Storskalig CI/CD med ett Eiffel-integrerat Jenkins-plugin

Examensarbete inom högskoleprogrammet Datateknik

Lukas Danielsson
Martin Hagentoft

INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

Examensarbete 2025

Storskalig CI/CD med ett Eiffel-integrerat Jenkins-plugin

Lukas Danielsson & Martin Hagentoft

CHALMERS

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Göteborg 2025

Storskalig CI/CD
med ett Eiffel-integrerat Jenkins-plugin
Lukas Danielsson & Martin Hagentoft

Handledare: Nikolai Dahlberg, Nexer
Supervisor: Sakib Sistek, Data- och informationsteknik
Examiner: Nicholas Smallbone, Data- och informationsteknik

Examensarbete 2025
Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Cover: The messages are peacefully flowing through the CI/CD-pipeline; unstoppable, silent, and bound for a purpose!

Typeset in LaTeX
Gothenburg, Sweden 2025

Enabling Large-scale CI/CD
with an Eiffel-integrated Jenkins Plugin
Lukas Danielsson
Martin Hagentoft
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Today, companies are facing a challenge of increasing software demand and complexity. A key part of the strategy is to overcome this by using continuous integration and deployment (CI/CD). One of the most popular tools for managing CI/CD flows is the automation server, Jenkins. At the time of writing this, Jenkins lacks native support for automation via standardized event-driven job triggering using protocols such as Eiffel. This forces companies to develop their own solutions, which often lead to poorly developed software and lack of company collaboration on an enterprise level. Together with Nexer, this thesis has set the goal to lay a foundation for a solution. By enabling Jenkins to trigger work instructions in an event-driven and automated way using Eiffel events. The development process followed an iterative, test-driven approach, supported by continuous validation of prototypes. The result is a functional Jenkins plugin capable of responding to Eiffel events, a reusable configuration system for defining filter rules, and a user-friendly interface for managing them. Additionally, a standalone component was developed to receive and process incoming Eiffel events. Looking forward, the thesis proposes a dedicated orchestration service to manage multiple EiffelManager instances. This concept would enhance scalability, observability, and fault tolerance, making it suitable for large-scale CI/CD infrastructures.
Overall, the work indicates that extending Jenkins to support event-driven workflows using Eiffel is both feasible and valuable, offering a path toward more standardized and collaborative enterprise CI/CD pipelines.

Sammanfattning

För att företag idag ska kunna möta den konstant ökande efterfrågan på mjukvara används kontinuerlig integration och driftsättning (CI/CD). En av utmaningarna är att mjukvaran och dess underhåll blir mer komplicerad med tiden. En vanlig kombination av de mest populära produkterna för att hantera CI/CD-flöden är Jenkins och RabbitMQ. I skrivande stund saknas det stöd för automation via standardiserad händelsestyrning enligt Eiffel-protokollet. Härmed finns ett tydligt behov av en lösning som kan fylla denna brist. Till följd av bristen på standardiserat stöd har företag tvingats ta fram egna lösningar, vilket ofta har lett till dåligt samarbete mellan organisationer och bristfälligt utvecklad mjukvara. Målet med detta arbete, som initerats av Nexer, är att möjliggöra för Jenkins att utlösa arbetsinstruktioner på ett händelsedrivet och automatiserat sätt med hjälp av Eiffel-event. Lösningen är utformad för att vara skalbar och flexibel och därmed kunna anpassas efter olika företags behov.
Arbetet har bedrivits iterativt med testdriven utveckling och prototypvalidering. Resultatet är ett fungerande Jenkins-plugin som kan utlösa arbetsinstruktioner baserat på inkommande Eiffel-event. Lösningen inkluderar återanvändbara inställningar, ett användarvänligt gränssnitt för hantering av filterregler samt implementering av en fristående mjukvara EiffelManager som lyssnar efter och behandlar Eiffel-event. Som ett framtida koncept föreslås en fristående tjänst/mjukvara som lyfter ut logiken att orkestrera instanser av EiffelManager i syfte att förbättra skalbarhet och övervakning. Konceptet har potential att effektivt övervaka och hantera en omfattande CI/CD-infrastruktur. Sammantaget indikerar arbetet att det är möjligt att utvidga Jenkins funktionalitet för att implementera händelsestyrda CI/CD-pipelines.

Nyckelord: CI/CD, Continuous Integration, Continuous Development, Jenkins, Eiffel, RabbitMQ, Pipeline

Förkortningslista

Nedan är en förkortningslista i alfabetisk ordning över de akronymer som har använts i detta examensarbete.

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CD	Continuous Development
CI	Continuous Integration
DevOps	Development Operations
EE	Eiffel Event
EM	Eiffel Manager
EO	Eiffel Orchestrator
JIT	Just-In-Time kompilator
JJ	automatiserade processer i Jenkins
JVM	Java Virtual Machine
UI	användargränssnitt

Innehåll

1
Introduktion

Det här kapitlet behandlar bakgrunden till arbetet, dess syfte, vilka mål som vill uppnås samt vilka avgränsningar som gjorts för att tydliggöra projektets omfattning.

1.1 Bakgrund

Allteftersom mjukvaruutvecklingen i världen ökar i både komplexitet och efterfrågan kräver detta strukturerade automatiserade arbetsflöden. Kontinuerlig integration och driftsättning(CI/CD)[1] är ett av de vanligaste uppläggen för att effektivisera arbetsflöden när det kommer till komplexa produkter. Enligt arbetets handledare på Nexer (se appendix B) är Automationsservern Jenkins[2] det mest populära verktyget för distribuering via CI/CD på grund av dess flexibilitet. Ofta används Jenkins tillsammans med RabbitMQ[3][4] för att möjliggöra extern, händelsestyrd kommunikation i ett CICD-flöde. Handledaren nämner att Jenkins är extremt begränsat, när det kommer till automatisering och visualisering av pipelinearbetsflöden. Det saknas stöd för att automatisera arbetsflöden med hjälp av ett standardiserat meddelandeprotokoll som Eiffel-protokollet[5] utvecklat av Ericsson. Handledaren nämner att detta ofta tvingar företag att utveckla sina egna lösningar vilket har lett till dåligt utvecklad mjukvara och bristande samarbete på företagsnivå.

1.2 Syfte

Syftet med detta arbete är att skapa grunden för en stabil och framtidssäker integration mellan Jenkins och Eiffel-protokollet. Syftet är att ta fram en lösning som bidrar till effektivare CI/CD-arbetsflöden med ökad modularitet och spårbarhet, samt att ta ett första steg med en utvecklad prototyp med basfunktioner. Prototypen ska vara designad för att enkelt kunna vidareutvecklas till en slutprodukt som i framtiden kommer möta den efterfrågan som finns och lösa de problem som ligger till grund för arbetet.

1.3 Mål

Arbetets mål är att leverera ett Jenkins-plugin skrivet i Java som:

- **Mål 1**:Gör det möjligt att konfigurera autentisering mot en eller flera RabbitMQ-servrar.
- **Mål 2**:Tillhandahålla ett användarvänligt gränssnitt där användare kan skapa, spara och återanvända filterregler.
- **Mål 3**:Göra det möjligt att knyta Jenkins-jobb (JJ) till specifika Eiffel-event (EE) via en plugin-konfiguration i jobbinställningarna.
- **Mål 4**:Initiera egenutvecklad mjukvara som mottager och tolkar inkommande EE och utlöser JJ när matchning uppstår.
- **Mål 5**:Lägga grunden för vidareutveckling mot en orkestreringslösning som möjliggör skalbar hantering av instanser av den egenutvecklade mjukvaran som beskrivs i mål 4.

1.4 Avgränsningar

Arbetet fokuserar enbart på stödet för inkommande Eiffel-event som triggermekanism i Jenkins. Följande aspekter har inte behandlats eller endast berörts ytligt:

- Visualisering av hela pipelineflöden enligt Eiffel-standarden implementeras inte.
- Säkerhetsaspekter utöver grundläggande autentisering (t.ex. rollbaserad åtkomst, certifikathantering) implementeras inte.
- Prestandatester under hög belastning genomförs inte i större skala, då lösningen främst testats i en kontrollerad utvecklingsmiljö.
- Det föreslagna EiffelOrchestrator konceptet realiseras inte, utan presenteras endast som ett framtida utvecklingsspår.
- Integrationer med andra CI-system än Jenkins eller vidare koppling till andra Eiffel-komponenter kommer ej implementeras.

2
Metod

I det här kapitel presenteras de metoder som kommer att användas.

2.1 Data- och informationsinsamling

Den inledande fasen i arbetet består av inläsning av baskunskap för att få en övergripande förståelse för hur Jenkins, Eiffel och RabbitMQ teknologierna fungerar. Källor som kommer att användas för projektets data- och informationssamling inkluderar:

- **Officiell dokumentation** – Samtliga av projektets verktygs/ramverks officiell dokumentation är välskrivna och kommer vara de mest betydelsefulla informationskällorna.
- **Presentationer** – Då detta projekt tar upp ett aktuellt problem som diskuteras inom företag, har det publicerats flertal presentationer angående detta på Eiffel-Communitys officiella YouTube-kanal.
- **Existerande plugin** – Existerande Jenkinsplugin som använder samma/liknande ramverk byggda av väletablerade utvecklare kommer att vara en värdefull källa för att se till att projektet får en igenkännbar och Jenkinstypad struktur.
- **Forskningsartiklar** – Arbetet kommer att använda tidigare forskning för att ta del av jämförelser av kommunikationsprotokoll samt innebörden av CI/CD.
- **Intervju** – För att stärka projektsyftets trovärdighet kommer vi genomföra intervjuer med vår handledare samt utvecklare på Ericsson som har arbetserfarenhet inom området.

2.2 Mjukvarudesign

Jenkinspluginets design kommer att prioritera funktionalitet men även ha fokus på användarvänlighet. Eftersom produkten är avsedd att användas av utvecklare med olika bakgrund och erfarenhet behövs en igenkännbar struktur som följer standarder för Jenkinsplugins. Pluginet ska även vara skalbart för att kunna tillämpas i både enklare projekt i mindre skala och mer avancerade projekt i större skala.

3
Teknisk bakgrund

För att förstå den tekniska lösningen som presenteras i detta arbete beskrivs de verktyg, ramverk och protokoll som används. Detta kapitel ger en översikt över de centrala mjukvaruplattformarna som projektet bygger på. Fokus ligger på Jenkins, ett verktyg för automatisering av CI/CD-processer, samt Eiffel-protokollet som möjliggör händelsestyrd kommunikation i komplexa pipelines. De viktigaste teknologierna som ingår i arbetet är RabbitMQ som meddelandekö, Java som programmeringsspråk, och Docker[6] som containerlösning. För versionshantering används GitHub[7].

