KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Olli Ritari

MONITORING A KUBERNETES APPLICATION

Opinnäytetyö
Marraskuu 2019

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Marraskuu 2019** **Tietojenkäsittelyn koulutusohjelma** Tikkarinne 9 80200 JOENSUU +358 13 260 600 (vaihde)

Tekijä(t)
Olli Ritari

Nimeke
Monitoring a Kubernetes Application

Toimeksiantaja
Siili Solutions Oyj

Tiivistelmä

Tämän opinnäytetyön aiheena oli Kubernetes teknologialla toteutetun mikropalveluarkkitehtuuriin nojaavan ohjelmiston monitorointi. Monitorointi toteutettiin Elastic Stack työkaluilla toimeksiantajan projektissa.

Opinnäytetyö koostuu teoriaosuudesta ja case esimerkistä. Teoriaosuudessa selostetaan Kubernetes teknologian periaatteet ja toiminta perusteellisesti, sekä esitellään Elastic Stack -työkalut ja niiden konfigurointiperiaatteet. Case esimerkissä kuvataan, kuinka monitorointi on toteutettu toimeksiantajan sisäisessä tuotekehitysprojektissa.

Työn tuloksena toimeksiantajalle rakennettiin toimiva monitorointiratkaisu, joka tarjoaa helposti toistettavan asennustavan myös tuleviin projekteihin. Työ tuotti myös uutta tutkimusta Kubernetes klusterien monitoroinnista. Aiempaa tutkimusta juuri monitoroinnin näkökulmasta ei ollut toteutettu.

Kieli englanti	Sivuja 43 Liitteet 0 Liitesivumäärä 0

Asiasanat
Kubernetes, Elastic Stack, monitorointi, mikropalveluarkkitehtuuri

THESIS
November 2019
Degree Programme in Business Information Technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author (s)
Olli Ritari

Title
Monitoring a Kubernetes Application

Commissioned by
Siili Solutions Oyj

Abstract

The aim of this thesis is monitoring a microservice application, that is deployed with Kubernetes. The monitoring is implemented with a set of tools called Elastic Stack in a project for the client.

The thesis consists of a theoretical part and a case example. The theoretical part explains Kubernetes in detail and introduces the Elastic Stack and its configuration. The case example describes how monitoring is implemented with these tools in an internal product development project for the client.

As a result, a complete and fully working monitoring system was built for the client. It also provides a repeatable installation example for future projects. Furthermore, new research was produced about Kubernetes, specifically from monitoring perspective that was not previously covered.

Language English	Pages 43 Appendices 0 Pages of Appendices 0

Keywords
Kubernetes, Elastic Stack, monitoring, microservices

Contents

1 Introduction

The current thesis examines the monitoring of a software application that is deployed and maintained with a technology called Kubernetes and monitored with a set of tools called Elastic Stack. This thesis consists of a theoretical part and a case example. The theoretical part introduces the fundamentals of Kubernetes with figures and configuration examples. It also explains what the parts of Elastic Stack are, and how they are configured. The case example shows in detail how monitoring is conducted with these technologies in an enterprise-level project and how the data is visualized to browser-based dashboards.

Kubernetes is a technology and an open-sourced project of Google. Kubernetes provides a way to implement microservice-architecture either in cloud computing infrastructure (like Amazon Web Services or Microsoft Azure) or on-premises servers. Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs (Amazon Web Services 2019a). Elastic Stack is a set of open-source tools from a company called Elastic. These tools range from acquiring, sending, refining and storing logs and data to aggregating and visualizing them to timelines and dashboards. Using Elastic Stack for monitoring allows accessing and visualizing all gathered data from one user interface.

2 Client

The client of this thesis is Siili Solutions, a Finnish consulting company that provides IT-services to its customers. Siili has over 700 experts employed. Most of them are consultants, who help Siili's customers in the digitalization of their businesses. Other employees are project coordinators, such as scrum masters, salespeople and human resources workers. Siili has ten offices located in Finland, Poland, Germany and the USA. (Siili 2019.)

The case example of this thesis explains how the Elastic Stack is implemented in an internal product development project of Siili Solutions. The product is KnoMe-redux, an enterprise system for managing customer demand, employee competence and employee interests. This system is built to replace an older system which is complicated to maintain and has fewer features and integration capabilities.

3 Objective

The objective of this thesis is to provide a fully functional and repeatable example of monitoring a specific application-deployment. The monitoring implementation is configured and run in KnoMe-redux project and a comprehensive report is written of it. The client can then use this example as one possible implementation model in future projects. Having multiple toolsets from which to choose from alleviates the time-consuming configuration of a new project and lets developers concentrate on more case specific challenges.

A secondary objective is to create new research about this subject. Kubernetes is a growing technology, and not much has been written specifically about the monitoring aspect. The Finnish Theseus database holds works only about Kubernetes in general or how a specific application is containerized and deployed into a cluster. No previous reports exist that cover monitoring in particular. This thesis can, therefore, broaden the field of research and provide a new perspective on application monitoring.

4 Theory

4.1 Kubernetes

“Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.” (Kubernetes 2019a.)

The principal idea of Kubernetes is to run and manage scalable workloads inside containers. The containers are similar to virtual machines (and even physical servers), in the way that they have their own filesystem, CPU and memory. Containers, however, are considered lightweight compared to VMs. Kubernetes is developed and maintained by Google. Originally it was made for their purposes, but Google decided to open-source the project in 2014. (Kubernetes 2019a.)

A container is a piece of software that packages, not only application code, but also all its dependencies, runtime, libraries and settings into a reliable and portable unit. Docker is a technology and a product of Docker, Inc. Docker is considered to be the de-facto technology for running containerized workloads (Messina 2018). The definition and declaration of a Docker container is a Docker container image. Container images become containers at runtime. Because containers isolate the software from its environment, they will always be run the same, regardless of the infrastructure. An orchestration of containers that share a network is called a cluster. (Docker. 2019.)

Kubernetes provides a framework to run containers as distributed systems resiliently. For example, it takes care of scaling requirements, failover, deployment patterns, storage orchestration, rollouts and rollbacks and management of secrets and configuration. (Kubernetes 2019a.) Kubernetes can manage other container runtimes besides Docker as well, for example, CRI-O and Containerd (Kubernetes 2019b).

So, Kubernetes lets the user configure some aspects, but also automates a lot of the infrastructure where containers are run. No coincidence, the name Kubernetes originates from Greek, meaning helmsman or pilot. (Kubernetes 2019a.)

4.1.1 Overview

In the heart of Kubernetes is the clusters *desired state*. The desired state defines what workloads to run, what images the containers use, how many replicas of each container to have or what networks are made available, to name a few. Once the desired state is configured, the *Kubernetes Control Plane* makes the current state of the cluster match it. The desired state is defined in configuration files and set by creating objects using the Kubernetes API. The API is typically accessed via the command-line interface, *kubectl*. (Kubernetes 2019c.)

Kubernetes API can be accessed without kubectl as well. There are client libraries that enable API calls directly from other programs. A number of third-party cluster-management software is available also, for example, Rancher and a Finnish product, Kontena. These provide graphical user interfaces and pre-configured environments for accessing, managing and monitoring clusters.

4.1.2 Architecture

A running Kubernetes cluster consists of a master and several worker nodes. The master is another term for the control plane, and the nodes are either VMs or physical machines. Nodes are not inherently created by Kubernetes, rather by the cloud provider where the cluster is running (they can also exist on-premises). Kubernetes does create a node object for each node, so it can do a health check and decide whether or not the node is eligible to run a Pod. (Kubernetes 2019d.)

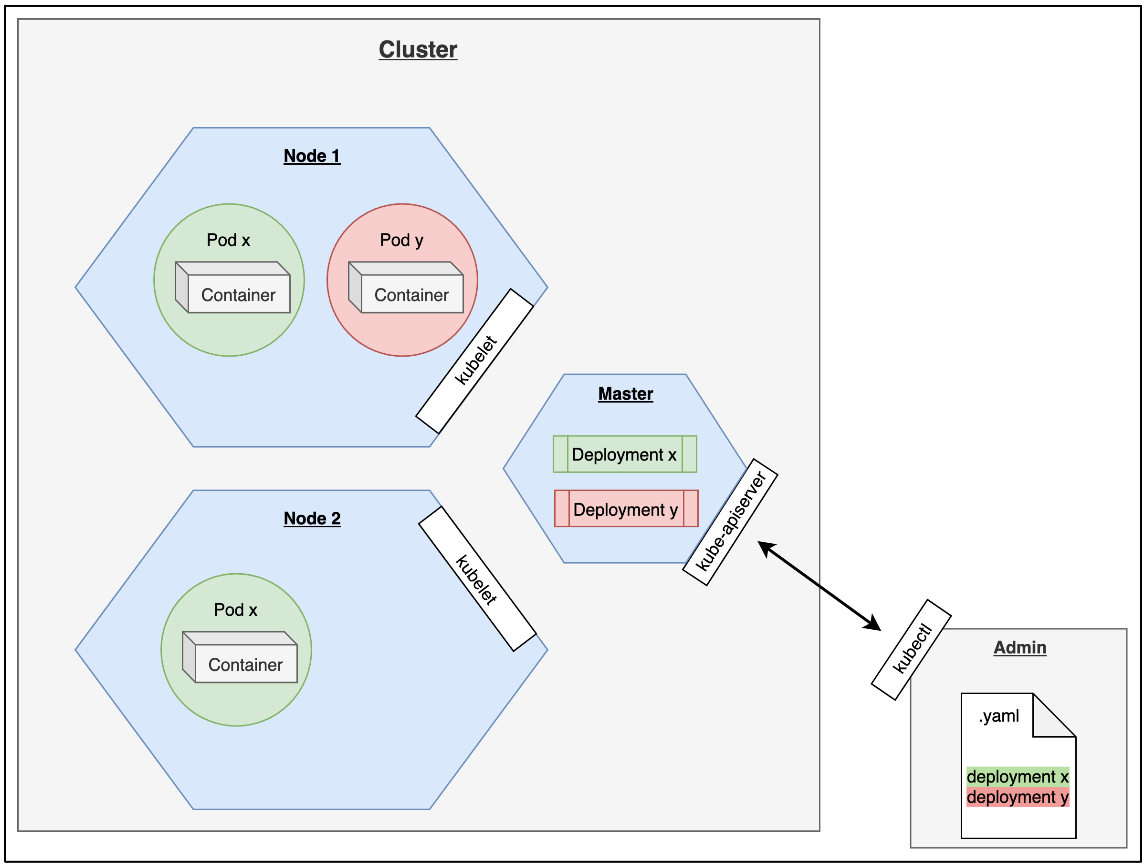


Figure 1. Simplified Kubernetes cluster.

Figure 1 illustrates a simplified Kubernetes clusters architecture. The cluster has two nodes, which could be cloud providers virtual machines for example, and a master that controls the nodes. The desired state of the cluster is defined in local yaml-files, which are pushed to the master's API using kubectl command-line tool. The master then creates objects for the resources and uses kube-scheduler and kube-control-manager to make the clusters state to match the desired state. In this case, the cluster runs two replicas of a Pod defined in deployment x and one replica of a Pod defined in deployment y.

4.1.3 Components

A functioning Kubernetes cluster requires various binary components. These components can be divided into three sections: Master components, Node components and Addons. (Kubernetes 2019e.)

Master components provide the cluster’s control plane. They make global decisions and respond to events. For example, they start a new Pod when the number of replicas is unsatisfied. Master components are typically run on the same machine, although they can reside on any machine in the cluster.

Master components are:

- kube-apiserver, which exposes the Kubernetes API.
- etcd, which is the distributed key value storage system where cluster state is backed.
- kube-scheduler, which selects a node for newly created Pods to run on.
- kube-controller-manager, which runs all controllers.
  - Node controller: Notices and responds to nodes going down
  - Replication Controller: Maintains the correct number of Pods for every replication controller object.
  - Endpoints Controller: Populates the Endpoints object (joins Services to Pods).
  - Service Account & Token Controllers: Create default accounts and API tokens for new namespaces.
- cloud-controller-manager, which runs all controllers that interact with the underlying cloud provider. (Kubernetes 2019e.)

Node components run on all of the nodes in the cluster. They maintain running Pods and provide the Kubernetes runtime environment.

Node components are:

- kubelet, which makes sure that containers are running in Pods inside the node.
- kube-proxy, which maintains network rules in nodes.
- Container Runtime, which is the software that is running containers (for example Docker). (Kubernetes 2019e.)

