INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 7, Issue 4 - V7I4-1158)
Available online at: https://www.ijariit.com

Multi regional GKE cluster implementation on GCP using Terraform

Aditya Ashok Illur
adityaaillur.is17@rvce.edu.in
RV College of Engineering, Bengaluru, Karnataka

Aaryaman T P Katoch
aaryamantpk.cs17@rvce.edu.in
RV College of Engineering, Bengaluru, Karnataka

Poornima Kulkarni
poornimapk@rvce.edu.in
RV College of Engineering, Bengaluru, Karnataka

Manas M. N.
manasmn@rvce.edu.in
RV College of Engineering, Bengaluru, Karnataka

ABSTRACT

With the increase of threats and attacks, it's more necessary than ever to be prepared, and implementing cloud endpoints for decryption and authorisation is getting increasingly difficult. Kubernetes is one of the finest methods for mitigating the risk of multi-regional failures. This paper describes ways to develop effective and efficient ways to achieve it with the best practices to build the global Kubernetes presence for the cloud authorisation endpoint with guidance to build infrastructure and DevOps practices that's necessary to make it operate in multi-region, high-availability presence and leveraging kubernetes and spanner. Also, automation using GitHub Style CI-CD

Keywords— *Endpoints, Decryption, Kubernetes, Cloud Authorisation, Multi-Region, Spanner, CI-CD*

1. INTRODUCTION

Google - Google as a consumer firm, over the past 18 years Google has been working hard to organise the world's information and make it widely accessible and valuable. They've created products such as Search, Android, Chrome, Maps, Play, YouTube, and Gmail, all now with over a billion users. However, it's not just the scale that's interesting about these global products. What's interesting is the infrastructure—Google's core technology infrastructure that's built-in order to achieve the mission. All of their products are powered by that infrastructure.

Google Cloud Platform(GCP)- Is built on the same infrastructure that powers Google. Super Flexible, Superior Economics, Always available storage, Fastest and Most reliable Network and Robust. GCP is broad, with a variety of services, as shown above exposing capabilities via REST, CLI Console. Application runtime services range from PaaS, to IaaS, to containers, Storage services, Networking & analytics and core infrastructure offered through Google's global network, with highest levels of security and reliability. Here deployment will be application in multi-cluster in different zones.

2. METHODOLOGY

As we move on to the implementation, we can have six different regions according to the locations required. The aim here is to deploy the applications in various locations and have a wider range of its availability of the services deployed through applications. This method in this paper has a generalised and detailed approach which can incorporate from one application to hundreds of working applications, which is all automated for creating, accessing, changing, deploying and improving the infrastructure anytime and anywhere and any number of times with ease.

Assuming we have to implement the infrastructure for six different regions as per the choice of locations as mentioned below, the resources can be deployed into the cloud(GCP) with the help of terraform.

- Asia-Northeast1 - Tokyo
- Asia-Northeast2 - Osaka
- Europe-West1 - Belgium
- Europe-West2 - London
- Us-Central1- Iowa
- Us-West3- Salt Lake City

A. Terraform Structure

The deployment of resources can be done through terraform. The files can be divided into 4 directories i.e, global, modules, regions and resources. Each leaf directory will have six files namely, backend.tf, main.tf, outputs.tf, providers.tf, terraform.tfvars, variables.tf and versions.tf. The GCS Buckets will be linked with all the backend.tf files, main.tf will contain the modules, outputs.tf will have the the code for desired outputs, provider.tf will have google as the provider, the .tfvars file will contain the values to be passed into the variables, variables.tf will have the variables declared and versions.tf will contain the terraform version and provider version can be specified. Global directory will contain 5 leaf directories - cloudbuild, gcs, iam, networking and spanner.

(a) Cloudbuild: Here the main.tf file will contain the modules for triggers and the tfvars file will have values for all the variables in main.tf.
(b) GCS (Google Cloud Storage): The main.tf file will contain the modules for 3 regions i.e, Asia, Europe, US and .tfvars file will have the projectid, region and storage classes defined.
(c) IAM (Identity and Access Management): Within this directory there can be a "service account" leaf directory where main.tf can include service accounts modules with the spanner service account modules and the .tfvars file will have the roles specified for those modules.
(d) Networking: Here there will be Private DNS, Firewall Rules, Nat Gateway and VPC Subnet leaf Directories, where there will be 6 subnets for 6 GKE Autopilot Clusters mentioned in the .tfvars file. IP CIDR Ranges for those 6 subnets are mentioned in the GKE Autopilot Cluster section below.
(e) Spanner: This includes main.tf containing 3 modules written for 3 spanner instances i.e, each instance for US, Europe and Asia projectid, number of nodes, labels, *deletion protection flag = true* and *ddl.*

The "Module" directory will contain the modules for cloudbuild, cloud dns, cloud load-balancer, cloud storage, cloud sql, compute Engine, data proc, GKE, iam, memorystore, networking, spanner and spanner database.

