Load Balancing in gRPC (K8s)

Ujala Singh · Follow

8 min read · Oct 14, 2023

Scope

This article addresses the challenges and complexities of implementing efficient gRPC load balancing within a Kubernetes cluster. The goal is to ensure optimal distribution of traffic across gRPC-based services to achieve high availability, scalability, and performance.

Abstract

gRPC, which stands for "General Remote Procedure Call," is a high-performance RPC (Remote Procedure Call) framework that was developed by Google. It builds upon the foundation of Protocol Buffers (protobuf) and leverages the capabilities of

HTTP/2.0 to provide a powerful and efficient communication protocol for building distributed systems.

Background

Load-balancing within gRPC happens on a per-call basis, not a per-connection basis. In other words, even if all requests come from a single client, we still want them to be load-balanced across all servers.

See gRPC is fantastic, with high throughput (about 7X faster than REST), less computation, bidirectional, etc(not discussed here, OUT_OF_SCOPE_ERROR). Most of these fantastic features are because gRPC uses HTTP/2 protocol that provides **multiplexing** where the same connection is reused for multiple requests as long as the connection persists. And this is our problem. Many new gRPC users are surprised that Kubernetes's default load balancing often doesn't work out of the box with gRPC.

The Solution?

In exploring load-balancing mechanisms within our Kubernetes environment, we delved into leveraging Envoy as a critical intermediary layer. Employing the ROUND_ROBIN strategy, we observed auspicious results as it efficiently distributed the load across the various pods of our server. This effectiveness is readily demonstrable through the metric graphs we've collected. However, it became evident that traditional Kubernetes connection-based load balancing didn't align with the specific requirements of gRPC. Given that gRPC operates atop the HTTP/2 protocol, which relies on multiplexing and stream-based communication, Kubernetes' standard load balancing was less than optimal. This strategic move was driven by the need for a more compatible and efficient solution to address the unique characteristics of gRPC communication, ensuring seamless and performant interactions within our Kubernetes cluster.

Along with this, we will also explore the scaling of the gRPC servers and envoy proxy based on the custom metrics.

- grpc_requests_per_second: Measures gRPC requests rate.
- envoy_cluster_upstream_per_second: Tracks upstream cluster traffic rate.

Overview

The communication flows from a gRPC client through Linkerd Proxy, which acts as a service mesh, to an Envoy proxy serving as a load balancer, and finally, to a gRPC server. This setup ensures efficient, reliable, and secure communication between our client and server.

- gRPC Client: Our gRPC client initiates communication. This could be any application or service that wants to interact with our gRPC server.
- Linkerd Proxy (Service Mesh): Linkerd is used as a service mesh to facilitate advanced networking capabilities such as load balancing, service discovery, and security. When the client sends a request, Linkerd acts as an intermediary.
- Envoy as a Load Balancer: Within the Linkerd service mesh, Envoy proxies are deployed to handle routing and load balancing. Envoy ensures that incoming requests are distributed to available gRPC server instances effectively, optimizing resource usage and using a headless service.
- gRPC Server: The gRPC server is the final destination for the client's request. It processes the request, executes business logic, and sends a response back to the client through the same flow, ensuring a seamless and efficient communication process.

This flow enables us to manage our gRPC communication more efficiently, providing benefits such as load balancing, failover, and transparent service discovery, all while securing our communication within a service mesh. It's a powerful architecture that enhances the reliability and performance of our gRPC-based applications.

Why Envoy?

We needed a layer 7 (L7) load balancer because they operate at the application layer and can inspect traffic in order to make routing decisions. Most importantly, they can support the HTTP/2 protocol.

I chose Envoy (A smarter load balancer) as a load balancer proxy for the below-mentioned reasons.

- A proxy server created by Lyft using C++ as a language to achieve high performance.

- It has built-in support for a service discovery technique it calls STRICT_DNS, which builds on querying a DNS record and expecting to see an A record with an IP address for every node of the upstream cluster. This made it easy to use with a headless service in Kubernetes.
- It supports various load balancing algorithms, among others “Least Request”.

There are a number of LB policies provided by the envoy. The most notable ones are ROUND_ROBIN (the default), and LEAST_REQUEST.

Why Headless Service?

Each connection to the service is forwarded to one randomly selected backing pod. But what if the client needs to connect to all of those pods? What if the backing pods themselves need to connect to all the other backing pods? Connecting through the service clearly isn’t the way to do this. What is?

For a client to connect to all pods, it needs to figure out the IP of each individual pod. One option is to have the client call the Kubernetes API server and get the list of pods and their IP addresses through an API call, but because you should always strive to keep your apps Kubernetes-agnostic, using the API server isn’t ideal.

