Deploying application on GCE using Kubernetes

So, we're trying to develop a cloud-native application using microservices philosophies as part of a Hackathon.

Starting Idea:

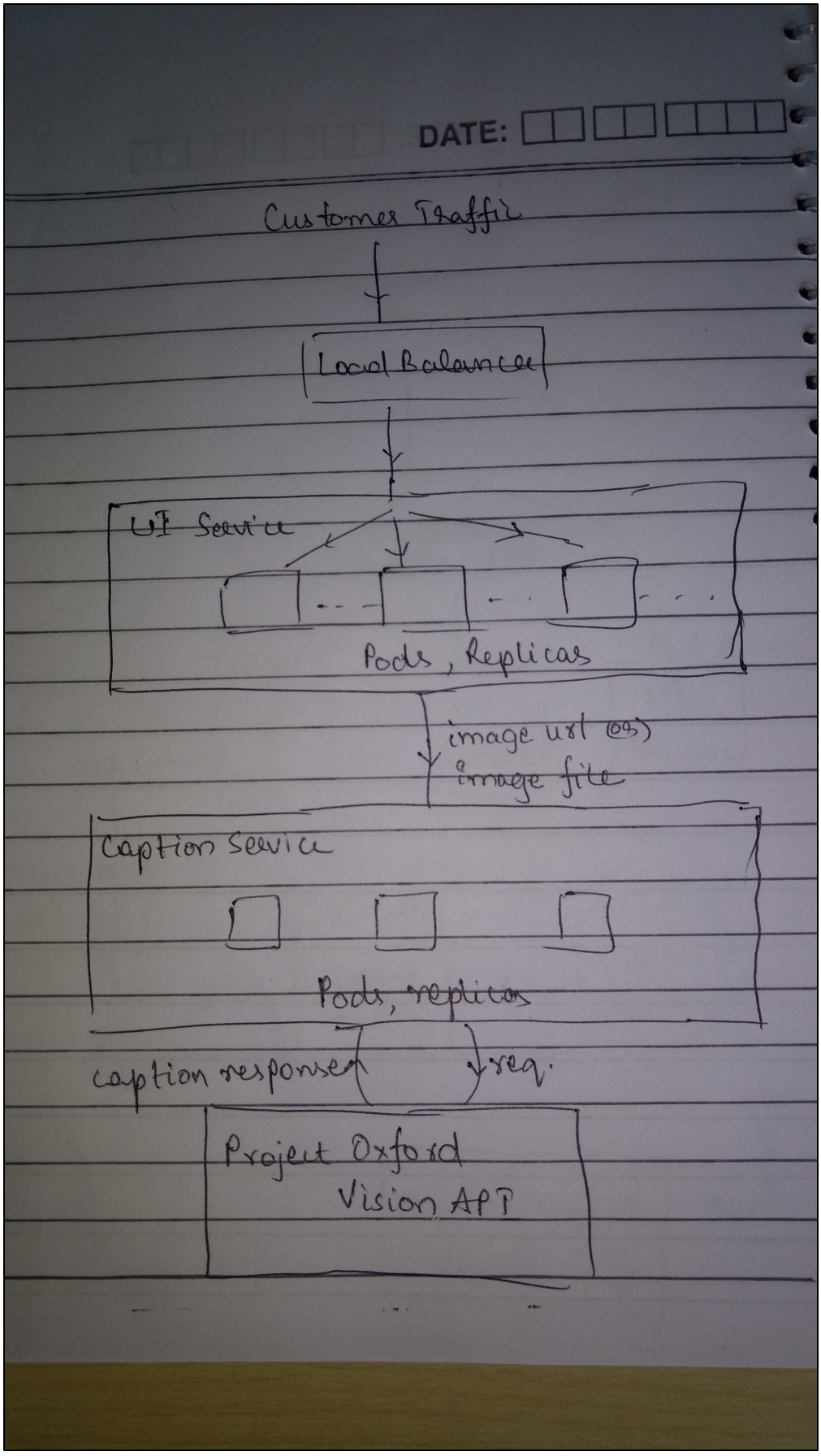
Image Content Captioning Service:

A service where the user will provide an image url or upload the image, The service return a text that describes the content of the image. e.g: A man in black shirt playing guitar.

