PENGEMBANGAN DASHBOARD UNTUK MONITORING CELAH KEAMANAN PADA CLUSTER KUBERNETES MENGGUNAKAN METODE CONTAINER IMAGES SCANNING

Disusun Oleh:

N a m a : Muhammad Riko Bediatra
NIM : 19523201

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2023

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGEMBANGAN DASHBOARD UNTUK MONITORING CELAH KEAMANAN PADA CLUSTER KUBERNETES MENGGUNAKAN METODE CONTAINER IMAGES SCANNING

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

N a m a : Muhammad Riko Bediatra
NIM : 19523201

Yogyakarta, 4 Oktober 2023

Pembimbing,

Dr. Sri Kusumadewi, S. Si, M. T.

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGEMBANGAN DASHBOARD UNTUK MONITORING CELAH KEAMANAN PADA CLUSTER KUBERNETES MENGGUNAKAN METODE CONTAINER IMAGES SCANNING

TUGAS AKHIR JALUR MAGANG

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 2 November 2023

Tim Penguji

Dr. Sri Kusumadewi, S. Si, M. T. _____

Anggota 1

Arrie Kurniawardhani, S. Si., M. Kom. _____

Anggota 2

Ari Sujarwo, S. Kom., M.I.T. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Muhammad Riko Bediatra

NIM : 19523201

Tugas akhir dengan judul:

PENGEMBANGAN DASHBOARD UNTUK MONITORING CELAH KEAMANAN PADA CLUSTER KUBERNETES MENGGUNAKAN METODE CONTAINER IMAGES SCANNING

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 23 November 2023

Muhammad Riko Bediatra

HALAMAN PERSEMBAHAN

Alhamdulillah saya panjatkan kepada Allah SWT karena dengan segala nikmat dan kesempatan yang telah diberikan kepada saya untuk dapat menyelesaikan tugas akhir saya yang diberi judul “Pengembangan Dashboard untuk Monitoring Celah Keamanan Pada Cluster Kubernetes Menggunakan Metode Images Scanning” dengan segala kekurangannya. Laporan tugas akhir ini saya persembahkan untuk seluruh keluarga dan kerabat dekat saya yang telah mendukung serta membantu secara mental, finansial, dan doa selama saya menempuh pendidikan di jurusan Informatika FTI UII. Laporan ini juga saya persembahkan kepada diri saya sendiri yang telah berusaha dan bertahan dalam menjalani setiap jalan yang ditempuh hingga sampa pada tahap ini.

HALAMAN MOTO

“Sesungguhnya Allah tidak akan mengubah keadaan suatu kaum hingga mereka merubah keadaan yang ada pada diri mereka sendiri.”

– Q.S. Ar Rad: 11

“*It's always easy to blame others. You can spend your entire life blaming the world, but your successes or failures are entirely your own responsibility.*”

– Paulo Coelho

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh,

Puji dan syukur penulis haturkan kepada Allah SWT yang telah melimpahkan rahmat serta hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir penjaluran magang ini dengan lancar. Tidak lupa *shalawat* serta salam penulis junjungkan kepada Nabi Muhammad SWT yang telah membimbing dan menuntun umat muslim kepada jalan yang diridhai oleh Allah SWT.

Laporan ini disusun sebagai salah satu persyaratan dalam kelulusan di penjaluran Magang pada Program Studi Informatika – Program Sarjana dan menjadi bukti serta dokumentasi dari pelaksanaan magang. Adapun selama pelaksanaan magang, penulisan laporan tengah, sampai menyelesaikan tugas akhir penulis banyak dibantu oleh banyak pihak sehingga penulis merasa lebih ringan dalam pengerjaan tugas – tugas tersebut. Oleh karena itu, penulis ingin mengucapkan banyak terima kasih kepada:

1. Allah SWT yang telah memberikan kesehatan, rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir ini.
2. Orang tua dan keluarga yang senantiasa berdoa untuk kelancaran dan selalu memberikan semangat serta dukungan nya kepada penulis.
3. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Jurusan Informatika dan Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku Ketua Program Studi Informatika – Program Sarjana.
4. Ibu Dr. Sri Kusumadewi, S.Si., M. T. selaku dosen pembimbing yang senantiasa membantu serta memberikan arahan selama penempuh penjaluran akhir.
5. Bapak dan Ibu dosen Program Studi Informatika, yang telah memberikan ilmu yang luar biasa bermanfaat selama saya menempuh studi.
6. Seluruh jajaran pimpinan di Badan Sistem Informasi Universitas Islam Indonesia yang telah memberikan kesempatan dan pengalaman magang.
7. Bapak Pandhu Bangun Asmoro selaku kepala tim dan seluruh anggota Tim Interopeng yang telah membantu pelaksanaan magang.
8. Rekan - rekan seperjuangan, INFINITY, yang telah memberikan dukungan serta bantuan selama pelaksaan tugas akhir.
9. Teman baik saya Jasmine Erina Firdaus serta teman–teman dekat kuliah saya selalu memberikan dukungan serta menemani selama proses magang hingga tahap penulisan tugas akhir.

Penulis menyadari masih banyak kekurangan dalam pembuatan tugas akhir ini. Oleh karena itu penulis menerima segala bentuk saran guna menyempurnakan laporan ini. Semoga laporan ini dapat bermanfaat bagi pembaca

Wassalamu'alaikum Wr. Wb.

Yogyakarta, 07 November 2023

Muhammad Riko Bediatra

SARI

Penulis melakukan kegiatan magang sebagai *software engineer* di Badan Sistem Informasi Universitas Islam Indonesia atau sering disebut BSI UII. BSI UII merupakan badan sistem informasi yang diperuntukkan untuk mendukung proses bisnis di lingkungan Universitas Islam Indonesia. Layanan yang diberikan oleh BSI dapat dinikmati oleh seluruh civitas akademik, mulai dari mahasiswa, dosen, hingga staf yang ada di lingkungan UII. Kantor BSI UII bertempat di lantai 4 Gedung GBPH Prabuningrat Kampus Terpadu Universitas Islam Indonesia Yogyakarta.

Proyek pengembangan *dashboard* untuk *monitoring* celah keamanan pada *cluster* Kubernetes dipilih untuk dikaji lebih lanjut dalam laporan tugas akhir ini. Proyek ini didasari karena saat ini BSI UII belum menggunakan *scanning images* pada *runtime cluster* serta saat ini *scanning images* dilakukan hanya pada *cloud registry* yang dimana diperlukan biaya yang harus dikeluarkan setiap melakukan *scanning images*. Hasil akhir dari proyek ini adalah dapat mendeteksi celah keamanan yang berada di dalam *container images* yang telah di-*deploy* pada *cluster* Kubernetes. Selain itu terdapat juga *dashboard* yang berisi informasi mengenai detail celah keamanan pada suatu *cluster*. Detail ini berisi mengenai jumlah total celah keamanan, jumlah total celah keamanan berdasarkan tingkat *vulnerability* (*critical, high, medium, low),* nama *container images* yang memiliki celah keamanan, versi aplikasi yang di-*install*, saran pembaharuan versi aplikasi, dan *vulnerability id* yang didapatkan dari *database* perusahaan Aqua Security. Aplikasi ini juga bersifat *opensource* yang di mana dalam penggunaannya, *user* tidak akan dipungut biaya apabila melakukan *container image scanning.*

Dalam pelaksaan proyek ini masih terdapat banyak kekurangan serta batasan dari *container scanning images* yang dilakukan. Oleh karena itu untuk proyek selanjutnya diharapkan dapat melakukan implementasi dan membandingkan hasil *images scanning* untuk mengetahui tingkat akurasi dari laporan celah keamanan yang dihasilkan. Tidak hanya itu pelaksanaan proyek selanjutnya dapat juga melakukan *container scanning* yang tidak terbatas hanya pada *deployment.*

Kata kunci: *Microservice*s, Kubernetes, *Container Images Scanning, monitoring*

GLOSARIUM

Client	Client merupakan komputer yang melakukan permintaan (*request)* layanan tertentu dari komputer *server*.
Dashboard	Jenis antarmuka pengguna grafis yang memberikan informasi mengenai data yang relevan dengan tujuan atau proses bisnis tertentu.
DevOps	Sebuah prinsip *developer* untuk mengkoordinasikan antar tim yaitu tim *development* dengan tim *operations* dengan efektif dan efisien.
Metrics	Sebuah tipe data yang berisi gabungan sekumpulan data mentah. Secara operasional data *metrics* menjadi titik data yang dapat digunakan untuk memahami apa yang terjadi pada kumpulan data yang luas.
Monitoring	Proses rutin pengumpulan data dan pengukuran kemajuan atas objektif program, memantau perubahan yang fokus pada proses dan keluaran.
Scanning Image	Proses melalukan pemindaian artefak pada sebuah image dengan tujuan mendapatkan sebuah informasi tertentu.
Vulnerability	Kerentanan atau kelemahan dalam sistem komputer yang dapat melemahkan suatu perangkat atau sistem secara keseluruhan.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Penggunaan teknologi dalam dunia pendidikan mulai mengubah kegiatan maupun proses bisnis yang dilakukan. Kegiatan yang sebelumnya dilakukan secara manual mulai berubah menjadi sistem informasi yang terkomputerisasi. Perubahan ini terjadi dikarenakan penggunaan teknologi sebagai media pembelajaran memiliki banyak manfaat. Manfaat yang dirasakan dalam penerapan teknologi ini tidak hanya pada siswa yang menggunakan, akan tetapi para pelaku bisnis didunia pendidikan ikut juga merasakannya. Hal ini dapat dilihat dari banyaknya universitas yang menggunakan sistem informasi untuk mendukung proses bisnis yang dilakukan sehari hari.

Perkembangan yang sangat pesat terhadap teknologi informasi di dunia pendidikan khususnya kegiatan perkuliahan, mengakibatkan banyak universitas yang mulai melakukan digitalisasi pada kegiatan sehari–harinya. Universitas mulai membuat departemen atau badan sistem informasi untuk mewujudkan hal ini, termasuk Universitas Islam Indonesia. Universitas Islam Indonesia juga mengikuti perkembangan teknologi informasi dengan mendirikan sebuah badan yang bergerak dalam layanan sistem dan teknologi informasi pada lingkup universitas. Badan ini diberi nama dengan Badan Sistem Informasi Universitas Islam Indonesia atau yang disingkat BSI UII.

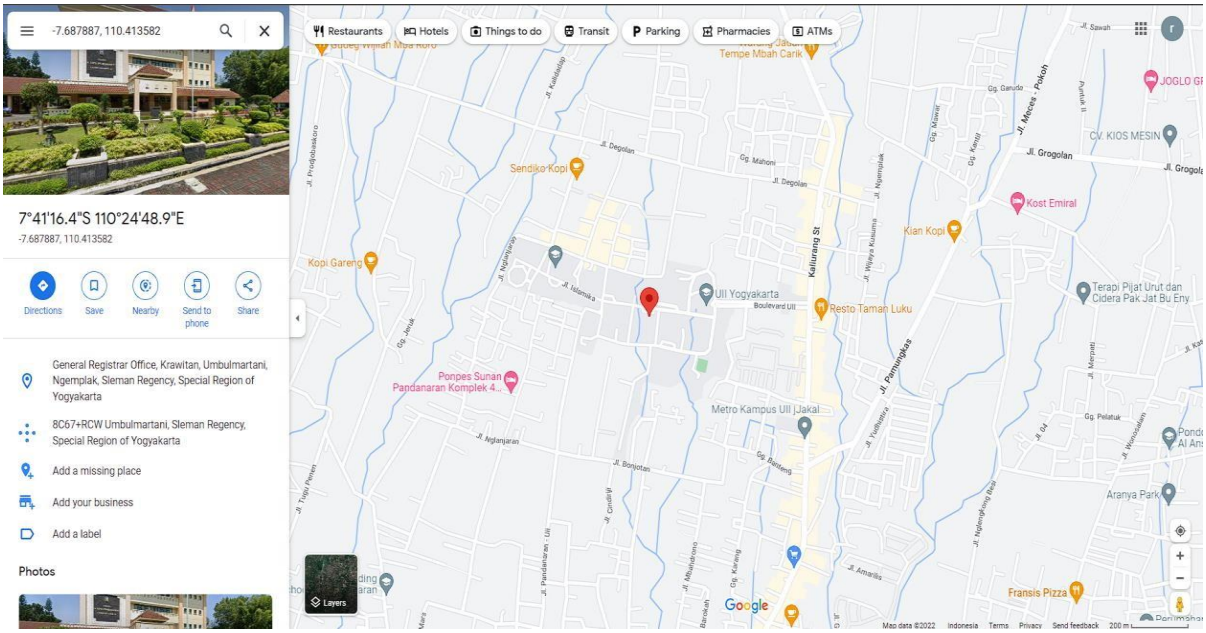
BSI secara umum bertugas mengawal perencanaan, pengembangan, operasi serta layanan sistem informasi di lingkungan UII dengan jumlah pengguna yang mencapai kurang lebih 30.000 pengguna yang terdiri dari dosen, tenaga pendidik, dan para pemangku kepentingan lainnya (*Sekilas BSI - BSI UII*, n.d.). Dalam menjalankan tugasnya BSI merumuskan tiga peran yaitu melayani, mendampingi, dan mengakselerasi. Ketiga peran ini dimainkan oleh BSI dengan mempertimbangkan konteks aktivitas atau masalah yang terjadi dilingkup universitas. Oleh karena itu untuk memenuhi standar dan peran yang dilakukan, BSI didukung oleh sumber daya manusia profesional yang terdiri dari *programmer*, analis dan konsultan dibidang sistem teknologi dan informasi.

Peran pertama BSI adalah sebagai pelayan bagi para pemangku kepentingan terkait dengan layanan yang selama ini sudah ada. Pelayan yang dimaksud dalam hal ini adalah menyediakan beragam sistem informasi pendukung bagi proses bisnis yang telah berjalan.

Selanjutnya BSI berperan dalam mendampingi para pemangku kepentingan dalam mengembangkan layanan yang sebelumnya tidak ada. Terakhir BSI berperan sebagai katalis perubahan di setiap jajaran yang ada di universitas, mulai dari tingkatan program studi sampai universitas itu sendiri. Dalam konteks ini BSI berperan dalam menimbulkan kesadaran terkait potensi sistem informasi yang sedang berkembang di pasaran ataupun pengembangan sistem informasi yang sudah ada demi meningkatkan proses bisnis di Universitas Islam Indonesia.

BSI berlokasi pada Jl. Kaliurang km. 14,5 Sleman, Yogyakarta. Tepatnya pada gedung GBPH Prabuningrat Kampus Terpadu Universitas Islam Indonesia. Ruangan kantor BSI sendiri terletak pada lantai 4 dan menyatu dengan gedung rektorat. Apabila dilihat dari depan maka tampak depan bangunan akan seperti gambar Gambar 1.1. Bila dilihat menggunakan Google Maps maka akan terlihat seperti Gambar 1.2.

Gambar 1.1 Tampak depan Gedung GBPH Prabuningrat Kampus Terpadu UII



Gambar 1.2 Lokasi GBPH Prabuningrat Kampus Terpadu UII dari GMaps

Penulis melaksanakan magang di Badan Sistem Informasi selama delapan bulan terhitung dari bulan September 2022 dan berakhir pada bulan Mei 2023 sebagai Software Engineering. Penulis ditempatkan pada tim interopeng. Tim interopeng sendiri bergerak pada bidang *development* dan *operation* (DevOps). DevOps (*development and operations*) adalah studi konseptual pengembangan dan pengiriman perangkat lunak terhadap infrastruktur dengan mengambil pendekatan kolaboratif dan integratif antara pengembang (Dev) dan operasional perangkat lunak (Ops) (Jha & Khan, 2018) Gambaran proyek – proyek yang dikerjakan seperti melakukan instalasi server, melakukan migrasi Rancher server, membuat *url shortener* dan mengerjakan tugas pada *backlog.* Setiap proyek dikerjakan dijalankan dengan menggunakan metode pengembangan *scrum*. Metode ini digunakan dikarenakan metode *scrum* merupakan metode pengembangan yang cepat dan adaptif (Grebić & Stojanović, 2021). Semua proses pengembangan dalam *scrum* dilakukan beberapa *sprint*, waktu *sprint* beragam menyesuaikan dengan *task* yang dikerjakan anggota tim yang lain. Waktu yang ditentukan oleh ketua tim interopeng berkisar satu hingga dua minggu.

