HÖGSKOLAN
DALARNA

Examensarbete

En jämförelse av molnbaserade realtidsanalystjänster

Skillnader i användbarhet mellan multi-tenant och single-tenant arkitekturer

A comparison of cloud-based real-time analytics

Differences in usability between multi-tenant and single-tenant architectures

Författare: Kalle Eljas, Dan Johansson
Handledare: Leif Åkerblom
Examinator: Bo Sundgren
Ämne/huvudområde: Informatik
Kurskod: IK2017
Poäng:15
Ventilerings-/examinationsdatum:

Vid Högskolan Dalarna har du möjlighet att publicera ditt examensarbete i fulltext i DiVA. Publiceringen sker Open Access, vilket innebär att arbetet blir fritt tillgängligt att läsa och ladda ned på nätet. Du ökar därmed spridningen och synligheten av ditt examensarbete.

Open Access är på väg att bli norm för att sprida vetenskaplig information på nätet. Högskolan Dalarna rekommenderar såväl forskare som studenter att publicera sina arbeten Open Access.

Jag/vi medger publicering i fulltext (fritt tillgänglig på nätet, Open Access):

Ja ☒ Nej ☐

Förord

Vi vill tacka Leif Åkerblom vår handledare vid Högskolan Dalarna för att han har stöttat oss, visat stort engagemang och varit ständigt tillgänglig under arbetets gång.

Dalarna University
Röda vägen 3
S-781 88 BORLÄNGE

Telephone: +46 (0)23 77 80 00
Fax: +46 (0)23 77 80 50
URL: http://www.du.se/

Sammanfattning

Internet of Things är ett samlingsbegrepp för den utveckling som innebär att olika typer av enheter kan förses med sensorer och datachip som är uppkopplade mot internet. En ökad mängd data innebär en ökad förfrågan på lösningar som kan lagra, spåra, analysera och bearbeta data. Ett sätt att möta denna förfrågan är att använda sig av molnbaserade realtidsanalystjänster. Multi-tenant och single-tenant är två typer av arkitekturer för molnbaserade realtidsanalystjänster som kan användas för att lösa problemen med hanteringen av de ökade datamängderna. Dessa arkitekturer skiljer sig åt när det gäller komplexitet i utvecklingen.

I detta arbete representerar Azure Stream Analytics en multi-tenant arkitektur och HDInsight/Storm representerar en single-tenant arkitektur.

För att kunna göra en jämförelse av molnbaserade realtidsanalystjänster med olika arkitekturer, har vi valt att använda oss av användbarhetskriterierna: *effektivitet*, *ändamålsenlighet* och *användarnöjdhet*. Vi kom fram till att vi ville ha svar på följande frågor relaterade till ovannämnda tre användbarhetskriterier:

- Vilka likheter och skillnader kan vi se i utvecklingstider?
- Kan vi identifiera skillnader i funktionalitet?
- Hur upplever utvecklare de olika analystjänsterna?

Vi har använt en design and creation strategi för att utveckla två Proof of Concept prototyper och samlat in data genom att använda flera datainsamlingsmetoder. Proof of Concept prototyperna inkluderade två artefakter, en för Azure Stream Analytics och en för HDInsight/Storm. Vi utvärderade dessa genom att utföra fem olika scenarier som var för sig hade 2-5 delmål. Vi simulerade strömmande data genom att låta en applikation kontinuerligt slumpa fram data som vi analyserade med hjälp av de två realtidsanalystjänsterna. Vi har använt oss av observationer för att dokumentera hur vi arbetade med utvecklingen av analystjänsterna samt för att mäta utvecklingstider och identifiera skillnader i funktionalitet. Vi har även använt oss av frågeformulär för att ta reda på vad användare tyckte om analystjänsterna.

Vi kom fram till att Azure Stream Analytics initialt var mer användbart än HDInsight/Storm men att skillnaderna minskade efter hand. Azure Stream Analytics var lättare att arbeta med vid simplare analyser medan HDInsight/Storm hade ett bredare val av funktionalitet.

Dalarna University
Röda vägen 3
S-781 88 BORLÄNGE

Telephone: +46 (0)23 77 80 00
Fax: +46 (0)23 77 80 50
URL: http://www.du.se/

Abstract

Internet of Things is a collective term for a development which means that different types of units can be equipped with sensors and computer chips that are connected to the internet. An increased amount of data results in an increased demand for solutions that can store, track, analyze and process data. One way to solve this is to use cloud-based real-time analytics. Multi-tenant and single-tenant are two types of architectures for cloud-based real-time analytics services, which can be used to solve the problems that come with handling the increased data volumes. These architectures differ in terms of complexity of the development.

In this report Azure Stream Analytics represents multi-tenant architecture while HDInsight/ Storm represent single-tenant architecture.

In order to compare these different architectures, we have evaluated them by making use of three usability criteria: effectiveness, efficiency and user satisfaction. Based on these criteria, we wanted answers to the following questions:

- What similarities and differences can we see in development times?
- Can we identify differences in functionality?
- What do developers think about the various analysis services?

We have used a design and creation strategy to develop two artifacts. Then we collected data from the development using multiple data collection methods. We used observations to document how we worked with the development of the analytical services, and to measure development times and identify differences in functionality. We have also used questionnaires to find out what users thought of the real-time analysis services.

The Proof of Concept prototypes consisted of two artifacts, one for Azure Stream Analytics and one for HDInsight/Storm. We evaluated them by performing five different scenarios, each of them had 2-5 different tasks. We simulated a data stream by creating an application that continuously generated random data that we analyzed with the two real-time analysis services.

We came to the conclusion that Azure Stream analytics was initially more useful than HDInsight/Storm but the differences decreased gradually. Azure Stream Analytics was easier to work with during the simpler analyzes but HDInsight/Storm had a wider choice of functionality.

Dalarna University
Röda vägen 3
S-781 88 BORLÄNGE

Telephone: +46 (0)23 77 80 00
Fax: +46 (0)23 77 80 50
URL: http://www.du.se/

HÖGSKOLAN
DALARNA

Innehållsförteckning

HÖGSKOLAN
DALARNA

Bilagor

Bilaga 1 - SUS-formulär

Bilaga 2 - Testordning scenarier och delmål

Bilaga 3 - Azure Stream Analytics källkod

Bilaga 4 - HDInsight/Storm källkod

Bilaga 5 - Azure Stream Analytics fältnoteringar

Bilaga 6 - HDInsight/Storm fältnoteringar

HÖGSKOLAN
DALARNA

Figurförteckning

Tabellförteckning

HÖGSKOLAN
DALARNA

Ordlista

Artefakt - Artefakt betyder i grunden konstgjort föremål (Wikipedia, 2015c). I den här rapporten syftar vi till artefakt som en utvecklad IT-produkt (Oates, 2006).

ASA - Azure Stream Analytics.

Azure - Microsofts molnplattform: en samling integrerade molntjänster som innefattar databearbetning, lagring, data, nätverk och applikationer. (Microsoft Azure, 2015a)

Datorkluster - Är ett antal parallellt arbetande datorer som delar på arbetsbördan.

Event - En enskild händelse som innehåller data.

Event Hub - Kösystem för events, tar emot event och lagrar dessa tillfälligt (max 7 dagar).

GA - General availability är en fas i en mjukvaruprodukts livscykel som innebär att alla nödvändiga kommersiella aktiviteter är färdigställda och är tillgänglig för allmänheten. (Wikipedia, 2015a)

HDIS - HDInsight/Storm.

Historisk data - Redan existerande data.

Multi-Tenant - I en multi-tenant arkitektur delar flera användare (tenants) på samma resurser, men användarna har endast tillgång till sina egna data. (Burgess, 2013)

PaaS - Platform as a Service är en tjänst som ger möjligheten att hyra hårdvara, operativsystem, lagring och nätverkskapacitet via internet. (Rittinghouse & Ransome, 2010)

PoC - Proof of Concept, en prototyp som ska kunna demonstrera en metods genomförbarhet. (Oates, 2006)

Realtidsdata - Data som blivit inmatat i ett system max några få sekunder innan analysen sker. (Mohammed, 2014)

Single-Tenant - Single-tenant är en arkitektur där en enda instans av ett program eller en tjänst samt stödjande infrastruktur tjänar endast en användare (en *tenant*) i taget. (Rouse, 2012)

SQL – SQL (Structured Query Language)är ett programspråk för att hämta och modifiera data i en relationsdatabas. (Wikipedia, 2015e). Ex: SELECT någonting FROM databas

1 Inledning

1.1 Bakgrund

Utvecklingen för vilken teknisk utrustning som kan vara uppkopplad mot internet har förändrats genom åren. Idag är det en självklarhet att bärbara datorer, smartphones och surfplattor är uppkopplade. Nästa steg i utvecklingen är det som IT-branschen brukar benämna som Internet of Things (IoT). (Augustsson, 2013)

Internet of Things är ett samlingsbegrepp för den utveckling som innebär att t.ex. fordon, maskiner, hushållsapparater eller portabla enheter kan förses med sensorer och datachip som är uppkopplade mot internet. I takt med att priserna pressas ned på sensorer och datachip så blir det ekonomiskt möjligt att ansluta fler saker till internet, vilket också kraftigt bidrar till att datamängden ökar. (Augustsson, 2013; Morgan, 2014)

Osman et al. (2013) skriver om hur den massiva datamängd som genereras av diverse IT-tjänster ställer höga krav på insamling och analys. De pekar på studier som visar att datamängden kommer att öka 26 ggr från år 2013 och fem år framåt. Philip Howard som är analytiker på Bloor Research förutspår att denna utveckling kommer att ge en ökad förfrågan på verktyg för att analysera data i realtid (Vijayan, 2015).

Trafikövervakning är ett exempel där det finns ett behov att analysera och bearbeta data i realtid. Information från uppkopplade bilar och övervakningskameror kan visa läget i trafiken så bilister kan välja en annan väg för att slippa köer. (Augustsson, 2013)

Patrick Moorhead, analytiker på Moor Insights & Strategy säger att realtidsanalys av stora datamängder kan kosta företag tiotals miljoner dollar i hård- och mjukvara. För företag som inte har resurser att sätta upp egna realtidsanalyssystem så kan molntjänster för realtidsanalys vara lösningen som sparar tid och pengar när det gäller underhåll och installation. Molntjänster kan också möjliggöra funktionalitet som företag inte skulle ha möjlighet att utveckla på egen hand. (Gaudin, 2014; Marston, 2011)

Microsoft Azure är en molnplattform med en samling integrerade molntjänster, två realtidsanalystjänster som tillkom år 2015 var HDInsight/Storm och Azure Stream Analytics. (Microsoft Azure, 2015a)

HDInsight är en molnplattform som möjliggör för företag att utnyttja stor datorkraft utan komplexiteten och kostnaden för installation, förvaltning och support (Rajesh, 2013). Vid realtidsanalys används HDInsight tillsammans med Apache Storm som är en open-source realtidsanalysplattform som kan bearbeta data i realtid. Dessa två plattformar erbjuds som en kombinerad tjänst i Azure som HDInsight/Storm. (Franks, 2015c)

HDInsight/Storm har en single-tenant arkitektur vilket innebär att varje användare äger sin egen instans av tjänsten. Instansen kan anpassas för att möta användarens specifika behov och önskemål, speciellt vid de tillfällen då användaren har tillgång till källkoden. (Feddersen, 2015; Khalil, 2013; Rouse, 2012;) Några problem med denna typ av arkitektur är dock att det krävs en hel del programmering och underhåll. (Santurkar et al., 2014)

Att använda multi-tenant arkitektur kan var lösningen på dessa problem. Multi-tenant arkitektur möjliggör en mer standardiserad instans som kan hantera flera användare samtidigt och genom detta använda mindre resurser. (Pathirage et al., 2011) Dock så betyder det att tjänster med multi-tenant arkitekturer har begränsade anpassningsmöjligheter för varje enskild användare eftersom en gemensam kodbas används. Tjänsterna måste därför istället kunna erbjuda breda konfigurationsmöjligheter. (Krebs et al., 2012)

Azure Stream Analytics är en realtidsanalystjänst som har en multi-tenant arkitektur. Azure Stream Analytics möjliggör för utvecklare att snabbt och enkelt kombinera dataströmmar med historisk data för att kunna göra analyser i realtid. Enligt Microsoft kommer Azure Stream Analytics att minska komplexiteten vid utveckling av realtidsanalysfunktioner. (Stokes, 2015; SQL server team, 2015; Schaffner, 2013)

1.2 Samarbetspartner

Altran är ett globalt IT- företag som är verksamt i över tjugo länder i Europa, Asien och Amerika. År 2011 etablerade Altan sitt kontor i Borlänge. Uppdraget är att hjälpa organisationer och företag att skapa och utveckla nya produkter och tjänster. Altrans kunder finns inom informationsintensiva verksamheter liksom inom forskning och utveckling. (Altran, u.å.)

Altran söker ett analysverktyg för realtidsdata som är enkelt att använda och som är snabbt och lätt att komma igång med. Altran ser att den globala trenden går mot användning av molntjänster vilket innebär att man inte behöver utveckla analysverktyget på egen hand. Detta gör att molnbaserade realtidsanalystjänster är extra intressanta för Altran.

Eftersom Altran är samarbetspartner med Microsoft så vill de kunna erbjuda sina kunder kompletta Microsoftlösningar, en del i detta är att sätta upp ett analysverktyg som kunderna med viss enkelhet själva kan använda i sin verksamhet för att kunna analysera sina datamängder. De tror att Azure Stream Analytics kan vara lösningen på problemet.

1.3 Problematisering

Den snabba tekniska utvecklingen gör att data genereras i stora mängder och dessa bör hanteras effektivt. Ett sätt att göra detta på är att utnyttja realtidsanalyser, men att skapa egna sådana system är resurskrävande och lösningen kan instället vara att använda molntjänster. Att använda molnbaserade realtidsanalystjänster med single-tenant arkitektur kräver tekniskt kunnande i utvecklingsarbetet medan en multi-tenant arkitektur är mer standardiserad och kräver mindre resurser.

För att kunna göra en jämförelse av molnbaserade realtidsanalystjänster med olika arkitekturer: multi-tenant respektive single-tenant, har vi valt att utgå ifrån användbarhetskriterierna: effektivitet, ändamålsenlighet och användarnöjdhet.

Användbarhet definieras enligt ISO-normen 9241-11 som följande:
Den grad i vilken användare i ett givet sammanhang kan bruka en produkt för att uppnå specifika mål på ett ändamålsenligt, effektivt och för användaren tillfredsställande sätt (användarnöjdhet) (Wikipedia, 2015b). Vi kom fram till att vi ville ha svar på följande frågor relaterade till ovannämnda tre användbarhetskriterier:

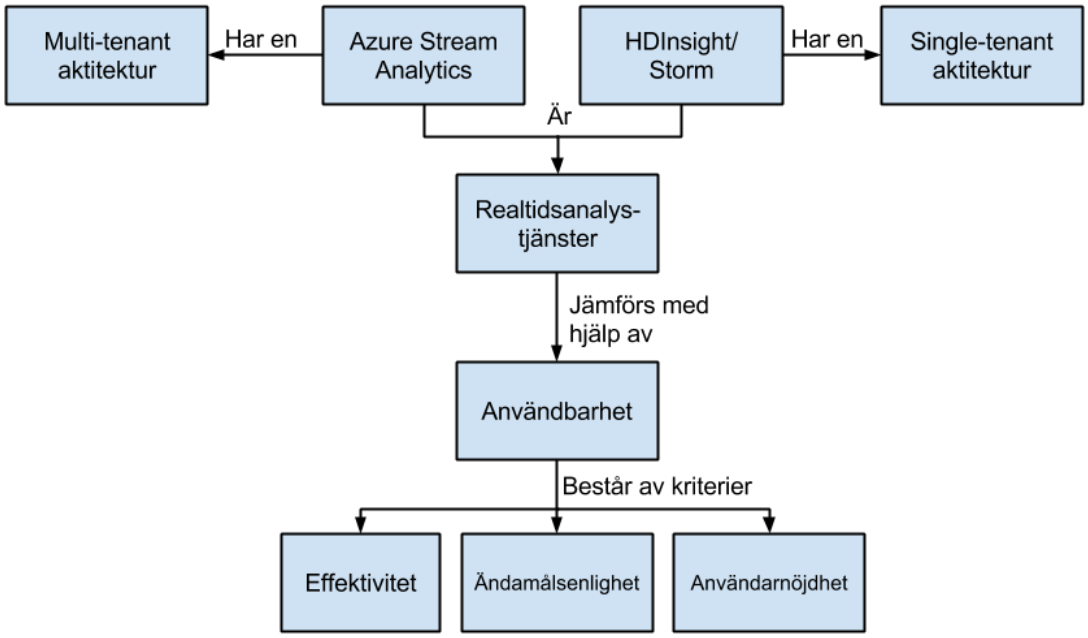
- Vilka likheter och skillnader kan vi se i utvecklingstider?
- Kan vi identifiera skillnader i funktionalitet?
- Hur upplever utvecklare de olika analystjänsterna?

1.4 Syfte

Syftet med arbetet är att med hjälp av två Proof of Concept (PoC) prototyper jämföra molnbaserade realtidsanalystjänster med olika arkitekturer: multi-tenant respektive single-tenant, samt förklara likheter och skillnader i användbarhet.

1.5 Begrepp

Figur 1 nedan beskriver hur nyckelbegreppen som används i den här rapporten är relaterade till varandra. Azure Stream Analytics och HDInsight/Storm är två molnbaserade realtidsanalystjänster med olika arkitekturer. Vi jämför realtidsanalystjänsterna med hjälp användbarhetskriterierna effektivitet, ändamålsenlighet, och användarnöjdhet.



Figur 1 - Begreppsgraf

1.6 Avgränsning

Vi kommer att avgränsa oss till att undersöka Microsofts molnbaserade realtidsanalystjänster Azure Stream Analytics och HDInsight/Storm under utvärderingen av Proof of Concept (PoC) prototyperna. I detta arbete representerar Azure Stream Analytics en realtidsanalystjänst med multi-tenant arkitektur medan HDInsight/Storm representerar en realtidsanalystjänst med single-tenant arkitektur. Anledningen till att vi valde just dessa instanser av molnbaserade realtidsanalystjänster var att vår samarbetspartner Altran erbjuder Microsoftprodukter och är därför intresserade av dessa. Vi kommer inte att behandla big data i den här rapporten utan kommer istället att fokuserat på strömmande data då detta var mer relevant för vårt arbete. Storleken på datamängden ansåg vi inte var relevant för vår undersökning utan det viktiga var att data kunde analyseras i realtid. Hädanefter kommer vi att använda oss av förkortningen ASA för Azure Stream Analytics.

2 Teori

2.1 Tidigare forskning

Pathirage et al. (2011) föreslår att multi-tenant arkitekturer kan användas för att låta företag outsourca sina mindre kritiska affärsfunktioner till någon tredje part som har en betala-för-det-du-använder modell. För att detta ska vara ekonomiskt hållbart behövs en molnbaserad mellanprogramvara som utnyttjar delning av resurser och kan minimera kostnader för resurser som inte används. Genom en multi-tenant arkitektur ska det bli möjligt att låta användare dela på samma resurser och samtidigt behålla integriteten för användarna. Även om den här konferensrapporten inte handlar om realtidsanalys så har den hjälpt oss att förstå vad multi-tenant arkitektur innebär och att det är ett vedertaget begrepp. Khalil et al. (2013) skriver i deras rapport att molntjänster är ett billigare alternativ till lokala lösningar men att det finns säkerhetsrisker. En möjlig lösning är att använda IDS (Intrusion Detection System) integrerat i molnet. De presenterar ett IDS som respekterar multi-tenant arkitekturer för att ge användare möjligheter att i stor utsträckning konfigurera en applikation utan att ha tillgång till källkoden. Krebs et al. (2012) skriver om multi-tenant arkitektur som ett nytt koncept och ger en översikt över viktiga saker att överväga vid användning av multi-tenant arkitekturer. De definierar multi-tenant och förklarar hur arkitekturen skiljer sig från liknande koncept. Vi har använt både Khalil et al. (2013) och Krebs et al. (2012) rapporter för att bygga upp en teori om både multi-tenant och single-tenant arkitekturer.

Osman et al. (2013) skriver om hur den massiva datamängd som genereras av diverse IT-tjänster ställer höga krav på insamling och analys. De pekar på studier som visar att datamängden kommer att öka 26 ggr från år 2013 och fem år framåt. Även om det finns existerande teknik som kan hantera stora datamängder så kan molntjänster och högpresterande analysverktyg leda till en smartare molndriven IT-värld. Osman et al. (2013) nämner att tidigare analystjänster som Map-reduce, Hive och Pig inte är lämpliga lösningar för molnbaserade realtidsanalyser. Däremot så argumenterar de för andra lösningar som Yahoo S4, Aurora, Apache Spark och Apache Storm. Den här rapporten var en nyckel för oss när vi började undersöka alternativ till den multi-tenant baserade realtidsanalystjänsten ASA. Enligt Osman et al. (2013) har single-tenant baserade Apache Storm flera liknade tjänster som t.ex. Apache Spark, Aurora och Yahoo S5. Vi anser att rapporten stödjer den generalisering av HDInsight/Storm som vi gör i slutsatsen av vår studie.

För att få större förståelse för hur Apache Storm fungerar använde vi oss av en rapport av Im et al. (2014) som föreslog ett ramverk för att använda parallella uppgifter med Apache Storm. Den innehöll lättförståelig information om Apache Storm och dess komponenter. Vi har inte använt den i utvecklingsarbetet utan bara under teoribildningen. Några tidigare arbeten om ASA har vi inte hittat, troligtvis av det enkla skälet att under tiden för litteratursökningen var ASA inte ännu släppt i general availability (GA). Vi fick använda oss av ett white paper från Microsoft som bestod av en introduktion, exempel på scenarier där ASA skulle användas samt information om arkitekturen och komponenterna. (Feddersen, 2015)

2.2 Realtidsanalys

I dagens konkurrensutsatta miljö har konsumenterna höga förväntningar. Därför är de affärsbeslut som är baserade på de mest aktuella uppgifter, de beslut som kan förbättra kundrelationer, öka intäkter och maximera driftseffektiviteten. Hastigheten på dagens databehandlingssystem har gjort att företag har skiftat fokus från den klassiska hanteringen av historiska data, till hantering av data i realtid. (Highleyman et al., 2013)

En väsentlig skillnad mellan realtidsanalys och analys av historiska data är att realtidsanalys baseras på realtidsdata och att analysen sker i nära realtid. Realtidsdata är data som blivit inmatade i ett system max några få sekunder innan analysen sker. Om ett beslut inte kan göras inom denna tidslinje så kan i många fall beslutet hinna bli meningslöst. Därför är det kritiskt att de data som behövs för ett specifikt beslut görs tillgängliga i rätt tid, och att analysen i sig kan göras på ett snabbt och pålitligt sätt. (Mohamed, 2014)

Ett exempel på system där realtidsanalys är tidskritiskt är *Indian Tsunami Early Warning* som varnar för att en tsunami eller jordbävning kan vara nära förestående. Systemet består av en mängd sensorer som ständigt genererar realtidsdata, dessa data måste analyseras i realtid för att en eventuell evakuering ska hinna genomföras innan katastrofen är ett faktum. (ibid.)

