En optimeringsprocess för databasfrågor i Qiiwi Interactives produkt Backpacker -

Utveckling av en process samt verktyg för databasfrågeoptimering i Amazon Web Services Aurora kluster baserande på MySQLs Slow Query Log

Examensarbete VT 2017:
Tim Langhans
Handledare Anders Norberg
Program Java Enterprise Utvecklare
Klass Java15
YRGO – Högre Yrkesutbildning Göteborg
Alingsås, 2017-05-31

YRGO

Sammandrag

I uppdrag av företaget Qiiwi Interactive i Alingsås utvecklades en process samt processverktyg för databasfrågeoptimering för deras mobilspel Backpacker. Backpacker använder sig av en PHP-webbservice i Amazon Web Services (AWS) med en MySQL databas i AWSs Aurora kluster. Ett eget system för övervakning, analys och optimering av databasfrågor byggdes upp i AWS och kallades för Analysservice. Systemets utformning är resultat av förstudier med tekniska ansvarige och framtida användare hos uppdragsgivaren. Analysservicen bygger på MySQLs Slow Query Log som hämtas från Backpackers produktionsdatabas. Loggen analyseras sedan vidare med Perconas PT-Query-Digest för att slutligen presentera analysdatan samt resultat från olika optimeringsverktyg i webbapplikationen Anemometer av Box Open Source. I arbetet beskrivs Analysservicen och de olika anpassningarna som genomfördes för att matcha denna med uppdragsgivarens specifika krav och önskemål. Dessutom beskrivs en möjlig process för att genomföra databasfrågeoptimeringar och ett konkret exempel på identifikation och optimering av en icke optimal databasfråga ges.

Förord

Denna rapport är undertecknads examensarbete på Yrkeshögskolprogrammet Java Enterprise Utvecklare i klass Java15 hos YRGO, Högre Yrkesutbildning Göteborg. Arbetet genomfördes under vårterminen 2017 och omfattningen motsvarar 15 yrkeshögskolepoäng (YH-poäng).

Ett stort tack går till herrarna från Qiiwi Interactive i Alingsås som tog emot mig på ett trevligt sätt under min LIA-praktik och tiden av mitt exjobb. Ett speciellt tack till Marcus Dale Rundberg för all hjälp. Jag hoppas att föreliggande arbete kommer till användning i företaget och gör lite nytta i framtiden.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

1.1.1 Om uppdragsgivaren och uppdraget

Qiiwi Interactive AB är ett mindre mjukvaruföretag med sju anställda i Alingsås, Västra Götaland. Företaget utvecklar mobilspel och pedagogiska applikationer samt tillhörande driftsystem. Qiiwis senaste produkt är mobilspelet Backpacker som levereras till spelare i Sverige via Googles Play Store för Android och Apples App Store för IOS-enheter. Vid skrivande stund har spelet ungefär 180-tusend aktiva spelare. Klientapplikationerna servas i från en webbservice skriven i PHP som persisterar spel- och användardata i en MySQL-databas kluster. Infrastrukturen köps som en service av Amazon Web Services (AWS), befinner sig alltså i molnet. Vid skrivande stund överstiger spelets behov, även under de mest intensiva speltiderna, inte mer än cirka 40% av databasens prestationsförmåga, enligt Amazons övervakningssystem. I den nära framtiden ska produkten Backpacker dock erbjudas till spelare i flera länder, den långsiktiga visionen av Qiiwi Interactive är till och med i hela världen, och antalet användare förväntas därför stiger avsevärt. Naturligtvis önskar företaget inte att köpa större databas-resurser från Amazon än det är nödvändigt. I första hand önskar Qiiwi därför att optimera sin databas-användning och högst prioriteras en granskning av alla databasfrågor som ställs till systemet samt en följande optimering av icke optimala frågor; rörande otillräckliga index, förhöjd användning av processor och minne, höga svarstider, höga skriv- eller låstider och andra vanliga problem som kan uppstå i sammanhanget.

Det finns vid nuläget inte något beprövat sätt att genomföra just den delen av databas-administrationen inom företaget. Har tydliga problem med databasfrågor uppmärksammats under utvecklingen av spelet, hanterades de individuellt och “för hand” genom att skriva om frågorna eller att optimera i databasschemat. Genom Amazons verktyg finns det dessutom bara en övergripande övervakning av databasens funktion, såsom minnesanvändning, CPU-användning et cetera. Denna är dock inte lämpade för att upptäcka de detaljerade svårigheterna som enstaka, icke optimala databasfrågor kan föra med sig.

1.2 Syfte

1.2.1 Det initiala uppdraget från Qiiwi Interactive

Qiiwis initiala uppdrag och målsättning med detta arbete är att genomföra en optimering av företagets databasfrågor rörande produkten Backpacker. Uppdragsgivarna gav utförandet en mycket stor grad av frihet. Det kontinuerliga skapande av en gemensam förståelse av uppdraget och dess utformning tillsammans med uppdragsgivaren ses därför som en viktig del av själva utvecklingsprocessen i föreliggande arbete. Den första perioden av utvecklingsprocessen lades mycket tid på just analys av problemsituationen samt ovan nämnda diskussion om hur den efterfrågade optimeringsprocessen för

databasfrågor bäst kunde utformas; se gärna avsnitten 2.2 om kravanalysen samt 2.3 som ger en sammanfattning av den gemensamma förståelsen av arbetsuppdraget.

Inom ramen av uppdraget hade det varit önskvärt att behandla funktionen av hela backend-systemet, webbservice och databas. Detta för att på kunna upptäcka prestationsproblem på ett mer realistiskt och relevant sätt. En utförligt analys borde börja med en benchmarking av hela systemet, ser dess beteende under längre perioder av olika belastningsgrader vilka skapas på ett kontrollerat sätt genom att penetrera systemet med välkonstruerad testdata. Detta inte minst för att få ett basvärde som kvalitetsmarkör för framtida optimeringsinsatser (se Schwartz et al. 2012). På grund av examensarbetets begränsade omfattning samt att många relevanta faktorer av systemet inte står under företagets direkta inflytande, utan Amazon Web Services, ska detta dock bara rekommenderas för det framtida arbetet med företagets infrastruktur.

1.2.2 Frågor som uppstår ur uppdraget

Ifrån det initiala uppdraget uppstår en rad frågor som måste besvaras för att kunna utföra föreliggande arbete. Följande frågor utgjorde en del av grunden till diskussionen som fördes med uppdragsgivaren under den gemensamma utformningen av uppdraget:

- Hur identifierar man en databasfråga med behov av optimering? Vad är de bästa mätvärdena att fokusera på?

Optimering av databasfrågor kan syfta på en förbättring av frågans syntax, men även på ändringar i databasens schema eller databasens hårdvarukonfiguration. Likaså finns det en uppsjö av olika relevanta mätvärden från databasens funktion, som inte sällan står i komplexa sammanhang till varandra. Det är därför av stor vikt för föreliggande arbete att hitta en gemensam förståelse och giltiga definitioner med ansvariga hos uppdragsgivaren av vad som ska anses som en icke optimalt fungerande databasfråga och vad optimeringen kan och ska fokusera på. Denna fråga ska besvaras i avsnitt 2.3 i föreliggande arbete.

- Vilken omfattning kan arbetet förväntas få?

För att kunna bedöma vilka insatser som krävs för en databasoptimering och hur en sådan process samt processverktyg ska vara utformade, måste man få en aning om hur många databasfrågor som överhuvudtaget ställs till systemet och vad databasen ungefär är sysselsatt med. Denna fråga ska besvaras i avsnittet 2.1 senare i texten.

- Vad händer med ändrade eller nya databasfrågor, samt efter förändringar i databasschemat?

Det initiala uppdraget från Qiiwi Interactive rörde en optimering av deras databasfrågor. Det efterfrågades egentligen inte någon utveckling av en process eller speciella verktyg för detta. Det måste diskuteras om en kontinuerlig process, som är anpassad till medarbetarnas tekniska önskemål, tidsresurser och behov vore mer lönsamt och önskvärt för systemet Backpacker i längden. Detta inte minst eftersom systemet fortfarande befinner sig under en rask utveckling.

- Vilka krav och ramar ställs på en optimeringsprocess och -verktyg?

För att kunna utveckla välanpassade processverktyg och den process som uppdragsgivaren önskar, måste både uppdragsgivarens önskemål och de givna tekniska ramar analyseras och bestämmas. Denna fråga ska besvaras i avsnitt 2.2 som berör kravanalysen.

1.3 Metod

Sammanfattningsvis kan arbetsmetodiken i föreliggande arbete beskrivas som lättrörlig och iterativ; flera idéer från den agila metodiken användes utan att dock tydligt följa en agil arbetsmetodik till punkt och pricka. Fördelen med ett iterativt arbetssätt och en stark ”kundorientering” sågs i den mycket öppna uppdragsförståelsen i början av arbetet och att en stor grad av personlig anpassning av produkten till uppdragsgivarens behov och kompetenser krävdes. En stor fördel för att kunna använda en sådan metodik var att föreliggande arbete genomfördes under tiden då undertecknad befann sig som LIA-praktikant (YH-utbildning) hos uppdragsgivaren. Detta skapade en närhet till de framtida användare och möjliggjorde spontana och dagliga möten samt kontinuerligt feedback under utvecklingsprocessen. Iterationerna i utvecklingen kunde anpassas flexibelt då arbetet inte skedde i team utan bara utfördes av undertecknad. Detta ledde till snabba, meningsfulla cykler i arbetet med mycket regelbundna avstämningar med uppdragsgivaren, respektive användare. Typiska agila element som kom till insats var en produktbacklog i Trello.com, User Stories som en del av kravanalysen och planering med uppdragsgivaren, samt att arbetet skedde i iterationer med kontinuerliga avstämningar och anpassningar i direkt kontakt med uppdragsgivaren och framtida användare.

