Hello Host IP Application

Project Definition :

Creating a greenfield OPS environment and deploy an application to displays IP address of the host machine.

System Architecture :

Application is running on a 2-tiers architecture - running Web-server and Application is different tiers with Load-balancer service for High Availability. In addition this provides additional security to Application server from being exposed to external access — running them behind private cluster IP Service.

Kubernetes Cluster is deployed with a NGINX server running Load Balancer service and 3 replicas of Nodejs Application Server with Cluster IP Service.

Application Server and Web-server images are created and deployed as Docker Images

Helm is used to create Application and Web server charts with parameters to automate orchestration of Kubernetes clusters running on Minikube.

Terraform code will automate the deployment of Kubernetes Cluster with IaC pipeline.

Minikube Dashboard to view the deployment

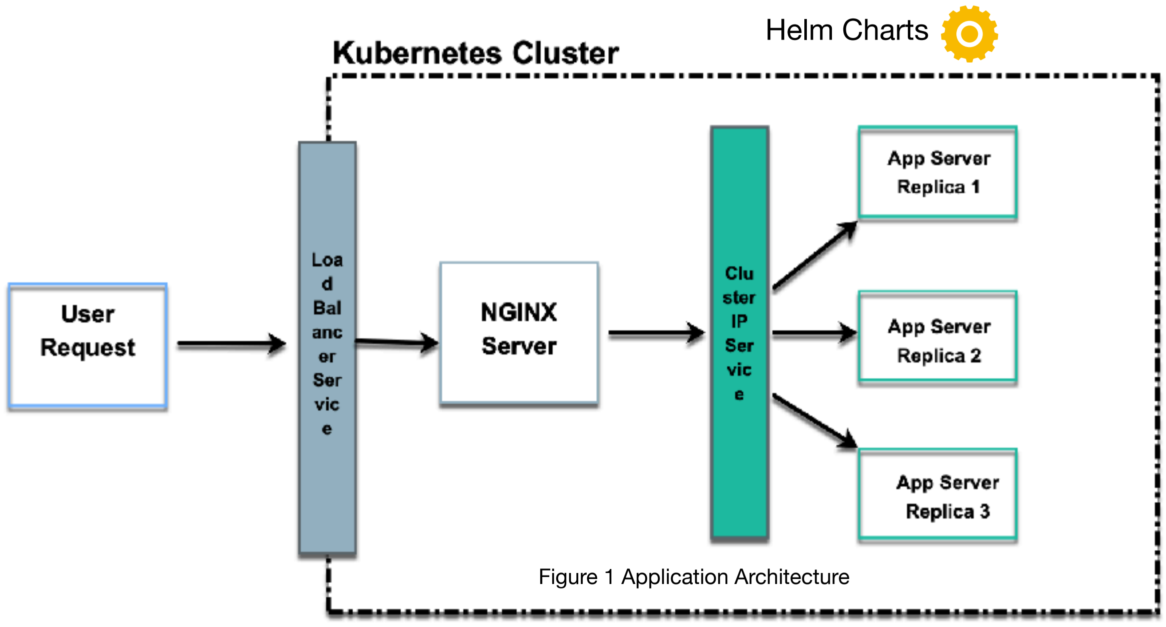


Figure 1 Application Architecture

Features :

High Availability - Application runs on 3 replicas increasing availability. Also increases Performance of application when heavy load.

Scale on demand - Rescaling is faster (change replica count in values.yaml) and requires no down-time

Self-healing - Kubernetes replaces pods in case of failure, upgrades with rolling updates and application can self heal in case node failure - supporting “zero-downtime” rolling updates

Portability - Run anywhere that has Kubernetes — whether in cloud AWS Cloud using EMR service, Google, Azure cloud, or on-premise datacenter. Currently application is running on on single system.

Automation - Using Helm charts and Terraform Devops tools automates the deployment across multiple environments and platforms.

Project Components :

1. Nodejs Application
2. NGINX Server
3. Docker Images
4. Kubernetes Cluster (MiniKube)
5. Helm Chart
6. Terraform

Approach : Developed first Application and Server deployments and run them successful, then converted to Helm Chart. Then created IaC with terraform.

Implementation Steps :

1. Nodejs application which return the IP address of host on which application is running
2. Application Server which runs Nodejs app script
3. App-server Docker image create for Nodejs application files to install Nodejs and run application server
4. Kubernetes cluster with Deployment, Service and pods created from the docker image
5. Application service run in minikube container

Webserver

6. WebServer - NGINX server which runs as a Loadbalancer with External IP
7. Web-server Docker image create for NGINX server files
8. Kubernetes cluster with Deployment, Service and pods created from the docker image
9. Application service run in minikube container

10. Helm Chart is used to orchestrate deployment of Kubernetes Cluster

11. Terraform to automate creation of Kubernetes Cluster. (Node : Terraform does not create Helm charts yet)

Technical Stack used in the Project :

For Application with Cluster IP services : Nodejs with express, networkInterfaces modules
For Web-server with Loadbalancer services : NGINX server

