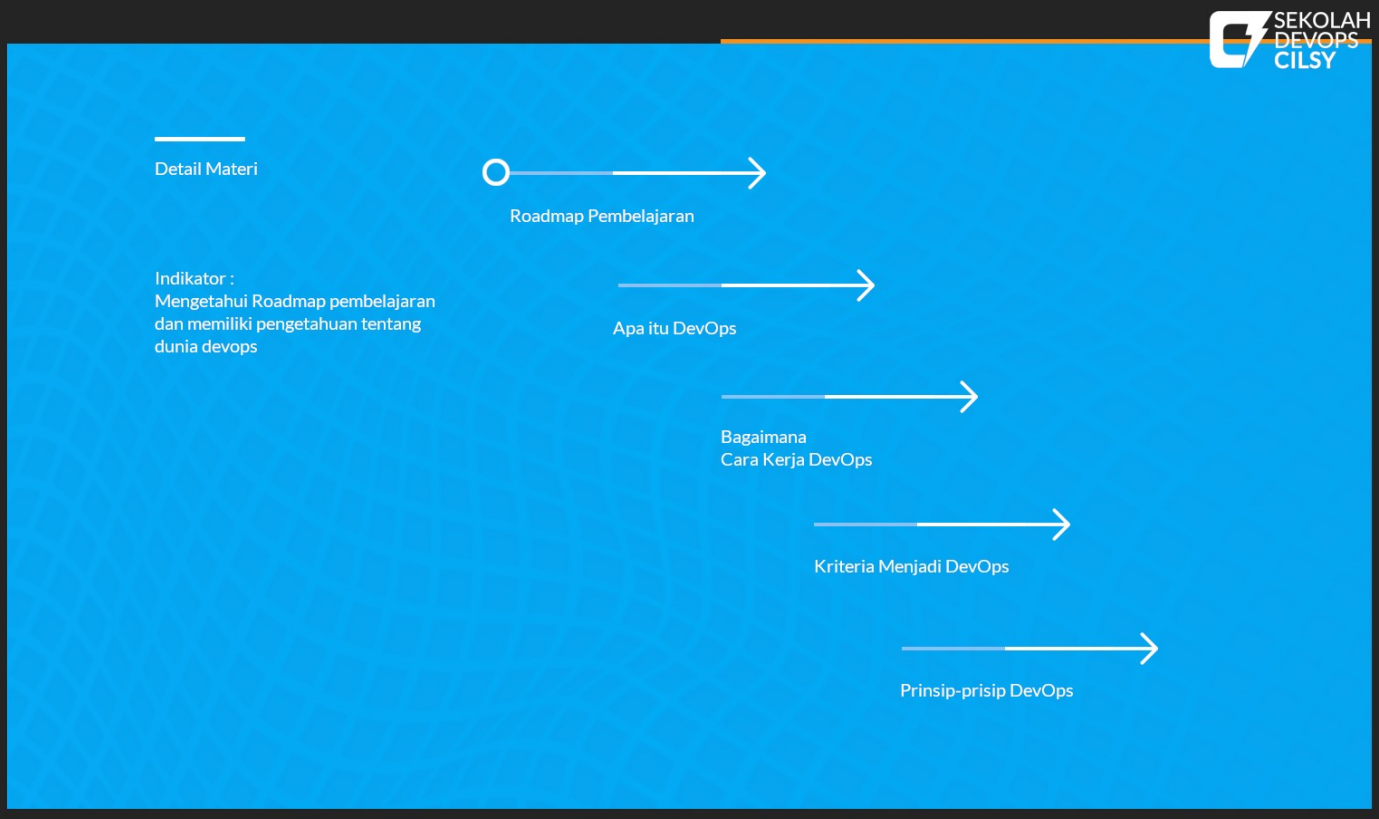
Bab 1

Pengenalan Roadmap SDC dan DevOps



Modul Sekolah DevOps Cilsy

Penulis : Adi Saputra, Irfan Herfiandana & Tresna Widiyaman
Editor : Rizal Rahman, Tresna Widiyaman, Muhammad Fakhri A.
Revisi Batch 5

Penerbit : **PT. Cilsy Fiolution Indonesia**
Web Site : https://cilsyfiolution.com , https://devops.cilsy.id

Daftar Isi

1.
Pengenalan DevOps

1.1. Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Mengetahui Roadmap Pembelajaran yang akan mereka jalani.
2. Mengenal masalah utama yang dipecahkan DevOps (Kenapa harus ada DevOps?).
3. Mengenal konsep dasar DevOps dan Software Development Lifecycle
4. Mengetahui sejauh mana prospek DevOps khususnya di Industri dalam negeri
5. Mengetahui skill fundamental yang dibutuhkan untuk menjadi seorang DevOps
6. Mengetahui prinsip dan cara kerja DevOps di industri IT
7. Mengetahui manfaat penerapan DevOps

1.2. Outline Materi

1. Tren Industri Aplikasi
2. Mengenal Proses Development sebuah Aplikasi (Development Cycle)
3. Masalah Utama yang Terjadi pada Proses Development Aplikasi.
4. Bagaimana DevOps Berkembang dan menjadi Solusi
5. DevOps sekarang & yang akan datang
6. *How DevOps Works*
7. DevOps di mata Industri

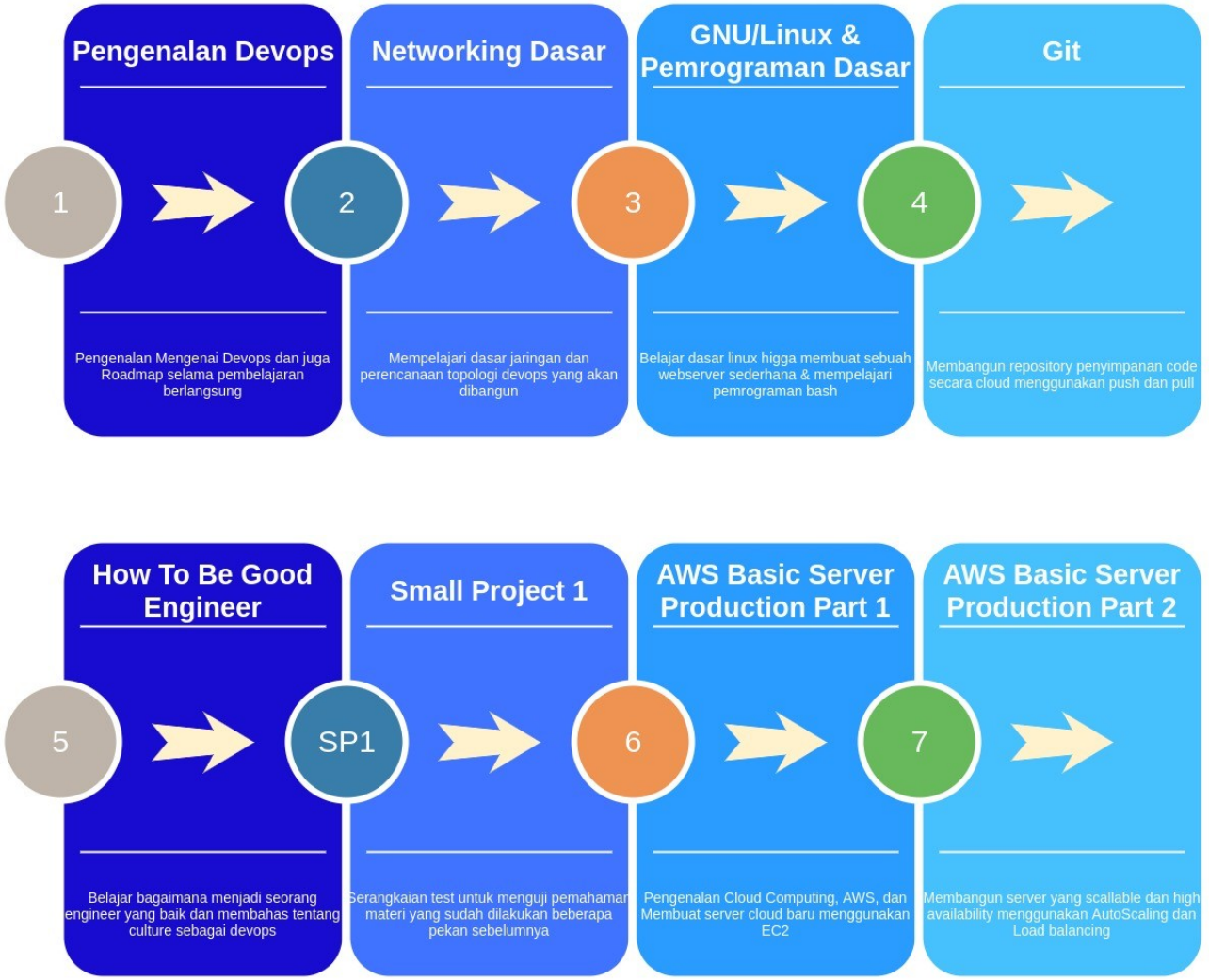
1.3. Roadmap Pembelajaran

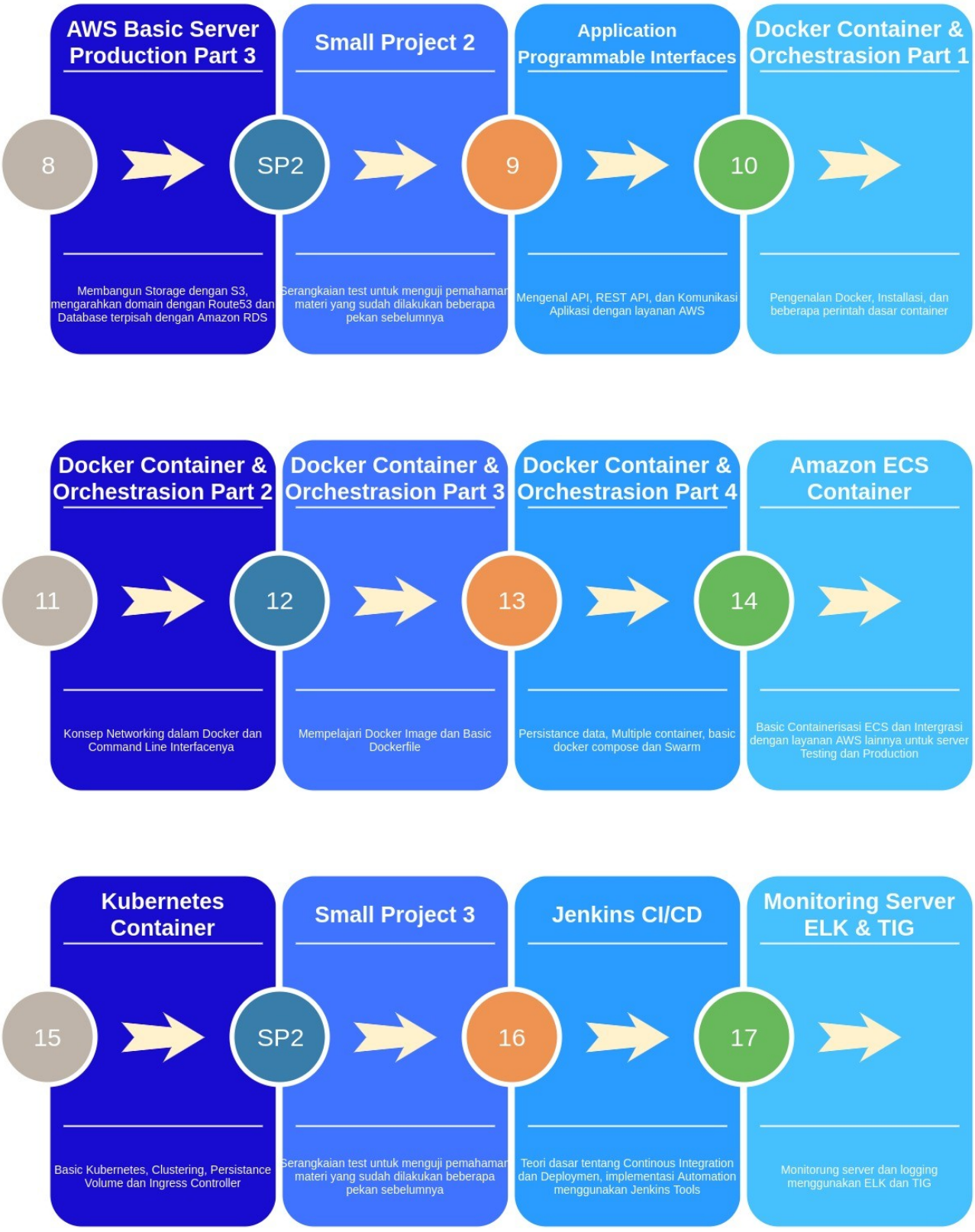
Sebelum memulai pembelajaran di Sekolah DevOps Cilsy, ada baiknya kita mengenal terlebih dahulu roadmap atau alur pembelajaran yang akan kita jalani selama masa pembelajaran 4 bulan kedepan. Diharapkan kita bisa mengetahui goal apa yang akan kita capai setelah menyelesaikan semua pembelajaran dan mengetahui sudah sampai tahap mana kita dalam proses pembelajaran yang berlangsung.

