/EasyNetQ/EasyNetQ/issues/581
[2017-06-04]

[68] RabbitMQ supports .NET Core
https://github.com/rabbitmq/rabbitmq-dotnet-client
[2017-06-04]

[69] MongoDB supports .NET Core
https://code.msdn.microsoft.com/How-to-using-MongoDB-with-74f3e1cf
[2017-06-05]

[70] Service Fabric XML example
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-manage-multiple-environment-app-configuration
[2017-06-06]

[71] MongoDB Supported Operating System
https://docs.mongodb.com/manual/administration/production-notes/
[2017-06-06]

[72] MongoDB BSON usage
http://bsonspec.org/implementations.html
[2017-06-06]

[73] RabbitMQ Log
https://www.rabbitmq.com/event-exchange.html
[2017-06-06]

[74] RabbitMQ Trace
https://www.rabbitmq.com/firehose.html
[2017-06-06]

[75] RabbitMQ Features
https://www.rabbitmq.com/features.html
[2017-06-06]

[76] Develop MongoDB for Azure
https://docs.microsoft.com/en-us/azure/cosmos-db/create-mongodb-dotnet
[2017-06-07]

[77] RabbitMQ Temporary Queue
https://www.rabbitmq.com/tutorials/tutorial-three-dotnet.html
[2017-06-07]

[78] ProxyWrapper
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-secure-communication
[2017-06-07]

[79] Service Fabric Application Configuration
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-manage-multiple-environment-app-configuration
[2017-06-07]

[80] Service Fabric Application Manifest
https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-manage-application-in-visual-studio
[2017-06-07]

[81] MongoDB Sharding
https://docs.mongodb.com/manual/sharding/
[2017-06-07]

[82] RabbitMQ Protocols
https://www.rabbitmq.com/protocols.html
[2017-06-07]

7.1 Figur Referenser

Figur 1: Skapad av examensarbetarna.
[2017-06-08]

Figur 2: Skapad av examensarbetarna.
Inspirerad av: https://i-msdn.sec.s-msft.com/dynimg/IC141963.gif
[2017-06-21]

Figur 3: Skapad av examensarbetarna.
[2017-06-06]

Figur 4: Skapad av examensarbetarna.
Inspirerad av:
https://www.rabbitmq.com/img/tutorials/python-five.png
[2017-06-21]

Figur 5: Skapad av examensarbetarna.
[2017-06-08]

Figur 6: Skapad av examensarbetarna
[2017-06-02]

Appendix F

Figur 7 : Skapad av examensarbetarna
Inspirerad av:
http://lostechies.com/derekgreer/files/2012/03/DirectExchange_thumb1.png
[2017-06-21]

Figur 8: Skapad av examensarbetarna
Inspirerad av:
http://lostechies.com/derekgreer/files/2012/03/FanoutExchange_thumb2.png
[2017-06-21]

Figur 9: Skapad av examensarbetarna
Inspirerad av:
http://lostechies.com/derekgreer/files/2012/03/TopicExchange_thumb2.png
[2017-06-09]

Figur 10: Skapad av examensarbetarna
Inspirerad av:
http://lostechies.com/derekgreer/files/2012/03/HeadersExchange_thumb2.png
[2017-06-09]

