University of Hradec Králové

Faculty of Informatics and Management

Department of Information Technology

Virtualization and virtualization clusters on Linux systems

Master's thesis

Author: JAN KOHOUT
Field of study: Applied Informatics

Supervisor: Ing. PAVEL BLAŽEK, Ph.D.

Luzern 2023

Declaration

I warrant that the thesis is my original work and that I have not received outside assistance. Only the sources cited have been used in this draft. Parts that are direct quotes or paraphrases are identified as such.

.....

Jan Kohout

April 23, 2023

Acknowledgements

I would like to acknowledge and express my deepest appreciation to my supervisor, Pavel Blažek, for his support. I am also grateful to my colleagues for their support and encouragement. I would like to thank Jean Sebastien Susset and Fadri Pestalozzi for their assistance and support and reviews. Lastly, I would like to extend my appreciation to all the participants who generously gave their time and left me very valuable feedback.

Anotation

The main objectives of this thesis are to conduct research on cloud technologies and design a distributed system based on modern principles. The system is built on cloud resources using microservices architecture design patterns and is adaptive through autoscaling. The entire solution meets the demand for modern approaches and standards. The master's thesis comprises of four primary chapters. The first chapter is an introduction that describes the motivation and focus of the research. The second chapter conducts research across various cloud providers and their solutions, and concludes with a decision on which provider to choose. The third chapter focuses on the system infrastructure's architecture provided by cloud services and designs it based on the knowledge gathered from research on each component. The fourth chapter describes the implementation of the system, including deployments and automations. The conclusions can be found in chapter number five.

Contents

List of Figures

List of Tables

1 Introduction

As a result of globalization, more companies have become international in recent years and doing business in a foreign country on the other side of the world is not uncommon. The high demand for their applications to be available globally causes a number of issues. Instead of building their own infrastructure, which costs more effort, money and human power, most of them decide to move to the cloud. Moving a company into the cloud is not an easy task; it requires a pure company transformation that could take several years of work, where multiple teams contribute, research and implement. Cloud technologies became globally available and reliable and this brought the opportunity to run different application stacks on multiple cloud solutions. There are two main fundamental cloud paradigms: public and private. The public cloud is mostly used by small businesses, whereas the private cloud is mostly used by large enterprises and can be highly customized. This thesis focuses on the transformation of small to midsize enterprises into the public cloud with a high demand for customization and security. As stated above, there are several fundamental requirements, which are scalability, reliability, accessibility and security. Security in the sense of IP security, advanced firewalls and data storage; scalability—to provide a solution that is automatically able to adapt to continuous traffic in heavily loaded production environments. Reliability in terms of adopting the service level objectives Service level objective (SLO) and service level agreements Service level agreement (SLA) is defined by the contract between the company and the customer and accessibility—the application should be accessible from the region where the customer is located. These requirements could be met by running applications in the cloud in virtualization environments and this master thesis provides research, decisions and architectural solutions. [14]

2 Cloud solutions and cloud providers

There are multiple cloud providers that provide their services around the globe. This chapter compares the biggest players in the cloud market and their Infrastructure as a service (IaaS), Platform as a service (PaaS) and Software as a service (SaaS) solutions. The comparison is made using publicly available metrics such as cloud market shares, security, networking, Virtual machine (VM) tiers, sizes, pricing and others.

2.1 Cloud market growth and market shares

In 2017, the cloud market began to expand rapidly as more businesses decided to transform their infrastructure and make it globally available rather than building their own hybrid or private cloud solutions.This trend was influenced in part by the COVID-19 pandemic, when more employees worked from home and desired to have their services available with the shortest possible connection response time; the same was true for clients abroad. The first news about the cloud market's progressive growth came in 2018, when many IT analytical firms examined stakeholders' interest in public cloud providers and the impact of big enterprise transformation. This information is proven by the Synergy report from 2019:

"New data from Synergy Research Group shows that across seven key cloud services and infrastructure market segments, operator and vendor revenues for 2018 passed the $250 billion milestone, having grown by 32% from 2017. IaaS & PaaS services had the highest growth rate at 50%, followed by hybrid cloud management software at 41%, enterprise SaaS and public cloud infrastructure both at 30% and hosted private cloud infrastructure services at 29%. In 2016 spending on cloud services first overtook spending on hardware and software used to build public and private clouds and in 2017 and 2018 the gap widened dramatically. Despite a strong 2018 uptick in the growth rate for spending on cloud infrastructure, in aggregate spending on cloud service markets continues to grow much more rapidly. Across the whole cloud ecosystem, companies that featured the most prominently among the 2018 market segment leaders were Microsoft, AmazonAWS, Dell EMC and IBM. They were followed by Salesforce, Cisco, HPE, Adobe and VMware. In aggregate these nine accounted for well over half of all 2018, total spend on hardware and software used to build cloud infrastructure exceeded $100 billion – somewhat evenly split between public and private clouds – though spend on public cloud continues to grow more rapidly. Infrastructure investments by cloud service providers helped them to generate over $150 billion in revenues from cloud infrastructure services (IaaS, PaaS, hosted private cloud services) and enterprise SaaS, in addition to which their infrastructure supports internet services such as search, social networking, email, ecommerce, gaming and mobile apps." [1] The data of the cloud market growth are visualized in the plot 1.

In the synergy report from October 2019, we can see one of the first comparisons between the four biggest players Amazon, Microsoft, Google and Alibaba and their market shares.

"New data from Synergy Research Group shows that the leading four providers of public cloud services accounted for 72% of the worldwide market for IaaS and PaaS in Q3, up from 57% at the beginning of 2016. Throughout that period Amazon's worldwide market share has held steady at around 40%, while Microsoft, Google and Alibaba have all steadily gained share. The four market leaders are followed by Salesforce, IBM, Oracle, Tencent, SinnetAWS and a large group of companies with minor market shares. Total worldwide spending on public IaaS and PaaS reached $20 billion in Q3, representing over 80% of the total cloud infrastructure services. The rest of the market is comprised of hosted and managed private

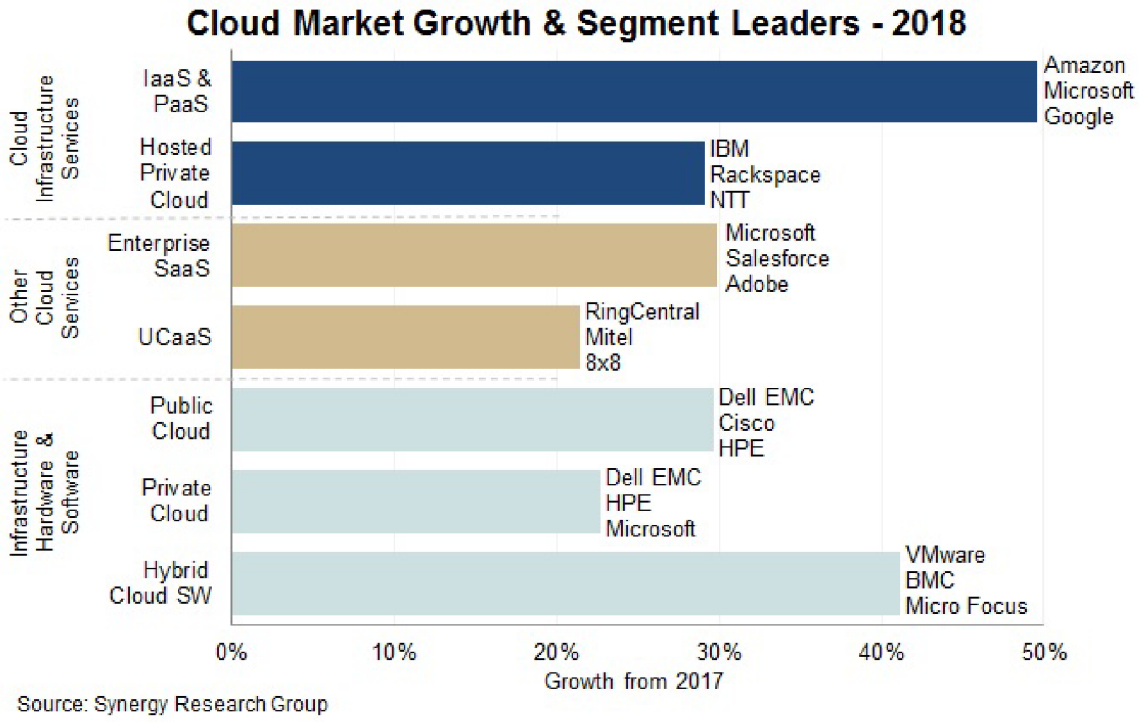


Figure 1: Cloud market growth 2018 [1]

cloud services, where IBM is the market leader and companies like Rackspace and OVH feature more prominently. With most of the major cloud providers having now released their earnings data for Q3, Synergy estimates that total cloud infrastructure service revenues (including IaaS, PaaS and hosted private cloud services) were well over $24 billion in the quarter, with revenues for the last four quarters now reaching $89 billion. The total market grew by 37% from the third quarter of 2018. The public IaaS and PaaS part of the market continues to grow more rapidly than private cloud services, with Q3 growth coming in at 40%. Geographically, the cloud market continues to grow strongly in all regions of the world." [2]

"It has taken just eight quarters for the public IaaS and PaaS markets to double in size and our forecast shows them doubling in size again over the next eleven quarters," said John Dinsdale, a Chief Analyst at Synergy Research Group. "It is particularly noteworthy that as spending on public cloud services continues to grow rapidly, the top four cloud providers are strengthening their grip on the market. Some of the companies outside the top four are actually growing at a reasonable pace, but the reality is that in aggregate they continue to lose ground to the market leaders. Outside of some niche services and geographic regions, this is a game where scale of operations, geographic footprint and global brand are key competitive advantages." [2]

The plot 2 depicts the growth of the cloud market and a comparison of the top four players. The comparison of the growth of four big players in the cloud market from different sources than Synergy Research Group are in the table 1 and it highly correlates.

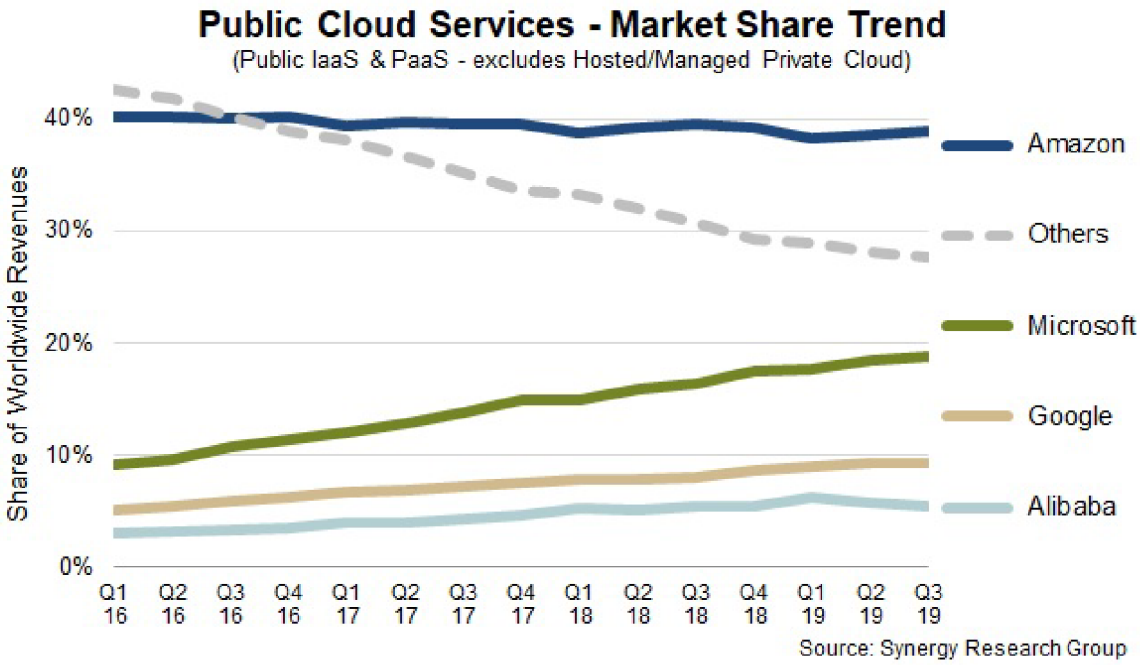


Figure 2: Cloud market growth Q3 2019 [2]

Table 1: Cloud market shares in %

Journal	Period	AWS	Azure cloud (AC)	Google cloud (GC)
Synergy [15]	Q4 2019	33%	18%	?%
Sam solutions [16]	2020	33%	19%	7%
Alto Palo [17]	probably Q1 2021	33%	18%	9%
Canalys [18]	Q1 2021	32%	19%	7%
Hosting seekers [19]	probably Q2 2021	30%	16%	10%
CISIN [20]	Q1 2021	32%	20%	7%
Channele2e [21]	Q1 2022	33%	21%	8%

2.1.1 Latest market shares

The latest data on cloud market shares are shown in the plot 3. The growth of Azure cloud services has increased by 2%. Amazon remains the market leader, but its share has increased by only 1% in the last few years, while Google's cloud services have increased by approximately 2%.

2.2 Cloud services

Cloud computing refers to the delivery of computing resources, such as storage, processing power and networking, over the internet. These resources can be accessed on-demand and are typically offered on a pay-per-use basis. Cloud computing can be a cost-effective and scalable solution for organizations that need to quickly and easily access computing resources. Azure, Google Cloud and AWS are three major providers of cloud computing services. Each provider offers a wide range of services, including IaaS, PaaS and SaaS. IaaS

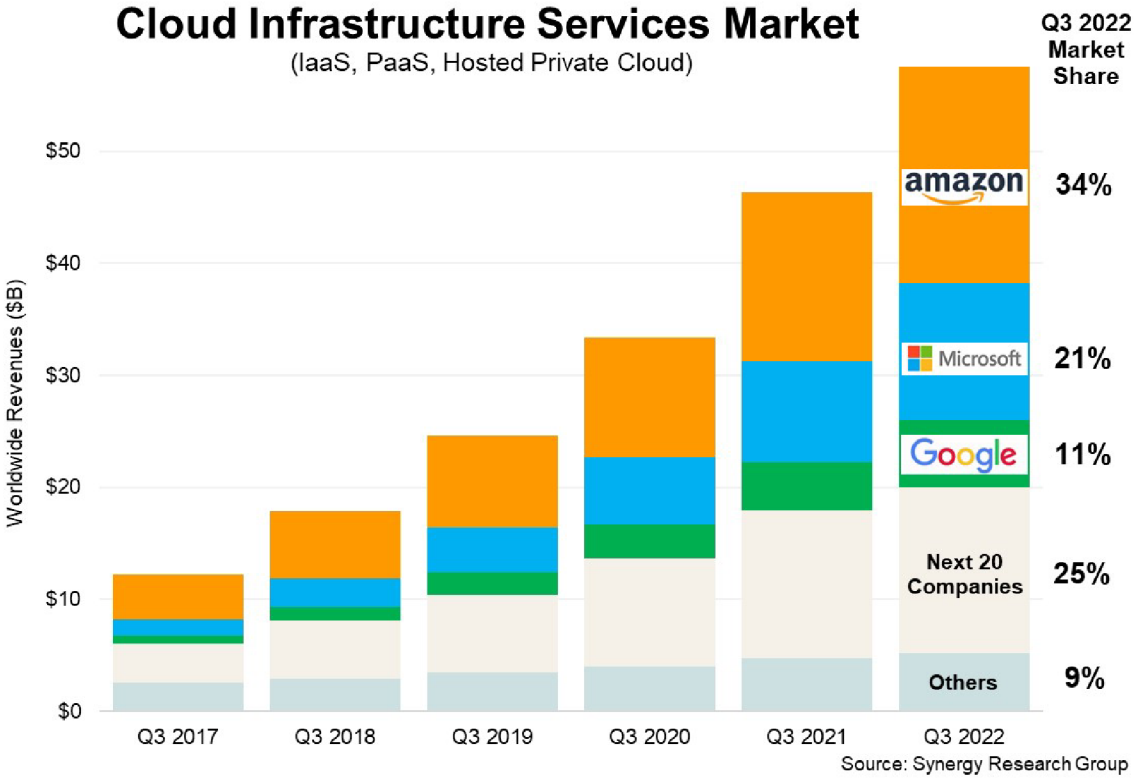


Figure 3: Cloud market growth Q3 2022 [3]

is a type of cloud computing that provides access to raw computing resources, such as storage, networking and processing power. IaaS allows organizations to rent these resources on a pay-per-use basis, rather than having to purchase and maintain their own infrastructure. Examples of IaaS offerings on Azure include virtual machines, storage and networking, while Google Cloud offers services such as Compute Engine, Cloud Storage and Cloud Networking. AWS also offers a range of IaaS services, including EC2, S3 and VPC. PaaS is a type of cloud computing that provides access to a platform for developing, testing and deploying applications. PaaS eliminates the need for organizations to purchase and maintain their own hardware and software infrastructure and allows them to focus on developing and deploying applications. Examples of PaaS offerings on Azure include Azure App Service, Azure Functions and Azure SQL Database, while Google Cloud offers services such as App Engine, Cloud Functions and Cloud SQL. AWS also offers a range of PaaS services, including Elastic Beanstalk, Lambda and RDS.

2.2.1 Cloud computing

Cloud computing refers to the delivery of computing resources, such as storage, processing power and networking, over the internet. These resources can be accessed on-demand and are typically offered on a pay-per-use basis. Cloud computing can be a cost-effective and scalable solution for organizations that need to quickly and easily access computing resources. One type of cloud computing service is IaaS, which provides access to raw computing resources such as storage and processing power. A key component of IaaS is

the VM, which is a software emulation of a physical computer that can be configured with a specific operating system and set of applications. Another type of cloud computing service is PaaS, which provides a platform for developing and deploying applications. PaaS includes container services, which allow applications to be packaged in a container that can be easily deployed on any infrastructure. Container registry is a service that stores and manages container images, while serverless functions are a type of cloud computing that allows organizations to run code without having to worry about the underlying infrastructure, compared in table 4.

Table 2: Comparison of cloud computing engine [12], [13]

Service	AWS	Azure cloud (AC)	Google cloud (GC)
IaaS	Amazon Elastic Compute Cloud (EC2)	Virtual Machines (Azure VMs)	Google Compute Engine (GKE)
PaaS	AWS Elastic Beanstalk	Cloud services	Google App Engine
Containers	Amazon Elastic Compute Cloud Container Service	AKS (Azure Kubernetes Service)	GKE (Google Kubernetes Engine)
Container Deployment		Container service	Container Engine
Container register	EC2 registry	Container registry	Container registry
Serverless Functions	AWS Lambda	Azure Functions	Google Cloud Functions

2.2.2 Cloud storage service

Cloud storage refers to the use of remote servers on the internet to store, manage and process data, rather than storing it on a local hard drive or server. Cloud storage can be a cost-effective and scalable solution for organizations that need to store and access large amounts of data. Cloud storage services are typically classified as either object storage or block storage. Object storage is a type of cloud storage that stores data as objects, which can be easily accessed and managed over the internet. Block storage is a type of cloud storage that stores data as blocks, which can be accessed and managed as part of a file system. Major providers of cloud storage services include Azure, Google Cloud and AWS, each offering a range of object storage and block storage services.

Table 3: Comparison of cloud storage [12], [13]

AWS	Azure cloud (AC)	Google cloud (GC)
Simple Storage Service (S3)	Blob Storage	Cloud Storage
Elastic Block Storage (EBS)	Queue Storage	Persistent Disk
Elastic File System (EFS)	File Storage	Transfer Appliance
Storage Gateway	Disk Storage	Transfer Service
Snowball	Data Lake Store	
Snowball Edge		
Snowmobile		

2.2.3 DaaS Solutions

Database as a service (DaaS) is a type of cloud computing service that provides access to a database over the internet. DaaS allows organizations to rent a database on a pay-per-use basis, rather than having to purchase and maintain their own database infrastructure. This can be a cost-effective and scalable solution for organizations that need to quickly and

easily access a database. DaaS is typically offered as part of PaaS, offering, that provides a platform for developing and deploying applications.