In "Regions" directory, it will have 6 directories according to 6 regions where respective main.tf files will be having the GKE private cluster values like projectid, region, network, subnetworks, zone etc. Flags like *http_load_balancing*, *enable_private_nodes*, *enable_private_endpoints* etc as per use case.

Coming to the last directory "Resources" will have two leaf directories to store Keys and Scripts. This would be the terraform code structuring. All of this can be easily deployed on the GCP through terraform.

B. GKE AutoPilot Cluster

Autopilot is an advanced version of the usual standard kubernetes cluster but here the provisioning and managing of the cluster's underlying infrastructure like nodes and node-pools are done automatically by the GKE itself. This makes it an optimised cluster and having things done automatically like scaling, maintenance, updation of clusters etc. by GKE rather than human manual intervention.

While creating the Autopilot cluster, we will have to pass the IP Classless Inter-Domain Routing (CIDR) values for Pod ranges and Service Ranges. The sample worked out CIDR ranges has to be calculated for IP addresses of six VPC subnets and in GKE cluster - six control plane IP's, six pod range IP's and six service range IP's. Sample IP's CIDR ranges are worked out as below, where the primary subnet CIDR is the subnet IP address.

Region	Location	Primary subnet CIDR	GKE Service Ranges	GKE Master Ranges	GKE Pod Ranges
asia-northeast1	Tokyo	10.1.0.0/20	10.1.16.0/20	10.1.32.0/28	10.1.64.0/18
asia-northeast2	Osaka	10.2.0.0/20	10.2.16.0/20	10.2.32.0/28	10.2.64.0/18
europe-west1	Belgium	10.3.0.0/20	10.3.16.0/20	10.3.32.0/28	10.3.64.0/18
europe-west2	London	10.4.0.0/20	10.4.16.0/20	10.4.32.0/28	10.4.64.0/18
us-central1	Iowa	10.5.0.0/20	10.5.16.0/20	10.5.32.0/28	10.5.64.0/18
us-west3	Salt Lake City	10.6.0.0/20	10.6.16.0/20	10.6.32.0/28	10.6.64.0/18

Fig 1. Networking - CIDR Ranges

C. Spanner

It is horizontally scalable, globally consistent, relational database service. It can scale upto 100s of 1000s of nodes. It is Automatic scaling, replicable and redundant. Since its multi-regional deployment and infrastructure is wider, Spanner would be a perfect fit.

D. Continuous Integration and Continuous Deployment (CI-CD Pipeline)

Git-Hub with cloud build can be used to achieve the CI-CD element, where the automation files will be uploaded on GitHub private repository. For security purposes 2-Factor Authentication can be set up. A total of five automation files will be required for CI-CD which is implemented using the Google Cloud service called Cloud Build. The repository will be linked to the six different region-specific triggers(a cloud build service) created and deployed using the terraform. Four of the files will be written in yaml language and the other one is Dockerfile. Here there can be any number of application files uploaded on Git-Hub repository, but in this scenario I am considering only one application file i.e index.html. So the files would be:

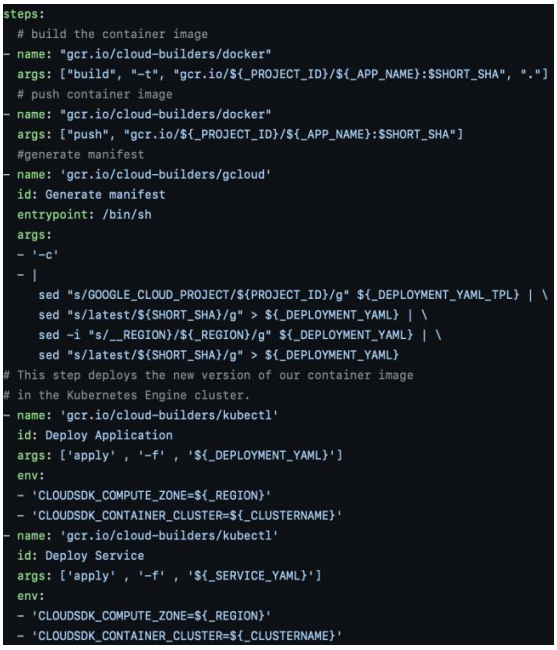
- Dockerfile
- index.html
- Cloudbuild.yaml
- Deployment.yaml
- Service.yaml