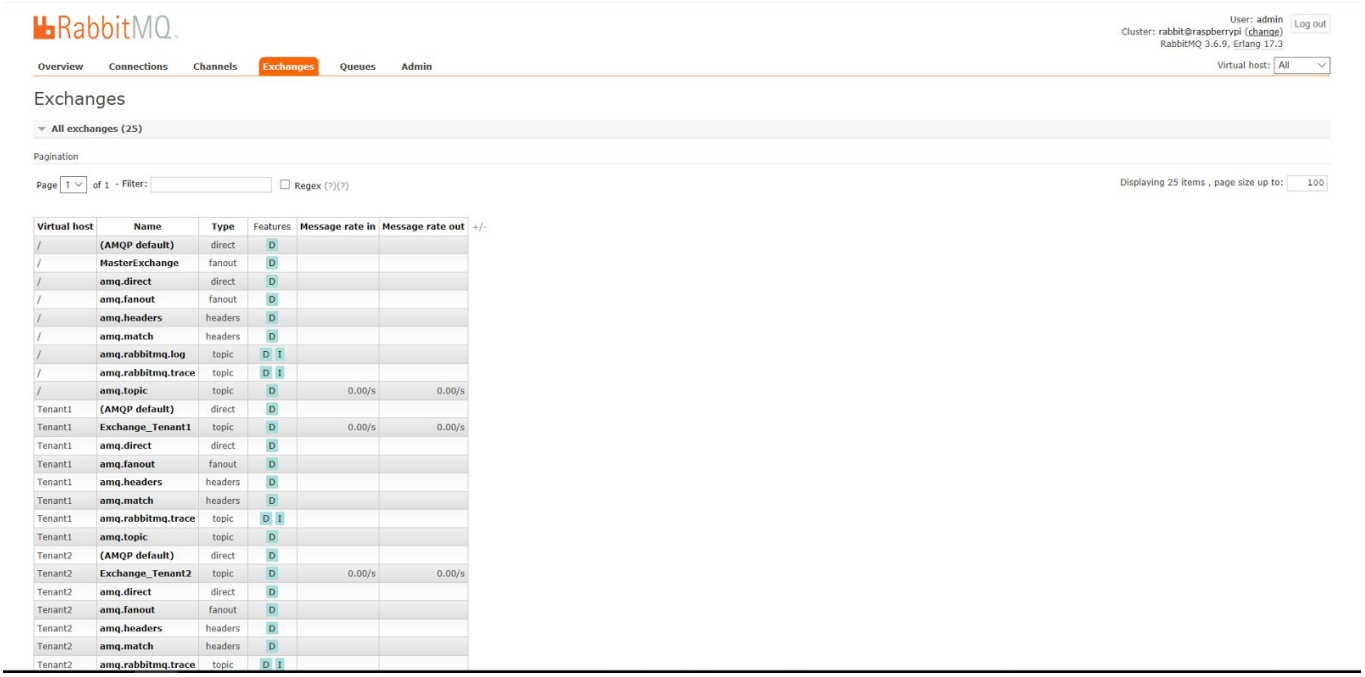
8 Appendix

Appendix A

Node-RED skickar MQTT-meddelande till två olika topic för två olika tenants

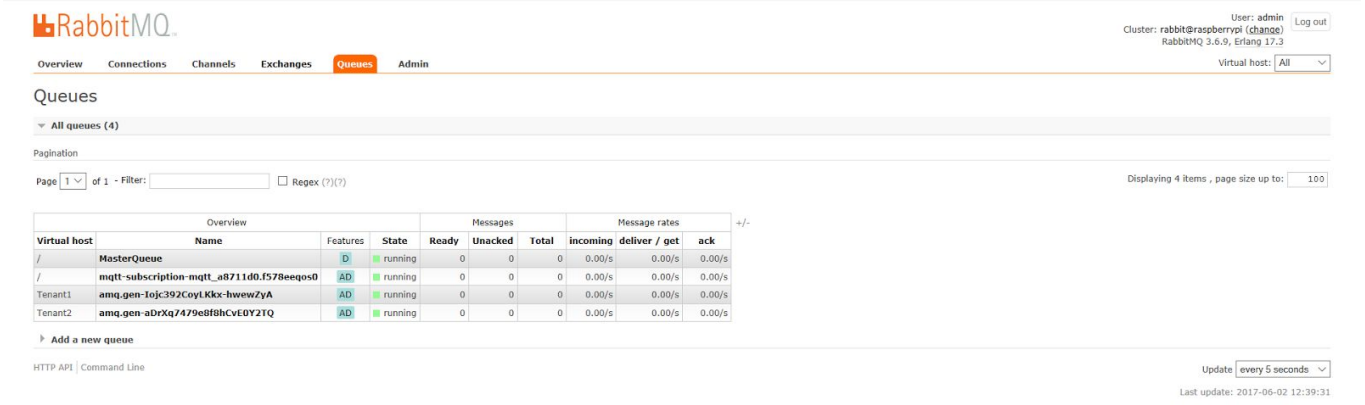
Appendix B

RabbitMQ sätter upp MasterExchange och MasterQueue för IoT-hub. Virtual Host för Tenant 1 och Tenant 2 sätts upp i Service Bus med vars en Tenant_Exchange.



