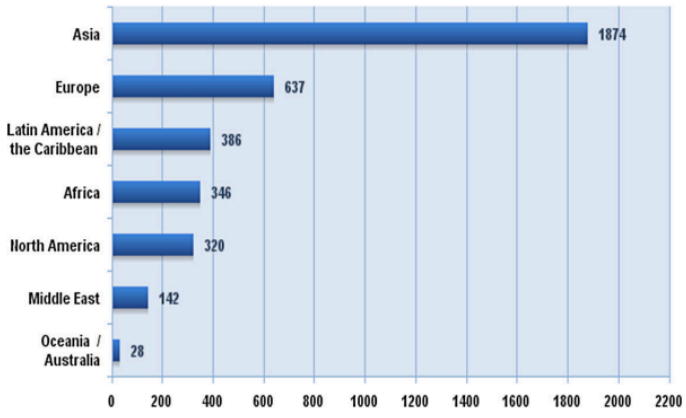
BAB 2
LANDASAN TEORI

2.1 Tinjauan Teori

2.1.1 Cloud Computing

Secara konsep menurut [4], makna komputasi dapat diterjemahkan menjadi banyak bagian seperti Gas, Air, Listrik yang dapat diakses oleh jutaan orang di dunia namun *computing* yang dimaksud disini adalah sumberdaya perangkat lunak, penyimpanan, *network* dan komunikasi. Kemunculan *cloud computing* menjadi salah satu lompatan besar di abad ke 20 dimana setiap orang dapat menyimpan, mengakses semua sumberdaya di *internet* tanpa menyimpannya di dalam memori perangkat mereka sendiri. *Internet* memungkinkan diadakannya *cloud computing*, evolusi dari *cloud computing* sering dikaitkan dengan masa depan dari *internet* itu sendiri.

Pengguna *internet* meningkat sangat pesat mulai dari tahun 1999 hingga 2013, 1 juta pertama pengguna *internet* meningkat pada tahun 2015, kemudian meningkat menuju angka 2 juta di tahun berikutnya, bahkan meledak lebih banyak hingga saat ini. Gambar 1.1 menunjukkan *chart statistic* pengguna *internet* di tahun 2017 bulan Maret.



Gambar 2.1 Pengguna *internet* di bulan Maret 2017 [4]

Cloud Computing diambil dari beberapa inovasi dunia *IT* (*Information Technology*) seperti virtualisasi, peningkatan kapasitas *internet* dan teknologi yang

berkembang di dunia *internet*. *The National Institute of Standards and Technology (NIST)* [4] menjelaskan terdapat 5 karakteristik dari *cloud computing* :

A. *On-demand self-service*, yang berarti tidak ada interaksi antar manusia dengan penyedia servis seperti *server time*, *network storage* dan semua servis *cloud* yang ada.

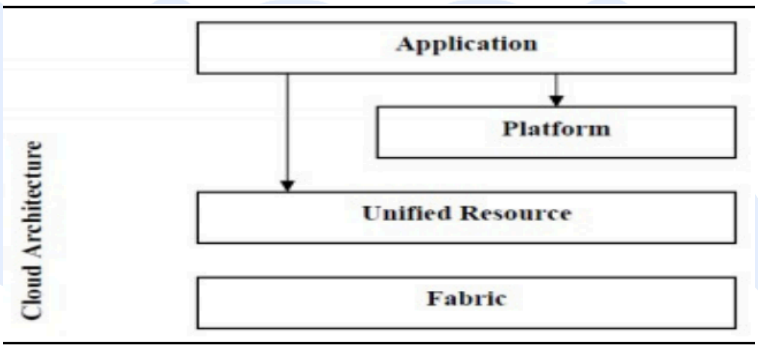
B. Cakupan jaringan yang luas yang berarti sumber daya yang dapat diakses darimanapun di seluruh dunia yang sudah dijangkau dengan adanya internet dan dapat diakses dengan perangkat yang ada seperti *laptop*, *handphone* milik pribadi maupun *workstation*.