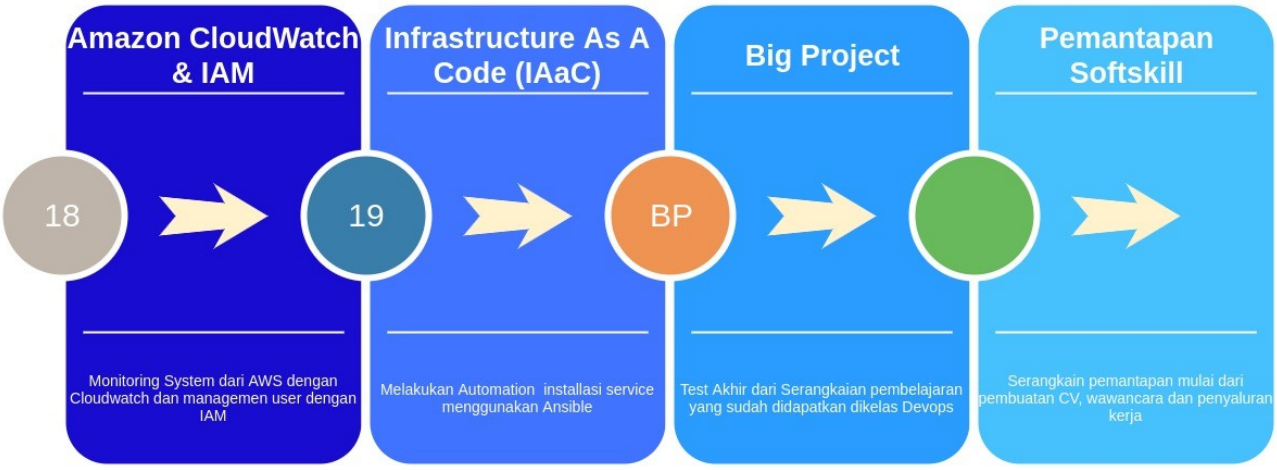
Roadmap disini akan ditampilkan dalam bentuk diagram proses, dan dijelaskan kembali nanti disetiap point pembelajaran yang akan kita jalani. Sehingga terus terpantau alur pembelajarannya.

1.3.1. Diagram Roadmap Pembelajaran

Berikut merupakan diagram roadmap dari pembelajaran yang akan kita jalani :




AWS Basic Server Production Part 3
8
Membangun Storage dengan S3, mengarahkan domain dengan Route53 dan Database terpisah dengan Amazon RDS
Small Project 2
SP2
Serangkaian test untuk menguji pemahaman materi yang sudah dilakukan beberapa pekan sebelumnya
Application Programmable Interfaces
9
Mengenal API, REST API, dan Komunikasi Aplikasi dengan layanan AWS
Docker Container & Orchestrasion Part 1
10
Pengenalan Docker, Installasi, dan beberapa perintah dasar container
Docker Container & Orchestrasion Part 2
11
Konsep Networking dalam Docker dan Command Line Interfacenya
Docker Container & Orchestrasion Part 3
12
Mempelajari Docker Image dan Basic Dockerfile
Docker Container & Orchestrasion Part 4
13
Persistance data, Multiple container, basic docker compose dan Swarm
Amazon ECS Container
14
Basic Containerisasi ECS dan Intergrasi dengan layanan AWS lainnya untuk server Testing dan Production
Kubernetes Container
15
Basic Kubernetes, Clustering, Persistance Volume dan Ingress Controller
Small Project 3
SP2
Serangkaian test untuk menguji pemahaman materi yang sudah dilakukan beberapa pekan sebelumnya
Jenkins CI/CD
16
Teori dasar tentang Continous Integration dan Deploymen, implementasi Automation menggunakan Jenkins Tools
Monitoring Server ELK & TIG
17
Monitorung server dan logging menggunakan ELK dan TIG



1.3.2. Penjelasan Roadmap Pembelajaran

Roadmap Pembelajaran	
Bagian	Penjelasan
1	Pengenalan Mengenai Devops dan juga Roadmap selama pembelajaran berlangsung
2	Mempelajari dasar jaringan dan perencanaan topologi devops yang akan dibangun
3	Belajar dasar linux higga membuat sebuah webserver sederhana & mempelajari pemrograman bash
4	Membangun repository penyimpanan code secara cloud menggunakan push dan pull
5	Belajar bagaimana menjadi seorang engineer yang baik dan membahas tentang culture sebagai devops
SP1	Serangkaian test untuk menguji pemahaman materi yang sudah dilakukan beberapa pekan sebelumnya
6	Pengenalan Cloud Computing, AWS, dan Membuat server cloud baru menggunakan EC2
7	Membangun server yang scallable dan high availability menggunakan AutoScaling dan Load balancing
8	Membangun Storage dengan S3, mengarahkan domain dengan Route53 dan Database terpisah dengan Amazon RDS
SP2	Serangkaian test untuk menguji pemahaman materi yang sudah dilakukan beberapa pekan sebelumnya
9	Mengenal API, REST API, dan Komunikasi Aplikasi dengan layanan AWS

10	Pengenalan Docker, Installasi, dan beberapa perintah dasar container
11	Konsep Networking dalam Docker dan Command Line Interfacenya
12	Mempelajari Docker Image dan Basic Dockerfile
13	Persistance data, Multiple container, basic docker compose dan Swarm
14	Basic Containerisasi ECS dan Intergrasi dengan layanan AWS lainnya untuk server Testing dan Production
15	Basic Kubernetes, Clustering, Persistance Volume dan Ingress Controller
SP3	Serangkaian test untuk menguji pemahaman materi yang sudah dilakukan beberapa pekan sebelumnya
16	Teori dasar tentang Continous Integration dan Deploymen, implementasi Automation menggunakan Jenkins Tools
17	Monitorung server dan logging menggunakan ELK dan TIG
18	Monitoring System dari AWS dengan Cloudwatch dan managemen user dengan IAM
19	Melakukan Automation installasi service menggunakan Ansible
BP	Test Akhir dari Serangkaian pembelajaran yang sudah didapatkan dikelas Devops
	Serangkain pemantapan mulai dari pembuatan CV, wawancara dan penyaluran kerja

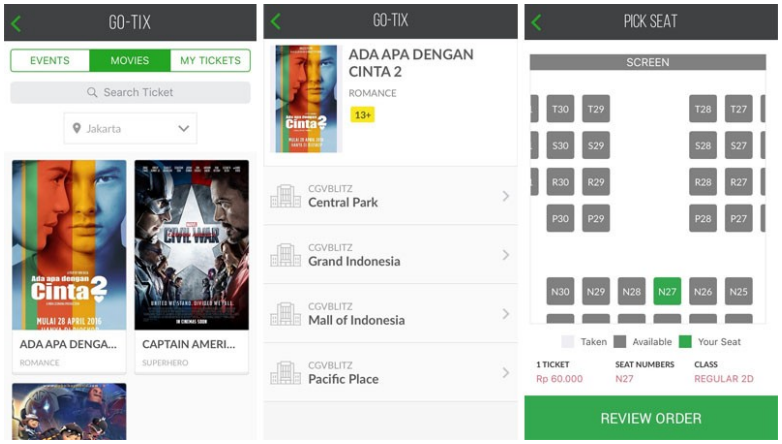
1.4. Tren Industri

1.4.1. Era Serba Aplikasi

Semua bermula dari dimulainya era Aplikasi. Dimana beberapa tahun kebelakang hingga saat ini semua perusahaan dan bisnis berlomba-lomba membuat produk-produk berbasis Aplikasi. Bahkan tak hanya produk yang tampil langsung ke pengguna, untuk kegiatan internal perusahaan pun sudah mulai dibuat serba aplikasi untuk meningkatkan produktifitas mereka. Sebegitu pentingnya, hingga kini pemanfaatan Aplikasi untuk perusahaan bukan lagi barang yang “nice to have” saja, tetapi sudah menjadi kebutuhan utama agar perusahaan dapat menjadi pemenang di industri mereka masing-masing.

Sebagai contoh, lihat saja bagaimana Cinema XXI yang dulunya merupakan raja di industri bioskop, langsung kocar-kacir ketika CGV muncul dengan kemudahan pemesanan tiket melalui aplikasi. Akibatnya kini Cinema XXI pun bergerak cepat untuk juga memunculkan aplikasi pemesanan tiket mereka sendiri agar tidak tertinggal.

Lihat juga bagaimana Bank BTPN yang namanya sebenarnya kurang terkenal dibanding BCA dan Mandiri, kini mendadak memimpin pasar perbankan setelah mereka meluncurkan aplikasi Jenius yang merevolusi pengalaman Banking menjadi super mudah.



Kerjasama CGV dengan Go-Jek menjadi Game Changer

Siapa yang berhasil memanfaatkan aplikasi untuk meningkatkan produktifitas perusahaan dan memuaskan pelanggan sebaik mungkin, mereka lah pemenangnya.

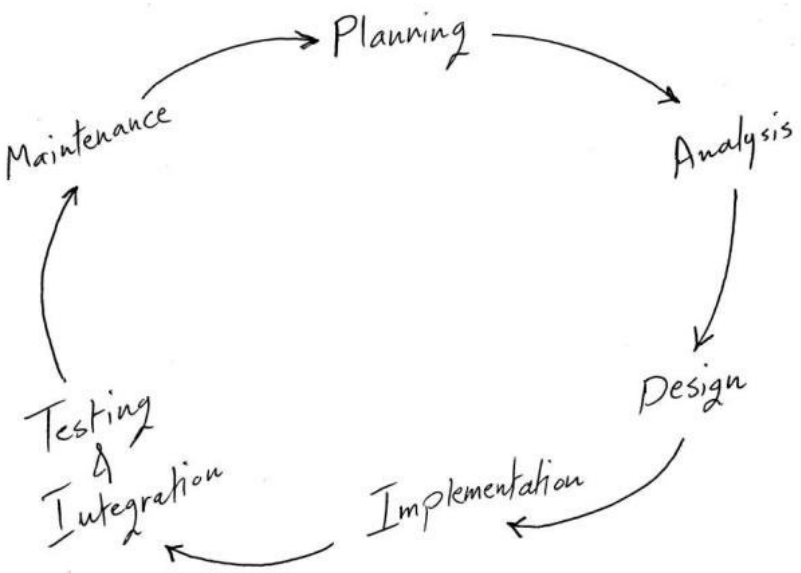
1.4.2. Exercise

1. Ceritakan contoh lain perusahaan yang menjadi game-changer berkat aplikasi.

1.5. Mengenal Proses Development Aplikasi

Dalam menghasilkan/menciptakan Aplikasi yang bagus, maka perusahaan pasti menerapkan suatu sistem dalam pembuatan aplikasi tersebut, atau bisa disebut Development Cycle. Intinya Development Cycle merupakan serangkaian proses dan metode bagaimana

pembuatan aplikasi (atau suatu fitur di aplikasi) dari awal sampai jadi. Mulai dari ide nya seperti apa, perancangannya, bentuk desainnya, hingga implementasi dan menghasilkan suatu output.



