Informatics, Science, and Technologies Journal
Vol 12 No 2 (2022) **Jurnal GERBANG STMIK Bani Saleh**
ISSN : 0853 – 6376
E-ISSN : 2622 – 3708

OPTIMALISASI PENGGUNAAN CPU CLUSTER RELAY SERVER EMAIL BLAST DENGAN KUBERNETES POD AUTOSCALE

Domo Pranowo K.[1] , M. Raihan Utomo[2], Mumtaz Muttakin[3]
[1]Domo Pranowo K., Teknik Komputer, STMIK Bani Saleh, domopranowo@stmik.banisaleh.ac.id
[2]M. Raihan Utomo, Teknik Informatika, STMIK Bani Saleh, mohammad06180047@stmik-banisaleh.ac.id
[3]Mumtaz Muttakin, Teknik Informatika, STMIK Bani Saleh, mumtazmuttakin@gmail.com

Abstrak

Saat ini salah satu media yang paling baik digunakan untuk menyebarkan informasi adalah *email*. Saat mendaftar Aplikasi atau sosial media, pasti akan ditanyakan dan diminta mengisi alamat email, dengan demikian penggunaan media *email* masih menjadi pilihan yang tepat untuk melakukan komunikasi dengan pelanggan seperti pemberitahuan promo atau pengumuman penting lainnya. Namun pengiriman *email* dalam jumlah banyak muncul beberapa permasalahan diantaranya : pengiriman *email* yang tidak sampai tepat waktu, proses penambahan *server* membutuhkan waktu lama, dan yang utama penggunaan *resource CPU* yang tidak optimal. Penelitian ini dilakukan untuk mendapatkan perancangan sistem yang bisa menyelesaikan permasalahan-permasalahan diatas. Pengumpulan data yang dalam penelitian ini menggunakan studi litertur dan melakukan pengujian system langsung menggunakan metode *blackbox testing*. Data yang didapatkan pada penelitian akan digunakan sebagai bahan analisis sistem *cluster relay server email blast* dengan *Kubernetes pod autoscale*. Adapun masalah yang timbul pada sistem tinjauan, waktu pengiriman *email blast* yang tidak cepat, lamanya proses penambahan *server*, penggunaan *resource CPU* kurang optimal. Dari masalah tersebut hasil akhir dari penelitian ini, penggunaan *software* pengelola resource berbasis *container* mampu mengoptimalkan mempercepat konfigurasi dan penggunaan CPU, dengan menerapkan *autoscale pod.* kemampuan *autoscale* juga mampu menambahkan *pod container* secara otomatis berdasarkan penggunaan *resource CPU*, waktu pengiriman *email* yang bisa lebih cepat dikarenakan sistem *cluster relay server email blast* menggunakan pertumbuhan *resource* CPU menyesuaikan besarnya email yan di kirim.

Kata Kunci: *Email, Mail Transfer Agent, Kubernetes, Relay, Cluster*

PENDAHULUAN

Pada zaman digital seperti sekarang ini, banyak sekali kebutuhan untuk mengirimkan sebuah informasi kepada banyak orang. Informasi bisa berupa transaksi pribadi seperti tagihan, informasi umum seperti iklan, promo, tawaran produk dan sebagainya. Media yang digunakan untuk menyebarkan informasi bisa dilakukan melalui aplikasi sosial media, *website*, dan juga *email*.

Saat ini email menjadi salah satu menjadi account yang pasti dimiliki pengguna saat menggunakan internet sampai digunakan untuk melakukan registrasi diaplikasi online ataupun layanan offline. Dan cukup efektif untuk menyebarkan atau menyampaikan informasi. Dengan jumlah layanan bersifat dinamis atau fluktuatif, dalam penelitian ini perlu alat bantu aplikasi *relay email blast*. Sebagai gambaran pada hari-hari biasa, mungkin *demand* kiriman email relatif normal hanya rata-rata 100-200 *email* per-jam, namun pada saat hari-hari libur besar tentu saja demand kiriman email akan meningkat drastis bahkan bisa mencapai ratusan ribu email per-jam, dengan jenis kiriman email blast berisi pemberitahuan, promo ataupun informasi lainnya yang ditujukan kepada para pelanggan. Aplikasi ini membutuhkan sebuah infrastruktur yang fleksibel memiliki kemampuan bekerja secara otomatis untuk mengatur konfigurasi yang cepat dan kebutuhan sumber daya CPU. Salah satu pilihan untuk

memenuhi kebutuhan tersebut adalah dengan menggunakan metode *scaling container*.

Menurut (Premanantha 2021) dalam jurnalnya yang berjudul *Improve Horizontal Pod Autoscaling in Container Orchestration to Adopt Frequent Oscillation in Service Requests* menyatakan bahwa "*A flexible infrastructure is required for applications with dynamic workloads to leverage performance measures and minimize resource costs*", lalu dalam jurnal lain dengan berjudul *Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration* yang ditulis oleh (Nguyen et al. 2020) menyatakan bahwa "*In Kubernetes, one of the most important features is autoscaling because it allows containerized applications and services to run resiliently without the necessity of human intervention*".

Berdasarkan latar belakang dan referensi di atas untuk optimalisasi layanan *email blast*, maka penulis akan melakukan penelitian dengan judul " OPTIMALISASI PENGGUNAAN CPU CLUSTER RELAY SERVER EMAIL BLAST DENGAN KUBERNETES POD AUTOSCALE ".

METODA

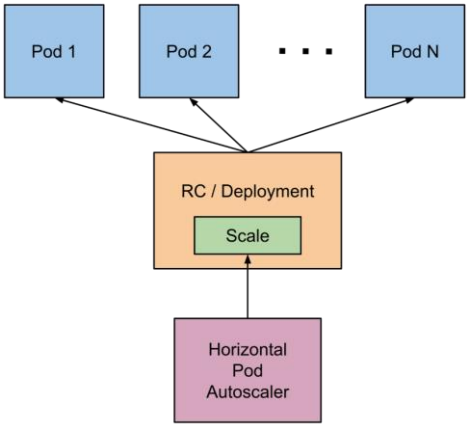
Berdasarkan kutipan (Sumbogo, Data, and Siregar 2018) dari jurnalnya yang berjudul "IMPLEMENTASI *FAILOVER* DAN *AUTOSCALING* KONTAINER *WEB SERVER NGINX* PADA *DOCKER* MENGGUNAKAN *KUBERNETES*" menyatakan bahwa "Terdapat pula sebuah metode *autoscaling* dimana sistem akan *memonitoring* sebuah *server cluster* secara berkala dimana ketika sistem mendeteksi layanan pada sebuah *server* tidak dapat memenuhi permintaan pengguna maka sistem akan menambahkan layanan tersebut ke *server* lain yang tersedia". Lalu jurnal yang sama juga menyatakan "*Scaling* merupakan kemampuan sistem untuk menyesuaikan sumber daya yang dimiliki seperti menurunkan atau menambah jumlah proses sesuai kebutuhan sistem tanpa mengganggu proses yang sedang berjalan". Dari penyataaan yang ada di jurnal tersebut, *autoscale* dirasa bisa secara fleksibel menambahkan *server* dalam sebuah *cluster* atau bahkan menguranginya, sesuai dengan kebutuhan layanan, dalam penelitian ini misalnya jumlah kiriman *email blast*-lah yang akan menentukan seberapa banyak *cluster* membutuhkan *server* tambahan atau seberapa banyak *server* yang bisa dikurangi untuk menghemat *resource* yang ada.