C. *Resource pooling*, dimana sumber daya yang dapat di tetapkan dan ditugaskan secara dinamis tergantung kebutuhan.

D. *Rapid elasticity*, dapat disebut sebagai *scalable* yang berarti servis dapat meningkat maupun turun sesuai dengan kebutuhan.

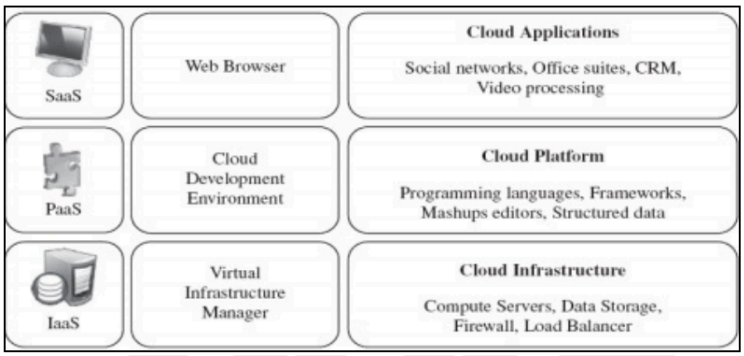
E. *Measures service*, kemampuan untuk mengoptimasi kegunaan sumber daya seperti ruang penyimpanan, *bandwith* serta akun *user* yang aktif.

Pada jurnal yang berjudul *Benefits And Challenges of The Adoption of Cloud Computing in Business* [5], dipaparkan arsitektur dari *Cloud Computing* sebagai berikut,



Gambar 2.2 Arsitektur *Cloud Computing*

Cloud dalam terjemahan berarti awan yang dapat diartikan sebagai *remote environments* dimana setiap developer dapat mengakses barbagai *resource* darimana pun selama terjangkau oleh koneksi *internet*, dalam *cloud computing* terdapat 4 lapisan dasar, lapisan pertama adalah *fabric* yang terdiri dari komponen fisik seperti komponen jaringan, unit komputasi dan sistem penyimpanan. Lapisan kedua yaitu *unified resource* yang merupakan *hardware* yang sudah di virtualisasi seperti server yang tidak lagi *on-premise* melainkan sudah terletak di dalam *cloud* itu sendiri. Lapisan berikutnya merupakan *platform* yang merupakan *layer* yang berisi *tools* yang berguna untuk mengurangi beban untuk menaik kan aplikasi secara langsung ke dalam *container virtual machines*, yang terakhir adalah *application* dimana merupakan *interface* dari *cloud* itu sendiri yang digunakan untuk proses operasional dari *user*, model servis yang terdapat di dalam dunia *cloud computing* yang terbagi menjadi 3 yaitu *IAAS*, *PAAS* dan *SAAS*, gambar 1.3 menunjukkan *service models* dari *Cloud Computing*.



Gambar 2.3 *Service Models of Cloud Computing*

A. *IAAS (Infrastructure as a Service)*

IAAS merupakan layer paling bawah di dalam struktur servis *model*, konsep dasar dari *IAAS* atau *Infrastructure as a Service* adalah sebagai infrastruktur dari aplikasi untuk berjalan *IAAS* menyediakan beberapa *resource* seperti penyimpanan, *processing unit*, jaringan, dsb. *IAAS* mempermudah *user* untuk melakukan *deployment* dan perancangan aplikasi tanpa memikirkan infrastrukur yang rumit dibelakangnya, dengan adanya *IAAS user* dapat mengatur *flow* dari *Firewall* maupun *Load Balancer* dengan mudah di *user interface* yang disediakan oleh *cloud*

provider seperti *Google Cloud* yang menyediakan *load balancer*, *Firewall*, dsb. [5]

B. ***PAAS (Platform as a Service)***

PAAS atau *Platform as a Service* merupakan *layer* yang sangat penting dan paling berguna untuk kelangsungan operasional, *PAAS* menyediakan *platform* yang langsung dapat digunakan oleh *user* untuk menjaga jalannya aplikasi dan proses implementasi dari aplikasi tersebut, *platform* yang disediakan oleh *cloud provider* antara lain seperti *database (Google Storage Bucket*), *operating systems* seperti *CentOS*, *Debian*, dan berbagai *OS Linu*x lainnya, selain itu juga *server* seperti *Compute Engine* yang disediakan oleh *GCP*. [5]

