CS580K - Cloud Computing Big Project

Live Video Streaming and Chat Web Application using Google Docker and Kubernetes

By
Bhanu Dobriyal
Tejesh Agrawal
Kasturi Vartak

Index

1. Abstract

- Created a Web Application with live Video and Chat features among users

- Build the Docker Image for the single container containing this app
- Uploaded this Image to Docker Hub
- Created cluster using Kubernetes cluster manager
- Deployed the containerized application over Kubernetes cluster
- Autoscaling the app using Kubernetes

2. Motivation

Live streaming video and chat feature

- Users can share a live video streaming and others can discuss about it over chat
- Similar to various live video streaming by Youtube , Facebook and Instagram
- Large user base and live streaming needs robust platform to orchestrate traffic
- Global video streaming market is 42.6 billion USD as of 2019

Kubernetes

- Use Kubernetes deployments to deploy pod workloads
- Allow applications to automatically respond to changes in their workloads and scale to meet demand

3. Web Interface

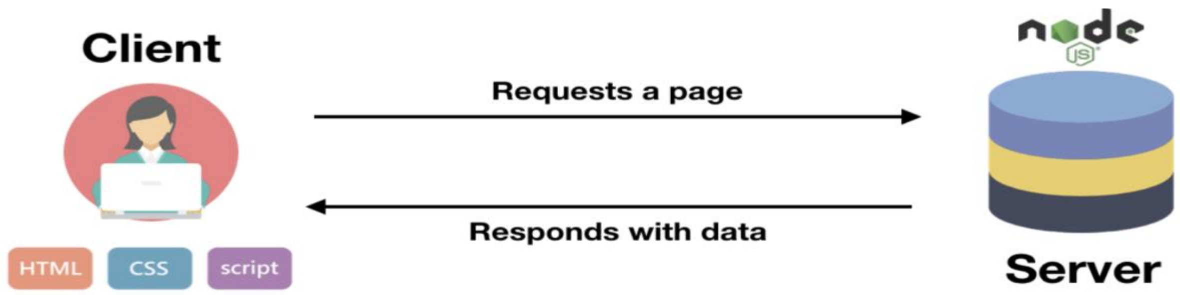
3.1 Video and Chat Interface

A key feature of our web application. We wanted to have live video streaming in our application , as this is a real time event generation media and would be a great example to study and understand the performance and behavior of our Cloud computing.

We studied various API documents like Facebook , Youtube and Google for their live video streaming but found challenges in their implementation to use their services for our application development.

Finally we were able to implement this feature using this capability by studying a WebRTC which adds real-time communication capabilities to an application. For the chat feature we used express.js framework to make use of the server, in responses from the client side which we developed using ejs engine and various javascript files to use socket.io library for making connections to server and render it over client side.

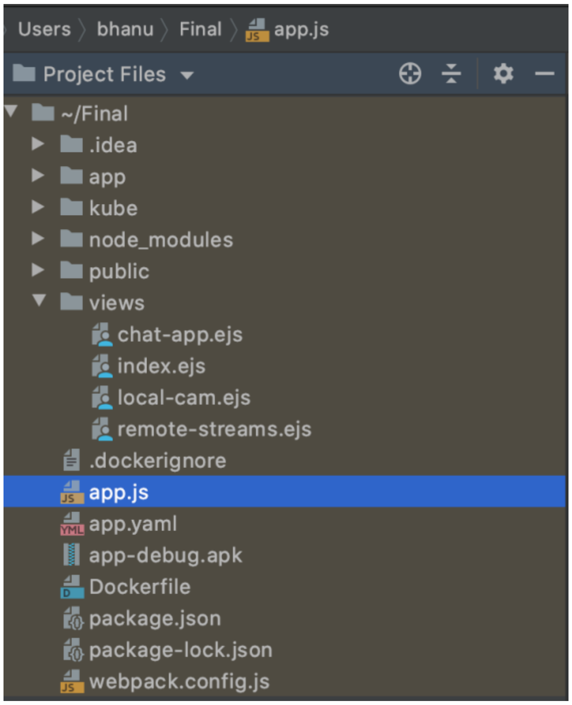
Below is the web architecture of the application.



Below is the technical stack used for this app.

- **Ejs -** template engine to simplify production of HTML
- **Websockets -** Protocol that allows bilateral synchronous exchange between client and server. Socket.io is the library used.
- **Express.js -** Framework based on Node.js which is JS back-end tech executed by server.
- **CSS -** For user interface template design and stylesheets.

Below is the file structure of the code of our application.



```
Users 〉 bhanu 〉 Final 〉 app.js
Project Files ▾
▼ ~/Final
    ▶ .idea
    ▶ app
    ▶ kube
    ▶ node_modules
    ▶ public
    ▼ views
        chat-app.ejs
        index.ejs
        local-cam.ejs
        remote-streams.ejs
    .dockerignore
    app.js
    app.yaml
    app-debug.apk
    Dockerfile
    package.json
    package-lock.json
    webpack.config.js
```

3.2 Integration

As these features are completely different from a user perspective , it required efforts to integrate it over one system in terms of real time connections across multiple users synchronously on the same server.

4. Docker

4.1 Packaging the app as a container

- In this we need to package and run the application using Linux and Docker containers by using mechanism to encapsulate apps with all of their dependencies into a single archive
- To run the app, we only need the archive.
- With Linux containers, you can start the application even if you don't have Node.js installed.
- First we need to install the Docker Community Edition (CE).
- You can do this by referring to the below link.
  https://docs.docker.com/get-docker/

```
$ git clone https://github.com/waft-media-corp/waaft.git
```