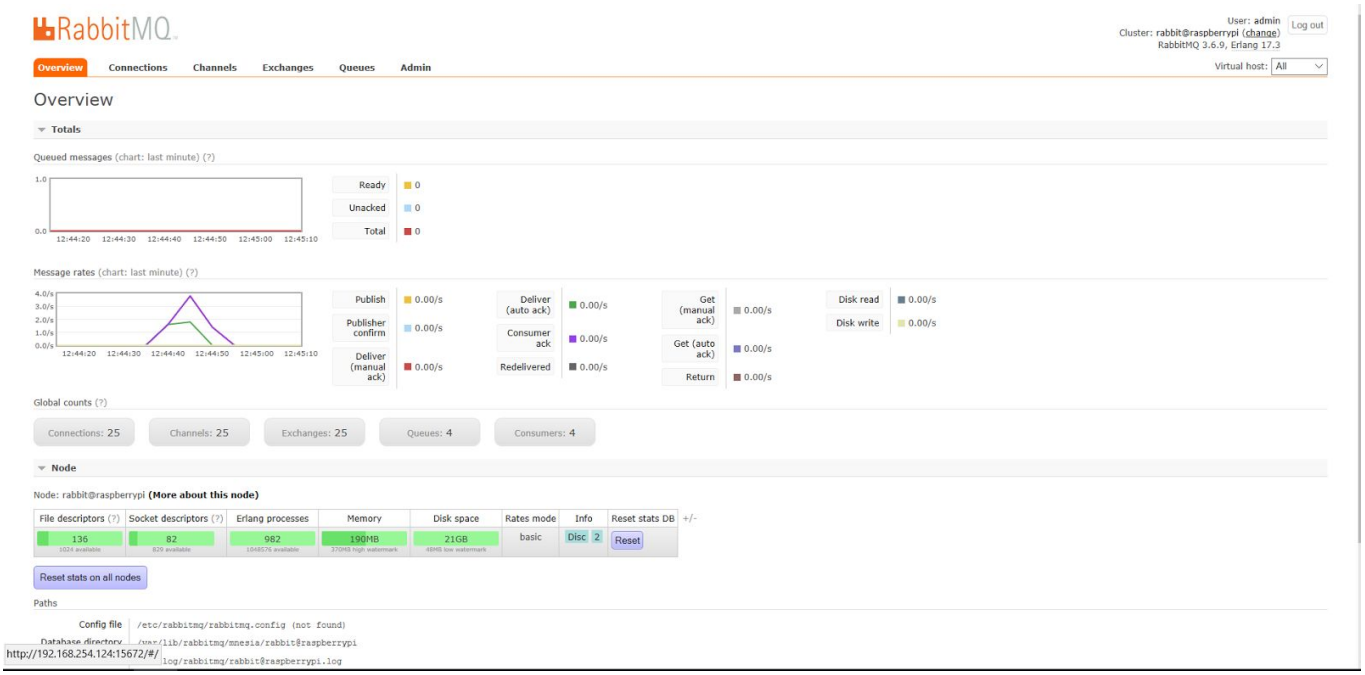
Virtual host	Name	Type	Features	Message rate in	Message rate out
/	(AMQP default)	direct	D		
/	MasterExchange	fanout	D		
/	amq.direct	direct	D		
/	amq.fanout	fanout	D		
/	amq.headers	headers	D		
/	amq.match	headers	D		
/	amq.rabbitmq.log	topic	D I		
/	amq.rabbitmq.trace	topic	D I		
/	amq.topic	topic	D	0.00/s	0.00/s
Tenant1	(AMQP default)	direct	D		
Tenant1	Exchange_Tenant1	topic	D	0.00/s	0.00/s
Tenant1	amq.direct	direct	D		
Tenant1	amq.fanout	fanout	D		
Tenant1	amq.headers	headers	D		
Tenant1	amq.match	headers	D		
Tenant1	amq.rabbitmq.trace	topic	D I		
Tenant1	amq.topic	topic	D		
Tenant2	(AMQP default)	direct	D		
Tenant2	Exchange_Tenant2	topic	D	0.00/s	0.00/s
Tenant2	amq.direct	direct	D		
Tenant2	amq.fanout	fanout	D		
Tenant2	amq.headers	headers	D		
Tenant2	amq.match	headers	D		
Tenant2	amq.rabbitmq.trace	topic	D I		

