

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research



A/Mira University of Bejaia
Faculty of Exact Sciences
Computer Science Department

Master Thesis

In
Computer science

Option
Network and Security Administration

Theme

Deploying and Securing a High Availability
Kubernetes Cluster

Made by:

M^r Anis Messaoudi & *M^r* Ahmed Lounici

Supervised by:

M^r Sofiane AISSANI, *M^r* Mohand MOKTEFI & *M^r* Youcef BELATTAF

the jury is made up of:

M^r Karim AKILAL
M^r Khaled BEDJOU
M^r Sidali BELHOCINE

U. A/M Bejaia, July 2022.

Abstract

The microservices architectural style has been gaining popularity in recent years. In this architectural style, small and loosely coupled modules are deployed and scaled independently to compose cloud-native applications. Microservices are maintained and tested easily and are faster at boot time. However, to fully leverage the benefits of the architectural style of microservices, it is necessary to use technologies such as containerization. Therefore, in practice, microservices are containerized in order to remain isolated and lightweight and are orchestrated by orchestration platforms such as Docker Swarm or Kubernetes.

Therefore, in order to realize this thesis we have invested in learning the best DevOps practices and mainstream containerization technologies mentioning Docker, and orchestration tools mentioning Docker Swarm and Kubernetes.

Keywords : Virtualization; Container; Docker; Kubernetes; Orchestrator; Microservices, Cloud-Native; Security; DevOps.

Agzul

Tawsit n lbenyan n tenfa timeçtaḥ d tin yettuyalen s imal d tagdudant xersum iseggasen-a ineggura. Deg tewsit, ilmuden imeçtaḥ yettwasduklen s unammun axfifan d tid yettwasxedmen ttwarsent yef sshellum d wudem amunnan iwakken ara d-xelqen isnassen "cloud natives". Tanfiwin timeçtaḥ d tid yettwaṭfen arnu ttwaæarḍent shala eḡlent deg tazwara. Ihi, iwakken ad nefk s umata yef tewsit n lbenyan n "microservices", ilaq ad teswasxdem tetiknulugit am "conteneurisation". Arnu yur-s, deg tigawt, Tanfiwin timeçtaḥ ttwaxedment "conteneurisés" iwakken ad qqiment d tmunanin arnu fessus-it, ttwaselḥunt s tyeryert am "Docker Swarm akked Kubernetes".

Iwakken ad nessiwed ad nexdem tarist-a n nnig turagt, nefka azal i ulmad n tigawin yufraren DevOps akked tiknulugit n conteneurisation, ad nebder Docker akked ullallen n uselḥu Docker Swarm akked Kubernetes.

Iwalen n Tsarut : Virtualization; Container; Docker; Kubernetes; Orchestrator; Microservices, Cloud-Native; Security; DevOps.

Résumé

Le style architectural des microservices devient de plus en plus populaire ces dernières années. Dans ce style, des petits modules faiblement couplés sont appliqués et mis à l'échelle de manière indépendante pour créer des applications cloud natives. Les microservices sont maintenus et testés facilement et sont plus rapides au démarrage. Cependant, afin de tirer pleinement parti du style architectural des microservices, il est nécessaire d'utiliser des technologies telles que la conteneurisation. Ainsi, en pratique, les microservices sont conteneurisés pour rester isolés et légers, et sont orchestrés par des plateformes d'orchestration comme Docker Swarm et Kubernetes.

Par conséquent, afin de réaliser cette thèse, nous avons investi dans l'apprentissage des meilleures pratiques DevOps et des technologies de conteneurisations, en mentionnant Docker et les outils d'orchestration Docker Swarm et Kubernetes.

Mots clés : Virtualisation; Conteneur; Docker; Kubernetes; Orchestrateur; Microservices, Cloud-Native; Sécurité; DevOps.

ملخص

اكتسب نمط التطبيقات الدقيقة شعبية في السنوات الأخيرة. في هذا، يتم نشر وحدات صغيرة ومقترنة وتوسيع نطاقها بشكل مستقل لتأليف تطبيقات سحابية. يتم صيانة الخدمات الصغيرة واختبارها بسهولة و التي تعتبر أسرع من حيث سرعة بدء التشغيل. ومع ذلك، للاستفادة الكاملة من فوائد الخدمات الدقيقة أو التطبيقات المصغرة، من الضروري استخدام تقنيات مثل الحاويات. لذلك، من الناحية العملية، يتم وضع الخدمات الدقيقة في حاويات من أجل إبقائها معزولة وخفيفة الحجم ويتم تنظيمها بواسطة منصات تنسيق مثل Docker Swarm و Kubernetes.

لذلك، من أجل تحقيق هذه الأطروحة، استثمرنا في تعلم أفضل ممارسات DevOps وتقنيات الحاويات Docker وأدوات التنسيق Docker Swarm و Kubernetes.

الكلمات المفتاحية: المحاكاة الافتراضية؛ حاوية ؛ Docker ؛ Kubernetes ؛ منسق ؛ التطبيقات المصغرة ؛ Cloud-Native ؛ الحماية ؛ DevOps.

Acknowledgements

First of all, we would like to thank our dearest parents, siblings and friends who have stood by us and shown us an endless amount of support and encouragement.

We would like to express our deepest gratitude to our supervisor Mr. Sofiane AISSANI for accepting to supervise our work and providing us proper guidance and encouragement throughout this research as well as thank him for his patience and understanding.

We would also like to express our appreciation for co-supervisor Mr. Mohand MOKTEFI for taking time to review our work and give us thoughtful comments and recommendations during the writing of this paper.

We would also like to thank our internship supervisor and director of EURL Tech Instinct Mr. Youcef BELATTAF for taking time to guide us during these last months of work and for providing the means and resources to help us in realizing this project.

Lastly, we thank the members of the jury for accepting to examine and evaluate our work.

Contents

General introduction	14
1 Background Concepts	16
1.1 Introduction	16
1.2 Virtualization	16
1.2.1 Virtual machine	16
1.2.2 Hypervisor	16
1.2.3 Containerization	17
1.2.4 VM vs Container	18
1.3 Microservices	19
1.4 Linux	19
1.4.1 Namespaces	19
1.4.2 Cgroups	20
1.5 Container Orchestration tools	20
1.6 DevOps	21
1.6.1 CI/CD	22
1.6.2 GIT	22
1.7 High Availability	22
1.7.1 High availability cluster	22
1.8 Network File System	23
1.9 Conclusion	23
2 Modern Deployment Infrastructure and Security	24
2.1 Introduction	24
2.2 Docker	24
2.2.1 Docker tools	25
2.3 Docker Swarm	26
2.3.1 Docker Swarm components and concepts	27
2.3.2 Network in Docker Swarm	27
2.3.3 Security in Docker Swarm	28
2.3.4 Storage in Docker Swarm	28
2.4 Kubernetes	30
2.4.1 Kubernetes architecture	30
2.4.2 Kubernetes workloads	31
2.4.3 Affinity	32
2.4.4 Networking in Kubernetes	33
2.4.5 Storage in Kubernetes	34
2.4.6 Security in Kubernetes	34

2.4.7	Kubernetes installation	36
2.4.8	Kubernetes cloud providers	36
2.5	Kubernetes vs Docker Swarm	37
2.6	Conclusion	37
3	Analysis of current deployment	38
3.1	Introduction	38
3.2	Analyzing the deployment architecture	38
3.3	Services	39
3.3.1	Nginx	39
3.3.2	RabbitMQ	40
3.3.3	Postgres	41
3.3.4	Jhipster Microservices	41
3.3.5	SIRH microservices	43
3.4	ELK Stack	45
3.4.1	Elasticsearch	45
3.4.2	Logstash	45
3.4.3	Kibana	46
3.4.4	APM Server	46
3.4.5	MetricBeat	46
3.4.6	HeartBeat	47
3.5	Microservices	47
3.5.1	Postgres Admin	47
3.5.2	SIRH Backoffice	48
3.5.3	SIRH Front	48
3.5.4	SIRH Employee-Portal	48
3.6	Critics about the current deployment	48
3.7	Conclusion	49
4	Deploying and configuring the Kubernetes Cluster	50
4.1	Introduction	50
4.2	New deployment architecture	50
4.3	Cluster setup and microservices deployment	51
4.3.1	Creating a High Availability MicroK8s cluster	51
4.3.2	NFS for Persistent storage	54
4.3.3	Deploying the ECK Stack	56
4.3.4	Deploying Patroni	61
4.3.5	Deploying pgAdmin	61
4.3.6	Deploying Falco	62
4.3.7	Deploying RabbitMQ	63
4.3.8	Deploying SIRH Company-Management	64
4.3.9	Deploying SIRH Trackability	65
4.3.10	Deploying SIRH Billing	65
4.3.11	Deploying SIRH Pay	66
4.3.12	Deploying Ingress	66
4.3.13	Pushing configuration files to GitHub	70
4.4	Conclusion	71
	References	73

List of Figures

1	Tech Instinct logo.[75]	14
1.1	Hypervisor types.[98]	17
1.2	Containers.[77]	17
1.3	Docker Engine Components.[89]	18
1.4	Container vs virtual machine.[96]	19
1.5	Namespaces.[76]	20
1.6	Cgroups.[97]	20
1.7	Container orchestration tools.[67]	21
1.8	DevOps practices.[91]	22
1.9	Network File System.[108]	23
2.1	Docker architecture.[81]	25
2.2	Docker Compose.[18]	25
2.3	Docker Hub.[93]	26
2.4	Docker Swarm architecture.[105]	26
2.5	Implicit pre-Container storage.[104]	28
2.6	Explicit Shared Storage.[104]	29
2.7	Shared Multi-Host Storage.[104]	29
2.8	Container Runtime Interface.[73]	30
2.9	Kubernetes architecture.[38]	31
2.10	Kubernetes Services.[2]	33
2.11	Kubernetes Ingress.[26]	34
3.1	Tech Instinct deployment architecture.	38
3.2	Nginx architecture.[87]	39
3.3	RabbitMQ architecture.[3]	40
3.4	Jhipster microservices architecture.[4]	41
3.5	ELK Stack architecture.[40]	45
4.1	New deployment architecture with Kubernetes.	50
4.2	MicroK8s installation command.	51
4.3	Generating token for node 2.	51
4.4	Joining node 2.	52
4.5	Generating token for node 3.	52
4.6	Joining node 3.	52
4.7	Nodes list.	52
4.8	Enabling storage and dns addons.	53
4.9	Checking MicroK8s cluster and HA status.	53
4.10	adding node-name label to nodes.	53

4.11	Installing NFS server.	54
4.12	NFS directory.	54
4.13	Allowing access to the masters only.	54
4.14	Installing the CSI driver for NFS.	55
4.15	Checking driver status.	55
4.16	Creating StorageClass for NFS.	55
4.17	Setting the StorageClass.	56
4.18	Installing CRDs.	56
4.19	Installing the operator with its RBAC rules.	56
4.20	deploying Elasticsearch.	57
4.21	Elasticsearch status.	57
4.22	deploying Kibana.	57
4.23	Kibana status.	57
4.24	Kibana Web interface.	58
4.25	deploying APM.	58
4.26	APM status.	58
4.27	deploying Heartbeat.	59
4.28	Heartbeat status.	59
4.29	Heartbeat Web interface.	59
4.30	deploying Metricbeat.	60
4.31	Metricbeat status.	60
4.32	deploying Logstash.	60
4.33	Logstash status.	60
4.34	Deploying Patroni.	61
4.35	Deploying Patroni.	61
4.36	Deploying pgAdmin.	61
4.37	pgAdmin Web interface.	62
4.38	Deploying Falco.	62
4.39	Deploying Falco-sidekick.	63
4.40	Falco-sidekick Web interface.	63
4.41	Deploying RabbitMQ operator.	63
4.42	Deploying RabbitMQ cluster.	64
4.43	RabbitMQ Web interface.	64
4.44	Deploying SIRH Company-Management.	64
4.45	SIRH Company-Management status.	65
4.46	Deploying SIRH Trackability.	65
4.47	SIRH Company-Management status.	65
4.48	Deploying SIRH Billing.	65
4.49	SIRH Billing status.	66
4.50	Deploying Pay.	66
4.51	SIRH Pay status.	66
4.52	Enabling Ingress addons.	66
4.53	Deploying Ingress rules.	67
4.54	Checking Ingress status	67
4.55	SIRH Front Authentication Web interface.	67
4.56	SIRH Backoffice Web interface.	68
4.57	SIRH Employee Portal Web interface.	68
4.58	SIRH Facturation Web interface.	69

LIST OF FIGURES

4.59 SIRH Front Web interface.	69
4.60 Setting up the repository.	70
4.61 Push to the public repository.	70
4.62 Push to the public repository.	71

List of Tables

1.1	Virtual machine vs container	18
2.1	Kubernetes vs Docker Swarm	37
3.1	Nginx information.	40
3.2	RabbitMQ information.	40
3.3	Postgres information.	41
3.4	SIRH Registry information.	42
3.5	SIRH Gateway information.	42
3.6	SIRH UAA information.	42
3.7	SIRH Company-Management information.	43
3.8	SIRH Pay information.	43
3.9	SIRH Trackability information.	44
3.10	SIRH Billing information.	44
3.11	SIRH Front-Authentication information.	44
3.12	Elasticsearch information.	45
3.13	Logstash information.	45
3.14	Kibana information.	46
3.15	APM-Server information.	46
3.16	MetricBeat information.	46
3.17	HeartBeat information.	47
3.18	PostgresAdmin information.	47
3.19	SIRH Facturation information.	47
3.20	SIRH Backoffice information.	48
3.21	SIRH Front information.	48
3.22	SIRH Employee-Portal information.	48

LIST OF ABBREVIATIONS

AKS	Azure Kubernetes Service
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
APM	Application Performance Management
CD	Continuous Deployment
CI	Continuous Integration
CLI	Command Line interface
CNCF	Cloud Native Computing Foundation
CPU	Central Processing Unit
CRI	Container Runtime Interface
CRD	Custom Resource Definitions
CSI	Container Storage Interface
CSP	Cloud Service Provider
DDoS	Distributed Denial of Service
DNS	Domain Name System
DVCS	Distributed Version Control System
EKS	Elastic Kubernetes Service
GKE	Google Kubernetes Engine
ELK	Elasticsearch, Logstash, and Kibana
GUI	Graphical user interface
HA	High Availability
IP	Internet Protocol
IT	Information Technology
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IMAP	Internet Message Access Protocol
KVM	Kernel-based Virtual Machines
K8S	Kubernetes
LB	Load Balancer
NAT	Network Address Translation
NFS	Network File System
OS	Operating System
PKI	Public Key Infrastructure
PoC	Proof of Concept
POP3	Post Office Protocol
PV	Persistent Volume
PVC	Persistent Volume Claim
RAM	Random Access Memory

LIST OF ABBREVIATIONS

RBAC	Role-based access control
REST	Representational State Transfer
SaaS	Software as a service
SIRH	Système d'Information Ressources Humaines
SCTP	Stream Control Transmission Protocol
SME	small and Medium-sized Enterprises
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell Protocol
SSO	Single Sign On
STOMP	Streaming Text Oriented Messaging Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VM	Virtual Machine
VMM	Virtual Machine Monitor
vCPU	virtual centralized processing unit
VPS	Virtual Private Server
YAML	YAML Ain't Markup Language

General Introduction

Nowadays, IT technologies have continuously evolved to meet the challenges and demands of users and organizations. Containerization has changed application development and deployment entirely. IT organizations have recognized the benefits of adopting containers for their workflows because it provides businesses with a way to automate the deployment of modern applications at scale. Containers also provide a more cost-effective method of deployment, as applications run faster, consume fewer resources, and are portable and environment-agnostic, all of which translate into cost savings. The usage of containers is growing, estimates that 90% of global organizations will be running containerized applications in production by 2026. In 2020, Cloud Native Computing Foundation (CNCF) survey found that containers usage has grown 300% since 2016, in May 2021 IDC study predicted a wholesale shift to containers in three years where 80% of workloads will shift to or be created with containers and microservices. It becomes extremely difficult to manage the container lifecycle and its management when numbers increase dynamically with demand. Container orchestration platforms solved the problem by automating the scheduling, deployment, scalability, load balancing, availability, and networking of containers. Many container orchestration tools are available in the market, such as Kubernetes, Docker Swarm, OpenShift, Nomad, and others. This raises the question about what's the difference between each platform and which orchestration platform is more suitable for a business.[94]

Tech Instinct is an Algerian startup, based in Béjaïa, specializing in IT consulting and production, founded in July 2018. It supports SMEs and start-ups to provide them with the best technical advice and helps them to come up with the right solutions to meet their needs. Tech Instinct possesses expertise in IT architectures, SaaS platform development, API development, and web and mobile applications. Its strength comes from the experience of its consultants, some of whom have been formed in the largest French companies. Tech Instinct offers quality services at least equivalent to on-shore services (in France and Europe).[74]



Figure 1: Tech Instinct logo.[75]

Tech Instinct is using Docker to build and run their container images, and Docker Swarm as a container orchestration tool, because it doesn't take as much time to learn and implement as other more complex orchestration tools.

In this thesis and internship, we have analyzed and criticized the current deployment of Tech Instinct and their services and applications, in order to propose a new deployment using the mainstream and the widely used orchestration tool Kubernetes, and also while taking into consideration the high availability (HA) of services, and applying solutions for the critics.

This thesis is divided into 4 parts. After talking about containerization and introducing the internship host organization Tech Instinct in the introduction, the rest of this thesis and the main contributions are structured as follows:

Chapter 1 Backgrounds Concepts, introduces the core concepts and important background information of the research topic, we have talked about virtualization and containerization technologies and the best development architecture that is Microservices, described and compared between virtual machines and containers, and also presented the underlying technologies of containers which are Namespaces and Cgroups. Then talked about orchestration tools and DevOps practices. Finally, we have described the file protocol NFS and the concept of High Availability.

Chapter 2 Modern Deployment Infrastructure and Security, introduces the modern deployment container virtualization technology mainly Docker, and the two container orchestration tools Docker Swarm and Kubernetes, their components, concepts, storage, security, and networking within the cluster. We have finished the chapter by making a comparison between the two orchestration tools.

In chapter 3 Analyze of current deployment, we have started by analyzing the deployment of Tech Instinct, each applications and microservices. After finishing the analyze phase, we have pointed out the issues about the current deployment and explained the potential consequences.

In chapter 4 Deploying and configuring the Kubernetes Cluster, we have defined a plan for the new deployment, then we have started on deploying the microservices and the applications.

Chapter 1

Background Concepts

1.1 Introduction

To better understand the scope of our project, in this chapter we introduce the background concepts related to virtualization and containerization to describe and explain the relationship between them. We will also discuss the practices and tools that we will use in our work.

1.2 Virtualization

Virtualization is a process of dividing the resources of a computer into multiple execution environments for creating and running a virtual version of a device or resource, like servers, storage resources, networks, or operating systems, by applying one or additional concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, quality of service, emulation, and many others.[99]

1.2.1 Virtual machine

A VM or Virtual Machine is an emulation of a physical machine by using software, it works like a computer within a computer. It runs on an isolated partition of its host computer with its own CPU power, memory, operating system, and other resources. The users can run applications on VMs and use them as they normally would on their workstations[121].