Table 4: Comparison of cloud computing engine [12], [13]

AWS	Azure cloud (AC)	Google cloud (GC)
Aurora	SQL Database	Cloud SQL
RDS	Database for MySQL	Cloud Bigtable
DynamoDB	Database for PostgreSQL	Cloud Spanner
ElastiCache	Data Warehouse	Cloud Datastore
Redshift	Server Stretch Database	
Neptune	Cosmos DB	
Database migration service	Table Storage	
	Redis Cache	
	Data Factory	

2.3 Security

Cloud security and compliance refer to the processes and measures put in place to ensure the secure and compliant use of cloud computing services. Cloud security involves protecting data stored in the cloud and preventing unauthorized access to it, while cloud compliance refers to ensuring that a company's use of cloud computing adheres to relevant laws, regulations and industry standards. Ensuring both security and compliance is essential for companies that use the cloud, as the cloud has become a critical part of many businesses and the amount of sensitive data being stored in the cloud has increased. A breach of security or a failure to adhere to compliance requirements can have serious consequences for a company, including legal and financial penalties, damage to its reputation and the loss of customer trust.

To address both cloud security and compliance, companies should have a clear understanding of relevant laws and regulations and implement policies and procedures to ensure that their use of cloud computing aligns with these requirements. In addition, they should implement technical controls such as encryption and secure authentication protocols to protect against data breaches and unauthorized access. Regular assessments and audits can also help companies ensure that they are adequately addressing both security and compliance.

There are various approaches that companies can take to ensure cloud security and compliance. One approach is to work with a third-party provider that specializes in helping companies achieve and maintain compliance with relevant laws and regulations. These providers can offer expert guidance on the specific requirements that companies need to adhere to, as well as assist with the development and implementation of compliance policies and procedures. They can also help companies assess and improve their technical controls and provide ongoing support to ensure that the company remains compliant over time. By working with these specialized providers, companies can have greater confidence that they are effectively addressing all aspects of cloud security and compliance and minimizing the risk of legal and financial consequences.

Another approach is for companies to develop an in-house compliance and security program. This involves establishing a dedicated team or department responsible for overseeing

compliance and security efforts, as well as implementing policies and procedures to ensure that the company is meeting all relevant requirements. This approach can be more cost-effective for companies that only use a few cloud computing services, but it may be more challenging for companies that rely heavily on the cloud and have a large number of compliance and security requirements to address. Regardless of the approach taken, it is essential that companies prioritize cloud security and compliance to protect their sensitive data and minimize the risk of legal and financial consequences.

2.3.1 Shared responsibility models

Shared responsibility models are an important concept in cloud computing, as they outline the specific responsibilities of the cloud provider and the customer when it comes to security and compliance. These models are meant to make it clear what each party's roles and responsibilities are, so that all the necessary steps are taken to protect sensitive data and follow laws and rules.

There are several different shared responsibility models that are commonly used by cloud providers. In a full responsibility model, the cloud provider assumes complete responsibility for all aspects of security and compliance, including the physical infrastructure, the operating system and the applications. In this model, the customer is responsible for managing their own data and applications within the cloud environment, but the provider handles all other security and compliance concerns.

In a partial responsibility model, the cloud provider and the customer share certain responsibilities. The provider may still be responsible for the physical infrastructure and the operating system, but the customer is responsible for managing their own data and applications and implementing necessary security measures. In this model, it is important for the customer to carefully understand their specific responsibilities and ensure that they are adequately addressing them.

In a full customer responsibility model, the customer assumes complete responsibility for all aspects of security and compliance, including the physical infrastructure, the operating system and the applications. In this model, the cloud provider is responsible for providing the necessary resources and infrastructure, but the customer is responsible for all other security and compliance concerns. This model is usually used when the customer has very specific security or compliance needs that the provider cannot meet.

No matter what model of shared responsibility is used, it is important for both the cloud provider and the customer to know and follow their specific responsibilities to make sure that cloud computing services are used in a safe and legal way.

Every cloud provider has a slightly different model.

The Amazon Shared Responsibility Model [4] is the most simple cloud provider model and can be seen in Figure 4. This model defines two main entities, which are the customer and Amazon web services (AWS) and explains them in a very simplistic manner. It should be mentioned that the customer is responsible for the security of its application stack, beginning with customer data, because the customer defines how the data will be stored and uses one of the AWSproducts. The customer also implements the identity and access management policies and fully configures them while using the access management services from a third-party provider or AWS.

The customer also selects the platform on which the application stack will run (Operating system (OS), Amazon Elastic Kubernetes Service (EKS), Amazon Elastic Con-

tainer Service (ECS), VM) and fully configures network and firewall policies while utilising AWS IaaS, SaaS and PaaS. The customer is also responsible for the client's data and its encryption while being stored on the cloud, as well as the encryption of communication between the client and the cloud, as well as internal communication between specific services within the cloud. For example , application — Transport Layer Security (TLS)—> database. On the other hand, AWS is responsible for its own cloud service stack, which includes a compute engine, storage, database and networking and gives recommendations for the best practises to the customer. AWS is also responsible for the availability of its services through declared availability zones and hardware in the data centers, as well as physical access to the machines and networks and SLA declared in the official terms of service. It should be mentioned that there is a difference between the responsibility for service availability and configuration. AWS provides the availability of the service while the customer configures it in the cloud.

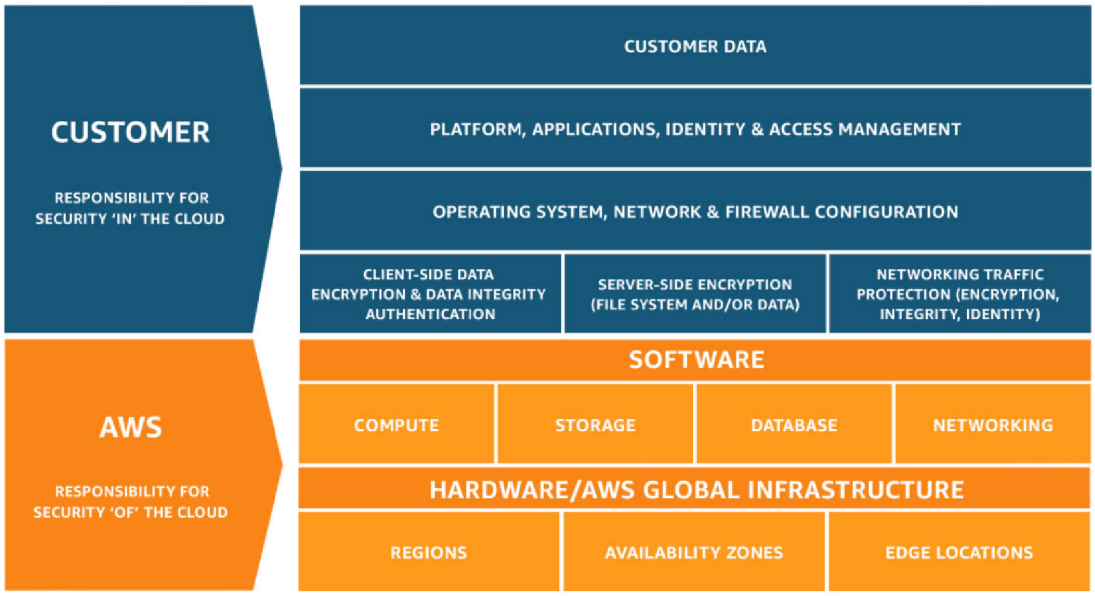


Figure 4: AWS shared responsibility model [4]

Shared responsibilities and shared fate on Google Cloud are shown in Figure 5. While the AWS model is based more on the common description, Google tries to compare its own IaaS, PaaS, SaaS solutions with On premise (on-prem). The Figure also shows two entities, a cloud provider and a customer. As could be seen, once the customer provides its own solutions On-prem, it takes full responsibility. With cloud-based IaaS solutions, the customer is in charge of selecting the guest operating system, data and content. This could be the use case of running the virtual machine in the network with specific access. With a PaaS cloud solution, the customer's responsibility begins at the point of web application security. This could be a use case for a web application running in the serverless computing engine [22]. The SaaS brings almost a whole pack of responsibilities to the cloud provider side and the most explanatory example is using the whole cloud solution by itself because it is SaaS.

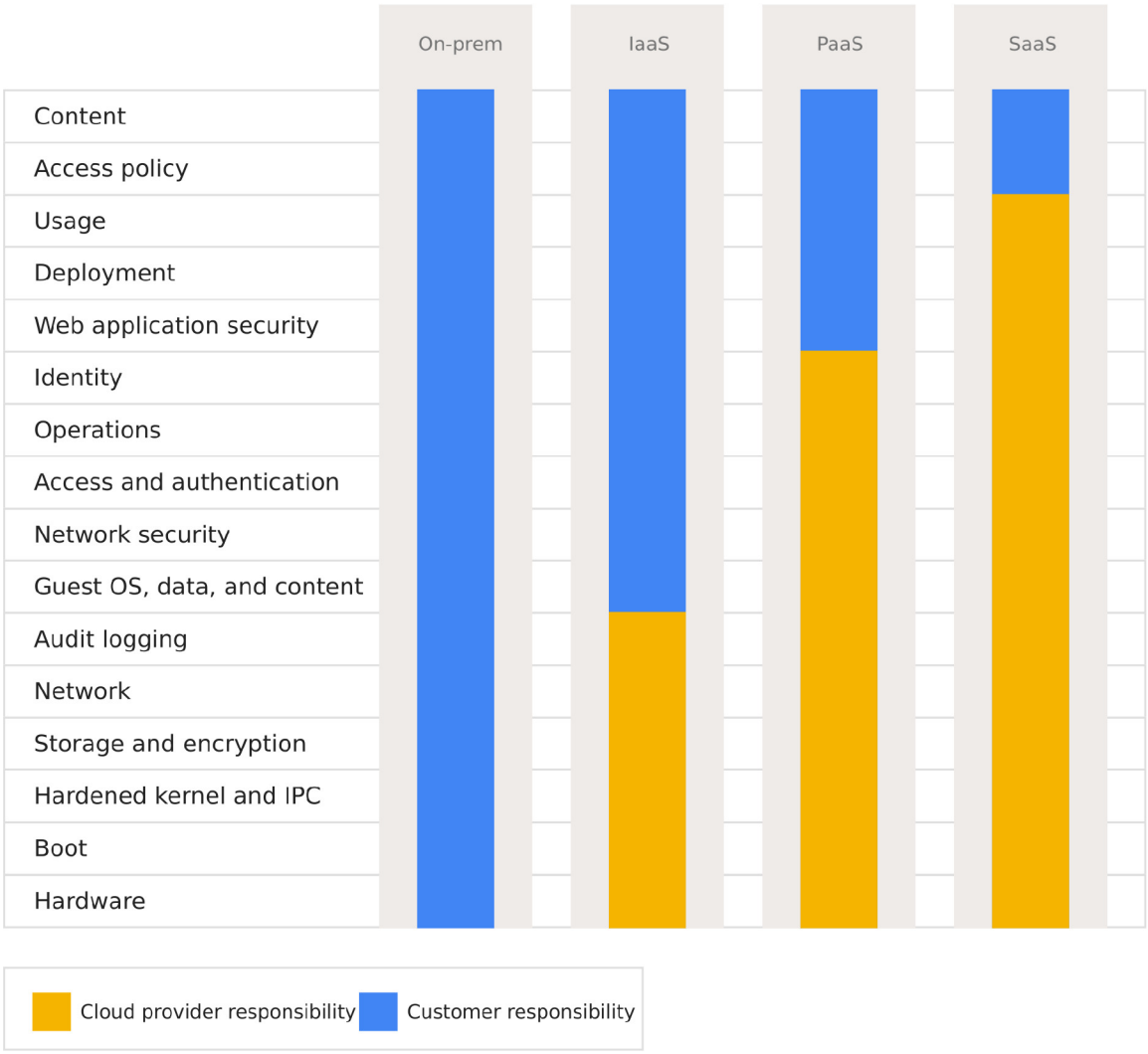


Figure 5: GC shared responsibility model [5]

Division of responsibility Azure approaches the problem from a slightly different angle. Unlike the previous two models, despite its origins in the Google model, Azure does not provide a straightforward responsibility policy, but rather shared responsibilities, as illustrated in Figure 6. On the other hand, it differs in terminology and is specific to provided cloud solutions via Microsoft, so there cannot be any further comparison done, especially in the case of highly configurable and customizable cloud solutions.

2.3.2 Security management

All three cloud providers implement different services for security management. Multifactor authentication (MFA) is supported via multiple authentication apps or text messages, which contain a unique code. The most commonly used apps are the Microsoft Authenticator app [23] or Google Authenticator [24]. The Single sign on (SSO) feature is provided via different systems. AWS has got newly its own Identity and access management (IAM) Center [25], which supports mostly same features as Azure active directory (AD), but it is

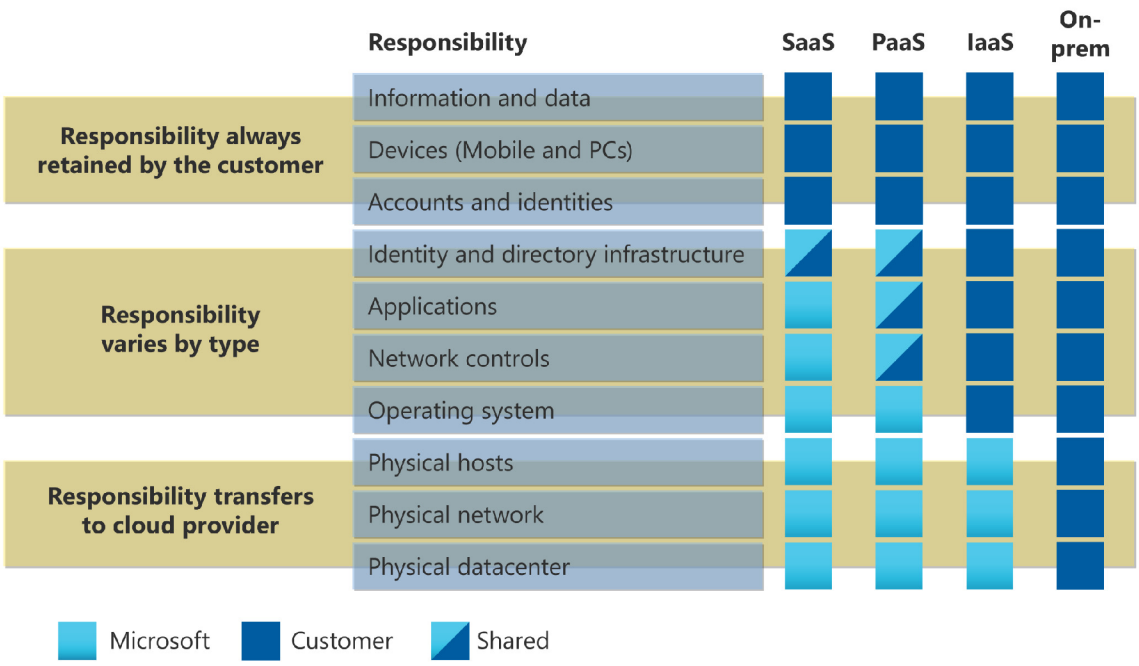


Figure 6: Azure shared responsibility model [6]

also recommended to use third party SSO service like Okta Universal Directory [26] or AD, which is the most advanced SSO provider on the cloud market. Google has got its own SSO services, which are called Cloud Identity or a Google Workspace account. There are several protocols that implement the federation of third-party SSO service

- OpenID Connect
- OAuth
- SAML
- password-based SSO
- linked-based SSO

These protocols provide the authorization and authentication process from identity provider to service provider. The service provider uses an identity provider to acquire the token that defines the user's identity. Based on this token, the user is able to log in to the service. Every SSO implementation includes Role based access control (RBAC) management, which is the method of assigning different permissions to users who manage multiple cloud resources. In Figure 7 are basic resources or entities to which users can gain access based on RBAC policies. This is not an infinite count of resources, but only a comparison of basic structure.

AWS consolidates permissions and resources into a single JSON file (called a policy). Permissions to resources are assigned to an identity via a policy. This is unlike Azure and GCP, which separate the permissions set from the scope; each assignment involves attaching permissions and resources to an identity. In addition, because Azure and GCP

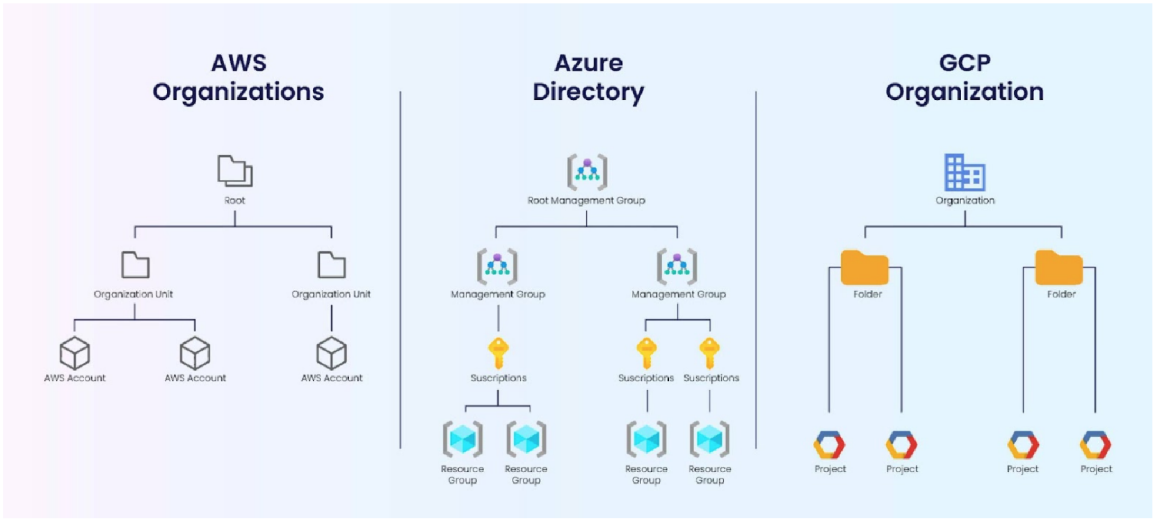


Figure 7: Basic cloud entities hierarchy

follow RBAC permissions, they enable inheritance. Inheritance means that if a role has a higher scope, it has a wider set of permissions. [27]

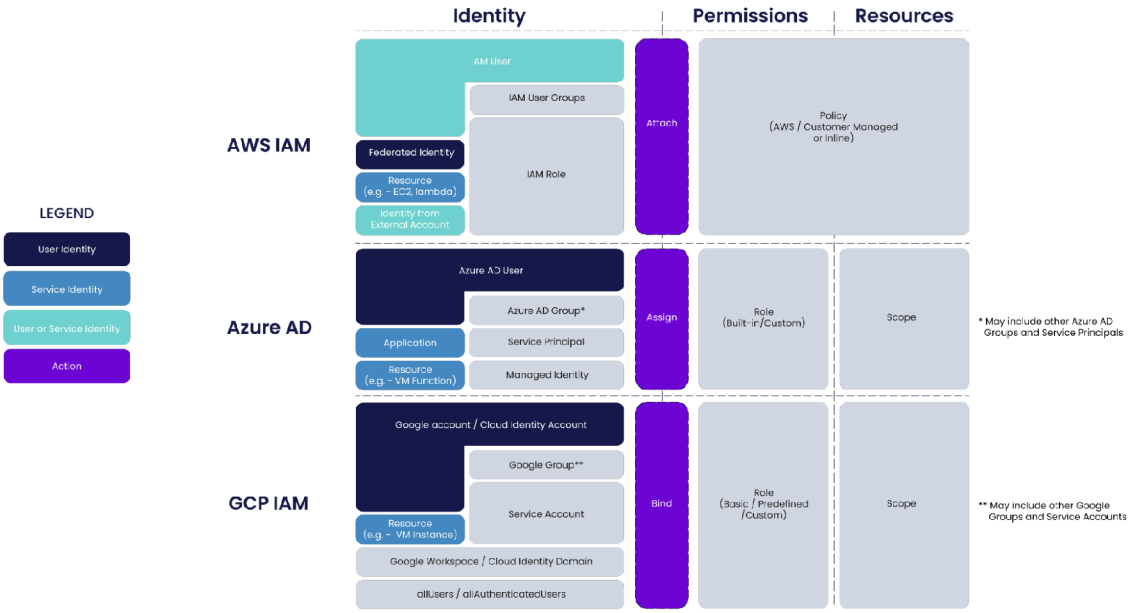


Figure 8: IAM policy differencies

2.3.3 Workload security protection

Workload security protection is a set of best practises implemented into the engine / service on the cloud platform that periodically checks all the running resources in the infrastructure and searches for potential vulnerabilities. This solution is very useful once an app stack is

running on a fully public cloud with internet access. Here is a list of potential vulnerabilities that should be investigated:

- VMs and their OS vulnerabilities
- Container services aka Kubernetes and serverless containers, memory leaks, overloads, thread leaks
- Databases and SQL injections
- Suspicious network traffic
- Distributed denial of service (DDOS)

Every cloud provider offers its own solution for complex continuous vulnerability checks and for network security maintenance (see table 5).