1.3.1 Intervju och utredning av Backpackers system

Undertecknad erhöll inom ramen av uppdraget lästillgång till source-koden av hela systemet och produktionsdatabasen som används i produkten Backpacker. För uppdraget relevant var främst databasschemat och webbtjänsten som styr speldatan och interaktionen med spelarna. Från sist nämnda ställs huvuddelen alla alla databasfrågor mot databasen. Det utfördes också korta intervjuer med utvecklarna i företaget om deras kunskaper och användning av olika befintliga verktyg och program. Sammanfattningsvis framkom en tydligt bild av både medarbetarnas främsta kompetenser och det befintliga systemet som den efterfrågade optimeringsprocessen behövdes matcha in.

1.3.2 User Stories

För att effektivisera kravanalysen, se en sammanfattning under avsnitt 2.2, och att få en tydligare bild av uppdragsgivarens önskemål på systemet under utveckling samlades User Stories med teknisk ansvariga och framtida användare hos företaget. User Stories är en arbetsmetodik i projekt- och mjukvaruutveckling där framtida användare och relevanta andra ombeds att beskriva önskade produktegenskaper på ett naturligt, vardagsspråkligt sätt. För mer information om och gällande definition av User Stories i föreliggande arbete se User Story 2017 i litteraturförteckningen. User Stories samlades på enkla papperskort vilket gjorde det lätt att sortera och skapa en prioriteringsordning. På grund av projektets förutsättningar blev metoden dock använt på ett något annorlunda sätt än vid många mjukvaruutvecklingsprojekt; varje Story kunde inte alltid förstås som en

exakt arbetsuppgift utan sågs mer som ett effektivt sätt att dokumentera och prioritera uppdragsgivarens önskemål och krav för att senare kunna hitta optimala lösningar.

1.3.3 Systemutveckling

I föreliggande arbete skapades ett helt system som processverktyg som kallades för Analysservice, se avsnitt 3.1, för att underlätta arbetet med optimeringsprocessen. Detta krävde olika anpassningar för att matcha företagets specifika krav och förutsättningar samt för att låta de olika berörde verktygen och programmen fungerar ihop. Infrastrukturen som driver systemet ställdes på grund av säkerhetsskäl av uppdragsgivaren i Amazon Web Services och berörs följaktligen inte i föreliggande arbete. Installationen och konfiguration av systemet på denna infrastruktur ingick dock som arbetsuppgifter.

Anpassningar av programkod skedde främst i PHP och Shell-skript och var möjligt tack vare att alla verktyg från tredje hand valdes som open source mjukvara med lämpliga licenser. Själva konfiguration och installation av systemet, ”provisioning”, skedde genom Shell-skript som skapades för föreliggande arbete och ska kopieras till och utföras på Analysservice-servern. All källkod och alla skript som behövs för att ta Analysservicen i drift finns i GitHub-repository som skapades för föreliggande arbetet och finns offentligt att tillgå, se Backpacker_Analysservice 2017 i mjukvaruförteckningen, avsnitt 6.2.

2 Förarbeten till optimeringsprocessen

I det här kapitlet ska en rad förarbeten beskrivas som skedde innan arbetet på den egentliga produkten kunde påbörjas men ansågs vara viktiga för att kunna skapa ett resultat som är av relevans för uppdragsgivaren och matchar deras behov. Arbetet med den egentliga produkten ska beskrivas senare i texten i kapitel 3. I avsnitt 2.1 ges en kort sammanfattning av ett stickprov av databasanvändningen som togs i början av föreliggande arbete. Avsnitten 2.2 och 2.3 sammanfattar resultat från den genomförda kravanalysen respektive den gemensamma uppdragsförståelsen som uppstod på grund av förarbetena.

2.1 Analys av databasanvändning och uppdragets omfattning

För att kunna planera optimeringsprocessen och förstå vad databasen egentligen arbetar med togs ett stickprov av produktionsdatabasens MySQL Slow-Query-Log, för mer information se avsnitt 3.2.1. Enligt Schwartz et al. 2012 kan det anses som tillräckligt för ett sådant stickprov att logga en timme under ”peak performance”, alltså databasens mest aktiva timme på dygnet. Illustration 2 visar utskriften av Perconas verktyg Pt-Query-Digest för den Slow-Query-Log som produktionsdatabasen samlade in under den timmen av högsta systembelastning under den aktuella veckan, klockan 21 – 22 på den 24:e april 2017. Observera att tiden i rapporten anges i ”Coordinated Universal Time” (UTC) alltså ligger 2 timmar efter.

```
1  # 4.6s user time, 50ms system time, 32.90M rss, 2.36G vsz
2  # Current date: Tue Apr 25 14:08:15 2017
3  # Hostname: Qiiwis-MacBook-Pro.local
4  # Files: mysql-slowquery.log.2017-04-24.20
5  # Overall: 22.63k total, 73 unique, 6.29 QPS, 0.02x concurrency _____
6  # Time range: 2017-04-24 19:00:02 to 20:00:01
7  # Attribute          total     min     max     avg     95%  stddev  median
8  # ============     ======= ======= ======= ======= ======= ======= =======
9  # Exec time            76s    72us      2s     3ms     2ms    40ms     1ms
10 # Lock time             2s    23us     2ms    89us   204us    54us    69us
11 # Rows sent          1.76M       0     317   81.65  313.99  112.14   11.95
12 # Rows examine       76.51M       0 470.17k   3.46k  918.49  25.19k  223.14
13 # Query size          6.71M      27   1.24k  310.80   1.04k  246.84  202.40
```

Illustration 1: Exempel av en Pt-Query-Digest rapport av MySQL Slow Query Log från en timme av hög belastning, del 1

Själva Slow-Query-Report från MySQL hade en storlek av ungefär 11MB och detta då det valdes att logga alla inkommande databasfrågor. Storleken förespråkar att det är utan vidare möjligt att konstant logga och analysera systemet på detta sätt utan att behöva oroa sig för höga resurskrav, speciellt med tanke på att processverktyg inte behöva spara själva Slow-Query-Log utan bara dess sammanfattade form som verktyget PT-Query-Digest skapar i en senare steg. Även Schwartz et al. 2012 upplever att det är ineffektivt och inte nödvändigt att läsa själva Slow-Query-Log i dess råa form, utan rekommenderar att alltid använda ett analysverktyg. Under den analyserade timmen ställdes ungefär 22600 databasfrågor, varav 73 kunde sammanfattas som unika, ”Fingerprints”, om man bortser från

deras variabla parameter; se funktionsvis av PT-Query-Digest under avsnitt 3.2.3 på sida 15. Under hela timmen var databasen enligt rapporten sammanlagt bara sysselsatt i 76 sekunder med att hantera databasfrågor, se ”Exec Time” i rad 9. I ögon fallande är också att databashanteraren undersökte fler än 60 gånger så många rader som den senare returnerade, jämför ”Rows sent” och ”Rows examine” in raderna 11 och 12 av rapporten i illustration 1. Enligt Schwartz et al. 2012 kan man förvänta sig ett ett-till-ett till ett-till-tio förhållande. Det höga förhållandet kunde tyda på att många databasfrågor rör administrativa uppgifter där genomsökningar av ett stort antal rader är vanliga eller att ett stort optimeringsbehov i databassökningarna föreligger, se också Severalnines 2015b.

```
15  # Profile
16  # Rank Query ID           Response time Calls R/Call V/M   Item
17  # ==== ================== ============= ===== ====== ===== ==========
18  #    1 0xFBD5F01B31D19A77 19.4068 25.5%    12 1.6172  0.00 SELECT use
19  #    2 0x77E39D0DE2AE9D0A  8.1717 10.7%  5766 0.0014  0.00 SELECT ach
20  #    3 0x0F986E6582CA4536  7.2838  9.6%  3603 0.0020  0.00 SELECT loc
21  #    4 0xDCC19D0B174C0768  4.7553  6.2%    60 0.0793  0.00 SELECT use
22  #    5 0x61D3EC071BDFAFE2  4.6470  6.1%    60 0.0775  0.00 SELECT use
23  #    6 0x05F1CB3F4F03D271  4.4431  5.8%    60 0.0741  0.00 SELECT use
24  #    7 0xD960D7525A9A608A  4.3014  5.7%    60 0.0717  0.00 SELECT use
25  #    8 0x305F319ADA4C2426  4.2679  5.6%    60 0.0711  0.00 SELECT use
26  #    9 0x7ADCDFF2134D2911  2.0813  2.7%  3844 0.0005  0.00 SELECT inf
27  #   10 0x47C68C1D6DEC8D5F  1.8218  2.4%     1 1.8218  0.00 SELECT ins
28  #   11 0xAF8DAC5302160628  1.6906  2.2%     8 0.2113  0.00 SELECT use
29  #   12 0xFCF833AA7DB53E8F  1.6299  2.1%     8 0.2037  0.00 SELECT use
30  #   13 0x596950D9F84DF906  1.1262  1.5%  1446 0.0008  0.00 UPDATE SEL
31  #   14 0xEDF38D27EF6AE36C  1.0990  1.4%     8 0.1374  0.00 SELECT ins
32  #   15 0x4D9995634430AF87  0.9794  1.3%   699 0.0014  0.00 UPDATE SEL
33  #   16 0x485B5F095D8CFDDF  0.9721  1.3%   166 0.0059  0.00 SELECT job
34  #   17 0x09A4491535A0F2E8  0.9150  1.2%   667 0.0014  0.00 UPDATE SEL
35  #   18 0x0207EDEF84B7BFB1  0.7780  1.0%  3167 0.0002  0.00 SELECT use
36  #   19 0x60355B5C6211D3DE  0.7756  1.0%     1 0.7756  0.00 SELECT ins
37  #   20 0xFA5E990A9AB44FD1  0.5124  0.7%     1 0.5124  0.00 SELECT pur
38  # MISC 0xMISC             4.4355  5.8%  2937 0.0015   0.0 <53 ITEMS>
```

Illustration 2: Exempel av en Pt-Query-Digest rapport av MySQL Slow Query Log från en timme av hög belastning, del 2

