Automatiska tester med Post-man i IBM Cloud

Valtteri Eronen

Examensarbete

Informationsteknik

2019

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	
Författare:	Valtteri Eronen
Arbetets namn:	Automatiska tester med Postman i IBM Cloud
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	

Sammandrag:
Examensarbetet redovisar för hur man kan skapa automation i form av automatiska tester för integrationer som är byggda med IBM:s egna utvecklade mjukvara App Connect Studio, hur man skapar Postman-samlingar och kör dem med kommandotolken och hur man konfigurerar de automatiska testerna till en delivery pipeline som är en del av en automatiserad verktygskedja. Verktygskedjan byggs upp i IBM Cloud med produkten IBM Cloud DevOps. Grund principen för att ha en delivery pipeline och automatiska tester är för att spara tid då man inte behöver utföra manuella tester varje gång det sker en ändring i integrationerna. Istället skulle verktygskedjan aktiveras och exekvera testerna på alla integrationerna varje gång det sker en ändring i versionhanteraren. Alla tester som skapats är regressionstester och varje enskilda integration har sin egen Postman samling med specifika tester just för denna enskilda integration. Postman-samlingarnas grundtanke är att samla ihop ett antal HTTP-anrop som kommunicerar med en REST API där varje integration fungerar som en slutpunkt som går att anropa i API:n. Själva regressionstesternas funktionalitet är i grund och botten det att man med ett HTTP-anrop skickar ett meddelande till API:n som pekar på integrationen som sedan efter prosseseringen skickar responsen tillbaka till API:n. Nästa HTTP-anrop i samlingen tar upp responsen av integrationen och utför regressionstesterna som sedan visas upp som lyckade eller misslyckade i loggarna i IBM Cloud. I examensarbetets teoretiska del ges det en översikt på olika nivåer av test som kan utföras. Det redovisas centrala skillnader mellan enhetstester, integrationstester och E2E-tester och när det lönar sig att använda dem. I den teoretiska delen redovisas också metoder för hur man kan skapa mjukvarutester beroende på hurdan mjukvaran är som bör testas. Metoderna som presenteras i examensarbetet är den s.k. ”Big Bang” metoden, ”Top-to-Bottom” och ”Bottom-to-Top.

Nyckelord:	IBM Cloud, Postman, automatiska tester, delivery pipeline,
Sidantal:	27
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	
Author:	Valtteri Eronen
Title:	Automatic tests with Postman in IBM Cloud
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	

Abstract:
The thesis presents how to create automation in the form of automatic tests for integrations built with IBM's own developed software App Connect Studio, how to create Postman collections and run them with the command prompt and how to configure the automatic tests for a delivery pipeline that is part of an automated toolchain. The toolchain is built in IBM Cloud with the product IBM Cloud DevOps. The basic principle of having a delivery pipeline and automatic tests is to save time when you do not have to perform manual tests every time there is a change in the integrations. Instead, the toolchain would be activated and execute the tests on all the integrations every time there is a change in the version control. All tests created are regression tests and each individual integration has its own Postman collection with specific tests for just this one integration. The basic idea of the Postman collections is to collect several HTTP requests that communicate with a REST API where each integration acts as an endpoint that can be called in the API. The functionality of the regression tests themselves is basically that with an HTTP request a message is sent to the API, which points to the integration which then after the processing sends the response back to the API. The next HTTP call in the collection addresses the response of the integration and performs the regression tests that then appear as successful or unsuccessful in the logs in IBM Cloud. In the thesis's theoretical part, an overview is given at different levels of tests that can be performed. Central differences between unit tests, integration tests and E2E tests and when it’s useful to use them. The theoretical part also describes methods for how to develop software tests, depending on what the software is that is going to be tested. The methods presented in the thesis work are the so-called "Big Bang", "Top-to-Bottom" and "Bottom-to-Top methods.

Keywords:	IBM Cloud, Postman, automatic tests, delivery pipeline
Number of pages:	27
Language:	Swedish
Date of acceptance:	

Innehåll

1 Inledning 1

1.1 Syfte och Målsättning 2
1.2 Metoder 3
1.3 Struktur 3

2 Mjukvarutester 4

2.1 Enhetstester (Unit testing) 5
2.2 Integrationstester (Integration Tests) 6
2.3 E2E tester 7

3 Testmiljö och Verktyg 8

3.1 IBM Cloud 8
3.1.1 IBM Cloud DevOps 9
3.1.2 App Connect Proffesional 10
3.2 Postman 13
3.2.1 Newman 14

4 Automatiska Tester med Postman 16

4.1 Miljöer 16
4.1.1 Lokal miljö 16
4.1.2 Global miljö 16
4.2 Skapandet av en samling 17
4.3 Regressionstester 18
4.3.1 Exempel 19
4.4 Strukturen i verktygskedjan 20
4.4.1 Skript 21
4.4.2 Omgivningarna i delivery pipeline (Dev/Test/Prod) 22
4.4.3 Konfigurering av omgivningen (Development) 22
4.4.4 Integration av testerna med delivery pipeline 23

5 Slutsatser 25

Källor / References 27

Definitioner

Hyper Text Protocol (HTTP): det protokoll som styr kommunikationen på webben. – HTTP styr hämtandet och skickandet av webbsidor mellan webbserver och webbläsare.

Användargränssnitt (UI): den typ av användargränssnitt som är allenarådande på persondatorer. Man hanterar datorn genom att på bildskärmen peka på, klicka på och flytta bilder (ikoner) som står för program och filer.

E2E Test (End-to-End Test): är en metod för att testa arbetsflödet på en applikation från början till slut.

Testdriven systemutveckling (TDD): skriven kod testas ofta då blir koden godkänd eller underkänd. Att skriva program som testar koden ingår i utvecklingsprojektet. Koden måste klara alla tester till 100 procent innan man får använda den.

Programmeringsgränssnitt (API): specifikation för hur program fungerar ihop: de regler man måste följa när man skriver program som ska fungera ihop med ett annat program. – Fördelen med programmeringsgränssnitt är att programmeraren inte behöver känna till alla detaljer i det bakomliggande programmet.

1 INLEDNING

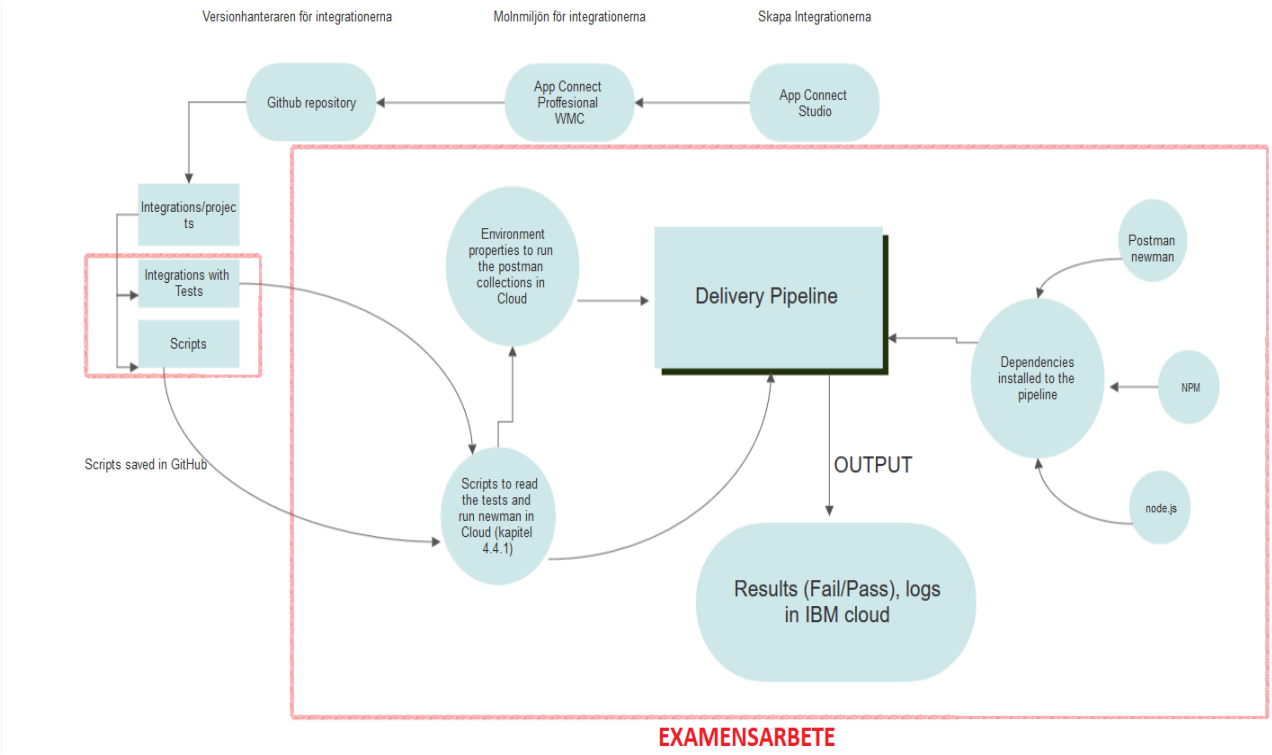
Det här examensarbetet baserar sej på ett projekt jag varit delaktig Iivid IBM Finland. Projektet handlar om en Webb/Cloud integration med diverse mikrotjänster. Grundidén med integrationen är att omvandla gamla datasystem och arbetsflöden till moderna och molnbaserade.

Kunden som beställt projektet är ett stort finländskt företag inom detaljhandelsbranschen. Projektet startade med att utreda vilka procedurer och verktyg som kommer att behövas för att göra upprätthållningen lätt och smidig för det teamet som kommer att upprätthålla projektet. En sak som kom fram var att skapa automatiska tester som en del av vår Continuous Integration Pipeline i IBM Cloud. Testerna som planerades var huvudsakligen regressionstester som testar logik som implementerats till de olika mikrotjänsterna.

I examensarbetet ingår inte en detaljerad redovisning av projektets processer och arbetsflöden, utan behandlar istället de olika ämnesområdena på ett mer generellt plan. Projektet fungerar mer som en grund för saker som jag kommer att redovisa för.