1.2.2 Hypervisor

Also known as a virtual machine monitor (VMM), is a computer software or firmware that makes running multiple virtual machines (Guest Machine) on a Host Machine with different operating systems possible by virtually sharing the host machine's hardware resources such as memory and CPU.[107]

Hypervisors can be divided into two categories:

Type-1 native or bare-metal hypervisors: Like KVM and it runs directly on the host's hardware.[100]

Type-2 or hosted hypervisors: Like Virtualbox, runs as a software layer on an operating system.[100]

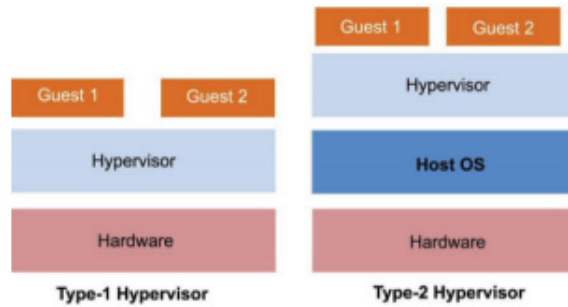


Figure 1.1: Hypervisor types.[98]

1.2.3 Containerization

Containerization is a form of virtualization, it is the packaging together of source code with its necessary components like libraries, frameworks, and other dependencies so that they are where applications run on an isolated lightweight executable called a container.[42]

1.2.3.1 Container

Container is a lightweight and portable package of software, it contains an application's code and needed configuration files, libraries and dependencies so that the application runs quickly and reliably from one computing environment to another. It runs as a process, isolated from all other processes using kernel Namespaces and Cgroups. Containers can run on local machines, VMs or even deployed on the Cloud, they can run on any Operating system.[78]

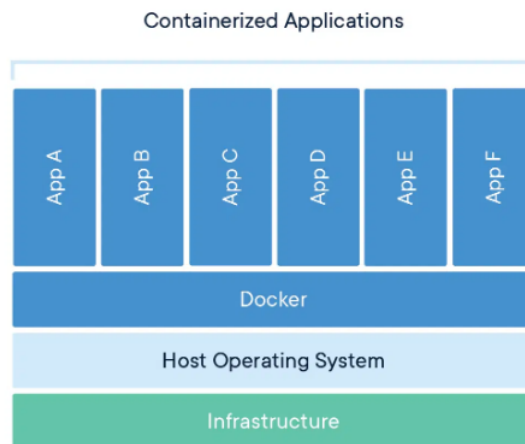


Figure 1.2: Containers.[77]

1.2.3.2 Container Runtime Engine

A container engine is a software platform that supports building and running containers based on container images. Today, the most widely known container engine is Docker, it can be installed on the host's operating system and becomes the medium for containers to share the operating system resources with other running containers on the same computing system. Container engines prepare storage to run the containers, allocate and isolate resources for use

of a container, and manage container's deployment and life-cycle. A user or an orchestrator can interact with the containers through the container engine, to give input, stop or start a container, pull or push container images from or to a repository.[90]

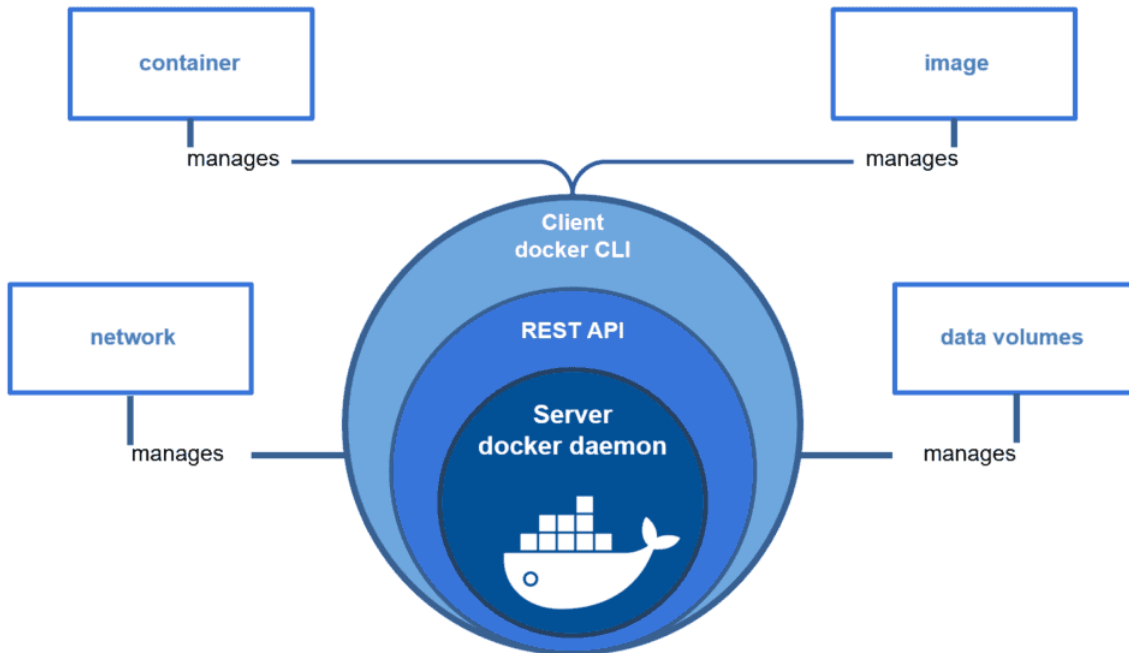


Figure 1.3: Docker Engine Components.[89]

1.2.4 VM vs Container

Virtual machines are based on hardware virtualization, whereas containers are based on OS virtualization, VMs and Containers are both “packages” that contain applications and everything they need to run, but VMs also include the operating system itself. This makes containers lightweight and faster to boot in comparison with VMs, and the fact that containers share the same OS, makes that OS and other containers vulnerable if a container gets compromised.[96]

Features	Container	Vitrual machine
Virtualization[96]	Operating System Virtualization	Hardware Virtualization
Boot time[122]	Fast	Relatively slow
Type of OS[122]	Same as Host OS	Multiple independent OS
OS isolation	Cgroups and Namespaces	Machine Isolation
Size[122]	Lightweight	à Large
Security[122]	Less secure	More secure
OS updates/upgrades[95]	On Host OS only	On Each VM

Table 1.1: Virtual machine vs container

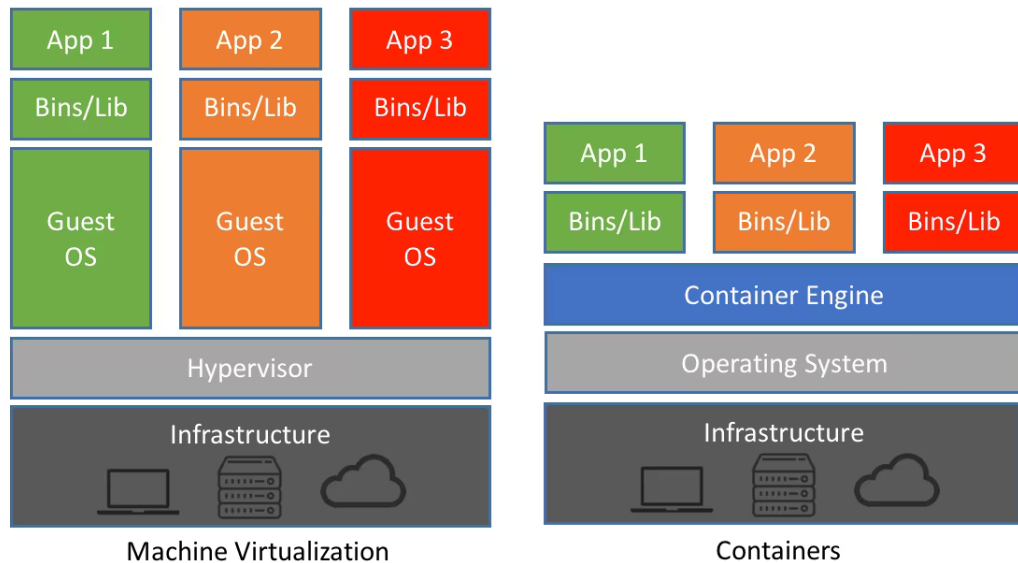


Figure 1.4: Container vs virtual machine.[96]

1.3 Microservices

Microservices are an architectural style that uses containers to develop a single application as a set of small services. Each service runs in its own process. The services communicate with clients, and often with each other, using lightweight protocols, often over messaging or HTTP. This architecture allows for each service to scale or update without disrupting other services in the application.[65]

1.4 Linux

Linux is a free open-source Unix-like operating system based on the Linux kernel, created on September 17, 1991, by Linus Torvalds. Users can modify and create variations of the source code, known as distributions such as Ubuntu, Fedora, and Arch. It's commonly used on servers, but Linux is also used for Desktop computers, smartphones, E-book readers like Kindle and gaming consoles, etc.[116]

1.4.1 Namespaces

Namespace is the underlying linux feature behind containerization technologies like Docker. It allows the system to restrict the resources that containerized processes see, and that ensures none of them can interfere with one another, thus isolating independent processes from each other. In other words, namespaces define the set of resources that a process can use. At a high level, they allow fine-grain partitioning of global operating system resources such as mounting points, network stack, and inter-process communication utilities. A powerful side of namespace is that they limit access to system resources without the running process being aware of the limitations.[117]

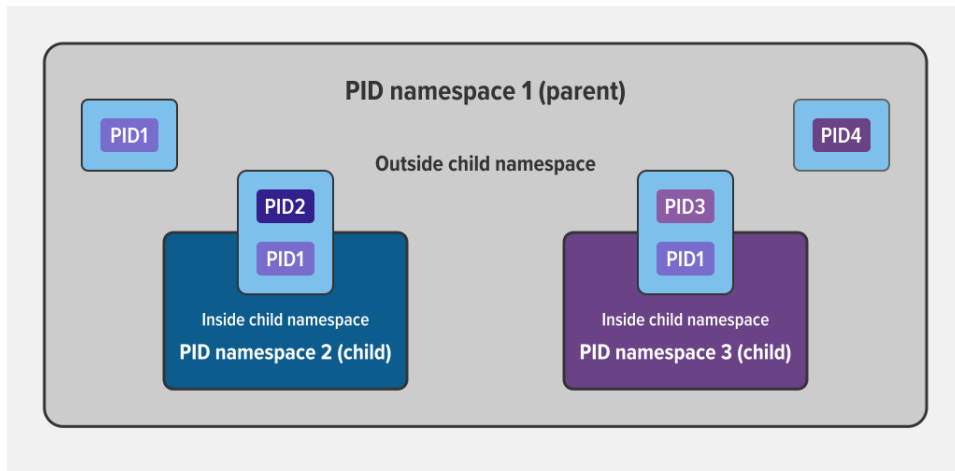


Figure 1.5: Namespaces.[76]

1.4.2 Cgroups

Control groups, usually referred to as Cgroups, are a Linux kernel feature which allow processes to be organised into hierarchical groups whose usage of various types of resources can then be limited and monitored. Cgroups are used to manage processes in many ways, such as limiting the CPU, I/O, and memory resources that are available to a process or group of process belonging to the Cgroup, change the priority of a group relative to other groups, measure a group's resource usage for accounting and billing purposes.[109]

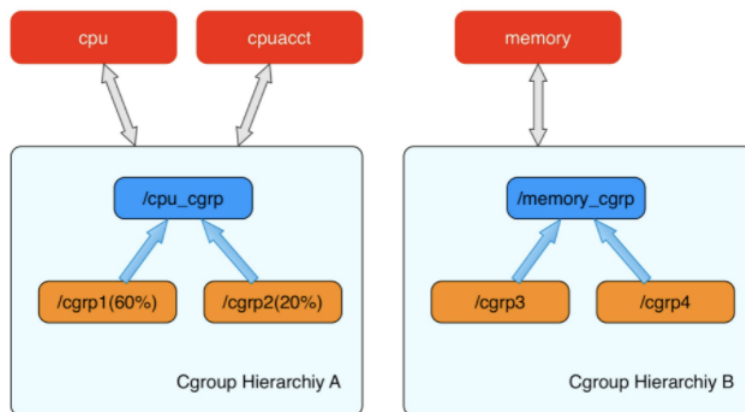


Figure 1.6: Cgroups.[97]

1.5 Container Orchestration tools

Containers can be made highly scalable, which can be created on-demand. It is good for a few containers but in the case of a cluster that consists of multiple nodes, on which tens, hundreds, or even thousands of containers are running, it becomes extremely difficult to manage the container life-cycle and its management when numbers increase dynamically with demand. Container orchestration solves the problem by automating the scheduling,

deployment, scalability, load balancing, availability, and networking of containers. Container orchestration is the automation of tasks such as:

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Load balancing and traffic routing
- Container availability
- Scaling or removing containers based on balancing workloads

Some of the well-known Container Orchestration tools are Kubernetes, OpenShift, and Docker Swarm.[88]

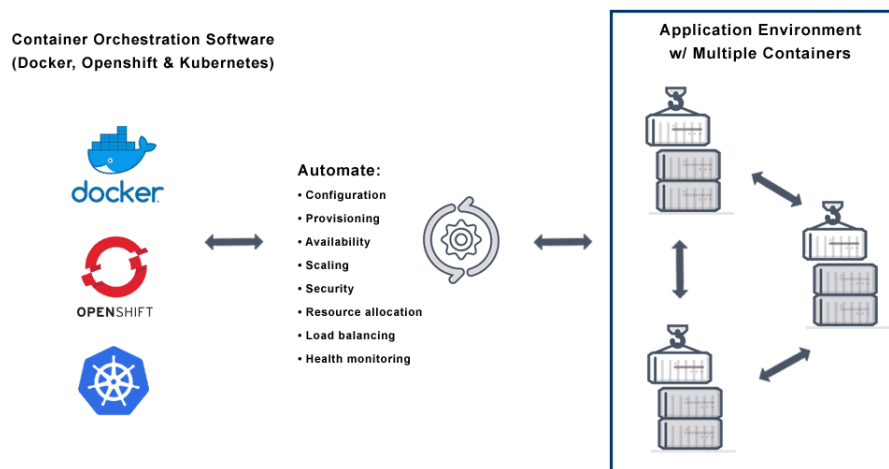


Figure 1.7: Container orchestration tools.[67]

1.6 DevOps

DevOps is a set of practices, tools that automate and integrate the processes between software development (Dev) and IT operations (Ops). DevOps to make short the systems development life cycle and provide continuous delivery with high software quality. DevOps is complementary to Agile software development, several DevOps aspects came from the Agile methodology.[111]

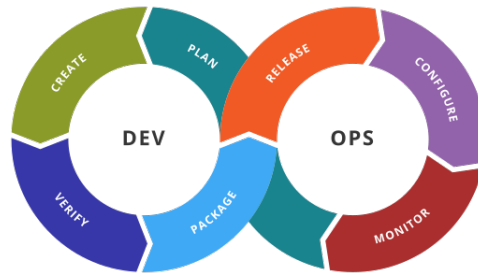


Figure 1.8: DevOps practices.[91]

1.6.1 CI/CD

CI/CD is the combined practices of Continuous Integration (CI) and Continuous Deployment (CD), they are the culture, operational principles, and set of practices that application development teams use to deliver code changes more frequently and reliably. Make it concrete. CI/CD is a DevOps and agile best practice, its automates integration and deployment, allowing software development teams to focus on meeting their business needs while ensuring code quality and software security.[110]

1.6.2 GIT

Git is a free and open-source Distributed Version Control System (DVCS), and it's a DevOps tool used to handle small to very large projects efficiently. Git is used to track changes in the source code, so you have a record of what has been done, and you can go back to specific versions that you need anytime and it enables multiple developers to work together on the same project and every developer with his own version of code, they can create branches to try to add some new features and merging those branches into the original code to implement their changes in case it was successful.[112]

1.6.2.1 GitHub

GitHub is an open-source web-based interface and cloud-based service that uses Git, It's a social networking site for programmers that many companies and organizations use to facilitate project management and collaboration.[113]

1.7 High Availability

High Availability (HA) is a characteristic that ensures that a system or application can operate continuously without any downtime or disruption and an agreed-on operational performance level is met.[114]

1.7.1 High availability cluster

Also known as HA cluster, fail-over cluster or Metrocluster Active/Active is group of computers that support server applications that can be reliably used with a minimum or null amount of down-time. If one server in a high availability cluster fails, the mission-critical app is immediately restarted on another server the moment the fault is detected.[115]

1.8 Network File System

Network File System (NFS) is a networking protocol for distributed file sharing developed by Sun Microsystems, it allows users to access files and directories located on a remote computer and perform actions like reading and writing as if they were on the local machine.[118]

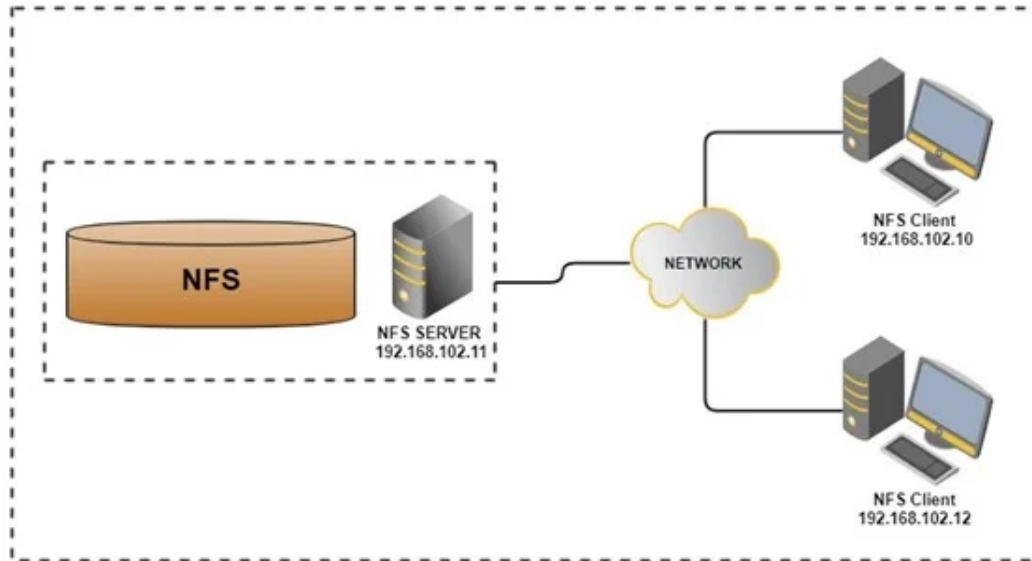


Figure 1.9: Network File System.[108]

1.9 Conclusion

In this chapter, we have presented the basics of containerization and virtualization, also we compared virtual machines and containers, and concluded that containers have an inherent advantage over VMs due to the improvement of several performance metrics such as size, storage, and boot-time. The next chapter will be devoted to talk about the modern deployment infrastructure and technologies such as Docker and container orchestration tools Docker Swarm and Kubernetes.

Chapter 2

Modern Deployment Infrastructure and Security

2.1 Introduction

Businesses around the world increasingly rely on the benefits of container technology to ease the burden of deploying and managing complex applications. Containers group all necessary dependencies within one package. They are portable, fast, secure, scalable, and easy to manage, making them the primary choice over traditional VMs. Nowadays, the most widely used container engine is Docker. But to scale containers, it needs a container orchestration tool—a framework for managing multiple containers. Today, the most prominent container orchestration platforms are Docker Swarm and Kubernetes. They both come with advantages and disadvantages, and they both serve a particular purpose. In this chapter, we will talk about Docker, also about Docker Swarm and Kubernetes, and examine both to identify which container orchestration tool is more suitable and for which scenario.

2.2 Docker

Docker (also known as Docker Engine) is a lightweight open-source software platform that was written by the team at Docker, Inc. It allows users to build, run and deploy applications quickly, and separate applications from the infrastructure. So, they can deliver software anywhere as long as Docker Engine is present since it uses OS-level virtualization to deliver software packages known as containers, which are portable and have everything an application needs to run including libraries, system tools, code, and runtime. Docker Engine is based on the client-server architecture, where the Docker client communicates with the server (Docker daemon) using a REST API. Docker daemon (Dockerd) does all the building, running, and distribution of containers.[81]

Some of the reasons to use Docker are:

- Fast, consistent delivery of your applications.
- Responsive deployment and scaling.
- Running more workloads on the same hardware.

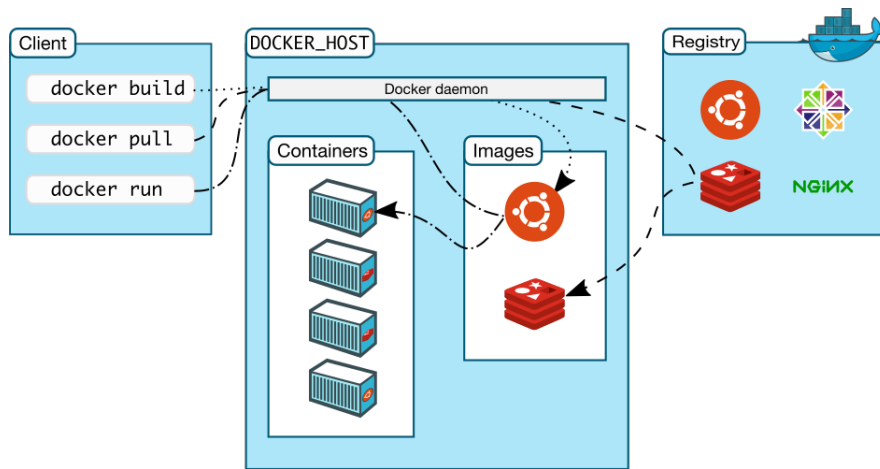


Figure 2.1: Docker architecture.[81]

2.2.1 Docker tools

2.2.1.1 Docker Desktop

Docker Desktop is an easy-to-install application for Mac and Windows environments, It's responsible for creating a Linux virtual machine to build and run containers in, it includes different Docker tools such as Docker Engine, Docker CLI, and Docker Compose. Docker Desktop is free for personal use, but a paid subscription is required for professional and business use.[79]

2.2.1.2 Docker compose

Docker Compose is a tool for defining and running multi-container Docker applications, A user can feed it a YAML configuration file (`docker-compose.yml`) that describes the services an application needs and how they interact with each other. Using Docker Compose, users can run multiple containers simultaneously and create multiple isolated environments on a single host.[84]

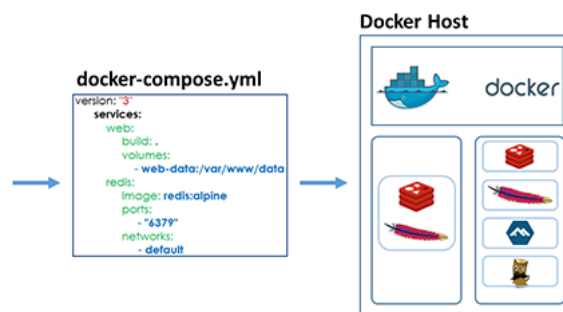


Figure 2.2: Docker Compose.[18]

2.2.1.3 Docker Hub

Docker Hub is the largest library and community provided by Docker for container images, Docker Hub is a cloud-based repository in which users and developer teams can get access to

free public repositories for storing and sharing images or can choose a subscription plan for private repositories, they can also push their own Docker images into private repositories to share between team members.[80]