3.1 Jenkins

Jenkins är en open-source automatiserad orkestrerare för mjukvaruutveckling och distribution som används inom CI/CD flöden. När kodförändringar sker under övervakning av Jenkins kan automatiserade bygg-, test och distributionssteg utföras. Jenkins är skrivet i Java och erbjuder ett bra stöd för utveckling av plugin och har som följd ett starkt tillhörande plugin-ekosystem. Denna egenskap hos Jenkins gör det möjligt att utöka dess funktionalitet för en uppsjö av plattformar som Kubernetes, Docker, Maven[8] och Git.

Både stora och små företag använder Jenkins för att påskynda och säkra kvalitén på mjukvaruutveckling. Jenkins använder en master-agent arkitektur där en Jenkins-controller styr hur agenter (noder) utför en handling i en pipeline och hur handlingen distribueras mellan maskinerna som utgör pipelinen. Mjukvarans bygg-, test- och distributionsflöde kan beskrivas som ett set väldefinierade handlingar eller arbeten som ska utföras i pipelinen. Detta refereras till som deklarativa eller skriptbaserade pipelines och denna typ av pipelines uttrycks ofta i form av källkod. Ett alternativ till att definiera pipelines som programkod erbjuds genom ett grafisk webbgränssnitt.

Till följd av Jenkins breda stöd för utbyggnad av funktionalitet och dess aktiva community, används Jenkins av både små och och stora utvecklingsteam för att minska manuellt arbete, skapa bättre kvalité samt påskynda leveransprocessen.

3.2 Eiffel

Eiffel är ett kommunikationsprotokoll i realtid som är oberoende av operativsystem, tillhörande mjukvara och är utvecklat av Ericsson för stora komplexa CI/CD-pipelines. En av grundtankarna bakom skapandet av Eiffel var att parter med varierande krav, förväntningar och förutsättningar ska kunna spåra händelser i pipelineflödet. På detta sätt skapas ett öppet och effektivt informationsutbyte av händelser mellan parter utan behov av direkt interaktion.

Traditionellt beskrivs en pipeline som ett set av konfiguerade jobb som utgör hela arbetsflödet från start till distribution. Eiffel betraktar i stället alla dessa aktiviteter som microtjänster med kommunikation sinsemellan. Varje handling inom CI/CD utgör ett event som delas på en global meddelandebuss. Ett event i sig är komplett med all nödvändig information för att beskriva just den aktiviteten.Utöver det kan även ett Eiffel-event referera till det event som låg till grund för dess uppkomst. Tillsammans bildar alla dessa events ett nätverk av riktade beroenden utan cirkulära referenser (riktad acyklisk graf) som innehåller all information om vad som händer i en CI/CD-pipeline i realtid.

3.3 RabbitMQ

RabbitMQ är en populär open-source meddelandeförmedlare mellan tjänster som behöver kommunicera med varandra. Meddelandeförmedlarens funktionalitet är skriven i Erlang och använder sig utav kommunikationsprotokollet Advanced Message Queueing Protocol (AMPQ) [4]. Biblioteken som är skrivna för att kunna integreras med andra program är utvecklade i flera språk, bland annat Java.

Producer-consumer[9] arktitekturen möjliggör att producenter kan lägga till ett meddelande i kön och direkt gå vidare till sin nästa uppgift, likaså kan konsumenter hämta ett meddelande från kön och gå vidare till nästa uppgift. Därmed bildar RabbitMQ en brygga som resulterar i låg koppling mellan tjänsterna.

RabbitMQ erbjuder bred flexibilitet vid riktning av meddelande till konsumenter. Kombinering av konfiguration av både växlingspunkt och meddelande är möjligt. Denna flexibilitet möjliggör riktande av meddelande till en eller flera mottagare av ett givet meddelande. RabbitMQ kan köras på fristående maskin, vilket möjliggör hög prestanda för både konsumenter och producenter.

3.4 Java

Java är ett plattformsoberoende och robust programmeringsspråk vilket har lett till att det är ett av de mest populära språken idag. Vidare har Java en lång och väletablerad användning inom webbutveckling, företagslösningar, inbyggda system, spel och mobilappar. Utveckling av språket började 1991 och första versionen släpptes 1995. Idag är Java tillgängligt som open source via OpenJDK.

Hög prestanda levereras genom en kombination av kompilering, tolkning via Java Virtual Machine (JVM) [10] och Just-In-Time (JIT) [11] kompilering. Genom att JIT utför analys av källkod kan den upptäcka de delar som exekveras oftare än andra och dessa kompileras till maskinkod i realtid. Java stödjer multitrådning som är ett krav för prestanda hos de applikationer som kräver parallell exekvering.

Java har en automatiserad minneshantering som förenklar utveckling av mjukvaran på så sätt att programmerare inte behöver fokusera på allokering eller frigöring av minne. Javas äutomatic garbage collection"innebär att objekt som inte refereras i koden längre frigörs och då minskar risken för minnesläckor, vilket i sin tur förenklar minneshanteringen.

3.5 Docker

Docker är en plattformsoberoende containeriseringsteknik som möjliggör paketering, utveckling och distribuering genom att skapa isolerade miljöer som kallas cointainrar. Docker har funnits tillgängligt sedan 2013 och är oerhört populär i sammanhang som mjukvaruutveckling, molnbaserade miljöer och Development Operations (DevOps)[12].

En container är en kompakt och smidig virtualiserad miljö där allt finns som behövs för att köra önskad mjukvara så som mjukvarubibliotek, systemresurser och tillhörande miljö som krävs för önskad exekvering. Till följd av detta kan containers köras på i stort sett vilken maskin som helst som har förmågan att köra Docker utan några som helst kompabilitetsvårigheter. Det är denna egenskap som vidare möjliggör att paketering, utveckling och distribuering är väldigt enkelt.

Containers kan man se som en förenklad dator med dess egna operativsystem, egna CPU-processer, minne och nätverksresurser. Utöver detta är Docker oerhört resurseffektivt då det kommunicerar direkt med underliggande operativsystems kärna för att få tillgång till systemresurser. Detta skiljer sig från en klassisk virtuell maskin där hela segment av underliggande operativsystemets resurser måste reserveras som senare inte är tillgängliga för någon annan process än den tillhörande virtuella maskinen. Till följd av denna effektivitet gällande systemresurser är det väldig enkelt och billigt att lägga till, ta bort, stanna, stoppa eller skala upp och ner antal containers.

3.6 GitHub

Github är en webbaserad tjänst för versionskontroll och molnlagring. Genom att lagra all källkod i molnet förenklas samarbete vid mjukvaruutveckling. Källkod kan göras tillgänglig för samarbetare och det finns möjlighetet att ställa in vem som får ändra på vad. Vidare finns det även stöd för att förändringar av källkod kräver genomröstning.

3.7 Maven

Maven är ett versionshanteringsverktyg vars användande fastställts som standard vid utveckling av Jenkins plugin.

3.8 Alpine Linux

Alpine Linux[13] är en minimal Linux-distribution som använts i det här examensarbetet för att köra egenutvecklad mjukvara med minimal systemresursåtgång.

3.9 JSON

JSON[14] ett meddelandeprotokoll som är plattformsoberoende och har hög flexibilitet gällande konfigurering och sammansättning av innehåll.

4
Genomförande

Här redogörs för examensarbetets utvecklingsprocess och hur de viktigaste stegen och milstolparna har utvecklats steg för steg.

4.1 Instudering

Initialt var det viktigt att skapa en övergripande förståelse för de teknologiska ramverken som tilldelats av Nexer. Dessa består huvudsakligen av Jenkins, Eiffel och RabbitMQ, men även Docker och GitHub har varit essentiella teknologier. Nedan följer en lista som specificerar nödvändig kunskap för att genomföra arbetet:

- **Jenkins** – CI/CD-motor: Här krävs övergripande kunskap om API strukturen, grundläggande pluginutveckling både backend och frontend samt hur man på bästa sätt hanterar externa signaler och kommunikation. Denna information hämtades främst från Jenkins officiella dokumentation och existerande verifierade plugins i öppen källkod.
- **Eiffel** – Kommunikationsprotokoll: Är uppbyggt av ett specifikt antal mallar i JSON-format. Utförande av examensarbetet krävde en förståelse för hur Eiffel-event är uppbyggda och hur man validerar dessa på bästa möjliga sätt. Informationen hämtades främst från Eiffel Communitys officiella dokumentation samt presentationer ifrån Eiffels årliga summit event publicerade på deras youtube-kanal[15].
- **RabbitMQ** – Denna komponent möjliggör kommunikation mellan system med hjälp av meddelandeköer. Här krävs inga djupare kunskaper men det krävs grundlig förståelse för hur man lyssnar på specifika köer och hur man filtrerar meddelanden, vilket förklaras tydligt i RabbitMQs officiella dokumentation.
- **Java, Docker och GitHub** – Dessa verktyg har varit en del av tidigare utbildning i programmet. Någon övergripande efterforskning på hur de fungerar har därför ej behövts.

4.2 Utveckling av Jenkins Plugin

Efter en noggrann instudering inleddes utvecklingen av Jenkins-pluginet EiffelTrigger, som utgör en del av det praktiska i arbetet.

4.2.1 Jenkins Plugin

Det första steget i utvecklingsprocessen påbörjades genom att skapa ett enkelt "Hello World-plugin för Jenkins. Detta var en viktig inlärningsfas som gav oss förståelse för hur Jenkins plugin-arkitektur fungerar, inklusive struktur, beroenden, byggsystemet Maven, samt hur UI-komponenter integreras med backend-logik i Java genom Jenkins Jelly-filer.
Genom att följa den officiella Jenkins plugin-guiden[16] kunde vi snabbt sätta upp ett minimalt fungerande plugin. Denna övning gav oss en stabil grund att vidareutveckla på, och gjorde det möjligt att experimentera med Jenkins inbyggda funktioner.
Hello World-projektet fungerade som en bas för vår pluginutveckling, där vi gradvis bytte ut komponenter och lade till funktionalitet för att uppfylla de uppsatta målen med detta arbete. Det hjälpte oss även att förstå hur plugins paketeras, testas och distribueras inom Jenkins ekosystem.

4.2.2 Behandling av delmål

EiffelTriggers grundläggande design för önskad funktionalitet baserat på examensarbetets mål kan man översiktligt beskriva genom att ange de nedan listade processerna i systemet:

- **Jenkins** – Har som uppgift att hantera noderna i CI/CD-flödet, alltså definiera, bygga och hantera jobb. I linje med Mål 1, 2 och 3 måste det även på ett användarvänligt sätt vara möjligt att konfigurera EiffelTrigger och för Mål 5 kunna kommunicera med övriga processer i systemet.