1.1 Syfte och Målsättning

I Figur 1 ges en visuall översikt över projektet som helhet



Figur 1 Projekt helhet

Delarna som har gjorts inom ramen för examensarbetet är omringade med rött. Examensarbetet går igenom varje enskilda del i figuren för att ge läsaren en bättre uppfattning hur man kan skapa automatiska tester med Postman och få de som en del av en delivery pipeline.

Syftet med examensarbetets praktiska del är att redovisa hur man kan använda Postman i samband med IBM Cloud för att skapa regressionstester till en Continuous Integration Pipeline. Syftet med att göra automatiska tester är att teamet som börjar och upprätthålla integrationerna inte behöver genomföra sina egna tester varje gång det sker någon mindre ändring, utan de färdigt gjorda testerna kollar direkt då man publicerar ändringarna till versionshanteringen. Då kan de gå och titta på loggarna på de tester som körts och kolla var någonstans felet har skett och lösa felet utan att hamna undergå något desto större undersökningar.

Målsättningen med det praktiska arbetet var att få regressionstesterna gjorda på alla gränssnitt där det har implementerats någon logik som modifierar på ingående data. I examensarbetet redogörs hur man kopplar automatiska testerna i en Continuous Integration Pipeline i IBM Cloud.

Målsättningen med den teoretiska delen av examensarbetet är att redovisa vad automatiska tester egentligen är och hurdana processer som krävs för att skapa tester

1.2 Metoder

För skapandet av testerna valdes Postman pga. att det har använts tidigare då man genomför manuella tester och därmed var Postman ett bekant verktyg. Sedan fanns det redan massor med färdigt gjorda HTTP anrop där lösenord, användare etc. har sparats som underlättade arbetet. Slutligen var det ändå ett beslut som gjordes av arkitekten och vårt Team Lead. (Postman 2019)
IBM Cloud DevOps är ett verktyg där du kan skapa en Continuous Integration Pipeline, DevOps valdes som verktyg pga. att det är IBM:s egna verktyg och resten av applikationen var byggd på IBM Cloud plattformen (IBM Cloud 2019).

1.3 Struktur

I kapitel 2 ges en översikt över olika typer av automatiska tester. I kapitel 3 redogörs arbetsmiljön och de verktyg som använts i praktiska delen av examensarbetet. I kapitel 4 går man igenom hela processen med att sätta upp en Pipeline i IBM Cloud. I kapitel 5 redovisas det vad man behöver veta för att skapa tester med Postman och vad som bör göras för att kunna integrera det till Pipelinen. I kapitel 6 går vi igenom testerna som skapats och motiveringarna varför vi skapat dessa testerna. I kapitel 7 redogörs resultaten dvs. hur lyckades de slutliga testerna, uppkom det problem och kunde testerna ha utvidgats eller förbättras på något sätt. I sista kapitlet dras en slutsats över hela examensarbetet.

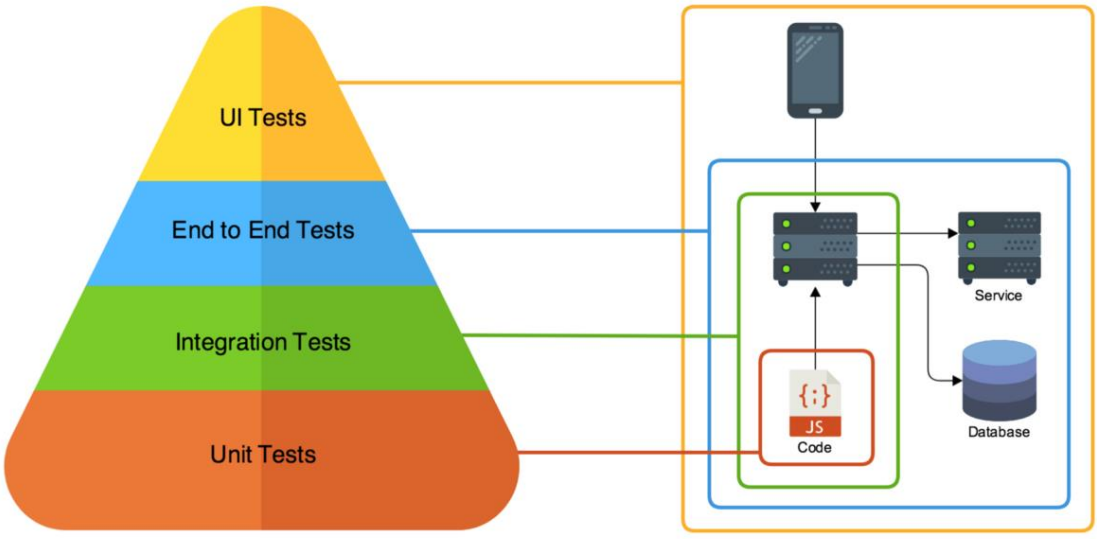
2 MJUKVARUTESTER

När man testar mjukvara så använder man sig oftast av olika nivås tester beroende på vad man vill nå med testerna. Man brukar ofta dela in testerna i 3–4 kategorier.

På lägsta nivån är Enhetstester (unit tests) som testar logik och enskilda funktioner. På följande nivå kommer integrationstester som testar hur koden i applikationen fungerar t.ex. ihop med en databas dvs. den testar om 2 eller flera system fungerar tillsammans. Nästa nivå är E2E tester dvs. ”end-to-end” tester som egentligen testar hela applikationens funktionalitet dvs. databaser, servrar, API:n etc.

Om applikationen som testas har ett grafiskt användargränssnitt så finns det någonting som kallas UI-tester där man kontrollerar om alla länkar fungerar och att t.ex. tiden som det tar att navigera mellan olika sidor inte tar för länge.

I Figur 2 finns det en grafisk representation på hur de ovannämnda testnivåerna kunde delas upp och den ger en bra bild på hur omfattande olika nivåer av testerna är (Peck 2017)

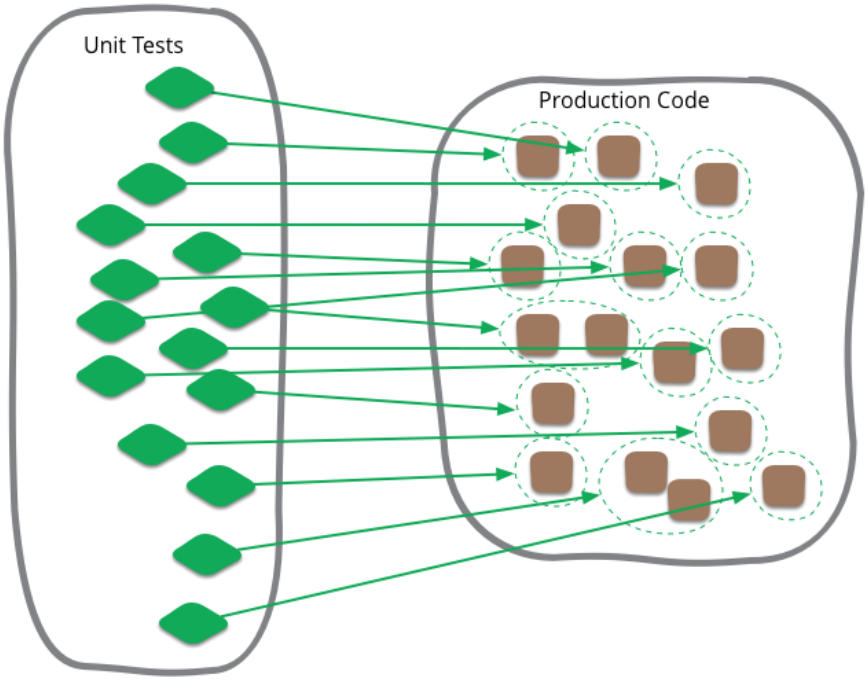


Figur 2 En testpyramid på olika nivåers tester i mikrotjänster.

2.1 Enhetstester (Unit testing)

Enhetstester (Eng. unit tests) är den mest använda formen att skriva tester och används huvudsakligen för att testa små mängder av kod, logik eller enskilda funktioner i en applikation. Enhetstester är lätta att skriva och borde inte ta längre en ett par sekunder att genomföra pga. att de inte är beroende av andra system eller funktioner utan endast av den koden som testas. Ett utvecklingsätt där man använder sig massor av enhetsterser är TDD (test driven development).Där är det vanligt att man skriver enhetstesterna färdigt före man börjar skriva den egentliga koden till applikationen och det fungerar som ett slags ”säkerhetsnät” för koden så att man skulle märka eventuella misstag i ett tidigt skede.

Ett exempel på en enhetstest kunde vara att applikationen i fråga är en kalkylator och du vill räkna vad 2 + 2 är och testet har lyckats om du får ett svar som är 4.