Menurut (Rodriguez and Buyya 2020) dari jurnalnya yang berjudul "*CONTAINERS ORCHESTRATION WITH COST-EFFICIENT AUTOSCALING IN CLOUD COMPUTING ENVIRONMENTS*" dalam bahasa inggris menyatakan bahwa "*Autoscaling: This problem is divided into two sub-problems: i) scaling out and ii) scaling in. On one hand, if after attempting to reschedule there is still an unschedulable task, the autoscaler should consider scaling out (provisioning a new VM) in order to increase the cluster's capacity to deploy the task. If there are unused VMs or the cluster resources are being utilized inefficiently, the autoscaler should consider shutting down unused VMs or consolidating tasks to increase the utilization of the cluster and deprovision unnecessary VMs"*. Dari pernyataan tersebut diketahui bahwa, *autoscale* bisa bersifat 2 arah, yaitu *scaling out* dan *scaling in*. *Scaling out* dilakukan pada saat kebutuhan layanan meningkat dan membutuhkan *server* tambahan, sedangkan *scaling in* dilakukan pada saat kebutuhan layanan lebih sedikit dari kondisi biasanya sehingga beberapa *server* tidak dibutuhkan

Horizontal Pod Autoscaling (HPA)

Dari (Nguyen et al. 2020) dalam jurnalnya yang berjudul *Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration*, "*In Kubernetes, HPA is a powerful feature that automatically raises the number of pods, to increase the application's overall computational and processing power, without having to stop the application's currently running instances. Once successfully created, these new pods are able to share the incoming load with the existing ones*"

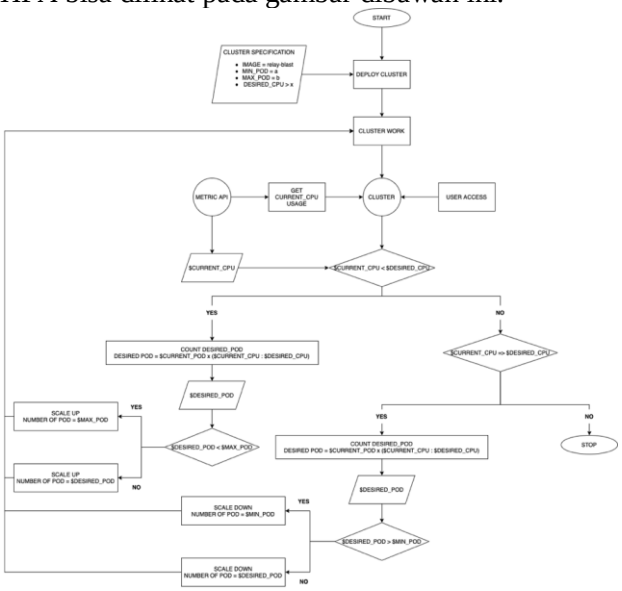
Horizontal Pod Autoscaling (*HPA*) merupakan kemampuan *Kubernetes* untuk menambahkan ataupun mengurangi jumlah *pod* di dalam sebuah cluster sesuai dengan penggunaan *resource cluster* atau gabungan *resource* semua *host* yang ada di *cluster*, tanpa perlu melakukan *stop* atau *restart cluster*. Setelah proses penambahan (*Scaling up*) ataupun pengurangan (*Scaling down*) oleh *HPA* beban layanan akan didistribusikan secara seimbang ke sejumlah *pod* yang ada di dalam *cluster*. Proses kerja *HPA* bisa dilihat pada gambar dibawah ini.



