**Infrastructure**

* The infrastructure is built on GCP with following components – VPC, Subnet, GKE cluster, Cloud NAT and Cloud DNS.
* All the GCP components were built using terraform. The terraform scripts can be found in /terraform directory of the repo.
* Python flask is used for the sample web application which is deployed at domain [**http://guptaji.tech**](http://guptaji.tech)
* Jenkins is used for the CI/CD process. Jenkins pipeline gets triggered automatically through webhook once updated code is pushed in GitHub repository.
* Jenkins master, Jenkins slave/agent and python blue/green applications are deployed in same GKE Kubernetes cluster under different namespaces.
* Jenkins is deployed in GKE cluster using helm chart available in official open-source repo.
* Jenkins slave/agent is installed with tools required for building image of python app. Tools like python, pip, kubectl and gcloud are installed using custom docker image present in /Jenkins directory of the repo
* Docker hub registry is used for storing the images of python app and Jenkins slave/agent.

**Security**

* All the Kubernetes worker nodes are defined as private i.e. they do not have public IP address assigned. These worker nodes use cloud NAT gateway to connect to docker hub in order pull python app and Jenkins slave/agent image.
* The credentials for dockerhub registry, GCP service account keys are stored in Jenkins credentials, values for which cannot be retrieved by any process other than Jenkins agent.
* Can be implemented – For additional security SSL certificate can be installed in domain guptaji.tech to make customer to load balancer traffic encrypted.
* Can be implemented – Jenkins server/load balancer should not have public ip. That should only be exposed within the organization network using private ip address.
* Can be implemented – WAF protection like cloud armor/AWS WAF/AWS shield for DDOS can be applied at load balancer.
* Can be implemented – Users should be given least permission and access should be governed using Kubernetes roles, cluster roles, role bindings and cluster role bindings.

**Scalability and Availability**

* This is a zonal Kubernetes cluster but in production environment regional cluster should be used which will ensure that Kubernetes control plane and worker nodes are distributed in multiple AZs.
* Autoscaling is enabled in the cluster to ensure that cluster can handle additional load. The new worker node will be added in node pool once there is increase in CPU/memory.
* Can be implemented – Horizontal pod autoscaling can be enabled for the python application deployment so that we can adjust number of replicas based on load.
* Can be implemented – Pod disruption budget can be set for the python application deployment to ensure that minimum number of pod(s) is always available to serve the traffic. Useful in case of node maintenance or Kubernetes version upgrade.
* Can be implemented – Gitops solution like ArgoCD can be added for all deployments with self-healing enabled. This will ensure that there is no downtime in the application due to changes made by manual intervention. All Kubernetes deployments are governed by application and project yamls.

**Cost**

* The nodes in this cluster are E type nodes which are recommended for general purpose use. The cost can be optimized by evaluating the resources required and changing the node type to reduce the cost.
* There should be resource field in all deployment yamls to ensure usage of CPU and memory are within the limits.
* Can be implemented **–** The size of backup/contingency application – Blue or Green can be scaled down to 0 (replica = 0) once primary application deployed and healthy.