- Docker containers are built from Dockerfile

```
$vi Dockerfile
```

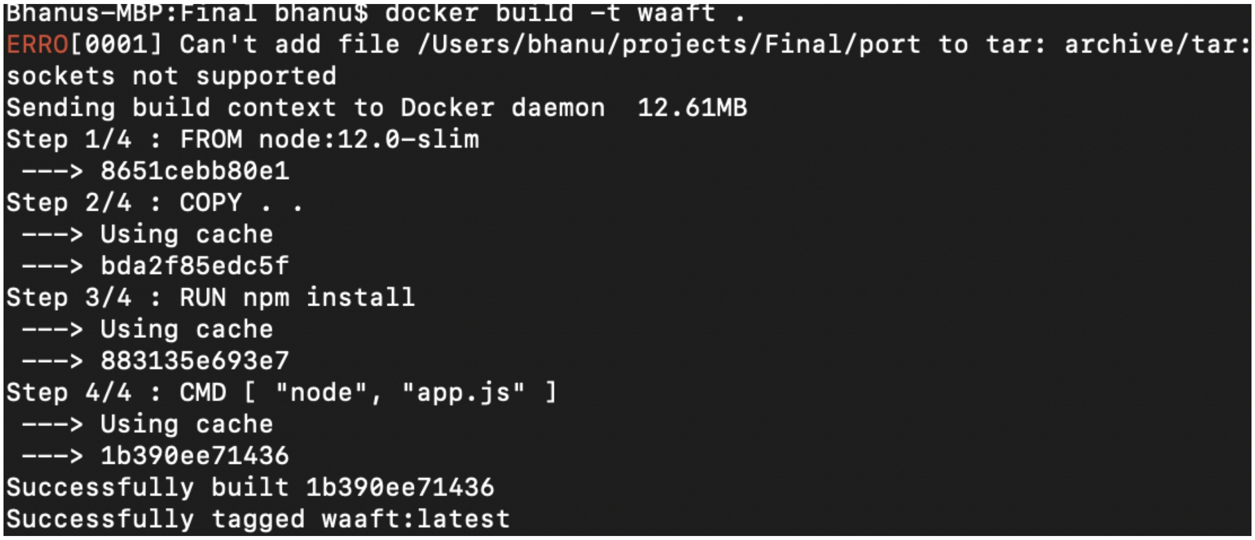
```
FROM node:12.0-slim
COPY . .
RUN npm install
CMD [ "node", "app.js" ]
```

Where,

- FROM defines the base layer for the container, in this case, a version of Ubuntu with Node.js installed
- COPY copies the files of your app into the container
- RUN executes npm install inside the container
- CMD defines the command that should be executed when the container starts

Then we have run the below command in terminal

```
$ docker build -t waaft .
```



```
Bhanus-MBP:Final bhanu$ docker build -t waaft .
ERRO[0001] Can't add file /Users/bhanu/projects/Final/port to tar: archive/tar:
sockets not supported
Sending build context to Docker daemon  12.61MB
Step 1/4 : FROM node:12.0-slim
 ---> 8651cebb80e1
Step 2/4 : COPY . .
 ---> Using cache
 ---> bda2f85edc5f
Step 3/4 : RUN npm install
 ---> Using cache
 ---> 883135e693e7
Step 4/4 : CMD [ "node", "app.js" ]
 ---> Using cache
 ---> 1b390ee71436
Successfully built 1b390ee71436
Successfully tagged waaft:latest
```

4.2 Building the container

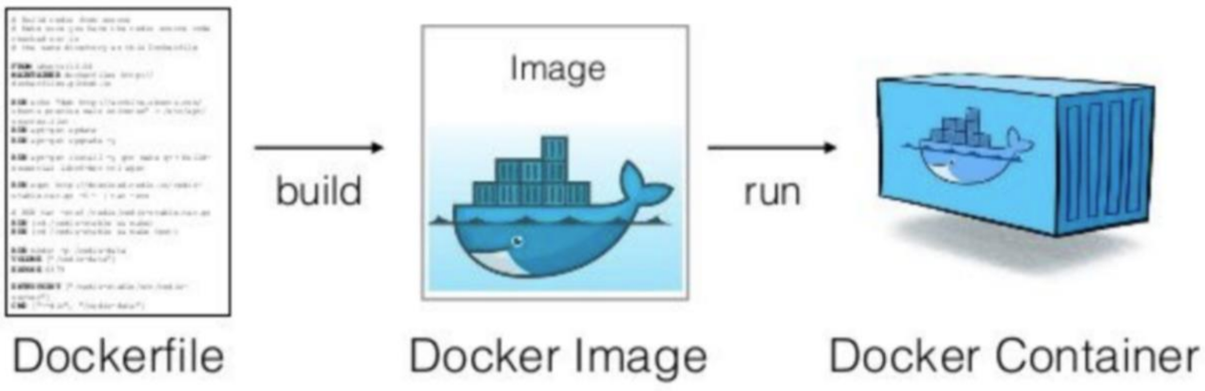
- Build a container image from our app with command:

  `$ docker build -t waaft .`
  `-t waaft` → defines the name ("tag") of your container

  `.` → is the location of the Dockerfile and application code — in this case, it's the current directory

  The command executes the steps outlined in the Dockerfile, one by one
  Output is a Docker image.

- A Docker image is an archive containing all the files that are packaged in a container.



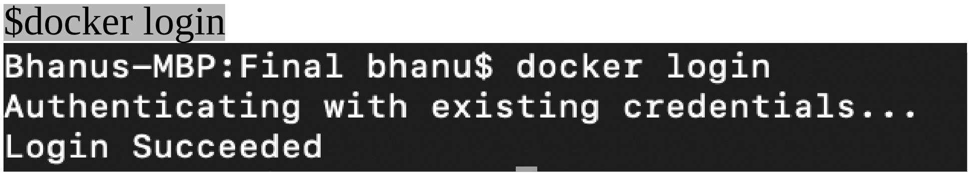
List docker images:
`$ docker images`

4.3 Uploading container image to Docker Hub

To ensure this container image can be used by anyone , we can push this image over Docker Hub.