Luckily, Kubernetes allows clients to discover pod IPs through DNS lookups. Usually, when you perform a DNS lookup for a service, the DNS server returns a single IP — the service’s cluster IP. But if you tell Kubernetes you don’t need a cluster IP for your service (you do this by setting the clusterIP field to None in the service specification), the DNS server will return the pod IPs instead of the single service IP. Instead of returning a single DNS A record, the DNS server will return multiple A records for the service, each pointing to the IP of an individual pod backing the service at that moment. Clients can therefore do a simple DNS A record lookup and get the IPs of all the pods that are part of the service. The client can then use that information to connect to one, many, or all of them.

```
$ nslookup grpc-server-headless.grpc-server.svc.cluster.local
Server:  10.96.0.10
Address: 10.96.0.10#53

Name: grpc-server-headless.grpc-server.svc.cluster.local
Address: 10.244.0.12
Name: grpc-server-headless.grpc-server.svc.cluster.local
```

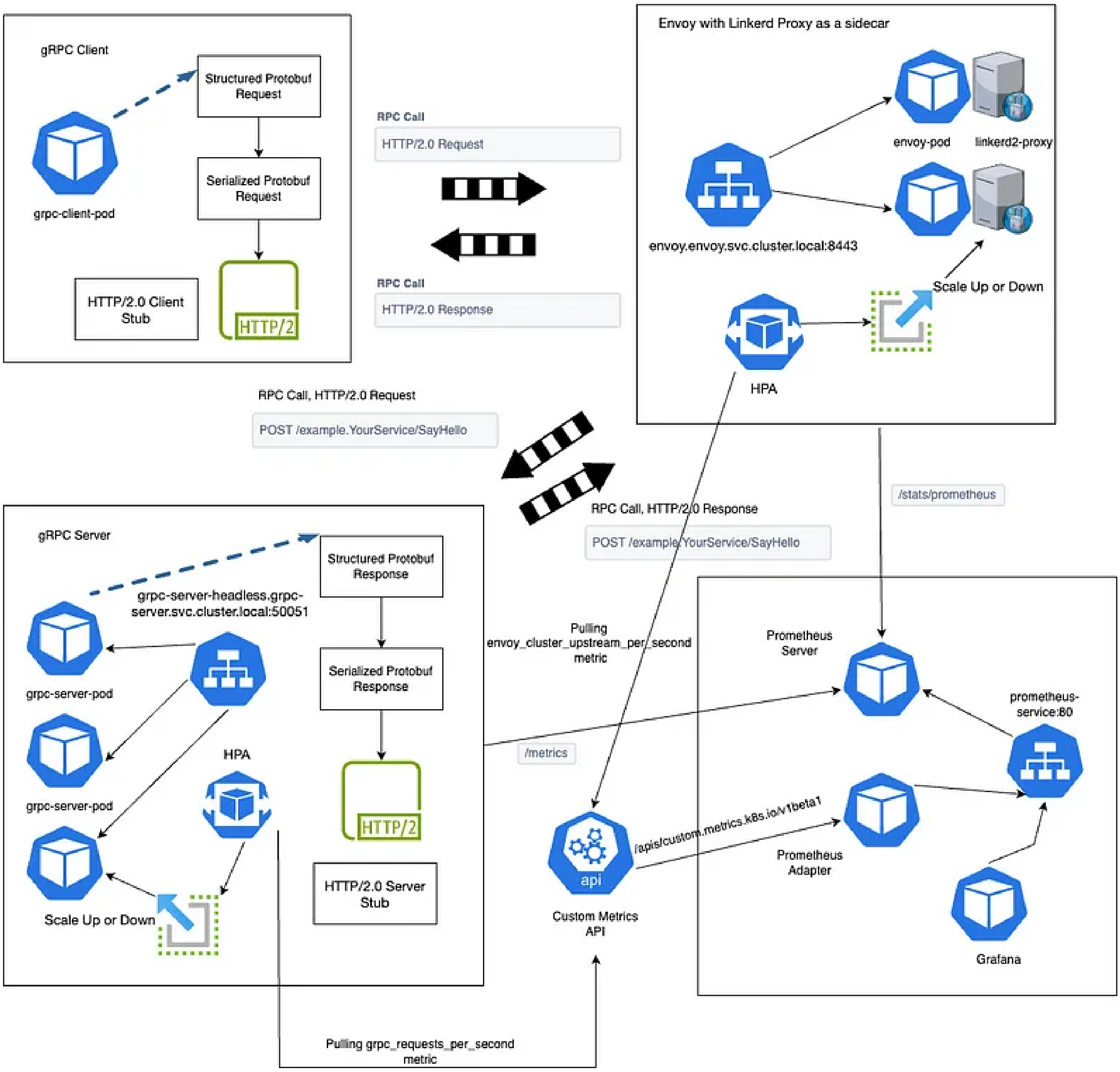
```
Address: 10.244.0.13

$ nslookup grpc-server.grpc-server.svc.cluster.local
Server:  10.96.0.10
Address: 10.96.0.10#53

Name: grpc-server.grpc-server.svc.cluster.local
Address: 10.101.229.142
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: grpc-server-headless
  namespace: grpc-server
spec:
  type: ClusterIP
  clusterIP: None
  selector:
    app: grpc-server
  ports:
    - protocol: TCP
      port: 50051
      targetPort: 50051
```

Architectural Diagram



Development Setup

Please follow the steps mentioned here on my GitHub repository.

Scaling and Monitoring

The Custom Metric API in Kubernetes is required if HPA is implemented based on custom metrics. Currently, most users use the Prometheus Adapter to provide the Custom Metric API. The Prometheus Adapter converts the received custom metric APIs to Prometheus requests and returns data queried from Prometheus to the Custom Metric API Server.