Realtidsanalyser förlitar sig inte bara på realtidsdata, utan ibland även historiska data som kan vara t.ex. kartor, tidigare transaktioner, situationer och beslut. Att hantera stora historiska datamängder tillsammans med realtidsdata kan var en stor utmaning, men genom att dela in dessa data i mindre set av data, så kan tiden för analysprocessen minskas. Att processa både realtidsdata och historiska data för att hitta samband är ofta resurskrävande för systemet. Därför är det viktigt att de beräkningsalgoritmer som används är optimerade och inte använder onödigt mycket beräkningskraft. Även om algoritmerna är optimerade så krävs det ofta stor beräkningskraft för att utföra realtidsanalyser, detta medför att högpresterande plattformar används. Dessa plattformar kan utrustas med särskild hårdvara för att utföra dessa algoritmer eller så kan plattformen använda sig av datorkluster för att fördela arbetsbördan. (ibid.)

2.3 Molntjänster

Molntjänster har utvecklats till att idag bli en av de mest använda och utbredda teknikerna då företag kan utnyttja kraften hos superdatorer utan att de behöver göra stora infrastruktur-investeringar samt att de bara behöver betala för de resurser som används. (Osman et al., 2013)

Molntjänster innebär att data och applikationer finns tillgängliga via internet istället för lokalt på en dators hårddisk. Hög prestanda och datorkraft är idag viktigt för många företag då IT-systemet ofta är en del av kärnverksamheten. Att använda sig av molntjänster som levererar IT-tjänster kan kraftigt minska kostnaderna både vad gäller underhåll och installation. Molntjänster kan också möjliggöra funktionalitet som företag inte skulle ha möjlighet att utveckla på egen hand. (Marston, 2011)

Platform as a Service (PaaS) är en av typ av molntjänst som erbjuder en plattform för applikationsutveckling. Med hjälp av verktyg som ingår i PaaS som t.ex. databaser, programvara och utvecklingsverktyg så kan användarna skapa programvaruapplikationer. I tjänsten PaaS ingår infrastruktur vilken också administreras av leverantören. Några exempel på PaaS är: Google AppEngine, Amazon Web Services och Microsoft Azure. (Bojanova, 2011)

2.3.1 Azure

Microsoft Azure är en molnplattform med en samling integrerade molntjänster. Azures molntjänster kan utnyttjas utan att man behöver investera i egen hårdvara. Azure erbjuder integrerade tjänster för databearbetning, lagring, nätverk och applikationer. Via användargränssnittet Management portal (Azureportalen) kan användaren administrera de flesta tjänster som Azure erbjuder. (Microsoft Azure, 2015a)

Microsoft Azure har en modell där nyckeln är att användare ska kunna be om resurser och använda dessa när behov finns, och kunna släppa resurserna när de inte längre används. Detta bör leda till att applikationer som inte används borde vara nästintill kostnadsfria. Därför borde PaaS som Azure stödja applikationer som ägs av flera användare inom samma serverutrymme medan resurser fördelas när användare behöver dem. Detta är möjligt att göra genom att köra en virtuell maskin (VM) per användare, s.k. single-tenant arkitektur, men detta är slöseri på resurser när användarna inte använder applikationen. Det tar tid att starta en VM, och ofta tar det så pass lång tid att de initiala förfrågningarna från användaren misslyckas, därför är det oftast inte praktiskt att tillhandahålla och starta VM's när en användare behöver resurser. Att använda multi-tenant arkitektur kan var lösningen på dessa problem. Multi-tenant arkitekturer möjliggör att en serverinstans kan hantera flera användare samtidigt och genom detta köra infrastrukturen med mycket mindre resurser. (Pathirage et al., 2011)

2.3.2 Multi-tenant arkitektur

En multi-tenant tjänst körs som en instans men används av många användare (tenants) samtidigt. En användare kan vara en enskild person eller flera personer, t.ex. kan ett företag med flera personer ses som en användare. Inom ett företag så är det vanligt att personer har olika rättigheter till viss information, därför är det också viktigt att personer kan tilldelas olika rättigheter i systemet/tjänsten även om alla personer i företaget ses som en användare (tenant). (Khalil et al., 2013)

I en multi-tenant arkitektur har varje specifik användare en egen vy mot tjänsten. I den vyn ingår de data och konfigurationer för tjänsten som är unika för varje användare. Genom att lagra data på olika partitioner så separerar man data så att de olika användarna endast har tillgång till sina egna data. De konfigurationer som en användare gör av en tjänst som att t.ex. ändra tema påverkar inte de övriga användarna. Eftersom det endast körs en instans av tjänsten så delar alla användare samma kodbas. Det betyder att kodbasen inte kan anpassas för varje enskild användare utan tjänsten måste istället kunna erbjuda breda konfigurationsmöjligheter för att användarna ska kunna anpassa den efter sina specifika behov. (ibid.)

Eftersom flera användare utnyttjar samma instans av en tjänst så minskar möjligheten för användaren att anpassa tjänsten. Detta innebär inte att funktionaliteten i tjänsten är begränsad, men att anpassningen av tjänsten för den enskilde användaren blir begränsad. Därför passar multi-tenant arkitekturer bra för användare som inte har behov av att anpassa tjänsten efter sina egna behov. (Burgess, 2013; Krebs et al., 2012)

Den ekonomiska aspekten gällande multi-tenant arkitekturer är särskilt tilltalande för mindre företag då man i denna typ av arkitektur delar många av tjänstens (instansens) kostnader mellan användarna. Detta bidrar till att även mindre företag och organisationer kan få tillgång till mer komplexa system som annars skulle ha varit för ekonomiskt betungande att investera i på egen hand. (Wikipedia, 2015d)

Det finns några utmaningar som en multi-tenant arkitektur måste ta sig an och lösa för att kunna ge alla användare likvärdigt hög kvalitet på tjänsten eller applikationen som erbjuds:

- Affinitet - Definierar hur varje person hos en användare binds till noder för att processas.
- Varaktighet/stabilitet - Hur ska varje användare och deras data hanteras? vilken typ av databasdesign ska användas för att säkerställa att data separeras mellan användare.
- Prestandaisolering - Hur ska resurser fördelas mellan varje unik användare för att balansera prestandan?
- Tjänstdifferentiering - Möjliggör att användare kan tilldelas resurser med individuellt anpassad kvalitet.
- Anpassningsbarhet - Tjänsten eller applikationen bör kunna erbjuda breda konfigurationsmöjligheter.

Det finns två huvudsakliga säkerhetsrisker med multi-tenant arkitekturer generellt. Detta gör att denna typ av arkitektur kräver högre säkerhetsstandard än single-tenant arkitektur. Den första risken gäller virtuell infrastruktur där en fysisk maskin delas av flera användare. Teoretiskt sett skulle en användare kunna kringgå den implementerade säkerheten och därmed kunna övervaka vad andra användare gör. Den andra risken är att en användares data kan ses av andra användare genom en dåligt implementerad åtkomsthantering eller någon form av bugg i mjukvaran. (Khalil et al., 2013)

2.3.3 Single-tenant arkitektur

I en single-tenant arkitektur tjänar en enda instans av ett program eller en tjänst samt stödjande infrastruktur en användare (en tenant) i taget. I en single-tenant arkitektur äger en användare en instans av en tjänst och den kan anpassas för att möta användarens specifika behov och önskemål, speciell vid de tillfällen då användaren har tillgång till källkoden.

Några fördelar med single-tenant arkitekturer är:

- Maximal integritet: eftersom varje användare äger sin instans av tjänsten, finns det mindre risk för att en annan användare av misstag eller genom spionage får tag på uppgifter som inte tillhör dem.
- Användaren kan endast modifiera och förändra sin egen instans, vilket innebär att andra användares instanser inte kan påverkas.
- Arbeten som kräver mycket datorkraft kan utnyttja systemet till fullo, eftersom resurser inte behöver delas med andra användare.

Några nackdelar med single-tenant arkitekturer är:

- Single-tenant system är generellt dyrare än multi-tenant lösningar.
- Även om systemet inte körs på full kapacitet så måste användaren betala för full kapacitet, detta medför att denna typ av system kan bli kostnadsineffektiva.
- Användaren är ansvarig för att installera uppdateringar, detta kräver en större teknisk kunskap jämfört med om leverantören ansvarar för uppdateringarna. Detta försvårar även supportarbetet eftersom användare kan ha olika versioner, i multi-tenant arkitekturer finns bara en version att hålla reda på.

Generellt kan man säga att single-tenant arkitektur är motsatsen till multi-tenant arkitektur där en tjänst tjänar flera användare samtidigt. (Johnson, 2013; Khalil, 2013; Rouse, 2012; Smith, 2013)

2.3.4 Event Hub

Event Hub är en skalbar molntjänst för att hantera events (händelser) i stor skala i Microsoft Azure. Event Hubs kan hantera miljontals events per sekund och möjliggör bearbetning och analys på stora datamängder som produceras av anslutna enheter. Med hjälp av någon typ av realtidsanalystjänst kan data som samlats in via Event Hubs omvandlas och lagras. (Microsoft Azure, 2015c)

2.4 HDInsight/Storm

2.4.1 HDInsight

Apache Hadoop är ett open-source mjukvaruramverk för distribuerad lagring och bearbetning av mycket stora datamängder i datorkluster (Hadoop, 2015). Distribuerad bearbetning handlar om att skala ut istället för att skala upp. Att skala upp betyder att man uppgraderar en enskild maskin, detta leder till höga kostnader och låg tillgänglighet (om maskinen går ner stannar allt upp). Skala ut innebär att arbetsbördan fördelas på flera maskiner vilket sänker kostnaden och ger en miljö med högre tillgänglighet. Ramverket är utformat för att kunna skala ut ett arbete från enstaka maskiner till tusentals maskiner som var och en erbjuder lokal bearbetning och lagring. (Apache Hadoop, 2015; Rajesh, 2013)

HDInsight är Microsofts implementation av Apache Hadoop i molnplattformen Azure. HDInsight är 100 % kompatibelt med Apache Hadoop och är byggt på open-source komponenter i samarbete med Hortonworks som står bakom Apache Hadoop. (Sarkar, 2014) HDInsight är en PaaS (Platform as a Service) som ska möjliggöra för företag att utnyttja kraften i Apache Hadoop utan komplexiteten och kostnaden för installation, förvaltning och support. Att använda HDInsight har enligt Rajesh (2013) några fördelar:

- Man kan starta i liten skala och expandera efter behov.
- Datorkluster kan skapas på några minuter utan komplexa installationer och inställningar.
- Man får förvaltad och tillgänglig infrastruktur.
- Datorkluster kan stängas ner när de inte behövs.

HDInsight kan skapa, konfigurera, skicka och övervaka Hadoop-arbeten med hjälp av flera programmeringsspråk, som C#, Java, Python och Pig. Dessa tillägg används vid arbeten med historisk data i Apache Hadoop. Vid realtidsanalys används HDInsight tillsammans med Apache Storm, en open-source realtidsanalysplattform som kan bearbeta realtidshändelser i stor skala. (Microsoft Azure, 2015b)

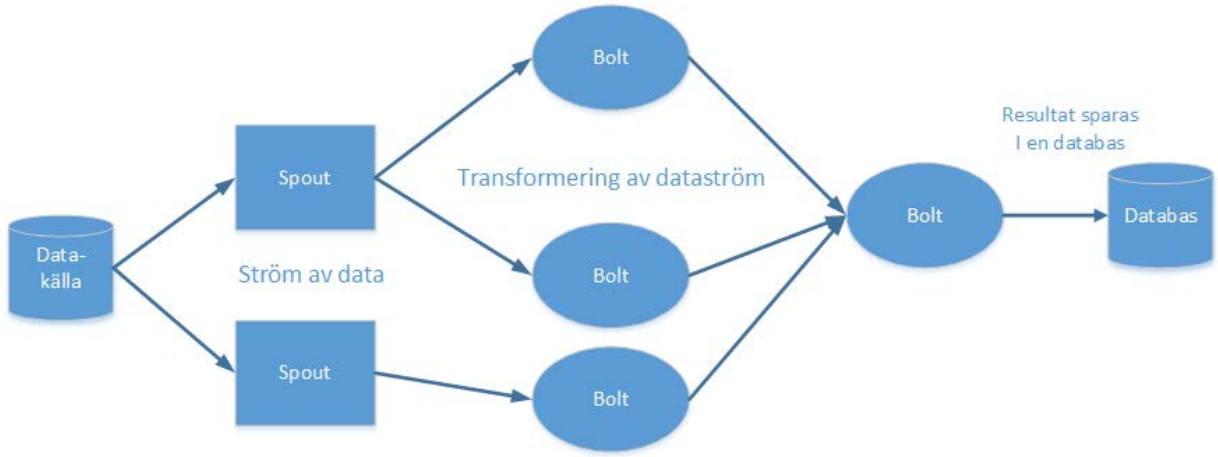
2.4.2 Apache Storm

Apache Storm är en open-source realtidsanalysplattform som gör att du kan bearbeta data i realtid med HDInsight. I fortsättningen kommer vi att benämna Apache Storm som endast Storm. Storm är skalbart, feltolerant och garanterar bearbetning av data genom att spela om data som inte framgångsrikt bearbetats första gången. Storm (som stand-alone) kan användas med vilket programmeringsspråk som helst, men som tjänst i Microsoft Azure finns det endast stöd för Java, C# och Python. (Apache, 2014; Franks, 2015c)

Storm körs på dedikerade Stormkluster i HDInsight, dessa kluster har en single-tenant arkitektur och kan använda flera olika typer av input- och outputströmmar. Som inputström kan Storm använda sig av bl.a. Event Hubs, Azure Service Bus och Apache Kafka. Som outputström kan Storm använda sig av bl.a. Event Hubs, Apache Cassandra och SQL-databaser. (Feddersen, 2015) HDInsight och Storm släpptes i GA som en gemensam tjänst i Azure 18 februari 2015 (Guthrie, 2015).

Storms topologi består av tre komponenter: *streams, spouts* och *bolts. Streams är* strömmar som kan beskrivas som en osorterad sekvens av nyckelvärdepar, *tuples*. Dessa nyckelvärdepar behandlas och transformeras till nya dataströmmar av komponenterna *spouts* och *bolts*. (Im et al., 2014)

Storm implementerar en dataflödesmodell där data strömmar genom ett nätverk av *spouts* och *bolts***.** En *stream* är flödet av data som kan beskrivas som en osorterad sekvens av *tuples*. En *spout* fungerar som en adapter som kopplar upp sig mot en datakälla, transformerar data till *tuples* och skickar ut dessa som en ström till *bolts. Bolts* är den del av Storms arkitektur som bearbetar data från en *spout* och kan utföras i flera steg. Exempel på detta kan ses i figur 2 nedan, där tre *bolts* utför tranformationer som senare behandlas i ytterligare en *bolt* som i sin tur sparar resultatet i en databas. Hädanefter kommer vi att referera till HDInsight och Storm som en kombinerad tjänst med förkortningen HDIS. (ibid.)



Figur 2 - Konceptmodell över Storms topologi (Mera, 2014)

2.5 Azure Stream Analytics

ASA är en molntjänst med en multi-tenant arkitektur som används för att analysera strömmande data i realtid (Feddersen, 2015). Enligt Sirosh (2015) på Microsoft så ska ASA göra det enklare att sätta upp realtidsanalyssystem som tar emot strömmande data från t.ex. sensorer, webbsidor, applikationer, infrastruktur-system samt alla typer av uppkopplade enheter. ASA använder sig inte av de traditionella programmeringsspråken som t.ex. Java och C#. Istället stödjer ASA ett SQL-liknande språk som enligt Sirosh (2015) drastiskt kommer att förenkla logiken för att analysera data i realtid. Att t.ex. beräkna medelvärde under en viss tidsperiod i realtid kan enligt Sirosh (2015) göras med ca fem rader SQL-kod i ASA jämfört med HDIS där det behövs flera hundra rader objektorienterad kod till samma uppgift. Det SQL-liknande språket i ASA är likt T-SQL som används för SQL Server, dock så har ASAs SQL-språk ett antal extra funktioner som t.ex. tidsbaserade fönster.

ASA är kompatibelt med flera av Microsoft Azures tjänster för dataintegrering samt datalagring: *Azure Event Hub, Azure Blob Storage* samt *Azure SQL Database*. ASA har ett grafiskt gränssnitt för att hantera kopplingar mot Azures dataintegrering och datalagringstjänster, i och med detta så behövs det inte skrivas någon kod för att sammankoppla ASA mot dessa tjänster. (Feddersen, 2015)

Möjligheten finns också att felsöka och testa SQL-frågan i webbläsaren innan den laddas upp och körs live i molnet. ASA släpptes i GA 16 april 2015. (Sirosh, 2015)

Resurser kan fördelas utifrån prestandabehov och användaren behöver endast betala för de resurser som används. Användaren ska kunna använda tjänsten i liten skala och vid behov öka prestandan och skala ut. ASA kan skalas från att analysera några kilobyte data per sekund till en gigabyte per sekund. Integreringen med Azure Event Hubs möjliggör att miljontals event per sekund kan tas emot av ASA och analyseras. ASA skalas automatisk beroende på hur många event som tas emot samt hur mycket datorkraft som behövs för att bearbeta SQL-frågan. Kostnaden för tjänsten baseras på mängden event som bearbetas samt datorkraften som behövs för att utföra analyserna. (Stokes, 2015)

Funktionalitet finns för att analysera strömmande data i kombination med historisk data s.k. referensdata eller data som förändras långsamt. Referensdata kan t.ex. användas för att hitta mönster i strömmande data. (ibid.)

2.6 Användbarhet

Användbarhet är ett samlat kvalitetsmått för en produkt (eller tjänst), och kan därför vara hög eller låg. Användbarhet definieras enligt ISO-normen 9241-11 som följande:
Den grad i vilken användare i ett givet sammanhang kan bruka en produkt för att uppnå specifika mål på ett ändamålsenligt, effektivt och för användaren tillfredsställande sätt (användarnöjdhet). (Wikipedia, 2015b)

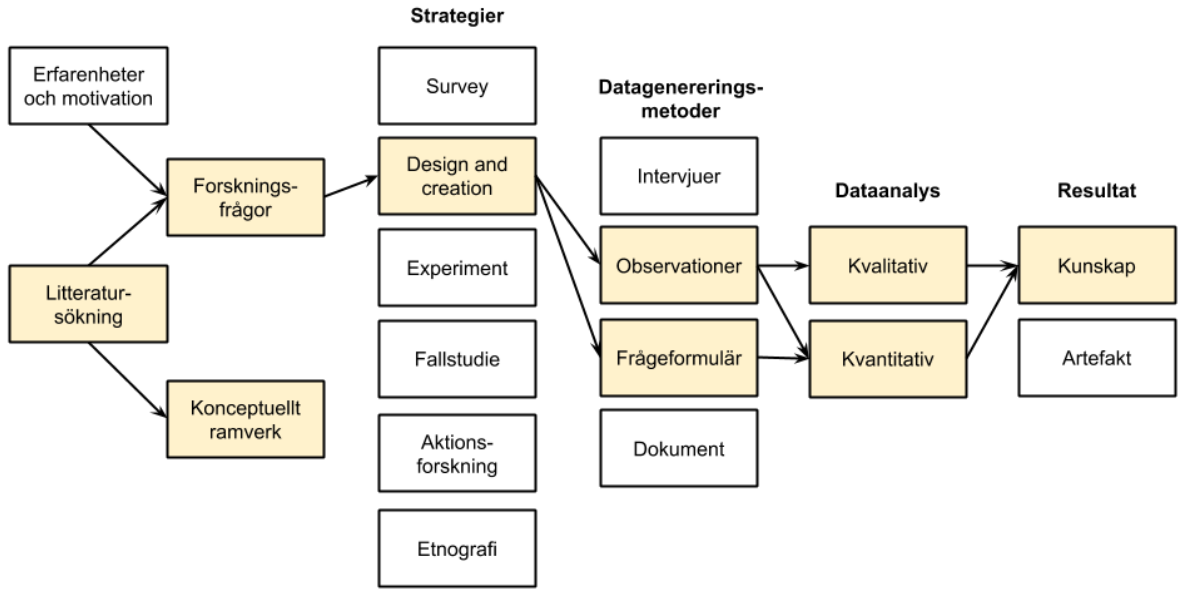
ISO-normen innehåller inga specifika metoder för att kunna utvärdera användbarhet utan endast generella principer som beskrivs nedan.

- *Ändamålsenlighet* anger den träffsäkerhet och fullständighet med vilken användarna uppnår specificerade mål.
- *Effektivitet* anger de resurser som förbrukas för att uppnå specifika mål.
- *Användarnöjdhet* anger den frihet från obehag, samt positiva attityder till användningen av produkten som användare upplever.

Det givna sammanhanget innefattar användare, uppgifter, utrusning (hårdvara, mjukvara och material) samt den fysiska och sociala miljö som en produkt används inom, alla dessa påverkar användbarheten av en produkt. (ISO OBP, 1998)

3 Metod

I metodkapitlet beskriver vi tillvägagångssättet i studien. Figur 3 nedan beskriver forskningsprocessen i studien och vilka steg vi valt att ta. Dessa steg beskrivs mer ingående i detta kapitel.



Figur 3 - Anpassad modell av Oates (2006) forskningsprocess

3.1 Litteratursökning

Steg 1

Eftersom våra baskunskaper i ämnet realtidsanalys var väldigt begränsade så började vi söka litteratur som kunde öka vår förståelse i ämnet. I denna inledande sökningsfas använde vi oss av sökord som: Azure Stream Analytics, realtidsanalys, molntjänst, real-time analytics, cloud computing, Hadoop, big data, Internet of Things, IoT. I tabell 1 kan vi se de koncept vi använt oss av och varianter på sökord för varje koncept. (Oates, 2006)

Tabell 1 - Koncept steg 1

Koncept 1 (K1)	Koncept 2 (K2)	Koncept 3 (K3)	Koncept 4 (K4)	Koncept 5 (K5)	Koncept 6 (K6)	Koncept 7 (K7)
Azure Stream Analytics	realtidsanalys	molntjänst	big data	Hadoop	Internet of Things	Azure
	real-time analytics	cloud computing		map-reduce	IoT	
	real-time analysis			Apache hadoop		
	real-time					

Det första steget i litteratursökningen utfördes på de vetenskapliga databaserna IEEE och Google Scholar. Det har i dessa databaser varit svårt att hitta litteratur som behandlar realtidsanalys som molntjänst. Som vi kan se i tabell 2 har vi endast hittat en artikel som behandlar realtidsanalys som molntjänst. Vi kan också se i tabell 2 att vi helt saknar sökträffar på koncept 1 som är Azure Stream Analytics. Under detta första steg i vår litteraturstudie så hittade vi artiklarna *Real-Time Big Data Analytics: Emerging Architecture* och *Towards Real-Time Analytics in the Cloud* som behandlade andra realtidsanalyssystem som t.ex. Apache Storm. I artikeln *Microsoft Azure Essentials: Fundamentals of Azure* fann vi att HDInsight var Microsoft Azures implementation av Apache Hadoop.