- AWS - AWS Control Tower
- MS Azure - Microsoft Defender for Cloud
- Google Cloud - Google Cloud Platform Security Overview

Table 5: Network security solutions comparison

AWS	Azure cloud (AC)	Google cloud (GC)
AWS shield	DDOS protection	Google Cloud Armor
Application Load Balancer	Application Gateway (WAF V2)	Cloud Load Balancing
NSGs	NSGs	Firewall rules
VPC endpoints	PEPs	Private service connect
VPC peerings	VNET Peerings	VPC peering
WAF	Azure Firewall	Firewall rules
VPC Flow Logs	Azure Network Watcher	Network intelligence center

2.3.4 Secrets management

All three cloud providers have services for storing important application and infrastructure secrets, which are accessible via specific Application Programming Interface (API) or Software Development Kit (SDK) libraries.

Table 6: Secrets management solutions

AWS	Azure cloud (AC)	Google cloud (GC)
Secrets manager	Azure Keyvault	Secret manager

2.3.5 VPN access

If a company decides to offer its product with high security standards, one of the best practises is to use Virtual private network (VPN) access only. It prevents accessibility from the internet and allows possible vulnerabilities to be mitigated, as mentioned with DDOS, SQL injections, etc. There are two implementations of VPN: VPN - Point to site (P2S) and Site to site (S2S), described in Figure 9. Not every cloud provider offers the same setup for VPN configuration as it's described in Table 7.

Table 7: Secrets management solutions

AWS	Azure cloud (AC)	Google cloud (GC)
AWS VPN	VPN Gateway	?
P2S and S2S	P2S and S2S	S2S
10 S2S connections	30 S2S connections	?

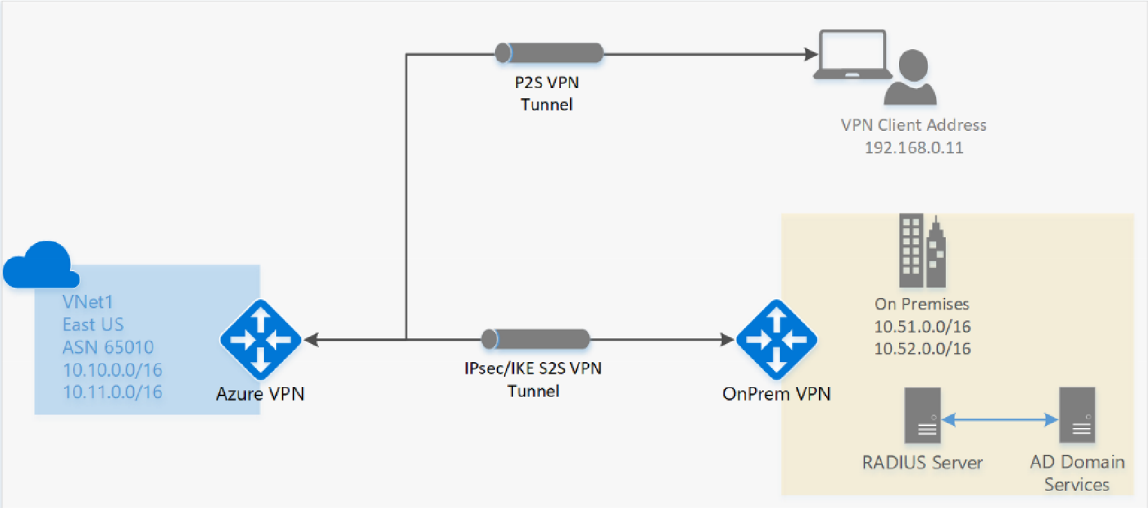


Figure 9: VPN S2S and P2S diagram [7]

2.3.6 Encryption

In terms of encryption, it is necessary to distinguish between three main approaches, in transit, at the host and at rest."At res" approach is done via symetric encryption and its focus is on actively running infrastructure resources, which are VM's OS discs and whole scale sets (IaaS), databases (DaaS), etc. To secure encryption keys, they should be stored encrypted, too. This process is called envelope encryption, which involves encrypting encryption keys. These are then stored in secrets storage with very limited user access defined via strict RBAC policies. Encryption fragmentation or partitioning is used to speed up the process and make it more user-friendly. For example, the VM OS disc is encrypted into multiple partitions with several different keys. Once there is demand for encryption or decryption, it runs in parallel. Envelope encryption and partitioning also prevent attackers from decrypting data easily, which makes it more or less impossible. [28] "At host", the approach is almost the same as at rest, but with a slight difference in the encryption target, which is the whole host running VMs. [29]

Encryption in transit is used in communication between several parties.

- When a client machine communicates with a server
- When a server communicates with another server
- When a server communicates with a non-cloud platform server (for example, Exchange Online delivering email to a third-party email server)

and is done via TLS which is common, obligatory approach. [30]

2.3.7 SaaS policies

Azure Security Center 10 is a security management tool provided by Microsoft Azure that helps customers protect their Azure resources. It provides a centralized view of security alerts and recommendations and allows customers to take actions to remediate potential threats. Azure Security Center also integrates with third-party security solutions, such as antivirus software and firewalls, to provide a more comprehensive security posture. [31]

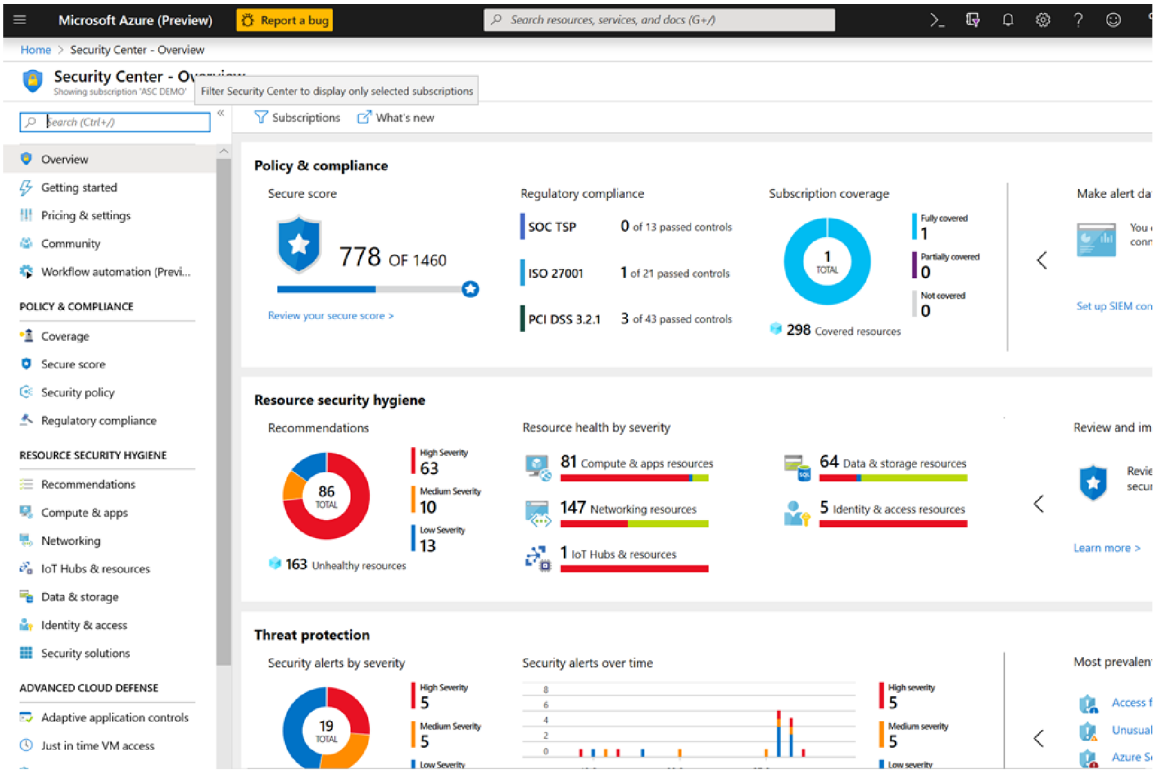


Figure 10: Azure security center [8]

The Google Security and Trust Center is a resource provided by Google Cloud that helps customers understand and implement security best practises for Google Cloud products. It includes a wide range of resources, including documentation, case studies and security assessments, to help customers understand the security features of Google Cloud products and how to use them effectively. Google Security and Trust Center also includes a security incident response team that is available to assist customers in the event of a security breach. [32]

Amazon Inspector 11 is a security assessment tool provided by Amazon Web Services (AWS) that helps customers identify potential security issues in their AWS resources. It performs a range of checks, including identifying insecure network configurations and vulnerable software versions and provides recommendations for remediation. Amazon Inspector can be used on an ad-hoc basis or as part of a regular security assessment schedule. [9]

In terms of security features, all three of these tools offer a range of capabilities to help customers protect their resources. Azure Security Center and Google Security and Trust Center both provide centralized security management and integration with third-party security solutions, while Amazon Inspector focuses on identifying potential security

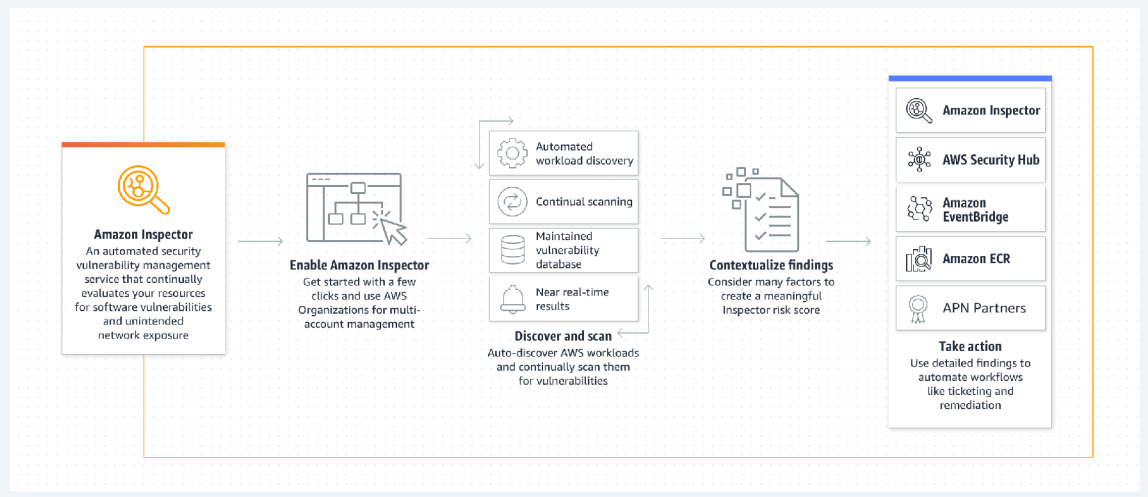


Figure 11: Amazon inspector [9]

issues and providing recommendations for remediation. Ultimately, the choice of which tool to use will depend on the specific needs and requirements of the customer.

2.3.8 Globally known standards

ISO 27001 is an international standard that outlines the requirements for an information security management system (ISMS). It provides a framework for establishing, implementing, maintaining and continually improving information security. The standard is intended to help organizations protect their sensitive data and ensure the confidentiality, integrity and availability of their information systems. One of the key features of ISO 27001 is its focus on risk management. The standard requires organizations to assess the risks to their information systems and implement controls to address those risks. This can include technical controls, such as firewalls and encryption, as well as non-technical controls, such as employee training and policies and procedures. ISO 27001 also requires organizations to regularly review and assess their information security controls to ensure that they are effective and up-to-date. To become certified to ISO 27001, organizations must demonstrate that they have implemented an ISMS that meets the requirements of the standard. This typically involves conducting a gap analysis to identify any areas where the organization's current practices do not align with the standard and implementing the necessary changes to bring the organization into compliance. Once the organization has implemented the necessary controls, it can undergo an audit by an accredited certification body to determine whether it is ready for certification. [33]

PCI DSS is a set of security standards designed to ensure that all companies that accept, process, store, or transmit credit card information maintain a secure environment. The Payment Card Industry Security Standards Council is in charge of these rules. The council is made up of major credit card companies like Visa, Mastercard and American Express.

PCI DSS consists of six main categories of requirements, known as the "PCI DSS Control Objectives":

- Build and Maintain a Secure Network: This includes requirements for protecting the network infrastructure, such as using firewalls to secure network access.
- Protect Cardholder Data: This includes requirements for protecting sensitive data, such as encrypting data transmission and storing data securely.
- Maintain a Vulnerability Management Program: This includes requirements for identifying and addressing vulnerabilities in the network and applications.
- Implement Strong Access Control Measures: This includes requirements for controlling access to cardholder data, such as requiring unique user IDs and strong passwords.
- Regularly Monitor and Test Networks: This includes requirements for monitoring and testing the network to identify security issues and ensure that controls are effective.
- Maintain an Information Security Policy: This includes requirements for establishing and maintaining an information security policy that outlines the company's commitment to protecting cardholder data.

To become compliant with PCI DSS, companies must demonstrate that they have implemented the necessary controls to meet the requirements of the standard. Usually, this means filling out a self-assessment questionnaire (SAQ) and letting a qualified security assessor (QSA) do an audit from the outside.

2.3.9 Cloud auditing

Cloud auditing refers to the process of reviewing and verifying the security and compliance of cloud computing environments. This can include reviewing the configuration of cloud resources, assessing the security of data stored in the cloud and verifying that the organization is adhering to relevant laws and regulations. Cloud auditing is an important part of ensuring the secure and compliant use of cloud computing services.

On Azure, cloud auditing can be performed using Azure Audit Logs. Azure Audit Logs provide a record of activity within the Azure environment, including actions taken by users, system events and resource changes. This information can be used to track changes to resources, identify security issues and verify compliance with policies and standards. Azure Audit Logs are stored in a central repository and can be accessed using the Azure portal, Azure Monitor, or the Azure Log Analytics API.

On Google Cloud, cloud auditing can be performed using Cloud Audit Logs. Cloud Audit Logs provide a record of activity within the Google Cloud environment, including actions taken by users, system events and resource changes. This information can be used to track changes to resources, identify security issues and verify compliance with policies and standards. Cloud Audit Logs can be accessed through the Google Cloud Console or the Cloud Audit Logs API.

On AWS, cloud auditing can be performed using AWS CloudTrail. AWS CloudTrail is a service that provides a record of activity within the AWS environment, including actions taken by users, system events and resource changes. This information can be used to track changes to resources, identify security issues and verify compliance with policies and standards. AWS CloudTrail can be accessed through the AWS Management Console or the CloudTrail API.

2.4 Conslusion

It is used AWS for the purpose of this master's thesis because it has the highest shares on the cloud market, AWS cloud services are the first ones to have started with cloud computing, so many bugs on AWS cloud are already resolved in comparison with the other cloud providers; and it has the most updated documentation.

3 Architecture of the system

This chapter presents the architecture of the distributed system designed in this thesis. The objective of this chapter is to provide a detailed explanation of the system's infrastructure design and the integration of cloud services to build the system. The architecture of the system was developed based on the knowledge gathered from the research conducted on each particular component of the system. The design of the system's infrastructure was focused on the microservices architecture design pattern and cloud resources, which provides scalability and availability. The chapter provides a comprehensive overview of the architectural decisions made during the design process, including the selection of infrastructure components and the configuration of the system's components. Furthermore, the chapter elaborates on the implementation of the infrastructure design, including the integration of cloud services and the development of a deployment strategy. Finally, this chapter concludes with an evaluation of the architecture's effectiveness in meeting the requirements and objectives of the system design.

3.1 Infrastructure as a code

Infrastructure as a Code (IaaC) is a practice that allows for the management of infrastructure resources, such as virtual machines, network settings and storage services, through code, rather than manual configuration processes. This helps to automate the deployment, management and scaling of IT infrastructure, improving the speed and consistency of infrastructure provisioning and reducing the risk of human error. IaaC also enables version control and collaboration, making it easier to track changes to infrastructure over time and collaborate with others on the development and management of infrastructure. The code used in IaaC is typically written in a high-level programming language and can be integrated into a broader Developers/Operations (DevOps) workflow, making it possible to automate the entire software delivery process, from development to production.

3.1.1 Terraform

Terraform is an open-source IaaC tool that allows users to define and manage infrastructure resources as code. It supports a wide range of popular cloud providers, including AWS, Google cloud (GC) and Azure cloud (AC), as well as on-premise and other infrastructure. With Terraform, infrastructure is described using a high-level configuration language called HashiCorp Configuration Language (HCL), which can be version controlled and shared among teams.

Terraform automates the creation, update and deletion of infrastructure resources by using provider APIs. This allows for the management of complex infrastructure in a predictable and consistent manner, reducing the risk of errors and making it easier to roll back changes if necessary. Terraform also provides a visualization of the resources it manages and the dependencies between them, making it easier to understand and manage large-scale infrastructure.

In summary, Terraform provides a unified way to manage and automate the provisioning of infrastructure, making it easier to build, change and version infrastructure as code. [34]

3.1.2 AWS CloudFormation

AWS CloudFormation is an IaaC service that helps to automate the deployment, management and scaling of AWS resources. It provides a simple way to create and manage AWS infrastructure in a predictable and repeatable manner. The difference between Terraform and AWS CloudFormation is, that AWS CloudFormation is fully integrated with AWS platform and cannot be used with other cloud providers. Terraform is much more universal and not vendor locking.

With AWS CloudFormation, infrastructure is defined using templates written in either JSON or YAML, which describe the desired state of the infrastructure. These templates can be version controlled, shared among teams and reused across multiple environments. Once a template is written, CloudFormation uses it to create and manage the defined AWS resources, such as EC2 instances, S3 buckets and Virtual private cloud (VPC)s.

AWS CloudFormation also provides features for monitoring, updating and rolling back changes to the infrastructure. This makes it easier to make changes to the infrastructure and track the history of those changes. Additionally, CloudFormation integrates with other AWS services, such as AWS CodePipeline and AWS CloudTrail, to provide a complete end-to-end solution for automating infrastructure deployment and management.

In summary, AWS CloudFormation is a tool that allows users to model and set up users AWS resources and manage infrastructure as code. It provides a centralized way to manage and automate the deployment, updates and deletion of AWS resources, making it easier to build, change and version infrastructure as code. [35]

3.1.3 Azure resource manager

Azure Resource Manager (ARM) is the deployment and management service for AC computing platform. It provides a way to automate the deployment, management and monitoring of Azure resources, such as virtual machines, storage accounts and network resources.

With ARM, infrastructure is defined using templates written in JSON, which describe the desired state of the infrastructure. These templates can be version controlled, shared among teams and reused across multiple environments. ARM uses these templates to create and manage the defined Azure resources in a consistent and repeatable manner.

ARM also provides features for monitoring, updating and rolling back changes to the infrastructure. This makes it easier to make changes to the infrastructure and track the history of those changes. ARM also integrates with other Azure services, such as Azure DevOps and Azure Policy, to provide a complete end-to-end solution for automating infrastructure deployment and management. [36]

3.1.4 Google cloud deployment manager

Google Cloud Deployment Manager is a service for deploying and managing infrastructure in Google Cloud Platform (GCP). It provides a simple way to create and manage GCP resources in a predictable and repeatable manner.

With Cloud Deployment Manager, infrastructure is defined using templates written in YAML, which describe the desired state of the infrastructure. These templates can be version controlled, shared among teams and reused across multiple environments. Cloud Deployment Manager uses these templates to create and manage the defined GCP resources, such as Compute Engine instances, Cloud Storage buckets and Google Kubernetes Engine clusters.

