Optimizing Performance and Resource Consumption of Cloud-Native Logging Application Stacks

Gergő Csáti[1], István Pelle[2], László Toka[2]
[1]Ericsson Hungary, [2]*MTA-BME Network Softwarization Research Group*,
Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Hungary
gergo.csati@ericsson.com, pelle.istvan@vik.bme.hu, toka.laszlo@vik.bme.hu

***Abstract*—Nowadays cloud-based applications and Internet of Things use-cases are becoming more and more common in the field of IT benefiting from the virtually limitless resources and microservice-based deployment options available in the cloud. Observability in such environments is key for tracing application execution to detect possible malfunctions and anomalies. Collecting logs can greatly help in this regard, however, a high volume of logging data can add huge costs for the maintenance of the infrastructure gathering monitoring data. In order to increase the profitability of the application, monitoring-related infrastructure needs to have the lowest cost possible while still being able to fully serve the application's monitoring needs. In this work, we investigate this aspect and provide an evaluation of the resource footprint of one of the most prominent log collection services, Elastic Stack, from the perspective of its write path.**

I. Introduction

The popularity of cloud-native (CN) and Internet of Things (IoT) applications has increased significantly recently and this trend is likely to continue. Microservice-based cloud architectures and on-demand scaling provide a good foundation for the former, while the latter can leverage data aggregation and analysis deployed to cloud resources. As cloud infrastructures substitute on-premises resources, many formerly non-CN applications are getting transitioned to CN environments today.

To achieve high reliability, applications running in cloud environments need to employ robust logging procedures as logs are generated for every important event during the operations of an application. Analyzing application logs can help to discover workflow characteristics, detect anomalies and malfunctions, and identify root causes of errors. Components of the Elastic Stack (ES) [4], one of the most popular logging frameworks, perform well in IoT-related CN logging. The stack's ability to process large amounts of data is demonstrated by Talaş *et al.* [2] on a Smart City use-case. Beyond the features of a simple search engine, the stack provides powerful tools for management and monitoring that are particularly important for IoT systems, as demonstrated by Calderon *et al.* [5] in a generic setting and by other works [3], [12] in a cloudified smart building scenario. As avoiding security threats is also a cornerstone in IoT, logging needs to align with this aspect as well. A robust implementation of security logging leveraging the ES is provided by Noura *et al.* [6] while others [14] investigate the capabilities of the stack regarding the security analysis of security-related logs.

978-1-6654-0601-7/22/$31.00

Although logging is often mandatory for microservices, it is always subordinate to application components. As CPU, memory, and disk resources should be used for the primary task of the application, heavy resource-optimization should be performed for logging solutions. Myriads of different logging scenarios make the optimization of such solutions not at all trivial. To help plan the logging stack backing up CN IoT applications, we investigate optimization possibilities within the stack. We create a controlled environment where emulated load is sent to the stack and we evaluate the stack's performance under varying configuration options. Thus, our contributions are threefold: *i)* we shortly review CN execution environments and the ES, our chosen logging solution, *ii)* we present measurement setups for evaluating the resource footprint of the ES for storing and querying logs, and *iii)* we evaluate its write-path-performance. According to these, we structure the rest of the paper in the following manner. In §II we paint a general picture of the background of the CN logging framework. In §III we discuss related work from the field of logging framework performance testing. Then we describe our measurement methodologies and setup in §IV and discuss measurement results in §V before closing our paper with §VI.

II. Background

The CN environment: Diverse services can be implemented and accessed through the network in a fast, flexible way exploiting the CN concept. One of the most efficient ways to utilize resources in the cloud is to use containerized systems. Docker [8] is an open-source containerization service with operating system-level virtualization. Containerization is especially useful when working with microservices and is essential in the CN world. To manage the multitude of containers in a large cluster, an orchestration system is required. The most popular of these is Kubernetes [16] which orchestrates the placement, resource assignment, and life cycle of application containers running in pods on cluster nodes.