Litteratur som har använts i rapporten från tabell 2 är ett konferenspapper om realtidsanalys i molnet av Osman et al. (2013) *"Towards Real-Time Analytics in the Cloud".* Majoriteten av det material som vi funnit behandlar dock till största del äldre system och lösningar som vi har valt att inte fokusera på.

Tabell 2 - Resultat av litteratursökning från steg 1

Publikation	**Typ av publikation**	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**
Towards Real-Time Analytics in the Cloud	Konferenspapper		X	X	X	X	
BUSINESS INTELLIGENCE AND ANALYTICS: FROM BIG DATA TO BIG IMPACT	Artikel i vetenskaplig tidskrift				X		X
Towards Efficient Big Data and Data Analytics: A Review	Konferenspapper				X	X	
Big Data: Issues, Challenges, Tools and Good Practices	Konferenspapper			X	X	X	
Big Data: A Review	Konferenspapper				X	X	
Real-Time Big Data Analytics: Emerging Architecture	Bok		X		X	X	
Microsoft Azure Essentials: Fundamentals of Azure	Bok			X			

Steg 2

Som vi kan se i tabell 3 så har vi i ett andra steg i litteratursökningen bytt ut och ändrat några av koncepten från tidigare tabell 1. Vi har frångått big data och istället fokuserat på strömmande data då detta kändes mer relevant för vårt arbete. Storleken på datamängden ansåg vi inte var relevant för vår undersökning utan det viktiga var att data kunde analyseras i realtid, därför har vi i tabell 3 istället strömmande data som koncept 4. Vi har även frångått Hadoop som koncept och istället fokuserat på HDInsight eftersom det är en implementation av Apache Hadoop i molnet. Vi kompletterade även vår sökning från tidigare steg med ytterligare ett koncept: Apache Storm.

Tabell 3 - Koncept steg 2

Koncept 1 (K1)	Koncept 2 (K2)	Koncept 3 (K3)	Koncept 4 (K4)	Koncept 5 (K5)	Koncept 6 (K6)
Azure Stream Analytics	realtidsanalys	molntjänst	strömmande data	HDInsight	Apache Storm
	real-time analytics	cloud computing	streaming data		Storm
	real-time analysis	cloud	data stream		
	real-time		realtidsdata		
			real-time data		

Vi kompletterade tidigare sökresultat från steg 1 genom att använda sökmotorn Google i kombination med IEEE och Google Scholar samt koncepten vi kan se i tabell 3. Detta för att kunna hitta lite färskare information som mer direkt behandlar vårt ämne. Det vi funnit och använt oss av för att få fram ett konceptuellt ramverk kommer från konferenspapper, bloggar, online artiklar, böcker och white papers. De bloggar, white papers, böcker och online artiklar vi hittat har vi synat extra noggrant för att se om källan är trovärdig. Vid utvärdering av bloggar och online artiklar har vi dels synat den som skrivit bloggen eller artikeln, och även sidan den är publicerad på. De white papers vi har hittat har vi varit noggranna med att särskilja trovärdig fakta från sådant som kan anses som reklam för företaget som står som utgivare. Konferenspapper och böcker har vi utvärderat genom att syna utgivare och författare. (Oates, 2006)

I tabell 4 ser vi de centrala publikationerna som har använts i rapporten. I vår sökning fann vi bl.a. ett konferenspapper av Im et al. (2014) "*Detecting a large number of objects in real-time using apache storm"* som vi använt för att få ökad förståelse för Apache Storm. Boken "*HDInsight Essentials*" av Rajesh (2013) har vi använt för att få ökad förståelse om HDInsight. För att få motsvarande kunskaper om Azure Stream Analytics använde vi b.la. ett white paper från Microsoft av Feddersen (2015) "*Real-Time Event Processing with Microsoft Azure Stream Analytics"* och en artikel av Stokes (2015) "*Introduction to Azure Stream Analytics*".

Tabell 4 - Resultat av litteratursökning från steg 2

Publikation	Typ av publikation	K1	K2	K3	K4	K5	K6
Detecting a large number of objects in real-time using apache storm	Konferenspapper		X		X		X
Real-Time Event Processing with Microsoft Azure Stream Analytics	White paper	X	X	X	X	X	X
HDInsight Essentials	Bok			X		X	
Announcing the General Availability of Azure Stream Analytics	Blogginlägg	X	X	X	X		
Introduction to Azure Stream Analytics	Artikel online	X	X	X	X		

Steg 3

Innan vi hittade det slutliga ämnet utförde vi nya sökningar med hjälp av kunskap från tidigare funnen litteratur. Efter att vi studerat publikationerna från steg 2 så upptäckte vi att Apache Storm tillsammans med HDInsight kunde utföra realtidsanalys i molnet. Vi förstod också att realtidsanalystjänsterna Azure Stream Analytics och HDInsight/Storm baserades på olika arkitekturer. Vi utförde en ny sökning med koncepten: multi-tenant, single-tenant enligt tabell 5.

Tabell 5 - Koncept steg 3

Koncept 1 (K1)	Koncept 2 (K2)
multi-tenant	single-tenant
multi-tenancy	single-tenancy
multi tenant architecture	single-tenant architecture
	dedicated

I tabell 6 ser vi de centrala publikationerna som har använts för att skapa ett konceptuellt ramverk kring arkitekturerna multi-tenant och single-tenant. Publikationen "*A multi-tenant architecture for business process executions*" hjälpte oss att förstå vad multi-tenant arkitektur innebär. Publikationerna "*Cloud computing architectures based multi-tenant IDS*" och "*Architectural Concerns in Multi-Tenant SaaS Applications*" använde vi i rapporten för att bygga upp teorin kring multi-tenant arkitektur. För att bygga upp teorin kring single-tenant arkitektur använde vi oss av publikationerna "*In the Cloud: Multi-Tenant vs. Single-Tenant ITSM"* och *"Single-Tenant vs Multi-Tenant Cloud ERP".*

Tabell 6 - Resultat av litteratursökning från steg 3

Publikation	Typ av publikation	K1	K2
A multi-tenant architecture for business process executions	Konferenspapper	X	
Cloud computing architectures based multi-tenant IDS.	Konferenspapper	X	X
Architectural Concerns in Multi-Tenant SaaS Applications	Konferenspapper	X	
In the Cloud: Multi-Tenant vs. Single Tenant ITSM	Artikel online	X	X
Single-Tenant vs Multi-Tenant Cloud ERP	Artikel online	X	X

3.2 Forskningsstrategi

Forskningsstrategin design and creation fokuserar på utveckling av IT-artefakter. Enligt March och Smith (1995) finns det fyra olika typer av artefakter:

- Konstruktioner: En samling begrepp som används i en IT-relaterad domän.
- Modeller: En kombination av konstruktioner som representerar en situation och används för att underlätta problemförståelse och utveckling av lösningar.
- Metoder: En serie steg som används för att utföra en uppgift.
- Instansieringar: En realisation av en artefakt i sin naturliga miljö som demonstrerar genomförbarheten och effektiviteten av artefakten.

Eftersom vi implementerar artefakterna i Azure vilket är analystjänsternas naturliga miljö, och sedan använder dem för att utvärdera användbarhet anser vi att de är av typen instansiering.

Design and creation är den förväntade forskningsstrategin inom datavetenskap. Enligt Oates (2006) är IT relativt nytt inom många områden och i ständig utveckling. Som en följd av den snabba utvecklingen så finns många områden där utveckling av nya IT-artefakter kan bidra till ny kunskap. Molntjänster för att analysera realtidsdata med en multi-tenant arkitektur som t.ex. ASA är ett nytt område. Därför anser vi att det är högst intressant och aktuellt att jämföra ASA med en annan analystjänst som t.ex. HDIS som har en single-tenant arkitektur.

Enligt Oates (2006) så kan den snabba utvecklingen ibland också vara en nackdel då det kan innebära att forskningen riskerar att bli inaktuell innan forskningsresultaten publicerats. Eftersom ASA släpptes i GA 16:e april 2015 så ser vi ingen risk att tjänsten blir inaktuell inom en snar framtid och således inte heller våra forskningsresultat. Tvärtom så tror vi att det kommer att dyka upp fler tjänster med multi-tenant arkitektur som ASA. Apache Storm har funnits en längre tid och som tjänst i Azure tillsammans med HDInsight har den funnits i GA sedan 28:e feb 2015.

Enligt Oates (2006) kan det vara riskabelt att arbeta med design and creation om forskarna inte besitter tillräckliga tekniska kunskaper. Under litteraturstudierna så läste vi om både ASA och HDIS och hur man skulle gå till väga för att kunna utveckla analyslogiken (artefakterna) till dessa tjänster. Vi var noggranna med att inte läsa för mycket om någon av tjänsterna för att undvika att testresultaten skulle bli missvisande. Därför valde vi att bara studera hur tjänsterna var uppbyggda logiskt och studerade inte några kodexempel på djupet. Det som framkom under litteraturstudien var att ASA använde sig av en SQL-liknande syntax och HDIS använde b.la. programmeringsspråket C# under utvecklingsarbetet. Vi ansåg att vi i egenskap av studenter på Systemvetenskapliga programmet hade tillräckliga kunskaper inom både C# och SQL för att kunna utföra studien på ett tillfredställande sätt. Vi anser även att våra kunskaper inom de båda programmeringsspråken är likvärdiga varandra, därför anser vi också att ingen av analystjänsterna hade någon fördel innan testerna utfördes.

I planeringen av design and creation måste man enligt Oates (2006) ta ställning till några saker: *systemutvecklingsmetodik, varför design and creation-projektet är forskning, utvärdering, användning av datagenereringsmetoder.*

3.2.1 Systemutvecklingsmetodik

I en forskningsrapport måste man enligt Oates (2006) förklara och dokumentera hur man arbetade sig igenom analys, design, implementation och testning. Man kan använda sig av någon existerande systemutvecklingsmetod eller utveckla en egen som är unik för forskningsarbetet. Det viktigaste är att läsaren kan följa hur man arbetat sig igenom

utvecklingen av artefakten. Vi har använt oss av de grundläggande stegen i systemutvecklingsmetodik som Oates beskriver: analys, design, implementation och testning. Den första systemutvecklingsmetoden (förutsättningar) beskriver hur vi har gått till väga för att skapa förutsättningarna för våra tester. Den andra (analystjänster) beskriver hur vi har arbetat oss igenom utveckling av analyslogiken i både ASA och HDIS.

3.2.1.1 Systemutvecklingsmetod förutsättningar

Steg 1 analys

Vi började med att fundera på vilken typ av strömmande data som vore lämplig att jobba med. Vi kom fram till att lämpliga data var:

- Data som strömmas kontinuerligt
- Uppdateras ofta, oftare än var 10:e sekund.
- Tillgänglig 24h/dygn
- Data som ständigt förändras (t.ex. vore en utomhus temperaturmätare inte lämplig eftersom temperaturer inte förändras särskilt ofta)

Vi kontaktade Trafikverket för att ta reda på om de hade någon öppen dataström som vi kunde använda oss av. Det visade sig att de data som fanns tillgänglig för oss inte var tillräckligt kontinuerlig då de endast uppdaterades var 30:e minut. Eftersom vi inte hittat någon lämplig existerande ström av data att jobba med, så bestämde vi oss för att utveckla en applikation som genererar simulerad data. Vi diskuterade kring vad som vore ett lämpligt och realistiskt scenario att utgå ifrån. Vi kom fram till att vi skulle simulera en trafikkamera som står utefter en väg och registrerar bilar som kör förbi kameran.

Steg 2 design

Vi valde att utveckla våra applikationer med programmeringsspråket C# och Visual Studio som utvecklingsverktyg eftersom analystjänsterna som vi använder i våra PoC är Microsoftprodukter. I Litteraturstudien framkom det att båda analystjänsterna kunde koppla upp sig mot Event Hubs för att ta emot och skicka strömmande data. Därför beslutade vi att använda Event Hubs för att hantera input- och outputströmmarna i våra PoC. För att få förståelse för hur man använder sig av Event Hubs så följde vi instruktioner av Damaggio (2015). Instruktionerna innehöll två konsolapplikationer, en inputapplikation som genererade slumpade tecken och skickade dessa som events till Event Huben samt en outputapplikation som hämtade eventen och presenterade innehållet i konsolfönstret.

Steg 3 implementation

Vi följde tidigare nämnda instruktionsguide och utvecklade applikationerna och testar dessa. Efter att testningen blev lyckad gick vi tillbaka till designfasen och gjorde om inputapplikationen så att den skulle passa vårt arbete. Vi skapade ett grafiskt gränssnitt där vi kunde styra flödet av data för att simulera att något hände på vägen som kameran är placerad vid.

Steg 4 testning

Vi testade den nya applikationen med det grafiska gränssnittet och när vi konstaterat att den fungerade kunde vi gå vidare med att utveckla analystjänsterna. Design, implementation och testfasen arbetade vi med iterativt när vi upptäckte att förbättringar behövdes.

3.2.1.2 Systemutvecklingsmetod analystjänster

Steg 1 analys
Vi inleder varje utvecklingsprocess med att läsa ett delmål och identifiera problemet som ska lösas. *Exempel på delmål: Registrera samtliga bilar som kör för fort oavsett väg.*

Steg 2 design
Baserat på de funktioner och/eller variabler vi identifierade i analysfasen diskuterar vi här logiken som krävdes för att lösa problemet. I ASA består logiken (artefakten) av en SQL-fråga och eventuell definition av inputs och outputs. I HDIS består logiken (artefakten) av C#-kod och eventuella databasuppkopplingar.

Steg 3 realisering och Implementering
I det här steget realiseras logiken som framkommit i designfasen kodmässigt. Eventuella databaser som behövdes för att lösa uppgiften skapades. Sedan laddas koden upp i respektive analystjänst och körs. Det vi ändrar på i respektive tjänst i det här steget kallar vi för analyslogik, vilket är detsamma som våra artefakter.

Steg 4 testning
I detta steg kontrollerades om vi kunde se förväntat resultat i outputapplikationen. Om inte förväntat resultat visades i outputapplikationen så upprepades utvecklingsprocessen från designfasen.

3.2.2 Design and creation som forskning

Enligt Oates (2006) så måste man motivera varför arbetet är design and creation-forskning och inte vanligt systemutvecklingsarbete. En artefakt kan ha en av tre roller: IT-artefakten i sig själv kan vara kunskapsbidraget, den kan vara ett hjälpmedel för att undersöka något eller en verklig slutprodukt där fokus ligger på utvecklingsprocessen. Exempelvis kan en artefakt som hjälpmedel innebära att en IT-applikation utvecklas med två olika mjukvaruprogram för att jämföra och utvärdera de olika programmen. I ett sådant fall är jämförelsen och utvärderingen kunskapsbidraget. (Oates, 2006)

Eftersom vi använde design and creation för att ta fram artefakter som används som hjälpmedel och inte är slutprodukter, så anser vi att kunskapsbidraget är av sådan natur att arbetet kan ses som vetenskapligt. Vårt kunskapsbidrag kommer att vara en utvärdering av användbarhet hos två analystjänster med skilda arkitekturer. Denna utvärdering är av intresse för alla som har eller kommer att få behov av att analysera realtidsdata.

3.2.3 Utvärdering

När en artefakt har utvecklats måste den enligt Oates (2006) utvärderas. Några av kriterierna som Oates (2006) nämner som kan användas för att utvärdera en artefakt är: funktionalitet, prestanda, pålitlighet och användbarhet. Vi har valt att utvärdera IT-artefakterna enligt användbarhetskriterierna *effektivitet*, *ändamålsenlighet* och *användarnöjdhet*.

Måttet på effektivitet och produktivitet är tiden det tar för en användare att utföra en uppgift, andelen slutförda uppgifter är en central del för att mäta ändamålsenlighet (Sauro, 2011). Mätning av subjektivitet i användbarhet görs oftast genom frågeformulär med attitydskalor, detta ger ett mätbart resultat av en användares subjektiva upplevelse (Brooke, 1996). Vår utvärdering kommer att leda till att vi kan dra slutsatser angående användbarhet i utvecklingen av analyslogiken i analystjänsterna som vi undersöker.

Om man inte behöver utvärdera artefakten i en verklig kontext kan man ha som mål att visa på Proof of Concept (PoC) (Oates, 2006). Vi använde oss av simulerad data för att utvärdera användbarhet genom att utföra våra scenarier och delmål i respektive analystjänst.

3.2.4 Användning av datainsamlingsmetoder

Enligt Oates (2006) är det inom forskningsstrategin design and creation vanligt förekommande att man använder sig av datainsamlingsmetoderna: *intervjuer, observationer, frågeformulär* samt *dokument.* Observationer kan man använda för att se hur människor arbetar (Oates, 2006). Vi kommer att använda oss av två typer av observationer, deltagande observationer använder vi för att se hur vi arbetat med utvecklingen av analystjänsterna samt systematisk observation för att kunna mäta effektivitet och ändamålsenlighet. Vi har använt oss av frågeformulär för att ta reda på vad användare tyckte om de utvecklade artefakterna.

3.3 Beskrivning av Proof of Concept

I detta kapitel beskriver vi de olika delarna i våra PoC prototyper och hur delarna hänger ihop med varandra. Figur 4 och figur 5 nedan visar dataflödet i våra PoC prototyper. I punktlistan nedan ger vi en kort beskrivning på hur data strömmar mellan de olika delarna.

- *Kamerasimulator* är den applikation som vi har utvecklat för att simulera en dataström, applikationen representerar ett flertal trafikkameror som är placerade vid olika vägar och registrera alla bilar som passerar. Dataströmmen som genereras innehåller data om registreringsnummer, hastighet, tidpunkt samt vägnummer.
- Dataströmmen skickas vidare till *Input Event Hub* som är en molntjänst för tillfällig lagring.
- Från Input Event Hub så skickas dataströmmen vidare till analystjänsten *ASA* i vår PoC 1 och till *HDIS* i PoC 2. Det är i detta steg som analysen av dataströmmen utförs. Det är också i detta steg som vi har utvecklat våra artefakter.
- När vi har utfört analysen och sorterat ut de data som var intressant så skickas dataströmmen vidare till *Output Event Hub.* Därefter skickas dataströmmen till *outputapplikationen* där vi kan se resultatet från vår analys i realtid.

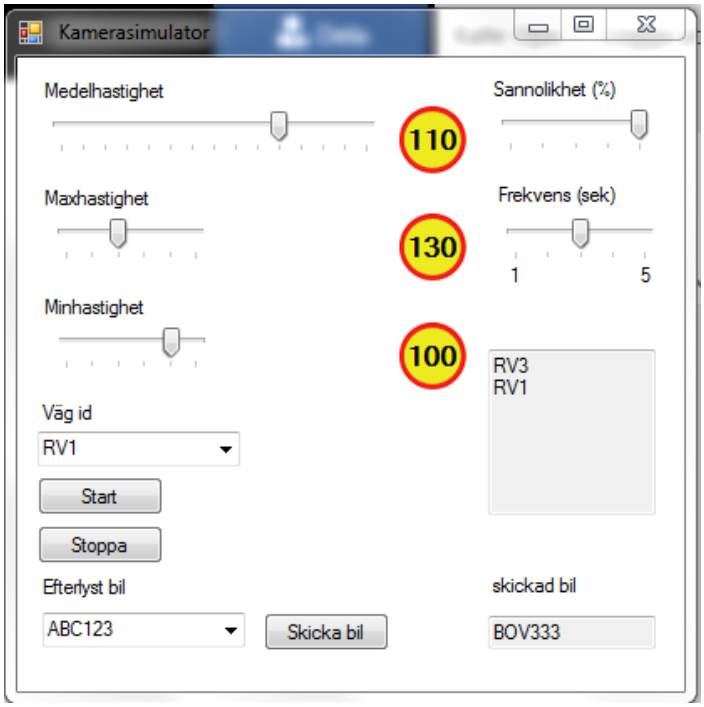
Figur 4 - PoC 1

Figur 5 - PoC 2

Kamerasimulator

Applikationen representerar en trafikkamera som är placerad utefter en fyrfilig väg där det kontinuerligt passerar bilar. Med kontinuerligt menar vi 1-4 bilar med ett tidsintervall på 1-5 sekunder, tidsintervallet är justerbart. Varje bil har ett registreringsnummer som kameran läser av, kameran lägger också till tidpunkt, hastighet på varje bil samt vägnummer. Vi kommer att simulera flera trafikkameror på flera olika vägar samtidigt för att öka datamängden.

När en bil passerar en kamera så skapas ett event som innehåller registreringsnummer, tid, bilens hastighet och vägnummer. Maxhastighet för varje kamera finns lagrad i en SQL-databas och det gör även en lista på efterlysta bilar som vi kommer att använda i ett scenario. I figur 6 ser vi det grafiska interfacet för kamerasimulatorn.



Figur 6 - Kamerasimulatorns grafiska interface

Eftersom trafikförhållandena på en väg i verkligheten ständigt förändras ville vi kunna hantera dessa förhållanden i realtid i vår applikation. Vi kan starta upp till fem kameror med unika väg-ID samtidigt. För varje instans av kamerasimulatorn kan man ändra följande parametrar:

- Medelhastighet för samtliga bilar som färdas på vägen.
- Maxhastighet anger hur fort bilarna kan köra.
- Minhastighet anger den lägsta hastigheten för bilarna.
- Sannolikhet anger hur många procent av bilarna som avviker från medelhastigheten.
- Frekvens hur ofta bilarna passerar kameran.
- Knappen skicka bil, skickar vald efterlyst bil förbi kameran.