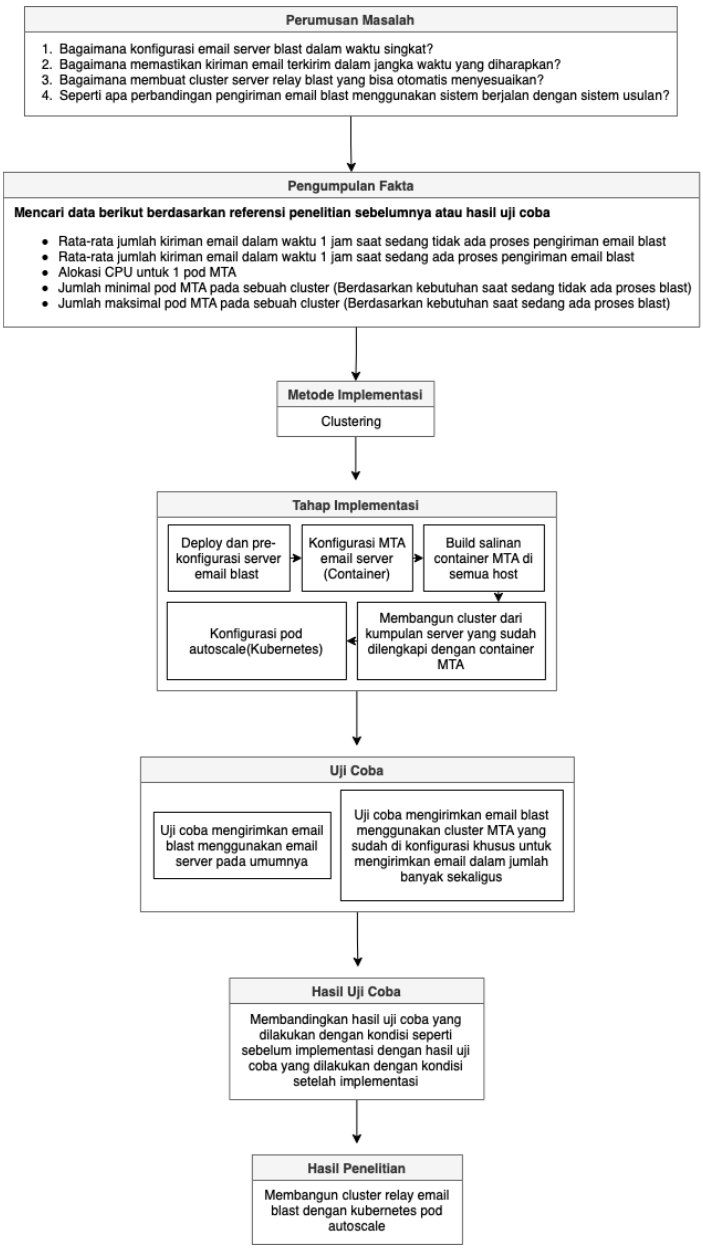
Gambar 1 Proses Kerja HPA

Sumber Gambar:
https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Sepeti apa algoritma penentuan jumlah *pod* yang dibutuhkan? Penentuan jumlah *pod* yang dibutuhkan dilakukan oleh *HPA* dengan cara membaca *metric* total penggunaan *resource CPU* atau *RAM* cluster, apabila total penggunaan *resource CPU* atau *RAM* cluster mencapai ataupun melewati batas yang ditentukan maka *HPA* akan otomatis melakukan *scale up* terhadap jumlah *container* yang ada, namun apabila total penggunaan *resource CPU* atau *RAM* cluster kurang dari batas yang ditentukan, maka *HPA* akan otomatis melakukan *scale down*. Algoritma *scaling HPA* bisa dilihat pada gambar dibawah ini.



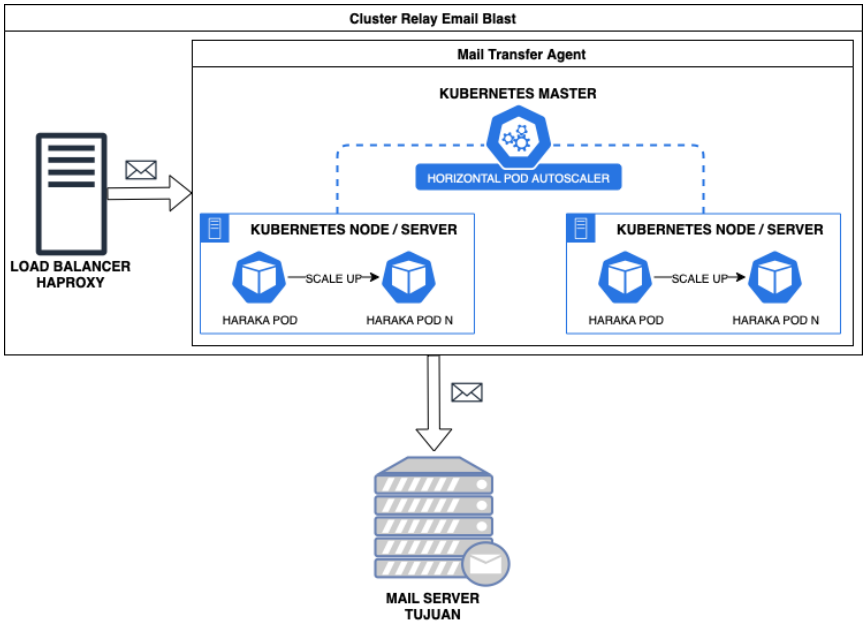
Gambar 2 Algoritma *Scaling HPA*



Topologi Sistem Usulan

Penulis sudah merancang topologi sistem usulan yang akan digunakan sebagai alternatif untuk mengurangi kekurangan yang ada. Berikut adalah rancangan topologi sistem usulan, lebih tepatnya topologi untuk komponen *cluster relay email blast*:

Informatics, Science, and Technologies Journal
Vol 12 No 2 (2022) **Jurnal GERBANG STMIK Bani Saleh**
ISSN : 0853 – 6376
E-ISSN : 2622 – 3708

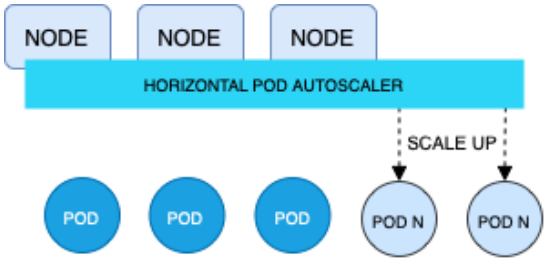


Gambar 3. Topologi Sistem Usulan

Berikut adalah penjelasan terkait dengan alur pada topologi sistem usulan *cluster relay email blast* yang ada di atas:

1. *Traffic* pengiriman *email blast* dari klien akan masuk ke *server load balancer HAProxy*.
2. Dari *server load balancer HAProxy email blast* akan disebarkan ke beberapa *pod mail transfer agent Haraka* dengan detail sebagai berikut:

a. *Pod* merupakan bagian terkecil dari *Kubernetes cluster*, lalu di dalam *pod* terdapat *container Haraka*.
b. *Pod* berjalan di atas *Kubernetes node*.
c. *Kubernetes node* dikendalikan oleh *Kubernetes master* yang memiliki tugas untuk melakukan *deployment pod Haraka* serta memastikan *pod* tersebut dalam kondisi *running*.
d. Selain itu, *Kubernetes master* juga dilengkapi dengan *horizontal pod autoscaler* yang memiliki kemampuan untuk menambahkan *pod (n)* apabila penggunaan *resource pod* meningkat dan melewati batas persentase yang ditentukan. Meningkatnya penggunaan *resource pod* diperkirakan akan berbanding lurus dengan jumlah pengiriman *email blast* dalam jangka waktu tertentu. Untuk memperjelas, berikut adalah topologi atau gambaran proses yang dilakukan oleh *horizontal pod autoscaler* apabila penggunaan *resource pod* melewati batas persentase yang ditentukan.