Med tanke på antal och omfång av de databasfrågorna som ställs på systemet blir det tydligt att det behövs en prioriteringsstrategi för vilka databasfrågegrupper som borde optimeras först för att arbeta på det effektivaste sättet, istället för att bearbeta frågorna slumpvist eller en efter en. Ur Illustration 2, som visar den andra delen av PT-Query-Digests första rapportsida, framgår det sammanfattningsvis att det är viktigt att även logga snabba databasfrågor. I den första och mest arbetstid-krävande gruppen av databasfrågor som förklarar ungefär 25% av all arbetstid i databasen, rad 18 i rapporten, bearbetas en databasfråga i genomsnittet i över 1,6 sekunder, se ”R/Call” vilket står för ”Response Time Per Call”. Just denna grupp av databasfrågor kan alltså i den egentligen meningen av ordet anses som en ”Slow Query”, en långsam databasfråga. Nästan alla övriga databasfrågegrupper körs betydligt snabbare, de utförs dock också mycket oftare och tar därför sammanlagt upp mer av databasens arbetstid än de

långsamma frågorna. Man kan därför anta att snabba databasfrågor med optimeringsbehov kan vara effektivare att prioritera i en optimering eftersom summan av arbetstiden som kan sparas in för många snabba frågor kan bli större en för några få extremt långsamma databasfrågor. Det framgår dessutom av rapporten att en relativ liten andel av alla frågegrupper upptar den största andelen av databashanterarens totala arbetstid och just sådana grupper måste processen naturligtvis fokusera på i första hand. Även Box Open Source, som tillverkar processverktyget Anemometer, rekommenderar därför, emot MySQL's Slow Query Logs ursprungliga idé, att alltid logga alla databasfrågor oberoende av deras bearbetningstid, se Towey 2012.

2.2 Uppdragsgivarens krav och önskemål

Från den gemensamma analysprocessen med medarbetare hos Qiiwi Interactive framkom önskemålet om ett system för att underlätta en kontinuerlig process av optimeringen av deras databasfrågor. Det önskades en webbapplikation som är tillgängligt för alla berörda medarbetare. Systemet skulle byggas upp i Amazon Web Services, där även resten av företagets infrastruktur befinner sig. Det fanns ett önskemål om open source produkter och att det dessutom skulle vara möjligt att anpassa verktygen enligt egna behov. Använda mjukvarorna skulle helst inte kosta något och från stora Enterprise-lösningar skulle avstås. Systemet skulle generellt vara så resurssnålt som möjligt. Viktigt för uppdragsgivaren var att verktyget även ger smidiga grafiska lösningar för att snabbt och tidseffektivt kunna övervaka databasanvändning och identifiera icke optimala databasfrågor. Förutom att datan från databasloggar ska sammanfattas på ett smidigt sätt, ska den även sparas över tid. Detta för att kunna jämföra resultat av enstaka frågor i tidslinjer för att se eventuella förändringar efter optimeringsåtgärder och dylikt. Det är viktigt att verktyget trots all dessa önskemål förblir lättförståelig och lättanvänt. Kunskaper om databasoptimering av MySQL-databaser kan dock förutsättas.

Enligt företagets önskemål ska verktyget inte få någon direkt tillgång till produktionsdatabasen eller dess MySQL-socket, utan bara använda sig av log-filer som sparas till ett externt filsystem. En läs-tillgång till produktionsdatabasen kan bara i nödfall tillåtas. Inga ändringar av databasen eller databasfrågor ska utföras av undertecknad eller på ett automatiskt sätt av något verktyg utan samförstånd med uppdragsgivaren. Infrastruktur som behövs för systemet under utveckling ställs av uppdragsgivarens systemadministratör med i högsta möjliga mån restriktiva rättigheter av säkerhets skäl.

2.3 Definitioner och ett gemensamt uppdragsförståelse

Under förarbeten till produkten genomfördes regelbundna diskussioner av nya kunskaper och delresultat med teknisk ansvariga medarbetare hos uppdragsgivaren. Dess vidare formulerades Use Cases och detaljrik information om önskemål och kompetenser hos uppdragsgivaren framkom vilket ledde till en tydligare bild av hur produkten skulle behöva utformas och enligt vilken strategi en optimering av databasfrågor kunde äga rum. Viktiga frågor som uppstod under initiala fasen av uppdraget kunde besvaras, se avsnitt 1.2.2.

- Hur identifierar man en databasfråga med optimeringsbehov? Vad är de bästa mätvärden att fokusera på?

I litteraturen framkommer inte det enda sättet att definiera en perfekt databas-prestation. Generell kan en optimering fokusera på hårdvaru-aspekter av en databas, den fysiska designen, eller på databasschema och datatyper, den logiska designen. Gemensamt har dock alla goda tillvägagångssätt att de måste föregå av en fas av analys av databasens funktion för att identifiera problem. För att göra detta på det mest effektiva sättet rekommenderar Schwartz et al. 2012 att profilera (på engelska ”profiling”) databasens funktion för att identifiera just de databasfrågorna som tar mest resurser i anspråk och dess optimering således kan förväntas ge största förbättringseffekterna. Enligt författarna brukar själva profileringen kräva den största tidsandelen i en optimeringsprocess; 90% av tiden borde planeras in för profileringen och bara 10% för själva optimeringsarbete. Enligt Schwartz et al. 2012 är **bearbetningstid per databasfråga**, alltså tiden databasen behöver för att utföra arbetet av en databasfråga, det måttet som bäst beskriver prestation, respektive resurskrav av en databasfråga. Även antalet av undersökta rader (”rows examined”) och antalet av returnerade rader (”rows returned”) är enkla men relevanta mått på hur optimalt en databasfråga kan anses hanteras. Andra vanliga mått som CPU användning, skalbarhet och så vidare beskrivs däremot som för starkt beroende av sin kontext, enligt Schwartz et al. 2012. Speciellt resursanvändning ska enligt författarna inte fokuseras på för mycket då dessa resurser, som CPU, eventuellt används för att göra databasens viktigaste uppgift bättre och snabbare. De senare versionerna av MySQL till exempel tenderar till att förbättra sin funktion genom att använda mer CPU. Uppdragsgivaren och undertecknad kom därför överens om att optimeringsprocessen ska prioritera:

1. De databasfrågorna som tar upp den största andelen av databashanterarens arbetstid; summa av bearbetningstid per databasfrågegrupp i en tidsintervall

2. Så kallade Outliers, databasfrågor som behöver extra lång tid att utföras vilket kan leda till en dålig användarupplevelse

- Vad händer med ändrade eller nya databasfrågor, samt efter förändringar på databasschemat?

Som speciellt viktigt under förarbeten, se även avsnitt 2.2 om kravanalysen, framkom att den utvecklade optimeringsprocessen ska ha en iterativ processkaraktär, som kan följa med i företagets utveckling som en kontinuerlig arbetsuppgift ”vid sidan om”, precis som övervakningen av resterande infrastruktur. Det ska vara möjligt att följa funktionen av databasfrågor över tid i form av tidslinjer, inte minst eftersom det primära måttet, bearbetningstid per databasfråga, också är beroende av kontext alltså funktion av övriga systemet och därför kan variera starkt. Med tanke på antalet utförda databasfrågor och tillgängliga relevanta mätvärden blir en snabb tolkning av resultat inte möjligt utan goda grafiska hjälpmedel. I förarbeten, se avsnitt 2.1, blev det speciellt tydligt att inte bara de långsamma databasfrågorna utan oftast även de snabba men hög frekventa databasfrågorna kan kräva lejonandelen av databashanterarens arbetstid. Följaktligen kan optimeringen bli effektivast om man inte förstår ”Slow-Query analys” bokstavligen. Uppdragsgivaren och undertecknad kom därför överens om att logga och analysera alla databasfrågor som ställs och inte bara de långsamma, vilket

visade sig vara fullt möjligt med ett externt system. Processen för en databasfråga ska vara iterativ men avslutas om vidare analys inte leder till en effektivt förbättring med hänseende till investerad tid. Schwartz el al. 2012 sammanfattar här, *”Optimization is not the same as improvement. Stop working when futher improvement is not worth the cost.”*. Med säkerhet kan tidpunkten när vidare optimering inte är värt det investerade arbetet bestämmas bäst av de tekniska medarbetarna.

3 Optimeringsprocessen

3.1 Beskrivning av hela optimeringsprocessen

I detta avsnitt ska det ges en översikt över den optimeringsprocessen som utvecklades i föreliggande arbetet. De olika stegen i processen beskrivs i mer detalj i de följande avsnitten. Där ska även förklaras hur de olika verktyg har använts och anpassats för att matcha företagets speciella behov och krav som ställdes på processen, vilka beskrevs tidigare i kapitel 2.

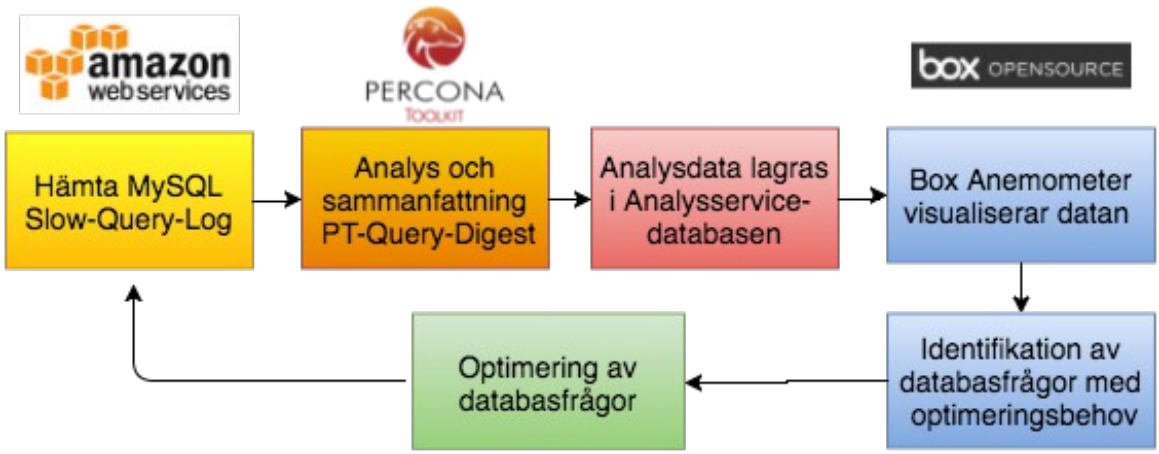

Illustration 3: Översikt över den utvecklade optimeringsprocessen för databasfrågor

Som illustration 3 visar börjar processen med att aktivera MySQLs Slow-Query-Log i produktionsdatabasen och samla in dessa rapporter i regelbundna avstånd. Loggen analyseras sedan med PT-Query-Digest dess output anpassades och sparades direkt i den databas som tillhör optimeringssystemet som byggs upp i föreliggande arbete. På detta system körs Box Anemometer som webbapplikation vilket ger den grafiska presentationen, mångfaldiga analys- och utvärderingsmöjligheter av Slow-Query data samt användargränssnittet till systemet. Även relevanta optimeringsverktyg, till exempel MySQLs EXPLAIN statement, visualiseras av Anemometer för planering av möjliga optimeringsåtgärder. De databasfrågorna som identifierats som icke optimala ska sedan optimeras i sista fasen av processen.