För att uppnå Mål 4 krävs följande processer:

- **Lyssnare** – En process som kontinuerligt lyssnar på inkommande Eiffel-Event ifrån RabbitMQ
- **Utlösare** – Signalerar Jenkins att bygga jobb.
- **Tolkare och validerare** – Tolkar och validerar inkommande event och gör beslut om eventet ska aktivera Utlösaren.

Listan utgör en enkel systembeskrivning av EiffelTrigger angivet i processer som i samråd med handledare från Nexer ansågs som Minimal Viable Product". En överskådlig förenklad bild av programmet som har de absolut nödvändigaste funktionerna som krävs för ett fungerande system.

4.2.3 Design för användarvänlighet i Jenkins

För att EiffelTrigger enkelt ska kunna användas av företag krävs en design som är igenkännbar och enkel att konfigurera. För att uppnå detta började vi fokusera på interaktiv design och utvecklade de följande tre verktygen som bidrar till en röd tråd i plugin-konfigurationen.

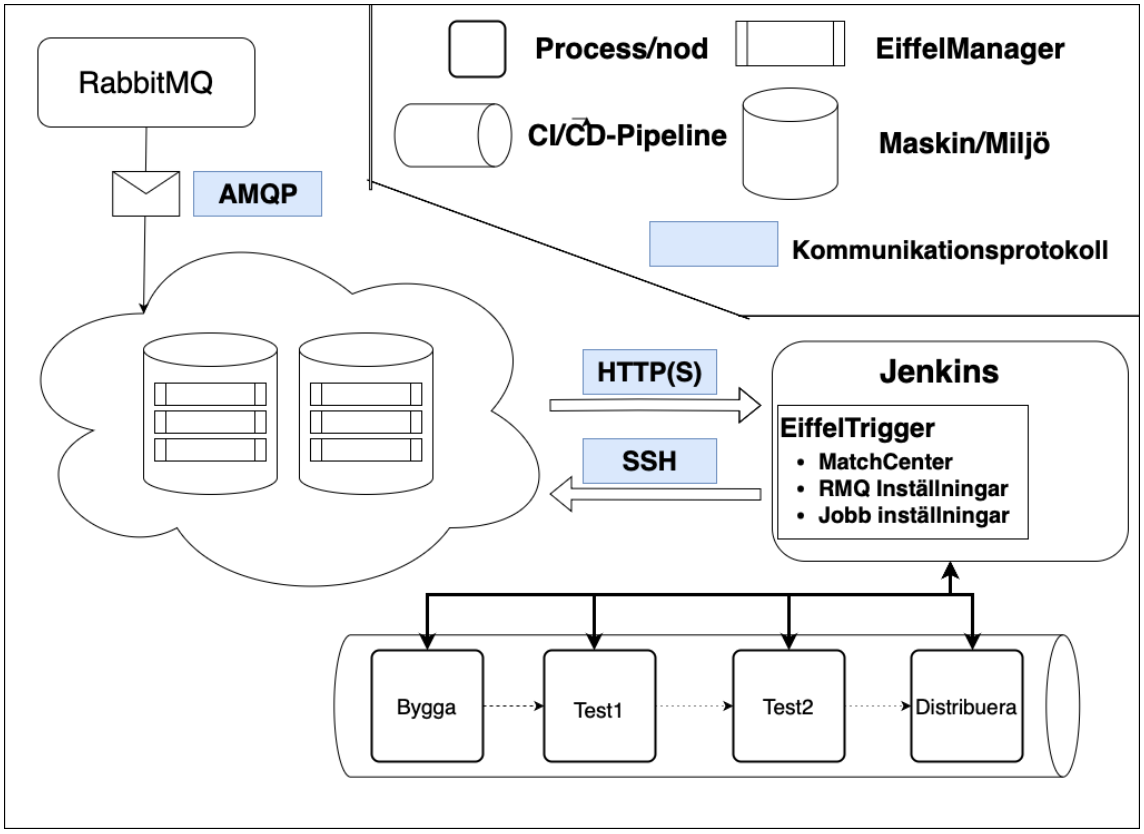
- **RabbitMQ Konfiguration** – Ett önskemål ifrån handledare från Nexer var att man enkelt ska kunna spara autentiseringsuppgifter till RabbitMQ i Jenkins systeminställningar. Detta verkställdes genom ett formulär som skapades i Systeminställningar där användaren kan fylla i, spara och sedan hämta sparade autentiseringsuppgifter. Användaren har även möjlighet att spara ner flertal set med uppgifter för att i framtiden kunna koppla upp sig till flera RabbitMQ servrar samtidigt.
- **Matchning** – För att kunna avgöra om ett inkommande EE ska utlösa ett jobb i Jenkins behöver det matcha med det specfika jobbets filterregler, vilket användaren som installerar pluginet ansvarar för. För att underlätta detta utvecklades ett eget verktyg som enkelt kan navigeras till via huvudmenyn. Detta verktyg bidrar till att användare enkelt kan skapa filterregelkortsom sedan kan återanvändas enkelt vid skapandet av ett utlösbart jobb. Processen blev ännu smidigare genom tillägget av en ny funktion som gör det möjligt för användaren att enkelt generera grunden för filterregeln genom att välja vilka event de ska kunna matchas mot och trycka på generera.
- **Konfiguera jobbutlösare** – För att aktivera EiffelTrigger och göra ett JJ utlösbart behöver det ställas in i jobbets konfigurationsmeny när det skapas/-definieras. En checkbox för själva plugin-aktiveringen och en meny som blir synlig först när checkboxen är checkad. Menyn består av två stycken val i form av rullgardinsmenyer där användaren väljer instans av RabbitMQ-uppgifter och Matchningskort vilka skapats i de föregående verktygen.

4.2.4 Lyssnare och utlösare

Processen att skapa en lyssnare, eller rättare sagt en RabbitMQ consumersom kollar och hämtar EE från en definierad kö, gjordes genom att implementera en RabbitMQ callback-metod. Här definierar man vilken kö man vill lyssna på och vad som händer om callbacken utlöses. I det här steget validerar och matchar EiffelTrigger det inkommande EE och avgör därmed om en signal ska skickas för att utlösa jobbet i Jenkins. För att skicka signalen hade vi olika val att välja mellan men landade i att HTTP(S)-förfrågan var det mest passande för ändamålet. Tankarna bakom det valet går att läsa om i rapportens diskussionskapitel.

5
Resultat och analys

I det här kapitlet presenteras resultaten som uppnåtts i eftersträvan att uppfylla målen för arbetet. Figur 5.1 visar en översikt av alla dellösningar och hur de samverkar.



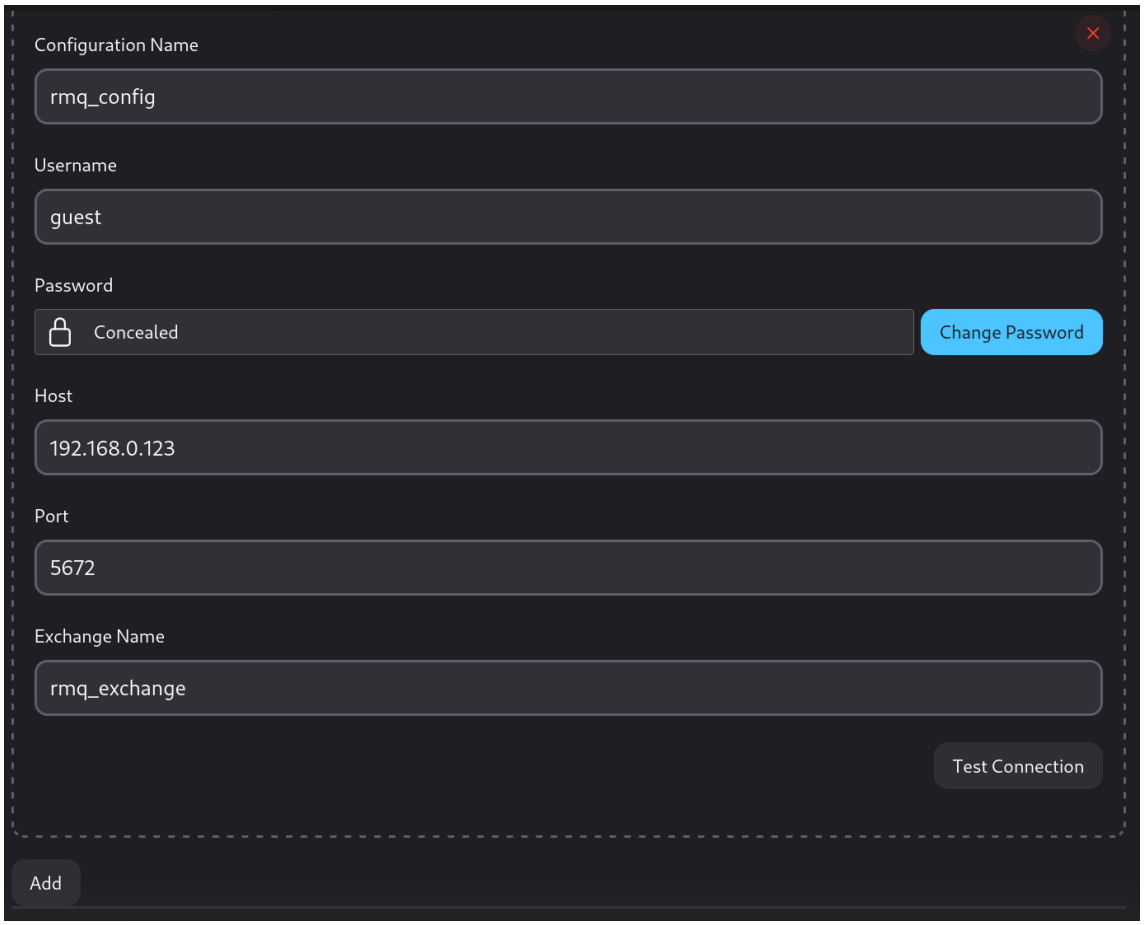
Figur 5.1: Översiktsblick av arbetet, dess dellösningar samt hur de samverkar.

5.1 Jenkins-plugin

I kapitel 5.1.1-3 redovisas det Jenkin-plugin som utvecklats i detta arbete och benäms som EiffelTrigger samt den externa mjukvaran som inititeras av EiffelTrigger för att lyssna, validera och reagera på EE.

5.1.1 Modul för konfiguration av RabbitMQ-autentisering

Figur 5.2 visar en skärmdump av det grafiska gränssnitt som EiffelTrigger lägger till i Jenkins systeminställningar. Genom detta gränssnitt matas uppgifter in så som användarnamn, lösenord, värd ("hosttex IP-adress) och port för en specifik RabbitMQ-instans. Uppgifterna kan sparas under ett godtyckligt namn för lättare återanvändning. Det finns också möjlighet att ta bort dessa vid ett senare tillfälle för att hålla listan med namn aktuell. Den nedre knappen ”Test connection” ger möjlighet att säkerställa att konfigurationen är giltig.

