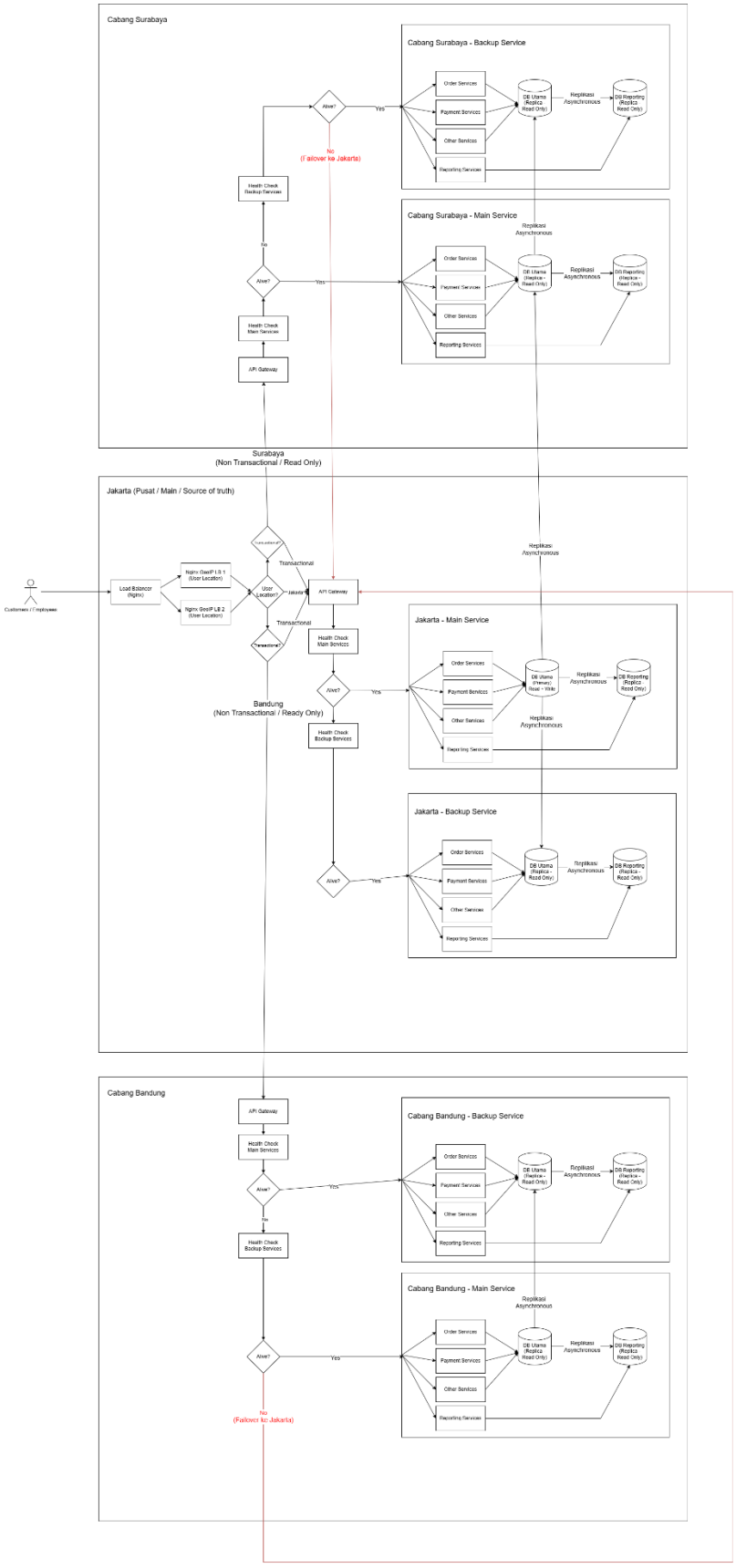
Contents

Arsitektur


Cabang Surabaya
Cabang Surabaya - Backup Service
Order Services
Payment Services
Other Services
Reporting Services
DB Utama (Replica - Read Only)
DB Reporting (Replica - Read Only)
Replikasi Asynchronous
Alive?
Yes
No
(Failover ke Jakarta)
Health Check Backup Services
Cabang Surabaya - Main Service
Health Check Main Services
API Gateway
Surabaya
(Non Transactional / Read Only)
Jakarta (Pusat / Main / Source of truth)
Load Balancer (Nginx)
Nginx GeoIP LB 1 (User Location)
Nginx GeoIP LB 2 (User Location)
User Location?
Jakarta
Transactional
Jakarta - Main Service
Jakarta - Backup Service
Bandung
(Non Transactional / Reacty Only)
Cabang Bandung
Cabang Bandung - Backup Service
Cabang Bandung - Main Service
(Failover ke Jakarta)

Penjelasan

Metodologi Pengerjaan

Dalam 1 minggu ini, saya berusaha mempelajari dan meningkatkan kemampuan saya di DevOps yang sebelumnya masih pada tahap dasar. Saya menggunakan berbagai referensi dan tools baik Google, ChatGPT, Youtube, dan pengalaman pribadi untuk menyusun tugas rancangan yang diminta. Soal-soalnya cukup menantang tapi saya sangat excited mengerjakanannya.

Dokumen dan Github yang ditulis, merupakan rangkuman yang diyakini oleh saya benar, setelah melewati proses pembelajaran, analisis, dan pemahaman mendalam. Saya akui masih banyak kekurangan dan saya masih belajar lebih jauh untuk lebih memahami dan menyempurnakan rancangan saya dan terbuka untuk belajar dari senior ataupun referensi lain.

Mohon izin juga, mungkin saya mencatat makna beberapa istilah untuk catatan agar mudah jika perlu review ulang. Izin diinfokan juga ada beberapa script yang saya copy paste dari referensi dan belum sempat saya lakukan testing karena masih dipelajari dan sayangnya sudah sampai batas waktu pengumpulan.