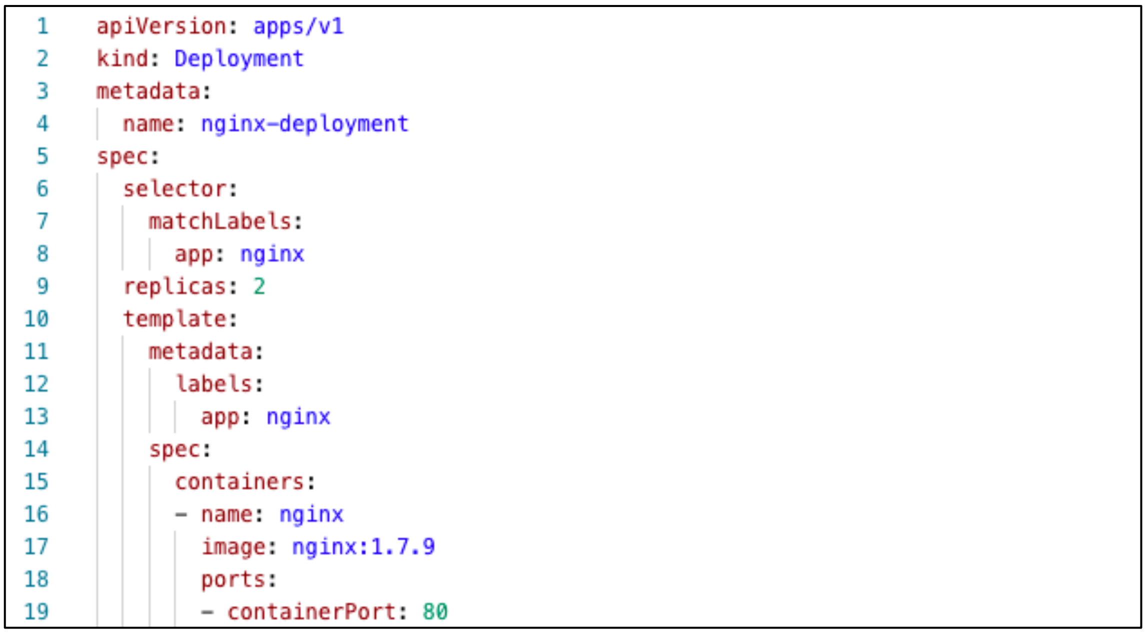
Addons use Kubernetes resources to implement cluster features. As the name suggests, addons are not required, but they can be useful. DNS addon provides a DNS server to the environment, which serves DNS records for Kubernetes services. Containers that are started by Kubernetes automatically include this DNS server in their DNS searches. Another addon, for example, is Web UI, which serves a web-based UI for managing and troubleshooting the cluster. (Kubernetes 2019e.)

4.1.4 Objects

Objects are the persistent entities in the Kubernetes system that represent the desired state. Configuration files are written and sent to the cluster's API, usually with the kubectl command-line interface. API calls are made with JSON in the request body. Usually, the configuration is written in a yaml-file, and kubectl converts the information to JSON when making the request. Kubernetes then creates objects based on this configuration. When the objects are created, Kubernetes will continuously work to ensure that the state of the cluster equals to the state defined in the objects. (Kubernetes 2019f.)

Kubernetes objects include two nested object fields, *spec* and *status*. The spec is provided by the user through the API and represents the desired state. Status is supplied and updated by the Kubernetes system and represents the current state. (Kubernetes 2019f.)

Figure 2 shows an example of a yaml configuration file. It describes one Deployment type object, which has the spec part of the object. In this case it describes an app that should be called *nginx*, which is based on the publicly available docker image *nginx:1.7.9*. It also describes that two simultaneously running instances are needed and port 80 should be open in these containers.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Figure 2. Yaml configuration example of a Deployment controller.

These yaml configuration files have four required fields: apiVersion, kind, metadata and spec. ApiVersion specifies which version of the Kubernetes API will be used. Kind states what kind of object is created, and metadata defines data that helps in identifying the object. Metadata includes name, UID, and optional namespace. Kubernetes supports multiple namespaces for separating resources inside the cluster. If no namespace is given, Kubernetes uses *default*. Spec field contains nested fields that are different for each type of Kubernetes object. These fields contain all the technical specifications that make up the desired state. (Kubernetes 2019f.)

4.1.5 Workloads

Workloads divide into two subcategories, *Pods* and *Controllers*. Pods are the basic execution units of a Kubernetes application. They are the smallest and simplest units in the object model that can be created and deployed. A Pod represents processes running on a cluster. It encapsulates an applications container, storage resources, a unique network IP and options on how to run the container. (Kubernetes 2019g.)

Pods can run either one or multiple containers. One container per Pod is the most common use case. It means basically that the Pod acts as a wrapper layer around the container and Kubernetes manages the Pod rather than the container directly. Multiple containers per Pod is used when containers are tightly coupled and need to share resources. One container is serving files from a volume and another container is refreshing and updating those files, for example. The other, updating container is also called a sidecar container. This sidecar implementation is also used in some addons and managing tools. They allow manipulation of the application container from within the cluster. (Kubernetes 2019g.)

Although single Pods can be described and created as objects, this is very rarely a valid use case. Because Pods are designed to be disposable and re-creatable, they are usually described with a higher-level abstraction, a *controller*. Because each Pod runs a single instance of an application, to scale the application horizontally (run multiple instances) the number of Pods is increased. This behavior is called *replication,* and these replicas are created and managed as a group by the controller.

Kubernetes Controllers are:

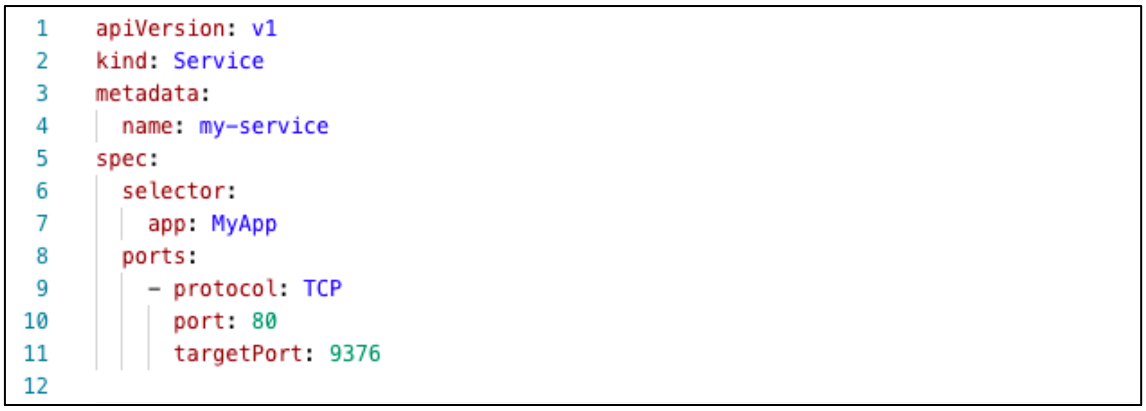
- ReplicaSet, which maintains a stable set of replica Pods. (Kubernetes 2019h.)
- Deployment, which provides declarative updates for Pods and ReplicaSets. A deployment can be modified dynamically after it has been created. For example, changing the image of containers causes a rolling update with no downtime. Deployment is a controller used to deploy stateless applications. (Kubernetes 2019i.)
- StatefulSet, which, like a Deployment manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These Pods are created from the same spec but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling. StatefulSets is a controller used to deploy applications with stable identifiers e.g. unique network identifiers or persistent storage. (Kubernetes 2019j.)

- DaemonSet, which ensures that all (rarely some) nodes in the cluster run a copy of a Pod. A typical use case is running a logs collection or a monitoring daemon on each node. (Kubernetes 2019k.)
- Carbage Collection, which deletes certain objects that once had an owner but no longer have. Carbage Collection controller is part of the cluster and is not manually deployed. (Kubernetes 2019l.)
- Job, which creates one or more Pods and ensures that they complete successfully. When a specified number of successful completions are reached, the job is complete. Deleting a job deletes all Pods it created. An example use case for a job is populating a database with a container. After the population is complete, the job completes, and the Pod is killed. (Kubernetes 2019m.)
- CronJob, which runs jobs on a time-based schedule, contrary to the job controller, which only runs a job once on its creation. An example use case could be a scheduled backup job, which runs daily or weekly. (Kubernetes 2019n.)

4.1.6 Services and Networking

Containers that are inside the same Pod can communicate using localhost. When a container communicates with entities outside the Pod, shared network resources need to be configured. Each Pod is assigned a unique IP address for this. However, since Pods are mortal, the set of Pods running at a given time can differ from the set running a moment later. This leads to the need for an abstraction layer, hence *Services*. (Kubernetes 2019o.)

Service defines a logical set of Pods and a policy by which to access them. The set of Pods targeted by a service is usually determined by a selector (figure 3). For example, a frontend Pod serving a web interface to a client needs to acquire data from a backend Pod that is running in 3 replicas. The frontend does not care from which of these three the response is coming from. In fact, it does not know how many replicas are running at a given time. Knowing the service is enough for the frontend, and the service abstraction will route the traffic accordingly. (Kubernetes 2019o.)