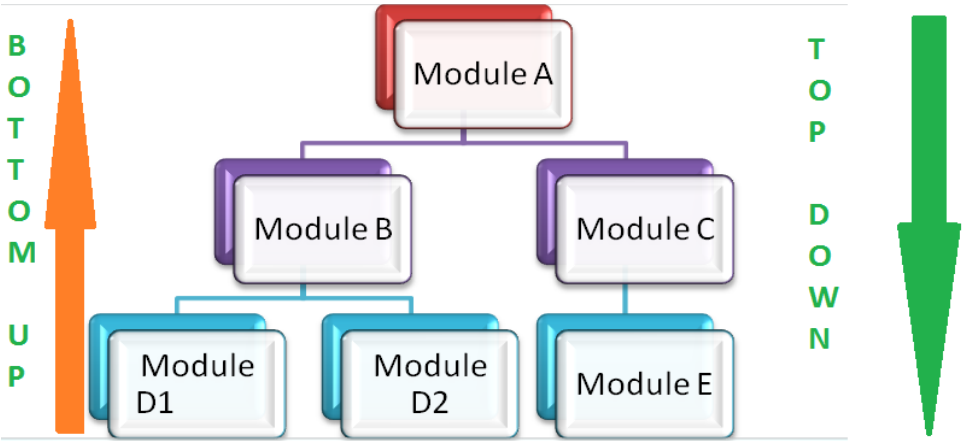
Figur 3 Enhetstester

Här ovan i figur 3 finns en representation vad enhetstester går ut på, varje enskilda test testar en liten del av hela applikationen. (Fowler 2014)

2.2 Integrationstester (Integration Tests)

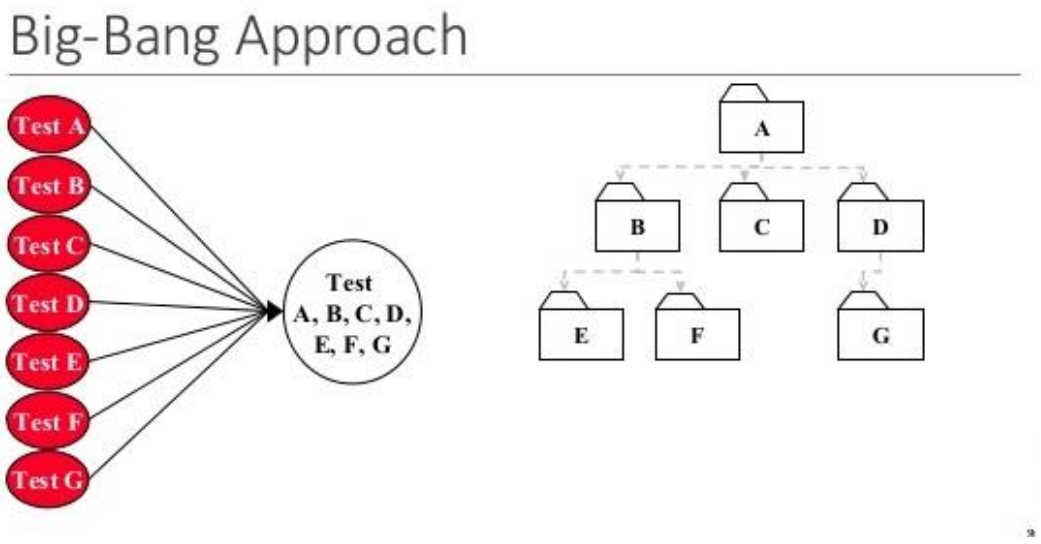
Integrationstester används då när man vill testa att flera moduler i en applikation fungerar tillsammans på ett önskat sätt.
Metoder för att skapa integrationstesterna:

”Top to Bottom” eller **”Bottom to Top”** metoden där man testar funktionaliteten mellan moduler som är beroende av varandra. I figur 4 kan man se en grafisk representation på hur Top to Bottom modellen fungerar. (Kanchan 2017)



Figur 4 Visar hur Top-to-Bottom modellen fungerar.

”Big Bang” metoden är mer sällsynt när man nuförtiden bygger upp applikationer som mikrotjänster som kan fungera självständigt utan att vara beroende av andra moduler. Då när man använder denna metod så är oftast alla moduler beroende av varandra och det går inte att testa separat. Nackdelen med att använda denna metod är att felhanteringen på enskilda moduler blir väldigt svår. I figur 5 kan man se hur man använder sig av Big Bang metoden där alla tester grupperas i ett större test (Kanchan 2017)



Figur 5 Big Bang metoden för Integrationstest

Integrationstesterna är mer komplexa än enhetstesterna för att det oftast kräver någon form av förberedelse innan man kan köra testerna. Ett exempel kunde vara att man måste sätta upp en databas.

Huvudsakligen så borde man ha mindre integrationstester än enhetstester och integrations tester används endast då när man testar hur system fungerar tillsammans eller då om en funktion är för komplex för att göra enhetstester.

Ett exempel på en integrationstest kunde vara att man försöker spara kundinformation till en databas. Då använder man sig av en funktion som sparar kundinformation och den funktionen försöker kommunicera med en databas.

2.3 E2E tester

E2E tester dvs. ”end-to-end” tester är även kända som ”acceptans tester” eller ”funktionalitetstest”. E2E tester testar hela funktionaliteten på en applikation och kan därför vara väldigt tidskrävande och svåra att skriva.

Testerna kan ta väldigt länge att köra p.g.a att den testar all logik och alla funktioner i applikationen. Om applikationen har ett användargränssnitt så inkluderas det också med i testerna.

Ett exempel på ett E2E test är att om applikationen i frågan är en webbutik med ett användargränssnitt så då måste man kolla att alla länkar fungerar, applikationen får kontakt med alla diverse databaser, ”köp korgen” fungerar på rätt sätt så att man kan sätta till och ta bort varor etc. Så det är fråga om väldigt stora tester som kan ta flera veckor eller t.om månader att skriva.

3 TESTMILJÖ OCH VERKTYG

Verktyg som användes för att skapa testerna och miljön för testerna var huvudsakligen Postman och IBM Cloud DevOps. Pipelinen har gjorts med IBM Clouds DevOps som möjliggör skapandet av verktygskedjor för applikationer. Själva testerna har skapats med Postman’s egna verktyg Postman Newman (Postman 2019b). Newman möjliggör att man kan köra Postman-samlingar (Postman 2019c) via kommandotolken precis på samma sätt som man skulle göra när man använder Postman applikationen. App Connect Professional (IBM 2018) används till att skapa Integrationerna och innehåller inbyggda anslutningar till hundratals molntjänster, förpackade och privata lokaltillämpningar, som inkluderar affärssystem (Eng. ERP - Enterprise Resource Planning), kundvårdssystem (Eng. CRM - Customer Relationship Management), databaser och webtjänster

3.1 IBM Cloud

IBM Cloud är en uppsättning av molntjänster som kan erbjuda publika, privata och hybrida miljöer i en katalog på över 170 produkter i 16 olika kategorier (IBM Cloud Products 2019).IBM Cloud erbjuder PaaS (Eng. platform as a service) och IaaS (Eng. infrastructure as a service)

PaaS är hårdvara, nätverk eller operativsystem som tjänst som tillhandahålls över internet. Det betyder att kunden har sina egna program men köper/hyr allting som behövs för att köra dem. (IT-Ord 2019e)

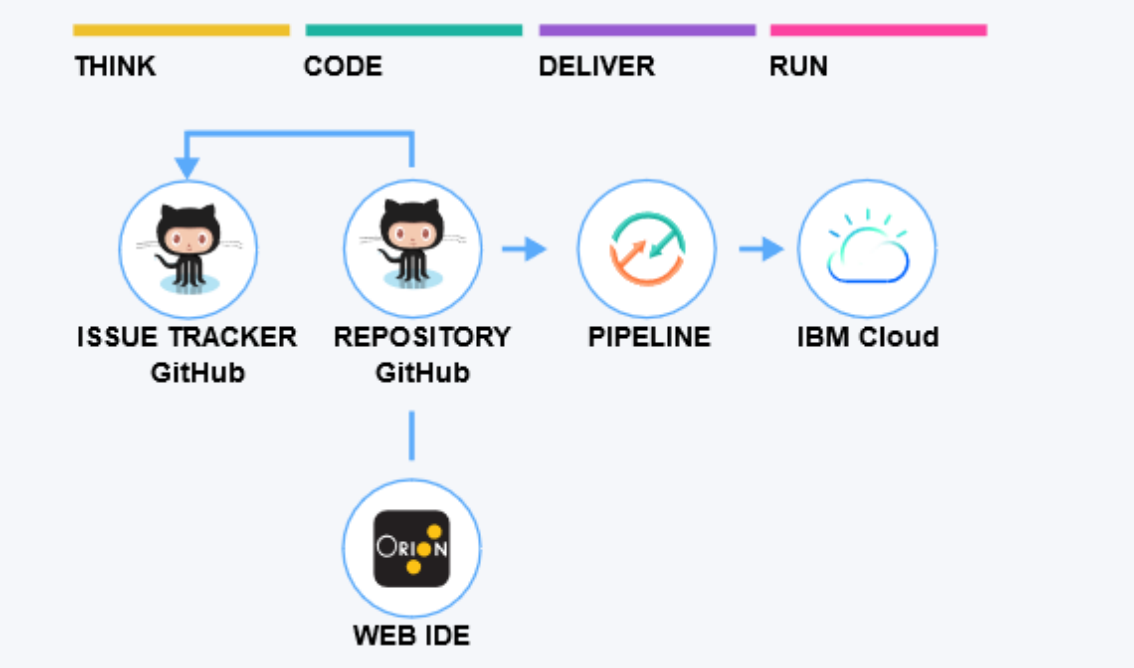
IaaS innebär att ett företag hyr servrar och annan utrustnings som ägs av ett annat företag. Kunden kör sina egna program, inklusive operativsystem, på utrustningen, som finns hos

uthyraren. Ofta betalar kunden bara för det utrymme och den kapacitet som faktiskt utnyttjas. (IT-Ord 2019f).

3.1.1 IBM Cloud DevOps

DevOps är ett allt mer vanligare sätt att leverera programvara som används av utvecklingsteam för att bygga, testa, distribuera program i korta iterationer utan att kvaliteten lider. DevOps är relevant för alla typer av mjukvaruprojekt oavsett arkitektur, plattform eller syfte. Vanliga användningsfall inkluderar: mobila applikationer, applikationsintegration och modernisering och multicloud hantering (IBM DevOps, 2019)

I figur 6 ges det en överblick över hur en applikation kunde byggas med IBM Cloud DevOps.



Figur 6 Applikationens struktur

THINK – är där man som en utvecklare sätter in alla sina idéer dvs. det fungerar som en backlog för applikationen var det finns alla uppgifter som skall göras.

CODE – där man har sin versionhanterare och sitt editeringsverktyg som man använder för att editera koden med.

DELIVER – är där man har byggt upp en delivery pipeline som oftast fungerar med en trigger som kunde t.ex. aktiveras då någonting ändras i versionhanteraren.