First, we have to make sure the metrics are present in Prometheus. These are the metrics we will be looking for:

```
grpc_requests_per_second
envoy_cluster_upstream_per_second
```

Next, we have to create a rule in the Prometheus Adapter:

```
prometheus:
  url: http://prometheus-service
  port: 80
  path: ""
rules:
  custom:
  - seriesQuery: 'envoy_cluster_upstream_rq'
    seriesFilters: []
    resources:
      overrides:
        kubernetes_namespace:
          resource: namespace
        kubernetes_pod_name:
          resource: pod
    name:
      matches: "^(.*)_rq"
      as: "${1}_per_second"
    metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy
  - seriesQuery: 'grpc_requests_total'
    seriesFilters: []
    resources:
      overrides:
        kubernetes_namespace:
          resource: namespace
        kubernetes_pod_name:
          resource: pod
    name:
      matches: "^(.*)_total"
      as: "${1}_per_second"
    metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy
```

You can look for the setup in the above-mentioned development setup link where I have explained it.

For troubleshooting purposes, I have also enabled logging into my gRPC server and client. Some sample logs are given below:

Server and Client Logs

Server Logs:

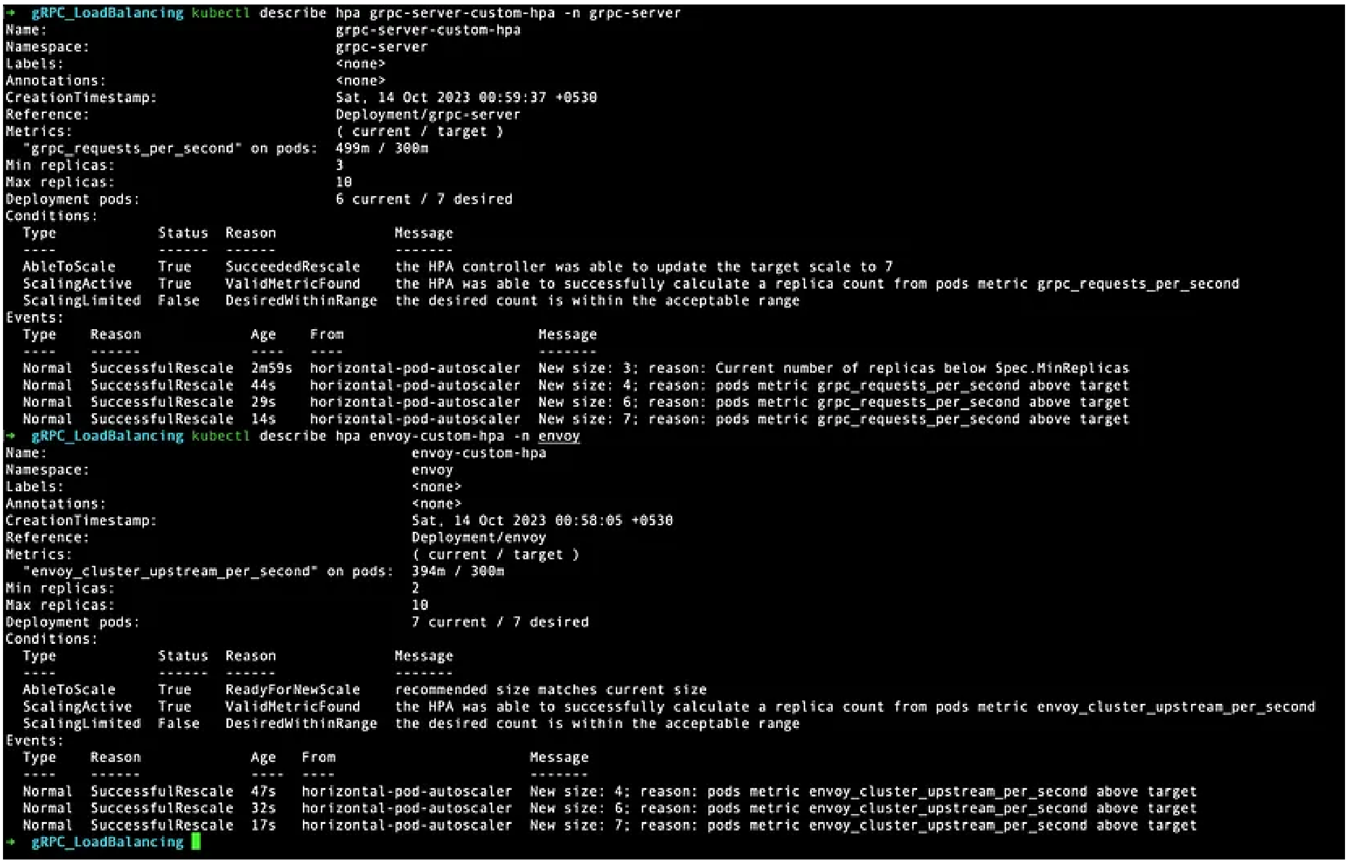
```
2023-10-14 15:09:45,053 - __main__ - INFO - Starting server. Listening on port
2023-10-14 15:10:29,687 - __main__ - INFO - Received a request from grpc-client
2023-10-14 15:10:34,697 - __main__ - INFO - Received a request from grpc-client
2023-10-14 15:10:54,758 - __main__ - INFO - Received a request from grpc-client
2023-10-14 15:11:04,784 - __main__ - INFO - Received a request from grpc-client
```