- Access Docker Hub

`$docker login`

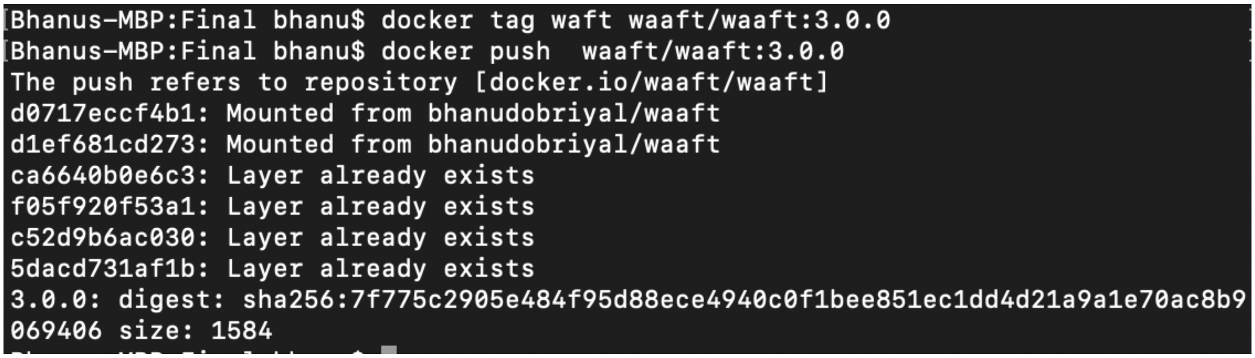


```
Bhanus-MBP:Final bhanu$ docker login
Authenticating with existing credentials...
Login Succeeded
```

//authenticate and login to docker hub

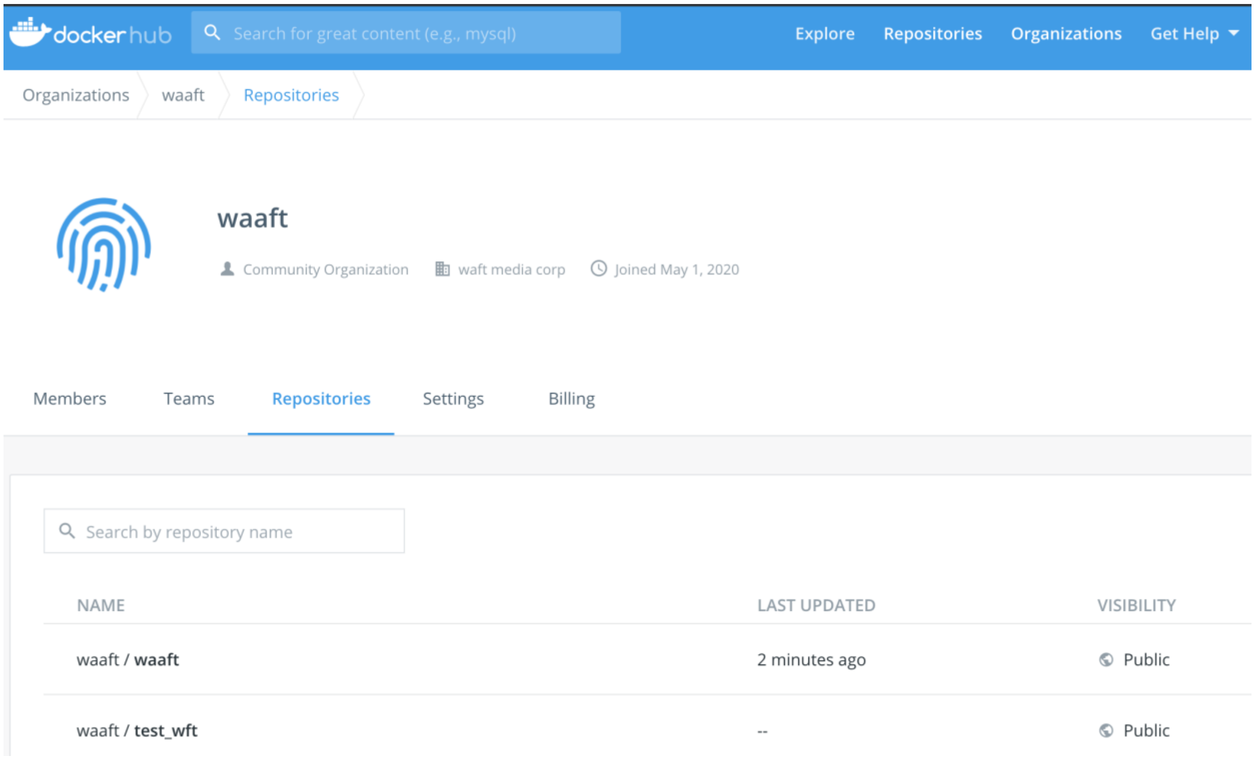
- Images uploaded to Docker Hub must have a name of the form username/image:tag:
  To rename your image according to this format, run the following command:
  `$ docker tag waft waaft/waaft:3.0.0`
  Waaft → organization name
  Waaft → image name with version 3.0.0

- Upload waft image to Docker Hub
  $ docker push waaft/waaft:3.0.0



```
Bhanus-MBP:Final bhanu$ docker tag waft waaft/waaft:3.0.0
Bhanus-MBP:Final bhanu$ docker push  waaft/waaft:3.0.0
The push refers to repository [docker.io/waaft/waaft]
d0717eccf4b1: Mounted from bhanudobriyal/waaft
d1ef681cd273: Mounted from bhanudobriyal/waaft
ca6640b0e6c3: Layer already exists
f05f920f53a1: Layer already exists
c52d9b6ac030: Layer already exists
5dacd731af1b: Layer already exists
3.0.0: digest: sha256:7f775c2905e484f95d88ece4940c0f1bee851ec1dd4d21a9a1e70ac8b9
069406 size: 1584
```

- Now, the image is now publicly available as waaft/waaft:3.0.0 on Docker Hub.



NAME	LAST UPDATED	VISIBILITY
waaft / **waaft**	2 minutes ago	Public
waaft / **test_wft**	--	Public

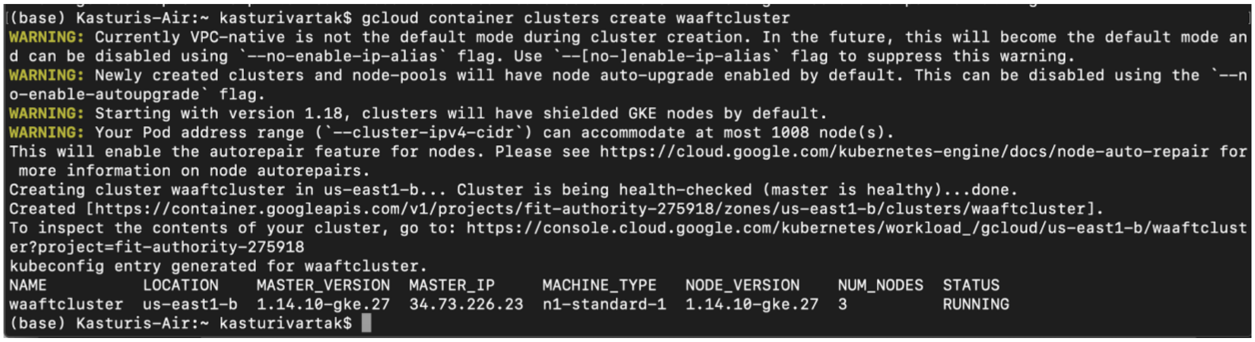
5. Deploying app to Kubernetes cluster

$gcloud init

- To launch an interactive Getting Started workflow for the gcloud command-line tool.

