POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master's Degree Thesis

Enhancing Multi-Cloud Security with Policy-as-Code and a Cloud Native Application Protection Platform

Supervisor:
Prof. Riccardo Sisto

Co-Supervisor:
Dr. Fulvio Valenza

Candidate:
Manuel Enrique Colotti

Company tutors
Reply
Dr. Francesco Borgogni
Dr. Luigi Casciaro
Dr. Ivan Aimale

Academic year 2022-2023

Summary

Over the past 15 years, the Cloud Computing paradigm has steadily gained popularity due to the advantages it offers, such as flexibility, scalability, and reliability. In today's business landscape, there is a notable trend toward the increased adoption of Multi-Cloud strategies by companies to provide services to customers, employees, and other businesses. A 2023 study conducted by Oracle [18] reveals that 98% of surveyed enterprises use at least two cloud infrastructure providers and 31% are using four or more; This widespread adoption underscores the necessity for the implementation of innovative security strategies to safeguard these complex environments. In this context, Cloud Native Application Protection platforms (CNAPPs), represent a novel tool utilized to enforce industry-standard compliance and security across one or more cloud platforms.

This thesis dissertation analyzes the field of Multi-Cloud Security, discussing some solutions proposed in the literature and demonstrating the design and showcasing a realistic enterprise Multi-Cloud infrastructure whose security has been assessed by integrating a CNAPP solution named Sysdig Secure. The work starts off with a description of Cloud Security and Cloud Governance key concepts, to continue with an examination of two different Security-as-a-Service solutions proposed in the literature. Subsequently, an entire chapter is dedicated to the Policy-as-Code paradigm and to how it can be effectively exploited to ensure compliance in DevSecOps pipelines (Section 4.2.3), Public Cloud platforms (Section 5.3.2), and diverse hosts (Section 5.3.3).

Following this, a proof-of-concept Multi-Cloud infrastructure that emulates the resources and features of a company cloud environment is proposed, along with a thorough description of all the integrations implemented from Identity and Access Management, Secrets Management, and Observability points of view. This work also demonstrates how a Cloud Native Application Protection platform has been successfully configured to provide Infrastructure-as-Code security (IaC-Sec), Cloud Security Posture Management (CSPM), and Cloud Workload Protection (CWP) within the previously introduced multi-cloud environment. Finally, the last chapter verifies the proper functioning of all the services offered by the multi-cloud infrastructure and evaluates the quality of the security assessments carried out by Sysdig Secure CNAPP.

Acknowledgements

I would like to thank my supervisor Professor Riccardo Sisto, for giving me the possibility to embark on this thesis journey and for all the pieces of advice he gave me during the development of this work. I would also like to thank my corporate tutors Francesco Borgogni, Luigi Casciaro, and Ivan Aimale for proposing such an interesting topic to work on, and for the support I was given during the whole period at Reply. My recognition also goes to all Liquid Reply colleagues who welcomed me warmly and were always available whenever I had questions or doubts. I also want to express my gratitude to my family, in particular my Mom and Dad who with their sacrifices gave me the possibility to study and after 5 years reach the goal I'm celebrating now. Last but not least I would like to thank my friends for their unwavering support, which inspired and motivated me to consistently give my best despite facing numerous obstacles along the way.

Contents

Chapter 1

Introduction

Over the past 15 years, the Cloud Computing paradigm has steadily gained popularity due to the advantages it offers, such as flexibility, scalability, and reliability. In today's business landscape, there is a notable trend toward the increased adoption of Multi-Cloud strategies by companies to provide services to customers, employees, and other businesses. As a result of this, enterprises often accumulate countless resources in the cloud, making it impractical to secure and verify the compliance of each of them individually. Therefore, it is becoming essential to think about new security strategies and implement novel solutions capable of safeguarding complex cloud environments. Cloud Native Application Protection Platforms (CNAPPs) constitute a new software category. They are engineered to encompass security and compliance capabilities developed specifically for defending and securing cloud-native applications. Among the wide range of features offered by CNAPPs, they excel in several critical areas. These include the capacity to identify security flaws within Infrastructure-as-Code templates (IaC Security), the detection of misconfigurations in cloud platforms and resources (CSPM and CIEM), and the ability to conduct vulnerability assessments and run-time threat detection on workloads. Apart from the previously described security capabilities, many CNAPPs also implement the Policy-as-Code paradigm. The latter allows for the definition of code-written policies that are enforced to certify cloud environments' compliance with regulatory and industry standards such as CIS, NIST, ISO, and many others. All in all, CNAPPs are an asset to the current cloud security scenario that cannot be missed when dealing with the security of multiple heterogeneous cloud platforms, and applications.

1.1 Objective

The thesis commences with a comprehensive examination of cloud security and includes a literature review of some Security-as-a-Service solutions proposed by researchers. Subsequently, an exploration of the Policy-as-Code paradigm highlights how it can be exploited to ensure cloud resources configuration compliance on diverse public Cloud Service Providers (Chapter 3 and Section 5.3.2), and Infrastructure-as-Code templates compliance in the context of DevSecOps CI/CD pipelines (Section 4.2.3). The primary objective of this study is to construct a robust and realistic enterprise Multi-Cloud infrastructure composed of several interconnected services, offering Identity and Access Management and Secrets Management capabilities to applications, as well as Observability throughout the whole environment. Furthermore, the study aims to seamlessly integrate this Multi-Cloud environment with a Cloud Native Application Protection Platform. This integration will empower the CNAPP to proactively detect resources' compliance and security issues during the development, deployment, and runtime stages. The study is focused on confirming the necessity and effectiveness of adopting a CNAPP solution to detect and remediate vulnerabilities that, if left unaddressed, have the potential to undermine the integrity and resilience of entire cloud environments in due course.

1.2 Thesis Structure

The work is structured as follows:

- **Chapter 2** introduces the subject of Cloud Security with a focus on Multi-Cloud environments and Cloud Governance. Furthermore, it proposes a literature review of Security-as-a-Service solutions in Cloud Environments.

- **Chapter 3** presents the Policy-as-Code paradigm and discusses how the latter can be exploited within DevSecOps pipelines in the context of Infrastructure-as-Code security. Sections 3.4, 3.5, and 3.6 on the other hand present from a theoretical point of view three domains of Cloud Security that can be approached with Policy-as-Code as well.

- **Chapter 4** begins by presenting two Infrastructure as Code (IaC) Security tools, followed by a detailed portrayal of their practical usage within a DevSecOps pipeline. Subsequently, it provides a detailed explanation of the resources that comprise the PoC Multi-Cloud infrastructure and their integration with each other.

- **Chapter 5** discusses how Sysdig Secure CNAPP was integrated with the Multi-Cloud infrastructure to assess its level of compliance and security. For each of the security domains Sysdig takes care of, some violations detected by the CNAPP are also illustrated.
- **Chapter 6** tests the Multi-Cloud infrastructure that was realized in Chapter 4 by verifying the correct functioning of all the proposed integrations. Moreover, an assessment of Sysdig's proficiency in identifying vulnerabilities, addressing misconfigurations, and mitigating runtime threats is also undertaken.
- **Chapter 7** presents the conclusions reached by this thesis and a set of improvements aimed at enriching the functionality and enhancing the security level of the existing multi-cloud infrastructure.

Chapter 2

Cloud Security Overview

2.1 Introduction to Cloud Computing

Cloud Computing can be summarized as the delivery of computing resources as a service. In this scenario, resources can be of any type: browser-based web applications, general-purpose storage, and servers employed by privates and companies to support and enhance their computing infrastructure.

Before the spread of cloud computing, a generic company or user would have needed to buy and personally maintain the entirety of the software and hardware resources they desired to use. Nowadays given the exponential availability of cloud resources, businesses, and consumers can access a broad variety of on-demand services; in this context, shifting from on-premise to networked remote distributed hardware and software, allows cloud users to no longer commit the time, money, and knowledge necessary to acquire and manage these computing resources on their own.

The National Institute of Standards and Technology (NIST) of the United States, defines Cloud Computing as: "*a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*"

This institute additionally formulated and specified five essential characteristics of cloud computing:

- **On-demand self-service**: Cloud resources can be accessed and supplied without the need for human intervention, thus allowing consumers to immediately gain access to the cloud services they signed up for. Another capability that businesses can develop and provide to their employees and customers is a mechanism for on-demand access to internal cloud resources according to predetermined logic.

- **Broad network access**: Assuming that they are authorized, users can access cloud services and resources through any device and from any networked location.
- **Resource pooling**: Multiple tenants share the resources of the cloud provider, yet individual clients' data is kept private from other clients.
- **Rapid elasticity**: Differently from on-premise software and hardware, cloud computing resources can be rapidly scaled to suit users' changing needs.
- **Measured service**: Cloud resource usage is metered, allowing companies and other users to only pay for the resources they are using.

2.1.1 Cloud Computing Models

Cloud Computing services can be delivered to users in three main models, each offering different levels of flexibility.

Infrastructure as a Service Infrastructure as a Service or IaaS is the on-demand delivery of computing infrastructure. Storage, networking, operating systems, and all kinds of infrastructural components are outsourced to a third-party cloud provider; in this scenario, individuals or companies will buy the needed computing resources on a pay-as-you-go model.

Platform as a Service Platform as a Service or PaaS is a model often chosen by software developers who want to focus more on development rather than on DevOps and administration. With this approach, it is possible to build and deploy an application in a tested and standardized environment without worrying about the installation, configuration, and maintenance of an entire infrastructure.

Software as a Service Software as a Service is represented by the usage of a complete application on a third-party system. These types of applications can be accessed on demand from any network-connected device without the need to install or maintain any software. SaaS model is very popular among companies because of its ease of adoption and use by customers; examples of SaaS applications include any of the different web-based services provided by Google such as Gmail, Docs, Sheets, and many others.

2.2 Cloud Security Key Concepts

Cloud Security consists of a set of technologies, controls, policies, and procedures employed to protect cloud-based infrastructures. These security measures are set up to safeguard cloud data, assist regulatory compliance, safeguard the privacy of clients, and define authentication guidelines for particular users and devices.

The growing migration of business-critical applications and data toward third-party cloud service providers is the main reason why nowadays cloud security is becoming so important. Even though the majority of CSPs already provide several cybersecurity tools with alerting and monitoring capabilities as part of their offer, those may not be enough to protect very large companies' infrastructures effectively.

To prevent breaches and data losses, avoid noncompliance, and grant business continuity, it is crucial to adopt specific security solutions and best practices aimed at reducing the overall risk of security incidents.

2.2.1 Cloud Security Main Challenges

The following are some of the challenges that can be faced in the context of cloud security:

- **Lack of visibility**: Since many cloud services are accessible outside of corporate networks and through third parties, it can be simple to lose track of how and by whom your data is being viewed.
- **Multitenancy**: Gartner defines multitenancy as *"a reference to the mode of operation of software where multiple independent instances of one or multiple applications operate in a shared environment. The instances (tenants) are logically isolated but physically integrated. The degree of logical isolation must be complete, but the degree of physical integration will vary."*. Public cloud service providers strongly rely on multitenancy to host multiple client infrastructures under the same physical one; therefore, it's feasible that hosted services of different clients will be infiltrated by malevolent attackers who are only targeting a specific company.
- **Access management** Access management in cloud environments may be very tough to achieve with the same level of restrictions that would be obtained in an on-premises environment; The risk of unauthorized access to cloud resources is even higher if we take into consideration organizations that still don't deploy bring-your-own-device (BYOD) policies and allow unfiltered access from any device or geolocation.

- **Compliance**: Cloud compliance is the art and science of complying with regulatory standards of cloud usage, following industry guidelines and local, national, and international laws. For businesses employing public or hybrid cloud installations, regulatory compliance management is sometimes a cause of complexity due to the vast amount of standards that have to be followed.
- **Misconfigurations**: Between 2018 and 2019, misconfigurations in cloud environments cost companies 5 trillion U.S. dollars, according to TrendMicro [21]. Some examples of misconfigurations include leaving default administrative passwords in place, or not creating appropriate privacy settings.

2.2.2 Cloud Security Solutions

Given the challenges of cloud security, some solutions have also been defined to cope with the dangers of cloud environments. The following list proposed by IBM [13] provides insight into some of the most fundamental solutions that any business working in the cloud has to consider to safeguard its infrastructure.

- **Identity and Access Management (IAM)**: The main purpose of Identity and Access Management is the provisioning of digital identities whose objective is to authenticate and authorize users to access specific on-premises and cloud-based services. Exploiting IAM tools, enterprises can deploy ad-hoc policy-driven enforcement protocols that can be easily tuned to restrict resource access only to authorized personnel. Another very important feature of identity and access management services is accountability; The latter refers to the principle of holding individuals or entities responsible for their actions and the consequences of those actions within a specific environment.
- **Data loss prevention (DLP)**: To ensure the security of cloud data it is necessary to put in place services capable of preventing data loss or destruction. Data loss prevention systems employ a combination of measures, such as remediation alerts and data encryption, to ensure that stored data is protected both at rest and in motion.
- **Security information and event management (SIEM)**: A solution to automate threat monitoring, detection, and response in cloud-based environments is a security information and event management system. With this security orchestration solution and its ability to aggregate logs coming from different platforms, IT teams can enforce their custom network security protocols while also being able to react to threats quickly.
- **Business continuity and disaster recovery**: Apart from adopting preventive measures to ensure cloud infrastructure security, companies should

also rely on disaster recovery solutions since data breaches and disruptive outages can still occur. With this kind of solution in place, enterprises get the ability to:

1. Proactively react to threats
2. Limit the damages of an attack
3. Recover lost data
4. Quickly resume normal business operations

2.3 Multi-cloud Environments

2.3.1 Introduction to Multi-Cloud

Multi-Cloud is an approach to cloud computing in which enterprises adopt more than one cloud deployment of the same type, from different vendors. The type of cloud deployment chosen by a company depends on the needs of the latter and can be classified into one of these two classes:

- **Public Cloud**: Computing services and infrastructure management is delegated to a third-party provider that sells these services according to one of the already explored cloud computing models, to many organizations.
- **Private Cloud**: Cloud computing model in which organizations can have their dedicated infrastructure. Private clouds can be either hosted in-house by the same organizations, inside colocation facilities, or by a private cloud provider.

Multi-Cloud deployments enable companies to allocate resources based on their needs thus improving overall flexibility, security, and resilience in the event of an outage with a single cloud vendor. Two additional benefits deriving from the adoption of a Multi-Cloud infrastructure are:

- **Proximity**: When poor response times become an issue for cloud users located far away from the organization's headquarters, relying on different cloud providers may be the perfect solution to deploy company services geographically near the end users in need. To accomplish this task, care has to be taken in choosing different cloud providers that have points of presence as scattered around the world as possible.

- **Reduced Costs**: Even though more resources will have to be managed across more than one cloud infrastructure, it can also be an opportunity for cost savings since companies get the opportunity to compare the costs of different services offered by each service provider, and choose the most suitable one.

2.3.2 Multi-Cloud Challenges

As an always-growing number of organizations are embracing multi-cloud environments, according to a survey from Gartner [7], 81% of respondents use two or more cloud providers, This presents new challenges in supporting existing and new application architectures and workloads, which are distributed across major clouds, at the edge, in colocation facilities, sovereign environments, and private data centers.

Additional problems arise considering that each cloud provider has its technology stack and delivers a unique set of capabilities that natively do not extend to other providers; these inconsistencies in the operating models of cloud infrastructures are a leading cause of increased complexity and overall risk.

IT professionals and developers have identified the following challenges to be the most demanding for the successful operation of a Multi-Cloud infrastructure:

- **Inconsistent infrastructure**: Without consistency among the deployed cloud infrastructures, teams end up working in isolated environments with limited flexibility in terms of strategies adopted to change business needs.

- **Continuously changing application landscape**: Support for DevOps, performance, and availability across multiple cloud environments is essential to allow faster development of applications and features, as well as to back the growing complexity of applications and infrastructures.

- **Inefficient management**: The lack of proper management tools and governance policies for multi-cloud infrastructures, significantly increases costs and risks from a security point of view.

- **Networking security**: Complexity in networking and in securing applications and data across clouds results in security flaws, risk exposure, and a larger attack surface. To tackle these problems, appropriate best practices should be followed when designing and operating a Multi-Cloud infrastructure.

- **Distributed workforce**: With more and more users accessing cloud resources from outside companies' networks, particular care has to be taken to guarantee an opportune user experience without jeopardizing security.

2.3.3 Multi-Cloud Security

During the last few years, it became clear that security in the cloud cannot be implemented in the same way as it was done on-premises. Likewise moving from securing a single cloud environment to securing many would ideally require well-defined procedures that should be adopted to implement security in a proactive way rather than in a reactive one.

This new approach to cloud security particularly stems from the way teams initially migrate to multi-cloud environments: These migrations are often driven by the need for speed and enhanced functionality, which typically results in the adoption of decentralized solutions exclusive to each team.

To support this new security paradigm appropriate elements have to be introduced and exploited; a Multi-Cloud security approach mainly consists of the policies and tools in charge of protecting workloads, applications, and data across cloud service providers. In the following paragraphs, the most important components of a Multi-Cloud security solution will be explored.

Automation
Since most vulnerabilities are due to misconfigurations, automation can reduce the likelihood of such events occurring. A security foundation can be achieved by automating deployments and by setting up policies and guardrails in any cloud environment. In the event of a deviation from a secure state, monitoring can be also leveraged to trigger automated tools that will restore original configurations. Additionally, automation can be particularly useful for threat monitoring, incident remediation, and ensuring that a common security posture is maintained regardless of the cloud provider.

Dashboarding
In the context of Multi-Cloud security, centralized monitoring is essential to collect logs and data from all cloud locations; to properly analyze this kind of information, visualization tools designed for Multi-Cloud become a key element to provide, through dashboards, a comprehensive view of all cloud environments.

Tooling
Security tools designed for Multi-Cloud deployments operate across cloud service providers and offer the following features:

- Standard integration patterns that can be followed to incorporate this solution with one or more providers
- Configurations and policies that can be enforced selectively on the deployed infrastructures
- Centralized dashboard and analytic solution that can be used to ingest and visualize data from multiple cloud providers

DevSecOps
DevSecOps methodology allows embedding security tooling into the software development lifecycle; by doing so it is possible to: detect potential security vulnerabilities as early as possible, remediate them more efficiently, and reduce the overall costs of software development. Given that the employed security tools were effective, the result will be a more secure product.

2.4 Cloud Governance

2.4.1 Introduction

Cloud Governance is a process aimed at defining, implementing, and enforcing a set of rules and policies adopted by companies to operate their services on the cloud. Cloud governance processes ensure that aspects like data security, asset deployment, and systems integration are planned and managed in a compliant way with respect to company policies and third-party regulations.

Organizations' cloud infrastructures and services deployed on the latter can benefit in different ways from the adoption of a well-defined governance framework:

- **Improve Cloud Resource Management**: A best practice suggested by major cloud providers is the breakdown of cloud systems into separate accounts each representing an organization's bounded context such as a department or a project. This kind of segregation aims at improving cost management and visibility as well as limiting the impact of potential attacks.

- **Reduce Shadow IT**: Shadow IT resources in large organizations encompass all those systems, services, devices, and software applications that are being used without explicit consent of the company's IT department itself. While these resources may prove to be useful to employees, they could also introduce security risks and violate compliance policies. With Cloud Governance employees are given a convenient way to access cloud systems and ask for cloud resources, thus reducing the need to turn to shadow IT.
- **Reduce Administrative Overhead**: In the scenario of a company not operating a governance framework, tracking of cloud accounts, costs, compliance issues, and many other business-specific aspects related to the cloud, would be managed for instance through spreadsheets, which are an inefficient and an unscalable approach when organizations are huge. Cloud Governance solutions enable centralizing policy definition, compliance, cost management, access control, and also facilitate security auditing and response to violations.
- **Improve Cloud Security**: A Cloud governance model defines the strategy to protect a company's data confidentiality, integrity, and availability. By designating and implementing such a solution, an organization won't have to be concerned about where services are actually deployed since visibility and enforcement of security controls will be granted by the governance solution.

2.4.2 Cloud Governance Framework Components

The definition of a cloud governance framework starts by identifying its primary components. Each of them will supervise independent management aspects of an organization's cloud environment.

1. **Cloud Financial Management**: Even though cloud services promise the reduction of IT costs, this only applies if there is a proper management of the latter. Financial management in the cloud can be obtained: by enforcing financial policies that clarify how the cloud will be used, defining budgets allocated to different organizations' cloud services, and reporting accurately the charges deriving from each deployed cloud resource.
2. **Cloud Operations Management**: Collection of processes that describe how services are deployed. Several processes that serve this purpose can be identified, such as:
   - Definition of resources allocated to services.
   - Determine expected Service-level agreements and ensure they are continuously met.

- Access controls concerns

  Adopting proper operations will greatly contribute to preventing the growth of shadow IT resources and reduce costs.

3. **Cloud Data Management**: In an environment where huge amounts of data are regularly collected and analyzed, it becomes essential to establish well-defined data management policies. Cloud governance in this context specifies ad-hoc techniques that can be employed to guarantee a proper administration of the data lifecycle in the cloud:

   - Set policies for data depending on their sensitivity.
   - Encrypt data both at rest and in transit.
   - Apply data masking while developing or testing, to reduce the possibility of sensitive leaks
   - Automate data lifecycle management.

   In recent years as a result of regulations like GDPR (General Data Protection Regulation) in the European Union or the PIPL (Personal Information Protection Law) in China, which companies must be compliant with in order to handle citizens' data, it became crucial to implement previously defined protection measures in order to continue providing cloud services without incurring fines.

4. **Cloud Security and Compliance Management**: Cloud Governance in the context of enterprise security determines organizations' security and compliance requirements whose enforcement must be made sure. Key security domains and activities that should be performed are:

   - Risk assessment
   - Identity and access management
   - Data management and encryption
   - Application Security
   - Disaster recovery

2.5 Cloud Security Literature Review

2.5.1 Analysis of Cloud Computing Problems

As cloud computing and all its derived paradigms kept on diffusing among every type of business, researchers started questioning its security and tried to enumerate the problems stemming from its adoption, along with proposing solutions to tackle these security issues. This paper by Mohamed Almorsy, John Grundy, and Ingo Müller [1] analyzes existing challenges and issues involved in the cloud computing security problem, and groups them into the following categories:

- **Architecture-related issues**
- **Cloud characteristic-related issues**
- **Service delivery model-related issues**
- **Cloud stakeholder-related issues**

Architecture-related issues

The cloud computing model as already discussed in section 2.1.1 has three service delivery models, namely IaaS, PaaS, and SaaS; moreover, it also has three main deployment models: private cloud and public cloud presented in section 2.3.1, and an additional one named Hybrid Cloud in which a private cloud gets integrated seamlessly with one or more public clouds.

Regarding the deployment models, the authors of the article identify the public cloud as the most vulnerable model since *"they are available for public users to host their services who may be malicious users."* [1]

From the perspective of the service delivery models, each may have different possible implementations, further complicating the definition and development of a standard security model. Furthermore, other service delivery models may coexist in a single platform thus increasing the complexity of the security management process.

Cloud stakeholder-related issues

In the cloud computing model, three different stakeholders can be identified:

- **Cloud Provider**: entity that delivers resources to the cloud consumers
- **Service Provider**: an entity that uses the cloud infrastructure to deliver a service to the end users
- **Service Consumer**: entity that uses cloud-based services

Each stakeholder manages security with different systems and processes and has distinctive expectations and capabilities concerning what should be offered to, and obtained from others. This leads to:

- Potentially conflicting security requirements defined by different tenants on a single service.
- Need for an agreement on the applied security properties by providers and consumers, however, no security specification notation that can be used by cloud stakeholders to represent this kind of property exists.
- Stakeholders identify assets, risks, impacts of the latter, and mitigation techniques according to some specifically adopted security management processes. With this approach, cloud providers tend to lack awareness of the services' security requirements deployed on their infrastructure, while cloud consumers may lose control of their asset's security.

Service delivery model-related issues

Given the proposed three service models 2.1.1, researchers identified key security vulnerabilities for each of them. Responsibility for these issues may be either of cloud providers or cloud consumers.

- **IaaS Issues**
  - **VM security**: securing virtual machines operating systems and workloads from common security threats. It is the responsibility of cloud consumers.
  - **VM images repository security**: protecting image repositories from the injection of malicious code into virtual machine files is a task that should be taken care of by cloud providers.
  - **Virtual network security**: securing network infrastructures shared by multiple tenants at the same time from attacks that target DNS servers, DHCP, and the IP protocol in general.
  - **VM boundaries security**: protecting virtual boundaries of virtual machines (i.e. managing access to shared resources such as CPU, I/O, and memory) running on the same physical server is the responsibility of the cloud provider.
  - **Hypervisor security**: a hypervisor is the software component in charge of mapping physical to virtualized resources. Compromising its security means exposing virtual machines to a large variety of attacks. In this

case, the responsibility to protect the hypervisor is both of the cloud provider and of the service provider (i.e. the company that realized that specific hypervisor)

- **PaaS Issues**
  - **SOA related security issues**: PaaS takes inspiration from the older Service-oriented Architecture model, therefore it also inherits its known security issues concerning DOS, Man-in-the-Middle, XML, and replay attacks.
  - **API Security**: Business and security functions offered by PaaS model may be delivered through an API; it is the duty of the cloud service provider to properly implement security controls and follow standards aimed at protecting this collection of APIs.
- **SaaS Issues** Security issues in the Software-as-a-Service model inherently derive from the previous two, as SaaS is built on top of those; this includes data locality, integrity, backups, and confidentiality issues. Maintaining the security of SaaS-based services is a shared responsibility among cloud providers and service providers (i.e. vendor of the software).

Cloud characteristic-related issues

From the point of view of cloud providers, the profitability of the cloud computing model derives from an increase in resource utilization while at the same time reducing costs. Such needs are met in the cloud computing world by relying on two key concepts: multi-tenancy and elasticity. Multi-tenancy allows resource sharing among tenants and can be realized in different ways:

1. Each tenant uses a dedicated instance with custom configurations.
2. Tenants use each a dedicated instance with standard configurations.
3. Tenants share the same instance.
4. Tenants are redirected by a load balancer to a specific instance which may be used also by other tenants.

The most dangerous approaches are the third and the fourth ones since tenants are sharing the same process. Secure multi-tenancy may be delivered if data isolation and location transparency of tenants' resources are enforced.

2.5.2 Security-as-a-Service in Cloud Environments

Security-as-a-Service (SECaaS) is a cloud model in which security services can be outsourced. Similarly to SaaS, Security-as-a-Service provides security services on a subscription basis. SECaaS solutions have gained in the last years a lot of popularity thanks to their capabilities and ease of integration in corporate infrastructures.

Nowadays a wide variety of Security-as-a-Service solutions have been proposed to protect cloud environments from the majority of threats such as data losses, vulnerability scanning, network security, intrusion protection, and many others. A concrete example is the Data Encryption-as-a-Service solutions proposed by major cloud providers like Google and Amazon [8] [2] that allow to automatically encrypt all data stored in Amazon Web Services buckets or in Google Cloud storage.

In the following paragraphs two different frameworks for providing data protection as a service in multi-cloud and federated environments will be discussed.

Security-as-a-Service in Multi-cloud and Federated Environments [19]

This paper, which was written by researchers Pramod S. Pawar, Ali Sajjad, Theo Dimitrakos, and David W. Chadwick proposes an application protection solution (i.e. Intelligent Protection) and a data protection solution (i.e. Secure Cloud Storage) delivered as a SaaS application that can be integrated into multi-cloud and federated environments. Both the host and the application protection solutions described in this paper aim at providing cloud operators the ability to dynamically provision protective functionalities while still maintaining full control over their services' security.