Client Logs:

```
2023-10-14 15:10:24,672 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:29,690 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:34,699 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:39,709 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:44,734 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:49,749 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:54,759 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:10:59,775 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:11:04,786 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:11:09,803 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:11:14,822 - __main__ - INFO - Received: Hello, Ujala! This messag
2023-10-14 15:11:19,849 - __main__ - INFO - Received: Hello, Ujala! This messag
```

Envoy Global Stats

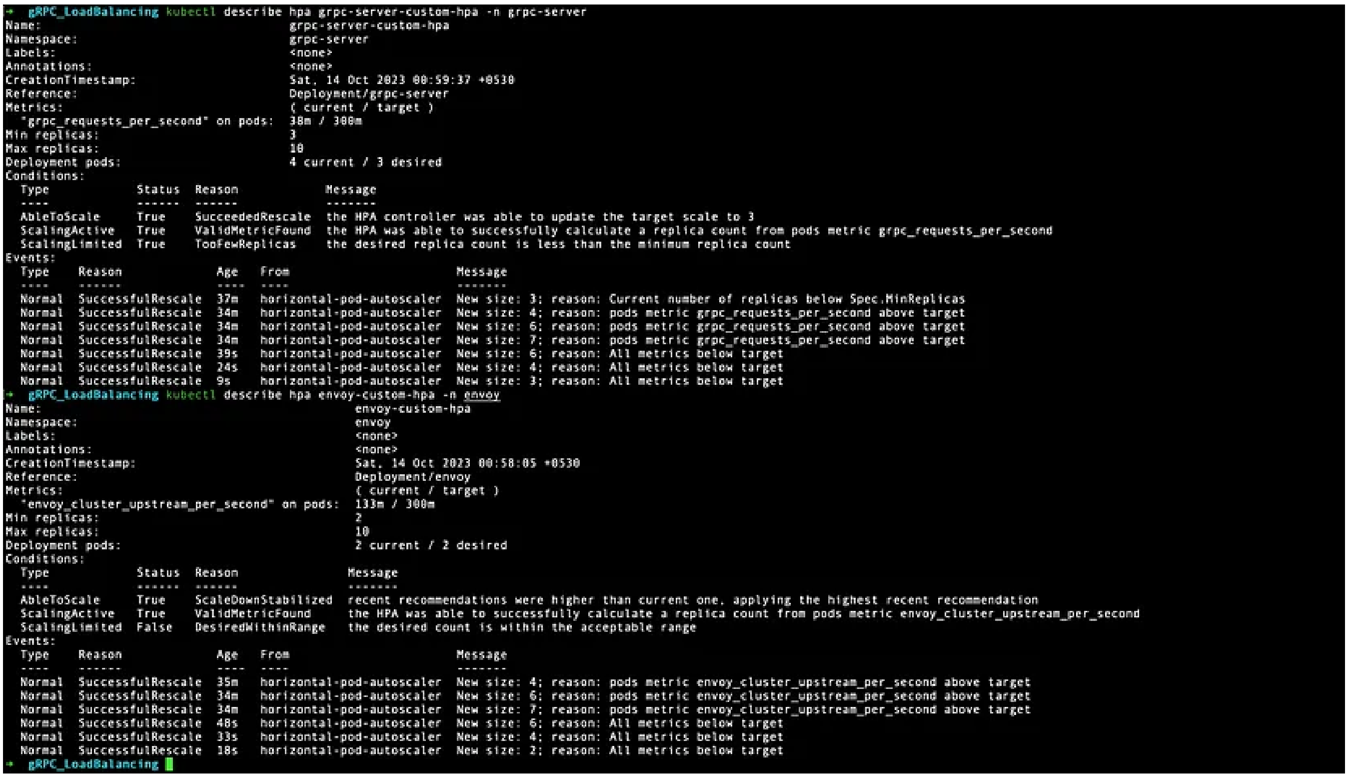
Envoy Service to Service Stats



```
→ gRPC_LoadBalancing kubectl describe hpa grpc-server-custom-hpa -n grpc-server
Name:                                                  grpc-server-custom-hpa
Namespace:                                             grpc-server
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Sat, 14 Oct 2023 00:59:37 +0530
Reference:                                             Deployment/grpc-server
Metrics:                                               ( current / target )
  "grpc_requests_per_second" on pods:                  499m / 300m
Min replicas:                                          3
Max replicas:                                          10
Deployment pods:                                       6 current / 7 desired
Conditions:
  Type            Status  Reason               Message
  ----            ------  ------               -------
  AbleToScale     True    SucceededRescale     the HPA controller was able to update the target scale to 7
  ScalingActive   True    ValidMetricFound     the HPA was able to successfully calculate a replica count from pods metric grpc_requests_per_second
  ScalingLimited  False   DesiredWithinRange   the desired count is within the acceptable range
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  2m59s  horizontal-pod-autoscaler  New size: 3; reason: Current number of replicas below Spec.MinReplicas
  Normal  SuccessfulRescale  44s    horizontal-pod-autoscaler  New size: 4; reason: pods metric grpc_requests_per_second above target
  Normal  SuccessfulRescale  29s    horizontal-pod-autoscaler  New size: 6; reason: pods metric grpc_requests_per_second above target
  Normal  SuccessfulRescale  14s    horizontal-pod-autoscaler  New size: 7; reason: pods metric grpc_requests_per_second above target
→ gRPC_LoadBalancing kubectl describe hpa envoy-custom-hpa -n envoy
Name:                                                  envoy-custom-hpa
Namespace:                                             envoy
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Sat, 14 Oct 2023 00:58:05 +0530
Reference:                                             Deployment/envoy
Metrics:                                               ( current / target )
  "envoy_cluster_upstream_per_second" on pods:         394m / 300m
Min replicas:                                          2
Max replicas:                                          10
Deployment pods:                                       7 current / 7 desired
Conditions:
  Type            Status  Reason               Message
  ----            ------  ------               -------
  AbleToScale     True    ReadyForNewScale     recommended size matches current size
  ScalingActive   True    ValidMetricFound     the HPA was able to successfully calculate a replica count from pods metric envoy_cluster_upstream_per_second
  ScalingLimited  False   DesiredWithinRange   the desired count is within the acceptable range
Events:
  Type    Reason             Age   From                       Message
  ----    ------             ----  ----                       -------
  Normal  SuccessfulRescale  47s   horizontal-pod-autoscaler  New size: 4; reason: pods metric envoy_cluster_upstream_per_second above target
  Normal  SuccessfulRescale  32s   horizontal-pod-autoscaler  New size: 6; reason: pods metric envoy_cluster_upstream_per_second above target
  Normal  SuccessfulRescale  17s   horizontal-pod-autoscaler  New size: 7; reason: pods metric envoy_cluster_upstream_per_second above target
→ gRPC_LoadBalancing
```