Node-RED och Konsol Konsument 1 och Konsol Konsument 2 binder sig med temporära köer



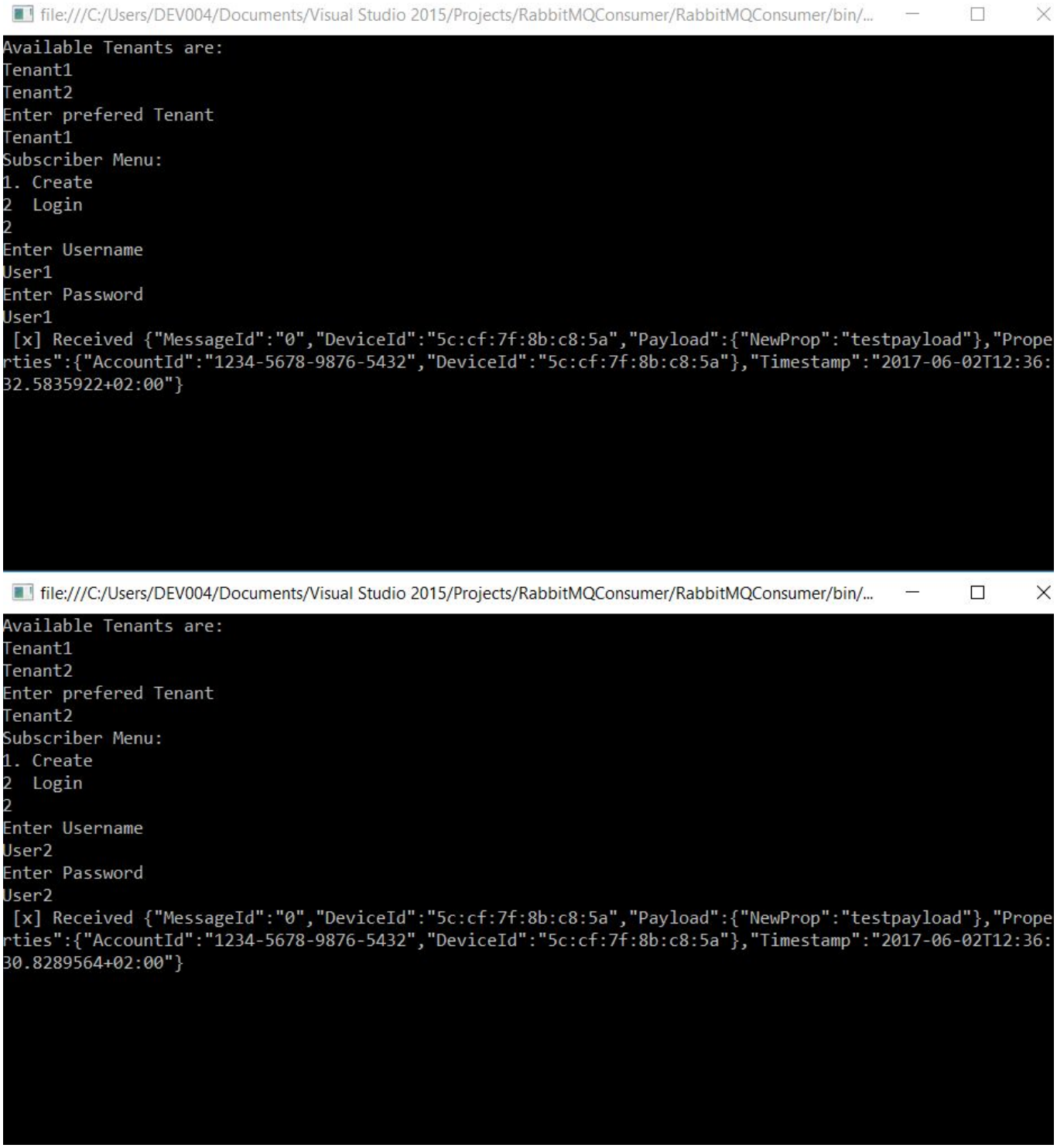
Overview				Messages			Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	MasterQueue	D	running	0	0	0	0.00/s	0.00/s	0.00/s
/	mqtt-subscription-mqtt_a8711d0.f578eeqos0	AD	running	0	0	0	0.00/s	0.00/s	0.00/s
Tenant1	amq.gen-1ojc392CoyLKkx-hwewZyA	AD	running	0	0	0	0.00/s	0.00/s	0.00/s
Tenant2	amq.gen-aDrXq7479e8f8hCvE0Y2TQ	AD	running	0	0	0	0.00/s	0.00/s	0.00/s

