Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Cluster infrastructure for LearnShell: monitoring and logging

Ilya Ryabukhin

Department of Applied Mathematics
Supervisor: Ing. Jakub Žitný

May 13, 2021

Acknowledgements

First of all, I'd like to thank my friends and family. Especially the family. They were the first people in my life who were believing that I'd finish my studies. Even in the time when I was not so sure about that. Their support is not possible to measure or evaluate. Mummy♥, daddy♥ and my sister♥, you are amazing!

I also want to thank doc. RNDr. Josef Kolář, CSc. He was the first person in the university who let me in, went with me through the campus and gave me the chance to enter without any exams. That's the kindness that I won't forget.

Also I want to thank several professors, who were helping me a lot to succeed in my studies. I'm talking about RNDr. Jiřina Scholtzová, Ph.D., Ing. Pavel Hrabák, Ph.D. and Ing. Michal Štepanovský, Ph.D. They were always respecting me in the same way as all other students.

Moreover, I want to thank all of my classmates that were helping me during my studies in the university. Especially Bc. Askar Kolushev. Without his support and passion to help I wouldn't prepare well to some subjects' exams or tests.

I want to thank the Mrs. Ludmila Facer, who was helping me to solve my administrative issues even out of her working hours or vacation. She was always asking about my study results and was worrying for me.

And my special thanks goes to two professors that taught me a lot and I won't forget them: Ing. Jan Trávníček, Ph.D. and of course Ing. Ladislav Vagner, Ph.D. These two are the most outstanding professors in my life. They have changed completely my mindset. It has started from very first lecture, where I've got late for 30 minutes. And starting that time and until nowadays they influenced me in the biggest way. Děkuji vám oběma za to.

Also my special thanks goes to my supervisor Ing. Jakub Žitný and my teammate Samuel Majoros. They were always helping with actual thesis.

And of course I should thank Netflix and Marvel for creating a great movies. Especially the Avengers and Spider-Man.

Declaration

In Prague on May 13, 2021 ....

Citation of this thesis

Ryabukhin, Ilya. *Cluster infrastructure for LearnShell: monitoring and logging.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Následující práce pojednává o analýze současného systému LearnShell, který je používán v současném vzdělávacím procesu na naší fakultě. V níže uvedeném textu se podívám na současnou situaci, sleduji všechna stávající úzká místa, navrhuji řešení pro některé z nich a definuji cestovní plán, jaké jsou další kroky, které zde lze provést. Vysvětlím také vybrané přístupy a technologický zásobník a porovnám je se současnými trendy na trhu.

Klíčová slova Kubernetes, Docker, DevOps, Shell, klastr, kontejner, orchestrace

Abstract

The following thesis is about analysis of current LearnShell system that is used in current education process on our faculty. In the text below I'll have a look on the current situation, observe all existing bottlenecks, propose solutions for some of them and define a roadmap of what are the next steps that can be done here. Also, I'll explain selected approaches and technology stack and compare it to the current market trends.

Keywords Kubernetes, cluster, Docker, Shell, container, DevOps, orchestration

Contents

List of Figures

Introduction

Motivation and objectives

Since I was starting my studies on the FIT faculty of Czech Technical University in Prague in 2017 one of the strongest memories was the enormous number of students trying to pass BIE-PS1 (Programming in Shell 1) subject. In my opinion, this is one of the most complicated subjects in the first semester. Moreover, that is also a compulsory one for all study branches, which means that there is no chance to fail it. Fortunately, I have passed this subject in the first semester, but lots of my classmates had retaken this course (there were some students that were taking it even more than 2 times).

It is worth to mention that this is a **practice-oriented course**, which means it is more about proper application of knowledge rather than just *cram knowledge*. And one of the reasons of failure could be the fact that we had the issues to get this practice. The only way to get somehow guided practice was attending the tutorials and completing the home assignments. But there were ***limited number of assignments*** and most of them already were solved during the tutorials. It led to the fact that the students have faced a problem of ***lack of practice materials***.

To solve this issue students needed to browse resources on the Internet to find some practice exercises. And if we consider the fact above that the course is practice-oriented, then we can conclude that the problem creates a consequence of ***lowering the chances to pass the course***. Another issue that needed to be fixed is ***manual assessment and exam tests check***. All the tests and exam during the semester were written on the paper and were manually checked. In comparison, another programming subject in the first semester, which is BIE-PA1 (Programming and Algorithmics 1), has its own evaluation system *Progtest*. Moreover, this system is used in other subjects on the faculty, so we can take as an axiom that ***the efficient automation of student's works evaluation is possible***.

All the issues covered above had motivated to create a *LearnShell* platform.

It makes students to learn the basics of Shell programming faster and easier. Moreover, the knowledge there should be really strong, as they are required throughout the whole education cycle. It means that system is assumed to teach in the ***most user-friendly way*** to motivate students to learn the subject not just to pass the subject, but to avoid possible issues with subjects that are kind of referencing to Shell.

So, the platform is designed to solve some challenges:

- practice of Shell exercises, an exercise is chosen randomly. The goal of that is ***to improve knowledge level of Shell*** and to ***find out the weaknesses in the knowledge base***;
- hosting of assessment tests, assignments and exams for BIE-PS1 subject. So, this system is aimed to be a ***single entrypoint to all practice part of the course***;
- automatic evaluation of exercise. This would let system to ***automate the evaluation process*** and ***decrease professor's load***.

Problems

Current system design is properly described in Figure 0.1.

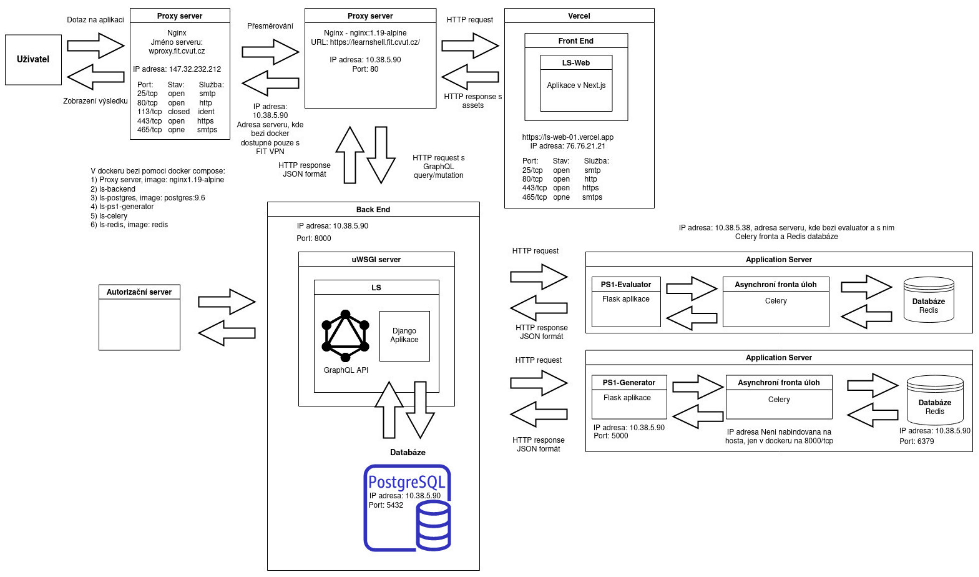


Figure 0.1: Current architecture in production environment

This solution has its own pros and cons. On the one hand, the solution is ***not designed as a monolith*** (most of the components are covered within isolated environment called *microservice*; this term will be properly defined and described in the following chapters) and is ***quite strong from the networking security perspective*** (solution has two proxy servers; one stands for default gateway, while second proxies the traffic through microservices). On the other hand, let us observe the negative factors of current implementation:

- the solution is scaled between two servers. So, there could be the problem of **server overhead**. Proxy is not able to handle that, simply because its goal is to forward the request to *some server that can fulfill it*, while we need to ***balance the load*** between the servers;
- another problem of current implementation is ***logging***. Logs are usually one of the most powerful tools for debugging. And here we have multiple servers to handle, so in current approach we should *manually integrate all logs*. This is non-efficient and risky approach, because some of the data may be lost during manual transfer. It means that ***proper logs collection*** is required. Also, it could be a benefit if we could enable some ***efficient logs search and monitoring***, since it would enable solution to scale faster and would not slow down the log analysis process;
- at the moment solution is *deployed manually* on multiple servers. And complexity of this operation would *increase linearly to the number of servers*. That is definitely not a best practice, so there is a need to implement some ***DevOps best practices***, especially ***CI/CD pipelines*** (it will be described further).