Salah satu proyek yang dikerjakan saat di BSI UII adalah implementasi *container images scanning* pada *cluster* yang sedang berjalan. Alasan proyek ini dilaksanakan karena pada saat ini ketika sebuah aplikasi ataupun servis dibuat menjadi sebuah *images*. *Images* tersebut akan disimpan pada Google Cloud Registry (GCR). Oleh karena itu, *scanning images* baru akan dilakukan setelah aplikasi atau servis tersebut berhasil tersimpan. GCR dalam melakukan

scanning images mengharuskan pengguna untuk membayar setiap melakukan *scanning*. Tidak hanya itu GCR hanya melakukan *scanning* apabila terdapat *images* yang baru di-*push* melalui Gitlab. Apabila sebuah *images* telah berada pada sebuah *cluster*, *images* tersebut tidak akan di-*scanning* karena sudah pernah dilakukan *scanning* pada saat *deployment.* Padahal, seiring berjalannya waktu lama kelamaan *images* yang telah lama digunakan dan tidak menggunakan versi terbaru dapat memiliki celah keamanan yang dapat disusupi oleh para *hacker.*

Berdasarkan permasalahan pada paragraf diatas, proyek pengembangan *container images scanning* untuk *monitoring* celah keamanan diangkat menjadi topik skripsi karena menurut penulis proyek ini sangat unik. Selain itu, proyek ini sekaligus menjadi salah satu proyek yang di dalamnya terdapat kontribusi penuh dari penulis. Proyek ini dilakukan dengan memanfaatkan aplikasi Trivy Operator sebagai *images scanning*. Aplikasi ini dipilih karena dapat diimplementasikan pada *cluster* yang sedang berjalan. Tidak hanya itu, aplikasi ini bersifat *opensource* sehingga dalam proses implementasi dan *scanning images* pengguna tidak akan dipungut biaya apapun. Hal menarik lainnya bagi penulis dalam menerapkan Trivy Operator adalah bahwa hasil *scanning* yang dilakukan dapat diolah menjadi sebuah *dashboard* untuk melakukan *monitoring* celah keamanan.

1.2 Ruang Lingkup

Pelaksanaan pengembangan *dahsboard* untuk *monitoring* celah keamanan dilaksanakan selama kurun waktu tiga bulan dengan periode November 2022 sampai dengan Januari 2023. Dalam proyek ini terdapat dua orang termasuk penulis yang berkontribusi. Kontribusi penulis pada proyek ini adalah sebagai berikut:

a. Melakukan instalasi Trivy Operator pada *local cluster kubernetes* untuk melakukan *container images* scanning.
b. Melakukan pengujian *container images scanning* menggunakan *images* aplikasi yang berasal dari Dockerhub.
c. Melakukan implementasi pada Trivy Operator agar *vulnerability report* yang dihasilkan dapat dibaca oleh Prometheus.
d. Mengelola *vulnerability report* yang berhasil dibaca Prometheus menjadi sebuah dashboard keamanan.

Batasan dari pengembangan aplikasi adalah *scanning images* yang dilakukan hanya mencakup satu *cluster kubernetes* bukan *multiple cluster kubernetes.*

1.3 Tujuan

Laporan ini bertujuan untuk memberikan gambaran secara detail terkait deteksi celah keamanan yang terdapat pada suatu *cluster kubernetes* menggunakan metode *container images scanning.* Selain itu juga dapat menjadi alternatif untuk menguatkan sistem keamanan suatu cluster kubernetes.

1.4 Manfaat

Manfaat dari pengembangan *dashboard* untuk *monitoring* celah keamanan menggunakan metode *images scanning* adalah sebagai berikut:

a. Memudahkan mendeteksi *images* apa saja yang memiliki celah keamanan pada suatu *cluster kubernetes*.
b. Memudahkan dalam melakukan *monitoring* celah keamanan apabila terdapat celah keamanan yang baru.
c. Memberikan informasi detail terkait celah keamanan disertai total jumlah celah keamanan, grafik garis total celah keamanan berdasarkan *namespace*, nama *images,* tingkat bahaya suatu celah keamanan, *resource*, versi yang di-*install*.
d. Memberikan rekomendasi versi aplikasi yang sudah terdapat pembaharuan celah keamanan.
e. Mengetahui dampak apa saja yang dapat terjadi apabila celah keamanan tidak diatasi berdasarkan basis data yang dimiliki oleh Aqua Security.
f. Memahami penggunaan Trivy Operator sebagai *tools* untuk melakukan *images scanning*

1.5 Sistematika Penulisan

Sistematika penulisan ini menjelaskan secara keseluruhan dari setiap materi yang terdapat dalam laporan ini. Sistematika penulisan terdiri dari lima bab yaitu:

a. Pendahuluan

   Bab ini menjelaskan dan menggambarkan latar belakang, ruang lingkup magang, tujuan, manfaat serta sistematika penulisan.

b. Landasan Teori dan Tinjauan Pustaka

   Bab ini membahas tentang teori – teori yang berhubungan dengan pengembangan aplikasi sebagai landasan pengembangan *dashboard* untuk *monitoring* celah pada *cluster kubernetes* keamanan menggunakan metode *images scanning*.

c. Pelaksanaan Magang

Bab ini menggambarkan kegiatan yang berisi implementasi selama proses kegiatan magang.

d. Refleksi Pelaksanaan Magang

Bab ini berupa penjelasan refleksi dan hasil yang diperoleh selama pelaksanaan magang pada proyek pengembangan *dashboard* untuk *monitoring* celah keamanan pada *cluster kubernetes* menggunakan metode *images scanning.*

e. Penutup

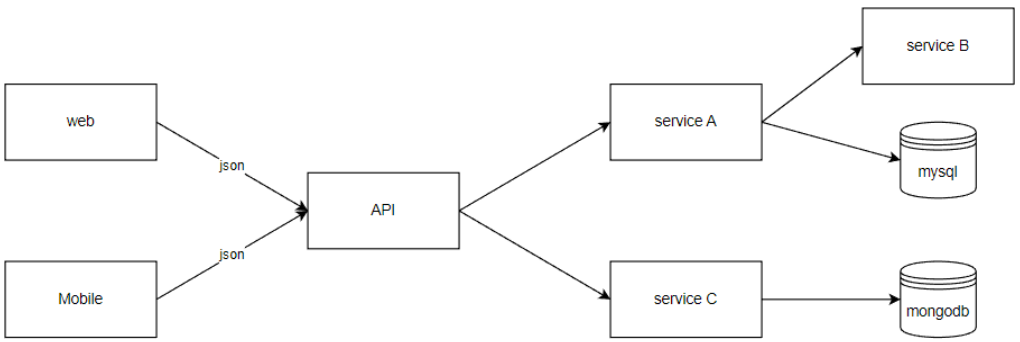
Bab ini berisi kesimpulan dan saran dari seluruh materi yang telah disajikan pada bab sebelumnya.

BAB II
LANDASAN TEORI DAN TINJAUAN PUSTAKA

2.1 Microservice

Microservice adalah kumpulan proses independen dan kecil yang berkomunikasi antara satu dengan yang lainnya untuk membentuk aplikasi kompleks yang agnostik terhadap bahasa API apa pun (Putra, 2019). Apabila disederhanakan, *microservice* membagi sebuah aplikasi menjadi beberapa layanan (*service)* yang lebih kecil dan dapat saling terhubung antar satu sama lain menggunakan API. Servis–servis ini terdiri dari block–block kecil yang terpisah dan fokus terhadap tugasnya masing–masing. Hal inilah yang menjadi salah satu perbedaan antara aplikasi *microservice* dan aplikasi yang bersifat monolitik.

Tujuan utama *microservices* adalah memecah dan merangkai aplikasi menjadi layanan-layanan kecil. Setiap layanan memiliki tanggung jawab masing-masing, tetapi tetap saling berkomunikasi. Sistem pada teknologi *microservices* juga didistribusikan secara independen yang hasilnya setiap layanan dapat beradaptasi dengan setiap perubahan kebutuhan (Lewis & Fowler, 2014). Salah satu contoh pengaruh dari penerapan *microservice* adalah bagaimana hubungan antara aplikasi dan *database*. Seperti pada Gambar 2.1, alih – alih berbagi skema *database* yang sama dengan *service* lain, masing–masing *services* dapat memiliki skema *database* nya sendiri. Dengan memiliki skema *database* sendiri dapat mengurangi adanya kemungkinan duplikasi data pada aplikasi. Tidak hanya terbatas pada penggunaan *database,* penerapan *microservice* juga memungkinkan setiap *service* menggunakan bahasa pemrograman yang berbeda beda.



Gambar 2.1 Contoh arsitektur *microservices* sederhana

Dalam setiap usaha penerapan teknologi tentunya tidak terlepas kan dari kelebihan dan kelemahan pada saat teknologi itu digunakan. Beberapa kelebihan pada penerapan *microservice* dipaparkan oleh Newman pada buku yang berjudul Building Micorservices (Newman, 2021). Kelebihan tersebut diantaranya:

a. Dengan sistem yang dibangun dalam bentuk *services* dapat memungkinkan *developer* menentukan sendiri setiap teknologi yang akan digunakan pada setiap *services*. Hal ini yang membuat teknologi yang digunakan pada arsitektur *microservices* lebih *heterogen* dibandingkan dengan menggunakan arsitektur monolitik.
b. Arsitektur berbasis *microservices* menjadi lebih tangguh terhadap kerusakan yang terjadi. Tidak hanya itu, apabila terjadi kerusakan dapat mudah dideteksi dan hanya terlokalisasi pada *service* tertentu.
c. *Scalling* dapat dengan lebih mudah dilakukan, karena hanya melakukan *deploy* pada aplikasi tertentu. Sehingga ketika *server* melebihi kapasitas, hanya beberapa *service* yang dilakukan *deploy* ulang, tidak keseluruhan aplikasi.

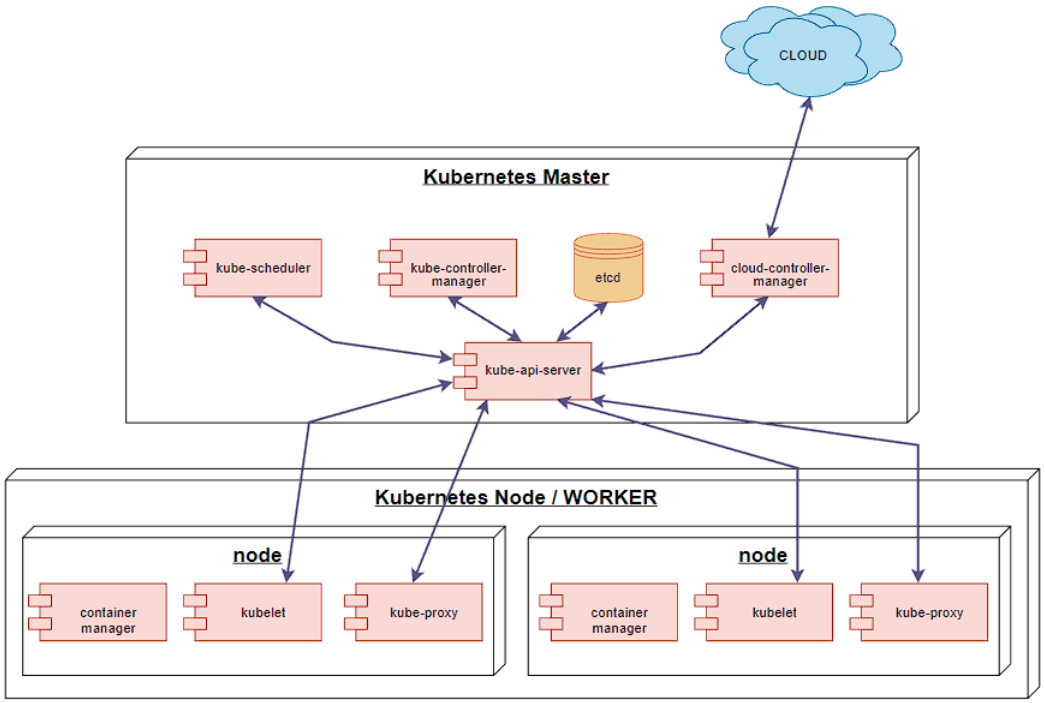
Sedangkan kekurangan dari penggunaan *microservices* adalah diperlukannya usaha lebih dalam membuat koneksi antar layanan dan mengatasi kegagalan di sebagian layanan. Tidak hanya itu dalam penerapannya dibutuhkan penggunaan memori yang lebih tinggi sehingga menyebabkan biaya yang dikeluarkan menjadi lebih besar.

2.2 Kubernetes

Seiring berkembangnya waktu dan pengembangan dalam dunia teknologi khususnya *microservices,* diperlukanlah sebuah aplikasi yang mampu melakukan *clustering manangemt container*. Hal ini dibutuhkan demi menjawab permasalahan aplikasi untuk dapat selalu tersedia dan menerima *traffic* yang banyak. Salah satu *platform* yang berfungsi untuk melakukan *clustering management container* adalah Kubernetes. Kubernetes adalah suatu *container orchestration* yang bersifat *opensource* yang diperuntukkan untuk menjalankan aplikasi *container*. Secara resmi kubernetes dikembangkan oleh Google, terinspirasi dari pengalaman selama satu dekade dalam proses menjalankan sistem yang scalable, reliable didalam container melalui application-oriented API (Haris et al., 2020).

Container orchestration yang dilakukan oleh Kubernetes meliputi manajemen *container* mulai dari proses *computing, scheduling, deployment,* hingga *networking*. Untuk mendukung

kinerja yang dilakukan, Kubernetes memiliki tiga komponen. Komponen ini saling terhubung dan membentuk arsitektur seperti pada Gambar 2.2.



Gambar 2.2 Arsitektur Kubernetes

a. Komponen Master

Komponen *master* menyediakan *control plane* yang digunakan oleh *cluster*. Komponen ini nantinya akan berperan dalam melakukan deteksi dan pemberian respons terhadap *event*s yang berlangsung di dalam suatu *cluster*. Dalam menjalankan tugasnya komponen *master* terbagi ke dalam beberapa komponen yang memiliki fungsi berbeda. Komponen tersebut yaitu:

1. Kube-apiserver

   Berfungsi untuk melakukan validasi dan konfigurasi data bagi objek API, seperti *services*, *pods*, *volume* dan lainnya.

2. Etcd

   Adalah ruang penyimpanan bagi *key value* dari konfigurasi data *cluster*.

3. Kube-scheduler

   Mempunyai tugas untuk mengatur bagaimana *events* pada *cluster* berdasarkan *rules* yang dibuat, *resources* yang tersedia, dan sebagainya.

Tidak hanya itu komponen ini juga berfungsi dalam menentukan *node* mana yang akan ditempatkan bagi *pod* baru.

4. Kube Controller Manager

   Melakukan *monitoring cluster* agar sesuai dengan konfigurasi data di dalam *node.*

5. Cloud Controller Manager

   Fungsi pada komponen ini hampir seperti *kube controller manager*, yang membedakan adalah komponen ini bekerja untuk melakukan *monitoring* pada layanan *cloud.*

b. Node Komponen

Komponen – komponen yang berada pada setiap *node* Kubernetes

1. Kubelet

   Komponen yang berfungsi untuk memastikan setiap *container* beroperasi didalam pod. Komponen ini dijalankan dalam bentuk agen dan berada didalam setiap *node.*

2. Kube Proxy

   Komponen yang berfungsi untuk meneruskan koneksi yang dituju dan memelihara aturan – aturan jaringan (*network rules).*

3. Container Runtime

   Komponen ini adalah perangkat lunak yang berfungsi dalam menjalankan *container*. Terdapat beberapa *runtime* yang didukung oleh Kubernetes diantaranya Docker, Containerd, CRI-O, dan lain – lain.

c. Addons

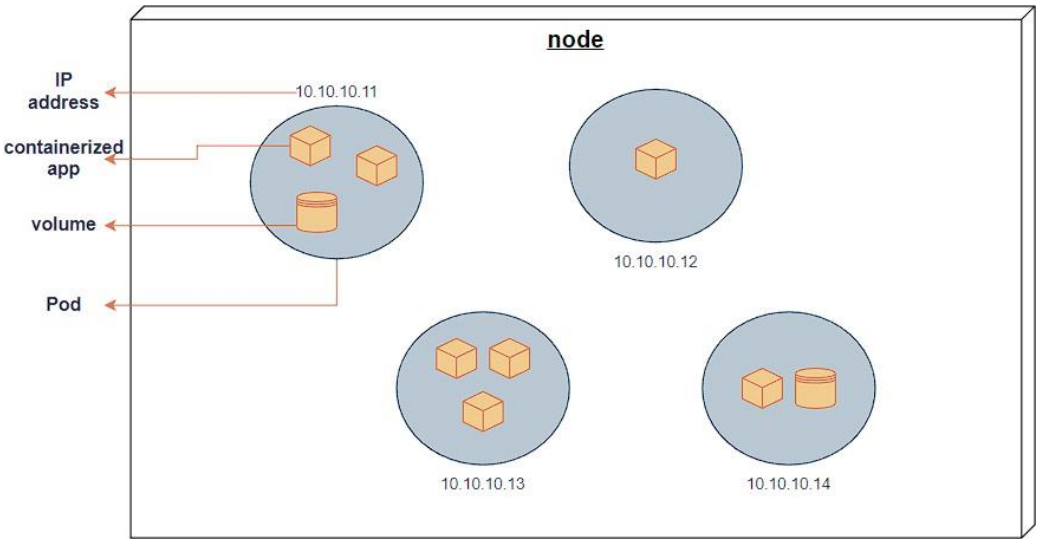
Merupakan tambahan fitur utama untuk mendukung komponen-komponen utama pada Kubernetes dan biasanya *addon* akan membuat Namespace sendiri dan langsung berkomunikasi dengan Namespace kube-system (The Kubernetes Authors, 2020).