Gambar 4. Topologi *HPA* Saat Penggunaan *Resource Pod* Meningkat

3. Dari *pod mail transfer agent Haraka*, *email blast* akan langsung dikirimkan ke *email server* tujuan.

Kebutuhan Hardware :

1. 3 VPS cloud dengan spesifikasi sebagai berikut:
   a. *vCPU* = 2
   b. *Memory (RAM)* = 4 *GB*
   c. *Space HDD* = 80 *GB*
2. 1 VPS cloud dengan spesifikasi sebagai berikut:
   a. *vCPU* = 1
   b. *Memory (RAM)* = 2 *GB*
   c. *Space HDD* = 50 *GB*
3. 1 VPS cloud dengan spesifikasi sebagai berikut:
   a. *vCPU* = 1

b. *Memory (RAM)* = 1 *GB*
c. *Space HDD* = 25 *GB*

Kebutuhan Software :
1. Linux Debian
2. Docker
3. Kubernetes
4. Haraka (Mail)
5. HAProxy
6. Mailhog

HASIL DAN PEMBAHASAN

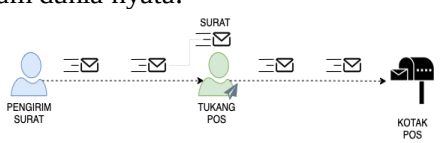
Implementasi Sistem

Pada tahap implementasi sistem, penulis melakukan penerapan beberapa perangkat lunak atau *software* sesuai dengan sistem usulan yang direncanakan. Setelah melakukan implementasi, sistem usulan akan diuji dan dibandingkan dengan sistem yang berjalan saat ini sehingga bisa didapatkan hasil penelitian yang diharapkan. Implementasi mencakup peran dan tugas *Haraka* sebagai *mail transfer agent (MTA)* berbasis *container*, *Kubernetes* sebagai infrastruktur sistem usulan, sampai dengan cara kerja *horizontal pod autoscaling (HPA)* pada *Kubernetes*.

Mail Transfer Agent Haraka

Pada tahap awal implementasi penulis telah melakukan pembuatan serta konfigurasi aplikasi *Haraka* sesuai dengan kebutuhan pada penelitian kali ini, yaitu untuk mengirimkan *email blast*. Dikarenakan fungsinya untuk mengirimkan *email* dalam jumlah banyak sekaligus, maka *Haraka* tidak dilengkapi dengan tambahan *plugin* seperti *antispam* ataupun *antivirus* supaya tidak menyebabkan *delay* proses pengiriman yang mungkin akan disebabkan oleh proses *scanning antispam antivirus*.

Haraka berperan sebagai *transfer agent* atau *relay agent* atau pengantar atau kurir atau tukang pos, dengan *email* sebagai objeknya. Fungsi *Haraka* hanya mengirimkan *email* dari pengirim ke penerima tanpa menyimpannya. Untuk memperjelas, berikut adalah gambaran peran *mail transfer agent Haraka* menggunakan analogi surat-menyurat dalam dunia nyata:

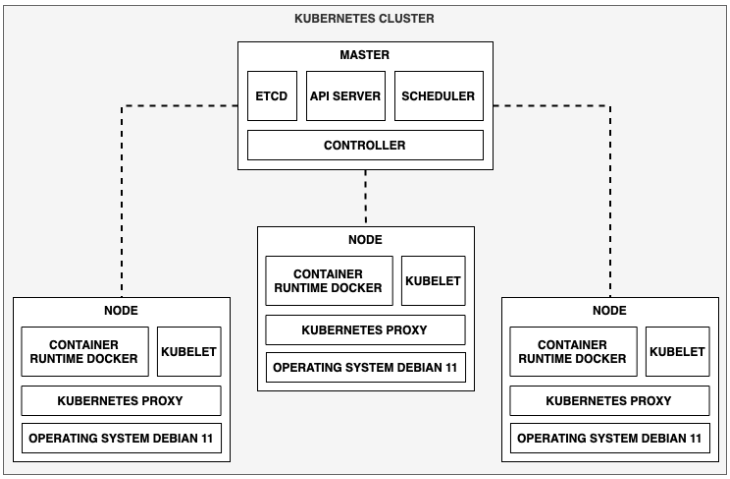


Gambar 4 Analogi Surat Menyurat

Dari analogi di atas, berikut adalah persamaannya dengan *mail transfer agent Haraka*:
1. Surat = *Email*
2. Pengirim Surat = Pengirim *Email*
3. Tukang Pos = *Mail Transfer Agent Haraka*
4. Kotak Pos = *Email server* penerima

Selain itu, aplikasi *Haraka* yang disiapkan oleh penulis juga sudah berbentuk *image container* yang dapat berjalan di atas *container runtime Docker*. Media *container* dipilih supaya proses konfigurasi pada *server* atau *node* baru bisa dilakukan secara cepat, hanya dengan *redeploy container* menggunakan *image* yang sudah di-*upload* ke *private registry*.

Infrastruktur *Kubernetes cluster* yang digunakan pada sistem usulan terdiri dari 3 server fisik atau *node* atau host dengan spesifikasi yang identik sehingga jika total spesifikasi *cluster*-nya memiliki *resource vCPU* sebanyak 6, *RAM* sebesar 12 *GB*, serta *space HDD* sebesar 240 *GB*. Selain itu, *Kubernetes cluster* juga dilengkapi dengan beberapa komponen yang akan menunjang kegiatan manajemen dan *orchestration container*. Berikut adalah gambaran terhadap komponen pada



Kubernetes cluster:

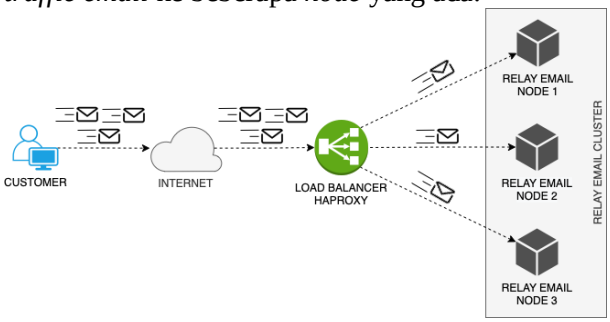
Gambar 5 Komponen *Kubernetes Cluster*

Load Balancer HAProxy

Sistem usulan yang menggunakan mekanisme beberapa server sekaligus untuk menangani suatu layanan *email blast* tentu saja akan membutuhkan *tools* yang akan menjadi gerbang utama *traffic email* dari *Customer* dan harus mampu mendistribusikan *traffic* tersebut ke beberapa *node (Load Balancer)*. Aplikasi *load balancer* yang penulis gunakan dalam sistem usulan adalah