ES – The logging framework: Thanks to the CN revolution, logging frameworks have transformed into robust systems with complex pipelines for handling all logging-related activities ranging from collecting through processing to storing. The Elastic Stack [4], [11], also known as the ELK Stack, is one of the most advanced log collection systems that consists of four main components: collectors (Beats), a router and processor engine (Logstash), a store and search engine (Elasticsearch)

and a visualization component (Kibana) that combine into a feature-rich log processing pipeline. *Filebeat* (one of the *Beats* services) specializes in low resource footprint file-based log aggregation and shipping. *Logstash* ingests data simultaneously from multiple sources in multiple formats and transforms them on-the-fly before routing them to log storage services. *Elasticsearch* stores, searches, and analyzes multi-format logging data. When the engine ingests a record, it indexes it in a document by assigning types to record fields dynamically or according to the set configuration. Indices organize data both physically and logically and are mapped to primary and replica shards. According to guidelines, these need to be kept between 10–50 GB in size and below 20 in number for every GB of heap memory configured, to provide good performance. Shards are automatically distributed among data nodes and rebalanced when nodes are added or removed (e.g., on failure) or in case of high disk usage. Data nodes are tied to physical hardware and handle CPU, RAM, or IO-intensive operations. While data nodes serve ingestion and queries, master nodes control the cluster, and ingest nodes help in data preprocessing, monitoring, and in securing connections with other components of the stack. Although one node can fill multiple roles, for reliability purposes these are best kept separately, ideally on separate hardware as well. *Kibana* is a GUI for the stack that provides pipeline and ingested data management, monitoring, and visualization. It offers various metrics regarding the Elasticsearch nodes, e.g., resource consumption, shard, and index information.

III. Related Work

For logging stacks, resource cost awareness and minimization of overhead are well-studied aspects. Log4Perf [7] acts at the application level and provides suggestions on where to insert logging statements. Log2 [10] optimizes log ingestion and storage by providing a cost-aware logging mechanism that uses a multi-level filtering mechanism: redundant, irrelevant logs are dropped while still adhering to a storage cost budget. Contrary to these, Wiriya *et al.* [15] investigate the scaling behavior of logging stacks in an environment utilizing virtual machines. Moving closer to our chosen logging platform, ES is evaluated against CouchDB by Gupta *et al.* [13]: they analyze the advantages and disadvantages of each platform. One of the main advantages of the CN concept, the possibility for auto-scaling, is investigated by Cholomskis *et al.* [1] with respect to logging stacks, where the ES handles the collection and aggregation of metrics.

Contrary to the above works, our investigation fully focuses on the ES components themselves and places them in a containerized, CN environment utilizing different setups and configurations to map resource utilization. In this sense, it is closest to Rally [9] and Elastic's own blog post on Elasticsearch sizing [17]. The former gives a benchmark tool for measuring deployments, albeit without publishing any comparison cases, while the latter investigates only throughput aspects without looking into compute resource utilization characteristics.

IV. Measurement setup

Depending on the log shipping method, we create multiple pipelines to mimic real-life cases. In each case, the pipeline starts with an emulated application component that generates synthetic log records.

Application emulation: We use a purpose-built Python application for generating lines of text with randomized content in a structured or unstructured format. The configurable application is deployed as a Kubernetes pod.

Pipelines: Our first log ingestion pipeline originates with Filebeat. As shown in Fig. 1, the tool is deployed as a sidecar container. In this case, the application appends its output into a file located on an *emptyDir* volume that is shared with Filebeat which reads the shared logs and forwards them to Logstash from where they arrive in Elasticsearch.

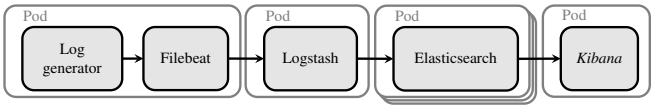

Fig. 1. Pipeline involving Filebeat

In our second pipeline setup, depicted in Fig. 2, we connect our application directly to Logstash and send log records via the HTTP protocol.