C. ***SAAS (Service as a Service)***

SAAS (Service as a Service) merupakan lapisan teratas, sebelum adanya teknologi *cloud* setiap perusahaan masih menggunakan *computing* yang bersifat *on premise* dimana perusahaan harus membeli atau menyewa *super computer* atau *server* dengan biaya *maintenance* yang sangat tinggi, *user* harus melakukan pemasangan kabel, pendingin ruangan, serta mengatur semua alur jaringan, sementara dengan *cloud computing* setiap hal tersebut sudah disediakan dari *cloud provider* sebagai servis yang dijual dari perusahaan mereka sehingga setiap perusahaan sekarang yang menggunakan teknologi *cloud* tidak lagi harus mengalokasikan banyak biaya untuk melakukan semua hal tersebut. [5]

2.1.2 DevOps

DevOps merupakan sebuah *role* atau peran di dalam dunia *IT* yang terbagi dari *Developer* dan *Operation*, secara harafiah tidak ada penjelasan yang spesifik mengenai *DevOps* namun Eoin Woods Endava di jurnalnya yang berjudul *"The Pragmatic Architect"* [6], bahwa *DevOps* adalah orang yang berperan untuk mengurangi waktu dalam *commit* atau menetapkan perubahan ke dalam sistem dengan kualitas yang sebaik-baiknya.

Sebelum naiknya *DevOps* ke permukaan sering terjadi konflik antar tim *Development* dan *Operations* yang dipaparkan oleh antara lain :

A. *Development* menyerahkan versi aplikasi yang baru kepada tim *operation* untuk dilakukan *testing* namun aplikasi tidak dapat berjalan di *production environments*,
B. Tim *operation* kemudian melaporkan kesalahan dari aplikasi ini ke tim *development* namun kemudian tidak dapat balasan dari tim *developer* karena komunikasi yang kurang baik,
C. Selanjutnya tim *development* pun membela diri dan berkata bahwa tidak dapat masalah apapun di aplikasi karena aplikasi dapat berjalan di *development environments*, kemudian konflik pun terjadi dimana 2 tim ini saling tunjuk untuk saling menyalahkan,
D. Setelah masalah dibawa ke meja Manejer dan Bos, mereka pun hanya dapat memerintahkan untuk meng-ekskalasi *problem* ini secepatnya,
E. Setelah di telaah lebih lanjut ternyata hanya terdapat masalah *minor* diantaranya seperti versi *database* yang berbeda diantara 2 *environments*, masalah teknis karena *cache* yang belum dibuang,
F. Kemudian aplikasi pun naik secara baik namun dengan waktu yang sangat panjang dan harus melewati begitu banyak konflik terlebih dahulu.

2.1.3 Continuous Integration

Dibalik semua masalah yang dipaparkan di sub-bab 2.1, *Continuous Integration* hadir untuk mengimplementasikan fungsi dari peran *DevOps* dimana *CI* merupakan metode yang berperan untuk secara berkesinambungan mengintegrasikan kode yang dibuat, ketika *developer* melakukan *push/commit* ke salah satu *repository* di *branch* tertentu maka *project* akan secara otomatis melakukan *build* dan memastikan bahwa kode berjalan dengan baik, Gambar 2.2 merupakan ilustrasi *CI/CD lifecycle*.