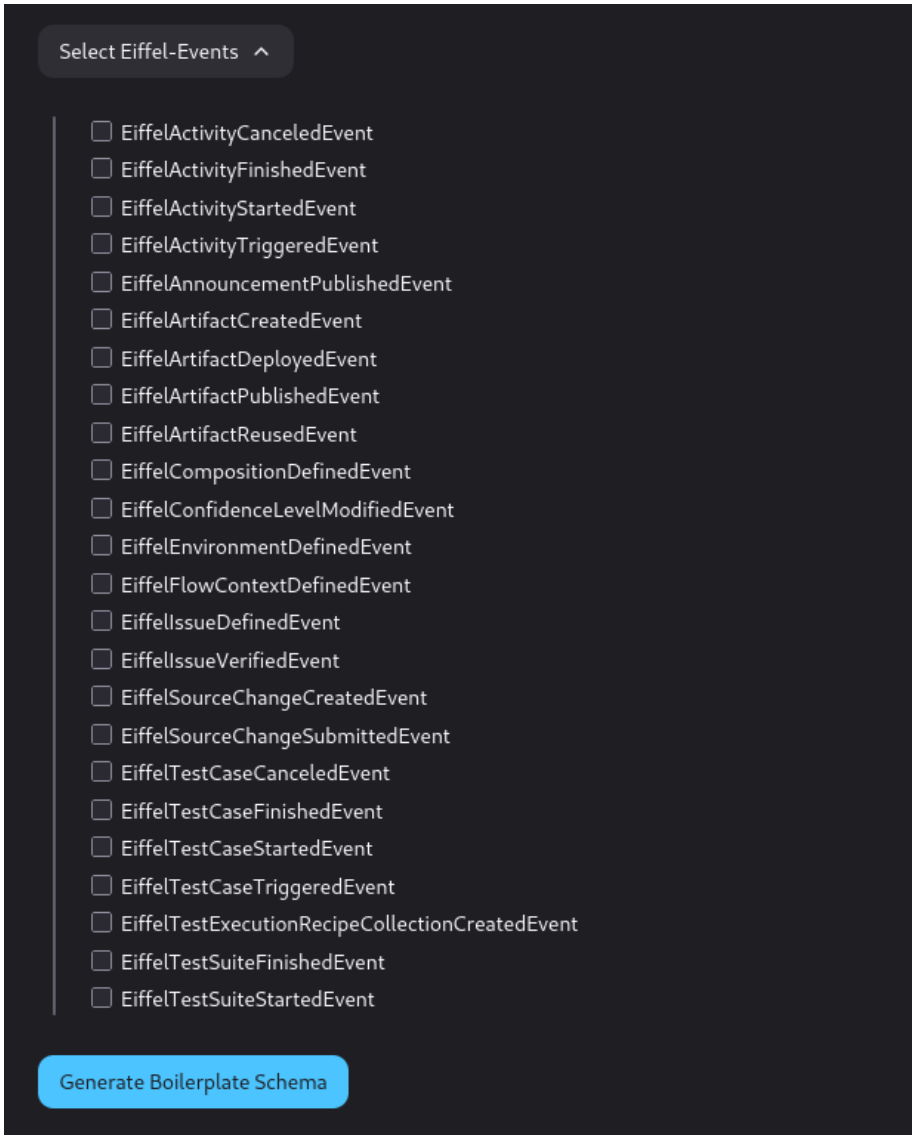


Figur 5.2: Grafiskt gränssnitt för att skapa och spara RabbitMQ-autentiseringen.

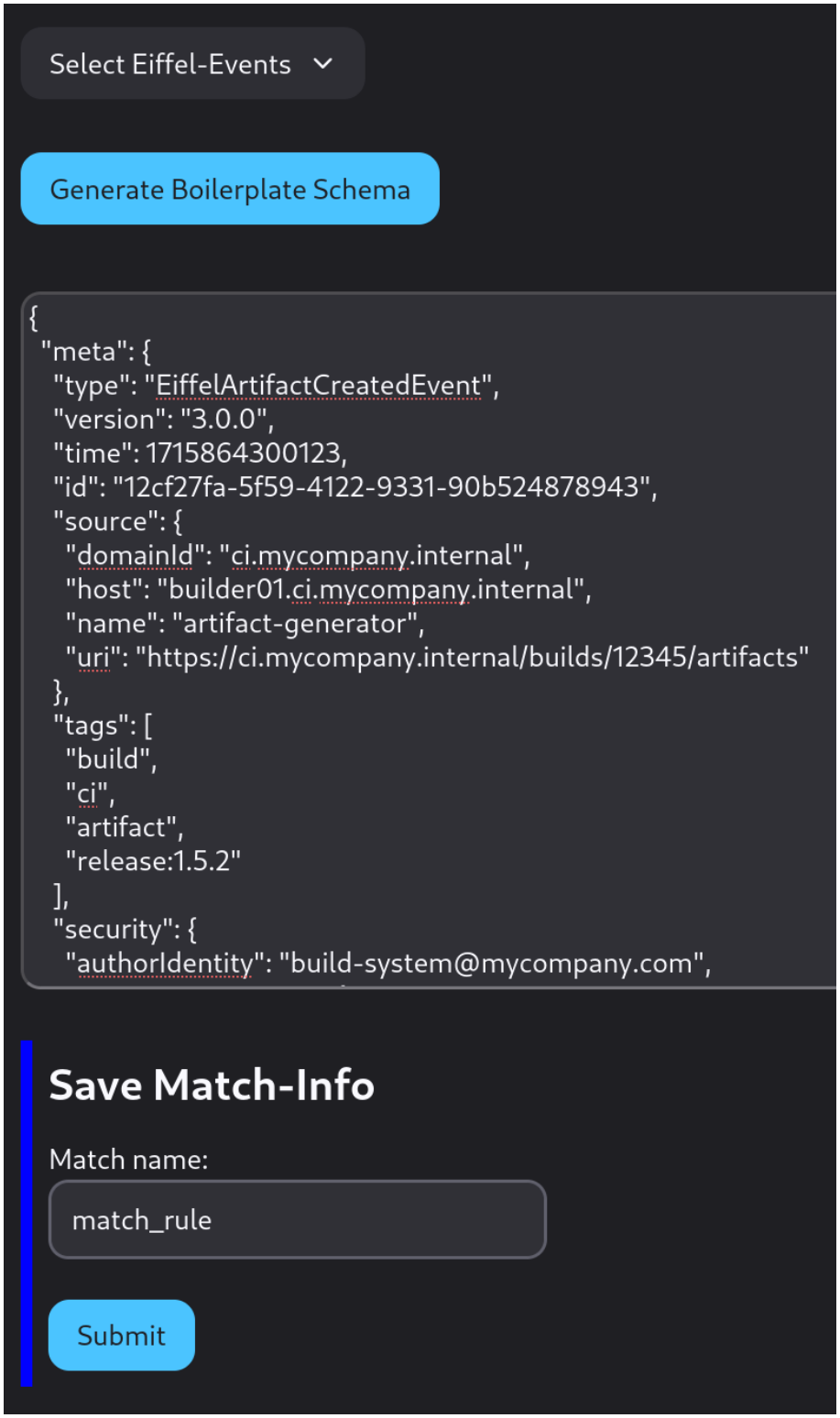
5.1.2 Modul för konfiguration av filterregler

I vårt grafiska gränssnitt Matchcenter skapas filterregler i form av JSON-schemas. För att underlätta inmatningen kan användaren generera en grundmall för filterregeln och välja en uppsättning EE ur listan. Listan med event är initialt dold för användaren men visas när användaren trycker på Select Eiffel-Events". För att generera grundmallen kan användaren bocka i önskade event i listan och sedan trycka på knappen Generate Boilerplate Schema". Därefter kan användaren i textform redigera filtret utefter önskad funktionalitet och spara under ett godtyckligt namn. Användaren kan se alla sparade filterregler i en lista längst ner på sidan och har även möjlighet att kopiera eller radera sparade filterregler, vilket sker genom ikonerna längst upp till höger för sparade filterregler, se figur 5.5. Följande delar redovisas i figur 5.3-5.

- Grafisk konfigurering av filterregler, figur 5.3
- Filterregel i redigerbar textform, figur 5.4
- Sparade filterregler, figur 5.5



Figur 5.3: Grafiskt gränssnitt för att göra urval bland samtliga EE.

