En Kort Introduktion till DevOps

Krystian Manczak

Blekinge Tekniska Högskola  
Webbprogrammering, Karlskrona, Sverige

# Abstrakt

**Dev**elopment och **Op**eration**s**, inom mjukvaruutveckling, oftast förkortat till DevOps, är ett kulturellt och praktiskt arbetssätt. Kulturellt, där man försöker genom en förstående teamanda uppnå positiva mål genom ständig återkoppling över hur mjukvaran avancerar från utveckling till produktion. Praktiskt, genom att försöka effektivisera och automatisera miljön mjukvaran befinner sig i, från utveckling till produktion.

Devops tillåter snabbare lansering av applikationer samt är dessa applikationer bättre förberedda och enklare att underhålla genom deras övergång till containeriserade mikrotjänster.

På grund av dess komplexitet som följer devops, så finns det en hel del att inta vilket kan överväga till att det aldrig blir av. Skiften över till devops innebär att man kanske blir tvungen att anställa nytt folk, börja arbeta på ett annat sätt, både praktiskt och mentalt. Tiden man får offra till att lära sig nya verktyg och metoder är också något att beakta.

Nyckelord: DevOps, devops kultur, devops praktiskt

# Introduktion

Den här uppsatsen är skriven som en slutlig rapport för en kurs som heter ”Kontinuerlig integration och driftsättning i molnet” på Blekinge Tekniska Högskola. Uppsatsen svarar på vissa frågor gällandes devops.

# Vad är DevOps?

Enligt “A Survey of DevOps Concepts and Challenges” skriver dem “DevOpsis [sic] a collaborative and multidisciplinary organizational effort to automate continuous delivery of new software updates while guaranteeing their correctness and reliability.” [1] DevOps är så mycket mer än att jobba på ett visst sätt med ett visst antal verktyg. ”Devops is a way of thinking and a way of working. It is a framework for sharing stories and developing empathy, enabling people and teams to practice their crafts in effective and lasting ways. It is part of the cultural weave that shapes how we work and why. Many people think about devops as specific tools like Chef or Docker, but tools alone are not devops. What makes tools “devops” is the manner of their use, not fundamental characteristics of the tools themselves.” DevOps är alltså mer än bara ett sätt, en metod att producera mjukvara efter. ”Devops is not just another software development methodology. Although it is related to and even influenced by software development methodologies like Agile or XP, and its practices can include software development methods, or features like infrastructure automation and continuous delivery, it is much more than just the sum of these parts. While these concepts are related and may be frequently seen in devops environments, focusing solely on them misses the bigger picture—the cultural and interpersonal aspects that give devops its power.” [2] Låt oss därför börja med den historiska och kulturella delen av devops innan vi går in på den tekniska och praktiska delen.

## DevOps kultur och historia

Datorer reflekterar människors gärningar genom vad vi programmerar datorer till att göra. Vi använder datorer för att underlätta vår vardag. På samma sätt har devops kommit till. Det handlar om att när man jobbar med något, så lär man sig mer om produkten och arbetsmiljön, och detta leder till att vilja effektivisera sin arbetsprocedur så mycket som möjligt.

Författarna bakom *The DevOps Handbook* ger en utmärkt förkaring: ”DevOps and its resulting technical, architectural, and cultural practices represent a convergence of many philosophical and management movements. While many organizations have developed these principles independently, understanding that DevOps resulted from a broad stroke of movements, a phenomenon described by John Willis (one of the co-authors of this book) as the “convergence of DevOps,” shows an amazing progression of thinking and improbable connections. There are decades of lessons learned from manufacturing, high reliability organization, high-trust management models, and others that have brought us to the DevOps practices we know today.”

