**Detailed Documentation**

This entire task is divided into following sub-tasks:

**1. Create public repo on git**

**2. Clone at local directory**

**3. Create Node.js App**

**4. Build and Run Docker Image**

**5. Load the static webpage**

**6. Create a GCP Project**

**7. Enable GCP API Services**

**8. Create GKE cluster using terraform**

**9. Transfer a local Docker image to Google Cloud Platform (GCP) Artifact Registry after creating it**

**10. Connect Artifact Registry with a Google Kubernetes Engine (GKE) cluster and deploy the pod**

**11. Install Helm**

**12. deploy prometheus on GKE cluster**

**13. Configure Prometheus RBAC Permissions**

**14. Create a Prometheus Kubernetes service**

Now let's deep down into each step:

This guide combines creating a Node.js application, building a Docker image, deploying it to Google Container Registry (GCR), and potentially deploying it to a Kubernetes cluster.

**Preparation:**

The guide assumes you have the following tools installed:

**Docker**

**Git**

**Node.js (and npm)**

**Google Cloud SDK (gcloud)**

**Terraform**

**Kubernetes**

# 1: Create a Public GitHub Repository

**Generate SSH Key Pair (Optional)**

If you don't have an SSH key pair for secure communication with GitHub, generate one using the following command in your terminal:

ssh-keygen -t rsa -b 4096 -C "my-email@gmail.com"

Follow the on-screen prompts to save the key securely.

**Add Public Key to GitHub**

Go to your GitHub account settings and navigate to "SSH and GPG keys" -> "New SSH key".

Enter a descriptive title (e.g., "Your Laptop") and paste the contents of your public key file (usually located at ~/.ssh/id\_rsa.pub).

Click "Add SSH key".

**Verify SSH Agent (Optional)**

Run the following commands in your terminal to verify your local SSH agent is running and has the private key loaded:

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id\_rsa

# 2. Clone Your Repository

Create a new directory for your project and clone your GitHub repository using the following command (replace <username> and <repository\_name> with your details):

git clone git@github.com:<username>/<repository\_name>.git

# 3. Create a Node.js App

**Create Node.js Project Directory**

Navigate to the cloned repository directory and create a directory for your Node.js application:

cd <repository\_name>

mkdir nodeapp

cd nodeapp

**Initialize Project**

Initialize the project with npm init -y to create a package.json file.

**Install Express.js**

Install Express.js, a popular Node.js web framework, using the following command:

npm install express

**Create Index File**

Create a file named index.js in your project directory and add the following code to define a simple HTTP server:

**Update package.json (Optional)**

To add a script for easily starting the server, update your package.json file with the following line under the scripts section:

JSON

"scripts": {

"start": "node index.js"

}

# 4. Build and Run Docker Image

**Create a .dockerignore File**

Create a file named .dockerignore in your project directory to specify files or directories to exclude from the Docker image. This helps keep the image size smaller.

node\_modules

npm-debug.log

**Build the Image**

Build a Docker image from your project directory using the following command:

docker build -t node-app .

This creates a Docker image named node-app.

**Run the Container**

Run the Docker container in detached mode (background) and map port 8080 of the host machine to port 3000 of the container:

docker run -d -p 8080:3000 --name heynodeapp node-app

You can verify if the container is running with docker ps.

# 5. Load the static webpage

Open your web browser and navigate to http://localhost:8080 (or the IP address of your machine followed by port 8080) to see the "welcome to my website" message.

# 6. Create a GCP Project

Make sure you’ve GCP account. Create a project in it.

# 7. Enable GCP API Services

Before we create any resource in GCP, make sure the APIs are enabled.

# 8. Create GKE cluster using terraform

The config files that we’ve used are main.tf, variables.tf, terraform.tfvars. Main.tf file have the cloud provider config details. Variables.tf have the variables declaration. Terraform.tfvars have the actual usage of variables declared in variables.tf file. Once these config files are created in terraform to create GKE cluster, run the terraform config files using commands as below:

Terraform init

Terraform validate

Terraform plan

Terraform apply

Check the logs for any error. If no error, check if the cluster is created or not.

# 9. Transfer a local Docker image to Google Cloud Platform (GCP) Artifact Registry after creating it

1.Authenticate with Google Cloud: Open a terminal or command prompt and run:

gcloud auth login

This will open a browser window where you can log in with your GCP credentials. Once authenticated, you will be able to access your GCP resources.