Probable Services: CaptionService, Ui-Service, CacheService

Architecture Diagram:

DATE:


Customer Traffic
Local Balancer
UI Service
Pods, Replicas
image url (or)
image file
Caption Service
Pods, replicas
caption response
req.
Project Oxford
Vision API

Description:

The crux of the application is tensorflow based model (Maching Learning) which was already build and open sourced by the brain team at Google. We hoped to replicate this model and make it as a service.

https://github.com/tensorflow/models/tree/master/im2txt : Model from tensorflow for (Show and Tell: A Neural Image Caption Generator).

Unfortunately we could not use this right now due to give time constraints of 3 days. The model takes about weeks-months , so we used **Azure** vi sion api, which attempts to do the same.

Features in Service:

1. url : add url to get the caption. Progress : Complete
2. upload image: upload the image to get the caption. Progress: Complete
3. Recent Captions: get the list of recently generated captions. Progress: InComplete

Technology Stack:

1. ui-service: NodeJS, Express, pug templates, twitter Bootstrap.
2. caption-service: NodeJS, Express, request
3. CacheService: redis cache. (Planned)

Learnings:

Progress:

First, we tried to get a basic Hello World application(Single Service) running to get an idea of how Kubernetes and GCE work

- Following the second link(in references), I created a GCE account etc.
- Got the docker, gcloud, node.js running
- Build a docker image locally and verify that it runs without any problems
- Push the docker image to the Google Cloud Registry(The image url be something like this gcr.io/$PROJECT_ID/...)
- Create a k8s cluster and set the number of nodes etc you might want in it. This step will create your cluster with the number of machines as requested.
- You might want to look at the billing page before proceeding further. Although Google is giving 300$ free credits for its cloud platform, as long as we don't use more than 5 nodes in our cluster, we shouldn't be priced.
- Now, go ahead and start the deployment using `kubectl run $name --image=$image` command
- Before doing the above step, you will need to download a key for authorization purposes. Look at this for more details: ADC(Application default credentials)

Now that we have a single service running, the next step is to build one more and have a complete web application. Right now, we are planning for 2 services - UI Service and Caption Service. For this part, we mainly followed the video tutorial to get an idea of how to configure each of these services.

- Once we have the docker images corresponding to the services(I'm assuming they are built), we can start spinning up servers
  - We can push this docker images to the Google Cloud Registry(gcr) or have it at some public location allowing Kubernetes to pull it.
- As talked about in the video, we need to create a yaml file for creating the deployment. For more info on what's a deployment, go here
  - Something useful I found about managing deployments: You can use the `kubectl edit` command to change any settings of the deployment later (E.g., If you want to increase the replicas for this deployment or you want to roll out a new image)
- Then, we need to create a service to expose these deployments to the outside world or to our other microservices
  - In the case of a front end service, you would add a Load Balancer in the yaml settings. We'll need to mention the port mapping here(The port outside world uses => the one this service is listening on)
  - For a backend service, we need to expose some port for other services to connect to. We'll need to mention this in this file(The port other services connect => the one this service is listening on)
- So, in order for the frontend service to connect to backend, we need some sort of DNS service which would translate the name we use to the IP address of the backend machines
  - As talked abt in this, `metadata.name`value is the hostname of the pod. So, make sure that the name you use while connecting to a microservice is the same as the one used in `metadata.name` of that microservice

Notes:

1. The video linked in the references talks about building two services on Google Cloud using Kubernetes. Though they use gRPC for connecting the services, you might want to watch it to understand the purpose of Replication Controller yaml and the service yaml files used to start up a deployment and service respectively.

Related articles

2. http://kubernetes.io/docs/hellonode/ - Hello World app on GCE

Deploying application on GCE using Kubernetes