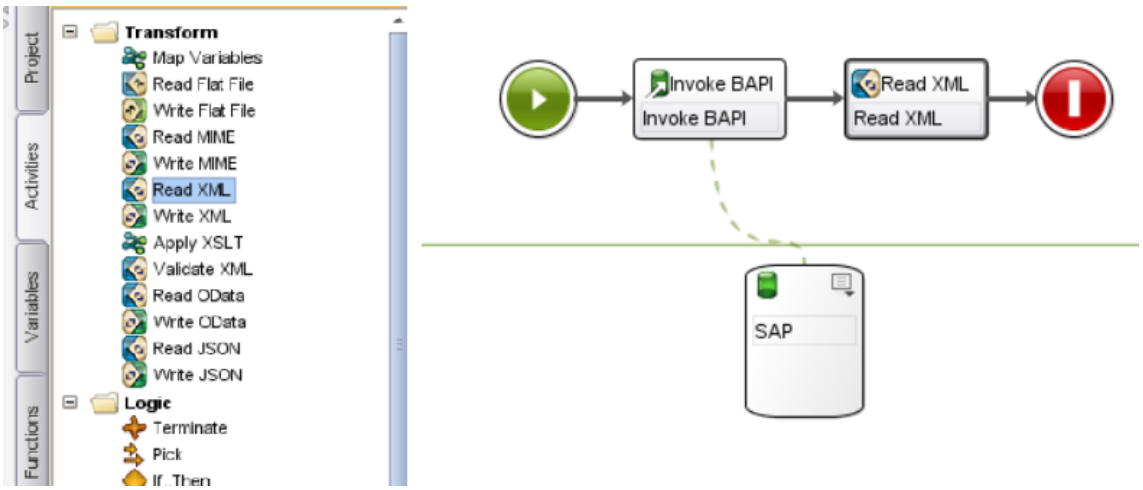
RUN – är där applikationen körs i IBM Cloud som t.ex. en Cloud Foundry App eller som en Docker / Kubernates container.

3.1.2 App Connect Proffesional

Integrationslösningarna skapas med IBM:s egen mjukvara App Connect Studio där man skapar ”arbetsflöden” dvs. en konfiguration som innehåller färdigt skapade adapters som möjliggör t.ex. mottagandet av information från SalesForce väldigt lätt.

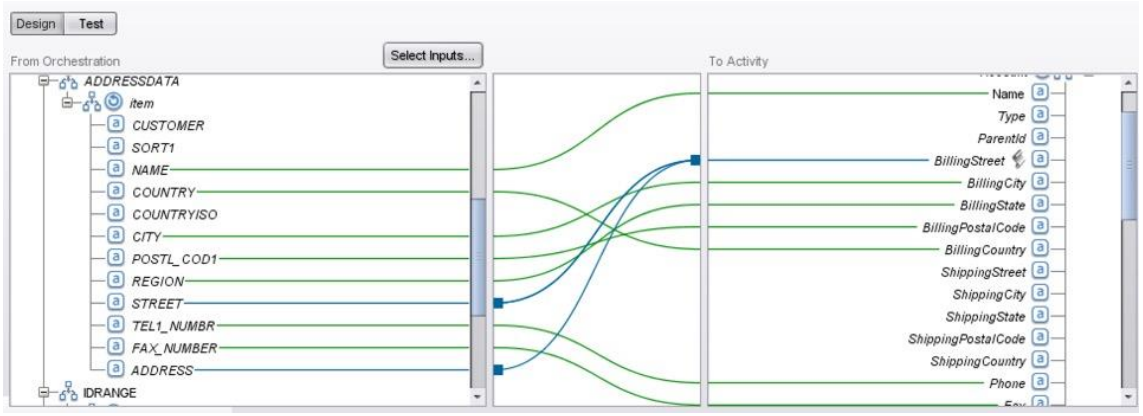
Integrationerna sker i stort sätt utan att behöva skriva kod. I stället kopplar man input data med det data man vill skicka till nästa aktivitet grafiskt med linjer, dvs. man fyller på det sättet ett tomt XML-schema med data. Mellan varje datakoppling går det att implementera egna skräddasydda funktioner som har skrivits i JavaScript.

Ett exempel kunde vara att man tar emot data från SAP med en färdigt inbyggd SAP adapter som tar kontakt till SAP servrar. Sedan får du som respons av SAP ett meddelande som kommer in i XML format som man läser med ”Read XML” aktivitet, i aktiviteten kan man fylla sitt eget schema lätt med att dra en koppling från responsdata till sitt egna schema (exempel på aktiviteter och hur konfigurationen kunde se ut i figur 7).



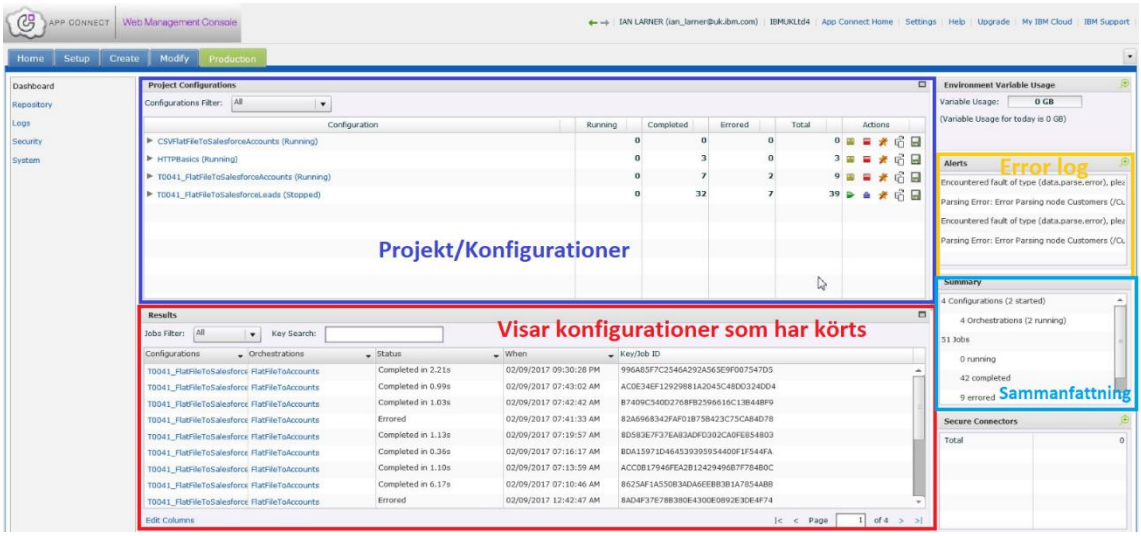
Figur 7 Exempel på aktiviteter och konfigurationen

I figur 8 visas det hur man drar kopplingar mellan responsdata och schemat som man vill fylla med data.



Figur 8 Read XML

App Connect Web Management Console (WMC) fungerar som ett grafiskt gränssnitt där man har en översikt på alla integrationer som har utvecklats. I figur 9 kan man se en överblick på App Connect Web management Console.



Figur 9 App Connect Web Management Console

App Connect Web management Console fungerar som en slags instrumentbräda till de integrationsprojekt som har skapats med App Connect Studio. Vyn går att skräddarsys enligt de behov man har. Standardvyn har en lista över projekt som har skapats och hur många gånger en konfiguration har körts. En log på alla konfigurationer som har körts, den berättar om jobbet som körts har misslyckats eller gått igenom utan problem genom att sätta statuset till errored eller avslutat.

På högra sidan finns det en detaljerad fellogg på hela omgivningen dvs. på alla projekt som körs i denna omgivningen. Där går det att identifiera fel i anslutningen till t.ex. en server eller om det har skett några fel i konfigurationerna på projektet t.ex. ett lösenord är fel. Under detaljerade felloggen finns det en sammanfattning över omgivningens alla projekt, hur många projekt det finns och säger vad för status dom har (undeployed, running, stopped)

3.1.2.1 Funktionalitet

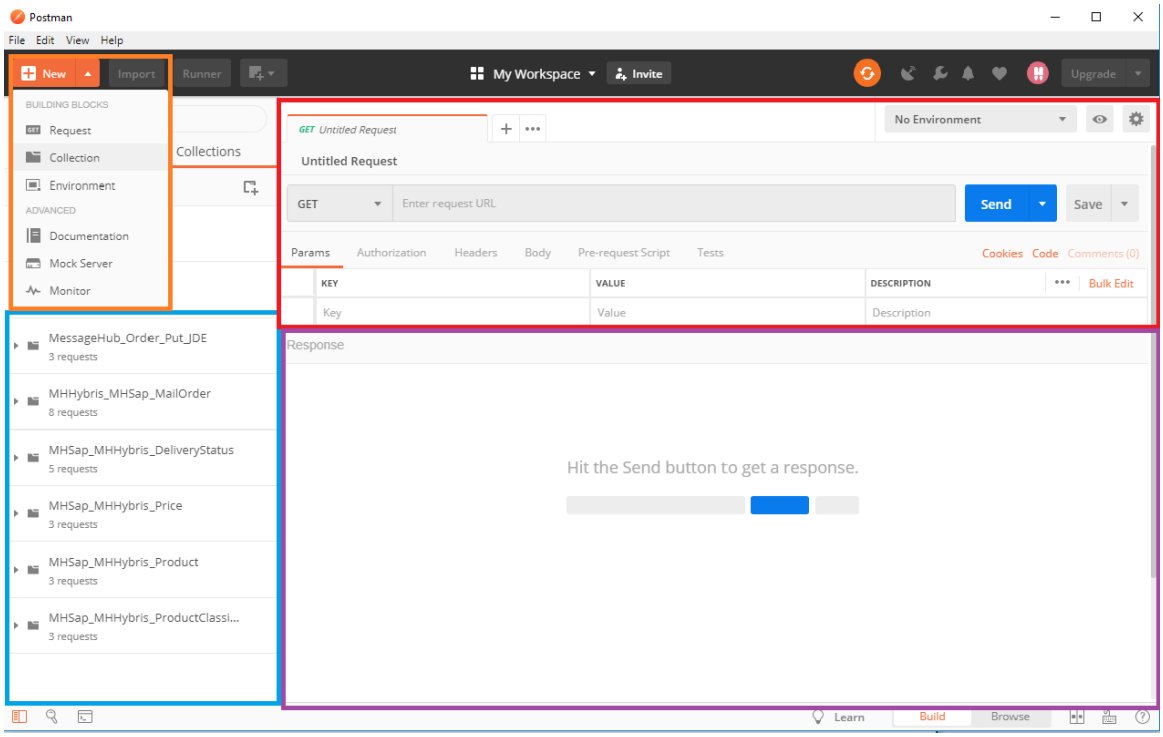
Sättet hur man startar ett projekt automatiskt är genom att skicka via API:n ett meddelande till en topic (t.ex. Orders) som är kopplat till en trigger som är kopplat till integrationen/projektet. Projektet processerar meddelandet och skickar det processerade meddelandet vidare till nästa topic varefter t.ex. slutanvändaren (kunden) tar meddelandet emot eller sedan fortsätter meddelandet vidare till nästa process.