$gcloud container clusters create waaftcluster

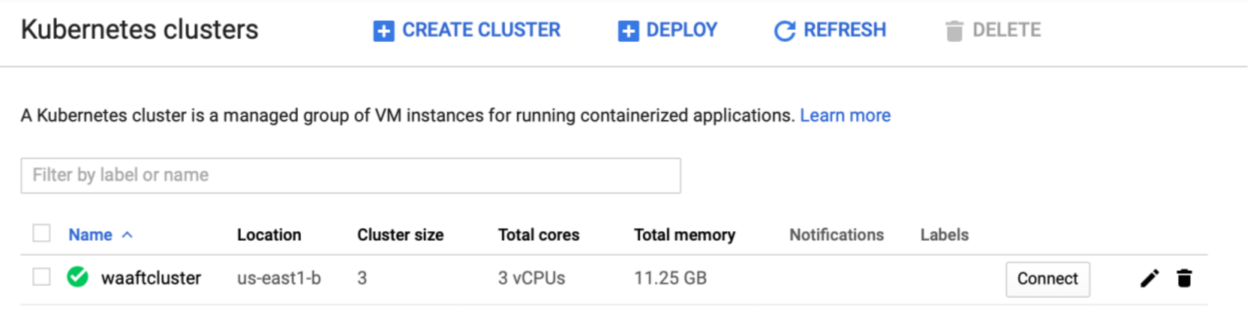
waaftcluster→ name of cluster



```
(base) Kasturis-Air:~ kasturivartak$ gcloud container clusters create waaftcluster
WARNING: Currently VPC-native is not the default mode during cluster creation. In the future, this will become the default mode an
d can be disabled using `--no-enable-ip-alias` flag. Use `--[no-]enable-ip-alias` flag to suppress this warning.
WARNING: Newly created clusters and node-pools will have node auto-upgrade enabled by default. This can be disabled using the `--n
o-enable-autoupgrade` flag.
WARNING: Starting with version 1.18, clusters will have shielded GKE nodes by default.
WARNING: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
This will enable the autorepair feature for nodes. Please see https://cloud.google.com/kubernetes-engine/docs/node-auto-repair for
 more information on node autorepairs.
Creating cluster waaftcluster in us-east1-b... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/fit-authority-275918/zones/us-east1-b/clusters/waaftcluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-east1-b/waaftclust
er?project=fit-authority-275918
kubeconfig entry generated for waaftcluster.
NAME          LOCATION    MASTER_VERSION  MASTER_IP      MACHINE_TYPE   NODE_VERSION    NUM_NODES  STATUS
waaftcluster  us-east1-b  1.14.10-gke.27  34.73.226.23   n1-standard-1  1.14.10-gke.27  3          RUNNING
(base) Kasturis-Air:~ kasturivartak$
```

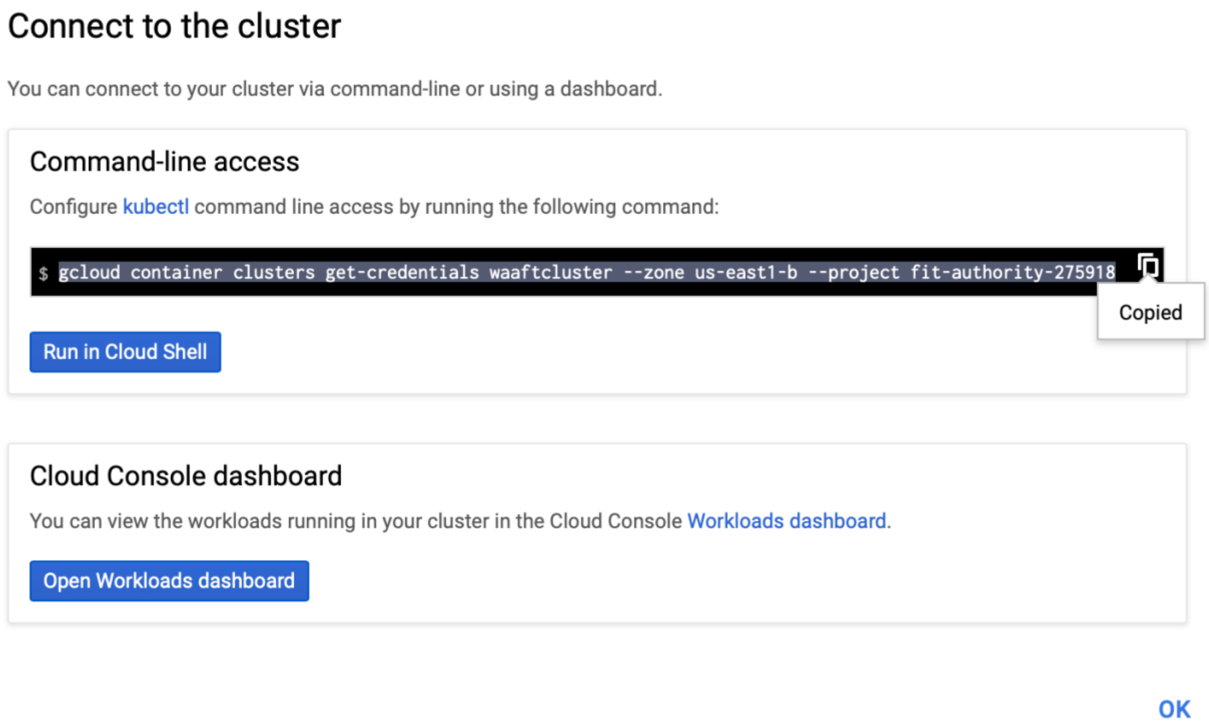
- To connect to a created cluster via command line:

Go to Kubernetes Engine→ Clusters → Connect



A Kubernetes cluster is a managed group of VM instances for running containerized applications. Learn more

□	Name ^	Location	Cluster size	Total cores	Total memory	Notifications	Labels		
□	✓ waaftcluster	us-east1-b	3	3 vCPUs	11.25 GB			Connect	



Connect to the cluster

You can connect to your cluster via command-line or using a dashboard.