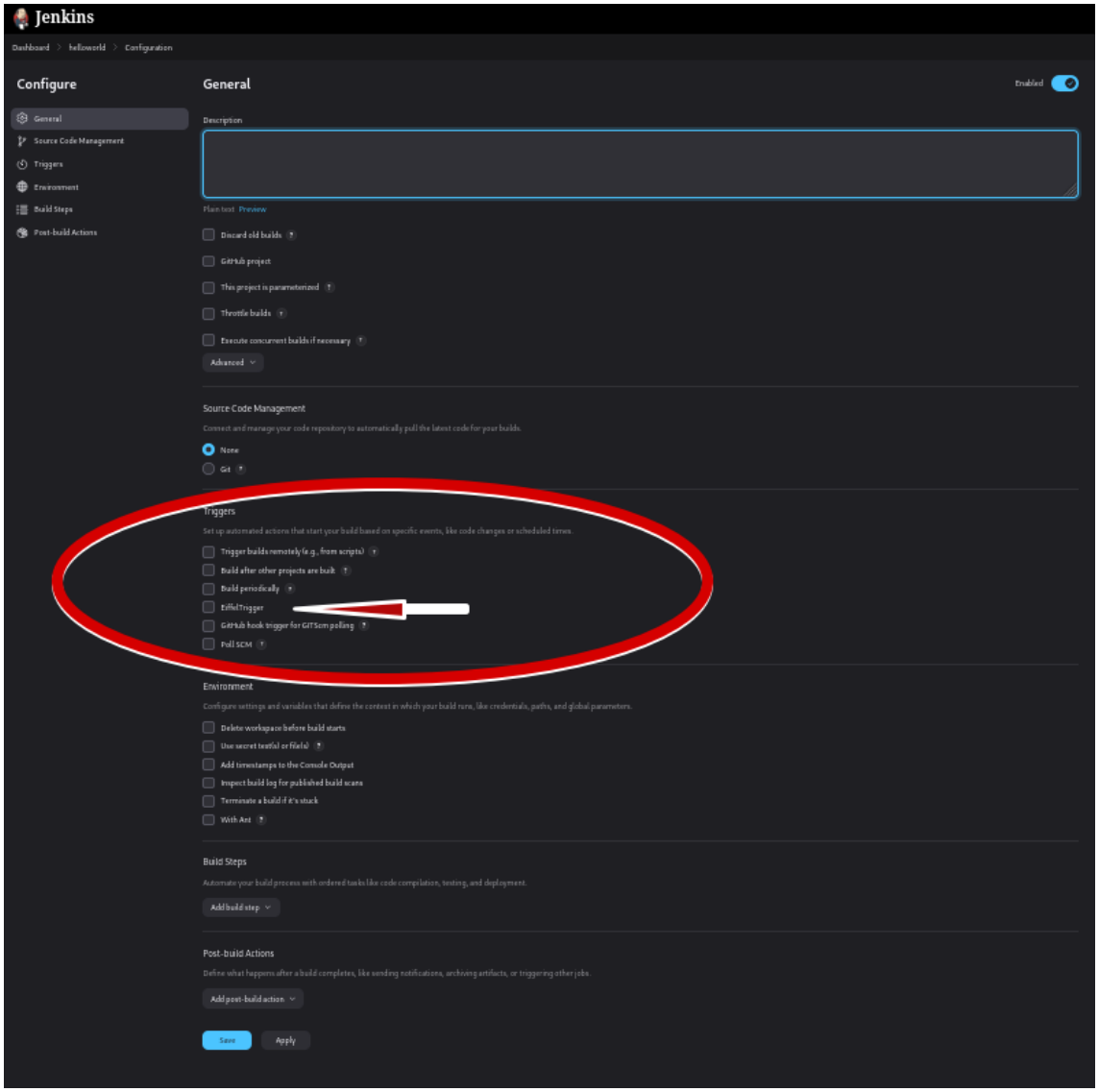


Figur 5.4: Filterregel representerad i redigerbar textformat samt inmatning för att spara under godtyckligt namn.

Figur 5.5: Figuren visar en sparad filterregel enligt JSON-schema representerad i textform.

5.1.3 Jenkins-plugin EiffelTrigger för konfiguration av utlösare i Jenkins-jobb

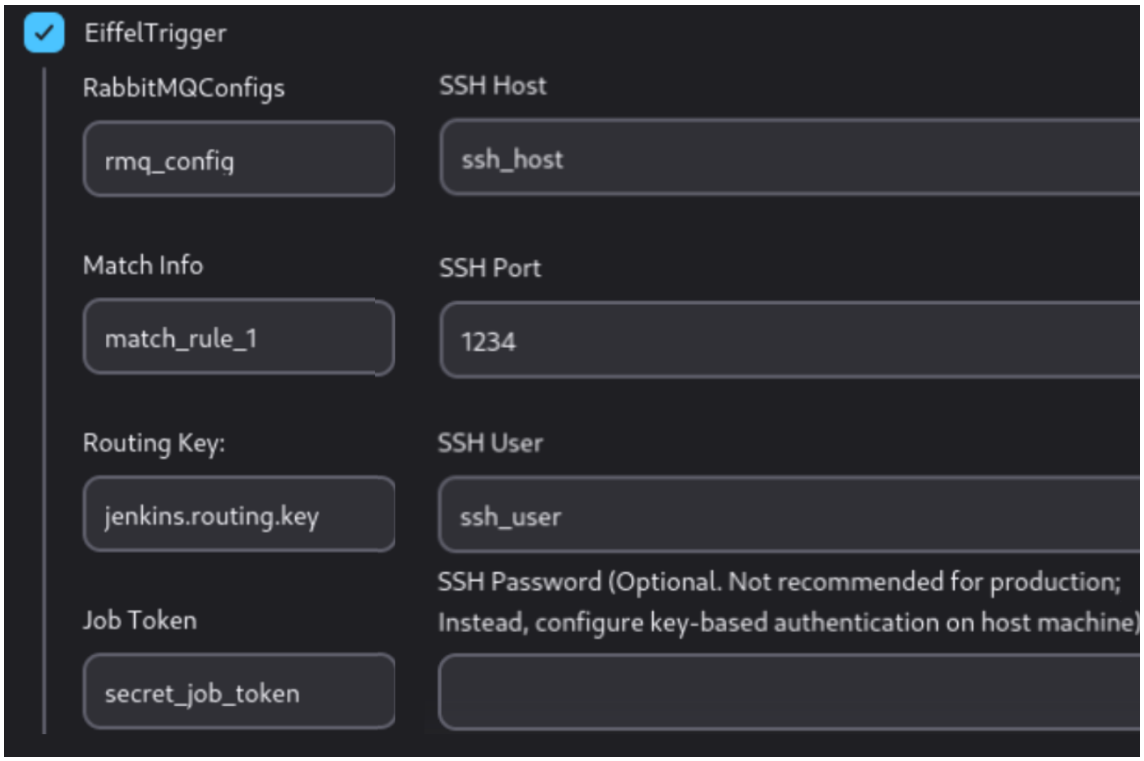
När ett nytt Jenkins-jobb skapas eller redigeras får användaren upp ett alternativ att göra jobbet utlösbart med EiffelTrigger i jobbkonfigurationsmenyn i **Triggers** sektionen, se Figur 5.6



Figur 5.6: Triggers menyn för Jenkins-jobb där EiffelTrigger är valbar (se där pilen pekar).

Då användaren valt EiffelTrigger presenteras åtta konfigurationsfält, se Figur 5.7.

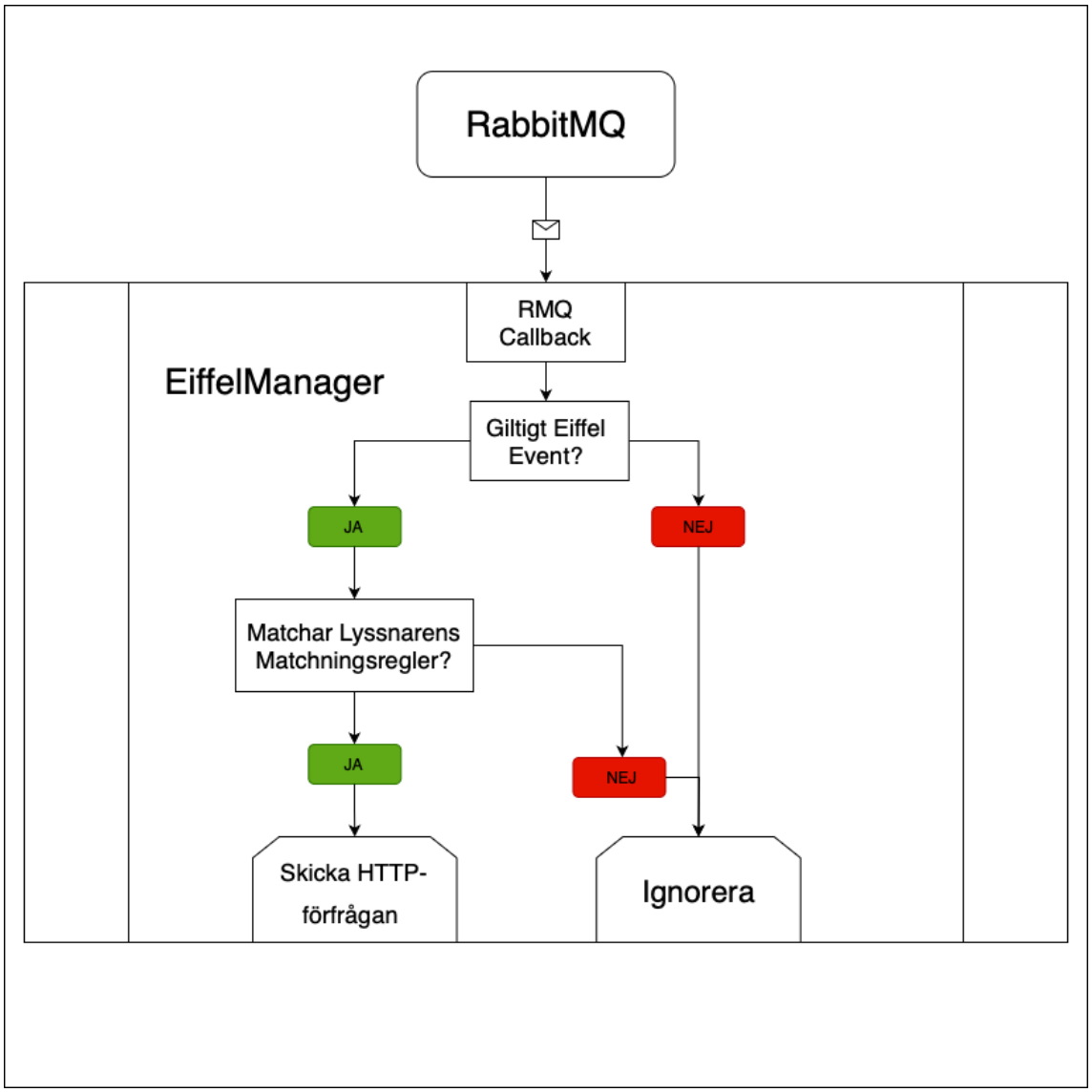
- Val av tillgängliga sparade RabbitMQ-autentiseringsuppsättningar.
- Val av tillgängliga sparade filterregler från Matchcenter.
- Inmatning av 'Routing Key' för placering av meddelande i förutbestämd RabbitMQ-kö.
- Job Token, används för att utlösa ett jobb i Jenkins.
- De sista fyra konfigurationsfälten används för inmatning av SSH-uppgifter i syfte att möjliggöra fjärranslutning till extern maskin för att starta en EM.



Figur 5.7: EiffelTrigger submenyn till Build Triggers

5.2 Utvecklad mjukvara; EiffelManager

Då ett EE anländer till en RabbitMQ-kö och innehållet uppfyller kraven definierade av en filterregel, kan en händelse i en CI/CD-pipeline utlösas automatiskt enligt konfigurering av JJ som sätts upp enligt kap 5.1.3. Utöver detta ger EiffelManager(EM) stöd för loggning, rudimentär felhantering samt hantering av ogiltiga händelser. EiffelTrigger lyssnar kontinuerligt på en förutbestämd RabbitMQ-kö. Då ett meddelande anländer till denna utlöses en Callback-funktion[9]. Detta visas schematiskt i Figur 5.8.



