Distributed Systems: Lab 2

Scalability & Traffic Management

Due: Oct 2, 2022

Ridwan Hossain (100747897)

Discussion

Problem

When a client wishes to access a service from a system, it is generally through an API call from which that request is made. In order to make the API request, the client must know about the endpoint of each service they wish to use. This paradigm can introduce issues when faced with various changes that may need to be made to the system. For example, if an API were to be modified, any refactors that may be required as a result must occur on both the client and server sides. Load balancing also introduces areas of contingency as when instances of a service are scaled up or down, the client side must be updated as well. A similar issue arises when different versions of the same service exist. Whenever there are changes in the portion of user traffic which are being directed to newer versions of the same endpoint, the client must be updated as well. These sorts of issues bring forth concerns regarding the scalability of a system in regards to maintaining a concurrent client-server connection.

Solution

A proposed solution to the problems mentioned above is to introduce an API gateway in order to manage incoming requests and outgoing responses from clients to the system. By using an API gateway, the client only needs to know about a single endpoint and allows modifications to the API to remain internal opposed to requiring modifications to be made on the client-side as well. By abstracting the various endpoints of the system's services, API calls from the client can remain simple while modifications are made to the system's backend. Furthermore, client calls may be rerouted to whichever services they require, allowing the configuration of the system's services to change without needing to notify or modify the client. The addition of a gateway also helps to better consolidate the principle of scalability as the increase or decrease of the number of services is encapsulated behind the gateway and hidden from the client. Additionally, an API gateway will allow for some extra control over how updates to the system are deployed to clients, as extra logic can be implemented to configure the release of updates to the clients.

Requirements for the Solution

In order to sufficiently implement an API gateway as a solution to the initial problem, a few requirements must be fulfilled in order to be executed sufficiently. Those requirements are as follows:

- Implement fault tolerance capabilities to mitigate a single failure point.
- Ensure the gateway exhibits adequate performance capabilities to meet scaling requirements.
- Implement load balancing functionalities within the system's gateway.
- Perform adequate load testing to mitigate the risk of cascading failures.
- Gateway routing can be based on IP, port, header, or URL.

- Use the gateway type required for whether your gateway services are regional or global.
- Limit public network access to the system's backend services.

Which of these requirements can be achieved by the procedures shown in parts 2 and 3?

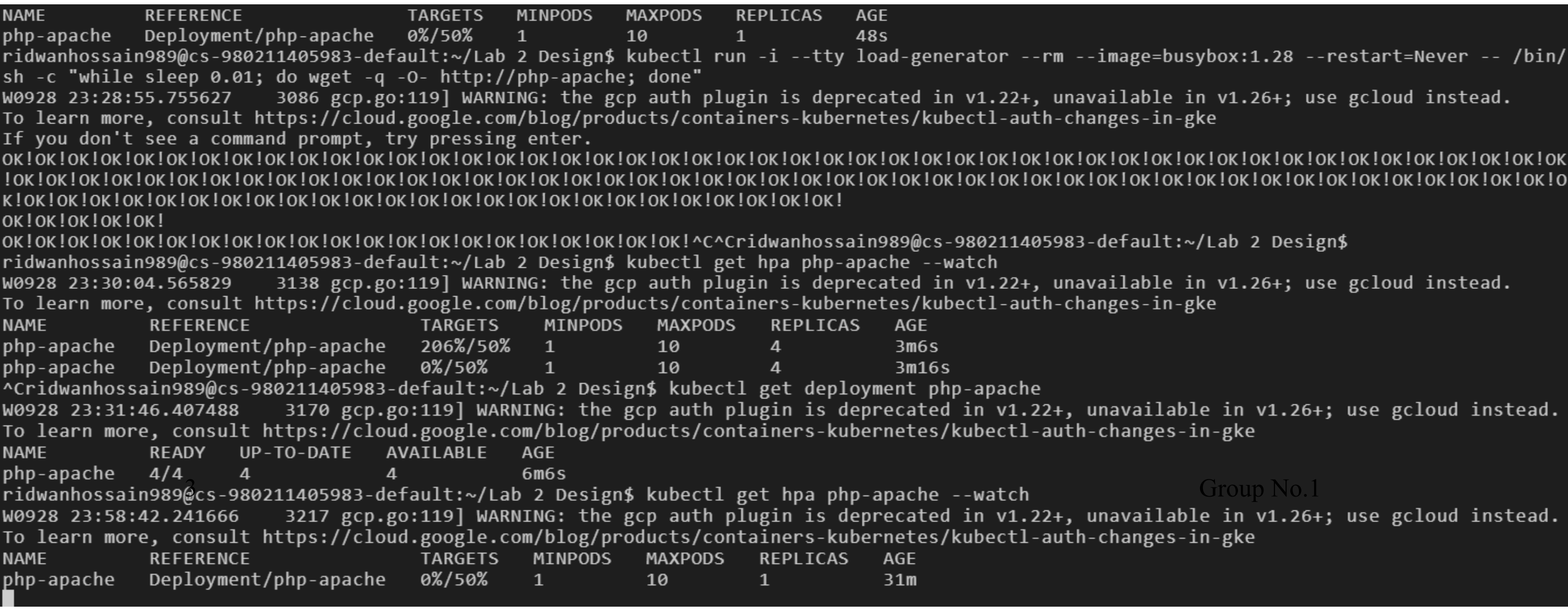
Part 2 achieves the requirement of ensuring load balancing within the system's gateway. As requests are made to the ambassador, it makes sure that 90% of the requests are distributed to the web-deployment server and 10% are distributed to the experiment-deployment server. The ambassador is performing load balancing functionalities and acts as this system's API gateway.