Selain komponen di atas terdapat pula beberapa object Kubernetes. *Object* Kubernetes tersebut diantaranya yaitu:

a. Pods

Sebuah pod merupakan unit terkecil yang dapat dibuat pada Kubernetes objek. Pod merupakan representasi dari sekumpulan *container application* dan *volumes* yang

berjalan pada satu *envirotment* (lingkungan) yang sama serta berbagi *resource* yang sama (Burns et al., 2022). Di dalam sebuah pod bisa terdapat satu ataupun lebih *container application* dan memiliki alamat IP dan *port* yang sama. Akan tetapi ini tidak berlaku apabila sebuah *container application* berada pada pod yang berbeda. Hal ini dikarenakan *container application* bersifat *isolated* dari pod lainnya. Oleh karena itu *container application* yang berada pada pod yang berbeda akan memiliki alamat IP dan port yang berbeda pula. Bahkan aplikasi yang berjalan pada sebuah pod di *node* yang sama dapat berada pada *server* yang berbeda. Gambar 2.3 merupakan visualisasi dari Kubernetes pods.

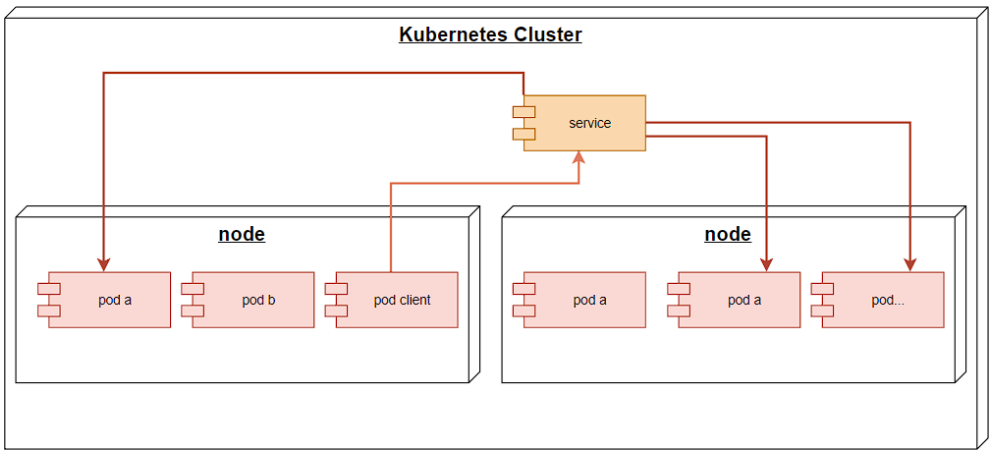


Gambar 2.3 Pods pada sebuah kubernetes

b. Services

Service merupakan tingkat abstraksi objek di mana mendefinisikan *container* sebagai sebuah gerbang untuk mengakses satu atau lebih pod pada *cluster* Kubernetes. Dengan adanya *service*, *client* cukup mengakses sebuah *service* dan otomatis akan diteruskan ke pod yang berada di belakang *service* tersebut. Selain itu IP *address* dan *port* yang dimiliki oleh *service* tidak akan berubah selama *service* tersebut ada. Sehingga *client* tidak perlu tahu alamat setiap pod, dan pod dapat bertambah, berkurang ataupun berpindah tanpa mengganggu *client.* Terdapat berbagai cara untuk menghubungkan antar pods menggunakan *service*, diantaranya

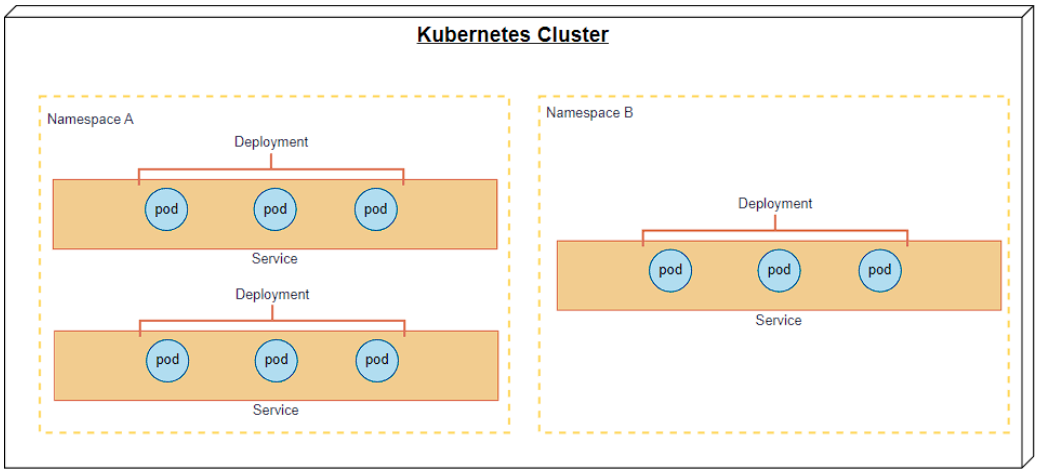
adalah *clusterIP*, *node port*, *load balancer*, dan lainnya. Gambar 2.4 merupakan visualisasi dari Kubernetes *service*.



Gambar 2.4 Service pada sebuah kubernetes

c. Namespace

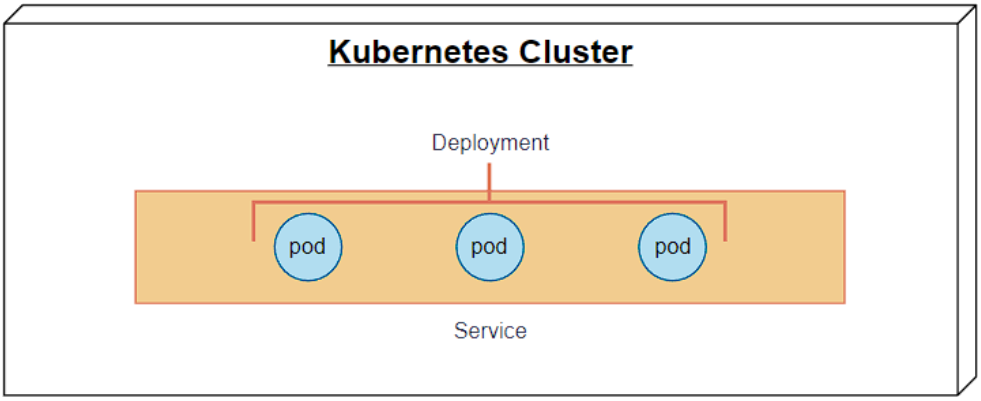
Kubernetes menggunakan *namespace* untuk melakukan pengelompokan objek pada *cluster*. Dengan adanya *namespace* dapat membantu dalam melakukan manajemen terhadap *environment.* Selain untuk manajemen *environment*, *namespace* juga dapat digunakan untuk tujuan pengaturan *resource* penggunaan objek yang ada di Kubernetes sehingga tidak mengganggu *resource* objek yang lain di dalam *namespace* yang berbeda (Kurniawan, 2020). Gambar 2.5 merupakan visualisasi dari *namespace* Kubernetes.



Gambar 2.5 Namespace pada sebuah kubernetes

d. Deployments

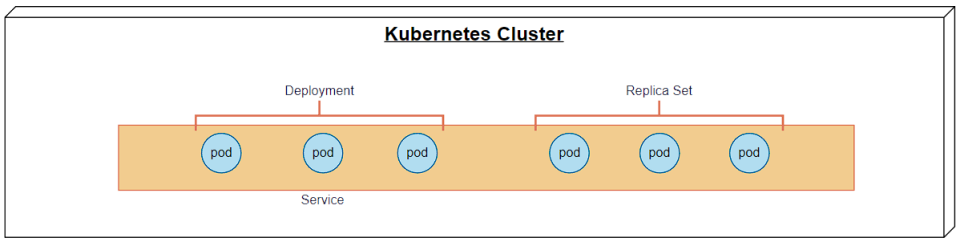
Deployment merupakan salah satu objek Kubernetes yang berfungsi dalam mengelola versi terbaru (Burns et al., 2022). Objek *deployment* memiliki fungsi untuk melakukan konfigurasi serta pembaharuan terhadap *pods*. Dengan fungsi ini apabila terdapat pembaharuan aplikasi, pengguna dapat dengan mudah melakukan pembaharuan versi aplikasi tersebut. Selain itu terdapat juga fungsi *rollback* yang dimana pengguna dapat mengembalikan versi aplikasi ke versi sebelumnya. Gambar 2.6 merupakan visualisasi deployment Kubernetes.



Gambar 2.6 Deployment pada sebuah cluster

e. ReplicaSets

ReplicaSet adalah objek pada Kubernetes yang digunakan untuk melakukan fungsi replikasi pada *Pod*s (Burns et al., 2022). Replicasets merupakan pengembangan dari *replica controller*. Fungsi dari *replica set* adalah membuat replika dari *pods* yang diinginkan. Nantinya setelah sebuah *pod* berhasil direplika, *pods* tersebut memiliki konfigurasi yang sama persis dengan induknya. Gambar 2.7 merupakan visualisasi dari *replicasets* Kubernetes.



Gambar 2.7 Replicasets pada sebuah cluster

2.3 Container Images Scanning

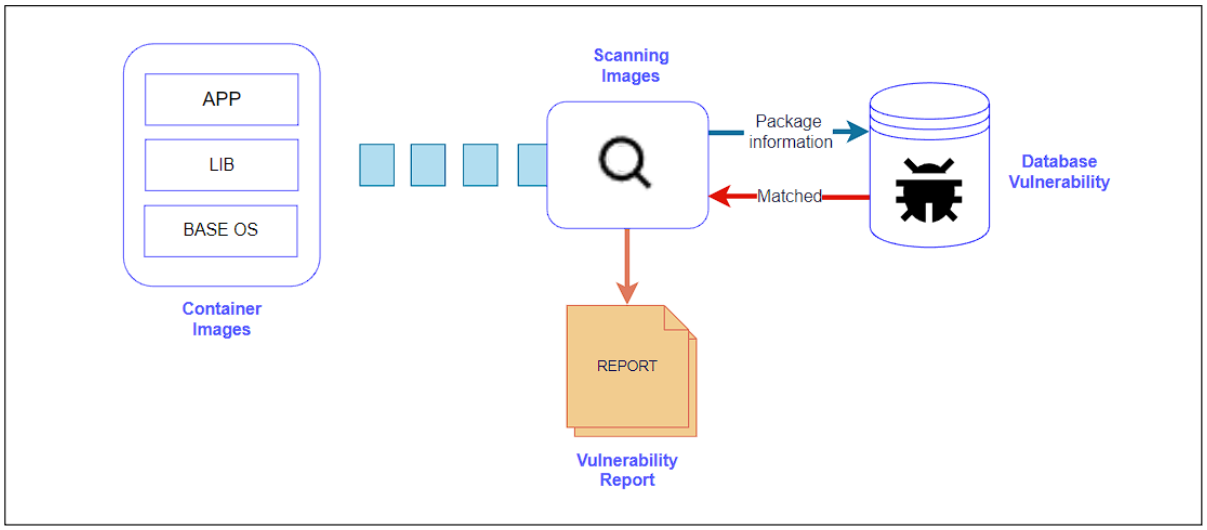
Containerization telah menjadi cara yang efisien untuk meningkatkan produktivitas dalam proses pengembangan teknologi informasi karena memiliki *envirotment* yang lebih ringan dan terisolasi dari *container* yang lainnya (Chen et al., 2022). Namun seiring dengan peningkatan penggunaan *container application,* diiringi juga dengan peningkatan potensi celah keamanan. Secara umum, celah keamana adalah sesuatu kekurangan pada sistem atau infrastruktur yang dapat diakibatkan oleh konfigurasi yang kurang tepat, *malware* ataupun kekurangan pada saat pembuatan aplikasi yang memungkinkan terjadi akses tanpa izin dan melakukan eksploitasi pada sistem tersebut. Celah keamanan yang biasanya terjadi pada *container images* cenderung dapat terjadi ketika paket yang memiliki versi yang lama atau sudah usang yang secara tidak sengaja dipasang atau ditambahkan pada suatu *container application* (Jaisinghani, 2022).

Dalam beberapa tahun terakhir, serangan keamanan menggunakan *container images* sering terjadi (Chen et al., 2022). Cara penyerang melakukan serangan dengan menyusup ke dalam *repository* dan menyuntikan kode *malware* ke dalam *images*. Selanjutnya para pengembang yang mengandalkan atau menggunakan *images* tersebut tidak sadar dan menggunakan *images* tersebut untuk pengembangan yang berulang. Hal ini mengakibatnya semakin cepat suatu aplikasi terinfeksi oleh sebuah *malware*.

Untuk mengatasi masalah yang ada, diperlukannya sebuah pencegahan untuk mengurangi celah keamanan pada *container images. Images scanning* menjadi sebuah pilihan dalam mendeteksi celah keamanan. Dengan melakukan *images scanning,* pengguna dapat mengetahui celah keamanan apa saja yang terdapat dalam *images* yang akan digunakan. Implemetasi *images scanning* dapat dilakukan pada beberapa tahap pengembangan aplikasi. *Scanning* dapat dilakukan pada saat melakukan *build images*, ataupun saat *images* sedang digunakan.

Terdapat perbedaan antara *images scanning* dengan deteksi celah keamanan yang lainnya. Perbedaan ini didapat dari cara suatu informasi celah keamanan didapatkan. Apabila menggunakan *Static Application Security Testing (*SAST) maka berfokus pada *source code* yang digunakan. Menurut penelitian yang dilakukan oleh Fauzan A., SAST murni melakukan analisa pada sumber kode atau *source code* dari aplikasi yang diujikan (Awanda Alviansyah & Ramadhani, 2021). Berbeda dengan *Dynamic Application Security Testing (*DAST). DAST melakukan injeksi kepada aplikasi untuk mengidentifikasi apakah terdapat sebuah celah keamanan ataupun tidak.

Terdapat banyak aplikasi yang menyediakan fungsi untuk melakukan *images scanning.* Secara umum pendekatan yang dilakukan dalam melakukan *scanning* antara aplikasi satu dengan yang lain cenderung sama. Apabila divisualisasikan maka akan tampak seperti Gambar 2.8. Langkah pertama yang dilakukan adalah melakukan *scanning* terhadap packages yang ada pada *container images. Packages* ini berisi mengenai informasi mengenai aplikasi, *library*, serta dasar sistem informasi yang digunakan oleh *container* tersebut. Selanjutnya informasi yang didapatkan akan dicocokkan dengan *database* celah keamanan yang dimiliki. Terakhir seluruh informasi yang telah dicocokkan dengan *database* yang dimiliki akan dijadikan suatu laporan celah keamanan.



Gambar 2.8 Scanning *images* untuk celah keamanan secara umum

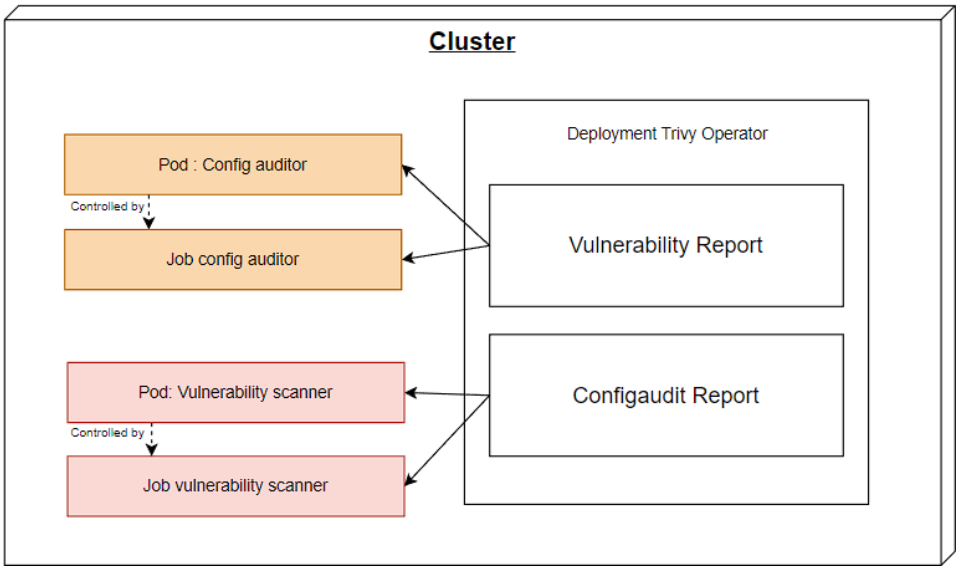
2.4 Trivy Operator

Trivy Operator merupakan aplikasi *container images scanning* yang dapat melakukan *scanning* pada *run time cluster.* Aplikasi ini dikembangkan oleh salah satu perusahaan yang bergerak pada bidang *cloud native securiy* yaitu Aqua Security. Perusahaan Aqua Security sendiri sudah banyak membantu perusahaan seperti Paypal, Juniper, Adobe, dan lainnya dalam meningkatkan keamanan pada aplikasi yang mereka gunakan. Tidak hanya itu, Trivy juga digunakan oleh aplikasi Gitlab CI sebagai sarana untuk melakukan *container scanning.*

Terdapat dua versi pada aplikasi Trivy yaitu Trivy Operator dan Trivy. Keduanya sama–sama dapat melakukan *scanning images.* Perbedaannya terletak pada kondisi *images* yang akan dilakukan *scanning.* Trivy dapat melakukan *scanning* pada Git *repository, container images,* atau konfigurasi IaC (Infrastructure as Code). Sedangkan Trivy Operator berfokus pada

Kubernetes Cluster. Hal ini dikarenakan Trivy Operator dapat di *deploy* di dalam sebuah *cluster* sedangkan Trivy hanya dapat di-*install* pada mesin lokal atau dipasang pada CI/CD *pipeline.*

Secara umum proses kerja Trivy Operator akan tampak seperti Gambar 2.9. Setelah Trivy Operator berhasil di-*deploy*, secara otomatis *controller* akan bertanggung jawab dalam membuat sebuah *job*. *Job* yang dibuat akan bertanggung jawab dalam membuat sekaligus melakukan pengawasan terhadap *agent scanning* yang berbentuk sebuah *pod.* Selanjutnya *agent* akan melakukan *scanning* terhadap *images* yang berada pada *cluster* kubernetes. Apabila *agent* telah selesai melakukan *scanning*, *controller* akan menghasilkan sebuah laporan dalam bentuk Kubernetes *resource* yaitu CRDs (Custom Resource Definition). Setelah semua tahap dilakukan, *controller* akan menghancurkan *job* serta *pods* yang digunakan untuk *scanning images.* Tidak hanya itu *controller* juga akan melakukan pengawasan terhadap perubahan yang terjadi pada *cluster.* Hal ini bertujuan agar apabila terdapat perubahan pada *cluster* seperti *deploy*, *update* ataupun *change* pada *cluster*, *controller* dapat melakukan *scanning* ulang terhadap perubahan.