To sum this all chapter up, we have a current solution, that has its pros and cons. It can and should be improved, so my goal is to analyse all the found issues and provide a solution to ***logging and monitoring*** related problems, including *CI/CD pipelines*.

Chapter 1

State-of-the-art

Nowadays more and more companies all around the world get better understanding of *Digital Transformation* term. It can be defined as *a process of using digital technologies to create or modify business processes* [7]. And the system that is given to improve is a great example of transforming the *slow and inefficient process* into ***modern and efficient one***, because it aims to substitute the manual assignments checks and improves education process.

It is not a secret that digital transformation is the way to cut the costs. But before the actual transformation let me introduce several types of costs that we will use further in this chapter. So, there are two types of expenses:

- ***Capital Expenses (CAPEX)***. This is a one-time expense that is used for long-term purpose. An example of such could be a hardware purchase [8]. Usually, this expense is quite huge, but it is done only once. So, we can conclude that CAPEX is preferred in *long-term initiatives and projects.*
- ***Operational Expenses (OPEX)***. This is an opposite to CAPEX: OPEX is an expense that is used for day-to-day operations. An example of such in our domain is usage of cloud provisioned infrastructure. This expense is relatively small, if we would compare it to CAPEX, but is done more regularly. So, the OPEX is the best solution in case there is an *innovative project* or the solution is on *MVP stage.*

Now we are aware of several expenses types and we can explore several ways of creating the solution. And all of them are on the different phase of digital transformation. Let us briefly go through all the possible approaches:

- ***Traditional approach***. In this case solution is designed in rather naive way, which could lead to several issues. The main one is that this solution *is not scalable enough*, which means that it would take a longer time to deploy it to some dedicated server. Also, it would require *additional low-level skillset* in infrastructure setup and support. This would lead to growth of both CAPEX and OPEX, which is usually not the best thing to start the project with.

- ***Cloud-native approach***. Such an approach brings lots of benefits to the solution. In this case solution has zero CAPEX, which is preferred in the very beginning. Also, building a solution in such manner would lead to *broaden vendors choice* that would make *lower OPEX* than in traditional approach because of huge amount of competitors. And depending on solution, it can even be deployed without infrastructure support. So, that is again lowering the OPEX. Moreover, the solution will be available depending on vendor's SLA, so it lead to *high availability* of solution.

Actually, huge penetration of cloud computing is one of the growth reasons of digital transformation [9]. And cloud computing does not always mean that solution will be hosted somewhere in external environment. Let us have a look on different types of cloud:

- ***Private cloud***. Basically, that is the most expensive offer. The only reason for that is that a company is required to purchase *specific hardware* that is usually more expensive than usual. Nevertheless, this offer guarantees that the data would not leave physical building of the data center. So, that is the *most secure solution.*

- ***Public cloud***. All the services are deployed in public data centers, but there are a strong security compliance rules. It means that there is a *shared responsibility*, where vendor is obliged to fulfill SLA and physical security, while company should maintain data security. This offer is the easiest and fastest to implement, but some companies would not establish it.

- ***Hybrid cloud***. This is the compromise that takes all advantages of previous two. It has everything that *does not have any obligatory requirements in public cloud* and everything that is *restricted to some physical location either by governmental or company's compliance - in private cloud.*

Another things to discover is all possible cloud offerings. All of them are used in their own specific scenarios, but we should be aware of all of them. So, there are three types of cloud solutions:

- ***Infrastructure-as-a-Service (IaaS)***. This offering transfers responsibility related to physical assets to the vendor. However, the company must *maintain all the software*, except OS. As the result, this is the *most flexible solution*, but there is still lots of tools to maintain.

- ***Platform-as-a-Service (PaaS)***. This is the most popular solution on the market at the moment [10]. PaaS hides the implementation from the user, but its usage is still a thing to be properly described, as *data security is still considered by company.* So, this solution is *a median between all offering* that requires less support.

- ***Software-as-a-Service (SaaS)***. This thing is also called ”black box”. Simply, that is a solution where the company should not care about anything like security. It can be configured even to be run locally, however, the company *doesn't control the data workflow.* In total, this solution is *the easiest to use, but there is no control over its work.*

To sum this chapter up, the current digital transformation process focuses mostly on creating cloud-oriented solutions, as the cloud computing opens up new growth opportunities. That is the reason why it is extremely important to build solution in cloud-native manner, since this approach would provide additional benefits from the operations and financial perspective.

Chapter 2

Analysis and architecture

2.1 Microservices Architecture

This term has been used several times in the text above, but, unfortunately, there was not convenient place to get so much into details. And now I would like to define and describe several terms, that would be useful in the following text. Probably, that is the **most important part of the thesis**.

2.1.1 Containers

Docker defines container as *standard unit of software that packages up code and all its dependencies so the* ***application runs quickly and reliably*** *from one computing environment to another* [11].

2.1.1.1 What is the difference from Virtual Machine?

The main difference between container and VM is described in the Figure 2.1. Let us highlight all the differences:

- Container is an abstraction of ***code and its environment***, so it abstracts the *application level.* In the meanwhile, VM abstracts *everything including hardware* [11].
- Multiple containers ***share single OS kernel***, which makes them to have a small size (usually measured in MBs), while each VM *copies whole OS and has separate kernel*, which would increase the image size up to tens of GBs [11].
- Including the previous fact, we can also conclude that ***containers is fast to boot*** in comparison with VM. It is quite obvious (each VM has to boot its own OS and only then the app, while container should boot only the app), but it is worth to mention.

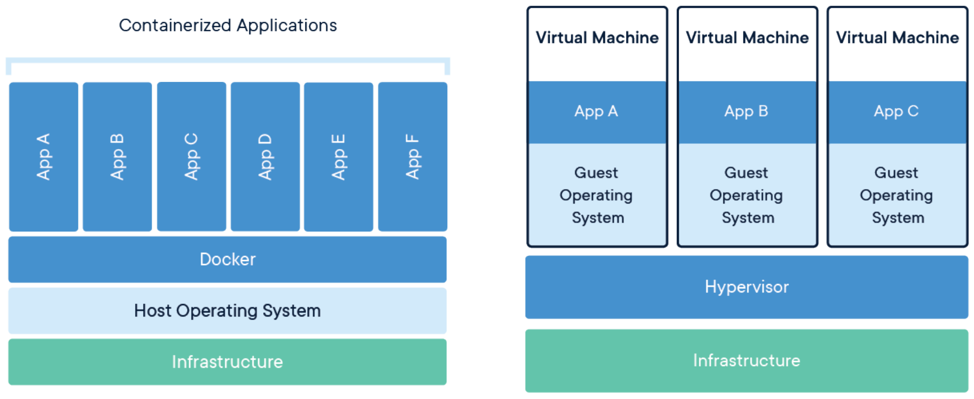


Figure 2.1: Difference between container and Virtual Machine [1]

The image also includes a part called *Docker*. But what is that and how is it related to the containers at all? It will be described in the next subsection.

2.1.1.2 Docker

So, in the previous section there was explained a concept of container. Now let us focus on the thing called ***Docker***. This is an *open-source software designed to facilitate and simplify application development. It is a set of platform-as-a-service products that create isolated virtualized environments for building, deploying, and testing applications* [1]. And as you would remember the previous section, virtualized environments is called a container. So, Docker and container is not the same thing. Let us explore the difference.

As it can be seen on Figure 2.2, there is a quite long way from executing Docker to a container creation. We are not yet interested in the *Kubernetes* part, it will be introduced further in this chapter. So, let us get through the main steps:

1. Docker calls a component called ***containerd***. This is *an open-source container runtime, where the container is further getting prepared* [2].

2. After that another component *Open Container Initiative (OCI)* is getting called. It creates and runs all container processes [2].

3. Then OCI calls *runc*, which is a command-line tool for running the container processes creaeted by its caller [2].

4. Only at this moment ***container is prepared***.

And to be precise, all of this time only container concept was introduced. Now it is time to get into another terms: ***what is Docker image and does it differ from Docker container***?

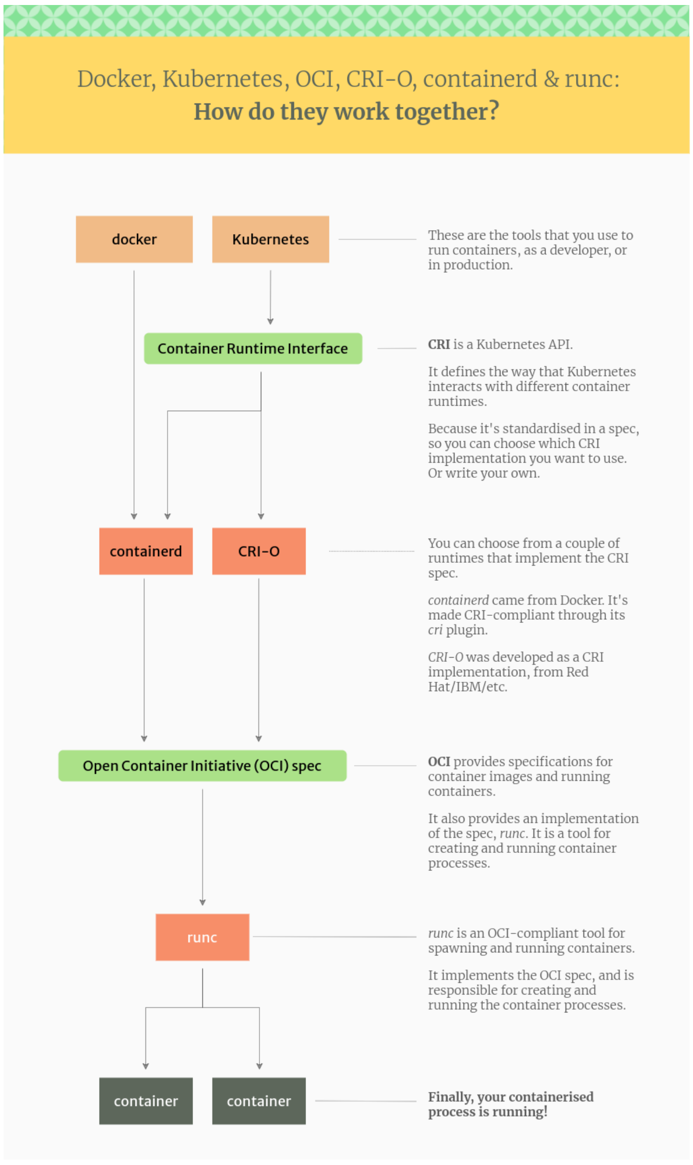

Docker, Kubernetes, OCI, CRI-O, containerd & runc:
How do they work together?
docker
Kubernetes
These are the tools that you use to run containers, as a developer, or in production.
Container Runtime Interface
CRI is a Kubernetes API.
It defines the way that Kubernetes interacts with different container runtimes.
Because it's standardised in a spec, so you can choose which CRI implementation you want to use. Or write your own.
containerd
CRI-O
You can choose from a couple of runtimes that implement the CRI spec.
containerd came from Docker. It's made CRI-compliant through its cri plugin.
CRI-O was developed as a CRI implementation, from Red Hat/IBM/etc.
Open Container Initiative (OCI) spec
OCI provides specifications for container images and running containers.
It also provides an implementation of the spec, runc. It is a tool for creating and running container processes.
runc
runc is an OCI-compliant tool for spawning and running containers.
It implements the OCI spec, and is responsible for creating and running the container processes.
container
container
Finally, your containerised process is running!

Figure 2.2: Difference between Docker and container [2]

2.1.1.3 Docker image vs Docker container

In the text above there was introduced the container concept and its relation to Docker. But further in the text there will be two separate terms with huge difference: *Docker container* and *Docker image*. Let us go through them.

First, let us define the ***image***. This is the *an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run* [2]. Figure 2.3 shows that image contains several *read-only layers* and one *container layer*. Each layer represents some instruction in image definition file (*Dockerefile*). All the details of image definition and best practices will be described in the next chapter.

The last layer in the container is read-write one and simply represents the application itself. Imagine that the image contains some OS and the last layer should be an executable that runs the app. In this case all the OS setup and dependencies installation is read-only, which is quite obvious (the OS setup is quite stable and should not change within the same app), while the last layer executes the app and it is a read-write one.

To sum up this section, we can think of Docker image as a ***template***, while container is an ***instance*** of this template.

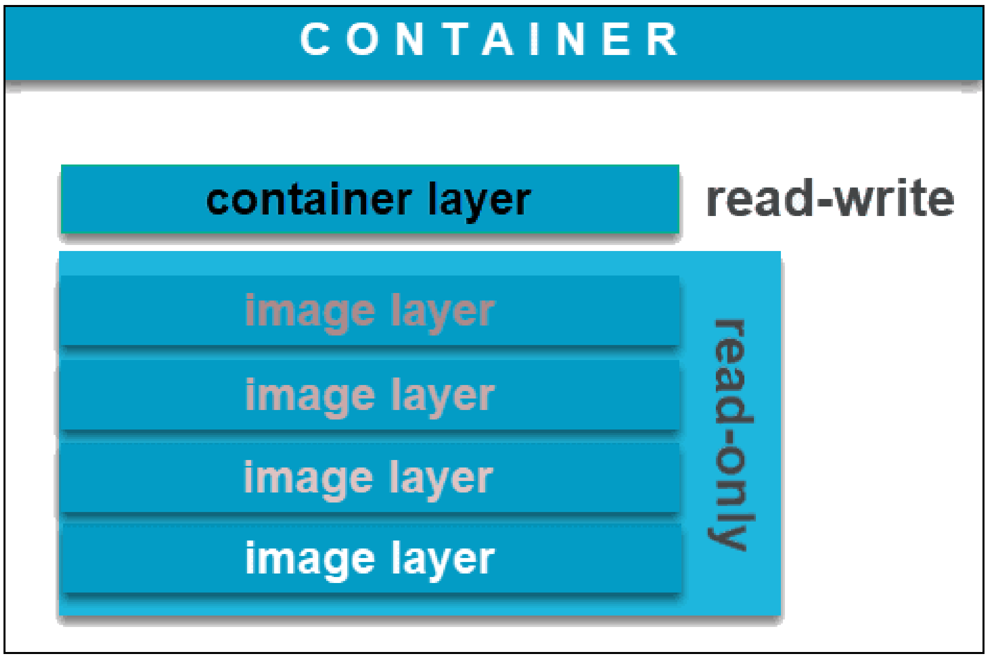


Figure 2.3: Docker container setup [2]

2.1.1.4 Alternatives to Docker

There a usual misunderstanding that Docker container and general container are the interchangeable terms. Well, that is not true. Simply because in 2017 99% of containers were done using Docker, but one year later the Docker share decreased up to 83% [12]. I would not go into all alternatives to Docker, my goal is just to show that there is at least something that is also used on the market. So, let us find it out.

The first project I want to talk about it ***LXD***. The share of such technology was around 1%. It *emulates the experience of operating Virtual Machines but* *in terms of containers and does so without the overhead of emulating hardware resources* [13]. I would personally assume that it looks like `chmod` command in Linux kernel (changing the permissions for individual users and groups of users), but to isolate the whole environments, not just a single file or directory.

The second project I would like to mention is *rkt* produced by CoreOS. It is assumed to be *more secure alternative to Docker* [13]. Also, rkt supports not just Docker images, but *appc* containers as well (another alternative to Docker, which is deprecated). In addition, rkt works out of the box in *Kubernetes* (it will be described further) and for that reason is the only main competitor with 12% share in 2018 [12].

To wrap this whole conversation about containers, here was introduced the most important concept that will be used further and will become a basis for further architecture and implementation details.

2.1.2 What is a microservice?

Now we know what is container. Now let us have a look at the ***microservices*** themselves. On the Figure 2.4 there is described an example of e-commerce application with microservices architecture.

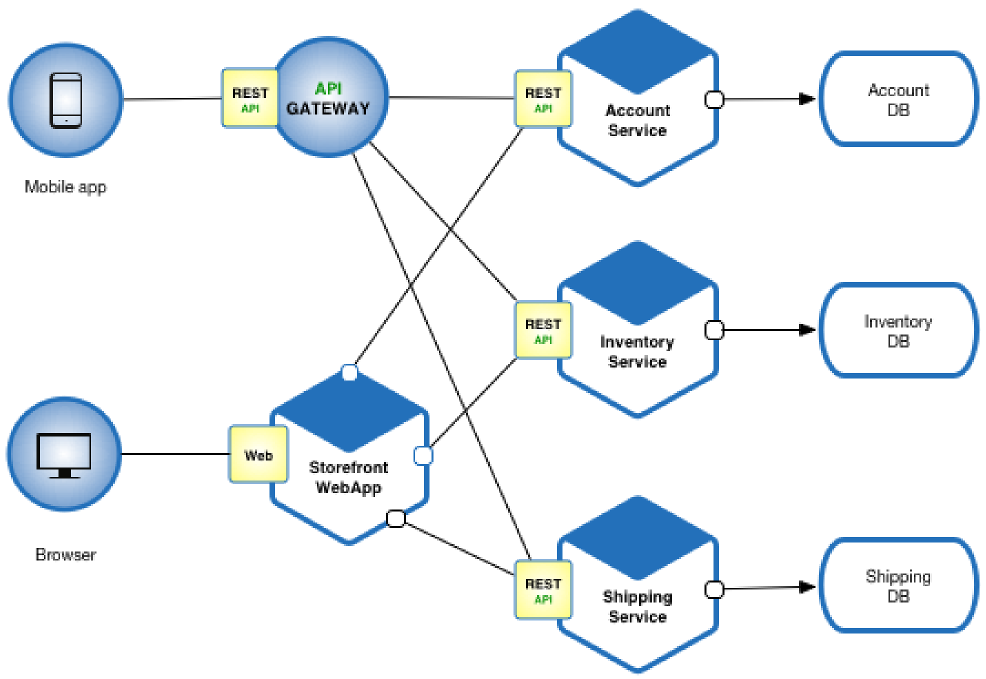


Figure 2.4: Application with microservices architecture [3]

Let us go through the main points of such an approach:

- All solution consists of small components, each of them is called a *microservice.* Each of them is isolated from each other, so it is a *container.*

- Each microservice connects to the only database. The reason for that is compliance to *Database-per-Service* pattern [14]. I would not go into details of the pattern, since that is a bit out of the scope, but the only thing that is sufficient to know is that ***each DB is also a container.***

This architecture approach is quite interesting, but how can we answer if *microservices architecture is good enough*? Well, we need to compare it with more traditional way, or the ***monolith architecture***.

2.1.3 Microservices vs monolith?

Now we know what is a microservice. So, it is time to find its pros and cons. Comparing to *monolith architecture* (see Figure 2.5), let us describe some meaningful advantages of microservices:

- The microservice concept is ***easier to understand and maintain***. It means that the code would be responsible just for single service. The monolith has the opposite approach and doesn't isolate the code, so it becomes less readable. That would leave each service relatively small and that is the reason of calling each service a *microservice.*

- The isolation of components brings a benefit of *loos coupling.* That is one the best practices in the architecture that simply states to make the components *as independent as possible.* This leads to ***faster deployment and less downtime***, since we need to deploy not the monolith service, but only its updated part, which also means that there could be even no downtime at all.

- Despite the loos coupling, there is an advantage of *high cohesion.* This is also one of architecture best practices that means that we can explicitly state *which services are called by another ones.* It brings ***better testability and maintainability***.

- Because of proper code isolation, there is a great ***fault isolation***. So, if some subsystem fails, then there is an option to "roll back" only the failing component. Moreover, this can be also used for easier adjustment of the system.

As all the pros of microservices are discussed, now we should go to its cons:

- Basically, *monolith is rather simpler than microservices.* For that reason, projects with 2-3 components would not need such a complication.

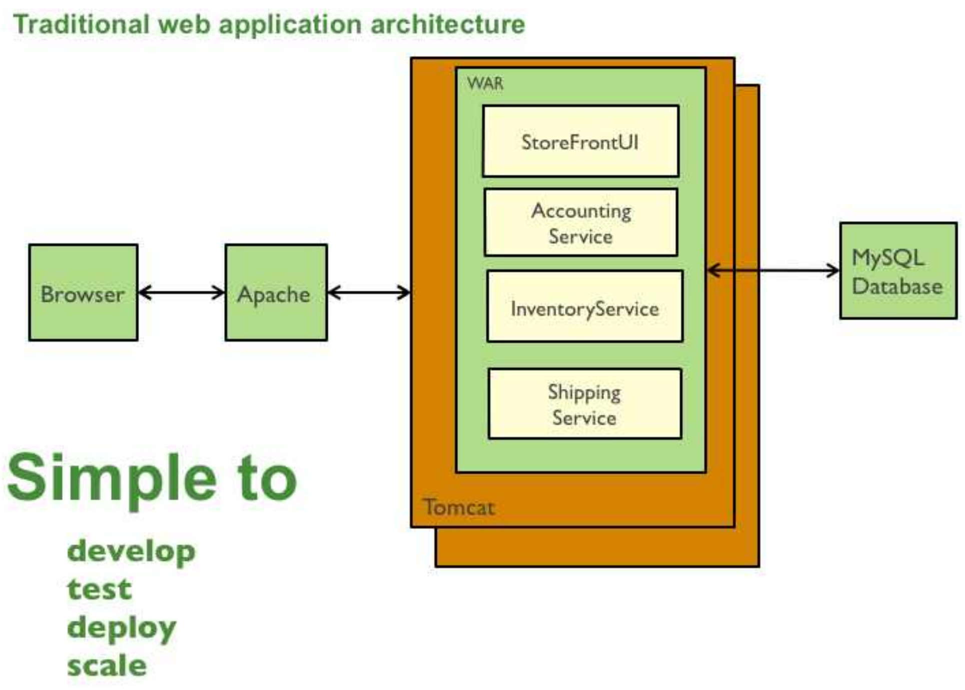


Figure 2.5: Application with monolith architecture [4]

- There is a huge question about *how to decompose the monolith into microservices*. In case of complicated data workflow that would lead to ***additional data synchronization***, just because the microservices are independent and not synced.
- Microservices use *huge amount of memory* in comparison with monolith. Just because each of them has a separate environment, which increases whole memory consumption ***by N times in case of N microservices*** in comparison with single monolith. [3].

So, let us wrap the section up. Here were discussed all the main things about microservices architecture. Also, there were discussed not only the basic concepts, but overall its pros and cons. Now it is time to move to specific technologies or approaches that would let us improve existing solutions.

2.2 Kubernetes

2.2.1 Introduction

Now we are aware of the microservices architecture basics. Now let us get a bit deeper in this topic, since all the ***implementation is mostly based on***

concepts from here.

Kubernetes (also known as K8S) is *portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation* [15]. The project was open-sourced by Google in 2014. This solution supports microserevices architecture and is used for ***load balancing*** and ***building a cluster***, which actually enables usage of multiple hardware units within the same solution. K8S itself consists of several components (see Figure 2.6), let us go through the K8S cluster architecture.

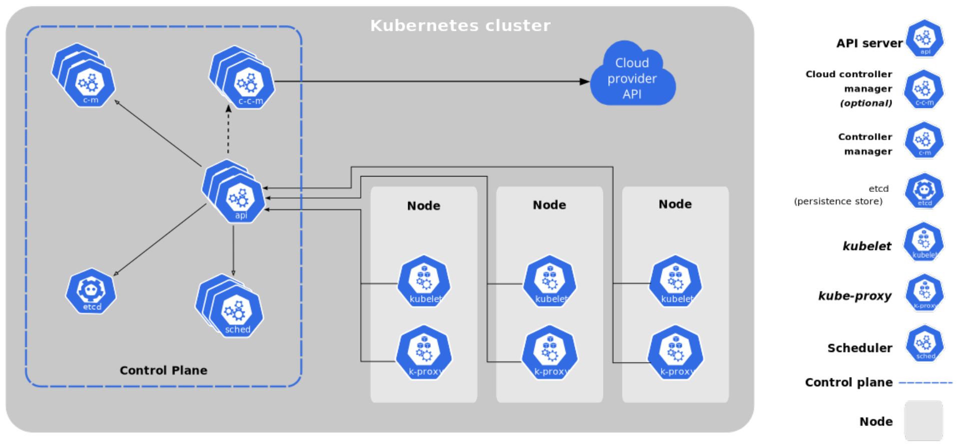


Figure 2.6: Kubernetes cluster architecture [5]

2.2.2 K8S Architecture

2.2.2.1 Control Plane

This is the most high-level component. It *makes global decisions about the cluster, as well as detecting and responding to cluster events* [5]. Despite the fact that it controls overall cluster it connects to the *cloud provider*. The thing is that each ***cloud provisioned K8S cluster has a default load balancer***. It means that load distribution between *nodes* (will be described further in this section) is managed automatically.

So, the control pane consists of several components:

- *kube-apiserver*. This is actually the ***public facing component*** that exposes the API and used in all the communications. Also, that is used for *horizontal scaling* (it will be described in the next section);

- *etcd*. This component represents ***highly available "key-value" storage***. In computer science there is a data structure called *Dictionary* or *Map* that is the most convenient to explain the component purpose on;

- *kube-scheduler.* This is one of the main features of controller plane. This component stands for ***pods scheduling***, meaning that the ones with no node will be assigned to any. The difference between *node* and *pod* will be described later in the text;
- *kube-controller-manager.* This is a component that consists of *multiple controllers* (functionality of nodes, jobs, endpoints, services) that are combined for simplicity reasons;
- *cloud-controller-manager.* This is an ***optional*** manager and used in ***cloud-provisioned environment*** and also consists of several controllers (cloud load balancing, deletion of nodes and services).

2.2.2.2 Node

Now it is time to learn more about ***node***. Simply, that is a ***physical or logical virtual machine*** [16]. It is used to run the cluster or its part on such a node (the cluster with single node is also called *minikube*). In production environment each K8S cluster has at least three nodes. And there should be two types of nodes: ***master*** and ***worker*** nodes. The master one is needed to *deploy a control plane*, while the worker node *runs the application itself.*

Each worker node has several components under the hood:

- *kubelet.* This component makes sure that ***each container runs in a pod*** [5]. It is worth to mention that kubelet manages not all containers, but just such that were created by the K8S.
- *kube-proxy.* This is network related component that ***maintains network rules on each of nodes***.
- *Container runtime.* This is a ***the software that is responsible for running containers*** [5]. That is actually the main part of Docker definition. Simply because Docker is one of possible container runtimes. Some of them were mentioned in the beginning of the chapter, but now it is time to talk about the K8S specific container runtimes. K8S supports several of them: previously mentioned *Docker*, *containerd* (the enterprise standard container runtime that was derived from Docker), *CRI-O* (lightweight K8S special container runtime) or *any other implementation of K8S CRI (Container Runtime Interface)* [5].

2.2.2.3 Pod

The last (and the smallest) component that is contained in K8S is a pod. Basically, that is a *is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers* [16].

Each node can have any amount of podes, while each pod should have *at least one container.* Since that, there are two main ways of using the pods:

- *One container per pod.* This is the ***most common way in K8S*** and is recommended in most of the cases.

- *Multiple containers per pod.* This is not so common pattern that enables to connect the containers called *co-located* ones. It has rather advanced usage scenarios and needs to be used only in specific cases [16].

2.2.3 Scaling

This term was used before in the text. Well, there is a time and place to define that. Basically, that is used to *make solution more available* and *decrease downtime.*

As it can be seen in Figure 2.7, there are two types or scaling available:

- *Vertical scaling.* This means that ***there is a single machine that increases or decreases its capacity***. This approach is quite usual in cloud provisioned infrastructures (you can select your app plan or VM type). But this approach is usually not applicable for some on-premise infrastructures, because on-premise servers could have not so powerful hardware. This type of scaling is available in K8S, but at this moment it is in one of preview versions, so it is quite risky to use it in production environment.

- *Horizontal scaling.* This is the opposite to its vertical sibling and means that ***multiple instances with the identical software and hardware setup are initialized***. This is an *only stable scaling in K8S* that is available. This approach can be easily applied on either cloud infrastructure or on-premise environment.

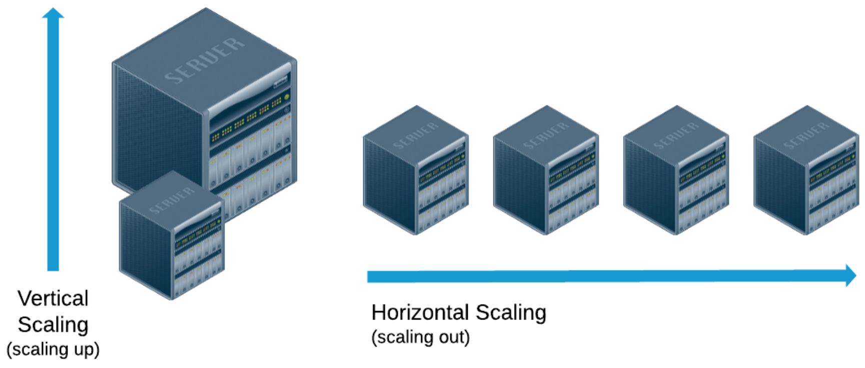


Figure 2.7: Difference between horizontal and vertical scaling[6]

So, let us wrap it up. We have learned about *Kubernetes*, have defined *pod* and *node* terms. Also, there was defined a difference between *horizontal* and *vertical* scaling. Now it is time to move forward and get to the core of the implementation: ***Logging and Monitoring***.

2.3 Logging and Monitoring

This part is the actual ***topic of the thesis***. Here will be defined both *logging* and *monitoring*, and their main advantages.

2.3.1 Logging

This term can be defined as *practice of collecting all the data produced by hardware and software* [17]. For example, that could be an authentication or authorization attempt, data sent as a response of API, usage data of CPU or RAM etc. This practice consists of several independent processes and multiple goals:

- ***Log aggregation***. This is an actual the process of collecting the data into some single place. Most frequently used way for that is simply write this data to some `.txt` or `.csv` file. It is also quite obvious that the ***log file has a huge size***.

- ***Log storage***. This stage was partially mentioned in the previous process. It is not about only the exact logs storage, but is about the *logs management*. In most of the companies and production systems there is a thing called ***retention period***, which is the *number of days that system's data should be kept*. So, it means that the ***out of date log files must be deleted***.

- ***Log enrichment***. At this moment we have somehow managed logs aggregation and storage. Now it is time to add some other information. It is not produced by the system itself, but can be useful further. Such an approach is called ***enrichment***. Such an information could be additional information about user, for instance, IP address and corresponding geographic location.

- ***Log analysis***. This is probably the main part of the whole process. everything has a purpose and logging is not an exception. It was mentioned above the log files are huge, so it makes lots of sense to analyze it not manually, but using some tools. So, there is a thing called *log analyzers*. This process has several goals, and the main ones are to ***find bottlenecks in the processes*** and ***comply with regulatory or security standards***. At first sight it main seem that log analysis is already a part of monitoring, but that will be described a bit later.

To sum this all up, the logging practice becomes a must have thing to implement nowadays. However, the log files are relatively huge in size and require proper maintenance. And that is the thing where ***monitoring*** comes into play. And we will describe it further.

2.3.2 Monitoring

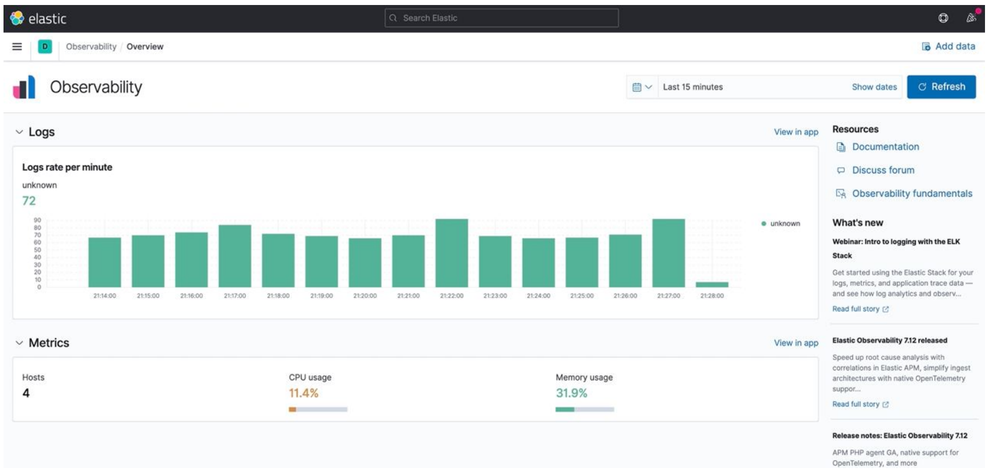
This term is widely used within topics that are related to the support. Broadly speaking, monitoring is the *process of collecting, aggregating, and analyzing the metrics provided by the components in your environment by using a monitoring solution* [18]. It consists of three main parts:

- ***Metrics***. Simply, that is a *raw data about resource usage or behavior that your monitoring system collects from any of the applications or services on your infrastructure* [18]. There are several inputs for them (CPU performance, disk space, network activity), and actually ***logs is one of possible metrics within the monitoring***.
- ***Monitoring***. This process is actually *build an environment to control the metrics* and *prepares a base for alerting.* This part can represent all the insights using graphics (maps, charts) and visualizing some metrics into some dashboard.
- ***Alerting***. This part is used to *notify the host about some anomaly in the monitored metrics.* It is quite clear that host would not notice some abnormal actions, because of huge amount of information. So, the goal of this part is to solve this issue.

So, now it is quite obvious that logs is one of possible metrics within the monitoring. Now, let us define several types of monitoring. There are some domain specific ones (e.g., *network* or *database monitoring*) that could be understood quite straight forward, but let us focus on two main types of monitoring:

- ***Real User Monitoring (RUM)***. This kind of monitoring focuses on *user-related types of activities.* This can be frontend load time, page-to-page customers conversion, spent time on each page etc. Thus, it can be any activity that is performed by the user while the app usage.
- ***Application Performance Monitoring (APM)***. Usually this type of monitoring is considered as the only possible, but that is not true. Its goal is to find the places where it is possible to *reduce costs* or *optimize the workflow.* And here is presented only the *application-related activities*, such as *system metrics.*

On the Figure 2.8 there is an example of how can monitoring solution look like. Simply, that is the combination of both charts and metrics. But let us



Webinar: Intro to logging with the ELK Stack

Get started using the Elastic Stack for your logs, metrics, and application trace data — and see how log analytics and observ...

Elastic Observability 7.12 released

Speed up root cause analysis with correlations in Elastic APM, simplify ingest architectures with native OpenTelemetry suppor...

Release notes: Elastic Observability 7.12

APM PHP agent GA, native support for OpenTelemetry, and more

Figure 2.8: Example of logging and monitoring solution

define two main reasons to have such a logging and monitoring embedded into the solution:

- ***Resource Management***. The logging and monitoring enables the solution stakeholders to define the bottlenecks in the processes and perform operational changes considering real data from the system. Also, such management could help to control if the solution meets the key requirements or SLA.

- ***Advanced troubleshooting***. Having logging and monitoring helps to find the issues faster and, consequentially, cheaper. This becomes more critical in the cluster environment, when there are multiple pods and nodes that should be properly managed. Also, there is an option to set up the alerting, so the administrator would be informed immediately after some accident happened. That could decrease the downtime and again help to meet the SLA or other non-functional requirements.

So, let us wrap the discussion about the logging and monitoring part. Here were described the goals of these two practices and advantages that they bring. Also, we have found out the difference between the kinds of metrics and the high level processes. And both logging and monitoring can be called *a DevOps practices*. And we will discuss the DevOps just now.

2.4 DevOps

Now it is time to go into another important topic. It it related to the logging and monitoring, as well as it connects all the things together.

This term is defined as a *set of practices that combines software development (Dev) and IT operations (Ops)* [19]. These practices become more and more important today. Furthermore, I would like to mention that there is a standard *ISO/IEC AWI TR 24586*, which is now under development by ISO and is aimed to standardize everything related to DevOps [20].

DevOps can be used as some kind of umbrella term for several practices. Obviously, some of them are related to the actual development (code version control, testing, CI/CD etc.) and other - to operations (releases, monitoring etc.). And ***monitoring is one of DevOps practices***. Let us meet with some others: *Agile*, *Continuous Integration* and *Continuous Delivery* or CI/CD.

2.4.0.1 Difference from Agile

Agile is another buzzword that is related to DevOps, but is different and is often used as a DevOps substitution. To be precise, that is not true. Agile has a definition of *practices involve discovering requirements and developing solutions through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).* [21]. So, it means that **agile can be used mostly in software development** and ***it has tighter application domain***, while DevOps combines practices from multiple domains.

2.4.0.2 CI/CD

This part is about one of practises mentioned above. Here I would like to define what is *Continuous Integration (CI)* and *Continuous Delivery (CD).*

But before I would go into that let me define the process that we would consider as a baseline for improvement. So, the current process looks like the following:

- ***Deployment is done manually***, which leads to *high human factor impact.*
- ***The images are built manually***, which leads to *high human factor impact.*
- ***The code is tested manually***, which leads to *high human factor impact.*

There was mentioned several times a ***high human factor impact***. Now it is time to describe its consequences, since that could be not that obvious:

- Manual processes take the time of developers that they can spend on something more meaningful. So, lack of automated processes ***decreases the productivity***.

- ***Existence of human factor increases the error rate***. Simply because human is able to get tired or lose some meaningful detail that would lead to error. And the cost of such an error can be extremely high (violation of service SLA, spending additional resources on resolving the issue, drop in company's quotes etc.)

All the processes described above are *easy to execute and do not change depending on input*. It means that this processes ***can be easily automated***. And including the fact that there is a high risk of human error, we can conclude that this processes may and must be automated.

And here is the place where CI/CD comes into play. It helps to solve the problems described above as they are the ***practices for automating the processes related to integration, test and deployment***. Each of them is described with the file called *pipeline*. CI/CD consists of two parts:

- CI stands for *Continuous Integration* and used for automated tests and builds. Usually this thing is done after each commit in some branch or pull request to it.
- CD is a *Continuous Delivery* and should deploy solution automatically. It is supposed to be triggered after successful CI operation. Usually because of security in production environment there is a tool called *manual approval* that helps to ***decrease the risk of putting the whole environment down***.

So, let us sum all the section up. Here was described the concept of DevOps and the difference between two popular practices: *CI/CD* and *Agile*. Now there is full amount of theory required to propose the changes to the existing solution. Let us have a look.

2.5 Solution Architecture

Now let us have a look on the solution architecture that is proposed to implement (see Figure 2.8). It includes everything that was covered in this chapter. There is not a detailed description (that is called a *design* and will be described in the next chapter) and is used just for understanding the high-level components.

There are some workflows that are proposed to implement:

1. There is a ***K8S LearnShell cluster*** where all the components are dockerized. It sends all the data from the cluster to the selected ***logging and monitoring solution***.
2. Despite the logging there are the processes to be improved. To be exact, that is about CI/CD. So, during the CI pipeline all the containers should be uploaded to ***private Docker registry***.

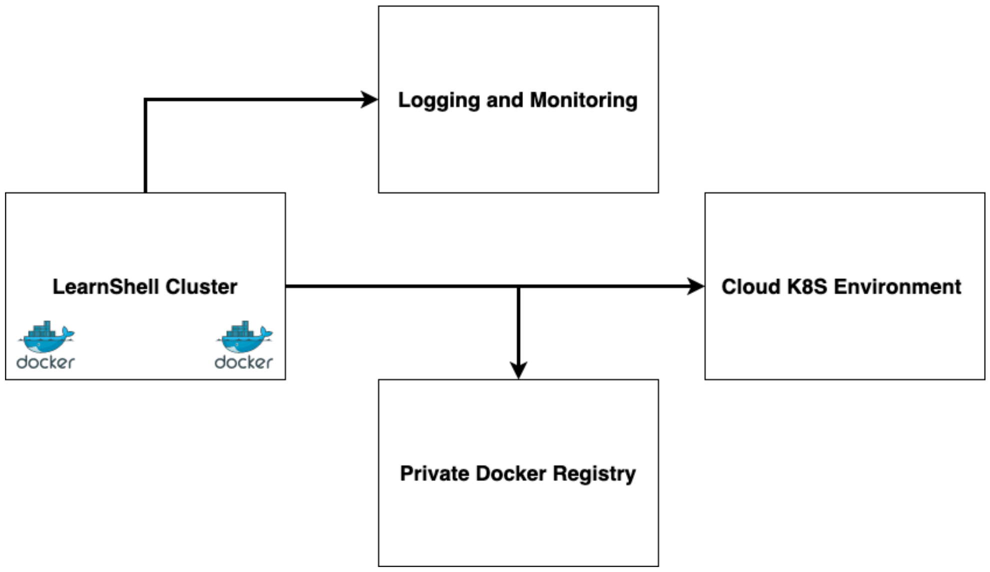


Figure 2.9: Proposed solution architecture

After all, we need to conclude this chapter. Here were discussed all the concepts that are crucial to understand the whole scope of work. Now, we can go into implementation details.

Chapter **3**

Realisation

Two chapters above were a bit abstract and not exact about the implemented solution. Now it is time to get more into that. In this chapter we will get deeper into the exact solution design and all implementation details.

3.1 Solution Design

In the previous chapter there was discussed an architecture of the solution. It is more high level, but here will be discussed all the details. It is completely described in the Figure 3.1. Now let us go into more details of the implementation:

- There are bunch of ***private GitHub repositories*** that are representing each component of the LearnShell solution.
- After the commit in the default branch there is a CI pipeline triggered that does two things: ***builds and pushes image into the Azure Container Registry*** and ***deploys the container of previously pushed image in Azure Kubernetes Service***. It is pushed securely, because in the middle of the process there were created a `imagePullSecret`, which provides another layer of security.
- In the meantime, the cluster itself should be ready for logging. There are 4 components needed to obtain that (it will be discussed further in details): *ElasticSearch* for real-time search, *Kibana* for visualization, *Metricbeat* for metrics collection and *Filebeat* for logs collection. All of them are downloaded using ***Helm charts***. These are the components that are already k8s-friendly.

Let us sum this section up. Here we know exactly about what and how should be implemented. Now let us move to the exact implementation.

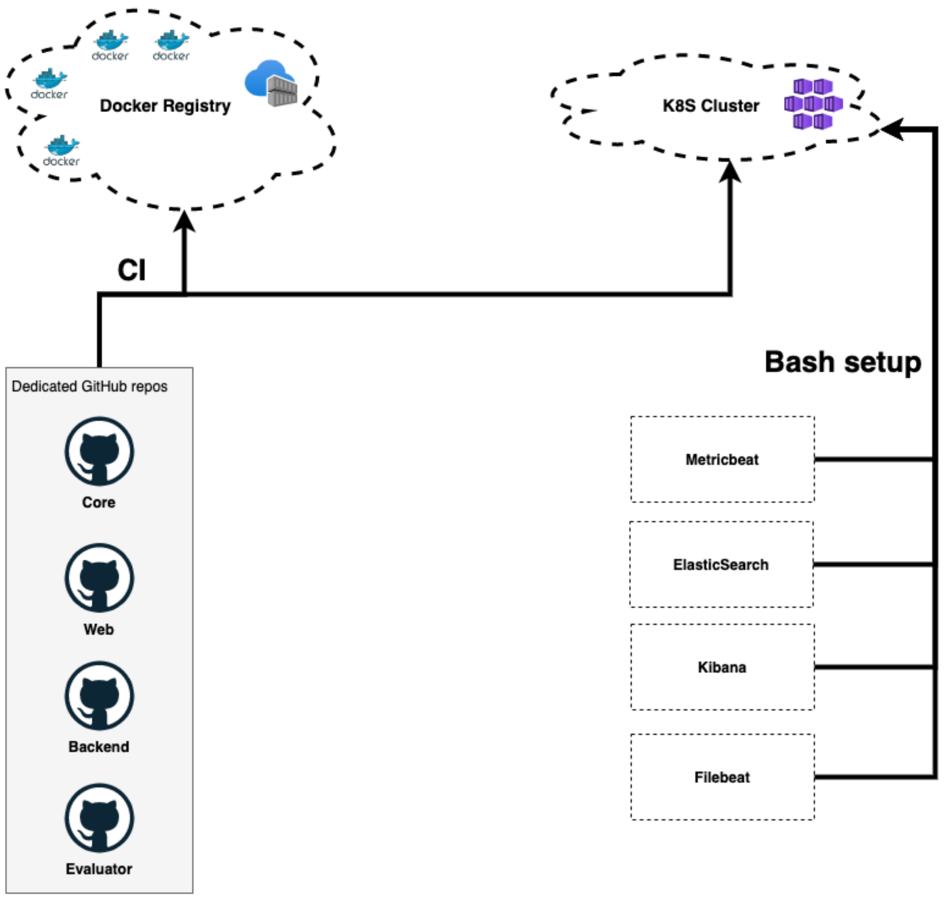


Figure 3.1: Proposed solution design

3.2 Implementing logging and monitoring

3.2.1 Solution selection

The monitoring becomes ***critical in the K8S environment***. The reason for that is quite simple: there is a tremendous amount of log files that are created by single node application, so this amount multiplies by number of nodes that are used in the cluster. In case of large companies that could be even thousands of nodes, so the amount of logs can be up to *tens or hundreds of TBs per day*. And one of the most commonly used solution is ***ELK***. It is combined out of three solutions: ElasticSearch, Logstah and Kibana. So, let us quickly go through all of them.

3.2.1.1 ElasticSearch

This is the most important part of ELK solution. It is a *NoSQL database providing distributed data storage*, which is based on Java that provides ***high compatibility*** [22]. Its main feature that it provides ***near real-time search engine*** that ***supports full-text search*** and can be called from RESTful API. It uses indexes for search, so it can bee possible to ***improve search speed***.

3.2.1.2 Logstash

This part connects the solution with the cluster itself. Its main goal is to *collect logs from several sources and transform them.* It is worth to say that it ***does not matter if the data is structured or not***. Additionally, it can ***find the geographical coordinates from IP address*** and ***anonymize PII data***, where the last feature provides additional security [22].

3.2.1.3 Kibana

This solution is a ***data visualization tool***. It connects all the dots together: Logstash provides the data, Kibana searches for given data in ElasticSearch and draws all the graphics in near real-time. This solution has several disadvantages (limited data export options [23]), but as a part of ELK solution that would not make any meaningful issues.

3.2.1.4 Alternatives to ELK

First of all, the Kibana is not the only solution to visualize the results of ElasticSearch. There is a solution called *Grafana*, which has started as a fork of Kibana and is more focused on the metrics, while Kibana supposes to be applied broader. Moreover, the Grafana requires ***explicit separation of data collection and storage***, so that could bring more issues to set all up.

Also there is an alternative to whole ELK stack called *Prometheus*. This solution is more focused on metrics only, while the ELK works with ***all types of data***, mostly logs. That is not a full alternative because of a bit different scope, but usually there is a trade off between these two solutions. I should also mention that there are some cases where Prometheus and ELK are used together.