Cloud Deployment Manager also provides features for monitoring, updating and rolling back changes to the infrastructure. This makes it easier to make changes to the infrastructure and track the history of those changes. Additionally, Cloud Deployment Manager integrates with other GCP services, such as Google Cloud Build and Stackdriver, to provide a complete end-to-end solution for automating infrastructure deployment and management. [37]

3.1.5 Summary

Despite the fact, AWS is being used for the practical part of the master's thesis, Terraform is the most suitable IaaC tool because it is universal and not vendor-locked in case the customer would want to use a specific platform in terms of ownership and control over it. Terraform also provides a wide range of modules for AWS and has descriptive and updated documentation and a large community too.

3.2 Helm package manager

Helm is a package manager for Kubernetes that allows users to easily deploy, manage and version their Kubernetes applications. With Helm, users can define their applications as packages, called charts, which include all the necessary Kubernetes resources, such as deployments, services and ingress controllers. It provides a simple and consistent way to install, upgrade and delete applications and allows users to manage dependencies between applications. Helm charts can be easily shared and reused, making it easier to collaborate and manage applications across teams and organizations. Helm also supports versioning and rollback of applications, which allows users to easily switch between different versions of an application, or roll back to a previous version if there are issues with a new deployment. This makes it easier to manage and maintain complex applications on Kubernetes. It has become a widely adopted standard for deploying applications on Kubernetes. It provides a simple and consistent way to package, deploy and manage Kubernetes applications and has a large and active community that contributes to the development and maintenance of Helm charts. It provides a standardized format for packaging and distributing Kubernetes applications, which makes it easier to share and reuse application packages across teams and organizations. It also provides a consistent and repeatable deployment process, which makes it easier to deploy and manage applications across different environments, such as development, testing and production. Helm has a large and active community, with a wide range of pre-built charts available for popular applications and services. Users can also create their own charts to define their own applications or to extend existing ones. [38]

3.2.1 Helm umbrella charts

An umbrella chart is a type of Helm chart that allows users to package multiple charts together as a single unit. This is useful for managing and deploying complex applications that are composed of multiple components or services. The umbrella chart includes a parent chart, which provides a top-level template and a set of values that are shared across all the child charts.

The child charts are typically independent Helm charts that define the different components or services of the application. The umbrella chart allows users to deploy all the child charts at once, with a single command and manage them as a single unit. This makes it eas-

ier to manage and deploy complex applications that are composed of multiple components or services. [39]

3.3 Kubernetes

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF).

Kubernetes provides a way to manage containers at scale, making it easier to deploy, manage and scale applications across a cluster of hosts. It uses a declarative model, where the desired state of the application is defined in configuration files called manifests. Kubernetes then takes care of reconciling the actual state of the application with the desired state, making any necessary changes to the cluster to ensure the desired state is met.

Kubernetes provides features for container management, such as automated deployment and scaling, self-healing and rollouts and rollbacks. It also provides features for networking, storage, security and monitoring, making it a complete platform for running containerized applications in production. [40]

There are several implementations of hosting Kubernetes on AWS which are described bellow.

3.3.1 Kubernetes control plane

The Kubernetes control plane is a set of components that manage the state of a Kubernetes cluster. It consists of several key components that work together to provide a platform for running and managing containerized applications.

The components of the Kubernetes control plane include:

- etcd: This is a distributed key-value store that stores the configuration data and state of the cluster. All Kubernetes objects, such as Pods, Services, and ConfigMaps, are stored in etcd.
- API server: The API server is the control plane component that provides a RESTful API for users and other components to interact with the cluster. It serves as the interface for all communication between the control plane and the nodes.
- Scheduler: The Scheduler is responsible for assigning Pods to nodes in the cluster. It takes into account the resource requirements of the Pod and the available resources on each node.
- Controller Manager: The Controller Manager is responsible for managing the various controllers that are responsible for maintaining the desired state of the cluster. It includes controllers for ReplicaSets, Deployments, and Services.
- Cloud Controller Manager: The Cloud Controller Manager provides a way for Kubernetes to interact with various cloud providers. It includes controllers for managing resources such as Load Balancers, Volumes, and Nodes.

Together, these components provide the necessary functionality for running and managing containerized applications on a Kubernetes cluster.

3.3.2 EKS

EKS is a fully managed service by AWS that makes it easier to run Kubernetes on AWS. EKS eliminates the need for users to manage their own Kubernetes control plane, as AWS takes care of this for them.

EKS is designed to be highly available, scalable and secure, making it a good fit for running production workloads. With EKS, users can create and manage Kubernetes clusters in EKS, including worker nodes, load balancers and networking. EKS integrates with other AWS services, such as Elastic Load Balancing, Amazon VPC and AWS IAM, to provide a seamless and secure Kubernetes experience.

EKS also supports the latest versions of Kubernetes, as well as popular Kubernetes tools and extensions, such as kubectl and Helm.

In summary, AWS EKS is a fully managed service that provides a simple, scalable and secure way to run Kubernetes on AWS. [41]

3.3.3 ECS

ECS is a fully managed container orchestration service by AWS. ECS allows users to easily run and scale containerized applications on AWS, without having to manage the underlying infrastructure.

With ECS, users can use Docker images to create containers and run them as tasks. These tasks are then grouped together into services, which can be load balanced and scaled automatically. ECS integrates with other AWS services, such as Elastic Load Balancing, Amazon VPC and AWS IAM, to provide a seamless and secure container experience.

ECSalso provides features for monitoring and logging, making it easier to troubleshoot issues and optimize the performance of users containers. Additionally, ECS integrates with other AWS services, such as AWS Fargate and AWS Batch, to provide more flexibility in how users run and manage users containerized workloads. [42]

3.3.4 Fargate instances

AWS Fargate is a compute engine for containers that allows users to run containers without having to manage the underlying infrastructure. With Fargate, users can run containers as a fully managed service, without having to provision, configure, or scale virtual machines.

Fargate allows users to launch containers in a matter of seconds, without having to worry about underlying infrastructure or server management. Users simply define containers and resources and Fargate takes care of launching and managing them on their behalf. This makes it easy to run and scale users applications, without having to worry about the underlying infrastructure.

Fargate is integrated with other AWS services, such as Elastic Load Balancing, Amazon VPC and AWS IAM, to provide a seamless and secure container experience. Fargate also provides features for monitoring and logging, making it easier to troubleshoot issues and optimize the performance of users containers. [43]

3.3.5 Summary

There are several advantages and disadvantages to each of the products described above. The main reason to use EKS over the other two managed solutions is its flexibility and Helm package manager support. Helm brings standardisation in terms of application deployment;

hence, many companies require standardised environments and do not like vendor locking. If a company wants to deploy applications on customer premises and in the cloud too, there is no other standard and widely used technology than Helm in combination with Kubernetes. This policy provides the company with a universal solution for easy deployment on a variety of Kubernetes-powered environments.

3.4 Autoscaling

Autoscaling is a feature of cloud computing services that allows users to automatically adjust the number of computing resources (such as virtual machines, containers, or servers) in response to changes in demand for their application or workload. The goal of autoscaling is to ensure that they have enough computing resources to handle the workload, without paying for more resources than are needed.

Autoscaling typically works by setting up policies that define thresholds for various metrics, such as CPU usage, network traffic, or queue length. When a threshold is breached, the autoscaling system will automatically provision additional resources to handle the increased workload. Similarly, if the workload decreases, the autoscaling system will automatically deprovision resources to save costs.

There are two main types of autoscaling:

- Vertical autoscaling: This involves adding or removing resources (such as memory, CPU or storage) to a single instance in response to changes in demand. This type of autoscaling is often used for applications that require large amounts of processing power or memory.
- Horizontal autoscaling: This involves adding or removing entire instances (such as virtual machines or containers) in response to changes in demand. This type of autoscaling is often used for applications that have a high number of requests, where adding more instances can help to distribute the workload.

3.4.1 Kubernetes built-in autoscaler and its integration with unoptimized applications

The built-in autoscaler in K8 is a feature of the Kubernetes engine. Its main feature is horizontal autoscaling, which automatically updates instances of workloads based on metrics from the K8s metrics server. Autoscaling is based on two metrics, which are provided by the metrics server: CPU load and memory load. These two metrics are the most suitable for 80% of optimised application loads that were developed to run Kubernetes or cloud native. If a company wants to transform its native Linux applications to the cloud, it can face multiple issues where the application does not provide real metric values.

One of the examples could be Java heap allocation. If Kubernetes engineers define a hard limit for memory allocation in a single Kubernetes The smallest deployable unit that represents a single instance of a running process in k8s cluster (POD), a Java application inside can easily allocate memory until it reaches the hard limit of the POD itself. Memory is allocated to its maximum limit despite the fact that it is not used inside the POD, just allocated. The metric server provides this allocation value, 90% of the memory used to the autoscaler and it triggers autoscaling. This can affect the cost-efficiency of the cluster; hence, it is overscaled and not really used at all. [44]

3.4.2 KEDA

Kubernetes Event-driven Autoscaling (KEDA) is a popular open-source project and Kubernetes operator that enables event-driven autoscaling for containerized workloads running in Kubernetes.

KEDA is designed to help cloud engineers scale Kubernetes workloads based on various events, such as the number of messages in a queue or the number of events in a stream. It allows users to automatically scale workloads up or down based on the number of incoming events or messages, so users can handle increased demand without wasting resources when there are fewer events to process.

KEDA can work with various event sources such as Kafka, Azure Service Bus, RabbitMQ and Prometheus and it can be integrated with Kubernetes native autoscaling, such as the Horizontal Pod Autoscaler (HPA), to provide a powerful and flexible autoscaling solution for Kubernetes workloads. By using KEDA, cloud engineers can ensure that applications are always running at the optimal level of performance and resource utilization, while also reducing operational overhead and cost. [10]

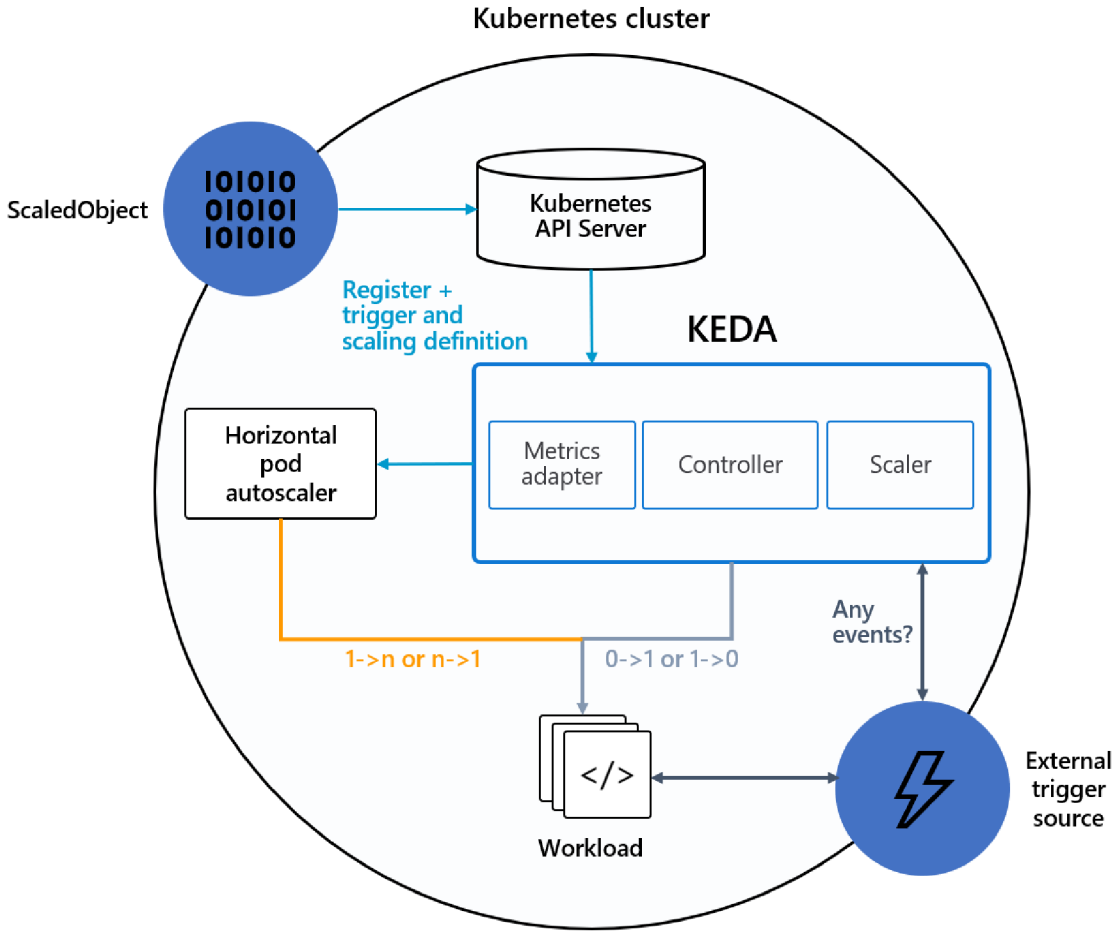


Figure 12: KEDA architecture diagram [10]

3.5 AWS DaaS solutions

Amazon provides multiple DaaS solutions:

- Relational - Traditional applications, enterprise resource planning (ERP), customer relationship management (CRM), ecommerce - Amazon Aurora, Amazon RDS, Amazon Redshift
- Key-values - High-traffic web applications, ecommerce systems, gaming applications - Amazon DynamoDB
- In-Memory - Caching, session management, gaming leaderboards, geospatial applications - Amazon ElastiCache, Amazon MemoryDB for Redis
- Document -Content management, catalogs, user profiles - Amazon DocumentDB (with MongoDB compatibility)
- Wide column - High-scale industrial apps for equipment maintenance, fleet management and route optimization - Amazon Keyspaces
- Graph - Fraud detection, social networking, recommendation engines - Amazon Neptune
- Time series - Internet of Things (IoT) applications, DevOps, industrial telemetry - Amazon Timestream
- Ledger - Systems of record, supply chain, registrations, banking transactions - Amazon Ledger Database Services (QLDB)

Consider the following for the purposes of this thesis: Amazon RDS with PostgreSQL, Amazon Aurora with PostgreSQL and Amazon Neptune. The first two are suitable, hence, it has got the support of PostgreSQL and the third one is designed for fraud detection solutions. [45]

3.5.1 Amazon RDS with PostgreSQL

With Amazon RDS for PostgreSQL, users can launch a managed PostgreSQL database instance in minutes, without the need to manually provision hardware, install software, or configure backups and maintenance tasks. Amazon RDS for PostgreSQL offers a range of features and benefits, including:

- Automated backups and point-in-time recovery: Amazon RDS automatically backs up your PostgreSQL database and enables you to recover your database to any point in time within your retention period.
- Scalability and high availability: Amazon RDS allows you to scale your PostgreSQL database up or down with a few clicks and provides built-in fault tolerance and high availability features.
- Security and compliance: Amazon RDS provides several security and compliance features, including network isolation, encryption at rest and integration with AWS Identity and Access Management (IAM).

- Monitoring and management: Amazon RDS provides detailed monitoring and management capabilities through its integration with AWS CloudWatch and the AWS Management Console.
- Read replicas: Amazon RDS allows you to create one or more read replicas of your PostgreSQL database for read-heavy workloads, providing improved read performance and scalability.

Amazon RDS for PostgreSQL supports several versions of PostgreSQL, including the latest version and provides several configuration options, including instance size, storage type and performance metrics. [46]

3.5.2 Amazon Aurora with PostgreSQL

Amazon Aurora is a high-performance, fully-managed relational database engine offered by AWS. It is designed to provide compatibility with popular open-source databases, including PostgreSQL, without sacrificing performance or availability.

Amazon Aurora with PostgreSQL is a version of Amazon Aurora that is compatible with PostgreSQL and provides many of the same benefits and features as the standard Aurora offering. These include:

- Performance and scalability: Amazon Aurora is designed to provide high performance and scalability, with low latency and high throughput for both read and write operations. It is also highly scalable, allowing you to add or remove capacity with ease.
- High availability: Amazon Aurora provides automatic failover and built-in replication to ensure high availability of your database, with minimal downtime.
- Security and compliance: Amazon Aurora provides several security and compliance features, including network isolation, encryption at rest and in transit and integration with AWS Identity and Access Management (IAM).
- Monitoring and management: Amazon Aurora provides detailed monitoring and management capabilities through its integration with AWS CloudWatch and the AWS Management Console.
- Compatibility with PostgreSQL: Amazon Aurora with PostgreSQL is designed to be compatible with PostgreSQL, allowing you to use many of the same tools and applications that you would use with a standard PostgreSQL database.
- Amazon Aurora with PostgreSQL also provides some additional features that are specific to PostgreSQL, such as support for advanced PostgreSQL features like JSONB and HSTORE and compatibility with popular PostgreSQL tools like pgAdmin.

[47]

3.5.3 Aurora VS. Amazon RDS

Amazon Aurora with PostgreSQL and Amazon RDS with PostgreSQL are both managed database services offered by AWS, but there are some key differences between the two:

- Performance: Amazon Aurora with PostgreSQL is designed to provide high performance and scalability, with low latency and high throughput for both read and write operations. It uses a unique storage architecture that allows it to achieve up to five times the performance of a standard PostgreSQL database, making it an ideal solution for applications that require high performance. In contrast, Amazon RDS with PostgreSQL provides good performance, but may not be able to match the performance of Aurora.

- High Availability: Both Amazon Aurora with PostgreSQL and Amazon RDS with PostgreSQL offer high availability, with automatic failover and built-in replication. However, Aurora provides faster and more seamless failover than RDS and also provides an active-active option for multi-region deployments.

- Scalability: Amazon Aurora with PostgreSQL is highly scalable, allowing you to add or remove capacity with ease, while Amazon RDS with PostgreSQL is less scalable, particularly for write-heavy workloads.

- Cost: Amazon RDS with PostgreSQL is generally less expensive than Amazon Aurora with PostgreSQL, particularly for smaller database instances. However, the cost of Aurora can be more competitive for larger database workloads, particularly when you factor in the potential performance benefits.

- Features: Amazon Aurora with PostgreSQL and Amazon RDS with PostgreSQL both offer many of the same features, including automated backups, security and compliance, monitoring and management and read replicas. However, Aurora provides some additional features that are specific to Aurora, such as the ability to scale to multiple regions and the ability to create up to 15 read replicas.

Features: Amazon Aurora with PostgreSQL and Amazon RDS with PostgreSQL both offer many of the same features, including automated backups, security and compliance, monitoring and management and read replicas. However, Aurora provides some additional features that are specific to Aurora, such as the ability to scale to multiple regions and the ability to create up to 15 read replicas.

3.5.4 Amazon Neptune

Amazon Neptune is a fully-managed graph database service offered by AWS. It is designed to be highly available, scalable and secure and is optimized for storing and querying highly connected data with complex relationships.

Some key features of Amazon Neptune include:

- Graph database functionality: Amazon Neptune is designed specifically for graph databases, which are used to model and query data with complex relationships. It supports popular graph query languages such as Gremlin and SPARQL and provides many graph-specific features, such as node and edge labels, property indexes and traversal caching.

- High availability and scalability: Amazon Neptune is designed to be highly available and scalable. It uses a distributed architecture that allows it to automatically replicate data across multiple Availability Zones and it can scale to support large-scale graph workloads.

- Security and compliance: Amazon Neptune provides several security and compliance features, including network isolation, encryption at rest and in transit and integration with AWS Identity and Access Management (IAM).
- Integration with other AWS services: Amazon Neptune integrates with other AWS services, such as AWS Lambda, Amazon S3 and Amazon CloudWatch, allowing you to build complex graph-based applications using a range of AWS tools and services.
- Fully managed: Amazon Neptune is a fully-managed service, which means that AWS takes care of the administrative tasks associated with running and maintaining a graph database. This allows you to focus on building your application and working with your data, rather than managing the underlying infrastructure.

[48]

3.6 Infrastructure layer architecture

Regarding the technological stack described above, a summary about the architecture could be made. Requirements are as follows:

- Application should be accessible at the location where it is used by clients and response time should be as low as possible as defined by the network throughput.
- Application should fulfill SLA of 99,999%.
- Application should be able to scale up in response to the amount of traffic and the number of requests that arrive at the load balancer or gateway.
- Application should implement standards for self-healing mechanisms to prevent outages.

As shown in the diagram 13, this architecture may meet requirements.