Command-line access

Configure kubectl command line access by running the following command:

```
$ gcloud container clusters get-credentials waaftcluster --zone us-east1-b --project fit-authority-275918
```

Cloud Console dashboard

You can view the workloads running in your cluster in the Cloud Console Workloads dashboard.

```
(base) Kasturis-Air:~ kasturivartak$ gcloud container clusters get-credentials waaftcluster --zone us-east1-b --project fit-authority-275918
Fetching cluster endpoint and auth data.
kubeconfig entry generated for waaftcluster.
```

```
vim deployment.yaml
```

```
(base) Kasturis-Air:~ kasturivartak$ kubectl apply -f deployment.yaml
deployment.apps/waaft created
```

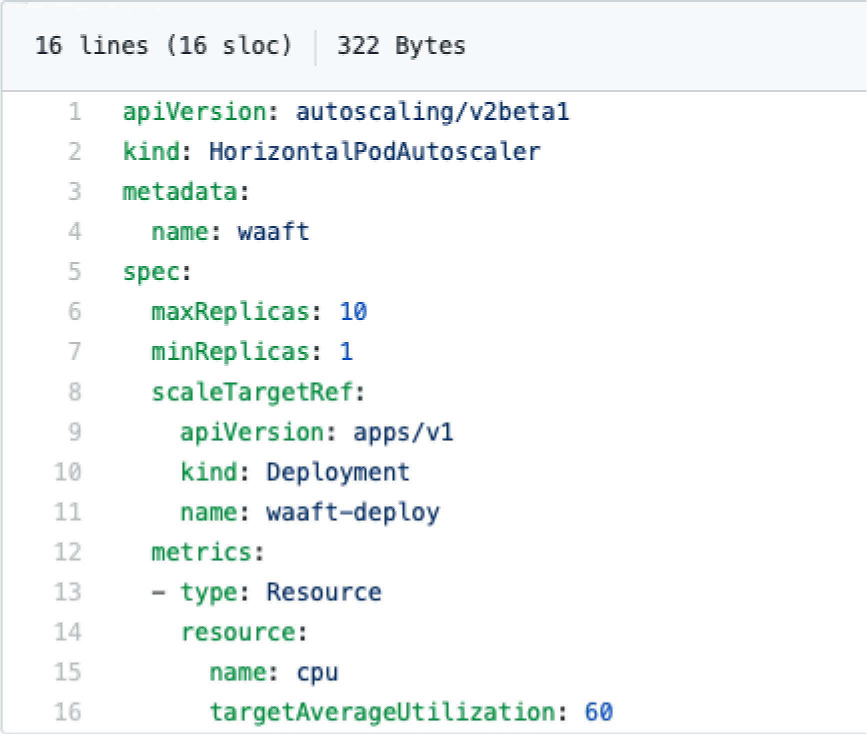
```
(base) Kasturis-Air:~ kasturivartak$ kubectl apply -f service.yaml
service/waaft created
```

6. Autoscaling using Kubernetes

We are using horizontal pod scaling to autoscale our application, suppose if the CPU utilization goes beyond 60 %, the pod replicas will increase by 1, we initialized a range for our replicas, min =1 and max =10.

When CPU utilization falls below 60%, then the autoscaler will adjust the replica count of the targeted deployment down;
When it goes above 60%, replicas will be added

Script for using the above conditions:



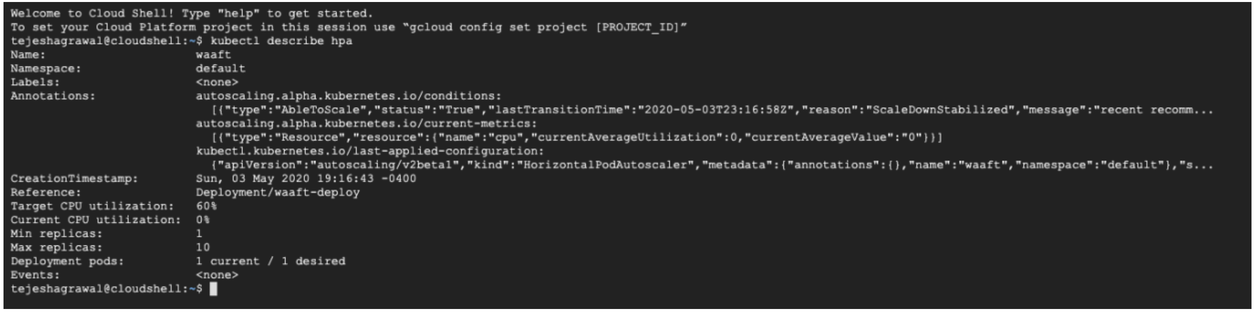
16 lines (16 sloc) | 322 Bytes

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: waaft
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: waaft-deploy
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 60
```

```
(base) Kasturis-Air:~ kasturivartak$ kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/kubecluster created
```

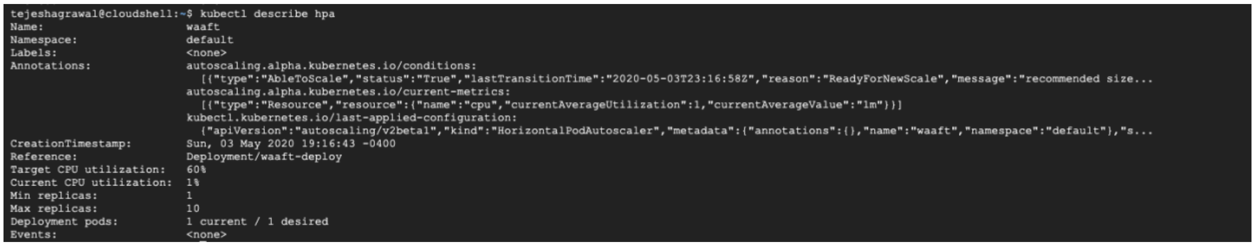
```
(base) Kasturis-Air:~ kasturivartak$ kubectl describe hpa
```

After describing hpa, we can see before running application CPU utilization is 1%