Gambar 2.9 Proses kerja Trivy Operator

2.5 Grafana

Grafana adalah sebuah *software opensource* yang membaca data *metrics* untuk dibuat menjadi sebuah grafik atau sebuah data tertulis. Grafana sering digunakan untuk melakukan analisis data dan *monitoring*. Grafana mendukung banyak *storage backends* yang berbeda untuk data *time series* (Source Data) (Mutiara et al., 2020). Grafana menggunakan panel –

panel yang dapat dikonfigurasi ataupun menggunakan *templating* dalam menampilkan visualisasi data.

Grafana mendukung banyak basis data seperti Prometheus, Graphite, INfluxDB, ElasticSearch, MySQL, PostgreSQL, dan lainnya. Walaupun setiap sumber data memiliki *query editor* yang berbeda, Grafana dapat mampu menggabungkan sumber data tersebut pada satu *dashboard*. Hal ini dapat terjadi karena setiap panel pada *dashboard* hanya terhubung dengan satu sumber data tertentu. Selain itu keuntungan dari penggunaan Grafana adalah tidak ada data yang dikirimkan ke *cloud vendor* untuk alasan keamanan dan lainnya.

2.6 Tinjauan Pustaka

Referensi yang digunakan pada penulisan tugas akhir ini mengacu terhadap beberapa penelitian – penelitian terdahulu, seperti yang terdapat pada Tabel 2.1.

Tabel 2.1 Tinjauan Pustaka

No.	Peneliti	Judul	Tahun	Tujuan	Metode	Hasil
1.	Bhunpinder Kaur, Mathieu Dugre, Aiman Hanna, dan Tristan Glatard	An analysis of security vulnerability in container images for scientific data analysis	2023	Analisa celah keamanan pada *container images* yang digunakan untuk data *analysis*	*container images scanning*	Peneiliti menemukan bahwa kebanyakan *container images* yang digunakan memiliki celah keamanan yang banyak. Peneliti juga menyarankan untuk menggunakan OS yang memiliki *long term support* dari pengembang.
2.	Omar Javed dan Salman Toor	Understanding the Quality Of Container Security Vulnerability Detection Tools	2021	Mempelajari kualitas dari aplikasi *container security vulnerability detection*	*detection coverage* dan *detection hit ratio*	Peneliti menemukan bahwa terdapat perbedaan antara hasil *report scanning* dikarenakan perbedaan *database* yang digunakan.
3.	Prawal Saxena	Container Image Security with Trivy and	2022	Implementasi Trivy pada AWS CI/CD	*AWS Service* dan Kubernetes	Trivy dapat melakukan *scanning images*

		Istio Inter-Service Secure Communication in Kubernetes		dan Istio sebagai validasi jaringan pada Kubernetes		melalui AWS CI/CD dan Istio dapat menerapkan aturan yang diterapkan pada jaringan internal *cluster* Kubernetes.
4.	Salma Rachman Dira dan Muhammad Arif Fadhly Ridha	Monitoring Kubernetes Cluster Menggunakan Prometheus dan Grafana	2022	Menerapkan Prometheus dan Grafana sebagai alat untuk melakukan *monitoring cluster*	Pengujian fungsional dan *performance*	Prometheus dan Grafana dapat melakukan monitoring dengan menapilkan data dalam bentuk grafik. Selain itu pada pengujian *performance* terdapat perubahan pada grafik seiring dengan *request* yang dilakukan oleh *user.*
5.	Arief Indriarto Haris, Rd. Angga Ferianda, Budhi Riyanto, Fajar Iman Nugraha, Januar Abadi	Pengamanan Container Orchestration Berbasis Kubernetes di Lembaga Penerbangan dan Antariksa Nasional (LAPAN)	2020	Mengetahui tindakan pengamanan yang diperlukan pada *container orchstration* berbasis Kubernetes	studi literatur, observasi, wawancara, dan analisis data	Dari 7 tindakan yang harus dilakukan dalam meningkatkan keamanan pada Kubernetes di lingkungan LAPAN salah satunya adalah melakukan *scanning image* dan *monitoring cluster* Kubernetes
6.	Panca Rizki Perkasa dan Evangs Mailoa	Adopsi DEVSECOPS Untuk Mendukung Metode Agile Menggunakan Trivy Sebagai Security Scanner Docker Image dan Dockerfile	2023	Melakukan implementasi Trivy agar dapat meningkatkan kesadaran dari *developer* dalam hal keamanan aplikasi serta mempercepat dalam	Studi literatur, implementasi, pengujian dan analisis	Trivy yang telah berhasil diimplementasikan akan membentuk SDLC yang bersifat *continous*. Selain itu Trivy memiliki bentuk format HTML yang dapat dengan mudah dibaca oleh *developer.*

				memperbaiki celah keamanan yang ada.		
7.	Milenia Oktaviana, Adityas Widjajarto, dan Ahmad Almaarif	Analisis Vulnerability Management Pada Container Docker Menggunakan Opensource Scanner Berdasarkan Standar Cyber Resilience Review (CRR)	2022	Melakukan uji coba dan membandingkan penggunaan *scanner* Anchore dan Aquasec	Uji coba dan analisis	Pengujian menggunakan Aquasec dan Anchore berhasil dilakukan serta dengan melakukan pembaharuan sesuai dengan rekomendasi yang diberikan mampu menurunkan jumlah tingkat celah keamanan yang ada. Selain itu pada penelitian ini juga didapati bahwa Aquasec memiliki *vulnerability recource* yang lebih banyak dibandingkan dengan Anchore.

BAB III
PELAKSANAAN MAGANG

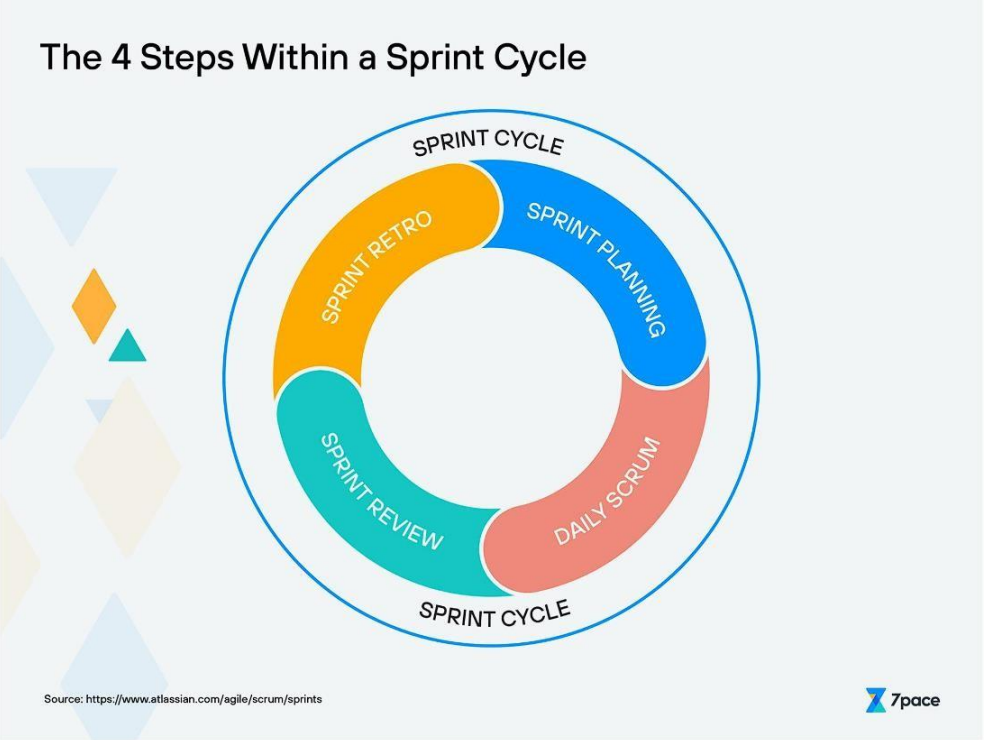
3.1 Manajemen Proyek

Penulis menjalani kegiatan magang sebagai *Software Engineer* BSI UII di masa peralihan pandemi Covid-19 yang terjadi di Indonesia. Oleh sebab itu, penulis dapat melakukan kegiatan magang di kantor (*work from office)* setiap hari dikarenakan sudah tidak ada lagi pembatasan yang dilakukan. Ditambah dengan kondisi kegiatan perkuliahan di kampus UII juga sudah dilaksanakan secara *luring* (luar jaringan) sehingga memungkinkan untuk melakukan segala aktivitas dilingkungan UII. Setiap minggunya penulis diwajibkan untuk memenuhi 20 jam kerja mulai dari jam 8 pagi dan selesai pada jam 12 siang.

Selama delapan bulan menjalankan kegiatan magang, penulis dituntut untuk dapat berkomunikasi aktif dengan mentor terkait teknis proyek maupun manajemen proyek. Dalam penjalanan proyek penulis menerapkan metode *sprint.* Metode ini digunakan karena dirasa lebih fleksibel terhadap perubahan dan kebutuhan perusahaan. Metode ini memiliki beberapa tahapan seperti yang ada pada Gambar 3.1.

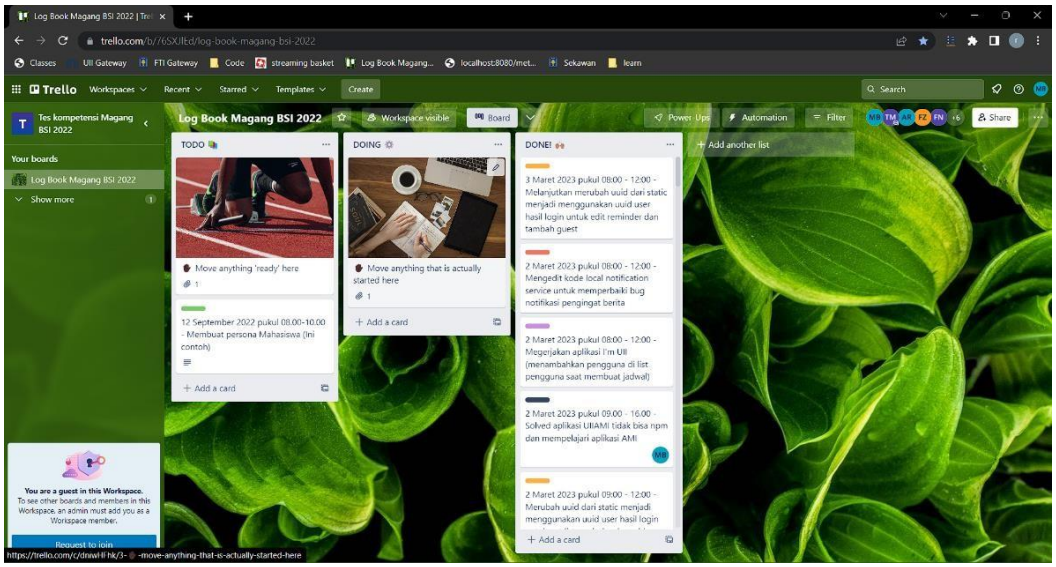
Pada setiap awal pengerjaan *task,* penulis bersama mentor akan melaksanakan *sprint planning* untuk menentukan *task* apa yang akan dikerjakan, kebutuhan apa saja yang diperlukan, serta batasan yang diperlukan selama pengembangan. Tahap selanjutnya adalah melakukan *daily scrum* atau melaksanakan *task* yang sudah ditentukan pada tahap sebelumnya. Setiap harinya penulis diwajibkan untuk melakukan *daily stand up.* Kegiatan ini bertujuan untuk memberikan informasi mengenai apa yang akan dikerjakan serta kendala yang dihadapi. Apabila terdapat kendala yang dihadapi, mentor akan membantu atau memberikan saran setelah sesi *daily stand up* selesai. Apabila terdapat *progress* yang belum selesai, penulis akan disampaikan kepada mentor untuk diberikan *feedback* untuk keputusan apakah dapat dilanjut ke langkah berikutnya atau perlu direvisi Selain itu penulis juga menggunakan aplikasi Trello seperti pada Gambar 3.2.

Apabila suatu *sprint* sudah selesai maka akan dilakukan *sprint review.* Pada tahap ini penulis memaparkan hasil yang dikerjakan selama *sprint* berlangsung. Tidak hanya itu, pada tahap ini juga pengerjaan penulis di *review* oleh mentor dan nantinya akan diputuskan oleh mentor apakah proyek ini telah selesai atau perlu dilakukan perbaikan dan akan dikerjakan pada *sprint* selanjutnya.



Gambar 3.1 *Sprint cycle*

(sumber: www.atlassian.com)



Gambar 3.2 Aplikasi Trello untuk manajemen proyek

3.2 Pelaksanaan Proyek

Pada pengerjaan proyek terdapat beberapa tahapan yaitu instalasi aplikasi, pembuatan *dashboard*, dan melakukan pengujian menggunakan *container images.* Pada proses pengerjaan

proyek ini, penulis membuat sebuah lokal *cluster* Kubernetes mengguakan Minikube serta dockerfile. Spesifikasi yang digunakan untuk membangun *cluster local* seperti pada Tabel 3.1. Alasan dipilihnya Minikube karena dapat dijalankan melalui *docker dekstop* dan tidak memerlukan sebuah *virtual machine* khusus. Hal ini sangat membantu penulis karena *operating system* yang digunakan selama melaksanakan kegiatan magang adalah Windows 11 Home Edition.

Tabel 3.1 Spesifikasi lokal *cluster*

No.	Software	Versi
1.	Windows	11
2.	Docker	20.10.17
3.	Minikube (3GB RAM, 4 CPUS)	1.26.1

3.2.1. Instalasi Trivy Operator

Tahap awal yang dilakukan adalah melakukan instalasi aplikasi Trivy Operator pada *cluster* lokal yang telah disiapkan. Terdapat dua cara untuk melakukan instalasi Trivy Operator yaitu dengan menggunakan *deployement* melalui *manual deploy* atau dengan menggunakan Helm chart. Penulis memilih menggunakan Helm Chart dikarenakan lebih mudah dalam melakukan kustomisasi aplikasi. Untuk melakukan konfigurasi hanya tinggal mengubah *file values.yaml.* Tidak hanya itu *file* ini lebih mudah dibaca dibandingkan *manual deploy.*

Hal pertama yang perlu disesuaikan pada *values.yaml* adalah *target namespcae.* Tujuan konfigurasi ini adalah menentukan *namespace* mana yang akan dilakukan *scanning images*. Sebelumnya penulis telah menyiapkan *namespace* dengan nama *frontend* dan *backend* yang nantinya akan digunakan untuk *deployment.* Selanjutnya diperlukan konfigurasi untuk berapa lama sebuah *report* akan bertahan sebelum dihapus. Hal ini dapat diatur dengan mengubah konfigurasi pada *report time to live* (reportTTL). Apabila tidak diatur, maka setiap kali *scanning* berhasil dijalankan *report* tersebut akan menumpuk pada *storage* yang digunakan. Pada proyek ini penulis menginginkan agar *record* hasil *report* hanya terdapat pada *storage dashboard* sehingga tidak membebani *storage* yang ada dengan melakukan dua kali penyimpanan.

Penulis dalam proyek ini ingin agar hasil *report* dapat diolah menjadi sebuah *dashboard*. Oleh karena itu diperlukan sebuah *service monitor* sebagai penjembatan antara *report scanning* dan *dashboard* yang akan dibuat. Untuk melakukan hal tersebut diperlukan menambah *service monitor* dan *label. Label* diberikan nama sesuai dengan *label* yang digunakan oleh Prometheus.

Apabila *label* pada aplikasi Trivy Operator dan Prometheus berbeda maka tidak akan terbaca dan data tidak dapat diolah.