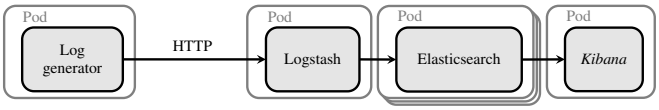

Fig. 2. Pipeline with direct HTTP connection to Logstash

Deployment and methodology: We deploy our pipelines to a Kubernetes cluster having 10 nodes each with 32 CPU cores and 64 GB of memory. For both Filebeat and the application, we allocate 1 CPU core, and 1 GiB of memory respectively. We set Logstash up with 2 CPU cores and 6 GiB of RAM with its JVM heap set to 4 GB according to guidelines. For Elasticsearch we use a setup of three nodes all with master, ingest, and data roles, having 1 CPU core, 2 GiB of RAM, with the JVM heap set to 1 GiB and 400 GB of persistent volume storage. Kibana is added to the pipeline to provide the necessary security configurations for Elasticsearch and monitoring. We track the volume and intensity of the ingested data, the throughput of the different scenarios, and potential log loss using the tool. We use the latest completely license-free versions of the ES components, i.e., version 7.10.2.

Measurements are started by reinstalling the collector components in the pipeline and deploying the application component that immediately starts generating log records. In certain cases, we use multiple application components to increase the load on the stack. We use Kubernetes' monitoring to query CPU and memory footprint having an averaging window of 30 s and collect peak and steady-state values. We supply CPU usage in percentage, where 100% is equivalent to 1 CPU core fully utilized.

V. Measurements and Evaluation

While multiple factors contribute to the resource footprint of the ES, here we focus on three of them.

Log record sizes and distributions: We experiment with two log record size distribution combinations: *i)* 0.5–5 kB/event, uniformly distributed, and *ii)* 10–64 kB/event having an exponential distribution with a decay factor of 17. The latter is selected to reduce load and resource consumption at the log generator. Using this distribution, out of 5 million log entries, on average, there are only 5 records sized approximately 64 kB, and the first 1 million log records are below 11 kB.

Load: In our experiments, we aim to reach loads that can be considered very high in real environments. To reach this, we use multiple instances of the generator application and set the log generation intensity to its technical maximum.

Sharding: We also evaluate different shardings in Elasticsearch to observe their effects in the various ingestion plans.

A. Ingesting high intensity 0.5–5 kB-sized logs with Filebeat

In this scenario, we measure the pipeline under heavy load and compare it to the lighter load case. We use Filebeat for this case, as it proved to be stable and reliable with the other ES components. The utilization of Filebeat also mimics real-life cases more realistically and has low overall footprint implications.

Our setup consists of two emulated application pods each containing one log generator container connected to one instance of Filebeat. We set the individual application containers to create 12.5 million log entries. Because of this large amount of data, we increase the number of shards in Elasticsearch to 12. This setup is motivated by the Elasticsearch guidelines, as it is recommended to not have shards greater than 50 GB. A summary of setup parameters and detailed results of this measurement can be observed in Tab. I.

It takes 417 minutes overall to ingest all 25 million log entries. The pipeline remains stable during the process and manages to ingest everything without any significant delays. Fig. 3 visualizes a summary of the peak CPU loads of each component in the stack. The larger intensity of logs results in a larger footprint, as can be expected. Filebeat's peak CPU usage is 17% percent while its RAM consumption peaks at 134 MiB. The CPU usage of Logstash peaks at 78% and it uses a maximum of 4.3 GiB of memory, which means it uses up all of the allocated JVM heap. The data nodes of Elasticsearch are working on full load. Each data node reaches its capacity in terms of CPU with an average usage of 91%. Elasticsearch can handle the load without any failures or lags.

TABLE I
Summary of cases using varying log ingestion intensities

	Low intensity	High intensity
Generation period	0.02	"max"
Ingestion time	10 min	417 min
Log size	0.5–5 kB evenly distributed	0.5–5 kB evenly distributed
Logs generated	29,834	25,000,000
Logs ingested	29,834	25,000,000
Logs lost	0%	0%
Average throughput	∼2,983/min	∼59,952/min
Peak CPU - Filebeat	1.4%	17% (avg. 13%)
Peak RAM - Filebeat	37 MiB	134 MiB
Peak CPU - Logstash	16%	78% (avg. 55%)
Peak RAM - Logstash	1 GiB	4.3 GiB
Peak CPU - ES nodes	7%	100% (avg. 91%)
Peak RAM - ES nodes	1.6 GiB	1.8 GiB