DevOps bygger på olika metoder ifrån olika organisationer ifrån olika tider: ”DevOps relies on bodies of knowledge from Lean, Theory of Constraints, the Toyota Production System, resilience engineering, learning organizations, safety culture, human factors, and many others. Other valuable contexts that DevOps draws from include high-trust management cultures, servant leadership, and organizational change management.” [3]

## DevOps praktiskt och tekniskt

För att arbeta efter devops (d**ev**elopment och **op**eration**s**)[4] så handlar det inte bara om att skapa mjukvara, utan också hur de anställda går tillväga för att utveckla mjukvaran. ”The heart of devops starts with people working not only as groups but as teams with a desire for mutual understanding. This can be described as a compact that teams will work together, communicate their intentions and the issues that they run into, and dynamically adjust in order to work toward their shared organizational goals.” [5]

Konceptet devops utgår ifrån att leverera en tjänst eller en produkt till en kund. ”In DevOps, we typically define our technology value stream as the process required to convert a business hypothesis into a technology-enabled service that delivers value to the customer.” När man börjar jobba på kundens idé, jobbar utvecklingsteamet efter en typisk agil eller iterativ process och börjar implementera detta i kod. Koden tillhandahålls genom versionshantering. Varje ändring i koden integreras och testas med resten av mjukvarusystemet. Integreringen innebär att bygga applikationen i ett byggsystem som bygger om applikationen för att se att alla komponenter samarbetar, och testerna, inkluderat säkerhetstester, testar att applikationen ska göra vad den är menat att göra. Som nämnt i förra citeringen, värde (”value”) sker endast när tjänsten körs i produktion. När man skickar applikationen till en produktionsserver, så behöver man försäkra sig om att driftsättningen inte orsakar fel och störningar såsom service-, driftavbrott eller säkerhetsfel, bland annat[6].

# Hur arbetar man inom DevOps?

## Dev

Effektiv **dev**ops innefattar flera steg, från utveckling och test till produktion. Man brukar följa en procedur när man utvecklar mjukvara, även känt på engelska som ”software development methodology”. Inom denna kategori brukar det agila (en. “Agile”) arbetssättet efterföljas. Det betyder dock inte att man alltid måste följa just detta arbetssätt. Det handlar mer om miljön koden arbetar inom. ”Devops is not so rigidly defined as to prohibit any particular methodology. While devops arose from practitioners who were advocating for Agile system administration and cooperation between development and operations teams, the details of its practice are unique per environment.” [7]

När man skriver kod vill man hantera detta på ett effektivt sätt. Man vill kunna snabbt ladda upp koden och testa den; man vill kunna gå tillbaka till en tidigare version ifall koden innehåller buggar; teamet vill arbeta på olika delar av koden i applikationen. Detta kallas för versionshantering. Ett välkänt versionshanteringssystem som många utvecklare använder heter Git.

Efter man skrivit koden vill man även testa den[7]. Man brukar ha ett automatiserande testramverk som går igenom olika testfall man skrivit för koden för att se att de nya funktioner man skapat går igenom. Ett sådant testramverk kan tex. vara PHPUnit för kod skriven i PHP. Säkerhetstester brukar också inkluderas, för att kolla sårbarheter i applikationen eller koden. Sådana säkerhetstester kan bestå av Bandit för Python. Denna sortens säkerhetstest brukar definieras som ”Static Application Security Testing (SAST), eller bara Static Code Analysis”. ”Dynamic Application Security Testing (DAST) letar efter sårbarheter i webbapplikationer genom att skanna och utföra attacker på applikationen.” [8] För det här ändamålet kan man använda ZAP. Dessa olika säkerhetstester brukar inkluderas i konceptet DevSecOps. Meningen med detta koncept är att inkludera säkerheten tidigt i utvecklingskedjan än försent i produktionskedjan.

När man känner att man är klar tex. med en viss funktion som gått igenom de lokala testerna, vill man gärna se hur implementeringen i applikationen beter sig i sin helhet som om den skulle köras på produktionsservern. Man brukar då pusha koden till sitt versionshanteringssystem. Efter man pushat koden, brukar en viss sida, som tex. TravisCI, bygga applikationen. Detta kallas för Continuous integration (CI). En sådan sida/tjänst känner av varje push man begått och bygger applikationen och kör testerna. Allt detta sker automatiskt. När inga fel uppstår förbereder man applikationen som ska köras på live servern för Continuous delivery (CD).

I CD stadiet befinner sig applikationen redo för att publiceras på produktionsservern för att nyttjas av slutanvändaren.

När nu applikationen är redo att släppas för produktion sker detta genom Continuous deployment, som också hänvisas ibland som CD. Skillnaden här är att continuous delivery ser till att saker kan släppas för produktion, medan continuous deployment verkligen ser till att det släpps för produktion.

