

1. Register the Container generated by the DockerFile with Cloud Build / Artifacts.

First of all we are going to create the docker repository on google artifact registry. So we enable the service, and then create a new repo, format: Docker, model: standard.

The screenshot shows the 'Edita el repositorio' (Edit repository) page in Google Artifact Registry. The page has a left sidebar with a back arrow and the title 'Edita el repositorio'. The main content area contains the following fields and options:

- Nombre** (Name): A text input field containing 'docker-repo'.
- Formato** (Format): A group of radio buttons with the following options:
 - ☒ Docker
 - ☐ Maven
 - ☐ npm
 - ☐ Python
 - ☐ Apt
 - ☐ Yum
 - ☐ Canalizaciones de Kubeflow
 - ☐ Go [VISTA PREVIA](#)
- Modo** (Mode): A group of radio buttons with the following options:
 - ☒ Estándar
 - ☐ Remoto [VISTA PREVIA](#)
 - ☐ Virtual [VISTA PREVIA](#)
- Tipo de ubicación** (Location type): A group of radio buttons with the following options:
 - ☒ Región
 - ☐ Multirregional
- Región** (Region): A dropdown menu showing 'europe-southwest1 (Madrid)'.
- Descripción** (Description): A text input field.
- Etiquetas** (Tags): A button labeled '+ AGREGAR UNA ETIQUETA'.
- Encriptación** (Encryption): A section with a warning message: 'Este recurso está encriptado de forma predeterminada con una clave administrada por Google. Si quieres administrar tu encriptación, puedes usar una clave administrada por el cliente en su...'.

The first approach consist on using Cloud Build to connect to a GitHub repo which contains all the files needed to build the image with the vue app.

But this plan fails as we do not have sufficient permissions to configure the GitHub repo with the Cloud Build service.

```

❗ Habilitar servicios: ha
cloudbuild.googleapis.com,secretmanager.googleapis.com
aT1xuiOR6oBEt2v08788vh0BHgUPrG
Not found or permission denied for service(s):
secretmanager.googleapis.com. Help Token: AT1-
yKGvLkxdUGirU33qIEJUe0Ru4Bj8pMzhKdJxS5TV9zgVpu4SZ
uWDP0FwQxRmd0BQqOGGtZqGc-
04sYWypAKbGYBkb8vUm7tMydo9katwh
REINTENTAR

❗ Habilitar servicios: ha
cloudbuild.googleapis.com,secretmanager.googleapis.com
aT1xuiOR6oBEt2v08788vh0BHgUPrG
Not found or permission denied for service(s):
secretmanager.googleapis.com. Help Token: AT1-
yKFErpgvt4euDjs4YIGuUI05IGZjXzGo3kk-
4VqIE1NOMzvFq3RwNm_cWqaENay_VndHTd9_s4wXCSMRILW
We530cYKxfKNDR_byni0I5oeE
REINTENTAR

✅ Habilitar servicio: hace 1 hora
secretmanager.googleapis.com
aT1xuiOR6oBEt2v08788vh0BHgUPrG

❗ Habilitar servicios: ha
cloudbuild.googleapis.com,secretmanager.googleapis.com
aT1xuiOR6oBEt2v08788vh0BHgUPrG
Not found or permission denied for service(s):
secretmanager.googleapis.com. Help Token: AT1-
yKFhTy0ZaUEUMSakb4ID_b-
DCmegvdZQcPmHdYhhyizB09xGumdrCBx8-
M7y4jMyXJQ5X5zHnENFolBwzp2MBWsbNCzCLNZzIqHGqIU2p

```

So a second approach has been applied. This one consists on cloning the github repo in our local machine, build the image from the Dockerfile provided and push the image to the already docker repository created in Google Artifacs.

Firstly we have to login and enable permissions to push images from our local repo to the Google repository.

Configure gcloud with registry:

```
gcloud auth configure-docker \
    europe-southwest1-docker.pkg.dev
```

Then login to the gcloud:

```
gcloud auth login
```

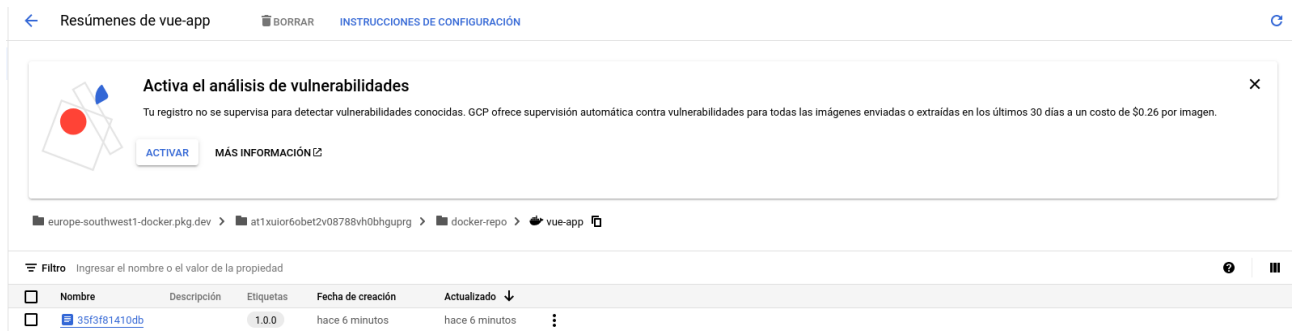
Finally we build locally our image with the following command:

```
docker build -t
europe-southwest1-docker.pkg.dev/at1xuior6obet2v08788vh0bhguprg/docker-repo/vue-
app:1.0.0 .
```