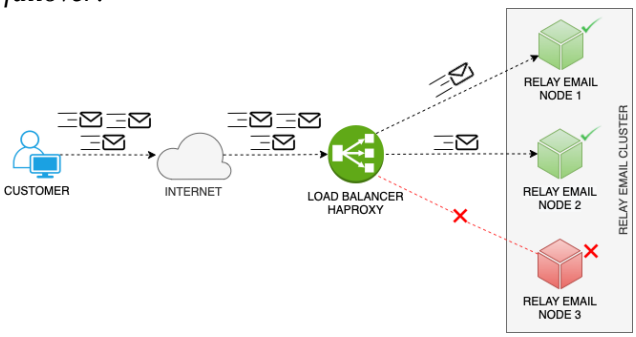
Informatics, Science, and Technologies Journal
Vol 12 No 2 (2022) **Jurnal GERBANG STMIK Bani Saleh**
ISSN : 0853 – 6376
E-ISSN : 2622 – 3708

HAProxy Load Balancer yang juga disediakan oleh *cloud provider Linode* karena *HAProxy* merupakan salah satu yang paling umum digunakan pada sistem operasi *Linux*. Berikut adalah gambaran *load balancer HAProxy* dalam melakukan distribusi *traffic email* ke beberapa *node* yang ada:



Gambar 6 Alur *Load Balancer HAProxy*

HAProxy bisa difungsikan juga sebagai *failover* untuk mengantisipasi apabila ada salah satu *node* yang *down* secara mendadak (*Unplanned downtime*) ataupun *down* dikarenakan ada perbaikan (*Planned downtime*). *HAProxy* akan otomatis mengalihkan *traffic* dari *node* yang kondisinya *down* ke *node* yang kondisinya normal *(Failover)*. Berikut adalah gambaran *saat HAProxy* melakukan *failover*:



Gambar 7 Alur Saat *Failover*

1.1.1 *Horizontal Pod Autoscaling (HPA)*

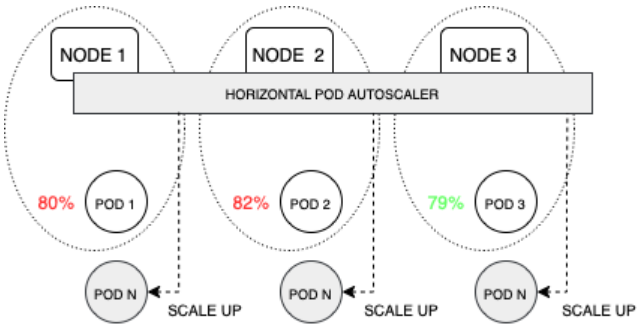
Sistem usulan yang dirancang oleh penulis memiliki kemampuan untuk melakukan *scaling* atau *expanding* secara otomatis supaya ukuran *cluster* bisa menyesuaikan sesuai dengan kebutuhan *request relay email* dari *Customer*. *Tools* yang digunakan untuk memenuhi kebutuhan tersebut adalah *Horizontal Pod Autoscaling (HPA)* yang secara *default* sudah ada pada *Kubernetes cluster*.

Proses *auto scaling* pada sistem usulan dipicu oleh *metric* penggunaan *CPU pod* yang akan meningkat apabila ada pengiriman *email* dalam jumlah banyak secara sekaligus ataupun dalam jangka waktu tertentu. Walaupun demikian, proses *auto scaling* tidak langsung dilakukan setiap ada peningkatan pengunaan *CPU*, namun hanya pada saat penggunaan *CPU* melewati persentase yang ditentukan oleh penulis. Pada sistem usulan, penulis menentukan batas atau limitasi penggunaan *CPU* dengan nilai 80%, sehingga *HPA* hanya akan melakukan *scaling up* pada saat penggunaan *CPU* salah satu *pod* mencapai atau melewati persentase *CPU* dengan nilai 80%. Nilai persentase 80% dipilih berdasarkan informasi dari forum diskusi pada*serverfault* (https://serverfault.com/questions/692034/whats-the-advisable-maximum-cpu-usage) serta pengalaman penulis dalam menggunakan aplikasi *email server* berbasis *Zimbra*.

Tabel 1 Contoh Skenario *HPA*

Horizontal Pod Autoscaling (HPA)			
Pod	***Node***	***CPU Usage***	***Trigger Autoscaling***
Pod 1	*Relay email node 1*	80%	*Yes*
Pod 2	*Relay email node 2*	82%	*Yes*
Pod 3	*Relay email node 3*	79%	*No*
Pod from Autoscaling Process			
Pod n	*Relay email node 1/2/3*	*N/A*	

Untuk mempermudah pemahaman cara kerja *HPA*, penulis akan memberikan contoh gambaran atau alurnya. Misalnya *cluster* terdiri dari total 3 *node* yang di dalamnya terdapat 1 *pod* masing-masing sehingga *cluster* tersebut memiliki *3 pod* yang menangani layanan yang sama, dalam hal ini layanan *email* dengan batas persentase penggunaan *CPU* pada setiap *pod* sebesar 80%. Pada saat tertentu, *pod* 1 mencapai persentase 80%, lalu *pod 2* mencapai persentase 82%, dan *pod 3* mencapai persentasi 79%, maka *HPA* akan melakukan *scale up pod* dengan jumlah tertentu (*Pod n*) yang dipicu dari penggunaan *CPU pod 1* yang mencapai 80% ataupun pengunaan *CPU pod 2* yang mencapai 100%.



Gambar 1 Alur Pada Skenario HPA

Proses *scale up* akan terus dilakukan sampai persentase penggunaan *CPU* semua *pod* dibawah limtasi yang ditentukan atau sampai memenuhi jumlah *pod* yang ideal. Berikut adalah rumus *HPA* dalam menentukan berapa jumlah *pod* yang ideal berdasarkan penggunaan *CPU pod*:

$$desired\_pod = current\_pod * \left(\frac{current\_cpu}{desired\_cpu}\right)$$

..... (4.1)

