


Dockerized Video Streaming System with Google Kubernetes Engine

Jaguar Perlas
Roy Ortega
Felix Ruttan

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

The Steps to Move to GKE

1. Connect Google Kubernetes Engine (GKE) with our local Kubernetes
2. Upload images to Google Cloud
3. Write deploy yml files
4. Solve tons of issues
5. Scale Automatically

Step 1. Connect GKE with our local Kubernetes

By following a guide from Google Cloud documentation, we were able to use our local CLI to control GKE.

1. Install the Google Cloud CLI

<https://cloud.google.com/sdk/docs/install>

2. Initialize gcloud CLI

https://cloud.google.com/sdk/docs/install-sdk#initializing_the_sdk

Initializing gcloud CLI included logging into our Google account, creating a project, and selecting a region.

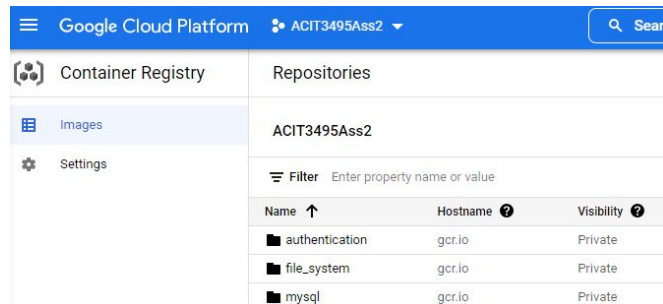
This process was made easy by Google.

Step 2. Upload Images to Google Cloud

Uploading images to Google Cloud was also done easily.

1. Build the docker image
 - a. `docker build -t mysql:latest .`
2. Tag the docker image
 - a. `docker tag mysql:latest gcr.io/<projectName>/mysql`
3. Push the image to Google
 - a. `docker push gcr.io/<projectName>/mysql`

Navigating to Google Cloud Platform Container Registry should have the pushed images



The screenshot shows the Google Cloud Platform Container Registry interface. The top navigation bar is blue with the text "Google Cloud Platform" and a dropdown menu showing "ACIT3495Ass2". A search bar is on the right. The left sidebar has three items: "Container Registry" (selected), "Images", and "Settings". The main content area is titled "Repositories" and shows a list of repositories for the project "ACIT3495Ass2". There is a filter input field with the placeholder text "Filter Enter property name or value". Below this is a table with three columns: "Name", "Hostname", and "Visibility". The table contains three rows of data: "authentication", "file_system", and "mysql", all with "gcr.io" as the hostname and "Private" as the visibility.

Name ↑	Hostname ?	Visibility ?
authentication	gcr.io	Private
file_system	gcr.io	Private
mysql	gcr.io	Private

Step 3. Write deploy yaml files

Best to keep deployment files separated to further make our system decoupled and independent

We've created 5 deployment files:

```
! deploy_authentication.yml
! deploy_file_system.yml
! deploy_mysql.yml
! deploy_upload_video.yml
! deploy_video_streaming.yml
```

For each deployment, a service is needed to enable communication. We created 5 services

```
! svc_authentication.yml
! svc_file_system.yml
! svc_mysql.yml
! svc_upload_video.yml
! svc_video_streaming.yml
```

Step 4. Solve issues

We were able to deploy our video streaming system, but there were a few major issues we needed to solve first:

Issues and Difficulties

- With scaling in mind, we can not have internal hard coded IP addresses
 - Using hard coded IPs means that replicas are not included in the data flow
- Replicas of user facing pods have no way of telling if the user already logged in from another replica
 - If a user logs in from Replica1 and is load balanced to Replica2, the user would need to log in again.
- Needed a way for replicated video storage and MySQL pods to share one volume
 - Otherwise, replicated video storage pods will have an empty storage
- We needed a way to automatically scale up/down our pods
 - With the above issues solved, we were able to manually scale our pods while maintaining full functionality. Now we have to automatically scale our system.
- Long process of pushing a change
 - It was time consuming to individually build, tag, push, and deploy every change made to our application