3.7 CI/CD

Continuous Integration/Continuous Deployment (CI/CD) is a set of practices and processes that enable organizations to deliver software quickly and reliably by automating the building, testing and deployment of code changes.

Continuous Integration (CI) involves automatically building and testing code changes as soon as they are committed to the source code repository, to detect errors early in the development cycle and ensure that new changes do not break the existing codebase. This process can include tasks such as code compilation, unit testing and static analysis.

Continuous Deployment (CD) goes one step further by automating the deployment of the application to production environments, once the code has passed the CI tests. CD can involve tasks such as building and packaging the application, deploying it to a testing environment, performing integration and acceptance testing and finally deploying it to the production environment.

The ultimate goal of CI/CD is to streamline the software development process, reduce the risk of errors and enable faster time-to-market for new features and bug fixes. By automating many of the manual processes involved in software development and delivery, organizations can focus more on developing quality code and less on the logistics of deploying it to production environments. [49]

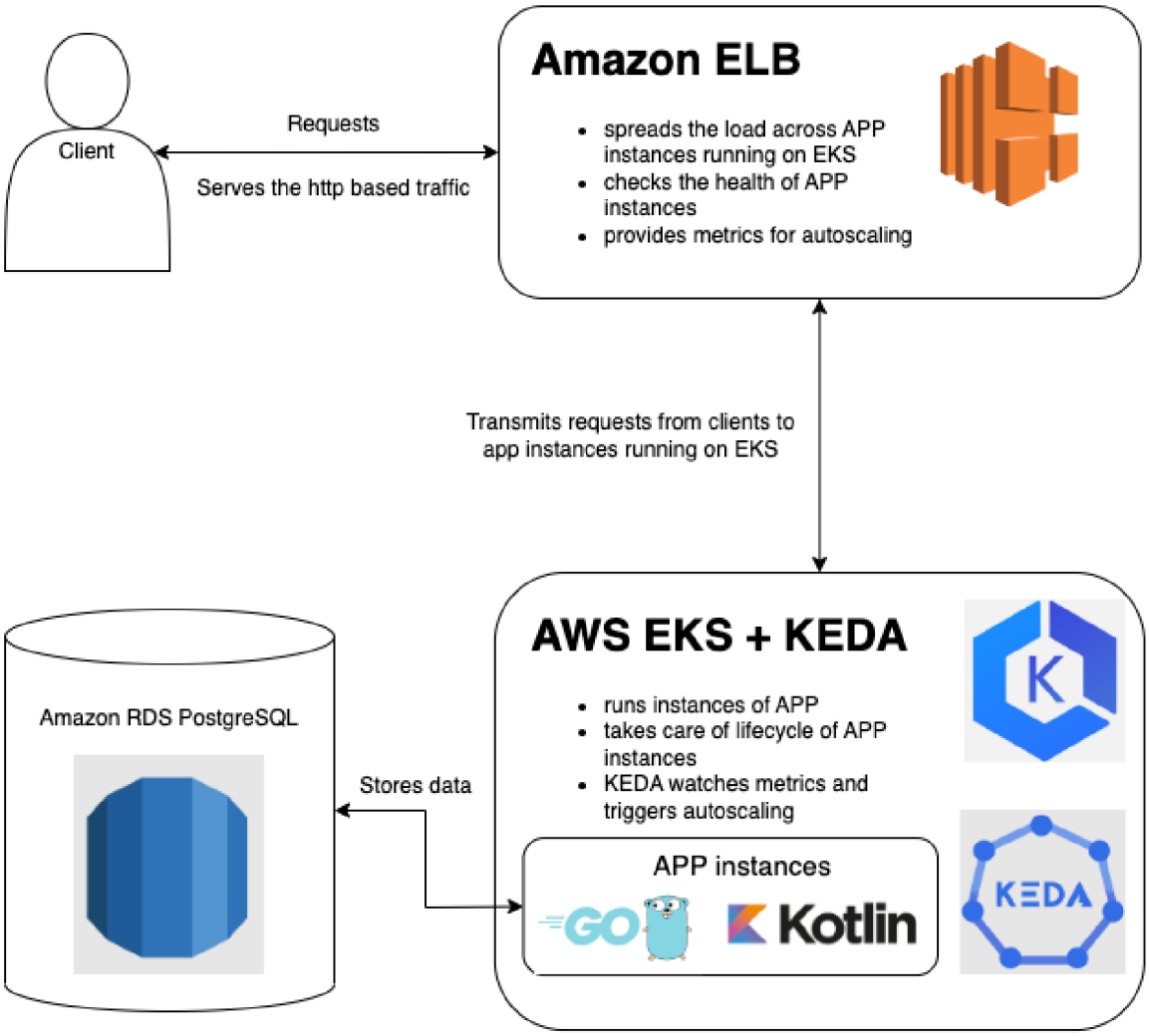


Figure 13: Infrastructure architecture diagram

3.7.1 Github actions

GitHub Actions is a popular cloud-based CI/CD service provided by GitHub. It allows developers to define custom workflows for building, testing and deploying their applications, directly from their GitHub repositories.

With GitHub Actions, developers can create workflows using YAML syntax and automate tasks such as building and testing code, deploying code to different environments and sending notifications based on various triggers such as code commits, pull requests, or issue creation. GitHub Actions supports a wide range of programming languages, tools and frameworks and allows you to use your own tools or choose from a variety of pre-built actions available in the GitHub Marketplace.

GitHub Actions provides a variety of built-in features that make it easy to customize workflows, such as environment variables, secrets and conditional logic. Developers can

also use GitHub Actions to integrate with other third-party tools such as Slack, AWS, or Docker and to orchestrate complex workflows across multiple repositories and projects. [50]

3.7.2 Infrastructure unit tests - Terratest

Terratest is an open-source testing framework for testing infrastructure code, particularly for Terraform. It is designed to help developers test their infrastructure code more thoroughly and automate the testing process, ensuring that their infrastructure deployments are reliable and secure.

Terratest provides a simple and easy-to-use interface for running tests against Terraform code, allowing developers to write automated tests in Go language that can be run in their preferred test runners like Go test, CircleCI or Jenkins. Terratest can test a wide range of cloud resources, including AWS, Azure, Google Cloud and Kubernetes.

Terratest allows developers to test for a variety of scenarios, such as testing the infrastructure code to ensure it provisions resources correctly, validating the network connectivity of resources, testing the deployment of complex infrastructure setups and more. It also provides many helper functions and utility libraries for common tasks, such as spinning up infrastructure resources for testing, programmatically querying cloud APIs and verifying the output of Terraform code.

Overall, Terratest is a powerful and flexible tool for testing infrastructure code, enabling developers to catch issues early and ensure that their deployments are reliable and secure. [51]

3.7.3 Unit tests

Unit testing is a software testing technique that involves writing and executing automated tests for small, individual units of code, typically at the function or method level. The purpose of unit testing is to validate that each unit of code performs as expected, with the aim of identifying and fixing defects as early in the development cycle as possible.

Unit tests are usually written by developers and they are designed to test the behavior of a single function or method in isolation, without any dependencies on external systems, databases, or services. This allows developers to test their code in a controlled and repeatable environment, ensuring that the function or method behaves as expected and meets the requirements specified in the design document.

Unit tests typically use a framework or library to automate the testing process and they are usually executed as part of a CI pipeline, which automates the build, testing and deployment process of the software. Unit tests are an essential part of the software development process, as they help to identify defects early in the development cycle, reducing the cost and time required to fix them.

3.7.4 Code inspection and vulnerability detection - Sonar Cloud

SonarCloud is a cloud-based software quality management platform that provides continuous code inspection and analysis to identify bugs, vulnerabilities and code smells in a variety of programming languages.

SonarCloud is designed to integrate with a wide range of development tools, such as code repositories, issue trackers, build systems and CI/CD pipelines. It analyzes the code for a variety of issues, including code complexity, security vulnerabilities, reliability and maintainability, providing developers with detailed reports on the quality of their code.

The platform uses a combination of static code analysis, dynamic code analysis and machine learning algorithms to detect and prioritize issues. It also offers intelligent code review capabilities, such as pull request analysis and branch analysis, which enables developers to identify issues earlier in the development cycle and fix them before they become larger problems.

In addition, SonarCloud provides features for tracking code quality metrics over time, monitoring the progress of code quality improvements and enforcing code quality standards across a team or organization. It can also integrate with security scanning tools to identify and remediate security vulnerabilities. [52]

3.7.5 ArgoCD

Argo CD is an open-source tool for continuous delivery and deployment of Kubernetes applications. It provides an automated way to deploy and manage applications across multiple environments, such as development, staging and production, with consistent configuration and version control. It is built on top of Kubernetes and uses Git as its source of truth for application deployment configuration. It continuously monitors the Git repository for changes and automatically deploys new versions of the application to the desired target environments when changes are detected. Argo CD uses a declarative approach to managing applications, where the desired state of the application is defined in a YAML file and the tool handles the deployment and synchronization of the actual state with the desired state. It supports various deployment strategies, including rolling updates, blue-green deployments and canary releases, which enable teams to deploy and test new features with minimal downtime and risk. It also provides a web-based user interface, a Command Line Interface (CLI) tool and an API for managing applications, tracking changes and performing rollbacks if needed. It offers features such as RBAC for managing access and permissions, a webhook system for integrating with external tools and automatic syncing of Helm charts, Kustomize overlays and other Kubernetes resources. [11]

The architecture of ArgoCD declarative deployments is shown in the Figure 14.

3.7.6 Release management and deployment strategies

Several deployment strategies used as part of the release process in production environments are listed here, regarding the previous chapter.

- Rolling Deployment: This strategy involves gradually rolling out new versions of an application by updating a subset of instances at a time, while keeping the rest of the instances running the previous version. This allows for gradual testing and validation of the new version, with minimal disruption to the user experience.

- Blue-Green Deployment: This strategy involves deploying two identical environments, one with the current version of the application (the blue environment) and the other with the new version (the green environment). Traffic is gradually routed from the blue environment to the green environment, allowing for thorough testing and validation of the new version before cutting over completely.

- Canary Deployment: This strategy involves deploying a new version of the application to a small subset of users, while keeping the rest of the users on the previous version. This allows for testing and validation of the new version with a smaller audience, before rolling it out to the rest of the users.

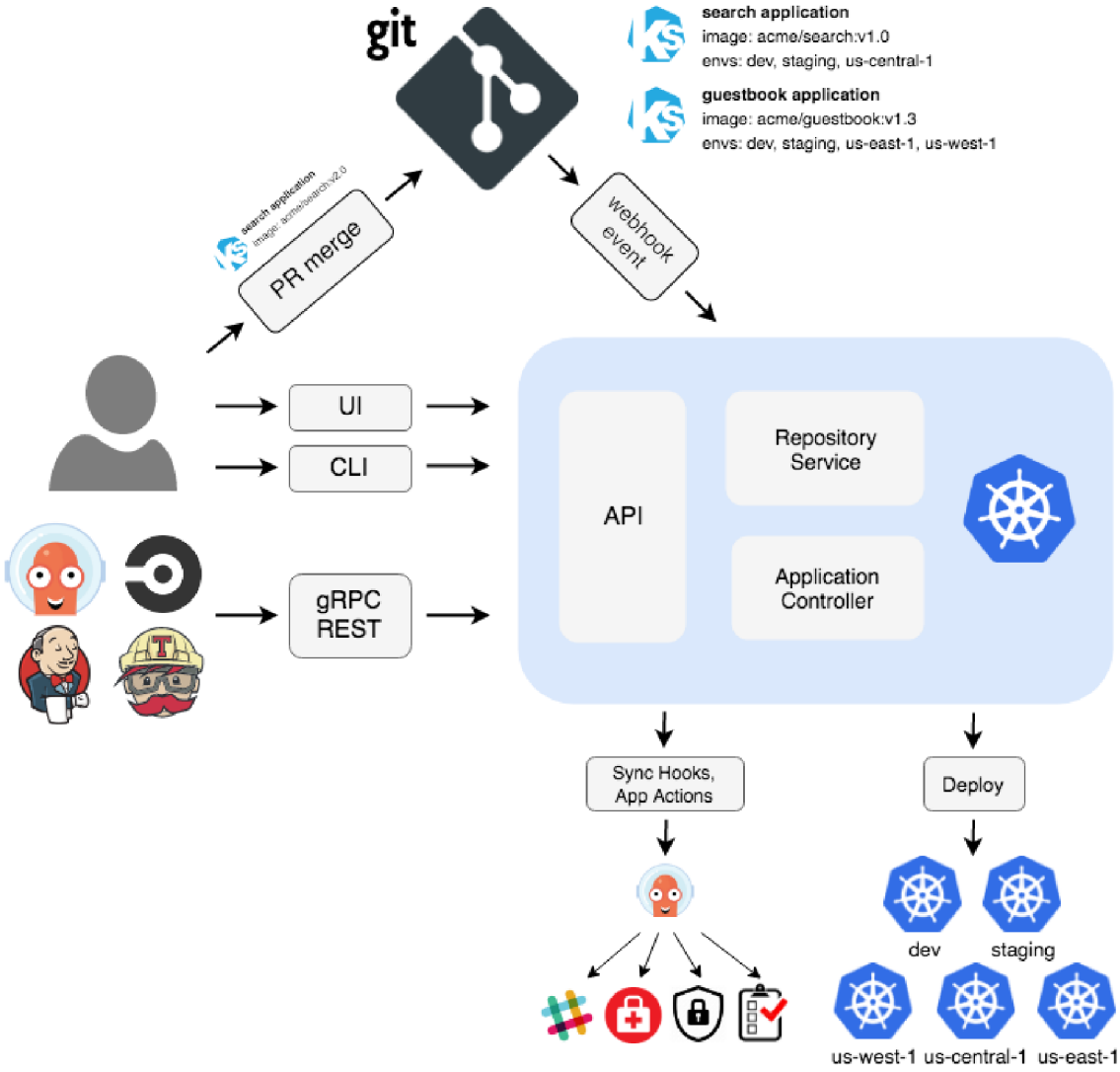


Figure 14: ArgoCD architecture [11]

- A/B Testing: This strategy involves deploying multiple versions of an application and randomly routing traffic to each version, allowing for testing and comparison of different features, designs, or algorithms. This can help optimize the user experience and improve engagement or conversion rates.
- Shadow Deployment: This strategy involves deploying a new version of the application alongside the current version, but not actually routing traffic to the new version. Instead, the new version is used to collect data and monitor behavior, allowing for testing and validation before rolling out the new version to users.
- Feature Toggles: This strategy involves deploying new features or functionality as part of the current version of the application, but hiding them behind a toggle or configuration flag that can be turned on or off. This allows for testing and validation of new features without impacting the user experience or requiring a full deployment.

3.7.7 New feature deployment life-cycle

Once development finishes new feature and test it on their side, there are several steps that should be performed in case of smooth delivery of this feature to the production environment. For this use case, there are three running environments:

- Development Environment: This is the first environment where the application is developed, tested and debugged. Developers work on the code and test it locally or in a shared development environment.
- User Acceptance Testing (UAT) Environment: The UAT environment is a replica of the production environment where the application is deployed for testing by stakeholders such as product owners, business analysts and users. This stage allows for functional and usability testing, as well as validation of non-functional requirements such as performance, security and scalability.
- Production Environment: Once the application has been fully validated in the staging environment, it is deployed to the production environment, where it is made available to users. At this stage, the application is monitored for issues and feedback is collected for continuous improvement.

The whole process is described in the Figure 15.

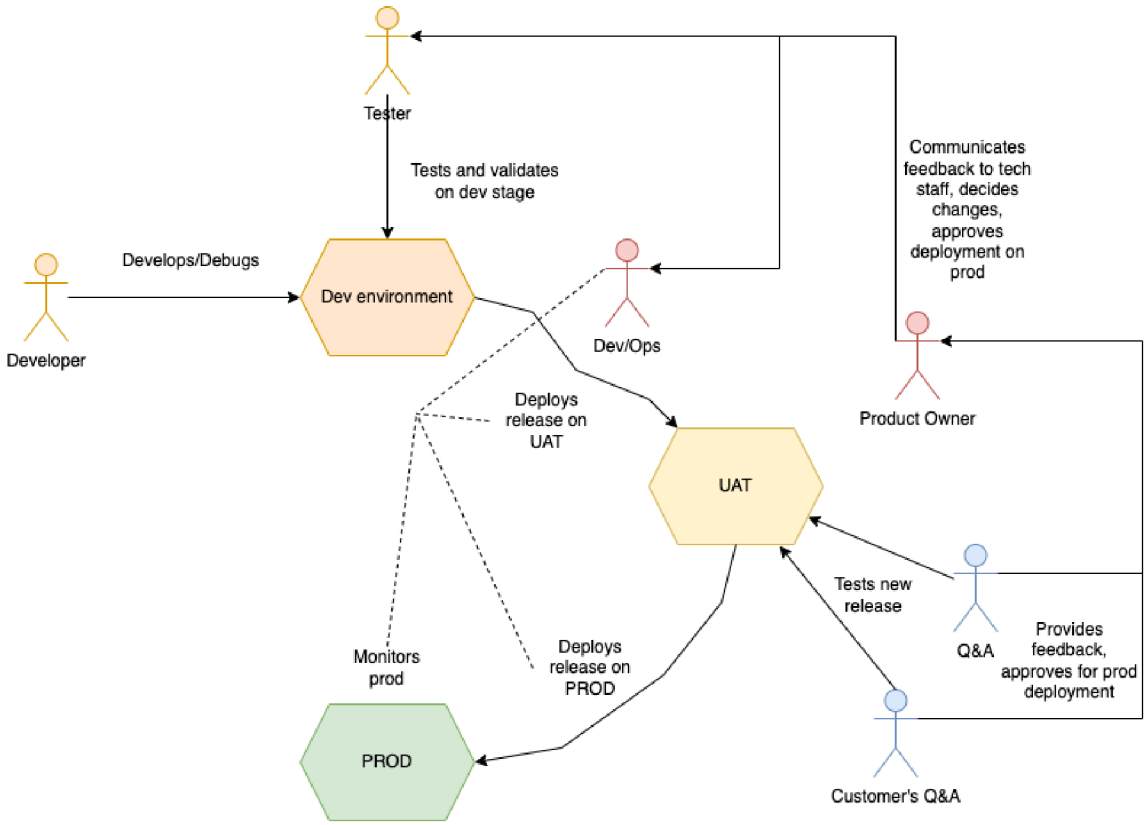


Figure 15: Deployment life-cycle

3.7.8 On demand deployment - demo environments and development environments

On-demand deployment is a deployment model in which resources are provisioned and released based on current demand, rather than being provisioned and allocated for a fixed period. This means that resources, such as compute, storage and network resources, are made available as needed and then released when they are no longer required, resulting in a more efficient use of resources. It is commonly used in cloud computing environments, where resources are provisioned dynamically based on workload requirements. This enables organizations to scale up or down based on demand without having to invest in additional hardware or infrastructure. It can also refer to the process of deploying software applications on an as-needed basis. This can be achieved through the use of automation and orchestration tools, which enable developers to quickly deploy new versions of applications or services in response to changing business requirements. On-demand deployment is widely used for demo environments when company tries to sell the piece of software to their potential customers. It should be quickly deployed based on the delivery team's demand to introduce the application to the customer. The customer is able to test all the delivered features themselves and then sign a potential contract if he decides to go for it. Diagram, which describes whole process is shown in Figure 16.

3.7.9 Production environments

Production environments differ from demo environments by High availability (HA) principles: more scalability, stability and less fluctuation of their instances. It is the live and operational environment where the software is deployed and used by end-users. This is the environment where the application is accessed and used by customers and where any issues or errors can potentially cause significant impact, such as financial losses, reputational damage, or even harm to individuals. Here are the key requirements that should be met by a production environment in comparison with a demo environment.

- Security: Production environments should have strong security measures in place to protect sensitive user data and prevent unauthorized access. This includes firewalls, intrusion detection and prevention systems, data encryption, access controls and more.
- Disaster Recovery: Production environments should have disaster recovery plans and processes in place to ensure business continuity in the event of a system failure or outage. This includes backups and redundancy measures to minimize downtime and data loss.
- Release Management: Production environments should have a robust release management process to ensure that new features, updates and patches are thoroughly tested before being released to end-users. This includes a process for rolling back changes if issues arise.