RabbitMQ total översikt av inkommande meddelande



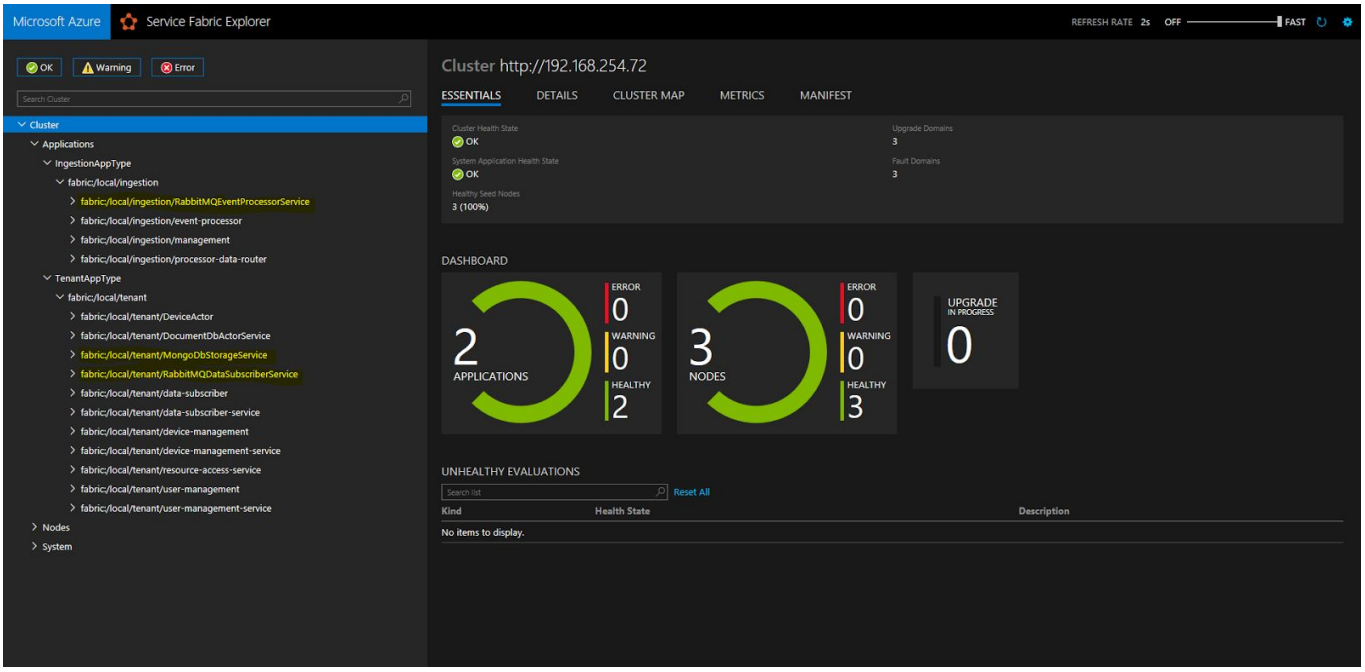
Appendix C

Konsol Konsument 1 och Konsol Konsument 2 får meddelande