Exempel på hur ett event som skickas kan se ut i JSON format:


```
{
   "CarList": [ { "Regnr": "VBX211", "Speed": 110 }, { "Regnr": "NEB436", "Speed": 115 }, {
"Regnr": "FMG198", "Speed": 105 }, { "Regnr": "JBB534", "Speed": 110 } ],
   "Time": "2015-05-11 16:37:20",
   "RoadId": "RV1"
}
```

Input Event Hub
Molntjänst från Microsoft som vi använder för att ta emot och tillfälligt lagra data från vår kamerasimulator. I vårt fall lagras data i Event Hub max 24 timmar. Input Event Hub sänder all data vidare till respektive analystjänst. Båda Event Hubs skapades i Azureportalen. För att kunna skapa båda Event Hubs, kamerasimulatorn och outputapplikationen följde vi instruktioner av Damaggio (2015).

Analystjänsterna
De båda analystjänsterna skapar en koppling till Event Hub (input) och tar emot all data som Event Hub levererar som en ström. Sedan sker en analys av dataströmmen och de data som är intressant för ett visst scenario plockas ut och skickas vidare till Output Event Hub.

Azure Stream Analytics
Vi skapade ett ASA-arbete via Azureportalen som vi sedan modifierade för att kunna lösa varje delmål. I ASA-arbetet började vi med att definiera input- och outputströmmarna, d.v.s. Input Event Hub och Output Event Hub. I modifieringen ingår utveckling av SQL-fråga och definiering av input och output vi använde utöver våra Event Hubs. För att utföra en analys av dataströmmen så ska en SQL-fråga utvecklas, SQL-frågan används för att sortera ut önskad data. För att testa att Event Hub-strömmarna fungerade började vi med att ställa en SELECT * fråga för att visa all data. (Evans, 2015) Exempel på hur en SQL-fråga ser ut finns i bilaga 3, där visar vi hur SQL-syntaxen ser ut i Scenario 5 delmål D.

HDInsight/Storm
Vi använde oss av en serie instruktioner som vi hittade i Azures dokumentation. Det var en introduktion till Storm i HDInsight som i flera steg beskrev hur man utvecklar Stormtopologier i C# samt laddar upp och startar dessa i Stormklustret i HDInsight.
(Franks, 2015)

Vi började med att skapa ett Stormkluster i HDInsight via Azureportalen. I portalen kan man själv välja klusterstorlek, vi valde en klusterstorlek på en (1) nod. Detta var rekommendationen när klustret skulle användas i testsyfte för att minimera kostnaderna. (Franks, 2015) Eftersom ett Stormkluster inte kan stoppas när det inte används, så tar vi bort klustret när vi är färdiga med det och skapar ett nytt när vi behöver det igen.

Vi följde Franks (2015) instruktion och skapade två Stormapplikationer i Visual Studio, en som tar emot data från en Event Hub och en som skriver till den. Eftersom vår Stormapplikation måste kunna kommunicera med två olika Event Hubs (se figur 5) så skapade vi en applikation som både kan läsa och skriva till två olika Event Hubs.

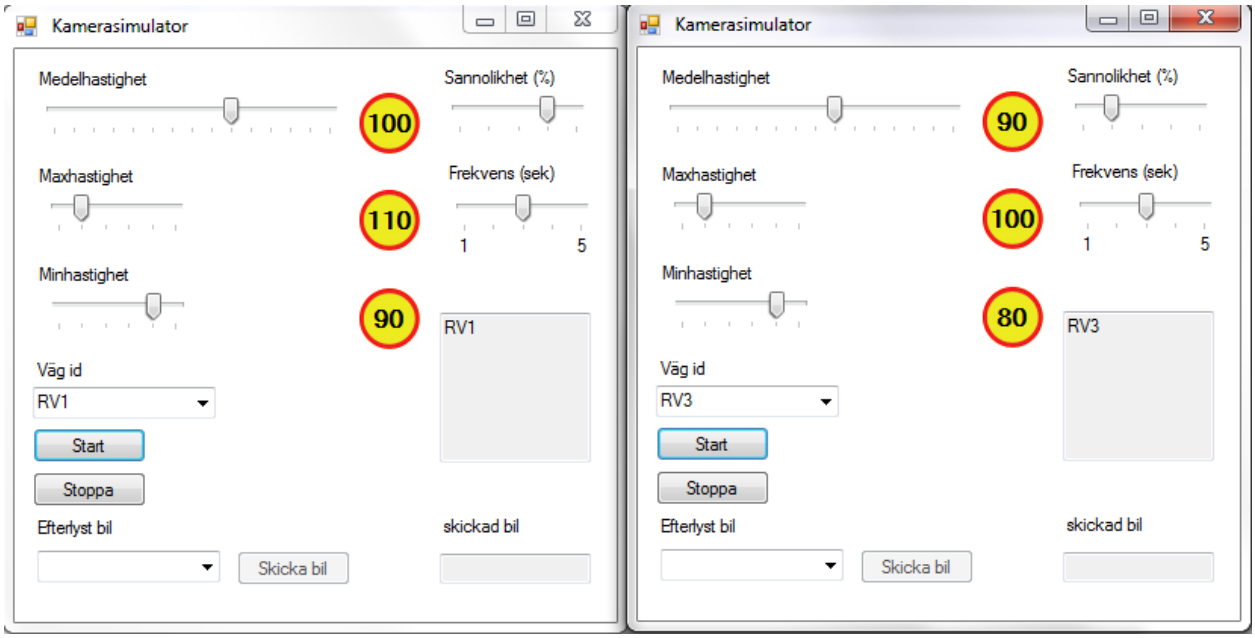
Först såg vi till att ha en fungerande topologi med en spout som sköter kopplingen till input Event Hub, samt två bolts, en för analyslogiken och en för kopplingen mot output Event Hub. För att kunna lösa varje delmål så ändrade vi nu bara i bolten för analyslogik, uppkopplingen mot Event Hubs var likadan under hela arbetet.

Exempel på C#-kod finns i bilaga 4, där visar vi hur koden i bolten för analyslogiken ser ut i Scenario 5 delmål D.

Output Event Hub
Användes för att ta emot och tillfälligt lagra data från våra analystjänster. I vårt fall lagras data i Event Hub max 24 timmar. Output Event Hub sänder all data vidare till outputapplikationen.

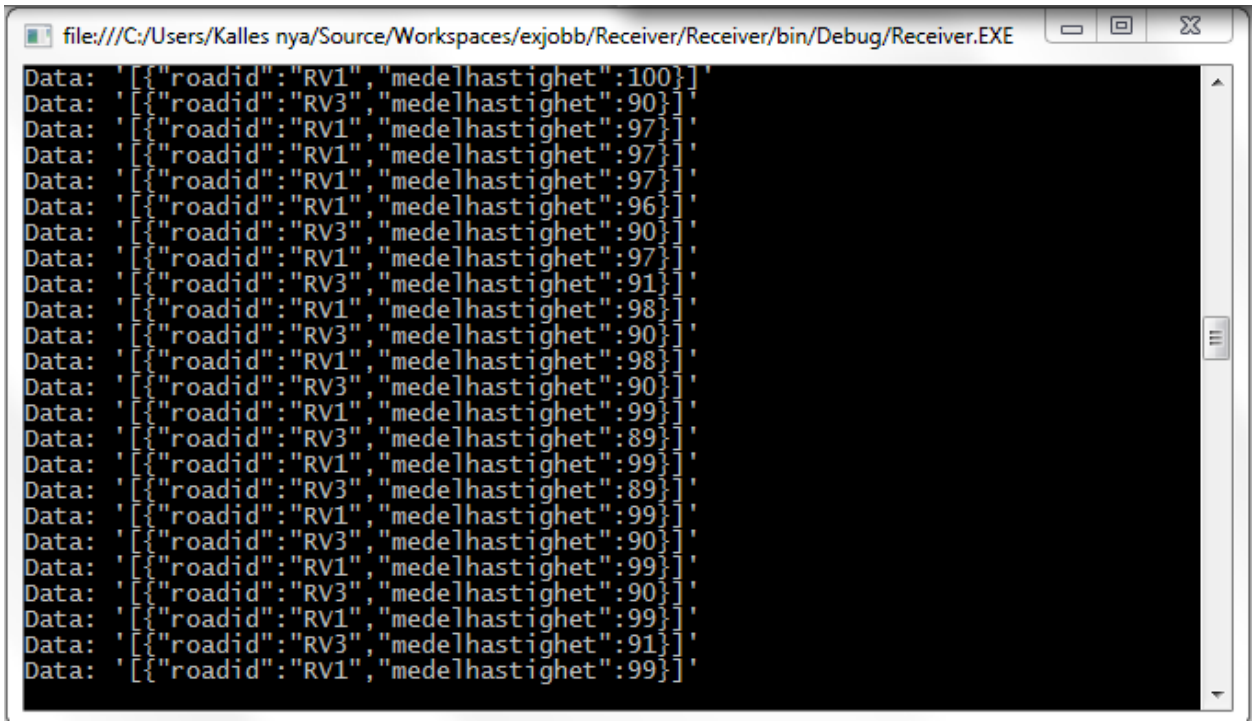
Outputapplikation

Outputapplikationen är en konsolapplikation som tar emot data från output Event Hub och visar resultatet i ett konsolfönster. I figur 7 kan vi se hur två instanser av kamerasimulatorn körs med olika vägar och medelhastighet. Max- och minhastighet anger hur fort och hur sakta bilarna i varje instans kan åka. Sannolikhet anger hur sannolikt det är att varje bil avviker från medelhastighet.



Figur 7 - Två instanser av kamerasimulatorn

I konsolapplikationen (figur 8) kan vi se resultatet som i det här fallet är scenario 4 delmål D (Beräkna medelhastighet på varje väg inom den senaste minuten).



Figur 8 - Outputapplikationen

3.4 Datainsamling

Vår primära datainsamling kommer att ske via observationer av PoC prototyperna vi ska utveckla. Vi kommer att utföra deltagande observationer på det som utförs i utvecklingen och göra kvalitativa fältnoteringar. För att komplettera dessa kvalitativa data kommer vi att samla in kvantitativ data från våra PoC prototyper via systematiska observationer baserat på användbarhetskriterierna effektivitet, ändamålsenlighet och användarnöjdhet.

3.4.1 Deltagande observationer

Enligt Oates (2006) innebär en deltagande observation att forskaren själv deltar i situationen som observeras. I vårt fall har vi observerat oss själva i utvecklingen av analystjänsterna och noterat så mycket som möjligt om det som hände. Vi använde oss av anteckningsverktyget OneNote där vi noterade händelser och våra upplevelser i utvecklingen av analystjänsterna, detta ger enligt Oates (2006) en rik beskrivning av det som observeras. För att göra noteringarna tydliga så strukturerade vi upp dem, först i respektive analystjänst, sedan delades dessa in i respektive scenario och till sist i varje delmål.

Processen i deltagande observationer är i regel väldigt tidskrävande, men desto längre tid som spenderas i en situation, desto mer är det troligt att forskaren kan lära sig. Vi startade observationerna förutsättningslöst, utan att definiera exakt vad som ska observeras och vad man ska fokusera på att notera. På så sätt kunde vi få en känsla av vad som faktiskt var intressant att observera, och i senare stadier fokusera på sådant som var intressant. (Oates, 2006)

3.4.2 Systematisk observation

En systematisk observation är enligt Oates (2006) när forskaren i förväg bestämmer vilka typer av händelser som skall observeras. Dessa händelser är fördefinierade i ett schema eller tabell som forskaren använder för att anteckna frekvens och varaktighet för den aktuella händelsen, denna typ av observation genererar vanligtvis kvantitativ data.

I den systematiska observationen har vi mätt utefter användbarhetskriterierna *effektivitet* och *ändamålsenlighet* vilket har genererat kvantitativ data. Vi har använt dessa kvantitativa data för att stödja den kvalitativa (Sauro, 2004). I vårt fall har vi i förväg definierat ett antal scenarier samt tillhörande delmål (se kapitel 3.4.6) som vi sedan har strukturerat upp i ett Excelark. Vi har i Excelarket noterat om delmålen blivit uppfyllda eller ej samt hur lång tid det tog att uppfylla varje delmål. De data som samlats in för delmålen används sedan för att beräkna effektivitet och ändamålsenlighet kopplat till scenariot. Enligt Oates (2006) så skall händelserna som skall mätas vara tydligt definierade och kategoriserade och inte överlappa varandra. Detta för att det ska vara enkelt för forskaren att notera när en händelse har inträffat samt när händelsen påbörjades och avslutades. I vårt arbete har vi kategoriserat i form av scenarier med tillhörande delmål. Scenarierna har ingen koppling till varandra dock så är flera av våra delmål en förutsättning för att nästa delmål ska kunna uppfyllas. Vi har observerat varje delmål separat, enligt Oates (2006) är det lättare att observera separata isolerade händelser istället för att mäta flera händelser som inträffar samtidigt.

3.4.3 Effektivitet

Enligt Sauro (2011) så är måttet på effektivitet och produktivitet tiden det tar för en användare att utföra en uppgift. Han säger att tiden för en uppgift bör starta när användaren har läst uppgiften och avsluta tiden när användaren har avslutat alla åtgärder. Enligt Sauro (2011) så finns det i huvudsak tre sätt att rapportera effektivitet:

1. Tiden att slutföra en uppgift: inkluderar endast tider där uppgiften blivit slutförd.
2. Tiden att misslyckas med en uppgift: Tiden som en användare lägger på en uppgift innan användaren ger upp eller slutför den felaktigt.
3. Tiden spenderad på en uppgift: Den totala tid en användare spenderar på en uppgift.

Vi har valt att mäta våra scenarier enligt punkt 1 och punkt 2. Vi anser att punkt 3 är för diffus för att kunna generera ett användbart mätresultat i vårt fall. Det finns enligt Sauro (2011) flera sätt att definiera maxtid för en uppgift, man kan använda sig av:

- Tider från liknade produkter, eller tjänster i vårt fall
- Tider från tidigare versioner av tjänsten
- Tider från ett test utan att använda sig av en dator

Enligt Sauro & Kindlund (2005) kan det vara svårt att sätta maxtid eftersom olika scenarier kan vara unika och det är inte alltid uppenbart vilken maxtid som kan vara lämplig. Sauro & Kindlund (2005) säger att man inte bör chansa fram maxtider, de ska på något sätt vara av betydelse för det specifika scenariot. Anledningen till att vi inte har definierat någon maxtid från början var dels att vi inte hittat eller kunnat ta fram någon av de tider som beskrivs ovan, samt att vi inte kunde göra någon kvalificerad uppskattning på hur lång tid varje scenario och delmål skulle ta. Speciellt första delmålet som till skillnad från de andra delmålen inkluderar uppkoppling till Event Hubs. Eftersom vi inte kunnat uppskatta tid innan testning så beslutade vi att först slutföra scenario 1 för att får en känsla av vilken maxtid som är lämplig för oss. Efter att ha slutfört scenario 1 uppskattade vi att lämplig maxtid var tio timmar.

Vi noterade starttiden efter att vi båda läst uppgiften (delmålet) och noterade stopptiden när vi kunde se önskat resultat i outputapplikationen eller om vi uppnått maxtiden 10 timmar. För scenario 1 hade vi dock ingen maxtid. I scenario 1 delmål A valde vi att inkludera uppkoppling mot både input- och output Event Hub. Det som inkluderades i mätningen var dels det praktiska utvecklingsarbetet, alltså själva kodningen. Vi räknade även med tiden det tog att leta information om hur man skulle gå tillväga, som t.ex. instruktioner eller guider.

Vi har endast räknat den tid som vi spenderat på att lösa uppgifter. När vi har haft tekniska hinder så har vi räknat bort den tiden från den inrapporterade tiden. Vi hade t.ex. problem med en router som hindrade oss att bidra till att lösa uppgiften. Det har även uppstått problem som berodde på andra orsaker som vi inte kunde lasta analystjänsterna för. Exempelvis så hade vi fel anslutning i koden som var vårt eget fel. Vi har räknat tiden som vi spenderat på varje delmål per person. Eftersom vi arbetade samtidigt på samma delmål så blev en timme alltså två timmar.

3.4.4 Ändamålsenlighet

Enligt Sauro (2011) så är andelen slutförda uppgifter en central del för att mäta ändamålsenlighet och det är data som är lätt samla in. Traditionellt mäts detta binärt (1=Uppgift slutförd, 0= Uppgift ej slutförd). Vi valde att lägga till ytterligare en typ av mätresultat, (2=Uppgift slutförd, men med en anmärkning). Anledningen var att vi i scenario 1 delmål C var tvungna att ändra på förutsättningarna för att kunna lösa uppgiften, delmålet gick alltså att lösa men med en brist vi ansåg var värd att notera. Vi visualiserade resultaten i en färgkodad tabell där grönt = godkänt, gult = godkänt med anmärkning samt rött = underkänt. En uppgift anses som slutförd när vi kan se förväntat resultat i outputapplikationen.

3.4.5 Användarnöjdhet

Vi använde oss av SUS (System Usability Scale) formulär som är en standard för att mäta användarnöjdhet i IT-system. SUS formulär använder 10 fördefinierade frågor som besvaras med en poängskala från 1-5, dessa frågor är standardiserade och inget vi själva kommit på. Skalan är en Likertskala där testpersonerna svarat i vilket grad de håller med de olika påståenden som SUS-formuläret ger. Resultatet blir en SUS-poäng på skalan 1-100 som gör att användarnöjdhet kan jämföras. (Brooke, 1996) Enligt Gatsou et al. (2013) är SUS det mest precisa formuläret om man har ett litet antal deltagare. Se bilaga 1 för att se SUS-formuläret.

Frågeformulär är en samling fördefinierade frågor i en bestämd ordning som passar när man vill ha kortfattad och standardiserad data. Ett väldesignat formulär ökar chansen att frågorna genererar de data man vill åt. (Oates, 2006)

Vi har utvärderat användarnöjdhet genom att vi båda individuellt fyllt i SUS-formulär efter varje scenario. Scenario 3 & 4 beslutade vi att slå ihop till ett SUS-formulär eftersom dessa scenarier gick väldigt fort att slutföra för båda analystjänsterna.

3.4.6 Scenarier

Här beskrivs våra scenarier(1-5) samt tillhörande delmål (a, b, c, o.s.v.) som vi har använt för att samla in data och utföra analyserna på.

1. Registrera bilar som färdas över tillåten hastighet. Bilarna som färdas över tillåten hastighet ska sparas i en databas.
    a. Skapa en uppkoppling mot Input Event Hub och Output Event Hub samt registrera samtliga bilar oavsett hastighet och väg.
    b. Registrera samtliga bilar oavsett hastighet men vid en specifik väg.
    c. Registrera samtliga bilar som kör för fort oavsett väg.
    d. Bilarna som kör för fort ska sparas i en SQL-databas. Det som skall sparas i databasen är: registreringsnummer, tidpunkt, väg-ID, hastighetsöverträdelse, tillåten hastighet.

2. Ta reda på vilken väg som har lägst trafikflöde under en viss period (inom den senaste minuten).
    a. Räkna totala antalet bilar som passerat alla kameror under en viss period (inom den senaste minuten).
    b. Räkna antalet bilar som passerat vid varje specifik kamera/väg inom den senaste minuten.
    c. Beräkna trafikflödet per väg i procent och rangordna vägarna med lägst trafikflöde högst upp.

3. När en bil passerar som finns i en SQL-databas/blob med efterlysta bilar presenteras den efterlysta bilen i outputapplikationen.
    a. Sortera ut efterlysta bilar
    b. Spara efterlysa bilar i SQL-databas (registreringnummer, väg-ID, tid)

4. Beräkna medelhastighet på varje väg inom den senaste minuten.
    a. Addera samtliga hastigheter för att visa totalsumma inom den senaste minuten.
    b. Beräkna medelhastighet på alla vägar.
    c. Addera samtliga hastigheter för att visa totalsumma inom den senaste minuten på varje väg.
    d. Beräkna medelhastighet på varje väg inom den senaste minuten.

5. Om mer än 80 % av bilarna på en väg färdas 20km/h långsammare än tillåten hastighet under en tidsperiod (15 sekunder) så triggas ett event som talar om att en trafikstockning är nära förestående.
    a. Registrera alla bilar som färdas 20km/h långsammare än tillåten hastighet på en specifik väg.
    b. Räkna antalet bilar som har färdats 20km/h långsammare än tillåten hastighet de senaste 15 sekunderna på alla vägar.
    c. Beräkna i procent andelen bilar som har färdats 20km/h långsammare de senaste 15 sekunderna på alla vägar.
    d. Skicka ett event till outputapplikationen om 80 % eller mer av bilarna på en väg färdas 20km/h långsammare än tillåten hastighet under en tidsperiod (15 sekunder).

3.4.7 Delmålens testordning

För att respektive analystjänst inte skulle få någon fördel p.g.a. vilken ordning delmålen utfördes, så planerade vi en förutbestämd testordning (se bilaga 2). Vid ett tillfälle så bröt vi testordningen, det var under delmål C i ASA som vi satte i vänteläge. Vi fortsatte istället med delmål C för HDIS för att sedan gå tillbaks till delmål C för ASA. Efter detta så följde vi testordningen enligt bilaga 2. Anledningen till att vi bröt testordningen under delmål C var att vi skickade en fråga till Microsoft som vi behövde få svar på för att kunna fortsätta med ASA delmål C.

3.4.8 Förutsättningar

För att kunna utföra testerna av ASA och HDIS var vi tvungna att skaffa Azurekonton eftersom det kostar pengar att använda dessa tjänster. Eftersom vi var studenter kunde vi genom Högskolan Dalarna få studentkonton som räckte i 3 månader och med en månadsbudget på 700 kr (Microsoft Azure, 2015d). Eftersom vi hade var sitt konto så hade vi totalt 1400kr att göra slut på per månad. Detta såg vi som en risk innan vi påbörjade testerna eftersom vi inte visste hur mycket det skulle kosta att utföra realtidsanalyser i de båda tjänsterna. Med tanke på att vi inte skickar några stora mängder data i vår simulator uppskattade vi ändå att 1400kr i månader skulle räcka.

3.5 Dataanalys

3.5.1 Deltagande observationer

Våra fältnoteringar har vi använt för att dokumentera hur vi arbetade med utvecklingen av analyslogiken. Vid analys av kvalitativ data i textformat ska man enligt Oates (2006) först se till att formatera data för att underlätta analysarbetet. Genom att vi från början sorterat våra fältnoteringar i scenarier och delmål samt att vi har skrivit dessa tillsammans, så var insamlade data redan kategoriserade och i ett format som var redo för analys.

Vi började med att läsa igenom alla våra fältnoteringar för att få en överblick över vad vi hade noterat angående analystjänsterna. Sedan identifierade vi nyckelteman i våra fältnoteringar. Vi började med att hitta segment i texten som vi direkt kunde anse som irrelevanta för vårt forskningssyfte, t.ex. datum, klockslag och kodexempel som vi dokumenterat på andra ställen. Efter att ha rensat onödig text identifierade vi segment som vi ansåg som relevanta för vår forskning. Sedan identifierade vi text som var viktig för att beskriva de relevanta segment som vi tidigare sorterat ut. Detta utfördes iterativt för varje scenario och analystjänst. (Oates, 2006)

3.5.2 Effektivitet

En kvantitativ dataanalys fokuserar på data som man kan utföra beräkningar på (Oates, 2006).