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376

```

Figure 3. Yaml configuration of a Service.

Figure 3 shows an example of a Service configuration file. It creates a service which targets TCP port 9376 on any Pod that has label app=MyApp. Kubernetes assigns an IP address for this service (also called a cluster IP), which is used by the Service proxies. As stated before, each node in the cluster runs a kube-proxy which is responsible for implementing this virtual IP for these services.

Publishing services has two basic options: internal and external. External services means exposing the service to connections from outside the Kubernetes cluster. The type of service is specified in the configuration yaml with ServiceType annotation. (Kubernetes 2019o.)

Possible service types are:

- ClusterIP, which exposes the service on an internal IP. This is the default ServiceType.
- NodePort, which exposes the service externally on each nodes IP and a static port. A ClusterIP where this service routes traffic is automatically created as well.
- LoadBalancer, which exposes the service externally using a cloud provider's load balancer. Both a NodePort and a ClusterIP are automatically created for the LoadBalancer to route traffic to. Example use case could be an applications frontend, which needs to be accessible from the internet and needs specific load balancing rules.

- ExternalName, which maps the service to the value of externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying is set up, so the redirection happens at the DNS level. Example use case could be a database that is outside the cluster and needs to be accessed via service. (Kubernetes 2019o.)

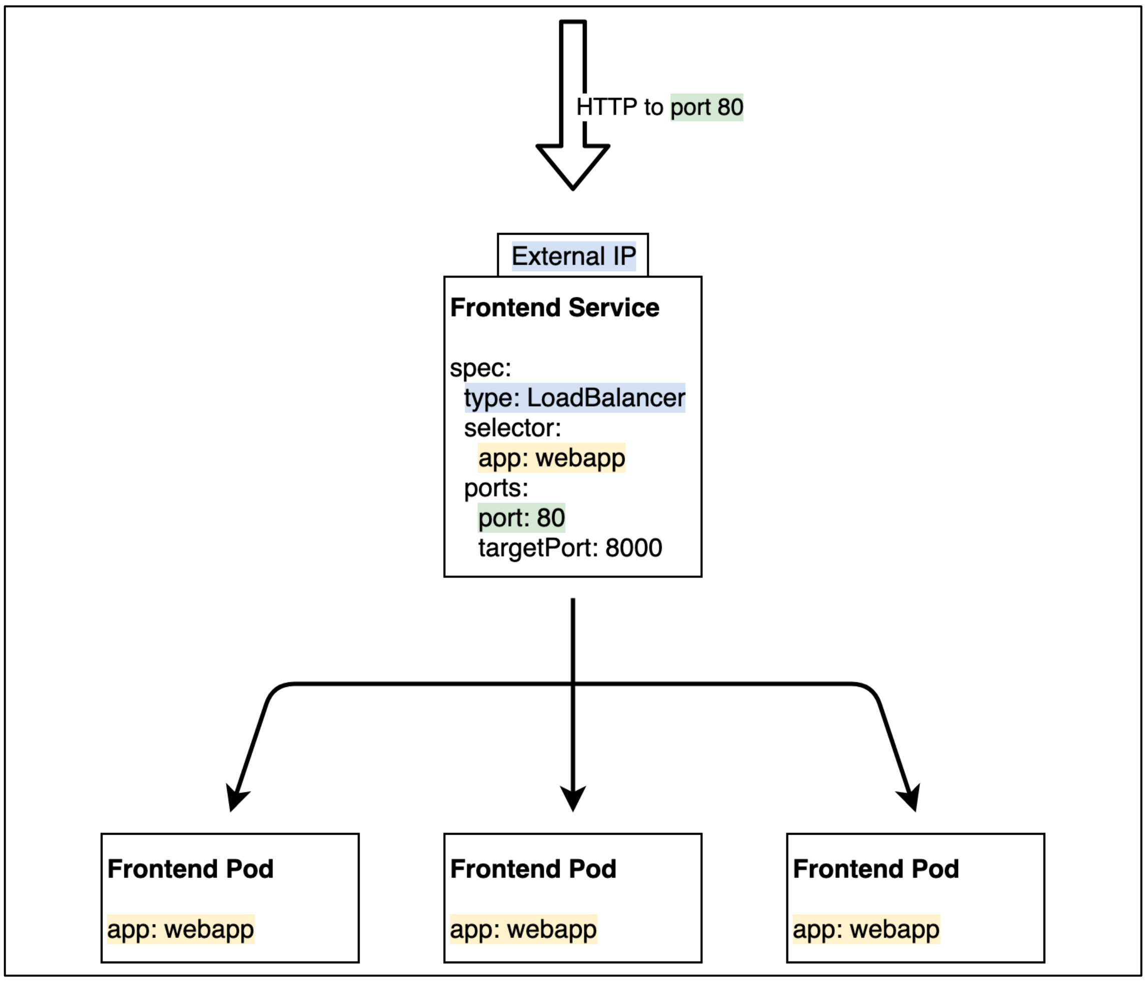


Figure 4. Kubernetes service example.

Figure 4 illustrates an example of a service that is of type LoadBalancer, so it has an external IP address. It has a selector for an app called webapp, so it routes traffic to Pods that have the same app defined in their configuration.

In addition to the IP address records, a Kubernetes cluster defines DNS names for all services and Pods. A normal service that has a cluster IP is assigned a DNS A record for a name in the form "*my-service.my-namespace.svc.cluster-do-*

main.example". DNS names for Pods are defined in the same manner: "*host-name.subdomain.my-namespace.svc.cluster-domain.example*", although Pods are usually accessed via their respected services. (Kubernetes 2019p.)

4.1.7 Storage

On-disk files inside a container are ephemeral, meaning that when the container dies, the files are gone. The Kubernetes Volume abstraction allows all containers in a Pod and all Pods created from a deployment to have the same volume. Kubernetes supports several types of Volumes, many of which are specific to cloud providers, for example, awsElasticBlockStore and azureDisk. (Kubernetes 2019q.)

Another type of volume is configMap, which provides a way to inject configuration data into Pods. This is a fairly common way to configure Pods that run generally available docker images from public repositories, and thus need customization to work properly. (Kubernetes 2019q.)

In addition to the ephemeral volumes, Kubernetes offers *PersistentVolume* subsystem. It is a resource in the cluster, like a node, which has a lifecycle independent of any Pod that uses it. The PersistentVolume API object captures the details of the storage, and it can be NFS, iSCSI or a given cloud provides own storage system. (Kubernetes 2019r.)

For a Pod to be able to access a PersistentVolume resource, it needs a *persistentVolumeClaim* in its volumes block. In other words, Pods do not consume PersistentVolumes as such but access them through claims. If no static PersistentVolumes (that match the given specs) are available, Kubernetes may try to provision a volume dynamically. This action requires a specific *StorageClass* object that specifies the provider, names and binding mode for the dynamic volume. The PersistentVolumeClaim then needs to request this StorageClass in its configuration. (Kubernetes 2019r.)

4.1.8 Infrastructure

Many cloud providers offer hosted Kubernetes clusters. These clusters use cloud providers’ virtual machines or bare metal servers as their nodes and various other resources, e.g. storage, load balancers or security features. The list of available cloud providers is extensive, and it consists of, at least, Amazon Web Services, Microsoft Azure, CloudStack, Google Cloud Platform, Oracle Cloud Infrastructure, OpenStack, OVirt, Photon, VSphere, IBM Cloud and Baidu Cloud. (Kubernetes 2019s.)

In addition to cloud providers’ clusters, Kubernetes can be run on local hardware as well. Using local data center or server rack needs more configuration than a cloud provided cluster. For example, networking needs to be implemented manually. (Kubernetes 2019t.)

For testing and development needs, Kubernetes can be run on a laptop using Minikube. Minikube runs a single-node cluster inside a virtual machine. It supports only some of the features of Kubernetes, namely DNS, NodePorts, ConfigMaps, Secrets, Dashboards, Container Runtime (Docker, CRI-O, containerd), Container Network Interface and Ingress. (Kubernetes 2019u.)

4.2 Elastic Stack

Elastic Stack (also called the ELK stack) is a set of tools from Elastic NV. Elastic NV is a company founded in Amsterdam, Netherlands 2012. Originally called Elasticsearch, the company changed its name in 2015 to distinguish the commercial entity from its opensource project, Elasticsearch (Kepes 2015). Elastic Stack provides a full range of needed components from shipping application or infrastructure metrics, storing that data and visualizing it in a graphical interface. The stack consists of four main parts: Elasticsearch, Kibana, Beats and Logstash. In addition to the core four, Elastic offers other products as well. One of the most notable is Elastic APM (Application Performance Monitoring). (Elastic 2019a.)

Figure 5 illustrates the basic data flow of Elastic Stack in a Kubernetes cluster. Beats, APM and Logstash ship data to Elasticsearch, while Kibana visualizes that data to the user. Kibana and all the arrow-shaped entities in the figure are deployed as Pods. Elasticsearch is a cluster itself, so it needs at least two Pods (a master node and a data node).

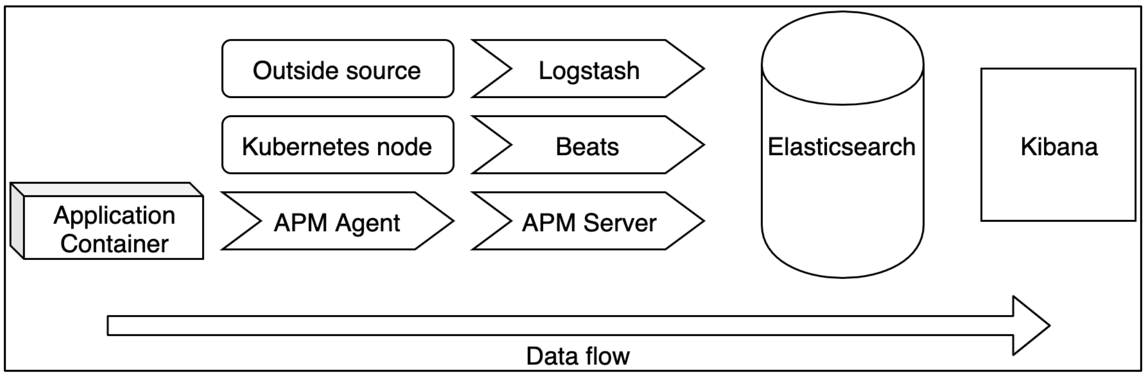


Figure 5. Elastic Stack data flow (simplified)

4.2.1 Elasticsearch

Elasticsearch is a distributed document store. It does not store information as rows of columnar data, rather as serialized JSON documents. A document is indexed and fully searchable within a second of its storing. An index is an optimized collection of documents, and each document is a collection of key-value pairs. Elasticsearch indexes all data in every field and each indexed field has a dedicated, optimized data structure. For example, text fields are stored as inverted indexes and numeric and geo fields as BKD trees. These per-field data structures are what make searching with Elasticsearch so fast. (Elastic 2019b.)

Elasticsearch has the ability to be schema-less, meaning that documents can be indexed without explicitly specifying all fields. This is called dynamic mapping, and when it is enabled, Elasticsearch automatically detects and adds new fields to indexes. All fields of the documents can also be manually mapped to get full control of the indexing. With defined rules and mappings, more sophisticated and faster searches can be conducted. For example, full-text string fields and exact value string fields behave differently and depending on the search type either of these can be preferred over the other. One field can also be indexed in multiple ways, so the same text field can be used for text search and aggregating the results. (Elastic 2019b.)

The document store and its capabilities are just one part of the service. Elasticsearch provides a REST API for managing, indexing and searching data. For testing and development, the API can be accessed from the command line or through the Developer Console in Kibana’s graphical user interface. For application-level, Elasticsearch offers clients for multiple programming languages. Third-party implementations also exist, for lesser-known languages. Search queries themselves can be accessed using Elasticsearch’s own JSON-style query language Query DSL. SQL-style queries can be constructed to search and aggregate data natively inside Elasticsearch, and JDBC and ODBC drivers enable a broad range of third-party applications to interact with Elasticsearch via SQL. (Elastic 2019c.)

Elasticsearch’s search capabilities are built on the Apache Lucene search engine library. Searching supports structured queries, full-text queries and complex queries that combine the two. Structured queries are similar to SQL queries. In other words, searching for specific fields in an index and sorting the matches by another field. Full-text queries find all documents that match the query string and are sorted based on relevance (how well they match the search terms). In addition to individual terms searching, Elasticsearch supports phrase searches, similarity searches, prefix searches and it can give autocomplete suggestions. (Elastic 2019c.)

Data analyzing is possible on Elasticsearch with aggregations. Aggregations enable building complex summaries of the data and gaining insights to key metrics, patterns and trends. With aggregations, averages, medians, time-series data, anomalies and other types of calculations can be returned from the index. Aggregations leverage the same data-structures as search, so they are also very fast. This enables analyzing and visualizing data in real time. Reports and dashboards update as the data changes. Because aggregations operate alongside search requests, searching and analytics happen at the same time, on the same data, in a single request. Elasticsearch also offers automated analysis with the utilization of machine learning. (Elastic 2019c.)

4.2.2 Kibana

Kibana is an analytics and visualization platform, designed to work especially with Elasticsearch. Kibana is used to search, view and interact with data stored in Elasticsearch indices. It offers a browser-based interface to create and share dynamic dashboards. Queries, visualizations and dashboards can be made through the graphical interface, so no coding or learning the Query DSL is required to get real-time access to the data in Elasticsearch. Just as Elasticsearch, Kibana is also open source. (Elastic 2019d.)

In order to visualize and explore data in Kibana, an index pattern must be created. Index patterns are what connect indices in Elasticsearch to Kibana. An index pattern can match a single index or multiple indices. Index patterns support wildcards and the ability to exclude. For example an index pattern “*index*, -index3*” would map together all indexes that begin with “*index*“, but explicitly leave out “*index3*”. When Kibana detects an index with timestamp values, it is required to choose a field to filter the data by time. If no field is specified, time filters cannot be used. (Elastic 2019e.)

From a monitoring perspective, Kibana can be divided into three basic elements: discover, visualize and dashboard. Discover enables data exploring on all indices in Elasticsearch. An index-pattern is selected, search queries can be made, and results filtered with custom filters. The number of matching documents is shown as well as field value statistics (Elastic 2019f). Visualizations are, as the name suggests, visual representations of some aggregation metric or entity (Elastic 2019g). Dashboards, in turn, are collections of visualizations that are grouped together and shown on the same page (Elastic 2019h).

4.2.3 Beats

Beats are open source data shippers that are installed as agents on servers. They send operational data to Elasticsearch. Elastic offers eight different types of Beats:

- Auditbeat, which ships audit data.
- Filebeat, which ships log files.
- Functionbeat, which ships cloud data.
- Heartbeat, which ships availability data.
- Journalbeat, which ships systemd journals.
- Metricbeat, which ships metrics.
- Packetbeat, which ships network traffic.
- Winlogbeat, which ships Windows event logs.