Secure Cloud Storage - Data Protection Service
Secure Cloud Storage (SCS) is a data protection service for public and private clouds that allows users to keep their sensitive data confidential by relying on a volume encryption service. This service can be deployed both following the SaaS model and on-premise, in either case, the user is the only one that can access decryption keys; The latter aspect offers users the capability to:

- Decrypt data on demand
- Apply policy-based key management to validate workload identity

Additional key technical features of SCS described by the authors of this paper are:

- Advanced Encryption Techniques
- Robust Auditing, Reporting, and Alerting

Intelligent Protection - Host and Application Protection Service
Cloud security service is designed to protect cloud infrastructure workloads from various threats. This system, offered to users via a web interface, encompasses many security functions such as intelligent intrusion detection and prevention, a bi-directional stateful firewall, anti-malware, integrity monitoring, incident reporting and analysis, and recommendation scans.

Data Protection as a Service in the Multi-Cloud Environment [5]

The existing Data-Encryption-as-a-Service solutions briefly described in section 2.5.2 offered by providers like Google [8] and Amazon [2], have some limitations. For instance, these encryption services are bound to a specific cloud provider and they don't include in their scope functionalities like access control policies and other basic security primitives.

These considerations led the papers' authors to develop their framework for providing Data Protection as a Service to data storage in Multi-Cloud environments. Concerning previous Data Encryption as a Service solution: *"this work aims to provide a full life-cycle of data security management and to maximize the flexibility of users in using the data encryption service, additionally this solution is extensible to work with different cloud platforms and security solutions"* [5].

Because of the previous motivations and also the intention to separate key management from encryption operations, researchers proposed an integrated framework made available as an additional component to Storage as a Service (STaaS).

As also stated by the authors, this work can be considered as an extension with more mature ideas and a comprehensive low-level design architecture to [19] since both aim at providing data encryption as a service. An analysis of the major objectives and features provided by this solution follows:

- Service available in cloud providers' marketplaces to be purchased and adopted by customers to protect data stored on one or more cloud platforms.
- Agent-based solution that prevents performance bottlenecks by encrypting data on the client side, ensuring no cloud provider with proper decryption keys will be able to access it.
- Policies for users' access control and the definition of application and environment constraints.
- Support for several cloud storage services such as object stores, virtual volumes, and big-data warehouses.
- Mobility of encrypted data from one storage service provider to another.
- Policy-based approval procedures are put in place to handle the decryption of protected data. This means that no decryption can happen without data having undergone such procedures.
- Compliance with data security regulations and custom pre-defined policies.

Security-as-a-Service: Summary

Even though the preceding discussed papers proposed two valuable Security-as-a-Service solutions for data protection across cloud platforms, they only cover a few aspects of the whole cloud and multi-cloud security landscape; apart from data protection, many other security matters like cloud resources misconfigurations and run-time threat detection need to be addressed. In a subsequent chapter of this work, a complete solution for multi-cloud security based on a third-party platform will be explored.

Chapter 3

Policy as Code

3.1 Introduction

Policy-as-Code (PaC) is a policy management approach that leverages the use of policies defined and enforced using code. By describing policies in the form of a text file written using a high-level language, it becomes more straightforward to adopt software development best practices like code versioning, test automation, and continuous deployment.

A policy in this context can be described as a rule that has to be followed, a condition that has to be fulfilled, or even a set of procedures automatically executed upon the occurrence of a specific event. Policies, usually depending on the framework and tools used to enforce them, can be written in either special-purpose policy languages like Rego, or other languages like YAML or Python.

Policy-as-Code can be compared in a way to the more spread Infrastructure-as-Code (IaC) paradigm, the latter is commonly used by IT operations teams to automatically provision infrastructure components leveraging a declarative language (e.g. YAML, JSON). PaC's primary objective on the other hand is to ease security operations, data management, and compliance management.

As of now, policy-as-code is the only alternative to the manual management of rules, conditions, and procedures; the reason why it is becoming more and more used across enterprise environments mainly derives from the benefits it offers, according to the American multinational cybersecurity company *Palo Alto Networks* [14]:

- **Efficiency**: By writing policies in the form of code they can be automatically enforced without requiring any manual action by a designated operator. Moreover defining clear code policies also allows for avoiding the majority of misinterpretations that stem from writing policies in natural human language.

- **Speed**: Automation of policy enforcement undoubtedly leads to an improved velocity in operations as opposed to a manual approach.
- **Visibility**: Rules defined utilizing code files greatly enhance visibility on what is happening in a system, also giving the possibility to easily review which remediation and alerting rules have been deployed.
- **Collaboration**: Both intra-team and inter-team collaboration benefit from having a uniform and centralized approach to policy management. This aspect is particularly important when coordination among professionals with different skill sets is required (e.g. developers, security engineers, operations engineers, ...).
- **Accuracy**: Management of policies using code is less error-prone with respect to manual system management.
- **Version Control**: If policies are tracked in a versioning control system as they evolve in time, IT operations teams acquire the ability to revert to a previous policy configuration. This is an extremely important feature whenever a new policy causes problems.
- **Testing and Validation**: Validation of code-written policies performed by specific auditing tools can significantly reduce the number of errors and misconfigurations that reach production environments.

3.2 ”Shift Left Security” Concept

The term *”Shift Left”* in the context of development operations is the practice of moving application correctness and security controls at earlier stages of the development lifecycle, thus moving these controls to the left in contrast with where they used to be performed in the past.

”Shift Left Security” in particular, is the practice of implementing security checks not only at the end of the development but during any phase of the development lifecycle. Up until the last few years, security testing like static analysis (SAST) and dynamic analysis (DAST) were only performed before application deployment; this practice led inevitably to long delays in the occurrence of errors that required developers’ intervention.

This novel approach to security controls’ implementation aims at designing software with built-in security best practices, but also at detecting and remediating potential vulnerabilities as early as possible. The following list provides a collection of recommendations and practices that can be adopted to shift security left:

- **Establish Shift Left Security Policies**: Defining security policies before the application's development has even started is a recommendable practice to set boundaries that will securely guide the development process.
- **Automate Security Controls**: Security operations teams should adopt automatic tools to detect, audit, and remediate potential threats to systems. Embracing this strategy will also lead to a sped-up software development life cycle.
- **Implement Security Fixes as Code is Written**: Developers should receive continuous feedback in terms of security about the code they're writing. By leveraging this feature developers get the possibility to quickly fix potential security issues that would have been neglected otherwise.
- **Enhance Visibility Into Application Security**: One of the main goals of shift-left security is to secure code during the whole software development lifecycle; to achieve this and the ability to remediate issues with ad-hoc software updates, visibility into application security is required.

3.3 Development, Security, and Operations

DevSecOps, short for development, security, and operations, it's an approach to automate the integration of security controls and best practices throughout the whole software development lifecycle: starting from the design, all the way to the delivery of the final product.

DevSecOps represents the natural evolution of DevOps that only takes care of development and operations, without considering security aspects. Nowadays this trend is changing dramatically, Among the predominant factors we identify the adoption of agile methodologies that have significantly reduced over the last decade the duration of individual software development cycles. In the context of Agile processes, DevSecOps seamlessly integrates infrastructure and application security by taking advantage of tools able to address security issues when they're easier and less expensive to remediate.

Another concept introduced by the DevSecOps approach is the shared responsibility of security. The latter is no longer an exclusive duty of security teams but it is also taken care of by development, and operation teams.

All in all the most important features introduced by DevSecOps practices can be summarized as follows:

1. **Improved and Proactive Security**: Cybersecurity processes are introduced from the beginning of the development; code is scanned and tested for vulnerabilities continuously and as soon as a security issue is detected, it can be immediately fixed, thus leading to fewer bugs in production environments and to lower development and maintenance costs.

2. **Fast Security Vulnerability Patching**: By employing proper vulnerability scanning and patching tools, the identification of common vulnerabilities, as documented in the most famous public databases (e.g. CVE, NVD, Exploit-DB,...), and their resolution becomes a rapid automatic process.

3. **Rapid and Cost Optimized Software Delivery**: Fixing security issues of projects managed in non-DevSecOps environments leads to a considerable waste of resources in terms of time and money. On the other hand, integrating security checks within every development phase allows for cutting costs, avoiding duplicate reviews and futile software rebuilds.

4. **Processes Repeatability**: Organizations adopting DevSecOps practices should aim to automate most of the processes carried out during the software development lifecycle. In such a scenario, security would be able to adapt to new requirements and to the changing environment in which it is applied. Additionally automating these processes grants the ability to repeat security controls whenever it is deemed necessary.

3.3.1 Infrastructure as Code Security

The growth of DevSecOps led to the diffusion of Infrastructure as Code Security which can be seen as one of the essential considerations in DevSecOps. While Infrastructure as Code (IaC) can be defined as the management and provisioning of infrastructure through code, IaC Security ensures that resources provisioned leveraging IaC template files are secure. To achieve this goal, IaC security relies on best practices and compliance requirements related to different areas of IT security, such as:

- Data Encryption
- Access Control Requirements
- Log Collection and Retention
- Network Segmentation

An example of IaC security is the identification of misconfigurations within IaC templates and modules; these resources are scanned against known policies to detect any violation that could pose risks to the security of a system. Since infrastructure as code was designed to be agnostic of the specific cloud infrastructure on which it is deployed, automation of IaC scanning is indispensably required to enforce and manage all existing policies that may differ across public or private cloud providers. For this purpose, some open-source frameworks and tools like Open Policy Agent (OPA) and Checkov, were made available.

To derive added value from IaC security, automation by itself is not enough; IaC-Sec scannings should be performed periodically not only to discover and remediate misconfigurations but also to avoid cloud drifting, which happens when the real-time state of a cloud infrastructure does not match the infrastructure-as-code configuration. In this situation, Continuous Integration (CI) and Continuous Delivery (CD) pipelines can be effectively employed to enforce compliance policies and to deploy resources defined within IaC templates.

Infrastructure as Code Security Best Practices

Considering that infrastructure as code security is a recent topic, there are only a few processes to implement it that can be recognized as well-established. Nevertheless, during the last decade, several best practices were outlined to ease the task of implementing IaC security in enterprise environments:

- **IaC Templates Hardening**: Security risks may be deriving from different aspects of IaC templates, like resources with known vulnerabilities, misconfigurations, or even from the usage of container images coming from unknown sources. An initial step to implement security is to harden IaC templates against these common issues.

- **Secrets Management**: Among the most common security risks, secrets management covers a predominant role. In the majority of cases, the risk of secrets disclosure is not bound to the secrets themselves, but rather to how these sensitive data are stored and managed: in not-so-isolated occurrences, inside public git repositories it has been noticed the presence of plaintext hard-coded secrets that could have been used by anybody to gain improper access to accounts, systems or even to entire company's infrastructures.

- **Communication Channels**: In a master-node architecture, that is the preferred one for the implementation of Infrastructure-as-Code management tools, security risks may arise if the master is not secured properly. Communications between the master and nodes should be encrypted, and multiple security mechanisms should be implemented in each node to adhere to the security-in-depth principle.

- **Configuration' Drift**: In the event of configuration changes directly in production environments, security issues may be introduced due to a deviation from the initially defined security posture. To avoid discrepancies between the original security posture definition and the actual one, ad-hoc IaC security practices can be taken advantage of.

- **Ghost Resources**: Resources deployed in cloud environments with no tags associated take the name of *Ghost Resources*. Preventing the number of these resources from growing exponentially is important to grant assets observability and detect potential threats that would have gone unnoticed.

- **Data Transmission Risks**: Securing data during transmission is as important as securing it while at rest. To meet the former requirement, care must be taken to avoid misconfigurations in TLS certificates and VPN connections.

- **Logs Auditing**: Every company's cloud infrastructure should be equipped with adequate tools to collect systems logs that can be analyzed in the event of a security incident.

3.3.2 GitOps Approach

While Nowadays software development lifecycle has almost been completely automated to support developers during their activities, infrastructure is partly still a manual process. In modern organizations' cloud infrastructures, GitOps uses Git repositories for automating the provisioning of cloud resources; these repositories shared by all team members contain the configurations and IaC templates used for deploying all the resources.

There are three core components required by GitOps to be successfully integrated into an enterprise environment:

- **Infrastructure-as-Code**: When infrastructure-as-code templates are tracked in a versioning control system like git, operations teams get the ability to rapidly deploy whichever version of the infrastructure is needed. This becomes particularly important in the event of a component failure caused by an update to its configuration.
- **Pull/Merge Requests**: Pull Requests (PRs) and Merge Requests (MRs) are extremely helpful in git repositories to support infrastructure updates. PRs and MRs promote collaboration among team members giving a way to review, comment, and eventually approve code changes submitted by others.
- **CI/CD Pipelines**: Continuous Integration (CI) and Continuous Delivery (CD) pipelines can be integrated into GitOps workflows to react to updates. When new code is merged, a pipeline that will perform operations like code compilation, automated tests, and deployment is triggered. GitOps pipelines have also the effect of avoiding any configuration drift by converging the current state of an infrastructure to the desired state as defined in the central git repository.

3.3.3 GitOps, DevOps, and DevSecOps

GitOps is the practice that guides software development and infrastructure provisioning relying on Git repositories. DevOps on the other hand is a culture that promotes team collaboration and shared responsibility, ultimately leading to reducing the whole software development lifecycle.

GitOps and DevOps may seem conceptually similar as both employ Git as a version control system and implement IaC processes and CI/CD pipelines, however, their difference lies in their scope of application:

GitOps workflows may be set up by DevOps teams to acquire their intrinsic advantages (3.3.3), nevertheless, it is not mandatory for operation teams to incorporate GitOps practices. This relationship between the two leads to identifying GitOps as a specialized method of DevOps that aims at relaxing the bounds between development and operations phases.

DevSecOps as already described in section 3.3 is an extension of DevOps, as such it presents the exact differences and is bound in the same way to GitOps as just explained for DevOps. When used in the context of DevSecOps environments, certain GitOps practices could be applied to implement security controls that wouldn't have been as effectively employed in a traditional DevOps environment.

3.3.4 CI/CD Platforms

A CI/CD platform is a set of tools and services that allows developers, engineers, and operations teams to successfully automate the process of building, testing, and deploying new code, along with the features it introduces to production environments. Every CI/CD platform aims to provide the elements prescribed by DevOps and DevSecOps frameworks (i.e. Development Speed, Code Quality, and Overall Efficiency) and offers a rich set of functionalities such as:

- **Continuous Integration (CI)**
- **Continuous Delivery (CD)**
- **Version Control Integration**
- **Automated Builds**
- **Automated Testing**
- **Deployment Automation**

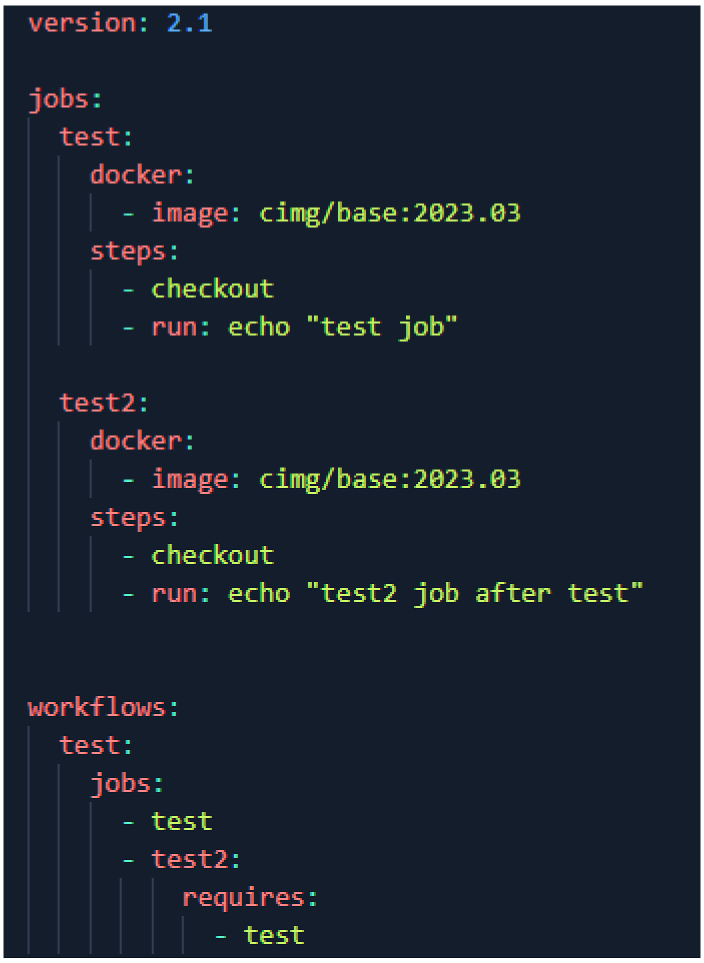
Many CI/CD platform solutions are available on the market; in the following paragraphs, the features provided by two of them will be analyzed to highlight their similarities and differences.

CircleCI

CircleCI is a CI/CD platform available as a SaaS solution through a Web Application; among its most important features there are:

- Integration with Version Control Systems of different vendors (e.g. GitHub, GitLab, BitBucket).
- YAML-based syntax that allows to define workflows, jobs, and their steps.
- Visualization of executed pipelines.
- Advanced caching mechanisms to speed up pipeline executions.
- Workflows executions inside Docker containers.
- Possibility to use reusable units of code to perform a large variety of tasks (i.e. Orbs).
- Matrix builds that allow running workflows with different inputs and in different environments.

Figure 3.1 shows a CircleCI example workflow that executes two simple jobs one after the other



```yaml
version: 2.1

jobs:
  test:
    docker:
      - image: cimg/base:2023.03
    steps:
      - checkout
      - run: echo "test job"

  test2:
    docker:
      - image: cimg/base:2023.03
    steps:
      - checkout
      - run: echo "test2 job after test"


workflows:
  test:
    jobs:
      - test
      - test2:
          requires:
            - test
```

Figure 3.1: CircleCI Workflow Definition

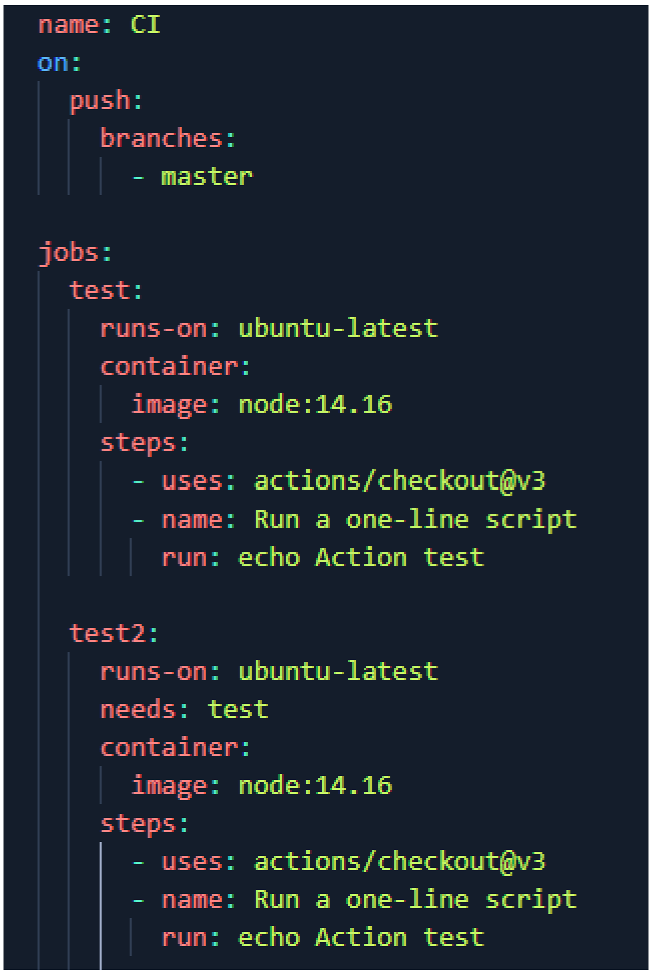
Github Actions

GitHub Actions is also a CI/CD platform that provides the capability of defining workflows and executing pipelines, but unlike CircleCI, GitHub Actions is seamlessly integrated with GitHub, hence with applications' and infrastructure's source code. Even though this feature introduces a lot of advantages, it also presents a major issue: Actions' workflows can only be defined for source code stored in GitHub repositories; this means that if a company is relying on platforms like GitLab or BitBucket, it will have to employ a different CI/CD platform than Actions. Additional features offered by this platform are:

- Deep integration with GitHub environment and the source code.

- YAML-based syntax to define jobs and their steps.
- Visualization of executed pipelines.
- Workflows executions inside Docker containers.
- Matrix builds (as explained for CircleCI 3.3.4)
- Huge ecosystem of Actions used to perform building, testing, and deploying tasks (similar concept to CircleCI Orbs)

The following figure 3.2 shows an equivalent GitHub Actions workflow as the one shown for CircleCI in figure 3.1.



```yaml
name: CI
on:
  push:
    branches:
      - master

jobs:
  test:
    runs-on: ubuntu-latest
    container:
      image: node:14.16
    steps:
      - uses: actions/checkout@v3
      - name: Run a one-line script
        run: echo Action test

  test2:
    runs-on: ubuntu-latest
    needs: test
    container:
      image: node:14.16
    steps:
      - uses: actions/checkout@v3
      - name: Run a one-line script
        run: echo Action test
```

Figure 3.2: GitHub Actions Workflow Definition

3.4 Cloud Security Posture Management

Cloud Security Posture Management (CSPM) is a Governance practice employed in cloud environments to detect misconfigurations and compliance risks. The term *Posture Management* in fact, refers to the configuration of resources and services of one or more cloud accounts, that are required to be compliant with well-defined industry standards. CSPM encompasses a wide range of tools, policies, and best practices that are leveraged to automatically discover and remediate compliance issues.

The following list presents some of the key capabilities of CSPM enterprise tools:

- **Detect and automatically remediate cloud misconfigurations**
  Example: *Azure Virtual Machine disk has not been encrypted, AWS EC2 instances exposing a public IP, ...*

- **Maintain an inventory of best practices scoped to each different cloud account**
  Since public clouds like Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP) offer each a different set of services with equally different configurations, several sets of policies and best practices need to be adopted depending on the specific cloud resources' vendor.

- **Ensure configurations compliance against security control frameworks or regulatory standards**
  To ease the task of ensuring compliance across heterogeneous environments, several cloud security frameworks and regulatory standards have been defined:

  - **CIS Benchmarks**: Provides prescriptive configuration recommendations for products of different vendors.
  - **NIST SP 800-53**: Comprehensive catalog of privacy and security controls for federal information systems, including cloud-based ones.
  - **ISO/IEC 27001**: World-acknowledged standard for information security management systems that provides a systematic approach to managing sensitive information and cloud assets.

- **Identify misconfigurations in storage buckets**: Ensuring the protection of data stored in the cloud from unauthorized access is critical. To accomplish this task, CSPM tools can monitor storage accounts (i.e. Azure equivalent of AWS S3 buckets) to detect unencrypted or un-protected data.

3.5 Cloud Infrastructure Entitlement Management

Cloud Infrastructure Entitlement Management (CIEM) is a security process leveraged in cloud and Multi-Cloud infrastructures, aimed at managing access rights, permissions, and privileges granted to identities. CIEM solutions can be put in place to apply the *Least Privilege* principle and to limit the risks deriving from granting higher privileges than what is needed to perform a specific task.

When working in the context of highly dynamic and ephemeral cloud infrastructures, proprietary IAM solutions provided by public cloud providers are usually not well suited to manage identities and privileges by themselves. CIEM tools can help address these issues by improving visibility and allowing for the detection and remediation of misconfigurations more rapidly.

CIEM mainly consists of three components:

1. **Entitlements Visibility**: Understand which identities are present in a given cloud or multi-cloud infrastructure and the permissions they are assigned.
2. **Permissions Rightsizing**: Apply the least privilege principle by granting only the necessary subset of permissions required to perform a task.
3. **Analytics and Compliance**: Rather than using rules, CIEM tools may rely on advanced analytics to detect entitlements misconfigurations and to align them according to compliance requirements.

Implementing a CIEM solution within organizations that rely on Multi-Cloud infrastructures is particularly important thanks to the benefit it introduces:

- Complete view of identities, permissions, and policies spread across multiple cloud environments.
- Automatic detection of malicious activities such as accounts being compromised, stolen access keys, and other suspicious user behaviors.
- Full compliance to regulatory standards related to user permissions.

3.6 Cloud Workload Protection

3.6.1 Introduction

In today's scenario where the number of attacks targeting enterprises and their cloud infrastructures is rising exponentially, it is becoming necessary to not only implement endpoint protection by limiting access to endpoint devices but also to secure resources at the workload level. Check Point's 2022 Cloud Security Report [4] affirms that 27% of businesses have suffered a security breach in their public cloud infrastructure; This highlights the importance of Cloud Workload Protection (CWP) practices, which are indispensable to secure workloads across different cloud environments.

The reason why workload protection is so important resides in the concept of *cloud workload*: the latter includes the application, the data managed by this application, and the network resources that connect users to it. If any of these components get compromised by an attack, the whole cloud-based application will not function as expected.

3.6.2 Cloud Workload Protection Platforms

Cloud Workload Protection Platforms (CWPP) are a technology that can be leveraged to implement the process of CWP. A CWPP solution works by initially discovering all the workloads that exist within an organization's cloud environments, and then by performing a vulnerability assessment aimed at discovering workloads' security flaws. Due to the heterogeneity of workloads and all the parts composing them, CWP platforms usually rely on the paradigm of Policy-as-Code to enforce regulatory and industry standards. Given the results of the vulnerability assessment, a CWPP solution should also provide a way to remediate each of the issues that were identified.

Dealing with security concerns that have been detected in vulnerability assessments it's not the only task that should be addressed by workload protection solutions. The most effective CWP platforms also provide additional features to both cloud and on-premise workloads such as:

- **Container and Kubernetes Security**
- **Runtime Protection**
- **CI/CD Pipeline Security**
- **Whitelisting**
- **Cloud Network Security**
- **Visibility and Discovery**
- **Intrusion Prevention**
- **Application Security**

All in all the proper implementation of CWP solutions within organizations' infrastructures addresses the challenge of managing and securing workloads distributed across several environments (e.g. Multi-Clouds and Hybrid Clouds) and introduces a wide variety of benefits that can be summarized as follows:

- **Workloads Monitoring**: Through workloads' behavior monitoring, CWPPs provide two very important features related to workload security: detection of intrusions and response. Upon detection of unauthorized access, a workload protection solution should be capable of alerting IT security teams as soon as possible to prevent further damage.
- **Workloads Visibility and Configurability**: Visibility inside individual workloads is also granted by these solutions, thus giving an insight into what is happening inside each resource, and a means of configuring them.
- **Centralized Log Management**: As each workload may be associated with different security technologies, it is of extreme importance for CWP platforms to offer security teams a comprehensive and centralized view that explains what is happening across various environments.
- **Vulnerability Management**: Performing periodical vulnerability assessments allows the detection of potential attack vectors such as libraries and accounts. Once identified these resources can be promptly eliminated before they put any workload in danger.

- **Threat Intelligence**: Threat Intelligence is defined by Crowdstrike as " *data that is collected, processed, and analyzed to understand a threat actor's motives, targets, and attack behaviors*. Using this data and being aware of common attack patterns, cloud workload protection solutions can quickly recognize and neutralize new potential threats.

Chapter 4

Cloud Security in a Realistic Multi-Cloud Infrastructure

4.1 Introduction

In Chapters 2 and 3 many concepts related to Cloud Security and Policy as Code were discussed from a theoretical point of view, revealing some of the most common best practices and tools that are nowadays globally employed in organizations. In Chapter 4, I will initially discuss and demonstrate the implementation of a practical IaC Security pipeline, Afterwards, I will also explain how the setup of a realistic corporate Multi-Cloud infrastructure was carried on.