Det system som i det föreliggande arbetet byggs upp sköter kontinuerligt alla steg från hämtning av de Slow-Query-Log-filerna till att visualisera data och resultat från ett flertal analysverktyg inom webbapplikationen Anemometer. Detta motsvarar alla steg i övre raden av illustrationen 3. Tillhörande MySQL databas som befinner sig i AWS RDS tjänsterna persisterar dessutom datan från alla insamlade PT-Query-Digest analyser. Detta system ska i det följande kortfattad kallas för ***Analysservice***.

3.1.1 Samla analysdata – MySQL Slow Query Log

Produktionsdatabasen befinner sig i Amazon Web Services (AWS) Aurora Kluster. Enligt standardinställningarna sparas då inte Slow-Query-loggen utan bara MySQLs Error log. Slow-Query-loggen måste aktiveras i databasens inställningar hos AWS (DB parameter group) där det är även viktigt att specificera att alla databasfrågor ska loggas, som överenskommen i föreliggande arbete (DB parameter: slow_query_log = 1, long_query_time = 0). Även utskriftsformatet av loggen måste väljas explicit då datan enligt standardinställningarna går till en ny databas-tabell som skapas i själva produktionsdatabasen (DB parameter: log_output FILE), vilket bryter mot uppdragsgivarens krav att lämna produktionsdatabasen i stort sätt orört av systemet. Nu skrivs filen till ett filsystem och heter mysql.slow_log vilken skapas på roterande schema för varje timme. Det är viktigt att hämta filen regelbundet eftersom AWS bara garanterar att spara dessa log-filer i 24 timmar. All detta information kan hittas i kapitlet ”Amazon Relational Database Service – User Guide” i AWS documentation 2017.

Hämtningen av filen sköts genom ett Shell-skript, ifrån Analysservicen, se *fetch-report-and-send-to-Anemometer.sh* i arbetets GitHub-repo Analysservicen_Backpacker. Via AWS-klienten får programmet tillgång till rapporten. Det är viktigt att AWS-användaren, respektive dess tillgångsnyckel, har rättigheterna att använda säkerhetsregel *DownloadDBLogFilePortion* i Amazons säkerhetsinställningar för produktionsdatabasen, alltså att hämta databas-loggar. AWS-klienten, som är ett kommandorads-verktyg, erbjuder minst fem olika sätt att hantera ”credentials”, alltså användarens tillgångsuppgifter. I Analysservicen valdes att placera de användaruppgifterna i en textfil i *.aws/credentials* i Home directory.

Hämtning, och senare analys som beskrivs i nästa avsnitt, sköts regelbundet genom en Cron uppgift som kör *fetch-report-and-send-to-Anemometer.sh* skriptet på Analysservicen. Föreliggande system hämtar filen var femte minut, vilket leder till att det grafiska verktyget får en uppdateringsfrekvens av fem minuter. Att hämta rapporten en gång per timme måste ses som minimum för att inte få glapp i övervakningen som skulle visa sig i ”noll-linjer” i den grafiska rapporteringen i Anemometer. En personlig hämtningsfrekvens kan väljas genom att ändra parametern i skriptet *setup-cron.sh,* se GitHub-repository.

Avslutande ska sägas att sättet att hämta Slow-Query-Log filen via AWS-klienten är en anpassning som avviker starkt ifrån Box Anemometers förväntade sätt att få tillgång till Slow-Query-Log. I originalversionen av programmet använder Anemometer MySQL-socketen av produktionsdatabasen. Eftersom detta inte var acceptabelt enligt kravspecifikationerna för Analysservicen hittades detta ”work-around”. Att just hämta Slow-Query-Log och att inte använda sig av andra sätt att få tillgång till databasfrågedata berodde på att loggen kan anses som mest informationsrik och därför värt extraarbetet, jämför Severalnines 2015a.

3.1.2 Analys och sammanfattning av Slow-Query-Log data

Den hämtade Slow-Query-Log filen analyseras sedan i Analysservicen med hjälp av verktyget PT-Query-Digest av Percona. Ett exempel på den första sidan av rapportens innehåll har redan visats

tidigare i texten, se illustrationerna 2 och 1. Verktygets rapport används i det här steget dock inte i textform som i illustrationerna utan överförs direkt till Analysservicens databas. Datan från rapporten anpassas till att matcha Box Anemometers funktionsvis genom PT-Query-Digests omfångsrika optionsflagor och parameter på kommandoraden. Analysdatan delas upp i två tabeller dess schema redan har skapats i databasen i ett tidigare skede, se skriptet *setup-history-db.sh* i produktens GitHub-repository, enligt Box. En tabell sparar själva information om databasfrågegrupper, ”*Fingerprints”*, medan en annan tabell sparar data rörande varje databasfrågegrupp över tid, så kallad ”*history”*. I Analysservicen anpassas PT-Query-Digest anropet genom att använda sig av miljövariabler som sätts på host-systemet som innehåller bland annat förbindelseuppgifterna till databasen.

3.1.3 Förvaltning och presentation av analysdata

Genom att Analysservicen kontinuerligt hämta nya databas-loggar, analyserar dem vidare och spara dem i sin databas växer databasens samling av information kontinuerligt. Det ska förtydligas att databasen inte spar data för varje databasfråga som ställs utan utnyttjar det sättet som PT-Query-Digest sammanfattar databasfrågor till ”Fingerprints”, se avsnitt 3.2.3 för mer information om verktyget. Detta möjliggör att jämföra aktuella resultat för varje grupp av databasfråga med samtliga, tidigare resultat.

Box Anemometer kan hantera resultat från olika databaser, vilket om än inte använt i föreliggande arbete, eventuellt kan gagna företaget i framtiden. Dessutom kan data visas i två olika former, antingen som tabell, ”Table Search mode”, eller som en interaktiv diagram, ”Graph Search mode” se bilaga 1. Som standardinställning sorteras datan enligt databasfrågegrupp, enligt PT-Query-Digests Fingerprint, och sorteras enligt gruppens summa av bearbetningstid i den valda tidsintervallen. Sorteringen motsvarar således arbetets främsta prioriteringsstrategi för identifikation av icke optimala databasfrågor; att använda sig av bearbetningstid per databasfråga som huvudkriterium.

3.1.4 Identifiering av icke optimala databasfrågor

Med Box Anemometer är det enkelt att snabbt hitta de databasfrågegrupperna som tar mest bearbetningstid i anspråk. Speciellt tydligt blir detta i det grafiska modus. Genom att zooma in eller att välja en tidsperiod från två datepicker-element i användargränssnittet samt sedan att följa länkarna till den valda databasfrågegruppen i diagrammet eller i tabellen kommer man snabbt från en överblicksbild till en sammanfattning av alla registrerade mätvärden av just denna databasfrågegrupp. Det blir därför enkelt att jämföra aktuella resultat med tidigare vilket ger ett intryck om frågan generellt är icke optimal eller om bearbetningstiderna på vissa mätpunkter har blivit problematiska beroende på en hög last på systemet eller andra kontextproblem just då. Anemometer ger nu även direkt tillgång till analysverktyg som MySQLs EXPLAIN statement samt Visual Explain, en hierarkiskt ordnat utskrift av EXPLAIN. Även resultat från andra av de vanligaste optimeringsverktygen presenteras, se avsnitt 3.2.4 för mer detaljinformation om Anemometers användargränssnitt. Detta ger en bra startpunkt för att planera optimeringsåtgärder.

För att kunna använda Box Anemometer i Analysservicen behövdes tjänsten anpassas på flera sätt. I PHP koden av verktyget tillades systemets databas som datakälla. Filen *datasource_backpacker.inc.php* i produktens GitHub-repo tillfogades i Anemometers kod på Analysservicens webbserver och måste läggas till i Anemometers config-katalog. För att kunna använda EXPLAIN-plugin i Anemometer behövde ytterligare anpassningar i källkoden göras, för att ge Analysservicen tillåtelse att köra en EXPLAIN-databasfråga på produktionsdatabasen. Själva tillgångsuppgifter till databaserna sätts in i PHP-koden ifrån miljö-variabler som sattes på Analysservicen via skriptet *export-env-variables.sh*, enligt *environment.properties*, se Github-repot. En användare för produktionsdatabasen måste naturligtvis först skapas, med rättigheter att utföra EXPLAIN och SELECT på de relevanta tabellerna. Just nödvändigheten att kunna utföra EXPLAIN bryter mot målsättningen för föreliggande arbete att skapa ett system som inte får någon direkttillgång till produktionsdatabasen. En annan lösning kunde dock inte hittas.