3.2 Postman

Postman är ett API verktyg där man kan skicka HTTP-anrop, skapa olika miljöer och dela upp anropen enligt egna arbetsytor (Eng. workspace). Det är möjligt att göra olika typer av HTTP-förfrågningar - GET, POST, PUT, PATCH och DELETE.

Anrop kan organiseras i samlingar (Eng. collections), man kan köra en eller flera samlingar på en gång med hjälp av Postman Collection Runner där man kan välja mängden iterationer, skapa fördröjningar mellan anropen och spara loggarna på alla anrop som körts (Postman 2019d). Man kan även skapa test för att t.ex. kontrollera att responstiden är inom en viss ram.

I figur 10 kan man se hur det grafiska gränssnittet ser ut i Postman.



Figur 10 Postman användargränssnitt

- man kan skapa nya anrop, samlingar och miljöer.
- man ser vilka samlingar man redan har skapat, inne i varje samling finns det flera anrop
- här finns allt som behövs för att skapa ett anrop, man ska sätta stigen som för till rätta plats i din API. headers, autentisering, meddelandet man skickar om det är en POST, förhandsskript, tester och val av miljö.
- här kommer responsen av anropet

3.2.1 Newman

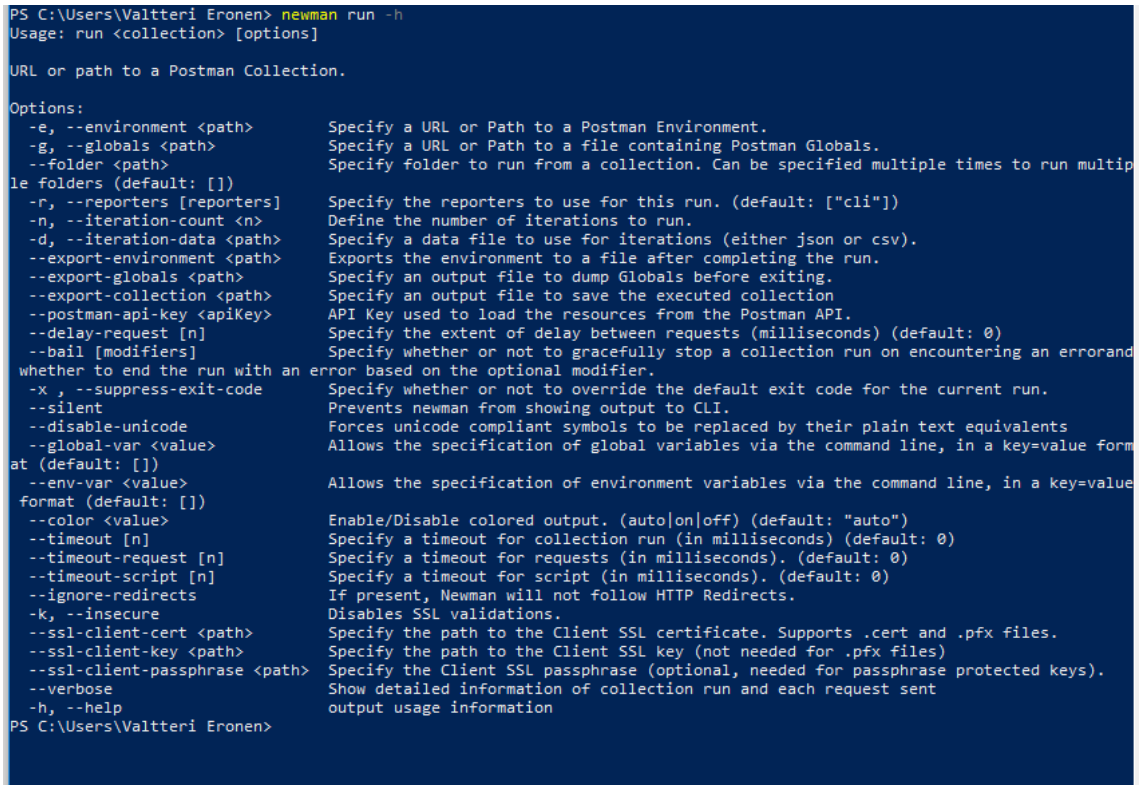
Newman är ett verktyg var man kan köra Postman samlingar (Eng. Collections)direkt från kommandotolken. Den är byggd så att man enkelt kan integrera den med kontinuerliga integrationssystem. Newman är byggd på Node.js (Postman 2019b).

Newman går att hämta från npm (Node Package Manager). Det kräver att man har installerat en node version (NodeJS, 2019) och en version av npm (npm 2019).

Installation av Newman:

```
$ npm install -g newman
```

I figur 11 kan man se vanligaste kommandon som man behöver för att köra samlingar via kommandotolken, varav viktigaste är de två första (-e, -g) som specificerar lokala och globala variabler som du använt in dina HTTP anrop.



Figur 11 Newman hjälp

Med Newman kan man köra samlingar som man exporterade till hårdskivan eller URL:en som pekar på din samling. Oftast behöver man också specificera i vilken miljö man är och då använder man sig av flaggan -e (lokal miljö) eller -g (global miljö). Miljön som man använder måste också exporteras lokalt till hårdskivan.

```
newman run .\MessageHub_APIs.postman_collection.json -e .\DEV.postman_environment.json
```

I kommandot ovan har man kört en Postman samling i en lokal miljö, om valda samlingen innehåller tester så kommer det att visas grafiskt i kommandotolken hur många tester har körts, vilka lyckades, mängden iterationer etc. I figur 12 kan man se ett exempel på detta, här har man testat att får man HTTP-koden 200 som betyder att man har fått en lyckad anslutning till API:n.

```
PS C:\Users\Valtteri Eronen> newman run .\MessageHub_APIs.postman_collection.json -e .\DEV.postman_environment.json
newman

MessageHub_APIs

□ MessageHub API
└ Query Message
  POST https://service.eu.apiconnect.ibmcloud.com/gws/apigateway/api/2e03bf6216f105a52e0bc3b7ae32994491bc5a2a64393ac5233
78df3f0ba18d5/messagehub/query [200 OK, 18.6KB, 1236ms]
  √  Status code is 200
```

	executed	failed
iterations	1	0
requests	1	0
test-scripts	2	0
prerequest-scripts	1	0
assertions	1	0
total run duration: 1390ms		
total data received: 17.72KB (approx)		
average response time: 1236ms [min: 1236ms, max: 1236ms, s.d.: 0ns]		

Figur 12 Resultat på den körda samlingen

4 AUTOMATISKA TESTER MED POSTMAN

4.1 Miljöer

Miljöer i Postman är ett sätt att hålla koll på miljöspecifika variabler mellan olika stadier t.ex. användarnamnet för att kontakta i API:n är A i utvecklingsstadiet och i textomgivningen är den B. Så då är det lätt att byta omgivning i Postman med att bara välja den omgivning (Eng. environment) du vill ha, utan att hamna manuellt gå och ändra på användarnamn. (se kapitel 3.2, figur 10).

4.1.1 Lokal miljö

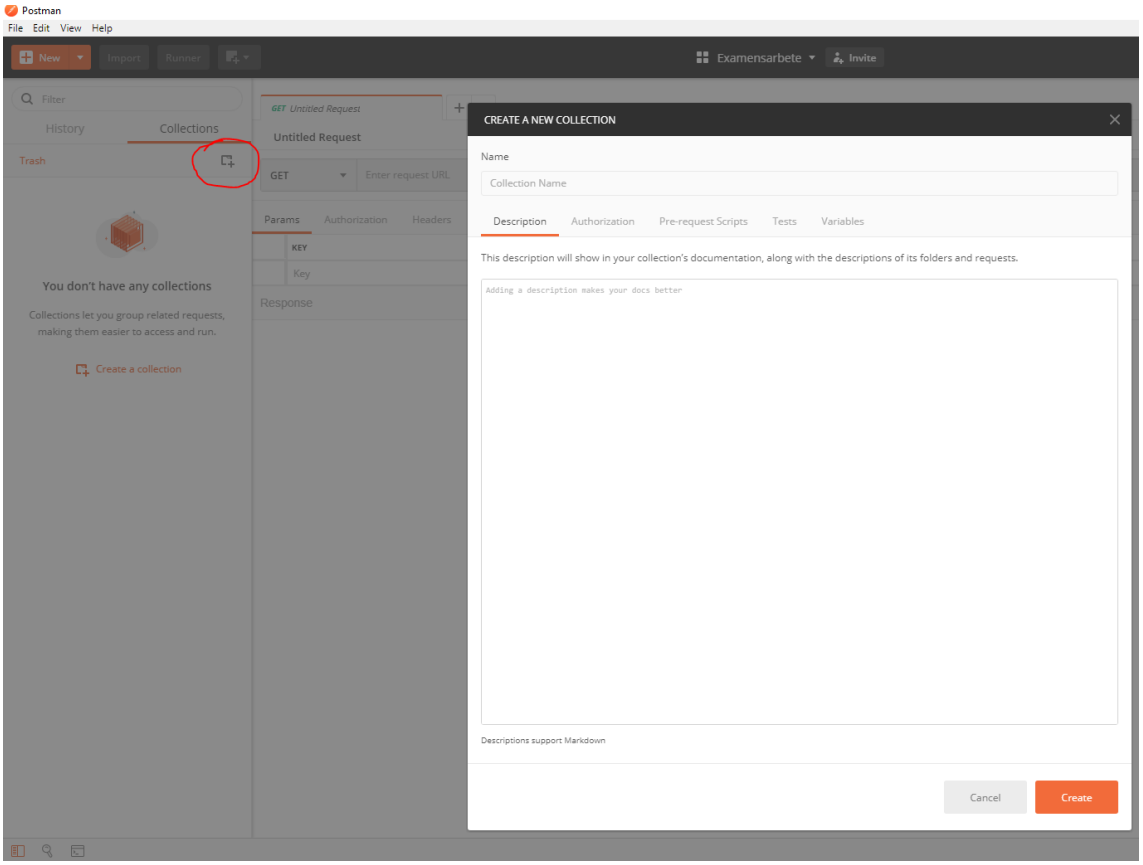
I en lokal miljö sparar man vanligtvis variabler som är specifika just för den miljön som t.ex. användarnamn, lösenord, API nycklar etc. Lokala variabler kan inte delas mellan olika anrop.