```
Welcome to Cloud Shell! Type "help" to get started.
To set your Cloud Platform project in this session use "gcloud config set project [PROJECT_ID]"
tejeshagrawal@cloudshell:~$ kubectl describe hpa
Name:                     waaft
Namespace:                default
Labels:                   <none>
Annotations:              autoscaling.alpha.kubernetes.io/conditions:
                            [{"type":"AbleToScale","status":"True","lastTransitionTime":"2020-05-03T23:16:58Z","reason":"ScaleDownStabilized","message":"recent recomm...
                          autoscaling.alpha.kubernetes.io/current-metrics:
                            [{"type":"Resource","resource":{"name":"cpu","currentAverageUtilization":0,"currentAverageValue":"0"}}]
                          kubectl.kubernetes.io/last-applied-configuration:
                            {"apiVersion":"autoscaling/v2beta1","kind":"HorizontalPodAutoscaler","metadata":{"annotations":{},"name":"waaft","namespace":"default"},"s...
CreationTimestamp:        Sun, 03 May 2020 19:16:43 -0400
Reference:                Deployment/waaft-deploy
Target CPU utilization:   60%
Current CPU utilization:  0%
Min replicas:             1
Max replicas:             10
Deployment pods:          1 current / 1 desired
Events:                   <none>
tejeshagrawal@cloudshell:~$
```

Once, we start the application, the CPU utilization gets increased by 1%. And now it is 1% from 0%.