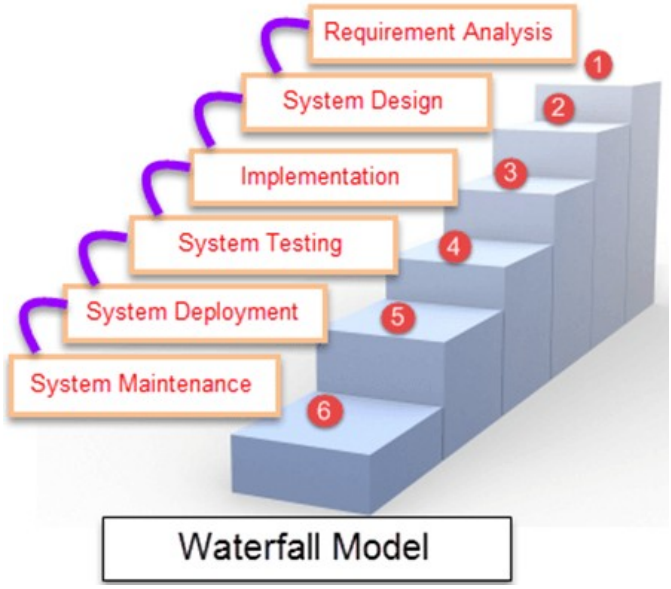
Ilustrasi Development Cycle

Misalnya ketika situs web Bhinneka.com ingin mengeluarkan fitur “Rating” pada kolom produk, nah proses dari awal desainnya, posisi tombolnya dimana, lalu proses pembuatan fiturnya itu sendiri, testing apakah berhasil jalan sesuai harapan atau tidak, proses analisa & revisi, sampai akhirnya fitur itu live dan bisa dinikmati oleh pengunjung situs web Bhinneka.com, itulah yang disebut sebagai Development Cycle.

Sebagai pengetahuan, metode Development Cycle yang paling terkenal ada 2. Yaitu metode Waterfall dan metode Agile. Pengetahuan terkait metode Development Cycle ini perlu dipahami oleh kita, agar nantinya lebih mudah dalam memahami bagaimana DevOps bekerja.

1.5.1. Model Waterfall

Pada intinya model waterfall ini adalah model Development Cycle yang benar-benar kaku dan sudah sangat jelas awal dan akhirnya. Benar-benar searah seperti layaknya air terjun (air yang sudah turun kebawah tidak akan bisa kembali lagi keatas).



Ilustrasi Waterfall

Contohnya begini : Perusahaan Sale Stock ingin membuat aplikasi absensi karyawan. Nah sebelum proses pembuatan aplikasi ini dimulai, maka sudah sangat jelas dan terdokumentasi seluruh spesifikasi dan alur dari aplikasi ini. Misalnya tombolnya warnanya merah, posisinya ditengah, ada fitur fingerprint, setelah klik tombol A maka akan muncul halaman B, dst.

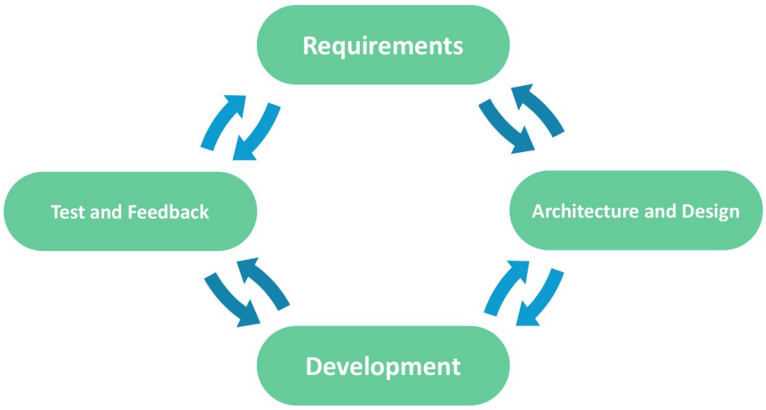
Setelah seluruh dokumentasi dan spesifikasi sudah dibuat, maka barulah proses pembuatan aplikasi dimulai. Output dari aplikasi yang dibuat, harus sama persis dengan spesifikasi awal. Tidak bisa ketika ditengah-tengah proses ngoding aplikasi, tiba-tiba ada penggantian yang tadinya ada fitur fingerprint menjadi fitur kamera. Atau ketika sedang proses testing, tiba-tiba ada perubahan untuk ulang lagi ke proses dokumentasi.

Jadi proses yang sudah berjalan, tidak bisa diulang kembali. Itulah Waterfall Development.

Karena sifatnya yang sangat kaku inilah, Waterfall Development mulai ditinggalkan di era yang sangat dinamis dan cepat berubah seperti saat ini. Waterfall Development lebih cocok digunakan pada pembuatan aplikasi-aplikasi besar yang sifatnya memang tidak akan berubah dalam waktu lama, misalnya seperti Microsoft Office atau Adobe Photoshop.

1.5.2. Model Agile

Untuk mengatasi kekurangan Waterfall Development, muncullah model Agile Development. Alih-alih menggunakan prinsip yang kaku, Agile mengedepankan prinsip iterasi dinamis secara terus menerus. Gambarannya dapat dilihat sebagai berikut :



Ilustrasi Agile Development

Dalam Agile Development, Aplikasi dan fitur yang sudah dibuat berdasarkan spesifikasi yang ditentukan sebelumnya, dapat berubah terus menerus berdasarkan hasil test dan feedback yang didapatkan. Dapat kita lihat pada gambar diatas bahwa pada Agile, tanda panah di masing-masing prosesnya adalah dua arah yang berarti dapat terus menerus melakukan penyesuaian.

Contohnya ketika Twitter pertama kali rilis, kita hanya bisa memposting 140 karakter. Namun ternyata pengguna merasa 140 karakter itu terlalu sedikit. Akhirnya Twitter pun

merevisi batasnya menjadi 300 karakter. Fitur/Aplikasi yang sudah dibuat, tidak ditinggalkan begitu saja. Tapi bisa terus berubah.

Berikut juga ada prinsip-prinsip utama dari metode Agile :

Prinsip-Prinsip Agile (Agile Manifesto)

Gambar diatas menyebutkan beberapa prinsip-prinsip lama pada model Waterfall tergantikan oleh prinsip-prinsip baru yang lebih mengedepankan kolaborasi dan perubahan.

Dengan menerapkan prinsip-prinsip Agile Development, diharapkan proses development aplikasi dapat lebih dinamis mengikuti kebutuhan pengguna. Sangat cocok diterapkan di era saat ini.

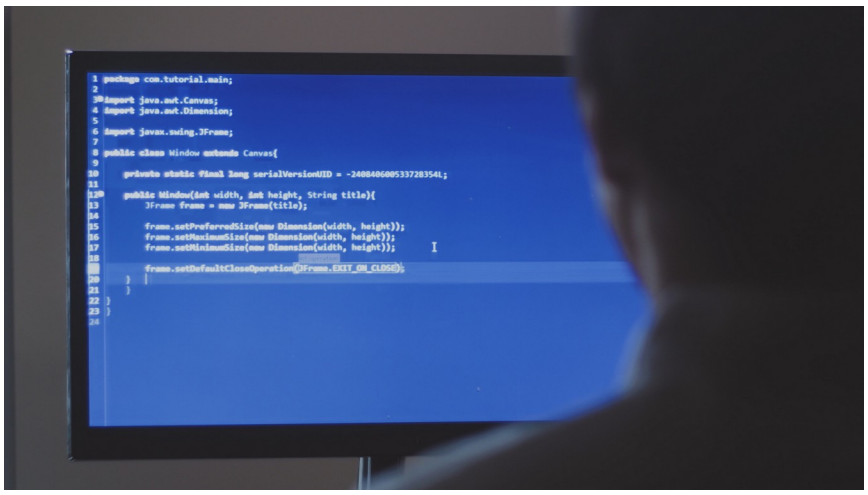
1.5.3. Siapa saja yang terlibat dalam Development Cycle?

Pada prinsipnya yang terlibat dalam proses Development Cycle cukup banyak. Tergantung juga dengan besar perusahaan maupun besar aplikasi yang sedang di kerjakan. Biasanya akan ada orang desainer yang membuat desainnya, ada Programmer yang melakukan coding, ada tim QA/tester yang memastikan apakah aplikasi/fitur yang dibuat sudah sesuai harapan, dll. Namun yang paling utama dalam bidang DevOps, kita bisa kerucutkan menjadi 2, yaitu tim Developer dan tim Operation.

1.5.3.1. Mengenal Developer

Developer (atau biasa disebut Programmer), adalah sang pembuat Aplikasi. Mereka menterjemahkan seluruh kebutuhan-kebutuhan, requirement, dan desain yang ada, menjadi bentuk kode, dan dari kode ini muncullah aplikasi yang diinginkan.

Misalnya Go-Jek ingin membuat fitur Go-Pay agar pengguna mereka bisa membayar tanpa pakai uang tunai. Para Developer ini lah yang “ngoding” agar fitur Go-Pay ini jadi.



Pekerjaan Developer

Hasil coding yang dibuat Developer ini nantinya harus diletakkan/di deploy ke Server agar bisa ditampilkan ke sisi pengguna. Server ini yang mengelola dan yang mengkonfigurasinya adalah tim Operation, yang artinya memang sangat erat kaitannya antara tim Developer dengan tim Operation secara pekerjaan.

1.5.3.2. Mengenal Operation

Operation ini adalah orang-orang yang pekerjaannya mengurusi Server. Mengurusi ini dalam artian memastikan agar kode-kode aplikasi yang sudah dibuat Developer dapat ditampilkan ke pengguna dengan baik. Tim Operation juga memastikan agar aplikasi ini tetap stabil dan tidak down. Tim Operation memastikan semua itu dapat tercapai dengan cara melakukan berbagai konfigurasi, setup dan monitoring di Server.

Pekerjaan Operation

Secara sederhana tugas sehari-hari Operation adalah menerima kode dari Developer, lalu men-deploy kodenya ke server agar aplikasi dapat dinikmati oleh pengguna, lalu melakukan konfigurasi dan monitoring pada Server agar aplikasi tidak down. Biasanya tim Operation juga sering disebut sebagai Sysadmin atau System Engineer.

Agar lebih mudah memahami kaitan antara tim Developer dengan tim Operation kita coba analogikan ini seperti sebuah perusahaan katering makanan dengan tokoh-tokoh sebagai berikut :

Developer = Koki
Aplikasi/Fitur = Makanan
Server = Sepeda Motor
Operation = Kurir

Koki yang membuat masakan sampai jadi. Nah Kurir yang bertanggung jawab memastikan agar masakan ini dapat tiba ke rumah pelanggan dengan baik. Kurir ini melakukan pengecekan dan pengaturan apakah ban Sepeda Motornya sudah dipompa, bensinnya sudah ada, lampunya sudah nyala, stnknya sudah dibawa, dll. Dari semua persiapan dan

pengaturan tersebut, harapannya Kurir dapat naik Sepeda Motor tersebut untuk mengantar makanan ke hadapan para pelanggan dengan lancar.

1.5.4. Exercise

1. Kenapa perlu ada Development Cycle?
2. Kenapa Model Agile lebih baik dibanding Waterfall?

1.6. Masalah Utama pada Proses Development.

1.6.1. Terlalu lambat!

Pada prakteknya, walaupun di environment (laptop/pc) para Developer fitur/aplikasi yang dibuat sudah sesuai yang diharapkan, ketika kode itu di deploy oleh tim Operation ke Server seringkali banyak errornya. Misalnya tadinya tombol A sudah bisa diklik, ternyata di server tombol tersebut tidak bisa diklik.