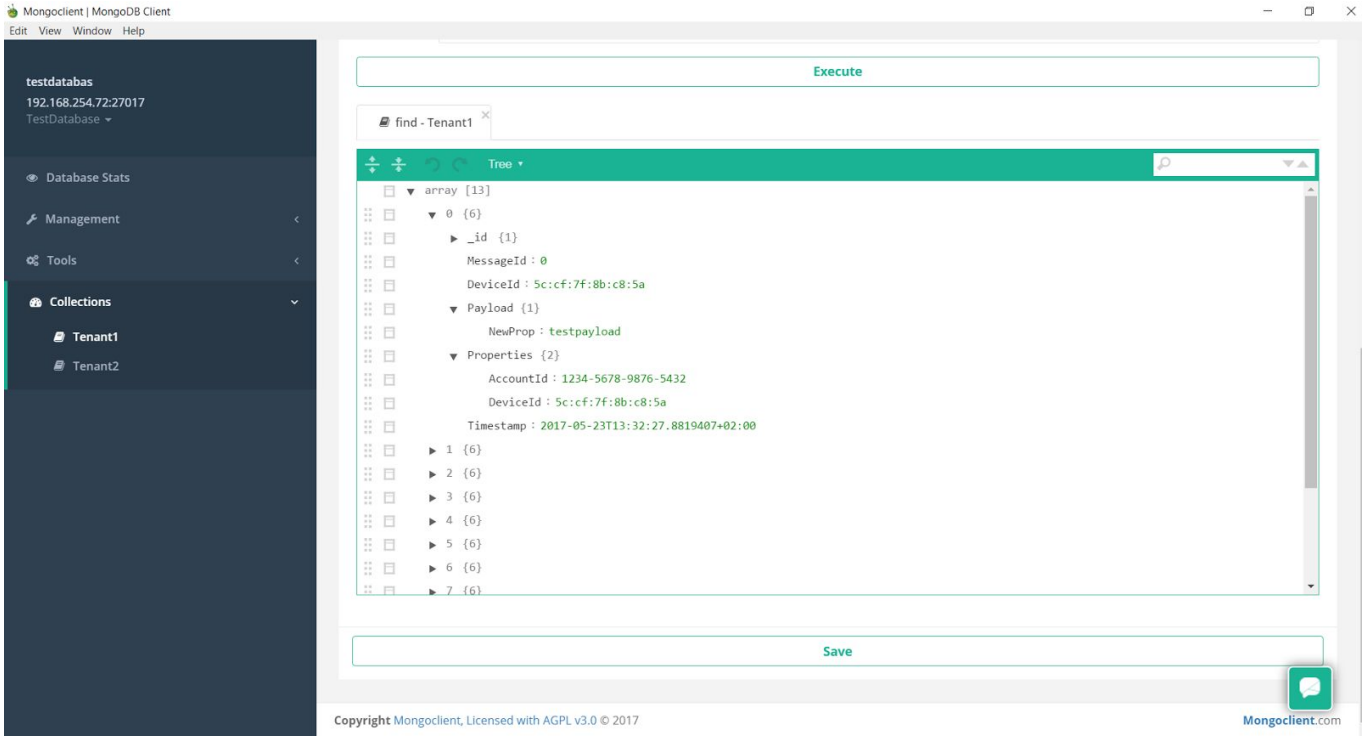
Appendix D

Service Fabric Explorer visar att mikrotjänsterna är igång på klustret