In the diagram 17 is shown how applications are deployed in a production environment. The difference between demo and production deployment is the step of deployment via ArgoCD that keeps the consistency of production deployments via continuous syncing.

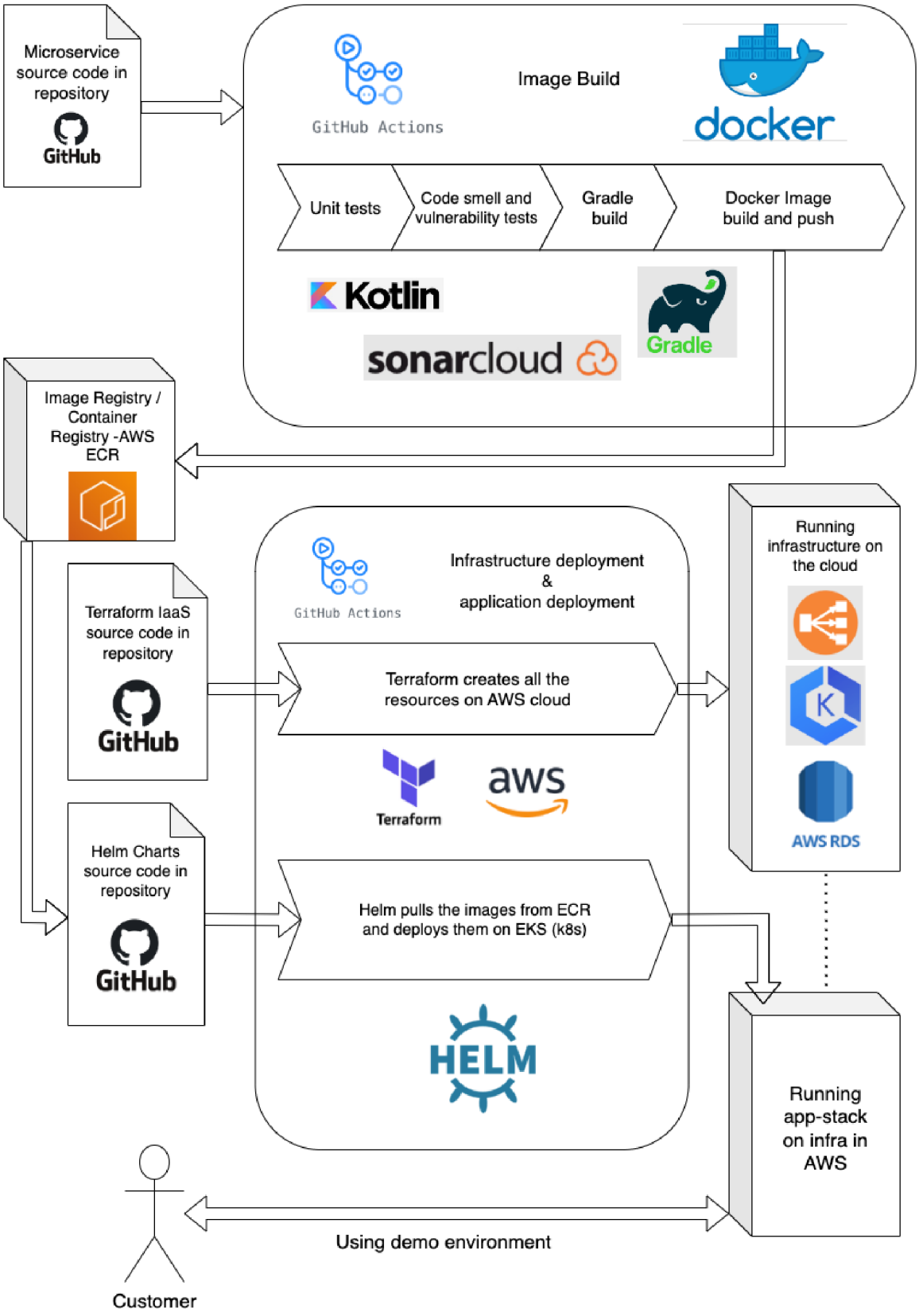


Figure 16: CI/CD demo/dev environments architecture

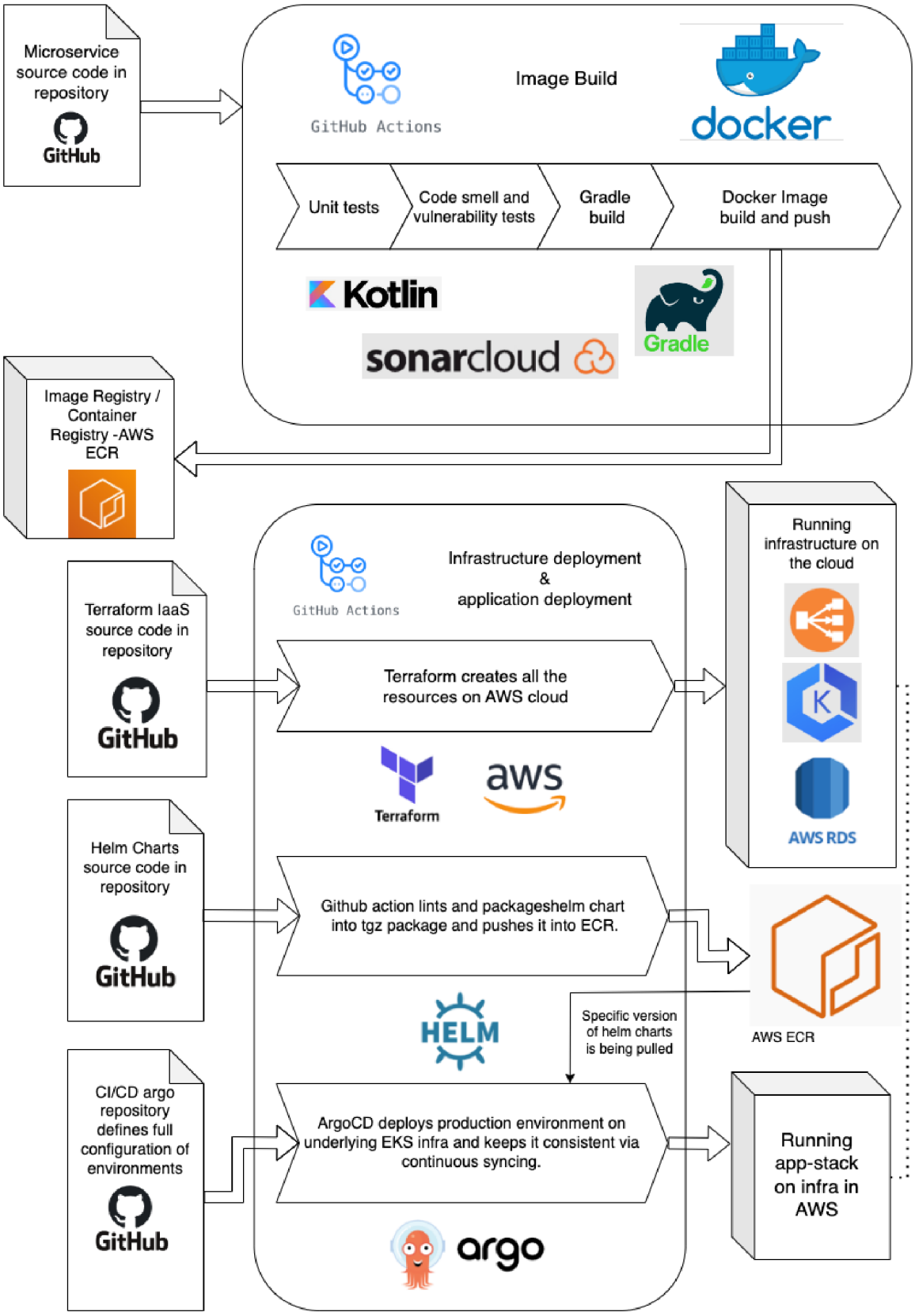


Figure 17: CI/CD production environments architecture

4 Implementation of the system

This chapter presents the implementation details of the distributed system designed in this thesis. The objective of this chapter is to provide a comprehensive overview of the implementation process, including the technical details, challenges faced and the solutions adopted. The implementation of the system was performed using cloud resources. The chapter also provides a detailed explanation of the development process, the testing methodology and the results of the testing. The implementation of the system includes the configuration of the infrastructure and the integration of the system with the cloud provider's services. Finally, this chapter concludes with an evaluation of the implementation and the lessons learned during the development process.

4.1 Terraform implementation and underlying infrastructure

This sub-chapter provides a detailed description of the implementation of the underlying Terraform infrastructure and demonstrates resource definitions through examples. By default, the Terraform script requires a strict structure and includes a few obligatory parts. These include the following:

- Definition of provider
- Variables file
- Definition of backend

The Terraform provider file includes the initial configuration that specifies which cloud infrastructure is being used as the provider. In this Master's thesis, the Terraform provider file contains information about the AWS cloud provider. As seen in the example, there are a few crucial definitions that need to be made. Firstly, the provider used is defined, which provides the driver to use the AWS API to create resources and it also specifies the region where the resources will be created. Secondly, it specifies the versions of the provider and the Terraform framework that will be used.

```
provider "aws" {
  region = var.region
}

terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  required_version = ">= 1.2.0"
}
```

A variables file is a text file that includes definitions for input variables used in the Terraform configuration. Input variables are values that are unknown at the time of writing the Terraform configuration and need to be provided when applying the configuration. By using a variables file, users can define values for input variables separately from the Terraform

configuration. This makes it easier to manage and reuse the same Terraform configuration with different input values.

Here is an example of variables that are used to create the infrastructure.

```
variable "region" {
  type = string
  default = "eu-west-1"
}

variable "availability_zone_1" {
  type = string
  default = "eu-west-1a"
}

variable "availability_zone_2" {
  type = string
  default = "eu-west-1b"
}

variable "environment_name" {
  type = string
  default = "test-demo"
}

variable "vpc_id" {
  type = string
  default = "vpc-047fe7f7538b42dc7"
}

variable "cidr_blocks" {
  type = list(string)
  default = ["","","",""]
}
```

This set of default variables enables the user to specify the region where the infrastructure is created, multiple availability zones, the name of the whole environment and the network ranges for multiple networks that are part of the architecture.

A backend is a configuration that determines how Terraform stores and retrieves state data. The backend file is a Terraform configuration file that specifies the configuration for the backend that Terraform should use. The backend file is usually named backend.tf and it includes a backend block that specifies the backend configuration. Here is an example of a backend configuration:

```
terraform {
  backend "s3" {
    bucket = "tf-states-bucket-<unique_identifier>"
    key = "test-customer-demo-<date-created>-<available-until>.tfstate"
    region = "eu-west-1"
  }
}
```

This backend example specifies the following: it will use an AWS S3 bucket to store the state file, specifies which bucket is being used (the bucket should have a unique name), specifies the name of the state file stored in the bucket and specifies a region.

Once these commands run:

```
terraform init #initialises backend and saves the state file in S3
terraform plan #downloads the content of the current state and
compares it with predefined resources by terraform, outputs
the plan, which contains resources, that should be created
```

the output in the S3 bucket is shown in Figure 18 And the partial example of the terraform

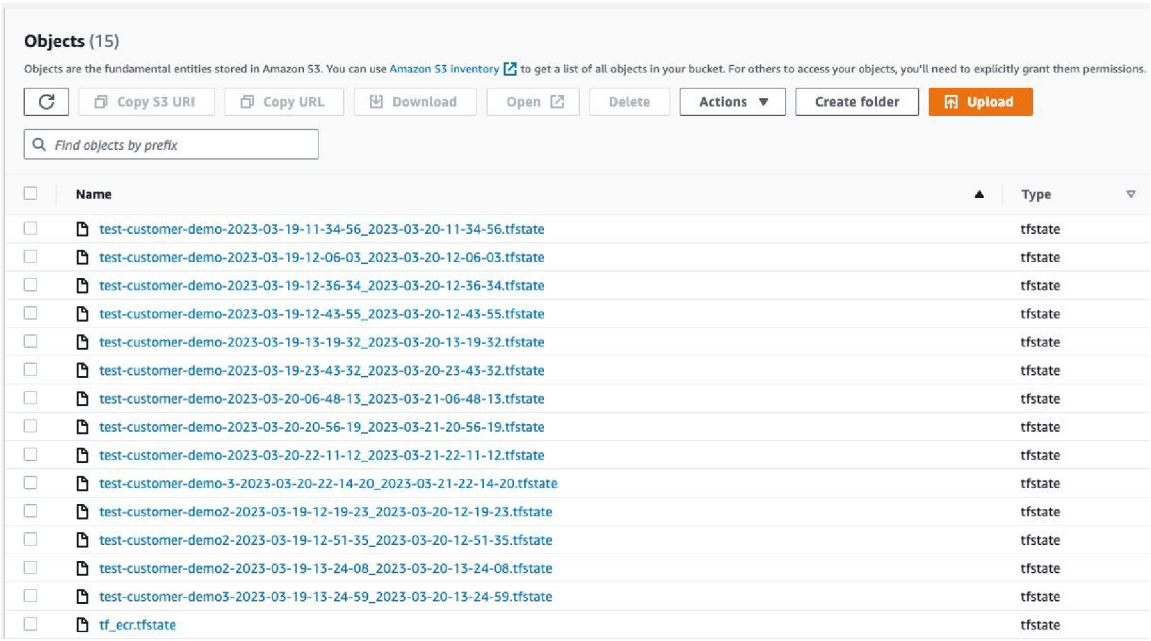


Figure 18: State files in S3

plan output is shown in Figure 19.

Regarding the AWS infrastructure, there are several elementary resources forming the underlying layer. These resources are as follows:

- S3 bucket serves as a Terraform state file storage (already created).
- ECR (Elastic Container Registry) for storing Helm charts and Docker images.
- VPC and networking (virtual private cloud definition and main network definition).
- AWS IGW (Internet Gateway) for accessing resources from the internet.
- Subnets (definition of subnets used for EKS and application networking inside and outside the internet).
- NAT Gateway (transfers network traffic between public and private networks).
- Route tables later assigned to subnets.

```
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create
 <= read (data resources)

Terraform will perform the following actions:

  # data.aws_iam_policy_document.eks_cluster_autoscaler_assume_role_policy will be read during apply
  # (config refers to values not yet known)
 <= data "aws_iam_policy_document" "eks_cluster_autoscaler_assume_role_policy" {
      + id   = (known after apply)
      + json = (known after apply)

      + statement {
          + actions = [
              + "sts:AssumeRoleWithWebIdentity",
            ]
          + effect  = "Allow"

          + condition {
              + test     = "StringEquals"
              + values   = [
                  + "system:serviceaccount:kube-system:cluster-autoscaler",
                ]
              + variable = (known after apply)
            }

          + principals {
              + identifiers = [
                  + (known after apply),
                ]
              + type        = "Federated"
            }
        }
    }

```

Figure 19: Terraform plan output

- EKS (Elastic Kubernetes Service) runs the application stack
- Nodes and node groups form the underlying layer that runs Kubernetes workloads (application stack).
- OIDC definition (OpenID Connect Provider).
- IAM for autoscaler (a role that allows the autoscaler to have permissions to control node groups and scale them).

As shown in Figure 20, Terraform deploys several layers that were mentioned in List 4.1. The networking is divided into two main groups: private and public. The public networking is connected to the internet gateway, while the private networking is connected to the NAT gateway. Each networking unit has one routing table and two subnets. The networking service dynamically provides IP addresses. The private networking provides networking to the node group layer, which comprises EC2 instance virtual machines that

provide computing power to the EKS cluster workload - the workload runs on these virtual machines. The private networking also provides networking for the app workload that runs on EKS. If there is a need to expose one of the services to the internet, it is routed through the internet gateway and an IP address is dedicated from one of the public subnets. The app stack images are pulled from ECR. Terraform deploys everything via GitHub Action and the current state of the infrastructure is saved in the S3 bucket in the same VPC on the cloud.

4.2 Github action automations

GitHub Actions are essentially sets of rules and definitions written in a YAML file that are executed by the GitHub Actions engine. These actions consist of several sections, including the following:

- Workflow dispatch: This section serves as the entry point for inputs configured by the user of the GitHub Action. [53]
- Jobs: This section is used to define groups of steps that are executed and jobs can run in parallel.
- Steps: This section contains the steps that are executed sequentially and they are contained within job groups.
- Plugins: These are used to execute specific steps, such as the EndBug/add-and-commit@v9 plugin.

The repository contains all the files required for creating the demo environment, including Terraform infrastructure scripts, Helm charts and an important file with CIDR blocks. This file lists all available IP ranges within a single VPC that is defined by the main IP range after it is created.

4.2.1 Demo environment creation automation

The automation process of the deployment is illustrated in diagram 21. Once the workflow dispatch is filled with user input, the action is triggered. It checks out the current git commit from the main branch and begins initializing carbon print. Carbon print is a set of files that defines the demo environment:

- backend.tf is a Terraform backend file that includes initial infrastructure configuration, as mentioned in chapte 4.1
- The files from and to contain the timestamp generated by the Python script based on user input of the lifetime of the demo environment. After the time specified in the to field is reached, the environment is automatically destroyed by a GitHub action cron job.
- Status is a file that contains the current status of the environment creation and can have values: initialized, progress, created.
- ccidr_blocks.csv contains IP ranges of all networks used for demo environment creation. These IP ranges are needed for the apply and destroy commands.

- tf_apply.sh and tf_destroy.sh contain the apply and destroy commands with all configuration parameters that are passed into Terraform variables.

Once the carbon print is initialized, it is committed to the main branch and the workflow proceeds to validate and initialize the Terraform backend with the cloud infrastructure. The carbon print is then updated to the progress status and the creation of the demo environment can begin. In the next step, the CIDR blocks are reserved from the main CIDR block file in the repository and a new file cidr_blocks.csv is created in the carbon print. Additionally, the tf_apply.sh and tf_destroy.sh scripts are generated. Another commit is made to the main branch to save the updated carbon print with all the changes. In Step 7, the tf_apply.sh command is used to create the infrastructure, as shown in Figure 20. In Step 8, Helm is used to install the components described in Chapter 4.3. Finally, the carbon print is updated to its final state and the status is changed to created. The customer now has access to the private demo environment.

4.3 Helm deployment and app stack deployment

This chapter describes the deployment of the app stack on the infrastructure layer created in sub-chapter 4.1. The entire deployment is done using multiple Helm charts.

The following applications are deployed on the EKS cluster:

- Node autoscaler: This is a scaler that scales the underlying EKS infrastructure - nodes in the node groups - and defines how many nodes should be started to run the application stack. This autoscaler is deployed as a plugin and is part of the Kubernetes control plane.
- Private and public load balancers.
- AWS CloudWatch collectors (for collecting metrics used by KEDA).
- KEDA POD autoscaler.
- The application itself.

4.3.1 Node autoscaler

The node autoscaler is designed to automatically scale the underlying layer of nodes in response to changes in the number of pods. When the number of pods increases, the control plane triggers the autoscaler to scale the nodes up. Conversely, when the number of pods decreases, the control plane triggers the autoscaler to scale the nodes down. The autoscaler consists of two main components: IAM roles and policies and the autoscaler deployment itself. The IAM roles and policies were already deployed using Terraform and are responsible for granting permissions to manage the node groups.

IAM policies consist of three main components: aws_iam_policy_document, aws_iam_role and aws_iam_policy. The aws_iam_policy_document entity specifies who has access, which in this case is the aws_iam_openid_connect_provider.eks.url, an OpenID provider integrated into the EKS cluster. The aws_iam_role entity defines the role itself, while the aws_iam_policy entity implements specific policies. The definition of these three entities is illustrated in Figure 22. fterward, the policy is attached to the role as shown in Figure 23.

The autoscaler deployment in the k8s cluster consists of the following entities. The ServiceAccount is responsible for binding to the IAM role created previously by Terraform.

The definition of this is shown in 24 where the binding to the AWS IAM role is visible via the annotation.

ClusterRole defines permissions inside the k8s cluster that has got entity assigned to the role as shown in Figure 25.

Role defines permissions insisde the k8s cluster that has got entity assigned to the role, but differs from the ClusterRole by the scope. ClusterRole is able to define the permissions to the cluster scoped entities such as nodes (important for the autoscaler), non-resource endpoints, namespaced resources (like deployments/pods) across all namespaces. Role has got scope only over the namespace. Definition of the Role could be seen in the Figure 26. ClusterRoleBinding and RoleBinding are kind of attachment between ServiceAccount, ClusterRole and Role. Definition of those could be seen in the Figure 27. Once the role is attached via binding to the ServiceAccount, the ServiceAccount has got permissions defined by the roles.