3.1.5 Optimering av databasfrågorna

Det sista momentet i en optimeringsgenomgång är att optimera de nu identifierade icke optimala databasfrågorna. Det är viktigt att fastställa att Analysservicen bara förser användaren med verktyg för att identifiera möjliga svårigheter med en databasfråga. Informationen härifrån är dock varken tillräckligt exakt, fullständig eller entydig för att med säkerhet kunna bestämma en färdig optimeringsstrategi, eller till och med automatisera en optimering. Optimering av databasfrågor är snarare en kreativ och komplex process; uppgiften kan beröra ändringar i databasens fysiska schema, ändringar på tabellschema men också ändringar av själva frågorna. Användaren som vill optimera databasfrågor måste således har en god förståelse av alla de berörda aspekterna, både tekniskt samt vad en databasfråga innehållsmässigt betyder. Detta blir tydligt vid ett exempel av en databasfrågeoptimering som ges i avsnitt 3.3 senare i texten. Så länge man inte ändra i själva syntaxen av databasfrågan, vilket ändrar frågans Fingerprint, kan man alltid lätt följa enstaka frågor och se effekterna av ens optimeringsarbete över tid via databasfrågans profil i Anemometer i Analysservicen. Processens flöde i illustration 3 visar optimeringsprocessen som en sluten cirkel som går i iterationer. När en specifik databasfrågegrupp ska anses som optimerad färdig bestämmer utförande användare ifrån resultaten som återges av Anemometers olika statistik.

3.2 Tredjepartsverktyg i optimeringsprocessen

I detta avsnitt ska olika verktyg från tredje part presenteras som användes för att skapa Analysservicen, systemet som byggdes i föreliggande arbete. I respektive avsnitt ska först en kort beskrivning och överblick över det beskrivna verktyget ges. Sedan beskrivs dess uppgift i Analysservicen och varje avsnitt ska avslutas med varför just detta verktyg valdes, om valet inte kan upplevas som en självklarhet.

3.2.1 MySQL's Slow-Query-Log

Detta är en speciell logg-fil som skrivs av MySQL databashanteraren och innehåller detaljerad, intern information som uppstår under bearbetning av en databasfråga. Ursprungligen var denna logg tänkt att

analysera långsamma databasfrågor, det vill säga sådana som pågår i minst en sekund, och verktyget hade en tidsskala som baserade på hela sekunder. Sedan version 5.6 av MySQL kan även snabbare eller alla databasfrågor loggas på en finare tidsskala, baserat på mikrosekunder. Ett exempel av en Slow-Query-Log utskrift för en databasfråga är given i illustration 4. Med tanke på att över 22000 databasfrågor ställdes under den timmen som analyserades under förarbetena, jämför avsnitt 1.3.1, förstår man omfånget av denna loggfil. Som nämnd tidigare i texten kan läsning av loggen i dess råa form inte rekommenderas för optimering av databasfrågor, utan bör sammanfattas med ett relevant verktyg (Schwartz et al. 2012).

```
/rdsdbbin/oscar/bin/mysqld, Version: 5.6.10-log (MySQL Community Server (GPL)). started with:
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                 Id Command    Argument
# Time: 170403  6:00:02
# User@Host: backpacker[xxxxxxx] @  [xxx.xx.x.xx]  Id: 25xxxxxx
# Query_time: 0.797761  Lock_time: 0.000242 Rows_sent: 0  Rows_examined: 316854
use backpacker;
SET timestamp=1491199202;
SELECT uc.user_id, u.locale_id FROM users_xxxxxxx uc
INNER JOIN users u ON u.id = uc.user_id
INNER JOIN inxxxxxxxxxx i ON i.user_id = uc.user_id
WHERE u.xxxxxxxx > 30 AND (SELECT SUM(sek_value) FROM xxxxxxx p
WHERE p.user_id = u.id) > 10 AND (SELECT SUM(sek_value) FROM xxxxxx p
WHERE p.user_id = u.id) < 50 AND (SELECT MAX(created) FROM xxxxxxx p
WHERE p.user_id = u.id) < DATE_SUB(now(), INTERVAL 1 WEEK)
AND (uc.xxxxxxxxx IS NULL OR uc.xxxxxxxx < DATE_SUB(now(), INTERVAL 1 WEEK))
GROUP BY u.id;
```

Illustration 4: Avidentifierat utdrag från en MySQL Slow-Query-Log fil som visar en databasfråga

3.2.2 MySQL's EXPLAIN Statement

EXPLAIN Statement är en inbyggd funktion i MySQL databashanteraren och i stort sätt synonymt med DESCRIBE Statement. EXPLAIN kan användas för att beskriva tabeller men även för att analysera databasfrågor som använder SELECT, DELETE, INSERT, REPLACE eller UPDATE. EXPLAIN används för att visa MySQLs Query Optimizers Execution Plan, alltså hur databashanterarens interna databasfrågeoptimering planerar att sedan utföra själva databasfråga. EXPLAIN avslöjar således vilka index som är relevanta för en databasfråga och vilka av dessa slutligen används, på vilket sätt tabeller är sammanlänkade, eller hur datan blir sorterade; i minnet, via diskswap eller dylikt. Likaså framkommer information om hur många rader som måste genomsökas för att lösa databasfrågan, samt hur många datarader som returneras. EXPLAIN levererar således viktig information för att kunna besvara frågor som:

- Blev en effektiv index använd eller kunde en mer effektiv index användas?
- Är tabellerna sammanlänkade på ett effektivt sätt?
- Kan datan levereras utan kostsamma omsorteringar?
- Hämtas och genomsöks bara den datan som behövs?
- Blev en fråga optimerad jämfört med tidigare EXPLAIN utskrifter?

I föreliggande arbete blev EXPLAIN vald för att kunna identifiera optimeringsmöjligheter av en grupp av databasfrågor. I Anemometer används en plugin för att även kunna visualisera utskriften av EXPLAIN samt VISUAL EXPLAIN i en grafisk form i användargränssnittet.

3.2.3 Percona - PT-Query-Digest

PT-Query-Digest är ett kommandoradverktyg för analysen av MySQL databasfrågor från Slow-Query-Log men även General-Log, Binary-Log files, MySQL PROCESSLIST Statement och tcpdump av MySQL data; kort sagt från alla sätt som MySQL kan generera databasfrågedata. Verktyget varit känt bland databasadministratörer sedan länge och har tidigare utvecklats och distribuerats under namnet MK-Query-Digest som del av Maatkit. Man kan säga att verktyget utgör en de facto standard för sin uppgift och beskrivs ofta som ett självklart val för databasfrågeoptimering. Första sidan av en exempelrapport från PT-Query-Digest visats redan tidigare i texten i avsnitt 2.1, se illustrationerna 2 och 1. Resten av rapporten består av mer detaljerade tabeller över analysdata för varenda grupp av databasfrågor som uppgavs på rapportens första sida. Själva rapporterna i textform används inte i föreliggande arbete utan datan presenteras genom Anemometers statistik. För mer information se gärna Percona 2016 och Severalnines 2015a.

En av de viktigaste analysfunktionerna av detta verktyg är förmågan att gruppera databasfrågor som bara skiljer sig i frågans variabla parameter. Som exempel skulle två databasfrågor som kommer från en enkel SELECT … WHERE sats som en gång fråga efter viss data för användare med id = 17 och nästa gång för användare med id = 99 utgöra två olika databasfrågor. Ändå skulle de grupperas i samma databasfrågegrupp av PT-Query-Digest. I PT-Query-Digest kallas sådana liknande grupper av databasfrågor för ”Fingerprints”, vilket betecknar ett hash-värde som är resultat av en intern algoritm som omformar databasfrågan. Här tas bland annat litterala värden och mellanslag bort och strängen omformateras till gemena. Enligt standardinställningarna rapporterar verktyget först vilka databasfrågor som är de långsammaste, matchar alltså det kriteriet som valdes som viktigaste mätvärde för optimeringen i föreliggande arbete, bearbetningstid per databasfråga.

Programmet har trots dess blyga yttre som kommandoradsverktyg många inställningsmöjligheter och funktioner. Viktigt att nämna är möjligheten att spara alla unika, nya databasfrågor till en databas-tabell. Detta aktiveras med flaggan ”--review”. Flaggan ”--history” aktiverar att mätvärdena för varje unika databasfrågegrupp blir sparade till ytterligare en databastabell vilket ger möjlighet att analysera trends i datan och ser performance över tid. Verktyget Anemometer, som beskrivs i nästa avsnitt längre ner i texten, utnyttjar just dessa två databaser i hög grad. Dessutom används en annan viktigt funktion av PT-Query-Digest vilket aktiveras genom flaggan ”--filter”. Här kan nya databas-attribut skapas programmatiskt och läggas till i de skapade databastabellerna. Detta används av Anemometer bland annat för att tillfoga en kolumn som persistera databas-host-namnet varifrån mätvärdena kommit, för att kunna separera mätserier från olika databaser.

PT-Query-Digest kan anses som en standard i optimering av databasfrågor och fick därför en given plats i urvalet av verktyg för föreliggande arbete. Det är snarare så att andra verktyg har valts för att

matcha PT-Query-Digest; Anemometer kan nog upplevas som huvudverktyget i Analysservicen men hela basen av Anemometers funktioner bygger på resultat från PT-Query-Digest.

3.2.4 Box Open Source - Anemometer

Anemometer är ett open source verktyg enligt Apache Licens, version 2.0. Detta innebär att uppdragsgivarna får använda mjukvaran fritt på alla sätt, även att ändra och att sprida den. Ändringar måste i så fall bara tydligt markeras och de befintliga licens-uppgifterna måste bibehållas. Anemometer främsta uppgift är att ge en grafisk användargränssnitt för analysen och presentationen av MySQL's Slow-Query-Log. Ett exempel på användargränssnittet i Graph Search modus kan ses i bilagan 1. Namnet Anemometer baserade Box Open Source på instrumentet i en väderstation som mäter vindens hastighet. SQL databasfrågor sägs vara som en vind - flyktig och svårt att gripa.