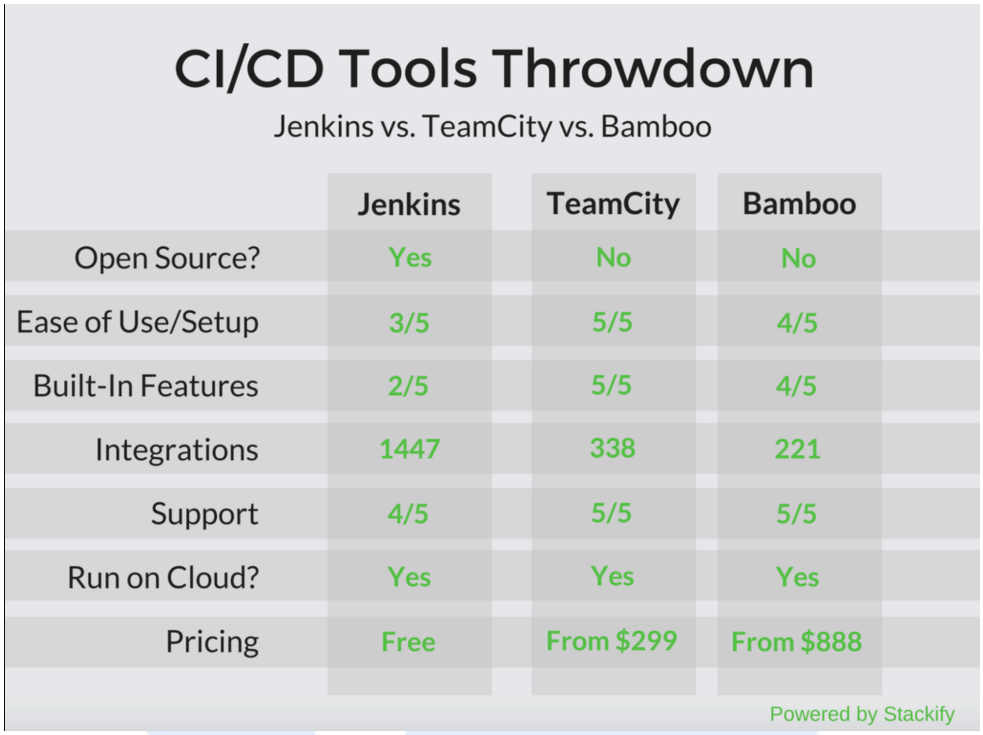
Gambar 2.4 *CI/CD Lifecyle* [7]

tertera beberapa persyaratan yang akan diterapkan untuk memastikan berjalannya *CI* [7], antara lain :

A. Menggunakan *version repository* untuk *source code* yang di *push* oleh *developer* seperti *Github, Gitlab, Bitbucket*, dkk,
B. *Testing* otomatis pada perubahan kode,
C. Otomasi proses *build*,
D. *Deploy* atau me-rilis kode ke dalam lingkup *pre-production* seperti *staging* dan *development*.

2.1.4 Jenkins

Jenkins, merupakan satu dari begitu banyak *tools* yang ada di dunia internet untuk otomasi proses *build* dan *deployment* namun banyak perusahaan besar yang bergerak di dunia *IT (Information Technology)* memilih *Jenkins* untuk proses *CI/CD* mereka karena *tools* ini tergolong aman dan bersifat *open-source* jadi tidak sepeser-pun keluar dari kantong perusahaan untuk menggunakan *tools* ini, lalu *Jenkins* merupakan salah satu *tools* yang tergolong lengkap di *integration* dan dapat *running* di *cloud provider* manapun, walaupun tergolong *free* alias *open source*, *Jenkins* memastikan bahwa *tools* ini sangat lengkap. Gambar 2.3 merupakan perbandingan *tools CI/CD*,



Gambar 2.5 Perbandingan beberapa *tools CI/CD [8]*

Secara harafiah dituliskan apa pengertian dari *Jenkins* itu sendiri dan apa kegunaan dari *tools* ini [8], *Jenkins* merupakan salah satu *tools* yang digunakan untuk proses implementasi *Continuous Integration* dengan *pipelines* atau alur kerja, *Jenkins* berkerja dengan *Jenkinsfile* yang kemudian akan di deteksi oleh mesin untuk menjalankan proses *build* dan *deployments*. Kelebihan menggunakan pipelines *Jenkins* antara lain :

A. *Code: pipelines* berjalan menggunakan kode yang sudah ditulis dan di cek di *source control* sehingga memberikan keluasaan kepada setiap tim di dalam organisasi untuk me-*review*, edit dan meng-iterasi *pipeline*.
B. *Durable:* dimana *pipelines* dapat melewati tahap yang tidak di rencanakan seperti *restart* yang direncanakan maupun tidak dari *controller Jenkins*.
C. *Pausable: pipeline* dapat di berhentikan sementara untuk menunggu *input* dari *developer* dan membutuhkan *approval* untuk melanjutkan kerja dari *pipelines* itu sendiri.

D. *Versatile: pipeline* dari *Jenkins* juga mendukung alur kerja yang cukup kompleks seperti adanya *looping*, *fork/join* dan kondisi *IF/ELSE*.
E. *Extensible: Jenkins* memiliki banyak sekali fitur yang mendukung integrasi dengan *plugin* lain.

2.1.5 Docker

Continuous Integration tidak jauh dengan *Docker*, 2 *tools* ini saling berhubungan untuk melakukan performa otomasi *proses build* dan *deployments*, *Docker* merupakan salah satu *tools* yang berfungsi sebagai *containerization tools* yang mulai dipublikasikan sejak 2013, dengan *Docker* maka setiap *Developer* dapat mengisolasi sebuah aplikasi dari *host* atau *virtual machines* yang ada sehingga di dalam satu *server* dapat menampung banyak aplikasi yang di kontainerisasi tanpa mengganggu sistem *host* itu sendiri. Relevansi *Docker* dengan *VM* adalah *Docker* akan menggunakan sumber daya dari *VM* itu sendiri seperti *CPU, RAM* dan *Memory*.

Alasan menggunakan *Docker container* sangat sederhana, konsep dari *container* adalah fleksibilitas dan beberapa alasan lain, yaitu:

A. Menciptakan *environments* untuk para *developer* seperti *production*, *staging* maupun *development* secara terisolasi.
B. Menciptakan *building blocks* untuk *service oriented* atau *microservice*, dengan adanya *Docker* dapat mempermudah untuk meng-isolasi aplikasi dan menggunakan *Docker* untuk menciptakan beberapa *microservice* dengan *container*.
C. Menciptakan sistem *CI/CD* dengan mudah.
D. Menciptakan *container* yang berdiri sendiri untuk melakukan *testing* dan *research* oleh para *developers*. [9]

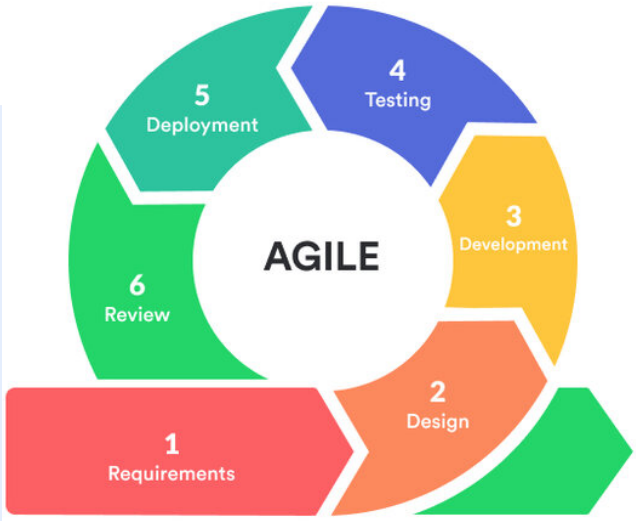
Kemudian juga dipaparkan beberapa komponen agar *Docker* dapat berjalan di dalam sebuah server, antara lain:

A. *Docker engine*, yang berguna sebagai *client* dari *Docker* atau dapat disebut sebagai *Docker daemon* yang memiliki tugas untuk mengeksekusi *container*.
B. *Images*, sebelum *Docker* dapat berjalan maka dibutuhkan *images* yang berperan sebagai *installer* dimana di dalam nya terkandung berbagai *dependencies*. *Docker images* dapat di *build* sendiri maupun diakses melalui *Dockerhub*.
C. *Registry*, berperan sebagai *storage* yang terbagi menjadi 2 yaitu *public* yang dapat diakses melalui Dockerhub maupun *private* melalui *images* yang dirancang dan *build* secara pribadi.
D. *Container*, setelah *images* berhasil ter-eksekusi maka akan terbentuk sebuah *container*, mudah dijelaskan *container* merupakan *images* yang berjalan.

2.1.6 Agile

Agile pada umumnya dikenal sebagai suatu cara pendekatan yang lebih *flexible* pada proses pengembangan dan manajemen suatu aplikasi, pendekatan *Agile* dikarakteristikan menjadi suatu pendekatan yang *iterative* dalam pengembangan suatu produk [10]. Dalam perkembangan dunia *digital* yang begitu pesat terkadang proses pengembangan aplikasi dibutuhkan dengan cepat salah

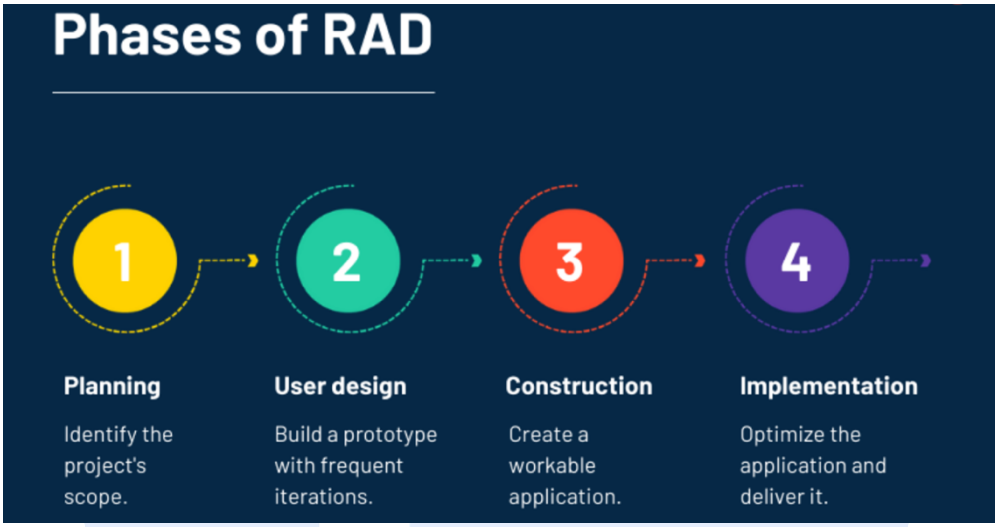
satunya adalah dengan implementasi metode *agile* untuk mempersingkat waktu *development*. Gambar 2.5 merupakan ilustrasi dari metode pengembangan *agile*,



Gambar 2.6 Metode pengembangan *Agile*

2.1.7 Rapid Application Development

RAD merupakan sebuah metode dan kerangka pikir untuk menciptakan atau men-*develop* sebuah aplikasi yang memakan waktu lebih cepat, *Rapid* sendiri diartikan sebagai cepat atau memakan waktu yang sedikit. RAD atau *Rapid Application Development* memiliki 4 fase yaitu *Requirements Planning*, *User Design*, *Construction* dan *Cutover Phase* (UAT), [11] gambar 2.4 merupakan visualisasi daripada setiap fase RAD (*Rapid Application Development*),



Gambar 2.7 *Rapid Application Development Phase* [11]

RAD atau Rapid Application Development mulai dikenalkan pada public sejak tahun 1970, RAD merupakan salah satu solusi dari dunia *IT* yang semakin berkembang dan banyak aplikasi yang harus di *develop* dalam waktu yang singkat, RAD serupa dengan metode *Agile* atau sebaliknya, pada metode RAD terdapat salah satu fase yang dinamakan *requirements planning* dimana tahap ini adalah tahap untuk proses *maintain* dokumentasi pada system yang diterapkan agar *developer* selanjutnya dapat merujuk pada dokumentasi yang sudah *exist* atau sudah dituliskan, komunikasi harus dijaga dengan baik pada proses *requirements planning* dan harus dilakukan dengan se-efektif mungkin. Tabel 2.1 merupakan perbedaan dari metode *Agile* dan RAD,

Tabel 2.1 Perbandingan *Agile* Vs. RAD [11]

	Agile	**RAD**
Prototype	No *Prototype*	Prototype lalu development
Working Style	*Group*	*Group* berdasarkan *skill* yang dimiliki
Testing	*Testing* dilakukan ketika *project* sudah selesai dirancang	*Testing* dilakukan pada *prototype*
Fokus	Efisiensi	Kecepatan *development*
***Suitable* project size**	*Large*	*Small*
Dokumentasi	*Less prioritized*	*Prioritized*

2.2 Penelitian Terdahulu

Penelitian terdahulu yang terlampir di tabel 2.2 menjadi salah satu acuan dalam melakukan penelitian serta dapat menjadi acuan untuk memperkaya teori serta pendukung ter-realisasinya penelitian ini.

Tabel 2.2 Penelitian Terdahulu

Jurnal 1 [12]	
Nama Jurnal	*Big Data, Cloud Computing, and Data Science Engineering, Studies in Computational Intelligence 844* (*4th* ed). (2019)
Judul Penelitian	*IoTDoc: A Docker-Container Based Architecture of IoT-Enabled Cloud System*
Peneliti	*Shahid Noor, Bridget Koehler, Abby Steenson, Jesus Caballero, David Ellenberger and Lucas Heilman.*
Latar Belakang	Penggunaaan cloud computing tradisional memiliki beberapa kekurangan dari segi biaya maintenance yang sangat mahal khususnya di platform cloud dan cukup menyulitkan bagi user untuk menginstall virtualisasi.
Metode Penelitian	Histori
Hasil Penelitian	Waktu *deployment* tanpa menggunakan *container* relatif lebih memakan waktu dibandingkan *deployment* menggunakan *container*, salah satu kendala yang dihadapi adalah proses *install* dan implementasi *platform* virtualisasi.
Jurnal 2 [13]	
Nama Jurnal	*Jurnal Ilmiah Teknologi Informasi Terapan vol. 8, no. 1,* (2021)
Judul Penelitian	Implementasi *Contionous Integration* dan *Continous Deployment* pada Aplikasi *Learning Management System* di PT. Millennia Solusi Informatika.
Peneliti	*Indra Guna Noviantama, Ari Purno Wahyu.*

Latar Belakang	Dikarenakan pengembangan aplikasi di PT. Millennia Solusi Informatika dilakukan secara terus menerus dan bersifat cepat, maka diperlukan suatu konsep yang mendukung setiap perubahan dapat terimplementasi dengan cepat, oleh karena itu dibutuhkan system CI atau continuous integration menggunakan tools Jenkins.
Metode Penelitian	Agile
Hasil Penelitian	Penerapan CI/CD mempermudah tim pengembang dan operasional untuk bekerja secara praktis karena semua proses ter-otomasi secara baik dengan sistem CI/CD.
Jurnal 3 [14]	
Nama Jurnal	Jurnal *Multimedia Networking Informatics, Vol. 6 No. 1 (2020)*
Judul Penelitian	Implementasi *DevOps* pada Pengembangan Aplikasi *e-Skrining Covid-19.*
Peneliti	*Tohirin, Sri Farida Utam, Septisan Rheno Widianto, Widhy Al Mauludyansyah*
Latar Belakang	SDLC (Software Development Life Cycle) memiliki beberapa fase yang penting di dalam pengembangan sebuah software, pada proses perancangan aplikasi skrining covid-19 dari penelitian terdahulu ke-3 terdapat masalah dimana proses build dari aplikasi memakan waktu yang sangat panjang, di lain sisi juga bila aplikasi ditemukan sebuah bug maka aplikasi akan diupdate dan ketika aplikasi sudah di update maka aplikasi akan memakan waktu untuk melakukan build ulang secara manual di dalam server, oleh karena itu dibutuhkan system CI menggunakan GitlabCI untuk mempercepat proses build secara otomatis.