Secara *default* tidak terdapat *description*, *links*, *cvss*, dan lainnya pada hasil *report*, namun untuk melengkapi hasil *report* dapat ditambahkan pada *additional vulnerability*. Selain itu penulis mengubah konfigurasi *default* pada *ignored unfixed*. Nilai awal yang ditetapkan oleh aplikasi Trivy sendiri bernilai *true* dan selanjutnya diubah menjadi *false*. Hal ini bertujuan untuk menampilkan semua data celah keamanan termasuk yang belum memiliki versi pembaharuan. Setelah semua konfigurasi ditambahkan maka akan tampak seperti pada Gambar 3.3.

```
# targetNamespace defines where you want trivy-operator to operate. By
# default, it's a blank string to select all namespaces, but you can specify
targetNamespaces: "frontend,backend"

# scannerReportTTL the flag to set how long a report should exist.
scannerReportTTL: "15m"

  # scanJobTimeout the length of time to wait before giving up on a scan job
scanJobTimeout: 15m

# scanJobsConcurrentLimit the maximum number of scan jobs create by the operator
scanJobsConcurrentLimit: 10

# metricsVulnIdEnabled the flag to enable metrics about cve vulns id
metricsVulnIdEnabled: true

# to install the trivy operator with the Service Monitor
serviceMonitor:
  enabled: true
  interval: ""
  labels: {relase: prometheus}

# additionalVulnerabilityReportFields is a comma separated list of additional fields which
additionalVulnerabilityReportFields: "Description,Target,Class,PackageType"

# ignoreUnfixed is the flag to show only fixed vulnerabilities in
ignoreUnfixed: false

compliance:
# cron this flag control the cron interval for compliance report generation (every 1 hour)
cron: 0 * * * *
```

Gambar 3.3 Konfigurasi instalasi Trivy Operator

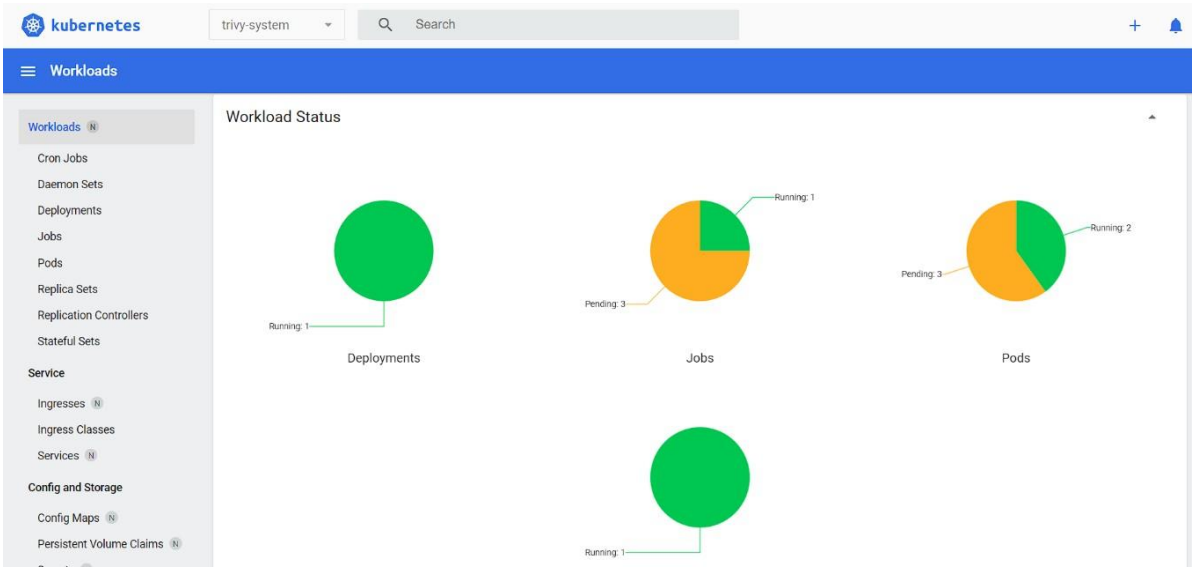
Langkah selanjutnya adalah melakukan instalasi aplikasi Trivy Operator menggunakan *file* konfigurasi yang telah dimodifikasi. Dengan melakukan perintah seperti pada Gambar 3.4 maka aplikasi akan terinstal sesuai dengan konfigurasi. Pada saat melakukan instalasi terdapat tambahan perintah untuk membuat *namespace* yang akan digunakan oleh aplikasi. Gambar 3.5 menunjukkan bahwa aplikasi berhasil terinstal pada *cluster* lokal yang telah dibuat. Saat pertama kali melakukan instalasi secara otomatis akan terbuat sebuah *pod* dan *jobs* yang bertugas untuk melakukan *scanning images* untuk pertama kali. *Pods* dan *job* akan tampak seperti pada Gambar 3.6 dan Gambar 3.7

```
helm install trivy-operator ./deploy/helm \
  --namespace trivy-system \
--create-namespace \
```

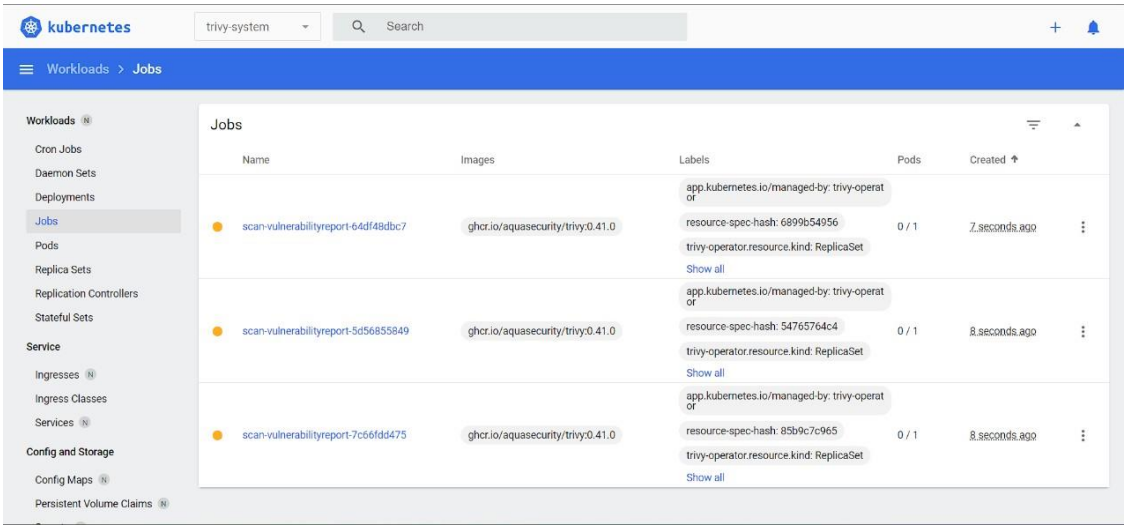
Gambar 3.4 Perintah instalasi Trivy Operator

```
NAME: trivy-operator
LAST DEPLOYED: Tue Jun  6 14:56:10 2023
NAMESPACE: trivy-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
You have installed Trivy Operator in the trivy-system namespace.
It is configured to discover Kubernetes workloads and resources in
all namespace(s).

Inspect created VulnerabilityReports by:

    kubectl get vulnerabilityreports --all-namespaces -o wide

Inspect created ConfigAuditReports by:

    kubectl get configauditreports --all-namespaces -o wide

Inspect the work log of trivy-operator by:

    kubectl logs -n trivy-system deployment/trivy-operator
```

Gambar 3.5 Instalasi Trivy Operator



Gambar 3.6 Workload status saat aplikasi Trivy pertama diinstal

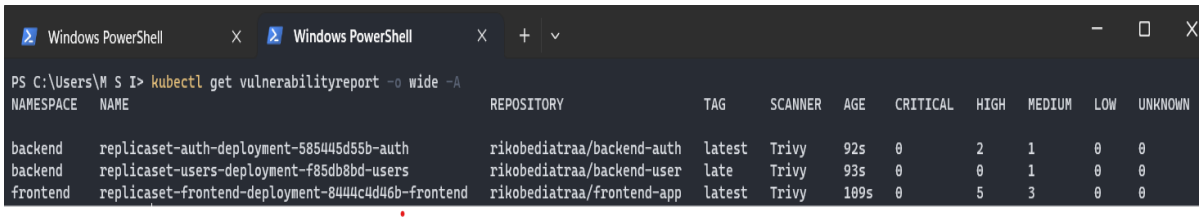


Gambar 3.7 Jobs yang menjalankan fungsi *scanning images*

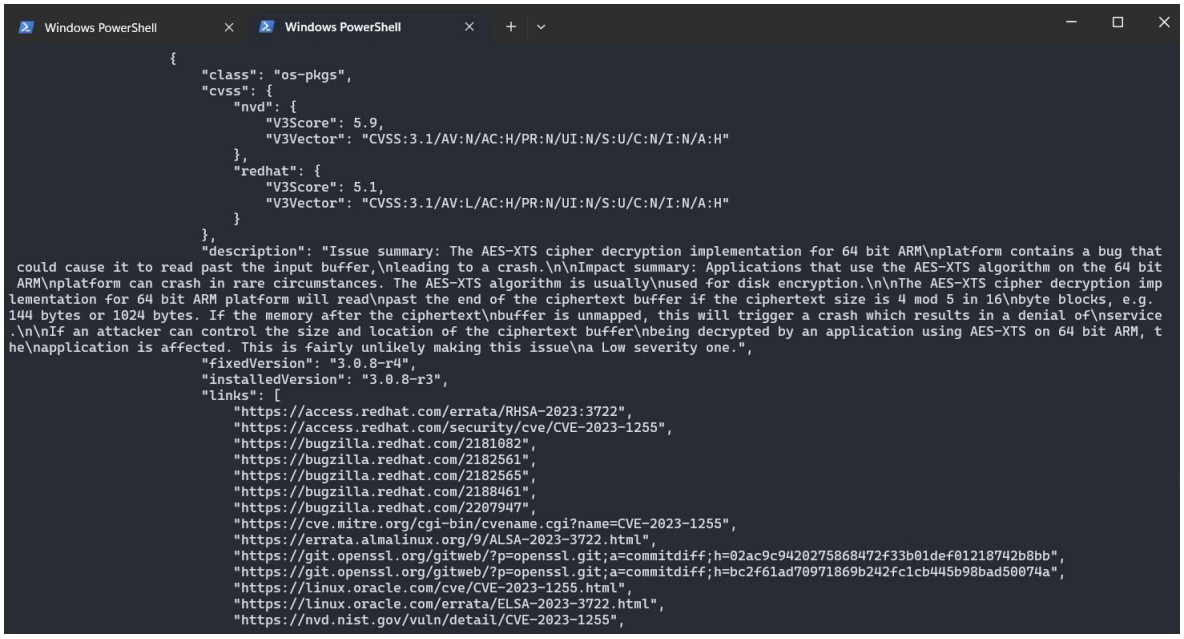
Untuk melihat hasil *report* dapat menggunakan perintah pada Gambar 3.8. Pada perintah pertama akan menghasilkan *report* berbentuk rangkuman celah keamanan pada setiap *deployment* berdasarkan jumlah *vulnerability* yang ada seperti yang terdapat pada Gambar 3.9. Berbeda dengan perintah yang kedua, perintah tersebut menghasilkan *report* yang berbentuk format JSON *file*. *Report* yang dihasilkan dalam bentuk JSON lebih lengkap dari pada format sebelumnya, hal ini dikarenakan terdapat data – data pelengkap seperti deskripsi, *fixed* dan install *version*, link, *score*, *severity*, dan lainnya seperti pada Gambar 3.10.

```
kubectl get vulnerabilityreport -o wide -A

atau

kubectl get vulnerabilityreport -o JSON -A
```

Gambar 3.8 Perintah untuk mendapatkan *report* celah keamanan



Gambar 3.9 Report dalam bentuk rekap

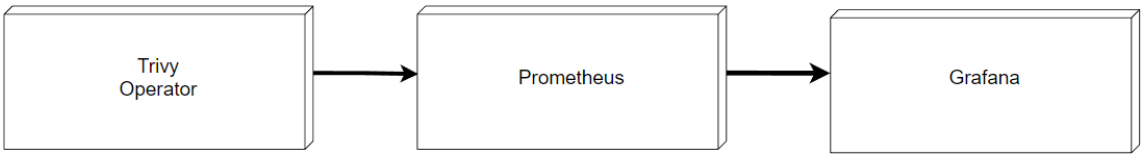


Gambar 3.10 Rekap dalam bentuk JSON file

3.2.2. Pembuatan Dashboard Grafana

Pada proyek ini penulis menggunakan Prometheus dan Grafana dalam melakukan visualisasi hasil *images scanning*. Prometheus sendiri adalah *toolkit* pemantauan dan peringatan sistem yang bersifat *opensource* dan dibuat oleh SoundCloud. Sebagai layanan *monitoring*, Prometheus memantau hal tertentu atau yang biasa disebut target. Target yang dimaksud dapat berbentuk *server* tunggal atau *endpoint* yang melalui HTTP, HTTPS, DNS, dan lainnya (Haris et al., 2020) Prometheus menyimpan hasil *metric* yang dikumpulkan dalam

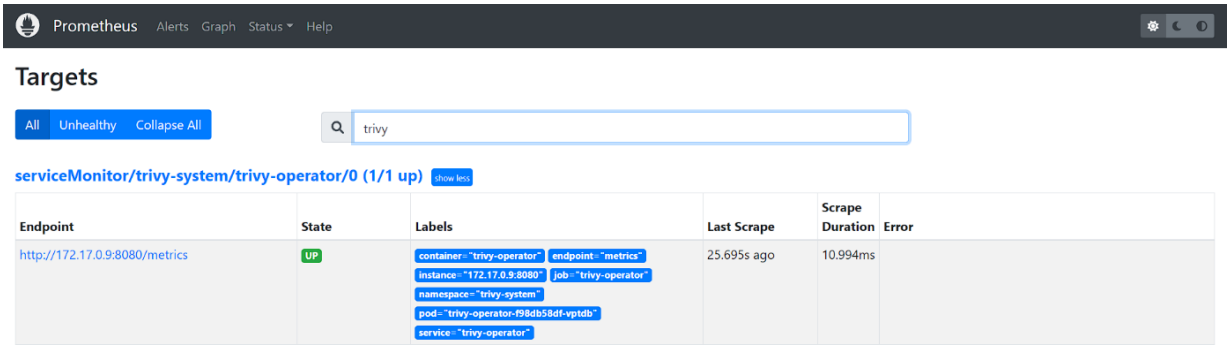
sebuah *time series database.* Nantinya *time series database* ini yang digunakan oleh Grafana untuk melakukan visualisasi data.



Gambar 3.11 Alur hasil *scanning images* hingga menjadi *dashboard*

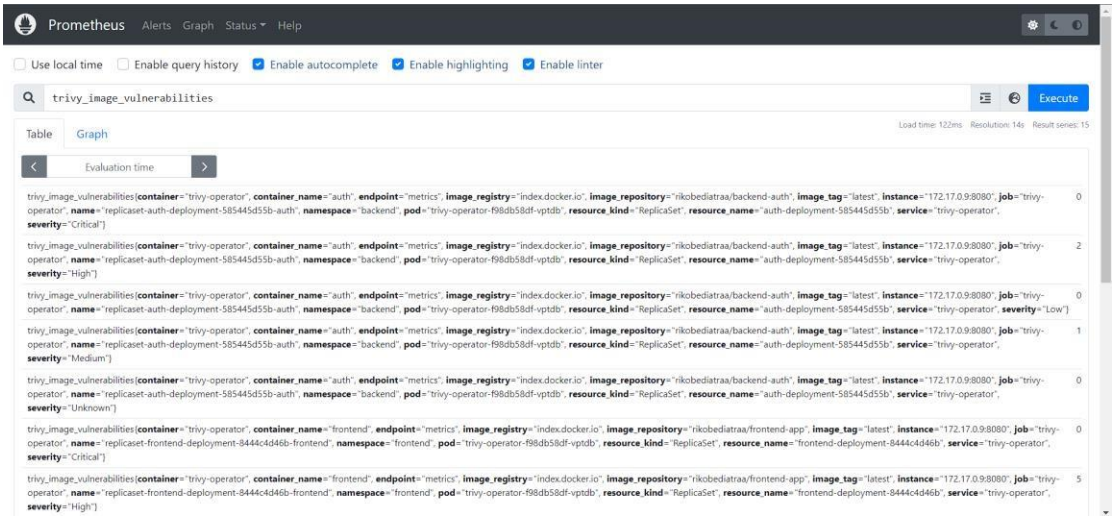
Gambar 3.11 merupakan visualisasi dari alur yang digunakan dalam menampilkan hasil *scanning images.* Pertama Trivy akan melakukan *scanning* dan menghasilkan *report* dalam bentuk JSON dan *metrics.* Selanjutnya data *metrics* ini akan ditangkap oleh Prometheus dan dikumpulkan dalam *time series database.* Terdapat dua *metrics* yang dihasilkan oleh Prometheus yaitu *images vulnerability* dan *vulnerability id. Images vulnerability* adalah sekumpulan *metrics* yang berisi detail celah keamanan berdasarkan *container images*. Sedangkan *vulnerability id* berisi mengenai detail celah keamanan berdasarkan data celah keamanan itu sendiri. Lalu Grafana menggunakan dan mengelola *database* yang dimiliki oleh Prometheus dalam bentuk panel untuk dijadikan sebuah dashboard.