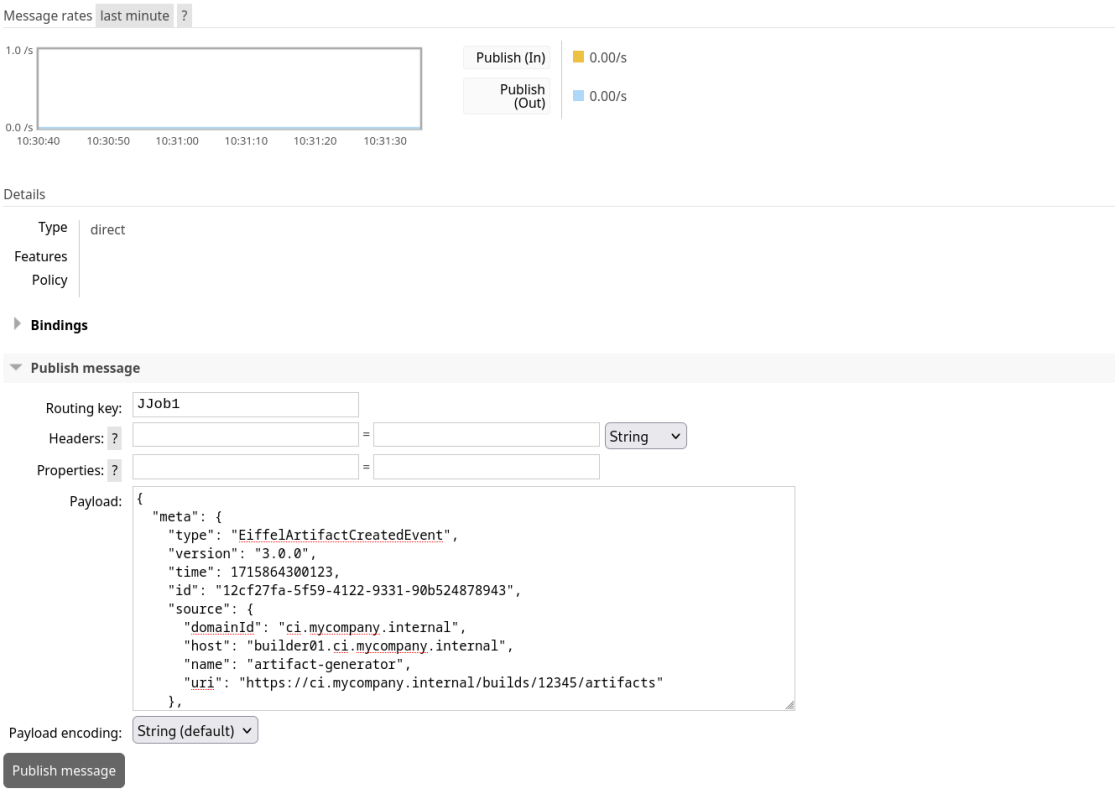
Figur 5.8: Schematisk beskrivning av hur EM lyssnar på RabbitMQ och möjliggör Callback-funktion för start av Jenkinsjobb.

I Callback-funktionen sker validering och verifiering, dvs. att meddelandet är ett EE och ett giltigt sådant. Uppfylls detta meddelar EM att starta det representerade jobbet i Jenkins genom att skicka en HTTP-förfrågan till Jenkins API.

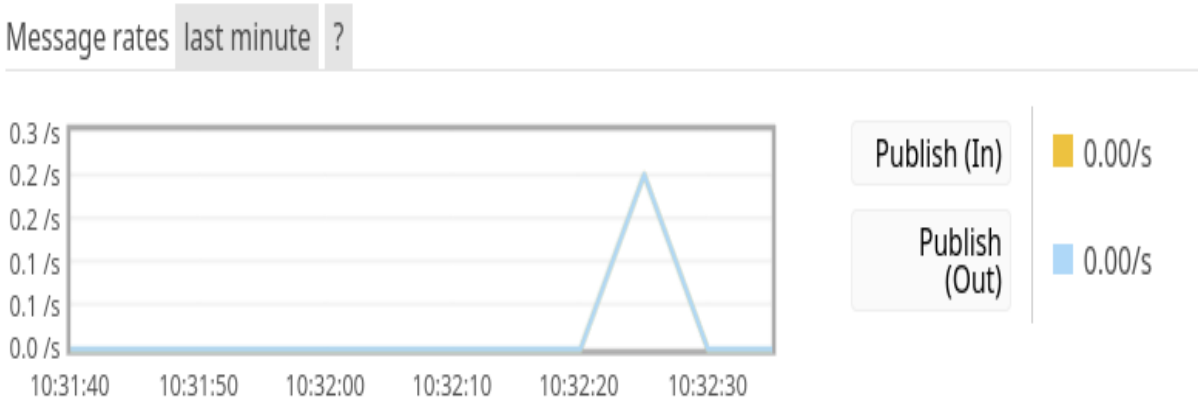
5.3 Demonstration

Nedan redogörs händelseförloppet för EM. Det EE som används finns redovisat i appendix A.

- Ett meddelande som innehåller ett giltigt EE publiceras i en RabbitMQ-kö, se figur 5.9.
- En tillfällig ökning av meddelandeinflödet (Messages rate) till RabbitMQ, som visas via dess användargränssnitt, berättar att meddelandet anlänt till kön, se figur 5.10.
- EM mottar ett EE, validerar, processar innehållet och skapar en arbetsinstruktion som skickas till Jenkins, se figur 5.11.
- Jenkins har mottagit och verkställt arbetsinstruktionen, se figur 5.12.

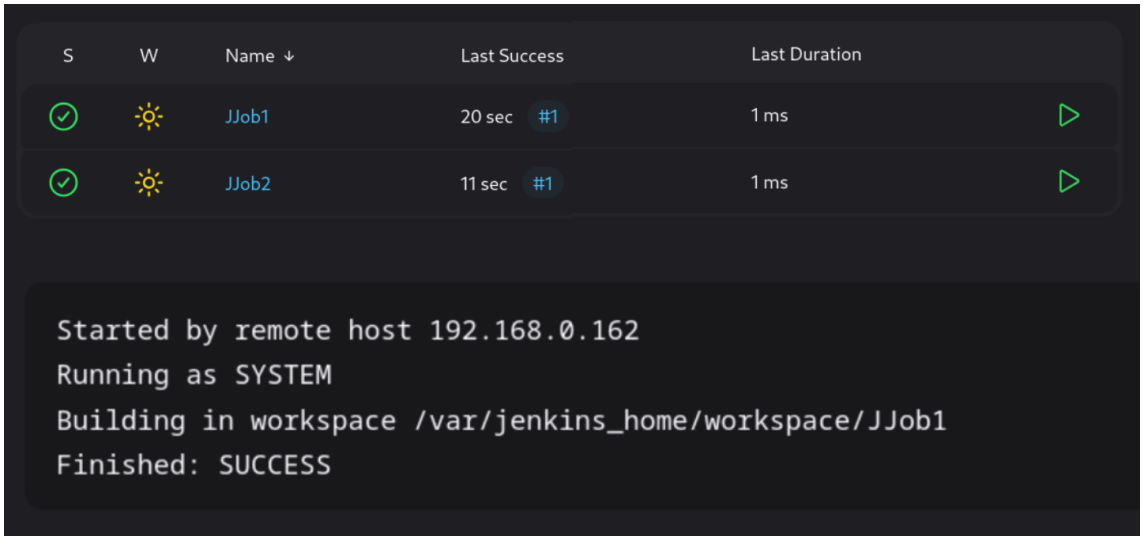


Figur 5.9: Meddelande som innehåller ett EE publiceras i en förutbestämd RabbitMQ-kö.



Figur 5.10: Meddelande som innehåller ett EE är mottaget av RabbitMQ-kö. Ökning av 'messages rate' i grafen viar att ett meddelande anlänt.

Figur 5.11: EM har mottagit ett EE, validerat meddelandet och beslutat om att skicka arbetsinstruktion till Jenkins.



Figur 5.12: Jobbet som är förutbestämt av EiffelTrigger har utförts.

6
Diskussion

I detta kapitel diskuteras genomförandet och resultaten av arbetet, vilka inkluderar tekniska lösningar, uppkomna komplikationer samt förslag på vidareutveckling. Det redogörs för examensarbetets framgångar, utmaningar och förbättringsområden som potentiellt hade underlättat arbetet.

6.1 Tekniska val och avvägningar

Jenkins-plugin skrivs i språket Java och detta gav oss möjligheten att använda Jenkins och RabbitMQ's API som båda är primärt i Java.
Vi hade ingen tidigare erfarenhet eller kompetens vad gäller Jenkins, RabbitMQ eller Eiffel. Det visade sig kräva en väldigt stor insats för att komma igång med utveckling av Jenkins-plugin. Det fanns inte bara krav på hur man skriver kod, integrerad utvecklingsmiljö (IDE), och Maven utan även på själva katalog och filstrukturen av projektet i IDEn.

Docker spelade en stor roll för utvecklingsmiljön och här hade vi inte heller tillräckligt med kunskap. Vi hade båda tidigare använt Docker i begränsad utsträckning men även här gjordes en rätt tilltagen insats för instudering. Tiden vi lade ner på att få Docker att fungera var väl investerad då vi kunde återställa en eller flera containers till ursprungligt skick utan mycket ansträngning eller tid förbrukats. Eftersom vi jobbade med examensarbetet på flera olika maskiner och geografiska platser användes en blandning av laptops och stationära datorer. En av oss var dessutom tvungen att köpa en ny laptop på grund av att den gamla gick sönder under arbetets gång. Att ha all mjukvara vi använde i projektet containeriserad underlättade enormt, då det gick snabbt att återställa utvecklingsmiljön på en ny maskin. Det var bara att installera Docker på den nya maskinen, ladda ner våra containers och arbetet kunde fortsätta.

6.2 Metod och genomförande

Med anledning av att vi endast varit två stycken i det här examensarbetet har har vi haft möten på veckobasis och daglig kommunikation för planering.
Den stora utmaningen i vårt genomförande har varit bristen på kunskap och insikt i företags arbetsflöden och system där mjukvaran vi utvecklat är tänkt att integreras. Framförallt en utmaning att göra ett program man inte själv jobbat med tidigare användarvänligt och att effektivt anpassa det efter dagens standarder. Detta bidrog till att fasen med information- och datainsamling tog mer tid än planerat, vilket i sin tur resulterade i mindre utvecklingstid. Ett studiebesök i början av projektet hade kunnat ge oss en mer grundläggande förståelse för arbetets tillämpning, vilket sannolikt hade effektiviserat den inledande fasen.

6.3 EiffelTriggers användbarhet i praktiken

Arbetsbeskrivningen är framtagen tillsammans med Nexer och grundar sig i att lösa ett aktuellt problem i företags CI/CD-utveckling. Behov och efterfrågan har även bekräftats i samtal med anställda på Ericsson. Vi har själva ingen möjlighet att praktiskt utvärdera vår framställda mjukvara i praktiken. Därför litar vi på vår handledares omdöme att det vi framställt utgör en god och lovande grund för vidareutveckling. De grafiska gränssnitten vi utvecklat har visat sig vara enkla att förstå och använda, givet att användaren är familjär med Jenkins. Gränssnittet är utvecklat med tanke på att det ska vara användbart för alla tänkbara CI/CD-projekt och inte något specifikt projekt eller företag. Mjukvaran har anpassats för fortsatt vidareutveckling och genom små justeringar kan prestanda, funktionalitet och användbarhet förbättras ytterligare.