In Service.yaml we will have to pass the TCP Protocol, ports and target port. Dockerfile is used for managing and creation of the container image using cloudbuild.yaml file. The CloudBuild file will have three build steps shown in the figure 4.

- Build the container Image
- Push the container image to Google Container Registry (GCR)
- Generate the Image Manifest

The next section will have the code to deploy the container image from the GCR to the Kubernetes Engine Cluster. Here we would have to pass image name path, id of the step, arguments like ['apply', '-f',' <filename.yaml>'] which should be repeated for deployment.yaml, Service.yaml and for all the files written in the yaml format.

The values supplied to the variables in the cloudbuild.yaml file should be user defined variables which will have the following syntax - ${_<VARIABLE>}. The sample snippet of the cloud build file is shown below.



```
steps:
  # build the container image
- name: "gcr.io/cloud-builders/docker"
  args: ["build", "-t", "gcr.io/${_PROJECT_ID}/${_APP_NAME}:$SHORT_SHA", "."]
  # push container image
- name: "gcr.io/cloud-builders/docker"
  args: ["push", "gcr.io/${_PROJECT_ID}/${_APP_NAME}:$SHORT_SHA"]
  #generate manifest
- name: 'gcr.io/cloud-builders/gcloud'
  id: Generate manifest
  entrypoint: /bin/sh
  args:
  - '-c'
  - |
     sed "s/GOOGLE_CLOUD_PROJECT/${PROJECT_ID}/g" ${_DEPLOYMENT_YAML_TPL} | \
     sed "s/latest/${SHORT_SHA}/g" > ${_DEPLOYMENT_YAML} | \
     sed -i "s/__REGION}/${_REGION}/g" ${_DEPLOYMENT_YAML} | \
     sed "s/latest/${SHORT_SHA}/g" > ${_DEPLOYMENT_YAML}
# This step deploys the new version of our container image
# in the Kubernetes Engine cluster.
- name: 'gcr.io/cloud-builders/kubectl'
  id: Deploy Application
  args: ['apply' , '-f' , '${_DEPLOYMENT_YAML}']
  env:
  - 'CLOUDSDK_COMPUTE_ZONE=${_REGION}'
  - 'CLOUDSDK_CONTAINER_CLUSTER=${_CLUSTERNAME}'
- name: 'gcr.io/cloud-builders/kubectl'
  id: Deploy Service
  args: ['apply' , '-f' , '${_SERVICE_YAML}']
  env:
  - 'CLOUDSDK_COMPUTE_ZONE=${_REGION}'
  - 'CLOUDSDK_CONTAINER_CLUSTER=${_CLUSTERNAME}'
```

Fig 2. Sample cloudbuild.yaml file

The variables are declared in capitals and starting with underscore in the left column of the trigger and the corresponding values for each variable is specified in the right column of the trigger section. The user defined environment variables will be replaced with the actual values while creating the trigger where we can pass the inputs as, variables and values as shown below

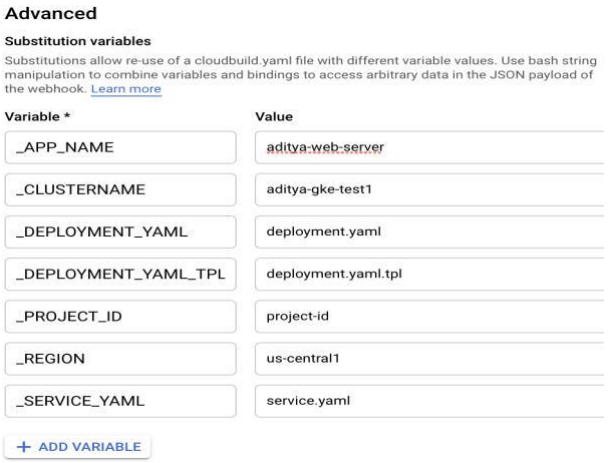


Fig 3. Substitution Variables Section in Trigger

Once These files are created on GitHub we can connect the repository to the 6 GKE Autopilots created one for each region by creating 6 CloudBuild Triggers once for each cluster which can also be created using terraform where we can pass the GKE Cluster name, Cloudbuild.yaml file name arguments. One of the triggers might resemble the illustration below.