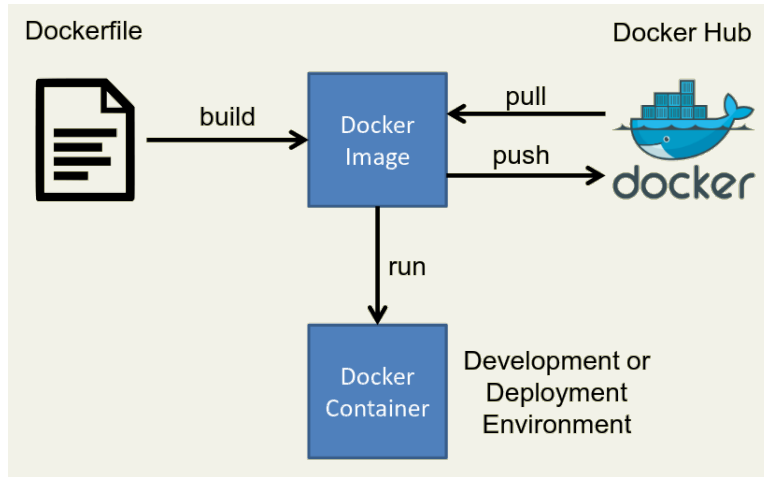


Figure 2.3: Docker Hub.[93]

2.3 Docker Swarm

Docker swarm is an open-source container orchestration and scheduling tool built and managed by Docker, It is the native clustering mode used by Docker that can be enabled to deploy and manage multiple containers across multiple machines or also called nodes which together form a cluster. One of the key benefits of using Docker Swarm is the high availability and load balancing, which means that it will make sure to distribute the workload across worker nodes and that each service will always be available.[85]

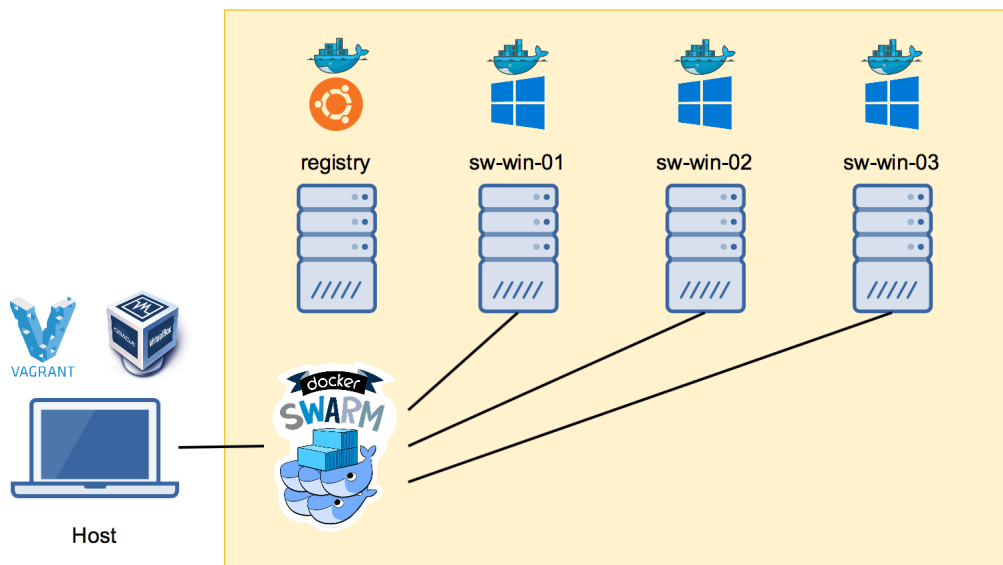


Figure 2.4: Docker Swarm architecture.[105]

2.3.1 Docker Swarm components and concepts

Below are the points for typical docker swarm components:

2.3.1.1 Nodes

A node is an instance of the Docker engine participating in the swarm, and there are two types of nodes:

- **Manager nodes**

Manager nodes are used to dispatch tasks to worker nodes, and for orchestrating and managing the functions of the swarm. a single node called the leader conducts orchestration tasks. If the leader node goes down, the other manager nodes select a new leader to resume the orchestration and the maintenance of the swarm state.[82]

The manager node consists of:

- **API:** receives commands from users and creates new services.[102]
- **Orchestrator:** takes the definition of service and creates tasks.[102]
- **Allocator:** assigns IP addresses to tasks and services.[102]
- **Scheduler:** schedules tasks and assigns them to worker nodes.[102]
- **Dispatcher :** all the worker nodes connect and respond to it. Each worker will report how many resources it has, and how many containers are run.[102]

- **Worker nodes :** receives tasks from manager nodes and execute required actions for the swarm, such as starting or stopping a container. By default, manager nodes also behave as worker nodes, but this behavior is configurable.[102]

2.3.1.2 Tasks

It carries a single Docker container and commands that define how that container will be launched and how it will work.[102]

2.3.1.3 Services

It is the definition of the tasks to execute on the manager or worker nodes.[102]

2.3.1.4 Load balancing

The Swarm manager uses ingress load balancing to expose the services to the outside.[103]

2.3.2 Network in Docker Swarm

To connect between the different containers on the different hosts in a Swarm cluster, Docker uses the overlay network, it handles the routing of each packet to and from the correct host and the correct container. When you configure a Swarm or join a Docker host to an existing Swarm, two new networks are created on that host: [86]

2.3.2.1 Ingress network

Ingress controls data traffic related to Swarm services, when a Swarm service is created and does not connect to a user-defined overlay network, it connects to the ingress network by default.[86]

2.3.2.2 Bridge network

It uses a software bridge that allows containers connected to the same bridge network to communicate while providing isolation from containers that are not connected to that bridge network.[86]

2.3.3 Security in Docker Swarm

Docker swarm manages security with the Public Key Infrastructure System (PKI), which is built into Docker, making it easy to safely deploy a container orchestration system. The nodes use Transport Layer Security (TLS) to authenticate, authorize, and encrypt the communications with other nodes in the Swarm. Role-based access control (RBAC), also known as role-based security, is used but only available in the Enterprise Edition. [83]

2.3.4 Storage in Docker Swarm

To store data in a Swarm cluster there are three types of storage to use:

2.3.4.1 Implicit pre-Container storage

It creates an implicit storage sandbox for the container, the directory `"/var/lib/docker/volumes"` will be created on the host. If the container is removed the data will be lost.[104]

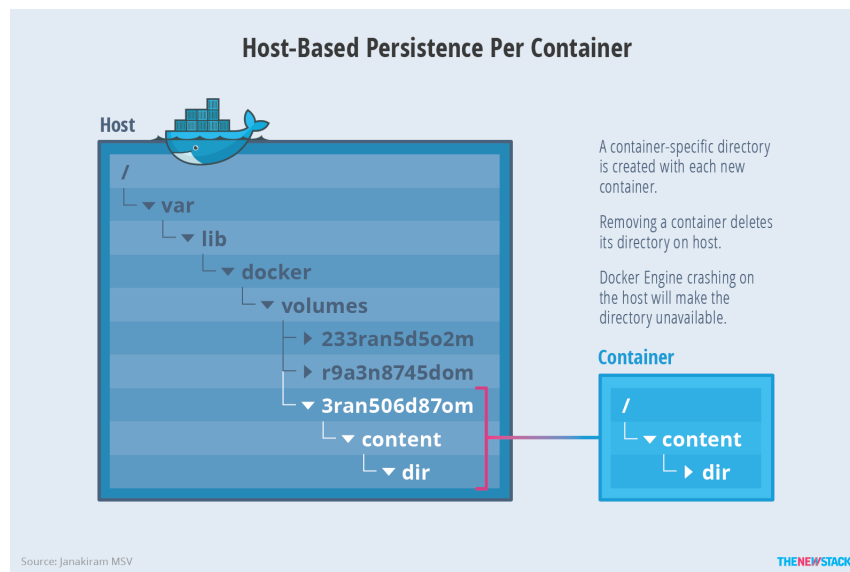


Figure 2.5: Implicit pre-Container storage.[104]

2.3.4.2 Explicit Shared Storage

It creates an explicit volume where the data will be stored, If the container is removed the data will be lost, also it can be mapped to a directory in the host, so If the container is removed the data is still on the host.[104]

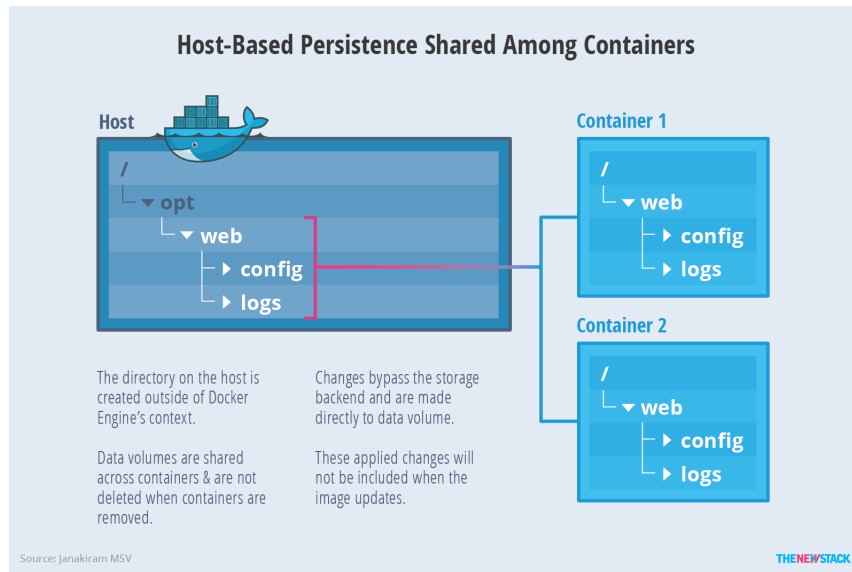


Figure 2.6: Explicit Shared Storage.[104]

2.3.4.3 Shared Multi-Host Storage

All types of storage we discussed already make the containers non-portable, the data residing on the host will not move with the container. By using distributed storage that is made available to all hosts to expose their data over the internet on shared filesystems like Ceph, GlusterFS, and Network File System (NFS).[104]

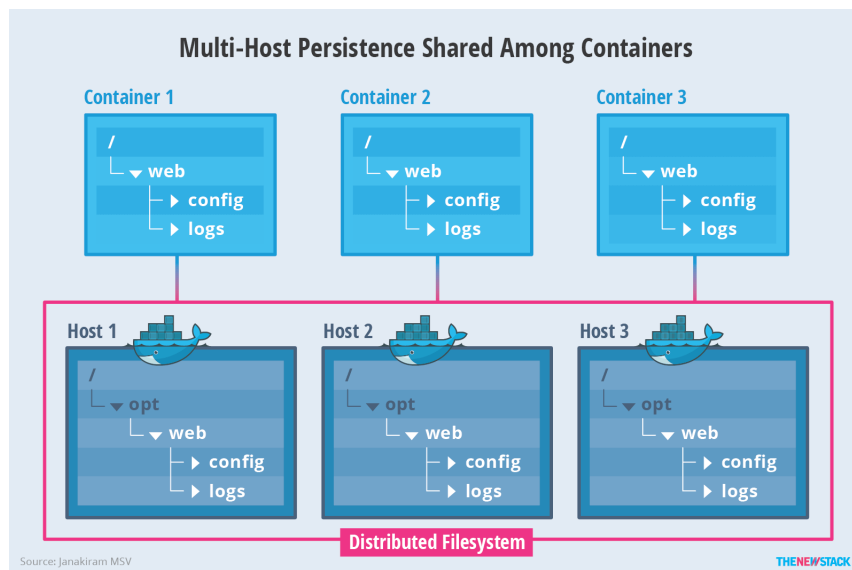


Figure 2.7: Shared Multi-Host Storage.[104]

2.4 Kubernetes

Often abbreviated as K8S, is an open-source platform container orchestration originally developed by Google, it designed to automate software deployment, scaling, and management of containerized applications.[69]

2.4.1 Kubernetes architecture

Kubernetes is an architecture that offers a couple of mechanisms for service discovery across a cluster.

2.4.1.1 Nodes

Nodes are divided into two types:

- **Worker node:** is a worker machine that runs the K8S workloads, it can be a physical machine or a virtual machine depending on the cluster. Kubernetes nodes are managed by a control plane, each node can host one or more pods.[49]

Each node runs three main components:

- **Kubelet:** is a software agent that runs on each node in the cluster, and communicates with the control plane. It allows the control plane to monitor the node.[37]
- **Container Runtime:** Kubernetes uses the Docker container as its default runtime to run the images within the pod. Kubernetes supports multiple runtimes using Container Runtime Interface(CRI) which is a plugin interface that enables kubelet to use a variety of container runtimes. There are two types of the container runtime, high-level and low-level, for example, Containerd is a high-level runtime that pushes and pulls images and manages the lifecycle of running containers by sending commands to a low-level container runtime such as runC. It is also possible for containerd to support multiple low-level container runtimes.[10]

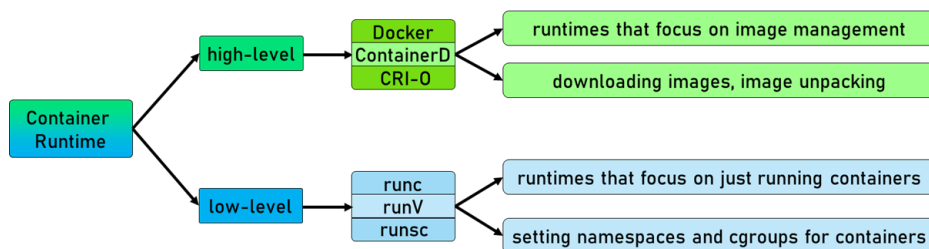


Figure 2.8: Container Runtime Interface.[73]

- **Kube-proxy:** is a network proxy that proxies the UDP, TCP, and SCTP networking of each Node, and provides load balancing. It is responsible for maintaining network rules on each node. The network rules enable network communication between nodes and pods.[35]
- **Master node:** or control plane node is a node that controls and manages a set of worker nodes and is responsible for making decisions about the cluster and pushing it towards the desired state. For a more high-availability Kubernetes cluster, two or more master

nodes can be used.[39]

It has the following components to help manage worker nodes:

- **Kube-APIserver:** acts as the frontend to the cluster. It's an entry point for all the REST commands and external communication to the cluster is via the API-Server.[33]
- **Etcd:** Is a key-value store that provides the backend database for Kubernetes. It stores and replicates the entirety of the Kubernetes cluster state.[50]
- **Kube-controller-manager:** is a daemon that manages the Kubernetes control loop, a control loop regulates the state of the system by watching the cluster state through the APIserver and makes changes to move the current state towards the desired state.[34]
- **Cloud-controller-manager:** The cloud-controller-manager runs in the control plane as a replicated set of processes The cloud-controller-manager allows the connection between the clusters and the cloud provider's API and only runs controllers specific to the cloud provider that is used.[8]
- **Kube-scheduler:** Monitors the newly created Pods without an assigned node, and selects a node that they can run on.[36]

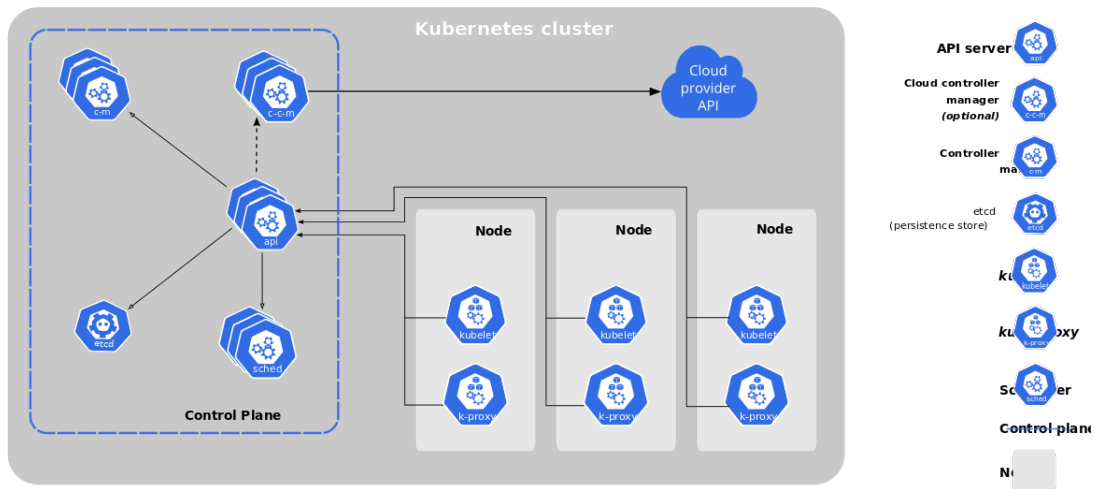


Figure 2.9: Kubernetes architecture.[38]

2.4.2 Kubernetes workloads

A workload is an application that runs on the K8S cluster, the pod is the smallest and simplest object, each pod gets its IP address with which it can interact with other pods within the cluster. Pod contains a container or more inside of it and they can communicate among themselves via localhost. Pods have a defined lifecycle, if the pods crash or stop, new pods need to be created to restore the normal state of the node. To automate the management of the pods, Kubernetes provides several built-in workload resources.[71]

2.4.2.1 ReplicaSet

ReplicaSet is a process that ensures that there is always a stable set of running pods for a particular workload. The ReplicaSet configuration defines the number of identical pods required, and if a pod stops or fails, additional pods are created to compensate for the loss.[52]

2.4.2.2 Deployment

Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods, when a Deployment is created, it creates a ReplicaSet, and then it creates pods according to the number specified. The Deployments scale the application by increasing the number of running pods or updating the running application.[16]

2.4.2.3 StatefulSet

work much as a Deployment does, it manages the deployment and scaling of a set of pods, and it guarantees the ordering and uniqueness of these pods.[57]

2.4.2.4 DaemonSet

ensures that all nodes (or some) are running exactly one copy of a pod. DaemonSets will even create the pod on new nodes that are added to the cluster. DaemonSets are exceptionally well suited for the Logs collection, node resource monitoring, and cluster storage.[12]

2.4.2.5 Operators

An Operator is an application-specific controller designed to extend the capabilities of Kubernetes and simplify and automate the packaging, deployment, and management of Kubernetes applications. Operators are clients of K8S API and they act as controllers for the Custom Resources[66]

2.4.2.6 Custom Resource Definitions

CRDs are a way to extend the Kubernetes API for use cases that are not necessarily available in a default Kubernetes installation. Like the other core Kubernetes resources, a CRD is defined as YAML. The Kubernetes API server will process CRDs as it does to any other resource, and report on the configuration content of a CRD to any authorized consumer of the Kubernetes API.[11]

2.4.3 Affinity

There are two types of affinity: Kubernetes Node affinity and Kubernetes Pod affinity.

2.4.3.1 Node affinity

Node affinity allows a Pod to specify a scheduling constraint to assign it to a node or a group of nodes.[5]

2.4.3.2 Pod Affinity

Pod affinity allows the user to specify the affinity constraint between the Pods using selectors, for example, a user might want that specific Pods always run together on the same node or that they run different nodes.[5]

2.4.4 Networking in Kubernetes

The Kubernetes network model specifies that every pod gets its IP address, containers within a pod share the pod IP address and communicate with each other, and the pods can communicate with the other pods in the cluster using pod IP addresses without using NAT.[9] The concepts and resources behind networking in Kubernetes are:

2.4.4.1 Service

It groups identical Pods together to provide a way of abstracting access to them. The group of pods backing each service is usually defined using a label selector.[56] This diagram illustrates how services do work:

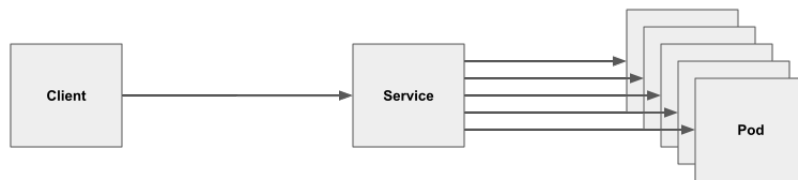


Figure 2.10: Kubernetes Services.[2]

There are four types of Services:

- **ClusterIP:** is the default service type, it allows services to be accessed only within the cluster via a virtual IP address, known as the service Cluster IP.[56]
- **NodePort:** is the most basic way to expose the services to the internet from the IP address of the node at the specified port number in the 30000-32767 range.[56]
- **LoadBalancer:** expose the service via an external Network Load Balancer. The exact type of network load balancer depends on which public cloud provider is integrated with the cluster. As long as the cloud provider supports the LB, This will be assigned to a fixed IP address in the cloud.[56]
- **ExternalName:** is a special type of service that does not have selectors and any assigned ports or endpoints. it serves as a way to return an alias to an external service residing outside the cluster.[56]

2.4.4.2 Ingress

Ingress Is an API object that gives HTTPS/HTTP routing policies to manage external users' access to the services in the cluster using particular domains or URLs, it easily configures traffic routing rules without having to create a set of load balancers or expose all services on a node. Ingress can also be used to terminate SSL / TLS before load balancing to the service.[26]



Figure 2.11: Kubernetes Ingress.[26]

2.4.4.3 DNS

DNS Is a built-in service that is launched automatically, Kubernetes cluster provides a DNS service, pods, and services are discoverable through the Kubernetes DNS service.[17]

2.4.4.4 Calico

Calico is open-source networking and network security solution for containers, virtual machines, and native host-based workloads. It enables Kubernetes workloads to communicate seamlessly and securely.[1]

2.4.5 Storage in Kubernetes

Kubernetes storage architecture is based on Volumes as a central abstraction. Volumes can be persistent or non-persistent.

