Microservices on Yelp Dataset deploy using Kubernetes

Abstract:

An application to annotate photos from Yelp dataset with Cloud Vision API, using Kubernetes and Microservice architecture to support portability, deployablity and scalability. The application makes use of Google’s Pub/Sub API for communication between microservices and Redis as backend storage. The Docker image container technology is used to build microservices and further pushed to google private container registry.

Introduction:

With the growing complexity of applications, not just the functionality wise but also the data generated and used, it highly important to leverage the application in various aspects such that they are up and running all the time(scalability), if one part of the application fails the other parts of the application should not be affected(Isolation), easy to be deployable on different cloud platforms and so on. Most of these suit of best practices called “Twelve-Factor Apps” can be bought into production of Software applications using technologies such Docker, Kubernetes and Microservice architecture.

In this project, a small application is built using the combination of these three technologies. In the below subsections, these concepts are briefly introduced to better understand the working of the application.

Docker

Modern applications are often built using different technologies with different versions based on the application requirement. Deploying many such application on a single Operating System could be a huge risk at times when there is a requirement of different versions of the same dependency. For instance, if you consider installing 2 versions of Nginx on the same OS, there would be a conflict with the namespace, network port making it logically cumbersome to do so. Although the concept of containerization is old, the ability to package applications with their dependencies into the container, making applications independent and isolated from other applications is huge progress to deploy. So it becomes very easy to run multiple containers using Docker. The docker image which is a snapshot of the container is the basis to build a container. The Docker container image is a packaging format that contains not only the application but also all the dependencies and runtime information. These images are hosted on public or private repository such as docker.hub, google storage.

There are 2 ways to start of with the docker image Dockerfile, previous docker image. Once the docker image is ready it takes simple commands listed below to build the container, push the final docker image to the storage repository, and pull the docker image whenever required.

Thus, it is easy and robust to create, distribute and run applications using Docker Containers with docker images and docker command-line tools. There are less or no restrictions for docker usage, as containers can be built on any machine with Docker installed it is highly in use by DevOps.

Kubernetes

Although Docker makes it easy to deploy and run applications using container technology when it comes to application configuration, service discovery, managing updates, secrets management and monitoring containers on the cluster, a better technology is required to leverage all of these tasks. Here comes Kubernetes, an open-source platform that provides a high level of abstraction and orchestration of containers deployed on one or more clusters, which in turn are treated as a single logical machine. Usually, a cluster has single Kubernetes master nodes that keep on running despite explicitly deleting, and zero or more worker nodes.

The master node is responsible for managing the cluster, whereas the worker nodes work like a VM, it consists of one or more pods, Volume, network ID and tools to handle container operations. A pod is the smallest unit of Kubernetes and it consists of one or more containers. All the containers in the pod have shared the same filesystem and IP address, this makes the communication between containers in a pod easy. Each of these pods created based on the scheme which is usually in YAML or JSON file format. The scheme covers important aspects of spec which specifies the Pod behavior, container name, container ports.

A pod without Services or Replication Controller cannot be accessed by the external client, neither scaling and distribution of the application are possible. Services provide an external interface for one or more pods. The Service schema definition has 3 important parameters: kind, metadata, and spec. The kind is set to Service to indicate a Kubernetes Service, which is deployment, pod in case of Kubernetes deployment, pod definition files. The label app and the name constitute the metadata. The spec mapping includes a ports mapping for port 80 with name HTTP. The selector is the key mapping in the spec and specifies a mapping to be used for selecting the Pods to expose via the Service. Therefore, the service diverts the network traffic to all its pods with the same label as the label selector specified in the Service spec, in a round-robin manner. There are 3 different types of Service: Load Balancer, Internal IP, Node port. If a Service type is ClusterIP, then the service is accessible only within the cluster via its internal port. Whereas if the service type is Node port then the service is accessible from outside the node port, which further routes the traffic to internal port Cluster IP of the service, that is automatically created. Similarly Load Balancer service type also automatically creates Node port and cluster IP. It gives access for the external user to ping the IP. In addition to this Load Balancer has the responsibility to balance the load between all the Pods in Service. Hence if the application is divided into 3 parts, the part which should

A Replication Controller manages the replication level of Pods as specified by the “replicas” setting in a Replication Controller definition or on the command line with the –replicas parameter. A Replication Controller ensures that the configured level of Pod replicas are running at any given time. If a replica fails or is stopped deliberately a new replica is started automatically.