```
tejeshagrawal@cloudshell:~$ kubectl describe hpa
Name:                     waaft
Namespace:                default
Labels:                   <none>
Annotations:              autoscaling.alpha.kubernetes.io/conditions:
                            [{"type":"AbleToScale","status":"True","lastTransitionTime":"2020-05-03T23:16:58Z","reason":"ReadyForNewScale","message":"recommended size...
                          autoscaling.alpha.kubernetes.io/current-metrics:
                            [{"type":"Resource","resource":{"name":"cpu","currentAverageUtilization":1,"currentAverageValue":"1m"}}]
                          kubectl.kubernetes.io/last-applied-configuration:
                            {"apiVersion":"autoscaling/v2beta1","kind":"HorizontalPodAutoscaler","metadata":{"annotations":{},"name":"waaft","namespace":"default"},"s...
CreationTimestamp:        Sun, 03 May 2020 19:16:43 -0400
Reference:                Deployment/waaft-deploy
Target CPU utilization:   60%
Current CPU utilization:  1%
Min replicas:             1
Max replicas:             10
Deployment pods:          1 current / 1 desired
Events:                   <none>
```

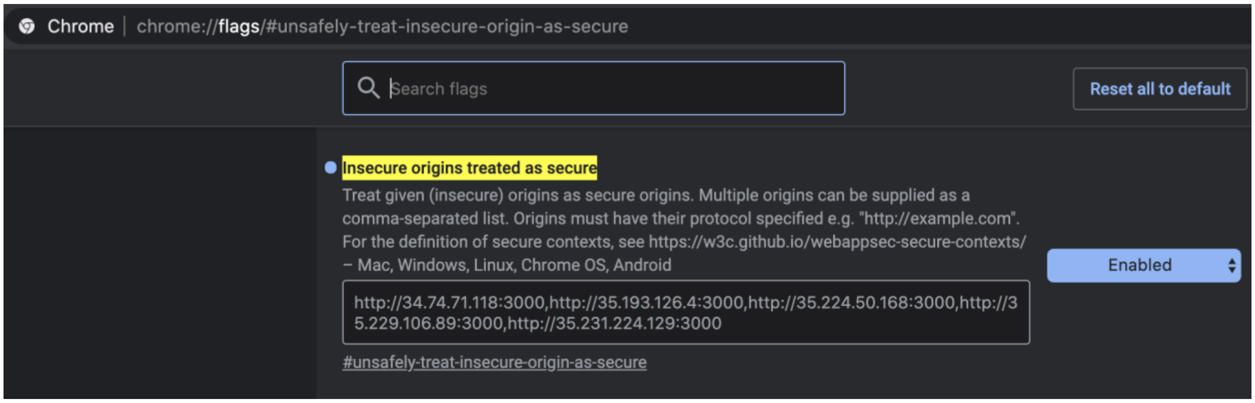
7. How to use this application

1. Open the static url given in the Links section on Chrome.
2. In a separate tab in chrome run the following in the URL section.

chrome://flags/#unsafely-treat-insecure-origin-as-secure

3. This would provide for the necessary accesses so that the Camera of the system between the peers can work. Click on enable thereafter.

   Then click on Relaunch.



Insecure origins treated as secure

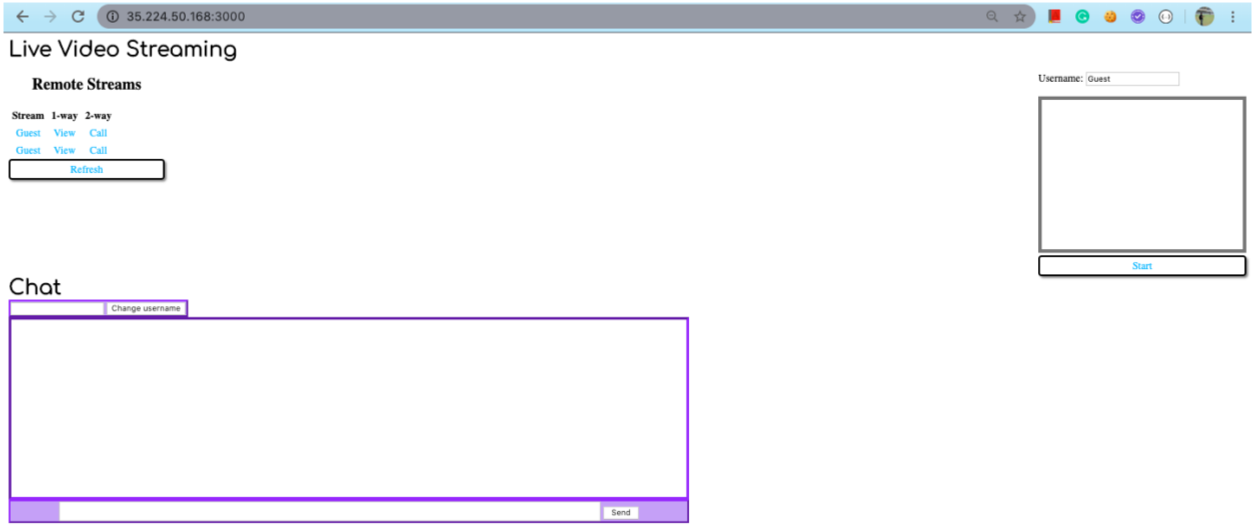
Treat given (insecure) origins as secure origins. Multiple origins can be supplied as a comma-separated list. Origins must have their protocol specified e.g. "http://example.com". For the definition of secure contexts, see https://w3c.github.io/webappsec-secure-contexts/ – Mac, Windows, Linux, Chrome OS, Android

http://34.74.71.118:3000,http://35.193.126.4:3000,http://35.224.50.168:3000,http://35.229.106.89:3000,http://35.231.224.129:3000

Enabled

#unsafely-treat-insecure-origin-as-secure

4. After putting url in the browser:



Live Video Streaming

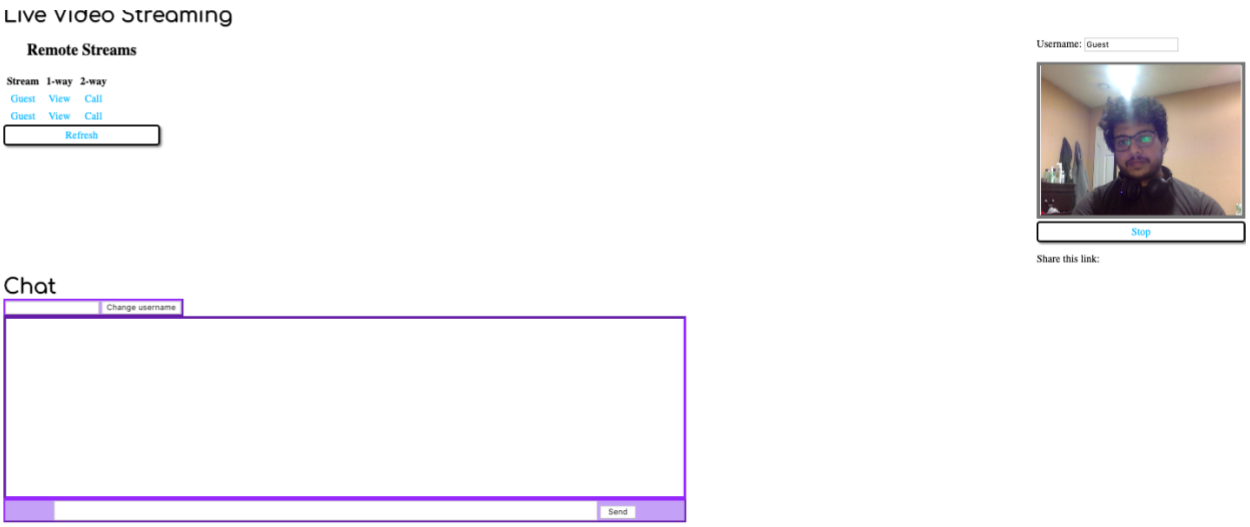
Remote Streams

Stream	1-way	2-way
Guest	View	Call
Guest	View	Call

Username: Guest

Chat

5. After we click on Start, and clicking refresh will give a sharing link.



Live Video Streaming

Remote Streams

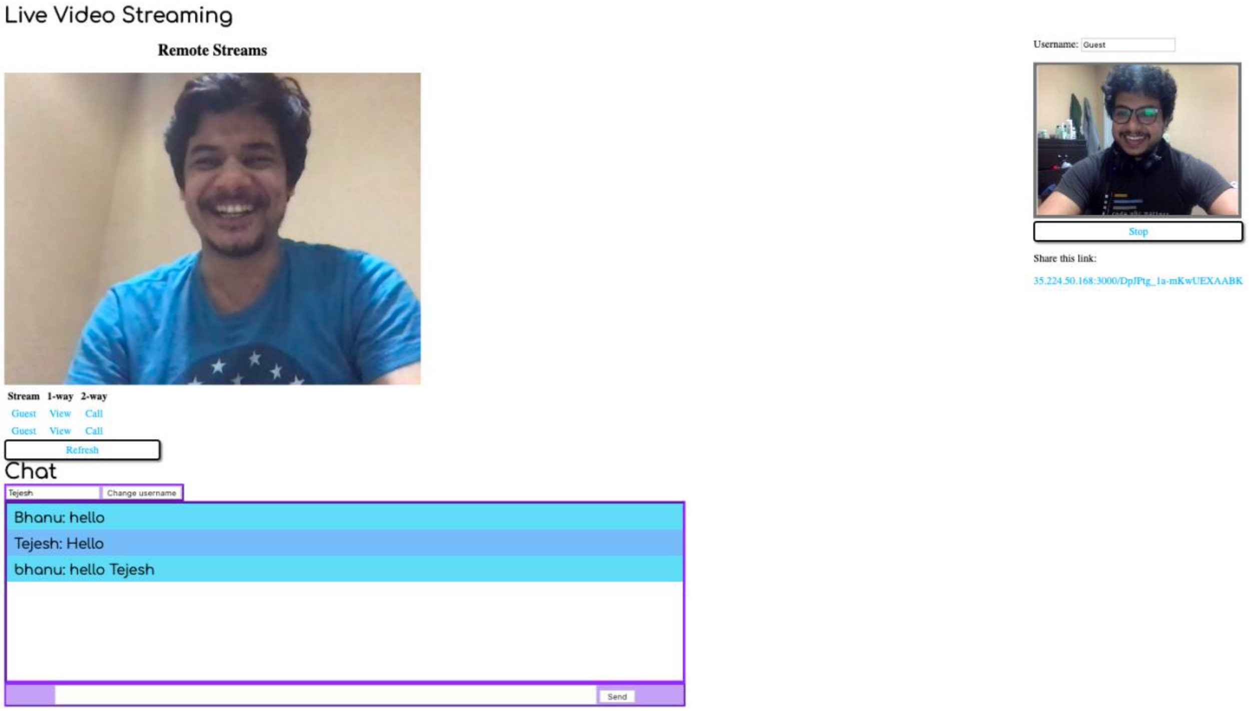
Stream	1-way	2-way
Guest	View	Call