Volume : Is a way to share file storage between containers in a Pod however, these Volumes are still tied to the pod lifecycle, so if the Pod gets destroyed, the volume gets destroyed with it.[64]

Persistent Volume : Unlike ordinary volumes, PVs are pieces of storage in the cluster as the nodes, their lifecycle independent of any individual pod that uses the persistent volume.[51]

Persistent Volumes Claim : Is a request from an application/user to create a persistent volume and mount it to the pods. If a Persistent Volume that meets the requirements exists or can be provisioned, the PVC will be bound to that PV.[51]

Storage Classe : Is an abstract underlying storage provider that enables dynamic storage provisioning using PVC.[58]

2.4.6 Security in Kubernetes

Kubernetes offers several built-in security features which are used to help in securing the cluster and the components.

2.4.6.1 The 4C's of cloud-native security

Those “C’s” represent the different layers that need to be secured to meet overall security goals and pass corresponding gates before exposing cloud-native applications to their customers.

- **Cloud** :The Cloud layer refers to the infrastructure that runs servers, Cloud service providers (CSPs) are responsible for setting up a secure cloud infrastructure. The most common issues found in today’s Cloud systems are misconfigurations and challenges with automation.[101]
- **Cluster** :The cluster layer consists of the Kubernetes components making up the worker nodes and control plane. There are three main cluster elements that organizations need to be concerned about: Cluster components, Cluster services, and Cluster networking.[101]
- **Container** :The container layer which is the container images contain vulnerabilities that can be scanned. The issues such as image security, the use of unknown sources, and weak privilege configurations are often overlooked by organizations. It is important to regularly update containers to reduce exposure to known vulnerabilities and to scan and audit every application running in containers.[101]
- **Code** :The code layer provides the highest level of security control, where it can restrict exposed endpoints, ports, and services to manage security risks, and also protect communication between both internal and external services using TLS encryption.[101]

2.4.6.2 Secret

A secret is a secure object which stores sensitive data, such as passwords, OAuth tokens, and SSH keys in the clusters. Storing sensitive data in Secrets is more secure than in Pod specifications or in a container image in plaintext, so by using Secrets, confidential data doesn’t need to be included in the application code.[55]

2.4.6.3 Role-based access control

RBAC is a method of granting and giving authorization to the users to access Kubernetes API resources to perform a certain action.[63]

The RBAC API have four kinds of Kubernetes object:

- **Role and ClusterRole**: roles manage the permissions within a particular namespace, so when a role is created, a namespace where it belongs needs to be specified, whereas ClusterRole is used for non-namespaced resources such as nodes.[63]
- **RoleBinding and ClusterRoleBinding**: a role binding grants the permissions that are defined in a role to users. It contains a list of subjects (users, groups, or service accounts), and a reference to the assigned role. Permissions can be granted within a namespace by using RoleBinding, or cluster-wide with a ClusterRoleBinding.[63]

2.4.6.4 ServiceAccount

In Kubernetes, service accounts are users managed by the Kubernetes API and used to provide an identity for Pods. Pods that want to interact with the API server will have to authenticate with a particular service account. By default, applications will authenticate as the default service account in the namespace they are running in.[6]

2.4.6.5 Transport Layer Security

is used as a default security configuration to encrypt and protect the network traffic in the cluster.[59]

2.4.7 Kubernetes installation

Kubernetes installation is one of the challenging topics of Kubernetes. This challenge occurs because a multitude of installation methods exists, like Minikube, Kubeadm, Microk8s, and other tools like Kubectl and Helm that used to interact and deploy on the cluster.

2.4.7.1 Microk8s

MicroK8s is a powerful, lightweight, and fully conformant Kubernetes distribution from Canonical. It's a minimalistic distribution focused on simplicity and performance by providing the functionality of core Kubernetes components, in a small footprint, scalable from a single node to a high-availability production cluster.[47]

2.4.7.2 Kubectl

Kubectl is the command-line interface tool that is installed with Microk8s and it is used to run commands to interact with Kubernetes clusters. It provides an easy way to perform tasks such as creating, managing, or deleting resources on your Kubernetes platform, Kubectl is an essential tool.[27]

2.4.7.3 Helm

Helm is a Kubernetes deployment tool for automating creation, packaging, configuration, and deployment of applications and services to Kubernetes clusters.[24]

2.4.8 Kubernetes cloud providers

The most popular and the major cloud providers provide managed Kubernetes services, such as Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), Red Hat's OpenShift and others.[60]

2.5 Kubernetes vs Docker Swarm

Features	Kubernetes	Docker Swarm
Installation[106]	Complex installation	Simple installation
GUI[106]	Kubernetes dashboard	No GUI
Scalability[106]	Fast and high scalability	Very fast and high scalability
Auto-Scaling[106]	Support auto-scaling	Does not support auto-scaling
Load Balancing[106]	Manual load balancing	Auto load balancing
Rolling Updates and Rollbacks[106]	Can deploy Rolling updates and does automatic Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks
Data Volumes [106]	Shared only with the containers in the same Pod	Shared with any container
Logging and Monitoring[106]	Built-in tools	Third-party party like ELK

Table 2.1: Kubernetes vs Docker Swarm

2.6 Conclusion

In this chapter, we have explored the two containers orchestration tool Kubernetes and Docker Swarm. Docker Swarm is a lightweight, easy-to-use orchestration tool with limited offerings compared to Kubernetes. However, Kubernetes is complex but powerful and provides self-healing, auto-scaling capabilities out of the box. Choosing an orchestration tool that is best depends on the business needs. In the next chapter, we will study and analyze Tech Instinct deployment and their microservices and applications.

Chapter 3

Analysis of current deployment

3.1 Introduction

After presenting in the previous parts the organization, and the concepts related to our thesis, and the two container orchestration tools, we dedicate this chapter to analyzing the Tech Instinct deployment such as networking, security, and storage, also their applications, and microservices in order to understand the interactions between and the workings of the infrastructure of the organization to identify the limits and issues with the current deployment, and this will help us come up with an approach to realize our migration.

3.2 Analyzing the deployment architecture

This deployment is the one used by Tech Instinct in production environment:

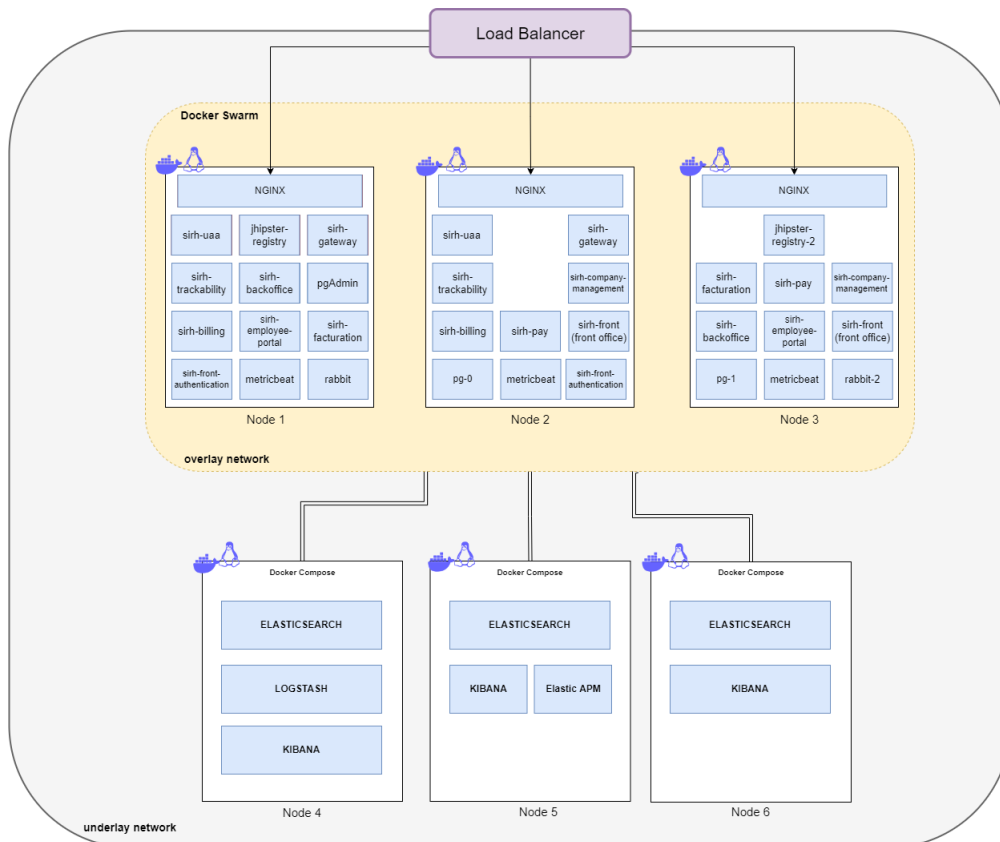


Figure 3.1: Tech Instinct deployment architecture.

All the nodes are virtual machines (VPS) running on a Linux Ubuntu server with 8 GO of RAM, 4 vCPU, and 200 GO of data storage.

The deployment architecture consists of two-part:

- Three nodes(nodes 1,2 and 3) run on the Docker Swarm cluster to manage the containers that run on it, all the VMs are connected and communicate on all the needed ports between them in a secure manner, and it can be reached from outside the cluster by using HTTPs or SSH protocols.
- Three nodes(nodes 4,5 and 6) run Docker images in the default mode and not in the Swarm mode. Each VM from the cluster is connected and communicates with all the nodes on all the needed ports, it can be reached from outside the cluster by using HTTPs or SSH protocols.

3.3 Services

In the current deployment set of services running on it:

3.3.1 Nginx

Nginx is an open-source software designed for web servers, reverse proxies, caching, load balancing, and media streaming. It also has HTTP server functionality, acting as a proxy server for email (IMAP, POP3, and SMTP) and as a reverse proxy and load balancer for HTTP, TCP, and UDP servers. It is designed for low memory usage and high concurrency. Instead of creating a new process for each web request, we use an asynchronous event-driven approach where the request is processed by a single thread.[70]

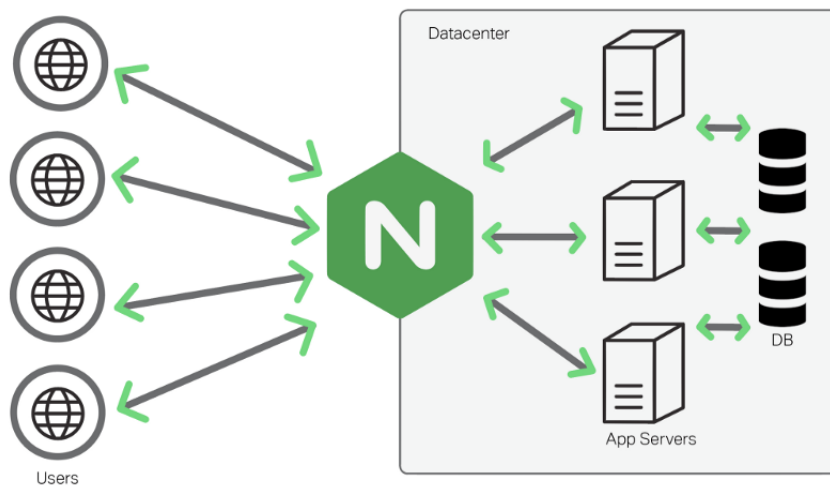


Figure 3.2: Nginx architecture.[87]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	80 : 80	HTTP
TCP	443 : 443	HTTPS	
Volumes	Host path	Container path	Desc.
	\$HOME/docker/volumes/nginx/nginx.conf	/etc/nginx/nginx.conf	nginx config
	\$HOME/docker/volumes/nginx/ssl/sirh-staging.tech-instinct.com/fullchain.pem	/etc/nginx/ssl/sirh-staging.fullchain.pem	
	\$HOME/docker/volumes/nginx/ssl/sirh-staging.tech-instinct.com/privkey.pem	/etc/nginx/ssl/sirh-staging.privkey.pem	
	\$HOME/docker/volumes/nginx/ssl/backoffice-staging.tech-instinct.com/fullchain.pem	/etc/nginx/ssl/sirhbo.staging.fullchain.pem	
	\$HOME/docker/volumes/nginx/ssl/backoffice-staging.tech-instinct.com/privkey.pem	/etc/nginx/ssl/sirhbo.staging.privkey.pem	
	\$HOME/docker/volumes/nginx/ssl/employee-staging.tech-instinct.com/fullchain.pem	/etc/nginx/ssl/employee.staging.fullchain.pem	
	\$HOME/docker/volumes/nginx/ssl/employee-staging.tech-instinct.com/privkey.pem	/etc/nginx/ssl/employee.staging.privkey.pem	
	\$HOME/docker/volumes/nginx/ssl/app.rh-partner.com/fullchain.pem	/etc/nginx/ssl/fullchain.pem	
	\$HOME/docker/volumes/nginx/ssl/app.rh-partner.com/privkey.pem	/etc/nginx/ssl/privkey.pem	
	\$HOME/docker/volumes/nginx/ssl/bo.rh-partner.com/fullchain.pem	/etc/nginx/ssl/bo.fullchain.pem	
	\$HOME/docker/volumes/nginx/ssl/bo.rh-partner.com/privkey.pem	/etc/nginx/ssl/bo.privkey.pem	
	\$HOME/docker/volumes/nginx/ssl/employee.rh-partner.com/fullchain.pem	/etc/nginx/ssl/employee.fullchain.pem	
\$HOME/docker/volumes/nginx/ssl/employee.rh-partner.com/privkey.pem	/etc/nginx/ssl/employee.privkey.pem		
Interactions	No interaction		
Node Affinity	On all the nodes		
Image	nginx:1.19.6		

Table 3.1: Nginx information.

3.3.2 RabbitMQ

RabbitMQ is a microservice known as a message-broker that supports multiple messaging protocols, such as the Advanced Message Queuing Protocol (AMQP), Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT), and other protocols.[43]

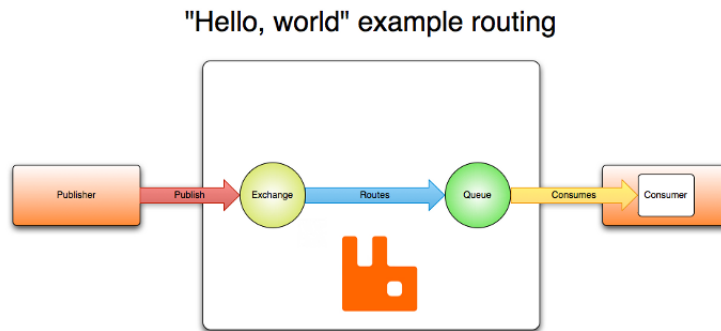


Figure 3.3: RabbitMQ architecture.[3]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	5672 : 5672	HTTP
TCP	15672 : 15672	HTTP	
Volumes	Host path	Container path	Desc.
	\$HOME/docker/volumes/rabbitmq/	/var/lib/rabbitmq	Rabbit data
	\$HOME/docker/volumes/rabbitmq/custom-config.config	/etc/rabbitmq/rabbitmq.config	Rabbit configs
	\$HOME/docker/volumes/rabbitmq/enabled_plugins	/etc/rabbitmq/enabled_plugins	Rabbit plugins
Interactions	No interactions		
Node Affinity	Node 1		
Image	rabbitmq:3.8.2-management		

Table 3.2: RabbitMQ information.

3.3.3 Postgres

Postgres is a microservice running the PostgreSQL database, Postgres is a free and open-source relational database management system, designed to work with different kinds of workloads, from single machines to data warehouses.[119]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	5433 : 5432	HTTP
Volumes	Host path	Container path	Desc.
	\$HOME/docker/volumes/postgres/	/bitnami/postgresql	Postgresql data
	\$HOME/docker/volumes/postgres/custom_conf/	/bitnami/postgresql/conf/conf.d/	Postgre configs
Interactions	No interactions		
Node Affinity	Node 3		
Image	docker.io/bitnami/postgresql-repmgr:13.1.0		

Table 3.3: Postgres information.

3.3.4 Jhipster Microservices

JHipster or Java Hipster is a free and open-source development platform to quickly generate, deploy, and develop modern web applications & microservice architectures.[30]

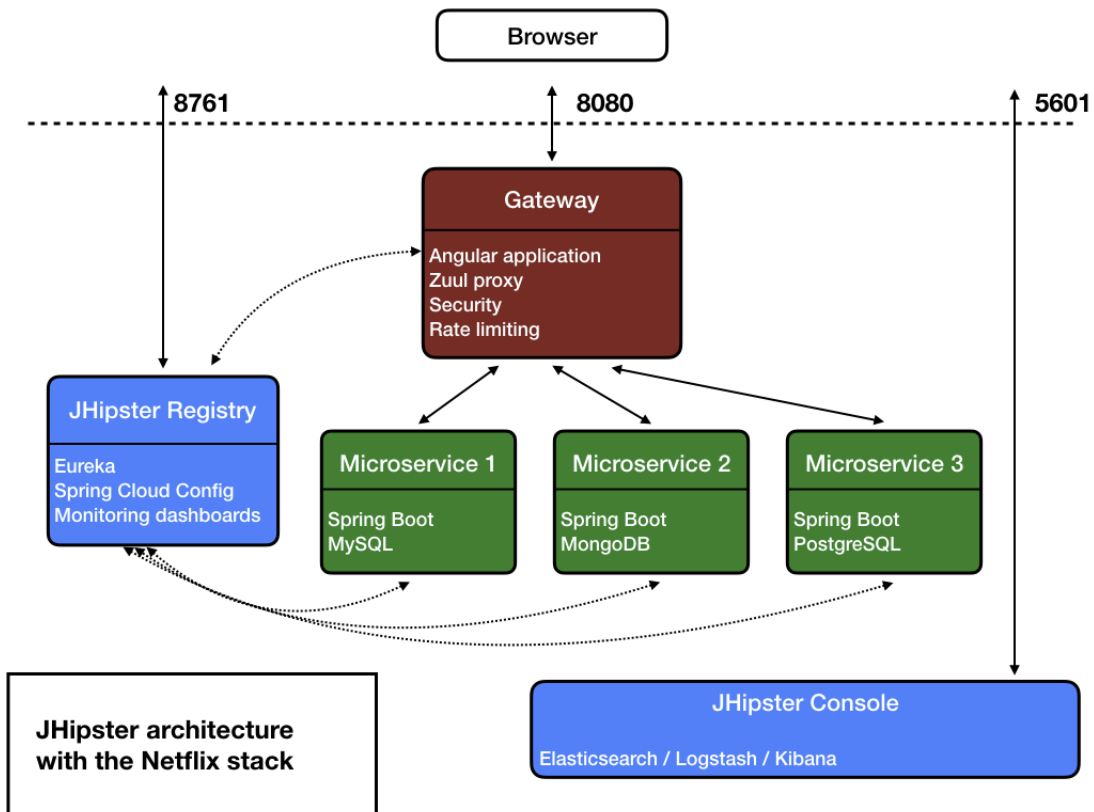


Figure 3.4: Jhipster microservices architecture.[4]

3.3.4.1 JHipster Registry

Registry is a key part of service discovery, it is a database microservice for the storage of data structures for application-level communication, also it contains the network locations of service instances.[31]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	8762 : 8761	HTTP
Volumes	No volume		
Interactions	No interactions		
Node Affinity	Node 1		
Image	jhipster/jhipster-registry:v6.3.0		

Table 3.4: SIRH Registry information.

3.3.4.2 SIRH Gateway

Gateway is a microservice that redirects and routes requests (layer 7 routings, usually HTTP requests) to the endpoints of the internal microservices.[4]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	8080 : 8080	HTTP
Volumes	No volumes		
Interactions	Service	Port	Desc.
	JHipster-Registry	8761	HTTP
	RabbitMQ	5672	HTTP
	APM-Server	8200	HTTP
	SIRH-Company-Management	8081	HTTP
	SIRH-Pay	8082	HTTP
	SIRH-Billing	8083	HTTP
Node Affinity	Not in Node 3		
Image	techinstinct/sirh-gateway:0.0.25		

Table 3.5: SIRH Gateway information.

3.3.4.3 SIRH UAA

UAA is a microservice that manages user authentication and authorization.[62]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	9999	HTTP
Volumes	No volumes		
Interactions	Service	Port	Desc.
	Postgres	5342	HTTP
	RabbitMQ	5672	HTTP
	Logstash	5000	HTTP
	APM-Server	8200	HTTP
	SIRH-Company-Management	8081	HTTP
	SIRH-Pay	8082	HTTP
	SIRH-Billing	8083	HTTP
Node Affinity	Not in Node 3		
Image	techinstinct/sirh-uaa:0.0.69		

Table 3.6: SIRH UAA information.

3.3.5 SIRH microservices

3.3.5.1 SIRH Company-Management

Company Management is an application built for businesses to manage companies, employees, establishments, and everything related to the organization of the company.

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	8081	HTTP
Volumes	No volumes		
Interactions	Service	Port	Desc.
	Postgres	5342	HTTP
	RabbitMQ	5672	HTTP
	Logstash	5000	HTTP
	APM-Server	8200	HTTP
	SIRH-UAA	9999	HTTP
	SIRH-Pay	8082	HTTP
	SIRH-Billing	8083	HTTP
	SIRH-Trackability	8084	HTTP
Node Affinity	Not on node 1		
Image	techinstinct/sirh-company-management:0.0.190		

Table 3.7: SIRH Company-Management information.

3.3.5.2 SIRH Pay

Pay is an application that manages payroll, social and tax returns, reports, and exports.