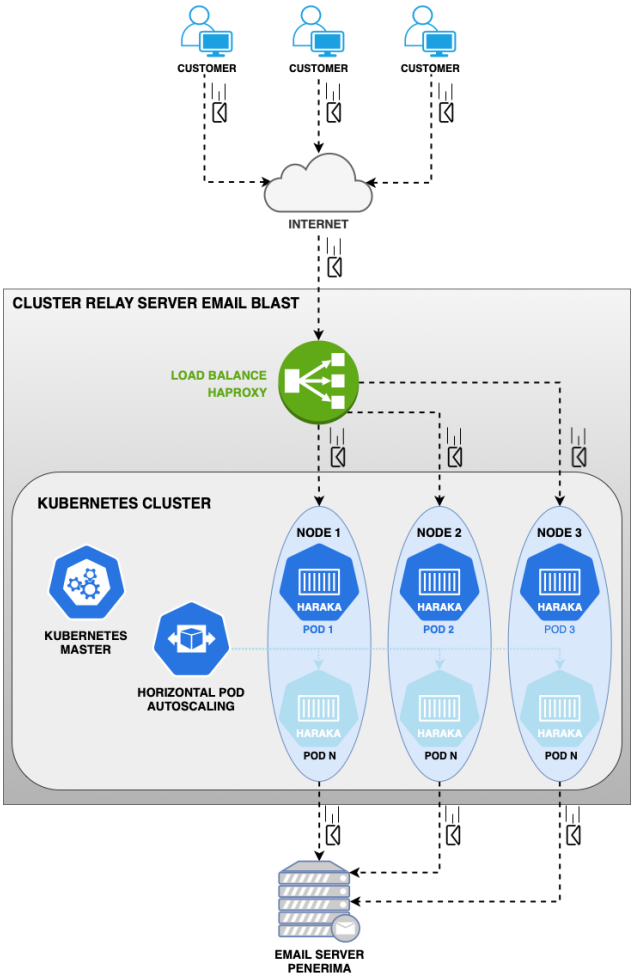
atau

$$jumlah\_pod\_ideal = jumlah\_pod\_berjalan * \left(\frac{penggunaan\_cpu}{batas\_penggunaan\_cpu}\right)$$

..... (4.2)

1.1.2 Integrasi Sistem Usulan

Sistem usulan yang diberikan merupakan hasil integrasi dari *mail transfer agent Haraka, Kubernetes cluster, load balancer HAProxy* serta *horizontal pod autoscaling*. Integrasi beberapa komponen tersebut pada sistem usulan diharapkan dapat mengurangi atau menyelesaikan kekurangan yang ada pada sistem berjalan. Berikut adalah topologi sistem integrasi yang dimaksud:



Gambar 2 Integrasi Pada Sistem Usulan

Berdasarkan gambar di atas, sistem usulan yang sudah terintegrasi akan menangani layanan *relay email blast*, mulai dari masuknya *email* yang berasal dari *email server Customer* sampai dengan melakukan *relay email* tersebut ke *email server* tujuan. Berikut adalah hal-hal yang dilakukan sistem usulan untuk melayani *request* layanan *relay email blast*:

1. Pertama-tama, *traffic* kiriman *email* akan masuk ke *load balancer HAProxy* yang menggunakan layanan *NodeBalancer* oleh *VPS provider Linode*.
2. Dari *load balancer HAProxy traffic* kiriman *email* akan didistribusikan ke 3 *node* yang merupakan bagian dari *Kubernetes cluster* dengan spesifikasi masing-masing *node* sebagai berikut.

a. *vCPU* = 2
b. *Memory (RAM)* = 4 *GB*
c. *Space HDD* = 80 *GB*
d. *Bandwidth/Network Transfer* = 4 *TB*

3. *Traffic email* yang masuk ke *node* ditangani oleh *pod* yang di dalamnya sudah terdapat *container Haraka*.

Informatics, Science, and Technologies Journal
Vol 12 No 2 (2022) **Jurnal GERBANG STMIK Bani Saleh**
ISSN : 0853 – 6376
E-ISSN : 2622 – 3708

4. *Kubernetes master* tidak menangani langsung *traffic email*, namun hanya memastikan semua komponen yang ada pada *Kubernetes cluster* bisa berfungsi sebagaimana mestinya untuk menunjang kegiatan penanganan *traffic email*.
5. *Horizontal Pod Autoscaling (HPA)* akan melakukan *auto scale up pod (n)* saat penggunaan *resource pod* meningkat dan melewati batas persentase yang ditentukan dengan jumlah *desired_pod (n)* sesuai dengan perhitungan berikut:

$$desired\_pod = current\_pod * \left(\frac{current\_cpu}{desired\_cpu}\right) \quad \text{..... (4.3)}$$

6. Lalu dari *pod*, *traffic email* akan di-*relay Haraka* ke *email server tujuan*.

Penggunaan *Resource CPU* Saat Pengiriman *Email* Normal

Sebelum melanjutkan ke tahap pengujian proses *autoscaling Kubernetes*, penulis memerlukan data penggunaan *resource CPU pod Haraka* dalam dalam mengirimkan *email normal non-blast* atau *email* dengan jumlah yang relatif normal. Data penggunaan *resource CPU* dalam menangani *request* pengiriman *email* normal yang dicari akan menjadi acuan penulis dalam menentukan spesifikasi minimal *CPU* yang dibutuhkan sebuah *pod Haraka* pada saat kondisi normal atau tidak dalam kondisi menangani *traffic email blast*.

Untuk melakukan pengetesan kebutuhan *resource CPU*, penulis perlu menentukan berapa jumlah *email* normal yang dikirimkan dalam jangka waktu tertentu untuk dijadikan acuan *penetration test* kepada *pod Haraka*. Penulis sudah melakukan pengumpulan data tersebut dari sistem pada tempat penelitian yang sudah digunakan secara rutin untuk melakukan *relay email* para *Customer*, dan telah didapatkan data berikut:

Tabel 2. Data Jumlah Kiriman *Email* Normal *Non Blast*

Jumlah *Server Sample*	**Jangka Waktu**	**Rata-rata Jumlah Kiriman *Email***				
		Hari Ke-				
		1	**2**	**3**	**4**	
11 *Server*	1 Jam	18083	17378	18363	17558	
1 *Server*	1 Jam	1643	1579	1669	1596	
Perkiraan Jumlah Kiriman *Email/Server*/Jam					1500-2000	

Berdasarkan data pada tabel di atas, secara normal 1 *server relay* rata-rata biasa menangani kiriman *email* sejumlah 1500-1600 dalam waktu 1 jam, namun beberapa *server sample* bisa mencapai 2000 *email* perjam, Dari data jumlah kiriman *email* yang didapat, penulis akan melakukan *penetration test* ke 1 *pod Haraka* dengan 2000 *request email* sekaligus menggunakan *tools Apache JMeter*. Berikut adalah statistik penggunaan *resource CPU pod Haraka* saat menangani *request* tersebut:

Gambar 10 Statistik Penggunaan *CPU Pod Haraka* Saat Menangani 2000 *Request Email*

