**Create a Kubernetes cluster which runs a simple go hello world application**

Created a sample go application and containerized and pushed to docker registry.

* docker build -t kube-app.
* docker tag kube-app shilpagokul95/hello-app:1.0.0
* docker login
* docker push shilpagokul95/hello-app:1.0.0

Created a deployment file in Kubernetes to deploy the containerized application using following command

* kubectl apply -f <deployment.yml>

Exposed the deployment as a service with type as NodePort using following command

* Kubectl expose deployment –port=9090 –target-port=8080 –name=hello-app-service –type=NodePort

Text

Description automatically generated

**How to make the service scalable?**

To manually scale the service, use the kubectl command or edit the deployment file as mentioned below. Horizontal pod autoscaling can also be enabled for autoscaling based on CPU consumptions

* Used kubectl scale command to scale the deployment to 4. “Kubectl scale –replicas=4 deployment/hello-app”
* The replicas can also be increased in the deployment file and reapplied which will in turn scale the number of pods to the mentioned count in deployment file

Graphical user interface, application

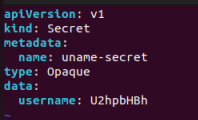
Description automatically generated

**What CI/CD pipeline to use?**

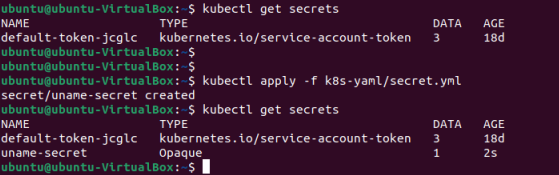
* Required files: main.go and main\_test.go file, dockerfile, deployment.yml and service.yml
* Configure Jenkins user to connect to the cluster
* Check**'**Specify another hook url**'** for GitHub configuration in Jenkins and add the URL shown in the repo’s webhooks
* Create the Jenkins pipeline job and select GitHub hook trigger for git scm polling and provide repository URL and credentials
* Configure Jenkins Credentials for GitHub and Docker Hub
* In Jenkins following steps are performed
  + Build – Build go binary and ensure that nothing is wrong in the build process
  + Test – Validate that the application works as expected
  + Publish- Build docker image and push it to registry
  + Deploy- Deploys to Kubernetes

**How to store or deploy secrets?**

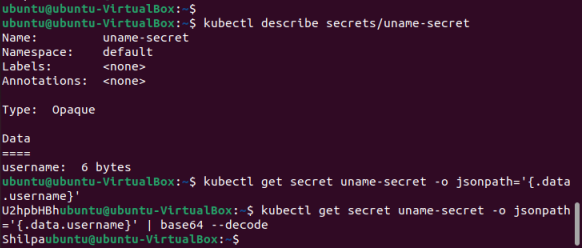
* Get the base64 encoded value of username
  + echo -n “shilpa” | base64
* Create a secret yaml and apply as shown below



* Apply secrets and confirm if the secrets are created



* Decode the secrets



* There are three main ways for a Pod to use a Secret:
  + As [files](https://kubernetes.io/docs/concepts/configuration/secret/#using-secrets-as-files-from-a-pod) in a [volume](https://kubernetes.io/docs/concepts/storage/volumes/) mounted on one or more of its containers.
  + As [container environment variable](https://kubernetes.io/docs/concepts/configuration/secret/#using-secrets-as-environment-variables).
  + By the [kubelet when pulling images](https://kubernetes.io/docs/concepts/configuration/secret/#using-imagepullsecrets) for the Pod.
* Trying option 2- Add as container environment variable. Add the env variable in the deployment and reapply. Login to the container and check if the env value is set
  + Kubectl exec -it <pod name> sh
  + echo $SECRET\_USERNAME

Text

Description automatically generated

Text

Description automatically generated

**How do you test how well your infrastructure scales? (When many requests come in)**

* Can perform load, endurance, and stress testing to ensure that the infrastructure scales properly
  + [Load test](https://www.techtarget.com/searchsoftwarequality/definition/load-testing)- To check how the system responds to a sudden increase in requests.
  + [Endurance test](https://www.techtarget.com/searchsoftwarequality/definition/Soak-testing)- To check how the system survives a constant, moderate load for longer duration of times. It can also be referred to as a soak test, referring to the long time the software spends under test.
  + [Stress test](https://www.techtarget.com/searchsoftwarequality/definition/stress-testing)- To check how the system responds under a heavy load, with an intent to find out the point at which system is stressed and ceases functioning.
* To avoid pod failures, can set memory and CPU limits in the deployment configuration or opt for Horizontal pod autoscaling which scales based on CPU consumption.

**How do you provide an SSL certificate to your service?**

* Create SSL certificate as secrets and mount it as volumes in the pod spec to access it from the containers.
  + Encode the SSL key and create secret
  + Specify the secret as volume mount in the pod spec to access the secrets
* To create a self-signed certificate for testing purposes, follow below steps
  + Generate a CA private key
    - *OpenSSL genrsa -out ca.key 2048*
  + Create a self-signed certificate, valid for 365 days.
    - *OpenSSL req -x509 -new -nodes -days 365 \*

*-key ca.key \  
-out ca.crt \  
-subj "/CN=yourdomain.com"*

* + Create TLS secret using kubectl command
    - *kubectl create secret tls my-tls-secret \  
      --key ca.key \  
      --cert ca.crt*