4.1.2 Global miljö

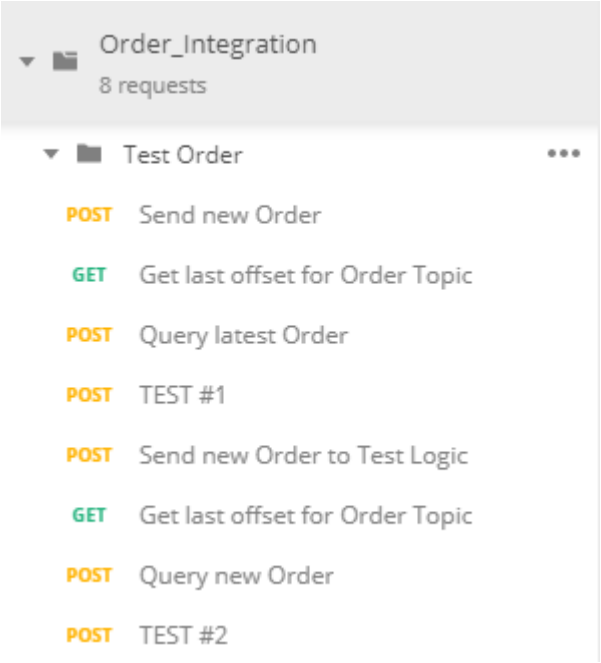
I en global miljö så sparar man variabler som går att användas i alla anrop och miljöer, är alltså inte bundna till någon viss miljö.

4.2 Skapandet av en samling

Man skapar en ny samling med att välja ”new collection” i postman som visas i figur 13. När man skapar en samling kan man välja om man vill sätta i förhand användarnamn, lösenord förhandsskript eller tester som körs vid varje anrop som är innanför den valda samlingen. In i en samling går det att lägga till HTTP-anrop, varje anrop går och exekveras manuellt en åt gången eller sen kan man exekvera anropen samtidigt med hjälp av Postmans Collection Runner (se kapitel 3.2).



Figur 13 Skapa en ny samling



Använder som exempel en samling där man har skapat regressionstester för en integration som sköter processeringen av beställningar. I figur 14 ser man den samlingen som skapats för integrationen. I denna samlingen skickar vi anrop till API:n där det finns topics som tar emot meddelandet och för det vidare till nästa stege där den processeras automatiskt (se kapitel 3.1.2.1).

Figur 14 Skapad samling

4.3 Regressionstester

Alla tester som skapats är regressionstester där man redan i förväg vet vilket värde som borde fås ut av integrationen efter den processerat ett nytt meddelande. Så man skulle kunna säga att enskilda testerna inom (se exemplet) anropen fungerar som enhetstester (se kapitel 2.1) och hela samlingen är som ett integrationstest (se kapitel 2.2) där man går igenom hela flödet på ett meddelande, i detta fall på en beställning.

Exempel på en enskild test inom ett HTTP anrop kunde se ut såhär:

```
tests["Currency OK"] = jsonAvailability.ZORDERS06.IDOC.E1EDK05.hasOwnProperty('KOEIN')
&& jsonAvailability.ZORDERS06.IDOC.E1EDK05.KOEIN === 'EUR';
```

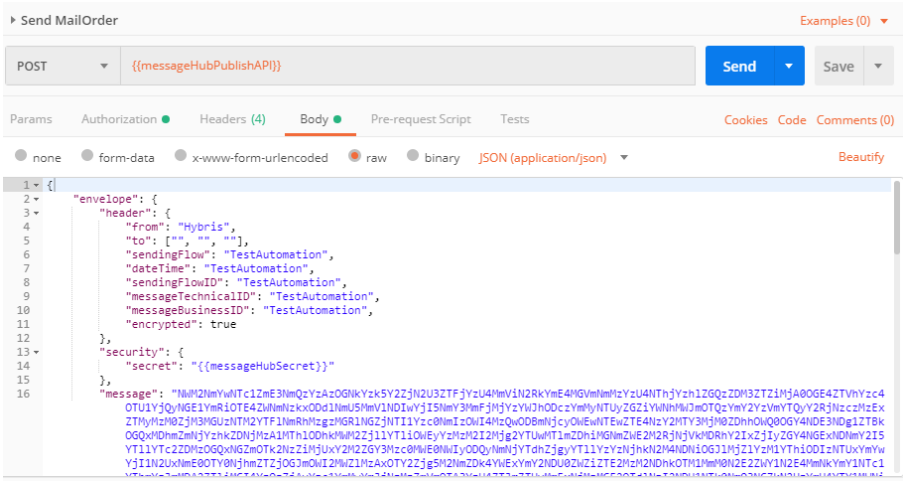
Ett enskilt test skapas med att skriva tests före hakparenteserna och inom parenteserna skriver man namnet på testet. Testet har gått igenom om ger svaret True dvs. det finns rätt information i XML-filen och testet har misslyckats om den returnerar False. Så testet fungerar egentligen som en IF-sats.

I exemplet ovan så går man igenom resultatet från integrationen med att först söka fram att det finns en nod i rätt segment i XML-filen som heter ”KOEIN” var valutan skall sättas och att värde för det är ”EUR”. Testerna skrivs i Postman med JavaScript.

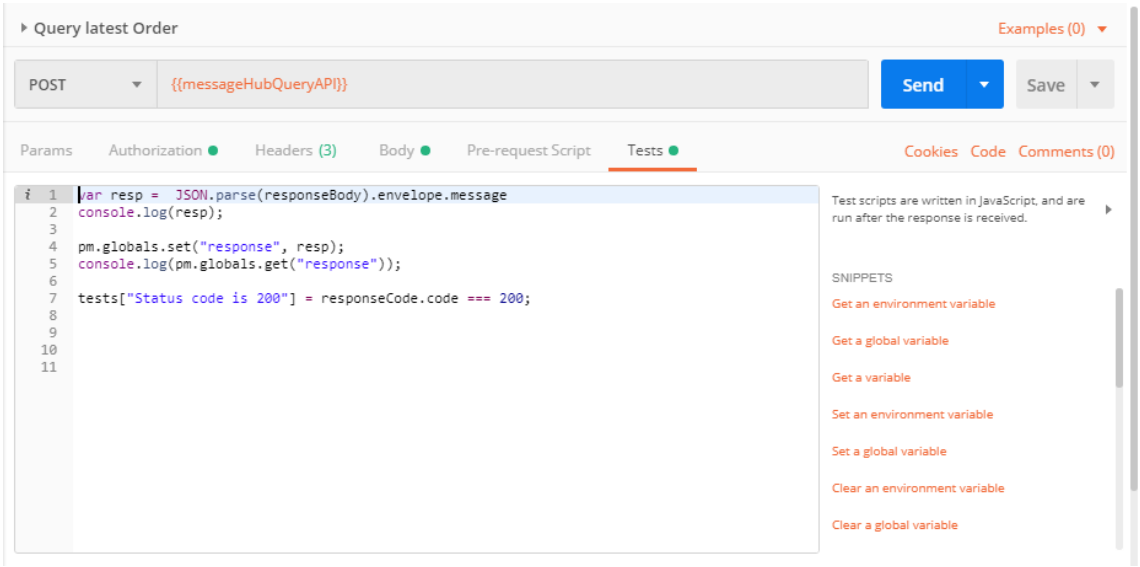
4.3.1 Exempel

Om man tar den samlingen som vi skapat i kapitel 4.2 som exempel så generella flödet med anropen för att skapa regressionstest är:

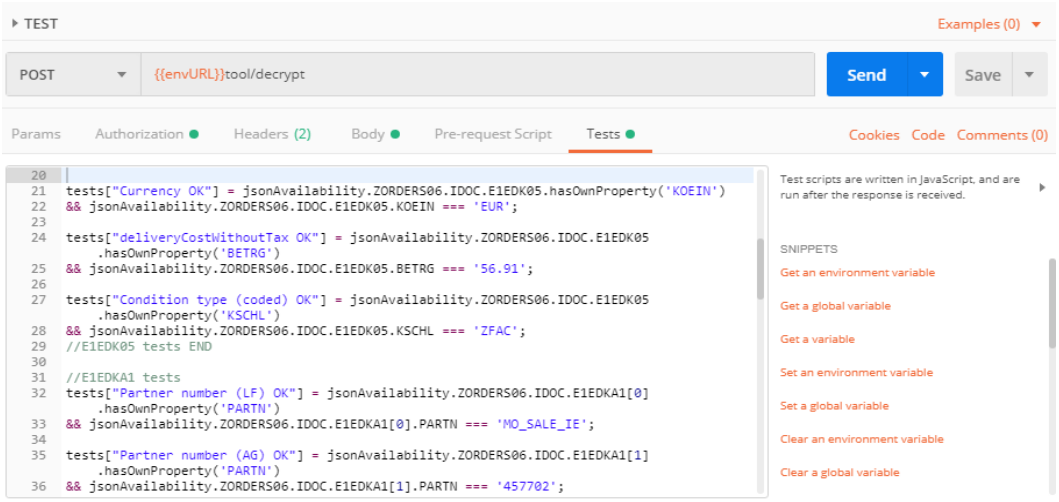
1. Skicka ett nytt meddelande till en topic (beställning)



2. Gör en GET anrop för att få senaste meddelandet från topic:en, sparar det senaste meddelandet som en global variabel (se kapitel 4.1.2) för att kunna komma åt det i nästa anropet i samlingen

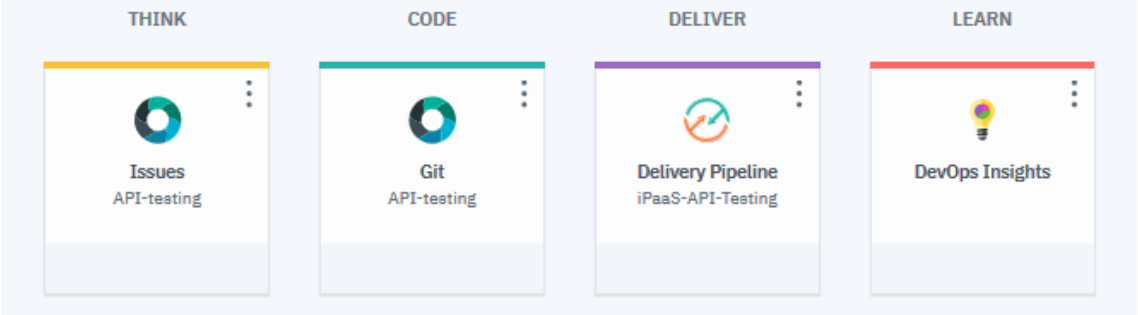


3. Meddelandet dekrypterats för att få den i XML-format, man gör testerna på det dekrypterade meddelandet för att kolla om informationen är fel eller korrekt.



4.4 Strukturen i verktygskedjan

Strukturen som verktygskedjan har är väldigt mycket lik det exemplet som kan hittas i kapitel 3.1.1, där vi har en Github repository där vi har koden, dvs. alla App Connect Studio projekt (se kapitel 3.1.2), en delivery pipeline som innehåller olika stadier (Developement, Test och Production) och Insights som kan ge diverse information av verktygskedjan i formen av en instrumentbräda. I figur 15 kan man se en översikt över verktygskedjan som används.



Figur 15 Verktygskedjan

4.4.1 Skript

För att få delivery pipeline:en konfigurerad och fungera på önskat sätt så har vi ett antal olika bash-skript.

buildGit.sh – sköter om ändringarna i Github, körs varje gång då en ändring körs, skapar en ny build.

createNewmanEnvironment.sh – skapar samban-det mellan variabler i Postman och vår pipeline. ”password” är det lokala variabelnamnet i Postman och ${APPCONNECT_PASSWORD} är motsvarigheten i delivery pipelinen. Sparar informationen i en json fil **Development.postman_environment**.

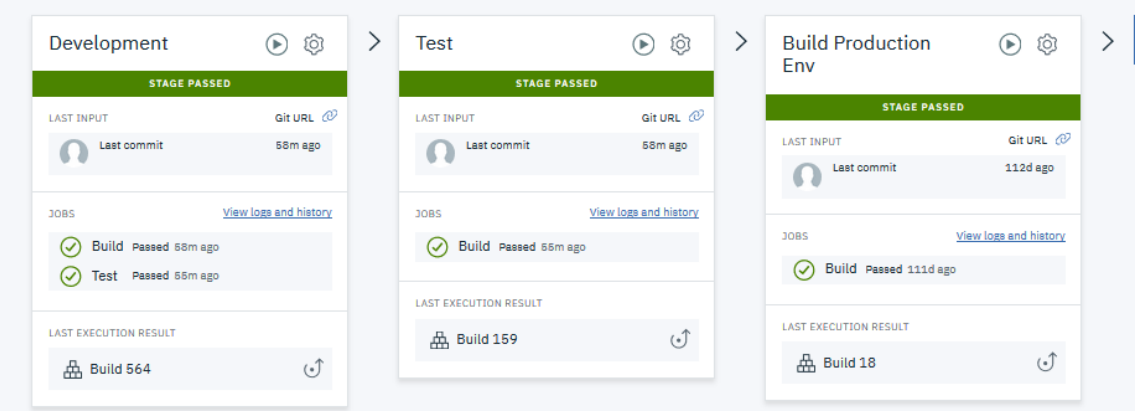
```
{
  \"enabled\": true,
  \"key\": \"password\",
  \"value\": \"${APPCONNECT_PASSWORD}\",
  \"type\": \"text\"
},
```

runTests.sh – läser från vår Github alla projekt som har en mapp som heter ”/tests/”, varefter den skapar en lista av kommandon som kör alla tester (se kapitel 3.2.1). Miljön som används har **createNewmanEnvironment.sh** skapat.

```
ls */tests/*.json > testlist.txt
while read row; do
  projectname=$(echo $row | cut -d "/" -f1)
  newman run $row -r junit --reporter-junit-export tests/TEST-$projectname.xml --silent --globals newman/Development.postman_environment.json
done <testlist.txt
```

4.4.2 Omgivningarna i delivery pipeline (Dev/Test/Prod)

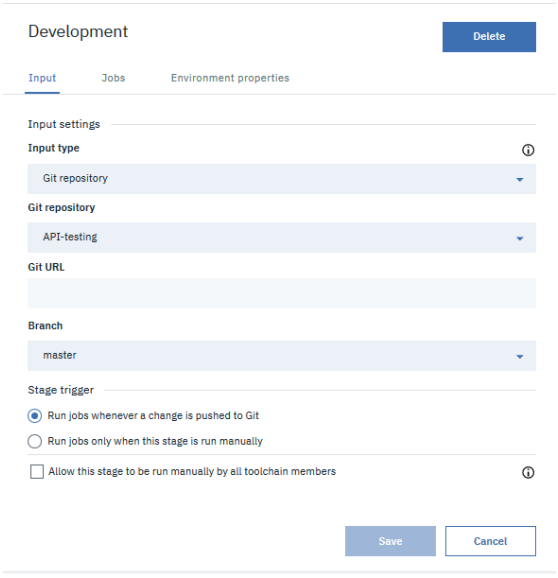
I delivery pipelinen (se kapitel 4.4, Figur 16) finns det tre stadier. **Development** där man gör ett nytt bygge (Eng. Build) och kör alla tester då man uppdaterar något till Github, **Test** och **Production** där man gör en enbart ett nytt bygge då man uppdaterar Github.



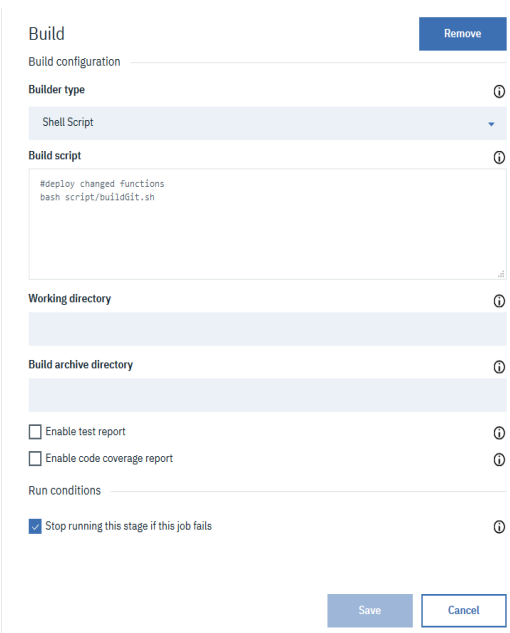
Figur 16 Delivery pipeline

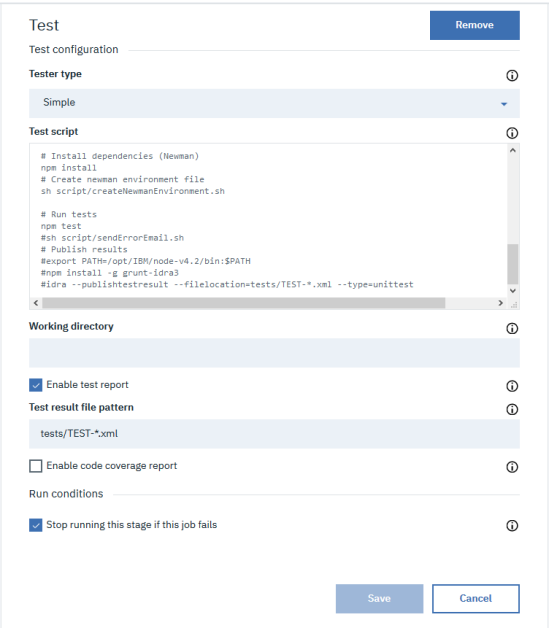
4.4.3 Konfigurering av omgivningen (Development)

För att konfigurera stadiet där man kommer att köra regressionstesterna så trycker man på kugghjulet i högra övre hörnet på den valda omgivningen. Först måste man välja inputen för omgivningen och i detta fall är de en Github repository som innehåller alla App Connect Studio projekt.



Sedan väljer man vilka job vi vill ha i omgivningen, i detta fall väljer man **Build** och **Test**. I **Build** så kör man skriptet **buildGit.sh** (se kapitel 4.4.1) och i **Test** så skapar man ett bash skript som innehåller installation av en version av npm och node (se kapitel 3.2.1), skapandet av miljövariablerna med att använda **createNewmanEnvironment.sh** skriptet, samla ihop alla tester och köra de med **runTests.sh**.