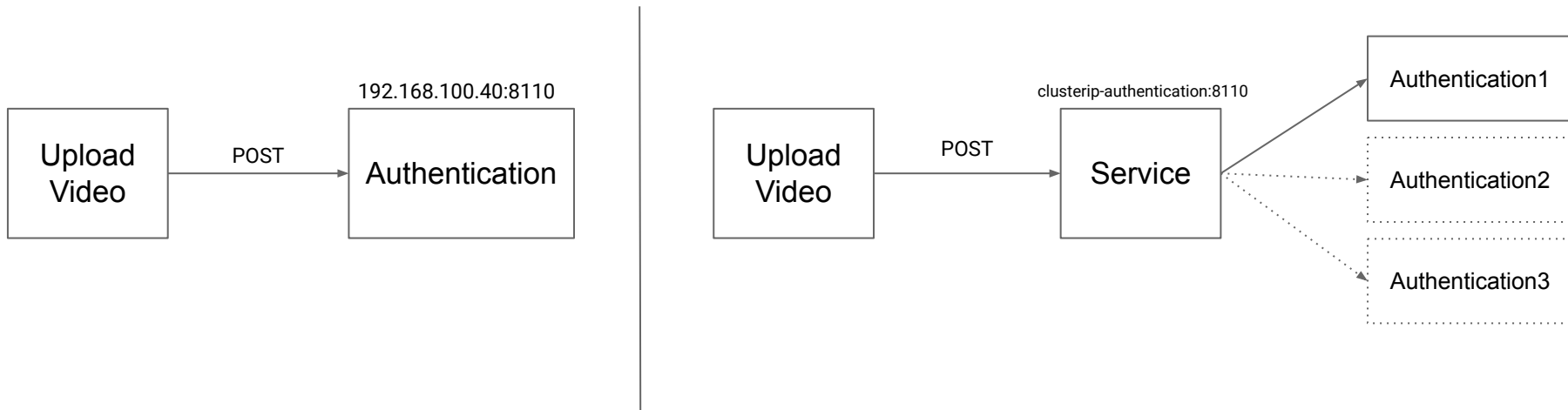
Addressing Issues (Hard Coded IPs)

With scaling in mind, we can not have internal hard coded IP addresses

Before: `axios.post('http://192.168.100.40:8110/')`

After: `axios.post('http://clusterip-authentication:8110/')`

To address this issue, we used the service name instead of the pod's IP address



Addressing Issues (Login)

Replicas of user facing pods have no way of telling if the user already logged in from another replica

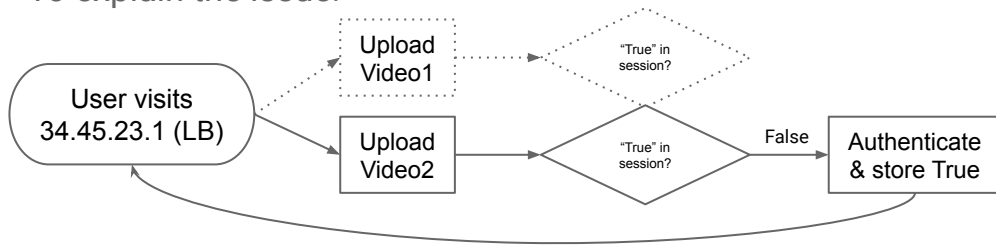
Before:

```
app.get('/', (req, res) => {  
  let sessionData = req.session  
  if (sessionData.authenticated !== true) {  
    res.redirect('/login')  
  } else {  
    res.sendFile(PATH + "index.html")  
  }  
});
```

After:

```
app.get('/', (req, res) => {  
  let cookieData = req.cookies  
  if (cookieData.authenticated !== 'true') {  
    res.redirect('/login')  
  } else {  
    res.sendFile(PATH + "index.html")  
  }  
});
```

To explain the issue:



- User needs to be load balanced to the same replica
 - Otherwise, the existing session would end, resulting in sessionData being cleared

Addressing Issues (Login Cont.)

Replicas of user facing pods have no way of telling if the user already logged in from another replica

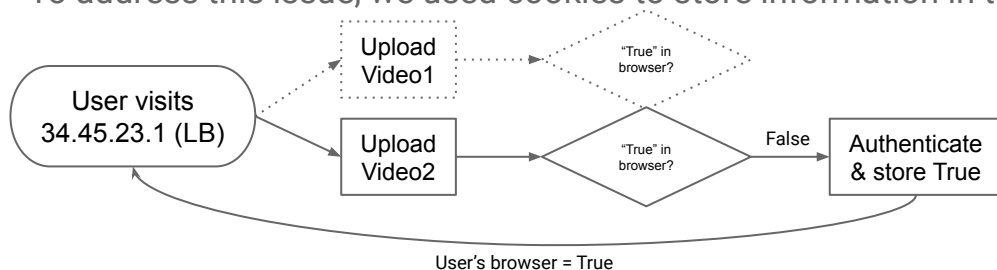
Before:

```
app.get('/', (req, res) => {  
  let sessionData = req.session  
  if (sessionData.authenticated !== true) {  
    res.redirect('/login')  
  } else {  
    res.sendFile(PATH + "index.html")  
  }  
});
```

After:

```
app.get('/', (req, res) => {  
  let cookieData = req.cookies  
  if (cookieData.authenticated !== 'true') {  
    res.redirect('/login')  
  } else {  
    res.sendFile(PATH + "index.html")  
  }  
});
```

To address this issue, we used cookies to store information in the user's browser