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	8081	HTTP
Volumes	No volumes		
Interactions	Service	Port	Desc.
	Postgres	5342	HTTP
	RabbitMQ	5672	HTTP
	Logstash	5000	HTTP
	APM-Server	8200	HTTP
	SIRH-UAA	9999	HTTP
	SIRH-Company-Management	8081	HTTP
	SIRH-Billing	8083	HTTP
	SIRH-Trackability	8084	HTTP
Node Affinity	Not on node 1		
Image	techinstinct/sirh-pay:0.0.191		

Table 3.8: SIRH Pay information.

3.3.5.3 SIRH Trackability

Trackability is an application that records the tracks and all the events that happen on the platform.

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	8084	HTTP
Volumes	No volumes		
Interactions	Service	Port	Desc.
	Postgres	5342	HTTP
	RabbitMQ	5672	HTTP
	Logstash	5000	HTTP
	APM-Server	8200	HTTP
	SIRH-UAA	9999	HTTP
	Elasticsearch	9200	HTTP
	SIRH-Company-Management	8081	HTTP
	SIRH-Pay	8082	HTTP
SIRH-Billing	8083	HTTP	
Node Affinity	Not on node 3		
Image	techinstinct/sirh-trackability:0.0.3		

Table 3.9: SIRH Trackability information.

3.3.5.4 SIRH Billing

SIRH Billing is an application built for commercial management (invoicing, stocks, products, purchases, etc).

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	8081	HTTP
Volumes	No volumes		
Interactions	Service	Port	Desc.
	Postgres	5342	HTTP
	RabbitMQ	5672	HTTP
	Logstash	5000	HTTP
	APM-Server	8200	HTTP
	SIRH-UAA	9999	HTTP
	SIRH-Company-Management	8081	HTTP
	SIRH-Pay	8082	HTTP
	SIRH-Trackability	8084	HTTP
Node Affinity	Not on node 3		
Image	techinstinct/sirh-billing:0.0.139		

Table 3.10: SIRH Billing information.

3.3.5.5 SIRH Front-Authentication

SIRH Front-Authentication is A web SSO system allows a user to log in using the SSO web service with one set of credentials for authentication, which are unique usernames and passwords. Then, this authentication allows them to access many other web-based applications and password-protected websites.

Protocol	Port (Published:Target)	Desc.
TCP	4202 : 80	HTTP
No volumes		
No interactions		
Not on node 3		
techinstinct/sirh-front-authentication:0.0.8		

Table 3.11: SIRH Front-Authentication information.

3.4 ELK Stack

The ELK Stack is a collection of three open-source products Elasticsearch, Logstash, and Kibana.

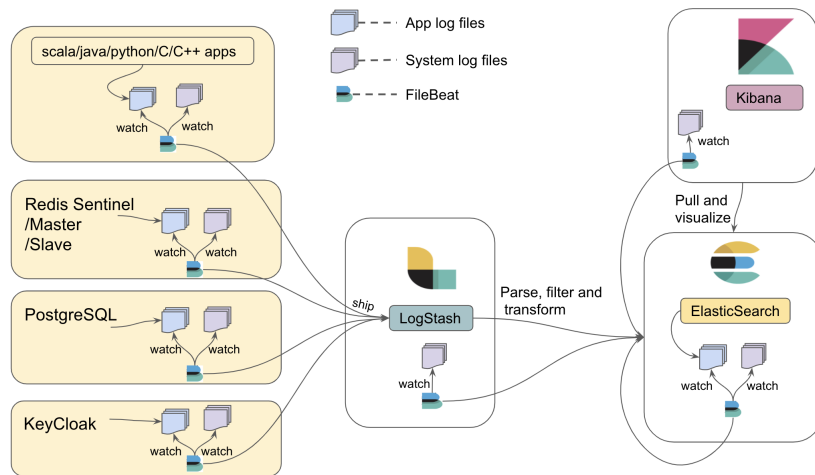


Figure 3.5: ELK Stack architecture.[40]

3.4.1 Elasticsearch

Elasticsearch is An open-source search and analysis engine that can quickly store, search, and analyze large amounts of data in near real-time and return answers in milliseconds.[68]

Networking	Protocol	Port (Published:Target)		Desc.
	TCP	9200 : 9200		REST API
	TCP	9300 : 9300		Nodes communication
Host path		Container path		Desc.
Volumes	\$HOME/docker/es01/data	/usr/share/elasticsearch/data		elasticsearch data
	\$HOME/docker/es01/config/elasticsearch.yml	/usr/share/elasticsearch/config/elasticsearch.yml		elasticsearch.yml file
	\$HOME/docker/es01/config/elasticsearch.keystore	/usr/share/elasticsearch/config/elasticsearch.keystore		Secrets keystore for secure settings
	\$HOME/docker/es01/config/es01.p12	/usr/share/elasticsearch/config/es01.p12		elasticsearch configs
Interactions	No interactions			
Node Affinity	Node 4 / Node 5 / Node 6			
Image	docker.elastic.co/elasticsearch/elasticsearch:7.9.0			

Table 3.12: Elasticsearch information.

3.4.2 Logstash

Logstash is a lightweight, open-source, server-side data processing pipeline that collects data from a variety of sources and manages events and logs.[41]

Networking	Protocol	Port (Published:Target)		Desc.
	TCP	5000 : 5000		HTTP
Host path		Container path		Desc.
Volumes	\$HOME/docker/logstash/config/logstash.yml	/usr/share/logstash/config/logstash.yml		logstash configs
	\$HOME/docker/logstash/config/sirh-conf.conf	/usr/share/logstash/config/sirh-conf.conf		sirh configs
	\$HOME/docker/logstash/config/logstash.keystore	/usr/share/logstash/config/logstash.keystore		Secrets keystore for secure settings
	"\$HOME/docker/logstash/config/ca.pem	/usr/share/logstash/config/ca.pem		Certificates authorities
Interactions	No interactions			
Node Affinity	Node 4			
Image	docker.elastic.co/logstash/logstash:7.9.0			

Table 3.13: Logstash information.

3.4.3 Kibana

Kibana is a free and open frontend application that works as data visualization dashboard software for Elasticsearch.[32]

Networking	Protocol		Port (Published:Target)		Desc.
		TCP		5601	
	Host path		Container path		Desc.
Volumes	\$HOME/docker/kibana/config		/usr/share/kibana/config		kibana configs
	\$HOME/docker/kibana/data		/usr/share/kibana/data		kibana data
	\$HOME/docker/kibana/plugins		/usr/share/kibana/plugins		kibana plugins
Interactions	No interactions				
Node Affinity	Node 4 / Node 5 / Node 6				
Image	docker.elastic.co/kibana/kibana:7.8.0				

Table 3.14: Kibana information.

3.4.4 APM Server

APM-Server is an application performance monitoring system built on the Elastic Stack. It allows tracking of key performance-related information such as requests, responses, database transactions, errors, etc.[20]

Networking	Protocol		Port (Published:Target)		Desc.
		TCP		5601	
	Host path		Container path		Desc.
Volumes	\$HOME/docker/kibana/config		/usr/share/kibana/config		kibana configs
	\$HOME/docker/kibana/data		/usr/share/kibana/data		kibana data
	\$HOME/docker/kibana/plugins		/usr/share/kibana/plugins		kibana plugins
Interactions	No interactions				
Node Affinity	Node 4 / Node 5 / Node 6				
Image	docker.elastic.co/kibana/kibana:7.8.0				

Table 3.15: APM-Server information.

3.4.5 MetricBeat

MetricBeat is a microservice that periodically collects metrics from the operating system and services running on the server. It takes the collected statistics and sends them to the specified output, such as Elasticsearch or Logstash.[44]

Networking	Protocol		Port (Published:Target)		Desc.
				No ports	
	Host path		Container path		Desc.
Volumes	\$HOME/docker/volumes/metricbeat/metricbeat.yml		/usr/share/metricbeat/metricbeat.yml:ro		metricbeat configs
	\$HOME/docker/volumes/metricbeat/metricbeat.keystore		/usr/share/metricbeat/data/metricbeat.keystore		Secrets keystore for secure settings
	\$HOME/docker/volumes/metricbeat/cert/		/usr/share/metricbeat/cert		metricbeat certificates
	/proc:/hostfs/proc:ro		/hostfs/proc:ro		monitor the Docker host
	/sys/fs/cgroup		/hostfs/sysfs/cgroup:ro		monitor the Docker host
	/		/hostfs:ro		report on disk usage
Interactions	No interactions				
Node Affinity	On all the nodes				
Image	docker.elastic.co/beats/metricbeat:7.10.1				

Table 3.16: MetricBeat information.

3.4.6 HeartBeat

Heartbeat is a lightweight daemon that is used to periodically check the status of the services and determines whether they are available and reachable.[23]

Networking	Protocol	Port (Published:Target)	Desc.
		No ports	
	Host path	Container path	Desc.
Volumes	\$HOME/docker/heartbeat/heartbeat.yml	/usr/share/heartbeat/heartbeat.yml:ro	heartbeat yml file
	\$HOME/docker/heartbeat/monitors.d/	/usr/share/heartbeat/monitors.d/	monitoring configs
	\$HOME/docker/heartbeat/cert/	/usr/share/heartbeat/cert/	heartbeat certificates
	\$HOME/docker/heartbeat/heartbeat.keystore	/usr/share/heartbeat/data/heartbeat.keystore	Secrets keystore for secure settings
Interactions	No interactions		
Node Affinity	Node 1 / Node 2 / Node 3 / Node 4 / Node 5 / Node 6		
Image	docker.elastic.co/beats/heartbeat:7.10.1		

Table 3.17: HeartBeat information.

3.5 Microservices

In the current deployment set of services running on it.

3.5.1 Postgres Admin

PgAdmin is a web-based GUI microservice used to interact with the Postgres database sessions and to connect the two instances of postgres that we have on the current deployment architecture.[92]

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	53350 : 53350	HTTP
	Host path	Container path	Desc.
Volumes	\$HOME/docker/volumes/pgadmin	/var/lib/pgadmin	PstotgresAdmin data
Interactions	No interactions		
Node Affinity	Node 1		
Image	dpage/pgadmin4:4		

Table 3.18: PostgresAdmin information.

3.5.1.1 SIRH Facturation

SIRH Facturation is an an AngularJS web application.

Protocol	Port (Published:Target)	Desc.
TCP	4201 : 80	HTTP
No volumes		
No interactions		
Not on node 2		
techinstinct/sirh-facturation:0.0.320		

Table 3.19: SIRH Facturation information.

3.5.2 SIRH Backoffice

SIRH Backoffice is an an AngularJS web application.

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	4401 : 80	HTTP
Volumes	No volumes		
Interactions	No interactions		
Node Affinity	Not on node 2		
Image	techinstinct/sirh-backoffice:0.0.104		

Table 3.20: SIRH Backoffice information.

3.5.3 SIRH Front

SIRH Front is an an AngularJS web application.

Networking	Protocol	Port (Published:Target)	Desc.
	TCP	4400 : 80	HTTP
Volumes	No volumes		
Interactions	No interactions		
Node Affinity	Not on node 1		
Image	techinstinct/sirh-front:0.0.489		

Table 3.21: SIRH Front information.

3.5.4 SIRH Employee-Portal

SIRH Employee-Portal is an an AngularJS web application.

Protocol	Port (Published:Target)	Desc.
TCP	4200 : 80	HTTP
No volumes		
No interactions		
Not on node 2		
techinstinct/sirh-employee-portal:0.0.86		

Table 3.22: SIRH Employee-Portal information.

3.6 Critics about the current deployment

- **Running older versions of applications and services**

Using older versions of applications and services can lead to crashes, decreased productivity, and cybersecurity vulnerabilities which can lead to security breaches.

- **Running multiple different instances of Postgres can cause data loss**

Two instances of Postgres running in master-slave mode lead to data loss if the master instance or both instances stop since the slave instance takes on services in read-only mode.

- **Running multiple different instances of Rabbitmq can cause message loss**

Only two instances of Rabbitmq running can cause message loss if both instances go down

- **Using storage from working nodes for containers**

Using local storage (on machines) for each container instance puts pressure on the network and consumes bandwidth while replicating data between different container instances on different nodes, and it can also cause data incoherence.

- **No mechanism to detect intrusions and report them**

It's true that the different applications and services are using secure protocols and strong passwords, but that does not exclude the possibility of someone intruding into the system, so having a mechanism in place to monitor the system and detect such intrusions is critical.

- **Usage of plaintext passwords in configuration files**

Passwords are used in plaintext in configuration files which can be stolen if someone gets access to the files.

3.7 Conclusion

After having completed our study and analysis of the current deployment of microservices and applications, we have identified the conditions to be met, and also we have described some of the issues that should be fixed in the new deployment. In the next chapter, we will plan our solution and put it into action.

Chapter 4

Deploying and configuring the Kubernetes Cluster

4.1 Introduction

In the previous chapter, we have analyzed and studied the current deployment, and defined the conditions that we will take into consideration in the new deployment. In this chapter, we will set a plan for the new deployment, after that we will start deploying the microservices and the applications.

4.2 New deployment architecture

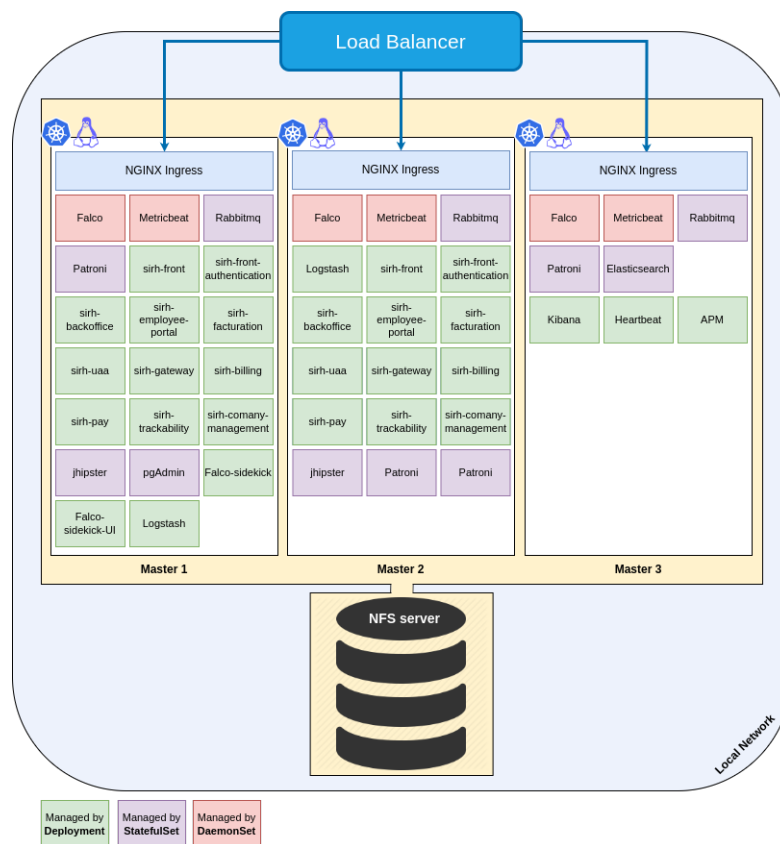


Figure 4.1: New deployment architecture with Kubernetes.

This diagram represents the new deployment architecture, where we have used 4 VMs (VPS) with 4 vCPU and 8 GO of RAM running on Ubuntu server 20.04 LTS, 3 of the machines are dedicated to running the cluster which means the total of its resources is 12 vCPU and 24 GO of RAM, and one is used as centralized storage using NFS protocol, with a storage capacity of 200 GO.

4.3 Cluster setup and microservices deployment

4.3.1 Creating a High Availability MicroK8s cluster

High availability add-on gets enabled on MicroK8s by default for a cluster that consists of three master nodes or more.[46]

4.3.1.1 Setting up the first node

MicroK8s installs as a snap package by running a single installing command on each node.[46] Snap is a software packaging and deployment system developed by Canonical for operating systems that use the Linux kernel.[120]

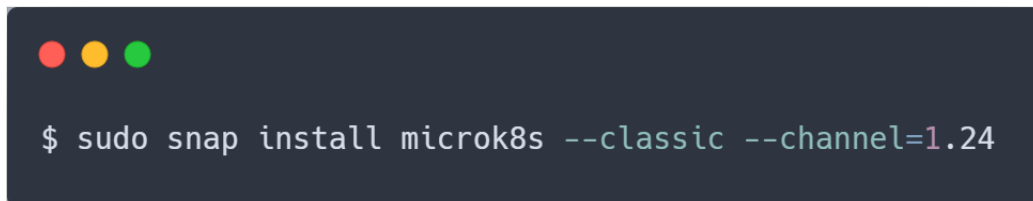
A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal shows a shell prompt followed by the command: `$ sudo snap install microk8s --classic --channel=1.24`

Figure 4.2: MicroK8s installation command.

4.3.1.2 Adding master nodes to the cluster

After installing MicroK8s on nodes 2 and 3 in the same way as in node 1 we add them to the cluster.[46]

- **Node 2 :** On the node 1, we run this command to generate a token:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal shows a shell prompt followed by the command: `$ microk8s add-node`. Below the command, the terminal displays the following text: `From the node you wish to join to this cluster, run the following:` followed by a code block: `microk8s join 1[REDACTED].144.167:25000/0b3d6f9b114fd5d7dbb4c0eb24805be0/ee9c56507ce2`

Figure 4.3: Generating token for node 2.

It returns a command with a joining token which should be executed on the MicroK8s node that we wish to join the cluster.[46]

```

$ microk8s join 1[REDACTED]2.144.167:25000/0b3d6f9b114fd5d7dbb4c0eb24805be0/ee9c56507ce2
Contacting cluster at 1[REDACTED]2.144.167
Waiting for this node to finish joining the cluster. . . .

```

Figure 4.4: Joining node 2.

- **Node 3** : By following the same previous steps used to add node 2, we add node 3.[46]

```

$ microk8s add-node
From the node you wish to join to this cluster, run the following:
microk8s join [REDACTED].144.167:25000/18f74f456f90a072bb76f4574f07cef2/ee9c56507ce2

```

Figure 4.5: Generating token for node 3.

```

$ microk8s join 1[REDACTED]2.144.167:25000/18f74f456f90a072bb76f4574f07cef2/ee9c56507ce2
Contacting cluster at 1[REDACTED]2.144.167
Waiting for this node to finish joining the cluster. . . .

```

Figure 4.6: Joining node 3.

To verify that the nodes are joined the cluster[46], we run this command:

```

$ microk8s kubectl get nodes

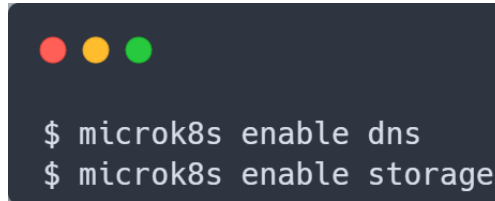
```

NAME	STATUS	ROLES	AGE	VERSION
vmi893702.[REDACTED].net	Ready	<none>	68s	v1.24.0-2+59bbb3530b6769
vmi893701.[REDACTED].net	Ready	<none>	3m48s	v1.24.0-2+59bbb3530b6769
vmi893700.[REDACTED].net	Ready	<none>	3d2h	v1.24.0-2+59bbb3530b6769

Figure 4.7: Nodes list.

4.3.1.3 Enabling storage and DNS add-ons

To enable storage and DNS on the cluster [46], we should run those command:

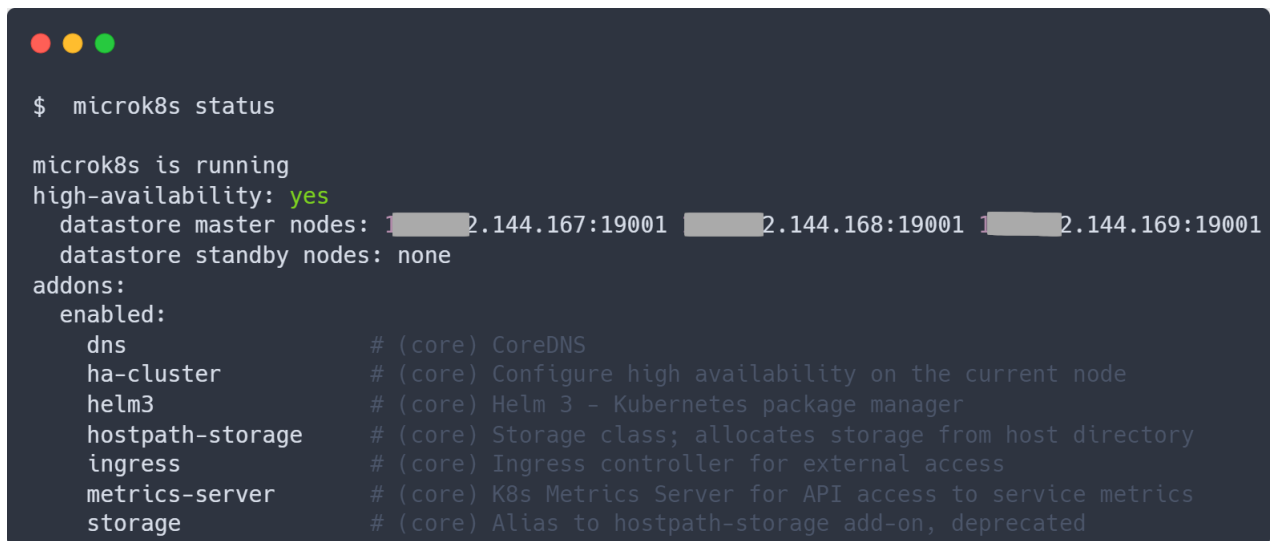


```
$ microk8s enable dns
$ microk8s enable storage
```

Figure 4.8: Enabling storage and dns addons.