And push the generated image to our google artifact registry:

docker push

europe-southwest1-docker.pkg.dev/at1xuior6obet2v08788vh0bhguprg/docker-repo/vue-app:1.0.0



Another option is to configure our project in the gcloud cli client and use Cloud Build service to build the image and push it to the artifactory. This way we do not have to use docker locally, instead we use Cloud Build.

gcloud config set project at1xuior6obet2v08788vh0bhguprg

gcloud builds submit --region=global --tag

europe-southwest1-docker.pkg.dev/at1xuior6obet2v08788vh0bhguprg/docker-repo/vue-app:1.0.0

```
6f429902848d: Pushed
1.0.0: digest: sha256:b5bc896b2c2e011af981d216f0b6d9b006e84b2d9aef9fce90f00af82d7338c8 size: 2206
DONE
-----
ID              IMAGES              CREATE_TIME          DURATION  SOURCE                                     STATUS
df4ca346-334b-4599-a3a2-206060f8b17 2023-03-25T11:40:28+00:00 2M12S gs://at1xuior6obet2v08788vh0bhguprg_cloudbuild/source/1679744426.590851-d8cca988a9f44d6abc3c8cf8ee1f0461.tgz europe-southwest1-docker.pkg.dev/at1xuior6obet2v08788vh0bhguprg/docker-repo/vue-app:1.0.0 SUCCESS
```

2. Generating a YAML file for Docker Composer.

Now it's time to create our compose file in order to run the application:

```
version: "3"
services:
  web:
    container_name: vue-app
    image: vue-app:1.0.0
    ports:
      - "3000:3000"
```

From this yaml file using kompose we can create the yaml deployment files, for the service and the deployment objects.

We are using version 3 of docker-compose, as we are going to run only 1 container we instance 1 service called web, the container created will have the name vue-app, the image which creates the container is the one in the registry, vue-app:1.0.0.

3. Generate the Terraform files in order to have the infrastructure as code and be able to deploy with Kubernetes.

Now it's time to provision the infrastructure required to deploy our app. First of all we are going to create a VPC and then a Kubernetes cluster all of that using Terraform.

By creating a VPC, we create an environment where our cluster will reside, so it is isolated from other resources.

Before creating all this infra, we have to configure some things: enable Compute engine API and install gcloud auth plugin.

```
gcloud services enable servicemanagement.googleapis.com
servicecontrol.googleapis.com cloudresourcemanager.googleapis.com
compute.googleapis.com container.googleapis.com containerregistry.googleapis.com
cloudbuild.googleapis.com
```

`sudo apt-get install google-cloud-sdk-gke-gcloud-auth-plugin`

Now we can start provisioning our infrastructure. In our terraform configuration file we define a resource VPC and a GKE resource, define all required variables and providers. Then we initialize the environment with terraform init and follow the workflow , terraform plan and terraform apply. So our infra is created:

```
Apply complete! Resources: 3 added, 0 changed, 2 destroyed.

Outputs:

kubernetes_cluster_host = "34.66.247.223"
kubernetes_cluster_name = "atlxiuor6obet2v08788vh0bhguprg-gke"
project_id = "atlxiuor6obet2v08788vh0bhguprg"
region = "us-central1"
```

To connect to our cluster we must configure kubectl client

```
Setting up google-cloud-sdk-gke-gcloud-auth-plugin (423.0.0-0) ...
albert@albert-B250-HD3P:~/dev/mediamarkt/learn-terraform-provision-gke-clusters$ gcloud container clusters get-credentials $(terraform output -raw kubernetes_cluster_name) --
region $(terraform output -raw region)
WARNING: Accessing a Kubernetes Engine cluster requires the kubernetes commandline
client [kubectl]. To install, run
$ gcloud components install kubectl

Fetching cluster endpoint and auth data.
kubeconfig entry generated for atlxiuor6obet2v08788vh0bhguprg-gke.
```

Then confirm that we have access to the cluster:

```
ip: 35.192.169.244
albert@albert-B250-HD3P:~/dev/mediamarkt/terraform-deploy$ kubectl get ns
NAME                STATUS    AGE
default             Active   101m
kube-node-lease     Active   101m
kube-public         Active   101m
kube-system         Active   101m
kubernetes-dashboard Active   35m
```

Now that we have created our cluster inside a VPC its time to deploy the application as a pod in our kubernetes cluster.