With these 2 crucial features scaling and replication factor, Kubernetes keep microservices up and running all the time. Hence, Kubernetes is production-ready, which provides dynamic container cluster orchestration in real time. Kubernetes as a cluster manager provides the feasibility for deploying Microservices by breaking an application into smaller, manageable, scalable components that could be used by groups with different requirements; Fault-tolerant cluster in which if a single Pod replica fails (due to node failure,for example), another is started automatically; Horizontal scaling in which additional or fewer replicas of a Pod could be run by just modifying the “replicas” setting in the Replication Controller or using the replicas parameter in the kubectl scale command; Higher resource utilization and efficiency;

Google Cloud Platform

Google Cloud platform gives the flexibility to scale quickly and handle intense data while having the luxury of not having to maintain the robust infrastructure, servers, networks etc and create business solutions. It provides Cloud shell, which comes with a package of a command-line tool, temporary VM instance of GEC, and access to Google API with implicit authorization. Also, it supports language such as Python, Java, Go, PHP, Ruby and .NET. Moreover, the command-line tool exclusively supports Cloud SDK gcloud command line tool, which is used for information on cloud and use of text editors like.. The other alternative to Cloud Shell is to download Google Cloud SDK and enable Authorization through some keys.

Google API is a set of application programming interface which allows communication with google services and integration of other services.It is great tool for developers to perform operations and use its features easily, google map API, google Visualization API , good AJAX search are few examples.

Cloud Pub/Sub API is a message passing product that is highly useful for communication between independent applications hosted on Google Cloud Platform. The concept of Cloud Pub/Sub has 2 endpoints sender and receiver and having one instance cloud pub/sub would allow interaction between many applications. The main advantage of Pub/Sub compared to other messaging tools like RabbitMQ is it is asynchronous and decouples publisher from Subscribers, that is any Client who subscribed as Sender or Publisher can send , Receive messages irrespective of the client on the other side.

In this project psq : Cloud Pub/Sub, a powerful, scalable and reliable messaging tool, implemented using Python is used. It has features similar to rq, simpleq and celery. It forms the basis for communication between microservices, especially main application and frontend.

Cloud Vision API is the most popular API that Google has till date. it is very easy and efficient to analyze the content of the image, which has state-of-the art tools for Image detecting features like: face, text, label and document text,web detection.

It is further made easy to use, through Cloud AutoML suite. Using Vision AutoML, it is just one click away to upload images and run pre-determined, custom machine learning models. It is built based on Google’s powerful technology of learning-to learn, neural network architecture. In fact, building custom ML model is just few steps. First, uploading training dataset with images labeled into google bucket or human-support to label images and the ML model is trained according to the provided dataset. And then test data is passed, and accuracy of prediction, classification of test data set is determined. However, this feature of Cloud AutoML is accessible to only limited customers, but the basic feature of labeling the images such as data in Google is quite possible through REST API and are available to use in different programming languages.

Redis is open source in memory database and is useful as database, cache and message broker. It has different data structures, remote, persistent and scalable to address wide variety of problems.

Yelp Dataset

Yelp provides an open-source dataset for the challenge with students and university grads. It is usually in JSON and CSV format that can be downloaded from their website. In addition to this Yelp also gives access to their data through Yelp-Fusion. Yelp Fusion provides REST API to get access to search, business, metadata. In order to make use of these REST API Authentication is required, which is recently modified to the Private Key authentication method, which is a simple 2 step process. Create an account, create manage app, fill in details and the private key is generated.

Approach

The main aim of the application is to label photos from Yelp dataset retrieved on passing location and search term such as food, dinner, using cloud vision API. The application is divided into 3 microservices frontend, backend, and main app. Each of these functionalities is explained below.