Docker for containerization of Application and Server deployments

Minikube cluster running in docker virtual machine - to deploy containerized application to Kubernetes cluster

Kubernetes(K8s) for automating deployment, scaling and management of Application and Server deployment containers

Helm for creating parameterized charts of Web-server and App-server applications and orchestration of Kubernetes cluster

Minikube dashboard - user interface to view Kubernetes deployments by traffic light
Terraform - to automate kubernetes deployment with Infrastructure as code tool

FOLDER STRUCTURE :

jvp-hostip-app (root folder)
--app (application folder)
----app.js (nodejs application code)
----Dockerfile
----package.json (nodejs package)
--server (server folder)
----default.conf (nginx server configuration file)
----Dockerfile

--helm
----templates
------app-deployment.yaml
------app-service.yaml
———values.yaml

Security

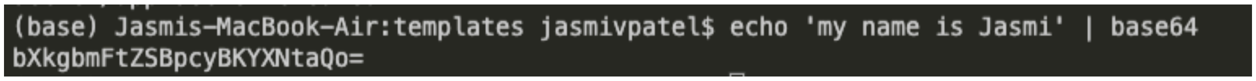
a.What Security considerations did you think about (secret storage, https, network security, etc)
I am using Secrets to pass security configuration (credentials and private keys) to application.

Secrets are similar to Config Map - and can be injected either using “environment variables” or “by mounting files”

Here I am using environment variables.

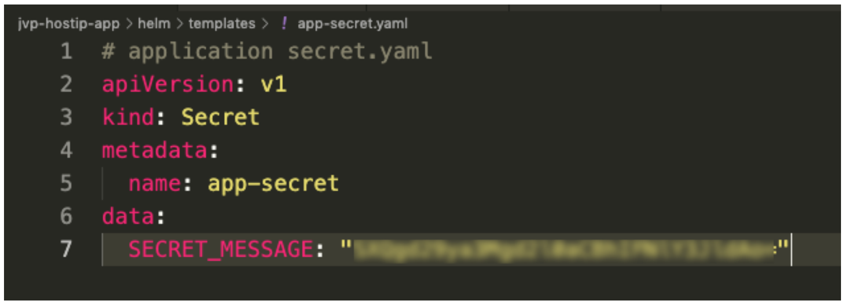
Secrets are distributed in pods running application and not written to physical storage keeping them secure, stored on master node as encrypted. We retrieve them in “Base64” encoded form.

Base64 encoding example :



```
(base) Jasmis-MacBook-Air:templates jasmivpatel$ echo 'my name is Jasmi' | base64
bXkgbmFtZSBpcyBKYXNtaQo=
```

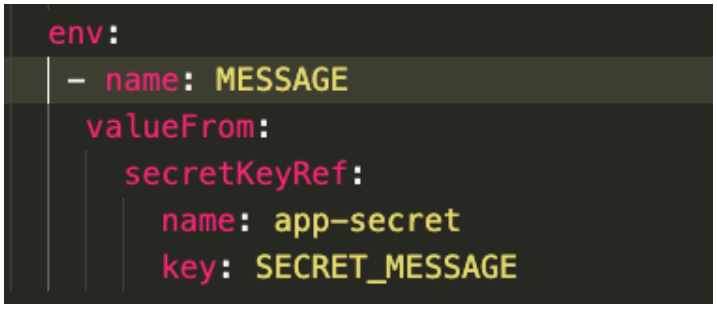
Secrets yaml file :



jvp-hostip-app > helm > templates > ! app-secret.yaml

```yaml
# application secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
data:
  SECRET_MESSAGE: "[redacted]"
```

Adding secrets as “Environment Variables” in app deployment file



```yaml
env:
- name: MESSAGE
  valueFrom:
    secretKeyRef:
      name: app-secret
      key: SECRET_MESSAGE
```

Ssecure application from common web attacks like DDOS, Cross-site scripting, SQL Injection by placing application servers with private “ClusterIP” behind LoadBalancer

With current design - we don’t want application server to be exposed directly to external traffic -- all request come through Loadbalancer.
Loadbalancer server can also be used for authentication and to funnel traffic through a single path. Also experimented with all three types of services - ClusterIP, NodePort and LoadBalancer.

I am also using container environment variables in values.yaml file.

I created Application **Readiness and Liveness Probes** to monitor health check of cluster services. Used in second version of application. Configured health-check within nodes application.

If deploying cluster on AWS - server credentials can be stored in “Secret Manager” or “Parameter Store”.

Loadbalancing service	Exposed to be external world
Clusterip service	Used by only othe rpods

Used Resource manifest --- controls deployment - what container to run and how many instance

---

How to build and use the project

Please refer to appendix for detailed step on Installations required for developing and running the project.