Terima kasih sudah memberi kesempatan saya untuk mengerjakan soal tes ini. Saya sangat senang jika saya bisa belajar dan berkontribusi lebih jauh di perusahaan bapak/ibu ke depannya khususnya di area IT DevOps/Software Development. Semoga laporan ini bermanfaat untuk kita semua.

Breakdown Masalah / Tantangan yang Dihadapi

1. Trafik di cabang tinggi -> saya usahakan sistem berada di dekat user. Biasanya kalau semakin jauh antara server dan user (misal beda kota), latency networknya jadi tinggi. Dengan dekat, diharapkan beban server terbagi, dan network tidak terlalu lama mendeliver datanya.
2. Ada gangguan jaringan antar kota -> saya usahakan minimalkan ketergantungan akses ke luar kota. Sehingga traffic sebanyak mungkin dihandle oleh sistem di dalam kota tersebut.
3. Data pada modul laporan inkonsistensi -> saya asumsikan ini diakibatkan proses replikasi yang lambat akibat beban kerja database tidak terbagi dengan baik atau terlalu menumpuk pada source of truth. Diharapkan dengan load balancer yang baik, proses replikasi berjalan lebih lancar dan optimal. Pembagian beban akan dibuat berdasarkan 2 hal :
    - Read berdasarkan lokasi user. Biasanya Read itu juga ga kecil data tarikannya. Sehingga jika user berada di Bandung maka tarik datanya dari server di

Bandung. Kalau usernya berada di Surabaya maka Tarik datanya dari server di Surabaya. Latency network lebih rendah, beban server terbagi.

- Write hanya ke main / source of truth artinya di DB Jakarta.

Narasi Aliran Request / Data

1. User (Customer / employee) akan mengakses ke siste. Anggap saja yang diakses Adalah sebuah web via https://ecommerce.knitto.co.id.
2. Request dari user pada awalnya akan diarahkan ke sistem di Jakarta untuk menentukan :
    a. Lokasi User Berasal
    b. Kegiatan yang dilakukan oleh user
3. Jika user melakukan kegiatan non transactional (misal : melihat list produk, reporting, dll) yang hanya sebatas read, maka diarahkan ke sistem terdekat yaitu berdasarkan lokasi user itu berada. Misal lokasi user berasal dari Bandung, maka sistem Bandung yang melayani permintaan user tersebut.
4. Jika user melakukan kegiatan transactional (misal : create order) maka sistem Jakarta yang melayani dan menyimpan data order baru tersebut. Server database Jakarta akan membagikan / menginfokan ke server database Bandung dan Surabaya melalui sistem replikasi. Metode replikasi yang diambil adalah asynchronous, hal ini dikarenakan resiko network antar kota cukup tinggi sehingga kita buat sistem Jakarta tidak perlu menunggu proses sinkronisasi beres di semua kota hanya untuk melayani user. Dengan begitu pelayanan terhadap user bisa lebih lancar dan cepat.
5. Cabang Surabaya dan Cabang Bandung meposisikan diri sebagai penerima salinan data dari pusat di Jakarta. Posisi database di cabang Surabaya dan cabang Bandung bersifat pasif (read only). Hal ini bertujuan untuk meminimalisir terjadinya split brain, karena hanya 1 database yang bisa dilakukan write (Jakarta) dan selebihnya menerima salinan dari Jakarta (Cabang Bandung dan Cabang Surabaya).
6. Selain dengan aktif-pasif database, split brain perlu ditambah proteksi ekstra yaitu metode Quorum. Quorum di sini adalah metode untuk menentukan database primary berikutnya ketika database primary saat ini mati (harus ganjil jumlah node Postgre nya, missal 3 node, 5 node, dst). Saya sebenarnya belum pernah terapkan Quorum di Postgre, dulu pernah nerapin yang sama di RabbitMQ. Kurang ini lebih ini sistem voting.
7. Database reporting dipisah dan disediakan di setiap cabang. Proses read ke data reporting biasanya cukup berat dan besar data tarikannya, sehingga wajib dipisah. Ada beberapa metode tambahan yang bisa diupayakan mengurangi beban yaitu dengan :
    a. Cache menggunakan Redis untuk data report yang tidak cepat berubah. Read/Write RAM lebih cepat daripada Read/Write ke Disk, jadi ini cukup membantu meringankan beban.
    b. SQL select dibuat seringan mungkin, artinya harus ada SQL lain yang membuat rekapan secara berkala. Hal ini bertujuan agar ketika user meminta report maka

sistem tidak perlu lagi mengkalkulasi data sebelum menampilkan report. SQL ringan maka kerja sistem lebih ringan. (Contoh Kalkulasi : misal harus menjumlahkan total penjualan item X di hari Senin atau menghitung total Rupiah penjualan di hari Sabtu).

8. Setiap cabang memiliki 2 VM service yaitu untuk service-main dan 1 service-backup. Maka saya buat suatu metode health check untuk mengecek apakah service-main hidup (biasanya pakai API /check).
    a. Skenario 1 (Jika cabang Bandung atau Cabang Surabaya ada gangguan sebagian / total) :
        i. Jika service-main hidup maka request akan dilayani oleh service-main
        ii. Jika service-main mati maka request akan dilayani oleh service-backup
        iii. Jika service-backup di cabang Surabaya/Bandung mati juga, akan di failover ke Jakarta sebagai prioritas utama.
    b. Skenario 2 (Jika Jakarta yang servicenya mati) :
        i. Service di sini artinya layanan seluruhnya mati bukan hanya DB yang mati.
        ii. service-main dan service-backup di Jakarta tidak bisa melayani request
        iii. Maka semua user tidak bisa melakukan kegiatan transaksional (missal create order).
        iv. User Bandung / area Bandung dan sekitarnya dapat membuka service yang sifatnya read only (missal : list produk), tapi tidak bisa transaksi.
        v. User Surabaya / area Surabaya dan sekitarnya dapat membuka service yang sifatnya read only (missal : list produk), tapi tidak bisa transaksi.
        vi. User Jakarta dan sekitarnya agar tidak error, bisa dibuat dialihkan ke service terdekat yaitu Bandung
        vii. Service di Jakarta harus diperbaiki. Tidak ada jalan lain.
    c. Skenario 3 (Jika hanya database yang mati / putus koneksi tapi bukan primary DB) :
        i. Service dianggap tidak berfungsi. Tanpa database, service hampir pasti tidak bisa melayani user. (Kecuali ada cache)
        ii. Mau tidak mau harus di failover. Dalam skema saya, failover akan diprioritaskan ke Jakarta (sebagai source of truth).
        iii. Server DB replikasi (yaitu Cabang Bandung / Cabang Surabaya) harus segera diperbaiki sesuai SLA.
    d. Skenario 4 (Jika DB Primary yang mati / putus koneksi) :
        i. Service DB Primary ada di Jakarta. Artinya Jakarta sebagai source of truth dan yang satu satunya bisa write DB tidak bisa melayani kegiatan transaksional.
        ii. Sistem akan menjalankan secara otomatis metode Quorum. Dan memilih pengganti primary, missal pindah ke DB Bandung. Jakarta dan Surabaya jadi read only.