## Ops

Effektiv dev**ops** ser även till att tänka på infrastrukturen applikationen befinner sig i, dansar och lever inom. Då varje människa vill befinna sig i en så hälsosam omgivning som möjligt för att kunna leva, njuta och förlänga sina dagar, på samma sätt förhåller det sig inom mjukvaruvärlden. Det räcker inte att bara slänga upp en färdig applikation och överlåta åt slumpen vad som händer med den. Det gäller att förstå sig på applikationen i den miljö den befinner sig i för att kunna förlänga dess kvalitet och livstid, med andra ord, dess uppetid.

Allt detta måste också ske på ett så effektivt sätt som möjligt. För att bibehålla en konsistens över sina applikationer i den miljö de befinner sig i, så måste man styra upp servern de ska laddas upp på. Detta uppstyrande måste ske genom fördefinierad kod så att man enkelt kan sköta ändringar ifrån denna infrastruktur som kod, även känt på engelska som ”Infrastructure as Code (IaC)”.

Genom att baka in i sina CI/CD pipelines färdig IaC kan man alltid bygga om servern precis som man vill det genom ”Configuration Management (CM)”. [9] CM betyder att genom IaC använda ett verktyg som kommunicerar med servern, om det nu är fysiska servrar eller Virtual Machines på företaget eller bland de digitala molnen, som Azure, och detta verktyg, tex. Ansible, bygger upp servrarna från scratch.

Då vi nu har en konsistens i vilken miljö våra applikationer ska befinna sig i, gäller det också att människorna inte förlitar sig på att bara leva i en hälsosam miljö utan ibland får göra kontroller över deras hälsa. Ett osynligt för våra ögon virus kan orsaka hemskheter i människors liv. Dessutom sprids det snabbt. Samma gäller våra dansande applikationer. Vi måste sätta upp kontroller över applikationernas hälsa så att vi helst i god tid kan kontrollera livslängden på en applikation eller varför den beter sig som den inte borde göra. Vad vi helt enkelt gör kallas för ”Monitoring” och ”Log management”[10]. Denna monitoring och logging innebär att vi installerar applikationer som avger olika metrics (mätvärden) över applikationernas välbefinnande. Monitoring innebär att övervaka ett system med dess olika processer. Log management, eller hantering av loggar, handlar om att samla in, lagra, analysera och hantera logg data. Application performance monitoring (APM), handlar om att övervaka prestanda och användarupplevelsen hos applikationer för att möta vissa kriterier. Prometheus kan man använda ”för att lagra tidsserie data” och Grafana för att visualisera denna data genom grafer och diagram[10].

Ansible, som vi nämnde om förut, fungerar bra för få servrar utan några speciella belastningar. Men när trycket ökar måste detta hanteras på ett mer effektivt sätt. Ett sätt att hantera livslängden, mjukvarulivscykeln (en. ”Software Development Life Cycle (SDLC)”) av applikationerna på de olika servrarna bemöts genom ”Container orchestration”[11]. Meningen är att kunna på ett snabbt sätt hantera förändringarna genom effektiv utveckling utan att riskera infrastrukturen och produktionsmiljön. Det är här mjukvarucontainrar kommer in i bilden. På en och samma hamn finns flera containrar, där kanske alla innehåller olika saker. Oavsett vad som finns på hamnen kan varje isolerad container fortsätta existera oavsett dess innehåll. Detta är en liknande bild över digitala mjukvarucontainrar. En container består oftast av en huvudapplikation som körs på ett operativsystem inuti containern. Dessa containrar är mer isolerade ifrån operativsystemet och hårdvaran de befinner sig på jämfört med en traditionell applikation körandes på servern, som är mer inbakad i servern. Detta gör det enklare för utvecklare att skapa en applikation i en container och driftsätta i produktion[7]. Ett nästintill standard verktyg idag inom container orchestration är Kubernetes (K8s). K8s erbjuder bla. ”Service discovery and load balancing”, ”Storage orchestration”, “Automated rollouts and rollbacks”, “Automatic bin packing”, “Self-healing” och “Secret and configuration management”[12].