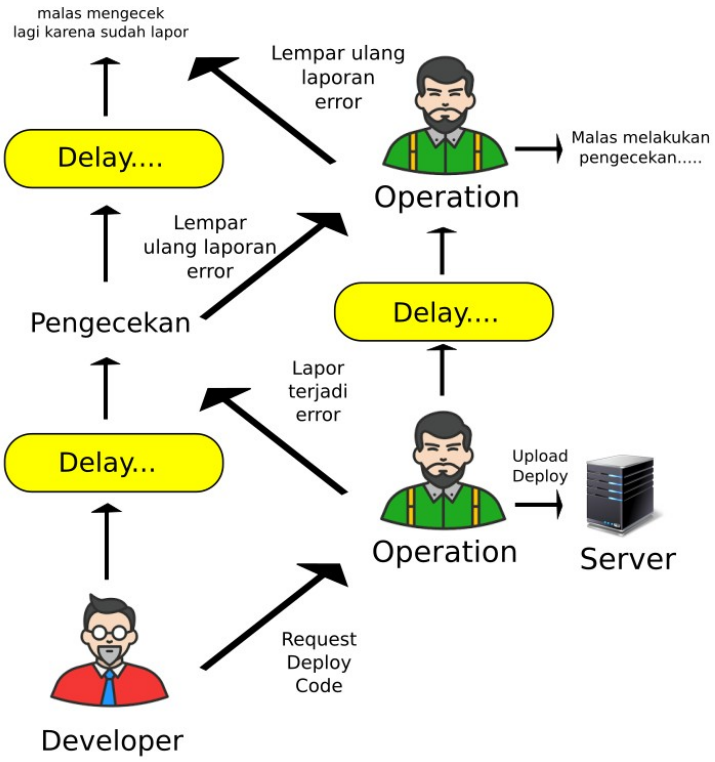
Akibatnya, antara kedua tim ini sering saling menyalahkan apakah errornya gara-gara coding atau gara-gara server. Istilah-istilah seperti “It’s not my code, it’s your server!” dan “It’s not my server, it’s your code!” sudah merupakan hal yang lumrah terjadi.

Developer dan Operation juga memiliki pandangan yang sangat berbeda. Developer menginginkan perubahan atau pengembangan berkala pada software mereka, sedangkan Operation menginginkan stabilitas pada server – server mereka. Antara tim Developer dan tim Operation menjadi seperti memiliki “bunker” masing – masing. Merasa departemen mereka benar-benar terpisah, cenderung “malas” untuk saling membantu.

Akibat seluruh perbedaan ini, sekarang kita coba lihat bagaimana contoh alur yang terjadi ketika misalnya terjadi error dalam sebuah aplikasi :

1. Tim Operation meminta tim Developer mengecek.

2. Ternyata tim Developer tidak mau mengecek karena mereka merasa kesalahan bukan di sisi mereka.
3. Akhirnya tim Developer meminta ulang tim Operation yang mengecek.
4. Tim Operation “malas” mengecek karena mereka merasa sudah mensetup server mereka dengan benar.
5. Begitu terus seterusnya.



Ilustrasi Alur Developer dan Operation yang Lambat

Bisa kita bayangkan akan berapa lama proses delay yang terjadi akibat saling lempar kesalahan tersebut. Lebih parah lagi kalau pada akhirnya tidak ada yang mau bertanggung jawab mencari solusi. Ujung-ujungnya proses delivery sebuat fitur/aplikasi baru menjadi lambat.

Delivery yang lambat ini sangat berpengaruh terhadap masa depan perusahaan. Selain cost operasional meningkat dan revenue perusahaan menurun, juga berpengaruh terhadap posisi perusahaan di industri. Masih ingat saat Snapchat masih terkenal di Indonesia sebelum ada Instagram Story? Andai saja Instagram begitu lambat dalam merilis fitur Instagram Story, maka tidak mungkin Instagram bisa mengalahkan Snapchat seperti saat ini.

1.6.2. Exercise

1. Kenapa lambatnya deliver aplikasi dapat berpengaruh terhadap meningkatnya cost operasional perusahaan ?

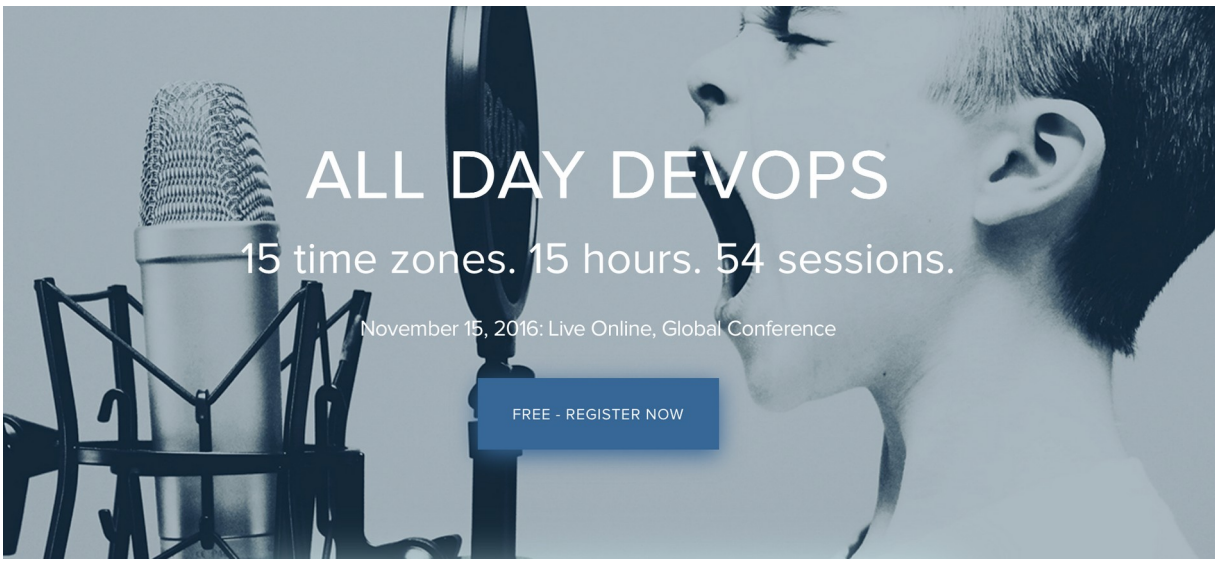
1.7. Bagaimana DevOps Berkembang dan Menjadi Solusi

Dibagian ini kita akan belajar darimana asal mula tren DevOps berasal hingga bagaimana DevOps dapat menjadi solusi dari seluruh masalah yang sudah kita pelajari sebelumnya.

1.7.1. Bagaimana Sejarahnya

Berangkat dari permasalahan-permasalahan diatas, pada awal 2009 seorang berkebangsaan Belgia, Patrick Dubois, bersama seseorang bernama Andrew Schaffer berhasil merumuskan konsep : “Agile System Administration” dalam salah satu momen pada acara diskusi terbuka yang bertemakan “Agile Infrastructure”. Ini merupakan konsep penerapan metode agile pada kegiatan System Administration.

Pada tahun 2010 juga, Flickr mempresentasikan kesuksesan mereka setelah berhasil mengolaborasikan Developer dan Operation dengan judul : *10+ Deploys per Day: Developer and Operation Cooperation at Flickr.* Dari situlah Patrick Dubois terinspirasi dan nama DevOps mulai tersebar. Dan di Belgia pula, DevOps Day pertama diselenggarakan oleh Patrick Dubois.

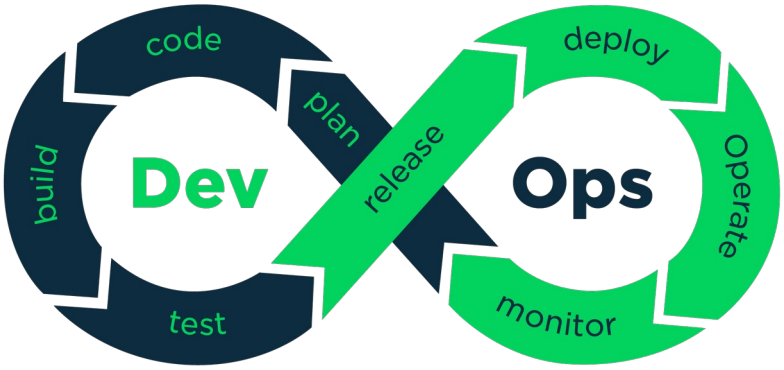


DevOps Day

1.7.2. DevOps sebagai Solusi

DevOps sebenarnya bukanlah suatu jabatan melainkan lebih tepat menyebutnya sebagai sebuah kultur. Sebuah kultur dimana tim Developer dan tim Operation bekerja sama dalam proses development aplikasi agar lebih cepat dan efisien.

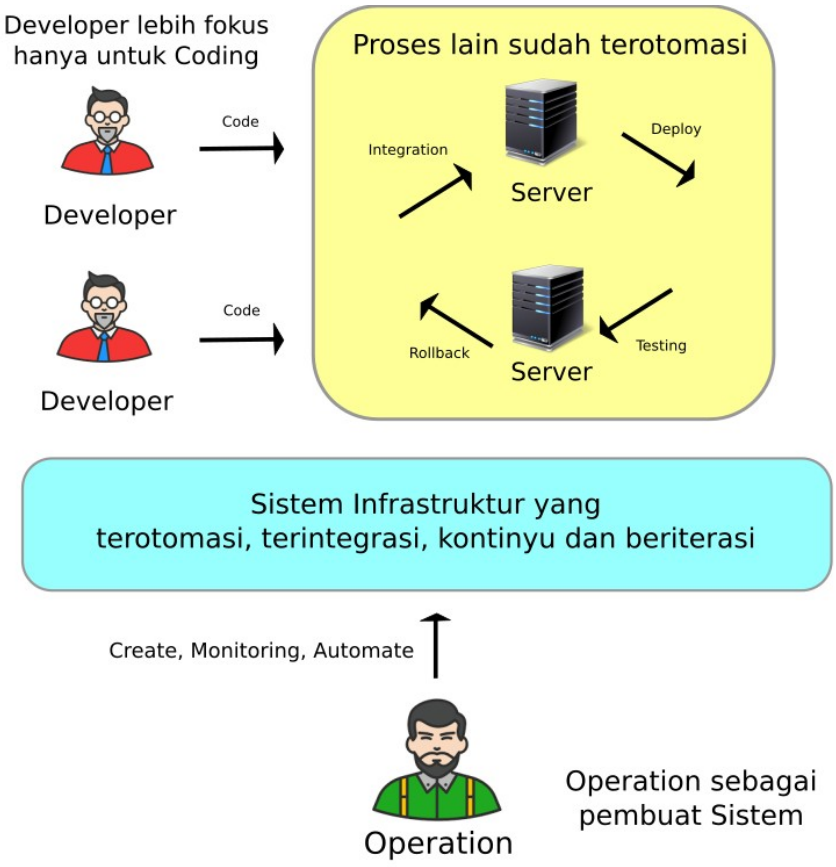
Tim Developer dan Tim Operation tidak lagi bekerja sendiri-sendiri melainkan benar-benar berkolaborasi menjadi satu persis seperti yang tergambarkan pada gambar dibawah :



Ilustrasi Penggabungan Developer dengan Operation

Harapannya dengan penerapan DevOps, terciptalah kultur sistem development yang Iteratif, Kolaboratif, Komunikatif, Kontinyu, Sistematis, dan Ter-otomatisasi.

Dalam kultur DevOps, Tugas Operation dari yang tadinya sebagai tukang mengkonfigurasi server dan tukang deploy code ke server secara manual, berubah menjadi tukang otomasi dan pencipta sistem baru yang kontinyu serta terintegrasi. Tim Operation akan membuat sistem agar tugas Developer menjadi lebih ringan. Developer cukup ngoding saja, tidak perlu lagi pusing-pusing memikirkan masalah Deploy dan segala macamnya. Seluruh proses dari bagaimana mengintegrasikan code-code yang dibuat oleh antar sesama develoepr, bagaimana testingnya, hingga code itu di deploy ke server sampai live bisa dinikmati pengguna, hampir semua akan berjalan serba otomatis.



Ilustrasi setelah penerapan DevOps

Produktifitas kedua tim ini pasti akan meningkat. Karena pekerjaan yang awalnya serba manual dan juga serba delay menjadi hilang sehingga kedua tim dapat lebih fokus mengerjakan hal-hal yang lebih penting, Seperti pengerjaan aplikasi-aplikasi berikutnya maupun melakukan riset dan pengembangan teknologi baru.

Transisi tugas tim Operation yang dulunya serba manual menjadi serba otomatis bisa kita analogikan dalam kasus sebuah Pabrik Roti :

Kita bisa bayangkan jika ada sebuah pabrik roti, maka tadinya Tim Operation ini adalah seorang kuli yang benar-benar mengaduk adonan rotinya sendiri, meng-oven sendiri, dan mem-packing sendiri roti-rotinya untuk naik produksi.

Sedangkan tugas Tim Operation yang baru adalah orang yang memanfaatkan mesin-mesin, robot-robot, dan alat-alat lainnya, kemudian mengatur seluruh mesin, alat dan robot tersebut sedemikian rupa agar bisa melakukan tugas-tugas mengaduk adonan, mengoven, dll secara otomatis sebisa mungkin tanpa campur tangan manusia lagi.

1.7.3. Kaitan DevOps dengan Agile

DevOps sangar erat kaitannya dengan Agile. Jika kita lihat kembali konsep Agile Development yang sudah dibahas sebelumnya, kita bisa melihat betapa Agile dan DevOps memiliki konsep yang serupa. Salah satunya, Agile mementingkan Individual dan interaksi, sebagaimana DevOps mementingkan proses, kultur dan komunikasi. Dibutuhkan pemahaman yang mendalam untuk menemukan persamaan ataupun perbedaan diantara keduanya, namun sejatinya Agile dan DevOps mampu memberikan solusi dari kekurangan Waterfall Development.

Berikut adalah sedikit ilustrasi perbedaan pendekatan proses Development Aplikasi antara Waterfall, Agile, dengan DevOps :

Ilustrasi Waterfall, Agile, dan DevOps

1.7.4. Benefit

Benefit dari penerapan DevOps diantaranya adalah :

1. Delivery Software yang berkelanjutan
2. Berkurangnya kompleksitas untuk memanage / mengatur
3. Resolusi masalah yang lebih cepat
4. Tim yang lebih produktif dan bahagia
5. Keterlibatan Tim yang lebih tinggi
6. Kesempatan yang lebih tinggi untuk pengembangan diri secara profesional
7. Keuntungan Bisnis dan berkurangnya cost operational perusahaan
8. Delivery fitur yang lebih cepat
9. Environment yang lebih stabil
10. Komunikasi dan kolaborasi yang semakin baik
11. Waktu untuk berinovasi lebih banyak

1.7.5. Exercise

Masing-masing 1 Peserta diminta untuk menjawab pertanyaan berikut berdasarkan pemahaman dan bahasa sendiri tanpa melihat modul/slide.

1. Kenapa DevOps dapat menjadi solusi?
2. Apa perbedaan utama DevOps dengan Agile ?

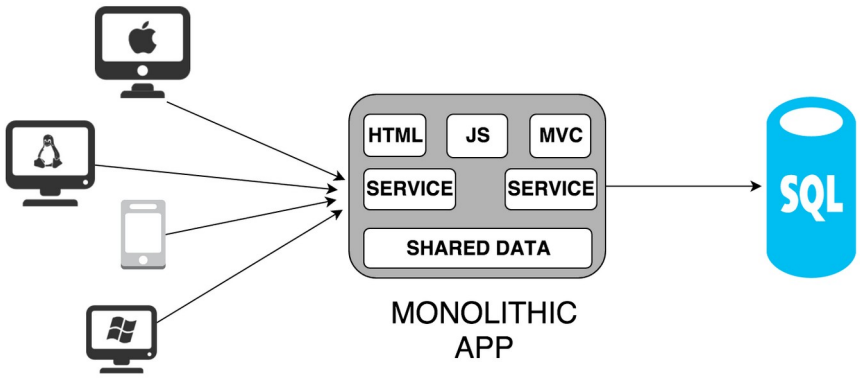
1.8. DevOps Sekarang dan yang Akan Datang

Beberapa tahun kebelakang, startup – startup di Indonesia seakan berebut untuk mengadopsi DevOps. Pertemuan – pertemuan DevOps pun mulai bergulir di Indonesia. Tidak hanya di DevOps Day, tapi juga di Agile conference dan lain sebagainya. Kemudahan dan ketersediaan tools yang beredar di internet makin membuat DevOps digandrungi dari tahun ke tahun. Apalagi dengan terlahirnya arsirtektur Microservices, aplikasi Monolithic sudah mulai ditinggalkan.

1.8.1. Mengenal Aplikasi Monolithic

Sekian lama kita telah hidup dari desain aplikasi yang memiliki arsitektur yang terpusat dan teknologi yang seragam, atau lebih dikenal dengan istilah Monolithic application architecture. Arsitektur aplikasi monolitik ini menggunakan kode sumber dan teknologi yang serupa untuk menjalankan semua tugas-tugasnya.

Untuk memudahkan pemahaman kita, kita ambil contoh aplikasi Wordpress. WordPress merupakan contoh paling mudah untuk menggambarkan arsitektur aplikasi yang bersifat monolitik, dimana dalam satu aplikasi kita dapat memiliki frontend sekaligus backend. Juga Semua fitur security, performance, manajemen konten, statistik, semuanya dibangun dengan menggunakan PHP dan database MySQL, dalam kode sumber yang sama.



Ilustrasi Monolithic Apps

Walaupun Monolithic Apps ini benar-benar sangat mudah untuk dibangun dan paling mudah untuk di jalankan di server, namun Monolithic App memiliki kekurangan besar, yaitu :

1. Ketika aplikasi menjadi besar (banyak yang akses) peforma akan menurun.
2. Ketika akan merubah teknologi pada aplikasi maka akan merubah secara keselutuhan aplikasi.
3. Jika terjadi error pada salah satu fungsi maka akan mempengaruhi keseluruhan aplikasi.
4. Jika terjadi error pada salah satu server juga akan mempengaruhi keseluruhan aplikasi.

Kekurangan-kekurangan Monolithic Apps ini pula lah sebagai salah satu faktor penyebab utama sering terjadinya error dan mis-koordinasi antara tim Developer dengan tim Operation.

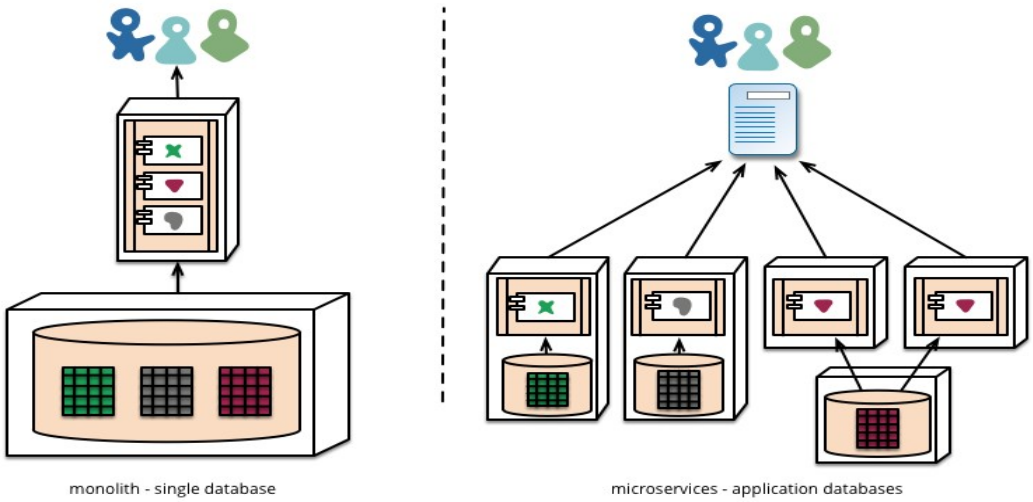
Perkembangan berikutnya adalah dengan memisahkan aplikasi berdasarkan user role-nya. Ada frontend yaitu aplikasi yang akan diakses oleh pengakses blog (user) dan Backend yang digunakan oleh kontributor konten dan admin blog. Bisa juga memisahkan halaman reporting, member dan lainnya, dengan tujuan agar satu fungsi tidak akan mengganggu fungsi yang lainnya.

Pada praktiknya pemisahan halaman ini dilakukan di level domain. Anggap saja kita gunakan subdomain www untuk frontend, subdomain admin untuk backend, subdomain report untuk reporting dan seterusnya. Dan masing-masing subdomain tersebut dapat dilakukan pemisahan server secara fisik. Pemisahan ini dilakukan agar proses scaling dan debuging dapat dilakukan dengan mudah.

1.8.2. Mengenal Microservices

Secara sederhana, arsitektur aplikasi Microservices ini menggunakan desain yang memecah aplikasi berdasarkan fungsinya secara spesifik. Tidak sekedar dengan memisahkan berdasarkan user-role atau subdomain saja, tetapi aplikasi akan di breakdown lebih rinci lagi dari sisi fungsionalitasnya. Aplikasi akan dirancang agar setiap fungsi bekerja secara independent. Dan setiap fungsi dapat menggunakan teknologi stack yang sesuai dengan kebutuhan, walaupun itu artinya akan terdapat teknologi yang berbeda-beda dalam satu aplikasi besar.

Berikut adalah sedikit ilustrasi perbedaan Microservices dengan Monolithic :



Ilustrasi perbedaan Microservices dengan Monolithic

Dengan pemisahan aplikasi berdasarkan fungsi-nya ini, pada akhirnya kita akan menemui keragaman teknologi dalam sebuah satu layanan digital. Misalkan dari layanan blog yang

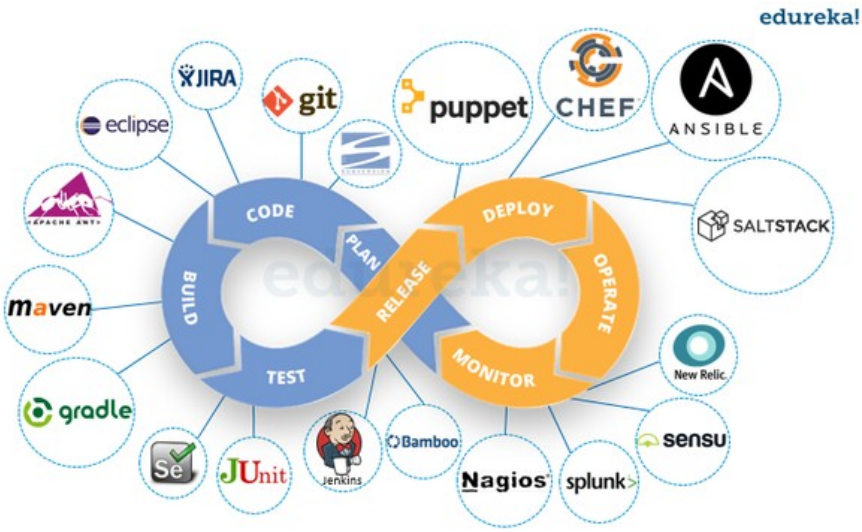
telah dicontohkan diatas, kita bisa coba pecahkan aplikasi blog tersebut menjadi fungsi konten, user management, komentar, rangking, search dan lainnya.

Pada bagian fungsi konten mungkin kita menggunakan PHP dan mysql, tetapi pada komentar kita menggunakan Python dan Mongodb, sedangkan di search menggunakan nodejs dan elasticsearch untuk penyimpanan datanya. Disini terlihat bahwa setiap fungsi / permasalahan teknis dapat diselesaikan dengan cara dan teknologi yang berbeda-beda.

Dalam konsep microservices, kita tidak hanya melalukan pemisahan di level aplikasi saja, tetapi dari sisi infrastruktur pula. Kita akan menemukan keragaman arsitektur infra, konfigurasi dan optimalisasi sistem yang berbeda.

1.8.3. Tools-Tools DevOps

Tools DevOps sangatlah beragam. Tergantung kebutuhan dan pada saat kapan tools itu dibutuhkan.



Ilustrasi Tools DevOps

Contohnya salah satu tools yang sangat menunjang Microservices dikarenakan fleksibiltas dan skalabilitasnya yang sangat baik dibandingkan dengan virtualization pada umumnya adalah Docker. Ukuran docker yang sangat kecil membuatnya sangat mudah dalam melakukan deployment, rollback ataupun update aplikasi.