Pada saat menangani 2000 *request email* sekaligus, *pod Haraka* bisa menghabiskan *resource CPU* maksimal 811 *millicore* atau 0,8 *core*. Sebenarnya dari data tersebut penulis sudah dapat mengambil kesimpulan berapa kebutuhan *resource CPU pod Haraka* untuk menangani *email* normal, namun penulis menggunakan *tools* yang bisa memberikan rekomendasi berapa alokasi *resource CPU* berdasarkan *request* layanan yang ditangani oleh sebuah *pod*. Berikut adalah rekomendasi dari *tools-tools* yang dimaksud:

Tabel 3. Rekomendasi Alokasi *CPU Pod*

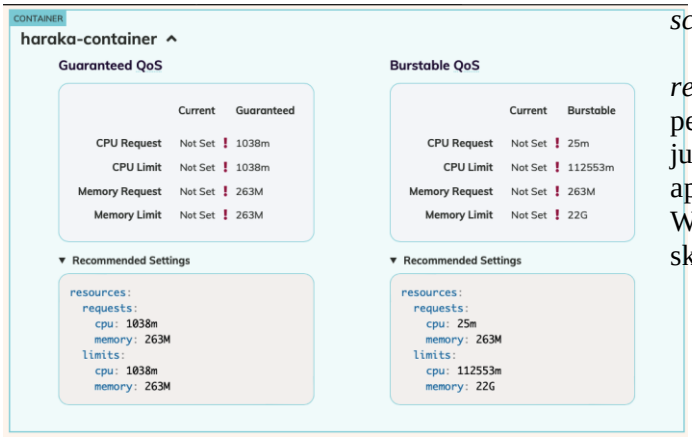
No	***Tools***	**Rekomendasi Alokasi *CPU***
1	*Vertical Pod Autoscaler (VPA)*	813 *millicore*
2	*Goldilocks*	1038 *millicore*

Sebagai acuan, penulis menggunakan *tools Vertical Pod Autoscaler* yang disediakan oleh *Kubernetes* untuk mendapatkan rekomendasi alokasi *resource CPU*. Berikut adalah rekomendasi alokasi *resource CPU pod* yang dihasilkan oleh *VPA*:

```
Status:
  Conditions:
    Last Transition Time:  2022-07-16T15:20:52Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  haraka-container
      Lower Bound:
        Cpu:     25m
        Memory:  262144k
      Target:
        Cpu:     813m
        Memory:  410771395
```

Gambar 11. Rekomendasi Alokasi *CPU* No. 1

Selain *VPA*, penulis juga menggunakan *tools Goldilocks* untuk mendapatkan rekomendasi alokasi *resource CPU*. Berikut adalah rekomendasi alokasi *resource CPU pod* yang dihasilkan oleh *Goldilocks*:



Gambar 12. Rekomendasi Alokasi *CPU* No. 2

Tools nomor 1 memberikan rekomendasi alokasi *CPU* sekitar 800 *millicore* untuk memenuhi kebutuhan 2000 *request email*, atau bisa dibilang sama dengan hasil penggunaan *CPU* yang didapatkan dari statistik. Namun *tools* nomor 2 memberikan rekomendasi alokasi *CPU* lebih banyak jika dibandingkan dengan statistik pengunaan *CPU* yaitu 1000 *millicore*. Dari data tersebut, penulis memutuskan untuk mengambil nilai alokasi *CPU* diantara nilai rekomendasi yang paling besar, yaitu sekitar 900-1000 *millicore*.

Alokasi CPU yang sudah ditentukan tersebut akan dibagi menjadi 3, mengacu pada jumlah *node* pada *Kubernetes cluster*. Dengan demikian, maka 1 *pod* akan diberikan minimum alokasi *CPU* sekitar 300 *millicore* atau 0,3 *core* dan maksimal sekitar 350 *millicore* atau 0,35 *core* untuk menangani paling tidak 2000 pengiriman *email* dalam jangka waktu 1 jam, lalu apabila ada pengiriman *email* lebih dari 2000 maka pada kondisi tersebut *Horizontal Pod Autoscaler* diharapkan dapat bekerja untuk melakukan *scale up* jumlah *pod*.

Response Action HPA Terhadap Penggunaan *CPU*

Selanjutnya, penulis melakukan pengujian terhadap *horizontal pod autoscaler Haraka*, lebih tepatnya melakukan pengujian bagaimana *response action* yang diberikan oleh *HPA* pada saat penggunaan *CPU pod* mencapai batas yang ditentukan, dimana meningkatnya penggunaan tersebut diduga akan berbanding lurus dengan fluktuasi pengiriman *email blast*. Adapun, jumlah minimal *pod* yang ditentukan adalah 3 *pod* dengan alokasi minimal *CPU* masing-masing sebesar 0,3 *core*, lalu *horizontal pod autoscaler* juga sudah dikonfigurasi oleh penulis untuk memberikan nilai 80% untuk dijadikan batas penggunaan *CPU pod* sebelum *horizontal pod autoscaler* melakukan *scaling* secara otomatis.

Namun, sebelum melakukan pengujian *response action HPA* saat mengirimkan *email blast*, penulis juga melakukan pengiriman *email* dalam jumlah normal, yaitu 2000 *email* untuk melihat apakah *HPA* akan melakukan *scaling* atau tidak? Walaupun jumlah *email* yang ditangani merupakan skala normal.

Tabel 4. *Response Action HPA* Terhadap Pengiriman *Email* Normal

***Autoscaling Pod Relay* Saat Terdapat 2.000 *Email* Sekaligus**			
Kondisi	***Pod***	**Rata-rata Penggunaan *CPU Pods***	**Persentase *CPU* dari Alokasi Minimum *CPU***
Before scale	3	290 *millicore*	96%
After scale	4	233 *millicore*	77%

Tabel 5. *Response Action HPA* Terhadap 10 Ribu Pengiriman *Email*

Pengujian Pengiriman 10 Ribu *Email* Dibagi 4 *Batch*-2.500 *Email*/*Batch*					
No. *Batch*	***Pod***	**Rata-rata *CPU***	**Persentase *CPU***	***Autoscale Pod***	**Persentase Setelah *Autoscale***
1	3	314	104%	+1	79%

Informatics, Science, and Technologies Journal
Vol 12 No 2 (2022) **Jurnal GERBANG STMIK Bani Saleh**
ISSN : 0853 – 6376
E-ISSN : 2622 – 3708

		millicore			
2	4	310 *millicore*	103%	+2	43%
3	6	281 *millicore*	93%	+1	0%
4	7	268 *millicore*	89%	+1	57%