6.4 Utmaningar och begränsningar

Detta kapitel redogör för de huvudsakliga utmaningarna och begränsningarna som identifierades under arbetets gång, samt hur dessa påverkade projektets genomförande och resultat.

6.4.1 Utmaningar

I början av arbetet hade vi stora svårigheter att förstå konceptet som föreslogs och vad det faktiskt var vi skulle göra. Hur passade alla delar ihop och hur skulle alla delar användas och utvecklas? Även om vi hade god kommunikation med handledare på Nexer krävde det sin tid att komma till den förståelse som behövdes för att kunna påbörja utvecklingsstadiet. Det är svårt att uppskatta tiden som gick åt här i proportion till utveckling av mjukvara och dokumentation. Vi skulle vilja påstå tidsåtgången var oproportionerlig för ett arbete av den här storleken. Hade arbetet pågått i ytterligare 2-4 månader hade tiden vi var tvungna att lägga initialt varit mer rimlig. Instudering av dokumentationen för Jenkins och hur Jenkins-plugin utvecklas tycker vi var omständlig. Även om dokumentationen i sig är välskriven var det svårt att sätta sig in i hur olika delar skulle användas.

Det finns en officiell grundläggande guide för utveckling av Jenkins-plugin. Det hade dock varit önskvärt om guiden inkluderade fler konkreta exempel. Vi stötte även på brister i dokumentationen med delar som var ouppdaterade, ej heltäckande och svåra att förstå. Här fann vi stort värde i AI-verktyg som effektiviserade inlärningsprocessen genom att tolka och förklara. Eftersom det finns gott om Jenkins-plugins tillgängliga utgjorde detta en bra grund för vältränade AI-verktyg. Ett Jenkins-plugin består av två delar, en skriven i Java som ansvarar för logik och en skriven i Jelly som ansvarar för det grafiska. Jelly är ett XML baserat skript-språk som vi hade väldigt begränsad eller noll erfarenhet av. Minsta fel i Jelly-koden resulterar i att ett Jenkins-plugin inte dyker upp i Jenkins användargränssnitt. Detta blev oerhört irriterande emellanåt eftersom komplexiteten av mjukvaruutvecklingen fick ytterligare en dimension. Är det vår programkod i Java som är fel eller är det Jelly-koden som inte stämmer? Det tog ytterligare tid för oss att etablera en metod att jobba med felsökning och buggjakt.

För att utveckla Jenkins-plugin krävs att den IDE du valt för projektet använder Maven som byggverktyg för att kompilera, hantera beroenden, packa pluginet, köra enhets- och integrationstester, samt struktur och integrationstester. Ingen av oss hade tidigare erfarenhet av Maven så även här var tidsåtgången lite för stor enligt vår bedömning.
Konceptuellt sett var Docker relativt enkelt att förstå och det är en väldokumenterad teknik. Den tiden vi avsatt för att få det att fungera på ett tillfredsställande sätt var väl värt det. Vad vi förstår är det en värdefull kunskap att besitta för att jobba vidare inom mjukvaruutveckling och CI/CD.

Efter att vi tagit in kunskap om enskilda tekniker och fått en grundförståelse för hur

de fungerar självstående var nästa steg att få teknikerna att fungera tillsammans. Vi började med att få RabbitMQ och EM fungera tillsammans. Det var inte självklart hur den kommunikationen skulle implementeras. Som tur är har RabbitMQ en god dokumentation med exempel på hur man kan lyssna på en kö och implementera en callback-funktion som vi fann väldigt hjälpsam.
På förslag av handledare på Nexer skulle vi undvika att Jenkins blir överbelastat genom att kringgå dess API. Därifrån kom idéen att utveckla en fristående mjukvara vars uppgift är att orkestrera EM instanser och kommunicera med Jenkins via websockets. Examensarbetes begränsade tid tvingade oss att avgränsa arbetet till att endast implementera grundläggande stöd för vidarebefordring av arbetsinstruktioner från EM till Jenkins. Därmed blev målet som sattes upp att implementera en grundläggande modul för vidarebefordring som senare kommer vara lätt att vidareutveckla.

6.4.2 Begränsningar

Allt eftersom vi gjorde framgångar i arbetet blev det mer och mer uppenbart hur värdefullt det hade varit att ha tillgång till ett simulerat inflöde av meddelanden till en RabbitMQ-kö. Syftet med detta hade varit att se hur EM hade presterat under en hög last även om inte den fullständiga visionen av produkten hann bli realiserad under utsatt tid för examensarbetet. Den komponenten vi var mest nyfikna på är ifall EM kunnat validera meddelanden i den takt de anlänt till förutbestämd RabbitMQ-kö.

Bristen på insikt i hur det faktiskt ser ut på företag i dagens läge var även något vi förstod vikten av då vi närmade oss slutet av arbetet. Under en diskussion med en medarbetare på Ericsson nämdes det att många företag är fastlåsta i gamla osmidiga och utdaterade tekniker i sina CI/CD-pipelines och att de finns ett stort behov av en ny lösning som är lätt att migrera till.

Eiffel "communitynsom vi hämtade mycket inspiration från och som består av anställda på Ericsson, Volvo och Axis uppfattade vi vara drabbade av dissonans i avseende hur man ska implementera Eiffel-protokollet för att åstadkomma en eventdriven pipeline. Det hade varit gynnsamt för arbetet ifall det hade funnits mera tydligt uttalade riktningar och riktlinjer.

6.5 Förslag på vidareutveckling

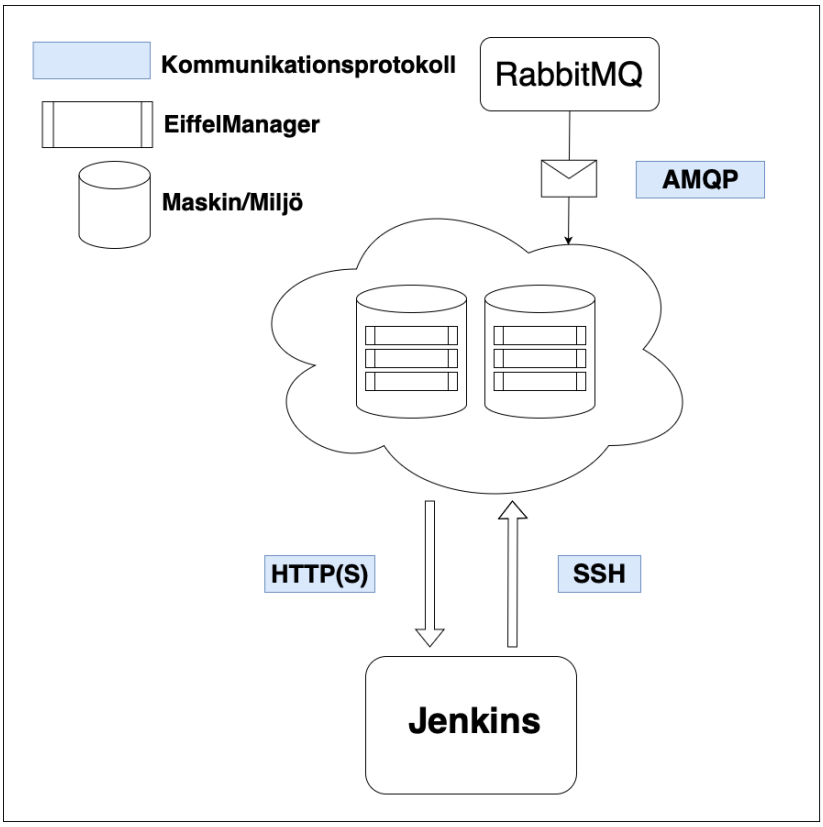
Under arbetets gång har vi identifierat flera utmaningar med den nuvarande designen. Dessa är främst kopplade till resursanvändning, processhantering och brist på överblick vid drift i större miljöer. Vi har därför skissat på en lösning för majoriteten av dessa problem genom att lägga till en mjukvara som vi kallar **Eiffel Orchestrator (EO)**.

Tanken med EO är att den ska avbelasta Jenkins, genom att lyfta ut och hantera logiken för orkestreringen av EM instanser i en annan miljö.

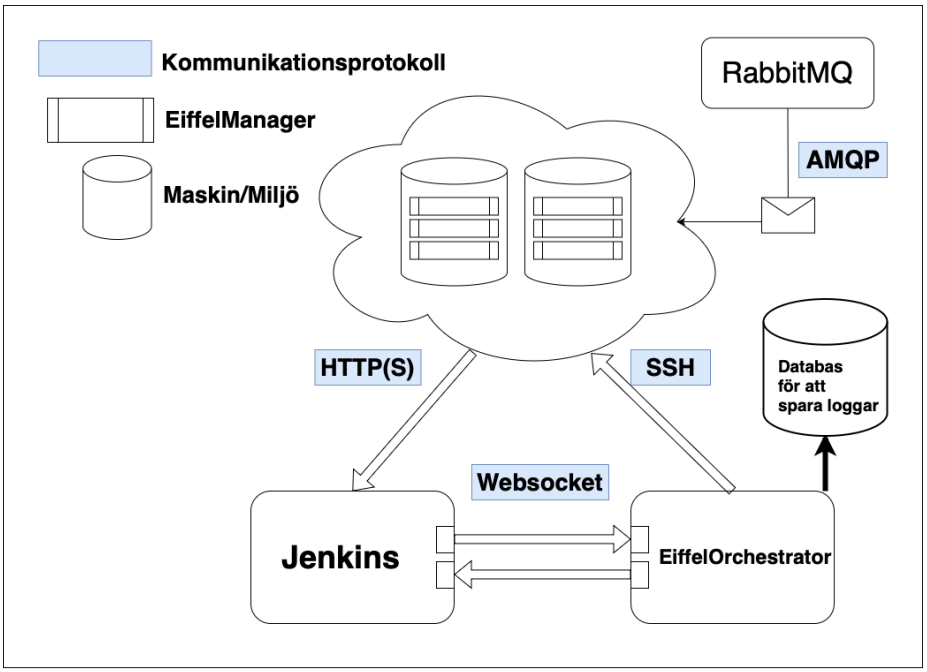
6.5.1 Funktion och ansvar

För att optimera prestanda ska Jenkins endast hantera konfiguration av pluginet, definiera och initiera exekvering av jobb. Detta innebär att EO behöver ha stöd för följande funktionalitet:

- När Jenkins definierar, uppdaterar eller raderar jobb behöver EO kunna ta emot och behandlar följande information:
  - Händelsetyp (skapande/ändring/radering).
  - SSH-uppgifter för att kunna initiera EM instanser på specifik maskin.
  - Uppgifter för att EM ska kunna ansluta och lyssna på RabbitMQ köer.
  - Filterregler ifrån Matchcenter, så att EM kan filtrera inkommande EE.
  - Nödvändig information för att EM ska kunna utlösa jobb i Jenkins via HTTP/HTTPS.
- Stöd för att kunna för att kunna orkestrera(skapa, uppdatera, stänga och övervaka) EM instanser. EM instanserna ska kunna köras på en eller flera maskiner.
- Info ifrån EM ska kunna kommuniceras tillbaka till Jenkins.
- Möjlighet att kunna lagra loggdata i en central databas på valfri server.