Beats can send data directly to Elasticsearch, or it can be sent via Logstash, where it can be further processed or enhanced before indexing to Elasticsearch. (Elastic 2019i.)

Since Beats is also an open source, it has a plethora of community created Beats as well. The list contains dozens of different projects from amazonbeat that reads data from an Amazon product to elasticbeat that reads status from the Elasticsearch cluster itself. Elastic offers no support or warranty to community provided add-ons. (Elastic 2019j.)

4.2.4 Logstash

Logstash is a data collection engine with real-time pipelining capabilities. It can be used for dynamically unifying data from disparate sources and normalizing it into destinations of choice, usually Elasticsearch. For example, logs from different services of an application might be in very different forms. Logstash can receive this data and filter it into a unified form. It can also be used to relay and refine data from outside sources to the Elastic Stack, like webhooks from GitHub, GitLab, JIRA or other sources. Just like many other Elastic products, Logstash is open source as well. (Elastic 2019k.)

4.2.5 APM

Elastic APM is an application performance monitoring system built on the Elastic Stack. It allows monitoring of software on the application level. With it, response times and spans can be tracked for incoming requests, database queries and more. It helps to pinpoint performance problems and to fix bottlenecks on applications. APM also automatically collects unhandled errors and exceptions. (Elastic 2019l.)

Elastic APM consists of Agents and a Server. APM Agents are open source libraries that are installed into the service like any other source code libraries. These libraries provide a way to collect performance data and errors. Some functionalities are automatic, and some require manual implementation (webserver middleware, wrapping functions). This data is buffered for a short period and sent on to APM Server. APM Server is a separate application that receives the data from agents, validates it, processes it to documents and stores the documents to Elasticsearch indices. (Elastic 2019m.)

In addition to the two main components, Elastic APM is also part of the Kibana UI. It is provided with a basic license and provides a pre-configured way to visualize APM data. Because the APM Server writes the data into a specific index in the Elasticsearch, only an index pattern is required to be created in Kibana. After this, the application appears automatically to the APM UI in Kibana. (Elastic 2019n.)

5 Case Siili

The case example of this thesis is an internal product development project in Siili Solutions Oyj called KnoMe-redux. It is an application for handling human resources in the company. It contains personal knowledge data of the people working at Siili, as well as data about Siili's customers, including projects made for these customers. An employee's contact info, skills, certificates, education and projects are shown in the employee's profile pages, while customers and projects have their own pages. These entities are linked together to create a network of people, skills and references. All entities can also be tagged to combine data further.

KnoMe-redux is built to replace an older system called KnoMe. Because KnoMe has its data stored in an unstructured document database, KnoMe-redux's relational data-model will unify entities and significantly improve user experience. Alongside the stricter data, KnoMe-redux also provides better maintenance and further development capabilities. Kubernetes and utilization of containers is a big part of this because independent parts of the system can be replaced or updated individually.

5.1 Architecture

KnoMe-redux is a distributed micro-service application. The processing logic is shattered to individual modules, and a Docker container is built to represent each module. Almost all containers are stateless so that they can be scaled up or down at any time. Exceptions are Elasticsearch and Redis containers, which are in-memory data stores and thus stateful by nature.

The backend of KnoMe-redux consists of eleven micro-services. Nine of these are custom written with Clojure, using embedded Jetty servers for handling HTTP-traffic. These Jetty servers have Ring wrappers, which abstract the details of HTTP into a simple and unified API (McGranaghan, Reeves & contributors

2018). Two other backend services are the before mentioned Elasticsearch and Redis, which are used as-is from Dockerhub repository. In addition to these eleven services, the KnoMe-redux cluster also includes an authorizing service. This is a third-party service called Authorizer, and it consists of an application container and an SQL database container. Authorizer is responsible for oauth2 login flow, ADFS integration and handling of user roles and groups.

The frontend of KnoMe-redux is a container that runs an NGINX based container with a Vue.js web application inside. NGINX is a gateway for incoming connections and serves the Vue.js user interface that is accessible with a web browser. NGINX works as a reverse-proxy for the frontend, enabling calls to be made to its router and it is then responsible for proxying calls to correct endpoints in the backend.

KnoMe-redux is deployed into an Amazon EKS cluster. EKS stands for Amazon Elastic Kubernetes Service. Despite the similar name, it has nothing to do with Elastic, co. Amazon EKS runs the Kubernetes management infrastructure across multiple AWS availability zones to eliminate a single point of failure (Amazon Web Services 2019b). In addition to the services running in Kubernetes, KnoMe-redux uses Amazon DynamoDB document database for storing data, AWS Lambda serverless functions with Simple Notification Service queues for system to system integrations and Amazon S3 object storage for storing logs, user pictures and other assets.

5.2 Implementing Elastic Stack

As KnoMe-redux is running in Kubernetes, the Elastic Stack needs to be configured accordingly. All services are running in containers in the cluster. From the stack, Elasticsearch, Kibana, Metricbeat, Filebeat, Logstash and APM are selected. Since Elasticsearch is a scalable cluster, a master-node is required along with a number of data-nodes. For this monitoring implementation, a single Elasticsearch data-node is used for storing all data. Data from different sources is stored to different indices in Elasticsearch.

Because of distributed servers and container orchestration, a Kubernetes application should be monitored on three separate levels: infrastructure, service and application. In addition to these, also the cluster itself can be monitored. (Datadog 2019.)

Infrastructure level monitoring means tracking metrics and statuses of the host servers. In practice, it means CPU and memory utilization and network traffic of the entire worker nodes. It is essential to know what type of loads the servers are running and how many Pods they can run. It is not cost-effective to schedule five worker nodes if an application only needs three, for example. Implementation wise, kube-state-metrics is a service that gathers this data and pushes it to an Elasticsearch index. Since host metrics can be read from outside the actual server, only one instance of kube-state-metrics is required in the cluster.

Even if infrastructure level monitoring would show that nodes are using significant resources, no further conclusions can be made from it. Service level monitoring is required to identify which Pods are using the resources of the node they are running on. Service level data can be acquired with Beats. In this monitoring implementation, Metricbeat and Filebeat are used. Filebeat ships logs from the stdout (standard output) of containers and Metricbeat ships resource usage statistics of each individual Pod. While kube-state-metrics only needs to be run as one instance in the whole cluster, Beats need to be present on all worker nodes to be able to access all Pods.

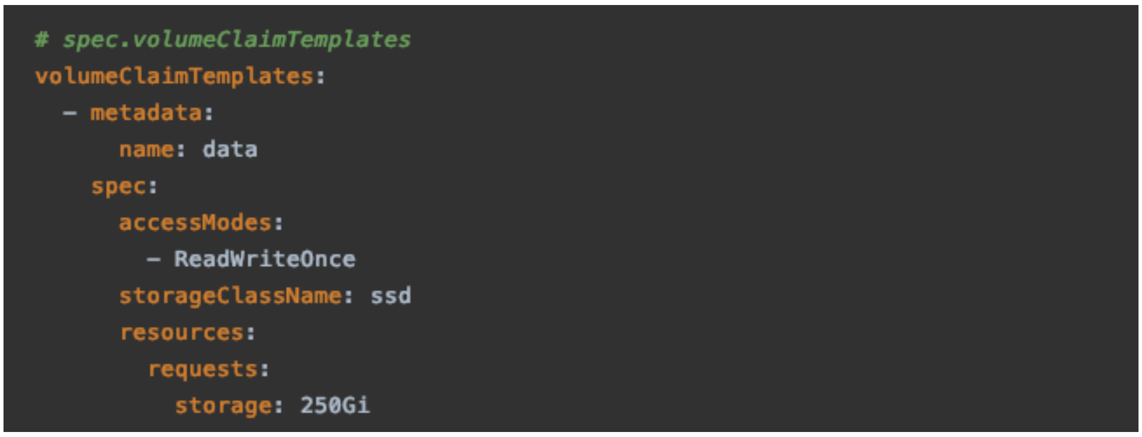
Knowing which Pods use what number of resources is good, but in order to understand why, monitoring needs to reach into the applications running inside the containers. For this, Elastic APM is used. Unlike infrastructure and service level monitoring, properly implementing Application Performance Monitoring means adding an agent and proper functionality to the source code. These agents then send the data they have gathered to an APM server, which is deployed to the Kubernetes cluster. The server, in turn, sends the data on to Elasticsearch.

Getting all this data becomes valuable only when it can be accessed and visualized. For this, Kibana is deployed. It gives access to Elasticsearch native queries and visualizing tools with a browser-based interface for creating informative dashboards. To secure access to this information, an oauth2 proxy is deployed and configured to use GitLab as a login provider with a list of allowed user accounts. Kibana's loadbalancer is then configured to route traffic to Kibana through this proxy.

Resources for the stack are deployed to two different namespaces in the cluster, depending on the type of resource. Elasticsearch, Kibana, Logstash and APM server are deployed to knomeredux-elk namespace, which is a custom namespace only for these resources. Kube-state-metrics and Beats are deployed to kube-system namespace because they need access to Kubernetes internal resources that recede there.

5.2.1 Elasticsearch

Elasticsearch is in the heart of this monitoring implementation. As stated before, Elasticsearch is a cluster and to configure it properly in Kubernetes infrastructure, at least one master Pod and one data Pod are required. All Elasticsearch resources are deployed to knomeredux-elk namespace. The master node is deployed with a Deployment controller, and it has a selector with app: elasticsearch and role: master. Data node is then configured to have a selector with app: elasticsearch and role: data. To keep the data node's storage persistent across Pod rescheduling, it is deployed as a StatefulSet, and a volumeClaimTemplate is added to it. Two hundred fifty gigabytes are requested for storing logs and other monitoring data (figure 6). Both the master and the data node are deployed with a default service object, configured with same selectors as the Pods. The services are given a clusterIP: None specification to keep them headless. This headless implementation combined with selectors modifies the DNS configuration to return records (addresses) that point directly to the Pods backing the services (Kubernetes 2019o).



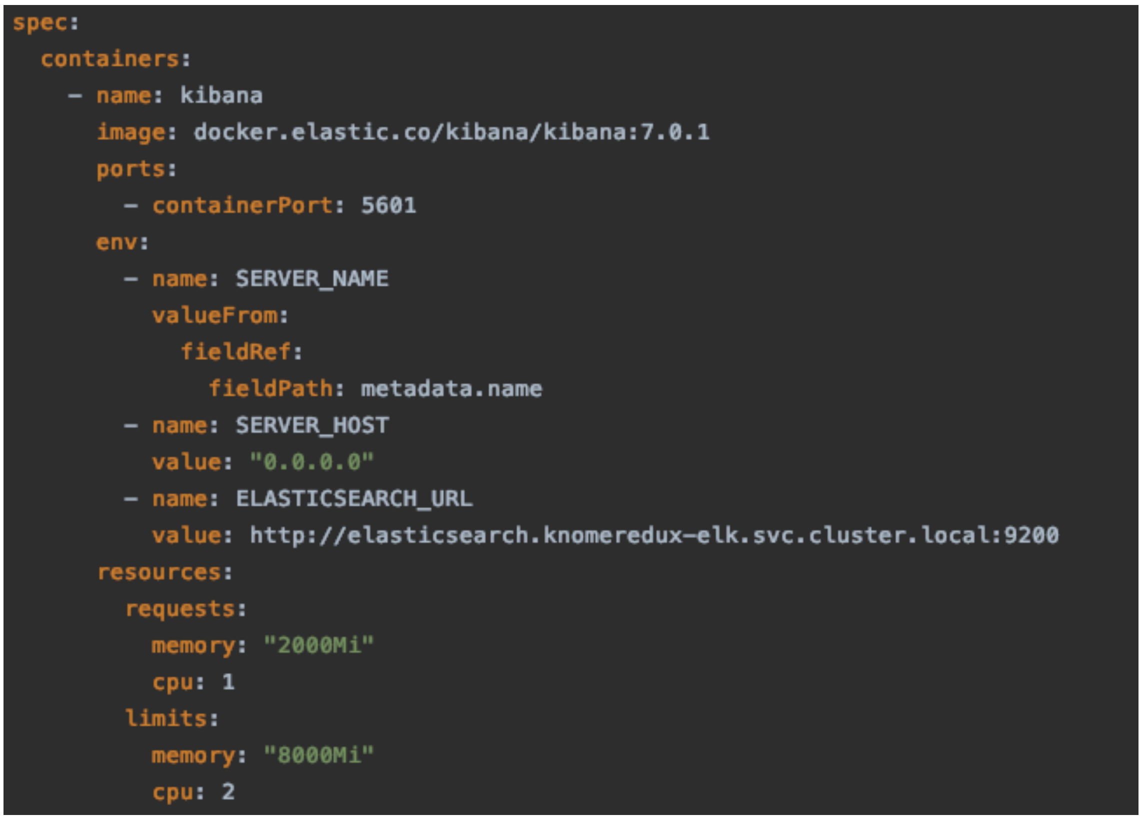
```
# spec.volumeClaimTemplates
volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes:
        - ReadWriteOnce
      storageClassName: ssd
      resources:
        requests:
          storage: 250Gi
```

Figure 6 Elasticsearch’s volumeClaim

Elasticsearch is extended with a CronJob. This CronJob uses a Docker container image called Curator. Curator is a tool for managing time-based indices in Elasticsearch. Curator is configured with five actions, one of which is a snapshot action and four are deleting actions. The snapshot action sends backup data to Amazon S3 bucket from all indices that are not logstash- prefixed and are older than one day. The four deleting actions delete metricbeat-, filebeat-, and amp- prefixed indices that are older than seven days. Logstash indices are deleted as late as 31 days old because Logstash is configured to receive data only from GitLab webhooks and it does not pile up as fast as the metrics data. Visualizations for ci/cd pipeline and the repository are also relevant on a longer time period.

5.2.2 Kibana

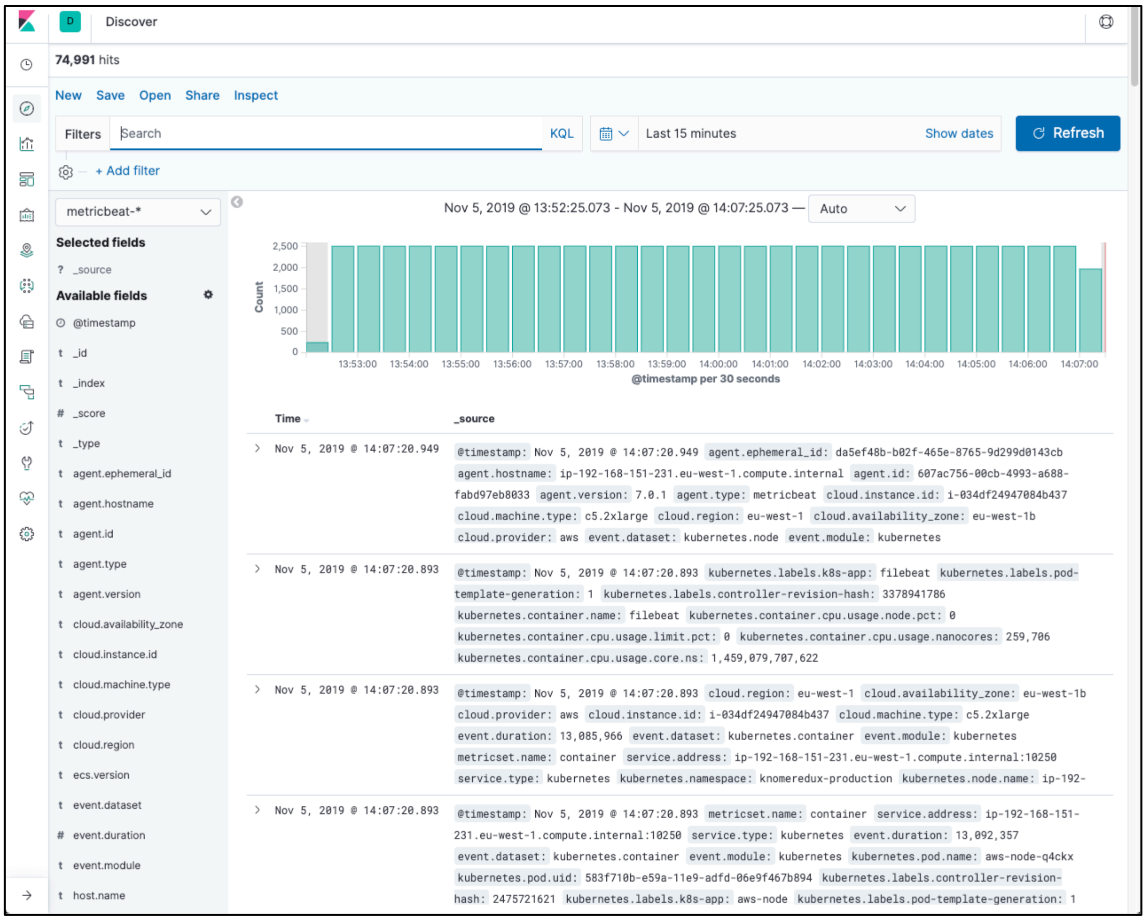
As stated before, Kibana is accessed through a web browser. To secure the information gathered by Elastic Stack, an oauth2 proxy is deployed to give access only to pre-defined users. Kibana itself is deployed with a Deployment controller and a default Service. The container spec of the controller can be seen in figure 7. The setup also has a LoadBalancer type Service. This service routes traffic from the internet to the oauth2 Pod, which (on successful login) redirects to the Kibana Service.