3.2.2 Implementing ELK

Now let us briefly go through the implementation details. In the Figure 3.2 is shown a part of `build.sh` script, which describes the ELK setup. So, let us have a look there:

```
helm repo add elastic https://helm.elastic.co

helm install -n learnshell elasticsearch elastic/elasticsearch
--set service.type=LoadBalancer

helm install -n learnshell kibana elastic/kibana
--set service.type=LoadBalancer

helm install -n learnshell metricbeat elastic/metricbeat

helm install -n learnshell  filebeat elastic/filebeat
```

Figure 3.2: Implementation of ELK

- First of all, there is bunch of ***helm*** repositories, which manages the kubernetes-ready services. They need to be downloaded, otherwise there would not be possible to upload needed each separate service, or ***chart***. In this case, there is `elastic` repository used.
- Some of them are set to be a ***load balancers***. It means that they are exposed with external IP address and can be accessed from the public web.

And in the result, the logging and monitoring service will be accessible from ***EXTERNAL-IP: 5601*** address, where `EXTERNAL-IP` is a *external IP address of Kibana.*

To conclude this section, the ELK setup is not requiring lots of time and effort, but would bring lots of meaningful advantages, so this solution would be definitely a useful one.

3.3 Implementing DevOps practices

3.3.1 CI pipeline

And now that is time to show CI pipeline. It triggers on each commit into the default branch and is ***required for each individual repository***. It has the high level parts:

- ***Build stage***. This stage has two jobs to be done: *build the image* and *push it to the private Docker registry.*
- ***Deployment stage***. It has a goal to *deploy a container of the previously pushed image to the K8S cluster.* But before that it is required to create

an entity called `imagePullSecret` that will ***provide another security layer while deploying into the cluster***.

Now we have a CI pipeline that is able to automate all the steps needed to maintain solution up to date relatively fast and easy. And this is time to analyse and conclude all the job described above. So, it will be in the next chapter.

Conclusion

To conclude all the discussion described above there is a need to discuss two main things: ***what are the advantages of the solution*** and ***what are the things to be improved***. So, let us get into the main insights.

Main insights

These are the things that can be considered as a positive solution outcome. Well, there are few of them:

- Now the cluster is able to ***simply collect and analyze the information from all the nodes*** in the cluster. It was discussed above, but briefly these steps solves a huge bottleneck in the cluster's scalability.
- There is a ***CI pipeline*** provided. This resolves the issues related to the solution maintenance and operations.

These are the main insights that are showing the main advantages. Now it is time to talk about the things to be improved.

Next steps

Not everything looks perfectly at the moment. Let us go in more details:

- Right now there are at least two components of logging and monitoring (ElasticSearch and Kibana) that are exposed to the public Internet. That is a ***huge security guidelines violation***, since that is completely not secure. Moreover, ***both of the services can be accessible through http***. That is done for testing purposes only and should be definitely improved before moving the solution into production environment.

- Another thing that can be improved is a ***automation of ElasticSearch index creation***. Now it requires additional index setup, which would not completely automate the process. It is worth to mention that the process is simple and repeats over the attempts, so ***such a process must be automated*** for the better result.

Now let us sum everything up. In this text were discussed lots of concepts that are becoming more and more popular nowadays (DevOps, K8S, Docker) and how they can be applied to the given LearnShell solution. Moreover, there was done a full documentation of all the solution in the GitLab READE file. Despite the theoretical knowledge, there was also a guide to the solution architecture, design and implementation phases. So, we can claim that ***all the problems stated in the introduction were resolved and current solution would bring a positive outcome***.

Links

- GitLab repository - https://gitlab.fit.cvut.cz/ryabuily/ls-azure
- GitHub repository - https://github.com/ilya2108/ls-web
- Kibana dashboard - http://20.82.215.142:5601/
- ElasticSearch - http://20.82.215.140:9200/
- Frontend - http://20.82.211.217:3000/profile

Bibliography

[1] Simic, S. Docker Image Vs Container: The Major Differences. Available from: https://phoenixnap.com/kb/docker-image-vs-container

[2] Donohue, T. The differences between Docker, containerd, CRI-O and runc. Available from: https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/

[3] Richardson, C. Pattern: Microservice Architecture. Available from: https://microservices.io/patterns/microservices.html

[4] Richardson, C. Pattern: Monolithic Architecture. Available from: https://microservices.io/patterns/monolithic.html

[5] Kubernetes. Kubernetes Components. Available from: https://kubernetes.io/docs/concepts/overview/components/

[6] Section. Scaling Horizontally vs. Scaling Vertically. Available from: https://www.section.io/blog/scaling-horizontally-vs-vertically/

[7] Salesforce. What is Digital Transformation? Available from: https://www.salesforce.com/in/products/platform/what-is-digital-transformation/

[8] Ross, S. CAPEX vs. OPEX: What's the Difference? Available from: https://cutt.ly/svRa039.

[9] Markets; Markets. Digital Transformation Market. Available from: https://www.marketsandmarkets.com/Market-Reports/digital-transformation-market-43010479.html?gclid=CjwKCAjwjuqDBhAGEiwAdX2cj-Y_PzImMNchPP1tZfUoJidfitwc_YaGV6WbJ_xOtxSSqwO6-YQE8hoCaeIQAvD_BwE

[10] Hou, T. IaaS vs PaaS vs SaaS Enter the Ecommerce Vernacular: What You Need to Know, Examples and More. Available from: https://www.bigcommerce.com/blog/saas-vs-paas-vs-iaas/

[11] Docker. What is a Container? Available from: https://www.docker.com/resources/what-container

[12] Doerrfeld, B. 5 Container Alternatives to Docker. Available from: https://containerjournal.com/topics/container-ecosystems/5-container-alternatives-to-docker/

[13] Academy, A. C. N. Docker Alternatives. Available from: https://www.aquasec.com/cloud-native-academy/docker-container/docker-alternatives/

[14] Richardson, C. Pattern: Database per service. Available from: https://microservices.io/patterns/data/database-per-service.html

[15] Kubernetes. What is Kubernetes? Available from: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[16] Kubernetes. Nodes. Available from: https://kubernetes.io/docs/concepts/architecture/nodes/

[17] sematext. Logging vs Monitoring: How are They Different & Why You Need Both. Available from: https://sematext.com/blog/apm-vs-log-management/

[18] sematext. The Complete Guide to Metrics, Monitoring & Alerting. Available from: https://sematext.com/blog/monitoring-alerting/#toc-what-is-monitoring-2

[19] Wikipedia. DevOps. Available from: https://en.wikipedia.org/wiki/DevOps

[20] ISO. Software and systems engineering — Agile and DevOps principles and practices. Available from: https://www.iso.org/standard/79010.html

[21] Wikipedia. Agile software development. Available from: https://en.wikipedia.org/wiki/Agile_software_development

[22] Rogerson, L. Prometheus vs. ELK. Available from: https://www.metricfire.com/blog/prometheus-vs-elk/

[23] Gupta, V. Grafana vs. Kibana. Available from: https://www.metricfire.com/blog/grafana-vs-kibana/

Appendix A

Acronyms

CI	Continuous Integration
CD	Continuous Delivery
K8S	Kubernetes
ELK	ElasticSearch, LogStash and Kibana
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
SaaS	Software-as-a-Service
SLA	Service Level Agreement
CAPEX	Capital Expenses
OPEX	Operational Expenses
MVP	Minimum Viable Product
OS	Operating System
VM	Virtual Machine
MB	Megabytes
GB	Gigabyte
TB	Terabyte
DB	Database
API	Application Programming Interface
CRI	Container Runtime Interface

CPU	Central Processing Unit
RAM	Random Access Memory
REST	Representational State Transfer
PII	Personal Identifiable Information
ISO	International Organization for Standardization
JVM	Java Virtual Machine
IP	Internet Protocol

Appendix B

Contents of enclosed CD

```
readme.txt ........... the file that contains all the technical explanation
src ....................................... the directory of source codes
├── learnShell .................................. implementation sources
└── thesis .............. the directory of LaTeX source codes of the thesis
text ......................................... the thesis text directory
└── thesis.pdf ......................... the thesis text in PDF format
```