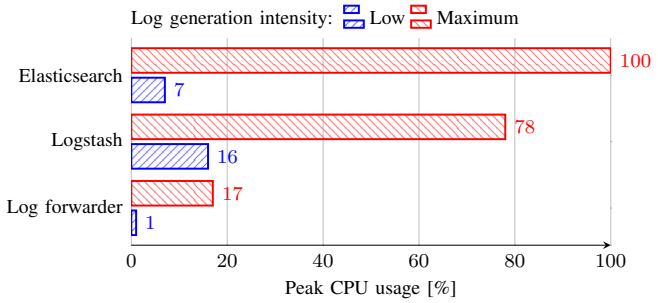

Fig. 3. Effect of different log intensities on peak CPU usage

Checking the rate of the ingestion throughput (in Kibana) shows that for approximately an hour the throughput remains stable around 60,000 logs/minute, but after that it starts to fluctuate. In our assessment this can be attributed to the data nodes getting really close to their limits. For intense loads like this, a three-node setup seems to be sufficient, but in an ideal case Elasticsearch should have more nodes to work with, in order to reach a more consistent ingestion rate.

B. Ingesting 10–64 kB-sized logs with Logstash via HTTP using different sharding plans

In this case, we measure the pipeline's footprint and performance under the load of 10–64 kB-sized logs. We compare measurements using different sharding plans, to see the effect of these different scenarios. With larger logs, the size of the storage becomes much more crucial. By using different sharding configurations, we can greatly influence Elasticsearch's storage management.

For these measurements, we deploy two application instances that create exponentially distributed log entries with maximum intensity. As per §IV, logs are directly sent from the application to Logstash via HTTP. In the following, we discuss the measurement results attained by utilizing two different setups that are also summarized in Tab. II.

In the first case, we aim to ingest 2 million logs into a single shard, with each data node having access to 100 GB of storage. Similarly to the previous measurement of §V-A, the load is significant on Logstash and the data nodes, as results shown in Fig. 4 attest. Logstash's peak CPU usage is 11% lower than in the case of the ingestion of the smaller logs with maximum intensity. The throughput is approximately a tenth of what we saw in §V-A, which is not surprising since the average size of an ingested log is also significantly larger, although not ten times larger.

As data nodes have 100 GB of storage allocated for them, the ingestion stops after 300 minutes because the single shard

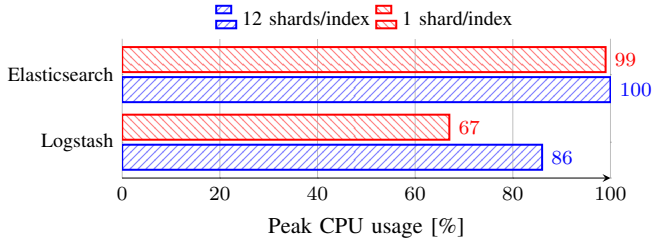

Fig. 4. Effect of sharding when ingesting 10–64 kB-sized records

is unable to allocate any more storage for the incoming logs. This illustrates why it is crucial to have more shards for large amounts of data as well as an issue where a low number of shards makes the complete Elasticsearch cluster get stuck while only the storage of a single node is filled up.

For the second measurement, we delete all the data from the nodes and redeploy them each with 400 GB of allocated storage. We set the number of shards for the index to 12 and start ingesting 5 million logs. This time the pipeline manages to ingest all the data in 919 minutes. The load on the data nodes and Logstash is similar to the load in the 1-shard-setup. The ingestion is slower overall because the increased number of shards also increases the overhead of the ingestion process. The average load on the data nodes is lower altogether, which indicates that in the ingestion's case, the size of the logs matters less than the amount of the logs.

VI. Summary

For CN IoT systems, a proper logging implementation is essential. IoT systems tend to generate a significant amount of logs, hence the optimization of the logging solutions behind these systems is a crucial, often non-trivial task. Elastic Stack can be a good solution in many use-cases, with a low resource cost when optimized properly.

Although our measurements are not exhaustive in terms of configuration options, we gathered important takeaways. These can be applied to the myriads of different use-cases that can be performed leveraging the ES while still keeping a low resource footprint in order to raise the profit realized on IoT or CN applications. First, the size of log records matters less than their aggregate number. This is most noticeable when it comes to the Elasticsearch data nodes, where CPU usage is significantly higher, and the ingestion happens with greater intensity, using smaller log entries. Second, sharding is very important for the indices. In order for the cluster to stay healthy, it is mandatory to have good shard management. Third, Logstash is one of the most demanding components in terms of resource consumption. In large systems, Logstash can be convenient to use for routing or to aggregate logs from disparate sources, but in simpler environments substituting it with Elasticsearch's ingest node pipeline can lead to lower overall resource consumption.