4.3.1.4 Checking MicroK8s cluster and HA status

After a restart of the cluster, to check the cluster status and whether High Availability is set [46], we run the following command:



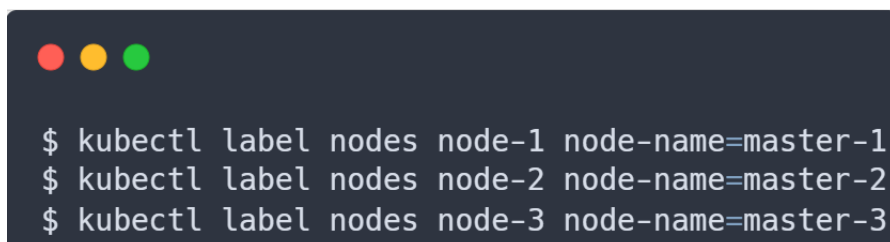
```
$ microk8s status

microk8s is running
high-availability: yes
  datastore master nodes: 2.144.167:19001 2.144.168:19001 2.144.169:19001
  datastore standby nodes: none
addons:
  enabled:
    dns # (core) CoreDNS
    ha-cluster # (core) Configure high availability on the current node
    helm3 # (core) Helm 3 - Kubernetes package manager
    hostpath-storage # (core) Storage class; allocates storage from host directory
    ingress # (core) Ingress controller for external access
    metrics-server # (core) K8s Metrics Server for API access to service metrics
    storage # (core) Alias to hostpath-storage add-on, deprecated
```

Figure 4.9: Checking MicroK8s cluster and HA status.

4.3.1.5 Setting node-name label for the nodes

Labeling nodes will help us in scheduling pods on specific nodes using Node affinity properties. [46]



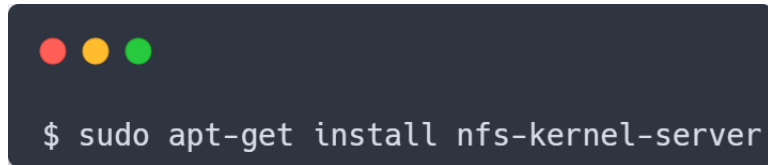
```
$ kubectl label nodes node-1 node-name=master-1
$ kubectl label nodes node-2 node-name=master-2
$ kubectl label nodes node-3 node-name=master-3
```

Figure 4.10: adding node-name label to nodes.

4.3.2 NFS for Persistent storage

4.3.2.1 Setting up the NFS server

We install the `nfs-kernel-server` package[48] by running the command shown below:

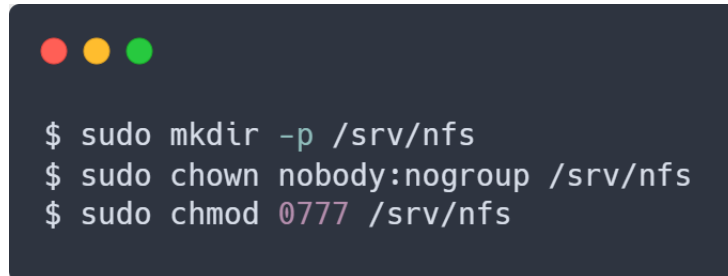
A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The command `$ sudo apt-get install nfs-kernel-server` is entered and displayed in white text.

```
$ sudo apt-get install nfs-kernel-server
```

Figure 4.11: Installing NFS server.

4.3.2.2 Creating an NFS Export Directory

Create a directory that is shared between client nodes, then remove any restrictions in directory permissions, and grant read, write, and execute permissions for all the contents inside the directory.[48]

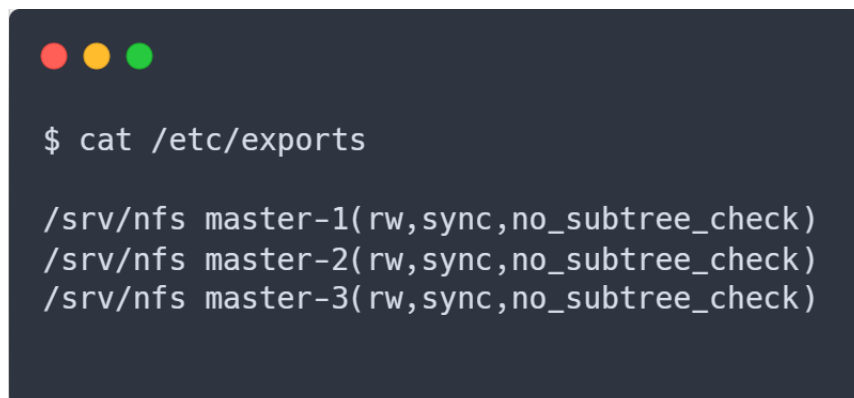
A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. Three commands are entered and displayed in white text:

```
$ sudo mkdir -p /srv/nfs
$ sudo chown nobody:nogroup /srv/nfs
$ sudo chmod 0777 /srv/nfs
```

Figure 4.12: NFS directory.

4.3.2.3 Allow the masters to mount the directory

Making sure that the nodes are allowed to mount this share. Node names are associated to their IP addresses in the `/etc/hosts` file.[48]

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The command `$ cat /etc/exports` is entered and displayed in white text. The output shows three lines of export configurations for the `/srv/nfs` directory, each allowing access to a master node with read-write permissions and no subtree check.

```
$ cat /etc/exports

/srv/nfs master-1(rw,sync,no_subtree_check)
/srv/nfs master-2(rw,sync,no_subtree_check)
/srv/nfs master-3(rw,sync,no_subtree_check)
```

Figure 4.13: Allowing access to the masters only.

4.3.2.4 Install the CSI driver for NFS

Deploy the NFS provisioner using the official Helm chart, then installing the Helm chart under the kube-system namespace.[48]

```

$ microk8s helm3 repo add csi-driver-nfs https://raw.githubusercontent.com/kubernetes-csi/csi-driver-nfs/master/charts

$ microk8s helm3 repo update

$ microk8s helm3 install csi-driver-nfs csi-driver-nfs/csi-driver-nfs \
  --namespace kube-system \
  --set kubeletDir=/var/snap/microk8s/common/var/lib/kubelet

```

Figure 4.14: Installing the CSI driver for NFS.

4.3.2.5 Checking the driver status

After deploying the Helm chart, we check the driver status by running a kubectl command.[48]

```

$ kubectl get pods --selector app.kubernetes.io/name=csi-driver-nfs --namespace kube-system

```

NAME	READY	STATUS	RESTARTS	AGE
csi-nfs-node-wnjrw	3/3	Running	0	6m52s
csi-nfs-node-5zlbz	3/3	Running	0	6m52s
csi-nfs-controller-75d6c9589d-tvqc7	3/3	Running	0	6m54s
csi-nfs-node-7bnb5	3/3	Running	0	6m54s

Figure 4.15: Checking driver status.

4.3.2.6 Creating a StorageClass for NFS

Creating a Kubernetes StorageClass to use the provisioner nfs.csi.k8s.io CSI driver, and applying it.[48]

```

$ cat nfs-storageclass.yml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-csi
provisioner: nfs.csi.k8s.io
parameters:
  server: nfs-server
  share: /srv/nfs
reclaimPolicy: Delete
volumeBindingMode: Immediate
mountOptions:
  - hard
  - nfsvers=4.1

$ kubectl apply -f nfs-storageclass.yml

```

Figure 4.16: Creating StorageClass for NFS.

After that, We set the nfs-csi StorageClass to be the default one by running the kubectl command.[48]

```
$ kubectl patch storageclass nfs-csi -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Figure 4.17: Setting the StorageClass.

4.3.3 Deploying the ECK Stack

4.3.3.1 Installing the CRDs

Custom Resource Definitions will be installed by running the following command:[15]

```
$ kubectl create -f https://download.elastic.co/downloads/eck/2.2.0/crds.yaml
customresourcedefinition.apiextensions.k8s.io/agents.agent.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/apmservers.apm.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/beats.beat.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/elasticmapsservers.maps.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/elasticsearches.elasticsearch.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/enterprisesearches.enterprisesearch.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/kibanas.kibana.k8s.elastic.co created
```

Figure 4.18: Installing CRDs.

4.3.3.2 Deploying the Operator with its RBAC rules

The Operator will automatically create and manage Kubernetes resources to achieve the desired cluster state of Elasticsearch, Kibana, APM Server, Beats, etc.

The ECK Operator will be installed with its RBAC rules[15], by running the following command:

```
$ kubectl apply -f https://download.elastic.co/downloads/eck/2.2.0/operator.yaml
namespace/elastic-system created
serviceaccount/elastic-operator created
secret/elastic-webhook-server-cert created
configmap/elastic-operator created
clusterrole.rbac.authorization.k8s.io/elastic-operator created
clusterrole.rbac.authorization.k8s.io/elastic-operator-view created
clusterrole.rbac.authorization.k8s.io/elastic-operator-edit created
clusterrolebinding.rbac.authorization.k8s.io/elastic-operator created
service/elastic-webhook-server created
statefulset.apps/elastic-operator created
validatingwebhookconfiguration.admissionregistration.k8s.io/elastic-webhook.k8s.elastic.co created
```

Figure 4.19: Installing the operator with its RBAC rules.

4.3.3.3 Deploying Elasticsearch

After creating Elasticsearch configuration YAML file, We deploy it on the cluster.[14]
Since this is only a PoC, the Elasticsearch cluster consists only of one node.

```
$ kubectl apply -f elasticsearch.yml
elasticsearch.elasticsearch.k8s.elastic.co/elasticsearch created
```

Figure 4.20: deploying Elasticsearch.

Checking Elasticsearch status.[14]

```
$ kubectl get elasticsearch
```

NAME	HEALTH	NODES	VERSION	PHASE	AGE
elasticsearch	green	1	8.2.1	Ready	1h

Figure 4.21: Elasticsearch status.

4.3.3.4 Deploying Kibana

After creating Kibana configuration YAML file, We deploy it on the cluster.[13]
Since this is only a PoC, the Kibana cluster consists only of one node.

```
$ kubectl apply -f kibana.yml
kibana.kibana.k8s.elastic.co/kibana created
```

Figure 4.22: deploying Kibana.

Checking Kibana status.[13]

```
$ kubectl get kibana
```

NAME	HEALTH	NODES	VERSION	AGE
kibana	green	1	8.2.1	1h

Figure 4.23: Kibana status.

Checking Kibana Web interface.

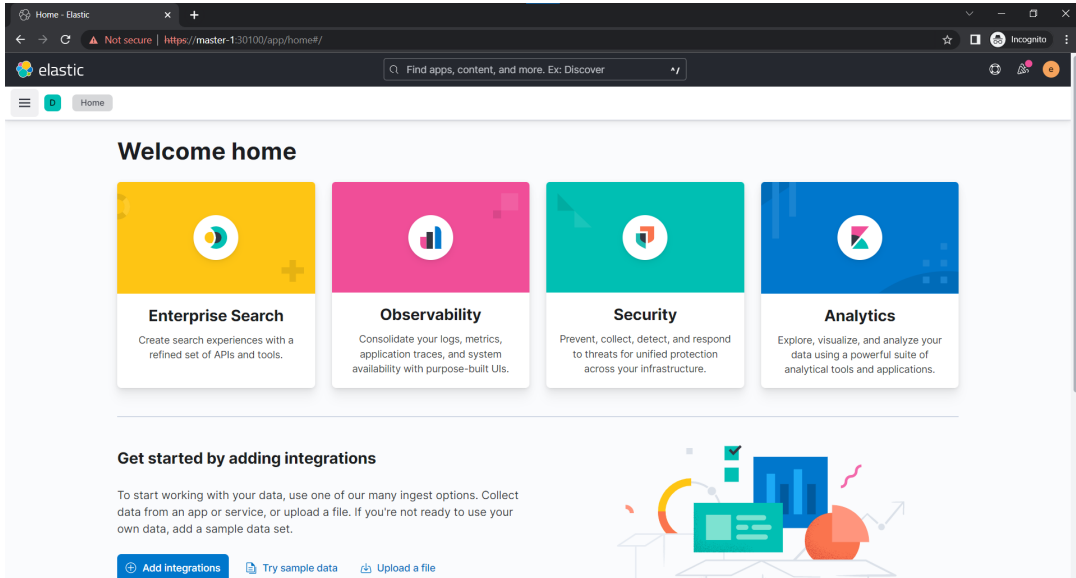


Figure 4.24: Kibana Web interface.

4.3.3.5 Deploying APM

After creating APM configuration YAML file, We deploy it on the cluster.[61]

```

$ kubectl apply -f apm.yml
apmserver.apm.k8s.elastic.co/apm-server created
  
```

Figure 4.25: deploying APM.

Checking APM status.[61]

```

$ kubectl get apm

NAME          HEALTH   NODES   VERSION   AGE
apm-server   green    1       8.2.1     1h
  
```

Figure 4.26: APM status.

4.3.3.6 Deploying Heartbeat

After creating Heartbeat configuration YAML file, we deploy it on the cluster.[54]

```

$ kubectl apply -f heartbeat.yaml

configmap/heartbeat-deployment-config created
deployment.apps/heartbeat created
clusterrolebinding.rbac.authorization.k8s.io/heartbeat created
rolebinding.rbac.authorization.k8s.io/heartbeat created
rolebinding.rbac.authorization.k8s.io/heartbeat-kubeadm-config created
clusterrole.rbac.authorization.k8s.io/heartbeat created
role.rbac.authorization.k8s.io/heartbeat created
role.rbac.authorization.k8s.io/heartbeat-kubeadm-config created
serviceaccount/heartbeat created

```

Figure 4.27: deploying Heartbeat.

Checking Heartbeat deployment status.[54]

```

$ kubectl get deployments

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
heartbeat     1/1     1             1           1h40m

```

Figure 4.28: Heartbeat status.

Checking Heartbeat Web interface.

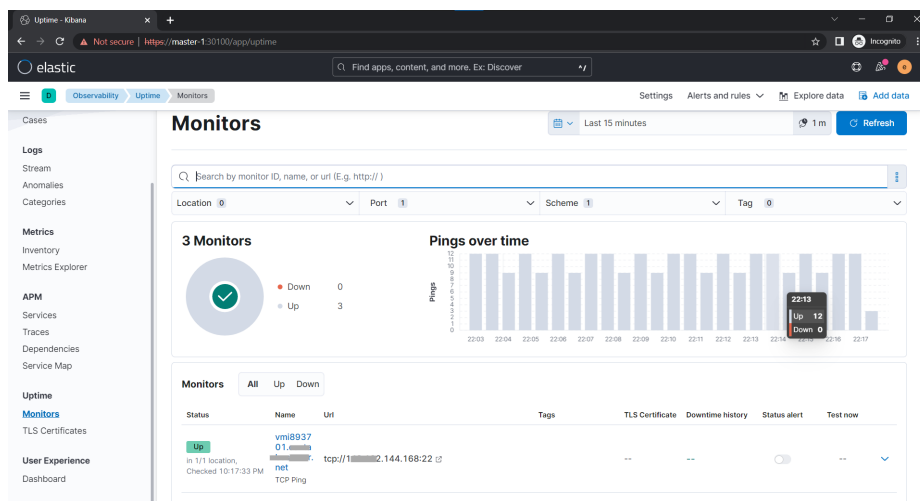


Figure 4.29: Heartbeat Web interface.

4.3.3.7 Deploying Metricbeat

After creating Metricbeat configuration YAML file, we deploy it on the cluster.[53]

```
$ kubectl apply -f metricbeat.yml

beat.beat.k8s.elastic.co/metricbeat created
clusterrole.rbac.authorization.k8s.io/metricbeat created
serviceaccount/metricbeat created
clusterrolebinding.rbac.authorization.k8s.io/metricbeat created
```

Figure 4.30: deploying Metricbeat.

Checking Metricbeat status.[53]

```
$ kubectl get beat

NAME          HEALTH   AVAILABLE   EXPECTED   TYPE          VERSION   AGE
metricbeat    green    3           3          metricbeat    8.2.0    1h
```

Figure 4.31: Metricbeat status.

4.3.3.8 Deploying Logstash

After creating Logstash configuration YAML file, we deploy it on the cluster.[72]

```
$ kubectl apply -f logstash/

configmap/logstash-configmap created
deployment.apps/logstash-deployment created
service/logstash created
```

Figure 4.32: deploying Logstash.

Checking Logstash status.[72]

```
$ kubectl get deployments

NAME                READY   UP-TO-DATE   AVAILABLE   AGE
logstash-deployment  2/2     2            2           1h
```

Figure 4.33: Logstash status.

4.3.4 Deploying Patroni

Patroni is a Python-based open-source PostgreSQL template and controller, Which can run and manage High-Availability Postgres clusters.[29]

4.3.4.1 Deploying Patroni

After creating Patroni configuration YAML files, We deploy it on the cluster.[22]

```
$ kubectl apply -f patroni/

service/patronidemo-config configured
statefulset.apps/patronidemo created
endpoints/patronidemo created
service/patronidemo created
service/patronidemo-repl created
secret/patronidemo created
serviceaccount/patronidemo created
role.rbac.authorization.k8s.io/patronidemo created
rolebinding.rbac.authorization.k8s.io/patronidemo created
clusterrole.rbac.authorization.k8s.io/patroni-k8s-ep-access created
clusterrolebinding.rbac.authorization.k8s.io/patroni-k8s-ep-access created
```

Figure 4.34: Deploying Patroni.

Checking Patroni status.

```
$ kubectl get pods -L role | grep patroni

patronidemo-1          1/1    Running    0          18d    replica
patronidemo-2          1/1    Running    0          18d    master
patronidemo-0          1/1    Running    0          18d    replica
```

Figure 4.35: Deploying Patroni.

4.3.5 Deploying pgAdmin

After creating PgAdmin YAML files, We deploy it on the cluster.[25]

```
$ kubectl apply -f pgAdmin/

deployment.apps/pgadmin created
secret/pgadmin created
statefulset.apps/pgadmin created
service/pgadmin-svc created
```

Figure 4.36: Deploying pgAdmin.

Checking pgAdmin Web interface.

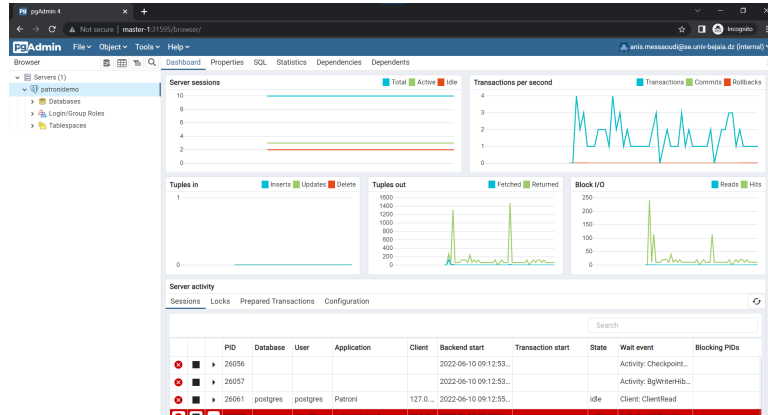


Figure 4.37: pgAdmin Web interface.

4.3.6 Deploying Falco

Falco is the open source tool for continuous risk and threat detection across Kubernetes, containers and cloud. It is continuously detecting unexpected behavior, configuration changes, intrusions, and data theft in real time. Falco has a lot reports output options like exporting into a file, Web server or a Slack channel, by default it uses a simple WebUI that works as output for displaying the latest events called Falcosidekick.[19]

4.3.6.1 Deploying Falco

After creating Falco configuration YAML files, We deploy it on the cluster.[7]

```

$ kubectl apply -f falco/

clusterrole.rbac.authorization.k8s.io/falco created
clusterrolebinding.rbac.authorization.k8s.io/falco created
configmap/falco created
daemonset.apps/falco created
serviceaccount/falco created

```

Figure 4.38: Deploying Falco.

4.3.6.2 Deploying Falco-sidekick

Falco-sidekick will be deployed using Helm charts[21], by running the following command :

```

$ microk8s helm3 install falcosidekick falcosecurity/falcosidekick --set webui.enabled=true --version 0.4.0

NAME: falcosidekick
LAST DEPLOYED: Sat Jun 4 17:27:57 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the URL for Falcosidekick by running these commands:
   kubectl port-forward svc/falcosidekick 2801:2801 --namespace default
   echo "Visit http://127.0.0.1:2801 to use your application"
2. Get the URL for Falcosidekick-UI (WebUI) by running these commands:
   kubectl port-forward svc/falcosidekick-ui 2802:2802 --namespace default
   echo "Visit http://127.0.0.1:2802/ui to use your application"

```

Figure 4.39: Deploying Falco-sidekick.

Checking Falco-sidekick Web interface.

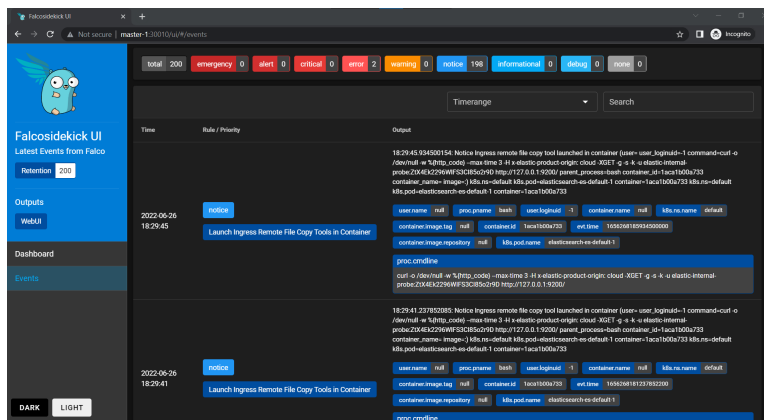


Figure 4.40: Falco-sidekick Web interface.

4.3.7 Deploying RabbitMQ

4.3.7.1 Deploying RabbitMQ Operator

RabbitMQ operator will be deployed[28], by running the following command :

```

$ kubectl apply -f https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml

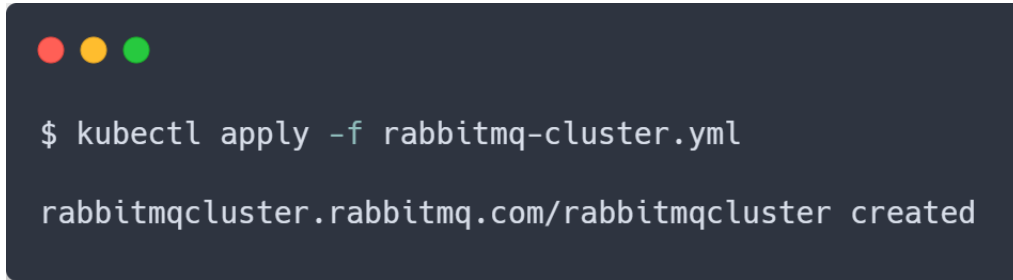
namespace/rabbitmq-system created
customresourcedefinition.apiextensions.k8s.io/rabbitmqclusters.rabbitmq.com created
serviceaccount/rabbitmq-cluster-operator created
role.rbac.authorization.k8s.io/rabbitmq-cluster-leader-election-role created
clusterrole.rbac.authorization.k8s.io/rabbitmq-cluster-operator-role created
rolebinding.rbac.authorization.k8s.io/rabbitmq-cluster-leader-election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/rabbitmq-cluster-operator-rolebinding created
deployment.apps/rabbitmq-cluster-operator created

```

Figure 4.41: Deploying RabbitMQ operator.

4.3.7.2 Deploying RabbitMQ cluster

After creating RabbitMQ cluster configuration YAML file[28], we deploy it on the cluster.