Without Terraform :

Steps :

1. Once code is created for app, create docker image in minikube docker container : and push it to docker hub

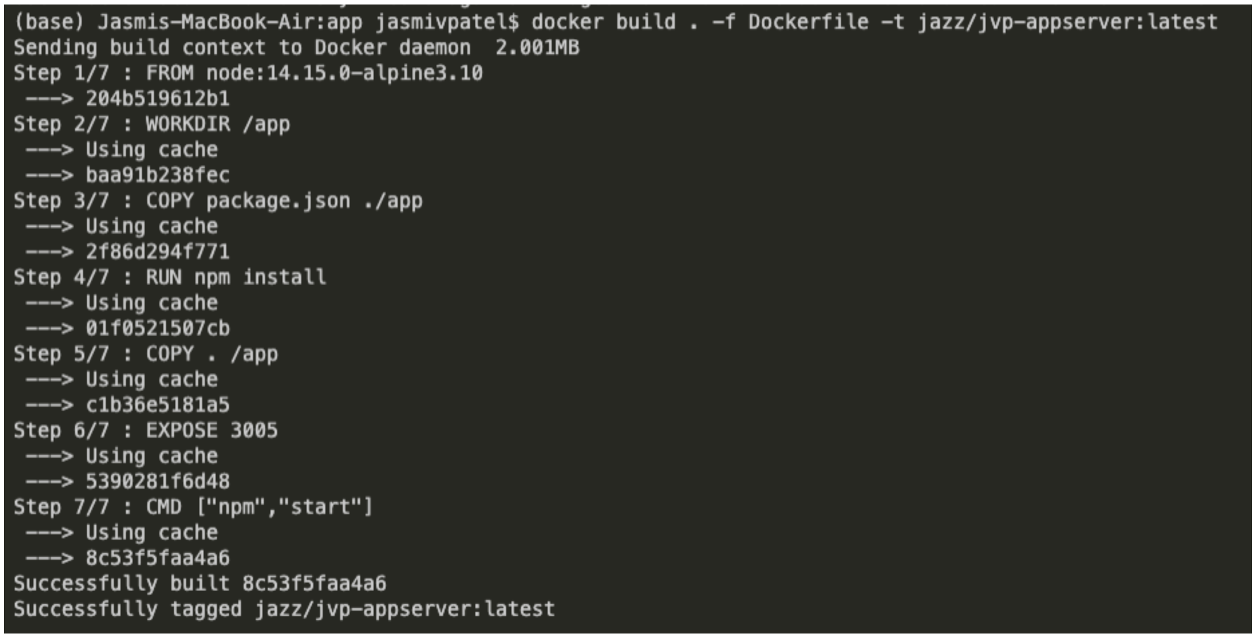
Root folder : *jvp-hostip-app*
Navigate to jvp-hostip-app/app folder and run command to create docker image

Docker login

run: docker login
run: eval $(minikube docker-env)
Cd app
run: docker build . -f Dockerfile -t jvp-appserver:latest

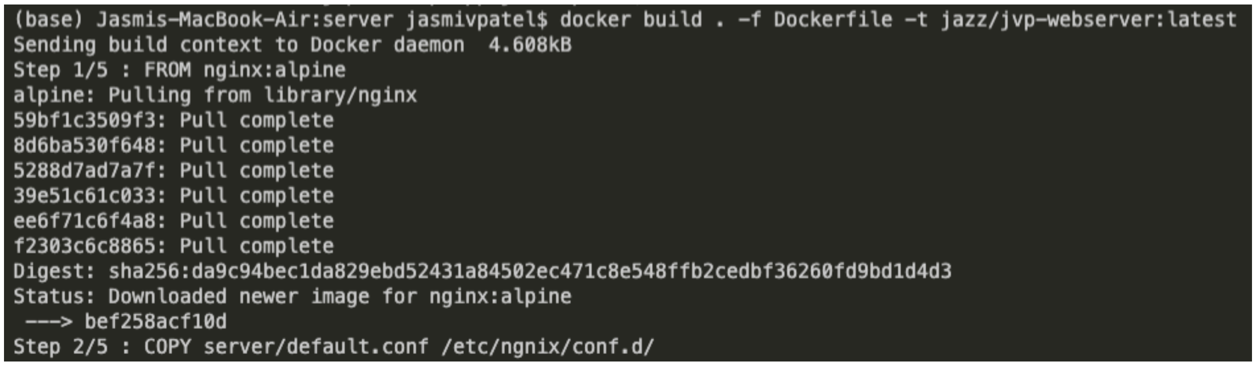
2. Once config is created for server, create docker image in minikube docker container :

Navigate to root folder : jvp-hostip-app



```
(base) Jasmis-MacBook-Air:app jasmivpatel$ docker build . -f Dockerfile -t jazz/jvp-appserver:latest
Sending build context to Docker daemon  2.001MB
Step 1/7 : FROM node:14.15.0-alpine3.10
 ---> 204b519612b1
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> baa91b238fec
Step 3/7 : COPY package.json ./app
 ---> Using cache
 ---> 2f86d294f771
Step 4/7 : RUN npm install
 ---> Using cache
 ---> 01f0521507cb
Step 5/7 : COPY . /app
 ---> Using cache
 ---> c1b36e5181a5
Step 6/7 : EXPOSE 3005
 ---> Using cache
 ---> 5390281f6d48
Step 7/7 : CMD ["npm","start"]
 ---> Using cache
 ---> 8c53f5faa4a6
Successfully built 8c53f5faa4a6
Successfully tagged jazz/jvp-appserver:latest
```