TABLE II
Summary of cases using different sharding setups

	Setup with 1 shard	Setup with 12 shards
Generation period	"max"	"max"
Ingestion time	300 mins	919 mins
Log size	10–64 kB exp. distribution	10–64 kB exp. distribution
Logs generated	2,000,000	5,000,000
Logs ingested	1,424,547	5,000,000
Used up storage	73.4 GiB	235.5 GiB
Average throughput	∼5,700/min	∼5,440/min
Peak CPU - Logstash	67%	86% (avg. 70%)
Peak RAM - Logstash	4 GiB	4 GiB
Peak CPU - ES nodes	99%	100% (avg. 60%)
Peak RAM - ES nodes	1.9 GiB	1.7 GiB

Acknowledgment

This work was supported by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund through projects *i*) no. 135074 under the FK_20 funding scheme and *ii*) 2019-2.1.13-TÉT_IN-2020-00021 under the 2019-2.1.13-TÉT-IN funding scheme. L. Toka was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. Supported by the ÚNKP-21-5 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

References

[1] A. Cholomskis et al. Cloud software performance metrics collection and aggregation for auto-scaling module. In *Information and Software Technologies*, pages 130–138, Cham, 2018. Springer.
[2] A. Talaş et al. Elastic stack in action for smart cities: Making sense of big data. In *ICCP*, pages 469–476, 2017.
[3] M. Bajer. Building an IoT Data Hub with Elasticsearch, Logstash and Kibana. In *FiCloudW*, pages 63–68, 2017.
[4] Elasticsearch B.V. Elastic Stack: Elasticsearch, Kibana, Beats & Logstash. Elasticsearch B.V. 2022. Accessed on: 20 January 2022. [Online]. Available: https://www.elastic.co/elastic-stack/, 2022.
[5] G. Calderon et al. Management and Monitoring IoT Networks through an Elastic Stack-based Platform. In *FiCloud*, pages 184–191, 2021.
[6] H. N. Noura et al. DistLog: A distributed logging scheme for IoT forensics. *Ad Hoc Networks*, 98:102061, 2020.
[7] K. Yao et al. Log4perf: Suggesting and updating logging locations for web-based systems' performance monitoring. *Empirical Software Engineering*, 25(1):488–531, 2019.
[8] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.
[9] D. Mitterdorfer. Announcing rally: Our benchmarking tool for Elasticsearch. Accessed on: 20 January 2022. [Online]. Available: https://www.elastic.co/blog/announcing-rally-benchmarking-for-elasticsearch, Apr 2016.
[10] R. Ding et al. Log2: A Cost-Aware logging mechanism for performance diagnosis. In *USENIX ATC 15*, Santa Clara, CA, 2015.
[11] R. K. Gupta et al. *Mastering elastic stack: Get the most out of the elastic stack for various complex analytics using this comprehensive and practical guide*. Packt, 2017.
[12] S. Dharur et al. Efficient surveillance and monitoring using the ELK stack for IoT powered smart buildings. In *ICISC*, pages 700–705, 2018.
[13] S. Gupta et al. A comparative study of Elasticsearch and CouchDB document oriented databases. In *ICICT*, pages 1–4, 2016.
[14] S. J. Son et al. Performance of ELK stack and commercial system in security log analysis. In *MICC*, pages 187–190, 2017.
[15] S. Wiriya et al. The enhancement of logging system accuracy for infrastructure as a service cloud. *Bulletin of Electrical Engineering and Informatics*, 9(4):1558–1568, 2020.
[16] The Kubernetes Authors. Kubernetes: Production-grade container orchestration. The Linux Foundation. 2022. Accessed on: 20 January 2022. [Online]. Available: https://kubernetes.io/, 2022.
[17] Y. Younes. Benchmarking and sizing your Elasticsearch cluster for logs and metrics. Accessed on: 20 January 2022. [Online]. Available: https://www.elastic.co/blog/benchmarking-and-sizing-your-elasticsearch-cluster-for-logs-and-metrics, Oct 2020.