Det som låter Anemometer stå ut från andra möjliga lösningar är dess flexibelt anpassningsbara möjligheter att rapportera och söka data, speciellt på ett grafiskt sätt. Webbgränssnittet byggdes med PHP, Bootstrap och JQuery samt med Flot för interaktiva HTML grafer vilket ger en mer flexibel och interaktiv känsla än andra open source mjukvaror som var möjliga kandidater för att uppfylla samma uppgift i föreliggande arbete. Möjligheter att samla in och förvalta data från PT-Query-Digest upplevdes likaså som mycket bra; resultat från tidigare databasfrågor kan sparas, data från olika MySQL databaser kan integreras samt att data kan grupperas, sorteras och sökas enligt alla möjliga mätvärden. Även vid en senare analys av identifierade databasfrågor erbjuder Anemometer många möjligheter; automatiskt blir en MySQL EXPLAIN Statement utfört och visualiserat, återgiven blir även CREATE Statement (databasschemat) och STATUS av berörda MySQL tabell. Det ges dessutom ett valfritt antal av de konkreta, alltså fullt parametriserade, databasfrågorna precis så som de har ställts mot databasen. Även för uppföljning och rapportering erbjuder Anemometer relativt många features, såsom möjligheten att kommentera och klassificera databasfrågor, samt att lätt integrera data i andra system eller dela rapporter genom möjlighet till JSON export, Permalänkar och en REST gränssnitt. En god introduktion till detta verktyg ges av Gavin Towey, databasadministratör hos Box Open Source, på konferensen Velocity in London 2012, se O'Reilly 2012.

Anemometer spelar en mycket central roll i föreliggande arbete och valdes sammanfattningsvis främst för dess höga matchning med uppdragets specifikationer. Verktyget är optimerad för att kunna följa samma grupper av databasfrågor över historiska mätpunkter, snabbt hitta icke optimala databasfrågor på översiktliga grafer på en mer generell plan för att sedan snabbt flytta sig direkt ner till enstaka frågors detaljer för att hitta de viktigaste informationerna för en optimering. Dessutom matchar Anemometer i hög grad kompetenserna som finns hos kunden, såsom PHP och JQuery, vilket möjliggör vidare anpassning vid behov. Kortfattat, ett system som kan växa med uppdragsgivarens framtida önskemål.

3.2.5 AWS Command Line Client

AWS klienten är ett kommandoradsverktyg som kan skapa och styra AWS produkter från kommandoraden eller Shell-skript. Med tanke på det stora antalet tjänster som AWS erbjuder samt

deras funktionsomfång förstår man hur kraftfullt verktyget är. Samtidigt levererar klienten en enhetlig och funktionell användarupplevelse. I föreliggande arbete användes AWS Command Line Client för att hämta Slow-Query rapporterna från filsystemet i produktionsdatabasen. Detta är en anpassning till arbetets kravspecifikation, eftersom en direkt tillgång till MySQL-socketen av produktionsdatabasen inte tilläts av säkerhetsskäl. Mer information om AWS kommandoradsverktyg kan hittas under kapitlet ”AWS Command Line Interface” i AWS documentation 2017.

3.3 Exempel på en komplett optimeringsgenomgång

3.3.1 Identifikation av en icke optimal databasfråga

Optimeringsgenomgången börjar med att identifiera icke optimalt fungerande databasfrågor. För det här exemplet valdes att börja processen i Box Anemometer användargränssnitt i Graph Search modus där all databasfrågeaktivitet på den andra maj 2017 ska granskas. Den resulterande grafiken som skapas av Anemometer visas i illustration 5.

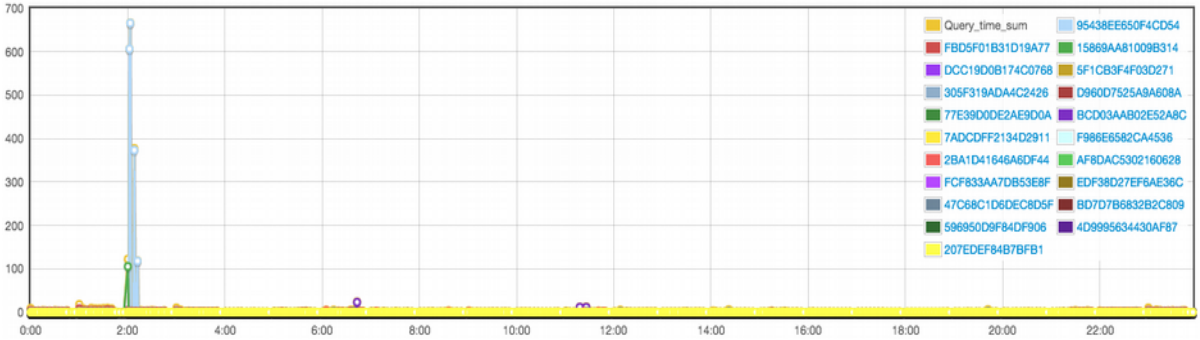

Illustration 5: Punktdiagram över summa av bearbetningstid per databasfrågegrupp under hela dygnet 2017-05-02

Mest iögonfallande är en mycket tidskrävande databasfrågegrupp, markerad i ljust blå, som bearbetas främst runt klockan 2 UTC som då sammanfattades i flera mätpunkter med bearbetningstider av över 600 sekunder, alltså över tio minuter. Med strategin att prioritera de frågorna som kräver mest bearbetningstid för att vara mest effektiv i optimeringen skulle man börja med just denna frågegrupp. En vidare analys av frågan, som dock inte ska beskrivas närmare här, resulterade dock inte i någon direkt optimeringsmöjlighet. Frågan kräver en genomsökning av stora delar av databasen i administrativt syfte och sker regelbundet. Skulle sådana administrativa uppgifter ställa till problem för funktionen av systemet kunde man replikera databasen för att köra sådana långsamma men nödvändiga uppgifter i sin egen databas så att de inte kan störa databasens funktion för Backpackers spelare. Med tanke på att databasen redan befinner sig i AWS Aurora kluster är en replikering mycket enkelt att genomföra.

Den automatiska skalningen av y-axeln i överblicksgrafiken i diagram 5 gör analysen av övriga databasfrågegrupper i stort sett omöjligt på grund av de extremvärdena vid klockan 2. Illustration 6 visar grafen för tiderna mellan klockan 04:00 och 21:20, alltså tiden som inte påverkas av de beskrivna

databasfrågorna som skapar extremvärden. Återigen är en databasfrågegrupp, markerad i röd, mycket framträdande.

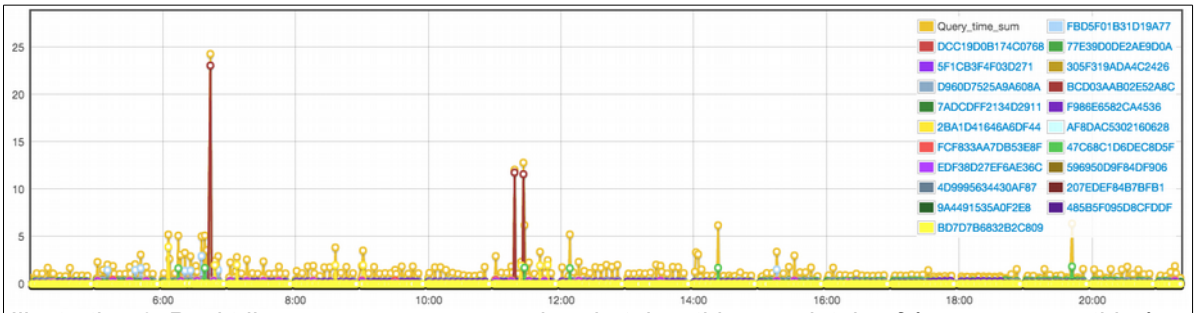

Illustration 6: Punktdiagram över summor av bearbetningstider per databasfrågegrupp över tid på 2017-05-02 mellan klockan 4:00 och 21:20

Även denna handlar om en administrativ fråga som likaså kunde köras i en extra databas. Tillsammans med grafikerna visar Anemometer också en tabell över den viktigaste statistiken för databasfrågorna som beskrivs i diagrammet, se tabell 1. Tabellen är sorterad efter summan av bearbetningstid och på plats ett, alltså mest sammanlagd bearbetningstid, ligger databasfrågegruppen med Fingerprint 77E39D0DE2AE9D0A. Denna databasfrågegruppen ska det föreliggande exemplet fokusera på och kallas för ”databasfrågegrupp under optimering” i det här avsnittet.

Tabell 1: Tabell över utvald statistik för de fem databasfrågegrupperna med den högsta summan av bearbetningstid mellan kl. 4:00 – 21:20 UTC

checksum	Query_ time_avg	ts_cnt	Query_time_sum	Lock_time_sum	Rows_sent_sum	Rows_exa-mined_sum
77E39D0DE2AE9D0A	0.00129237	45189	58.4013149	3.76644199	4757760	14273280
DCC19D0B174C0768	0.07285355	678	49.3947069	0.05106000	674	135077720
BCD03AAB02E52A8C	11.6094226	4	46.4376907	0.00121899	4	0
5F1CB3F4F03D271	0.06844431	678	46.4052441	0.04983399	995	135078041
305F319ADA4C2426	0.06590874	680	44.8179479	0.04854599	1108	135476166

Databasfrågor av denna grupp bearbetas inte iögonfallande långsamt med en genomsnittligt bearbetningstid av bara 1,3ms (se kolumn ”Query_time_avg”). Däremot utförs de relativt ofta; 45189 gånger under valda tidsperiod(”ts_cnt”). Den relativt höga totala bearbetningstiden kan inte förklaras med lås-och-vänte-problem (”Lock_time_sum”), alltså att databashanteraren väntar på sin tur i stället för att verkligen utföra det egentliga arbetet. Bara 3.8 sekunder tillbringar databasfrågorna sammanfattningsvis med att vänta på ett lås under sina sammanlagt 58 sekunder av bearbetningstid, enligt tabellen. Betraktar man antalet av raderna i databastabeller som databashanteraren genomsöker, ”Rows_examined_sum”, som ligger över 14,2 miljoner, för att sedan returnerar ungefär en tredje del 4,7 miljoner rader data, ”Rows_sent_sum”, kan man förmoda att databasfrågan eventuellt inte är helt optimal och borde analyseras vidare.