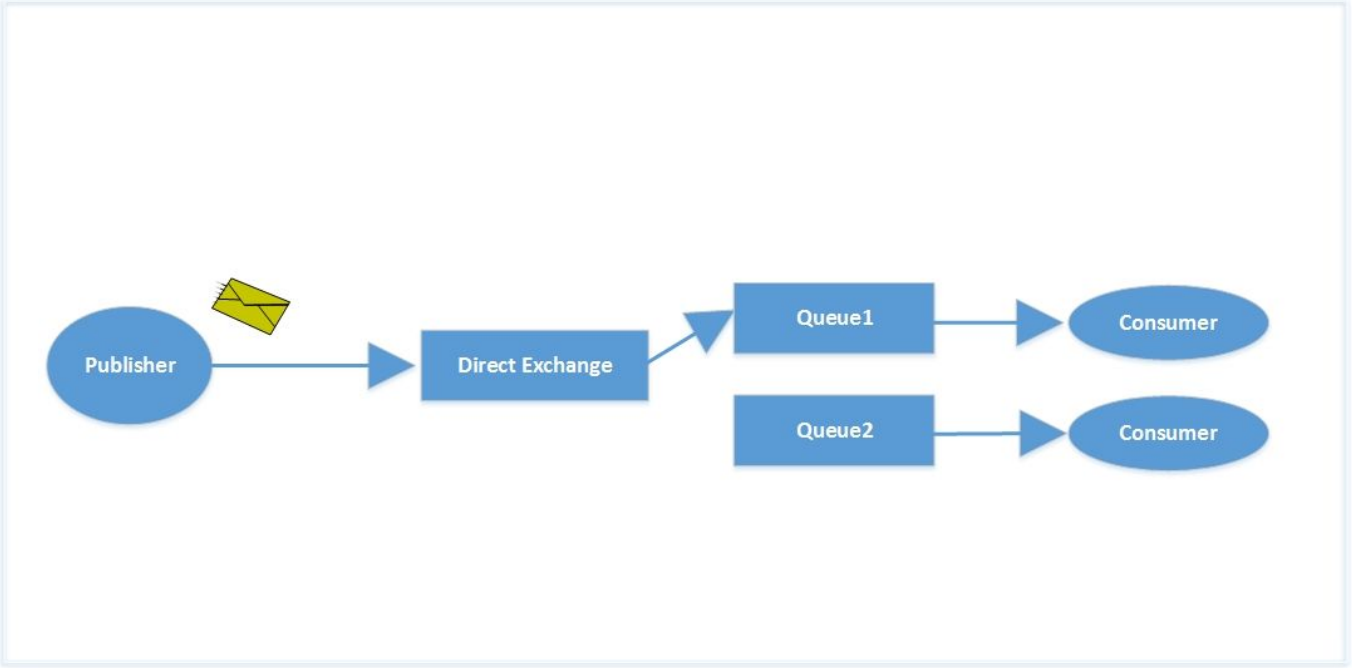
Appendix E

Mongo Client visar att meddelanden lagras i objektdatabasen

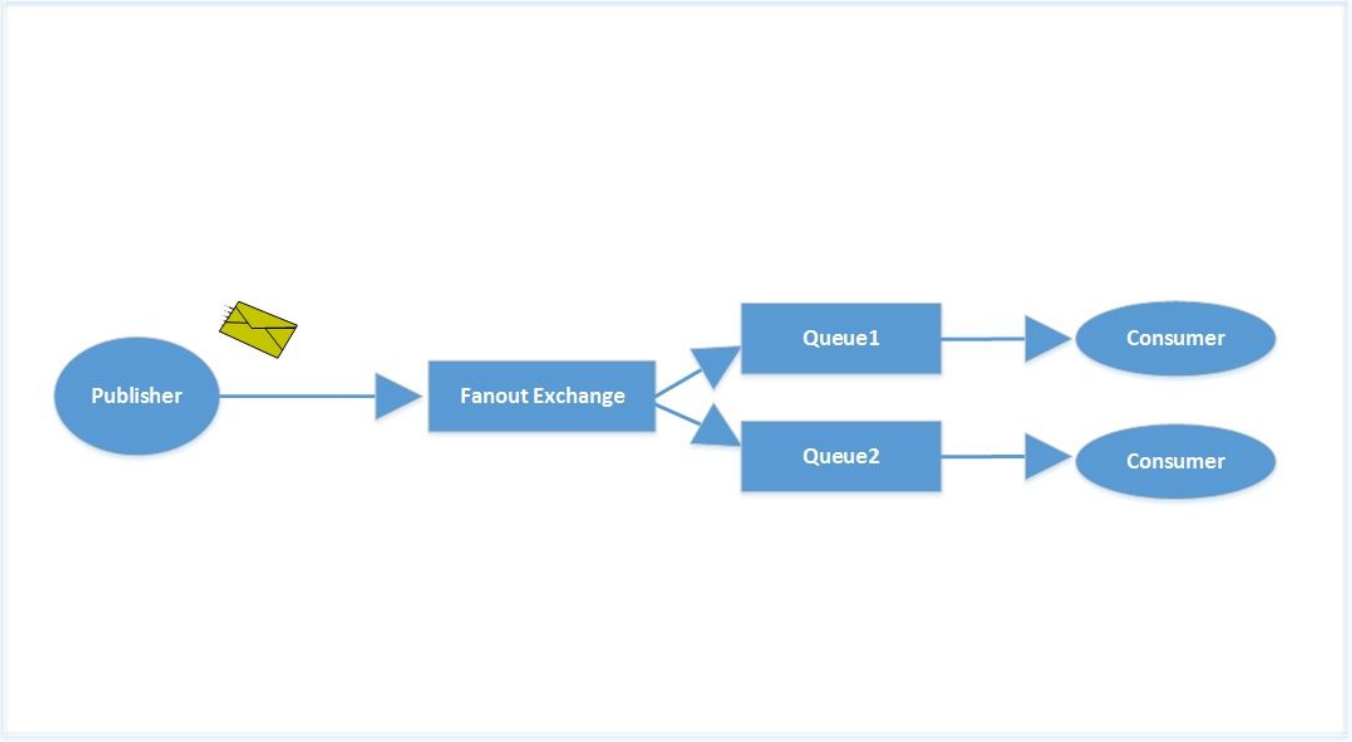


Appendix F