4.2 Infrastructure as Code Security

4.2.1 Checkov

Checkov is a static code analysis tool developed by Prisma Cloud that can detect misconfigurations in Infrastructure as Code files. By itself, Checkov already includes a pre-defined set of almost 750 policies, but it also allows writing custom policies in either Python or YAML languages. Since this open-source tool is capable of scanning many IaC file types such as Terraform for AWS, Google, Azure, and Oracle public cloud providers, Helm charts, Kubernetes, Docker templates, and others, it is widely utilized inside DevSecOps pipelines to detect and prevent the deployment of misconfigured resources on public and private clouds. Additionally, Checkov compliance scans against common industry standards, (e.g. CIS Benchmarks, AWS, Azure, GCP Best Practices Benchmarks) are another feature that makes this tool a great choice for companies with strong compliance requirements.

4.2.2 Sysdig

Sysdig is a Cloud Native Application Protection Platform (CNAPP) that offers a variety of functionalities to protect Cloud and Multi-Cloud environments. As this platform will be extensively discussed in Chapter 5, here I will only describe its IaC scanning tool. Sysdig currently supports GitHub, Bitbucket, GitLab, and Azure DevOps; once integrated into one of the previous version control platforms, this technology can be used to scan incoming Pull Requests against pre-defined policies. As it was for Checkov, the main purpose of this tool is to leverage the Policy-as-Code paradigm to detect security issues that could expose resources to a wide range of attacks.

4.2.3 IaC Security Implementation Scenario

Introduction

In the following paragraphs, a realistic scenario where DevSecOps and IaC Security practices were successfully implemented, is shown. In particular, the developed DevSecOps pipeline will cover the aspects of scanning Terraform files and K8s templates for compliance, and then only in case there are no security issues, the automatic deployment of Terraform resources on Azure public cloud will also be performed.

Environment and Tools

The environment chosen for this demonstration is a GitHub repository that ideally represents the code base where organization developers push new or updated IaC templates that are then deployed on their public cloud infrastructure. The IaC security tools that have been integrated are the ones discussed in the two previous sections, namely:

- **Checkov**: A specific GitHub Action starts the code scanning process and once finished produces a SARIF file reporting all detected misconfigurations.
- **Sysdig IaC Sec Scanning**: This tool is offered by Sysdig developers in the form of a GitHub Application that will run whenever a new pull request is opened on the GitHub platform. Specific configurations to modify the behavior of this tool may be applied directly from Sysdig's SaaS dashboards that will be shown in Chapter 5

GitHub Workflows

GitHub Actions CI/CD platform has been chosen over CircleCI due to several factors, such as:

- Seamless integration of GitHub Actions with GitHub platform
- Ease in defining multiple different workflows
- Availability of *Actions* for any kind of task

To automate the code scanning and deployment processes I've realized two different workflows written using GitHub Actions syntax. Each workflow is executed in reaction to particular events and performs specific operations to successfully deploy new or modified resources.

Pull Request Code Scanning Workflow

The first workflow is automatically triggered whenever a user opens or modifies a pull request whose objective is to merge new IaC code in a production environment. Since it is wanted to ensure that this code doesn't introduce any vulnerability or misconfiguration, the triggered pipeline will use Checkov GitHub Action to scan all the files within the pull request and produce a report in SARIF format that will be made available by GitHub itself in a specific dashboard. Even though not mentioned in the GitHub Workflow, SysDig IaC security scanning will also be launched as a consequence of the opening of a new pull request.

The snippet below shows this first GitHub Action Workflow code; the latter is followed by a high-level description of the performed jobs:

Code Scanning Workflow:

```yaml
name: Infrastructure as Code Security checks performed by Checkov
permissions: read-all

on:
  pull_request:
    types: [opened, reopened]

  push:
    branches:
      - '*'
```

```
jobs:
  IaC_Sec_Checkov_Scan:
    permissions: read-all
    name: 'Checkov IaC Sec Scan'
    runs-on: ubuntu-latest
    defaults:
      run:
        shell: bash

    steps:
      - uses: actions/checkout@v3
      - name: Check if pull request is opened/reopened
        id: pr_status
        uses: octokit/request-action@v3.x
        with:
          route: GET /repos/RunCor399/Terraform-IaCSec/pulls/${{
              github.event.pull_request.number }}
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

      - name: Checkov GitHub Action
        uses: bridgecrewio/checkov-action@v12
        if: ${{ steps.pr_status.outputs.data.state == 'open' ||
            steps.pr_status.outputs.data.state == 'reopened' }}
        with:
          output_format: cli,sarif
          output_file_path: console,results.sarif

      - name: Upload SARIF file
        uses: github/codeql-action/upload-sarif@v2

        if: success() ||failure()
        with:
          sarif_file: results.sarif
          ref: ${{ github.head_ref }}
          sha: ${{ github.sha }}
```

High-Level Job Description:

1. The first lines under the **on** tag specify that this workflow will only be triggered when a pull request is opened, re-opened, or updated.
2. **Check if pull request is opened/reopened**: As it is written this workflow triggers at every push in any branch, however, this is not the ideal behavior since the entire pipeline would be executed every time some code is modified. To make sure that security checks are only performed when a Pull Request is created or updated, the first step of **IaC_Sec_Checkov_Scan** job ensures that there is an open pull request associated with a push.
3. **Checkov Execution**: If the previously described conditions are met, this second step will launch Checkov code scanning by using the official *bridgecrewio/checkov-action* Action, and specifies that an output report in SARIF format has to be produced.
4. **Upload SARIF file Job**: The previously generated security report is now made available to GitHub so that it can be visualized within GitHub Repository's Security tab.

Azure Deployment Workflow

Once it is made sure that the newly introduced code is compliant with the required standards, a pull request may be merged in a production environment by an administrator. When this happens, the second workflow that I've realized will once again scan the code for security issues, and assuming everything is compliant, will then proceed to deploy the new or updated infrastructure on Azure.

For brevity purposes, I have omitted from the following workflow the Checkov code scanning job as its definition mirrors that of the previous workflow.

Deployment Workflow:

```
name: Terraform Azure Deploy
permissions: read-all

on:
  pull_request:
    types:
      - closed
```

```
env:
  TF_VAR_azure_sp_key: ${{ secrets.AZURE_SP_SECRET }}
  TF_VAR_storage_account_key: ${{
     secrets.TERRAFORM_STATE_STORAGE_ACCOUNT_KEY }}
  main_ref: "/refs/heads/main"

jobs:
  is_executed:
    name: 'Workflow execution controller'
    runs-on: ubuntu-latest
    outputs:
      output1: ${{ steps.step1.outputs.SHALL_EXECUTE }}
    defaults:
      run:
        shell: bash
    steps:
   - name: Shall execute set
      id: step1
      run: echo "SHALL_EXECUTE=true" >>$GITHUB_OUTPUT

  if_merged:
    if: github.event.pull_request.merged == true
    runs-on: ubuntu-latest
    needs: is_executed
    steps:
   - run: |
        echo The PR was merged

  IaC_Sec_Scan:
    // Omitted Job

  terraform-azure-deploy:
    name: 'Terraform Azure Deploy'
    runs-on: ubuntu-latest
    needs: IaC_Sec_Scan
    defaults:
      run:
        shell: bash
    steps:
   - name: Checkout
      uses: actions/checkout@v3
```

```
- name: Terraform
  uses: hashicorp/setup-terraform@v2

- name: 'Terraform Init'
  id: init
  run: terraform -chdir=./src init
     -backend-config="access_key=${{
     secrets.TERRAFORM_STATE_STORAGE_ACCOUNT_KEY }}"

- name: 'Terraform Validate'
  id: validate
  run: terraform -chdir=./src validate -no-color

- name: 'Terraform Plan'
  id: plan
  run: terraform -chdir=./src plan -no-color
  continue-on-error: true

- name: Terraform Apply
  id: apply
  run: terraform -chdir=./src apply -auto-approve
```

High-Level Job Description:

1. The first lines define once again the execution conditions: this workflow will only be triggered when a pull request is closed. Furthermore, one of the subsequent jobs (*if_merged*) ensures that infrastructure deployment is only carried out when a PR is merged.

2. **Terraform-Azure-Deploy Job**: Leveraging Terraform GitHub Action I was able to define a job capable of automatically deploying IaC-defined resources directly on my Azure Cloud account. The steps to achieve this include:

   (a) **Initializing Terraform Backend** (*Terraform Init*)

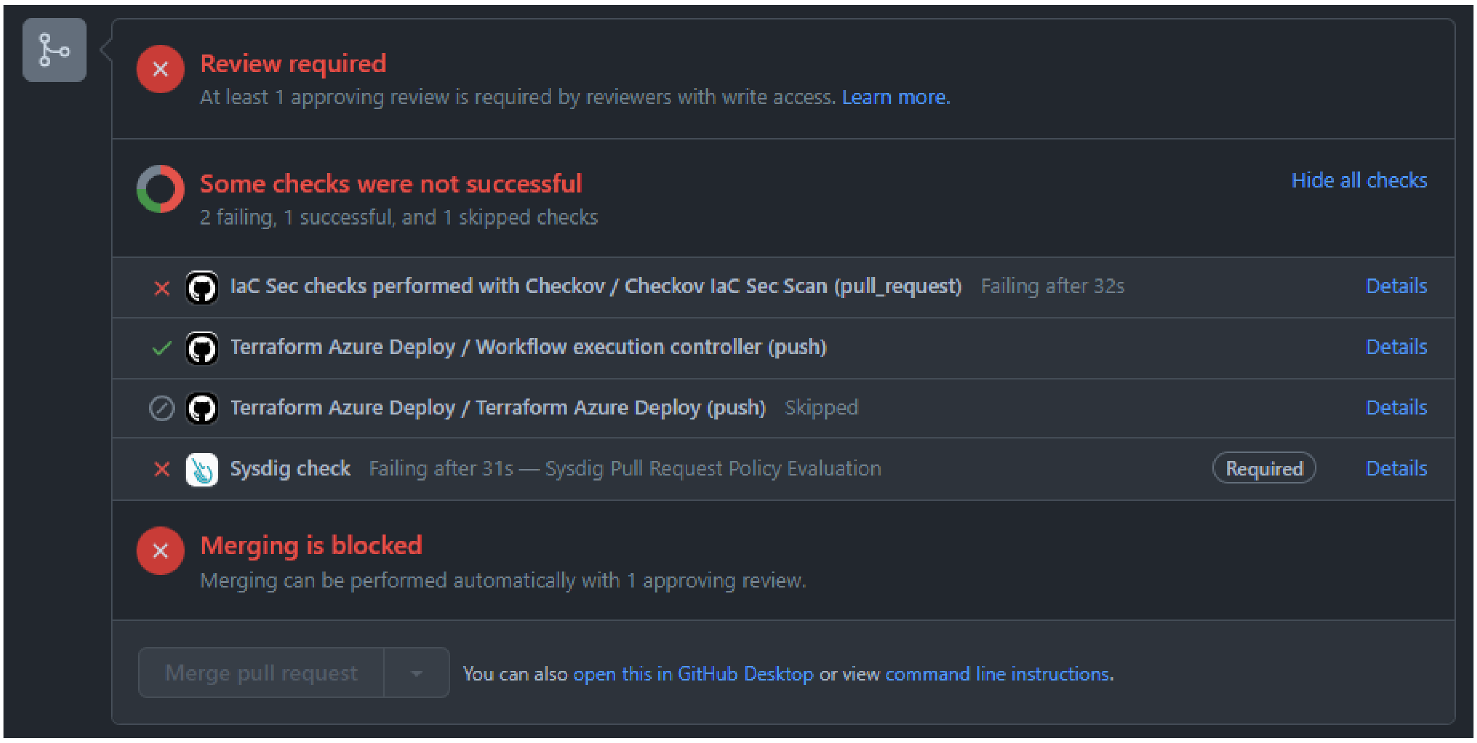
   (b) **Validating Terraform Syntax** (*Terraform Validate*)

   (c) **Planning Resources Deployment** (*Terraform Plan*)

   (d) **Applying the Plan** (*Terraform Apply*)

Assessment Results

The execution results of the first DevSecOps pipeline can be conveniently viewed from within the Pull Request that triggered the workflow. The following image shows a run of the pipeline where both Checkov and Sysdig detected security issues that prevent an administrator from successfully merging the code on the main branch (i.e. the production environment).



Review required
At least 1 approving review is required by reviewers with write access. Learn more.

Some checks were not successful
2 failing, 1 successful, and 1 skipped checks

Hide all checks

✗ IaC Sec checks performed with Checkov / Checkov IaC Sec Scan (pull_request) Failing after 32s Details
✓ Terraform Azure Deploy / Workflow execution controller (push) Details
⊘ Terraform Azure Deploy / Terraform Azure Deploy (push) Skipped Details
✗ Sysdig check Failing after 31s — Sysdig Pull Request Policy Evaluation Required Details

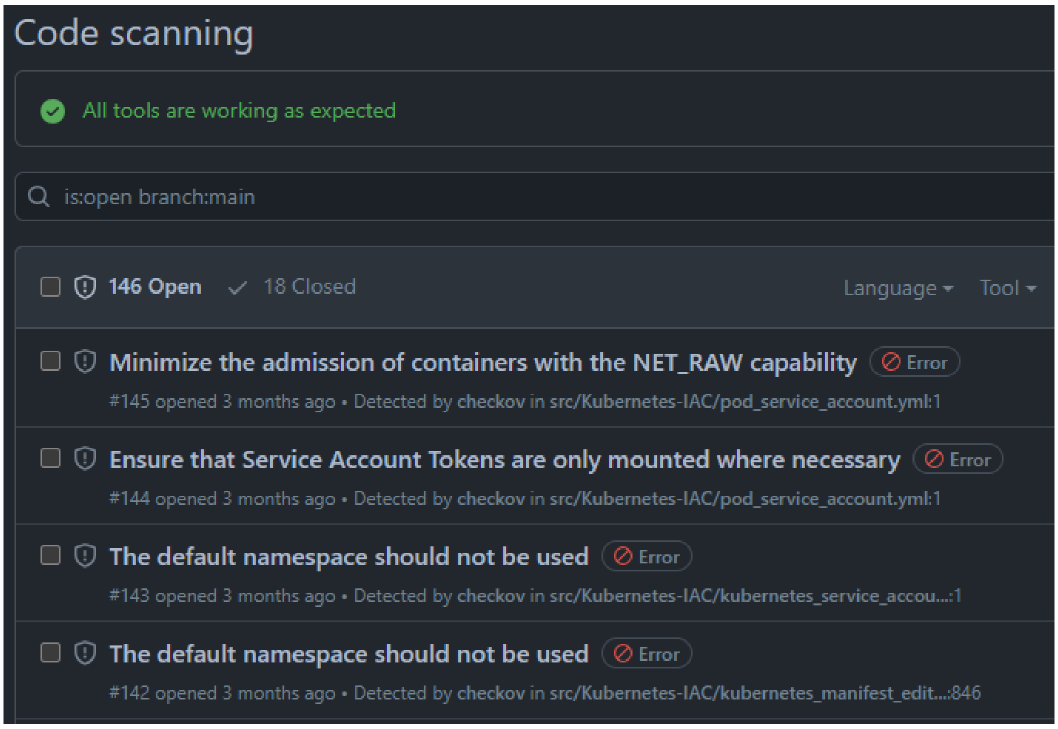
Merging is blocked
Merging can be performed automatically with 1 approving review.

You can also open this in GitHub Desktop or view command line instructions.

Figure 4.1: Pull Request Merging Blocked

At this point, both Checkov and Sysdig have already produced an assessment report that documents the detected security issues, the latter can be visualized in GitHub *"Security Code Scanning"* page, the former by clicking *"Details"* link beside Sysdig code scanning check.

The next two images provide just a summary of Checkov and Sysdig detections; nevertheless, a thorough analysis regarding the most severe issues that were detected has also been performed in Chapter 6



Code scanning

All tools are working as expected

is:open branch:main

146 Open ✓ 18 Closed Language Tool

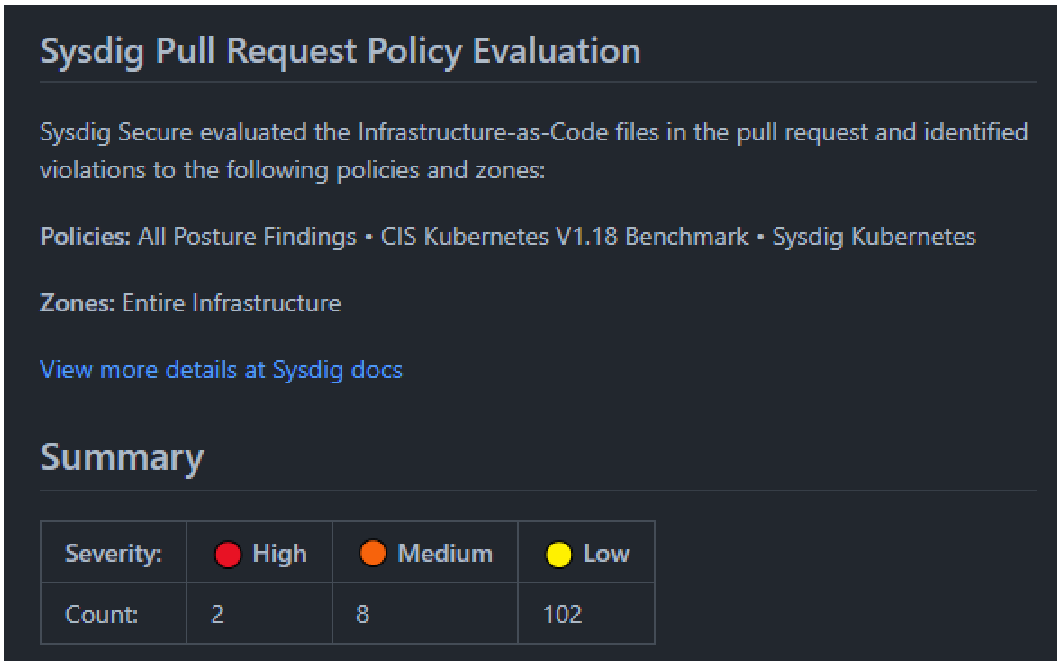
Minimize the admission of containers with the NET_RAW capability Error
#145 opened 3 months ago • Detected by checkov in src/Kubernetes-IAC/pod_service_account.yml:1

Ensure that Service Account Tokens are only mounted where necessary Error
#144 opened 3 months ago • Detected by checkov in src/Kubernetes-IAC/pod_service_account.yml:1

The default namespace should not be used Error
#143 opened 3 months ago • Detected by checkov in src/Kubernetes-IAC/kubernetes_service_accou...:1

The default namespace should not be used Error
#142 opened 3 months ago • Detected by checkov in src/Kubernetes-IAC/kubernetes_manifest_edit...:846

Figure 4.2: Checkov Detections



Sysdig Pull Request Policy Evaluation

Sysdig Secure evaluated the Infrastructure-as-Code files in the pull request and identified violations to the following policies and zones:

Policies: All Posture Findings • CIS Kubernetes V1.18 Benchmark • Sysdig Kubernetes

Zones: Entire Infrastructure

View more details at Sysdig docs

Summary

Severity:	🔴 High	🟠 Medium	🟡 Low
Count:	2	8	102

Figure 4.3: Sysdig Detections

4.3 Cloud Service Providers

4.3.1 Microsoft Azure

Launched in 2010, Azure is Microsoft's public cloud computing platform and as one of the most used public cloud providers (As of 2021, 56% of enterprises worldwide use Azure Cloud), it offers more than 200 services and products that organizations can leverage to build their applications. In the context of this work, the Azure Platform was relied upon to host and operate the majority of resources composing the Multi-Cloud infrastructure I designed.

Among these resources noteworthy are Identity and Access Management (IAM), Secret Management, and Observability services. These latter play a key role in centralizing Multi-Cloud authentication, authorization, and monitoring capabilities.

4.3.2 Amazon Web Services

Amazon Web Services or AWS is another public cloud computing platform that, similarly to Microsoft Azure, empowers organizations with a wide variety of cloud services. AWS can be considered the first hyper-scale public cloud provider in the world, Since launching in 2006 its revenue has never stopped growing, surpassing 80 billion U.S. dollars in 2022 [20].

To be able to set up a Multi-Cloud infrastructure, Reply granted me access to an AWS account which I utilized to deploy some cloud resources that would ideally represent the cloud-offered services of an enterprise. However, differently from the infrastructure I've designed in Azure Platform, the resources that will be deployed on AWS were architectured by a company colleague during his thesis work [3].

4.4 Setup of the Infrastructure

After explaining the IaC Security implementation scenario and introducing the two major public cloud providers worldwide, I will now present all the cloud resources that have been employed along with an explanation of their roles and how they have been integrated.

4.4.1 Terraform

Automation of infrastructure deployment is essential not only to obtain operations repeatability but also to implement an effective CI/CD pipeline capable of reducing the overall software development lifecycle duration. HashiCorp Terraform in this scenario is an Infrastructure-as-Code tool that allows for the definition of cloud resources using human-readable configuration files. Terraform workflow can be partitioned into three main phases:

1. **Write**: Definition of resources for one or more public/private clouds inside Terraform files.
2. **Plan**: Terraform software ingests *.tf* configuration files and produces an execution plan describing which resources it will create, update, or destroy based on the current infrastructure state.
3. **Apply**: As soon as the previously generated plan is approved, Terraform starts performing the proposed operations updating both its local state and the remote state of the infrastructure.

Terraform was chosen over other tools like AWS CloudFormation or Azure ARM templates because it offers a cloud service provider-agnostic solution for deploying resources across major cloud platforms.

To determine which resources need to be created, updated, or removed, Terraform by default uses a local state file that represents the current state of the remote infrastructure. When a plan is applied, Terraform updates the local state file and synchronizes the remote infrastructure state to match the file's content.

Terraform and its features were extensively used in the context of this work to define every single resource that will be deployed on both public cloud platforms. However to track state changes, instead of using a local state file, I have decided to exploit another Terraform feature named **Remote Backend**: the latter allows to store the current state of an infrastructure remotely in a file storage (Azure Storage Accounts, AWS S3 buckets, etc.). By adopting such an approach, operations teams get a consistent view of the remote cloud infrastructure that can now be modified asynchronously.

The following code represents an example of a Terraform file used to set up a remote backend and to create an Azure resource group.

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "3.52.0"
    }
  }

  backend "azurerm" {
    resource_group_name = "terraform"
    storage_account_name = "terraformstate1603709092"
    container_name = "tfstate"
    key = "terraform.tfstate_aks"
  }
}

provider "azurerm" {
  subscription_id = var.credentials["subscription_id"]
  client_id = var.credentials["client_id"]
  client_secret = var.azure_sp_key
  tenant_id = var.credentials["tenant_id"]
  features {}
}

resource "azurerm_resource_group" "aks_resource_group" {
  name = var.aks_resource_group.name
  location = var.aks_resource_group.location
}
```

4.4.2 Workloads: Azure

Azure Kubernetes Service

Azure Kubernetes Service (AKS) is a Managed Kubernetes service provided by Microsoft Azure that allows one to effortlessly deploy and manage Kubernetes Clusters. Thanks to its capabilities which range from workload deployment and service discovery to load balancing and storage orchestration, Kubernetes is nowadays the most used orchestration system in the world.

After having deployed an AKS instance on my Azure Subscription relying on Terraform provisioning capabilities, I decided to host in this Kubernetes Cluster a full-fledged microservices-based application provided by Google[9]. This application, composed of several microservices each serving a specific purpose, showcases an example of a service provided by an organization through a public cloud.

Apart from the suite of microservices of which *Online Boutique*[9] is composed, I've also developed an additional Spring service able to interact with the following cluster-external resources:

- **Hashicorp Vault**: As explained in 4.4.4, Vaults enable to secure, store, and control access to passwords, encryption keys, tokens, and certificates. By being able to interact with a Vault, the Spring microservice can query or persist secrets of any kind without the need for embedding sensitive data within the application code.
- **Azure MySQL Database**: By making use of temporary keys supplied by a Vault, this Spring application will also be able to interact with an Azure MySQL instance. A more in-depth explanation of the integration among the Spring microservice, HashiCorp Vault, and Azure MySQL Database, has been provided in Section 4.4.4, Paragraph *"Integration among Vaults and Workloads"*.

These next two extracts of code respectively show a part of the Terraform script that was used to deploy the Kubernetes Cluster on Azure and the commands allowing the provisioning of the services within the cluster itself.

Terraform AKS Code:

```
resource "azurerm_kubernetes_cluster" "aks_cluster" {
  name = "aks-cluster"
  location = azurerm_resource_group.aks_resource_group.location
  resource_group_name =
      azurerm_resource_group.aks_resource_group.name
  dns_prefix = "aks-cluster"

  default_node_pool {
    name = "default"
    node_count = 2
    vm_size = "standard_d2_v3"
    vnet_subnet_id = azurerm_subnet.aks_node_subnet.id
  }

  identity {
    type = "SystemAssigned"
  }

  azure_policy_enabled = false
  http_application_routing_enabled = false
}
```

Microservices Provisioning Script:

```
#!/bin/bash
az login

# Automatic configuration of local kubectl
az aks get—credentials —g aks—resource—group —n aks—cluster


# Deploy Service Account and Test Pod
kubectl apply —f ./kubernetes_service_account.yml
kubectl apply —f ./pod_service_account.yml

# Deploy Online Boutique
kubectl apply —f ./kubernetes_manifest_edited.yml
```

```
# Deploy Spring DB Connector microservice
kubectl apply -f ./db_connector_manifest.yml

# Deploy Kubernetes Dashboard
kubectl apply -f
    https://raw.githubusercontent.com/kubernetes/dashboard/
   v2.7.0/aio/deploy/recommended.yaml

echo "Dashboard URL:
    http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/
   services/https:kubernetes-dashboard:/proxy/"


# Proxy access to kubernetes dashboard
kubectl proxy
```

Azure MySQL Database

Among the many database solutions offered by Azure, I've opted to deploy on my infrastructure a simple MySQL Database that will be queried to retrieve generic-purpose data by the Spring application discussed above, as well as by Hashicorp Vault to obtain temporary access credentials that may be used to authenticate other microservices.

4.4.3 Workloads: AWS

Amazon Elastic Kubernetes Service

Amazon Elastic Kubernetes Service or EKS is the equivalent of Azure Kubernetes Service but on the Amazon Web Services Platform. As such it still offers a Managed Kubernetes instance that can be leveraged by businesses to build, secure, operate, and maintain Kubernetes Clusters hosted on the cloud.

To successfully deploy a Managed Kubernetes Cluster, I had first to set up a new Virtual Private Cloud (VPC). AWS VPCs can be seen as virtual networks that allow to: define networking rules, security groups, route tables, and create public and private subnets. Although in the context of this work, no workloads are running inside the EKS Cluster, I deemed it essential to establish an environment that reflected as much as possible an enterprise cloud infrastructure.

4.4.4 Multi-Cloud Secret Management

Secret Management is an essential practice adopted in large environments that allows centralizing the administration of sensitive data required by workloads for authentication, authorization, and many other tasks. Implementing a practical secret management framework, besides giving the possibility to protect data at rest, also reduces the risks of data leaks and unauthorized access to services.

In my Multi-Cloud infrastructure use case I've chosen to deploy a Vault accessible not only by the workloads operating within the same cloud environment (i.e. Azure) but also by resources that reside on other public or private clouds (AWS in this case), thus completely centralizing the management of secrets belonging to multiple cloud environments. HashiCorp, in its blog [11], discusses the significance of centralizing secrets management and explains how it can be achieved using HCP Vault and its extensive range of integrations.

In the subsequent sections, an overview of the resources utilized to implement this use case is presented, along with a thorough description of how Vault's capabilities are leveraged by workloads.

HashiCorp Vault

HashiCorp Vault is an identity-based secrets management system that not only securely stores passwords, encryption keys, tokens, and certificates, but also controls access to these secrets by authenticating entities seeking to use them. A formal description of Vault Core workflow, as defined by HashiCorp developers, follows:

1. **Authenticate**: During this phase clients supply information that allows Vault if they are who they claim to be. If authentication is successful Vault generates a token associated with a specific security policy.

2. **Validate**: Vault relies on third-party trusted sources such as Kubernetes, LDAP, or GitHub to validate clients' legitimacy.

3. **Authorize**: Policies defined in a Vault instance determine the API endpoints a client has access to given its identity. Such a policy mechanism implemented by Vault provides a declarative way to permit or deny access to certain operations or data selectively.

4. **Access**: The token issued after client authentication is the piece of data that grants access to secrets, keys, and encryption capabilities. A best practice to reduce the risk of data leaks is to limit the lifetime of this token; when it expires a client may once again perform authentication to receive a fresh one.

In the Multi-Cloud infrastructure proposed in this paper, a Vault has been provisioned to serve requests coming from workloads either running in Azure or AWS provider. The reason for choosing this Vault solution from HashiCorp lies in its capability to integrate with various platforms such as Kubernetes, Azure, and generic databases. At the end of 2021, HCP Vault had surpassed 100 integrations with 75 partners, confirming its leadership in identity-based security solutions and centralized secrets management [12].

To host the just mentioned Vault I've chosen to employ an Azure Virtual Machine that could be reachable from outside Azure provider using a public IP address. The entire deployment and Vault configuration process was automated with ad-hoc Terraform and template files: using these two resources together allows one to set up a working Vault in a matter of minutes. Apart from the virtual machine, several other resources were deployed on my Azure Subscription to take advantage of an additional useful Vault functionality named **Auto-Unseal**, discussed in detail in section 4.4.4.

Note that while the use of a public IP address to access Vault is not the most secure option, it was chosen for the sake of simplicity and expediency. A more secure approach, such as access through a VPN, is recommended for production environments.

Terraform Script used to deploy Vault Azure VM