```
spec:
  containers:
    - name: kibana
      image: docker.elastic.co/kibana/kibana:7.0.1
      ports:
        - containerPort: 5601
      env:
        - name: SERVER_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: SERVER_HOST
          value: "0.0.0.0"
        - name: ELASTICSEARCH_URL
          value: http://elasticsearch.knomeredux-elk.svc.cluster.local:9200
      resources:
        requests:
          memory: "2000Mi"
          cpu: 1
        limits:
          memory: "8000Mi"
          cpu: 2
```

Figure 7 Container spec of Kibana’s Deployment

Kibana Pods need the URL of Elasticsearch as an environment variable to access the gathered data. The Kubernetes DNS name of Elasticsearch’s service object is used. Otherwise, Kibana is used as it comes from the Dockerhub image repository. Needed visualizations and dashboards can be constructed manually from the UI or uploaded with an HTTP post query. It is advised to export these from Kibana to keep them backed up. Figure 8 shows Kibana’s Discover screen.



```
Time                         _source
Nov 5, 2019 @ 14:07:20.949   @timestamp: Nov 5, 2019 @ 14:07:20.949 agent.ephemeral_id: da5ef48b-b02f-465e-8765-9d299d0143cb
                             agent.hostname: ip-192-168-151-231.eu-west-1.compute.internal agent.id: 607ac756-00cb-4993-a688-
                             fabd97eb8033 agent.version: 7.0.1 agent.type: metricbeat cloud.instance.id: i-034df24947084b437
                             cloud.machine.type: c5.2xlarge cloud.region: eu-west-1 cloud.availability_zone: eu-west-1b
                             cloud.provider: aws event.dataset: kubernetes.node event.module: kubernetes
Nov 5, 2019 @ 14:07:20.893   @timestamp: Nov 5, 2019 @ 14:07:20.893 kubernetes.labels.k8s-app: filebeat kubernetes.labels.pod-
                             template-generation: 1 kubernetes.labels.controller-revision-hash: 3378941786
                             kubernetes.container.name: filebeat kubernetes.container.cpu.usage.node.pct: 0
                             kubernetes.container.cpu.usage.limit.pct: 0 kubernetes.container.cpu.usage.nanocores: 259,706
                             kubernetes.container.cpu.usage.core.ns: 1,459,079,707,622
Nov 5, 2019 @ 14:07:20.893   @timestamp: Nov 5, 2019 @ 14:07:20.893 cloud.region: eu-west-1 cloud.availability_zone: eu-west-1b
                             cloud.provider: aws cloud.instance.id: i-034df24947084b437 cloud.machine.type: c5.2xlarge
                             event.duration: 13,085,966 event.dataset: kubernetes.container event.module: kubernetes
                             metricset.name: container service.address: ip-192-168-151-231.eu-west-1.compute.internal:10250
                             service.type: kubernetes kubernetes.namespace: knomeredux-production kubernetes.node.name: ip-192-
Nov 5, 2019 @ 14:07:20.893   @timestamp: Nov 5, 2019 @ 14:07:20.893 metricset.name: container service.address: ip-192-168-151-
                             231.eu-west-1.compute.internal:10250 service.type: kubernetes event.duration: 13,092,357
                             event.dataset: kubernetes.container event.module: kubernetes kubernetes.pod.name: aws-node-q4ckx
                             kubernetes.pod.uid: 583f710b-e59a-11e9-adfd-06e9f467b894 kubernetes.labels.controller-revision-
                             hash: 2475721621 kubernetes.labels.k8s-app: aws-node kubernetes.labels.pod-template-generation: 1
```

Figure 8 A screenshot of Discover in Kibana

5.2.3 Kube-state-metrics

Kube-state-metrics is not a part of the Elastic stack, but it works seamlessly together with it. It acquires metrics data on the infrastructure level, for example, state and resource usage of nodes. It also sees the state of Pods, but nothing more specific about them. While kube-state-metrics is capable of scraping data from Kubernetes' endpoints, it needs a Metricbeat instance to ship the data onwards. A default Dashboard template of Kubernetes overview can be added to Kibana, which illustrates state metrics data (figure 9).

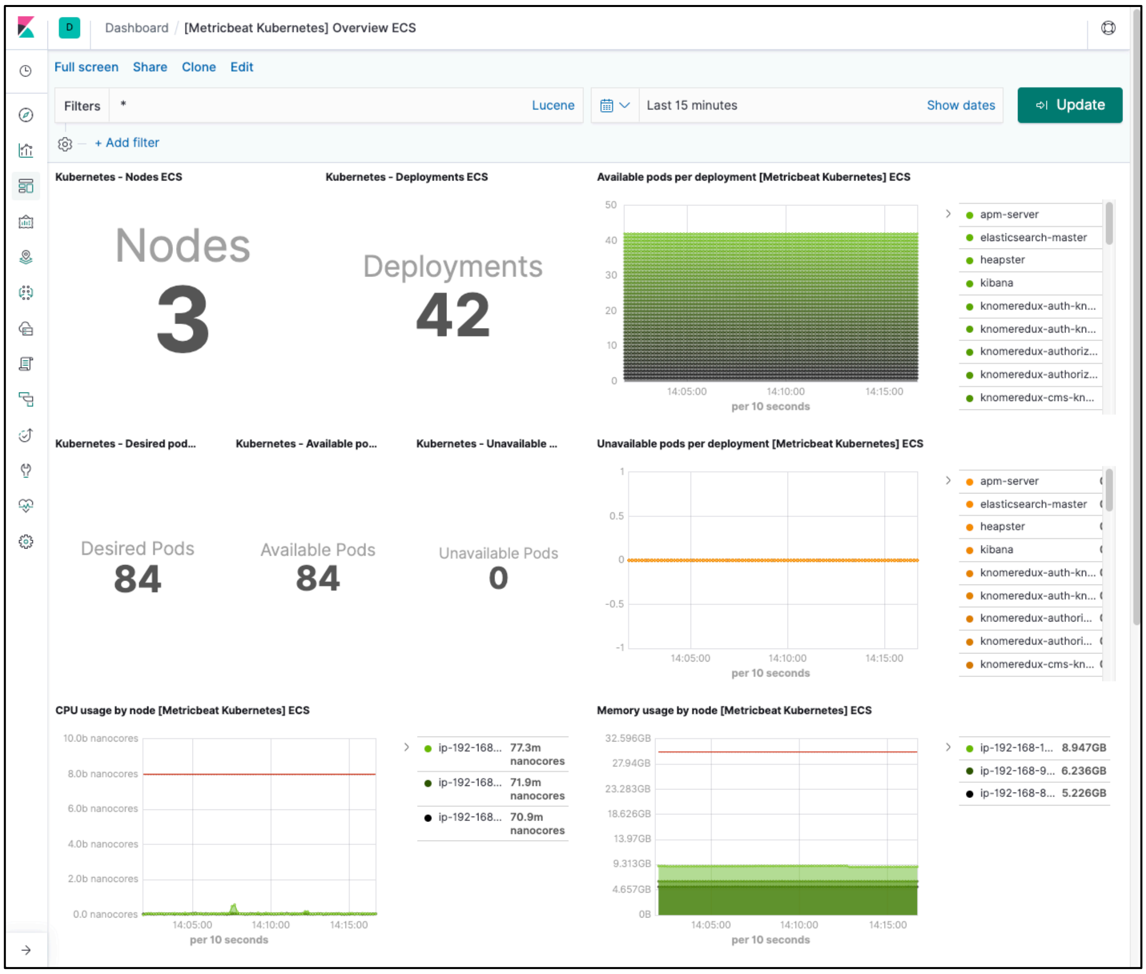
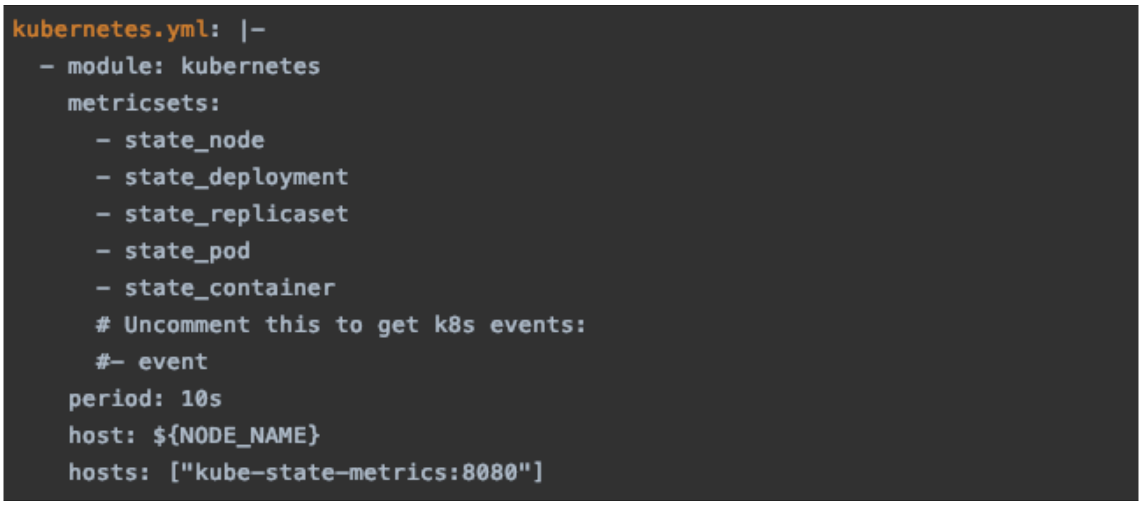

Dashboard / [Metricbeat Kubernetes] Overview ECS
Full screen
Share
Clone
Edit
Filters
Lucene
Last 15 minutes
Show dates
Update
+ Add filter
Kubernetes - Nodes ECS
Nodes
3
Kubernetes - Deployments ECS
Deployments
42
Available pods per deployment [Metricbeat Kubernetes] ECS
per 10 seconds
apm-server
elasticsearch-master
heapster
kibana
knomeredux-auth-kn...
knomeredux-authoriz...
knomeredux-cms-kn...
Kubernetes - Desired pod...
Kubernetes - Available po...
Kubernetes - Unavailable ...
Desired Pods
84
Available Pods
84
Unavailable Pods
0
Unavailable pods per deployment [Metricbeat Kubernetes] ECS
CPU usage by node [Metricbeat Kubernetes] ECS
ip-192-168... 77.3m nanocores
ip-192-168... 71.9m nanocores
ip-192-168... 70.9m nanocores
Memory usage by node [Metricbeat Kubernetes] ECS
ip-192-168-1... 8.947GB
ip-192-168-9... 6.236GB
ip-192-168-8... 5.226GB

Figure 9 Kubernetes Overview Dashboard in Kibana (from template)

Kube-state-metrics is deployed with a Deployment controller and a default clusterIP service object to the kube-system namespace. It needs heightened access inside the cluster to access needed data so custom role and cluster role are configured. To get this data properly to Elasticsearch, it needs a singleton instance (one Pod in any of the nodes) of Metricbeat in the cluster to receive and relay the data. This Metricbeat instance is configured independently of the other Metricbeat Pods that ship service level data. It has a configuration that is loaded as a volume to the deployment. The configuration, shown in figure 10, defines what metrics are collected on what frequency.



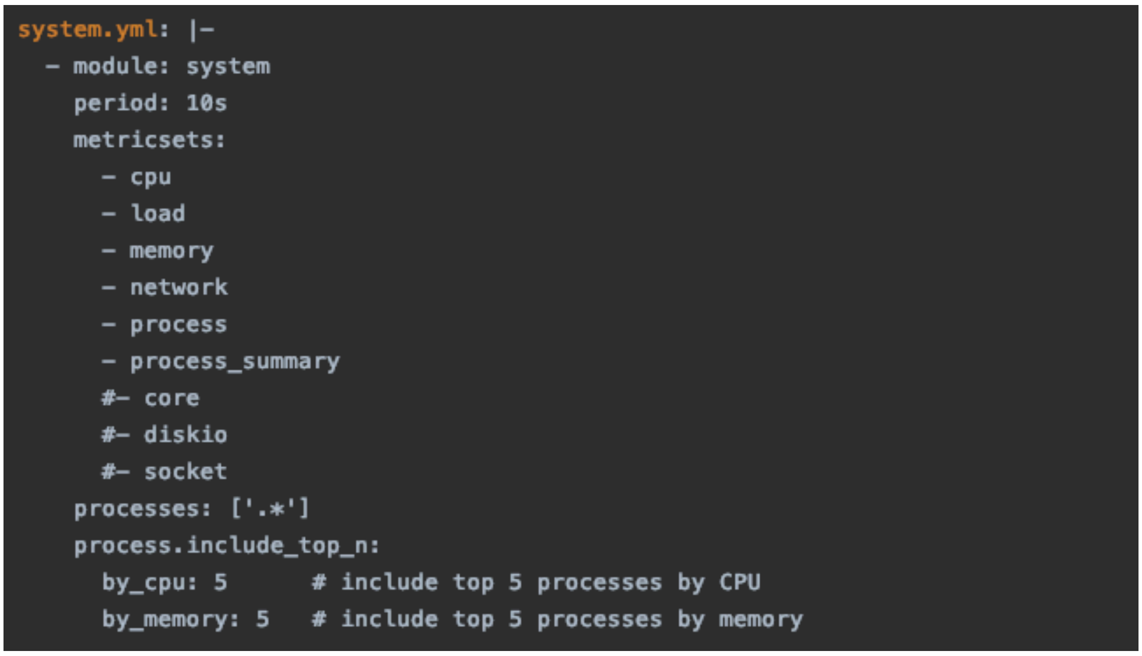
```
kubernetes.yml: |-
  - module: kubernetes
    metricsets:
      - state_node
      - state_deployment
      - state_replicaset
      - state_pod
      - state_container
      # Uncomment this to get k8s events:
      #- event
    period: 10s
    host: ${NODE_NAME}
    hosts: ["kube-state-metrics:8080"]
```

Figure 10 Kube-state-metrics' module configuration

5.2.4 Beats

Beats implementation includes Metricbeat and Filebeat. Both are deployed with DaemonSet controllers so that each node gets one Pod for shipping data. They also have custom ClusterRoles to gain needed access inside the cluster. Metricbeat needs a generic configuration file to describe an output endpoint of Elasticsearch and how to load modules into it. The output endpoint is the DNS name of the Elasticsearch service. This configuration is loaded into the Pods as a volume called config. The modules themselves are defined in two configurations which are called system.yml and kubernetes.yml, and they are loaded into the Pods as a single volume called modules.

Kubernetes.yml configures what metricsets are gathered, how often they are shipped, and how the host machine is discovered. The selected metrics are node, Pod, container, volume and system. Host machine means the worker node where each Pod is assigned to. It also has the optional ssl verification settings, which are disabled since the stack is running in a private virtual cloud network inside a locked down cluster network. One of the defined metricsets is system, and system.yml (figure 11) configures this in a bit more detail. It defines that from the system module CPU, load, memory, network, process and process summary metrics are gathered and shipped. It is possible to filter what processes are monitored, but in this implementation a wildcard is used, so no filter is applied.