e. Skenario 5 (Jika 2 service mati / putus koneksi misal Bandung dan Surabaya mati) :
    i. Service / layanan dalam kondisi darurat. Layanan tidak bisa melakukan transaksi / hanya read only.
    ii. DB karena menggunakan metode Quorum. Maka quorum tidak terpenuhi karena 2 dari 3 server mati. Artinya tidak ada jalan lain selain server-server segera diperbaiki.
    iii. Pendekatan ini agar tidak terjadi split brain.

9. Database akan dilakukan juga full backup rutin 1 hari sekali terutama untuk database PRIMARY dan Hasil Replikasi. Hasil backup disimpan di setiap cabang agar backup nya tersebar dan ada beberapa salinan.

Tabel Service Level Agreement (SLA) yang harus dipenuhi berdasarkan skema sistem di atas :

Kondisi	Urgency Level (Impact to Business)	Target Waktu Respon Terhadap Masalah	Target Waktu Penyelesaian
Service (Main+Backup) 3 kota mati total	High	1 menit	60 menit
Service 2 (Main+Backup) dari 3 kota mati	High	3 menit	30 menit
Service (Main+Backup) 1 dari 3 kota mati (tapi bukan primary)	High	5 menit	15 menit
Database reporting down / tidak dapat diakses di semua cabang	High	3 menit	60 menit
Service Main mati Tetapi Service Backup hidup	Medium	10 menit	15 menit
Database reporting down / tidak dapat diakses di salah satu cabang	Medium	15 menit	15 menit
Server lambat akibat lonjakan di luar dugaan (anomali)	Medium	10 menit	60 menit

Justifikasi Trade-Off

Aspek	Dikorbankan	Benefit yang Didapat
Availability	Kalau 2 dari 3 kota down total. 1 kota terakhir tetap tidak bisa melayani user secara penuh.	Bebas dari masalah Split Brain
Workload	Jakarta akan sibuk sebagai pusat write / transaksi	Integritas data aman, karena hanya 1 sumber data baru. Selebihnya replikasi.
Workload	Server Jakarta harus kerja lebih keras karena ada kegiatan mengirimkan data ke cabang untuk replikasi	Salinan data ada di cabang. User yang dekat dengan cabang bisa akses data yang di cabang tersebut, tidak perlu semua ke pusat. Sistem lebih responsive karena beban terbagi. Load Balancing.
Cost	Server jumlahnya lebih banyak artinya cost lebih banyak untuk infrastruktur	Akses dan Availability lebih terjamin karena ada pembagian beban kerja.

Langkah-Langkah Penyiapan Skema

Provisioning

Dibagi 3 :

1. Control Plane
    a. Menjalankan API Server, Scheduler
    b. Tidak dipakai untuk workload aplikasi
2. Worker – General Purpose
    a. API Gateway
    b. Service dari Aplikasi (Order, Inventory, dll)
    c. Tugas-tugas ringan
3. Worker – I/O Intensive
    a. Database – Misal saya pakai PostgreSQL

IaC (Infrastructre as Code) :

```
cluster "jakarta" {
node_pool "general" { count = 3 }
node_pool "database" { count = 3 }
```

```
}

cluster "bandung" {
node_pool "general" { count = 2 }
node_pool "database" { count = 2 }
}

cluster "surabaya" {
node_pool "general" { count = 2 }
node_pool "database" { count = 2 }
}
```

Note :

1. Jakarta pusat write / menjadi primary (transaksional). Posisinya vital, sehinggal node_pool dipasang paling besar
2. Cabang Bandung dan Surabaya. Menangani traffic juga, Cuma karena posisinya replicated dan efisiensi cost. Maka dibuat lebih rendah node_poolsnya.

Networking

Container Network Interface (CNI)

Rekomendasi dari hasil searching adalah Calico, katanya stabil dan support network policy.

CIDR – Rentang IP Address

Per cluster:

1. Pod CIDR: 10.X.0.0/16
2. Service CIDR: 10.Y.0.0/16

Contoh:

- Jakarta Pod CIDR: 10.10.0.0/16
- Bandung Pod CIDR: 10.20.0.0/16
- Surabaya Pod CIDR: 10.30.0.0/16

Ingress & Egress

Ingress (Jalur Masuk Sistem) : Via Nginx

Flow : Internet -> Load Balancer -> Nginx -> API Services -> Database

Egress (Jalur Keluar Sistem) : NAT Gateway (IP yang digunakan user)

Flow : Pod -> NAT Gateway -> Internet

DNS (Domain Name Server)

Ada 1 DNS Server. Rekomendasi yang ditemukan di Internet itu CoreDNS. Saya belum pernah setting ini.

Cuma dulu pernah konfigurasi High Availability (HA). Jadi aplikasi akses service / DB itu via DNS. Kalau misal server nya harus switch dari server A ke B karena primary nya pindah tinggal routing ulang IP yang dipointing oleh DNS tersebut ke server yang baru / aktif.