This could be done with a simple kubectl apply -f <file_name.yaml> but we are going to use terraform to deploy the deployment. We are going to deploy a deployment instead of a pod to give robustness to the app, as our deployment will have 2 replicas. Then it supports more traffic and in case one replica falls, we will continue giving service.

So we need another terraform configuration file for our deployment, inside it we are going to define a new resource of type `kubernetes_deployment`.

Also we need to define the providers used, in this case `gcloud` and `kubernetes`.

Note: it is important to mark that, at the time of deploying the app, the kubelet could not pull the image from the registry, so we had to add in our `gke` terraform conf file the following lines in order to kubelet have permissions to pull the image:

```
oauth_scopes = [  
    "https://www.googleapis.com/auth/cloud-platform"  
]
```

(All Terraform files are provided in the solution)

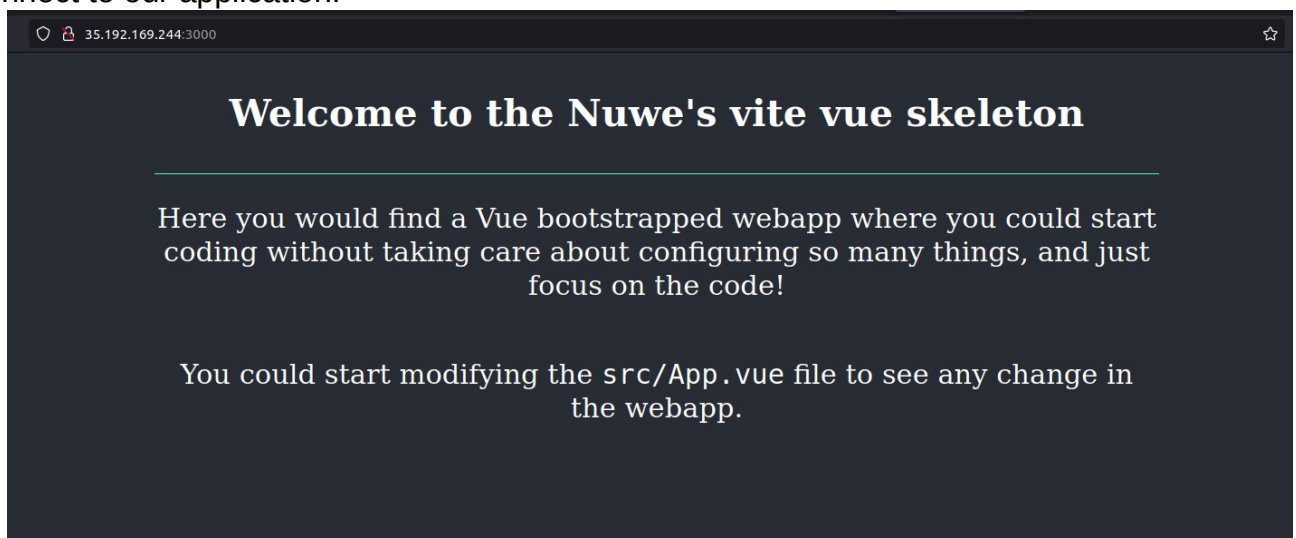
Then we have to make our app accesible to the internet, so we have 2 options here:

- 1- Provision a service of type `ClusterIP` and an ingress in order to expose our pod.
- 2- Provision a service of type `LoadBalancer`

We opt for the second one, as we only need to provision one resource. So we proceed to deploy our `LoadBalancer` service type with terraform.

As done before, we add the code to instanciate our service in the same conf file where we defined our deployment.

Finally accesing our `LoadBalancer` external IP providing the port of the app, we can connect to our application:



4. Answer the question to check the understanding of the Minimum Least Privilege in the Roles assignment.

For a Devops role, it is a must to have permissions of all services used along the CI/CD process. This includes having permissions to the secret manager. As the process followed in this documentation is nice, but it would be better if we can connect from our GitHub repository, which is a common practice in companies, instead of having to clone locally the repo and then use Cloud Build to build and push the image. In other words, if we want our GitHub repo to be the SCM, we need to configure it with our CI tool, in this case Cloud Build.

Focusing on the question, the roles for a devops in order to create clusters in kubernetes should be: roles/container as this role enables permissions to administer clusters and access to all API objects.

For a Finance workers, the correct role should be roles/billing. As it brings full access to the billing accounts and administer them.

DOCUMENTATION

Terraform GKE module:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/container_cluster#argument-reference

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/container_node_pool