```
system.yml: |-
  - module: system
    period: 10s
    metricsets:
      - cpu
      - load
      - memory
      - network
      - process
      - process_summary
      #- core
      #- diskio
      #- socket
    processes: ['.*']
    process.include_top_n:
      by_cpu: 5      # include top 5 processes by CPU
      by_memory: 5   # include top 5 processes by memory
```

Figure 11 Part of Metricbeat's system.yml configuration

Filebeat needs a configuration file as well. Like in Metricbeat, it is loaded to the Pods as a volume called config. For Filebeat, the config defines inputs of the system logs it is gathering and an Elasticsearch output. The inputs are then defined in a configuration called kubernetes.yml that is loaded as a volume called inputs. This implementation has only one type of input, docker. A wildcard query is used for container ids to get logs from all containers.

Figure 12 shows how Metricbeat's data is visualized in a Kibana Dashboard. This Dashboard and all its visualizations are custom made for KnoMe-redux. It shows the CPU and memory usage for each Pod. Namespace can be selected from a dropdown menu since different stages are deployed into different namespaces. Figure 13 shows a zoomed view of two Pods' resource usage.

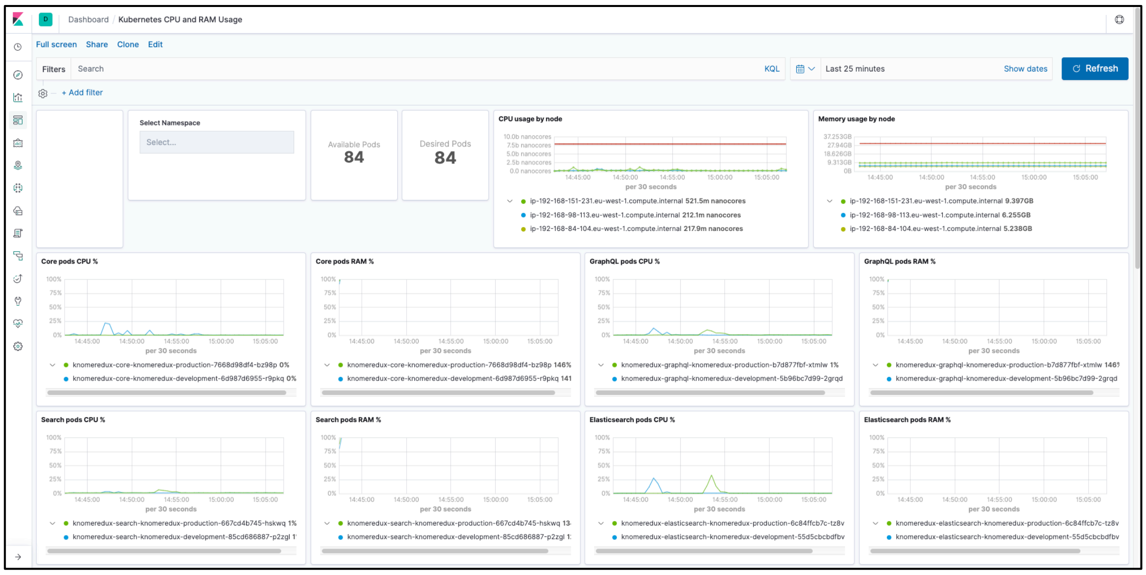


Figure 12 Kubernetes CPU and RAM Usage Dashboard in Kibana

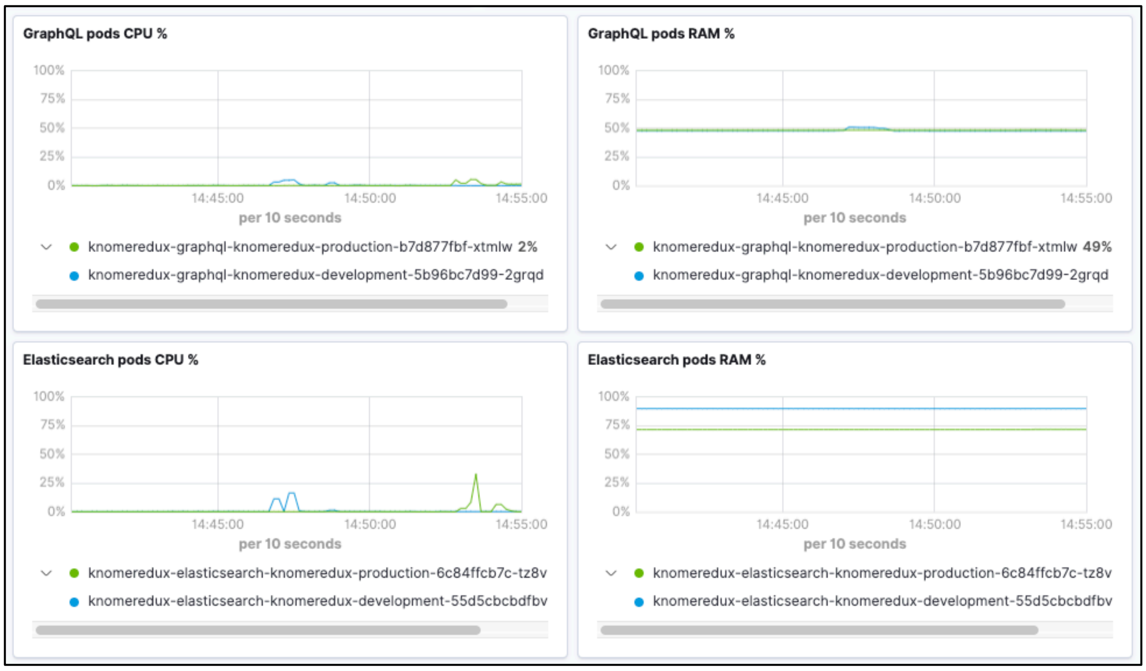
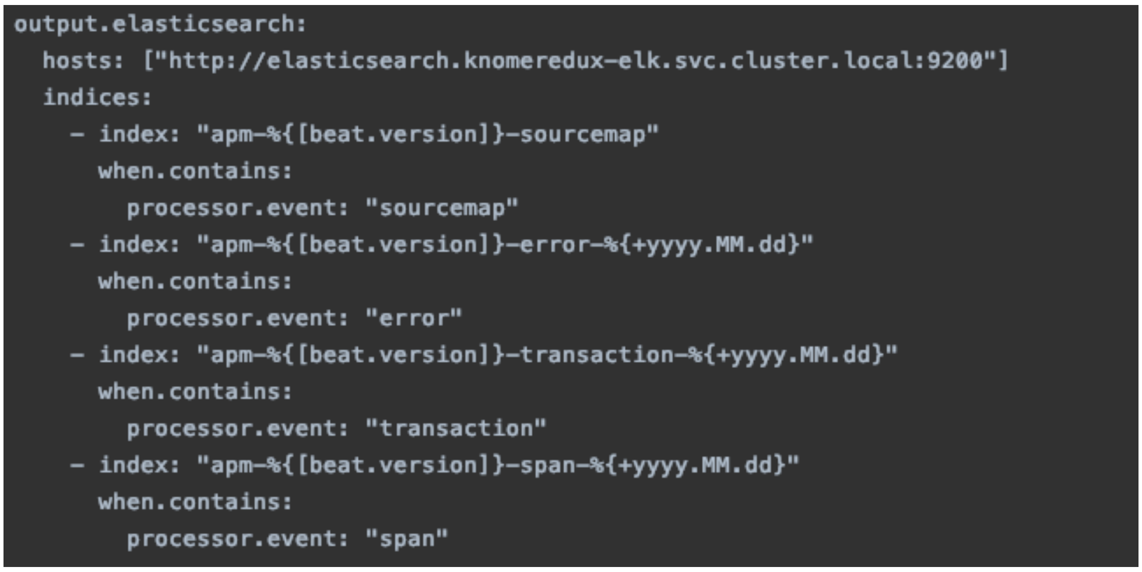


Figure 13 Closeup on Pods' resource usage

5.2.5 APM

Application Performance Monitoring is implemented in a bit different fashion than regular Beats shippers. It consists of a number of agents that ship data from inside the application containers and one server that relays this data to specified Elasticsearch indices.

APM-server is deployed to the Kubernetes cluster with Deployment controller and a clusterIP Service. It also needs some finer definitions, and these are done with apm-server.yml configuration. In it, the server host is defined along with the number of shards to use for the agent data and what codec to use for refining it. Output is also configured for Elasticsearch (figure 14). In addition to the host's address, it needs the indices to be specified for each type of APM-agent message.

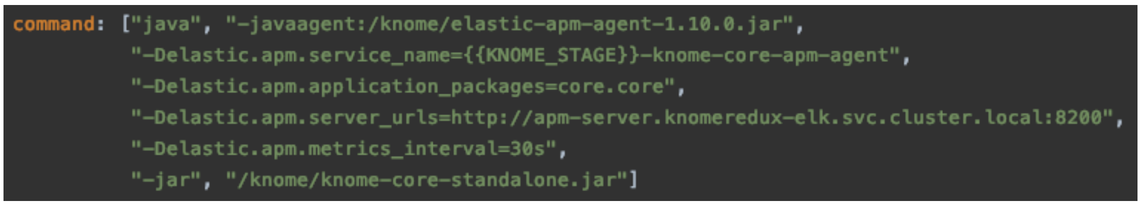