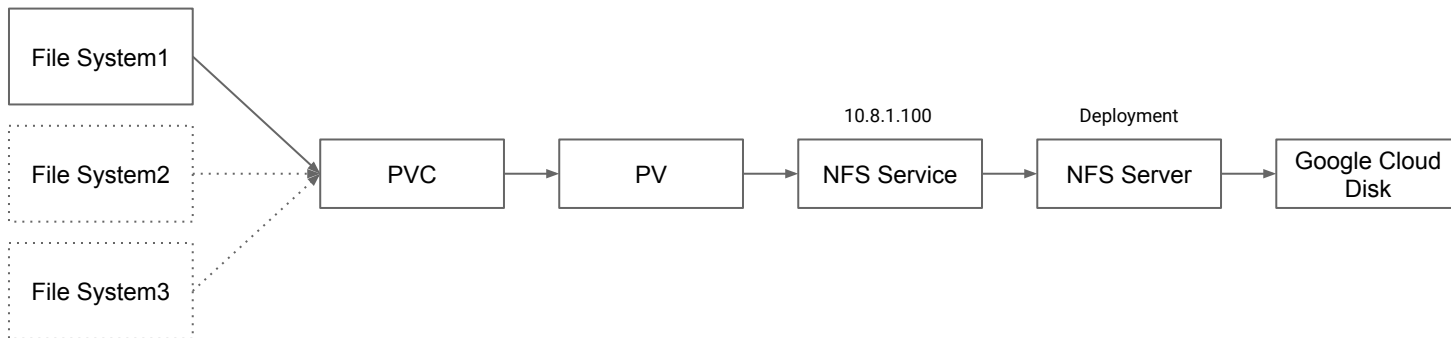
- Cookies will persist in the user's browser
 - Allows user to stay logged in throughout multiple replicas

Addressing Issues (Shared Volumes)

Needed a way for replicated video storage and MySQL pods to share one volume

- Setting up volumes in GKE by following Week 8 was unsuccessful
 - Default storage class is “standard”, which does not support ReadWriteMany
- HostPath volume plugin also does not support ReadWriteMany
- From research, the best way is to use a Network File System (NFS) to support ReadWriteMany

Topology of solution:



Addressing Issues (Automate start/clean up)

Long process of pushing a change

- With a lot of yaml files, it was time consuming to start from a clean slate

To address this issue, we've created a Shell script to automatically start our system from a clean slate

```
kubectl delete --all deployments
kubectl delete --all pv
kubectl delete --all pvc
kubectl delete --all hpa
kubectl delete -f hpa_components.yaml
kubectl delete service clusterip-authentication
kubectl delete service clusterip-file-system
kubectl delete service clusterip-mysql
kubectl delete service lb-upload-video
kubectl delete service lb-video-streaming
```

```
docker build -t jperlas/upload_video:latest ./upload_video
docker build -t jperlas/mysql:latest ./mysql_db
docker build -t jperlas/authentication:latest ./authentication
docker build -t jperlas/video_streaming:latest ./video_streaming
docker build -t jperlas/file_system:latest ./file_system
```

```
kubectl apply -f deploy_mysql.yml
kubectl apply -f deploy_upload_video.yml
kubectl apply -f deploy_authentication.yml
kubectl apply -f deploy_video_streaming.yml
kubectl apply -f deploy_file_system.yml
```

```
docker tag jperlas/upload_video:latest gcr.io/acit3495ass2/upload_video
docker tag jperlas/mysql:latest gcr.io/acit3495ass2/mysql
docker tag jperlas/authentication:latest gcr.io/acit3495ass2/authentication
docker tag jperlas/video_streaming:latest gcr.io/acit3495ass2/video_streaming
docker tag jperlas/file_system:latest gcr.io/acit3495ass2/file_system
```

```
docker push gcr.io/acit3495ass2/upload_video
docker push gcr.io/acit3495ass2/mysql
docker push gcr.io/acit3495ass2/authentication
docker push gcr.io/acit3495ass2/video_streaming
docker push gcr.io/acit3495ass2/file_system
```

```
kubectl apply -f pvc_nfs_server.yml
kubectl apply -f deploy_nfs_server.yml
kubectl apply -f svc_nfs_server.yml
kubectl apply -f pv_nfs.yml
kubectl apply -f pvc_nfs.yml
```

```
kubectl apply -f svc_mysql.yml
kubectl apply -f svc_upload_video.yml
kubectl apply -f svc_authentication.yml
kubectl apply -f svc_video_streaming.yml
kubectl apply -f svc_file_system.yml
```

Step 5. Scale Automatically

Horizontal Pod Autoscaling (HPA)

- Automatically updates a workload resource (Deployment), with the aim of automatically scaling the workload to match demand
- If tracking only CPU, no further installation is needed

