

อธิบาย



1. Dockerfile

เพื่อลด process step ทำเป็น Multi stage

image ที่นำมาใช้ สิ่งที่ต้องคำนึงถึง

- 1.1 มุมมองด้านความปลอดภัย ควรไม่มีช่องโหว่เลยหากเป็นไปได้ โดนตรวจสอบที่ https://hub.docker.com/_/busybox/tags ดูที่ column "Vulnerabilities" หรือใช้ software scan-image
- 1.2 มุมมองของขนาด image ควรมีขนาดเล็ก และมี Feature เท่าที่จำเป็น ต้องใช้
- 1.3 ***หากมีการ scan code review ด้วย sonarqube ด้วยจะดีมาก***

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

TAG 1.37.0 Last pushed 9 days by dojanky		docker pull busybox:1.37.0 Copy	
Digest	OS/ARCH	 Vulnerabilities	Compressed size 
dd8eb0e3533f	linux/386	None found	2.11 MB
42279ede3600	linux/amd64	None found	2.06 MB
2e4c77aba6a9	linux/arm/v5	None found	1.7 MB
+7 more...			

```
1 FROM golang:1.21.4 AS builder
2 WORKDIR /app
3 COPY go.mod ./
4 RUN go mod tidy
5 COPY main.go .
6 RUN go build -o hello-api
7 # FROM gcr.io/distroless/base-debian12
8 FROM busybox:1.37.0
9 WORKDIR /app
10 COPY --from=builder /app/hello-api /app/
11 EXPOSE 8080
12 CMD ["/app/hello-api"]
13
```

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 จุดสังเกต

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

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

```
12     labels:
13         app: myapp
14     spec:
15         serviceAccountName: file-read
16         containers:
17             - name: myapp
18               image: docker.io/wachira90/proof-concept:prod1
19               env:
20                 - name: TZ
21                   value: "Asia/Bangkok"
```



6. argocd

file มีการแยก Application dev และ prod และ มีการทำ permission

```
1  ---
2  apiVersion: argoproj.io/v1alpha1
3  kind: Application
4  metadata:
5      name: hello-api-prod
6      namespace: argocd
7  spec:
8      destination:
9          namespace: prod
10         server: https://kubernetes.default.svc
11     project: default
12     source:
13         repoURL: https://github.com/wachira90/xxxxxxx.git
14         targetRevision: prod
15         path: .
16     syncPolicy:
17         automated:
18             prune: true
19             selfHeal: true
```

```

1  ---
2  apiVersion: argoproj.io/v1alpha1
3  kind: AppProject
4  metadata:
5    name: argocd-deploy-permission
6    namespace: prod
7    finalizers:
8      - resources-finalizer.argocd.argoproj.io
9  spec:
10   description: deploy prod
11   sourceRepos:
12     - "https://github.com/wachira90/xxxxx.git"
13   destinations:
14     - namespace: "prod"
15       server: "https://kubernetes.default.svc"
16   clusterResourceWhitelist:
17     - group: ""
18       kind: "Service"
19     - group: "apps"
20       kind: "Deployment"
21     - group: "autoscaling"
22       kind: "HorizontalPodAutoscaler"
23     - group: "networking.k8s.io"
24       kind: "Ingress"

```

การใช้งานแบบ GUI BROWSER

การติดตั้ง ARGOCD

kubectl create ns argocd

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

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

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

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

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

GET PASSWORD LOGIN

```
kubectrl -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/argocd/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 เพื่อป้องกันการล้น

หากเป็นการทำงานใน 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 ข้อเสนอแนะเพิ่มเติม****

- 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 ตรวจสอบช่องโหว่ ฯลฯ
- การสำรองข้อมูล Backup แบบ 321
- อาจมีระบบ SIEM หากจำเป็น และ การทำ proxies ฝั่งขาเข้า
- ตรวจสอบความปลอดภัยเบื้องต้น ตามหลัก OWASP และ VA/scan Pentest