```
output.elasticsearch:
  hosts: ["http://elasticsearch.knomeredux-elk.svc.cluster.local:9200"]
  indices:
    - index: "apm-%{[beat.version]}-sourcemap"
      when.contains:
        processor.event: "sourcemap"
    - index: "apm-%{[beat.version]}-error-%{+yyyy.MM.dd}"
      when.contains:
        processor.event: "error"
    - index: "apm-%{[beat.version]}-transaction-%{+yyyy.MM.dd}"
      when.contains:
        processor.event: "transaction"
    - index: "apm-%{[beat.version]}-span-%{+yyyy.MM.dd}"
      when.contains:
        processor.event: "span"
```

Figure 14	APM-server's output configuration

The APM-agent is configured on the application level, so the steps are very different from other data shippers. Since the backend is composed of containers written with Clojure, two dependencies are added to the project.clj file of each micro-service: *apm-agent-api* and *clojure-elastic-apm*. The before mentioned is made by Elastic, co. and provides the basic functionalities for the APM-agent. The latter is an open-source library made under Yleisradio's GitHub account. To have the executables for the APM-agent, they need to be installed on the Docker containers that are running the code. Since Clojure runs on top of JVM (Java Virtual Machine), a java agent executable can be used. This is done by adding a wget (a Linux command line program) script to the Dockerfile that downloads the agent as a *jar* file. Jar files are compressed Java applets with their requisite components (Oracle 2019).

In addition to downloading the agents and adding libraries to the codebase, the actual container needs to be started with a specific command to configure the agent correctly. This Java agent could be configured with environment variables as well, but since the containers in KnoMe-redux are running embedded Jetty servers, it needs to be started with a specific startup command. For example, a Deployment called core – which has its own code bundled to a jar file called *knome-core-standalone.jar* – uses the command in figure 15. The command defines what agent to use and gives that agent some specifications. The *service name* is given with a parameter, which is shown in the command inside double braces.

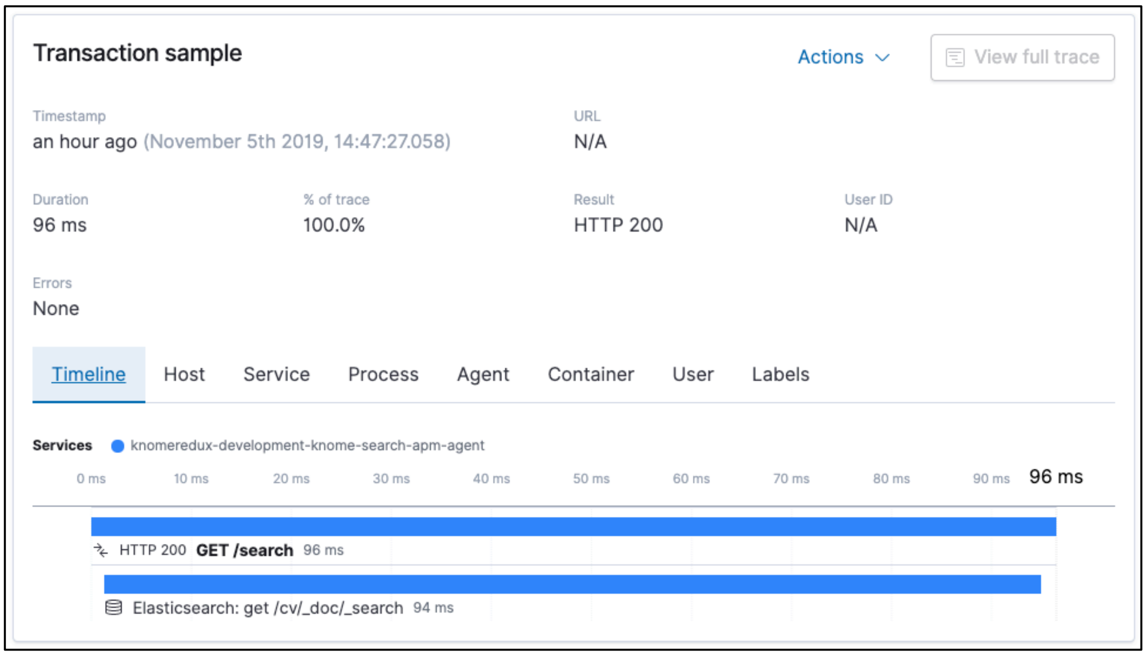


```
command: ["java", "-javaagent:/knome/elastic-apm-agent-1.10.0.jar",
          "-Delastic.apm.service_name={{KNOME_STAGE}}-knome-core-apm-agent",
          "-Delastic.apm.application_packages=core.core",
          "-Delastic.apm.server_urls=http://apm-server.knomeredux-elk.svc.cluster.local:8200",
          "-Delastic.apm.metrics_interval=30s",
          "-jar", "/knome/knome-core-standalone.jar"]
```

Figure 15. Container startup command in Deployment configuration.

One trackable metric for Elastic APM is *transaction span*. Transaction span in this context means the time it takes for a given part of the application to run. For example, one function can be wrapped to apm-transaction to track its span. Using it with a Ring middleware tracks the time a container takes to process the incoming HTTP request and send a response. To have this metric, *wrap-apm-transaction* function is used. It is part of the clojure-elastic-apm dependency library. As a middleware function, it is part of the Ring app and therefore all incoming HTTP queries launch the span tracking.

Figure 16 shows how Kibana's APM shows one transaction span. It is a GET request to */search* endpoint in a microservice called *Search*. Search offers a REST API for GraphQL and constructs Elasticsearch queries to the actual Elasticsearch container based on the HTTP requests it receives from GraphQL. The /search endpoint does searches to an index that holds users' complete CVs, based on given parameters. The transaction in figure 16 has taken 96 milliseconds, of which Elasticsearch has used 94 milliseconds. Kibana has APM functionalities preinstalled, so no visualizations or dashboards are needed for accessing APM data.

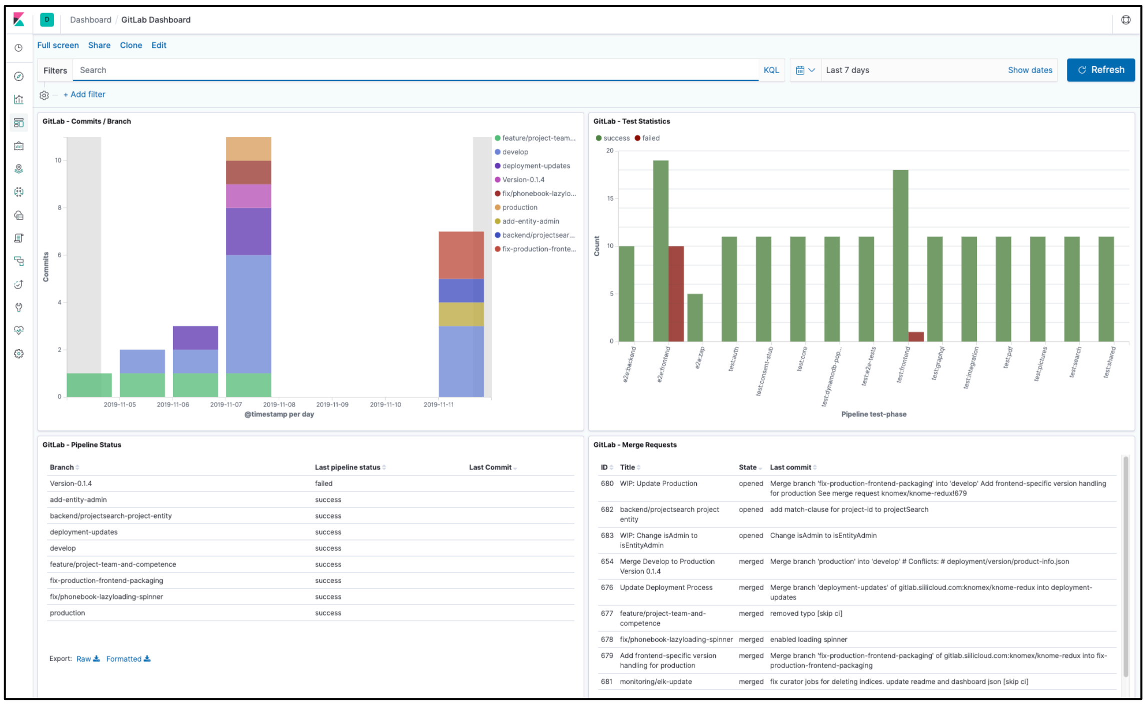

Transaction sample

Timestamp: an hour ago (November 5th 2019, 14:47:27.058)
URL: N/A
Duration: 96 ms
% of trace: 100.0%
Result: HTTP 200
User ID: N/A
Errors: None

Figure 16 APM transaction span in Kibana

5.2.6 Logstash

Logstash is a part of the Elastic Stack, and in this implementation, it is configured to allow an integration to the GitLab ci/cd pipeline of KnoMe-redux. GitLab has a webhook-based integration method, in which selected changes or actions in GitLab are sent out with HTTP requests. In KnoMe-redux, these HTTP messages go to a Lambda function in AWS. This Lambda function checks the source and integrity of the message and then sends these messages on to the LoadBalancer endpoint of Logstash inside the Kubernetes cluster. This endpoint is secured with an SSL certificate, and because GitLab does not have access to KnoMe-redux AWS resources like secrets and certificates, the Lambda function is needed in the middle. Figure 17 shows how data that comes from GitLab via Logstash is visualized in Kibana.



GitLab - Pipeline Status

Branch	Last pipeline status	Last Commit
Version-0.1.4	failed	
add-entity-admin	success	
backend/projectsearch-project-entity	success	
deployment-updates	success	
develop	success	
feature/project-team-and-competence	success	
fix-production-frontend-packaging	success	
fix/phonebook-lazyloading-spinner	success	
production	success	

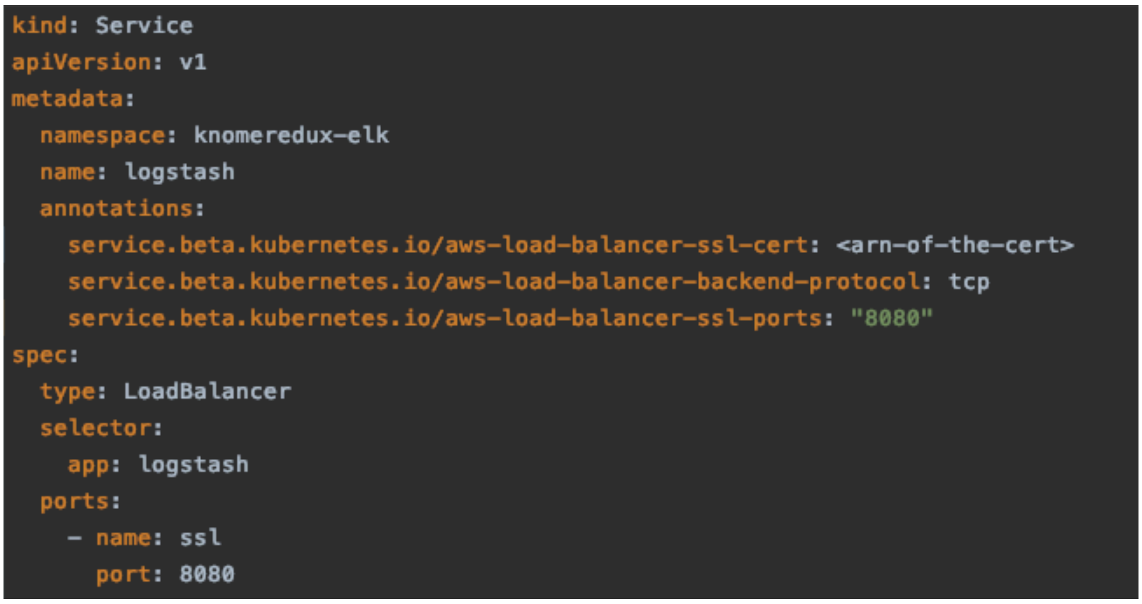
Export: Raw Formatted

GitLab - Merge Requests

ID	Title	State	Last commit
680	WIP: Update Production	opened	Merge branch 'fix-production-frontend-packaging' into 'develop' Add frontend-specific version handling for production See merge request knomex/knome-redux!679
682	backend/projectsearch project entity	opened	add match-clause for project-id to projectSearch
683	WIP: Change isAdmin to isEntityAdmin	opened	Change isAdmin to isEntityAdmin
654	Merge Develop to Production Version 0.1.4	merged	Merge branch 'production' into 'develop' # Conflicts: # deployment/version/product-info.json
676	Update Deployment Process	merged	Merge branch 'deployment-updates' of gitlab.siilicloud.com:knomex/knome-redux into deployment-updates
677	feature/project-team-and-competence	merged	removed typo [skip ci]
678	fix/phonebook-lazyloading-spinner	merged	enabled loading spinner
679	Add frontend-specific version handling for production	merged	Merge branch 'fix-production-frontend-packaging' of gitlab.siilicloud.com:knomex/knome-redux into fix-production-frontend-packaging
681	monitoring/elk-update	merged	fix curator jobs for deleting indices. update readme and dashboard json [skip ci]

Figure 17 GitLab dashboard in Kibana

The Logstash Kubernetes resources are deployed with a Deployment controller and a LoadBalancer service (figure 18). The Deployment is also configured with a configMap, which defines HTTP port 8080 as the input and the cluster-specific DNS address of Elasticsearch as the output. Index is also specified to have a static prefix with the current date.