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-file-system
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: deploy-file-system
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 10
```

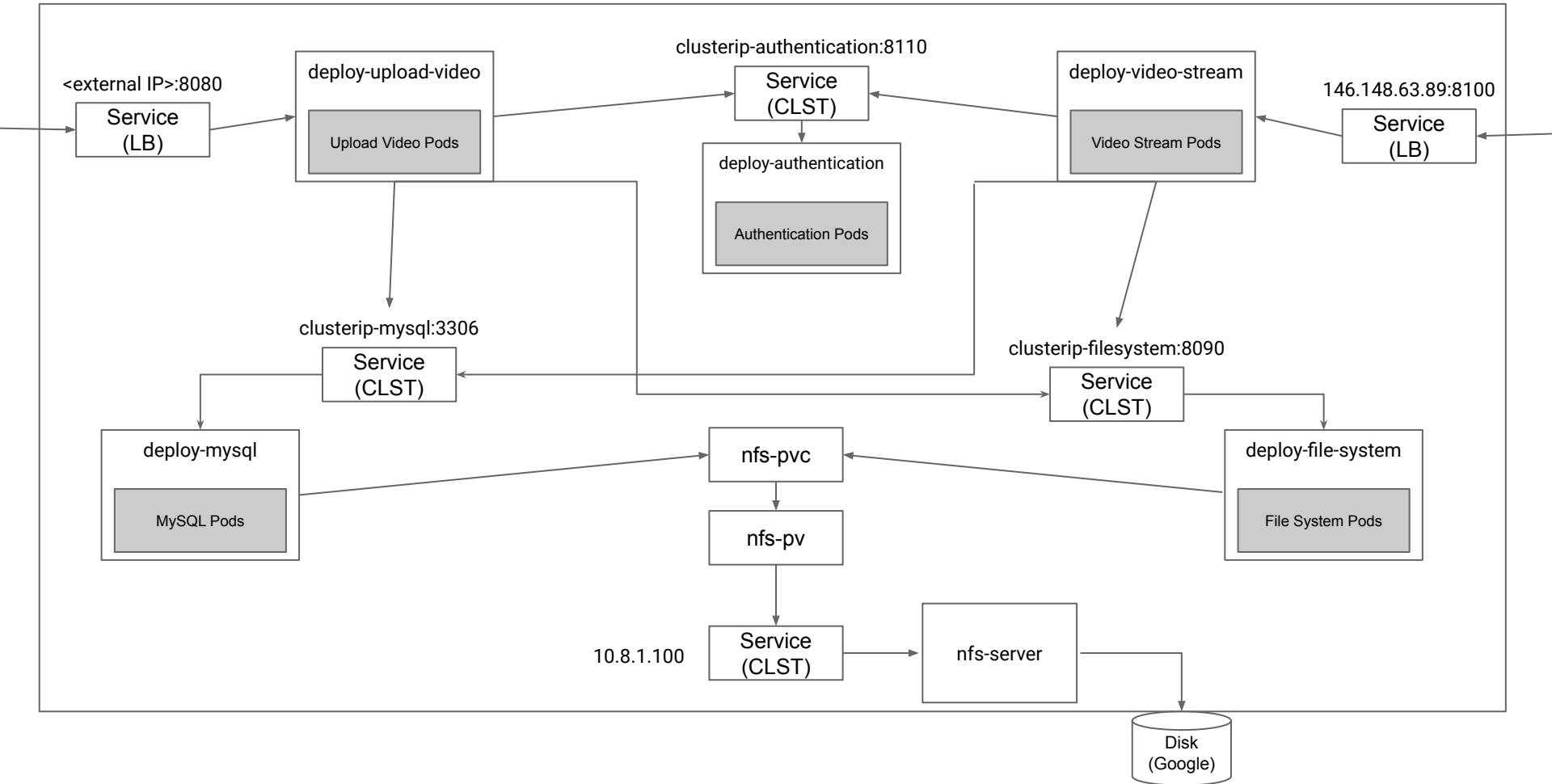
```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-upload-video
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: deploy-upload-video
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 10
```

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-authentication
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: deploy-authentication
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 10
```

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-video-stream
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: deploy-video-streaming
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 10
```

Full Topology

Google Kubernetes Engine Cluster-1



Further Issues (To-Do)

- HorizontalPodAutoScaling component is buggy
 - Sometimes doesn't grab CPU metrics from pods
 - Slow to scale down
- Deleting components via "kubectl delete -f <file>.yaml" sometimes gets stuck
 - Had to manually delete via GUI in Google
- Difficulties in trying to scale MySQL
 - Issues with MySQL requiring the folder in the NFS to be empty
 - Issues with locks when MySQL replicas are present
 - Not able to finish

Demo

Questions?