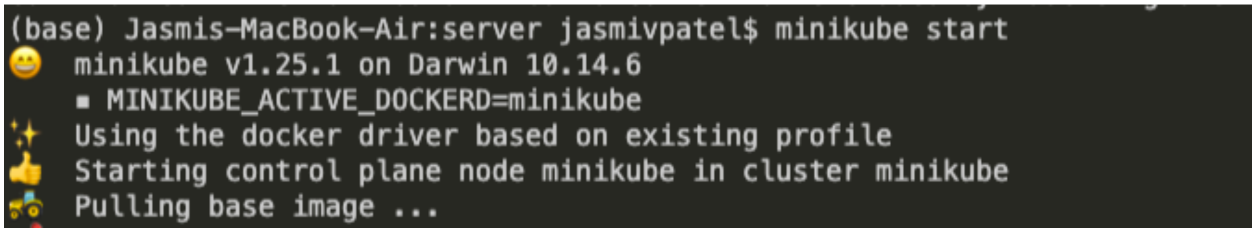
run: docker build . -f Dockerfile -t jvp-webserver:latest



```
(base) Jasmis-MacBook-Air:server jasmivpatel$ docker build . -f Dockerfile -t jazz/jvp-webserver:latest
Sending build context to Docker daemon  4.608kB
Step 1/5 : FROM nginx:alpine
alpine: Pulling from library/nginx
59bf1c3509f3: Pull complete
8d6ba530f648: Pull complete
5288d7ad7a7f: Pull complete
39e51c61c033: Pull complete
ee6f71c6f4a8: Pull complete
f2303c6c8865: Pull complete
Digest: sha256:da9c94bec1da829ebd52431a84502ec471c8e548ffb2cedbf36260fd9bd1d4d3
Status: Downloaded newer image for nginx:alpine
 ---> bef258acf10d
Step 2/5 : COPY server/default.conf /etc/ngnix/conf.d/
```

3. Confirm minikube is running

Run: Minikube start

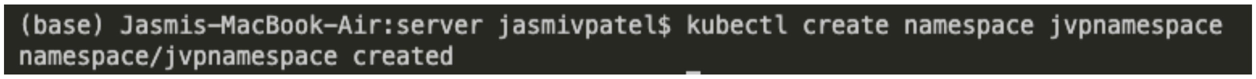


```
(base) Jasmis-MacBook-Air:server jasmivpatel$ minikube start
😄  minikube v1.25.1 on Darwin 10.14.6
    ▪ MINIKUBE_ACTIVE_DOCKERD=minikube
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
```

run: Minikube dashboard

4. Create kubectl namespace

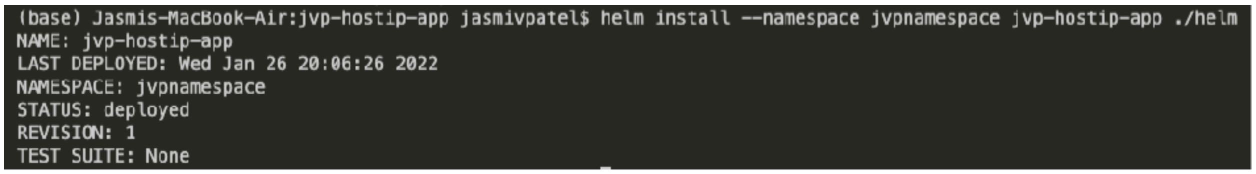
run: kubectl create namespace jvpnamespace



```
(base) Jasmis-MacBook-Air:server jasmivpatel$ kubectl create namespace jvpnamespace
namespace/jvpnamespace created
```

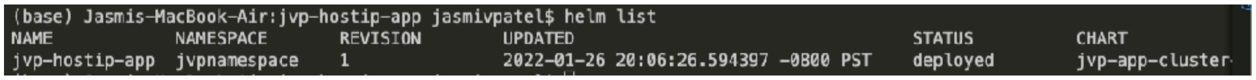
5. Run Helm chart to create cluster, deployment, pods and services

Confirm in root folder: jvp-hostip-app
run: helm install --namespace jvpnamespace jvp-hostip-app ./helm



```
(base) Jasmis-MacBook-Air:jvp-hostip-app jasmivpatel$ helm install --namespace jvpnamespace jvp-hostip-app ./helm
NAME: jvp-hostip-app
LAST DEPLOYED: Wed Jan 26 20:06:26 2022
NAMESPACE: jvpnamespace
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Confirm helm : helm list