Vi började med att summera tiderna för varje scenario för att få totaltid per scenario. Sedan summerade vi dessa till en totaltid för samtliga scenarier för varje analystjänst.

Enligt Oates (2006) så kan man använda sig av tabeller och diagram för att på ett enklare sätt visualisera mönster i data. Vi använde oss av tabeller för att visa detaljer för varje scenario. Stapeldiagram användes för att visualisera totaltid för varje scenario och tjänst. Vi har även använt ett linjediagram för att visa tid för varje delmål för samtliga scenarier.

3.5.3 Ändamålsenlighet

Vi använde oss av tabeller för att visualisera antalet delmål som blev godkända, godkända med anmärkning samt underkända delmål för varje analystjänst.

3.5.4 Användarnöjdhet

Efter att vi individuellt fyllt i varsitt SUS-formulär efter att ett scenario blivit slutfört så beräknade vi våra individuella resultat till ett medelvärde per fråga. I SUS formuläret beräknas alla udda frågor med skalans position minus 1. Jämna frågor beräknas med 5 minus skalans position. Svarens totala summa multipliceras med 2.5 för att få ett SUS-resultat på skalan 1-100. (Brooke, 1996) För att underlätta analysen använde vi Excels beräkningsfunktion.

Efter att alla scenarier var avslutade sammanställde vi SUS-poängen för varje scenario i ett Excelark och beräknade även en genomsnittlig SUS-poäng för varje analystjänst. Vi visar SUS-resultaten med hjälp av stapeldiagram som visualiserar skillnader i SUS-poäng per scenario samt genomsnittlig SUS-poäng.

3.6 Metodkritik

Utvärdera sig själv i utvecklingsarbetet är en stor utmaning, det är viktigt att förutsättningarna är klara och att datainsamlingen är väl planerat innan observationsfasen.

För att undersöka analystjänsternas användbarhet så definierade vi i förväg fem tydliga scenarier med tillhörande delmål. Dessa scenarier och delmål har vi använt som grund i vår empiriska undersökning och därför var det extra viktigt att dessa definierades innan undersökningen startade. I denna fas av arbetet bestämde vi även hur mätningarna utifrån våra användbarhetskriterier skulle utföras: *effektivitet, ändamålsenlighet samt användarnöjdhet.*

I inledningen av arbetet så var tanken att vi skulle mäta användarnöjdhet genom att andra utvecklare skulle testa de olika analystjänsterna och sedan fylla i SUS-formulär. Men p.g.a. tidsbrist har detta inte varit möjligt. Vi undersökte istället möjligheten att samla in SUS-data genom att ha en workshop med utvecklare hos vår samarbetspartner, vilket skulle spara tid jämfört med om utvecklare individuellt skulle utveckla något. Men vi saknade nu både tid och resurser för att kunna utföra någon workshop. Den enda lösning som vi kom på var att vi själva skulle fylla i SUS-formuläret efter varje avslutat scenario för att vi skulle få några data att mäta användarnöjdhet med. Vi gjorde detta individuellt utan att ha någon kontakt med varandra. Även om vi endast har samlat in data från oss själva (två personer), så tycker vi ändå att vi fått ett trovärdigt resultat eftersom de stämmer bra överens med de data vi fick från de systematiska observationerna. Vi är väl medvetna om att det hade varit önskvärt att göra en bredare insamling.

För att minimera risken för att våra mätresultat skulle påverkas p.g.a. problem som kan uppstå när man arbetar vid en dator som man själv inte äger. Därför valde vi att arbeta hemifrån från våra personliga datorer. Genom erfarenhet vet vi att det kan uppstå problem med brandväggar, administratörsrättigheter och liknande om man arbetar från en dator som man inte äger och har full kontroll över. Under hela arbetet har vi haft kontakt via applikationen Skype och det har fungerat bra för oss. Eftersom vi inte sitter på samma plats och arbetar har vi ibland upplevt svårigheter i kommunikation när vi inte kunnat se varandras skärmar. I utvecklingsarbetet har vi använt oss av Microsofts Team Foundation Server, detta har underlättat hanteringen av de olika Visual Studioprojekt vi använt samt av den kod vi skapat.

Vi hade stora bekymmer när det gällde tidmätningen under arbetets gång. Vi noterade starttiden när vi läste frågan och noterade sedan stopptiden när vi kunde se förväntat resultat i outputapplikationen. Vi hade mycket tekniska problem som inte berodde på analystjänsterna under flera av delmålen. Detta resulterade i att vi drog av tiden där vi hade tekniska problem från den rapporterade tiden. Det visade sig att detta var svårare än vi först trodde. Ibland hade vi problem en längre tid utan att förstå vad problemet egentligen berodde på. Det var svårt att i efterhand uppskatta hur lång tid som egentligen spenderats som inte borde vara med i den inrapporterade tiden. Vi insåg tidigt att vi kommer att ha svårt att notera den exakta tiden på grund av oförutsedda problem. Vi har därför uppskattat bortfall i tid så gott vi har kunnat men kunde konstatera att ju längre tid som gått desto svårare var det att uppskatta den exakta tiden. Därför blev de flesta tider över timmen avrundade till halvtimmar. Detta har naturligtvis påverkat sluttiderna, men eftersom vi endast har analyserat de sluttiderna som avvikit kraftigt så anser vi att det inte påverkar slutresultatet.

4 Resultat av våra tester

Här redovisar vi resultaten från vår datainsamling. Vi har strukturerat avsnittet på användbarhetskriterierna effektivitet, ändamålsenlighet och användarnöjdhet samt våra fältnoteringar som innehåller kommentarer som kan relateras till alla tre kriterier.

4.1 Effektivitet

Här redovisas resultaten av tidmätningen för varje analystjänst, tiderna som visas är utvecklingstider och är i formatet timmar: minuter. Båda tabellerna visar tider för varje delmål. Vi har även beräknat en totaltid för varje scenario och totaltid för alla scenarier.

I tidmätningen för ASA (tabell 7) kan vi utläsa att totaltiden för alla scenarier var 36:00. Scenario 2 Delmål C var det enda delmål där maxtiden gick ut och fick därför 10 timmar.

Tabell 7 - ASA effektivitet

	delmål A	delmål B	delmål C	delmål D	Totaltid
Scenario 1	04:00	00:20	09:30	01:00	14:50
Scenario 2	01:00	00:20	10:00		11:20
Scenario 3	00:20	00:20			00:40
Scenario 4	00:10	00:10	00:10	00:10	00:40
Scenario 5	00:20	00:10	06:00	02:00	08:30
Totalt					36:00

I tidmätningen för HDIS (tabell 8) kan vi utläsa att den totala tiden för alla scenarier blev 49:20.

Tabell 8 - HDIS effektivitet

	delmål A	delmål B	delmål C	delmål D	Totaltid
Scenario 1	12:30	16:00	03:00	01:00	32:00
Scenario 2	04:00	03:00	03:00		10:00
Scenario 3	01:00	00:40			01:40
Scenario 4	00:30	00:10	00:10	00:10	01:00
Scenario 5	00:30	00:40	01:30	02:00	04:40
Totalt					49:20

4.2 Ändamålsenlighet

I tabell 9 och tabell 10 nedan kan vi se godkända och underkända delmål. Delmål som är godkända är markerade med grön bakgrund och underkända med röd. Delmål som är markerade med gul bakgrund indikerar att delmålet är godkänt men att det finns en anmärkning.

Tabell 9 - ASA ändamålsenlighet

	delmål A	delmål B	delmål C	delmål D
Scenario 1	G	G	G	G
Scenario 2	G	G	U	
Scenario 3	G	G		
Scenario 4	G	G	G	G
Scenario 5	G	G	G	G

Tabell 10 - HDIS ändamålsenlighet

	delmål A	delmål B	delmål C	delmål D
Scenario 1	G	G	G	G
Scenario 2	G	G	G	
Scenario 3	G	G		
Scenario 4	G	G	G	G
Scenario 5	G	G	G	G

I scenario 1 delmål C så var förutsättningen från början att använda referensdata från en SQL-databas. I vårt fall gällde det att hämta tillåten hastighet för aktuell väg. Det visade sig att ASA endast kunde använda referensdata från blob-storage. Delmålet kunde slutföras med förutsättningen att ASA hämtade referensdata från blob-storage istället för en SQL-databas, detta gjorde att delmålet blev godkänt men med en anmärkning.

ASA scenario 2 delmål C uppnådde vi maxtiden utan att delmålet var uppfyllt, därför blev delmålet underkänt.

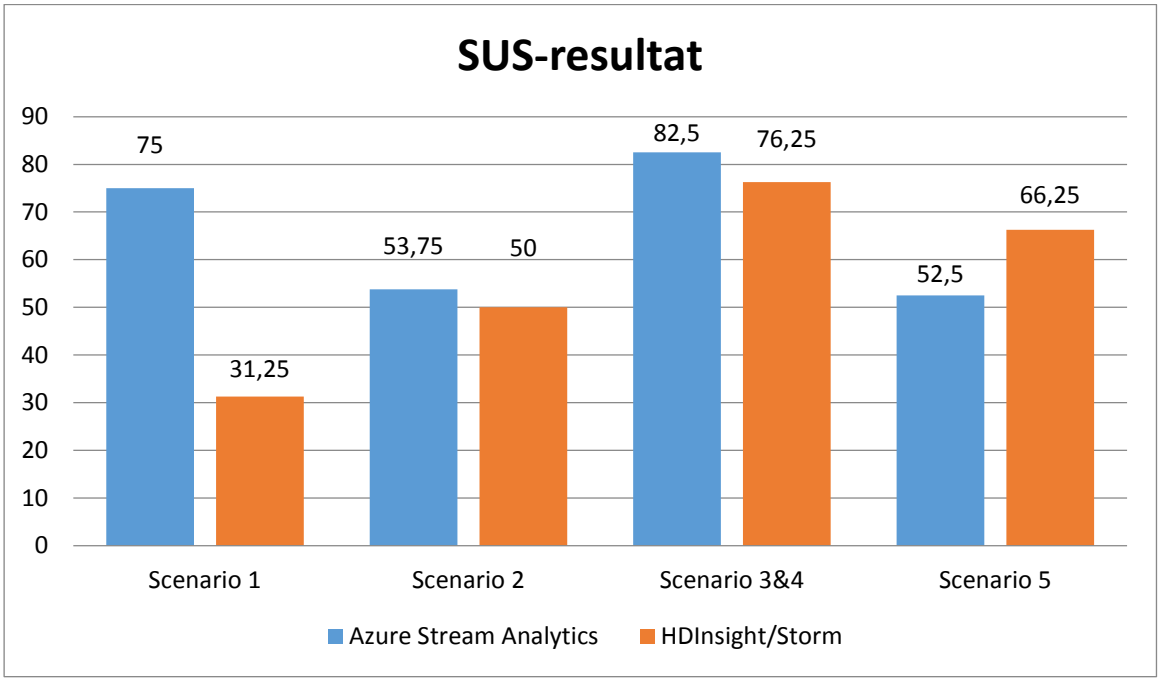
ASA scenario 5 delmål C ville vi beräkna andelen bilar som färdades minst 20km/h långsammare än tillåten hastighet de senaste 15 sekunderna. Problemet var att vi inte kunde räkna totala antalet bilar om det inte fanns någon bil som färdades långsamt de senaste 15 sekunderna. Detta resulterade i att ASA inte skickade någon data till outputapplikationen när det färdades 0 % långsamma bilar på en väg. De data som skickades till outputapplikationen var inte heller helt konsekvent då ASA ibland skickade samma data flertalet gånger till outputapplikationen. Vi satte ändå delmålet som godkänt men med en anmärkning eftersom beräkningen vi ville göra fungerade och skickades till outputapplikationen.

ASA har 14 godkända delmål, ett underkänt samt två delmål som är godkända med anmärkning. HDIS har godkänt på samtliga delmål.

4.3 Användarnöjdhet

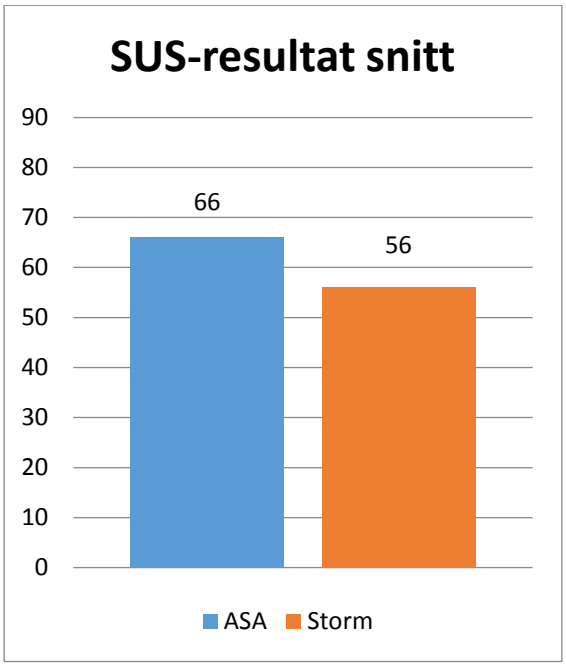
I detta kapitel presenterar vi våra SUS-resultat vilket är ett mått för vad användare tycker om användbarhet i en tjänst.

I figur 9 nedan kan vi se SUS-poäng per scenario för ASA och HDIS.



Figur 9 - SUS-resultat per scenario

I figur 10 ser vi ett genomsnittligt SUS-resultat för alla scenarier, där kan vi utläsa att ASA har en högre användarnöjdhet än HDIS totalt sett.



Figur 10 - SUS-resultat i snitt

4.4 Fältnoteringar

Här redovisar vi de fältnoteringar vi gjort under våra deltagande observationer. Vi har delat upp redovisningen på analystjänst, scenario och delmål. Det som redovisas här är information som är relevant för studien. För vidare information se bilaga 5 för ASA och bilaga 6 för HDIS.

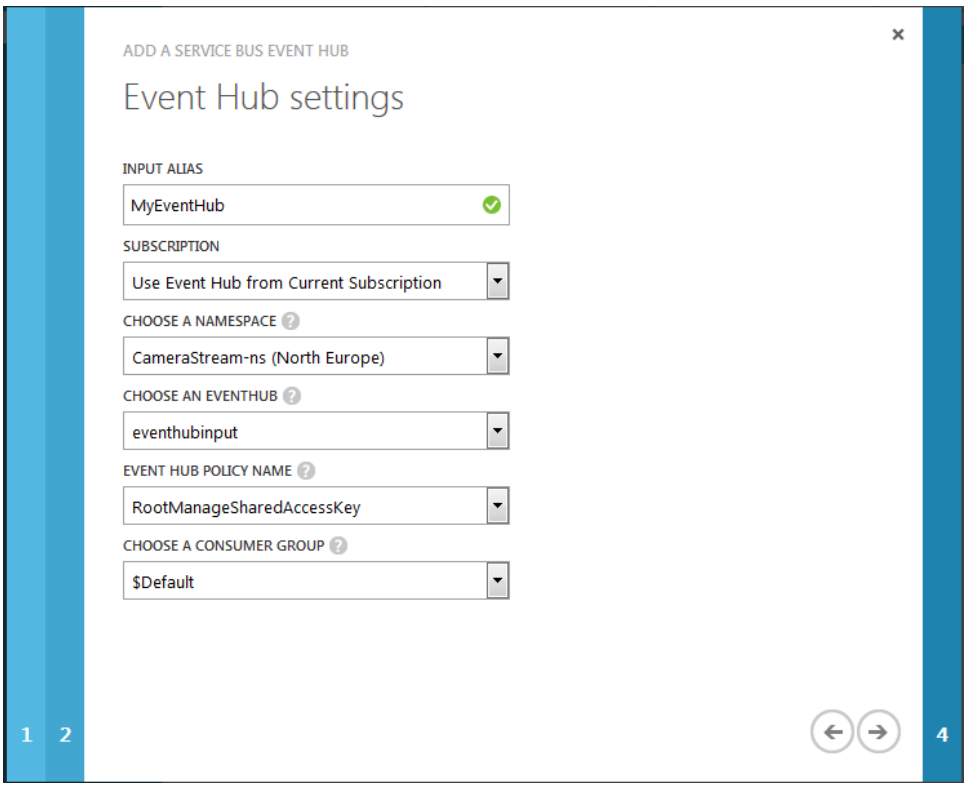
4.4.1 Azure Stream Analytics

Scenario 1

Delmål A

Vi följde guiden som fanns i Azureportalen innehållande fem steg om hur man skapar ett ASA arbete. Efter att ha genomfört delmål A så tycker vi att det var relativt enkelt att sätta upp ASA-tjänsten med tanke på att den var helt ny för oss. Det var dock svårt att via instrumentpanelen se om ASA hade tagit emot något event eller inte eftersom instrumentpanel hade en fördröjning på flera minuter.

Det var tydligt var man skulle definiera de olika delarna: inputs, SQL-fråga och outputs i det användarvänliga gränssnittet. När vi valde t.ex. inputs så kunde man via dropdown-menyerna få hjälp med vad man kunde välja, t.ex. så fanns en lista med de Event Hubs som vi redan hade skapat. Figur 11 visar gränssnittet för att välja input av typen Event Hub. I ASA kan man endast använda två typer av inputs som strömmande data, Event Hubs och blob-storage.



Figur 11 - ASA input interface

Delmål C

För att testa vår SQL-fråga så använde vi oss av den inbyggda testfunktionen där man kunde ladda upp en JSON- eller CSV- fil och testa SQL-frågan mot denna. Testfunktionen sparade oss tid då vi nu slapp starta ASA-arbetet för att testköra mot strömmande data. Det gick att ladda upp filer som representerade både referensdata och inputströmmar. Att starta ett ASA-arbete tog ca 1,5 minut.

Vi ville jämföra hastigheten från varje bil mot referensdata i form av tillåten hastighet som ligger sparat i en SQL-databas. Men efter att ha läst dokumentationen kunde vi konstatera att det inte gick. ASA kunde endast använda blob-storage som referensdata i filformaten JSON eller CSV.

Scenario 2

Delmål A

Vi läste i Azures dokumentation och hittade aggregatfunktionen COUNT, funktionen skulle användas tillsammans med någon form av window funktion för att bestämma inom vilken tidsram som man vill räkna. En av dessa var slidingwindow där man kunde välja att räkna antalet bilar inom en viss tidsram som ständigt är i rörelse, och man utgår alltid ifrån aktuell tid. När vi använde aggregatfunktioner med tidsfönster så fick vi dock inte ut något resultat i testfunktionen.

Delmål C

Eftersom vi skulle räkna ut en procentsats av trafikflödet för varje väg så behövde vi både det totala antalet bilar och antalet bilar per väg. Det visade sig att det var väldigt svårt att få ut båda dessa siffror i samma fråga, vilket behövs för att kunna räkna ut en procentsats. Efter mycket testande så gick maxtiden ut och vi var tvungna att avsluta delmålet. Vi kom så långt med delmålet att vi med viss felmarginal kunde se totala antalet bilar samt totala antalet bilar per väg.

Scenario 3 & 4

Dessa scenarier gick snabbt, delmålen påminde om tidigare delmål och därför visste vi hur vi skulle lösa uppgifterna.

Scenario 5

Delmål C

Det uppstod nu ett problem med att vi inte kunde se något resultat i outputapplikationen om alla bilar körde enligt tillåten hastighet. Det var först när någon bil körde 20km/h långsammare som ASA skickade data till outputapplikationen. Vi försökte använda T-SQL funktionerna ISNULL och COALESCE för att lösa vårt problem men ingen av dessa funktioner stöds av ASA enligt våra tester. Vi noterade även att resultaten inte alltid var konsekventa, antalet bilar som körde långsamt kunde ibland vara fler än totala antalet bilar.

4.4.2 HDInsight/Storm

Scenario 1

Delmål A

Eftersom vi tidigt insåg att HDIS bestod av många komponenter som var nya för oss så behövde vi hitta instruktioner som började från grunden. Det tog lång tid innan förstod hur Stormtopologin och dess komponenter hängde ihop. Informationen som presenterade via instrumentpanelen var svår för oss att förstå då det fanns väldigt lite förklaringar om vad denna skulle användas till. Via instrumentpanelen kunde vi dock se att analystjänsten arbetade men vi kunde inte se något förväntat resultat. Instrumentpanelen hade en fördröjning på några sekunder. Det visade sig att man var tvungen att använda sig av en hybrid-topologi och använda Javaklasser för att kunna hämta och skicka data till Event Hubs.

Delmål B

Det var svårt att förstå hur vi skulle skapa kopplingen mellan Javaspout, C#bolt och Javabolt så att data kan flöda mellan dessa komponenter. Vi kunde inte heller testa topologin lokalt längre eftersom vi inte visste hur man gjorde detta med Javabolts och Javaspouts, instruktioner om detta saknades. Det var tidskrävande att behöva ladda upp topologin och köra den i stormklustret varje gång vi behövde testa topologin. Eftersom det finns väldigt lite exempel på internet om hur man bygger Stormtopologier i C# så fanns det heller inte mycket hjälp att få. Problemet var att vi var tvungna att översätta de data som skickades mellan C# och Javakomponenterna. Vi noterade i Azureportalen att Stormklustret kostade pengar även om vi inte analyserade någon data. Klustret gick heller inte att stänga av utan vi var tvungna att ta bort det helt.

Delmål C

Eftersom det är tidskrävande att ladda upp en topologi i Stormklustret för att testa om kod och logik är korrekt så började vi nu parallellt att använda oss av en konsolapplikation. I konsolapplikationen testade vi koden lokalt i den mån det var möjligt vilket sparade tid. Tiden det tog att ladda upp varierade något men det tog ca 2,5 minut att ladda upp och köra en topologi.

Delmål D

Eftersom att det visade sig att ASA inte kunde hantera referensdata från en SQL-databas utan bara kunde använda blob-storage, så ändrade vi förutsättningarna för ASA testerna. Det visade sig att HDIS kunde hantera både SQL-databaser och blob-storage.

Scenario 2

Delmål A

Det svåra med det här delmålet var att komma på hur vi logiskt skulle lösa uppgiften med att arbeta inom ett tidsfönster i ständig rörelse.

Scenario 3 & 4

Precis som i ASA gick det snabbt att lösa scenario 3 & 4 eftersom det även här påminde om tidigare delmål och därför visste vi hur vi skulle lösa uppgifterna.