Deployment is the autoscaler application itself, that has got assigned ServiceAccount with ClusterRole/Role defined permissions and is deployed to the kube-system namespace, which is degault namespace of k8s where the control plane related applications are deployed. Definition of the autoscaler deployment could be seen in the Figure 28. Whole mechanism of IAM policies, permissions and ServiceAccounts is also explained in the diagram 29.

Atoscaler could be configured for particular purpose as is shown in the Figure 30. There are few important parameters that should be meant in context with the current use case:

- node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/application_cluster - This specifies particular nodes, that are being scaled. If nodes do not have these tags, they are not scaled.
- scale-down-utilization-threshold - If one the node resources (memory utilization, cpu utilization) increases above 50%, autoscaler triggers up. If nodes are utilized under 50% autoscaler triggers down.
- scale-down-delay-after-add - if nodes are utilized under 50% for 15 seconds, autoscaler triggers scale down.

4.3.2 Private and public load balancers

Private and public load balancers are essentially Kubernetes services based on Nginx, which include specific annotations defining the type of AWS LB, whether it's internal or external. In the Figure 31 is example of full private load balancer. TThere are two important annotations: service.beta.kubernetes.io/aws-load-balancer-type, which specifies the type of AWS network load balancer as nlb and service.beta.kubernetes.io/aws-load-balancer-internal, which specifies whether the load balancer is internal or external. The second annotation is not necessary for public load balancers, as AWS LB is public by default unless another configuration via annotations is provided.

4.3.3 AWS CloudWatch collectors

For the purpose of this master thesis, there is used Fluent Bit. It is a lightweight and efficient data collector and forwarder designed for cloud-native environments and modern infrastructures. It is an open-source project developed by the Fluentd community and written in C language. Fluent Bit is deployed as a Daemon Set and runs on each node,

collecting logs from containers that run on every node and then sends the data to AWS Cloud Watch, which saves the data to an S3 Bucket. The logs can be viewed by executing Cloud Watch queries from the S3 Bucket and they are displayed as tables and graphs. A detailed log collection architecture is illustrated in the diagram 32.

4.3.4 KEDA deployment and data flow architecture

For querying metrics from Cloud Watch by KEDA Scale Set, access logs should be transmitted into custom metric, that is later queried by Scale Set itself. This is done by query filter and creation custom metric into new Cloud Watch namespace. Only metric could be queried in Cloud Watch API so there is no other possibility then creating it out of log query. In the diagram 33 could be seen whole architecture. Logs are stored in log groups within Log insights. A custom metric can query logs in their original state and parse them using a simple regex. This custom metric periodically queries logs to make them available for external queries through the Cloud Watch API. KEDA ScaledObject is a Kubernetes custom resource that is bound to a specific deployment using an annotation (in this case, Nginx). It periodically queries the custom metric from the Cloud Watch API and compares it to configured thresholds. When the metric reaches the threshold, it triggers scaling up and sends a request to the KEDA engine, which runs in the same Kubernetes cluster. The request includes information about the number of instances to which the deployment should be scaled up and the deployment identifier (in this case, the name of the deployment is Nginx). The Kubernetes default pod scaler scales up the deployment, which remains in this state until the queried custom metric decreases below the threshold value. Once the cooldown period is triggered and there are no further increases in the custom metric value, the deployment is scaled back down to its initial state. The custom metric can be based on a large amount of diverse information contained in logs, but this always depends on the specific use case. Potential log query filters could include:

- Error rate (5xx)
- Latency
- Response time
- Number of timeouts

The configuration of the KEDA ScaledObject could be seen in the Figure 34. It contains few important parameters:

- pollingInterval: 30 - a length of period in which the scaled object executes metric API call
- cooldownPeriod: 0 - a period after which all pods are scaled down to initial state once the metric value decreases under the certain threshold
- minReplicaCount: 1 - minimum number of instances of scaled application
- maxReplicaCount: 50 - maximum number of instances of scaled application
- namespace: CustomLogQueries - a namespace, where is located the metric
- expression: SELECT SUM(nginx_all_http_requests_1) FROM CustomLogQueries - the query targeting our metric

- metricName: nginx_all_http_requests_1 - metric name
- targetMetricValue: 1300 - a threshold upon the scaling up triggers
- minMetricValue: 0 - a value provided if pulling from the metric ends up with no value
- awsRegion: eu-west-1 - a region of metric location
- awsEndpoint: - could be targeted specific endpoint (/ by default)
- metricCollectionTime: 600 - a size of the interval from now to past in seconds
- metricStat: Average - an agregation function applied on metric query
- metricStatPeriod: 60 - equivalent to group by function, in this case 1 second
- metricUnit: Count - the units for the custom metric can vary, such as bytes per second, bits and so on.
- metricEndTimeOffset: 0 - the offset of collection time (now - offset)

The metric threshold for the ScaledObject is configured to be 1300 requests per minute and when this threshold is reached, the number of nginx replicas scales up. The scaling policy is defined by the function numberOfPods = numberOfRequests / metricThreshold for numbers greater than 1. If the number is less than 1, the count of nginx replicas is 1. If there are no requests, the number of nginx replicas is determined by the minReplicaCount parameter.

4.3.5 Load test of KEDA integration

The load testing is managed by a user-friendly tool called Locust [54], which allows easy creation and execution of load tests using Python. The output of Locust is a report consisting of several graphs that are easy to read and export to various formats for use with other data analytic tools. The design specifies a minimum of 27.5 requests per second, equivalent to 1650 requests per minute, with a minimum aggregation value of 60 seconds in the current metric pull. Load test was running in 10 rounds with increase of requests per every round. Results could be seen in the table 8. The prove of scale up based on the number of the request per minute could be seen in the graph 35, it correlates and ScaledObject is configured well.

Table 8: Results of load test

Round	Threads	Spawns/s	Requests/s (avg)	Requests/minute (avg)	nginx replicas
1	1	1	28	1678	2
2	2	1	54	3284	3
3	3	1	86	5165	5
4	4	2	111	6638	5
5	5	5	144	8445	7
6	7	5	190	11276	9
7	9	7	250	15040	12
8	12	10	290	17362	14
9	14	14	320	20255	16
10	20	30	479	28737	24

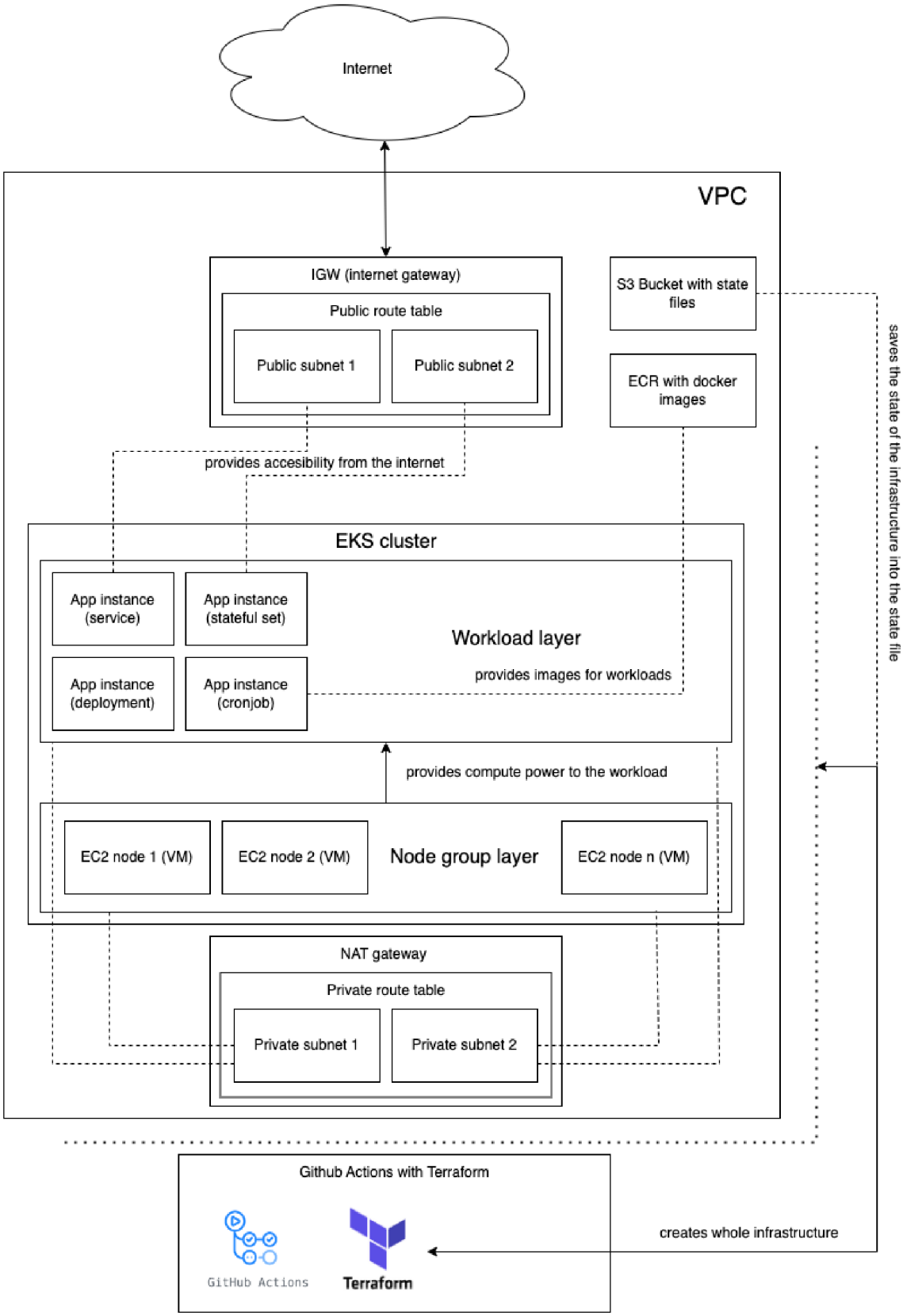


Figure 20: Infrastructure architecture diagram

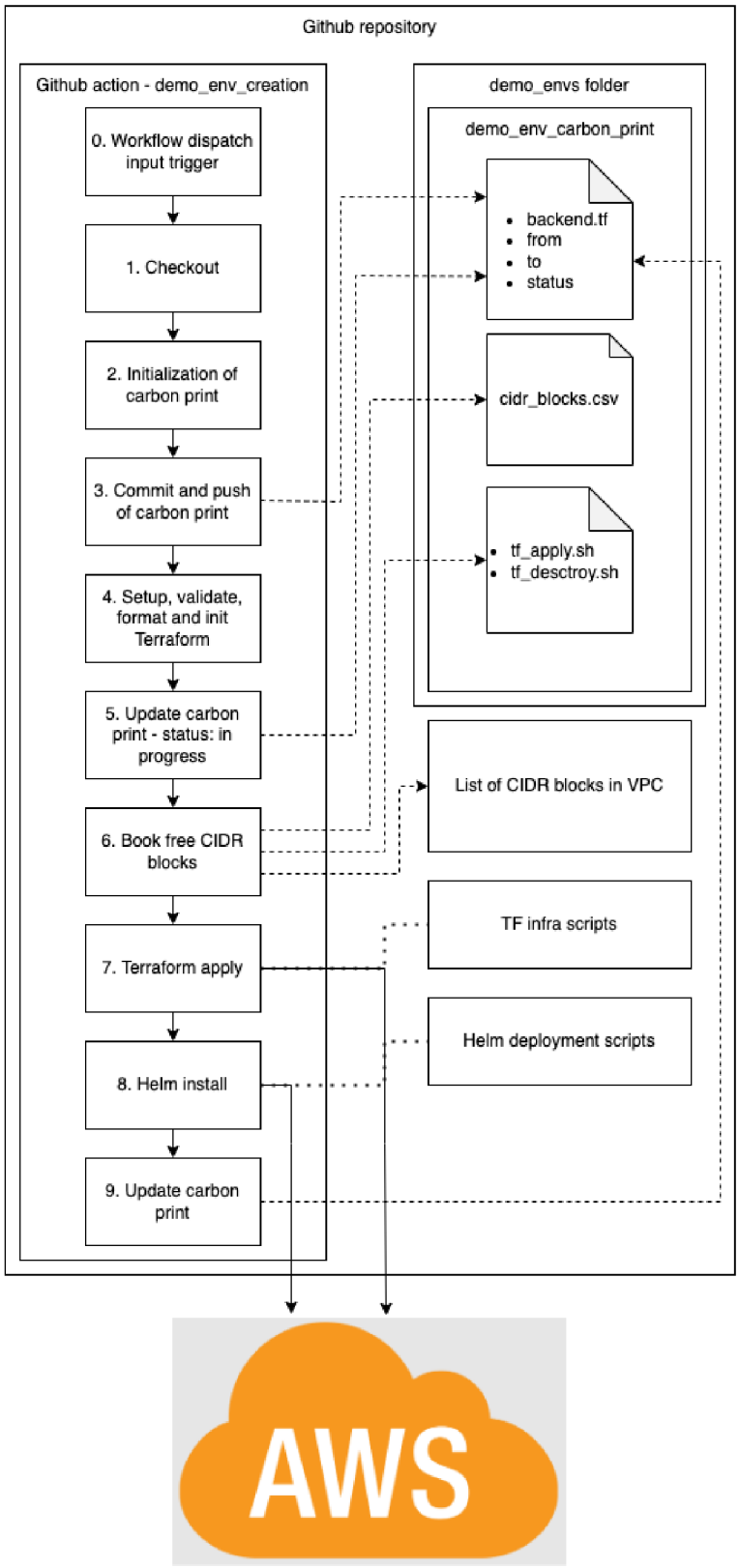
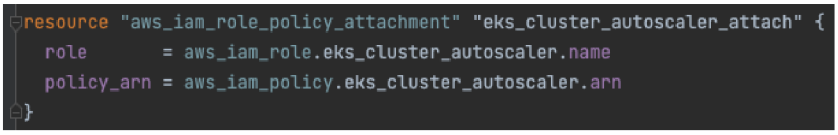


Figure 21: Github action infrastructure creation architecture

```
data "aws_iam_policy_document" "eks_cluster_autoscaler_assume_role_policy" {
  statement {
    actions = ["sts:AssumeRoleWithWebIdentity"]
    effect  = "Allow"

    condition {
      test     = "StringEquals"
      variable = "${replace(aws_iam_openid_connect_provider.eks.url, "https://", "")}:sub"
      values   = ["system:serviceaccount:kube-system:cluster-autoscaler"]
    }

    principals {
      identifiers = [aws_iam_openid_connect_provider.eks.arn]
      type        = "Federated"
    }
  }
}

resource "aws_iam_role" "eks_cluster_autoscaler" {
  assume_role_policy = data.aws_iam_policy_document.eks_cluster_autoscaler_assume_role_policy.json
  name               = "eks-cluster-autoscaler-${var.environment_name}"
}

resource "aws_iam_policy" "eks_cluster_autoscaler" {
  name = "eks-cluster-autoscaler-${var.environment_name}"

  policy = jsonencode({
    Statement = [{
      Action = [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "autoscaling:SetDesiredCapacity",
        "autoscaling:TerminateInstanceInAutoScalingGroup",
        "ec2:DescribeLaunchTemplateVersions"
      ]
      Effect   = "Allow"
      Resource = "*"
    }]
    Version = "2012-10-17"
  })
}
```

Figure 22: IAM source code



```
resource "aws_iam_role_policy_attachment" "eks_cluster_autoscaler_attach" {
  role       = aws_iam_role.eks_cluster_autoscaler.name
  policy_arn = aws_iam_policy.eks_cluster_autoscaler.arn
}
```

Figure 23: Policy and role attachment

Figure 24: Service account

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-autoscaler
rules:
  - apiGroups: [""]
    resources: ["events", "endpoints"]
    verbs: ["create", "patch"]
```

Figure 25: Cluster role

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: cluster-autoscaler
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["create","list","watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    resourceNames: ["cluster-autoscaler-status", "cluster-autoscaler-priority-expander"]
    verbs: ["delete", "get", "update", "watch"]
```

Figure 26: Role

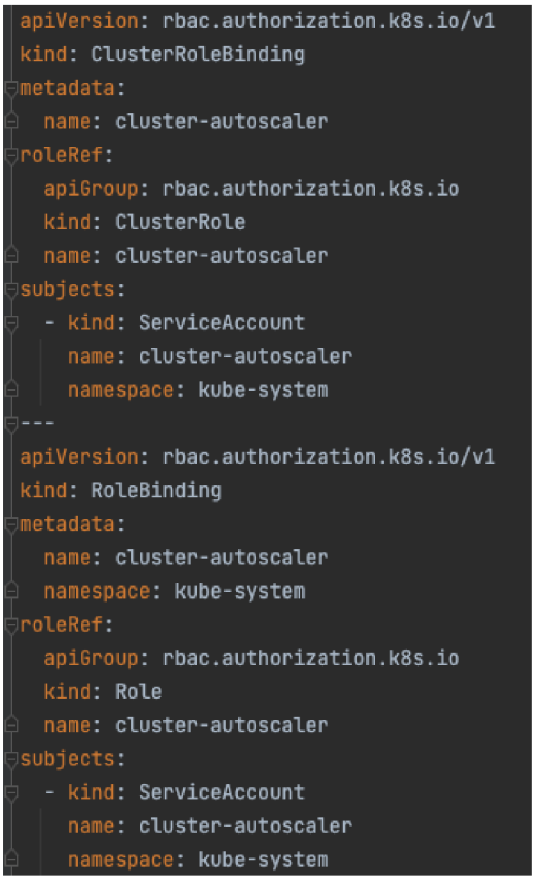


Figure 27: ClusterRoleBinding and RoleBinding

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cluster-autoscaler
  namespace: kube-system
  labels:
    app: cluster-autoscaler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cluster-autoscaler
  template:
    metadata:
      labels:
        app: cluster-autoscaler
    spec:
      serviceAccountName: cluster-autoscaler
      containers:
        - image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.21.0
          name: cluster-autoscaler
          resources:
            limits:
              cpu: 100m
              memory: 600Mi
            requests:
              cpu: 100m
              memory: 600Mi
```

Figure 28: Node autoscaler deployment

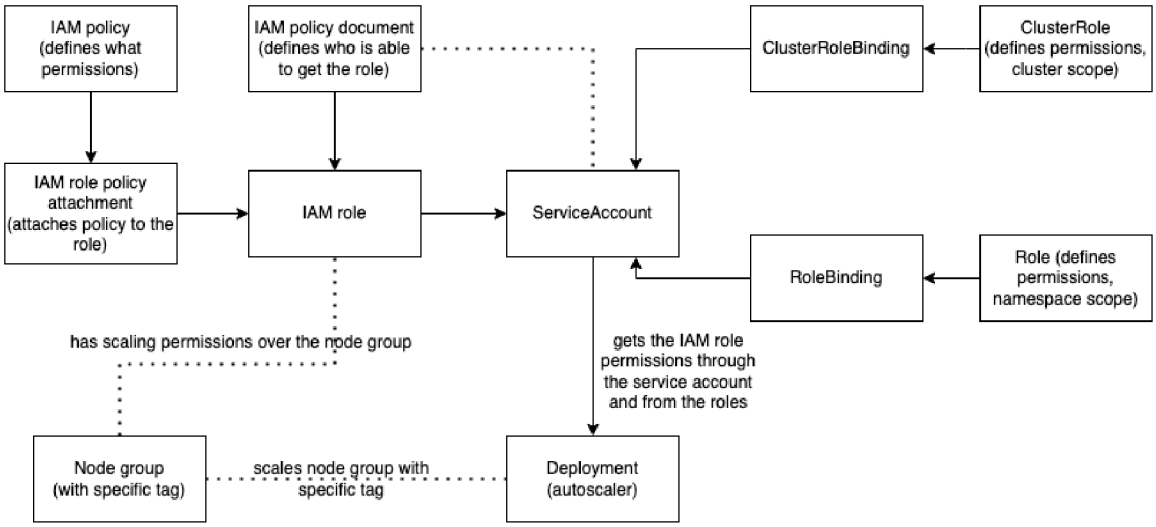


Figure 29: IAM and role schema