Figur 6.1: Schemadiagram som representerar arbetets nuvarande systemdesign



Figur 6.2: Schemadiagram som representerar en vidareutveckling av den nuvarande systemdesignen.

6.5.2 Jämförelse med nuvarande modell

Som figur 6.1(nuvarande systemdesign) och figur 6.2(vidareutveckling) visar är skillnaden att den vidareutvecklade modellen har en extra komponent som lyfter ut logik ifrån Jenkins. Detta kommer hypotetiskt resultera i följande:

- Minskad resursanvändning i Jenkins. Eiffel Orchestrator hanterar lyssnare externt.
- Möjlighet till lastbalansering.
- Förbättrad felsökning och analys genom loggcentralisering.

6.6 Slutsats av diskussion

Arbetet med att utveckla EiffelTrigger som integrerar Eiffel-protokollet med Jenkins har varit en spännande utmaning. Under arbetets gång har vi lärt oss och fått utökad förståelse för hur företag sätter upp och arbetar med CI/CD-miljöer. Vi har lärt oss att använda ett flertal aktuella verktyg/mjukvaror som används inom området och fått insikt i vilka typer av problem som kan uppstå längs vägen.
En av de största utmaningarna var bristen på tillgång till att testa vår slutprodukt i en verklig miljö. Detta gjorde det svårt att förstå hur våra lösningar skulle implementeras och prestera i dessa miljöer.

7
Slutsats

Det här arbetet har bidragit med ett steg närmare en robust lösning för företag som använder Jenkins, för att enkelt kunna sätta upp CI/CD-pipelines som är händelsestyrda med Eiffel-protokollet. För att minska belastningen på Jenkins har arbetet haft fokus på att förflytta logik för bearbetning av Eiffel-event utanför Jenkins. Detta genom att introducera en ny mjukvara som lyssnar, validerar och filtrerar event från en RabbitMQ server. För att sänka tröskeln för att implementera lösningen har arbetet även haft som fokus att skapa nya gränssnitt som bidrar till en användarvänlig miljö.
Arbetet har avgränsats till att matcha examensarbetets deadline. Avgränsning sattes till att endast utveckla grunden för en mer komplex lösning. För att underlätta vidareutveckling presenteras även en skissad lösning som vi anser vara nästa naturliga steg. Den skissade lösningen kompletterar funktioner som den nuvarande lösningen saknar. Detta inkluderar funktioner som förmåga att kunna orkestrera EiffelOrchestor instanser utanför Jenkins och spara loggar på en extern server.
Sammanfattningsvis har arbetet inte bara lagt grunden för en praktiskt tillämpbar lösning, utan även etablerat ett framtidsperspektiv som lägger grunden för en robust händelsehantering i Jenkins.
Med detta sagt ska det betonas att en av de största utmaningarna i arbetet var bristen på tillgång till att testa vår slutprodukt i en verklig miljö. Det har alltså varit svårt att avgöra hur vår lösning skulle implementeras och prestera i en verklig miljö. Därför går det bara att dra hypotetiska slutsatser som indikerar på att lösningar som presenteras i detta projekt är funktionella i praktiken.

Litteraturförteckning

[1] S. Nandgaonkar and V. Khatavkar, "Ci-cd pipeline for content releases," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, 2022, pp. 1–4. [Online]. Available: 10.1109/GCAT55367.2022.9972129

[2] "Jenkins user documentation | jenkins docs," 2025, [Hämtad: 20-May-2025]. [Online]. Available: https://www.jenkins.io/doc/

[3] "Rabbitmq: One broker to queue them all | rabbitmq," [Hämtad: 20-May-2025]. [Online]. Available: https://www.rabbitmq.com/

[4] "Amqp 0-9-1 model explained | rabbitmq," [Hämtad: 20-May-2025]. [Online]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts

[5] "Eiffel community | eiffel docs," [Hämtad: 20-May-2025]. [Online]. Available: https://eiffel-community.github.io/

[6] "What is docker? | docker docs," [Hämtad: 20-May-2025]. [Online]. Available: https://docs.docker.com/get-started/docker-overview/

[7] "About github and git - github docs," [Hämtad: 20-May-2025]. [Online]. Available: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

[8] "Introduction – maven," [Hämtad: 20-May-2025]. [Online]. Available: https://maven.apache.org/what-is-maven.html

[9] "Rabbitmq tutorial - producer/consumer | rabbitmq," [Hämtad: 20-May-2025]. [Online]. Available: https://www.rabbitmq.com/tutorials/tutorial-three-java

[10] Oracle, "Chapter 6. the java virtual machine instruction set," [Hämtad: 20-May-2025]. [Online]. Available: https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html

[11] IBM, "Ibm sdk, java technology edition 8," 2 2024, [Hämtad: 20-May-2025]. [Online]. Available: https://www.ibm.com/docs/en/sdk-java-technology/8?topic=reference-jit-compiler

[12] "Vad är devops? så här fungerar devops | microsoft azure," [Hämtad: 20-May-2025]. [Online]. Available: https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-is-devops

[13] "Alpine user handbook - alpine linux documentation," [Hämtad: 20-May-

2025]. [Online]. Available: https://docs.alpinelinux.org/user-handbook/0.1a/index.html

[14] JSON, “Json,” [Hämtad: 20-May-2025]. [Online]. Available: https://www.json.org/json-en.html

[15] “Eiffel community | youtube,” [Hämtad: 20-May-2025]. [Online]. Available: https://www.youtube.com/@EiffelCommunity

[16] “Plugin tutorial,” 2025, [Hämtad: 20-May-2025]. [Online]. Available: https://www.jenkins.io/doc/developer/tutorial/

A

Exempel på ett giltigt Eiffel-event

```
"{
   "meta": "EiffelArtifactCreatedEvent",
   "version": "3b.0.0,
   "time": 17158643001123,
   "id": 12cf27fa-598-412-932-9331-90b524878943",
   "source": {
      "domainid" "ci.mycompany.internal",
      "host" "builder01.ci.mycompany.Internal",
      "name" "artifact-gentrator",
      "uri" "https://ci.mycompanyinternal/build/12345/artifocts"
      "tags": [
         "build",
         "ci",
         "artiffact" "release:1.52?"
      },
      "secuity": {
         "authorIdentity": "build-system@mycompany.com",
         "integrityProtection" [
            "alg": "sha256",
            "tags" ["main","bindry","stable"],
            "size" 14523672,
            "hash" "d7a8fb53877790809469ca9abcb0082e4f8d65151
                    e46d2cd72d0bf7c9e352",
            "sha256": "app-1.5.3.jar.asc",
            "tags" ["signature",gpg"],
            "size" 612
            "hash" "94ee058335e587e501cc4bf90613e014f00AA7b0bc
                    c7c648fd865a2af8a23cc22"
         ]buildCommand": "/gradlew build"
         "implements": "urn:eiffel:document:bi-spec@1.5.2",
         "name"s"con.mycompany.buiidspec",
         "version": "1.5.2"
      "repository": {
         "name" "artifact-repo",
         "uri" "https:/artifacts.mycov.com/repo/com/mycompany/
      },
      "links": [
         "type" "CNTEXT", 6e14600e-bb6o-4c73-ae25-7f58c2295aa"
         "type" "CAUSE", 2a75aeb-112-4959-9e57-0bda46ec50b3"
         "type" "ENVIRONMENT", fb0a4bce-fc03-4e20-e6e6e53ed19"
         "type" "PREVIOUS_VERSION", 90ab4532-f0ea-41e-a91a-c-
      ], c15792f8cBE"
    }
```

Figur A.1: Ett giltigt Eiffel-meddelande som är rikt på data.

B

Intervju

Det här är en intervju med arbetets handledare från Nexer. Syftet med intervjun är att få en djupare förståelse för projektets bakgrund, mål och förväntningar, samt identifiera vilka behov och problem som detta initiativ försöker adressera. Fokus ligger på att samla in underlag kring syfte, användningsområden och den planerade nyttan med lösningen.

B.1 Berätta kortfattat vad din roll är och vad du jobbar med samt hur det återkopplar till Eiffel

Min roll är bred. Jag är en systemadnistratör, cloud ingenjör och CI/CD ingenjör samt "software integrator". Eiffel kommer främst in i min roll som CI/CD ingenjör samt software integrator. Det är i den rollen som Eiffel kommer till användning för att få traceability och modularitiet för att bygga komplexa mjukvaror .

B.2 Hur ser dagens CI/CD utveckling ut i praktiken?

Dagens CI/CD är extremt spridd. Alla använder olika teknologier för att bygga mjukvaror. Inom bara ett företag kan du ha 5 olika CI/CD mjukvaror. Jenkins är utan tvekan den mest populära inom embedded pga dess flexibilitet. På grund av denna flexibiliteten integreras den ofta med alla olika protokoll och infrastrukturer. Inklusive rabbitmq.

B.3 Hur ser det nuvarande stödet ut för händelsestyrda arbetsflöden i Jenkins, och varför räcker det inte till?

Stödet är extremt begränsat. Det finns plugins som stöttar händelsestyrda flöden för individuella produkter, t ex gerrit/bitbucket. Men det finns knappt några som stödjer hela flöden.

B.4 Hur kan det här arbetet bidra till att lösa dessa brister och förbättra arbetsflödena i praktiken?

När du har komplexa produkter så är det många delar som ska gå ihop. Detta arbetet gör det möjligt att bygga samman komplexa applikationer på ett enkelt vis genom att snabbt reagera på avslutade tester. Samt att skapa produkter när beroenden är uppfyllda. Det är så mycket som stora företag kämpar med i silon, detta kan hjälpa att lösa beroenden mellan dem!

B.5 Hur stor är den nuvarande efterfrågan på att integrera Eiffel-protokollet i Jenkins?

Inom embedded världen i Göteborg så är efterfrågan stor . Alla företagen kämpar med samma problem. Hur bygger man en komplex mjukvara utan att skapa silon. Pluginet möjliggör denna kommunikation mellan alla silos och kan garantera den brist på kvalitet som existerar hos de flesta företagen.

Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY