อธิบาย

1. Dockerfile

เพื่อลด process step ทำเป็น Multi stage

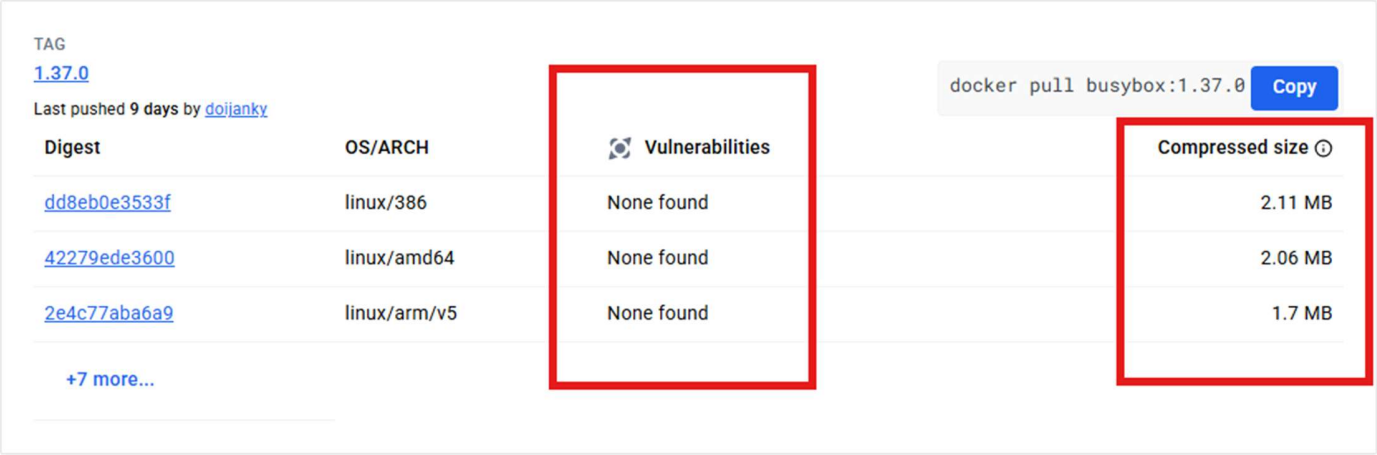
image ที่นำมาใช้ สิ่งที่ควรคำนึงถึง

1.1 มุมมองด้านความปลอดภัย ควรไม่มีช่องโหว่เลยหากเป็นไปได้ โดนตรวจสอบที่ https://hub.docker.com/_/busybox/tags ดูที่ column "Vulnerabilities" หรือใช้ software scan-image

1.2 มุมมองของขนาด image ควรมีขาดเล็ก และมี Feature เท่าที่จำเป็น ต้องใช้

1.3 ***หากมีการ scan code review ด้วย sonarqube ด้วยจะดีมาก***

https://hub.docker.com/_/busybox/tags




```
FROM golang:1.21.4 AS builder
WORKDIR /app
COPY go.mod ./
RUN go mod tidy
COPY main.go .
RUN go build -o hello-api
# FROM gcr.io/distroless/base-debian12
FROM busybox:1.37.0
WORKDIR /app
COPY --from=builder /app/hello-api /app/
EXPOSE 8080
CMD ["/app/hello-api"]
```

ส่งงาน ข้อ 1

Answer\1-Answer\Dockerfile

2. REPO Registry

คำสั่งสำหรับ Build และนำขึ้น บน docker Repo

for production Environment พร้อมใส่เลข tag version กำกับ

docker buildx build -t docker.io/wachira90/proof-concept:prod1 . --no-cache

for dev Environment พร้อมใส่เลข tag version กำกับ

docker buildx build -t docker.io/wachira90/proof-concept:dev1 . --no-cache

คำสั่ง buildx เป็นคำสั่งที่ใหม่กว่า images บางอย่างหากต้องการไม่เก็บ cache build ให้ใส่คำสั่ง --no-cache

การนำ image ขึ้น Repo (ต้อง login ก่อน โดยคำสั่ง "docker login")

docker push docker.io/wachira90/proof-concept:prod1

docker push docker.io/wachira90/proof-concept:dev1

3.Kustomize manifest

มีการสร้าง 2 Environment dev และ prod ใช้ namespace ตามชื่อ

kubectl create ns dev

kubectl create ns prod

ใน folder “Answer\3-Answer\kustomiz” ให้ run command เพื่อ verbose ทดสอบ value

kubectl apply -n prod -k overlays/prod --dry-run=client -o yaml

kubectl apply -n dev -k overlays/dev --dry-run=client -o yaml

คำสั่งนำไปใช้งาน

kubectl apply -n prod -k overlays/prod

kubectl apply -n dev -k overlays/dev

อธิบาย จุดสังเกตุ

DEV

kustom-dev.192-168-1-10.nip.io

มี AutoScale POD CPU และ RAM แยกไว้ หากเกินการใช้งาน 90 % (min 1 max 2), ScaleDown 1 min

image แยก ENV => wachira90/proof-concept:dev1

PROD

kustom-prod.192-168-1-10.nip.io

มี AutoScale POD CPU และ RAM แยกไว้ หากเกินการใช้งาน 90 % (min 1 max 6), ScaleDown 1 min

image แยก ENV => wachira90/proof-concept:prod1

4. GKE cluster Terraform

อธิบาย

ไฟล์

0-locals.tf

ปรับแต่ง project และ project_id , API region ควรเลือก ใกล้ไทย เพราะ ใช้งาน ในไทย

1-providers.tf

โหลด provider ที่จำเป็นสำหรับ project

2-apis.tf

load local each service API

3-vpc.tf

create vpc

4-subnets.tf

create subnet

5-nat.tf

nat network

6-firewalls.tf

firewalls เปิดใช้งานเท่าที่จำเป็น

7-gke.tf

GKE demo location => asia-southeast1-a

8-gke-nodes.tf

autoscaling => total_min_node_count = 2, total_max_node_count = 5

 # การใช้งาน

 # โหลด provider ที่จำเป็น

terraform init

 # test validate

terraform validate

 # ตรวจสอบการเปลี่ยนแปลง

terraform plan

 # สร้าง iac

terraform apply -auto-approve

5. การใช้ Workload Identity

การใช้ service account

```yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: file-read
  annotations:
    iam.gke.io/gcp-service-account: file-read@proof-concept-452415.iam.gserviceaccount.com
```

```yaml
      labels:
        app: myapp
    spec:
      serviceAccountName: file-read
      containers:
        - name: myapp
          image: docker.io/wachira90/proof-concept:prod1
          env:
            - name: TZ
              value: "Asia/Bangkok"
```

6.argocd

file มีการแยก Application dev และ prod และ มีการทำ permission

```yaml
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: hello-api-prod
  namespace: argocd
spec:
  destination:
    namespace: prod
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/wachira90/xxxxxxx.git
    targetRevision: prod
    path: .
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

```yaml
---
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: argocd-deploy-permission
  namespace: prod
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  description: deploy prod
  sourceRepos:
    - "https://github.com/wachira90/xxxxx.git"
  destinations:
    - namespace: "prod"
      server: "https://kubernetes.default.svc"
  clusterResourceWhitelist:
    - group: ""
      kind: "Service"
    - group: "apps"
      kind: "Deployment"
    - group: "autoscaling"
      kind: "HorizontalPodAutoscaler"
    - group: "networking.k8s.io"
      kind: "Ingress"
```

การใช้งานแบบ GUI BROWSER

 # การติดตั้ง ARGOCD

kubectl create ns argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

kubectl patch svc argocd-server -n argocd -p '{"spec": {"type":"LoadBalancer"}}'

```
kubectl port-forward svc/argocd-server -n argocd 8080:443

# ( access svc port 443 to external  https://x.x.x.x:8080 )

kubectl port-forward svc/argocd-server -n argocd --address x.x.x.x 8080:443
```

GET PASSWORD LOGIN

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d
```

การใช้งานแบบ CLI

ติดตั้ง

```
curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
```

permission

```
chmod +x /usr/local/bin/argocd
```

login argocd

```
argocd login argocd.example.com --insecure
```

การ add git

argocd repo add https://github.com/wachira90/xxxxx.git --username your-username --password your-password

สร้างแอปพลิเคชัน ArgoCD ที่ชี้ไปยังที่เก็บ Git

```
argocd app create hello-app \
  --repo https://github.com/wachira90/xxxxx.git \
  --path app-folder \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace prod
```

อธิบาย

hello-app ชื่อแอปพลิเคชัน

--repo URL ของที่เก็บ Git

--path เส้นทางภายในที่เก็บที่มีการแสดง Kubernetes (Helm charts, Kustomize หรือ YAML)

--dest-server คลัสเตอร์ที่แอปจะถูกปรับใช้ (https://kubernetes.default.svc สำหรับภายในคลัสเตอร์)

--dest-namespace เนมสเปซสำหรับแอปพลิเคชัน

7.CI/CD gitlab-runner

FILE => Answer\7-Answer\.gitlab-ci.yml

GITLAB

CI/CD จะใช้ gitlab-runner ทำ CI/CD โดยจะทำแบบ Semi Auto เพื่อป้องกันการลั่น

หากเป็นการใช้งานใน Production (รูปแบบ trigger ทั่วไปจะใช้แบบ merg branch) ทางผมจะเลือกแบบ file change "version.txt" โดยเนื้อหาภายในจะเป็น "prod1" , "prod2" ...

หลักการทำงาน

1. จับไฟล์ "version.txt" Change ใน branch main และ อ่านค่ามาใส่ใน tag version build
2. pull code โดยเก็บ หลักฐาน sha hash วันที่ต่างๆ เพื่อตรวจสอบย้อนหลัง
   - git pull --no-edit
   - git status
   - git show -s
3. build image
4. re login repo
5. push image
6. การ set image tag ใน Kube

****ความคิดเห็นส่วนตัว และ ข้อแนะนำเพิ่มเติม****

- **ระบบควรมีการทำ image cache เพื่อความรวดเร็ว**
- **ระบบควรมี Monitor Tools Grafana, Alert Condition ฯลฯ**
- **ระบบควรมี Tracing Tools, jaeger ฯลฯ**
- **ระบบควรมีการ Review Scan Code, Review Scan Image**
- **ระบบควรมีรอบการทำ MA ฝั่ง Software ตรวจหาช่องโหว่ ฯลฯ**
- **การสำรองข้อมูล Backup แบบ 321**
- **อาจมีระบบ SIEM หากจำเป็น และ การทำ proxies ฝั่งขาเข้า**
- **ตรวจสอบความปลอดภัยเบื้องต้น ตามหลัก OWASP และ VA/scan Pentest**

วชิร ดวงดี

2025-03-05

7solutions