Part 3 achieves the requirement of limiting public network access to the system's backend services. The implementation of the simple NodeJS application are abstracted by the load balancing service which processes the requests received through the external IP address and returns the appropriate result from the corresponding API endpoint.

Design

Some assistance was acquired from the following link to aid with autoscaling the deployment of a pod:
https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

A yaml file titled php-apache.yaml was created in order to create a deployment that runs a container using the public hpa-example image and exposes it as a service. After creating the deployment, a horizontal autoscaler was created in order to generate more replicas of the pod if the service load gets too high. In the line 'kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10', we defined that the autoscaler will increase and decrease the number of pod replicas to try and maintain an average CPU utilization of 50% across all pods, with a minimum number of deployed replicas at 1 and the maximum of 10 replicas. We then used the line 'kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"' in order to increase the service load by send client requests in a looping fashion.



```
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          48s
ridwanhossain989@cs-980211405983-default:~/Lab 2 Design$ kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/
sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
W0928 23:28:55.755627    3086 gcp.go:119] WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.26+; use gcloud instead.
To learn more, consult https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!^C^Cridwanhossain989@cs-980211405983-default:~/Lab 2 Design$
ridwanhossain989@cs-980211405983-default:~/Lab 2 Design$ kubectl get hpa php-apache --watch
W0928 23:30:04.565829    3138 gcp.go:119] WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.26+; use gcloud instead.
To learn more, consult https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
NAME         REFERENCE               TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   206%/50%   1         10        4          3m6s
php-apache   Deployment/php-apache   0%/50%     1         10        4          3m16s
^Cridwanhossain989@cs-980211405983-default:~/Lab 2 Design$ kubectl get deployment php-apache
W0928 23:31:46.407488    3170 gcp.go:119] WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.26+; use gcloud instead.
To learn more, consult https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   4/4     4            4           6m6s
ridwanhossain989@cs-980211405983-default:~/Lab 2 Design$ kubectl get hpa php-apache --watch
W0928 23:58:42.241666    3217 gcp.go:119] WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.26+; use gcloud instead.
To learn more, consult https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          31m
```

As shown in the image above, as the php-apache service exhibits higher loads, we can see that the CPU consumption is approximately 206% per pod, so the horizontal autoscaler scales the number of replicas to 4. I then ended the request loop, and after some time, the CPU consumption dropped, resulting in the horizontal autoscaler to reduce the number of replicas back down to 1.