Både i tabellerna och i graferna i Anemometers gränssnitt finns det direkta länkar till en profilsida för varje berörda databasfrågegrupp. Således kan man snabbt ”förflytta sig ner” från översiktsgrafiker och

-tabeller till så kallad ”historisk” information, alltså alla hittills samlade mätvärden rörande den valda databasfrågegruppen. Illustration 7 visar ett sådant översiktsdiagram över alla mätpunkter av summa av bearbetningstid för databasfrågegruppen under optimering. Det mest i ögonfallande i grafiken är periodiska svängningar under dygnsrytmen med de lägsta bearbetningstiderna ungefär under kl 2-4 nattetid och regelbundna platåer av högsta bearbetningstider mellan 7 - 23 UTC dagtid. Detta hänger förmodligen samman med att databasfrågorna brukar ställas mindre ofta under nattetid då spelaktiviteten är lägre. En kort kontroll i statistiken visar att det ställs ungefär 1600 frågor per timme under nattimmarna och under dagtid ställs det ungefär 2800 per timme. Ovanligt med grafiken är att det regelbundet förekommer mätvärden som ligger mycket nära noll-linjen, kunde alltså utföras mycket snabbare än vid övriga tillfällen. Detta kan inte förklaras av undertecknad.

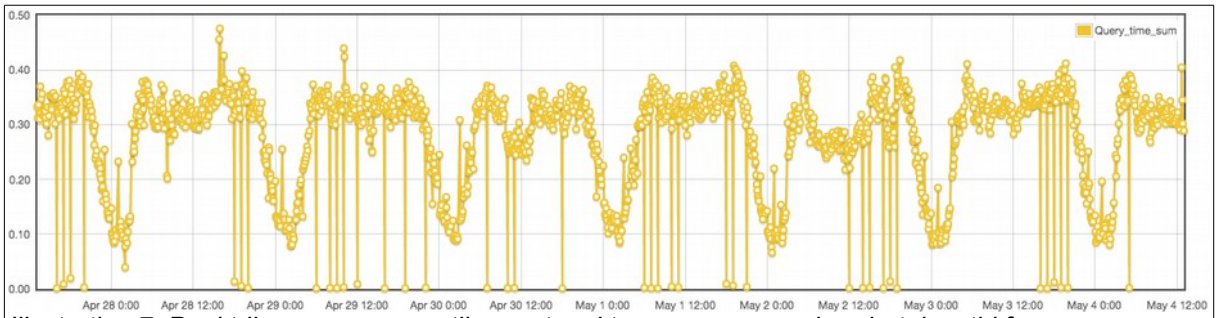

Illustration 7: Punktdiagram över samtliga mätpunkter av summa av bearbetningstid för databasfrågegruppen under optimering, 77E39D0DE2AE9D0A

Förutom överblicksgrafen visar Anemometer även konkreta databasfrågor med sina parameter ur denna grupp, precis så som de ställdes emot databasen, se text 1 längre ner i texten som visar en avidentifierad databasfråga för att kunna förstå strukturen av själva databasfrågan under optimering.

Text 1: Avidentifierat exempel av en databasfråga från databasfrågegruppen under optimering, 77E39D0DE2AE9D0A

```
SELECT
t1.id,
t1.sort_order,
t1.achievement_type,
t1.achievement_value,
t1.coins_gained,
...
t2.completed,
t3.name,
t3.description
FROM milestones t1
LEFT OUTER JOIN table2 t2 ON t2.user_id = 99999 AND t2.milestone_id = t1.id
INNER JOIN table3 t3 ON t3.milestone_id = t1.id AND t3.locale_id = 2
WHERE t1.is_active = 1
```

Jämfört med de flesta övriga databasfrågorna som ställs mot systemet är databasfrågan under optimering relativt simple. Frågan består av en enkel SELECT sats med WHERE. Datan som ska hämtas är specificerad noga, det finns inte några wildcards (”*”). Det hämtas alltså ingen onödig

information. Via två joins hämtas data från två ytterligare tabeller. Själva frågan handlar innehållsmässigt om milstenar i spelet, alltså viktiga uppdrag eller händelser som spelaren bör klara av under spelets gång. Varje gång en spelare loggar in i spelet, eller installerar den på nytt, utförs denna databasfråga för att hämta information över alla aktiva milstenar (”… WHERE a.is_active = 1”, jämför text 1). Själva datan ska sedan sparas på spelarens enhet och inte hämtas på nytt på samma dygn. Första join tjänar till att kontrollera för alla milstenar om spelaren redan har klarat av den (”t2.completed”) eller inte. Den andra joinen tjänar till för att hämta lokaliseringsinformation, alltså översättningen av milstenens beskrivning och namn på språket som spelaren spelar spelet i (”t2.name och t2.description”).

Dessutom finns det genom Anemometers gränssnitt möjligheten att se resultatet av CREATE satsen, alltså själva tabellschema, från alla av databasfrågegruppens berörda databas-tabeller samt utskriften av TABLE STATUS satsen. Likaså visas en tabell med 90 dagars historia av statistik av just denna databasfrågegrupp.

EXPLAIN utskriften från Anemometer, se tabell 2, ger ett möjligt svar på varför databasfrågan tar en stor del bearbetningstid. Trots två möjliga index som kunde användas för SELECT uppgiften, varav ”IX_is_active” även har lagts till för ändamålet, se kolumn ”possible_keys”, väljer databashanteraren att inte använda någon av dem. Den väljer istället strategin ”ALL” vilket betyder att alla databasrader i första tabellen från databasfrågan genomsökts sekventiellt. Varför indexen inte används och om själva databasfråga kunde ställas på ett bättre sätt ska svaras på under den konkreta optimeringen för denna databasfrågegrupp som beskrivs i följande avsnitt.

Tabell 2: EXPLAIN-utskrift för databasfrågegrupp under optimering, 77E39D0DE2AE9D0A

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	ALL	PRIMARY,IX_is_active				105	Using where
1	SIMPLE	ad	eq_ref	UK_ach_loc	UK_ach_loc	8	backpacker.a.id,const	1	
1	SIMPLE	ua	eq_ref	PRIMARY,IX_user	PRIMARY	8	const,backpacker.a.id	1	

De två joins till övriga tabeller verkar vara optimalt utförda. Kolumnen ”type” visar i båda fall ”eq_ref” vilket kan anses som näst bästa join-typen i MySQL. Detta betyder att kolumnen, eller i aktuella fall de två kolumnerna, som blir joinade har en index och raden i den andra tabellen kan unikt ordnas till, till exempel genom lämpliga index och en god SQL-frågesyntax. EXPLAIN utskriften visar också att det för båda joins precis förväntas returneras en enda rad, jämför kolumnen ”rows”.

3.3.2 Optimering av den icke optimala databasfrågan

Från identifieringen och första analysen av databasfrågegruppen under optimering, 77E39D0DE2AE9D0A, i förra avsnittet framgick sammanfattningsvis att frågan har en till synes optimal syntax (SELECT med WHERE), optimala joins och även optimala index. En eventuellt optimeringsbehov misstänktes eftersom inte någon av de möjliga index i SELECT... WHERE satsen används samt att det genomsöks tre gånger så många rader som det slutligen returneras.

Varför någon av indexen i huvudtabellen inte används blir tydligt med blick på innehållet i databastabellen som har 105 rader varav alla uppfyller WHERE-satsen, ”is_active = 1”. MySQLs Query Optimizer väljer då att ignorera indexen och genomföra en sekventiell sökning istället vilket brukar vara mer optimal när en stor andel rader av en databas ska hittas, se gärna ”8.3.1 How MySQL Uses Indexes” i MySQL 5.6 Reference Manual 2017.

Att precis tre gånger så många rader examineras som returneras kan likaså inte optimeras vidare och har sitt svar i sättet på hur MySQL skapar joins. MySQL väljer för de flesta av sina joins att använda en ”Nested-Loop Join Strategy”. För varje rad i första tabellen utförs direkt en join till den andra tabellen och sedan den tredje tabellen och en temporär tabell skapas som innehåller datan som direkt hämtas och kopieras över för de värden som kopplas ihop i join-processen av databashanteraren. Sedan hittas alla matchade rader som uppfyller join-satsen i sista tabellen och läggs till i den temporära tabellen. Har dessa hittats går databashanteraren ett steg tillbaka till andra tabellen och hittar alla matchande rader för detta join, sökningen genomförs alltså baklänges till alla matchande rader har hittas och samma procedur kan utföras för nästa rad från första tabellen. De joins i föreliggande exempel är optimala och bara en enda rad behöver returneras för varje join, ”type = eq_ref” se förra avsnitt. Således genomsöks för varje rad i huvudtabellen (1 + 1 + 1) rader, alltså tre gånger 105 rader, för att sedan returnera de 105 raderna som den temporära tabellen innehåller.

Anmärkningsvärt är att databasfrågan utfördes ungefär 45000 gånger under den betraktade tiden. Analysen av 90 dagars historia tabellen, som automatiskt visas av Anemometers databasfrågegrupp-profilsida, visar att mellan 50000 och 60000 frågor av denna databasfrågegrupp ställts dagligen till systemet. I motsatsen till detta fanns det under samma tid bara ungefär 35000 dagliga spelare. Databasfrågan borde inte ställas betydligt oftare än det, jämför beskrivningen av databasfrågan i förra avsnitt. Resultaten tyder följaktligen på att cachning av resultatet på klientenheten inte fungerar helt så som tänkt och bör ses över, vilket redan meddelades de teknisk ansvariga hos uppdragsgivaren.

Som den bästa optimeringslösningen för databasfrågan under optimering anses sammanfattningsvis att splittra upp frågan och arbeta mer med cachning av datan på serversidan. Det är möjligt att ta bort första join-satsen som ska hämta den spelare-specifika datan. Andra joinen hämtar den språkliga anpassning, lokaliseringen, och vid skrivande stund finns det bara fyra olika språkversioner. Det kunde således skapas fyra olika cachade versioner av databasfrågan, en för varje möjlig språkinställning i applikationens cache. Spelaren kunde sedan hämta sin specifika spelaredatan, som tidigare var den första join-satsen, direkt från andra tabellen (”table2”) enligt exemplet i text 1 och matcha in denna data i resultaten som tidigare hämtades från cachen. Själva index ”IX_Is_Active” i milsten-tabellen kunde förmodligen tas bort för att optimera schemat ännu mer, eftersom den inte kommer till användning men kräver resurser.