```
$ kubectl apply -f rabbitmq-cluster.yml
rabbitmqcluster.rabbitmq.com/rabbitmqcluster created
```

Figure 4.42: Deploying RabbitMQ cluster.

Checking RabbitMQ Web interface.

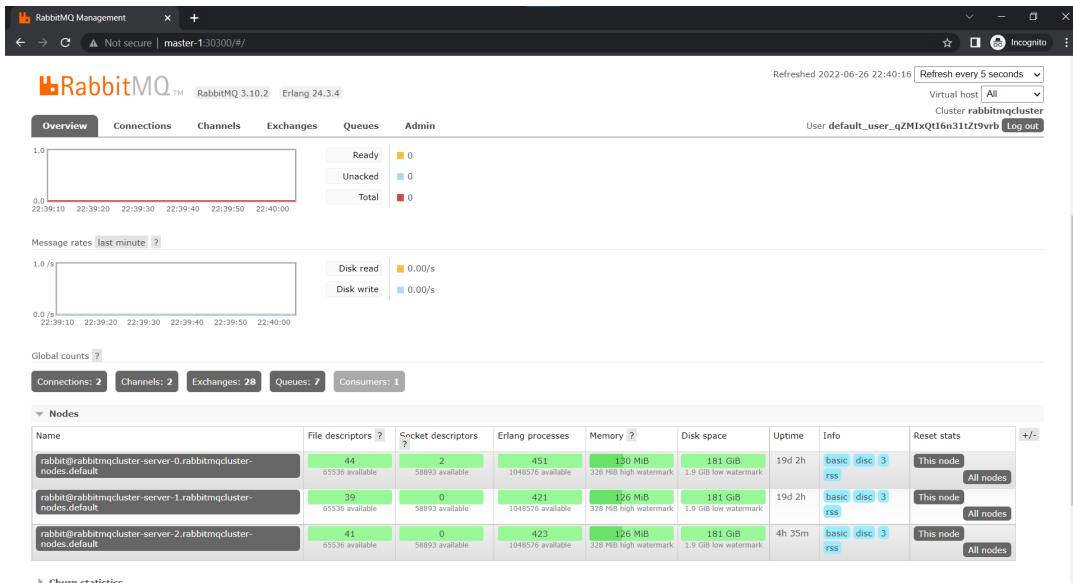


Figure 4.43: RabbitMQ Web interface.

4.3.8 Deploying SIRH Company-Management

After creating SIRH Company-Management configuration YAML files, We deploy it on the cluster.



```
kubectl apply -f company-management-k8s/
```

Figure 4.44: Deploying SIRH Company-Management.

Checking SIRH Company-Management status.

```
kubectl get deployment
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
companymanagement                   1/1      1              1            45h
```

Figure 4.45: SIRH Company-Management status.

4.3.9 Deploying SIRH Trackability

After creating SIRH Trackability configuration YAML files, We deploy it on the cluster.

```
kubectl apply -f trackability-k8s/
```

Figure 4.46: Deploying SIRH Trackability.

Checking SIRH Trackability status.

```
kubectl get deployment
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
webapp-trackability                 1/1      1              1            45h
```

Figure 4.47: SIRH Company-Management status.

4.3.10 Deploying SIRH Billing

After creating SIRH Billing configuration YAML files, We deploy it on the cluster.

```
kubectl apply -f billing-k8s/
```

Figure 4.48: Deploying SIRH Billing.

Checking SIRH Billing status.

```
kubectl get deployment
NAME                                READY  UP-TO-DATE  AVAILABLE  AGE
companymanagement                   1/1    1            1          45h
```

Figure 4.49: SIRH Billing status.

4.3.11 Deploying SIRH Pay

After creating SIRH Pay configuration YAML files, We deploy it on the cluster.

```
kubectl apply -f pay-k8s/
```

Figure 4.50: Deploying Pay.

Checking SIRH Pay status.

```
kubectl get deployment
NAME                                READY  UP-TO-DATE  AVAILABLE  AGE
webapp-pay                          1/1    1            1          45h
```

Figure 4.51: SIRH Pay status.

4.3.12 Deploying Ingress

4.3.12.1 Enabling Ingress addons

This addon adds an NGINX Ingress Controller for MicroK8s. It is enabled by running the command:[45]

```
$ microk8s enable ingress

Infer repository core for addon ingress
Enabling Ingress
ingressclass.networking.k8s.io/public created
namespace/ingress created
serviceaccount/nginx-ingress-microk8s-serviceaccount created
clusterrole.rbac.authorization.k8s.io/nginx-ingress-microk8s-clusterrole created
role.rbac.authorization.k8s.io/nginx-ingress-microk8s-role created
clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress-microk8s created
rolebinding.rbac.authorization.k8s.io/nginx-ingress-microk8s created
configmap/nginx-load-balancer-microk8s-conf created
configmap/nginx-ingress-tcp-microk8s-conf created
configmap/nginx-ingress-udp-microk8s-conf created
daemonset.apps/nginx-ingress-microk8s-controller created
Ingress is enabled
```

Figure 4.52: Enabling Ingress addons.

4.3.12.2 Deploying Ingress rules

Ingress rules can be deployed from the configuration YAML file, by running the following command: [26]

```
$ kubectl apply -f ingress-svc.yml
ingress.networking.k8s.io/ingress-svc created
```

Figure 4.53: Deploying Ingress rules.

4.3.12.3 Checking Ingress status

After deploying ingress rules, we checking it status.

```
$ kubectl get ingress
NAME          CLASS    HOSTS
ingress-svc  public  www.tech-instinct-new.com,bo.tech-instinct-new.com + 3 more...
```

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
ingress-svc	public	www.tech-instinct-new.com,bo.tech-instinct-new.com + 3 more...	127.0.0.1	80	62s

Figure 4.54: Checking Ingress status

Checking SIRH Front Authentication Web interface.

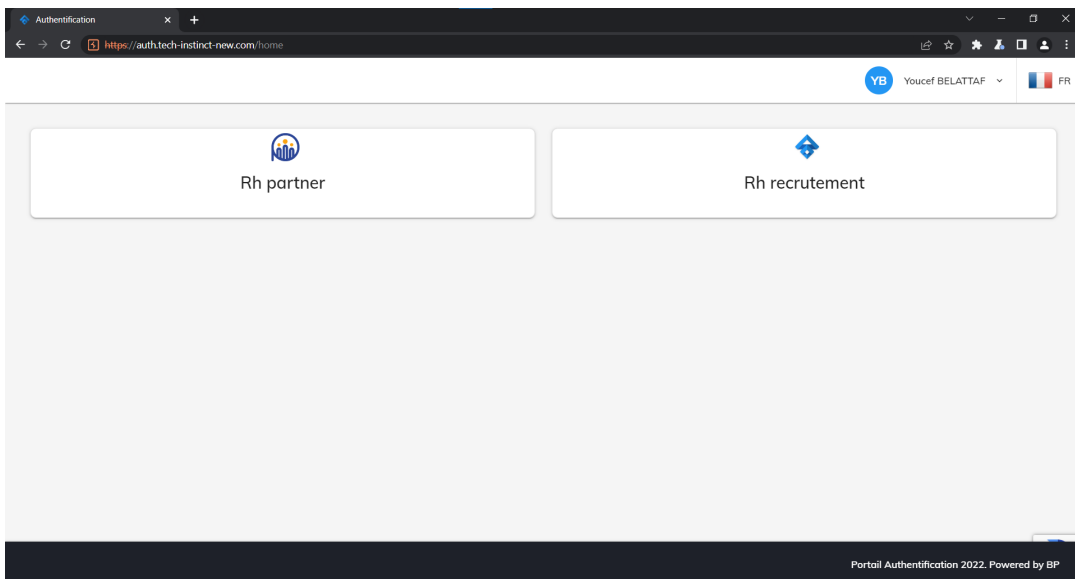


Figure 4.55: SIRH Front Authentication Web interface.

Checking SIRH Backoffice Web interface.

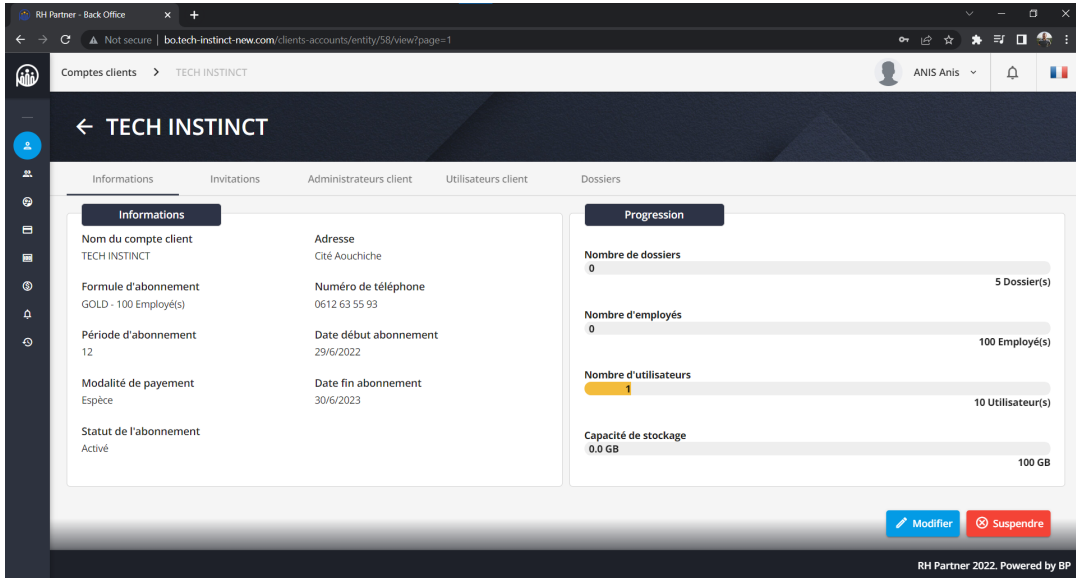


Figure 4.56: SIRH Backoffice Web interface.

Checking SIRH Employee Portal Web interface.

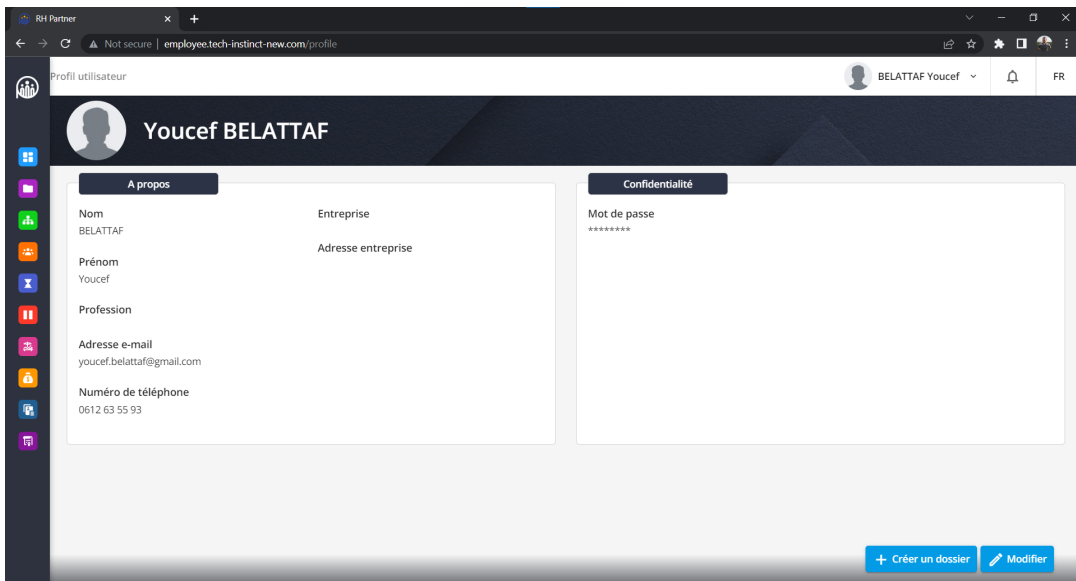


Figure 4.57: SIRH Employee Portal Web interface.

Checking SIRH Facturation Web interface.

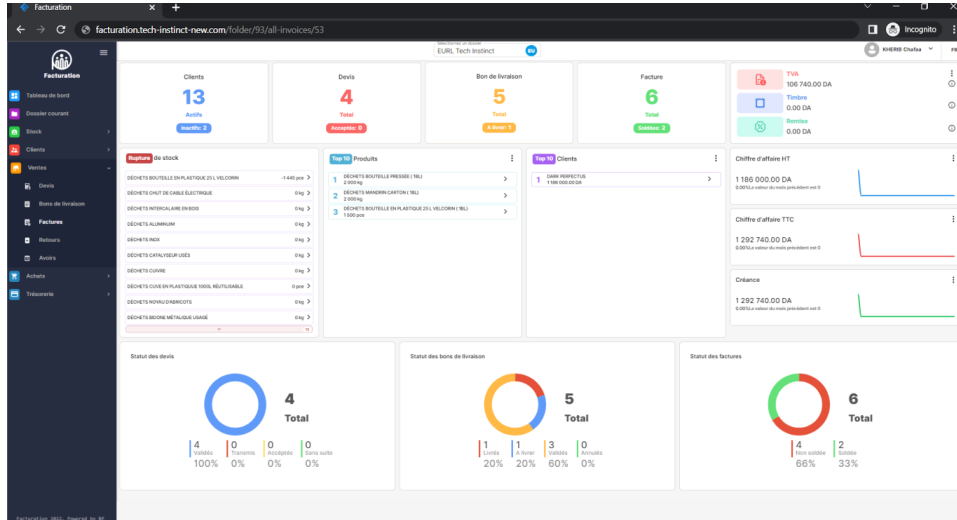


Figure 4.58: SIRH Facturation Web interface.

Checking SIRH Front Web interface.

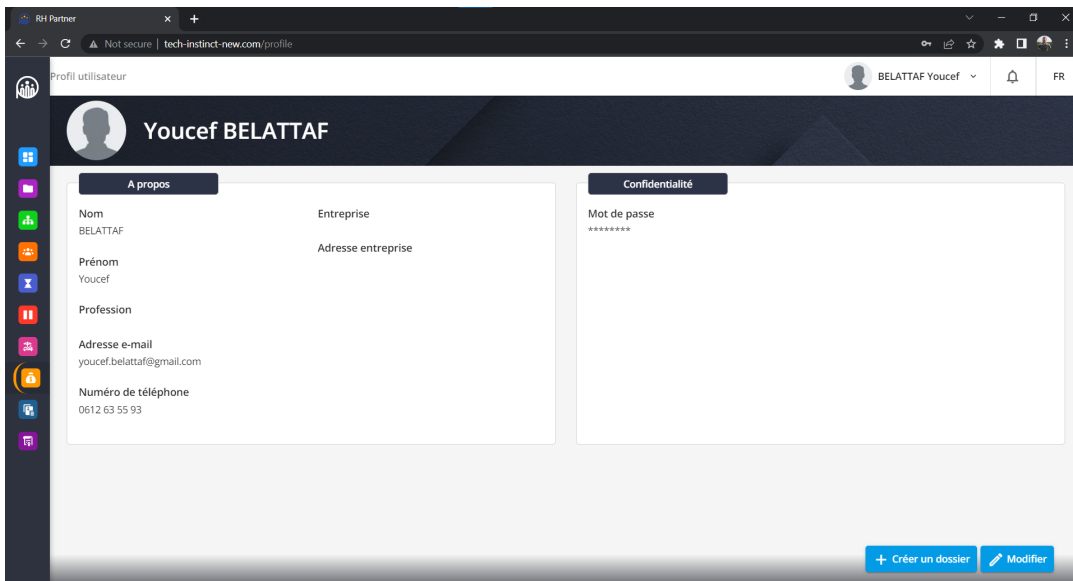


Figure 4.59: SIRH Front Web interface.

4.3.13 Pushing configuration files to GitHub

We push the configuration files of the project to our public GitHub repository `wh0kn0ws/k8s-deployment`

4.3.13.1 Setting up the repository

We start by initializing the local repository that contains our configurations files, then we link it to the public GitHub repository, and we commit the new changes.

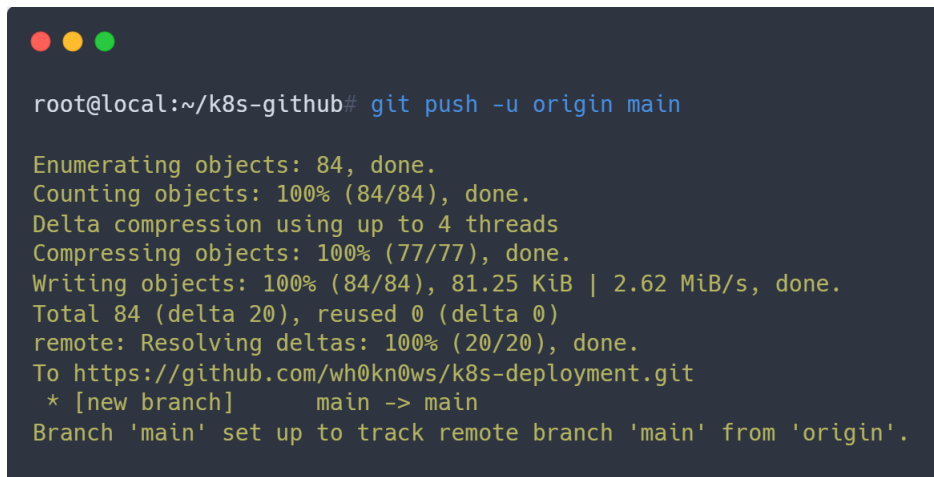


```
root@local:~/k8s-github# git push -u origin main
root@local:~/k8s-github# git add .
root@local:~/k8s-github# git commit -m "K8s commit"
root@local:~/k8s-github# git branch -M main
root@local:~/k8s-github# git remote add origin https://github.com/wh0kn0ws/k8s-deployment.git
```

Figure 4.60: Setting up the repository.

4.3.13.2 Pushing the configuration to the public repository

We push the configuration files to the public GitHub repository.



```
root@local:~/k8s-github# git push -u origin main
Enumerating objects: 84, done.
Counting objects: 100% (84/84), done.
Delta compression using up to 4 threads
Compressing objects: 100% (77/77), done.
Writing objects: 100% (84/84), 81.25 KiB | 2.62 MiB/s, done.
Total 84 (delta 20), reused 0 (delta 0)
remote: Resolving deltas: 100% (20/20), done.
To https://github.com/wh0kn0ws/k8s-deployment.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Figure 4.61: Push to the public repository.

4.3.13.3 Checking GitHub repository

After pushing the configuration files to the remote repository successfully, we can clearly see the presence of the files in the public repository on GitHub platform.