```
resource "azurerm_linux_virtual_machine" "tf_vm" {
  name = var.vm_name
  location = var.location
  resource_group_name = azurerm_resource_group.vault.name
  network_interface_ids = [azurerm_network_interface.tf_nic.id]
  size = "standard_a2_v2"
  custom_data = base64encode(data.template_file.setup.rendered)
  computer_name = var.vm_name
  admin_username = "azureuser"

  admin_ssh_key {
    username = "azureuser"
    public_key = var.public_key
  }

  os_disk {
    name = "${var.vm_name}-os"
    caching = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }
  source_image_reference {
    publisher = "Canonical"
    offer = "UbuntuServer"
    sku = "18.04-LTS"
    version = "latest"
  }

  boot_diagnostics {
    storage_account_uri =
        azurerm_storage_account.tf_storageaccount.primary_blob_endpoint
  }
}
```

Azure Key Vault

Azure Key Vault is one of Microsoft's proprietary Vault solutions, that helps solve problems related to secrets management, key management, and certificate management. The reason why I have opted to include an Azure Key Vault instance in my infrastructure is that it plays a primary role in supporting HashiCorp's Vault Auto-Unseal functionality explained in the next section.

Vault Auto-Unseal

Every time a Vault server is booted, it starts in a *sealed* state and can't decrypt the data it contains. To acquire the ability to perform any operation, the Vault needs first to be unsealed. Unseal operation requires the input of a certain number of keys to be successful; these special keys are generated by Shamir's Secret Sharing algorithm only the first time the Vault boots. When the Vault is supplied with all the required keys, it is then able to use the same Shamir's cryptographic algorithm to build the master key from which other keys originate, and decrypt secured data.

The previously introduced Azure Key Vault makes the auto-unseal process possible by internally storing each of the pieces of the master key that are to be used for the unsealing. By doing so, every time the HashiCorp Vault boots it will demand Azure Key Vault the necessary keys to unseal itself, thus effectively eliminating the need for human intervention. Further information about this technique can be retrieved directly from HashiCorp documentation [10]

A schema showing the components and the workflow of the auto-unseal feature is depicted in figure 4.4. Additionally, an explanation of the role of each component and the auto-unseal procedure is available under the mentioned schema.

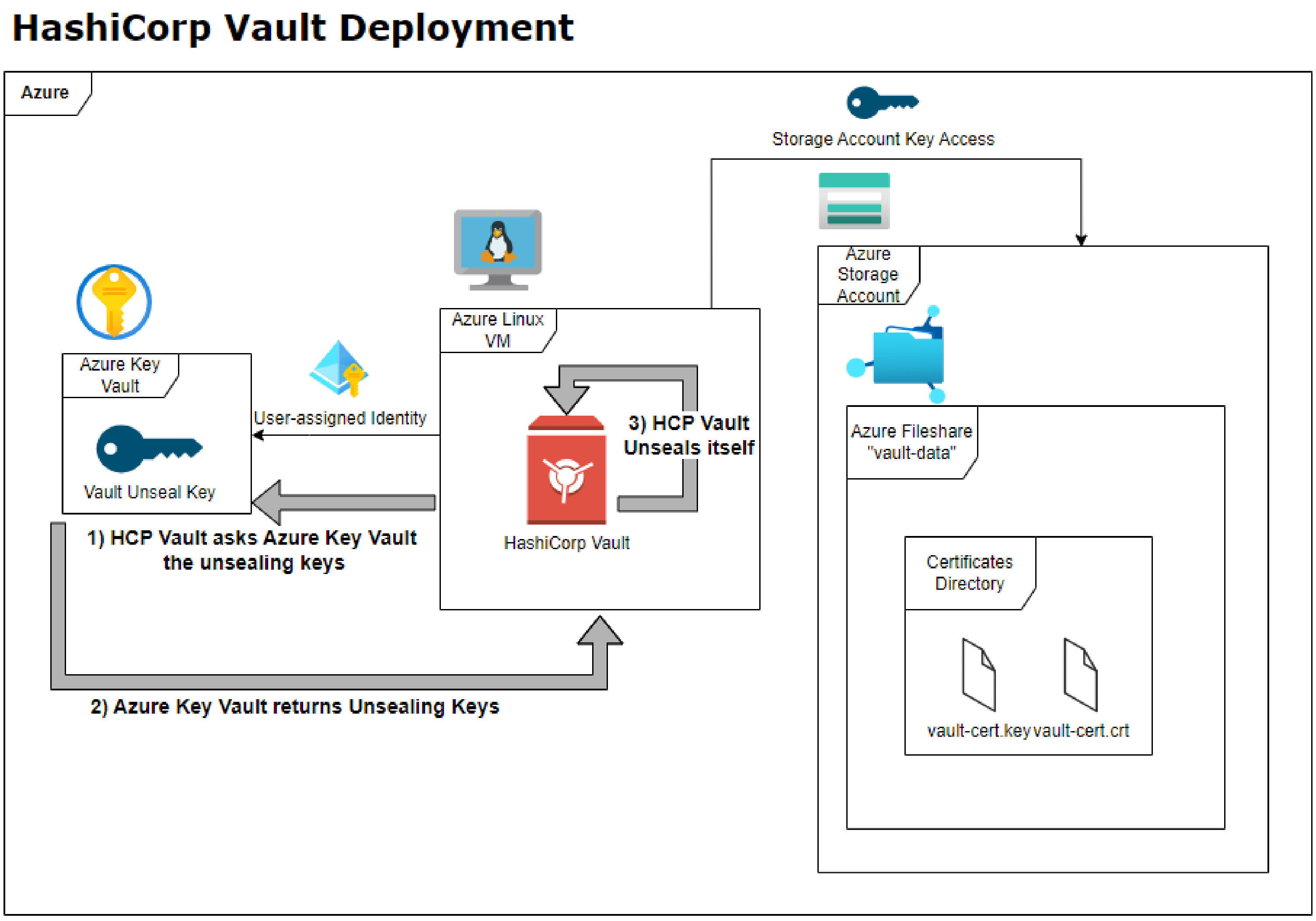

HashiCorp Vault Deployment
Azure
Storage Account Key Access
Azure Storage Account
Azure Linux VM
Azure Key Vault
User-assigned Identity
Vault Unseal Key
3) HCP Vault Unseals itself
HashiCorp Vault
1) HCP Vault asks Azure Key Vault the unsealing keys
2) Azure Key Vault returns Unsealing Keys
Azure Fileshare "vault-data"
Certificates Directory
vault-cert.key
vault-cert.crt

Figure 4.4: HashiCorp Vault Auto-Unseal

Components:

- **Azure Linux Virtual Machine**: Computing resource hosted on Azure Platform that hosts HashiCorp Vault.
- **HashiCorp Vault**: Vault containing passwords, access keys, tokens, and certificates
- **Azure Key Vault**: Vault that supports HCP Vault auto-unseal functionality by storing unsealing keys.

- **Azure User-assigned identity**: Azure Identity that allows HashiCorp Vault to authenticate with Azure Key Vault and to request unsealing keys.
- **Azure Storage Account**: Storage account containing a file share that has been attached to HCP Vault Virtual Machine.
- **Storage Account Access Key**: Secret key that allows a storage account and its file shares to be attached to an Azure Virtual Machine.
- **Vault Certificates**: PKI certificate and associated private key used to establish a TLS channel supporting server authentication between workloads (clients) and HCP Vault (server).

Auto-Unseal Workflow: (First Vault Start-up)

1. **HCP Vault** generates 5 unsealing keys starting from a single master key
2. **HCP Vault** saves each of the unsealing keys within Azure Key Vault
3. **Azure Key Vault** is now ready to offer unsealing keys when needed by HCP Vault

Auto-Unseal Workflow: (Subsequent Vault Start-ups)

1. **HCP Vault** starts up in a sealed state and requests Azure Key Vault the necessary unsealing keys.
2. **Azure Key Vault** authenticates the VM hosting the Vault relying on the *Azure User-Assigned Identity*; if authentication succeeds, it proceeds by returning the necessary unsealing keys.
3. **HCP Vault** uses the unsealing keys to reconstruct the master key and with the latter decrypts all data stored within the Vault.

Integration among Vaults and Workloads

The two integration scenarios I've implemented have as protagonists a HashiCorp Vault, a microservice hosted in a Kubernetes cluster, and an instance of an Azure MySQL Database. The first scenario explains how a generic Kubernetes workload acquires the ability to authenticate and obtain secrets secured inside an HCP Vault.

The second scenario on the other hand shows how that same workload is also able to obtain from the HCP Vault a pair of temporary database credentials that can be used to operate on an Azure MySQL Database instance. Even though the presented use cases only consider Azure Kubernetes workloads, there wouldn't have been problems nor differences in implementing the same two scenarios but using Kubernetes workloads running in an AWS Cluster.

The objective of these two integrations is to establish a trusted relationship between Kubernetes workloads, a Vault, and a MySQL database, based on a secret-less interaction between the parties. The list below includes a brief explanation of the role of each resource that allowed me to successfully implement these two scenarios.

- **Azure Kubernetes Cluster**: Cluster Kubernetes running the workload that interfaces with HCP Vault. An additional duty delegated to this cluster is to validate JWT tokens received by HCP Vault.
- **Spring Microservice**: Application that uses **Spring Cloud Vault** library to authenticate to HCP Vault and to request secrets or MySQL database credentials.
- **Pod Service Account**: Service Accounts are a resource in Kubernetes environments that allow the assignment of an identity to Pods, together with a collection of permissions. The Spring microservice discussed above has been conveniently equipped with a Service Account that allows it to authenticate to HashiCorp Vault.
- **HashiCorp Vault**: Vault instance deployed on Azure configured to allow Kubernetes authentication by pods equipped with a specific service account, and to interact with the Azure MySQL Database instance for the generation of temporary access keys.
- **Azure MySQL Database**: This MySQL instance deployed on Azure Platform, apart from providing the classic functionalities of a relational database, it has also been configured to allow HashiCorp Vault to generate temporary database access credentials that can be provisioned to workloads.

Vault Secrets Retrieval Scenario
This first scenario introduced above and schematized in figure 4.5 whose implementation gives Kubernetes workloads the ability to authenticate and request secrets stored in a Vault, is now thoroughly analyzed to provide additional information about the logic behind it.

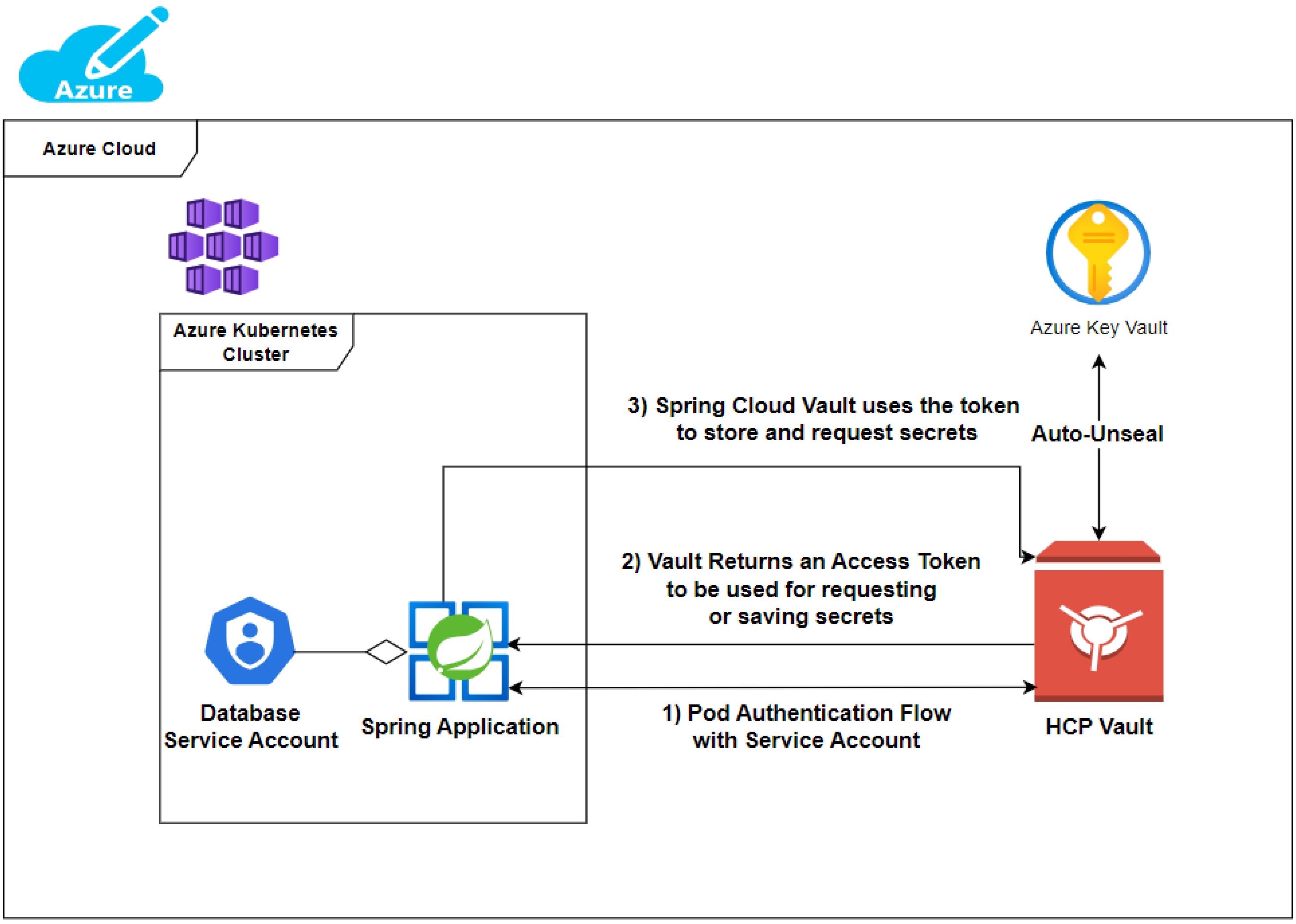

Azure
Azure Cloud
Azure Kubernetes Cluster
Azure Key Vault
3) Spring Cloud Vault uses the token to store and request secrets
Auto-Unseal
2) Vault Returns an Access Token to be used for requesting or saving secrets
Database Service Account
Spring Application
1) Pod Authentication Flow with Service Account
HCP Vault

Figure 4.5: HashiCorp Vault Secrets Retrieval

Secret Retrieval Workflow

1. **Pod Authentication Flow with Service Account**: During this initial phase a Pod willing to interact with the Vault is first required to prove its identity. The latter authentication flow can be summarized by the following steps:

   (a) **Pod => Vault**: The Pod contacts HCP Vault and sends it the JWT token associated with its Service Account.

   (b) **Vault => Kubernetes API Server**: The Vault forwards the JWT token to Kubernetes API Server to verify its validity

(c) **Kubernetes API Server => Vault**: Kubernetes API Server validates the JWT token's claims and signature, and returns a response to the Vault.

(d) **Vault => Pod**: If validation is successful the Vault generates and returns the Pod a token associated with a well-defined set of permissions. The granted permissions depend on the role defined within the service account and on Vault security policies.

2. **Access Vault Secrets**: The previously received session token can now be used by the Pod to retrieve secrets from the Vault. HashiCorp Vault uses a role-based authorization schema to permit or deny access to specific secrets.

Vault Database Access Scenario

The second scenario schematized in figure 4.6 preserves the same *"Pod to Vault"* authentication flow explained before, but it additionally implements a useful functionality for workloads that are willing to interact with an external database. Normally applications would just internally store database login details inside environment variables or configuration files; however, such an approach greatly endangers the confidentiality of database access credentials. An alternative technique that allows mitigating the risk of database credentials disclosure relies on HashiCorp Vault.
By configuring the HashiCorp Vault instance appropriately, which includes granting it permission to create temporary MySQL database users, Kubernetes microservices now have the possibility to request temporary database credentials directly from the Vault after authenticating.

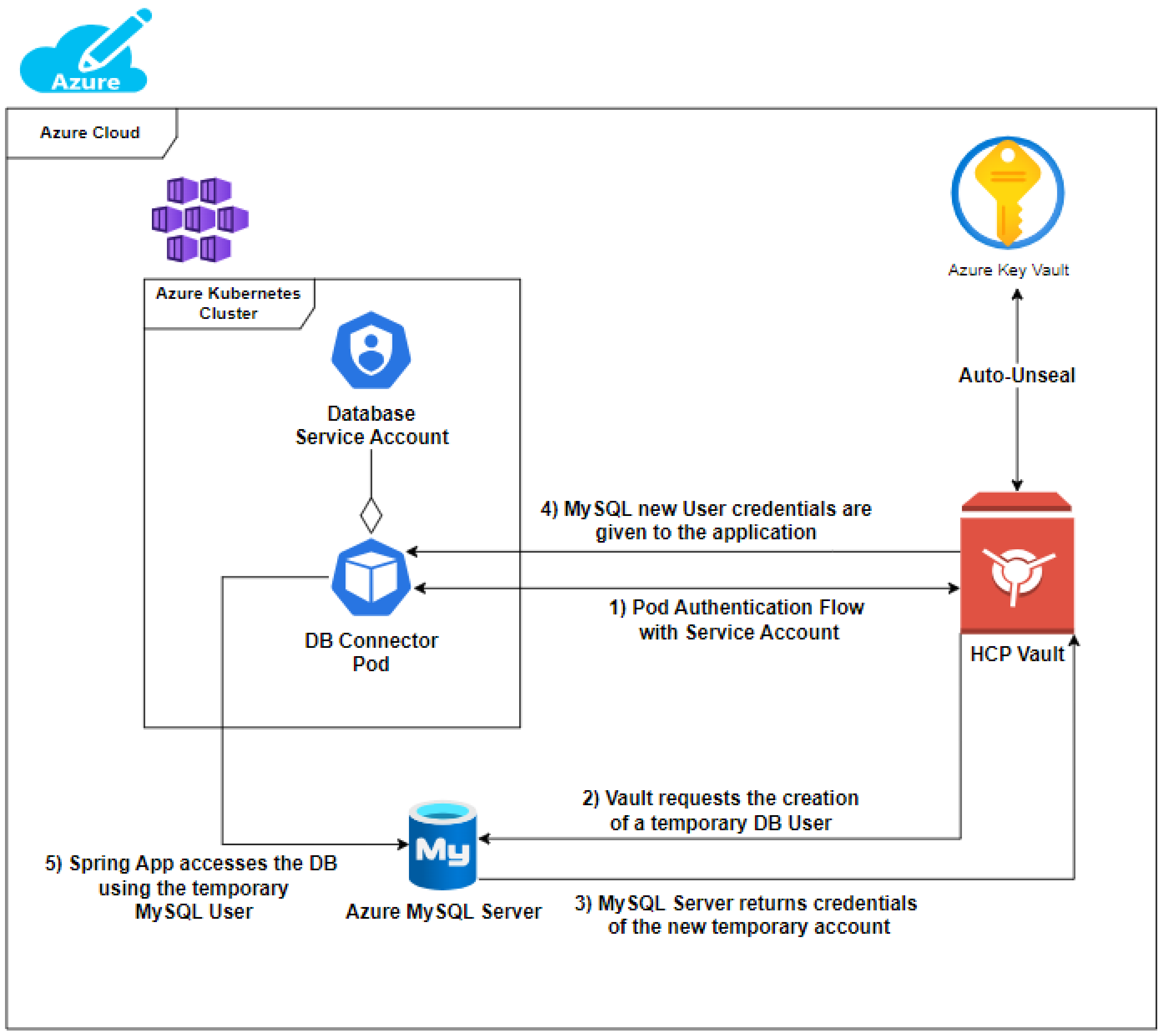

Azure
Azure Cloud
Azure Kubernetes Cluster
Database Service Account
DB Connector Pod
Azure Key Vault
Auto-Unseal
HCP Vault
4) MySQL new User credentials are given to the application
1) Pod Authentication Flow with Service Account
2) Vault requests the creation of a temporary DB User
3) MySQL Server returns credentials of the new temporary account
5) Spring App accesses the DB using the temporary MySQL User
Azure MySQL Server

Figure 4.6: HashiCorp Vault Database Access

Similarly to the previous *"Secret Retrieval"* use case, the following workflow highlights the actors and their interactions that enable the concrete implementation of this newly introduced scenario.

1. **Pod Authentication Flow with Service Account**: The same authentication flow discussed in the first scenario is yet again implemented and exploited

by microservices willing to prove their identity.

2. **Pod => HCP Vault**: Once authentication to the Vault has been performed, the Spring microservice uses Spring Cloud Vault to request the creation of temporary database credentials.

3. **HCP Vault => MySQL Server**: HCP Vault has the necessary authorization level to ask the MySQL server for the issuance of temporary database credentials.

4. **MySQL Server => HCP Vault**: The fresh database credentials are returned to the Vault.

5. **HCP Vault => Pod**: The pod receives the temporary database credentials it asked for.

6. **Pod => MySQL Server**: The received credentials can now be used by the Pod to perform queries on the Database. The database permissions a Pod receives depend once again on the role defined within the Service Account and on specific policy configurations that have been applied on the Vault. In my specific use case, the Pod was granted the *db_access_role* role, which allowed it to execute *CREATE*, *SELECT*, *INSERT*, *DELETE*, *DROP* statements on whichever database and table of the MySQL server. In a production implementation, a best practice would be to limit query execution capabilities according to workloads' roles.

4.4.5 Multi-Cloud Identity and Access Management

Organizations relying on multiple cloud platforms nowadays are facing enormous challenges when it comes to managing users' identities. The main problem is that every single public cloud provider offers its own Identity and Access Management solution: Azure Active Directory by Microsoft, IAM Identity Center by Amazon, and Google Cloud Identity by Google. Not adopting a Multi-Cloud Identity and Access Management strategy would lead to an uncontrolled replication of the same identities across different providers, thereby introducing increased management complexity and serious risks to accounts' security. Furthermore, implementing a Centralized Multi-Cloud approach to Identity and Access Management entails numerous benefits, including:

1. **Users are associated with a single identity shared among cloud providers**

2. **Organizations' security enhanced** thanks to the enforcement of Access Management policies across all systems.

3. **Identities can be managed from a single dashboard**

4. **Ability to compel users adherence to secure password policies**

The Identity and Access Management solution I've implemented in my Multi-Cloud infrastructure uses the Keycloak IAM tool as the only Identity Provider. After a brief explanation of how Keycloak was deployed in cloud and an overview of the established IAM federation among AWS, Azure, and Keycloak; a realistic *Business-to-Employees* scenario in which cloud users can authenticate with Keycloak and automatically gain access to the AWS Management Console is shown.

Keycloak

Keycloak is an open-source Identity and Access Management solution that provides Single Sign-On and Single Sign-Out capabilities and supports protocols such as SAML 2.0, OpenID Connect, and OAuth 2.0. Given the number of supported protocols, Keycloak is the perfect tool for heterogeneous types of applications with different security demands. In addition to the supported protocols, my Reply supervisors suggested the adoption of Keycloak in this thesis to determine its suitability as an Identity Provider in a Multi-Cloud infrastructure.

Using a collection of ad-hoc Terraform and Docker Compose scripts, I have successfully deployed a Keycloak instance on an Azure Web App and exposed it to be reachable both from Azure and other cloud platforms resources. The following Terraform file shows the main Azure resources required for the correct provisioning of this IAM tool:

Keycloak deployment in Azure Web App

```
resource "azurerm_linux_web_app" "keycloak-webapp" {
  name = "keycloak-webapp"
  resource_group_name = var.keycloak-resource-group.name
  location = var.keycloak-resource-group.location
  service_plan_id = azurerm_service_plan.keycloak-sp.id

  https_only = true

  app_settings = {
      "KEYCLOAK_USER" = "admin"
      "KEYCLOAK_PASSWORD" = sensitive(var.secrets.admin_password)
      "DOCKER_REGISTRY_SERVER_URL" =
         "https://registry.hub.docker.com/v2/"
      "WEBSITE_ENABLE_APP_SERVICE_STORAGE" = true
  }

  site_config {
      always_on = false
  }

  depends_on = [ azurerm_service_plan.keycloak-sp ]

}

resource "azapi_update_resource" "update_linux_web_app" {
  resource_id = azurerm_linux_web_app.keycloak-webapp.id
  type = "Microsoft.Web/sites@2022-03-01"
  body = jsonencode({
      properties = {
      "siteConfig" = {
        "linuxFxVersion" =
           "COMPOSE|${base64encode(file("keycloak_service.yaml"))}"
      }
      "appSettings" = {
        "keycloakFrontendUrl" = join("/",
           [azurerm_linux_web_app.keycloak-webapp.default_hostname,
           "auth"])
      }
      }
  })

  depends_on = [ azurerm_linux_web_app.keycloak-webapp ]
}
```

The Docker Compose file below on the other hand contains the Keycloak service definition that Azure takes care of setting up within the Web App:

Keycloak Service