```yaml
kind: Service
apiVersion: v1
metadata:
  namespace: knomeredux-elk
  name: logstash
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-ssl-cert: <arn-of-the-cert>
    service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
    service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "8080"
spec:
  type: LoadBalancer
  selector:
    app: logstash
  ports:
    - name: ssl
      port: 8080
```

Figure 18 Logstash service configuration (SSL certificate’s identity hidden)

5.3 Pros and cons

Implementing monitoring for a relatively complex application deployment like Kubernetes is not a simple task. Monitoring needs to reach infrastructure, service and application levels, and the extracted data needs to be aggregated and visualized in a way that provides value for the developers and customers alike. Value for the developers means giving visibility to performance bottlenecks at application level and resource demands at service level. Having factual data of the performance is needed for reaching fast and reliable performance, which is expected of modern applications. Value for customers translates, in the end, to money saved. Having efficient software that is running on appropriate hardware is cost-effective.

Elastic Stack, in particular, is not the most straightforward way of accomplishing monitoring. The stack is quite massive, and it needs a lot of configuring. There are other solutions available for monitoring specific levels or parts of a deployment that are easier to set up. Some of them work “out of the box” or need just a few startup parameters to work. However, Elastic Stack reigns above these simple solutions in unifying all data under one user interface. All parts of the stack run around Elasticsearch as a data storage and Kibana provides a native and intuitive way to visualize and learn from that data. Elastic Stack is also open source, so it does not cost anything to use. Ultimately, going through the trouble of configuring and deploying the stack is paid back in the extensive value that accessing all information from one place gives to all user groups.

6 Conclusion

This thesis was based on a business need of its client, Siili Solutions Oyj. It researches the theories of Kubernetes and Elastic Stack and provides a case example of implementing the stack on an existing Kubernetes cluster. Two objectives were given, and both of them were achieved to some degree.

The secondary objective of this thesis was to create new research about monitoring Kubernetes. This was reached quite well. This thesis is very solution heavy, and it has no comparative or exploratory aspects of different monitoring products. It does, however, give a detailed view of one option and provides a good theoretical base of Kubernetes, on which to build on. Future research could give similar perspectives on some other monitoring solution or make comparisons on multiple solutions.

The primary objective was to provide a working implementation example for the client and document the theory and configuration of it. This objective was fulfilled well since the technical solution is in use in KnoMe-redux. Reusing the configuration has not been tested since no other suitable projects have yet emerged. The theoretical part of this thesis can be used for achieving knowledge about Kubernetes in the client company, while the case example provides a way for minimizing the inconvenience of setting up a monitoring stack and going straight to producing value.

List of References

Amazon Web Services. 2019a. Amazon Elastic Kubernetes Service. Amazon Web Services, Inc. https://aws.amazon.com/microservices/. 26.11.2019

Amazon Web Services. 2019b. Amazon Elastic Kubernetes Service. Amazon Web Services, Inc. https://aws.amazon.com/eks/. 5.11.2019.

Datadog. 2019. Monitoring in the Kubernetes era. Datadog, Inc. https://www.datadoghq.com/blog/monitoring-kubernetes-era. 5.11.2019.

Docker. 2019. What is a Container? Docker Inc. https://www.docker.com/resources/what-container. 26.9.2019.

Elastic. 2019a. The Elastic Stack. Elasticsearch B.V. https://www.elastic.co/products/elastic-stack. 5.11.2019.

Elastic. 2019b. Elasticsearch Reference - Data in: documents and indices. Elasticsearch B.V. https://www.elastic.co/guide/en/elasticsearch/reference/7.0/documents-indices.html. 5.11.2019.

Elastic. 2019c. Elasticsearch Reference - Information out: search and analyze. Elasticsearch B.V. https://www.elastic.co/guide/en/elasticsearch/reference/7.0/search-analyze.html. 5.11.2019.

Elastic. 2019d. Kibana Guide - Introduction. Elasticsearch B.V. https://www.elastic.co/guide/en/kibana/7.0/introduction.html. 5.11.2019.

Elastic. 2019e. Kibana Guide - Index Patterns. Elasticsearch B.V. https://www.elastic.co/guide/en/kibana/7.0/index-patterns.html. 5.11.2019.

Elastic. 2019f. Kibana Guide - Discover. Elasticsearch B.V. https://www.elastic.co/guide/en/kibana/7.0/discover.html. 5.11.2019.

Elastic. 2019g. Kibana Guide - Visualize. Elasticsearch B.V. https://www.elastic.co/guide/en/kibana/7.0/visualize.html. 5.11.2019.

Elastic. 2019h. Kibana Guide - Dashboard. Elasticsearch B.V. https://www.elastic.co/guide/en/kibana/7.0/dashboard.html. 5.11.2019.

Elastic. 2019i. Beats Platform Reference - Beats overview. Elasticsearch B.V. https://www.elastic.co/guide/en/beats/libbeat/7.0/beats-reference.html. 5.11.2019.

Elastic. 2019j. Beats Platform Reference - Community Beats. Elasticsearch B.V. https://www.elastic.co/guide/en/beats/libbeat/7.0/community-beats.html. 5.11.2019.

Elastic. 2019k. Logstash Reference - Logstash Introduction. Elasticsearch B.V. https://www.elastic.co/guide/en/logstash/7.0/introduction.html. 5.11.2019.

Elastic. 2019l. APM Overview - Overview. Elasticsearch B.V. https://www.elastic.co/guide/en/apm/get-started/7.0/overview.html. 5.11.2019.

Elastic. 2019m. APM Overview - Components and documentation. Elasticsearch B.V. https://www.elastic.co/guide/en/apm/get-started/7.0/components.html. 5.11.2019.

Elastic. 2019n. Kibana Guide - APM Getting Started. Elasticsearch B.V. https://www.elastic.co/guide/en/kibana/7.0/apm-getting-started.html. 5.11.2019.

Kepes, B. 2015. Elasticsearch changes its name, enjoys an amazing open source ride and hopes to avoid mistakes. Forbes. 10.3.2015. https://www.forbes.com/sites/benkepes/2015/03/10/elasticsearch-changes-its-names-enjoys-an-amazing-open-source-ride-and-hopes-to-avoid-mistakes. 5.11.2019

Kubernetes. 2019a. What is Kubernetes. The Kubernetes Authors. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. 26.9.2019.

Kubernetes. 2019b. Container runtimes. The Kubernetes Authors. https://kubernetes.io/docs/setup/production-environment/container-runtimes/. 26.9.2019.

Kubernetes. 2019c. Concepts. The Kubernetes Authors. https://kubernetes.io/docs/concepts/. 26.9.2019.

Kubernetes. 2019d. Nodes. The Kubernetes Authors. https://kubernetes.io/docs/concepts/architecture/nodes/. 26.9.2019.

Kubernetes. 2019e. Components. The Kubernetes Authors. https://kubernetes.io/docs/concepts/overview/components/. 26.9.2019.

Kubernetes. 2019f. Understanding Kubernetes Objects. The Kubernetes Authors. https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/. 26.9.2019.

Kubernetes. 2019g. Pod Overview. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/. 26.9.2019.

Kubernetes. 2019h. ReplicaSet. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/. 26.9.2019.

Kubernetes. 2019i. Deployment. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. 26.9.2019.

Kubernetes. 2019j. StatefulSets. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/. 29.9.2019.

Kubernetes. 2019k. DaemonSet. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/. 26.9.2019.

Kubernetes. 2019l. Carbage Collection. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/garbage-collection/. 26.9.2019.

Kubernetes. 2019m. Jobs – Run to Completion. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/. 26.9.2019.

Kubernetes. 2019n. CronJob. The Kubernetes Authors. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/. 26.9.2019.

Kubernetes. 2019o. Service. The Kubernetes Authors. https://kubernetes.io/docs/concepts/services-networking/service/. 26.9.2019.

Kubernetes. 2019p. DNS for Services and Pods. The Kubernetes Authors. https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/. 26.9.2019.

Kubernetes. 2019q. Volumes. The Kubernetes Authors. https://kubernetes.io/docs/concepts/storage/volumes/. 26.9.2019.

Kubernetes. 2019r. Persistent Volumes. The Kubernetes Authors. https://kubernetes.io/docs/concepts/storage/persistent-volumes/. 26.9.2019.

Kubernetes. 2019s.Cloud Providers. The Kubernetes Authors. https://kubernetes.io/docs/concepts/cluster-administration/cloud-providers/. 26.9.2019.

Kubernetes. 2019t. Cluster Administration Overview. The Kubernetes Authors. https://kubernetes.io/docs/concepts/cluster-administration/cluster-administration-overview/. 26.9.2019.

Kubernetes. 2019u. Installing Kubernetes with Minikube. The Kubernetes Authors. https://kubernetes.io/docs/setup/learning-environment/minikube/. 26.9.2019.

McGranaghan, M, Reeves, J & contributors. 2018. Readme-file. Ring project repository. https://github.com/ring-clojure/ring/blob/master/README.md. 5.11.2019.

Messina, D. 2018. 5 years later, Docker has come a long way. Docker Blog. 20.3.2018. https://blog.docker.com/2018/03/5-years-later-docker-come-long-way/. 26.9.2019.

Oracle. 2019. JAR File Overview. Oracle Corporation. https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html. 14.11.2019.

Siili. 2019. Frontpage. Siili Solutions Oyj. https://www.siili.com/. 11.11.2019.