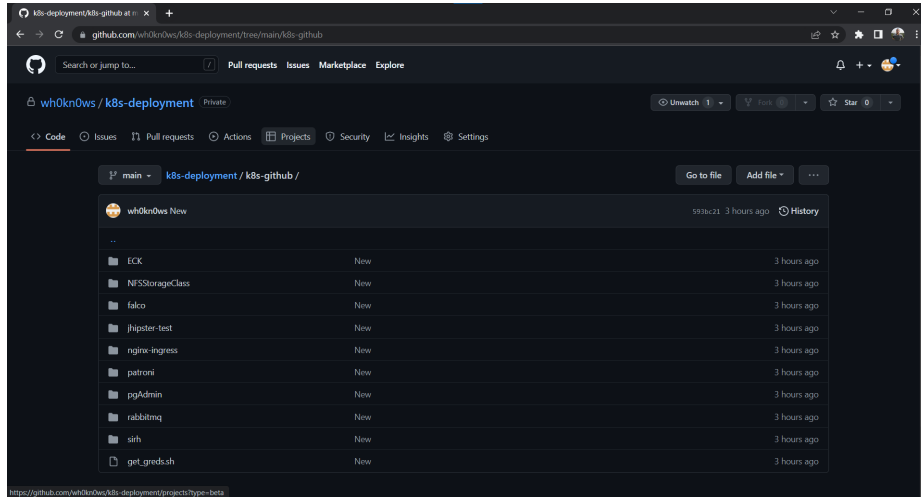


Figure 4.62: Push to the public repository.

4.4 Conclusion

In this chapter, we have presented a diagram that explains the overall architecture and the different components of the new deployment, after that, we have started setting up a highly available Kubernetes cluster, NFS server, and deployed microservices and applications successfully. After finishing with the deployment, we created remote public repository to push the configuration files to share and save the files, and to keep track of the future changes that could be made to the deployment.

General conclusion and future perspectives

Most companies and organizations are adopting the cloud-native approaches because of their performance efficiency, which further lies in technologies like containerization, orchestration, and microservices capable of providing highly scalable, light-weighted, portable, and flexible solutions. Through this work, we aimed to review, analyze, and critique Tech Instinct deployment and its applications and microservices, and then proposed a solution by migrating from a Docker Swarm cluster to a high-availability Kubernetes cluster.

In the future, We will be adding more nodes to the cluster to distribute the workloads more efficiently and reduce the pressure on them, configure auto-scaling rules basing on traffic to reduce downtime and respond to the high demand, protect against potential DDoS attacks, and will also add more storage servers and configure a High Availability storage cluster.

Bibliography

- [1] About calico. <https://projectcalico.docs.tigera.io/about/about-calico>. (Accessed on 06/10/2022).
- [2] About kubernetes services. <https://projectcalico.docs.tigera.io/about/about-kubernetes-services>. (Accessed on 06/06/2022).
- [3] Amqp 0-9-1 model explained — rabbitmq. <https://www.rabbitmq.com/tutorials/amqp-concepts.html>. (Accessed on 06/06/2022).
- [4] Api gateway. <https://www.jhipster.tech/api-gateway/>. (Accessed on 06/06/2022).
- [5] Assigning pods to nodes | kubernetes. <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node>. (Accessed on 06/27/2022).
- [6] Authenticating | kubernetes. <https://kubernetes.io/docs/reference/access-authn-authz/authentication/>. (Accessed on 06/06/2022).
- [7] charts/falco at master · falcosecurity/charts · github. <https://github.com/falcosecurity/charts/tree/master/falco>. (Accessed on 06/29/2022).
- [8] Cloud controller manager | kubernetes. <https://kubernetes.io/docs/concepts/architecture/cloud-controller/>. (Accessed on 06/06/2022).
- [9] Cluster networking | kubernetes. <https://kubernetes.io/docs/concepts/cluster-administration/networking/>. (Accessed on 06/06/2022).
- [10] Container runtime interface (cri) | kubernetes. <https://kubernetes.io/docs/concepts/architecture/cri/>. (Accessed on 06/06/2022).
- [11] Custom resources | kubernetes. <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>. (Accessed on 06/27/2022).
- [12] Daemonset | kubernetes. <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>. (Accessed on 06/06/2022).
- [13] Deploy a kibana instance | elastic cloud on kubernetes [2.3] | elastic. <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-deploy-kibana.html>. (Accessed on 06/29/2022).
- [14] Deploy an elasticsearch cluster | elastic cloud on kubernetes [2.3] | elastic. <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-deploy-elasticsearch.html>. (Accessed on 06/29/2022).

- [15] Deploy eck in your kubernetes cluster | elastic cloud on kubernetes [2.3] | elastic. <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-deploy-eck.html>. (Accessed on 06/29/2022).
- [16] Deployments | kubernetes. <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>. (Accessed on 06/06/2022).
- [17] Dns for services and pods | kubernetes. <https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/>. (Accessed on 06/06/2022).
- [18] Docker compose | neo kobo. <https://neokobo.blogspot.com/2017/04/docker-compose.html>. (Accessed on 06/05/2022).
- [19] The falco project | falco. <https://falco.org/docs/>. (Accessed on 06/15/2022).
- [20] Free and open application performance monitoring | apm user guide [8.2] | elastic. <https://www.elastic.co/guide/en/apm/guide/current/apm-overview.html>. (Accessed on 06/06/2022).
- [21] Github - falcosecurity/falcosidekick: Connect falco to your ecosystem. <https://github.com/falcosecurity/falcosidekick>. (Accessed on 06/29/2022).
- [22] Github - zalando/patroni: A template for postgresql high availability with etcd, consul, zookeeper, or kubernetes. <https://github.com/zalando/patroni>. (Accessed on 06/15/2022).
- [23] Heartbeat by elastic | docker hub. https://hub.docker.com/_/heartbeat. (Accessed on 06/06/2022).
- [24] Helm. <https://helm.sh/>. (Accessed on 06/06/2022).
- [25] How to deploy pgadmin in kubernetes. <https://www.enterprisedb.com/blog/how-deploy-pgadmin-kubernetes>. (Accessed on 06/15/2022).
- [26] Ingress | kubernetes. <https://kubernetes.io/docs/concepts/services-networking/ingress/>. (Accessed on 06/06/2022).
- [27] Install tools | kubernetes. <https://kubernetes.io/docs/tasks/tools/>. (Accessed on 06/06/2022).
- [28] Installing rabbitmq cluster operator in a kubernetes cluster — rabbitmq. <https://www.rabbitmq.com/kubernetes/operator/install-operator.html>. (Accessed on 06/29/2022).
- [29] Introduction — patroni 2.1.4 documentation. <https://patroni.readthedocs.io/en/latest/>. (Accessed on 06/26/2022).
- [30] Jhipster - full stack platform for the modern developer! <https://www.jhipster.tech/>. (Accessed on 06/06/2022).
- [31] Jhipster registry. <https://www.jhipster.tech/jhipster-registry/>. (Accessed on 06/06/2022).

- [32] Kibana: Explore, visualize, discover data | elastic. <https://www.elastic.co/kibana/>. (Accessed on 06/06/2022).
- [33] kube-apiserver | kubernetes. <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/>. (Accessed on 06/06/2022).
- [34] kube-controller-manager | kubernetes. <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/>. (Accessed on 06/06/2022).
- [35] kube-proxy | kubernetes. <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/>. (Accessed on 06/06/2022).
- [36] kube-scheduler | kubernetes. <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/>. (Accessed on 06/06/2022).
- [37] kubelet | kubernetes. <https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/>. (Accessed on 06/06/2022).
- [38] Kubernetes components | kubernetes. <https://kubernetes.io/docs/concepts/overview/components/>. (Accessed on 06/06/2022).
- [39] Kubernetes components | kubernetes. <https://kubernetes.io/docs/concepts/overview/components/>. (Accessed on 06/06/2022).
- [40] Logging aggregator · tmt common software (csw). https://tmtsoftware.github.io/csw/commons/logging_aggregator.html. (Accessed on 06/06/2022).
- [41] Logstash: Collect, parse, transform logs | elastic. <https://www.elastic.co/logstash/>. (Accessed on 06/06/2022).
- [42] Master 2 - thesis, deploying and securing a kubernetes cluster - online latex editor overleaf. <https://www.overleaf.com/project/62801418836fa94030a2ee98>. (Accessed on 06/08/2022).
- [43] Messaging that just works — rabbitmq. <https://www.rabbitmq.com/>. (Accessed on 06/06/2022).
- [44] Metricbeat overview | metricbeat reference [8.2] | elastic. <https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html>. (Accessed on 06/06/2022).
- [45] Microk8s - addon: Ingress. <https://microk8s.io/docs/addon-ingress>. (Accessed on 06/28/2022).
- [46] Microk8s - high availability (ha). <https://microk8s.io/docs/high-availability>. (Accessed on 06/29/2022).
- [47] Microk8s - microk8s documentation - home. <https://microk8s.io/docs>. (Accessed on 06/06/2022).
- [48] Microk8s - use nfs for persistent volumes. <https://microk8s.io/docs/nfs#heading--nfs>, journal=microk8s.io, language=en ,. (Accessed on 06/29/2022).

- [49] Nodes | kubernetes. <https://kubernetes.io/docs/concepts/architecture/nodes/>. (Accessed on 06/06/2022).
- [50] Operating etcd clusters for kubernetes | kubernetes. <https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/>. (Accessed on 06/06/2022).
- [51] Persistent volumes | kubernetes. <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>. (Accessed on 06/06/2022).
- [52] Replicaset | kubernetes. <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>. (Accessed on 06/06/2022).
- [53] Run metricbeat on kubernetes | metricbeat reference [8.3] | elastic. <https://www.elastic.co/guide/en/beats/metricbeat/current/running-on-kubernetes.html>. (Accessed on 06/29/2022).
- [54] Running heartbeat on kubernetes | heartbeat reference [8.3] | elastic. <https://www.elastic.co/guide/en/beats/heartbeat/current/running-on-kubernetes.html>. (Accessed on 06/29/2022).
- [55] Secrets | kubernetes. <https://kubernetes.io/docs/concepts/configuration/secret/>. (Accessed on 06/06/2022).
- [56] Service | kubernetes. <https://kubernetes.io/docs/concepts/services-networking/service/>. (Accessed on 06/06/2022).
- [57] Statefulsets | kubernetes. <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>. (Accessed on 06/06/2022).
- [58] Storage classes | kubernetes. <https://kubernetes.io/docs/concepts/storage/storage-classes/>. (Accessed on 06/06/2022).
- [59] Tls | kubernetes. https://kubernetes.io/docs/tasks/tls/_print/. (Accessed on 06/06/2022).
- [60] Top 6 kubernetes as a service providers and why you need them - aqua. <https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-as-a-service/>. (Accessed on 06/06/2022).
- [61] Use an elasticsearch cluster managed by eck | elastic cloud on kubernetes [2.3] | elastic. <https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-apm-eck-managed-es.html>. (Accessed on 06/29/2022).
- [62] Using jhipster uaa for microservice security. <https://www.jhipster.tech/using-uaa/>. (Accessed on 06/06/2022).
- [63] Using rbac authorization | kubernetes. <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>. (Accessed on 06/06/2022).
- [64] Volumes | kubernetes. <https://kubernetes.io/docs/concepts/storage/volumes/>. (Accessed on 06/06/2022).
- [65] What are microservices? <https://microservices.io/index.html>. (Accessed on 06/05/2022).

- [66] What is a kubernetes operator? - aqua. <https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-operators/>. (Accessed on 06/10/2022).
- [67] What is container orchestration? definition & related faqs | avi networks. <http://vinetworks.com/glossary/container-orchestration/>. (Accessed on 06/05/2022).
- [68] What is elasticsearch? | elastic. <https://www.elastic.co/what-is/elasticsearch>. (Accessed on 06/06/2022).
- [69] What is kubernetes? | kubernetes. <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. (Accessed on 06/05/2022).
- [70] What is nginx? - nginx. <https://www.nginx.com/resources/glossary/nginx/>. (Accessed on 06/06/2022).
- [71] Workloads | kubernetes. <https://kubernetes.io/docs/concepts/workloads/>. (Accessed on 06/06/2022).
- [72] How to deploy logstash and filebeat on kubernetes with eck and ssl | by raphael de lio | medium. <https://raphaeldelio.medium.com/deploy-logstash-and-filebeat-on-kubernetes-with-eck-ssl-and-filebeat-d9f616737390> Nov 2020. (Accessed on 06/29/2022).
- [73] Kubernetes deprecating docker & kubernetes container runtimes: What you need to know | hcl blogs. <https://web.archive.org/web/20210923151939/https://www.hcltech.com/blogs/kubernetes-deprecating-docker-and-kubernetes-container-runtimes-what-you-need-know> Sep 2021. (Accessed on 06/06/2022).
- [74] A propos - tech-instinct. <https://tech-instinct.com/a-propos/>, January 2021. (Accessed on 06/05/2022).
- [75] Tech-instinct logo. "<https://tech-instinct.com/>", January 2021. (Accessed on 06/05/2022).
- [76] What are namespaces and cgroups, and how do they work? ₂₀₂₁, Jul 2021.
- [77] What is a container? - docker. <https://www.docker.com/resources/what-container/>, Nov 2021. (Accessed on 06/08/2022).
- [78] What is a container? - docker. <https://www.docker.com/resources/what-container/>, November 2021. (Accessed on 06/05/2022).
- [79] Docker desktop overview | docker. <https://docs.docker.com/desktop/>, Jun 2022. (Accessed on 06/05/2022) Documentation.
- [80] Docker hub quickstart | docker documentation. <https://docs.docker.com/docker-hub/>, Jun 2022. (Accessed on 06/05/2022).
- [81] Docker overview | docker documentation. <https://docs.docker.com/get-started/overview/>, Jun 2022. (Accessed on 06/05/2022).

- [82] How nodes work | docker documentation. <https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/#manager-nodes>, Jun 2022. (Accessed on 06/05/2022).
- [83] Manage swarm security with public key infrastructure (pki) | docker documentation. <https://docs.docker.com/engine/swarm/how-swarm-mode-works/pki/>, Jun 2022. (Accessed on 06/05/2022).
- [84] Overview of docker compose | docker documentation. <https://docs.docker.com/compose/>, Jun 2022. (Accessed on 06/05/2022).
- [85] Swarm mode overview | docker documentation. <https://docs.docker.com/engine/swarm/>, Jun 2022. (Accessed on 06/05/2022).
- [86] Use overlay networks | docker documentation. <https://docs.docker.com/network/overlay/>, Jun 2022. (Accessed on 06/05/2022).
- [87] Abdalrhmanalkraien. Intro to nginx web server (part 1). <https://medium.com/javarevisited/intro-to-nginx-web-server-part-1-bb590fad7035>, Feb 2022. (Accessed on 06/06/2022).
- [88] Asad Ali. 14 container orchestration tools for devops. <https://geekflare.com/container-orchestration-software/>, Jul 2020. (Accessed on 06/05/2022).
- [89] Kumar Atul. Docker architecture | docker resource isolation | lifecycle. <https://k21academy.com/docker-kubernetes/docker-architecture-docker-engine-components-container-lifecycle/>, June 2020. (Accessed on 06/05/2022).
- [90] Stephen J. Bigelow. A breakdown of container runtimes for kubernetes and docker. <https://www.techtarget.com/searchitoperations/tip/A-breakdown-of-container-runtimes-for-Kubernetes-and-Docker>, October 2021. (Accessed on 06/05/2022).
- [91] Wikimedia Commons. File:devops-nshe free media repository. <https://commons.wikimedia.org/w/index.php?title=File:Devops-toolchain.svg&oldid=504012285>, 2020. (Accessed on 06/05/2022).
- [92] Aveek Das. An overview of pgadmin – postgresql management tool. <https://www.sqlshack.com/an-overview-of-pgadmin-postgresql-management-tool/>, Jun 2021.
- [93] durgeshkashyap. Dockerfile tutorial | dockerfile example | 2021 - infohubblog. <https://infohubblog.com/dockerfile-tutorial-dockerfile-example.html>, Aug 2021. (Accessed on 06/05/2022).
- [94] Richard Hatheway. Why enterprise it organizations will benefit from application containerization. <https://www.cio.com/article/189567/why-enterprise-it-organizations-will-benefit-from-application-containerization.html>, November 2021. (Accessed on 06/05/2022).
- [95] Gerend Jason. Containers vs virtual machines. <https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm>, Oct 2021. (Accessed on 06/05/2022).

- [96] Doug Jones. Containers vs. virtual machines (vms): What's the difference? =<https://www.netapp.com/blog/containers-vs-vms/>, Mar 2018. (Accessed on 06/05/2022).
- [97] Stefanie Lai. Layer-by-layer cgroup in kubernetes. =<https://medium.com/geekculture/layer-by-layer-cgroup-in-kubernetes-c4e26bda676c>, Nov 2021. (Accessed on 06/05/2022).
- [98] Dac-Nhuong Le, Raghvendra Kumar, Gia Nhu Nguyen, and Jyotir Moy Chatterjee. *Cloud Computing and Virtualization*, page 23. John Wiley & Sons, Inc., Hoboken, NJ, USA, Mar 2018. (Accessed on 06/05/2022).
- [99] Dac-Nhuong Le, Raghvendra Kumar, Gia Nhu Nguyen, and Jyotir Moy Chatterjee. *Cloud Computing and Virtualization*, page 16. John Wiley & Sons, Inc., Hoboken, NJ, USA, Mar 2018. (Accessed on 06/05/2022).
- [100] Dac-Nhuong Le, Raghvendra Kumar, Gia Nhu Nguyen, and Jyotir Moy Chatterjee. *Cloud Computing and Virtualization*, page 22. John Wiley & Sons, Inc., Hoboken, NJ, USA, Mar 2018. (Accessed on 06/05/2022).
- [101] Magno Logan. Securing the 4 cs of cloud-native systems: Cloud, cluster, container, and code - security news. <https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/securing-the-4-cs-of-cloud-native-systems-cloud-cluster-container-and-code>, May 2020. (Accessed on 06/30/2022).
- [102] Marek Moravcik and Martin Kontsek. Overview of docker container orchestration tools. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, page 476, Košice, Slovenia, Nov 2020. IEEE. (Accessed on 06/05/2022).
- [103] Marek Moravcik and Martin Kontsek. Overview of docker container orchestration tools. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, page 477, Košice, Slovenia, Nov 2020. IEEE. (Accessed on 06/05/2022).
- [104] Janakiram MSV. Managing persistence for docker containers – the new stack. <https://thenewstack.io/methods-dealing-container-storage/>, Sep 2016. (Accessed on 06/05/2022).
- [105] Stefan Scherer. Run a local windows docker swarm. <https://stefanscherrer.github.io/build-your-local-windows-docker-swarm/>, March 2016. (Accessed on 06/05/2022).
- [106] Jay Shah and Dushyant Dubaria. Building modern clouds: Using docker, kubernetes & google cloud platform. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, page 0189, Las Vegas, NV, USA, Jan 2019. IEEE.
- [107] WMWare Staff. What is a hypervisor? <https://www.vmware.com/topics/glossary/content/hypervisor.html>. (Accessed on 06/05/2022).
- [108] *ZindagiTechnologies. Nfsisfile-levelstorage,allocatedbysharedstorage,Feb2022. [Online;accessed6-June-2022].*
- [109] Wikipedia contributors. Cgroups — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Cgroups&oldid=1091179269>, 2022. (Accessed on 06/05/2022)=.

- [110] Wikipedia contributors. "ci/cd — Wikipedia, the free encyclopedia". <https://enCI/CD&oldid=10809>, 2022. (Accessed on 06/05/2022).
- [111] Wikipedia contributors. Devops — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=DevOps&oldid=1090566794>, 2022. (Accessed on 06/05/2022).
- [112] Wikipedia contributors. Git — Wikipedia,note = (Accessed on 06/05/2022), the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Git&oldid=1089059306>, 2022.
- [113] Wikipedia contributors. Github — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=GitHub&oldid=1095216695>, 2022. [Online; accessed 2-July-2022].
- [114] Wikipedia contributors. High availability — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=High_availability&oldid=1085124228, 2022. [Online; accessed 6-June-2022].
- [115] Wikipedia contributors. High-availability cluster — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=High-availability_cluster&oldid=1085114380, 2022. [Online; accessed 26-June-2022].
- [116] Wikipedia contributors. Linux — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Linux&oldid=1091428024>, 2022. (Accessed on 06/05/2022).
- [117] Wikipedia contributors. Linux namespaces — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Linux_namespaces&oldid=1090068250, 2022. (Accessed on 06/05/2022).
- [118] Wikipedia contributors. Network file system — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Network_File_System&oldid=1089090313, 2022. [Online; accessed 6-June-2022].
- [119] Wikipedia contributors. Postgresql — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=PostgreSQL&oldid=1089176863>, 2022. [Online; accessed 6-June-2022].
- [120] Wikipedia contributors. Snap (software) — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Snap_\(software\)&oldid=1095218727](https://en.wikipedia.org/w/index.php?title=Snap_(software)&oldid=1095218727), 2022. [Online; accessed 2-July-2022].
- [121] Wikipedia contributors. Virtual machine — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Virtual_machine&oldid=1086757768, 2022. (Accessed on 06/05/2022).
- [122] Anuj Kumar Yadav, M. L. Garg, and Ritika. *Docker Containers Versus Virtual Machine-Based Virtualization*, volume 814, page 147. Springer Singapore, Singapore, 2019. (Accessed on 06/05/2022).