Misal ada DNS :
https://api.nusantaramart.co.id , yang pointing awalnya ke 10.10.0.1

Kemudian 10.10.0.1 down (Jakarta), kita belokin ke Bandung. Jadi DNS https://api.nusantaramart.co.id pointing ke Bandung yaitu misal 10.20.0.1

Saya baca-baca ternyata ada acara biar pointing otomatis, yaitu menggunakan HA Controller di Kubernetes , Patroni (PostgreSQL), dll. Akan saya coba nanti.

mTLS

Hasil dari searching bisa pakai Istio (Buat Kubernetes). Akan saya coba nanti.

Data Layer

Topologi Transaksi VS Read Replica

1. Service Main :
    a. 1 Database Primary (Write + Read)
    b. Database Stand By (Read Only) di setiap cabang (Karena ada cabang Bandung dan cabang Surabaya maka total : 2 DB Stand By)
    c. 1 Database Reporting (Read Only) di setiap cabang (Karena ada cabang Bandung dan cabang Surabaya maka total : 2 DB reporting)
    d. Metode Replikasi yang dipakai : Async Streaming Replication
2. Service Backup (Tiap Cluster Cabang ada Service Backup), dibuat sama dengan Service Main:
    a. 1 Database Primary (Write + Read)
    b. 1 Database Stand By (Read Only) di setiap cabang (Karena ada cabang Bandung dan cabang Surabaya maka total : 2 DB Stand By)

c. 1 Database Reporting (Read Only) di setiap cabang (Karena ada cabang Bandung dan cabang Surabaya maka total : 2 DB reporting)
d. Metode Replikasi yang dipakai : Async Streaming Replication

Deployment dengan StatefulSet

Cocok untuk : Database, Message Queue, Storage System

Alasan : Pod punya identitas tetap, urutan jelas, storage harus menempel pada Pod

Jadi deployment nya pakai StatefulSet khususnya untuk deploy database (Saya pakai PostgreSQL)

Persistent Volume Claim (PVC)

Persisten Volume Claim (PVC) adalah kebutuhan/permintaan storage disk permmanen untuk Pod. Pod bisa mati kapan saja, bisa pindah node, bisa diganti. Database butuh disk, datanya harus tetap terjaga.

Maka :

1. setiap Pod memiliki 1 PVC.
2. Implementasi Main Service :
    a. Postgre Utama Jakarta (Storage : 500 GB)
    b. Postgre Utama Cabang Bandung (Storage : 500 GB)
    c. Postgre Utama Cabang Surabaya (Storage : 500 GB)
    d. Postgre Reporting Jakarta (Storage : 500 GB)
    e. Postgre Reporting Bandung (Storage : 500 GB)
    f. Postgre Reporting Surabaya (Storage : 500 GB)
3. Implementasi Backup Service :
    a. Postgre Utama Jakarta (Storage : 500 GB)
    b. Postgre Utama Cabang Bandung (Storage : 500 GB)
    c. Postgre Utama Cabang Surabaya (Storage : 500 GB)
    d. Postgre Reporting Jakarta (Storage : 500 GB)
    e. Postgre Reporting Bandung (Storage : 500 GB)
    f. Postgre Reporting Surabaya (Storage : 500 GB)

StorageClass

Tipe Storage : SSD / NVME

ReclaimPolicy (Delete/Retain): Retain (Sehingga ketika PVC dihapus, disk fisik tetap ada, data tidak langsung hilang dan disk harus dihapus manual)

Alasan /Pertimbangan pilih Retain :

Risk	Delete	Retain
Human Error	Beresiko karena data langsung hilang	Aman, karena data masih disimpan di disk
Bug Deployment	Fatal	Bisa Recovery
Audit	Susah	Aman

Anti-Affinity

Memastikan antara DB primary dan standby tidak berada di node yang sama.

```
affinity:
podAntiAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
- labelSelector:
matchLabels:
app: postgres
topologyKey: kubernetes.io/hostname
```

Backup/Restore

Terlepas sudah async ke replica database, tetap akan dilakukan full backup database harian. Setiap cabang akan dibackup.

Selain Backup harian, kita akan backup dengan pendekatan Point In Time Recovery (PITR). Karena saya menggunakan Postgre, di PostgreSQL ada yang Namanya WAL (Write-Ahead Log ; catatan perubahan semua data). Jadi database bisa digeser-geser restore nya berdasarkan timestamp.

Restore nya :

1. Full Backup untuk jaga-jaga kalau service PostgreSQL nya down total dan tidak bisa direcovery. Atau misal ada kerusakan hardware sehingga disk tidak terbaca.
2. Restore berdasarkan timestamp (via WAL) jika diperlukan, misal akibat human error dan terjadi salah write.

HA di Aplikasi

Deployment

Hampir semua service aplikasi jika direstart aman karena tidak menyimpan data di service tersebut dan aman direstart kapan saja (stateless). Oleh karena itu metode yang digunakan adalah Deployment.

Service yang dimaksud adalah service order, service payment, service report, dll.

Setiap cluster akan dibuat beberapa replica.

- Cluster Jakarta : 2 replica pod (1 main 1 backup)
- Cluster Bandung : 2 replica pod (1 main 1 backup)
- Cluster Surabaya : 2 replica pod (1 main 1 backup)

Replica ini dipakai untuk jaga-jaga kalau yang utama/main service down.

PodDisruptionBudget (PDB)

Menjamin minimal pod hidup saat maintenance.

```
minAvailable: 1
```

Anti Affinity

Pod aplikasi dirancang tersebar di beberapa node.

Readiness

Readiness lebih ke arah apakah aplikasi siap menerima/melayani request/traffic?