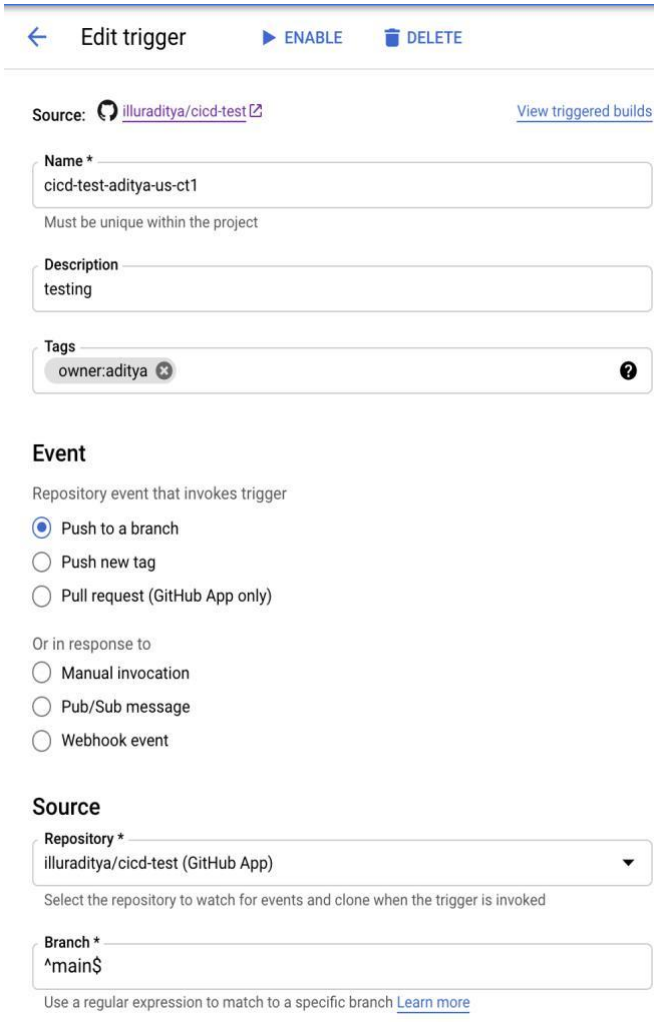


Fig 4. Sample trigger configuration

Once this is done, the CI-CD part is complete and whenever any code is changed in the GitHub the trigger in GCP Console will automatically run and the changes in the infrastructure will take place in the applications deployed on the Kubernetes Pods. We can then get the status by running commands 'Kubectl get pods' and the next command to get.

```
(base) adityaillur@Adityas-MacBook-Pro cloudbuild % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
web-server-aditya-57d575774-lttz2   1/1     Running   0          10m
(base) adityaillur@Adityas-MacBook-Pro cloudbuild %
```

Fig 5. Kubectl get pods

the services in the form of External IP of the application by doing the 'kubectl get services'.

```
(base) adityaillur@Adityas-MacBook-Pro cloudbuild % kubectl get service web-server-aditya
NAME              TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)        AGE
web-server-aditya LoadBalancer   10.5.29.82                    80      /TCP   3m22s
```

Fig 6. Kubectl get services

When we paste the obtained external IP on the browser the output will be shown with the updated code. Hence its an automatic deployment and management system of applications in multiple regions.

3. CONCLUSION

Through this method, we can have a structured way of deploying Resources across multiple regions which is achieved through terraform with the automatic management of the application which is achieved by the CI-CD process. Unit testing can be performed to check if all the applications deployed are accessible in all the locations. If some applications are not feasible to be deployed on the GKE Auto Pilot then such applications can be deployed on Standard GKE Cluster. Advantage of this approach is that the requirement of applications in multiple locations can run faster and have a smooth experience due to the drastic reduction of latency as there is Multi-Regional spanner instance running.