## Kulturell devops

Efter ett projekt har avslutats följer en retrospektiv dialog där bla. följande frågor ställs: *Vad hände?* *Vad gick bra?* och *Vad gick dåligt?* Intentionen med en sådan dialog är att kunna applicera svaren på nästkommande projekt.

Om en negativ incident skulle ske eller driftstopp av något slag, som man inte planerat, så har man en postmortem (undersökande) diskussion när det händer. Meningen är att lära sig genom att man går igenom saker över vad som hände; man låter personer inblandade i incidenten ge sin förklaring och man försöker åtgärda dessa incidenter för att öka säkerheten och undvika liknande fel framöver.

När man utför dessa saker nämnt försöker man göra det i en anda av ”blamelessness”. Fokus ligger på att lära sig än att straffa. Det betyder inte att låta dem gå skuldfria. Man vill få dem känna sig bekväma när de nämner en incident, även om det orsakade något dåligt[7].

# För- och nackdelar med devops

Jag har valt under rubriken fördelar att inte bara ta med en lista med de olika fördelarna utan försökt ge en viss förståelse varför fördelar uppstår när man jobbar inom devops.

## Fördelar

Devops handlar om att automatisera flödet, från utveckling till produktion. Det handlar om att effektivisera sitt arbetssätt så mycket som möjligt, detta sker oftast genom automatisering och feedback[13].

* Tack vare det automatiska flödet genom CI och CD kan applikationer snabbare lanseras för produktion.
* Applikationer är oftast bättre förberedda och enklare att underhålla pga. mikrotjänster och molnteknologier (en. ”cloud technologies”).
* Tack vare feedback loopen så reducerar man fel i produkten. Detta sker pga. att mjukvaran är mer anpassad till marknaden.
* Devops betyder gemensamt ansvar, vilket leder till bättre team engagemang och produktivitet[15].

### Det första sättet

Devops tillåter ett snabbt arbetsflöde från utveckling till drift för kunden. För att maximera detta arbetsflöde behöver man bla. synliggöra arbetet, reducera batch storlekarna och bygga in kvalitet genom att förhindra defekter[6].

För att förstå hur det går med arbetet, ifall det går bra eller dåligt, behöver man göra denna arbetsprocess synlig. Detta kan ske tex. genom en kanban bräda.

Istället för att leverera mycket kod åt gången, vilket kan visa sig ta mycket jobb när en och annan bugg inträffar, så koncentrerar man sig på att leverera små batcher åt gången. För att minska lead tiden och öka kvaliteten behöver man minska batch storleken[14].

### Det andra sättet

Under alla stadier av utvecklingen ser man till att feedback återskapas. Detta sker för att undvika samma problem i framtiden, eller upptäcka dem snabbare, och tillåter att man skapar kvalitet vid ett tidigt stadie av koden samt en viss nyttig kunskap om applikationen som kan belöna sig vid ett senare stadie också[6].

### Det tredje sättet

Tillit inom devops teamen genom dess kultur öppnar upp en disciplinerad och vetenskaplig väg för experiment och risktagande, vilket underlättar organisatoriskt lärande[6]. (Organisatoriskt lärande är en process där en organisation förbättrar sig själv genom att använda den erfarenhet den samlat på sig för att skapa kunskap. Denna kunskap sprids inom denna organisation[16].)

Genom de ständigt förbättrade återkopplingsslingorna (en. ”feedback loops”) skapar man säkrare arbetssystem och är bättre lämpade att våga ta risker och utföra experiment som hjälper företaget/organisationen lära sig snabbare jämfört med deras konkurrenter[6].

## Nackdelar

Devops i sin natur är väldigt komplex. För den som är ny till devops, så finns det en hel del nytt att ta in och lära sig. Skifta över till devops är heller inte en enkel sak man gör över en kort period. Man behöver tänka på att bygga om sina applikationer. Nya verktyg tillkommer man måste lära sig hantera. Kanske behöver man bygga om teamet och anställa nya medlemmar[15]? Man skulle kunna sammanfatta nackdelarna inom vissa utmaningar att ha i åtanken.