Teknologi-teknologi seperti Infrastructure as Code contohnya, juga dapat menghadirkan otomasi pada level server, sehingga membuat Provisioning server menjadi lebih ringkas, cepat dan konsisten. Berbagai instalasi yang biasa kita lakukan setelah instalasi OS, bisa kita persiapkan sebelumnya termasuk pemilihan OS, hardware, dsb.

Adalagi beberapa tools untuk Orchestration yang sangat digandrugi belakangan ini yaitu Docker Swarm, Kubernetes dan ECS. Fleksibilitas, skalabilitas dan fitur – fitur nya yang luar biasa menjadikan mereka primadona di dunia DevOps.

Azure sphere juga menjadi inovasi penting seiring terus berkembangnya IoT. Bisa dibayangkan betapa mudah dan cepatnya proses update tiap device IoT jika mengadopsi DevOps.

Sedangkan untuk best practice tools-tools yang sering digunakan DevOps di industri di Indonesia adalah :

1. Amazon Web Services
2. Git
3. Ansible
4. Jenkins
5. Docker

Banyaknya tools yang ada ini tidak perlu membuat kita khawatir. Karena salah satu keunikan DevOps juga adalah, ketika kita berpindah dari satu perusahaan ke perusahaan lainnya, kita tidak perlu bingung dengan perbedaan tools DevOps yang dipakai. Karena apapun tools yang dipakai, tetap memiliki konsep yang sama. Sehingga tidak memakan waktu lama untuk mempelajari tools baru.

1.8.4. Exercise

Masing-masing 1 Peserta diminta untuk menjawab pertanyaan berikut berdasarkan pemahaman dan bahasa sendiri tanpa melihat modul/slide.

1. Kenapa Monolithic lebih buruk dari Microservices?
2. Kenapa DevOps masih akan menjadi tren kedepan?

1.9. How DevOps Works

Efisiensi yang ditimbulkan oleh DevOps membantu menciptakan kondisi dimana tim Developer menjadi memiliki waktu yang lebih untuk bekerja, untuk melakukan riset maupun mengembangkan produknya dibanding disibukkan dengan masalah deployment. Mari kita bahas beberapa cara kerja DevOps untuk melihat bagaimana hal ini dapat terjadi.

1.9.1. Cara Kerja DevOps – Continuous Everything

Prinsip utama dari cara kerja DevOps adalah Continuous Everything. Artinya dalam setiap proses pekerjaannya harus selalu menanamkan mindset bahwa tidak ada proses yang hanya 1x dikerjakan lalu selesai, melainkan apapun harus tetap berjalan dengan baik secara terus menerus.

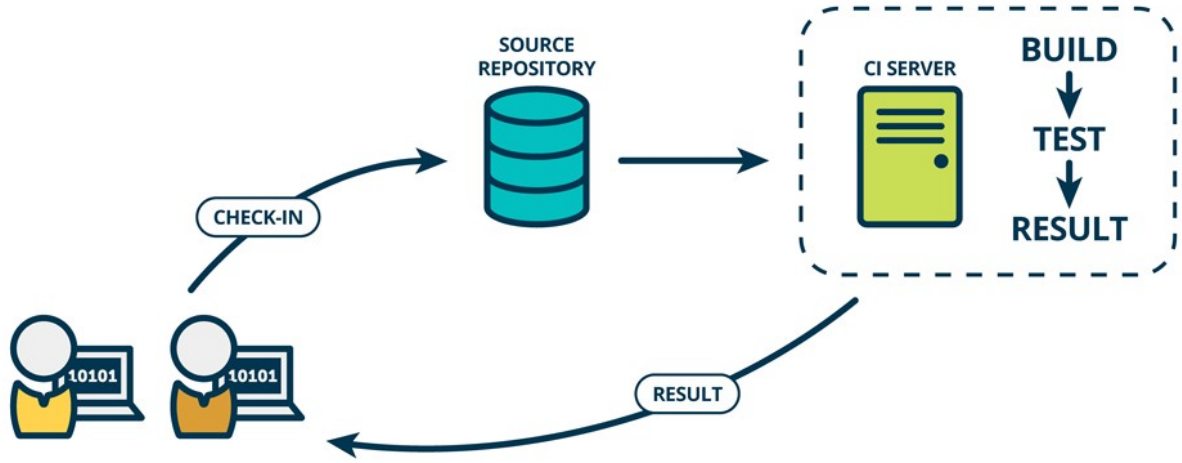
1.9.1.1. Continuous Integration

Perlu diketahui bahwa setelah tim Developer berhasil membuat code-code mereka berjalan di laptop mereka, tahap selanjutnya adalah bagaimana agar code-code ini dapat berfungsi juga di server. Disinilah ada yang namanya tahapan mem-build. Intinya kode-kode ini akan di kemas menjadi sebuah bentuk yang siap untuk nanti di deploy ke Server. Nah sebelum adanya DevOps, tahapan untuk membuild ini dilakukan secara manual yang tentunya sangat tidak efektif dan rentan kesalahan.

Continuous Integration adalah step untuk meng-otomasi proses build itu tadi. Trigger yang dipakai bisa dari kegiatan push, merge ke branch yang dilakukan oleh Developer menggunakan salah satu tools DevOps yaitu Git.

Di dalam step ini terdapat juga Continuous Testing untuk memvalidasi hasil build aplikasi sesegera mungkin agar meminimalisir error-error yang mungkin terjadi pada fase berikutnya.

Setelah semua selesai, didapatkanlah hasil build yang bisa berupa artifact, Docker Image, ataupun bentuk hasil publish lainnya (tergantung metode dan tools yang digunakan). Dari sini ada 2 tahapan yang bisa kita pilih, Continuous Deployment, atau Continuous Delivery.



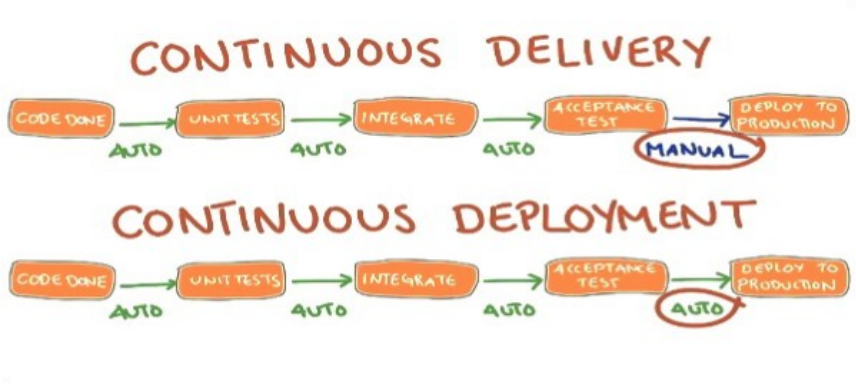
Ilustrasi Continuous Integration

1.9.1.2. Continuous Deployment

Pada step ini, hasil build yang sudah lulus test dalam bentuk artifact, docker image, ataupun package lainnya, langsung dideploy ke server – server yang diinginkan sehingga terciptalah sebuah produk. Semuanya berjalan otomatis setelah trigger untuk Continuous Integration terjadi.

1.9.1.3. Continuous Delivery

Pada Continuous Delivery, yang menjadi trigger untuk deployment adalah campur tangan manual dari manusia. Kita yang menentukan aplikasi mana dan versi dengan build number berapa yang akan di deploy. Continuous Delivery diberlakukan pada fase yang cukup riskan dan yang berhubungan langsung dengan pengguna. Berikut adalah ilustrasi perbedaan antara Continuous Deployment dan Continuous Delivery :



Ilustrasi perbedaan Continuous Delivery dan Continuous Deployment

1.9.1.4. Continuous Improvement

Penerapan kultur DevOps lebih memacu terjadinya banyak perubahan-perubahan, peningkatan-peningkatan, perbaikan-perbaikan yang sebenarnya kecil-kecil, tetapi dilakukan secepat dan sesering mungkin. Efeknya terjadi inovasi terus menerus yang perusahaan akan terus maju dan mampu tetap bersaing di industri.

1.9.1.5. Continuous Value for Customer

Seluruh proses Continuous diatas berfokus pada value yang dirasakan oleh pengguna secara terus menerus. Inilah salah satu keuntungan besar dari DevOps. Ketika kita menerapkan Continuous Everything dalam development aplikasi, ujung-ujungnya yang paling merasakan keuntungannya adalah pasar atau pengguna. Pengguna akan mendapatkan value yang mereka butuhkan dan inginkan secara cepat. Dan ketika pengguna merasa diuntungkan, yang terjadi adalah: revenue meningkat. Sebuah win-win solution yang ditawarkan oleh DevOps.

Ilustrasi Continuous Value Delivery

1.9.2. Mengenal Environment

Dalam pengembangan aplikasi, Environment dapat diartikan sebagai sebuah “wadah” untuk aplikasi itu sendiri. Wadah disini terdiri dari sekumpulan komponen yang mendukung agar aplikasi dapat berjalan dengan baik, diantaranya :

1. Server
2. Sistem Operasi
3. Database
4. Development Tools
5. dll

Ilustrasi Environment

Misalnya agar website pemerintahan Kota Bandung dapat berjalan dengan baik, maka dibutuhkan Environment berupa :

1. Sebuah Server dengan spek dual core, RAM 4 GB, harddisk 1TB.
2. Servernya diinstall Sistem Operasi Ubuntu Server 16.04 LTS
3. Didalam Ubuntu Server tersebut diinstall layanan Database server Mysql versi 5.5 dan Webserver NGINX versi 3.5

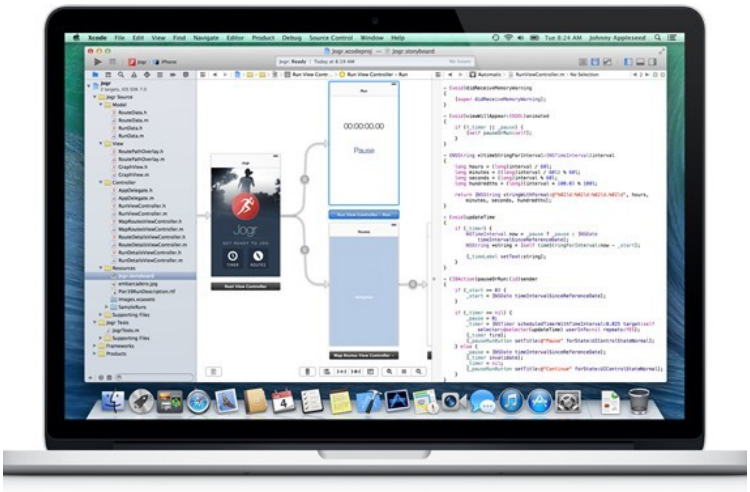
Analogi sederhana Environment pada dunia DevOps ini sama seperti Environment pada dunia nyata. Misalnya ketika kita ingin mempunyai tanaman kopi yang subur, maka

Environment minimalnya misalnya tanahnya harus berada di ketinggian minimal di 2000mdpl, kelembaban 17%, dan seterusnya.

Biasanya, walaupun aplikasi yang dikembangkan hanya 1, namun tetap membutuhkan beberapa Environtment yang dibagi sesuai kebutuhan dan kondisi. Setiap perusahaan memiliki terminologi/penyebutan Environment yang berbeda-beda, tetapi best practicenya biasanya di bagi sebagai berikut :

1.9.2.1. Environment Dev

Ini adalah Environment tempat pengembangan software terjadi, sederhananya di laptop/PC masing – masing Developer. Seluruh coding yang pertama kali dilakukan di Environment Dev ini. Sehingga biasanya di masing-masing laptop Developer akan menyesuaikan spesifikasi dan sistem operasinya, dan diinstall berbagai service dan software yang diperlukan. Misalnya jika Developer aplikasi untuk Iphone, maka dia wajib menggunakan Macbook dan diinstall tools XCode. Tidak bisa misalnya menggunakan komputer Windows biasa.