4. REFERENCES

[1] Orest Lavriv, Mykhailo Klymash, Ganna Grynkevych, Olga Tkachenko, Volodymyr Vasylenko, 'Method of cloud system disaster recovery based on "Infrastructure as a code" , IEEE Cloud automation Letters (Volume: 16, Issue: 1, Jan. 2019);

[2] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, Jianqing Fu, 'Cloud Storage as the Infrastructure of Cloud Computing, Published in**:** 2018 International Conference on Intelligent Computing and Cognitive Informatics

[3] A Generic Architecture for Composite Cloud as a ServiceMahmud Hossain, Rasib Khan, Shahid Al Noor, Ragib Hasan2016 IEEE 9th International Conference on Cloud Computing (CLOUD)Year: 2016 | Conference Paper | Publisher: IEEE

[4] Portable Autoscaler for Managing Multi-cloud Elasticity Yohan Wadia, Rohini Gaonkar, Jyoti Namjoshi 2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies

[5] Cloud Event Programming Paradigms: Applications and Analysis Garrett McGrath, Jared Short, Stephen Ennis, Brenden Judson, Paul Brenner 2016 IEEE 9th International Conference on Cloud Computing (CLOUD)

[6] A "No Data Center" Solution to Cloud Computing Tessema Mengistu, Abdulrahman Alahmadi, Abdullah

Albuali, Yousef Alsenan, Dunren Che 2017 IEEE 10th International Conference on Cloud Computing (CLOUD)

[7] Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform Jay Shah;Dushyant Dubaria 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)Year: 2019 | Conference Paper | Publisher: IEEE

[8] Yevgeniy., Brikman (2017). Terraform Writing Infrastructure as Code. O'Reilly Media. ISBN 9781491977057. OCLC 978667796

[9] Turnbull, James (2016). The Terraform Book. ISBN 9780988820258.

[10] Atkins, Martin (2017-11-16). "HashiCorp Terraform 0.11". HashiCorp Blog. Retrieved 2020-12-17

[11] HashiCorp. "HashiCorp Terraform - Provision & Manage any Infrastructure". HashiCorp: Infrastructure enables innovation. Retrieved 2020-04-15.

[12] O'Gara, Maureen (July 26, 2013). "Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud". SYS-CON Media. Archived from the original on September 13, 2019.

[13] Orest Lavriv, Mykhailo Klymash, Ganna Grynkevych, Olga Tkachenko, Volodymyr Vasylenko, 'Method of cloud system disaster recovery based on "Infrastructure as a code" , IEEE Cloud automation Letters (Volume: 16, Issue: 1, Jan. 2019);

[14] "Kubernetes 1.21: Power to the Community". Kubernetes. Retrieved 2020-04-09.

[15] Ellingwood, Justin (2 May 2018). "An Introduction to Kubernetes". *DigitalOcean*. Archived from the original on 5 July 2018. Retrieved 20 July 2018. One of the most important primary services is an API server. This is the main management point of the entire cluster as it allows a user to configure Kubernetes' workloads and organizational units. It is also responsible for making sure that the etcd store and the service details of deployed containers are in agreement. It acts as the bridge between various components to maintain cluster health and disseminate information and commands.

[16] "Don't Panic: Kubernetes and Docker". Kubernetes Blog. Retrieved 2020-12-22

[17] O "Container Attached Storage: A primer". *Cloud Native Computing Foundation*. 2018-04-19. Retrieved 2020-10-09.

[18] Gagliordi, Natalie. "Google releases Cloud TPU beta, GPU support for Kubernetes | ZDNet". *ZDNet*. Retrieved 2018-09-08.

[19] "Cloud DNS, VPN, HTTPS load balancing ... Google looks at rivals, thinks: Yeah, we'll do all that". Retrieved 2018-09-08.

[20] "Introducing Certified Kubernetes (and Google Kubernetes Engine!)". *Google Cloud Platform Blog*. Retrieved 2018-09-08.

[21] "Google Cloud SQL now Generally Available with an SLA, 500GB databases, and encryption". *Google Cloud Platform Blog*. Retrieved 2018-09-08.

[22] "An update on container support on Google Cloud Platform". *Google Cloud Platform Blog*. Retrieved 2018-09-08

[23] "Cloud Spanner: TrueTime and external consistency". *Google Cloud*. Retrieved 2020-11-24.

[24] Sacolick, Isaac (2020-01-17). "What is CI/CD? Continuous integration and continuous delivery explained". *InfoWorld*. Retrieved 2021-06-01.