### Den första utmaningen

Den första utmaningen inom devops är att göra flytten. Om man är beroende av legacy system när man ska bygga om sina applikationer till mikrotjänster och anpassa dem ev. för molnet kan bli en utmaning få vågar ta sig an. Med detta i åtanken kanske man behöver anställa nytt folk med erfarenhet inom devops och anpassa teamet för den kulturella delen också[15]? Detta kan innebära skifta arbetssätt och mentalitet till en helt annan nivå.

### Den andra utmaningen

Behöver man anställa nytt folk med erfarenhet inom devops så kan det kosta. Inte nog med att försöka hitta dessa också. Enligt en artikel *DevOps Zone* skrev år 2019, som hänvisar till PayScale, gällandes devops specialister i USA, så skriver dem ”DevOps specialists with hands-on experience are hard to find. Most specialists in the field have 1-4 years of experience, according to Payscale. That is why DevOps engineer positions are among the top most difficult jobs to fill.” [15]

### Den tredje utmaningen

Det kan bli svårt att hitta de rätta verktygen som passar företagets/teamets arbetsmiljö. Sen kan det visa sig att de verktygen man valde inte alls passar. Inte nog med detta, det krävs tid för teamet att lära sig de nya verktygen/metoderna också[15].

# Referenser

[1] The ACM Digital Library, *A Survey of DevOps Concepts and Challenges*, (https://dl.acm.org/doi/10.1145/3359981), hämtad 2020.

[2] Effective DevOps, *Chapter 1. The Big Picture*, (https://learning.oreilly.com/library/view/effective-devops/9781491926291/ch01.html), hämtad 2020.

[3] The DevOps Handbook, *Part I, A BRIEF HISTORY*, (https://learning.oreilly.com/library/view/the-devops-handbook/9781457191381/DOHB-pt\_01\_text.xhtml), hämtad 2020.

[4] TechnologyAdvice, *DevOps - development and operations*, (https://www.webopedia.com/TERM/D/devops\_development\_operations.html), hämtad 2020.

[5] Effective DevOps, *Chapter 2. What Is Devops?*, (https://learning.oreilly.com/library/view/effective-devops/9781491926291/ch02.html), hämtad 2020.

[6] The DevOps Handbook, *1 Agile, Continuous Delivery, and the Three Ways*, (https://learning.oreilly.com/library/view/the-devops-handbook/9781457191381/DOHB-ch\_01.xhtml), hämtad 2020.

[7] Effective DevOps, *Chapter 4. Foundational Terminology and Concepts*, (https://learning.oreilly.com/library/view/effective-devops/9781491926291/ch04.html), hämtad 2020.

[8] dbwebb, *SAST vs. DAST*, (https://dbwebb.se/kurser/devops-v1/kmom05#sast-dast), hämtad 2020.

[9] dbwebb, *Infrastructure as Code och Configuration Management*, (https://dbwebb.se/kurser/devops-v1/kmom03#iac-cm), hämtad 2020.

[10] dbwebb, *Kmom04: Monitoring*, (https://dbwebb.se/kurser/devops-v1/kmom04), hämtad 2020.

[11] dbwebb, *Kmom06: Container orchestration*, (https://dbwebb.se/kurser/devops-v1/kmom06), hämtad 2020.

[12] Kubernetes, *What is Kubernetes?*, (https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/), hämtad 2020.

[13] Reinventing ITIL® in the Age of DevOps: Innovative Techniques to Make Processes Agile and Relevant, *1. Introduction to DevOps*, (https://learning.oreilly.com/library/view/reinventing-itil-in/9781484239766/html/464131\_1\_En\_1\_Chapter.xhtml), hämtad 2020.

[14] The DevOps Handbook, *2 The First Way: The Principles of Flow*, (https://learning.oreilly.com/library/view/the-devops-handbook/9781457191381/DOHB-ch\_02.xhtml), hämtad 2020.

[15] DevOps Zone, *Benefits and Challenges of Taking the DevOps Route*, (https://dzone.com/articles/benefits-and-challenges-of-taking-the-devops-route), hämtad 2020.

[16] Valamis, *Organizational Learning*, (https://www.valamis.com/hub/organizational-learning), hämtad 2020.

[17] PayScale, Average Development Operations (DevOps) Engineer Salary, (länk), hämtad 2020.