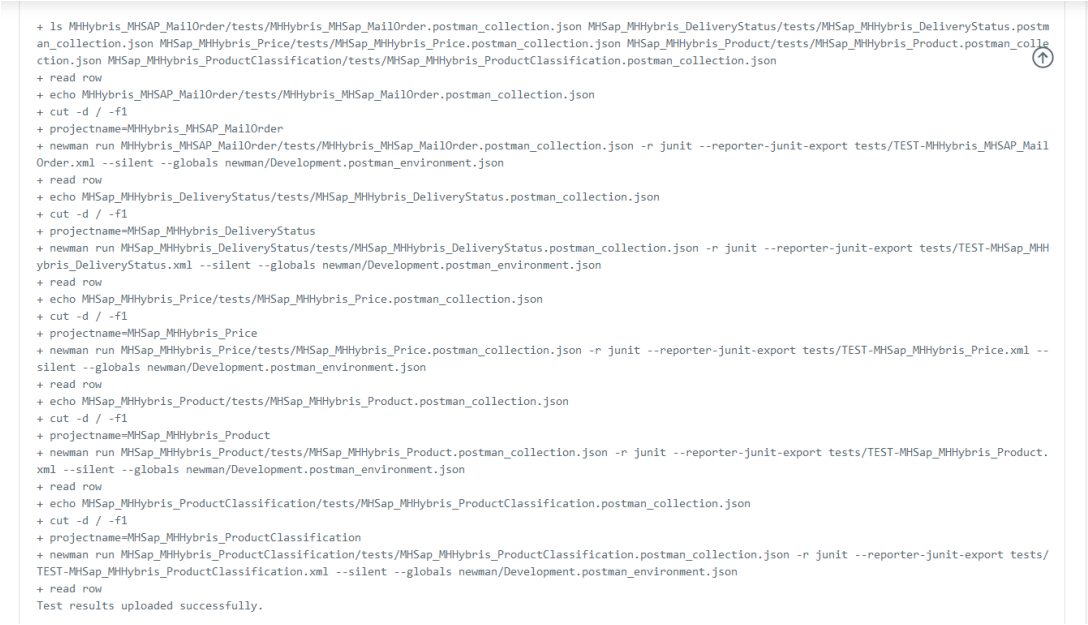


Sedan definierar man variablerna som skall användas i omgivningen. I variablerna finns användarnamn, lösenord, API nycklar etc. och dessa variabler kopplas ihop med variablerna i Postman samling som innehåller testerna, **createNewmanEnvironment.sh** sköter om det (se kapitel 4.4.1).

4.4.4 Integration av testerna med delivery pipeline

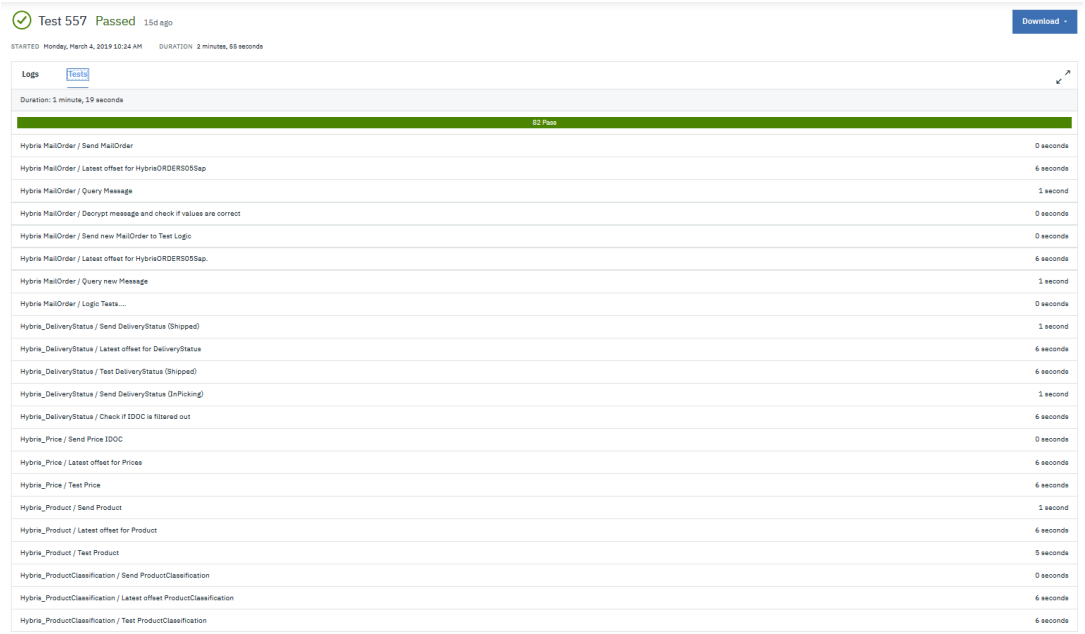
För att få den samling som man skapat (se kapitel 4.2, figur 14) som en del av testerna i delivery pipelinen måste man lägga den exporterade samlingen filen i respektive projekts ”/tests/” folder i versionshanteringen och publicera ändringarna till Github.

Efter ändringarna är publicerade så påbörjas automatiskt en ny build som gör själva buil-den varefter testerna påbörjas med att den går igenom alla testerna som skapats med hjälp av skripten som konfigurerats i Development-omgivningen (se kapitel 4.4.3). När allting är färdigt så ger den loggar på testerna som gått igenom, se figur 17 där kan man se att den har samlat alla projekts tester till en lista och går igenom alla tester.



Figur 17 Logarna på körda tester

Visar också vilka tester som har lyckats eller misslyckats, i figur 18 finns det ett exempel på detta.



Figur 18 Resultaten på körda tester

5 SLUTSATSER

Målsättningen med examensarbetet var att skapa regressionstester som en del av en Continuous delivery pipeline där testerna körs varje gång det sker en ändring i versionhanteraren så att de som kommer att upprätthålla integrationerna i fortsättningen skulle ha det lättare.

I examensarbetet redovisades hur man skapar en delivery pipeline med IBM Cloud DevOps, hur man konfigurerar den att fungera som en del av verktygskedjan och hur man bygger upp en infrastruktur som stöder exekverandet av tester som är skapade med Postman.

Tester som skapats är enbart regressionstester som testar integrationer gjorda med App Connect Studio. Testerna var lätta att skapa för att det finns bara ”ett rätt svar” på vad som man vill att skall komma ut efter integrationen har processerat meddelandet. Alla tester gjordes på samma sätt (se kapitel 4.3) som gjorde testandet väldigt klart, man behövde enbart ange noden i XML-filen och värdet som skulle finnas där.

Denna typ av regressionstest är ett bra val vid testning av de integrationer som beskrivits i examensarbetet just av den enkla orsaken att integrationer är skapade med App Connect Studio. App Connect Studio ger inte ut någon information som skulle berätta om de olika stegen som sker i (”koden”) då den processerar meddelandet utan man ser hela responsens/resultatet först efter processeringen. Det hämtar med sig vissa begränsningar i att hurdana tester det går att utföra.

Målsättningen var att skapa automatiska tester på alla integrationer men jag har lyckats skapa tester enbart på de integrationerna som är 100 % i molnmiljön. Integrationer som kommunicerar med SAP eller en annan extern instans (till exempel en FTP server) har jag inte lyckats skapa tester för eftersom jag inte har möjlighet att styra dataflödet som kommer från en yttre instans och vi har inte kommit överens hur man skulle förvara lösenorden och användarnamnen för yttre instanserna på ett vettigt och säkert sätt.

Men med lite planering och skapandet av en miljö i molnet där man simulerar yttre processerna vore det möjligt att skapa tester på alla integrationer. Men det skulle kräva massvis med tid att skapa en simulerad miljö, så för tillfället är det någonting vi inte har tänkt implementera.

KÄLLOR / REFERENCES

IBM, 2018, IBM App Connect Professional, Tillgänglig: https://www.ibm.com/support/knowledgecenter/en/SSTTDS_11.0.0/com.ibm.ace.home.doc/appconnpro.html Hämtad: 15.02.2019

IT-Ord, 2019a , HTTP, Tillgänglig: https://it-ord.idg.se/ord/http/ , Hämtad: 05.02.2019

IT-Ord, 2019b, UI. Tillgänglig: https://it-ord.idg.se/ord/anvandargranssnitt/ , Hämtad: 05.02.2019

IT-Ord, 2019c, TDD, Tillgänglig: https://it-ord.idg.se/ord/test-driven-development/ , Hämtad: 05.02.2019

IT-Ord, 2019d, API, Tillgänglig: https://it-ord.idg.se/ord/programmeringsgranssnitt/ , Hämtad: 05.02.2019

IT-Ord, 2019e, platform-as-a-service, Tillgänglig: https://it-ord.idg.se/ord/platform-as-a-service/ , Hämtad: 05.02.2019

IT-Ord, 2019f, infrastructure-as-a-service, Tillgänglig: https://it-ord.idg.se/ord/infrastructure-as-a-service/ , Hämtad: 17.02.2019

Techopedia, 2019, End-to-End Test, Tillgänglig: https://www.techopedia.com/definition/7035/end-to-end-test , Hämtad: 05.02.2019

Postman, 2019, Postman, Tillgänglig: https://www.getpostman.com/ , Hämtad: 8.2.2019

Postman, 2019b, Command line integration with Newman, Tillgänglig: https://learning.getpostman.com/docs/postman/collection_runs/command_line_integration_with_newman/ , Hämtad: 8.2.2019

Postman, 2019c, Collections, Tillgänglig: https://www.getpostman.com/collection , Hämtad: 22.02.2019

Postman, 2019d, Collection Runner, Tillgänglig: https://learning.getpostman.com/docs/postman/collection_runs/starting_a_collection_run/ , Hämtad: 17.02.2019

IBM Cloud Products, 2019. Products, Tillgänglig: https://console.bluemix.net/docs/services/ContinuousDelivery/devops_intro.html#devops_intro , Hämtad: 08.02.2019

IBM Cloud, 2019, Continious Delivery, Tillgänglig: https://console.bluemix.net/docs/services/ContinuousDelivery/devops_intro.html#devops_intro , Hämtad: 08.02.2019

IBM DevOps, 2019, DevOps, Tillgänglig: https://www.ibm.com/cloud/devops , Hämtad:17.02.2019

Peck, N., 2017, Microservice Testing: Unit Tests, Tillgänglig: https://medium.com/@nathankpeck/microservice-testing-unit-tests-d795194fe14e , Hämtad: 08.02.2019

Fowler, M., 2014, UnitTest, Tillgänglig: https://martinfowler.com/bliki/UnitTest.html , Hämtad: 08.02.2019

Kanchan, P., 2017, Why is Integration testing important in Software Testing Life Cycle?, Tillgänglig: https://bitwaretechnologies.com/integration-testing-important-software-testing-life-cycle/ , Hämtad: 06.02.2019

NodeJS, 2019, Download nodejs, Tillgänglig: https://nodejs.org/en/download/ , Hämtad: 19.03.2019

Npm, 2019, Install npm, Tillgänglig: https://www.npmjs.com/get-npm , Hämtad: 19.03.2019