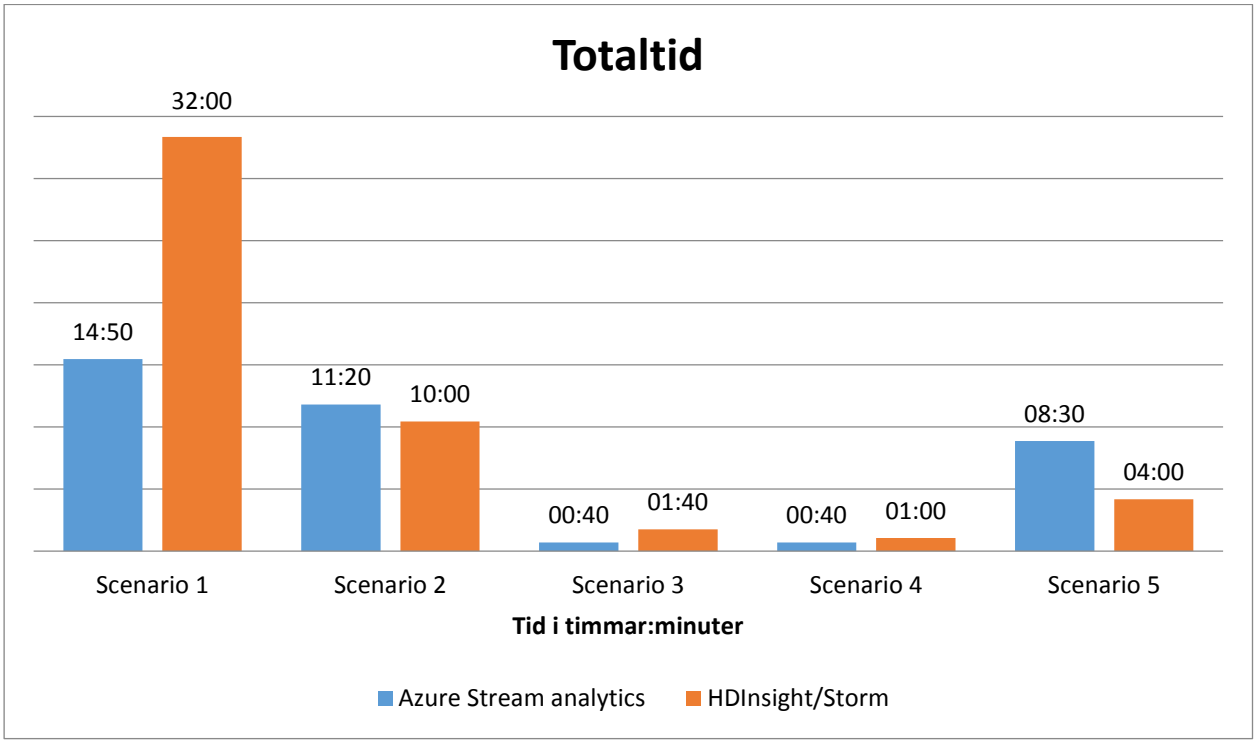
Scenario 5

Vi hade egentligen inga större svårigheter i detta scenario heller, utmaningen låg i att tänka ut hur vi logiskt skulle lösa uppgifterna.

5 Analys

5.1 Effektivitet

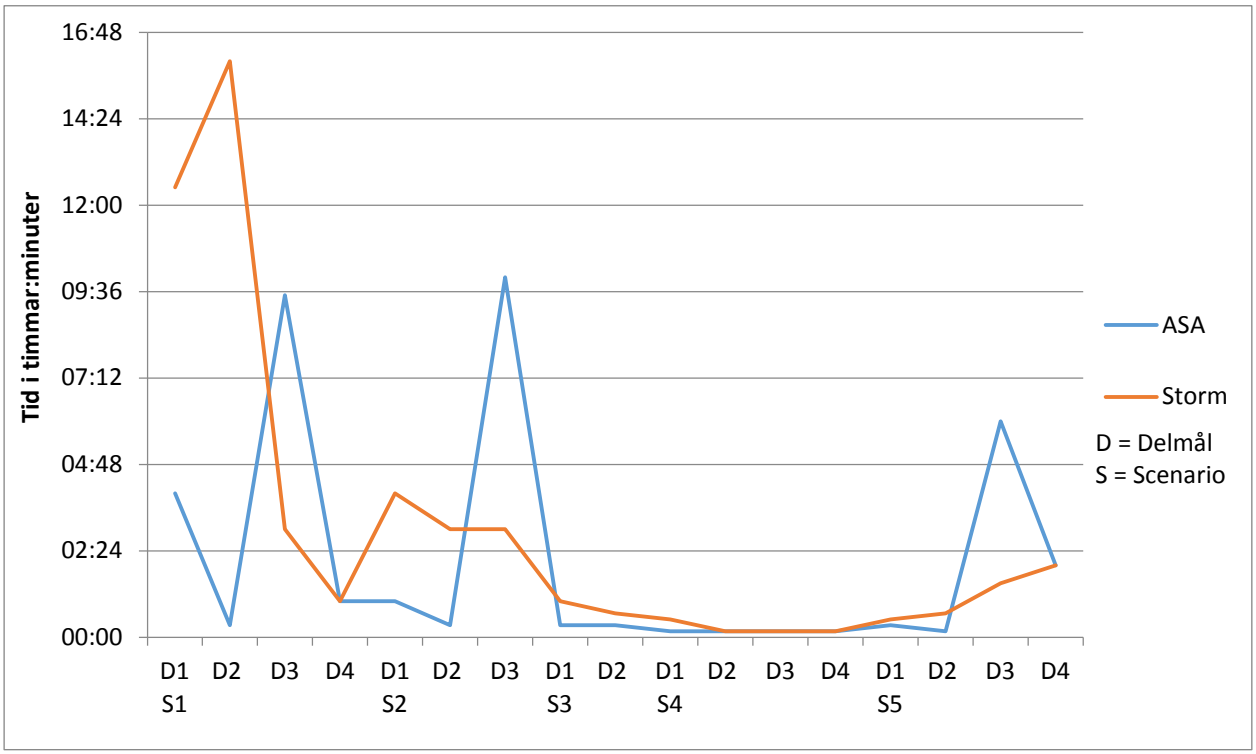
Figur 12 nedan visar totaltid per scenario för båda analystjänsterna. Vi kan se att det inledande scenario 1 har mest tid i båda analystjänsterna. Anledningen till detta är att scenario 1 delmål A inkluderade uppkoppling till Event Hubs som inte behövde göras i de senare scenarierna. En annan faktor var att båda tjänsterna var nya för oss och det tog tid att lära sig hur de fungerade. Scenario 2 hade även den relativt hög totaltid på båda scenarierna men mindre än scenario 1, speciellt för HDIS. Detta beror främst på att vi lärt oss grunderna i de båda tjänsterna, hur de fungerar och framför allt, hur man kan testa funktionaliteten. ASA hade ett delmål (delmål C) där maxtiden gick ut som påverkat totaltiden, annars hade den tiden också sjunkit rejält. Scenario 3 & 4 hade istället väldigt låga och analystjänsterna mellan, likvärdiga totaltider vilken indikerar att vi nu kan hantera båda tjänsterna bättre än i scenario 1 och 2. Det beror också på att de uppgifter som ingick i dessa scenarier kunde lösas på liknande sätt som i de tidigare scenarierna. I scenario 5 steg totaltiden igen, främst för ASA som hela tiden legat under HDIS i totaltid. Detta berodde främst på att scenariot var lite mer komplicerat än de tidigare.



Figur 12 - Totaltid per scenario

För att vidare illustrera de olika analystjänsternas inlärningskurva så har vi visualiserat tiderna för alla delmål i figur 13. Enligt Sirosh (2015) så ska ASA göra det enklare att sätta upp realtidsanalyssystem som tar emot strömmande data. Om vi studerar figur 13 kan vi se att HDIS har en högre tröskel än ASA vilket styrker Sirosh (2015) påstående. Men det kan vara värt att notera att efter halva vår undersökning ligger tjänsterna på i stort sett likadana tider.

Vi ser också i att ASA har tre toppar som indikerar att tjänsten är ojämnare än HDIS. Detta kan bero på att ASA som tjänst är nyare än HDIS (Sirosh, 2015) och att det inte finns särskilt mycket dokumentation som hjälp i utvecklingen.



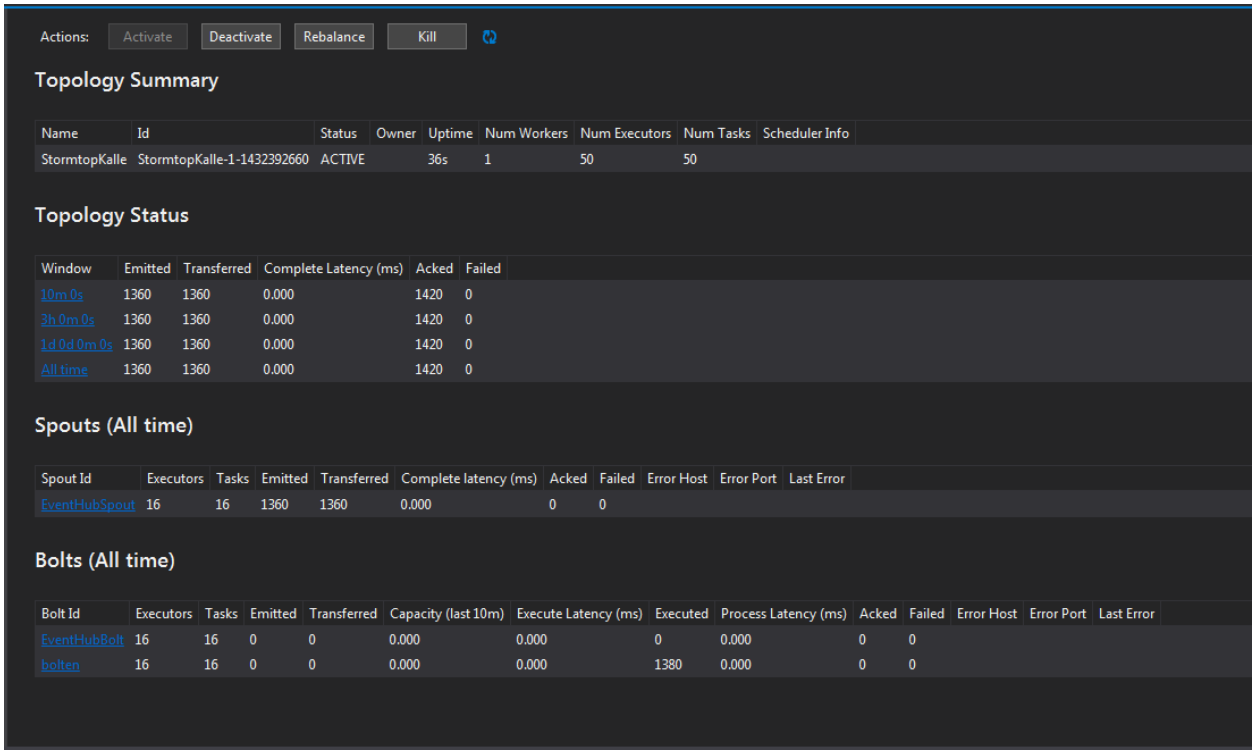
Figur 13 - Tider per delmål

Scenario 1

Om vi tittar på scenario 1 (tabell 11), som enligt figur 12 avviker kraftigt från de övriga, så kan vi se att delmål A med ASA tog 4 timmar, medan HDIS tog 12,5 timmar. När det gällde ASA så fick vi börja med att bekanta oss med tjänstens delar i Azureportalen. Det gick relativt enkelt att sätta upp ASA-tjänsten då det grafiska gränssnittet var till god hjälp (figur 11). Med HDIS så följde vi instruktioner i flera steg för att få förståelse för hur de olika delarna i Stormtopologin fungerade. I båda tjänsterna upplevde vi problem med att det var svårt att se om tjänsterna fungerade. Vi började med att försöka använda båda tjänsternas instrumentpaneler för att kunna utläsa om analysen fungerade, men vi upplevde att ingen av analystjänsterna levererade någon annan information än att analystjänsterna arbetade.

Figur 14 - ASA instrumentpanel

Vi upplevde ändå att ASA's instrumentpanel (figur 14) var tydligare än HDIS (figur 15) samtidigt som HDIS innehöll lite mer information. Båda visade begränsat med information i en första vy, och hade detaljer i andra vyer. ASA's instrumentpanel hade fördröjning på flera minuter medan HDIS hade fördröjning på några sekunder.



Figur 15 - HDIS instrumentpanel

Som vi kan se i tabell 11 så har delmål B tagit betydligt längre tid med HDIS. I delmål A behövde vi inte utföra någon analys på de inkommande data men detta var ett krav i delmål B då vi ville sortera ut bilar som färdades på en specifik väg. Eftersom delmål A inte analyserade data så måste vi addera tiderna från delmål A och B för att få ut tiden det tog innan vi kunde börja analysera data. Det tog alltså ca 28 timmar för HDIS och ca 4 timmar för ASA.

Tabell 11 - Tider scenario 1

	delmål A	delmål B	delmål C	delmål D
Azure Stream Analytics	04:00	00:20	09:30	01:00
HDInsight/Storm	12:30	16:00	03:00	01:00

Scenario 2

Scenario 2 har en hög totaltid för båda analystjänsterna. När det gäller HDIS så fick vi själva tänka ut logiken för hur man räknar bilar inom en viss tidsperiod i realtid vilket tog tid. ASA hade färdiga funktioner för att räkna händelser under en tidsperiod vilket sparade tid. Som vi kan se i tabell 12 så uppnådde ASA maxtiden för delmål C vilket kraftigt påverkade totaltiden. Att ASA uppnådde maxtiden för delmål C berodde på att det blev svårt att kombinera totalt antal bilar med antal bilar per väg för att göra en beräkning.

Tabell 12 - Tider scenario 2

	delmål A	delmål B	delmål C
Azure Stream Analytics	01:00	00:20	10:00
HDInsight/Storm	04:00	03:00	03:00

Scenario 3 & 4

Som vi kan se i tabell 13 och tabell 14 så har ASA låga tider på samtliga delmål i scenario 3 och 4. Detta beror till stor del på att de föregående scenarierna innehöll liknande uppgifter som vi har tagit lärdom av. Som vi kan se i tabell 13 så har HDIS högre tider än ASA på delmål A och B i scenario 3. I tabell 14 kan vi se att HDIS har en högre tid på delmål A men resterande delmål håller samma låga tider som ASA.

Tabell 13 - Tider scenario 3

	delmål A	delmål B
Azure Stream Analytics	00:20	00:20
HDInsight/Storm	01:00	00:40

Tabell 14 - Tider scenario 4

	delmål A	delmål B	delmål C	delmål D
Azure Stream Analytics	00:10	00:10	00:10	00:10
HDInsight/Storm	00:30	00:10	00:10	00:10

Scenario 5
Scenario 5 kombinerade funktioner från tidigare scenarier och är logiskt sett det mest komplicerade scenariot, detta förklarar varför det tog längre tid än de två föregående. I tabell 15 kan vi se att de två första delmålen i båda analystjänsterna ligger på relativt låga tider medan de avslutande delmålen stiger. I ASA's fall hade vi problem som liknade delmål C i scenario 2. Men nu kunde vi lösa problemet vi hade med att kombinera data från olika källor som hade aggregatfunktioner. Problemet med att vi inte kunde se resultat om inte WHERE-villkoret uppfylldes i ASA lyckades vi inte lösa, men vi fick ut ett godkänt resultat på delmål C ändå.

Tabell 15 - Tider scenario 5

	delmål A	delmål B	delmål C	delmål D
Azure Stream Analytics	00:20	00:10	06:00	02:00
HDInsight/Storm	00:30	00:40	01:30	02:00

5.2 Ändamålsenlighet

Som vi kan se i tabell 16 nedan så fick HDIS samtliga delmål godkända, ASA fick ett delmål underkänt då maxtiden uppnåddes. I scenario 1 delmål C så var förutsättningen från början att använda referensdata från en SQL-databas. I vårt fall gällde det att hämta tillåten hastighet för aktuell väg. Det visade sig att ASA endast kunde använda referensdata från blob-storage. Delmålet kunde slutföras med förutsättningen att ASA hämtade referensdata från blob-storage istället för en SQL-databas. Eftersom HDIS kan hantera referensdata från både SQL-databas och blob-storage så kunde vi här notera att ASA hade en brist gällande ändamålsenlighet.

Problemet i scenario 5 delmål C, var att ASA skulle hålla reda på variabler och göra alla beräkningar på något sätt i samma SQL-fråga. I HDIS var det enklare att dela upp alla delar som behövs i beräkningen eftersom det är skrivet i ett objektorienterat programmeringsspråk.

Vi upptäckte att det inte finns särskilt mycket dokumentation eller exempel om funktionaliteten i ASA. För att lösa vissa problem i ASA's SQL fick vi söka allmänna svar om SQL, och alla funktioner stöds inte ännu i ASA, som t.ex. ISNULL och COALESCE. Även HDIS saknar viss funktionalitet i C#, t.ex. så används Javabolts och Javaspouts vid uppkoppling mot Event Hubs (Franks, 2015b).

Tabell 16 - Ändamålsenlighet

	Godkända	Godkända med anmärkning	Underkända
Azure Stream Analytics	14	2	1
HDInsight/Storm	17	0	0

5.3 Användarnöjdhet

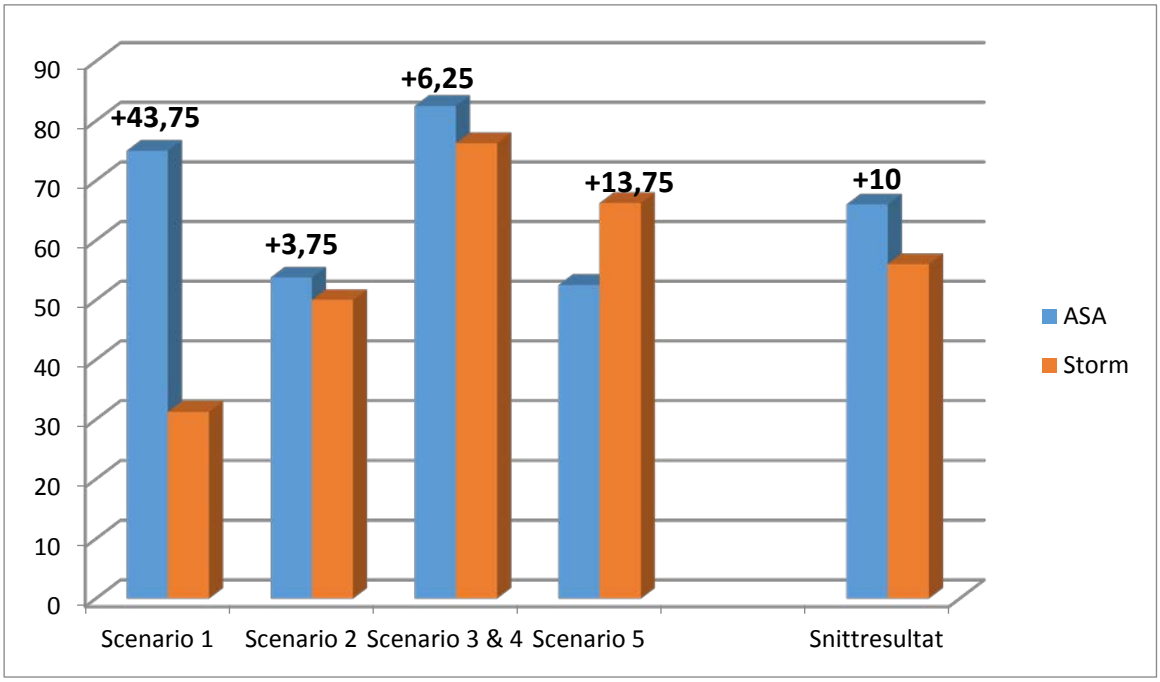
I figur 16 kan vi se att ASA har 43,75 SUS-poäng (p) mer är HDIS i scenario 1. ASA har alltså drygt dubbelt så mycket SUS-poäng som HDIS, vilket tyder på att vi tyckte att ASA var betydligt mer användbart än HDIS.

I scenario 2 så har SUS-poängen jämnat ut sig mellan tjänsterna och vi ser endast en skillnad på 3,75p. ASA har 21,25p minde jämfört med föregående scenario medan HDIS har 18,75p mer.

I scenario 3 & 4 så har båda tjänsterna höga SUS-poäng. Detta beror troligtvis på att vi inte hade några problem i utvecklingen i någon av tjänsterna. ASA har dock 6,25p mer än HDIS.

I scenario 5 kan vi för första gången se att HDIS har en högre poäng än ASA, skillnaden är 13,75p vilket är den näst största differensen mellan tjänsterna. Vi har inte noterat några större problem i utvecklingen av analyslogik i HDIS i detta scenario, vilket kan förklara den höga poängen för HDIS. ASA hade däremot flera noteringar om problem i utvecklingen av SQL-frågor.

Som vi kan se i figur 16 så har ASA ett högre snittresultat än HDIS vilket tyder på att vi tycker att ASA är mer användbart. Det kan dock vara noterbart att vi tyckte att HDIS var mer användbart än ASA i det sista scenariot. Om man jämför differensen mellan scenarierna så kan man se tecken på en stigande trend för HDIS. Med tanke på att differensen var 43 poäng till ASA's fördel på scenario 1 men snittet blev till slut endast 10 poäng, så tyder detta på att vi tyckte att HDIS blev mer användbart under arbetets gång.



Figur 16 - SUS-poäng differens

6 Resultat och diskussion

Enligt våra mätningar så gick det betydligt fortare att sätta upp en ASA-tjänst än en HDIS-tjänst, enligt resultatet på våra SUS-mätningar var det också tydligt att vi föredrog ASA framför HDIS i det inledande scenariot. Det användarvänliga grafiska gränssnittet i Azureportalen var till stor hjälp när man skulle definiera input och output till ASA. Det krävdes ändå lite förståelse för vilka komponenter som stöds i ASA, men det krävs ingen programmeringskunskap för att skapa dessa. Även om det gick fortare att sätta upp en ASA-tjänst så har vi noterat att efter halva vår undersökning ligger tjänsterna på i stort sett likadana tider, se figur 12 (s.34).

Det tog betydligt längre tid att sätta upp HDIS jämfört med ASA innan vi kunde börja analysera några data. Även om man har goda programmeringskunskaper så krävs det att man har kunskap om Stormtopologin och vilka komponenter som topologin innehåller för att kunna sätta upp en realtidsanalystjänst som HDIS.

Vi ser i utvecklingstiderna att ASA har tre avvikande toppar som indikerar att tjänsten är något ojämnare än HDIS, detta kan vi även se tecken på i SUS-resultaten. Detta kan bero på att ASA som tjänst är nyare än HDIS och att det inte finns särskilt mycket dokumentation eller exempel om funktionaliteten i ASA. För att lösa vissa problem i ASA's SQL fick vi söka allmänna svar om SQL, och alla funktioner stöds inte ännu i ASA, som t.ex. ISNULL och COALESCE. Vi tror att detta är funktionalitet som med tiden kan komma att implementeras, men i nuläget har vi sett att det finns begränsningar.

För att strömma data till och från Event Hubs så behöver man i HDIS använda sig av en hybrid-topologi som använder sig av Javaklasser i en C# miljö. Detta medför att man måste översätta de data som skickas mellan Java och C# komponenterna. Vi tycker att det är anmärkningsvärt att Microsoft Azures implementation av Apache Storm är beroende av att använda sig av Javaklasser för att skicka data mellan Microsofts molntjänster. Att HDIS har stöd för både Java och Python i kombination med C# är bra, men samtidigt känns det ändå som en begränsning att man i vissa fall är tvungen att kombinera de olika programmeringsspråken. HDIS fick väldigt lågt SUS-poäng i scenario 1 vilket till stor del berodde på att systemet var svårt att sätta upp.

För att testa SQL-frågan i ASA fanns en testfunktion där man kunde ladda upp en JSON- eller CSV- fil som man kunde köra frågan emot. Eftersom det tog ca 1,5 minut att köra igång ett ASA-arbete så sparade vi tid på att använda testfunktionen. Testfunktionen hade dock några begränsningar, när vi använde aggregatfunktioner med tidsfönster så fick vi inte samma resultat i testoutputen som i outputapplikationen. För att testa om en HDIS-topologi fungerade så var vi tvungna att ladda upp topologin i HDIS-klustret vilket var tidskrävande, vi saknade en liknande testfunktion som finns i ASA.

Vi läste i dokumentationen att ASA hade stöd för både SQL-databaser och blob-storage. Det framgick dock inte att som referensdata gick det endast att använda blob-storage. Att inte kunna använda SQL-databasen som referensdata tycker vi är en begränsning värd att notera.

En väsentlig skillnad mellan tjänsterna är att i HDIS så måste man utveckla all logik själv, medan i ASA finns det färdig funktionalitet att tillgå. Ett exempel vi stötte på var slidingwindow-funktionen i ASA som sorterade event i ett tidsbestämt fönster, den logiken fick vi tänka ut själva och implementera i HDIS.

Sett över alla scenarier så har vi sett en trend i både effektivitet och användarnöjdhet. Initialt var vi mer nöjda med ASA samt att utvecklingstiden i scenario 1 även den talade till ASA's fördel. Men under arbetets gång har vi sett att utvecklingstiden och SUS-resultaten pekat på samma trend, nämligen att HDIS har visat lägre utvecklingstider samtidigt som SUS-resultatet ökat desto längre arbetet har pågått. I det avslutande scenariot kan vi se att HDIS har lägre utvecklingstid och bättre SUS-resultat än ASA.

7 Slutsats

Här presenteras slutsatser där vi återkopplar till syfte och problem, generaliserar samt diskuterar kring våra slutsatser.

7.1 Återkoppling till syfte och problem

Syftet med arbetet var att jämföra molnbaserade realtidsanalystjänster med olika arkitekturer: multi-tenant, som vi utvärderat med hjälp av Azure Stream Analytics (ASA) samt single-tenant, som vi utvärderat med hjälp av HDInsight/Storm (HDIS). Genom utvärderingarna har vi kunnat förklara likheter och skillnader med hjälp av användbarhetskriterierna *effektivitet*, *ändamålsenlighet* och *användarnöjdhet*. För att kunna uppfylla syftet har vi svarat på följande frågor:

Vilka likheter och skillnader kan vi se i utvecklingstider?

Effektivitet
Det tog betydligt längre tid att sätta upp HDIS jämfört med ASA innan vi kunde börja analysera några data. Även om det inledningsvis gick fortare att sätta upp en ASA-tjänst så har vi noterat att efter halva vår undersökning ligger tjänsterna på i stort sett likadana utvecklingstider. I ASA finns en inbyggd testfunktion som kan användas för att verifiera SQL-frågan vilket sparade tid, detta saknades dock i HDIS. I våra undersökningar kan vi utläsa att ASA har tre tidsavvikelser som indikerar att tjänsten är något ojämnare än HDIS.

Kan vi identifiera skillnader i funktionalitet?

Ändamålsenlighet
För att strömma data till och från Event Hubs i HDIS så är man tvungen att använda sig av Javaklasser i en C# miljö medan man i ASA använder Azureportalens användarvänliga grafiska gränssnitt.

Vad gäller testning så hade ASA en inbyggd testfunktion för att verifiera SQL-frågan, denna testfunktion var dock ibland inkonsekvent då den inte alltid visade samma resultat som i outputapplikationen. HDIS saknade denna typ av inbyggda testfunktion.

En väsentlig skillnad mellan tjänsterna är att i HDIS så måste man utveckla all logik själv, medan i ASA finns det färdig funktionalitet att tillgå. Samtidigt så saknar ASA viss funktionalitet medan det finns betydligt fler valmöjligheter att lösa uppgifter med HDIS. I HDIS finns möjligheten att kombinera flera olika objektorienterad programmeringsspråk tillsammans med t.ex. SQL för att lösa en uppgift.

Hur upplever utvecklare de olika analystjänsterna?

Användarnöjdhet
Enligt resultatet från våra SUS-mätningar var det också tydligt att vi föredrog ASA framför HDIS i det inledande scenariot. HDIS fick väldigt lågt SUS-poäng i scenario 1 vilket till stor del berodde på att systemet var svårt att sätta upp. ASA hade väldigt ojämna resultat sett över arbetets gång medan HDIS hade en tydlig positiv trend.

7.2 Generalisering

Vi har studerat Azure Stream Analytics och HDInsight/Storm som i studien representerat molnbaserade realtidsanalystjänster med multi-tenant respektive single-tenant arkitekturer. Eftersom vi inte har hittat någon tjänst som kan likställas med ASA under studiens gång, kan vi bara generalisera om eventuella framtida tjänster som kan vara uppbyggda på liknande sätt. ASA kan karaktäriseras som en typisk multi-tenant tjänst genom att användaren är väldigt begränsad i anpassningsmöjligheterna. En stor del i ASA's karaktär är även att analyslogiken består av ett SQL-liknande språk. Vi upplever ändå att en sådan typ av tjänst som ASA har en plats i framtidens berg av realtidsanalystjänster.

HDIS som representant för realtidsanalystjänster med single-tenant arkitektur kan generaliseras till liknade tjänster, närmast till hands ligger Apache Spark. Även om Apache Spark och Apache Storm inte är uppbyggda på samma sätt så anser vi att de ändå är tillräckligt lika för att en generalisering ska kunna göras.

7.3 Diskussion om slutsats

Vi upptäckte tidigt att det var mycket svårare att komma igång med HDIS än ASA, detta pekar både våra tider, fältnoteringar och SUS-undersökning på. I ASA underlättade det väldigt mycket att ha ett grafiskt gränssnitt när man skulle definiera diverse inputs och outputs. Eftersom man väljer ur en dropdown-lista med tillgänglig inputs och outputs var det tydligt vad som var möjligt att välja på, vilket gjorde att det var svårt att göra fel. Det var mer krävande att lära sig hur HDIS fungerade eftersom man själv var tvungen att definiera både inputs, outputs och logiken mellan komponenterna i HDIS.

Både ASA och HDIS är relativt nya tjänster och vi tror att båda kommer att få utökad funktionalitet med tiden. ASA är mer beroende av detta eftersom den är mer standardiserad än HDIS. För att få ytterligare funktionalitet i ASA så måste Microsoft möjliggöra funktionaliteten eftersom ASA har en multi-tenant arkitektur. Den enda begränsningen vi hittat i HDIS är att man måste kombinera olika programmeringsspråk för att få tillgång till viss funktionalitet, i vårt fall gällde det kopplingen till Event Hubs. Eftersom HDIS har en single-tenant arkitektur kan användaren anpassa tjänsten, men detta kräver mer tid och kunskaper i programmering. Visserligen krävs det kunskaper i SQL för att kunna använda ASA, men vi anser att det inte är i samma utsträckning som kunskaperna som krävs för att utveckla topologier i HDIS.

Under studiens gång så har vi upptäckt att det finns betydligt fler valmöjligheter att lösa uppgifter på när det gäller HDIS, vi kan kombinera objektorienterad programmering tillsammans med t.ex. SQL vilket gör HDIS till en flexibel tjänst. ASA har för oss varit väldigt lätt att arbeta med så länge som den analys vi vill utföra matchar med den funktionalitet som ASA stödjer. Dock så känner vi att ASA är begränsad till den färdiga funktionalitet som erhålls, om det ska utföras mer komplicerade beräkningar så har vi upplevt att HDIS är mer användbart.

Källförteckning

Konferenspapper

Bojanova, I. (2011). Analysis of Cloud Computing Delivery Architecture Models. In: *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on, 2011/03/22-2011/03/25.* Biopolis, Singapore: IEEE. ss.453-458

Im, D-H. Cho, C-H. Jung, I. (2014). Detecting a large number of objects in real-time using apache storm. *Information and Communication Technology Convergence (ICTC), 2014 International Conference on .* 22-24 Oct. 2014. Busan:IEEE.ss.836 - 838

Khalil, E., Enniari, S., Zbakh, M., (2013). Cloud computing architectures based multi-tenant IDS. *Security Days (JNS3), 2013 National* , vol., no., pp.1,5, 26-27 April 2013

Krebs, Rouven. (2012). Architectural Concerns in Multi-Tenant SaaS Applications. *Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*. Conference on Cloud Computing and Services Science. SciTePress.

Marston, S., Li, Z., Bandyopadhyay, S., Ghalsasi, A. (2011). Cloud Computing – The Business Perspective. *Proceedings of the 44th Hawaii International Conference on System Sciences - 2011.* IEEE

Mera, D. Batko, M. Zezula P. (2014). Towards Fast Multimedia Feature Extraction: Hadoop or Storm. *Multimedia (ISM), 2014 IEEE International Symposium on.* 10-12 Dec. 2014. Taichung:IEEE.ss.106 – 109

Mohamed, N. (2014). Real-Time Big Data Analytics: Applications and Challenges. In: *High Performance Computing & Simulation (HPCS), 2014 International Conference on, 2014/07/21-2014/07/25.* Bologna: IEEE. ss.305-310

Osman, A., El-Refaey, M., Elnaggar, A. (2013). Towards Real-Time Analytics in the Cloud. In: O'Conner, L. *2013 IEEE Ninth World Congress on Services, 2012/01/30-2012/02/02* . Santa Clara, CA: IEEE.ss.428-435

Pathirage, M., Perera, S., Kumara, I., & Weerawarana, S. (2011, Juli). A multi-tenant architecture for business process executions. In *Web services (icws), 2011 ieee international conference on* IEEE. ss.121-128

Santurkar, S., Arora,A., Chandrasekaran,K. (2014). Stormgen - A Domain specific Language to create ad-hoc Storm Topologies. *IEEE Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on Computer Science and Information Systems.* 2014/08/07-2014/08/10. Warsawa:IEEE.ss.1621–1628

White papers

Feddersen, C. (2015). *Real-Time Event Processing with Microsoft Azure Stream Analytics.* Microsoft Corporation.

Highleyman, B., Holenstein, P.J., Holenstein, B.D. (2013). *The Evolution of Real-Time Business Intelligence and How To Achieve It Using Gravic Shadowbase.* Malvern PA*:* Gravic, Inc. Hämtad 2015/04/10 från http://www.gravic.com/shadowbase/pdf/white-papers/The%20Evolution%20of%20Real-Time%20Business%20Intelligence.pdf

Böcker

Rajesh, N. (2013). *HDInsight Essentials.* Packt publishing. ISBN:9781849695374

Rittinghouse, J.W., Ransome, J.F. (2009). *Cloud computing: implementation, management, and security*. CRC press.

Sarkar, D. (2014). *Pro Microsoft HDInsight, Hadoop on Windows*. ISBN13: 978-1-4302-6055-4

Schaffner, J. (2013). *Multi Tenancy for Cloud-Based In-Memory Column Databases: Workload Management and Data Placement.* Springer Science & Business Media.
ISBN 978-3-319-00497-6

Internetkällor

Apache. (2014). Hämtad 2015/04/21 från https://storm.apache.org/

Apache Hadoop. (2015, 27 mars). *What is Apache Hadoop?* Hämtad 2015/04/20 från https://hadoop.apache.org/

Augustsson, T. (2013, 18 december). Nästa stora steg för internet. *Svenska Dagbladet.* Hämtad 2015/04/09 från http://www.svd.se/naringsliv/digitalt/nasta-stora-steg-for-internet_8830082.svd

Burgess, RJ. (2013, 10 Juni). The Difference between Multi and Single-Tenant SaaS Architecture. Hämtad 2015/05/24 från http://blog.goiwx.com/blog/bid/296154/The-Difference-between-Multi-and-Single-Tenant-SaaS-Architecture

Chen, X. (2014). Storm: Debugging topology running in storm cluster.[Blogginlägg]. Hämtad 2015/05/12 från http://czcodezone.blogspot.se/2014/12/storm-debugging-topology-running-in.html

Damaggio, E. (2015). *Get started with Event Hubs.* Hämtad 2015/05/09 från http://azure.microsoft.com/sv-se/documentation/articles/service-bus-event-hubs-csharp-ephcs-getstarted/

Evans, K. (2015, 26 februari). Using Stream Analytics with Event Hubs [Blogginlägg]. Hämtad 2015/05/04 från http://blogs.msdn.com/b/kaevans/archive/2015/02/26/using-stream-analytics-with-event-hubs.aspx

Franks, L. (2015a). *Develop C# topologies for Apache Storm on HDInsight using Hadoop tools for Visual Studio.* Hämtad 2015/05/12 från http://azure.microsoft.com/sv-se/documentation/articles/hdinsight-storm-develop-csharp-visual-studio-topology/

Franks, L. (2015b). *Process events from Azure Event Hubs with Storm on HDInsight (C#).* Hämtad 2015/05/12 från http://azure.microsoft.com/sv-se/documentation/articles/hdinsight-storm-develop-csharp-event-hub-topology/

Franks, L. (2015c). *Storm on HDInsight overview.* Hämtad 2015/04/21 från http://azure.microsoft.com/sv-se/documentation/articles/hdinsight-storm-overview/

Gaudin, S. (2014, 26 juni). Computerworld. *Google focused on big data, real-time analysis in the cloud.* Hämtad 2015/04/10 från http://www.computerworld.com/article/2491340/big-data/google-focused-on-big-data--real-time-analysis-in-the-cloud.html

Guthrie, S. (2015, 18 Feb). Azure: Machine Learning Service, Hadoop Storm, Cluster Scaling, Linux Support, Site Recovery and More [Blogginlägg]. *ScottGu's Blog*. Hämtad 2015/06/15 från http://weblogs.asp.net/scottgu/azure-machine-learning-service-hadoop-storm-cluster-scaling-linux-support-site-recovery-and-more

Hadoop. (2015). *Welcome to Apache Hadoop! What Is Apache Hadoop?* Hämtad 2015/06/09 från http://hadoop.apache.org/index.html

ISO OBP (Online Browsing Platform). (1998). *ISO 9241-11:1998*. Hämtad 2015/06/09 från https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en

Johnson, D. (2013). Single-Tenant vs Multi-Tenant Cloud ERP. Hämtad 2015/06/11 från http://cloudtweaks.com/2013/01/single-tenant-vs-multi-tenant-cloud-erp/

Microsoft Azure. (2015a). *Vad är Microsoft Azure.* Hämtad 2015/04/13 från http://azure.microsoft.com/sv-se/overview/what-is-azure/

Microsoft Azure. (2015b). *HDInsight.* Hämtad 2015/04/21 från http://azure.microsoft.com/sv-se/services/hdinsight/

Microsoft Azure. (2015c). *Händelsenav.* Hämtad 2015/05/26 från http://azure.microsoft.com/sv-se/services/event-hubs/

Microsoft Azure. (2015d). *Try Microsoft Azure*. Hämtad 2015/05/30 från https://www.microsoftazurepass.com/azureu

Morgan, J. (2014, 13 maj). A Simple Explanation Of 'The Internet Of Things'. Hämtad 2015/04/09 från http://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/

Rouse, M. (2012). S*ingle-tenancy, techtarget*. Hämtad 2015/05/20 från http://searchcloudapplications.techtarget.com/definition/single-tenancy

Sirosh, J. (2015, 16 april). Announcing the General Availability of Azure Stream Analytics [Blogginlägg]. Hämtad 2015/04/21 från http://blogs.technet.com/b/machinelearning/archive/2015/04/16/announcing-the-general-availability-of-azure-stream-analytics.aspx

Smith, K. (2013). In the Cloud: Multi-Tenant vs. Single Tenant ITSM. Hämtad 2015/06/11 från http://apmdigest.com/in-the-cloud-multi-tenant-vs-single-tenant-itsm

SQL server team. (2015). The Ins and Outs of Azure Stream Analytics – Real-Time Event Processing. [Blogginlägg]. Hämtad 2015/05/24 från http://blogs.technet.com/b/dataplatforminsider/archive/2014/10/30/the-ins-and-outs-of-azure-stream-analytics-real-time-event-processing.aspx

Stackoverflow. (2011). *Can I use multiple "with"?.* Hämtad 2015/05/18 från http://stackoverflow.com/questions/5375634/can-i-use-multiple-with

Stokes, J. (2015). *Introduction to Azure Stream Analytics.* Hämtad 2015/04/22 från http://azure.microsoft.com/en-us/documentation/articles/stream-analytics-introduction/

Vanhoutte, S.(2015, 21 Jan) Azure Stream Analytics, scenarios and introduction. Hämtad 2015/05/04 från http://www.codit.eu/blog/2015/01/azure-stream-analytics-getting-started/

Vijayan, J. (2015, 23 feb). Streaming Analytics: Business Value From Real-Time Data. *Datamation.* Hämtad 2015/04/09 från http://www.datamation.com/data-center/streaming-analytics-business-value-from-real-time-data.html

Wikipedia. (2015a). *Software release life cycle.* Hämtad 2015/04/13 från http://en.wikipedia.org/wiki/Software_release_life_cycle

Wikipedia. (2015b). *Användbarhet.* Hämtad 2015/04/11 från http://sv.wikipedia.org/wiki/Anv%C3%A4ndbarhet

Wikipedia. (2015c). *Artefakt.* Hämtad 2015/06/09 från http://sv.wikipedia.org/wiki/Artefakt

Wikipedia. (2015d). *Multitenancy.* Hämtad 2015/06/11 från http://en.wikipedia.org/wiki/Multitenancy

Wikipedia. (2015e). *Structured_Query_Language.* Hämtad 2015/06/11 från https://sv.wikipedia.org/wiki/Structured_Query_Language

Metodik

Brooke, J. (1996). SUS - A quick and dirty usability scale. *Usability Evaluation In Industry,* CRC Press, 11 juni 1996

Gatsou, C. Politit, A. Zevgolis, D. (2013). Exploring inexperienced user performance of a mobile tablet application through usability testing. *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems.* IEEE. pp.557–564

March, S., Smith, J. (1995). Design and natural science research on information technology. Decision Support Systems, 15.ss.251-266

Oates, B J. (2006). *Researching information systems and computing*. SAGE publications Ltd.

Sauro, J. (2004). Premium Usability: Getting the Discount without Paying the Price *in ACM Interactions Volume 11, July-August.*

Sauro, J. & Kindlund E. (2005) "How long Should a Task Take? Identifying Specification Limits for Task Times in Usability Tests" *in Proceeding of the Human Computer Interaction International Conference (HCII 2005), Las Vegas, USA*

Bilagor

Bilaga 1 - SUS-formulär

Storm		Scenario 1	Scenario 2	Scenario 3 & 4	Scenario 5
1	Jag tror att jag skulle använda den här tjänsten ofta	2	2	2	2
2	Jag tyckte att tjänsten var onödigt krånglig	0,5	2	3	3
3	Jag tyckte att tjänsten var enkel att använda	1	2	3,5	2,5
4	Jag tror att jag skulle behöva hjälp av en tekniskt kunnig person för att kunna använda tjänsten	1,5	2	3	3
5	Jag tycker att tjänstens olika funktioner var väl integrerade.	1	2	3	2
6	Jag tycker att tjänsten var inkonsekvent	2	2	3	3
7	Jag tror att de flesta människor skulle lära sig tjänsten väldigt snabbt	1	2	3	2
8	Jag tycker att tjänsten kändes klumpig	2	1,5	3,5	3
9	Jag kände mig bekväm i användandet av tjänsten	1	2	3,5	3
10	Jag behövde lära mig flera saker innan jag kunde börja använda tjänsten	0,5	2,5	3	3
SUS-poäng per scenario		31,25	50	76,25	66,25

SUS-poäng i snitt	**56**

	Azure Stream Analytics	Scenario 1	Scenario 2	Scenario 3&4	Scenario 5
1	Jag tror att jag skulle använda den här tjänsten ofta	2	2	2	2
2	Jag tyckte att tjänsten var onödigt krånglig	3	2,5	4	1,5
3	Jag tyckte att tjänsten var enkel att använda	4	2,5	4	2,5
4	Jag tror att jag skulle behöva hjälp av en tekniskt kunnig person för att kunna använda tjänsten	2,5	1	3,5	1,5
5	Jag tycker att tjänstens olika funktioner var väl integrerade.	3,5	2	3	2,5
6	Jag tycker att tjänsten var inkonsekvent	2,5	2,5	3	2
7	Jag tror att de flesta människor skulle lära sig tjänsten väldigt snabbt	3,5	2	3	2,5
8	Jag tycker att tjänsten kändes klumpig	3,5	2,5	3,5	2
9	Jag kände mig bekväm i användandet av tjänsten	3,5	3	4	2,5
10	Jag behövde lära mig flera saker innan jag kunde börja använda tjänsten	2	1,5	3	2
SUS-poäng per scenario		75	53,75	82,5	52,5

SUS-poäng i snitt	**66**

Bilaga 2 - Testordning scenarier och delmål

Scenario 1	Delmål
ASA	A
HDIS	A
HDIS	B
ASA	B
ASA	C
HDIS	C
HDIS	D
ASA	D

Scenario 2	Delmål
HDIS	A
ASA	A
ASA	B
HDIS	B
HDIS	C
ASA	C

Scenario 3	Delmål
ASA	A
HDIS	A
HDIS	B
ASA	B

Scenario 4	Delmål
HDIS	A
ASA	A
ASA	B
HDIS	B
HDIS	C
ASA	C
ASA	D
HDIS	D

Scenario 5	Delmål
ASA	A
HDIS	A
HDIS	B
ASA	B
ASA	C
HDIS	C
HDIS	D
ASA	D

Bilaga 3 - Azure Stream Analytics källkod

Delmål D

```
WITH slowcars AS(Select
A.roadid,
Count(flat.ArrayValue.speed) as slowc
FROM eventhubinput A
CROSS APPLY GetElements(A.Carlist) AS flat
JOIN speedlimit S
ON A.RoadId = S.roadid
WHERE flat.ArrayValue.Speed <= S.speed - 20
GROUP BY A.roadid, SlidingWindow(second, 15)),
allcars AS(
Select
A.roadid,
COUNT(*) as allc
FROM eventhubinput A
CROSS APPLY GetElements(A.Carlist) AS flat
GROUP BY A.roadid, SlidingWindow(second, 15)
)

select
'Varning, köer nära förestående' as meddelande,
A.roadid,
cast(B.slowc as float)/cast(A.allc as float)*100 as procent
into eventhuboutput
from allcars A
JOIN slowcars B
ON DATEDIFF (second, A, B) BETWEEN 0 AND 0
WHERE B.slowc <= A.allc AND cast(B.slowc as float)/cast(A.allc as float)*100
>=80
```

Bilaga 4 - HDInsight/Storm källkod

```
Scenario 5
Delmål D

public void Execute(SCPTuple tuple)
{
      _sqlConn = OpenConnSql();
      string eventValue = (string)tuple.GetValue(0);

      if (eventValue != null)
      {

            _kamera = _json_serializer.Deserialize<Camera>(eventValue);

            int speedlimit = 0;
            SqlCommand cmd = new SqlCommand("SELECT maxspeed FROM roads where
roadId = '" +
            _kamera.RoadId + "' ", _sqlConn);
            SqlDataReader reader = cmd.ExecuteReader();

            if (reader.Read())
            {
                  speedlimit = reader.GetInt32(0);
            }
            reader.Close();

            foreach (var car in _kamera.CarList)
            {
                  SqlCommand insertAllCars = new SqlCommand(
                     "INSERT INTO countcars (regnr, time, roadid, speed) " +
                     "VALUES ('" + car.Regnr + "', '" + _kamera.Time + "','" +
                     _kamera.RoadId + "','" + car.Speed + "')", _sqlConn);
                  insertAllCars.ExecuteNonQuery();
            }

            DateTime camtime = Convert.ToDateTime(_kamera.Time);
            DateTime timelimit = camtime.AddSeconds(-15);

            SqlCommand deleteAllCars = new SqlCommand(
                  "DELETE FROM countcars " +
                  "WHERE time < '" + timelimit + "'", _sqlConn);
            deleteAllCars.ExecuteNonQuery();

            SqlCommand sqlSelect = new SqlCommand(
            "select COUNT(*) from countcars WHERE speed <= " +
            speedlimit + " - 20 AND roadid = '"+_kamera.RoadId+"'", _sqlConn);
            SqlDataReader readerCount = sqlSelect.ExecuteReader();

            readerCount.Read();
            int countCars = readerCount.GetInt32(0);
            readerCount.Close();

            SqlCommand sqlSelectAllCars = new SqlCommand(
            "select COUNT(*) from countcars WHERE roadid = '" +
            _kamera.RoadId + "'", _sqlConn);
            SqlDataReader readerAllCarsCount =
sqlSelectAllCars.ExecuteReader();

            readerAllCarsCount.Read();

            int countAllCars = readerAllCarsCount.GetInt32(0);
```

```
            double dCountAllCars = System.Convert.ToDouble(countAllCars);
            double dCountCars = System.Convert.ToDouble(countCars);
            double procent = Math.Round(((dCountCars/dCountAllCars)*100), 1);

            if (procent>=80)
            {
                  _ctx.Emit(new Values(
                  "slow: " + countCars +
                  ",All: " + countAllCars +
                  "  Procent: " + procent +
                  "%, Varning på väg: " + _kamera.RoadId));
            }

            _sqlConn.Close();

      }
}
```

Bilaga 5 - Azure Stream Analytics fältnoteringar

Scenario 1

Delmål A

Vi följde guiden som fanns i azureportalen innehållande fem steg om hur man skapar ett ASA arbete, vi valde blob storage som output enligt Vanhoutte (2015). Vi startade ASA arbetet men vi kunde inte se förväntat resultat från vår SELECT * fråga i vår outputblob, vi fick heller ingen feedback från ASA's inbyggda instrumentpanel som vi kunde använda för att se de data som vi skickade.

Efter att ha studerat instrumentpanelen till vår Input Eventhub vid testandet av vår färdigställda kamerasimulator så visste vi att instrumentpanelerna i Azures portal hade varierande latency och inte speglade händelserna i realtid. Eftersom ASA's instrumentpanel hade en fördröjning på flera minuter så var det svårt att avgöra om ASA tagit emot event eller inte.

Vi hade satt policy names i ASA arbetet till Send och Listen enligt den tutorial vi följde när vi skapade våra Event Hubs, vi provade att istället använda RootManagedAccessKey som verkade vara en defaultpolicy. Efter att ha ändrat input policyn så kunde vi nu se förväntat resultat i vår Blob Storage (samtliga bilar som vi skickade).

Vi insåg att vi hade nog fel arkitektur på vår PoC, i nuläget hade vi bara en eventhub som input och saknade då en output från ASA som vi kunde koppla till outputapplikationen. Vi skapade en Output Event Hub som tog emot data från ASA som skickades vidare till outputapplikationen för presentation. Eftersom tillägget av Output Event Hub ingår i förutsättningarna och inte i utvärderingen av ASA så räknade vi inte den tiden det tog att ändra arkitekturen av vår PoC.

Efter att ha genomfört delmål A så tycker vi att det var relativt enkelt att sätta upp ASA-tjänsten med tanke på att den var helt ny för oss. Det var tydligt var man skulle definiera de olika delarna: inputs, SQL-fråga och outputs i det användarvänliga gränssnittet. När vi valde t.ex. inputs så kunde man via dropdown-menyerna få hjälp med vad man kunde välja, t.ex så fanns en lista med de Event Hubs som vi redan hade skapat. I ASA kan man endast använda två typer inputs som strömmande data, Event Hubs och blob-storage.

Delmål B

För att få ut bilar som färdas på en bestämd väg (RV1 i det här fallet) lade vi till ett WHERE villkor till SQL frågan.

Delmål C

Eftersom vårt event som skickas till ASA innehåller en JSON-sträng innehållande en nästlad array med bilar så har detta medfört problem när vi har försökt att plocka ut hastighet på bilarna. Vi hittade inte någon lösning på detta i dokumentationen så därför skickade vi en fråga till Microsoft om det är möjligt att plocka ut de data som finns i "CarList". Vi fick svar och implementerade lösningen från Microsoft.

För att testa vår SQL-fråga så använde vi oss av den inbyggda testfunktionen där man kunde ladda upp en JSON- eller CSV- fil och testa SQL-frågan mot denna. Tesfunktionen sparade oss tid då vi nu slapp starta ASA-arbetet för att testköra mot strömmande data. Det gick att ladda upp filer som representerade både referensdata och inputströmmar. Att starta ett ASA-arbete tog ca 1,5 minut.

Nu ville vi jämföra tiden från varje bil mot referensdata som ligger sparat i en SQL-databas. Men efter att ha läst dokumentationen kunde vi konstatera att det inte gick. ASA kunde endast använda blob storage som referensdata i filformaten JSON eller CSV). Vi lade in våra vägar och deras maxhastighet i blob storage i form en JSON fil istället för en SQL-databas. Vi valde JSON eftersom vi hanterar det formatet på andra ställen. Sedan modifierade vi SQL frågan med hjälp av dokumentation på Azure så vi kunde matcha dataströmmen mot referensdata i vårt blob storage och plocka ut bilar som körde över tillåten hastighet.

Delmål D

Lade SQL-satsen från delmål C i en WITH-tabell och använde två SELECT på WITH-tabellen. en SELECT till Output Event Hub och en till Output SQL-databas.

Scenario 2

Delmål A

Vi läste i Azures dokumentation och hittade aggregatfunktionen COUNT, funktionen skulle användas tillsammans med någon form av window funktion för att bestämma inom vilken tidsram som man vill räkna. En av dessa var slidingwindow där man kunde välja att räkna antalet bilar inom en viss tidsram som ständigt är i rörelse, och man utgår alltid ifrån aktuell tid. När vi använde aggregatfunktioner med tidsfönster så fick vi dock inte ut något resultat i testfunktionen.

Delmål B

För att räkna antalet bilar som passerat vid varje specifik väg så använde vi COUNTfunktionen tillsammans med GROUP BY för att kategorisera antal bilar vid varje väg.

Delmål C

Eftersom vi skulle räkna ut en procentsats av trafikflödet för varje väg så behövde vi både det totala antalet bilar och antalet bilar per väg. Det visade sig att det var väldigt svårt att få ut båda dessa siffror i samma fråga, vilket behövdes för att kunna räkna ut en procentsats. Vi insåg ganska snabbt att vi behövde två tillfälliga tabeller med två olika SELECT-satser, en för totala antalet bilar och en för totalt antal bilar per väg, så vi försökte skapa två WITH-tabeller. Vi kunde inte hitta några exempel i dokumentationen om hur man skapade flera WITH-tabeller i ett ASA arbete så vi trodde inte att detta var möjligt. Vi hittade senare ett inlägg på stackoverflow (2015) om hur man kunde skapa flera WITH-tabeller. Nu kunde vi med viss felmarginal se totala antalet bilar samt totala antalet bilar per väg, men utskriften till konsolapplikationen blev otydlig då resultaten blev duplicerade.

Den maxtid vi hade bestämt att vi skulle lägga på varje delmål gick ut och vi kunde inte slutföra uppgiften.

Scenario 3

Delmål A

Detta delmål gick snabbt, det påminde om det delmål där vi hämtade tillåten hastighet baserat på väg-ID och därför visste vi hur vi skulle lösa uppgiften. Vi kollade av om registreringsnumren som passerade kamerorna fanns med i vår referensdata innehållande efterlysa bilar.

Delmål B

För att kunna spara de efterlysa bilarna i en SQL-databas så behövde vi göra två SELECT-satser, vi skapade därför en WITH-tabell. Den första SELECT-satsen visar den efterlysta bilen i outputapplikationen och den andra sparar den efterlysta bilen i SQL-databasen.

Scenario 4

Delmål A

Vi summerade alla hastigheter med hjälp av slidingwindow och aggregatfunktionen SUM.

Delmål B

Vi beräknar medelhastighet med hjälp av slidingwindow och aggregatfunktionen AVG.

Delmål C

Vi beräknar summan av samtliga bilars hastigheter per väg med hjälp av slidingwindow och aggregatfunktionen SUM, vi kategoriserar på väg-ID med funktionen GROUP BY.

Delmål D

Vi beräknar medelhastighet med hjälp av slidingwindow och aggregatfunktionen AVG, vi kategoriserar på väg-ID med funktionen GROUP BY.

Scenario 5

Delmål A

Vi gör en JOIN på vår referensdata som innehåller tillåtna hastigheter och sedan har vi ett WHERE-villkor där vi plockar ut de bilar som minst kör 20km/h långsammare än tillåten hastighet.

Delmål B

Vi lägger till COUNT och Slidingwindow för att räkna alla bilar inom 15 sekunder.

Delmål C

Vi skapade först två WITH-tabeller, en som räknade alla bilar inom 15 sekunder och en som räknade bilar som körde minst 20km/h under tillåten hastighet. Sedan skapade vi en SELECT-sats med JOIN på båda WITH-tabellerna. Det uppstod nu ett problem med att vi inte kunde se något resultat i outputapplikationen om alla bilar körde enligt tillåten hastighet. Det var först när någon bil körde 20/h långsammare som ASA skickade data till outputapplikationen. Detta berodde på att när vi hade ett WHERE-villkor på COUNT-funktionen så fick inte SELECT-satsen något värde. vi försökte ge COUNT-funktionen ett defaultvärde när den inte har något att räkna men lyckades inte. Vi försökte använda T-SQL funktionerna ISNULL och COALESCE för att ge COUNT ett defaultvärde, men ingen av dessa stöds av ASA enligt våra tester.

Vi noterade även att resultaten inte alltid var konsekventa, antalet bilar som körde långsamt kunde ibland vara fler än totala antalet bilar, vi löste det genom att lägga till en WHERE

[slowcars] <= [allcars] på SELECT-satsen, men fortfarande fick vi dubbel data då och då, men annars fungerade det.

Delmål D

Vi lade till ett WHERE villkor som anger att när fler än 80% av bilarna på en väg körde 20km/h långsammare än tillåten hastighet, så skickas en varning till outputapplikationen.

Bilaga 6 - HDInsight/Storm fältnoteringar

Scenario 1

Delmål A

Eftersom vi tidigt insåg att HDIS bestod av många komponenter som var nya för oss så behövde vi hitta en tutorial som började från grunden. Att börja utveckla topologier som kunde hämta och skicka data via Event hubs var för komplicerat för oss att börja med, därför kunde vi inte direkt starta med scenario 1 delmål A.

Vi började därför med att följa en tutorial på ett projekt som gick ut på att räkna ord. Projektet innehöll färdiga klasser för att utveckla egna topologier. I tutorialen fanns färdig kod som behövdes för att få igång projektet. Koden som fanns i tutorialen var bristfälligt kommenterad och därför så behövde vi lägga mycket tid på att försöka förstå koden. (Franks, 2015)

Nästa steg var att ladda upp projektet till stormklustret via Visual Studio. När projektet var uppladdat så startade analystjänsten. Sedan kunde man via Storms instrumentpanel se information om varje klass (spouts, bolts). Informationen som presenterade via instrumentpanelen var svår för oss att förstå då det fanns väldigt lite förklaringar om vad denna skulle användas till. Via instrumentpanelen kunde vi se att analystjänsten arbetade men vi kunde inte se något förväntat resultat i form av hur många ord som räknades. Instrumentpanelen hade en fördröjning på några sekunder innan man kunde se om något hände.

Det var först efter att vi gjort en klass som testade topologin lokalt som vi kunde se en loggfil som visande ett förväntat resultat. Efter att vi nu fått någorlunda grepp om hur topologin fungerar gick vi vidare till en tutorial som inkluderade Event Hubs.

I nästa tutorial (Franks, 2015b) började vi om från början och skapade två nya Stormprojekt. Det ena projektet hade till uppgift att generera data (Eventhubwriter) och skicka denna data till en Event Hub. Det andra projektet (Eventhubreader) hade till uppgift att ta emot data från samma Event Hub och lägga resultatet i en tabell.

Efter att vi hade fått projekten i tutorialen att nästan fungera felfritt så började vi bygga om koden för att passa in i vårt arbete. Det som inte fungerade var att Eventhubreader inte lade någon data i tabellen, även om tabellen skapades. Detta tog upp en stor del av tiden eftersom det var svårt att se om data flödade som den skulle i övrigt, eftersom vi inte skulle ha någon tabell i det här delmålet så beslutade vi att bygga om den ändå till slut.

Vi skapade ett nytt projekt där vi slog ihop logiken från Evenhubreader och Eventhubwriter, vår applikation skulle både läsa och skriva till två olika Event Hubs. Det visade sig att man var tvungen att använda Javaklasser för att kunna hämta och skicka data till Event Hubs. Eventhubwriter skrev till Event Huben med hjälp av en Javabolt, en Javaklass som hanterar uppkoppling mot Event Hub (Eventhubwritern hade för övrigt en C#spout). Eventhubreadern i sin tur använde en Javaspout som tog emot data från eventhuben (Eventhubwritern hade för övrigt en C#bolt). Vi löste detta genom att låta en Javaspout ta emot data från vår input Event Hub och sedan använda en Javabolt för att skicka ut samma data till vår output Event Hub.

Delmål B

Under delmål A så behövdes ingen C#bolt för att analysera data, men under detta delmål så behövdes C#bolten för att plocka ut bilar som färdades på en specifik väg. Det var svårt att förstå hur vi skulle skapa kopplingen mellan Javaspout, C#bolt och Javabolt så att data kan flöda mellan dessa.

Först fick vi problem med att få igenom några data överhuvudtaget. Vi kunde inte heller testa topologin lokalt längre eftersom vi inte visste hur man gjorde detta med Javabolts och Javaspouts, tutorial om detta saknades. Det var tidskrävande att behöva ladda upp topologin och köra den i stormklustret varje gång vi behövde testa topologin. När vi väl fick igenom data såg den inte ut som förväntat men det verkade som att kopplingarna och flödet av data fungerade. Vi hade stora problem med att hitta felet till varför vi inte fick ut det resultat vi förväntade oss eftersom vi inte kunde felsöka koden och därmed kunde vi inte hitta vart i flödet felet uppstod. Eftersom det finns väldigt lite exempel på internet om hur man bygger Stormtopologier i C# så fanns det heller inte mycket hjälp att få.

Vändpunkten kom när vi hittade en bloggpost (Chen, 2014) där det stod att man kunde använda sig av vanliga konsolutskrifter och sedan hitta utskrifterna i loggfiler via instrumentpanelen. Nu kunde vi skriva ut data som vi fick in i C#bolten och upptäckte då att data från Javaspouten var korrekt. Eftersom vi nu kunde konstatera att data från Java till C# var korrekt, men data från C# till java blev inte som förväntat, så kunde vi anta att det kunde ha med serializer/deserializer att göra. Vi hade en seriallizer redan, då testade vi att deklarera en deseriallizer för att översätta de data från C#bolten så att de skulle kunna läsas av Javabolten.

Nu kunde vi alltså börja analysera data i C#bolten. Eftersom data som kommer in är en JSON-sträng så ville vi först objektifiera den till ett kameraobjekt så att vi lättare kan utföra analysen.

Vi noterade genom Azureportalen att Stormklustret kostade pengar även om vi inte använde det. Klustret gick heller inte att stänga av utan vi var tvungna att ta bort det helt.

Delmål C

Eftersom det är tidskrävande att ladda upp en topologi i Stormklustret för att testa om kod och logik är korrekt så började vi nu parallellt att använda oss av en konsolapplikation. I konsolapplikationen testade vi Stormkod lokalt i den mån det var möjligt vilket sparade tid. Tiden det tog att ladda upp varierade något men det tog ca 2,5 minut att ladda upp och köra en topologi.

När Excecutemotden tar emot ett kameraevent så gör vi en fråga mot databasen som hämtar upp max tillåten hastighet baserat på väg-ID. Sedan kollar vi av om det är någon av bilarna i Carlist som kör över den tillåtna hastigheten. Vi skapar en ny JSONsträng av kameraobjektet som bara innehåller bilar som kört för fort och skickar den till outputapplikationen bara när det finns någon som kör för fort.

Delmål D

Eftersom att det visade sig att ASA inte kunde hantera referensdata från en SQL-databas utan bara kunde använda blob-storage, så ändrade vi förutsättningarna för ASA testerna. Därför ville vi undersöka om Storm kunde läsa data från blob-storage för att ta reda på om detta var en begränsning endast hos ASA. Det visade sig att Storm kunde hantera både SQL-databaser och blob-storage. Detta test gjordes fristående från övriga delmål och togs därför inte med i tidsberäkningen av delmål D. Vi valde ändå att fortsätta Stormtesterna med SQL-databas eftersom vi tyckte att det var mer logiskt att arbeta med.

Vi skapade en SQL-databas där vi sparade de bilar som körde över tillåten hastighet.

Scenario 2

Delmål A

Vi började med att fundera ut hur vi skulle lösa uppgiften logiskt. Vi skulle ha en räknare som håller koll på tiden och räkna antal bilar under den senaste minuten. Vi löste det genom att skapa en SQL- databas där vi lade ned alla bilar som passerade och tog bort de som var äldre än en minut för att sedan räkna de bilar som fanns kvar.

Delmål B

Vi började med att lägga till en kolumn med väg-ID i vår SQL-tabell, sedan gjorde vi en SELECT-sats med en COUNT-funktion som vi grupperade baserat på väg-ID. Vi loopar igenom vårt resultat från SELECT-satsen och skickar antalet bilar som passerat kamerorna vid de specifika vägarna till outputapplikationen.

Delmål C

Vi började med att skapa ett nytt objekt som innehåller ett väg-ID och trafikflöde i procent som vi lade till en lista. Sedan sorterade vi listan och skickar resultatet till outputapplikationen.

Scenario 3

Delmål A

Vi hämtar in registreringsnummer på efterlysta bilar som finns i en SQL-databas. Sedan kontrollerar vi varje event om det innehåller någon efterlyst bil. Vid träff så skickar resultatet till konsolapplikationen.

Delmål B

Vi lade till en INSERT från föregående delmål för att spara funna efterlysta bilar i en SQL-databas.

Scenario 4

Delmål A

Vi lägger in alla bilar i en SQL-databas med registreringsnummer, tid, väg-ID och hastighet. Sedan tar vi bort de bilar som är äldre än en minut och summerar hela hastighetskolumnen med SUM samt skickar resultatet till outputapplikationen.

Delmål B

Samma som delmål A men vi ändrar i vår SQL-fråga från SUM till AVG för att beräkna medelhastighet på samtliga vägar under den senaste minuten.

Delmål C

Samma som delmål B men vi ändrar i vår SQL-fråga från AVG till SUM, samt kategoriserar på väg-ID med GROUP BY-funktionen för att visa totalsumman av alla hastigheter den senaste minuten för varje väg.

Delmål D

Samma som delmål C men vi ändrar från SUM till AVG för att beräkna medelhastighet för varje väg under den senaste minuten.

Scenario 5

Delmål A

Vi hämtar tillåten hastighet från SQL-databas och plockar ut de bilar som kör minst 20km/h under tillåten hastighet, ersätter bilar i kameraobjektet och skickar resultatet till outputapplikationen.

Delmål B

Nu lägger vi ner alla bilar som kör minst 20km/h långsammare än tillåten hastighet i en SQL-databas, tar bort alla som är äldre än 15 sekunder och räknar alla bilar i databasen.

Delmål C

Vi lägger ner alla bilar som passerat kameran i en SQL-databas och plockar bort de som är äldre än 15 sekunder. Sedan räknar vi bilar som kört minst 20km/h långsammare än tillåten hastighet samt räknar alla bilar inom 15 sekunder. Sedan beräknar vi i procent hur många som kört 20km/h långsammare än tillåten hastighet och skickar samtliga värden till outputapplikationen.

Delmål D

Vi hämtar ut totala antalet bilar samt bilar som kört långsamt på en specifik väg. Sedan beräknar vi i procent hur många som kört minst 20km/h långsammare än tillåten hastighet på den vägen. Till sist så skickar vi en varning till outputapplikationen om andelen långsamma bilar är högre eller lika med 80 %.