Setelah berhasil mengimplementasikan Trivy Operator, hal yang harus dilakukan selanjutnya adalah memeriksa bahwa *metrics* yang dihasilkan telah dibaca oleh Prometheus sebagai sebuah target. Untuk memastikannya dapat dilihat dengan mengakses target pada *service* dari Prometheus. Gambar 3.12 menunjukkan bahwa Trivy Operator berhasil terbaca oleh Prometheus. Dari Gambar 3.12 juga dapat diketahui bahwa Trivy menggunakan *port* 8080/metrics sebagai *endpoint* untuk mendapatkan hasil *scanning images.* Selain itu terdapat juga informasi mengenai *label* yang digunakan, waktu terakhir data diambil, berapa lama pengambilan data dan status dari target.

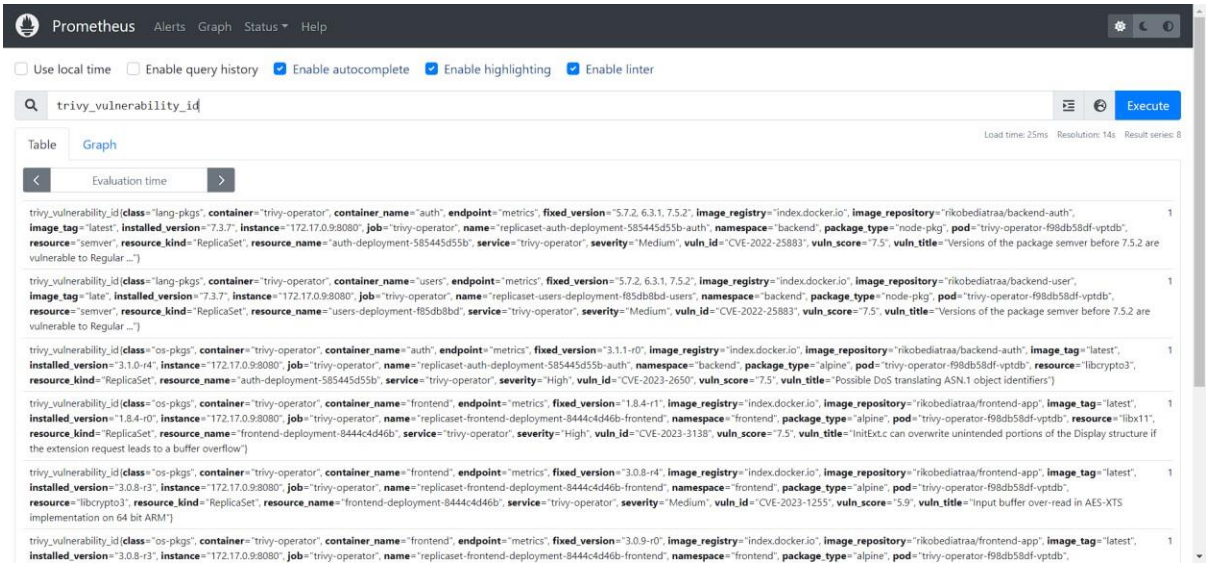


Gambar 3.12 Target trivy operator yang dibaca Prometheus

Terdapat beberapa *metrics* yang dapat digunakan untuk mengelola data. Namun pada proyek ini hanya menggunakan dua *metrics* yaitu *vulnerability id* dan *image vulnerability.* Kedua metrics ini digunakan karena menampung data berisi informasi celah keaman pada *container*. Gambar 3.13 menunjukkan *metrics* berdasarkan *images vulnerability* sedangkan Gambar 3.14 menunjukkan *metrics* berdasarkan *vulnerability id.* Perbedaan antara kedua *metrics* ini adalah *images vulnerability* menampilkan data celah keamanan berdasarkan *images* apa saja yang digunakan pada *namespace* sedangkan *vulnerability id* menampilkan data berdasarkan jenis celah keamanannya.



Gambar 3.13 Metrics images vulnerability



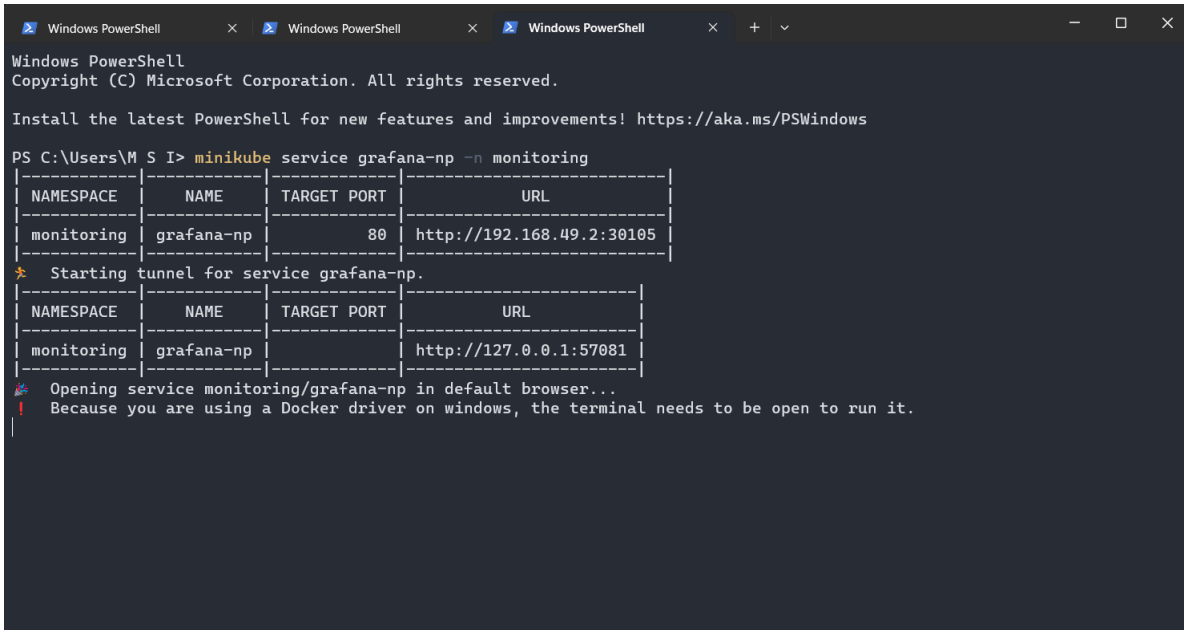
Gambar 3.14 Metrics berdasarkan vulnerability id

Selanjutnya adalah pembuatan *dashboard.* Untuk mengakses Grafana dapat dilakukan dengan cara membuka *service* Grafana agar dapat diakses dari luar *cluster*. Gambar 3.15 merupakan perintah yang digunakan untuk membuka *service* Grafana. Pada perintah pertama berfungsi untuk membuat *service* dengan tipe NodePort. *Service* nodeport merupakan sebuah sistem untuk meneruskan permintaan ke dalam *cluster* berdasarkan port (Wahyu et al., 2018). Pada perintah pertama terdapat beberapa parameter yaitu *type* atau tipe yang menentukan tipe *service*, *target port* dan *port* yang akan digunakan. Perbedaan antara *target port* dan *port* adalah *port* merupakan *internal service* yang digunakan untuk mendengarkan *request.* Sedangkan *target port* merupakan *port* yang bertugas untuk meneruskan ke dalam *container.* Sehingga parameter yang digunakan pada *port* dapat berubah sesuai dengan keadaan, namun parameter *target port* harus disesuaikan dengan aplikasi Grafana saat pertama kali melakukan instalasi. Perintah kedua adalah perintah yang digunakan untuk melihat apakah *service* berhasil dibuat dengan cara melihat daftar *service* yang ada pada *namespace* monitoring. Perintah terakhir digunakan untuk membuka *service* agar dapat diakses dari luar *cluster*. Dengan menggunakan perintah minikube, secara otomatis minikube akan membuat sebuah *tunel* bagi *service* yang akan dibuka. Gambar 3.16 menunjukkan minikube berhasil membuat *tunel* dan terdapat URL yang dapat diakses untuk aplikasi Grafana.

```
#kubectl expose pod/grafana-np --type=NodePort --target-port=30105 --port=80 -n monitoring

#kubectl get service -n monitoring
```

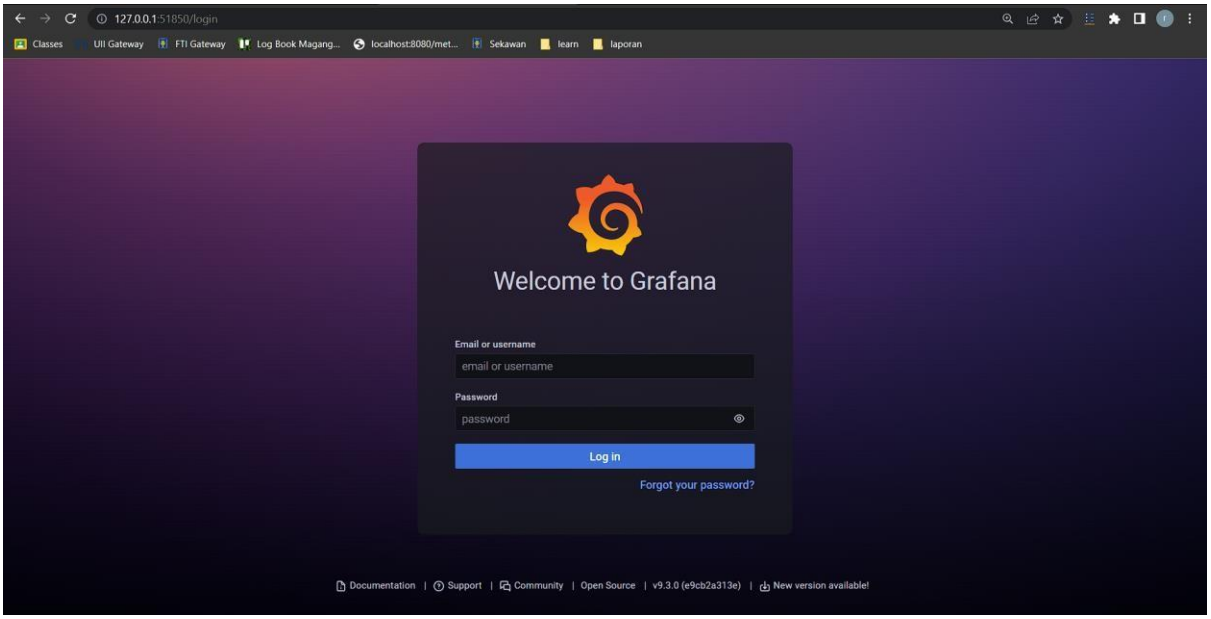
```
#minikube service grafana-np -n monitoring
```

Gambar 3.15 Perintah untuk membuka *service* Grafana

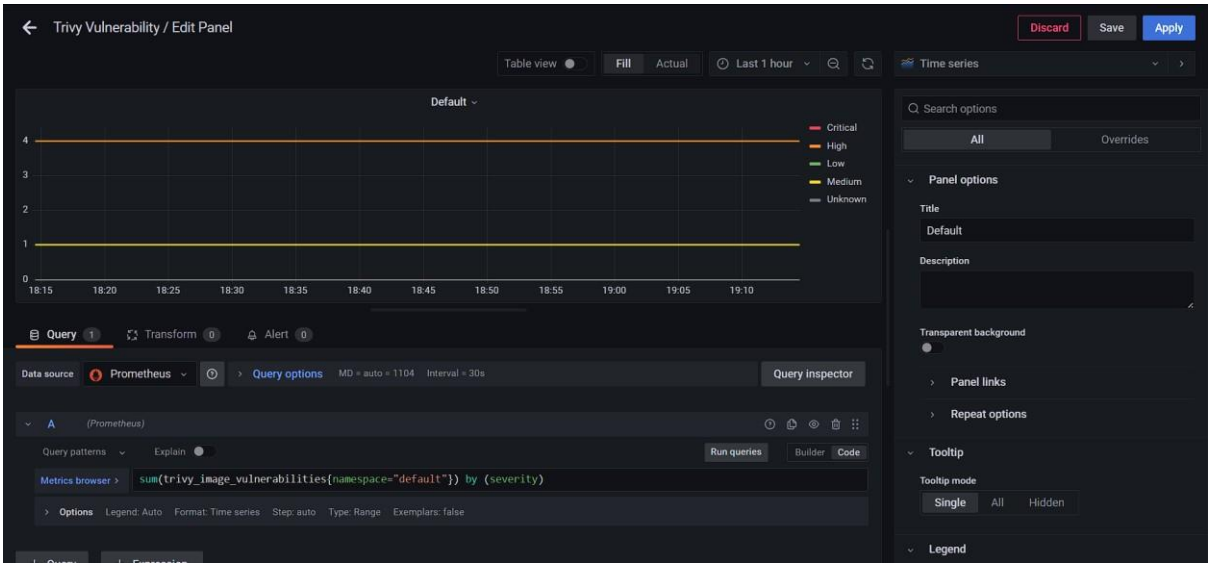


Gambar 3.16 Service grafana

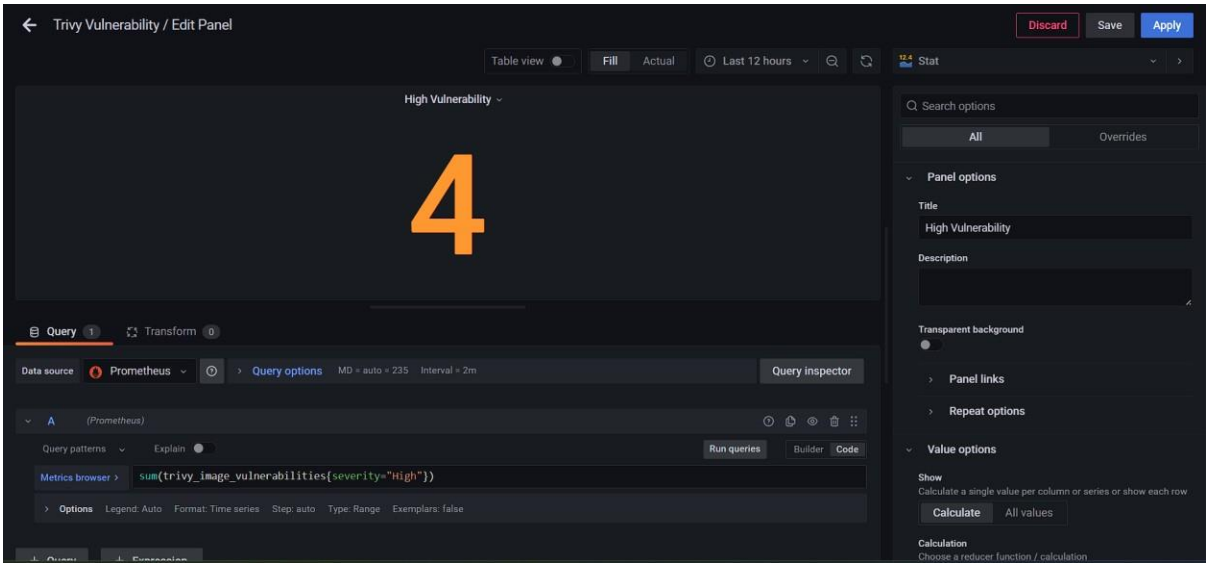
Gambar 3.17 merupakan halaman awal Grafana yang telah berhasil diakses melalui *url.* Setelah berhasil mengakses Grafana, penulis menggunakan panel–panel yang telah disediakan untuk mengelola data hasil *scanning images.* Dalam proses pengelolaan data Grafana menyediakan kustomisasi tampilan data seperti membuat *time series* pada Gambar 3.18, angka pada Gambar 3.19, dan tabel seperti yang terdapat pada Gambar 3.20. Alasan penulis menggunakan *time series* adalah untuk menampilkan grafik garis mengenai jumlah *vulnerability.* Penggunaan angka dimaksudkan agar dapat dengan mudah mengetahui jumlah *vulnerability* berdasarkan tingkat kerentanannya. Terakhir penggunaan tabel agar mudah membaca data – data detail mengenai celah keamanan.