```
version: '3'

services:
  keycloak:
    image: jboss/keycloak:latest
    container_name: keycloak
    volumes:
      - ${WEBAPP_STORAGE_HOME}/data:/opt/jboss/keycloak
          /standalone/data
    restart: always
```

Azure, AWS and Keycloak Federation

Centralization of Identity and Access Management in a Multi-Cloud infrastructure initially requires establishing specific trust relationships between the diverse Cloud Providers and the designated IAM tool. In the context of this thesis work, I've designed a Federation consisting of Azure and AWS platforms that act as Service Providers, and Keycloak being the Identity Provider that first verifies users' identity and then redirects them to the proper Service Provider.

Both trust relationships between Keycloak and Azure/AWS may be set up leveraging SAML 2.0 protocol: an open standard created to provide users the capability of authenticating against a system (**Identity Provider**) and gaining access to other systems deemed trustworthy (**Service Providers**). SAML which stands for *Security Assertions Markup Language*, uses XML digitally signed documents exchanged over HTTP connections to represent users' identities. These XML documents that take the name of **Assertions** are produced by the Identity Provider and consumed by Service Providers to verify that a user has been authenticated.

Even though implementing a complete Multi-Cloud Identity and Access Management strategy would have required establishing both federations cited above, Ultimately I've opted for only implementing the one between Keycloak and AWS provider. The reason for this choice is that despite the robust feature set provided by Azure's IAM tool, Azure Active Directory, as opposed to AWS's IAM Identity

Center, doesn't support a fully external identity provider setup where user information is solely stored in a third-party platform like Keycloak. As a result, an enterprise implementing such a federation would need to provision users both in Azure AD and in Keycloak, thus losing all the benefits of having a unique identity per employee.

In the upcoming list, I have outlined the main configuration operations that were carried out to establish the trust relationship between Keycloak and the AWS provider that would allow users to access the AWS Management Console only using their Keycloak identity.

- **Keycloak IdP Configuration**: Creation of a new Keycloak client that will register AWS as a SAML Service Provider. To achieve this, it is required to set up AWS SAML endpoints that will be used for redirecting users.
- **AWS Configuration**
    1. Set up of a new SAML Identity Provider in AWS Identity and Access Management dashboard.
    2. Creation of custom AWS roles associated with specific permissions sets. Users authenticating against Keycloak will assume one of these roles depending on specific mappings.
- **Users, Groups, and Roles Mappings**: During this last step, custom roles previously created on AWS are now mapped and associated with Keycloak groups. By doing so users belonging to a group are only granted a limited amount of permissions over the entire AWS Cloud Account, thus implementing an effective Role-based Access Control strategy.

Authentication Flow in a B2E Scenario

Having introduced the federation between AWS and Keycloak, it is now possible to discuss a real-life usage scenario within which users of a company use their unique identity to access one or more cloud accounts. The **Business-to-Employees** scenario I have explained and schematized below showcases the situation in which a user is willing to log into an AWS account using its Keycloak identity.

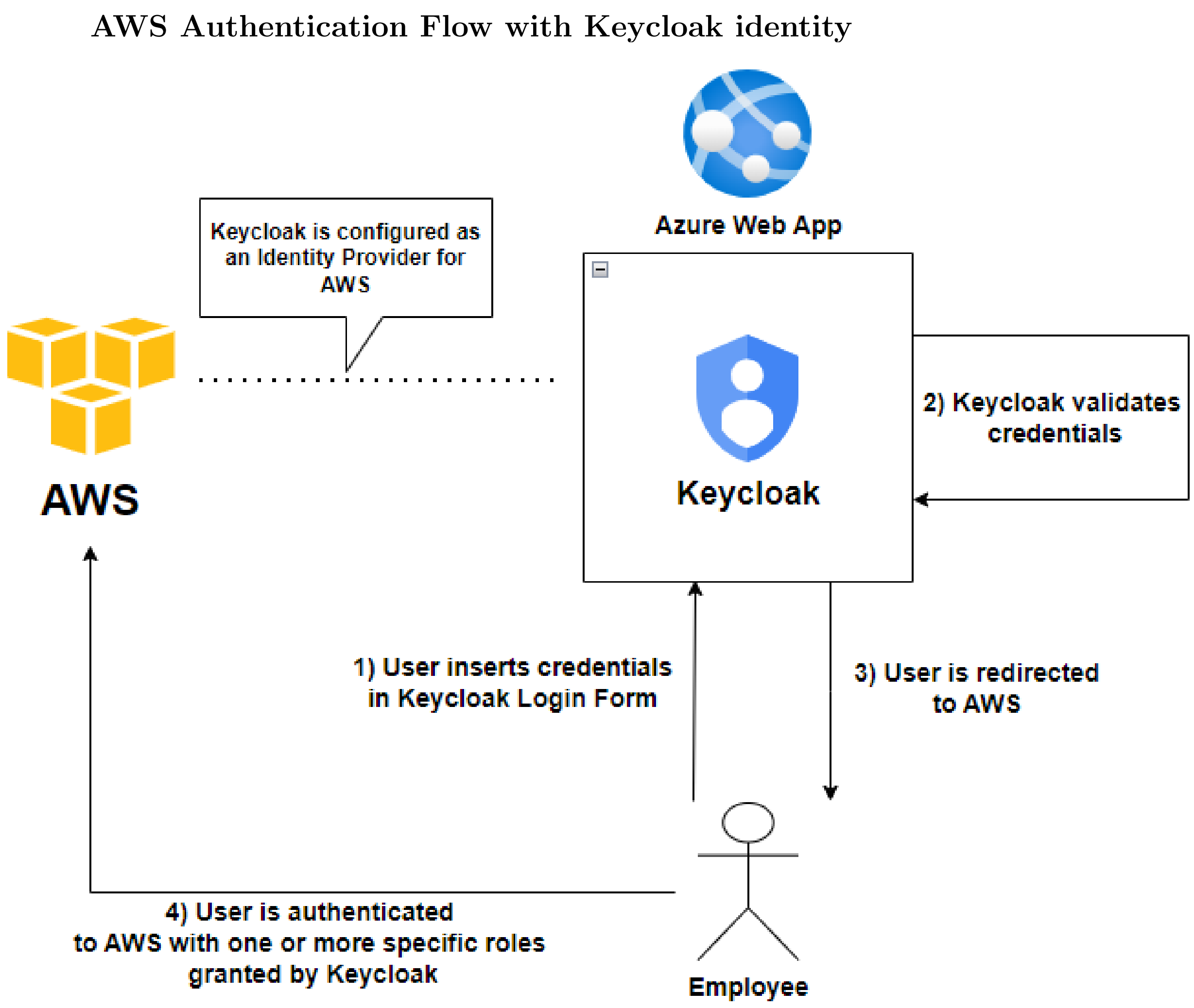

AWS Authentication Flow with Keycloak identity
Azure Web App
Keycloak is configured as an Identity Provider for AWS
AWS
Keycloak
2) Keycloak validates credentials
1) User inserts credentials in Keycloak Login Form
3) User is redirected to AWS
4) User is authenticated to AWS with one or more specific roles granted by Keycloak
Employee

Figure 4.7: AWS and Keycloak Federation

B2E Authentication Workflow

1. An employee who intends to access the AWS Management Console of the organization he is working for is first redirected to Keycloak where he can perform authentication using his unique identity.

2. Keycloak validates the employee's identity and associates a set of permissions according to his role.

3. If Keycloak credentials are correct, the employee is redirected to the AWS Management Console where he results as an authenticated user and can perform only the operations granted by his role.

4.4.6 Multi-Cloud Infrastructure Observability

Observability, which is usually defined as *"the ability to measure a system's current state based on the data it generates"*, is a crucial concept in the context of distributed systems and applications. In the Cloud Computing world, observability also refers to the tools and practices that allow for the collection, correlation, and analysis of streams of data coming from heterogeneous workloads. When shifting to a Multi-Cloud environment where there may be thousands of resources spread across different cloud platforms, implementing an observability strategy becomes essential for providing developers and DevOps teams visibility and insights into applications.

The forthcoming discussion covers the design and implementation steps I performed to successfully address observability in my Multi-Cloud infrastructure composed by Azure and AWS public cloud providers.

Azure Monitor

Azure Monitor is a comprehensive monitoring solution provided by Microsoft that eases the task of collecting and analyzing data coming from both cloud and on-premise environments. Apart from being able to collect data coming from resources like applications, virtual machines, operating systems, databases, and networks, Azure Monitor also offers the functionality of storing data in a common platform so that it can be made available to other aggregation, correlation, analysis, and visualization tools.

Among the many solutions provided by Azure Monitor, I have specifically employed **Log Analytics Workspace** which provides dedicated environments for logging data coming from Azure services. This collects monitoring data from all

Azure resources and forwards it to another Azure-managed service for visualization. In section 4.4.6 a description of the platform adopted for data visualization is provided while section 4.4.6 shows an overall view of the multi-cloud observability infrastructure design.

AWS CloudWatch

CloudWatch is the solution provided by Amazon for monitoring resources and applications running on AWS cloud accounts. In a similar way to Azure Monitor, CloudWatch enables real-time monitoring of AWS workloads like EC2 instances, EBS volumes, RDS instances, and many others.

Within my AWS infrastructure, I've decided to leverage CloudWatch capabilities to collect and forward monitoring data to the same Azure-managed visualization service mentioned above. By adopting such an approach, workload metrics coming from different cloud providers can be easily analyzed from a single data visualization platform, thus achieving the goal of implementing a centralized Multi-Cloud observability strategy.

Azure Grafana Managed Service

Grafana is the solution for data visualization suggested by Reply's tutors due to its features and availability in Azure. Grafana is an open-source platform developed by Grafana Labs that allows users to analyze and monitor data coming from a broad variety of different data sources. Dashboards are the key element of Grafana's platform for giving meaning to collected data: users can use pre-made dashboards or build custom dashboards that adapt to specific needs.

My Multi-Cloud infrastructure scenario includes a Managed Grafana instance deployed on Microsoft's Azure Platform. The latter Grafana instance, which is only accessible by authorized users, aggregates monitoring data coming both from Azure Log Analytics Workspace and AWS CloudWatch, and makes it available with ad-hoc dashboards.

Observability Infrastructure Design

Following the introduction of the main resources allowing the realization of a Multi-Cloud observability strategy, this section first presents an overview of the Terraform scripts specifically developed to set up the previously described observability stack, followed by a diagram representing the infrastructure design of the latter.

Azure Observability Stack Terraform Script

```
# Azure Log Analytics Workspace
resource "azurerm_log_analytics_workspace"
   "log_analytics_workspace" {
    name = var.log−analytics−workspace.name
    location = var.log−analytics−workspace.location
    resource_group_name = var.observability_rg.name
    sku = "PerGB2018"

    depends_on = [azurerm_resource_group.observability_rg]
}

data "azurerm_subscription" "primary" {}

# Grafana Instance
resource "azurerm_dashboard_grafana" "grafana-dashboard" {
  name = var.grafana−dashboard.name
  resource_group_name = var.observability_rg.name
  location = var.grafana−dashboard.location

  auto_generated_domain_name_label_scope = "TenantReuse"
  public_network_access_enabled = true
  api_key_enabled = false
  deterministic_outbound_ip_enabled = false
  zone_redundancy_enabled = false

  sku = "Standard"

  identity {
    type = "SystemAssigned"
  }

  depends_on = [azurerm_resource_group.observability_rg]
}

# Grafana instance is assigned the necessary permissions to
   scrape monitoring data
# from Azure Monitor Log Analytics Worksapace
resource "azurerm_role_assignment"
   "grafana-reader-role-assignment" {
  scope = data.azurerm_subscription.primary.id
  principal_id =
      azurerm_dashboard_grafana.grafana-dashboard.identity[0].principal_id
  role_definition_name = "Monitoring Reader"
}

# User that will be authorized to access Grafana Dashboards
data "azuread_user" "ad_user" {
  user_principal_name = "s297014@studenti.polito.it"
}

# User is assigned with the role of Grafana Admin
resource "azurerm_role_assignment"
    "grafana-admin-role-assignment" {
  scope = azurerm_dashboard_grafana.grafana-dashboard.id
  principal_id = data.azuread_user.ad_user.id
  role_definition_name = "Grafana Admin"
}
```

The upcoming list summarises the Azure resources provisioned by the Terraform script shown above:

- **azurerm_log_analytics_workspace**: Azure Log Analytics Workspace collecting monitoring data produced by Azure workloads.
- **azurerm_dashboard_grafana**: Azure Managed Grafana is a solution employed for the aggregation, correlation, and analysis of data collected from heterogeneous data sources.
- **azurerm_role_assignment**: Auxiliary resources used to assign the proper data access permissions to Grafana instance and to users supposed to be managing Grafana.

As a result of the provisioning of all the previously described resources, the implemented observability stack reflects the infrastructure design diagram depicted in figure 4.8.

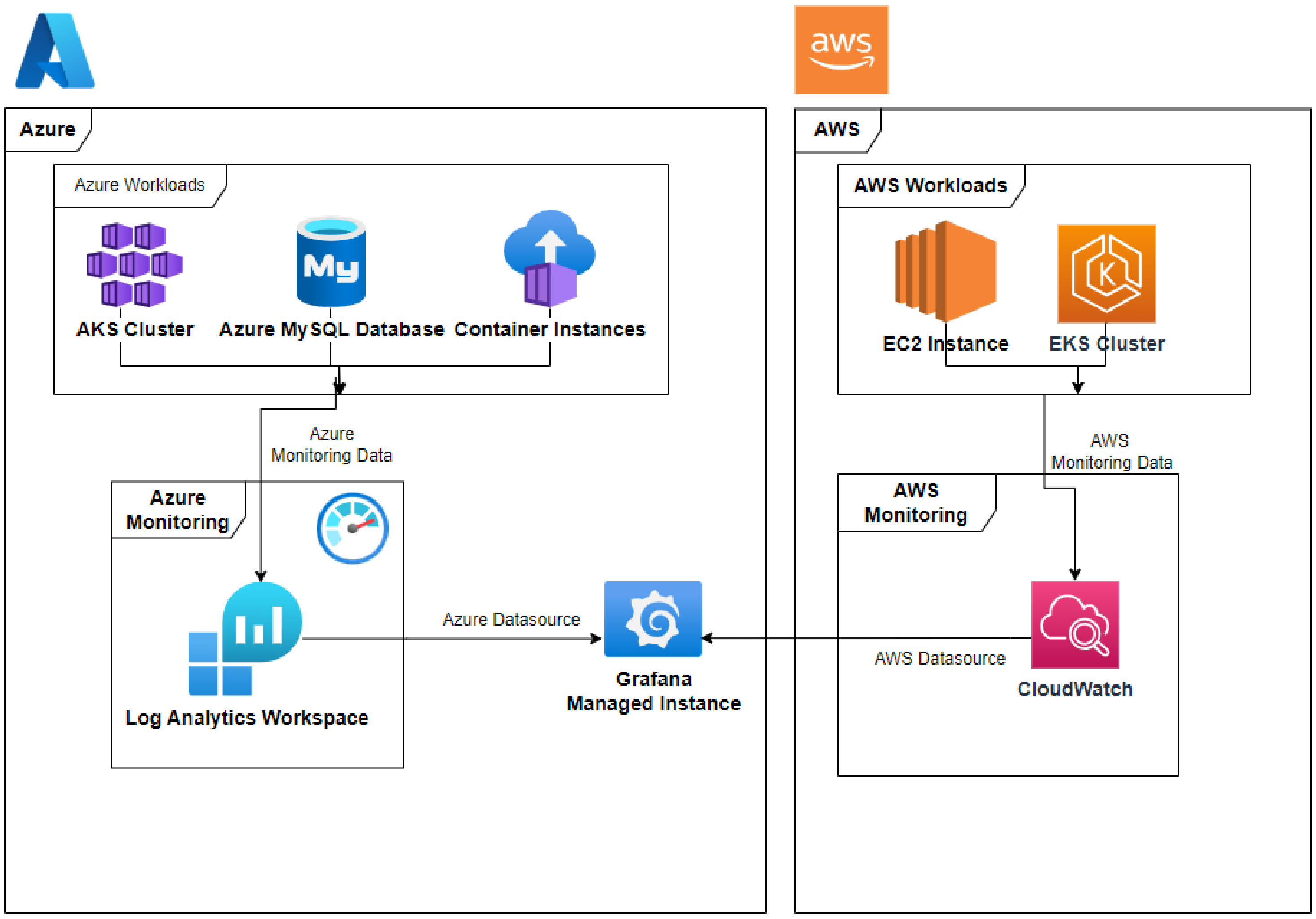


Figure 4.8: Multi-Cloud Observability Infrastructure Design

Components Interactions:

1. Azure and AWS workloads produce monitoring data collected respectively by Azure Log Analytics Workspace and AWS CloudWatch.
2. Grafana instance, which has visibility outside of the Azure cluster, scrapes monitoring data both from Azure Log Analytics Workspace (internal network) and AWS CloudWatch (Internet).
3. Data collected by Grafana is visualized through pre-made or custom Dashboards.

4.4.7 Considerations on the Multi-Cloud Infrastructure

The Multi-Cloud infrastructure realized and thoroughly analyzed in this chapter has the intention of representing an organization's cloud environment as realistically as possible. Due to this, strategies to address Multi-Cloud Identity and Access Management, Secret Management, and Observability, which represent some of the most common challenges, were explored. Implementing these frameworks across multiple cloud environments is challenging due to the interoperability required among heterogeneous technologies. However, not addressing these challenges would have resulted in the deployment of redundant technologies, complicating overall infrastructure management and expanding the attack surface of the entire Multi-Cloud environment. Chapter 5 focuses entirely on another extremely important topic already introduced from a theoretical point of view in Chapter 2: **Cloud Security**.

By adopting a specific Cloud Native Application Protection Platform (CNAPP), I've integrated into my Multi-Cloud environment a collection of vital controls that ensure workloads' security as well as their compliance against industry and regulatory standards, thus covering the best practices discussed in CSPM 3.4, CIEM 3.5, and CWP 3.6.

Chapter 5

CNAPP Case Study: Sysdig Secure

5.1 Introduction

A Cloud Native Application Protection Platform is a security tool that intends to replace multiple independent security solutions each focused only on a single security aspect. This all-in-one cloud-native platform aims at simplifying the monitoring, detection, and response of potential cloud security threats and vulnerabilities; moreover, having the possibility of using a single complete tool to manage the security of one or more cloud environments leads also to the minimization of management complexity and facilitation of DevSecOps operations.

CNAPPs were designed to solve many of the problems arising from the exponential growth in the use of cloud-native technologies, such as:

- **Advanced Observability and Risk Quantification**: Combining all previous security solutions in a single one improves the detection of risks within complex cloud infrastructures and the ability of security teams to respond to those risks.
- **Centralize Cloud Security**: The consolidation of reporting, scanning, and threat detection enabled by this single security platform accounts also for the minimization of human errors that were due to the presence of multiple different tools, and for the reduction of time taken by security teams before being notified of a potential security threat.
- **Secure Software Development**: As CNAPPs enable to detect and rapidly solve misconfigurations, it is a common practice nowadays to integrate these

security solutions within CI/CD pipelines to scan IaC configurations before they get deployed in a production environment (As also demonstrated in section 4.2).

The Cloud Native Application Protection Platform solution examined in this dissertation is referred to as **Sysdig Secure**. Sysdig Secure ranked as the second top-rated CNAPP solution according to Gartner [6], encompasses a multitude of features that can be leveraged to protect several cloud environments at the same time. Apart from Sysdig's reputation, the other factor that led me to choose this platform is the ongoing partnership between Liquid Reply and Sysdig itself, which allowed me to use the tool for free.

Some of the security features provided by Sysdig Secure are:

- **Infrastructure-as-Code Security**: Sysdig's GitHub application discussed in section 4.2, enables scanning of IaC templates within CI/CD pipelines, facilitating the detection of potential security issues due to misconfigurations, early in the software development lifecycle.
- **Cloud & Kubernetes Security Posture Management**: Sysdig provides two different Posture modules: one for handling Cloud Accounts compliance and one for Kubernetes compliance. Cloud and Kubernetes resources are persisted in an inventory that enhances resources and violations visibility.
- **Cloud Infrastructure Entitlements Management**: Identity and Access module available only for AWS accounts, that enumerates users, roles, groups, and IAM policies and performs risk assessments based on user configurations and privileges.
- **Container Registry Scanning**: Sysdig module that allows scanning for obsolete or vulnerable container images stored within private container registries.
- **Network Security**: Sysdig Network Security can be used in the context of Kubernetes clusters to:
  1. Visualize internal cluster's network topology
  2. Track each Pod's ingress and egress communications
  3. Generate Kubernetes network policies based on the configurations applied in Sysdig's dashboard.
- **Vulnerability Management & Runtime Threat Detection**: Adopting a Vulnerability Management strategy and Runtime threat detection techniques

are essential to provide an additional layer of defense and to safeguard workloads during their execution. to fulfill this need, Sysdig's runtime scanner offers:

- Automatic observation and reporting about all running workloads, thus providing a real-time view of workloads' current state.
- Periodic vulnerability assessments that guarantee an up-to-date view of running-workloads vulnerabilities.

The subsequent exposition presents the approach followed to profitably integrate Sysdig Secure CNAPP's most relevant security features with the Multi-Cloud infrastructure whose design and setup were extensively discussed in Chapter 4 In section 5.2, dedicated emphasis has also been directed toward the explanation of the different types of policies deployed by Sysdig's platform.

5.2 Sysdig Policies

Policies defined within this CNAPP are the element that allows threats to be detected across entire cloud infrastructures. Sysdig categorizes policies into three groups based on the specific cloud security issue they address: Posture, Vulnerability, and Threat Detection Policies.

Posture Policies
Regarding Posture Policies, Sysdig defines three additional concepts: Controls, Zones, and Policies. A Control is a precise rule that assesses the compliance of a resource's posture setting; Policies on the other hand can be seen as a container that aggregates related Controls, and A Zone likewise, encompasses a group of Policies that can be selectively applied to Cloud Accounts, Kubernetes Clusters or specific Hosts. The platform already includes numerous pre-made policies that belong to the most famous Compliance Benchmarks such as CIS, DISA, NIST, and PCI-DSS; nonetheless, custom ones may also be added to suit specific needs.

Vulnerability Policies
Sysdig's Vulnerability Policies allow for scanning Pipeline, Runtime, and Host vulnerabilities leveraging *"vulnerability"* and *"image configuration"* rules: the formers are in charge of detecting common software package vulnerabilities, while the latter instead analyze container images and their metadata to detect potential threats.

Threat Detection Policies Sysdig Threat Detection Runtime Policies encompass a collection of rules and configurations designed to monitor clusters and hosts, detect and respond to security violations in real time, and promptly alert security teams through dedicated communication channels like Slack, Microsoft Teams, and Emails. Out-of-the-box Sysdig offers and maintains a diverse set of policies that safeguard cloud infrastructures from intrusions, malware, and DDoS attacks. However, as with Vulnerability and Posture Policies, custom Threat Detection policies and rules can be imported as well.

5.3 Integration of Sysdig Secure

5.3.1 IaC Security

The DevSecOps pipelines showcased in section 4.2 include Sysdig Git integration as one of the two scanning tools set up to detect misconfigurations and security issues in infrastructure-as-code files. Such integration, which is available for the major version control systems like GitHub, GitLab, BitBucket, and Azure DevOps, can be configured at ease to assess the compliance level of infrastructure templates contained in repositories and enforce custom policies whose adherence is required in safety-critical environments.

Given that all infrastructure-as-code templates I developed have been tracked on GitHub adopting a monolithic repository strategy, the integration of Sysdig's IaC-Sec tool was finalized by installing on the previously mentioned repository, a Github Application manufactured by Sysdig itself.

Figures below respectively depict how Sysdig's GitHub App can be easily installed (5.1) and managed from Sysdig's SaaS dashboard (5.2).

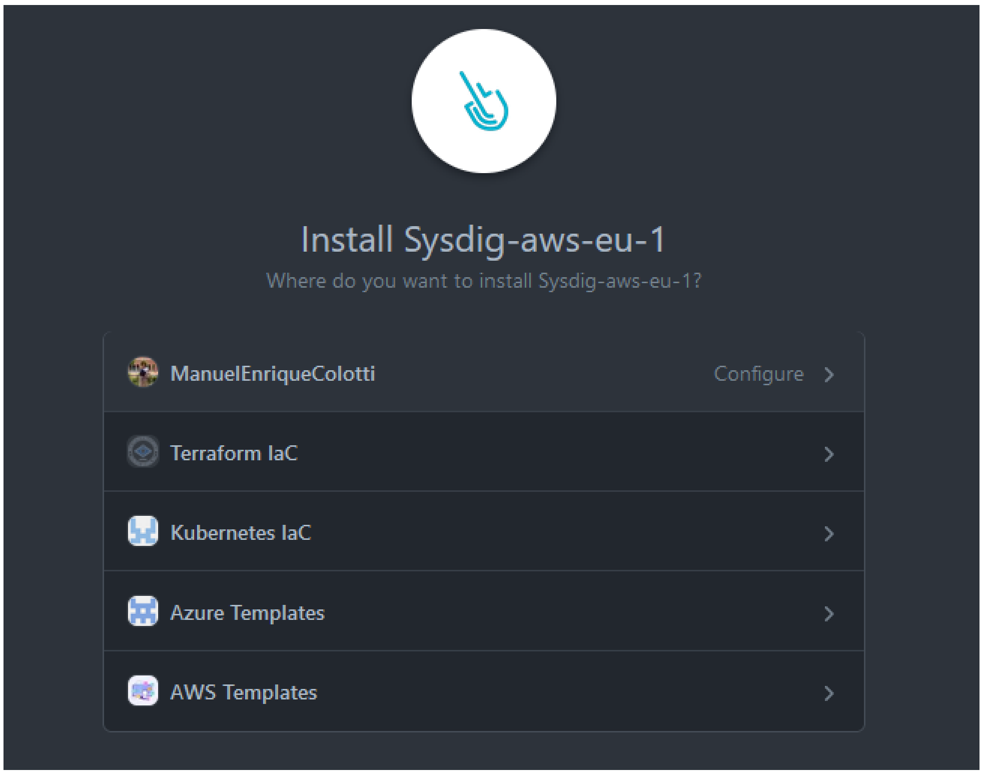
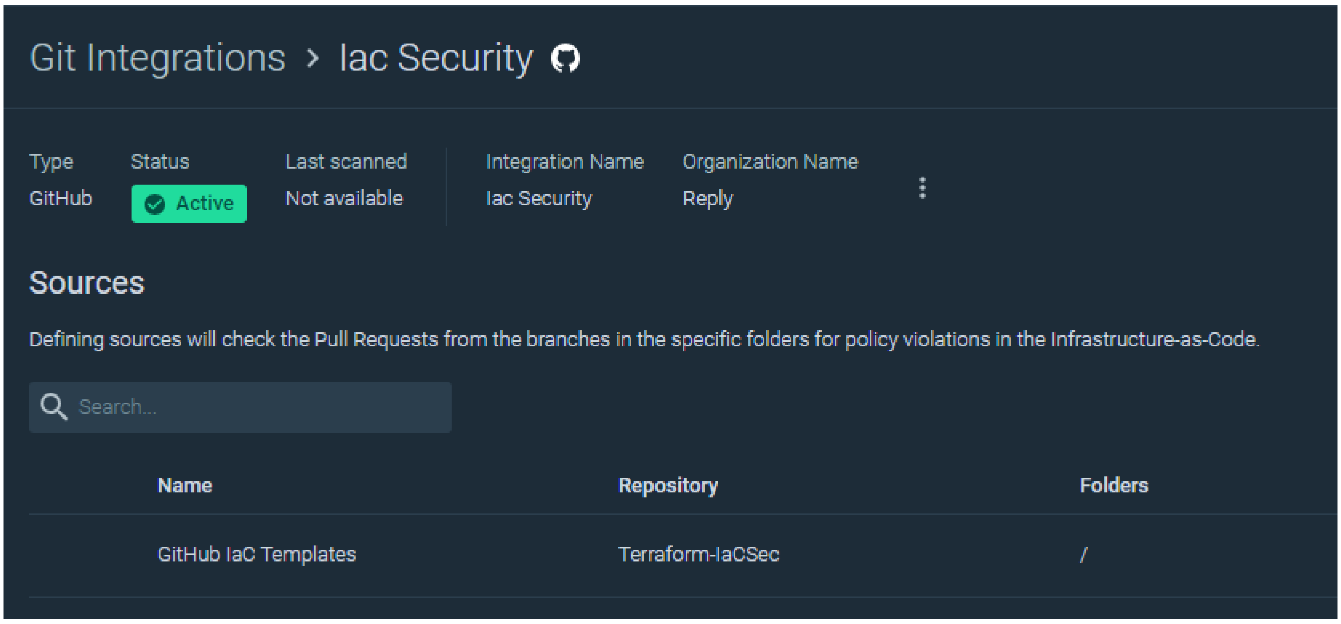


Figure 5.1: Github Sysdig App Installation



Type	Status	Last scanned	Integration Name	Organization Name
GitHub	Active	Not available	Iac Security	Reply

Sources

Defining sources will check the Pull Requests from the branches in the specific folders for policy violations in the Infrastructure-as-Code.

Name	Repository	Folders
GitHub IaC Templates	Terraform-IaCSec	/

Figure 5.2: Git Integration Dashboard

5.3.2 Compliance

As previously introduced in section 5.1, Sysdig offers two different posture modules that can be leveraged to enforce compliance policies: **Cloud** and **Kubernetes Security Posture Management**. The upcoming paragraphs explore the available integration approaches, focusing on those selected to implement Posture management within my Multi-Cloud infrastructure. Subsequently, an integration scenario for what concerns users' identities and privileges (CIEM) is also presented, accompanied by explanatory images.

Cloud Security Posture Management

Sysdig offers two different strategies to integrate Cloud Security Posture Management as part of a specific Cloud Infrastructure. The first strategy is an agentless installation which has the advantage that it only requires a few IAM roles and permissions to be deployed on a cloud to operate successfully. The second one, on the other hand, relies on an agent to be present and running within the cloud environment it is wanted to safeguard. Even though this second strategy requires more resources to be deployed, it offers Runtime Threat Detection, which was not included in the agentless installation.

Amid the array of available methods, Sysdig supplies some Terraform scripts designed to seamlessly integrate one of the two CSPM strategies. The following examples showcase two Terraform scripts, demonstrating both agentless and agent deployment, which were employed to integrate Cloud Security Posture Management within an Azure cloud account.

Agentless CSPM Deployment