Liveness

Liveness Lebih ke arah apakah aplikasi running dengan benar, misal tidak hang? Beberapa cara :

1. Harus dibuat notifikasi monitoring jika resource server habis.
2. Menggunakan Zabbix untuk monitoring resource
3. Membuat listener kalau hang maka otommatis restart

Tabel Spesifikasi Hardware

1. Perkiraan core vCPU, RAM, dan storage per node pool, per cluster (pisahkan untuk service rendah/tinggi).

   Note : 1 vCPU = 1 thread ; 0.5 vCPU = 500m = 0.5 thread

		CPU	Memory	Storage
Service Berat	Per Cluster	128 vCPU	128 GB	1 TB
	Per Node	64 vCPU	64 GB	500 GB
Service Medium	Per Cluster	64 vCPU	64 GB	400 GB
	Per Node	8 vCPU	8 GB	50 GB
Service Ringan	Per Cluster	8 vCPU	8 GB	100 GB
	Per Node	4 vCPU	4 GB	50 GB

   Contoh Service Berat : Database

   Contoh Service Medium : API dan Back End Aplikasi

   Contoh Service Ringan : Service Reporting

   Cluster itu kumpulan Node.

2. Kapasitas LB/Gateway, throughput target (RPS), dan headroom 30–50%.

   Target :

   - Peak/Max : 3000 RPS
   - HeadRoom 30 – 50% artinya 3900 – 4500 RPS

3. Proyeksi jaringan (bandwidth antar-cluster) dan RTO/RPO target.

   - Proyeksi Jaringan (Bandwidth antar-cluster) :
     - Replikasi DB + Backup/WAL : 50 Mbps per cluster
     - Aplikasi : 10 Mbps per cluster
   - RTO (Recovery Time Objective – Batas boleh down) : 3 menit (failover otomatis)
   - RPO (Recovery Point Objective – Batas maksimal data yang boleh hilang saat gagal) : Metode Replikasi yang digunakan Async. Asumsikan maksimal 10 detik karena masalah jaringan.

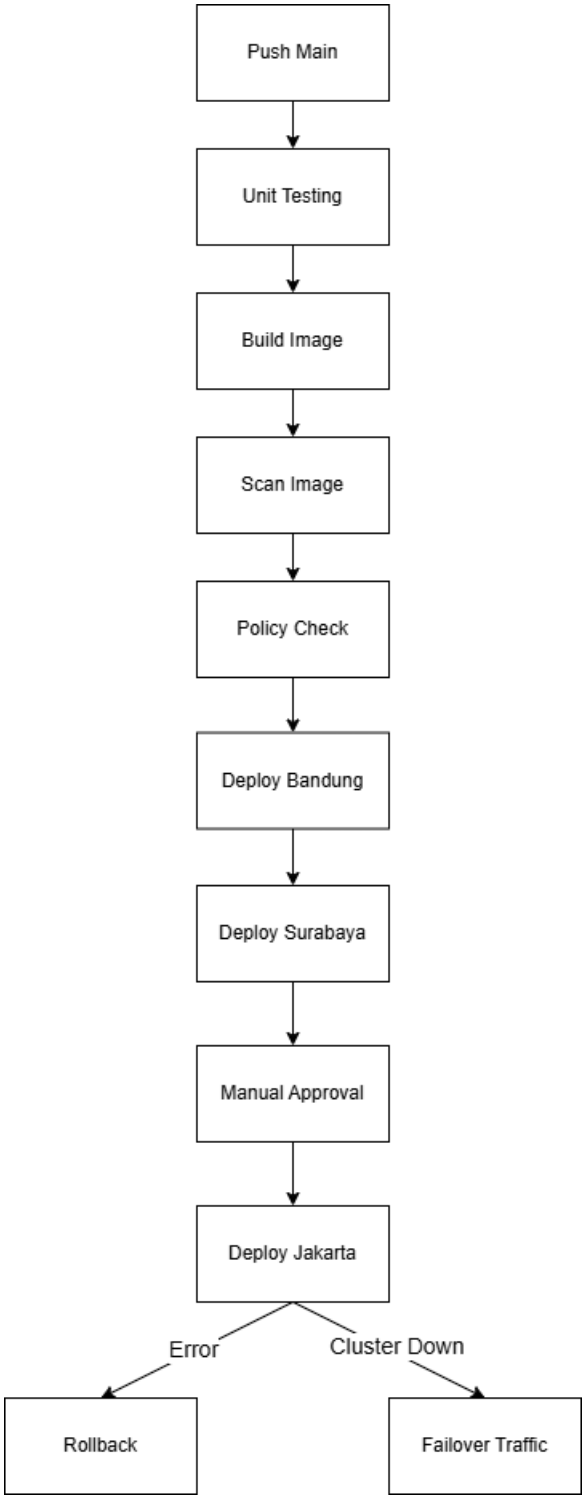
File Workflow YAML (Belum saya tes)

```
name: prod-multicluster-deploy

on:
  push:
    branches: ["main"]
  workflow_dispatch:

env:
  REGISTRY: ghcr.io/nusantaramart
  IMAGE_NAME: payment-service
  KUBE_VERSION: "1.29"

jobs:
  test-build-scan:
    runs-on: ubuntu-latest
    steps:
     - name: Checkout code
       uses: actions/checkout@v4

     - name: Set up Docker Buildx
       uses: docker/setup-buildx-action@v3

     - name: Login to GHCR
       uses: docker/login-action@v3
       with:
         registry: ghcr.io
         username: ${{ github.actor }}
         password: ${{ secrets.GITHUB_TOKEN }}

     - name: Unit Test
       run: |
         echo "Run unit tests here"

     - name: Build Image
       run: |
         docker build-t $REGISTRY/$IMAGE_NAME:${{ github.sha }} .
         docker push $REGISTRY/$IMAGE_NAME:${{ github.sha }}

     - name: Image Scan (Trivy)
       uses: aquasecurity/trivy-action@0.20.0
       with:
         image-ref: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}
         severity: CRITICAL,HIGH

  deploy-bdg-canary:
    needs: test-build-scan
    runs-on: ubuntu-latest
    environment: production
    steps:
     - uses: actions/checkout@v4
```

```
    - name: Set kubectl
      uses: azure/setup-kubectl@v4
      with:
        version: ${{ env.KUBE_VERSION }}

    - name: Configure kubeconfig Bandung
      run: echo "${{ secrets.KUBECONFIG_BANDUNG }}" | base64 -d > kubeconfig

    - name: Deploy Canary to Bandung
      run: |
        KUBECONFIG=kubeconfig \
        helm upgrade --install payment k8s/payment \
         --set image.tag=${{ github.sha }} \
         --set canary.enabled=true

  deploy-sby-canary:
    needs: deploy-bdg-canary
    runs-on: ubuntu-latest
    environment: production
    steps:
     - uses: actions/checkout@v4

     - name: Set kubectl
       uses: azure/setup-kubectl@v4
       with:
         version: ${{ env.KUBE_VERSION }}

     - name: Configure kubeconfig Surabaya
       run: echo "${{ secrets.KUBECONFIG_SURABAYA }}" | base64 -d > kubeconfig

     - name: Deploy Canary to Surabaya
       run: |
         KUBECONFIG=kubeconfig \
         helm upgrade --install payment k8s/payment \
          --set image.tag=${{ github.sha }} \
          --set canary.enabled=true

  approve-main:
    needs: deploy-sby-canary
    runs-on: ubuntu-latest
    environment:
      name: production-main
      url: https://status.nusantaramart.id
    steps:
     - name: Manual Approval Gate
       run: echo "Awaiting approval to deploy Jakarta"

  deploy-jkt:
    needs: approve-main
    runs-on: ubuntu-latest
```

```
  steps:
   - uses: actions/checkout@v4

   - name: Set kubectl
     uses: azure/setup-kubectl@v4
     with:
       version: ${{ env.KUBE_VERSION }}

   - name: Configure kubeconfig Jakarta
     run: echo "${{ secrets.KUBECONFIG_JAKARTA }}" | base64 -d > kubeconfig

   - name: Deploy Full Release to Jakarta
     run: |
       KUBECONFIG=kubeconfig \
       helm upgrade--install payment k8s/payment \
        --set image.tag=${{ github.sha }} \
        --set canary.enabled=false

rollback-on-failure:
  if: failure()
  needs: [deploy-bdg-canary, deploy-sby-canary, deploy-jkt]
  runs-on: ubuntu-latest
  steps:
   - name: Rollback via Helm
     run: |
       echo "Trigger helm rollback or traffic shift via gateway"
```