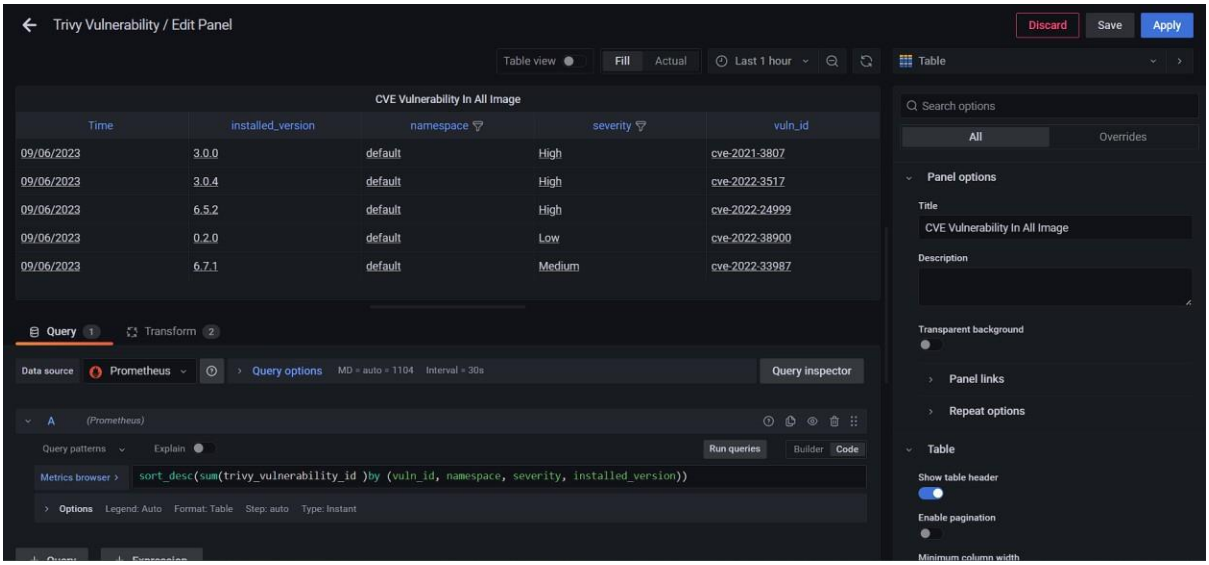
Gambar 3.17 Aplikasi Grafana



Gambar 3.18 Panel time series



Gambar 3.19 Panel angka



Gambar 3.20 Panel tabel

Langkah terakhir adalah melakukan uji coba menggunakan *deployment.* Pada tahap ini penulis melakukan *deployment* pada kedua namespace yaitu *frontend* dan *backend*. Selain itu untuk menguji *dashboard* penulis menguji dengan melakukan *upgrade version* dari aplikasi. Pengujian ini dilakukan untuk menguji apakah *scanning images* berhasil dilakukan, *dashboard* berhasil menerima data, dan rekomendasi yang diberikan dapat dilakukan. Tahapan ini menggunakan *deployment* seperti pada Tabel 3.2.

Tabel 3.2 Spesifikasi *deployment* pada proses pengujian

No.	Deployment	Namespace	Parent Images	Pengujian 1 (versi base os)	Pengujian 2 (versi base os)
1.	frontend-app	frontend	node js	14-alpine	16-alpine
2.	frontend-app	frontend	nginx	1.19-alpine	1.25-alpine
3.	backend-user	backend	node js	14-alpine	16-alpine
4.	backend-auth	bakend	node js	14-alpine	16-alpine

File untuk melakukan *deployment* seperti pada Gambar 3.21. Secara garis besar pada penugasan ini semua aplikasi yang di-*deploy* memiliki *service* nya masing–masing. Sehingga pada saat melakukan *deployment*, *service* yang digunakan perlu didefinisikan ulang. Tidak hanya itu agar setiap aplikasi dapat saling terkoneksi diperlukan *port* yang berbeda–beda. Hal ini untuk mencegah agar antar *service* tidak saling bertabrakan yang dapat menyebabkan *service* tidak terhubung dengan aplikasi semestinya.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: //nama deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: //images yang digunakan
          env:
            - name: AUTH_ADDRESS
              value: "auth-service.default"
            - name: TASKS_FOLDER
              value: tasks

kind: Service
metadata:
  name: //nama service
```

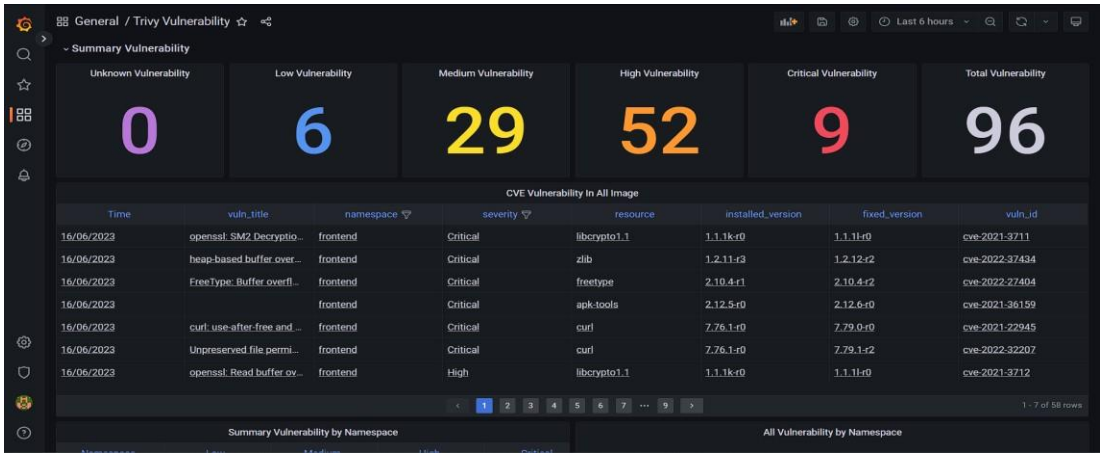
```
spec:
  selector:
    app: //nama app
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Gambar 3.21 File deployment aplikasi

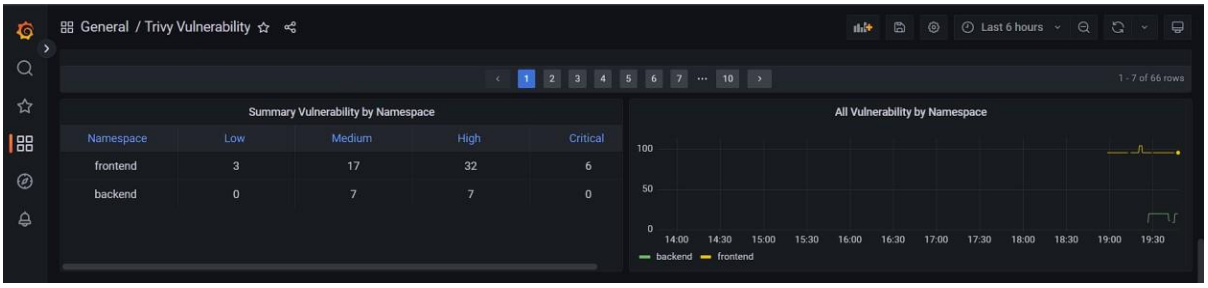
3.3 Hasil Proyek

3.3.1 Tampilan Dashboard

Tampilan dashboard telah terintegrasi dengan aplikasi *data source* dan secara garis besar tampilan terdiri dari dua bagian yaitu rangkuman celah keamanan dan celah keamanan berdasarkan *namespace*. Gambar 3.22 dan Gambar 3.23 merupakan tampilan rangkuman celah keamanan. Tampilan ini bertujuan untuk mengetahui jumlah keseluruhan celah keamanan berdasarkan tingkat suatu celah keamanan. Tampilan ini juga memberikan informasi mengenai detail celah keamanan, mulai dari nama celah keamanan, *namespace* tempat *container* berada, *resource* yang menjadi celah keamanan, versi yang diinstal dan versi yang sudah dapat pembaharuan. Tidak hanya itu terdapat juga *vulnerability id* yang dapat terhubung dengan halaman *database vulnerability* yang dimiliki oleh Aqua Security. Selanjutnya terdapat panel yang memberikan informasi mengenai jumlah celah keamanan berdasarkan *namespace* yang ada. Terakhir terdapat sebuah grafik untuk melakukan *monitoring* total celah keamanan berdasarkan *namespace.* Fungsi panel ini adalah agar pengguna dapat mengetahui apabila suatu waktu terdapat penurunan ataupun lonjakan celah keamanan yang terjadi pada suatu *namespace*.

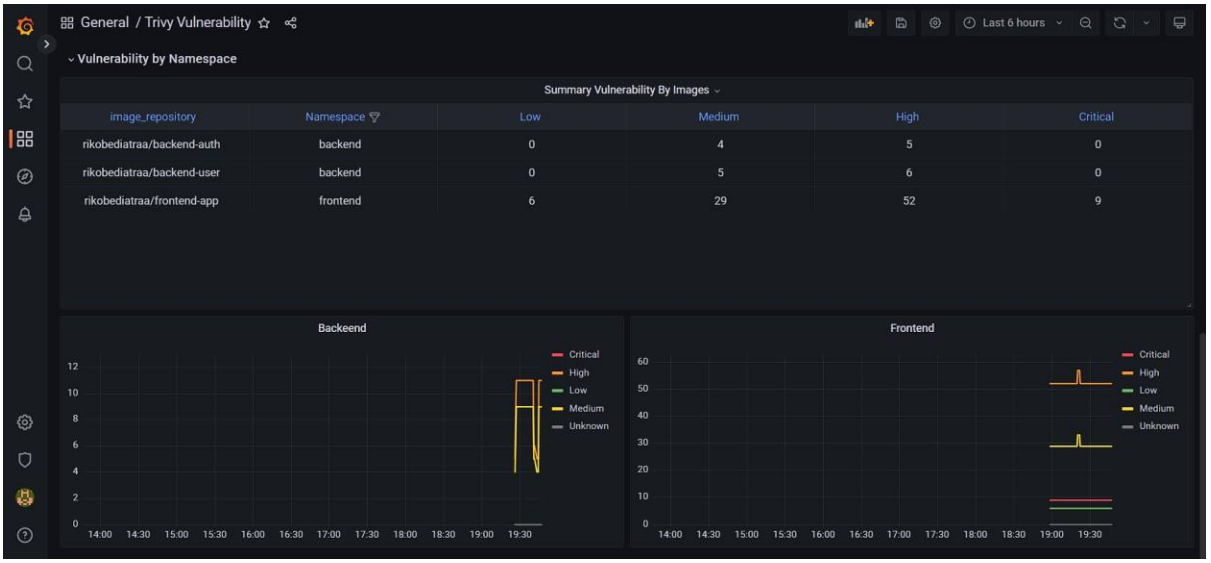


Gambar 3.22 Tampilan dashboard rangkuman celah keamanan (1)



Gambar 3.23 Tampilan dashboard rangkuman celah keamanan (2)

Selanjutnya terdapat bagian mengenai informasi celah keamanan berdasarkan *namespace* seperti pada Gambar 3.24. Pada bagian ini secara garis besar terdapat dua panel. Panel pertama berisi informasi mengenai jumlah celah keamanan berdasarkan tingkat celah keamanannya. Pada panel ini juga terdapat informasi mengenai *images repository* yang digunakan agar pengguna dapat mengetahui *images* mana yang memiliki tingkat celah keamanan paling tinggi. Selanjutnya terdapat panel berisi grafik yang menunjukkan jumlah keamanan pada setiap *namespace*. Grafik ini berisi lima garis yang masing – masing melambangkan jumlah celah keamanan menurut tingkatannya. Fungsi dari panel ini adalah untuk memberikan informasi kepada pengguna apabila di dalam suatu *namespace* terdapat lonjakan ataupun penurunan celah keamanan. Hal yang membedakan dari grafik pada rangkuman celah keamanan adalah panel ini memberikan informasi detail setiap tingkat celah keamanan menurut tingkatannya.



Gambar 3.24 Tampilan celah keamanan berdasarkan namespace

3.3.2 Hasil Pengujian Scanning Images

Pada saat telah berhasil melakukan *deployment*, aplikasi Trivy berhasil melakukan *scanning container images.* Tidak hanya itu konfigurasi yang telah dilakukan berhasil dilaksanakan dengan baik. Hal ini didapati pada bahwa *scanning* hanya dilakukan pada *namespace* yang telah ditentukan pada saat awal instalasi seperti pada Gambar 3.25. Dalam hal ini *scanning* hanya dilakukan pada *namespace frontend* dan *backend.* Dapat dilihat juga aplikasi *scanning* dapat dilakukan secara berkala sesuai dengan waktu yang telah ditentukan pada konfigurasi awal yaitu 15 menit. Dari Gambar 3.25 didapati juga bahwa setiap kali *scanning images* dilakukan akan terdapat jeda yang membuat hasil *scanning* tidak ada. Hal ini dikarenakan pada proses *scanning* ulang, laporan yang sebelumnya di dapatkan akan dihapus terlebih dahulu sebelum melakukan *scanning* lagi. Hal ini dilakukan agar laporan keamanan tidak terduplikasi dan membuat hasil laporan menjadi tidak akurat.

Gambar 3.25 Panel total scanning images

Selanjutnya setelah melakukan *deployment* didapatkan bahwa setiap *container images* memiliki jumlah *vulnerability* yang berbeda. Rangkuman jumlah *vulnerability* akan tampak seperti yang terdapat pada Gambar 3.26 dan Gambar 3.27. Terdapat hal yang menarik dari hasil yang didapat yaitu terdapat perbedaan antara jumlah keamanan yang terdapat pada *container* yang terdapat pada *namespace backend.* Dapat dilihat walaupun memiliki *base operating system* yang sama yaitu Node 14 Alpine, namun terdapat jumlah perbedaan pada celah keamanannya.

Summary Vulnerability by Namespace

Namespace	Low	Medium	High	Critical
frontend	3	17	32	6
backend	0	7	7	0

Gambar 3.26 Hasil celah keamanan berdasarkan namespace

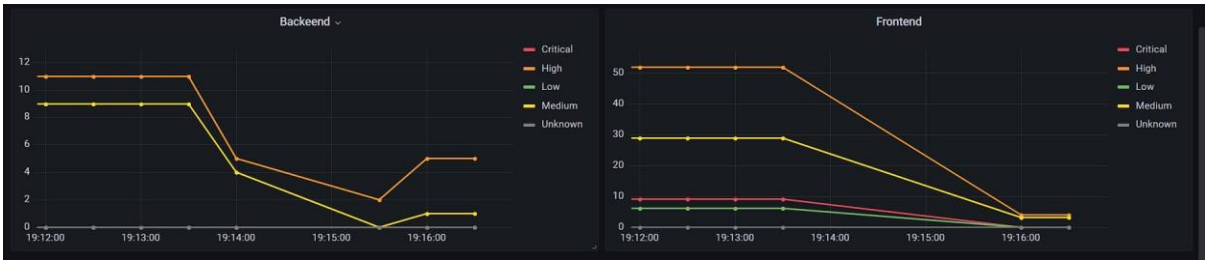
Gambar 3.27 Hasil celah keamanan berdasarkan *images* yang digunakan

Dilihat lebih lanjut bahwa perbedaan celah keamanan ini terdapat pada *operating system package* yang diinstal. *Package* yang membedakan terdapat pada *container images* backend-user yang menggunakan *package* axios seperti yang terdapat pada Tabel 3.3. Dari informasi ini didapatkan bahwa penggunaan *package* axios pada versi Node 14 terdapat celah keamanan yang memungkinan penyerang untuk melakukan *bypass* pada *proxy* dan memunculkan kemungkinan untuk melakukan serang DOS (*denial of service)*. Selain itu dari informasi ini juga menunjukan bahwa *scanning images* yang dilakukan tidak hanya pada *operating system* yang digunakan, namun meliputi juga *packages* yang diinstal pada saat proses sebuah *images* dibuat.

Tabel 3.3 Celah keamanan yang berbeda pada container backend-user

Severity	Kode CVE	Title
Medium	CVE-2020-28618	nodejs - axios: allows an attacker to bypass a proxy by providing a URL that responds with a redirect to a restricted host or IP address
High	CVE-2021-3749	nodejs - axios: Regular expression denial of service in trim function

Terakhir penulis melakukan uji coba dengan menaikkan versi *operating system* yang digunakan. Hal ini dikarenakan *packages* yang memiliki celah keamanan tersinstal bersamaan dengan OS yang digunakan, sehingga akan sulit apabila akan hanya melakukan *upgrade version* pada *package* yang memiliki celah keamanan saja. Setelah penulis menaikkan versi OS yang lebih baru terdapat penurunan celah keamanan seperti yang terdapat pada Gambar 3.28. Pada juga Gambar 3.29 dapat dilihat penurunan celah keamanan juga terjadi pada setiap *container images* yang telah dinaikkan versi OS.



Gambar 3.28 Grafik jumlah *vulnerability*

Summary Vulnerability By Images

image_repository	Namespace	Low	Medium	High	Critical
rikobediatraa/backend-auth	backend	0	0	2	0
rikobediatraa/backend-user	backend	0	1	3	0
rikobediatraa/frontend-app	frontend	0	3	4	0

Gambar 3.29 Jumalah celah keamanan berdasarkan container images

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Relevansi Akademik

Penulis selama delapan bulan melaksanakan magang di BSI UII mendapatkan banyak pengalaman dan pembelajaran dari sisi akademik. Tentunya hal ini merupakan hal yang baik bagi penulis. Terdapat beberapa pembelajaran terkait akademik yang didapatkan oleh penulis antara lain arsitektur kubernetes, penerapan Trivy sebagai *images scanning tools* dan pembuatan *dashboard.*

4.1.1 Arsitektur Microservice dan Kubernetes

Terjun secara langsung menjadi seorang DevOps (Development and Operation) yang secara langsung dapat mengakses sebuah *cluster orchestration* menjadi pengalaman baru yang sebelumnya belum pernah dilakukan pada bangku kuliah. Ruang lingkup yang harus dikuasai dalam arsitektur kubernetes sangat luas. Sebelumnya penulis perlu memahami terlebih dahulu konsep dari *microservices.* Dalam konsep ini penulis mempelajari bagaimana sebuah aplikasi dibagi menjadi bagian–bagian yang lebih kecil namun tetap dapat saling terhubung satu sama lain. Hal inilah yang membuat arsitektur *microservices* menjadi lebih fleksibel dikarenakan setiap komponen nya dapat berdiri sendiri. Tidak hanya itu, penulis juga mempelajari bahwa dengan menerapkan arsitektur *micorservices* dapat merespons perubahan teknologi ataupun proses bisnis yang lebih baik. Hal ini dikarenakan ketika terjadi suatu perubahan pada alur bisnis atau pengembangan teknologi, *developer* dapat melakukan perubahan hanya pada aplikasi, bagian aplikasi, ataupun fitur tertentu tanpa mengubah keseluruhan aplikasi.