HPA Auto Scale-Up

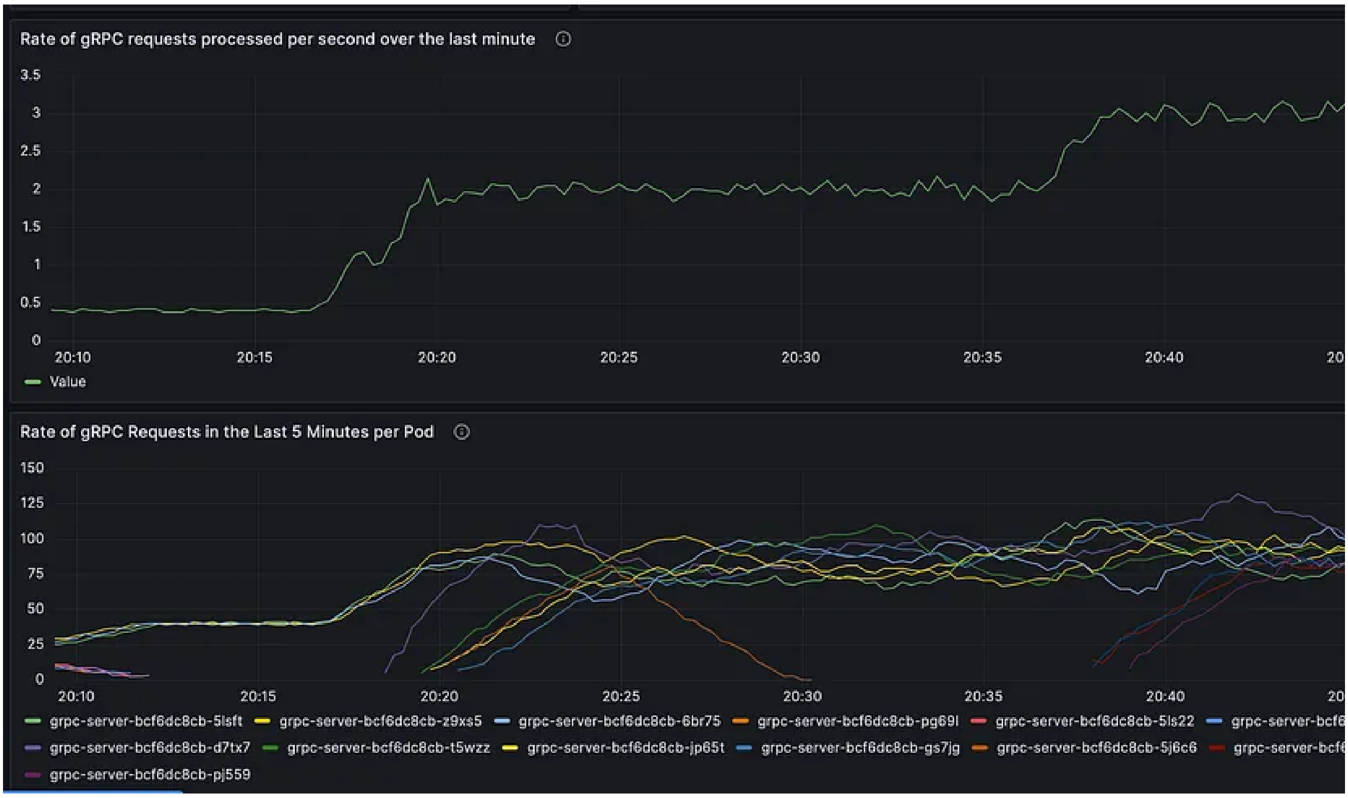
gRPC load balancing of servers while scaling up



```
→  gRPC_LoadBalancing kubectl describe hpa grpc-server-custom-hpa -n grpc-server
Name:                                         grpc-server-custom-hpa
Namespace:                                    grpc-server
Labels:                                       <none>
Annotations:                                  <none>
CreationTimestamp:                            Sat, 14 Oct 2023 00:59:37 +0530
Reference:                                    Deployment/grpc-server
Metrics:                                      ( current / target )
  "grpc_requests_per_second" on pods:         38m / 300m
Min replicas:                                 3
Max replicas:                                 10
Deployment pods:                              4 current / 3 desired
Conditions:
  Type            Status  Reason            Message
  ----            ------  ------            -------
  AbleToScale     True    SucceededRescale  the HPA controller was able to update the target scale to 3
  ScalingActive   True    ValidMetricFound  the HPA was able to successfully calculate a replica count from pods metric grpc_requests_per_second
  ScalingLimited  True    TooFewReplicas    the desired replica count is less than the minimum replica count
Events:
  Type    Reason             Age   From                       Message
  ----    ------             ----  ----                       -------
  Normal  SuccessfulRescale  37m   horizontal-pod-autoscaler  New size: 3; reason: Current number of replicas below Spec.MinReplicas
  Normal  SuccessfulRescale  34m   horizontal-pod-autoscaler  New size: 4; reason: pods metric grpc_requests_per_second above target
  Normal  SuccessfulRescale  34m   horizontal-pod-autoscaler  New size: 6; reason: pods metric grpc_requests_per_second above target
  Normal  SuccessfulRescale  34m   horizontal-pod-autoscaler  New size: 7; reason: pods metric grpc_requests_per_second above target
  Normal  SuccessfulRescale  39s   horizontal-pod-autoscaler  New size: 6; reason: All metrics below target
  Normal  SuccessfulRescale  24s   horizontal-pod-autoscaler  New size: 4; reason: All metrics below target
  Normal  SuccessfulRescale  9s    horizontal-pod-autoscaler  New size: 3; reason: All metrics below target
→  gRPC_LoadBalancing kubectl describe hpa envoy-custom-hpa -n envoy
Name:                                                  envoy-custom-hpa
Namespace:                                             envoy
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Sat, 14 Oct 2023 00:58:05 +0530
Reference:                                             Deployment/envoy
Metrics:                                               ( current / target )
  "envoy_cluster_upstream_per_second" on pods:         133m / 300m