```
(base) Jasmis-MacBook-Air:jvp-hostip-app jasmivpatel$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART
jvp-hostip-app  jvpnamespace    1               2022-01-26 20:06:26.594397 -0800 PST    deployed        jvp-app-cluster-
```

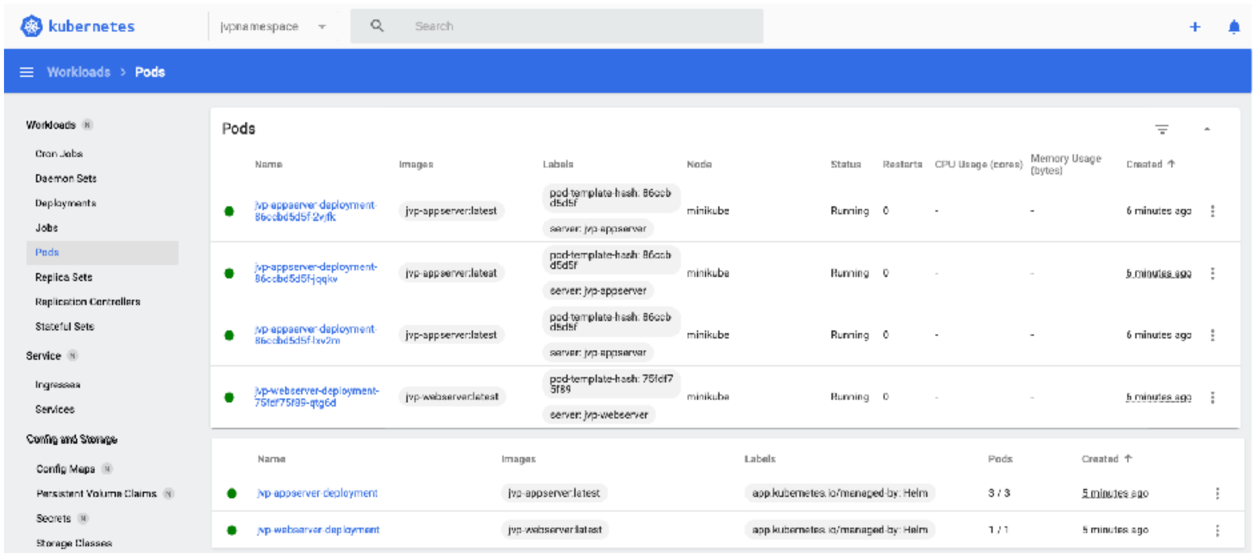
To upgrade :
Run:helm upgrade --namespace jvpnamespace jvp-hostip-app ./helm

6. Confirm deployment

Confirm pods created :

Run:Kubectl get pods

Or
Run : minikube dashboard
It will open web browser interface as below :



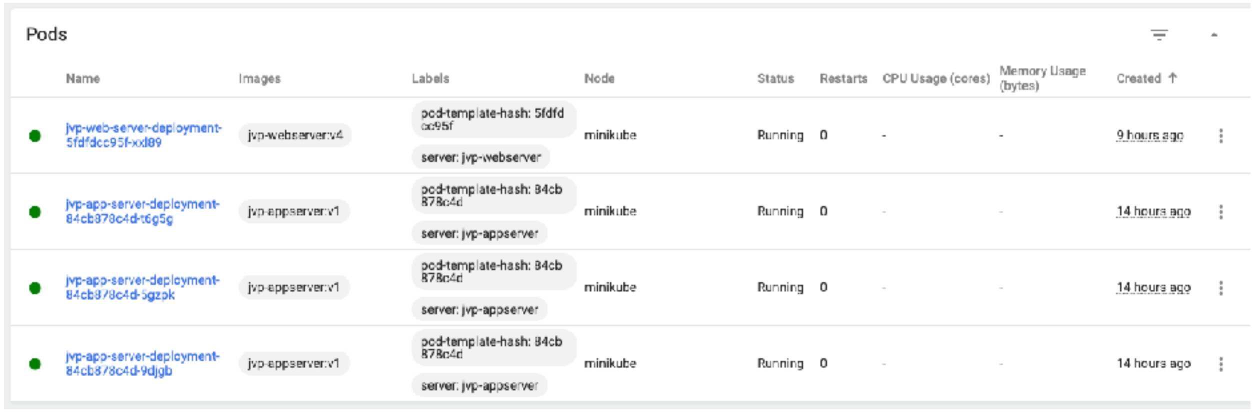
Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	jvp-appserver-deployment-86ccbd5d5f-2vjfk	jvp-appserver:latest	pod-template-hash: 86ccbd5d5f server: jvp-appserver	minikube	Running	0	-	-	6 minutes ago
●	jvp-appserver-deployment-86ccbd5d5f-jqqkv	jvp-appserver:latest	pod-template-hash: 86ccbd5d5f server: jvp-appserver	minikube	Running	0	-	-	5 minutes ago
●	jvp-appserver-deployment-86ccbd5d5f-lxv2m	jvp-appserver:latest	pod-template-hash: 86ccbd5d5f server: jvp-appserver	minikube	Running	0	-	-	6 minutes ago
●	jvp-webserver-deployment-75fdf75f89-qtg6d	jvp-webserver:latest	pod-template-hash: 75fdf75f89 server: jvp-webserver	minikube	Running	0	-	-	5 minutes ago

	Name	Images	Labels	Pods	Created ↑
●	jvp-appserver-deployment	jvp-appserver:latest	app.kubernetes.io/managed-by: Helm	3 / 3	5 minutes ago
●	jvp-webserver-deployment	jvp-webserver:latest	app.kubernetes.io/managed-by: Helm	1 / 1	5 minutes ago

Pods :

Replica Set :

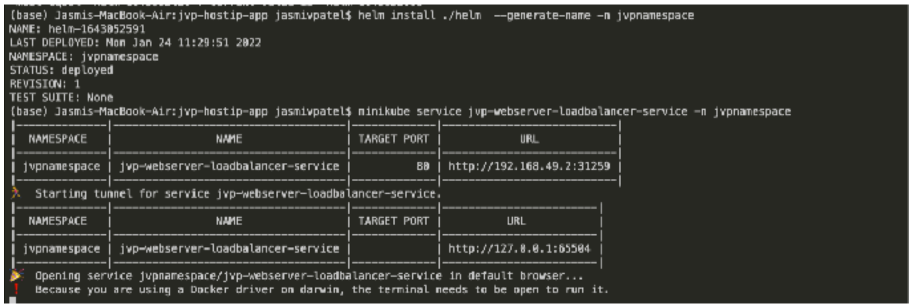


Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	jvp-web-server-deployment-5fdfdcc95f-xxl89	jvp-webserver:v4	pod-template-hash: 5fdfdcc95f server: jvp-webserver	minikube	Running	0	-	-	9 hours ago
●	jvp-app-server-deployment-84cb878c4d-t6g5g	jvp-appserver:v1	pod-template-hash: 84cb878c4d server: jvp-appserver	minikube	Running	0	-	-	14 hours ago
●	jvp-app-server-deployment-84cb878c4d-5gzpk	jvp-appserver:v1	pod-template-hash: 84cb878c4d server: jvp-appserver	minikube	Running	0	-	-	14 hours ago
●	jvp-app-server-deployment-84cb878c4d-9djgb	jvp-appserver:v1	pod-template-hash: 84cb878c4d server: jvp-appserver	minikube	Running	0	-	-	14 hours ago

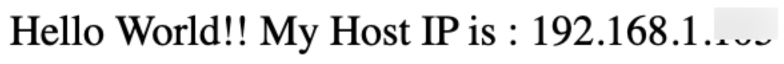
7. Run loadbalancer service :

minikube service jvp-webserver-loadbalancer-service -n jvpnamespace