Contoh Environment Dev

1.9.2.2. Environment Test

Environment Test adalah Environment yang ditujukan untuk mengetes perubahan yang terjadi di sebuah aplikasi, sehingga ketika error terjadi, hal tersebut bisa ditanggulangi secepat mungkin. Environment Test sama sekali tidak berhubungan langsung dengan pengguna. Jadi ketika dilakukan berbagai trial dan error disini tidak akan mempengaruhi apapun.

Contohnya Developer sedang mendevelop fitur Add to Cart pada situs toko online perusahaannya, maka seluruh proses coba-coba terhadap codingan-codingan yang dibuat akan ditest dulu di Environment ini. Ketika benar-benar sudah dipastikan tidak ada masalah dan sudah sesuai spesifikasi, barulah code-code fitur tersebut bisa dilanjutkan ke Environment berikutnya.

Seluruh spesifikasi server, sistem operasi, service-service yang diinstall pada Environment Test dibuat semirip mungkin dengan Environment Production untuk meminimalisir error.

1.9.2.3. Environment Staging

Environment Staging adalah Environment replika dari Environment Production, dimana semua instalasi, konfigurasi, migrasi dipastikan sudah berjalan baik sebelum dilempar ke Environment Production. Karena walaupun sudah melewati proses testing di Environment Test, kemungkinan terjadi error masih tetap ada. Di Environment inilah hal tersebut dipastikan. Seluruh spesifikasi server, sistem operasi, service-service yang diinstall dan segala macamnya juga benar-benar dibuat sama persis dengan Environment Production. Sehingga dapat diasumsikan apa yang sudah berjalan di Environment Staging, pasti juga berjalan di Environment Production.

1.9.2.4. Environment Production

Environment Production adalah Environment utama dan Environment yang paling sensitif karena langsung bersinggungan dengan pengguna, dimana kualitas dan minimum downtime adalah hal – hal yang sangat dipertahankan. Ketika code sudah masuk ke Environment Production, artinya code tersebut sudah akan tampil langsung ke pengguna.

Pada intinya seluruh environment diatas adalah wadah-wadah yang disesuaikan untuk fungsinya masing-masing pada sebuah Development Cycle aplikasi. Awal tim Developer buat code maka di Environment Dev di laptop sendiri, Setelah sudah siap untuk di masukkan ke server untuk di test-test maka dimasukkan ke Environment Dev. Setelah sudah benar-benar dipastikan codenya berhasil dan lulus segala macam test, maka di masukkan ke Environment Staging dulu sebagai sebuah “Gladi Resik”. Ketika sudah benar-benar siap untuk dinikmati oleh pengguna, barulah dimasukkan ke Environment Production.

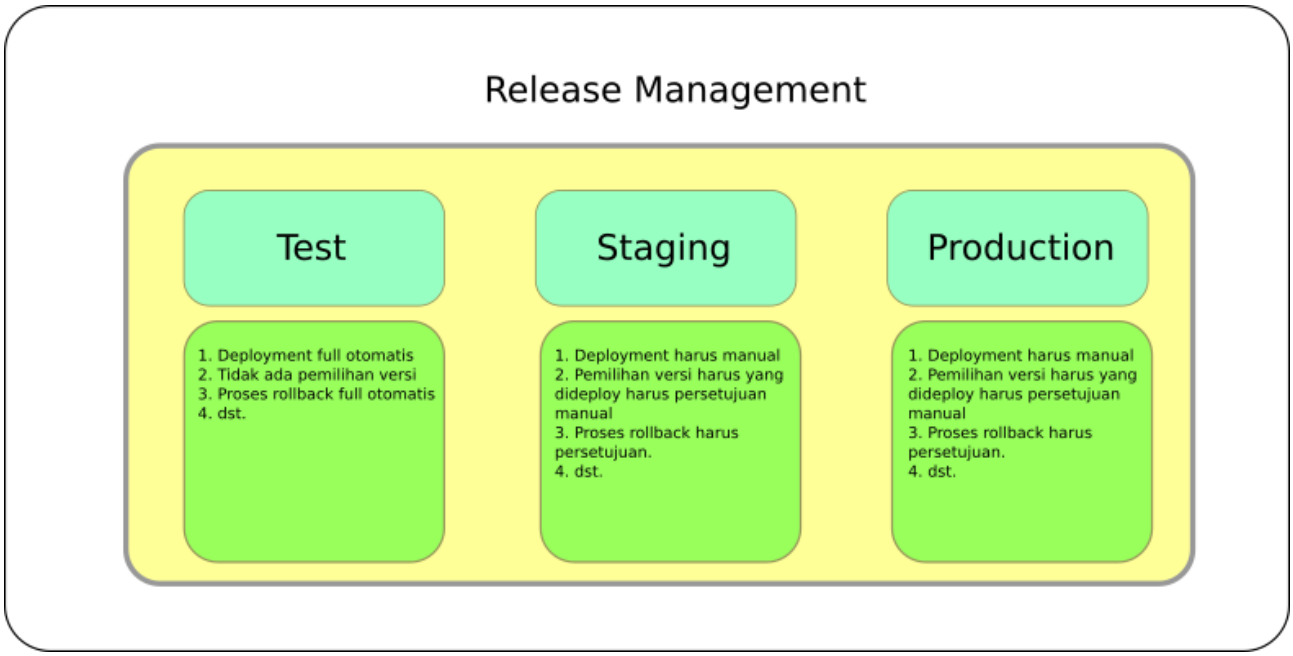
1.9.3. Release Management

Dengan adanya kebutuhan dan kondisi yang berbeda dari masing-masing Environment ini lah yang membuat Release Management diperlukan. Release Management itu sendiri adalah proses pengelolaan, perencanaan, penjadwalan, dan pengendalian aplikasi yang dibangun melalui berbagai tahap dan environment.

Intinya adalah kita harus membuat sistem yang baik (baik disini berarti sistemnya dapat terotomasi, terintegrasi dll sesuai cara kerja DevOps yang sudah dibahas sebelumnya) terkait bagaimana alur sebuah aplikasi dari pertama kali dibuat sampai benar-benar di rilis ke pengguna. Goalnya adalah tetap dapat melakukan eksperimen dan inovasi sebanyak mungkin, namun dengan tingkat error yang sekecil mungkin.

Misalnya walaupun kita ingin proses development seotomatis dan secepat mungkin, kita tetap tidak bisa serta merta mengadopsi Continuous Deployment ke Environment Staging atau Production. Faktor yang menyebabkan error pasti sangat banyak. Bayangkan ketika Continuous Deployment dilakukan ke Environment Production tetapi konfigurasi yang dibawa masih belum matang, yang terjadi adalah downtime tanpa diketahui. Kenapa? Karena Continuous Deployment sifatnya serba otomatis. Sehingga lebih tepat kita menerapkan Continuous Delivery pada Environment Staging dan Production, agar masih ada sisi approval secara manual untuk meminimalisir lolosnya konfigurasi yang belum matang ke pengguna.

Sebaliknya, kita juga tidak bisa menerapkan Continuous Delivery pada semua Environment. Misalnya pada Environment Test. Jika diterapkan Continuous Delivery, maka sisi Agile nya akan berkurang. Akan sangat memakan waktu dan tenaga apabila deployment di test server tidak berjalan secara otomatis. Sehingga pada Environment Test lebih cocok menggunakan Continuous Deployment.



Ilustrasi Release Management

Dengan strategi Release Management yang baik, maka akan sangat menunjang dalam penerapan kultur DevOps yang efektif di perusahaan.

1.9.4. Metrik DevOps

Ilustrasi Data-data yang diukur

Selain praktek – praktek di atas, ada beberapa metrik yang harus diperhatikan, karena semua yang terjadi di dalam DevOps harus terukur. Kecepatan automation - baik itu build, test atau deploy, kecepatan recovery ketika terjadi insiden, frekuensi deployment, dan lain sebagainya. Beberapa metrik yang umum digunakan adalah sebagai berikut :

1. Tingkat Availability – Ini metric untuk mengukur seberapa stabil aplikasi kita yang sudah live. Semakin jarang down, maka semakin baik.
2. Mean Time to Detection (MTTD) – Ini metric yang dapat mengukur seberapa cepat kita bisa mendeteksi adanya error pada aplikasi. Semakin cepat kita mendeteksi error, maka semakin cepat kita memperbaikinya.
3. Mean Time to Recovery (MTTR) – Ini metric yang dapat mengukur seberapa cepat kita bisa memperbaiki kegagalan sistem (downtime, lambat diakses, dll) sampai kembali normal. Semakin cepat, tentunya semakin baik.
4. Lead Time – Ini metric yang dapat mengukur berapa waktu yang dibutuhkan dari pertama kali aplikasi/fitur dikerjakan sampai bisa Live di Environment Production. Semakin cepat maka semakin baik.

Semua metrik ini akan sangat tergantung dan berbeda-beda di tiap perusahaannya. Namun pada dasaranya fungsinya sama, Metrik – metrik itulah yang akan membuktikan seberapa efektif DevOps di tempat kita.

1.9.5. Seberapa Penting Kultur DevOps

Semua cara kerja DevOps sebelumnya tidak akan pernah bisa berjalan, tanpa adanya kultur kolaborasi dan komunikasi yang baik. Karena DevOps berjalan bukan karena kita merekrut seseorang yang mengetahui teknologi dan tools nya. Pondasi sesungguhnya dari DevOps adalah manusia dan kulturnya, bukan di teknologi ataupun toolsnya. Tools hanyalah sebatas sarana. Akan sangat membuang energi apabila seseorang yang berpengalaman dalam bidang DevOps hadir dengan tim yang tidak mengadopsi kultur DevOps. Karena yang akan terjadi adalah tidak akan berkembangnya komunikasi yang baik, antar departemen masih mengunci diri masing – masing, hingga perputaran informasi yang sangat diperlukan untuk menjalankan DevOps jadi terhambat. Akibatnya pelaksanaan pun tidak akan se-efektif yang diharapkan.

Seperti yang sudah dibahas pada bab awal, pada dasarnya memang akan sangat mudah bagi tim Developer untuk tidak peduli terhadap apa yang terjadi dengan persoalan Server apabila urusan tersebut sudah diberikan kepada tim Operation. Bahkan hal yang umum terjadi adalah saling menuding ketika terjadi permasalahan di Environment.

Ilustrasi saling menyalahkan antara tim Developer dengan tim Operation.

Oleh karena itu dibutuhkan juga sebuah kultur Shared Responsibilities. Kultur yang satu ini akan membuat semua elemen dalam Development Cycle memiliki rasa bertanggung jawab atas pengembangan produk dan infrastrukturnya. Apabila kultur Shared Responsibilities sudah diterapkan, Tim Developer dan Operation akan berbagi tanggung jawab akan pekerjaannya. Tim Developer akan mengetahui apa saja masalah-masalah yang dirasakan dari Tim Operation, begitu pun sebaliknya. Hal itu akan memicu kedua Tim untuk saling bekerja sama mencari solusi, bukan mencari kesalahan. Lebih fokus kepada intropeksi dan improvement agar insiden bisa diminimalisir.

Pada akhirnya, kultur DevOps di perusahaan tidak akan terjadi dalam semalam. Jika sebuah perusahaan ingin mengimplementasi DevOps, mereka harus siap untuk menyatukan semua tim yang ada. Bagaimana membuat berjalannya kolaborasi dan komunikasi dalam perusahaan tersebut, untuk merumuskan tujuan yang sama.

Jika DevOps dianalogikan sebagai jalan, tools hanyalah kendaraan yang kita bawa. Dengan skill dan kendaraan yang tepat, kita akan melaju cepat ke tujuan. Tapi kultur, adalah infrastruktur, jalan itu sendiri. Bayangkan jika kita memilih Lamborghini dengan keadaan jalan tol yang berlubang, sampah berserakan dan berdebu, atau yang berlumpur. Secepat apapun kendaraannya, kita akan kewalahan.

Analogi Teknologi DevOps yang tidak didukung oleh kultur

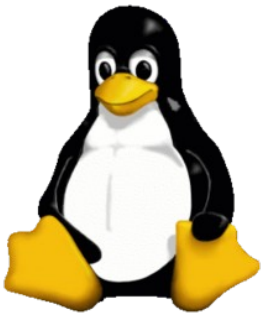
Itulah DevOps, sebuah kultur yang dibentuk untuk mengembangkan tidak hanya produk yang berkualitas, tapi juga semua orang yang terlibat di dalamnya.