Tabel 6. *Response Action HPA* Terhadap 30 Ribu Pengiriman *Email*

Pengujian Pengiriman 30 Ribu *Email* Dibagi 4 *Batch*-7.500 *Email*/*Batch*					
No. *Batch*	***Pod***	**Rata-rata *CPU***	**Persentase *CPU***	***Autoscale Pod***	**Persentase Setelah *Autoscale***
1	3	336 *millicore*	112%	+6	58%
2	9	268 *millicore*	89%	+2	40%
3	11	268 *millicore*	89%	+2	40%
4	13	228 *millicore*	76%	-	-

Waktu Pengiriman *Email Blast*

Penulis juga melakukan pengujian pengiriman *email blast* untuk mendapatkan berapa lama waktu yang dibutuhkan untuk mengirimkan *email blast* dengan jumlah tertentu. Berdasarkan jumlah pengiriman *email blast* yang dikirimkan oleh 1 *Customer*, yaitu sekitar 15 sampai 20 ribu *email*, maka penulis melakukan pengujian dengan cara membandingkan waktu pengiriman *email blast* dengan jumlah 10 ribu, 15 ribu sampai 20 ribu jika menggunakan sistem *dummy* dan jika menggunakan sistem usulan.

Sebagai informasi, penulis melakukan pengujian dengan aplikasi *mail campaign* berbasis web yang merupakan salah satu layanan di tempat penelitian, Berikut adalah data pengujian waktu pengiriman *email blast* sistem *dummy* dan sistem usulan:

Tabel 7. Perbandingan Waktu Pengiriman *Email Blast*

Jumlah *Email*	**Waktu Pengiriman**	
	Sistem Usulan	**Sistem *Dummy***
10 Ribu	13 menit	11 menit
15 Ribu	13 menit	13 menit
20 Ribu	14 menit	20 menit

PENUTUP

Simpulan

1. Berdasarkan uji coba yang dilakukan, terbukti bahwa *cluster* tersebut dapat mengirimkan *email blast* sebanyak 20 ribu lebih cepat 6 Menit
2. Rancangan *cluster* yang dibuat oleh penulis juga dilengkapi *Kubernetes* yang memiliki *tools Horizontal Pod Autoscaling (HPA)*. *HPA* terbukti mampu secara otomatis untuk menyesuaikan jumlah *pod*, baik mengurangi (*Scale down*) ataupun menambah (*Scale up*) sesuai dengan penggunaan *resource CPU pod container mail transfer agent* yang berbanding lurus dengan jumlah pengiriman *email blast*.
3. Setelah melakukan implementasi sistem usulan hasil perancangan, serta membandingkan proses pengiriman *email blast* didapatkan hasil pengujian bahwa sistem usulan lebih baik dalam proses penambahan atau konfigurasi *server* baru pada sistem usulan dapat dilakukan dengan singkat.

Saran

Dari hasil dan kesimpulan penelitian, berikut adalah saran yang dapat diberikan serta dipertimbangkan untuk pelaksanaan penelitian selanjutnya :

1. Pada penelitian selanjutnya, sebaiknya dilakukan analisis pada variabel *event learning* yang ada pada *HPA Kubernetes*.
2. Pada penelitian selanjutnya, sebaiknya perancangan sistem *cluster relay server email blast* dilengkapi dengan *autoscale* pada *node*, namun dengan kemampuan penambahan *IP address node* baru ke *SPF record* secara otomatis

DAFTAR PUSTAKA

[1] Adiputra, Firmansyah. 2015. "Container Dan Docker Teknik Vertualisasi Dalam Pengelolaan Banyak Aplikasi Web." *Jurnal SimanteC* 4 (3): 167–76.

[2] Burns, Brendan, Joe Beda, and Kelsey Hightower. 2019. *Kubernetes: Up and Running: Dive into the Future of Infrastructure, Edition 2*.

[3] Dwiyatno, Saleh, Edy Rakhmat, and Oki Gustiawan. 2020. "Implementasi Virtualisasi Server Berbasis Docker Container." *Prosisko* 7 (2): 165–75. https://e-jurnal.lppmunsera.org/index.php/PROSISKO/article/view/2520/.

[4] Ghani, Sachel. 2019. "SUCCESS FACTORS OF E-MAIL MARKETING," no. 3434: 13.

[5] Lehtinen, Kim. 2022. “Scaling a Kubernetes Cluster,” 73.
[6] Liimatainen, Parima. 2020. “How to Improve Customer Engagement by Email Marketing,” 36.
[7] Malviya, Nitesh. 2020. *Docker—A Must Have Tool for Developers*.
[8] Pedro, João. 2020. “Distributed Mail Transfer Agent,” 113.
[9] Penelitian, Jurnal, Ilmu Komputer, Fitri Komariyah, and Harum Argyawati. 2016. “IMPLEMENTASI SERVER CLUSTER HIGH AVAILABILITY PADA WEB SERVER Pendahuluan Saat Ini Yang Informasi Sangat Merupakan Penting Bagi Computer Merupakan Beberapa Teknologi Sumber Yang Daya Memanfaatkan Kebutuhan Komputer Tunggal Yang Kemudian Bekerja Bersama-Sa” 4 (2): 78–88.
[10] Premanantha, Divya. 2021. “IMPROVE HORIZONTAL POD AUTOSCALING IN CONTAINER ORCHESTRATION TO ADOPT FREQUENT OSCILLATION IN SERVICE REQUESTS.”
[11] Rismayadi, Ali Akbar, Salman Topiq, and Rinto Nurtantho. 2020. “Membangun Mail Server Berbasis Linux Menggukan Postfix Admin.” *Jurnal Responsif* 2 (1): 92–98.
[12] Safyan, Gigi. 2018. *Mastering Kubernetes: Master the Art of Container Management by Using the Power of Kubernetes, 2nd Edition, Edition 2*. 2nd ed.
[13]Saputra, Andika, and Melwin Syafrizal. 2012. “Perancangan Dan Implementasi Mail Server Pada Cv. Sanjaya Anugerah Sejahtera (Isp Jogjaringan) Berbasis Open Source.” *Data Manajemen Dan Teknologi Informasi (DASI)* 13 (2): 1.
[14]Sumbogo, Yosua Tito, Mahendra Data, and Reza Andria Siregar. 2018. “Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes.” *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer (J-PTIIK) Universitas Brawijaya* 2 (12): 6849–54.

Informatics, Science, and Technologies Journal
Vol 12 No 2 (2022) **Jurnal GERBANG STMIK Bani Saleh**
ISSN : 0853 – 6376
E-ISSN : 2622 – 3708