```
(base) Jasmis-MacBook-Air:jvp-hostip-app jasmivpatel$ helm install ./helm  --generate-name -n jvpnamespace
NAME: helm-1643052591
LAST DEPLOYED: Mon Jan 24 11:29:51 2022
NAMESPACE: jvpnamespace
STATUS: deployed
REVISION: 1
TEST SUITE: None
(base) Jasmis-MacBook-Air:jvp-hostip-app jasmivpatel$ minikube service jvp-webserver-loadbalancer-service -n jvpnamespace
|--------------|------------------------------------|-------------|---------------------------|
|  NAMESPACE   |                NAME                | TARGET PORT |            URL            |
|--------------|------------------------------------|-------------|---------------------------|
| jvpnamespace | jvp-webserver-loadbalancer-service |          80 | http://192.168.49.2:31259 |
|--------------|------------------------------------|-------------|---------------------------|
🏃  Starting tunnel for service jvp-webserver-loadbalancer-service.
|--------------|------------------------------------|-------------|------------------------|
|  NAMESPACE   |                NAME                | TARGET PORT |          URL           |
|--------------|------------------------------------|-------------|------------------------|
| jvpnamespace | jvp-webserver-loadbalancer-service |             | http://127.0.0.1:65504 |
|--------------|------------------------------------|-------------|------------------------|
🎉  Opening service jvpnamespace/jvp-webserver-loadbalancer-service in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Result : a browser window will open to display result



Hello World!! My Host IP is : 192.168.1.[redacted]

8. Output :

One time I was able to direct nginx to Apiserver — but then couldn't redirect service to Apiserver next time due to network poxy port issue.