1.9.6. Skill dan Background DevOps

Setelah semua semua penjelasan DevOps di atas, dari mana kita harus memulainya? Apa saja skill yang harus dipelajari? Background apa yang cocok untuk pekerjaan ini? Mari kita bahas skill dan knowledge dasar apa saja yang harus dimiliki agar kita bisa memulai karir sebagai seorang DevOps Engineer.

1.9.6.1. Linux

Sederhananya Linux adalah sebuah Sistem Operasi yang sama persis seperti Windows maupun MacOS. Namun Linux lebih cocok untuk kebutuhan Server, karena dianggap lebih stabil, ringan, aman dan harganya gratis.



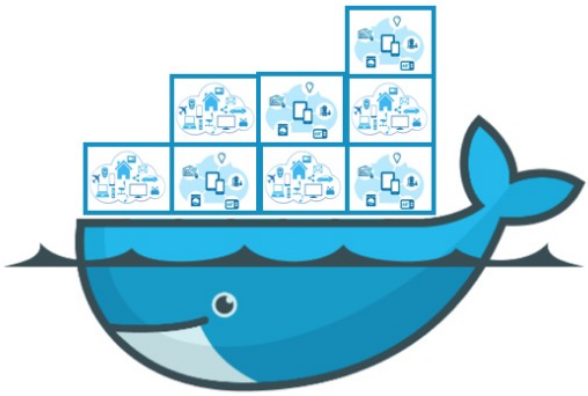
Logo Linux

Tools-tools dan layanan yang ada di Linux secara fitur dan kualitas tidak kalah dengan yang berbayar. Pemrograman cross platform pun semakin marak. Developer mulai beralih ke Linux karena banyak sekali bahasa pemrograman yang disupport untuk bisa berjalan di Environment Linux. Selain itu Linux juga memiliki bahasa scripting bernama Bash yang sangat powerful untuk menangani kebutuhan-kebutuhan kompleks sehubungan dengan macam-macam tugas DevOps nantinya.

Kita akan sehari-hari berurusan dengan Linux, oleh karena itu ini menjadi salah satu skill wajib yang harus dikuasai oleh seorang DevOps.

1.9.6.2. Docker/ Teknologi Kontainerisasi

Seperti yang sudah kita bahas sebelumnya, penerapan Microservices sangat penting dalam Development Aplikasi. Dan Docker adalah tools yang sangat mendukung tipe arsitektur microservices dimana tiap aplikasi memiliki environment nya masing – masing, terisolasi, tapi saling terkoneksi dengan menggunakan teknologinya yang bernama Container. Update akan lebih mudah karena kita hanya fokus pada service yang dimaksud tanpa perlu takut mengganggu service lainnya. Jika salah satu service mengalami downtime pun akan lebih cepat melakukan recovery dengan adanya container orchestration.



Logo Docker dan Ilustrasi Container

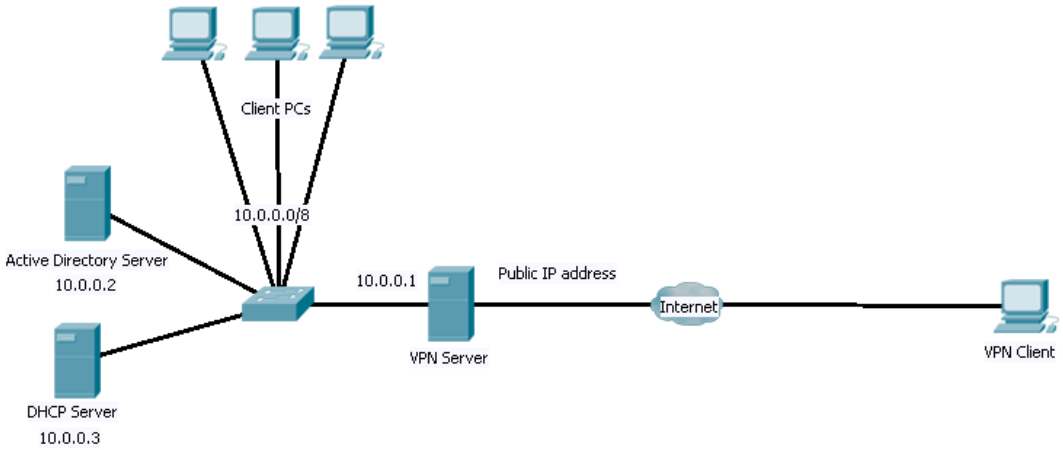
1.9.6.3. Cloud Computing

Semakin banyak orang beralih ke Cloud server karena flexibility, high availability dan servis yang ditawarkan sangat beragam. Initial cost yang harus dikeluarkan pun jauh lebih kecil dibandingkan server on-premise (server fisik). Kecepatan provisioning juga menjadi salah satu benefit, tidak usah menunggu berhari – hari untuk membeli dan menyalakan satu ataupun banyak server. Dalam hitungan jam, bahkan menit, server kita bisa langsung tersedia dengan beberapa kali klik saja. Beberapa provider Cloud Computing yang akan sering digunakan oleh DevOps adalah Amazon Web Service, Google Cloud Platform dan Azure.

Logo Amazon Web Services

1.9.6.4. Jaringan dan Keamanan

Poin penting dari sisi Operation yang memang harus diterapkan dimana – mana adalah ilmu-ilmu dasar dari jaringan dan keamanan. Minimal kita harus tau bagaimana cara login ke Server dari luar kota, tau bagaimana menghubungkan antara Server, tau bagaimana cara agar Server bisa diakses dari internet, tau bagaimana konsep komunikasi antar perangkat di jaringan, hingga tau cara membaca gambar topologi sebuah jaringan.



Contoh Topologi Jaringan

Hal ini sangat berguna ketika ingin mengadopsi berbagai teknik-teknik dan solusi DevOps seperti Infrastructure as Code, Configuration Management, scripting, dan lain sebagainya. Selain itu tujuan dari DevOps adalah menyediakan aplikasi untuk pengguna dengan aman. Karena seluruh aplikasi yang kita deliver ke pengguna tidak akan berarti apabila aplikasi itu

ternyata sering down ataupun malah berhasil di hack oleh orang yang tidak bertanggung jawab.

1.9.6.5. Soft Skill

Kolaborasi, komunikasi dan fleksibilitas yang baik sangat diperlukan untuk membangun kultur DevOps yang sebenar – benarnya. Pohon yang rindang tanpa akar yang baik akan jatuh ketika angin besar melanda. Jangan sampai hal itu terjadi di tempat kita. Maka dari itu, kita harus perkuat akar dari DevOps itu sendiri, yaitu kultur kolaborasi dan komunikasi.

1.9.7. Exercise

Masing-masing 1 Peserta diminta untuk menjawab pertanyaan berikut berdasarkan pemahaman dan bahasa sendiri tanpa melihat modul/slide.

1. Kenapa harus memisahkan Environment ke dalam beberapa fase?
2. Kenapa kita tidak bisa menerapkan Continue Deployment di semua Environment?
3. Kenapa DevOps harus mempelajari Linux?
4. Kenapa kultur DevOps begitu penting dibandingkan tools?

1.10. DevOps di Mata Industri

1.10.1. Fakta Pengaruh DevOps terhadap Perusahaan

Angka State of Devops 2016

Angka di atas adalah perbandingan High performing organizations dibandingkan lower-performing peers nya. Angka di atas bukan rekaan semata, tetapi didapat dari survey selama 5 tahun oleh Puppet, pemain besar di Configuration Management. Survey itu didapat lebih dari 25 ribu IT Profesional, yang bertujuan untuk memahami lebih dalam bagaimana DevOps memberi dampak kepada departemen IT dan performa perusahaan. Dan mereka mengonfirmasi bahwa perusahaan perlu melakukan investasi yang sama baik kepada staff maupun teknologinya. Selain angka yang luar biasa di atas, ada penemuan – penemuan penting yang harus kita ketahui:

1. High performing Org. memiliki pekerja yang lebih loyal
2. Mereka juga menghabiskan 22% waktu lebih sedikit dalam melakukan pekerjaan yang di luar rencana, dan mereka memiliki 29% waktu lebih untuk pekerjaan baru, seperti fitur atau code baru.
3. Melakukan eksperimen terhadap pengembangan produk dapat meningkatkan performa
4. Melakukan transformasi digital dapat menghasilkan keuntungan yang cukup besar

Hal – hal di atas adalah benefit yang ditemukan dalam perusahaan – perusahaan yang mengadopsi DevOps. Mengapa ada High performing dan lower-performing organizations? Bukan kah semua yang mengadopsi DevOps seharusnya memiliki performa yang baik? Kembali kepada bagaimana dibentuknya DevOps dalam suatu perusahaan. Seperti yang yang sudah disebutkan, DevOps tidak terjadi dalam satu hari, mungkin lower-performing organisasi yang disebutkan di atas belum lama mengadopsi DevOps, atau mungkin proses dan kultur mereka belom terbentuk sehingga mereka masih dalam proses menyempurnakan setiap aspek DevOps yang sedang terjadi.

Setelah mengetahui State of DevOps 2016, mari kita melangkah ke 2017.

Table 1: Changes in IT performance of high performers, 2016 to 2017

IT performance metrics	2016	2017
Deployment frequency	200x more frequent	46x more frequent
Lead time for changes	2,555x faster	440x faster
Mean time to recover (MTTR)	24x faster	96x faster
Change failure rate	3x lower (1/3 as likely)	5x lower (1/5 as likely)

Tabel perbedaan State of DevOps tiap tahun

Kita bisa lihat bahwa nilai statistiknya semakin sempit, yang artinya High-performer vs lower-performer memiliki gap yang semakin kecil, perusahaan – perusahaan memiliki hasil yang semakin baik dan semakin matang dalam mengadopsi DevOps di tempatnya masing – masing. Tapi berbanding terbalik dengan stabilitasnya, kita bisa liat waktu untuk recovery dan kemungkinan failure semakin tinggi. Itu menyiratkan bahwa low-performers masih bekerja untuk meningkatkan aspek kecepatan mereka, tapi belom berinvestasi dalam membangun kualitasnya. Hasilnya adalah angka failure yang besar dan waktu yang lebih banyak untuk recovery. High-performers mengerti bahwa mereka tidak perlu mengutamakan kecepatan untuk mendapatkan stabilitas, karena dengan membangun kualitas, mereka mendapatkan keduanya.

1.10.2. Exercise

Masing-masing 1 Peserta diminta untuk menjawab pertanyaan berikut berdasarkan pemahaman dan bahasa sendiri tanpa melihat modul/slide.

1. Kenapa tingginya frekuensi deployment justru tidak berbanding lurus dengan kualitas DevOps itu sendiri?

1.11. Summary

Berikut adalah rangkuman poin-poin penting materi yang sudah dipelajari dari bab ini :

1. Perusahaan yang mampu cepat berinovasi dan beradaptasi yang akan menjadi pemenang di industri.
2. Development Cycle yang paling cocok di era saat ini adalah model Agile. Dan DevOps dengan Agile menganut prinsip-prinsip utama yang mirip.
3. Masalah utama yang dipecahkan oleh DevOps sebenarnya adalah lambatnya delivery aplikasi akibat ketidak harmonisan tim Developer dengan tim Operation.
4. DevOps mengkolaborasikan tim Developer dengan tim Operation agar tercipta proses development dan delivery aplikasi yang lebih cepat dan efisien.
5. Yang paling utama adalah Kultur DevOpsnya itu sendiri dibanding tools dan teknologinya.
6. Prinsip kerja DevOps adalah Continuous Everything. Mulai dari Continuous Integration, Continuous Testing, hingga Continuous Deployment dan Continuous Delivery, demi mencapai Continuous Value untuk pengguna.
7. Skill dan Knowledge dasar yang harus dimiliki untuk menjadi seorang DevOps adalah : Linux, Docker, Cloud Computing, Network & Security, dan Softskill komunikasi serta kolaborasi.