4 Slutsats

I föreliggande arbetet skapades en optimeringsprocess för databasfrågor, speciellt utformad för att matcha uppdragsgivarens tekniska krav och kompetenser i arbetsgruppen samt den befintliga tekniska infrastrukturen. Den huvudsakliga arbetstiden lades på att bygga upp ett centralt processverktyg, Analysservicen, som möjliggör inhämtning av analysdata, den visuella och statistiska presentation av denna samt presentation av resultat från relevanta optimeringsverktyg. Således avlastas användaren från dessa rutinuppgifter och kan direkt börja med de egentliga arbetsuppgifterna; att identifiera databasfrågor med optimeringsbehov och att utveckla hypoteser för konkreta optimeringsåtgärder.

Optimering av databasfrågor visade sig vara en betydligt mer komplex uppgift än antagen i början av föreliggande arbete. Från sökningar i litteratur och på internet framkom inte det enda, bästa sättet att ta sig an uppgiften. Det finns en uppsjö av möjliga kvalitetskriterier och mätvärden och befintliga optimeringsverktyg är ofta mycket grundläggande i deras funktion. En viktigt delmoment av föreliggande arbete var det därför att arbeta fram en gemensam uppdragsförståelse med uppdragsgivaren. För att genomföra framgångsrika frågeoptimeringar krävs det samtidigt breda kunskaper och en god förståelse rörande databasteknik, systemet som driver databasen, schemadesignen av databasen, hur databashanteraren bearbetar olika typer av SQL-frågor men även frågornas innehållsmässiga, programmatiska innebörd. Det senare förutsätter i slutändan gedigna kunskaper om hela produkten Backpacker. Databasfrågeoptimering kan således bli en nästintill kreativ uppgift som teoretiskt kan sluta med att databasfrågor komplett skrivs om eller att det bestäms att en ny hårdvara behöver införskaffas. För undertecknad hade en fullständig databasoptimering snabbt överstigit ramen av ett examensarbete. Som tur var är kompetenserna i företaget mycket goda i ovan nämnda områden, det saknades dock en tydlig optimeringsprocess för databasfrågor samt hjälpverktyg för att underlätta denna och just det skapades i föreliggande arbete. Under förarbeten, se kapitel 2, blev det tydligt att optimeringen måste ske kontinuerligt och iterativt på grund av att systemet befinner sig under en rask utveckling och för att kunna följa ändringar av databasens funktion över tid. Även under utveckligen av optimeringsprocessen visade det sig vara mycket givande och till och med nödvändigt att använda sig av en iterativ utvecklingsmetodik med regelbundna, ibland även dagliga, avstämningar med uppdragsgivaren och framtida användare. Detta just för att kunna matcha behoven som finns hos dem och således utvecklar den bäst möjliga produkten.

Processen och Analysservicen har inte kommit till användning till fullo hos uppdragsgivaren än. Det första intrycket beskrevs dock som positivt, enligt användarnas kommentarer. Dessutom uppskattades att systemet kan anpassas vid behov eftersom mjukvarulicenserna tillåter detta och tekniken valdes så att den matcha kompetenserna i företaget; PHP, MySQL, AWS et cetera. Analysservicen upplevdes överlag leda till en stor reduktion av arbetsbelastning eftersom man får ett enhetligt system som automatiserar alla nödvändiga arbetssteg för att direkt kunna börja identifiera databasfrågor med optimeringsbehov samt omedelbart få resultaten från de vanligaste optimeringsverktyg till sitt förfogande. Således kan frågeoptimeringen nästintill göras ”vid sidan om” och bli en del av databasövervakningen. Under några genomgångar av optimeringsprocessen inom ramen av

föreliggande arbete, se även exempel-optimeringen i avsnitt 3.3, framkom flera punkter som uppfattades som speciellt gynnsamma och positiva med Analysservicen och den utvecklade processen:

- Det är viktigt att kunna sammanfatta (Fingerprints) och pre-analysera log-data (PT-Query-Digest)
- Det är viktigt att kunna följa hanteringen av en databasfråga över tid, ”history-database”
- Det är viktigt att logga **alla** databasfrågor och inte bara de långsamma
- Det är viktigt med grafiska verktyg och snabba sökmöjligheter eftersom datamängden blir mycket stor över tid

Möjliga förbättringar av processen och Analysservicen som redan identifierades under utvecklingen och delvis även diskuterats med teknisk ansvariga rör:

- Möjlighet att lägga till en PHP-backtrace i själva databasfrågan

Detta skulle underlätta att hitta koden varifrån databasfrågan ställdes, till exempel när man vill ändra själva frågan inom ramen av optimeringen. Nackdelen med en sådan lösning är en viss performance overhead samtidigt som fördelarna minskas av att kodbasen är mycket väl känd av utvecklarna i företaget, vilka hittar respektive databasfråga fort ändå.

- Att genomföra en mer omfattande benchmarking

Som redan diskuterat i avsnitt 1.2.1, hade det varit optimalt att börja optimeringsprocessen med att profilera hela systemet med en väl genomfört benchmarking. Detta ska rekommenderas för det framtida arbetet med infrastrukturen.

- Ansible för enklare deployment av Analysservicen

Att konfigurera Analysservicen och att ta den i drift kunde optimeras genom att använda provisioning-verktyg som Ansible. Eftersom sådana verktyg inte ha någon förankring i företaget och bara en enda maskin skulle sättas upp avstod vi i föreliggande arbete ifrån det. Icke desto mindre kunde detta förenkla och förenhetliga det avsevärt att ta ytterligare Analysservices i drift.

- Backup av analysdatan

I föreliggande arbete skapades inte någon backup-plan för analysdatan i Analysservicens databas. Detta mestadels för att systemet snarare är en ”proof of concept” och inte används i produktionen än. Skulle detta ske i framtiden behöver dock även backuplösningar diskuteras.

5 Ordlista

Analysservice	Det valda namnet på det centrala databasoptimeringsverktyget som utvecklas i föreliggande arbete. En Amazon Linux EC2 instans samt en RDS MySQL databas som driver alla steg i optimeringsprocessen från hämtningen av databas-loggar till visualisering av data och analysresultat.
Anemometer	Open Source verktyg från företaget Box Open Source för grafisk analys av databasfrågor på bas av PT-Query-Digest analyser av MySQLs Slow-Query-Log
AWS	Amazon Web Services - Vid skrivande stund största leverantör av infrastruktur i molnet
Fingerprint	Hashcode som skapas av PT-Query-Digest för att gruppera databasfrågor. Databasfrågor som bara skiljs åt på grund av dess variabla parameter erhåller samma Fingerprint som entydigt identifierar en sådan grupp.
PT-Query-Digest	Open Source, kommandoradsverktyg för analys av MySQLs Slow-Query-Log som tillhandahålls av företaget Percona
query	databasfråga på engelska
UTC – Coordinated Universal Time	Standard för världstid sedan 1972. Motsvarar svensk tid genom att addera en timme under perioder av vinter- och två timmar under perioder av sommartid.

6 Källor

6.1 Litteratur

Litteraturförteckningen följer Harvard standard, se Umeå Universitet 2017.

AWS documentation. 2017. *Amazon Web Services*. https://aws.amazon.com/documentation/. (Hämtad 2017-03-29).

MySQL 5.6 Reference Manual. 2017. *MySQL*. https://dev.mysql.com/doc/refman/5.6/en/. (Hämtad 2017-05-22).

O'Reilly. 2012. *Gavin Towey Lightning Demo Velocity Europe 2012*. https://www.youtube.com/watch?v=YZEA00bQsq8&enablejsapi=1. (Hämtad 2017-04-20).

Percona LLC. 2016. *Percona Toolkit Documentation,* Release 2.2.20. https://learn.percona.com/manuals-success-percona-toolkit-3-0.(Hämtad 2017-03-29).

Percona. 2014. Box - Anemometer and Raingauge. *MySQL Conference and Expo 2014*. https://www.percona.com/live/mysql-conference-2014/sites/default/files/slides/Anemometer%20-%20Towey%20-%20Percona%20Live%202014.pdf. (Hämtad 2017-04-21).

Schwartz, Zaitsev, Tkachenko. 2012. *High Performance MySQL*. 3. Uppl. Cambridge: O'Reilly.

Severalnines. 2015a. Become a MySQL DBA blog series - Analyzing your SQL Workload using pt-query-digest. *Severalnines Blog*. 26 augusti. https://severalnines.com/blog/become-mysql-dba-blog-series-analyzing-your-sql-workload-using-pt-query-digest. (Hämtad 2017-05-03).

Severalnines. 2015b. Become a MySQL DBA blog series - Deep Dive SQL Workload Analysis using pt-query-digest. *Severalnines Blog*. 14 september. https://severalnines.com/blog/become-mysql-dba-blog-series-deep-dive-sql-workload-analysis-using-pt-query-digest. (Hämtad 2017-05-02).

Towey, G. 2012. Optimizing MySQL Performance at Scale with Anemometer. *Box Blogs.* https://blog.box.com/blog/optimizing-mysql-performance-at-scale-with-anemometer-2/. (Hämtad 2017-04-21).

Umeå Universitet. 2017. *Referenser Harvard*. http://www.ub.umu.se/skriva/skriva-referenser/referenser-oxford. (Hämtad 2017-05-04).

User Story. 2017. *Wikipedia*. https://en.wikipedia.org/wiki/User_story. (Hämtad 2017-05-15).

6.2 Mjukvaror

Backpacker_Analysservice. 2017. *GitHub*. https://github.com/Langhans/Backpacker_Analysservice.

Box Open Source. 2014. *Anemometer – SQL Slow Query Monitor.* https://github.com/box/Anemometer.

7 Bilagor

7.1 Bilaga 1: Box - Anemometers grafiska användargränssnitt i Graph Search modus