Initial Setup

As mentioned above, the application requires 2 important API cloud vision API and Pub/Sub API, which have to enabled for the specific project id, the application would be started, in google cloud console. The best part for a software developer to test the working application is to launch directly using gcloud command-line tool, as it doesn’t require authentication setup.

For Cloud SDK installed on the local environment, setting up the authentication is crucial. For this, it is first required to create a service account and download service account key which is usually in JSON file format. Then set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS = [PATH], where Path is the file path of the JSON file downloaded from Google Console Dashboard.

Frontend Microservice

The frontend of the application plays a key role as the load balancer service for the entire application. It is basically a dynamic web-page, which allows the user to enter location, for example, “San Francisco, CA” and term like “food”,”dinner” . Based on these inputs, photos are fetched. As the frontend service is deployed as a load-balancer service an external IP is provided which enables user to access outside of the cluster through a web-browser.

Backend Microservice

The backend of the application is the storage service through Redis which is important for storage of images and their respective labels determined using Cloud Vision API. The redis is accessible through redis image specified in backend.yaml file. The data is stored into redis instance via Mainapp and retrieved in frontend in order to populate the page with resulting images-label pair.

MainApp Microservice

The mainapp provides the actual functionality of the application, starting from scraping the data to generating the desired output. The major functionalities involved in the service is briefly discussed.

The yelp\_images.py has 4 functions query\_api(), get\_business(), search() and request(); As the given location and term are passed to query\_api(), it sends a GET request and as a result business id’s of at most 10 businesses are extracted from the response object. For each business id, a get\_business() is called to retrieve business details. From the response object of business details, photos are extracted. All the photos are returned to the calling function in main.py.

In vision.py, for image passed, a post request is sent to Vision API to annotate the image, in particular, label detection feature is requested. The response object consists of further details of the image such as score, confidence, location, and so on but we are mainly interested in the label which is a description parameter of string type.

In storage.py, StrictRedis class is imported to instantiate redis object. The redis in memory storage is very useful because of it’s ability to store objects in key-value pair. Taking advantage of this all labels are stored with key labels and label-image pair is also stored. This makes very easy to retrieve data in the frontend service, by just simply looking for the associated imageurl for the label in the list of labels.

To summarize, main.py brings together all the above functionalities, it retrieves the data from yelp\_images.py, passes photos to vision.py to label each one of them and stores using storage.py. As pub/sub enqueues the whole process in main.py, once the task is done, the frontend gets triggered.

Pods, Services, Deployments

There is one yaml file for each of the microservice, which includes schema for service as well as deploymenet. The kubectl create -f <file.yaml> command included in the Makefile creates the pods,service and deployments. As discussed above the important parameter in general are type, label selector, replica, container image and the port where they are exposed.

In this application the frontend service is created as type load balancer. This exposes the service outside of the cluster through an external IP.

Once each of the pods are deployed, the application is production ready. The scalability is maintained with replica factor, which ensures deletion or failure of one or 2 pods doesn’t stop the application from running. Also the constant updates are made easy with rollback.

Execution

The execution of the application is incorporated through Makefile. Just by running make all inside yelplabel directory, will spin the docker container images, creation of pods,services and deployments. By executing the command kubectl get pods list all the pods created and kubectl get services will show the services up and running with their given IP address. The external IP of the frontend service, is the available to curl or run in the browser. This is also the load-balancer for the application, so on request the traffic is diverted to other services within the internal port and the entire application is up and running.

Challenges

The debug aspect is the difficult because of the layers of abstraction over the microservices, the deployments and everything works fine if there exists an error within one of the microservices, which gets hard to figure out at times. To an extent we can make use of kubectl logs and google cloud provide error report.

Conclusions

The application follows MVC architecture workflow which important dier in application deployment. With Kubernetes not just orchestration of docker components but the flexibility, scalability for the deployment of microservices is highly achieved.

Future Scope

I attempted to make use of Cloud AutoML, Vision API to label MNIST dataset. Which is a huge dataset of NIST authorized handwritten dataset, highly used for testing the accuracy of ML models for Computer Vision and Image Analysis. Unfortunately, In order to customize ML model is limited to an approved user. Using the same or more technology stack, I would like to work on MNIST dataset to detect and label handwritten images.