Diagram Alur Release-Rollback-Failover



Penjelasan :

1. Ketika perubahan di push ke Git, dilakukan unit testing untuk memastikan tidak ada bug. Jika bebas bug maka dibuat image baru (build image) yang digunakan sebagai template sistem yang akan dideploy.

2. Image baru hasil build akan melewati Scan Image menggunakan tools khusus. (Contoh tools adalah Trivy)
3. Sebelum deploy, image yang dibuat pastikan sudah sesuai kebijakan / standar perusahaan (policy check). Menggunakan tools seperti Open Policy Agent / Confest.
4. Karena Jakarta adalah primary dan source of truth, maka lebih baik update / deploy dulu ke cabang (Bandung dan Surabaya). Ini akan lebih aman.
5. Konsep proses deploy adalah forward / maju, jadi jika gagal maka di rollback semuanya. (Menggunakan syntax : kubectl rollout undo / helm rollback)
6. Jika ketika deploy menyebabkan salah satu cluster tidak menyala. Misal kita tentukan cluster cabang down lebih dari 3 menit maka lakukan failover ke cluster yang nyala agar menjaga availability sistem

Dashboard Sketsa, Contoh Query SLI/Alert, dan Runbook

Metrics

Dipakai untuk monitor CPU, RAM, Latency, Error Rate.

Menjadi dasar ketika menampilkan notifikasi alert dan SLO.

Sebelum lanjut, berikut Service Level Objective (SLO) yang saya rancang :

Area	Minimal yang Disebut Aman
Request User Sukses Diproses	>= 99.6%
Latency	< 300 ms
Availability	99.95%

SLI, SLO, SLA

Istilah	Arti
Service Level Indicator (SLI)	Kondisi layanan sekarang seperti apa? Metrik actual yang dipakai untuk mengukur kualitas layananan.
Service Level Objective (SLO)	Target internal kita apa?
Service Level Agreement (SLA)	Janji ke customer apa?

Logs

1. ELK

Kepanjangan dari :	Makna
Elasticsearch	Penyimpanan dan Search
Logstash	Proses & Parsing Log
Kibana	Kibana (Visualisasi & Search UI)

Dipakai untuk : Search log epat, dashboard, dan analisis insiden

2. PLG

Kepanjangan dari :	Makna
Promtail	Agen Kirim Log
Loki	Storage Log
Grafana	UI & Dashboard

Dipakai untuk sama kaya ELK, tapi lebih ringan

3. Retensi
    1. Retensi Log Aplikasi : 60 hari
    2. Retensi Log Audit : 365 hari (1 tahun)
    3. Retensi Log Security : 180 hari

4. Personally Identifiable Information Policy (PII Policy)
   Saya akan tampilkan berdasarkan data berikut :
    1. Username : inimyusername
    2. Full Name (masking) : Mich*** And*** Hus***
    3. Email (masking) : mye***@iniemailcontoh.com
    4. Error Message : “Create order error”

Traces

Traces adalah kegiatan untuk melihat jejak perjalanan suatu request dari awal sampai akhir. Saya menemukan tools OpenTelemetry bisa digunakan dan efektif untuk traces, metrics, dan logs.

Misal :

Masalah checkout order nomor 3 lambat

Trace : checkout nomor 3

Step / Span :
- API Gateway (5 ms)
- Order Service (20 ms)
- Payment Services (3500 ms) → Lambatnya di sini
- PostgreSQL (50 ms)

Contoh lain :
Payment Services -> call Bank API -> Timeout
Berarti dari trace, call bank API adalah penyebab timeout.

Catatan :

Istilah	Makna	Contoh (Analogi)
Metric	Seberapa sering / seberapa parah?	Jumlah paket per hari.
Log	Apa yang terjadi?	Catatan kejadian.
Trace	Terjadi di bagian mana?	Resi pengiriman suatu paket.