Min replicas:                                          2
Max replicas:                                          10
Deployment pods:                                       2 current / 2 desired
Conditions:
  Type            Status  Reason                 Message
  ----            ------  ------                 -------
  AbleToScale     True    ScaleDownStabilized    recent recommendations were higher than current one, applying the highest recent recommendation
  ScalingActive   True    ValidMetricFound       the HPA was able to successfully calculate a replica count from pods metric envoy_cluster_upstream_per_second
  ScalingLimited  False   DesiredWithinRange     the desired count is within the acceptable range
Events:
  Type    Reason             Age   From                       Message
  ----    ------             ----  ----                       -------
  Normal  SuccessfulRescale  35m   horizontal-pod-autoscaler  New size: 4; reason: pods metric envoy_cluster_upstream_per_second above target
  Normal  SuccessfulRescale  34m   horizontal-pod-autoscaler  New size: 6; reason: pods metric envoy_cluster_upstream_per_second above target
  Normal  SuccessfulRescale  34m   horizontal-pod-autoscaler  New size: 7; reason: pods metric envoy_cluster_upstream_per_second above target
  Normal  SuccessfulRescale  48s   horizontal-pod-autoscaler  New size: 6; reason: All metrics below target
  Normal  SuccessfulRescale  33s   horizontal-pod-autoscaler  New size: 4; reason: All metrics below target
  Normal  SuccessfulRescale  18s   horizontal-pod-autoscaler  New size: 2; reason: All metrics below target
→  gRPC_LoadBalancing
```

HPA Auto Scale-Down

gRPC load balancing of servers while scaling down



gRPC server's behavior without load balancing

Conclusion

As I have already mentioned above Kubernetes' default load balancing does not seamlessly integrate with gRPC due to its specific requirements (as shown above). However, we found an effective solution by implementing Envoy as an L7 load balancer and incorporating Linkerd as a side proxy within the Envoy pods. This configuration not only addressed the challenges posed by gRPC but also provided a robust and efficient load-balancing mechanism, enhancing the overall performance and reliability of our services.

External Links

The GitHub Repository Containing the files used for the above approach discussed in this piece can be found here.

Grpc Load Balancing Kubernetse Envoy Envoy Proxy

Follow

Written by Ujala Singh

7 Followers

DevOps Enthusiast | Think Globally, Act Locally. No Proxy.

More from Ujala Singh

Ujala Singh

Sentiment Analysis on Twitter Data Using Apache Hive And MapReduce

Twitter is one of the renowned social media that gets a huge amount of tweets each day. In this tutorial, we will be analyzing tweets...