Selanjutnya penulis mempelajari *tools* yang digunakan dalam melakukan orkestrasi sebuah *cluster.* Dalam proyek ini penulis menggunakan Minikube sebagai aplikasi untuk melakukan manajemen terhadap *cluster.* Dengan menggunakan Minikube penulis merasa lebih mudah dalam melakukan management *cluster*, hal ini dikarenakan Minikube memiliki fitur yang menyediakan sebuah tampilan (UI) dibandingkan aplikasi lainnya yang menggunakan terminal dalam melakukan manajemen sebuah *cluster.*

Tidak hanya *tools*, penulis juga mempelajari alur sebuah aplikasi dari *code* menjadi suatu aplikasi yang dapat diakses oleh pengguna. Ketika sebuah aplikasi telah selesai dibuat oleh *developer*, aplikasi tersebut harus diubah terlebih dahulu menjadi sebuah bentuk bernama

images. Selanjutnya perlu dibuat sebuah *file deployment* agar aplikasi tersebut dapat di-*deploy* pada suatu *cluster.* Langkah ini perlu dilakukan agar aplikasi dapat berjalan dengan baik pada sebuah *cluster* dan dapat diakses oleh pengguna.

4.1.2 Penerapan Trivy Sebagai *Container Images Scanning*

Penerapan Trivy sebagai *container images scanning* adalah inti dari proyek ini. Berbeda dengan penelitian yang dilakukan oleh Oktaviana dan Saxena yang di mana pada penelitian nya Trivy digunakan untuk melakukan scanning pada docker images, penulis melakukan implementasi Trivy pada *cluster* yang sedang berjalan. Pada proyek ini penulis berhasil untuk melakukan implementasi, sehingga aplikasi Trivy dapat melakukan *container scanning* pada sebuah cluster sesuai dengan konfigurasi yang diinginkan. Selain itu didapati bahwa aplikasi Trivy melakukan *scanning* pada *operating system* yang digunakan, *packages* yang diinstal, serta *dependency* yang digunakan saat sebuah *images* dibuat (*build images*).

Pada tahap implementasi penulis mendapati bahwa saat pertama kali aplikasi dijalankan pada sebuah cluster, secara otomatis akan terbuat *pods* yang berfungsi untuk melakukan *scanning* pada *deployment* yang ada. Scanning dapat dibatasi hanya pada *deployment* yang memiliki nama *namespace* yang sama. Hal ini akan membantu mengurangi pemakaian sumber daya seperti RAM dan CPU dikarenakan aplikasi tidak melakukan *scanning* pada keseluruhan *cluster*. Pada proses melakukan *scanning, pods* yang terbuat akan mengunduh data celah keamanan yang dimiliki oleh aplikasi *scanning*, dalam hal ini *database* yang digunakan berasal dari Aqua Security. Setelah berhasil diunduh *pods* akan secara otomatis menyamakan celah keamanan yang ada pada *database* dan hasil *scanning.* Selanjutnya apabila *pods* telah selesai melakukan tugasnya maka akan menghasilkan sebuah laporan celah keamanan. Secara otomatis *pods* yang telah selesai melakukan tugasnya akan menghancurkan diri nya sendiri namun tetap meninggalkan laporan celah keamanannya. Laporan ini nantinya dapat berbentuk rangkuman, JSON maupun *metrics* celah keamanan.

Setelah berhasil diimplementasikan dan dilakukan uji coba, didapati bahwa aplikasi Trivy mampu melakukan *scanning container images* pada *running cluster*. Informasi yang didapati adalah celah keamanan yang terdapat pada *base OS, package*, serta *dependency* pada *container images.* Tidak hanya itu Trivy juga mampu memberikan rekomendasi perbaikan versi untuk menanggulangi ancaman terhadap celah keamanan yang ada. Oleh karena itu dengan penerapan Trivy pada *Running Cluster* dapat menjadi salah satu cara untuk mengurangi celah keamanan pada sebuah aplikasi atau sistem berbasis *microservices*.

4.1.3 Pembuatan Dashboard Menggunakan Grafana

Pembuatan *dashboard* adalah hasil akhir dari seluruh proses yang sebelumnya telah dilaksanakan mulai dari pembelajaran arsitektur hingga implementasi Trivy sebagai *container images scanning. Dashboard* inilah yang nantinya akan digunakan oleh perusahaan untuk membantu dalam melakukan *monitoring* celah keamanan pada *cluster* kubernetes. Grafana dipilih sebagai aplikasi pembuatan *dashboard* dikarenakan Grafana mampu mengubah *data time series database* (DTSB) menjadi grafik dan visualisasi yang indah (Rahman et al., 2020). Grafana juga mampu terintegrasi dengan aplikasi Prometheus. Sehingga laporan celah keamanan yang dihasilkan dapat diubah dan diolah menjadi suatu *dashboard*. Dengan membuat visualisasi data dengan bentuk *dashboard* dapat lebih mudah dalam melakukan *monitoring* pada celah keamanan yang ada. Tidak hanya itu, Grafana juga dapat dijalankan pada *running cluster*.

Setelah penulis melakukan proyek ini, didapati bahwa Grafana tidak mendukung penggunaan *multiple cluster* Kubernetes. Hal ini dikarenakan apabila memiliki lebih dari satu *cluster* seperti *cluster production* dan *development*, pengembang harus melakukan instalasi aplikasi Grafana pada setiap *cluster*. Sehingga apabila membuat *dashboard* pada suatu *cluster* dan akan digunakan pada *cluster lain,* pengembang perlu melakukan *export* dalam bentuk JSON dan melakukan *import* pada *cluster* yang akan dituju.

Sebelumnya penulis belum pernah membuat *dashboard* yang bertema celah keamanan dan data yang digunakan berasal dari sebuah *metrics*, maka dari itu penulis dalam proses pembuatannya mendapatkan bantuan dari mentor dan mencari referensi dari internet. Mengelola data hingga akhirnya dapat ditampilkan pada sebuah *dashboard* merupakan pengalaman baru yang didapatkan dalam kegiatan magang ini. Dalam proses nya, penulis mempelajari bahwa tidak semua data perlu ditampilkan pada sebuah *dashboard.* Untuk membuat sebuah *dashboard* perlu diketahui dahulu siapa yang akan menggunakan *dashboard* tersebut. Setelah mengetahui penggunanya, diperlukan komunikasi secara aktif untuk mengetahui data apa saja yang diperlukan oleh pengguna.

4.2 Pembelajaran Magang

Pada proses kegiatan magang ini banyak sekali pengalaman yang baru pertama kali penulis rasakan. Tidak hanya pengalaman pertama dalam magang namun juga menjadi pengalaman pertama penulis terjun langsung pada proyek yang berkaitan dengan *cyber*

security. Pada proyek ini penulis mempelajari bagaimana sebuah arsitektur kubernetes diterapkan serta *tools* yang digunakan dalam melakukan orkestrasi. Selain itu penulis juga mempelajari bahwa banyak cara yang dapat dilakukan untuk mencegah kejahatan *cyber* yaitu salah satunya dengan menerapkan *images scanning.* Berbeda dengan *code scanning*, *images scanning* melakukan *scanning* pada *base OS, packages,* serta *dependency* yang digunakan sebuah *images* ataupun *container images.* Dalam penjalanan proyek penulis mempelajari bahwa banyak *tools* yang dapat digunakan untuk melakukan *images scanning*, akan tetapi dalam proyek ini penulis menggunakan Trivy sebagai aplikasi *container images scanning.*

Pada awal kegiatan *planning*, harapan *mentor* dengan implementasi *container images scanning* adalah dapat mengurangi biaya *images scanning* yang harus dikeluarkan dari penggunaan Google Cloud Registry (GCR). Hal ini dapat dilakukan apabila telah melakukan analisis serta membandingkan akurasi celah keamanan yang dihasilkan oleh Trivy dengan hasil yang didapat dari GCR. Namun penulis mendapatkan kendala dikarenakan kurangnya pengalaman yang dimiliki dalam metode *images scanning* dan *cyber security,* sehingga hasil proyek hanya sampai implementasi pada *cluster development*.

Pada saat pelaksanaan proyek, penulis ikut merasakan penerapan pengembangan aplikasi secara *agile.* Di mana setiap hari penulis dituntut untuk dapat mengikuti kegiatan *daily meeting.* Dari kegiatan ini penulis belajar untuk berkomunikasi secara aktif kepada seluruh anggota tim. Penulis juga mempelajari pentingnya komunikasi secara efektif, dikarenakan komunikasi yang efektif sangat membantu apabila terdapat kendala ketika penjalanan proyek berlangsung. Selain itu dengan penerapan metode ini juga penulis belajar untuk bekerja secara fleksibel dan menyesuaikan dengan kebutuhan yang ada. Hal ini didasari karena pengembangan *agile* dapat dengan mudah beradaptasi dengan kebutuhan dari proses bisnis, sehingga ini juga menyebabkan para *developer* harus dapat mengimbangi fleksibilitas dari metode ini.

Terakhir, penulis belajar untuk meningkatkan manajemen diri serta memperluas pandangan tentang dunia kerja. Hal ini dirasakan penulis ketika melakukan kegiatan magang diiringi dengan mata kuliah yang masih diambil di kampus. Penulis dituntut untuk membagi waktu antara kuliah, magang dan istirahat. Hal ini bertujuan agar baik pada saat magang maupun kuliah, penulis dapat melaksanakan dengan sebaik – baiknya.

BAB V
PENUTUP

5.1 Kesimpulan

Serangkaian proses telah berhasil dilakukan mulai dari implementasi aplikasi, pengolahan data, hingga pembuatan *dashboard monitoring*. *Dashboard monitoring* celah keamanan berhasil dibuat dengan menampilkan dua bagian yaitu rangkuman celah keamanan dan celah keamanan berdasarkan *namespace*. Dari *dashboard* yang dibuat dapat diketahui detail celah keamanan, mulai dari nama celah keamanan, *namespace* tempat *container* berada, *resource* yang menjadi celah keamanan, versi yang diinstal dan versi yang sudah dapat pembaharuan. Selain itu terdapat sebuah grafik untuk melakukan *monitoring* total celah keamanan berdasarkan *namespace.* Dalam melakukan implementasi aplikasi sangat penting untuk menentukan *resources* yang digunakan, *namespace* yang dipilih serta *behaviour* dari aplikasi *container images scanning*. Hal ini nantinya akan berpengaruh terhadap laporan yang dihasilkan.

Implementasi *container images scanning* cocok diterapkan pada *cluster* kubernetes guna memperkuat keamanan sebuah *cluster.* Hal ini dikarenakan *scanning* dapat dilakukan pada *deployment* yang sedang berjalan atau digunakan. Selain itu hasil *scanning* memberikan laporan mengenai celah keamanan apa saja yang terdapat pada sebuah *running cluster* serta rekomendasi perbaikannya. Tidak hanya itu, penggunaan Trivy sebagai *container images scanning* tidak dipungut biaya apapun. Hal ini dikarenakan aplikasi Trivy bersifat *opensource* sehingga dapat diimplementasikan tanpa biaya yang harus dikeluarkan. Namun terdapat kekurangan pada aplikasi yang digunakan untuk *scanning container images* yaitu aplikasi ini menggunakan *single database* yang diambil dari Aquasec. Sehingga *update* dan akurasi celah keamanan sangat bergantung pada *database* yang disediakan.

5.2 Saran

Adapun saran bagi BSI UII adalah dapat menerapkan *container scanning images* pada *cluster production* sebagai salah satu upaya untuk mengurangi celah keamanan. Bagi pelaksanaan magang selanjutnya adalah dapat melakukan analisis dan perbandingan dengan aplikasi *container images scanning* yang lain. Dengan melakukan implementasi dan membandingkan hasil *images scanning* dapat mengetahui tingkat akurasi dari laporan celah keamanan yang dihasilkan. Tidak hanya itu pelaksanaan magang selanjutnya dapat juga

melakukan *container scanning* yang tidak terbatas hanya pada *deployment* saja namun dapat meliputi *role base access control (RBAC), exposed secret* dan kesalahan pada *configuration.*

DAFTAR PUSTAKA

Awanda Alviansyah, F., & Ramadhani, E. (2021). *Implementasi Dynamic Application Security Testing pada Aplikasi Berbasis Android*.

Burns, B., Beda, J., Hightower, K., & Evenson, L. (2022). *Kubernetes: Up and Running*. O'Reilly Media. https://books.google.co.id/books?id=KeB-EAAAQBAJ

Chen, L., Xia, Y., Ma, Z., Zhao, R., Wang, Y., Liu, Y., Sun, W., & Xue, Z. (2022). SEAF: A Scalable, Efficient, and Application-independent Framework for container security detection. *Journal of Information Security and Applications*, *71*, 103351. https://doi.org/https://doi.org/10.1016/j.jisa.2022.103351

Grebić, B., & Stojanović, A. (2021). Application of the Scrum Framework on Projects in IT Sector. *European Project Management Journal*, *11*(2), 37–46. https://doi.org/10.18485/epmj.2021.11.2.4

Haris, A. I., Ferianda, Rd. A., Riyanto, B., Nugraha, F. I., & Abadi, J. (2020). PENGAMANAN CONTAINER ORCHESTRATION BERBASIS KUBERNETES DI LEMBAGA PENERBANGAN DAN ANTARIKSA NASIONAL (LAPAN). *Jurnal Teknoinfo*, *14*(1), 1. https://doi.org/10.33365/jti.v14i1.501

Jaisinghani, G. (2022). Vulnerability Management in the Age of Containers. *International Journal of Information Security (IJIS)*, *1*(1), 1–5. https://doi.org/10.17605/OSF.IO/WJSV6

Javed, O., & Toor, S. (2021). *Understanding the Quality of Container Security Vulnerability Detection Tools*. http://arxiv.org/abs/2101.03844

Jha, P., & Khan, R. (2018). A Review Paper on DevOps: Beginning and More To Know. *International Journal of Computer Applications*, *180*(48), 16–20. https://doi.org/10.5120/ijca2018917253

Kaur, B., Dugré, M., Hanna, A., & Glatard, T. (2021). An analysis of security vulnerabilities in container images for scientific data analysis. *GigaScience*, *10*(6). https://doi.org/10.1093/gigascience/giab025

Kurniawan, R. W. (2020). Implementasi Mutual Transport Layer Security (mTLS) Pada Arsitektur Microservices Dengan Istio Di Kubernetes.

Mutiara, R., Politeknik, F., & Jakarta, N. (2020). Implementasi Sistem Monitoring Menggunakan Prometheus dan Grafana . https://www.researchgate.net/publication/342511231

Newman, S. (2021). *Building Microservices* (2nd ed.). O'Reilly Media, Inc.

Oktaviana, M., Widjajarto, A., & Almaarif, A. (2022). Analisis Vulnerability Management Pada Container Docker Menggunakan Opensource Scanner Berdasarkan Standar Cyber Resilience Review (CRR). *Jurnal Sistem Komputer Dan Informatika (JSON)*, *4*(1), 77. https://doi.org/10.30865/json.v4i1.4787

Putra, R. A. (2019). Analisa Implementasi Arsitektur Microservices Berbasis Kontainer Pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (OPENDAYLIGHT DEVOPS COMMUNITY). *Sistem Informasi, Teknologi Informasi Dan Komputer (Just IT)*, *9*(2), 150–162. https://jurnal.umj.ac.id

Rahman, D., Amnur, H., & Rahmayuni, I. (2020). *Monitoring Server dengan Prometheus dan Grafana serta Notifikasi Telegram* (Vol. 1, Issue 4). https://doi.org/https://doi.org/10.30630/jitsi.1.4.19

Rizki Perkasa, P., & Mailoa, E. (2023). AGILE MENGGUNAKAN TRIVY SEBAGAI SECURITY SCANNER DOCKER IMAGE DAN DOCKERFILE. *Jurnal Indonesia : Manajemen Informatika Dan Komunikasi*, *4*, 856–863. https://doi.org/10.35870/jimik.v4i3.291

Saxena, P. (2022). *Container Image Security with Trivy and Istio Inter-Service Secure Communication in Kubernetes MSc Research Project Cloud Computing*.

Sekilas BSI - BSI UII. (n.d.). Retrieved May 12, 2023, from https://bsi.uii.ac.id/sekilas-bsi/

Wahyu, M., Santosa, I., Primananda, R., & Yahya, W. (2018). *Implementasi Load Balancing Server Basis Data Pada Virtualisasi Berbasis Kontainer* (Vol. 2, Issue 12). http://j-ptiik.ub.ac.id

LAMPIRAN