Definisi SLI berdasarkan SLO

Bagian ini membahas bagaimana cara mengukur suatu layanan dikatakan bagus atau jelek. Menentukan kapan alert diinfokan ke pengawas agar melakukan tindakan dan menentukan apa tindakan yang harus dilakukan berdasarkan masala hapa yang terjadi.

Formula :

1. Availability
   Versi ringkas :

   ```
   SLI (dalam persen) = request sukses / total request * 100%
   ```

   Versi detail (misal service payment):

   ```
   sum(rate(http_requests_total{service="payment",status!~"5.."}[5m]))
   /
   sum(rate(http_requests_total{service="payment"}[5m]))
   ```

2. Latency
   Versi ringkas :

   ```
   SLI = p95 response time
   ```

   Versi detail (misal service payment):

   ```
   histogram_quantile(
   0.95,
   sum(rate(http_request_duration_seconds_bucket{service="payment"}[5m])) by
   (le)
   )
   ```

3. Error Rate
   Versi ringkas :

   ```
   SLI = request 5xx / total request
   ```

Versi detail :

```
sum(rate(http_requests_total{service="payment", status=~"5.."}[5m]))
/
sum(rate(http_requests_total{service="payment"}[5m]))
```

Dimana : request 5xx itu request yang dapat kode response error

Burn Rate

Dengan SLO yang menetapkan minimal availability di 99.9% per 30 hari maka didapat error budget (batas wajar kegagalan sistem) di 0.1% per 30 hari.

Kalau dikonversi ke dalam menit, artinya didapat maksimal down / kegagalan sistem di 43 menit dalam 30 hari.

Burn Rate sendiri lebih kea rah kecepatan menghabiskan error budget.

Kalau dalam 30 hari hanya boleh 43 menit down, artinya dalam 1x down rata-rata hanya 1.4 menit. Jika pakai rumus penyeberan merata dimana setiap hari terjadi down time maksimal 1.4 menit.

Maka saya buat tablel panduan berikut untuk alert berdasarkan burn rate :

Batas Window	Burn Rate	Apa yang terjadi?	Level Urgensi
Dalam 1 jam	14 menit (jatah 10 hari)	Outage Service Besar	Urgent
Dalam 1 jam	1.4 menit (jatah 1 hari)	Berarti 1 hari down habis jatah 24 hari. Masih Outage Besar.	Urgent
Dalam 24 jam (1 hari)	2.8 menit (jatah 2 hari)	Down pelan-pelan	Medium (harus di tangani maksimal 3 hari operasional)
Dalam 24 jam (1 hari)	1.4 menit (jatah 1 hari)	Masih aman. Tetapi masih berpotensi bahaya jika ada anomali besar (misal down kritis).	Medium (harus dicek maksimal 14 hari operasional) – Bisa gunakan sistem tiket (antrian)
Dalam 24 jam (1 hari)	0.7 menit (jatah 0.5 hari)	Masih aman. Tapi sebaiknya dipantau berkala.	Low (Perlu dipantau berkala) – Bisa gunakan sistem tiket (antrian)

Runbook

Runbook lebih membahas panduan Langkah demi Langkah tentang apa yang harus dilakukan ketika suatu insiden terjadi.

Misal ada laporan konsumen tidak bisa membayar. Muncul error, bahwa pembayaran gagal.

1. Triage (5 menit pertama)
    1. Periksa apakah koneksi Internet yang mengarah kea rah API pembayaran berhasil terhubung? Misal via Ping.
    2. Periksa apakah service payment hidup?
    3. Periksa apakah ada error via dashboard monitoring?
    4. Kalau ada error, cek sumber errornya. Apakah dari DB/App/Response eksernal (response eksternal = API pihak ke 3) ?
    5. Apakah resource server penuh? Lihat di dashboard
2. Mitigasi Cepat
    1. Tambahkan resource jika tersedia
    2. Hapus log yang tidak dipakai
    3. Kalau errornya dari aplikasi dan tidak tahu solve nya bagaimana tetapi diketahui baru-baru ini baru melakukan deploy. Maka rollback.
    4. Kalau masalahnya payment provider. Biasanya sudah disiapkan link backup ke provider payment lain. Bisa lakukan switch payment provider dulu.
    5. Kalau masalahnya di network, bisa restart router atau cek firewall apakah keblokir / hang di sana.
3. Eskalasi (Bisa paralel)
    1. Hubungi tim infra network jika network
    2. Hubungi tim payment provider jika error di payment provider
4. Recovery Check
    1. Latency jaringan Kembali normal
    2. Error log app/DB/eksternal sudah tidak muncul atau di bawah SLO
5. RCA (Root Cause Analysis - setelah incident)
   Buat reporting dengan tujuan mencari sumber masalah utama / akar masalah dari suatu kejadian/insiden. Bahan evaluasi dan antisipasi ke depannya.

   Contoh dari pengalaman sendiri :
   Server hang -> restart -> hidup -> selesai -> terulang hang lagi setelah beberapa hari -> restart lagi -> hidup -> dan seterusnya

   Dengan RCA, kit acari sebab hang. CPU terindikasi 100% setelah beberapa hari, setelah dicek ternyata aplikasi membuka transaksi database tapi lupa commit/rollback/menutup transaksi. Solusinya diperbaiki aplikasinya agar dipastikan setelah buka transaski DB dilakukan commit/rollback. Masalah selesai dengan jelas dan tidak terulang.