```
terraform {
  required_providers {
    sysdig = {
      source = "sysdiglabs/sysdig"
    }
  }
}

provider "sysdig" {
  sysdig_secure_url = "https://eu1.app.sysdig.com"
  sysdig_secure_api_token = ""
}
```

```
provider "azurerm" {
  features { }
  subscription_id = "8eb30f69-69f6-4ff0-99ea-f9edd2274036"
}

module "sysdig-sfc-agentless" {
  source =
     "sysdiglabs/secure-for-cloud/azurerm//modules/services/cloud-bench"
  subscription_id = "8eb30f69-69f6-4ff0-99ea-f9edd2274036"
}
```

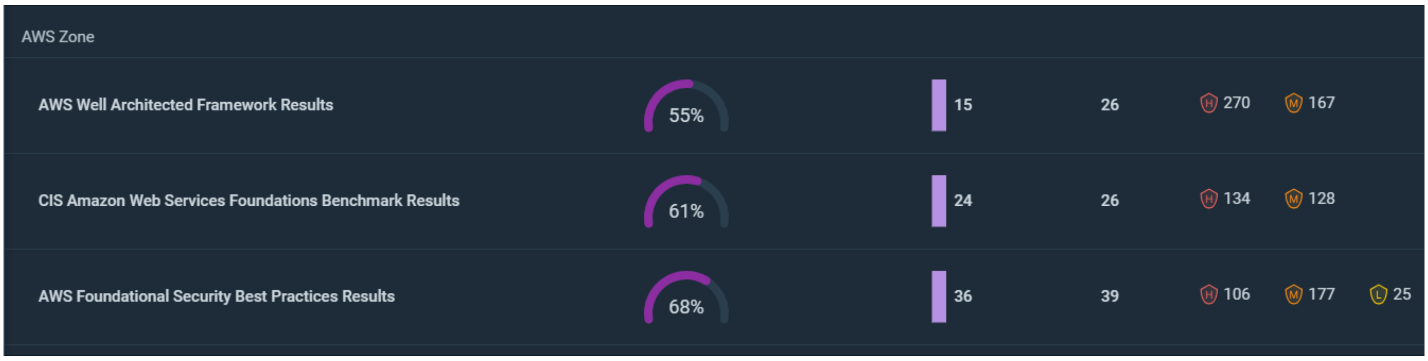
Agent CSPM Deployment

```
terraform {
  required_providers {
    sysdig = {
      source = "sysdiglabs/sysdig"
    }
  }
}

provider "sysdig" {
  sysdig_secure_url = "https://eu1.app.sysdig.com"
  sysdig_secure_api_token = ""
}

provider "azurerm" {
  features { }
  subscription_id = "8eb30f69-69f6-4ff0-99ea-f9edd2274036"
}

module "single-subscription" {
  source =
     "sysdiglabs/secure-for-cloud/azurerm//examples/single-subscription"
  deploy_active_directory = false
  location = "uksouth"
}
```

After having performed one of the two installations and customized Policies and Zones to conduct Posture assessments within a predefined scope, Sysdig will take care of periodically analyzing the configuration of cloud resources. It will then generate a compliance report that enumerates all misconfigured assets along with the necessary steps to remediate each issue and restore the environment's compliance.

Upcoming images depict some of the Posture Management dashboards provided by Sysdig as soon as a Compliance scan is completed.

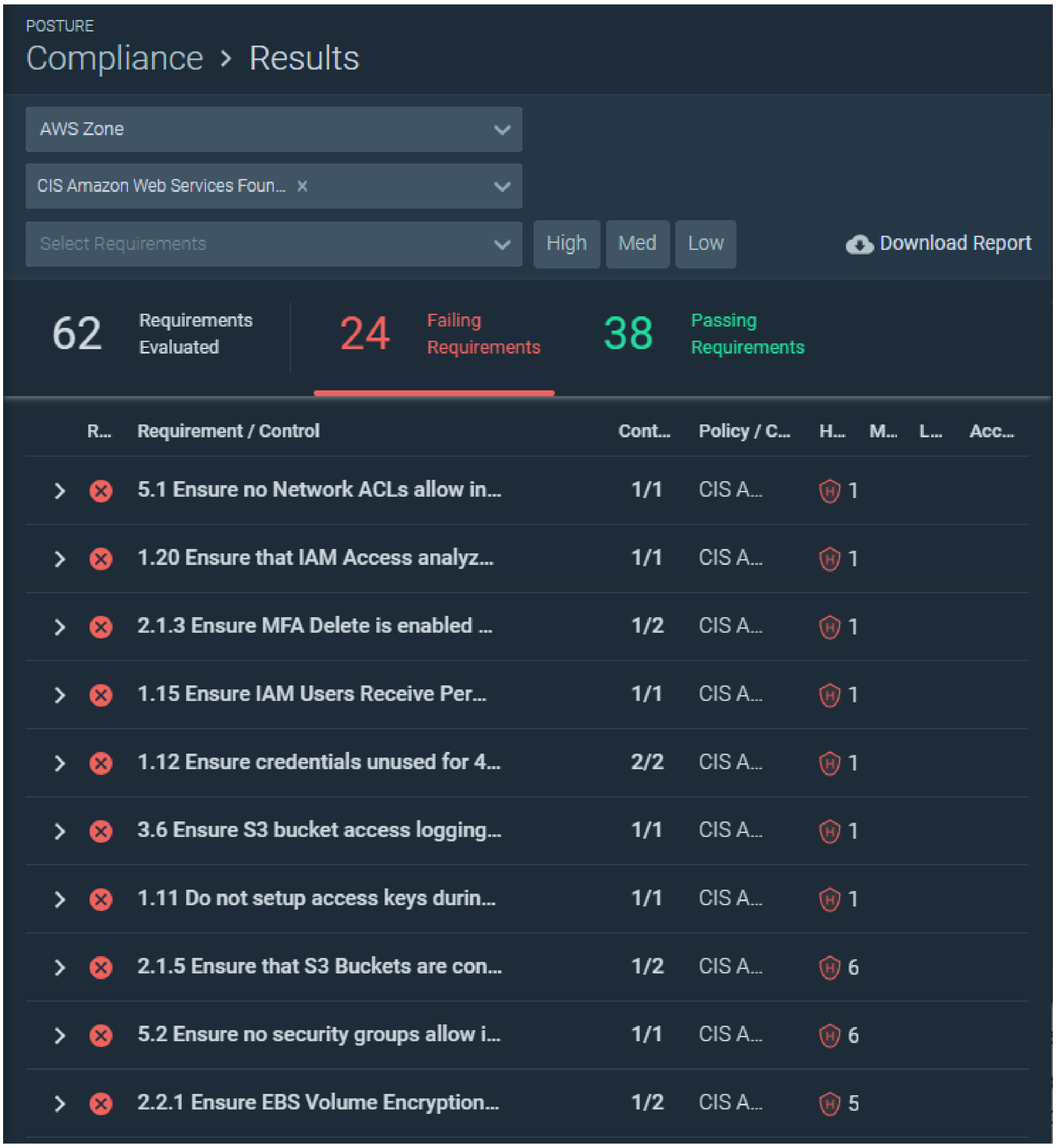
- **Figure 5.3** shows the three policies used by Sysdig to assess the compliance level of AWS's Cloud Account.
- **Figure 5.4** gives an overview of which and how many requirements passed the posture checks enforced by CIS AWS Foundations Benchmarks.



AWS Zone

AWS Well Architected Framework Results	55%	15	26	H 270	M 167	
CIS Amazon Web Services Foundations Benchmark Results	61%	24	26	H 134	M 128	
AWS Foundational Security Best Practices Results	68%	36	39	H 106	M 177	L 25

Figure 5.3: AWS Posture Assessments Overview


POSTURE

Compliance > Results

AWS Zone

CIS Amazon Web Services Foun... ×

Select Requirements | High | Med | Low | Download Report

62 Requirements Evaluated | 24 Failing Requirements | 38 Passing Requirements

R...	Requirement / Control	Cont...	Policy / C...	H...	M...	L...	Acc...
✗	5.1 Ensure no Network ACLs allow in...	1/1	CIS A...	1			
✗	1.20 Ensure that IAM Access analyz...	1/1	CIS A...	1			
✗	2.1.3 Ensure MFA Delete is enabled ...	1/2	CIS A...	1			
✗	1.15 Ensure IAM Users Receive Per...	1/1	CIS A...	1			
✗	1.12 Ensure credentials unused for 4...	2/2	CIS A...	1			
✗	3.6 Ensure S3 bucket access logging...	1/1	CIS A...	1			
✗	1.11 Do not setup access keys durin...	1/1	CIS A...	1			
✗	2.1.5 Ensure that S3 Buckets are con...	1/2	CIS A...	6			
✗	5.2 Ensure no security groups allow i...	1/1	CIS A...	6			
✗	2.2.1 Ensure EBS Volume Encryption...	1/2	CIS A...	5			

Figure 5.4: AWS Compliance Results against CIS AWS Foundations Benchmark

Kubernetes Security Posture Management

Kubernetes Security Posture Management, or KSPM, is an additional module Sysdig offers that complements CSPM. Unlike CSPM, KSPM focuses on identifying component misconfigurations within Kubernetes Clusters. Examples of misconfigurations that can be spotted by a KSPM tool are:

- Network configuration errors
- Over permissive user privileges
- Over permissive access to secrets

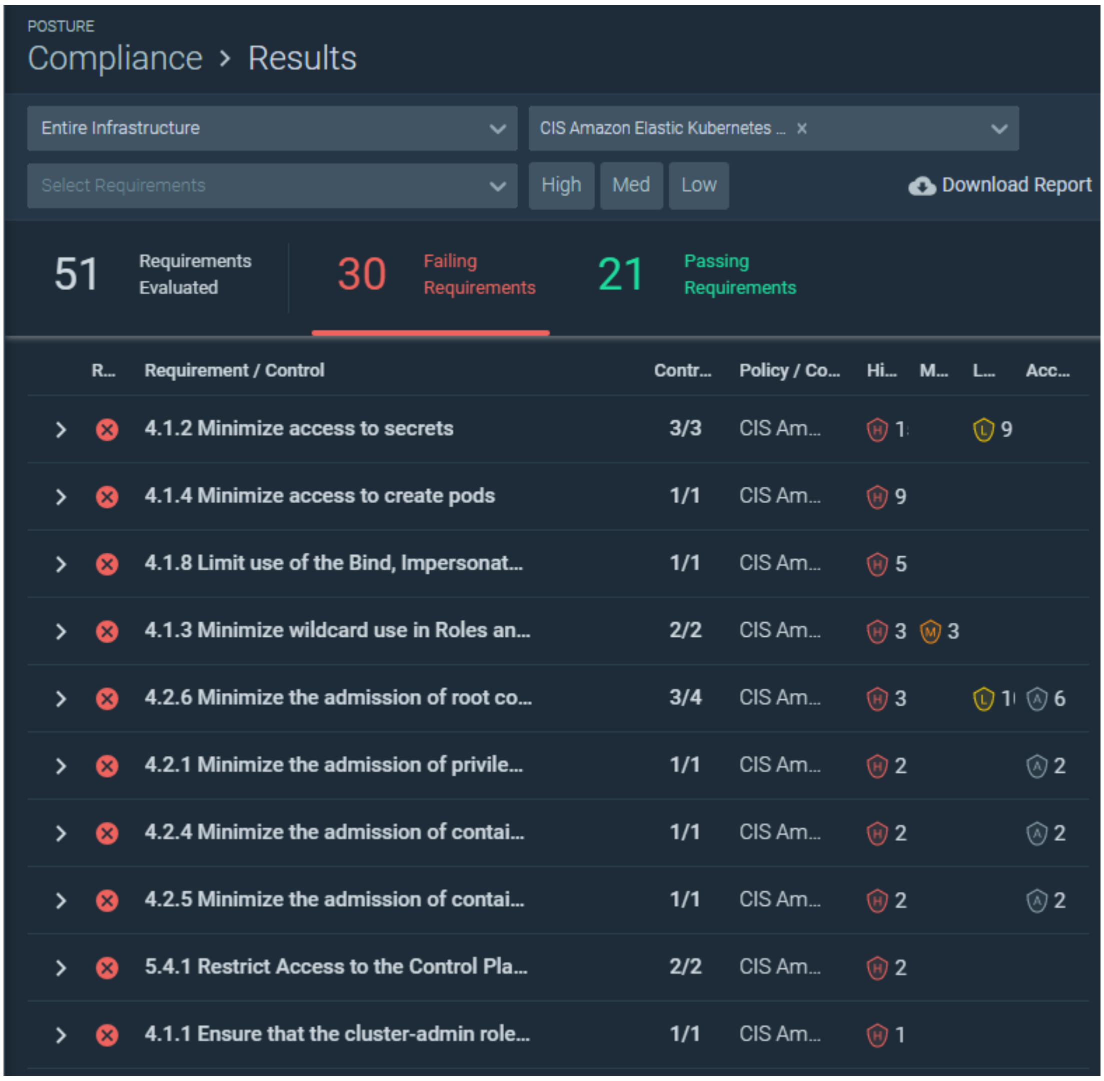
Integrating KSPM into a Kubernetes cluster can only be accomplished by deploying a Sysdig agent within the Kubernetes environment. Besides scanning component configurations to identify potential compliance issues, this agent can also detect package vulnerabilities and runtime threats, similarly to the previous CSPM agent installation strategy.

Below, the provided code showcases the simple commands that can be executed to install Sysdig's agent using an **Helm chart**:

```
helm repo add sysdig https://charts.sysdig.com
helm repo update
helm install sysdig-agent
    --namespace sysdig-agent --create-namespace \
    --set global.sysdig.accessKey=<ACCESS_KEY> \
    --set global.sysdig.region=<SAAS_REGION> \
    --set
       nodeAnalyzer.secure.vulnerabilityManagement.newEngineOnly=true
       \
    --set global.kspm.deploy=true \
    --set nodeAnalyzer.nodeAnalyzer.benchmarkRunner.deploy=false \
    --set nodeAnalyzer.nodeAnalyzer.hostScanner.deploy=true
    --set global.clusterConfig.name=<CLUSTER_NAME> \
sysdig/sysdig-deploy
```

If the installation is successful, the agent initiates the collection of Kubernetes Cluster component configurations, verifies the posture, and generates one or more reports. These reports are accessible through Sysdig's Compliance dashboards and can be used to address all identified misconfigurations.

The image below provides an overview of the misconfigurations detected by enforcing the CIS Amazon Elastic Kubernetes Benchmarks.


POSTURE

Compliance > Results

Entire Infrastructure | CIS Amazon Elastic Kubernetes ... ×

Select Requirements | High | Med | Low | Download Report

51 Requirements Evaluated | 30 Failing Requirements | 21 Passing Requirements

	R...	Requirement / Control	Contr...	Policy / Co...	Hi...	M...	L...	Acc...
>	✗	4.1.2 Minimize access to secrets	3/3	CIS Am...	1		9	
>	✗	4.1.4 Minimize access to create pods	1/1	CIS Am...	9			
>	✗	4.1.8 Limit use of the Bind, Impersonat...	1/1	CIS Am...	5			
>	✗	4.1.3 Minimize wildcard use in Roles an...	2/2	CIS Am...	3	3		
>	✗	4.2.6 Minimize the admission of root co...	3/4	CIS Am...	3		1	6
>	✗	4.2.1 Minimize the admission of privile...	1/1	CIS Am...	2			2
>	✗	4.2.4 Minimize the admission of contai...	1/1	CIS Am...	2			2
>	✗	4.2.5 Minimize the admission of contai...	1/1	CIS Am...	2			2
>	✗	5.4.1 Restrict Access to the Control Pla...	2/2	CIS Am...	2			
>	✗	4.1.1 Ensure that the cluster-admin role...	1/1	CIS Am...	1			

Figure 5.5: Kubernetes Compliance Results against CIS Amazon Elastic Kubernetes Benchmarks

Cloud Infrastructure Entitlements Management

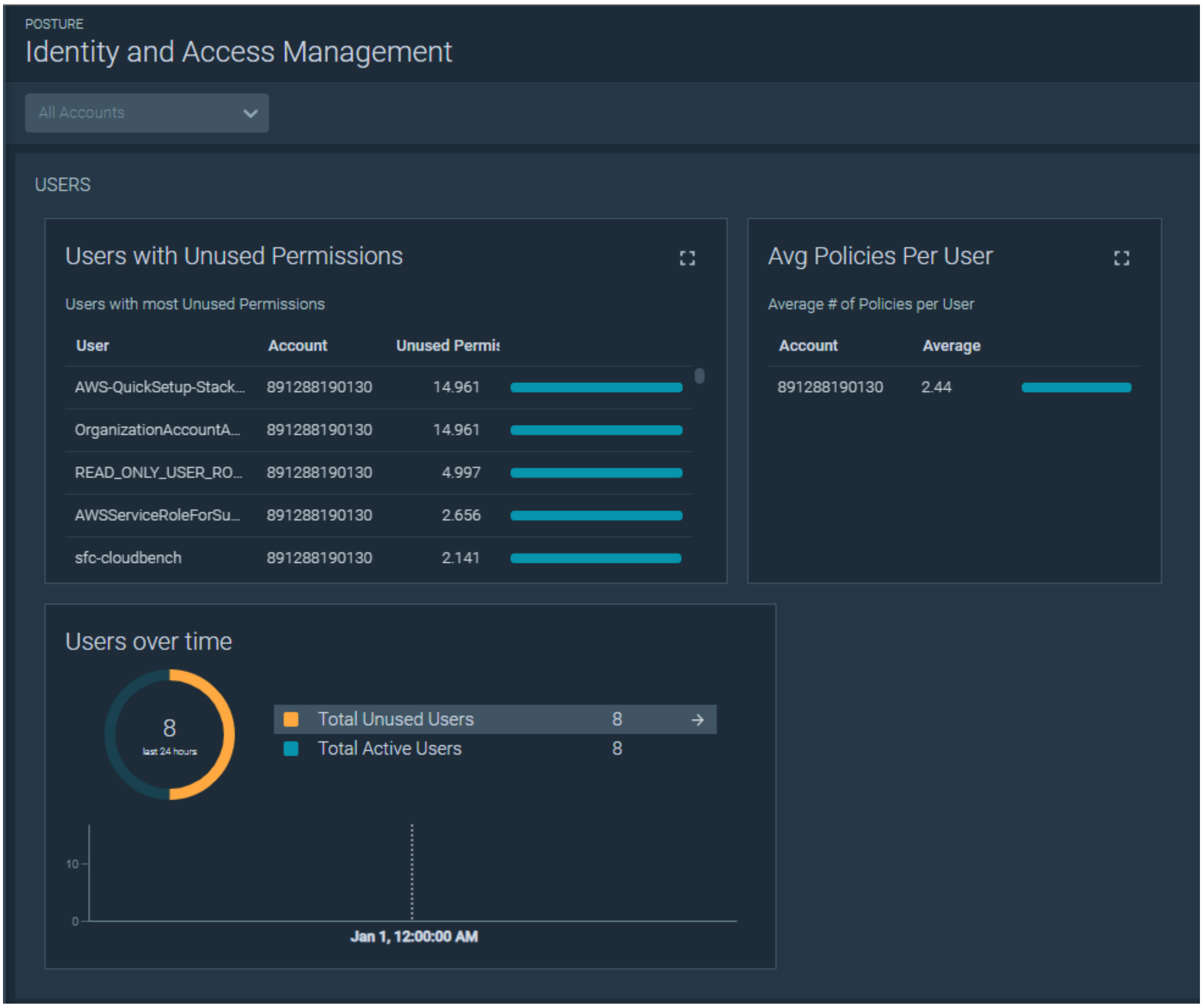
The last compliance component offered by Sysdig is an Identity and Access module whose first objective is to give an overview of users, groups, roles, and permissions within a cloud account, and to detect overly permissive policies that may pose risks to the environment's security. As of the current implementation, Sysdig's CIEM module can only be used in the context of AWS cloud accounts and can be set up either by using a CloudFormation template or by using Terraform.

Among the two available configuration methods for establishing Cloud Infrastructure Entitlements Management, the preferred choice was to deploy this compliance module on Reply's AWS account, using the Terraform templates made available by Sysdig's developers. Mirroring the deployment approach of the CSPM agent outlined earlier, setting up the CIEM module described in the Terraform file below also enables comprehensive Threat Detection coverage across the entirety of the AWS account.

CIEM Terraform Module for AWS Accounts

```
terraform {
    required_providers {
        sysdig = {
            source = "sysdiglabs/sysdig"
        }
    }
}

provider "sysdig" {
    sysdig_secure_url = "<SYSDIG_SECURE_URL>"
    sysdig_secure_api_token = "<SYSDIG_SECURE_API_TOKEN>"
}