Metode	Agile
Hasil Penelitian	Proses pengembangan aplikasi berjalan lancar dan mudah karena adanya *build* harian setiap kali terjadi *commit/push* dari *developer*.
Jurnal 4 [19]	
Nama Jurnal	*Ultima InfoSys : Jurnal Ilmu Sistem Informasi*, *9*(1). (2018)
Judul Penelitian	*Pregnancy Consultation Application Development based on Cloud Computing.*
Peneliti	Argo Wibowo.
Latar Belakang	Indonesia memiliki tingkat kehamilan yang sangat tinggi, menurut Direktur Kesehatan Keluarga Indonesia sekitar 400.000 ibu meninggal setiap bulannya diakbatkan oleh informasi tentang kesehatan janin yang kurang memadai, oleh karena itu muncul sebuah gagasan untuk menciptakan suatu aplikasi berbasis *Cloud Computing* untuk memberikan informasi yang relevan kepada calon ibu.
Metode	*Prototype*
Hasil Penelitian	Penerapan sistem *Cloud Computing* dengan mengandalkan *web server* dan juga *google firebase* sebagai *database* berhasil dikembangkan, dan aplikasi pun berhasil di kembangkan dengan *UI* yang semudah mungkin untuk mempermudah penggunaan dari aplikasi ini.

Tabel 2.2 merupakan penelitian terdahulu yang diambil dari jurnal *Big Data, Cloud Computing, and Data Science Engineering, Studies in Computational Intelligence 844,* dari penelitian terdahulu ini masalah dan hasil penelitian memiliki kesamaan dengan penilitian yang dilakukan khususnya pada *tools docker* yang digunakan, *Docker* digunakan sebagai *platform* virtualisasi untuk mengkontainerisasi *virtual machines* agar berjalan lebih efisien dari sisi biaya *maintenance virtual machines*, namun pada penelitian terdahulu ini peneliti tidak menggunakan *system* CI untuk mempercepat proses *development* dari aplikasi yang dikembangkan.

Dari penelitian terdahulu yang ke-2 (dua) terdapat kesamaan dengan penelitian yang dilakukan, salah satunya adalah latar belakang masalah dari penelitian terdahulu dilandasi dengan masalah banyaknya permintaan aplikasi di PT. Millennia Solusi Informatika sehingga diimplementasikan system CI (*Continuous Integration*) menggunakan *open source tools Jenkins*. Penelitian terdahulu ke-2 mengangkat metode *Agile* sedangkan penelitian yang dilakukan menggunakan metode RAD (*Rapid Application Development*). Alasan peneliti tidak menggunakan metode *Agile* adalah karena metode *Agile* membutuhkan tenaga *professional* untuk melakukan pengembangan system karena dokumentasi pada metode *Agile* tidak terlalu di prioritaskan.

Pada penelitian terdaulu ke-3, proses implementasi CI (*Continuous Integration*) dilandasi dengan latar belakang masalah yang sama dengan penelitian Rancang Bangun Sistem *Continuous Integration Pipeline* menggunakan *Jenkins* di PT. Emporia Digital Raya, dilandasi karena proses *development* yang cukup memakan waktu pada proses *build* dan *deployment* sehingga tercetus ide untuk meng-otomasi semua proses dari *build* hingga *deployment*. Penelitian terdahulu ke-3 tidak menggunakan *Jenkins* sebagai *system* CI melainkan langsung menggunakan *GitlabCI* yang sudah disediakan oleh *Gitlab*, kekurangan dari *GitlabCI* adalah tidak banyak *plugin* yang dapat ditambahkan kedalam proses *build* dan *deployment* serta penelitian terdahulu ke-3 menggunakan *shared runner* atau *worker* oleh *Gitlab* sehingga spesifikasi lebih rendah dibandingkan *worker independent Jenkins* yang dapat diatur

spesifikasinya sedemikian rupa sehingga proses *build* dapat berjalan lebih cepat dan maksimal.

Pada penelitian terdahulu ke-4, proses perancangan aplikasi dengan sistem *cloud* dikembangkan sedemikian rupa dengan *web server* dan menggunakan *database cloud* yaitu *Google firebase*. Rancang bangun pada penelitian ini juga menerapkan sistem *Cloud Computing* dengan server yang sudah tidak di *maintain* secara pribadi namun menggunakan *cloud provider Google cloud*.