Dashboard Sketsa

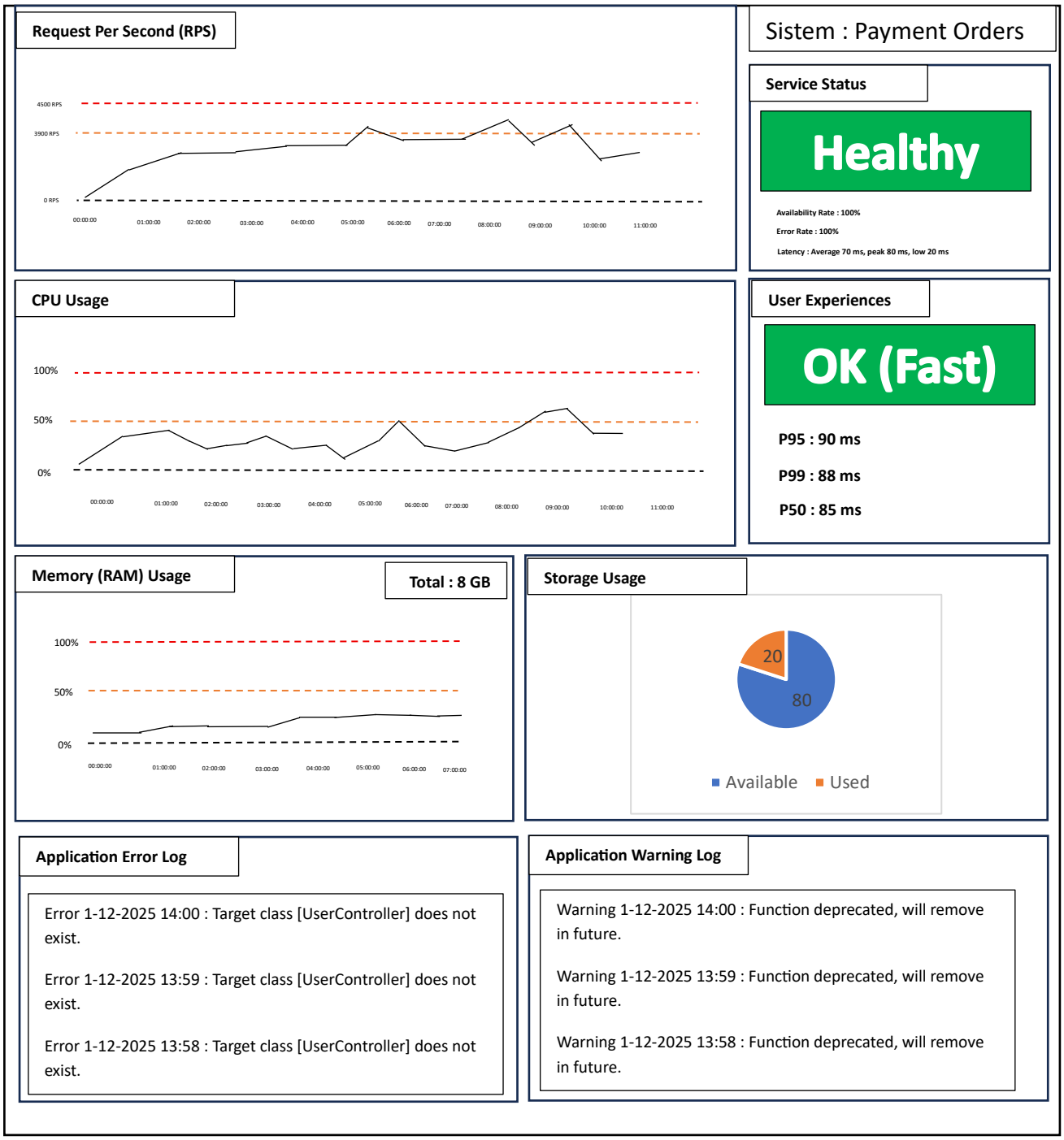
Daftar Widget :

1. Widget Service Status (Traffic Light)
    - ● Aman / Healthy (Warna Hijau)
    - ● Bermasalah / Need Attention (Warna Orange)
    - ● Tidak Aman / Unhealthy (Warna Merah)

    Diambil dari availability, error rate, dan latency. Tujuannya agar cepat mengindetifikasi ada masalah atau tidak.

2. Widget Traffic Request Per Second (RPS)
    - Ada diagram garis / grafik garis
    - Garis pertama threshold (atau batas RPS yang dirancang yaitu maksimal 4500 RPS)
    - Garis kedua yaitu tingkat RPS saat ini (Current RPS)
    - Tujuannya agar kelihatan apakah current RPS mendekati batas maksimal dan kelihatan anomali RPS dalam bentuk spike (bentuk lancip di grafik garis)
3. Widget P95, p99, p50
    - Sistem mungkin up dan bisa diakses, tetapi user experience tidak terlihat jelas apakah lambat atau cepat aksesnya.
    - P95 berarti misal 95% user akses sistem < 300 ms artinya masih baik, hanya 5% yang di atas itu. Tapi kalau misal hasilnya 2 detik, artinya ga sesuai SLO ya berarti jelek.
    - Dibuat varian p99 dan p50 agar kelihatan juga
    - Bisa juga dibuat warna warna indicator seperti sebelumnya biar cepat dipahami ketika monitoring ● (Aman), ● (Bermasalah), ● (Tidak aman)

Sketsa


Request Per Second (RPS)
4500 RPS
3900 RPS
0 RPS
00:00:00
01:00:00
02:00:00
03:00:00
04:00:00
05:00:00
06:00:00
07:00:00
08:00:00
09:00:00
10:00:00
11:00:00
Sistem : Payment Orders
Service Status
Healthy
Availability Rate : 100%
Error Rate : 100%
Latency : Average 70 ms, peak 80 ms, low 20 ms
CPU Usage
100%
50%
0%
User Experiences
OK (Fast)
P95 : 90 ms
P99 : 88 ms
P50 : 85 ms
Memory (RAM) Usage
Total : 8 GB
Storage Usage
20
80
Available
Used
Application Error Log
Error 1-12-2025 14:00 : Target class [UserController] does not exist.
Error 1-12-2025 13:59 : Target class [UserController] does not exist.
Error 1-12-2025 13:58 : Target class [UserController] does not exist.
Application Warning Log
Warning 1-12-2025 14:00 : Function deprecated, will remove in future.
Warning 1-12-2025 13:59 : Function deprecated, will remove in future.
Warning 1-12-2025 13:58 : Function deprecated, will remove in future.