provider "aws" {
    region = "<AWS-REGION>; ex. us-east-1"
}

module "secure_for_cloud_aws_single_account_ecs" {
    source =
        "sysdiglabs/secure-for-cloud/aws//examples/single-account-ecs"
}
```

After the setup is completed, Sysdig's agent starts collecting Identity and Access data pertinent to the AWS account in which it is deployed. In approximately one day from the beginning of the information gathering, data concerning users, roles, groups, and policies will be showcased on Sysdig's compliance dashboards. An example of CIEM's Overview dashboard offered by Sysdig is depicted in figure 5.6.



Users with Unused Permissions

Users with most Unused Permissions

User	Account	Unused Permis
AWS-QuickSetup-Stack...	891288190130	14.961
OrganizationAccountA...	891288190130	14.961
READ_ONLY_USER_RO...	891288190130	4.997
AWSServiceRoleForSu...	891288190130	2.656
sfc-cloudbench	891288190130	2.141

Avg Policies Per User

Average # of Policies per User

Account	Average
891288190130	2.44

🟧	Total Unused Users	8
🟦	Total Active Users	8

Figure 5.6: AWS Account CIEM Overview Dashboard

5.3.3 Vulnerability Management

Developing an effective vulnerability management plan requires, as a primary step, the identification of the key lifecycle stages that need to be addressed. Sysdig distinguishes three distinct stages, each enabling the management of vulnerabilities during a specific phase of the development lifecycle.

1. **Pipeline Stage**: The definition and update of custom container images may easily introduce software vulnerabilities that can be exploited by malicious users. To tackle this problem Sysdig offers a container image scanning tool that can be either integrated in CI/CD pipelines or directly used from a command line interface. The adoption of such a tool allows the detection of software package vulnerabilities even before an image is used within a container.

2. **Registry Stage**: Sysdig's Container Registry Scanner is a tool that seamlessly integrates with widely employed image registry solutions for both private and public clouds, serving as an additional layer of defense between pipeline and runtime stages. The periodic scanning of container images stored within image registries leads to two primary advantages:

   (a) Identification of recently discovered vulnerabilities that impact the software residing within a registry.

   (b) Detection of vulnerabilities affecting third-party software installed after the pipeline scanning's completion.

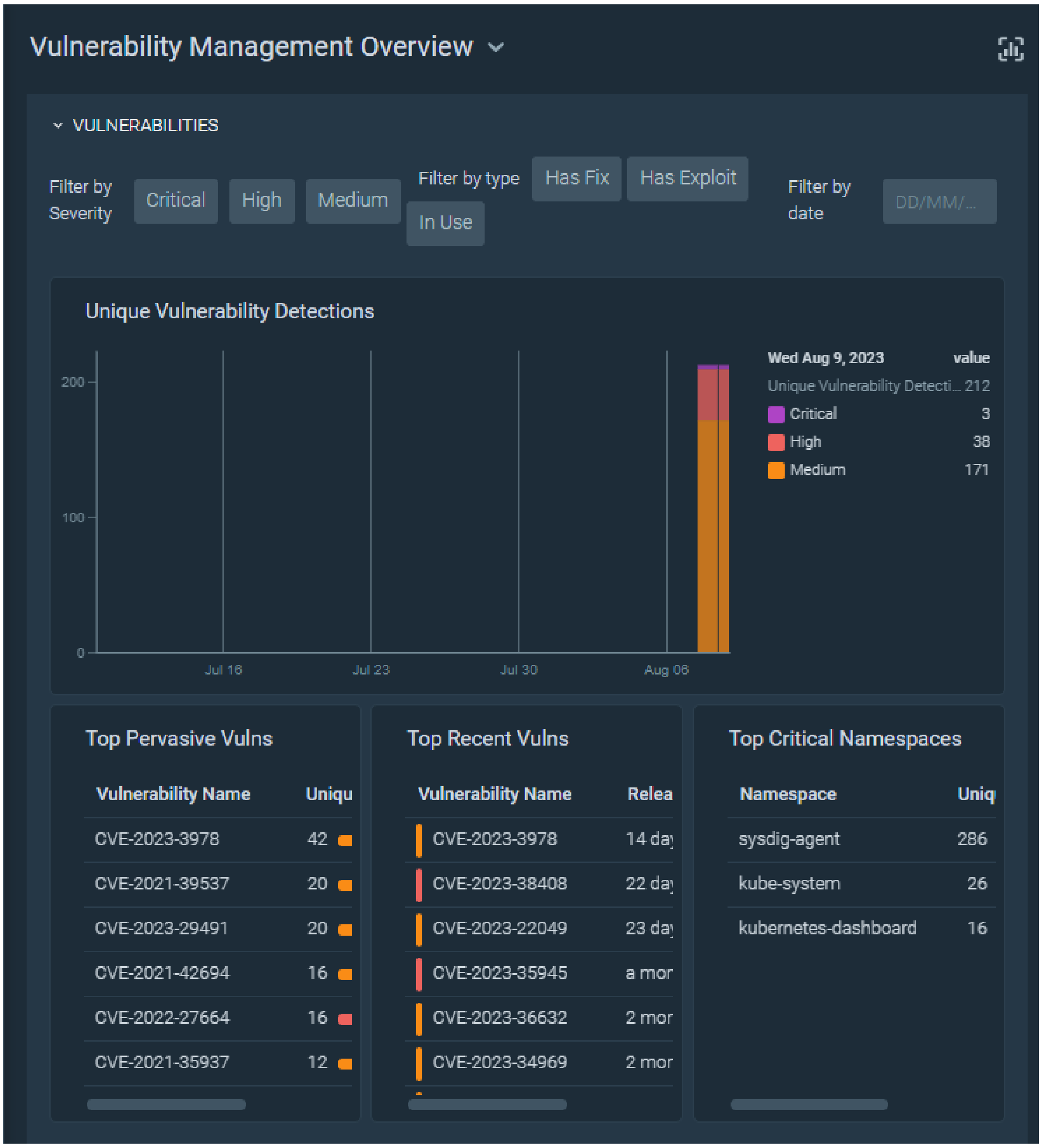
3. **Runtime Stage**: Sysdig Secure employs specialized agents to detect threats to workloads during runtime. These agents conduct scans on the images in use, enabling the identification of newly discovered vulnerabilities, and highlighting the most critical ones. They can be deployed across various environments, including:

   - **Kubernetes Clusters**: Helm chart installation
   - **Hosts**: container or package installation
   - **AWS Elastic Container Service (ECS) on EC2**
   - **AWS Elastic Container Service on Fargate**

Among these three stages, I've decided to exclusively focus on the runtime one. The motive behind this derives from the fact that in my use-case scenario, no container images are being developed and no private images registry was provisioned. Consequently, the implementation of vulnerability management for the pipeline and registry stages would have been futile. Conversely integrating runtime vulnerability management into my Multi-cloud infrastructure enables periodic vulnerability assessments on workloads as they operate.

Since the key workloads of my infrastructure are the two Kubernetes clusters deployed in Azure and AWS cloud platforms, as they symbolize a collection of services offered by a business to its customers, two separate Sysdig agents were set up to operate within these clusters. The installation procedure reflects the one described in section 5.3.2 that uses a single helm chart to deliver runtime threat detection, host scanning, runtime image scanning, and compliance.

The subsequent images provide an overview of the vulnerabilities discovered in the Kubernetes Cluster hosted on Amazon Web Services after letting Sysdig's agent operate for some hours.



Vulnerability Management Overview

VULNERABILITIES

Filter by Severity: Critical | High | Medium

Filter by type: Has Fix | Has Exploit | In Use

Filter by date: DD/MM/...

Unique Vulnerability Detections

Wed Aug 9, 2023	value
Unique Vulnerability Detecti...	212
🟪 Critical	3
🟥 High	38
🟧 Medium	171

Top Pervasive Vulns

Vulnerability Name	Uniqu
CVE-2023-3978	42 🟧
CVE-2021-39537	20 🟧
CVE-2023-29491	20 🟧
CVE-2021-42694	16 🟧
CVE-2022-27664	16 🟥
CVE-2021-35937	12 🟧

Top Recent Vulns

Vulnerability Name	Relea
CVE-2023-3978	14 day
CVE-2023-38408	22 day
CVE-2023-22049	23 day
CVE-2023-35945	a mor
CVE-2023-36632	2 mor
CVE-2023-34969	2 mor

Top Critical Namespaces

Namespace	Uniq
sysdig-agent	286
kube-system	26
kubernetes-dashboard	16

Figure 5.7: AWS Kubernetes Cluster Vulnerability Management Dashboard

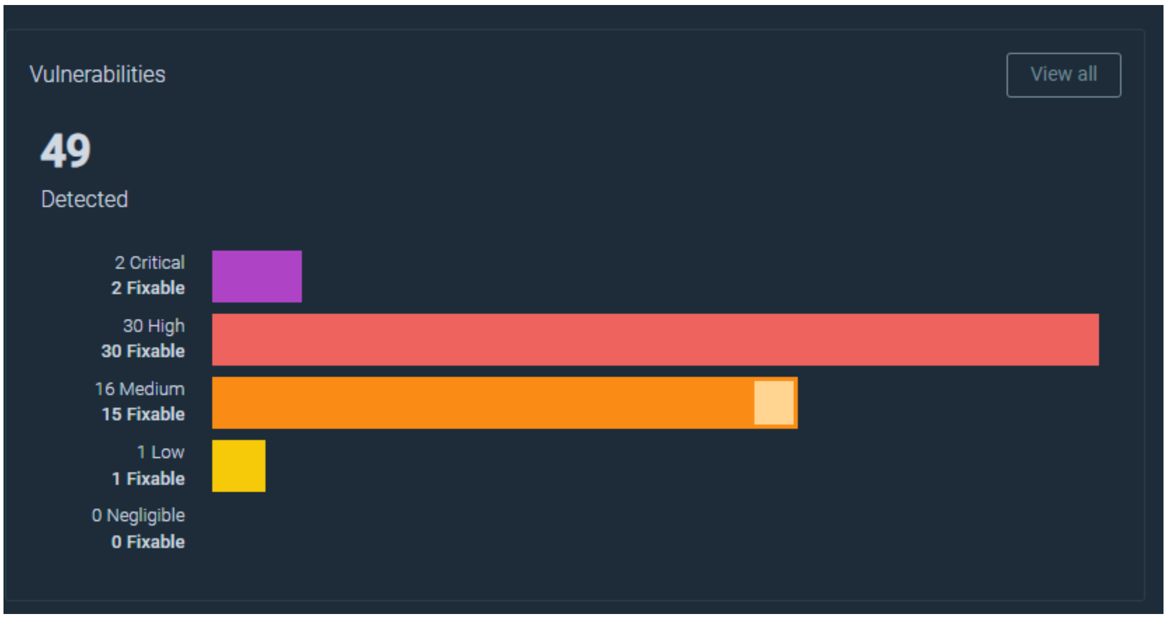


Figure 5.8: AWS Kubernetes Cluster Vulnerabilities Overview

5.3.4 Runtime Threat Detection

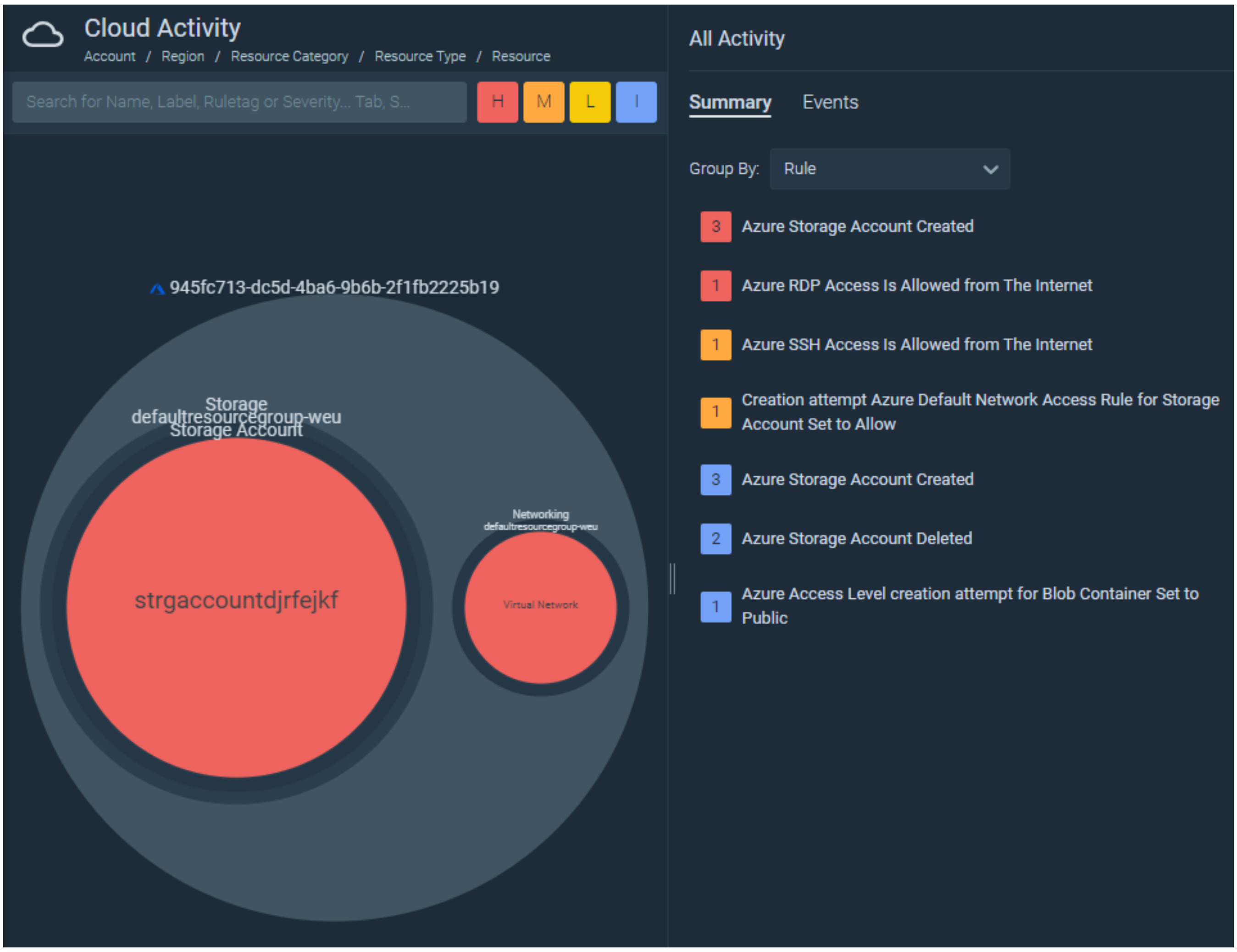
Runtime Threat Detection represents the final feature that has been integrated into the Multi-cloud infrastructure showcased in chapter 4. It is designed to identify and respond to security violations and anomalous activities. Sysdig Secure offers the option of using pre-made and customized policies to specify which runtime events should be detected, notified, and consequently responded to.

The runtime events that can be detected with the available policies may belong to several different environments; for instance, it is possible to spot and be alerted of security violations on three different levels: Cloud Accounts, Hosts, and Kubernetes clusters. For each of these scopes, Sysdig already provides bundles of rules that allow the identification of the most common threats, some examples are:

- **Cloud Account Scope**
  - Multi-Factor Authentication Deactivated for a User **(Azure)**
  - Azure RDP/SSH Access Is Allowed from the Internet **(Azure)**
  - CloudTrail Trail Deleted **(AWS)**
  - Console Root Login without MFA **(AWS)**
- **Hosts Scope**
  - Base64'd ELF file on Command Line **(Linux)**
  - Netcat Remote Code Execution in Container **(Linux)**
  - Detect reconnaissance scripts **(Linux)**
- **Kubernetes Cluster Scope**
  - Create Privileged Pod
  - Detect Attach/Exec Attempts to a Pod
  - Detect the Join of an Untrusted Node

The images depicted below show some runtime events captured by Sysdig after certain potentially malicious actions were taken on Azure's Cloud Account and a Kubernetes Cluster (Figures 5.9 and 5.10).

Azure Cloud Account Runtime Threat Detection

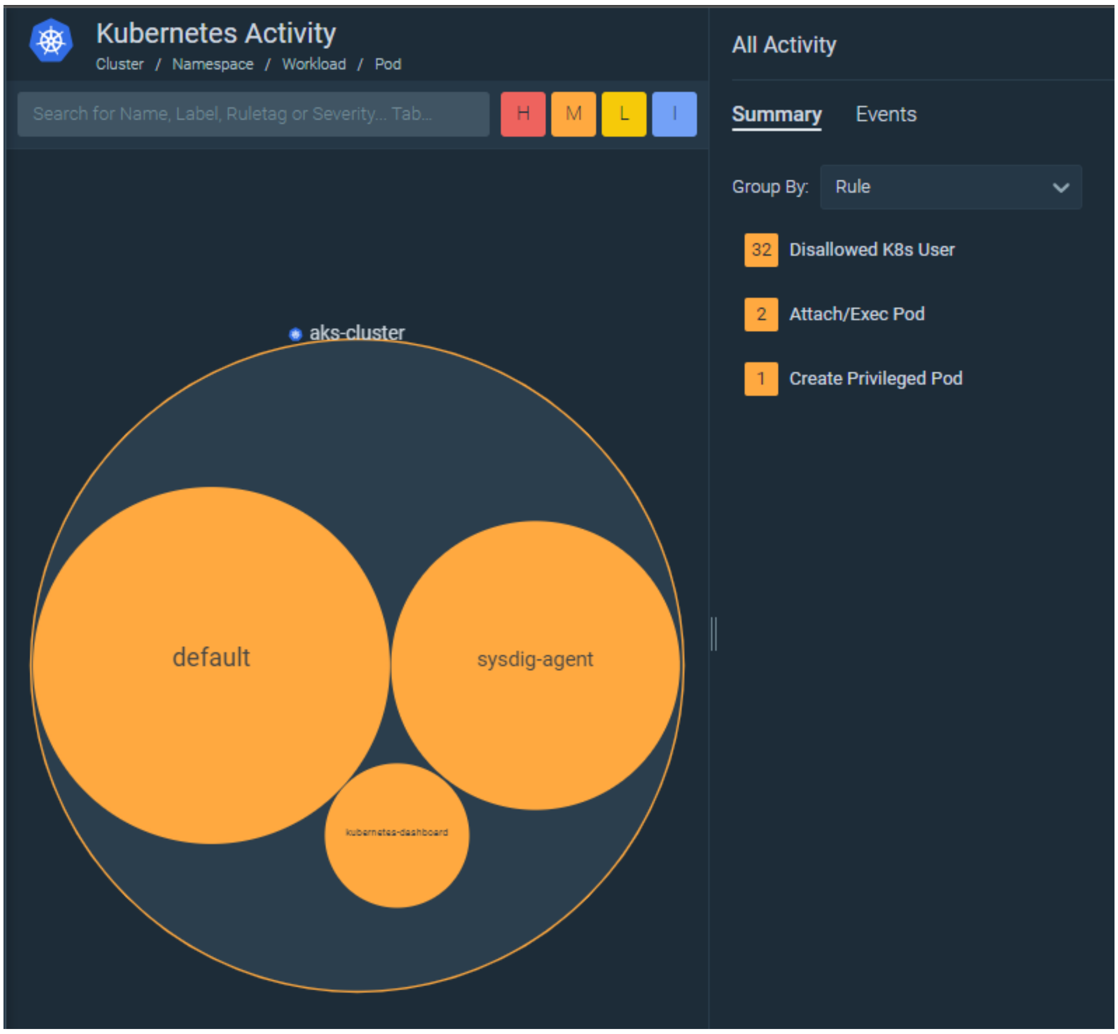


Group By: Rule

- 3 Azure Storage Account Created
- 1 Azure RDP Access Is Allowed from The Internet
- 1 Azure SSH Access Is Allowed from The Internet
- 1 Creation attempt Azure Default Network Access Rule for Storage Account Set to Allow
- 3 Azure Storage Account Created
- 2 Azure Storage Account Deleted
- 1 Azure Access Level creation attempt for Blob Container Set to Public

Figure 5.9: Azure Cloud Account Runtime Violations Overview

Kubernetes Cluster Runtime Threat Detection



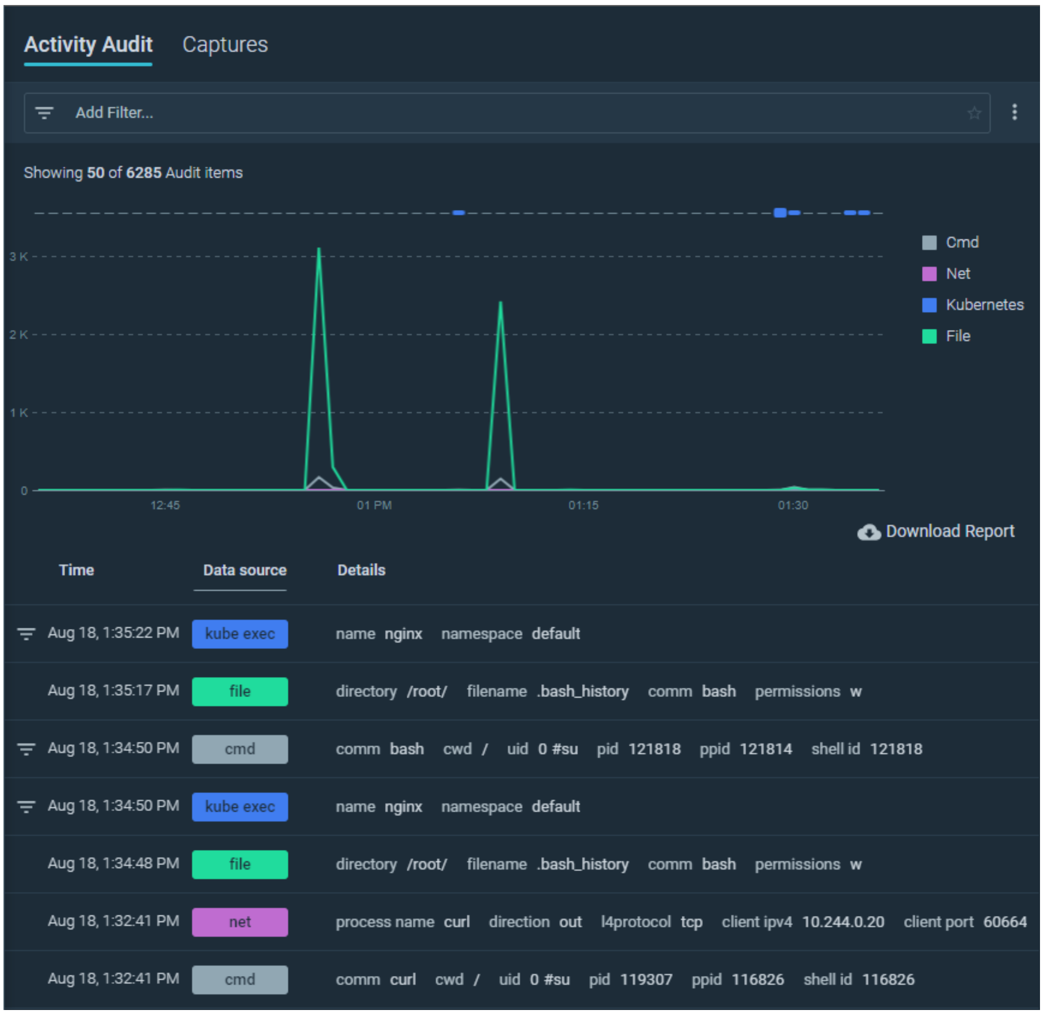
Group By: Rule

32 Disallowed K8s User
2 Attach/Exec Pod
1 Create Privileged Pod

Figure 5.10: Kubernetes Cluster Runtime Violations Overview

Runtime Auditing

By deploying an enhanced version of Sysdig's Kubernetes Agent - an additional Admission Controller, and some other resources can be automatically configured on the cluster. These latter allow Sysdig to audit every activity performed within the cluster such as modified files, execution of CLI commands, and opening of remote network connections. Figure 5.11 shows the activities audited by the cluster agent after I intentionally opened a shell inside a pod (*blue tag*), invoked curl to perform an HTTP request to a remote server (*grey and pink tags*), and inevitably modified the *.bash_history* file belonging to the root user (*green tag*).



Showing **50** of **6285** Audit items

Time	Data source	Details
Aug 18, 1:35:22 PM	kube exec	name nginx namespace default
Aug 18, 1:35:17 PM	file	directory /root/ filename .bash_history comm bash permissions w
Aug 18, 1:34:50 PM	cmd	comm bash cwd / uid 0 #su pid 121818 ppid 121814 shell id 121818
Aug 18, 1:34:50 PM	kube exec	name nginx namespace default
Aug 18, 1:34:48 PM	file	directory /root/ filename .bash_history comm bash permissions w
Aug 18, 1:32:41 PM	net	process name curl direction out l4protocol tcp client ipv4 10.244.0.20 client port 60664
Aug 18, 1:32:41 PM	cmd	comm curl cwd / uid 0 #su pid 119307 ppid 116826 shell id 116826

Figure 5.11: Kubernetes Cluster Runtime Audit

Chapter 6

Testing and Results

The objective of this chapter is to test the Multi-Cloud infrastructure that was designed in Chapter 4 by verifying the correct functioning of all the proposed integrations. In addition to this preliminary validation, a thorough analysis of the results obtained by Sysdig in the context of Infrastructure-as-Code Security, Cloud Security Posture Management, Vulnerability Management, and Runtime Threat Detection, is also carried out.

6.1 Test Environment

All tests are conducted on an ASUS VivoBook Pro using a Windows Operating System. Tests on the IaCSec tools are performed using a GitHub Actions CI/CD pipeline executed on a GitHub Runner with 2 CPU cores and 7 GB of RAM. The IaCSec tools posed under evaluation are Checkov v12.1347.0 and Sysdig's GitHub Application. Posture assessments of Azure and AWS Cloud Service Providers are performed by Sysdig enforcing respectively *CIS Azure Foundations* and *CIS AWS Foundations* Benchmarks. The Runtime Threat Detection correctness tests are performed by leveraging a Python script to automatically modify specific configurations of my Multi-Cloud environment, and Sysdig to verify the results obtained; such as the total amount of violations detected and the minimum, maximum, and average alerting times.

6.2 Multi-Cloud Infrastructure Validation

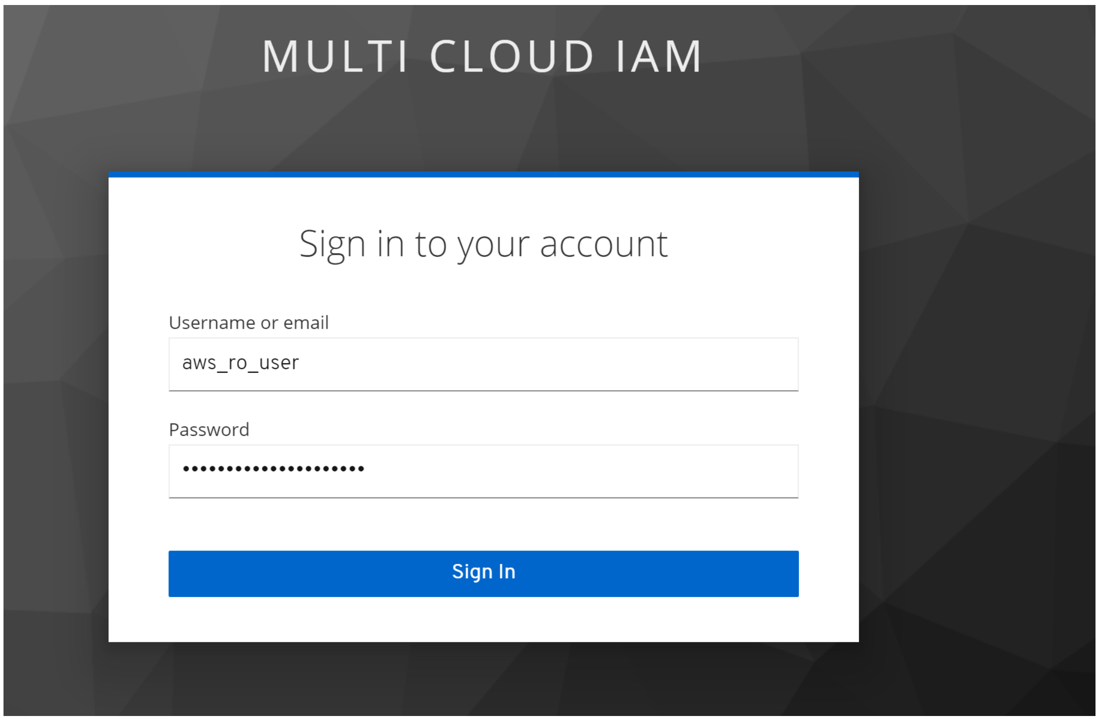
The following sections validate the proper functioning of the integrations proposed in the context of Identity and Access Management, Secrets Management, and Observability within the PoC Multi-Cloud infrastructure realized in Chapter 4. For each of the three integrations, a practical use case is presented accompanied by explanatory images to corroborate the work carried out in the previous chapters.

6.2.1 Multi-Cloud IAM Framework Validation

This validation aims to test the federation established between Keycloak acting as an Identity Provider for Reply's AWS Management Console. The proposed Business-to-Employee scenario shows how a company employee can use his Keycloak identity to obtain access to an AWS cluster with a well-defined set of permissions. To verify the correct functioning of such a use case I created two different Keycloak users:

1. **aws_ro_user**: User belonging to a group that only allows him to display information about AWS EC2 instances.
2. **aws_ec2_full_access_user**: User belonging to the group mentioned above and to an additional one that allows him also to create, modify, or delete AWS EC2 instances.

The two images below show respectively Keycloak login page filled with the credentials of *aws_ro_user* and the AWS account information we are displayed upon successful login and redirect to the AWS Platform.



aws_ro_user

••••••••••••••••••••

Figure 6.1: Keycloak Login Page

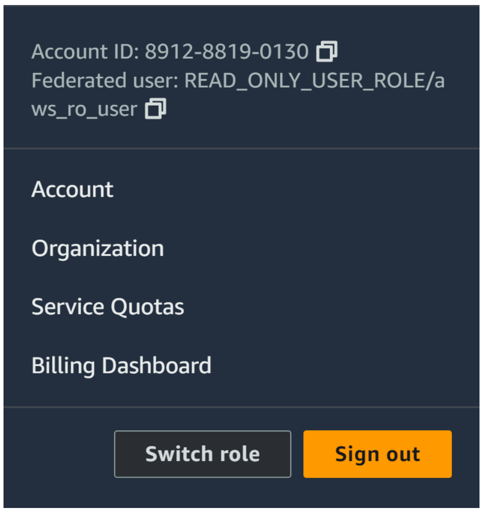
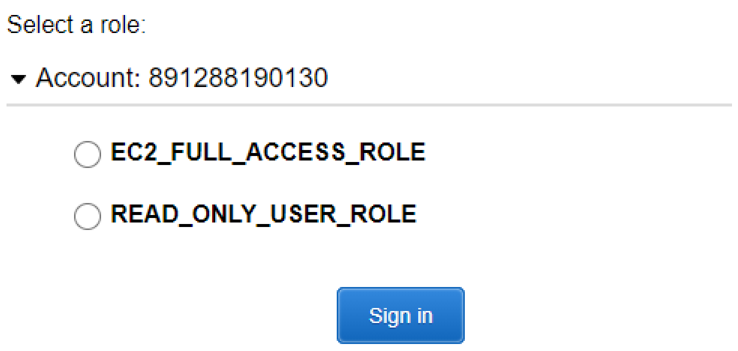

Account ID: 8912-8819-0130
Federated user: READ_ONLY_USER_ROLE/aws_ro_user

Figure 6.2: AWS Management Console Account

When a user accesses the *aws_ec2_full_access_user* account, which is associated with multiple Keycloak groups, the AWS Management Console presents the user with the opportunity to select a role to assume while working on AWS. This noteworthy feature, as illustrated in figure 6.3, facilitates the association of a single identity with various permissions sets, enabling the possibility to perform tasks with the least privilege required to carry them out.


Select a role:

▼ Account: 891288190130

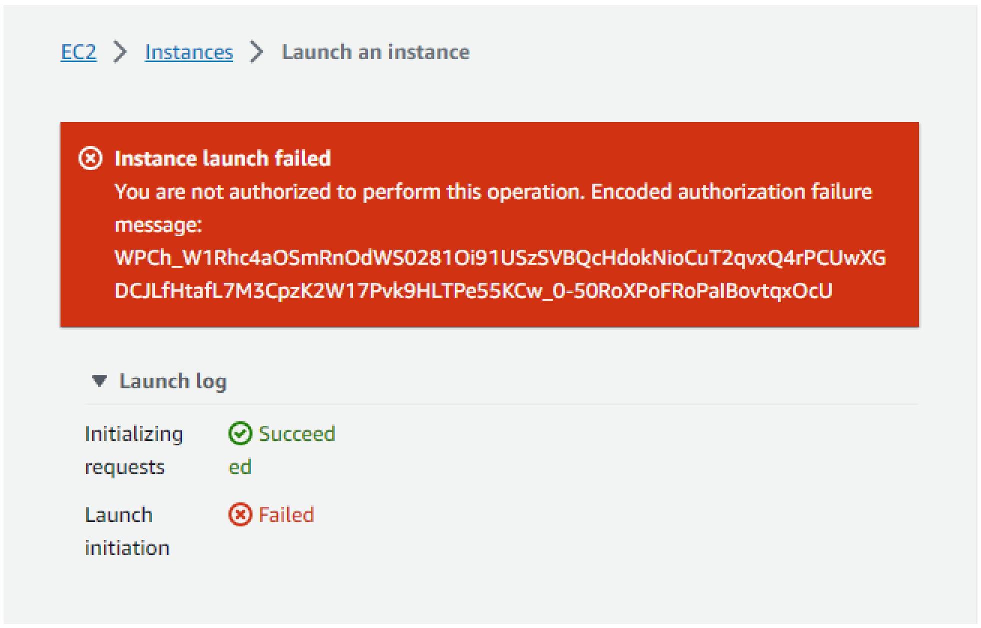
○ EC2_FULL_ACCESS_ROLE

○ READ_ONLY_USER_ROLE

Figure 6.3: AWS Management Console Role Selection

After verifying the functionality of delegated authentication enabled by Keycloak and AWS federation, I proceeded to confirm that the two showcased users were restricted to performing only specific actions on the AWS platform, thus adhering to Role-based Access Control principles. Initially, I accessed *aws_ro_user* account which is exclusively granted Read-Only permissions over EC2 instances, and attempted to launch a new container instance. Then I logged in *aws_ec2_full_access_user* account, chose **EC2_FULL_ACCESS_ROLE**, and tried to perform the same container launch action.

As anticipated, the first EC2 launch action is prevented by AWS, accompanied by an explicit lack of authorization warning (see Figure 6.4). In contrast, when that same instance launch was initiated by a user associated with EC2_FULL_ACCESS_ROLE role, it executed successfully without any issues (see Figure 6.5). These results underscore the effectiveness of adopting such an IAM framework for centralizing company employees' identity and permission management when dealing with one or more Cloud Service Providers.

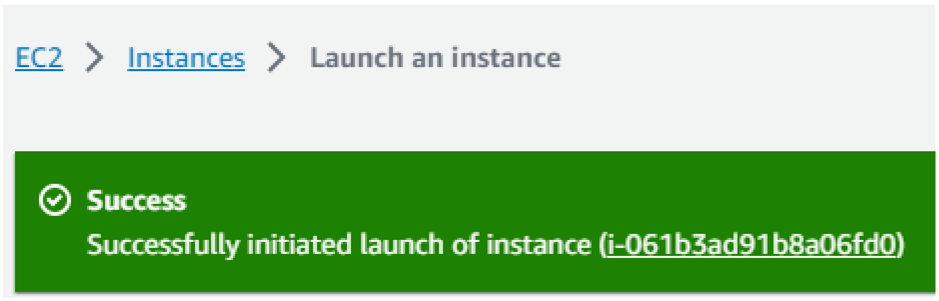

EC2 > Instances > Launch an instance

Instance launch failed
You are not authorized to perform this operation. Encoded authorization failure message:
WPCh_W1Rhc4aOSmRnOdWS0281Oi91USzSVBQcHdokNioCuT2qvxQ4rPCUwXG
DCJLfHtafL7M3CpzK2W17Pvk9HLTPe55KCw_0-50RoXPoFRoPalBovtqxOcU

▼ Launch log

Initializing requests	Succeeded
Launch initiation	Failed

Figure 6.4: EC2 Instance Launch Failed


EC2 > Instances > Launch an instance

Success
Successfully initiated launch of instance (i-061b3ad91b8a06fd0)

Figure 6.5: EC2 Instance Successfully Launched

6.2.2 Multi-Cloud Secrets Management Framework Validation

The objective of this Secrets Management Framework validation is to ensure the functionality of the integration discussed in section 4.4.4. This integration involves the configuration and connection of one or more Kubernetes clusters with an HCP Vault, and a MySQL database, enabling the following capabilities:

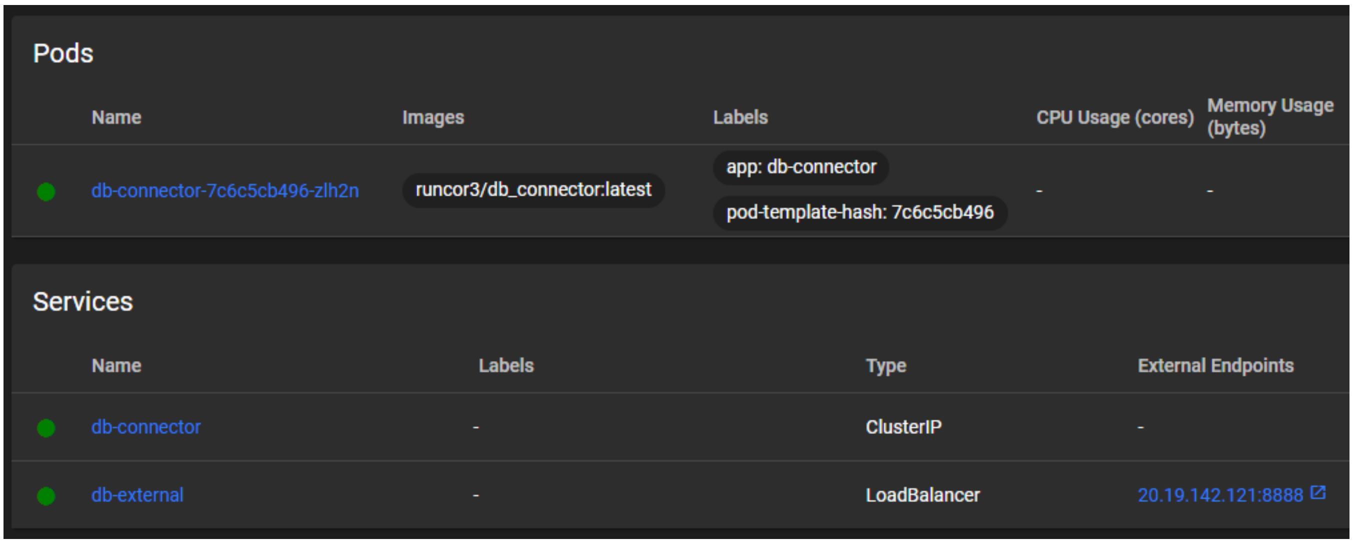
- Role-based Access Control for Kubernetes workloads seeking secrets from an HCP Vault
- Secret-less authentication for Kubernetes workloads with a MySQL Database, utilizing temporary credentials negotiated by HCP Vault.

In the scenario proposed there is a generic microservice, referred to as **DB Connector** as previously identified in figure 4.6. This microservice operates within an Azure Kubernetes Cluster and aims to authenticate itself and assume a specific role that allows it to query secrets from a remotely located HCP Vault solely using its Kubernetes Service Account. Once the Service Account's authenticity is verified by HashiCorp Vault, the Vault then interacts with a MySQL Database instance to generate a pair of temporary credentials. The microservice subsequently utilizes these credentials to perform CRUD operations on the database.

DB Connector is a Spring microservice that uses Spring Cloud Vault to interact with HashiCorp Vault, and Spring Data JPA to execute queries on a target MySQL Database. When the microservice starts, it attempts to authenticate to the Vault using the JWT token related to its Service Account. If the workload's identity is confirmed, the latter can subsequently request HCP Vault for a pair of credentials that can be used by Spring Data to access the MySQL Database.

Figures 6.6, 6.7a, and 6.7b show respectively:

- The **DB Connector** microservice running on my AKS cluster and reachable from the internet using a public IP address.
- The new **test_db** database, automatically created by Spring Data JPA upon successful authentication to HCP Vault.
- The two tables that have been created by *DB Connector* within *test_db* database.



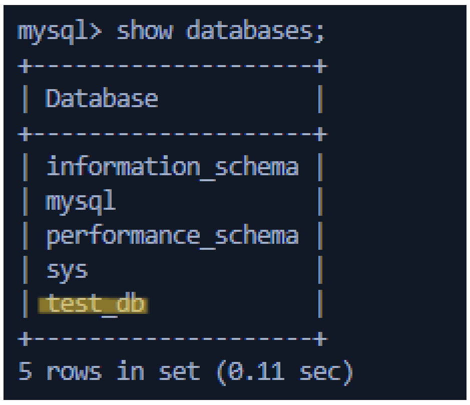
Pods

	Name	Images	Labels	CPU Usage (cores)	Memory Usage (bytes)
●	db-connector-7c6c5cb496-zlh2n	runcor3/db_connector:latest	app: db-connector pod-template-hash: 7c6c5cb496	-	-

Services

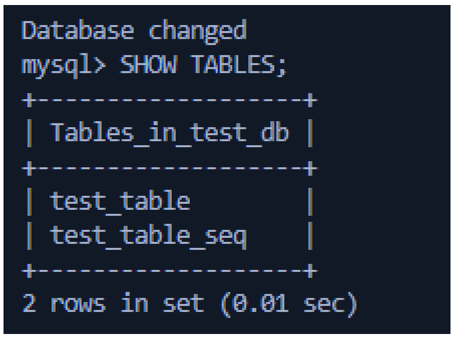
	Name	Labels	Type	External Endpoints
●	db-connector	-	ClusterIP	-
●	db-external	-	LoadBalancer	20.19.142.121:8888

Figure 6.6: DB Connector Microservice on Azure Kubernetes Cluster



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test_db            |
+--------------------+
5 rows in set (0.11 sec)
```