8 min read · May 8, 2020

6

Ujala Singh

Deploy WordPress and MySQL on K8s with the help of Custom Build Docker Images

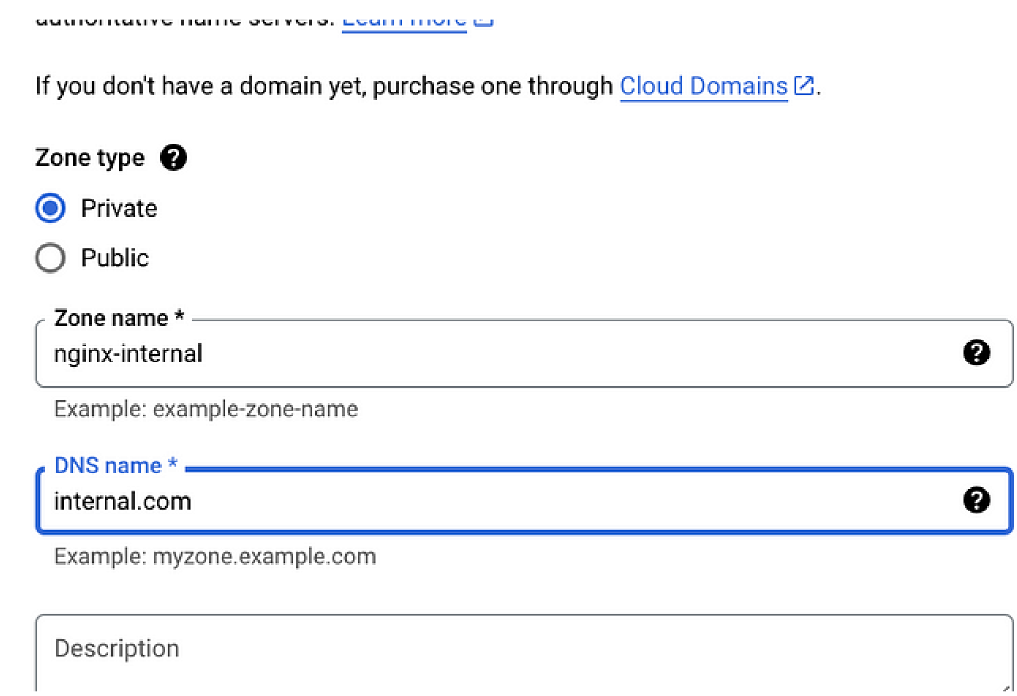
WordPress is a popular platform for editing and publishing content for the web. In this tutorial, I'm going to walk you through how to...

8 min read · Jul 16, 2021

1

See all from Ujala Singh

Recommended from Medium



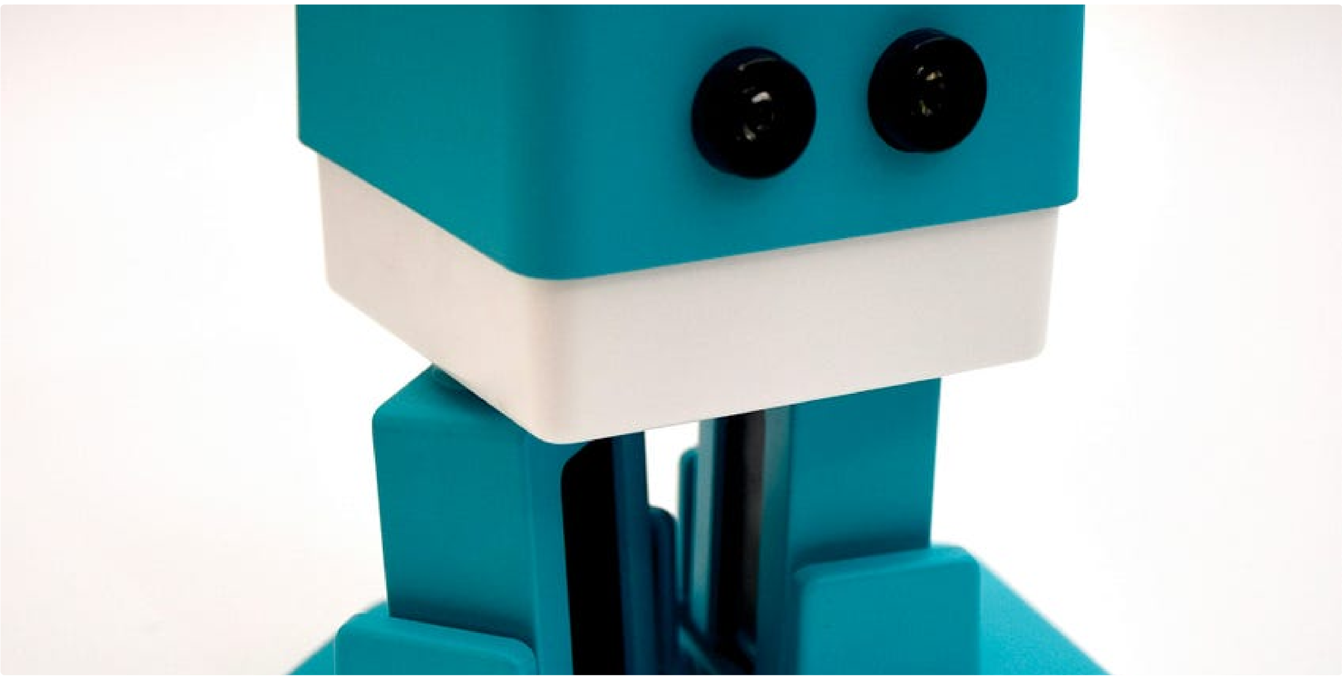
Nikhil YN

Configuring Internal Ingress GKE

Introduction: In the world of container orchestration and microservices, Google Kubernetes Engine (GKE) stands out as a leading platform...