WITH Terraform :

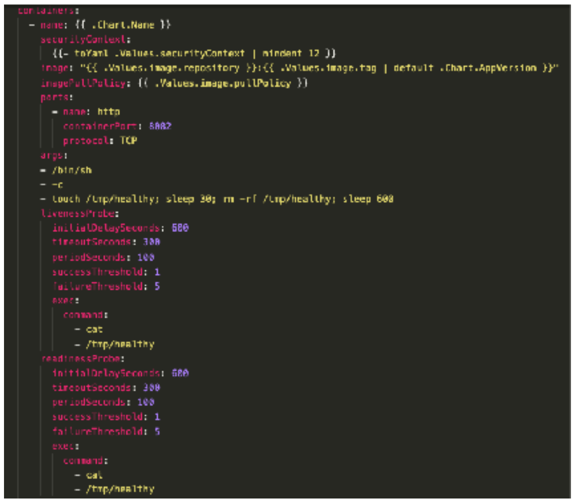
Folder - jvp-terraform

Navigate to folder jvp-terraform

1. Check parameters in inputs.tfvars files (personalize as required)
2. Run : terraform init (to initialize terraform providers)
3. Run : terraform plan -var-file=inputs.tfvars
4. Run : terraform apply -var-file=inputs.tfvars

Health-check and Monitoring :

1. Configured Liveness and Readiness probes to monitor health of services and configured rules in deployment file.



```
containers:
  - name: {{ .Chart.Name }}
    securityContext:
      {{- toYaml .Values.securityContext | nindent 12 }}
    image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
    imagePullPolicy: {{ .Values.image.pullPolicy }}
    ports:
      - name: http
        containerPort: 8082
        protocol: TCP
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      initialDelaySeconds: 600
      timeoutSeconds: 300
      periodSeconds: 100
      successThreshold: 1
      failureThreshold: 5
      exec:
        command:
          - cat
          - /tmp/healthy
    readinessProbe:
      initialDelaySeconds: 600
      timeoutSeconds: 300
      periodSeconds: 100
      successThreshold: 1
      failureThreshold: 5
      exec:
        command:
          - cat
          - /tmp/healthy
```

Discussions :

b.What you would do if you had more time

If there was more time — I would
- add custom code for liveness and readiness probe
- automate deployment of application on AWS/Google cloud containers
- experiment on more steps on securing application
- convert 2 tier application to 3 tier application for separating web-application-database layers
- debugging more on networking aspect of Kubernetes

c.How would you improve the deployment process

- Currently using Helm and Terraform ways of deployment. I will like to experiment with both Terraform and Helm together to discover additional advantages and further optimize the deployment.
- Experiment with rolling update - try to create “one touch deployment” process with Gitlab DevOps pipelines for automated deployment and easy rollback.
- Adding auditing capabilities as its required once application is larger and more complex as a next step.

d.What's broken about ops, and how would you fix it?

- Currently a networking aspect of how request and response flows between nginx server and application through cluster ip interface is not very clear. I will like to fix that part.
- Also try adding liveness and readiness probes to application for better monitoring.

Conclusion :

This is a very simple multi-tier application running at a small scale which creates prototype of High-availability application using Kubernetes, Helm and IaC (Terraform) - featuring the benefits of this design like High Availability, Portability, Performance, Scalability and Devops Automation.

Appendix

INSTALLATION STEPS :

Nodejs Application :

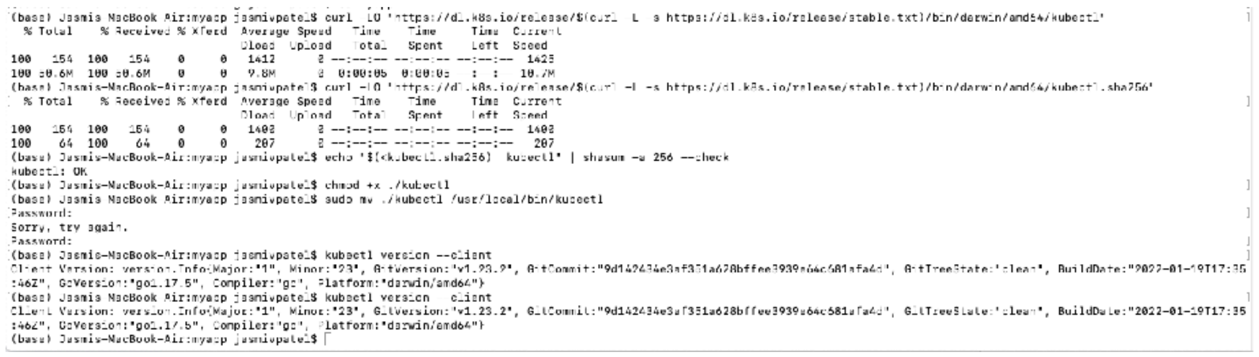
- Used Express package to handle request and response to nodejs application
- “use strict” — to indicate that code should be executed in strict mode and does not have any variables undefined

Installation step (without Terraform)

1. Installed Nodejs - Version v16.12.2
2. Docker - 4.4.2
3. Kubectl (mac-intel)

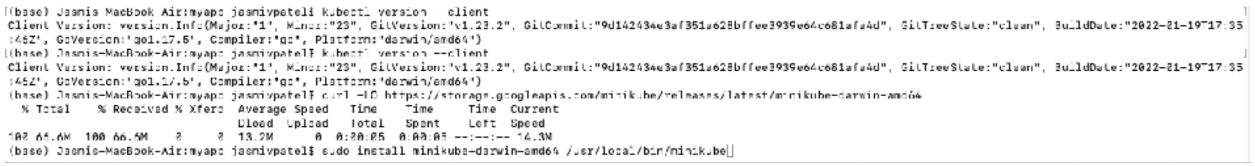
kubectl controls the Kubernetes cluster manager