2. Configure the GCP Project and Region: Set your default GCP project and region:

gcloud config set project [PROJECT\_ID]

gcloud config set compute/region [REGION]

Replace [PROJECT\_ID] with your project’s ID and [REGION] with the region closest to your location (e.g., us-central1).

3. Create an Artifact Registry Repository: Create a repository in Artifact Registry to store your artifacts:

gcloud artifacts repositories create [REPO\_NAME] \

--repository-format=docker \

--location=[REGION] \

--description="My Artifact Repository"

Replace [REPO\_NAME] with a name for your repository.

4. Tag the Local Docker Image: Tag your local Docker image with the desired name and version:

docker tag [IMAGE\_NAME] gcr.io/[PROJECT\_ID]/[REPO\_NAME]/[IMAGE\_NAME]:[TAG]

Replace [IMAGE\_NAME], [PROJECT\_ID], [REPO\_NAME], and [TAG] with appropriate values.

5. Push the Tagged Image to Artifact Registry: Push the tagged image to Artifact Registry:

docker push gcr.io/[PROJECT\_ID]/[REPO\_NAME]/[IMAGE\_NAME]:[TAG]

This command will upload your Docker image to Artifact Registry.

6. Verify the image has been successfully pushed by checking the Artifact Registry console or using the gcloud artifacts command:

gcloud artifacts images list --repository=my-repo --location=us-central1

# 10. Connect Artifact Registry with a Google Kubernetes Engine (GKE) cluster ad deploy the pod

1. Verify GKE version: Ensure your GKE cluster is running a supported version. Starting with GKE 1.14, clusters can pull images from Artifact Registry without additional configuration. For earlier versions, you’ll need to configure imagePullSecrets.

2. Create an imagePullSecret: In your GKE cluster, create an imagePullSecret named artifact-registry using the following command:

kubectl create secret docker-registry artifact-registry \

--docker-server=https://LOCATION-docker.pkg.dev \

--docker-email=SERVICE-ACCOUNT-EMAIL \

--docker-username=\_json\_key \

--docker-password="$(cat KEY-FILE)"

Replace LOCATION with the region where your Artifact Registry repository is located, SERVICE-ACCOUNT-EMAIL with the email address of the service account used by your GKE cluster, and KEY-FILE with the path to the JSON key file for that service account.

3. Grant permissions to the service account: In the Google Cloud project where Artifact Registry is located, grant the required permissions to the service account used by your GKE cluster. This can be done using the IAM console or the gcloud command-line tool:

gcloud artifacts repositories add-iam-policy-binding REPOSITORY \

--location=LOCATION --member=SERVICE-ACCOUNT-EMAIL --role=roles/artifactregistry.reader

Replace REPOSITORY with the ID of your Artifact Registry repository and LOCATION with the region where it’s located.

4. Configure the GKE cluster: Update your GKE cluster to use the artifact-registry imagePullSecret. You can do this by editing the cluster’s configuration file or by using the gcloud command-line tool:

gcloud container clusters update CLUSTER\_NAME --image-pull-secrets=artifact-registry

Replace CLUSTER\_NAME with the name of your GKE cluster.

5. Verify image pull: Once the configuration is complete, verify that your GKE cluster can pull images from Artifact Registry by running a command like:

kubectl run my-pod --image=LOCATION-docker.pkg.dev/REPOSITORY/MY-IMAGE:TAG

This should successfully pull the image from Artifact Registry and deploy the pod.

# 11. Install Helm

Before you install Helm, you must start your Minikube Kubernetes using the following command:

minikube start --driver=docker

Installing Helm on Windows:

choco install Kubernetes-helm

# 12. Deploy Prometheus on GKE cluster

First, we’ll search for the Prometheus Helm Charts. To search for the Prometheus Helm, run this command:

helm search hub prometheus

To install Prometheus Helm Chart on Kubernetes Cluster, run this helm install command:

helm install prometheus prometheus-community/prometheus

To view the deployed Kubernetes resources, run the following kubectl command:

kubectl get all

Output will enlist all the objects of the prometheus such as pods, deployments, services, daemonsets, replicasets and statefulsets.

# 13. Configure Prometheus RBAC Permissions

Before deploying Prometheus and Grafana to production, you’ll configure that RBAC privileges using a ClusterRole. You’ll then use a ClusterRoleBinding object to bind this ClusterRole to a ServiceAccount

To add your RBAC Permissions to your Cluster, run this command:

kubectl apply -f rbac-permissions.yaml

# 14. Create a Prometheus Kubernetes service

Prometheus Kubernetes Service will expose the Prometheus application that is running to allow external access.

kubectl apply -f prometheus-service.yaml