5 min read · Sep 13, 2023

9



Akhilesh Mishra

You should stop writing Dockerfiles today— Do this instead

Using docker init to write Dockerfile and docker-compose configs

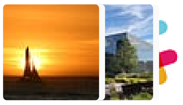
5 min read · Feb 9, 2024

1.95K 20

Lists

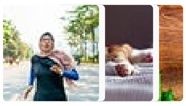
Staff Picks

584 stories · 760 saves



Stories to Help You Level-Up at Work

19 stories · 485 saves



Self-Improvement 101

20 stories · 1359 saves

Productivity 101

20 stories · 1247 saves

Talha Şahin

High-Level System Architecture of Booking.com

Take an in-depth look at the possible high-level architecture of Booking.com.

· 8 min read · Jan 10, 2024

2.6K 19

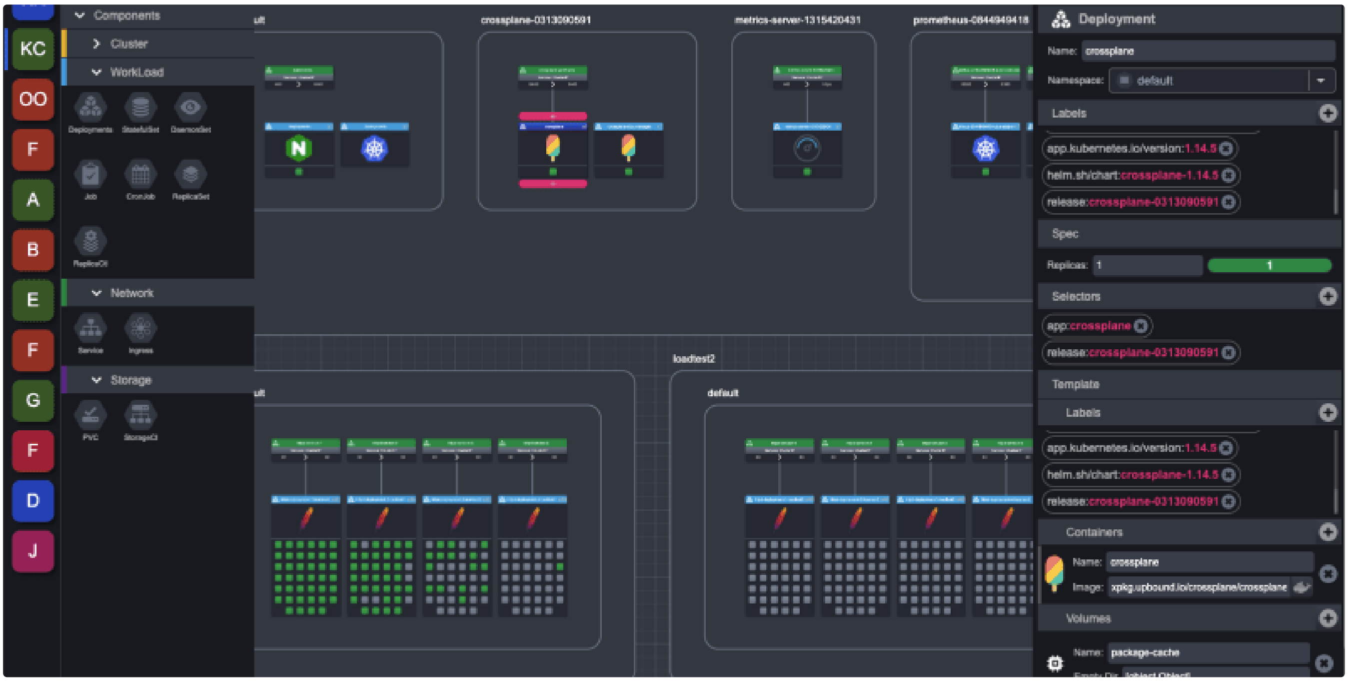
Anders Jönsson

Lessons From Our 8 Years Of Kubernetes In Production — Two Major Cluster Crashes, Ditching Self…

Cluster Crashes, Battling Complexity, Scaling, Power Of Helm, Tracing & Observability, From Self-Managed On AWS To Managed On AKS, And More

12 min read · Feb 5, 2024

791 10



Guillermo Quiros in ITNEXT

K8Studio Kubernetes IDE

It's been an exhilarating journey since we first embarked on the K8studio project four years ago. Although there were pauses along the way…

4 min read · Jan 16, 2024

863 13

```
          readOnly: true
        - mountPath: /etc/nginx/lua/plugins/modify_request/
          name: lua-scripts-volume
          readOnly: true
      dnsPolicy: ClusterFirst
      nodeSelector:
        kubernetes.io/os: linux
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      serviceAccount: my-ingress-nginx
      serviceAccountName: my-ingress-nginx
      terminationGracePeriodSeconds: 300
      volumes:
      - name: webhook-cert
        secret:
```

Wade Xu

Mastering Advanced Ingress-Nginx Techniques: Unleash the Power of Lua Scripts

In the dynamic world of Kubernetes and micro services, Ingress controllers play a pivotal role in routing traffic to your applications…

7 min read · Oct 8, 2023

18

See more recommendations