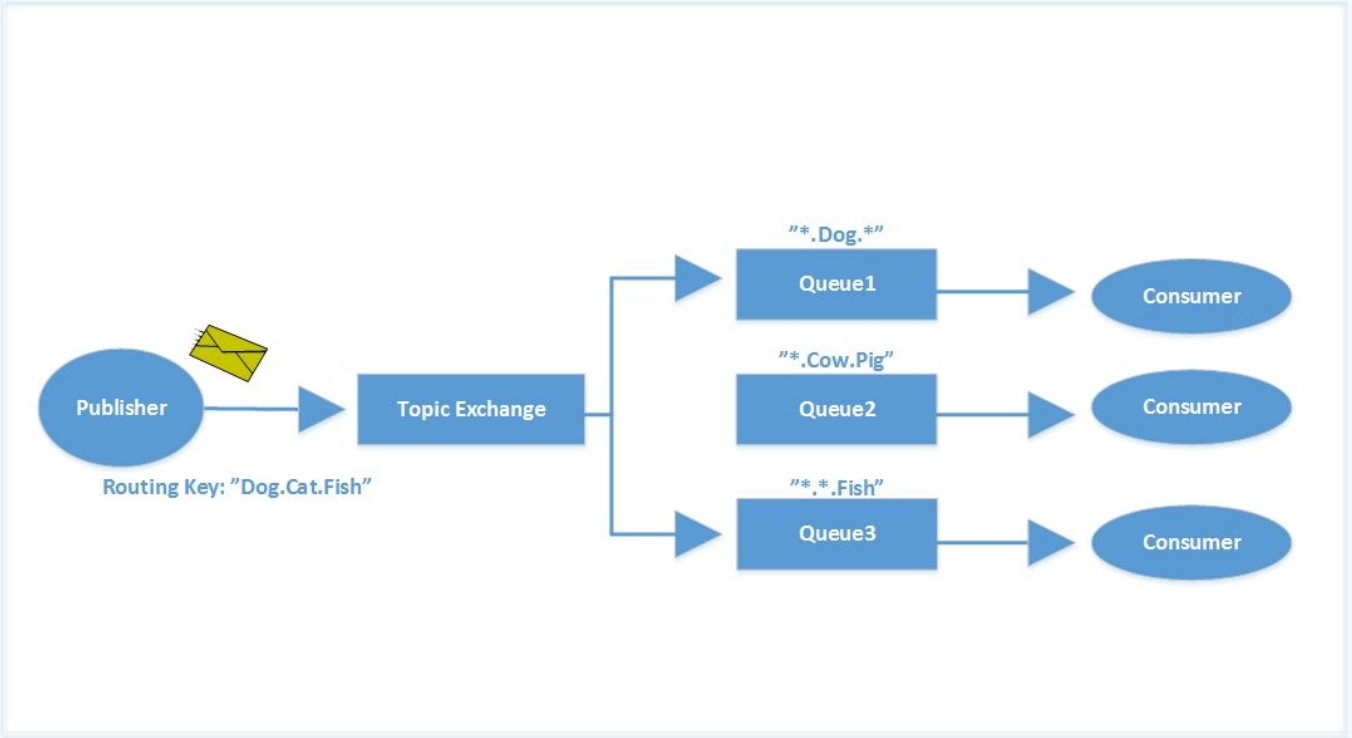
Figur 7: Direct Exchange



Figur 8: Fanout Exchange



Figur 9: Topic Exchange



Figur 10: Header Exchange