Installation : https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/



```
(base) Jasmis MacBook Air:myapp jasmivpatel$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0   1412      0 --:--:-- --:--:-- --:--:--  1425
100 50.6M  100 50.6M    0     0   9.8M      0  0:00:05  0:00:05 --:--:-- 10.7M
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl.sha256"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0   1408      0 --:--:-- --:--:-- --:--:--  1408
100    64  100    64    0     0    207      0 --:--:-- --:--:-- --:--:--   207
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ echo "$(<kubectl.sha256)  kubectl" | shasum -a 256 --check
kubectl: OK
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ chmod +x ./kubectl
(base) Jasmis MacBook Air:myapp jasmivpatel$ sudo mv ./kubectl /usr/local/bin/kubectl
Password:
Sorry, try again.
Password:
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.2", GitCommit:"9d142434e3af351a628bffee3939e64c681afa4d", GitTreeState:"clean", BuildDate:"2022-01-19T17:35
:46Z", GoVersion:"go1.17.5", Compiler:"gc", Platform:"darwin/amd64"}
(base) Jasmis MacBook Air:myapp jasmivpatel$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.2", GitCommit:"9d142434e3af351a628bffee3939e64c681afa4d", GitTreeState:"clean", BuildDate:"2022-01-19T17:35
:46Z", GoVersion:"go1.17.5", Compiler:"gc", Platform:"darwin/amd64"}
(base) Jasmis-MacBook-Air:myapp jasmivpatel$
```

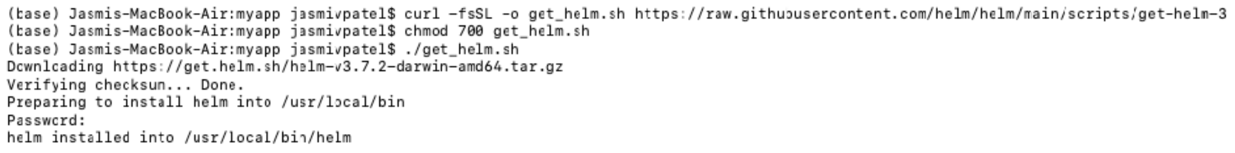
4, Install minikube



```
(base) Jasmis MacBook Air:myapp jasmivpatel$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.2", GitCommit:"9d142434e3af351a628bffee3939e64c681afa4d", GitTreeState:"clean", BuildDate:"2022-01-19T17:35
:46Z", GoVersion:"go1.17.5", Compiler:"gc", Platform:"darwin/amd64"}
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.2", GitCommit:"9d142434e3af351a628bffee3939e64c681afa4d", GitTreeState:"clean", BuildDate:"2022-01-19T17:35
:46Z", GoVersion:"go1.17.5", Compiler:"gc", Platform:"darwin/amd64"}
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 66.6M  100 66.5M    0     0  13.2M      0  0:00:05  0:00:05 --:--:-- 14.3M
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ sudo install minikube-darwin-amd64 /usr/local/bin/minikube
```

minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.

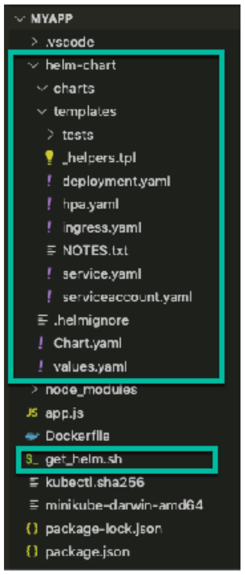
Here it is run on Docker



```
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ chmod 700 get_helm.sh
(base) Jasmis-MacBook-Air:myapp jasmivpatel$ ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.7.2-darwin-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
Password:
helm installed into /usr/local/bin/helm
```

5. Install Helm Chart
Helm v3.8.0-rc.2

https://helm.sh/docs/intro/install/



```
MYAPP
  > .vscode
  helm-chart
    charts
    templates
      > tests
      _helpers.tpl
      deployment.yaml
      hpa.yaml
      ingress.yaml
      NOTES.txt
      service.yaml
      serviceaccount.yaml
    .helmignore
    Chart.yaml
    values.yaml
  > node_modules
  app.js
  Dockerfile
  get_helm.sh
  kubectl.sha256
  minikube-darwin-amd64
  package-lock.json
  package.json
```

```
$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

Use Helm to create resource manifest which controls deployment - what container to run and how many instances.

Debugging : ssh into pods

```
kubectl exec -it jvp-webserver-deployment-68589cdcd9-bq5f6  -- sh
kubectl --namespace jvpnamespace port-forward $POD_NAME 8080:$CONTAINER_PORT
kubectl get deployments -n jvpnamespace
kubectl get pods -n jvpnamespace
```

Bug with tool : https://github.com/kubernetes/minikube/issues/549