Guest	View	Call

Username: Guest

Share this link:

Chat

6. Shared link will be joined by the receiver end. Here we can see that we even chat with the peer synchronously as we video stream.



Live Video Streaming

Remote Streams

Stream	1-way	2-way
Guest	View	Call
Guest	View	Call

Username: Guest

Share this link:

35.224.50.168:3000/DpJPtg_1a-mKwUEXAABK

Chat

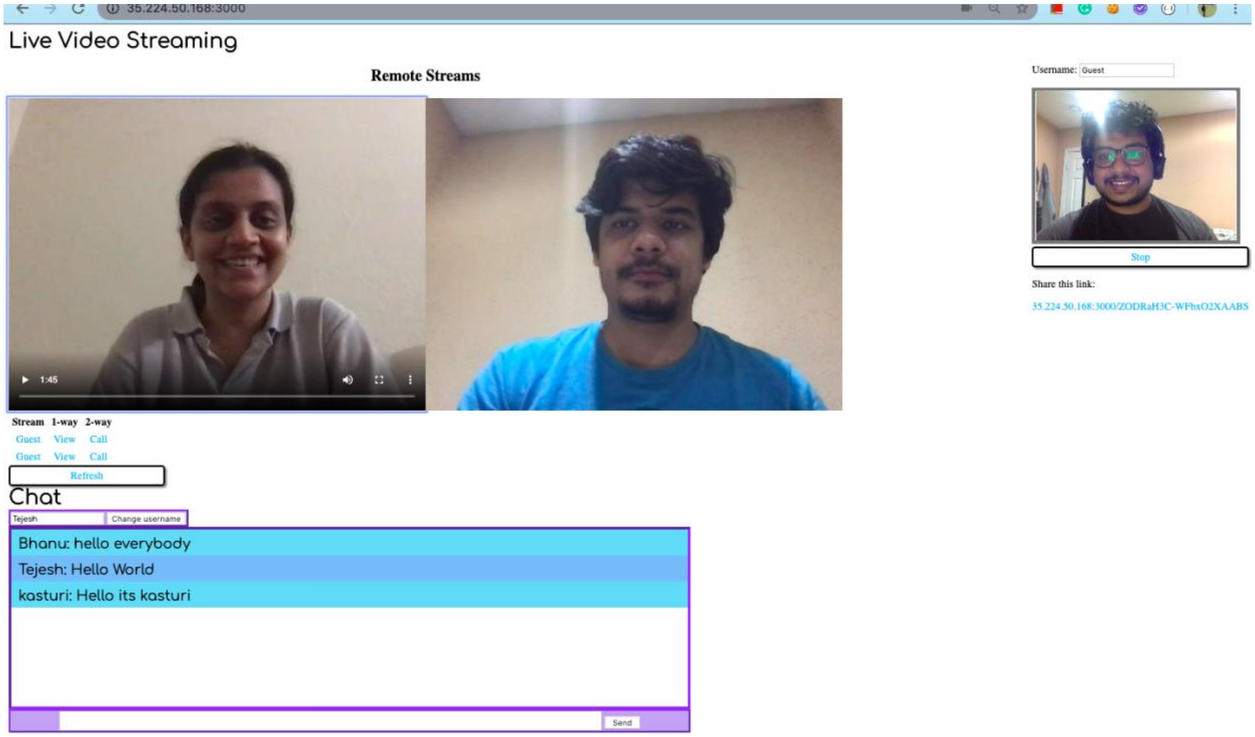
Tejesh

Bhanu: hello

Tejesh: Hello

bhanu: hello Tejesh

7. We can use this with multiple users as well. Below is the screenshot of our team members using the application for demo.



Live Video Streaming

Remote Streams

Username: Guest

Share this link:

35.224.50.168:3000/ZODRaH3C-WFbxO2XAABS

Stream	1-way	2-way
Guest	View	Call
Guest	View	Call

Chat

Tejesh Change username

Bhanu: hello everybody

Tejesh: Hello World

kasturi: Hello its kasturi

8. Challenges

- Instead of Youtube API live streaming, we decided to go and study ffmpeg for camera access, libx24 to encode the live camera video to a format (.avi) video to display on browser, because the video will be hosted on youtube platform, but later we decided to move to WebRTC.

- Instead of nginx, we are currently hosting applications on Kubernetes.Before this we hosted the application directly on GCP app engine, but it consumes GCP credits very fast. So we moved it to Kubernetes cluster and autoscaled it based on CPU utilization.
- Challenges while integrating the Chat and Video features of the application.
- We tried to integrate this application with a login page which uses MongoDb but couldn't implement it as it provided challenges for the runnability of this application ,as it used the same port number to hit the server.

9. Future Scope

- Https, to access the camera on browser, we have to create SSL certificates, for now a user has to manually enable connection for a given url to enable camera through browser.
- Login and Signup pages for authenticating users accessing the feeds. MongoDB connectivity to store this information at backend.
- We will introduce Nginx as well, so that different images such as mongoDB and application docker image can be combined and hosted on a single url.

10. Links

Static Link:

http://35.224.50.168:3000/

11. References

WebRTC:

https://webrtc.org

https://medium.com/@noufel.gouirhate/build-a-simple-chat-app-with-node-js-and-socket-io-ea716c093088

Kubernetes:

https://learnk8s.io/nodejs-kubernetes-guide

Autoscaling: https://hub.packtpub.com/implementing-horizontal-pod-autoscaling-in-kubernetes-tutorial/