ICAI 2023
12th International Conference on Applied Informatics
Eger, Hungary, 2023

Challenges in service discovery for microservices deployed in a Kubernetes cluster – a case study

Baasanjargal Erdenebat[a], **Bayarjargal Bud**[b], **Tamás Kozsik**[a]

[a]Department of Programming Languages and Compilers,
ELTE Eötvös Loránd University, Budapest, Hungary
baasanjargal@inf.elte.hu, ORCID: 0000-0003-0471-7183
tamas.kozsik@elte.hu, ORCID: 0000-0003-4484-9172

[b]National University of Mongolia
bud.bayarjargal@gmail.com

Abstract

As Kubernetes becomes one of the most wide-spread infrastructures of the cloud-native era, its containerization possibilities and tools are gaining more and more popularity. The main goal of this paper is to evaluate the service discovery mechanisms and DNS management (CoreDNS) of Kubernetes, and to present an experiment on service discovery challenges. In large scale Kubernetes clusters, the total number of running pods, services, requests, and workloads can be high, and the increased number of HTTP-requests often result in resource utilization concerns, e.g., spikes of errors [2, 5]. The memory usage of Kubernetes DNS is predominantly affected by the number of pods and services in the cluster [3, 4]. Other factors include the size of the filled DNS answer cache, and the rate of queries received per CoreDNS instance [3].

In this paper, we investigate optimization possibilities with respect to performance and scaling for CoreDNS in Kubernetes. We propose a new implementation of plugins, which ensures the reliability of service discovery and DNS resolution functionality. We introduce a solution for the CoreDNS concerns and elaborate on the functionality of our implementation within service discovery. Experimental results in a real-world case show that our solution for the CoreDNS ensures consistency of the workload. Compared with the default CoreDNS configuration, our

Table 1. Measurement data

Pod/Service count	Max memory (MB)	Max CPU (cores)	Average response time (sec)	Network load Receive (MB/sec)	Network load Transmit (MB/sec)
0	19	0.1	0.0008	0.0031	0.0047
250	776	2.74	0.67	1.48	2.55
1054	8914	5.27	4.02	2.70	4.66
2000	16664	9.14	8.19	3.75	6.12

customized approach achieves better performance in terms of number of errors for requests, average latency of DNS requests, and resource usage rate.

When we first encountered resource consumption issues and a spike of errors, we turned the configuration of CoreDNS [1] to troubleshooting and debugging the main pain points. Our initial idea was to increase the number of replicas for the application to see whether this improves the performance and reduces errors. As we drilled down further with the application developer, we found that most of the failures were related to DNS resolution. That is where we started to experiment with the performance of DNS resolution in Kubernetes.

We have carried out a stress test on service discovery to identify bottlenecks and issues. For our experiment, we used a cluster with one master and 10 worker nodes, which were set up with the default settings of Kubernetes. We executed Java-based front-end applications and microservices as Kubernetes pods and Kubernetes services on that cluster.

The figures presented in Table 1 are based on data collected from the tests using the following setup.

- Master node: n1-standard-1 (16 vCPU, 32 GB memory)
- Worker nodes: n1-standard-2 (32 vCPUs, 125 GB memory)
- Networking: calico-3.19.1
- Kubernetes Version: 1.21.3
- CoreDNS Version: 1.7.0

As shown in Table 1, resource consumption and network load drastically increased when the total number of services and pods were raised. When over 1054 pods/services were running, the system consumed high amount of resources such as up to 16 GB memory and 9 CPU cores. Average response time for 2000 pods/services was around 8.19 seconds, which resulted in high latency and increased error rates. The CoreDNS function with default configuration occasionally crashed when running 500 external services and pods in the Kubernetes cluster. After this incident, we adjusted the memory resource “request/limit” in the CoreDNS deployment up to 8 GB from 170 MB and increased the total number of instances to four.

It turned out that for the current challenges adding extra CoreDNS instances or configuring HPA (Horizontal Pod Autoscaler) for the cluster based on number of requests, resource consumption, and number of workloads running on the cluster did not provide an appropriate solution, especially for large-scale clusters in which numerous projects and environments were being developed simultaneously. A specific solution was needed to properly address the concerns of request/response latency, network load, spike errors, and resource consumption.

Our proposed solution is the following:

- development of a technique to automatically increase related resources based on request frequency, and
- customize CoreDNS behavior by using specific plugins for addressing our needs and requirements.

References

[1] CoreDNS: *DNS and Service Discovery*, https://coredns.io/, Accessed: 2023.

[2] A. Heidari, N. Jafari Navimipour: *Service discovery mechanisms in cloud computing: a comprehensive and systematic literature review*, Kybernetes 51.3 (2022), pp. 952–981, doi: https://doi.org/10.1108/K-12-2020-0909.

[3] Kubernetes: *Using CoreDNS for Service Discovery*, https://kubernetes.io/docs/tasks/administer-cluster/coredns, Accessed: 2023.

[4] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, M. Parashar: *Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing*, in: Cloud Computing: Principles, Systems and Applications, ed. by N. Antonopoulos, L. Gillam, Springer London, 2010, pp. 195–217, isbn: 978-1-84996-241-4, doi: https://doi.org/10.1007/978-1-84996-241-4_12.

[5] N. Singh, Y. Hamid, S. Juneja, G. Srivastava, G. Dhiman, T. Gadekallu, M. Shah: *Load balancing and service discovery using Docker Swarm for microservice based big data applications*, Journal of Cloud Computing 12 (2023), doi: https://doi.org/10.1186/s13677-022-00358-7.