```
./cluster-autoscaler
--v=4
--stderrthreshold=info
--cloud-provider=aws
--skip-nodes-with-local-storage=false
--expander=least-waste
--node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/application_cluster
--balance-similar-node-groups
--skip-nodes-with-system-pods=false
--scale-down-utilization-threshold=0.5
--scale-down-non-empty-candidates-count=30
--scale-down-delay-after-add=15s
--scale-down-delay-after-delete=0
--scale-down-delay-after-failure=3m
--scale-down-unneeded-time=15s
```

Figure 30: Autoscaler configuration

```yaml
apiVersion: v1
kind: Service
metadata:
  name: private-lb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: nlb
    service.beta.kubernetes.io/aws-load-balancer-internal: 0.0.0.0/0
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: web
```

Figure 31: Private loadbalancer

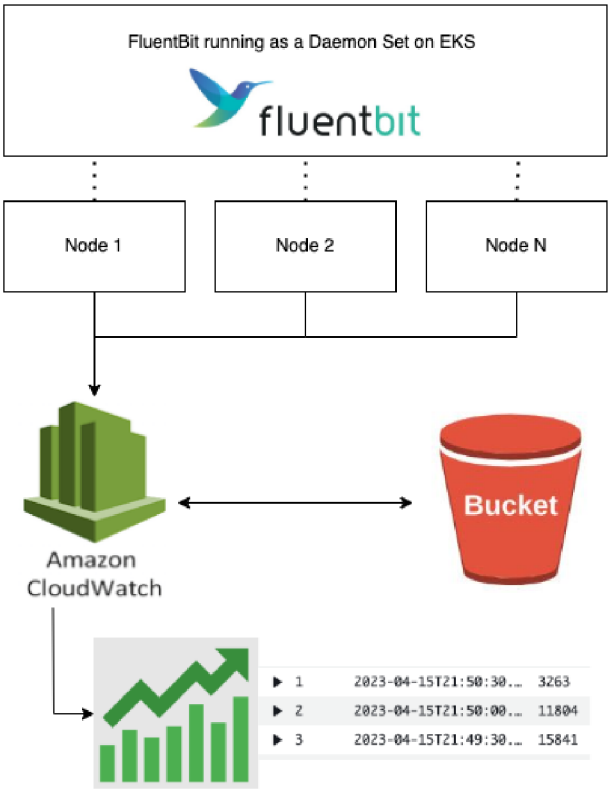


Figure 32: FluentBit log collection schema

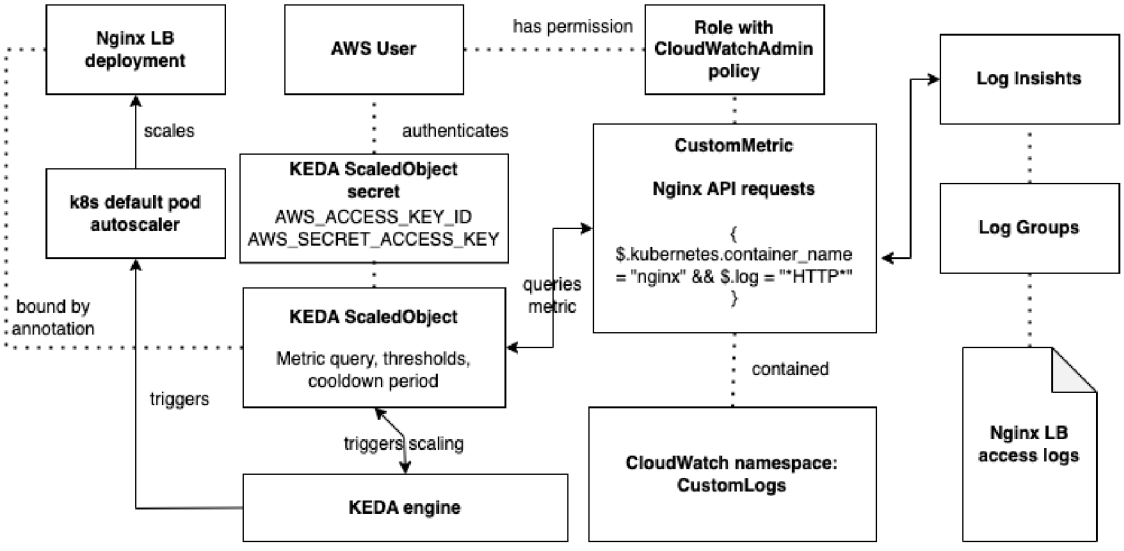
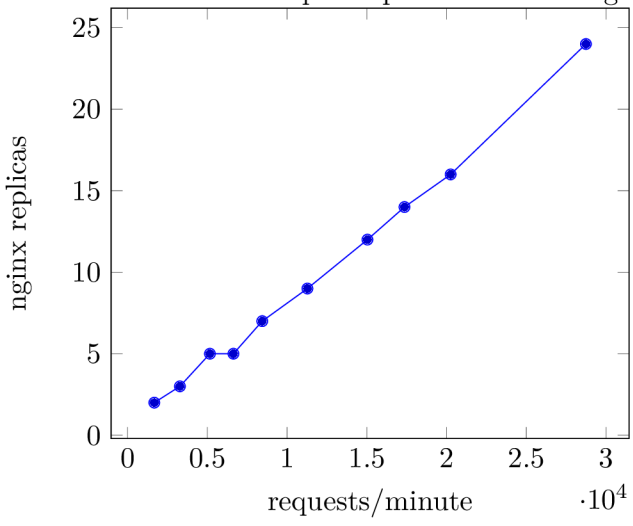


Figure 33: Custom metric architecture and log quering by ScaleObject

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: aws-cloudwatch-queue-scaledobject-nginx
  namespace: default
spec:
  scaleTargetRef:
    name: nginx
  pollingInterval: 30
  cooldownPeriod: 0
  minReplicaCount: 1
  maxReplicaCount: 50
  triggers:
    - type: aws-cloudwatch
      metadata:
        namespace: CustomLogQueries
        expression: SELECT SUM(nginx_all_http_requests_1) FROM CustomLogQueries
        metricName: nginx_all_http_requests_1
        targetMetricValue: "1300"
        minMetricValue: "0"
        awsRegion: "eu-west-1"
        awsEndpoint: ""
        metricCollectionTime: "300"
        metricStat: "Average"
        metricStatPeriod: "60"
        metricUnit: "Count"
        metricEndTimeOffset: "0"
      authenticationRef:
        name: keda-trigger-auth-aws-credentials
```

Figure 34: ScaledObject deployment and configuration

Figure 35: Correlation between requests per minute and nginx replicas



5 Conclusion of results

The output of the practical part of the master's thesis is the implementation of the infrastructure and CI/CD pipeline designed in the third chapter. There was discussed the implementation of the entire infrastructure created by Terraform, based on EKS, networking, load balancing, log ingestion by FluentBit, node autoscaling and pod autoscaling by KEDA, with custom metrics stored in an S3 bucket managed by AWS Cloud Watch. The infrastructure and deployment could be automatically created from GitHub Actions and provides on-demand deployment in several minutes. The last part of the chapter proved and tested the concept of pod autoscaling through a sequential load test of the Nginx load balancer and the results can be seen in Graph 35.

6 Conclusion

In the second chapter, the cloud market was researched and initial observations were made. The results from Synergy Research were compared with other sources and were found to highly correlate (table 1). The chapter compared multiple cloud providers and their shares in the growing cloud market, as well as their services. Security aspects of the cloud, such as encryption and shared responsibility models, were also defined and compared. Based on the research, the AWS cloud provider was chosen.

The third chapter discussed the infrastructure design and modern principles of cloud infrastructure development, such as infrastructure as code, Helm and its charts and virtual environments like Kubernetes. The chapter concluded with a choice of suitable technologies to run the application stack. The second part of the chapter focused on scaling the system and its components to meet the demand of real-time processing and requests. It compared suitable DaaS options and researched modern CI/CD principles, complemented by automated vulnerability detection and GitOps principles like ArgoCD. Release management was also mentioned, which partly defined the diversification of production-wise environments and demo or development environments. The output of this chapter was the architecture of the infrastructure and CI/CD pipeline, which can be seen in Figure 16.

The fourth chapter implemented a suitable architecture of the infrastructure layer for running enterprise applications based on distributed system design and microservices. Later on, it was implemented by Terraform, Helm, Github Actions, and KEDA. The result is an on-demand environment created in less than 30 minutes that runs on Kubernetes in the cloud and is fully automatically scalable.

This master thesis has proven several principles of modern cloud development and virtualization, that were tested and can be highly recommended to every company, that has got a high demand of moving into cloud and provide there a distributed enterprise system as SaaS.

References

[1] RENO, N. 2018 Review Shows $250 billion Cloud Market Ecosystem Growing at 32% Annually. *Synergy*, Jan. 2019. Available from: https://www.srgresearch.com/articles/2018-review-shows-250-billion-cloud-market-ecosystem-growing-32-annually

[2] RENO, N. Amazon, Microsoft, Google and Alibaba Strengthen their Grip on the Public Cloud Market. *Synergy*, Oct. 2019. Available from: https://www.srgresearch.com/articles/amazon-microsoft-google-and-alibaba-strengthen-their-grip-public-cloud-market

[3] RENO, N. Q3 Cloud Spending Up Over $11 Billion from 2021 Despite Major Headwinds; Google Increases its Market Share. *Synergy*, Oct. 2022. Available from: https://www.srgresearch.com/articles/q3-cloud-spending-up-over-11-billion-from-2021-despite-major-headwinds-google-increases-its-market-share

[4] team, A. Shared Responsibility Model. *aws.amazon.com*, Nov. 2022. Available from: https://aws.amazon.com/compliance/shared-responsibility-model/

[5] team, G. Shared responsibilities and shared fate on Google Cloud. *cloud.google.com*, July 2022. Available from: https://cloud.google.com/architecture/framework/security/shared-responsibility-shared-fate

[6] Terry Lanfear, D. B., Ann Marie Hitchcock. Shared responsibility in the cloud. *learn.microsoft.com*, Aug. 2022. Available from: https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility

[7] McGuire, C.; team. About Point-to-Site VPN. *learn.microsoft.com*, Sept. 2022. Available from: https://learn.microsoft.com/en-us/azure/vpn-gateway/point-to-site-about

[8] Team, A. Azure security center. *azure.microsoft.com*, Nov. 2019. Available from: https://azure.microsoft.com/en-gb/blog/new-azure-security-center-and-azure-platform-security-capabilities-2/

[9] team, A. Amazon inspector. *aws.amazon.com*, Jan. 2023. Available from: https://aws.amazon.com/inspector/

[10] Team, K. Kubernetes Event-driven Autoscaling. *keda.sh*, Feb. 2023. Available from: https://keda.sh/docs/2.0/concepts/

[11] Team, A. ArgoCD. *argo-cd.readthedocs.io*, Feb. 2023. Available from: https://argo-cd.readthedocs.io/en/stable/

[12] Team, G. Google Platform Services Comparison. *cloud.google.com*, Jan. 2023. Available from: https://cloud.google.com/free/docs/aws-azure-gcp-service-comparison

[13] Ekuan, M.; team. Azure Platform Services Comparison. *learn.microsoft.com*, Jan. 2023. Available from: https://learn.microsoft.com/en-us/azure/architecture/aws-professional/services

[14] Hilton, A. SRE fundamentals SLI vs SLO vs SLA. *Google*, May 2022. Available from: https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-sli-vs-slo-vs-sla

[15] RENO, N. Synergy Cloud Market Shares. *Synergy*, Feb. 2020. Available from: https://www.srgresearch.com/articles/incremental-growth-cloud-spending-hits-new-high-while-amazon-and-microsoft-maintain-clear-lead-reno-nv-february-4-2020

[16] Solutions, S. Sam Solutions Cloud Market Shares. *Sam Solutions*, Jan. 2021. Available from: https://sam-solutions.us/aws-vs-azure-v-s-google-cloud-which-is-better/

[17] Murray, C. Alto Palo Cloud Market Shares. *Alto Palo*, Apr. 2021. Available from: https://alto-palo.com/blogs/google-cloud-vs-aws-vs-azure-choose-the-right-cloud-platfor

[18] analytics team, C. Canalys Cloud Market Shares. *Canalys*, Apr. 2021. Available from: https://www.canalys.com/newsroom/global-cloud-market-Q121

[19] seekers, H. Hosting Seekers Cloud Market Shares. *Hosting Seekers*, Aug. 2021. Available from: https://www.hostingseekers.com/blog/aws-vs-azure-vs-google-cloud-platform

[20] CISIN. CISIN Cloud Market Shares. *CISIN*, Jan. 2022. Available from: https://www.cisin.com/coffee-break/technology/aws-vs-azure-vs-google-cloud-market-share-2021.html

[21] Panettieri, J. Channele2e Cloud Market Shares. *Channele2e*, Apr. 2022. Available from: https://www.channele2e.com/news/cloud-market-share-amazon-aws-microsoft-azure-google/

[22] team, G. Serverless. *cloud.google.com*, Nov. 2022. Available from: https://cloud.google.com/serverless/

[23] team, M. Download and install the Microsoft Authenticator app. *support.microsoft.com*, Dec. 2022. Available from: https://support.microsoft.com/en-us/account-billing/download-and-install-the-microsoft-authenticator-app-351498fc-850a-45da-b7b6-27e523b8702a

[24] team, G. Google Authenticator. *play.google.com*, Dec. 2022. Available from: https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&gl=US&pli=1

[25] team, A. AWS Single Sign-On (AWS SSO) is now AWS IAM Identity Center. *aws.amazon.com*, July 2022. Available from: https://aws.amazon.com/about-aws/whats-new/2022/07/aws-single-sign-on-aws-sso-now-aws-iam-identity-center/

[26] team, O. Okta universal directory. *www.okta.com*, Dec. 2022. Available from: https://www.okta.com/products/universal-directory/

[27] team, E. AWS, Azure and GCP: The Ultimate IAM Comparison. *ermetic.com*, July 2022. Available from: https://ermetic.com/blog/cloud/aws-azure-and-gcp-the-ultimate-iam-comparison/

[28] Baldwin, M.; team. Azure Data Encryption at rest. *learn.microsoft.com*, Nov. 2022. Available from: https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest

[29] Baldwin, M.; team. Server-side encryption of Azure Disk Storage. *learn.microsoft.com*, Aug. 2022. Available from: https://learn.microsoft.com/en-us/azure/virtual-machines/disk-encryption

[30] Gluck, D.; team. Encryption for data-in-transit. *learn.microsoft.com*, Sept. 2022. Available from: https://learn.microsoft.com/en-us/compliance/assurance/assurance-encryption-in-transit

[31] Team, A. Azure security center. *azure.microsoft.com*, Jan. 2023. Available from: https://azure.microsoft.com/en-us/services/security-center/

[32] team, G. Google Security and Trust Center. *cloud.google.com*, Jan. 2023. Available from: https://cloud.google.com/security/

[33] org team, I. ISO Standard 27001. *www.iso.org*, Jan. 2023. Available from: https://www.iso.org/standard/45170.html

[34] team, T. Terraform. *www.terraform.io*, Feb. 2023. Available from: https://www.terraform.io/

[35] team, A. AWS CloudFormation. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/cloudformation/index.html

[36] team, M. Azure Resource Manager. *azure.microsoft.com*, Feb. 2023. Available from: https://azure.microsoft.com/en-us/get-started/azure-portal/resource-manager

[37] Team, G. Google Cloud Deployment Manager. *cloud.google.com*, Feb. 2023. Available from: https://cloud.google.com/deployment-manager/docs

[38] Team, H. Helm Package Manager. *helm.sh*, Feb. 2023. Available from: https://helm.sh

[39] Christopher Parker, M. Helm Umbrella Charts. *itnext.io*, June 2020. Available from: https://itnext.io/helm-3-umbrella-charts-standalone-chart-image-tags-an-alternative-approach-78a218d74e2d

[40] team, K. Kubernetes Engine. *kubernetes.io*, Feb. 2023. Available from: https://kubernetes.io/

[41] Team, A. AWS Elastic Kubernetes Service. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html

[42] Team, A. AWS Elastic Container Service. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html

[43] Team, A. AWS Fargate Instances. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html

[44] Team, K. Kubernetes build-in autoscaler. *kubernetes.io*, Feb. 2023. Available from: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[45] Team, A. AWS Database Solutions. *docs.aws.amazon.com*, Feb. 2023. Available from: https://aws.amazon.com/products/databases/

[46] Team, A. AWS RDS for Postgres. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html

[47] Team, A. AWS Aurora for Postgres. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Aurora.AuroraPostgreSQL.html

[48] Team, A. AWS Neptune. *docs.aws.amazon.com*, Feb. 2023. Available from: https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

[49] Team, R. CI/CD principals. *redhat.com*, May 2022. Available from: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[50] Team, G. Github actions. *github.com*, Feb. 2023. Available from: https://docs.github.com/en/actions

[51] Team, G. Terratest. *gruntwork.io*, Feb. 2023. Available from: https://terratest.gruntwork.io/

[52] Team, S. Sonar cloud. *sonarsource.com*, Feb. 2023. Available from: https://www.sonarsource.com/products/sonarcloud/

[53] Team, G. Workflow dispatch. *docs.github.com*, Feb. 2023. Available from: https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows

[54] Team, L. Locust. *locust.io*, Feb. 2023. Available from: https://locust.io/

Seznam zkratek

AC Azure cloud

AD Azure active directory

API Application Programming Interface

ARM Azure Resource Manager

AWS Amazon web services

CD Continuous Deployment

CI Continuous Integration

CI/CD Continuous Integration/Continuous Deployment

CLI Command Line Interface

DaaS database as a service

DDOS distributed denial of service

DevOps Developers/Operations

ECS Amazon Elastic Container Service

EKS Amazon Elastic Kubernetes Service

GC Google cloud

GCP Google Cloud Platform

HA High availability

HCL HashiCorp Configuration Language

HPA Horizontal Pod Autoscaler

IaaC Infrastructure as a Code

IaaS infrastructure as a service

IAM identity and access management

KEDA Kubernetes Event-driven Autoscaling

MFA Multifactor authentication

on-prem on premise

OS Operating system

P2S point to site

PaaS platform as a service

POD The smallest deployable unit that represents a single instance of a running process in k8s cluster

RBAC role based access control

S2S site to site

SaaS software as a service

SDK Software Development Kit

SLA service level agreement

SLO service level objective

SSO Single sign on

TLS Transport Layer Security

UAT User Acceptance Testing

VM virtual machine

VPC virtual private cloud

VPN virtual private network

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. Jan Kohout**

Studium: I2000982

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Virtualizace a virtualizační clustery na Linuxových systémech**

Název diplomové práce AJ: Virtualization and virtualization clusters on Linux systems

Cíl, metody, literatura, předpoklady:

Cílem diplomové práce je prostudování, pochopení a návrh virtualizačního clusteru jeho automatická konfigurace a dimenzace pro určitý typ aplikací. Řešení má být založeno na open source software.

1, Úvod do problematiky virtualizace

2, Rešerše jednotlivých řešení na trhu

3, Návrh provedení a automatizace

4, Návrh teoretické dimenzace

5, Implementace a provedení základních výkonových testů

6, Shrnutí

7, Závěr

1, Virtualization, A Beginner's Guide 978-0071614016, 978-0071614016
2, Understanding the Linux Kernel Daniel P. Bovet,Marco Cesati 0596005652 (ISBN13: 9780596005658)
3, Hardware and Software Support for Virtualization 1627056882 (ISBN13: 9781627056885)
4, Inside the Machine 1593271042 (ISBN13: 9781593271046)
5, Docker Deep Dive (Kindle Edition) Nigel Poulton
Kubernetes: Up & Running (Paperback) Kelsey Hightower,Brendan Burns,Joe Beda 1491935677 (ISBN13: 9781491935675)
6, Docker: Up & Running: Shipping Reliable Containers in Production Sean P. Kane, Karl Matthias 1491917571 (ISBN13: 9781491917572)
7, The Book of Xen: A Practical Guide for the System Administrator Chris Takemura, Chris Takemura 1593271867 (ISBN13: 9781593271862)

Zadávající pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Pavel Blažek, Ph.D.

Datum zadání závěrečné práce: 21.1.2020