(a) MySQL Databases List



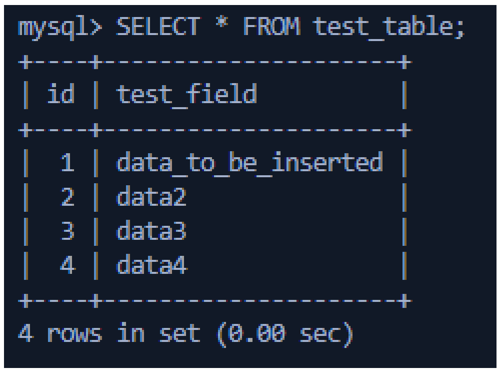
```
Database changed
mysql> SHOW TABLES;
+-------------------+
| Tables_in_test_db |
+-------------------+
| test_table        |
| test_table_seq    |
+-------------------+
2 rows in set (0.01 sec)
```

(b) test_db Tables List

Figure 6.7: Azure MySQL Database accessed by DB Connector microservice

In addition to automatically creating a new database and two tables, *DB Connector* microservice exposes two simple REST APIs for reading and writing data in these tables. This serves as additional evidence of the successful integration of the three entities discussed earlier. Executing the simple cURL command written below, which targets the microservice public IP address (see Figure 6.6), allows for the persistence and retrieval of new data from the MySQL Database, as confirmed by the results presented in Figure 6.8.

curl 20.19.142.121:8888/api/tests/data_to_be_inserted



```
mysql> SELECT * FROM test_table;
+----+---------------------+
| id | test_field          |
+----+---------------------+
|  1 | data_to_be_inserted |
|  2 | data2               |
|  3 | data3               |
|  4 | data4               |
+----+---------------------+
4 rows in set (0.00 sec)
```

Figure 6.8: test_db's Tables Data

Thanks to the interoperability of HashiCorp Vault, MySQL Databases, and Kubernetes Clusters, the secret management framework developed and tested in this dissertation demonstrates effective scalability and operation across two heterogeneous cloud environments. Furthermore, this framework significantly contributes to the implementation of Role-based Access Control for workloads and mitigates the need to hard-code secrets into application source code.

6.2.3 Multi-Cloud Observability Framework Validation

The Multi-Cloud Observability implementation discussed in section 4.4.6 represents the final integration within my infrastructure that remains to be verified. The objective of this validation is to ensure that data originating from various sources distributed across multiple cloud environments, is accurately collected, aggregated, and available for visualization through a unified analytics platform.

The tests conducted in this section were performed on the same infrastructure illustrated in Figure 4.8. This infrastructure consists of an Azure Grafana instance collecting data from Azure via a Log Analytics Workspace and from AWS through CloudWatch (as shown in Figure 6.9).

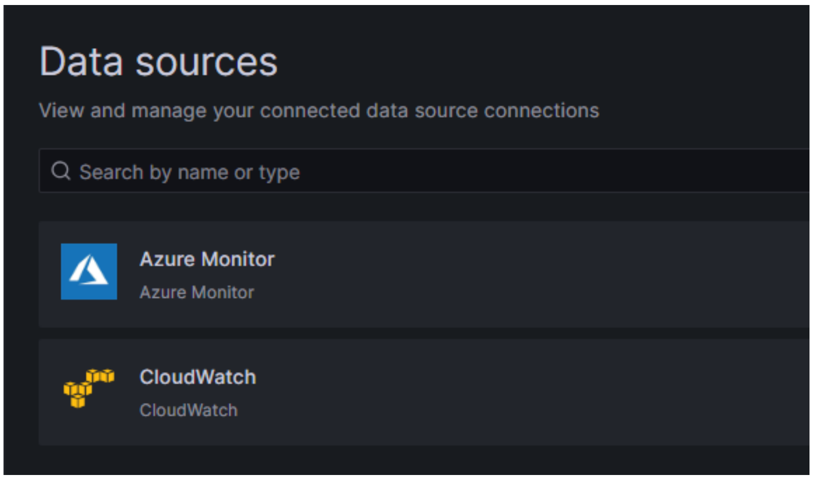


Figure 6.9: Grafana Data Sources

Azure Kubernetes Cluster Monitoring Test

After successfully deploying Grafana and configuring an Azure Kubernetes Cluster (AKS), I established a Grafana dashboard to collect a wide range of data from the aforementioned cluster. Figure 6.10 offers an overview of the cluster's CPU and Memory utilization during idle periods.

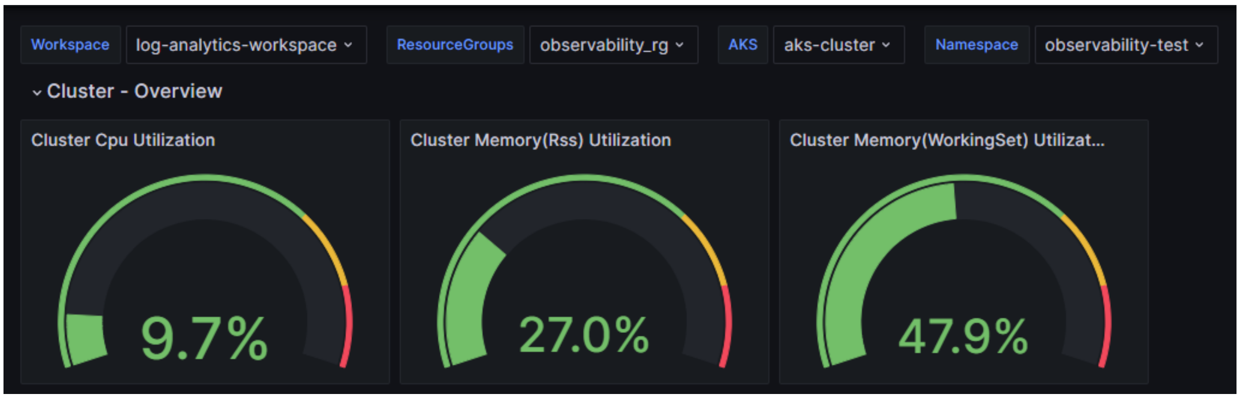


Figure 6.10: Grafana AKS Monitoring (Idle)

To validate the proper functioning of the observability framework for Azure data collection, I intentionally modified the state of my AKS cluster by deploying a substantial number of pods. As illustrated in Figure 6.11, this operation resulted in a noticeable spike in CPU utilization, accompanied by a moderate increase in Memory utilization. In Figure 6.12, we observe a histogram that unequivocally confirms the successful deployment of the pods within a dedicated Kubernetes namespace named **observability-test**. Together these two figures prove Grafana's ability to collect and visualize monitoring data from the Azure environment effectively.

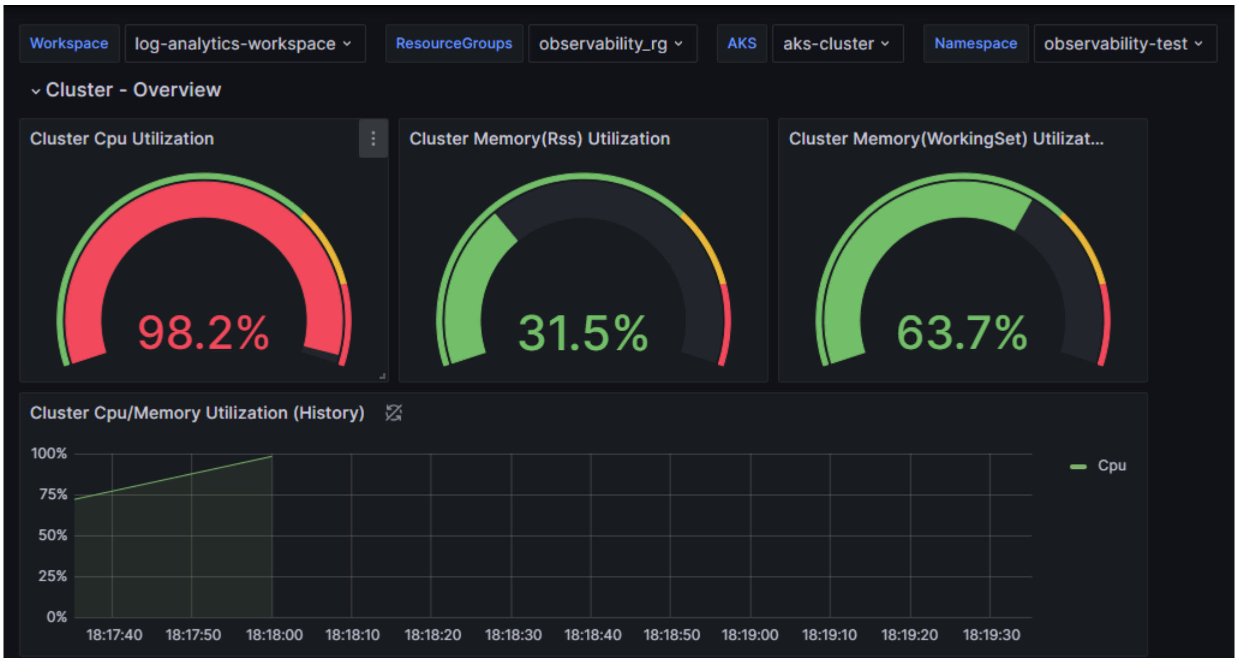


Figure 6.11: Grafana AKS Monitoring (In Use)

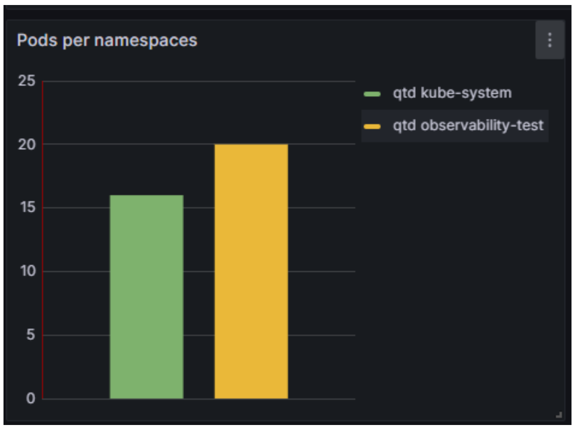
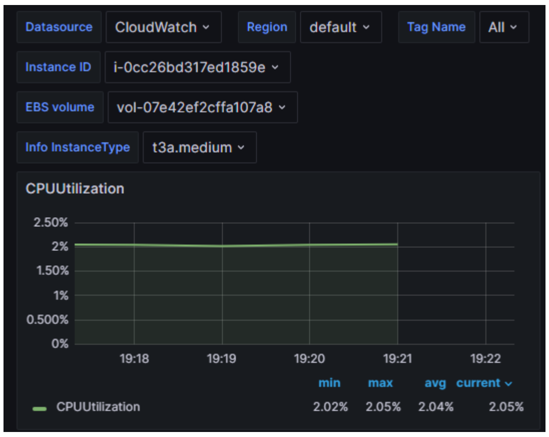


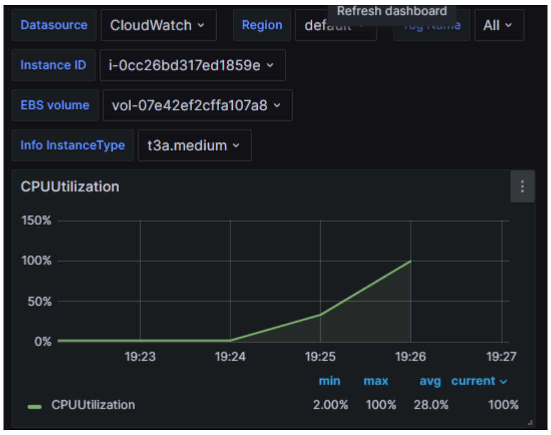
Figure 6.12: Grafana AKS Monitoring Pod Count

AWS Elastic Kubernetes Cluster Monitoring Test

To verify that AWS CloudWatch monitoring data were effectively being collected and visualized by Grafana, I've deployed an additional Kubernetes Cluster (EKS), this time on AWS, running on a single EC2 node. The two histograms depicted below respectively show the CPU utilization trend of the EC2 instance, before and after a consistent amount of pods were deployed on the EKS cluster.



(a) EC2 Instance CPU Utilization IDLE



(b) EC2 CPU Instance Utilization in Use

Figure 6.13: EC2 Monitoring Data

Further validation of the accuracy of AWS monitoring data collection and visualization is provided by the histogram in Figure 6.14. This histogram illustrates a brief rise in network utilization on the node, attributed to the updates automatically downloaded by each Pod.

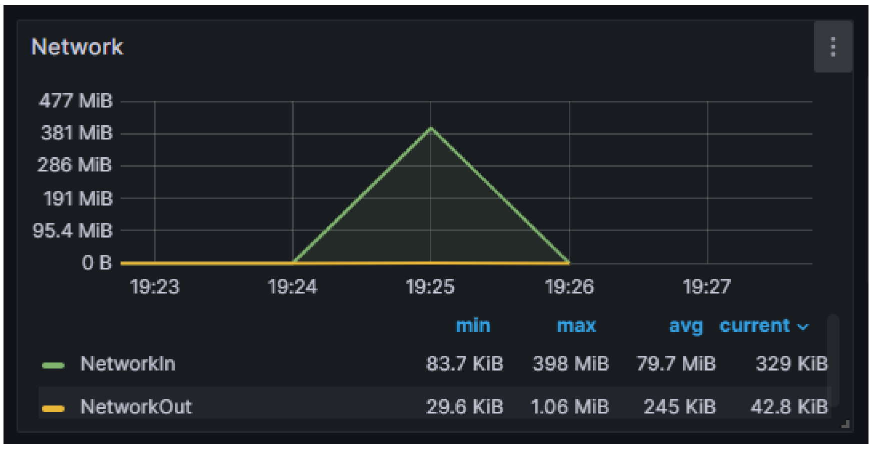


Figure 6.14: EC2 Network Utilization

6.3 Infrastructure-as-Code Security Results' Analysis

This evaluation rigorously assesses the efficacy of the two tools employed to ensure IaC Security. The examination centers around a set of Terraform and Kubernetes template files utilized for provisioning various components within the ultimate Multi-Cloud infrastructure. The data presented in the analysis stems from reports generated by the CI/CD pipeline and has been thoroughly examined to extract meaningful metrics. Each control result has been manually reviewed to categorize it as a: *true positive*, *true negative*, *false positive*, or *false negative*. A control result can be identified as:

- **True Positive:** a misconfiguration that effectively exists is identified.
- **True Negative:**: the absence of a misconfiguration is correctly identified as such.
- **False Positive:**: a misconfiguration that doesn't exist is identified.
- **False Negative:**: a misconfiguration that remains undetected by a tool.

6.3.1 Checkov Results' Evaluation

Table 6.1 shows the result of the classification.

Class	**Number of results**
True positives (TP)	240
False negatives (FN)	15
False positives (FP)	0
True negatives (TN)	1319

Table 6.1: Checkov Classification Results.

From the cardinality of the previous classes, I have derived two additional measures: the *"True Positive Rate"* (TPR) and the *"False Positive Rate"* (FPR), which respectively represent the probability that a control result is a true positive or a false positive. They can be calculated as follows:

$$TPR = \frac{TP}{TP + FN} \qquad (6.1)$$

$$FPR = \frac{FP}{FP + TN} \qquad (6.2)$$

Table 6.2 shows the value obtained for these metrics:

Metric	**Value**
TP rate	0.941
FP rate	0

Table 6.2: Checkov TPR and FPR values.

A visual representation of these two measures is depicted in Figure 6.15, where TPR and FPR respectively correspond to the X and Y coordinates of a point plotted in a two-dimensional space. As these values jointly represent probabilities, the X and Y axes are limited to continuous values within the range of 0 to 1.

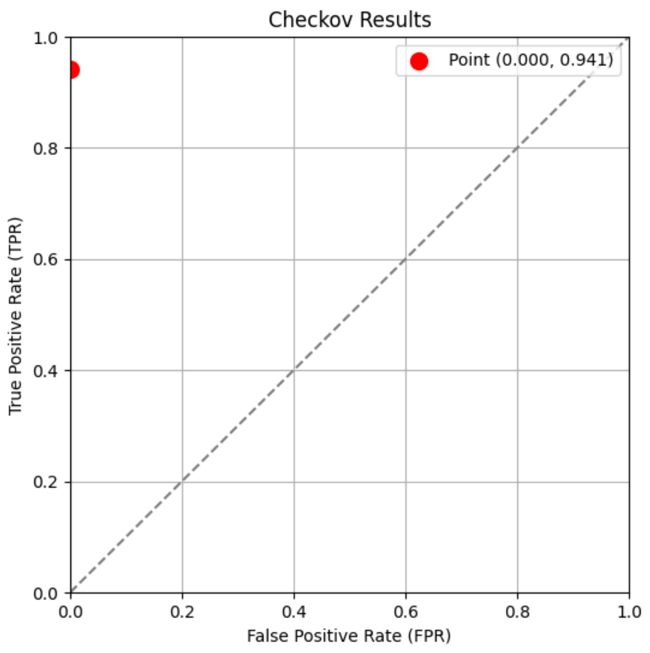


Figure 6.15: Checkov Results Graph

Tools are designed to maximize TPR while minimizing FPR; Therefore, the proximity of the point (FPR, TPR) to the ideal point (0, 1) on the graph indicates the accuracy of the results. When a tool detects misconfiguration randomly, the point (FPR, TPR) will fall along the dashed diagonal depicted in the graph. A point located above this diagonal signifies that the tool performs better than random chance in detecting misconfigurations. Conversely, a point located below this line suggests that the tool's performance in detecting misconfigurations is worse than random chance. The graph in Figure 6.15 showcases a very good result considering that point (0, 0.941) is above the diagonal and extremely close to the point (0,1)

6.3.2 Sysdig IaC-Sec Tool Results' Evaluation

The same metrics used to evaluate the results of Checkov's analysis have also been used in this section to review the results obtained by Sysdig's IaC-Sec tool.

Table 6.3 shows the result of the classification.

Class	**Number of results**
True positives (TP)	71
False negatives (FN)	33
False positives (FP)	5
True negatives (TN)	316

Table 6.3: Sysdig Classification Results.

Table 6.4 shows the value obtained for *True Positive* and *False Positive* rates:

Metric	**Value**
TP rate	0.682
FP rate	0.0155

Table 6.4: Sysdig TPR and FPR values.

Figure 6.16 shows the graphical representation of point (FPR, TPR) in a two-dimensional space.

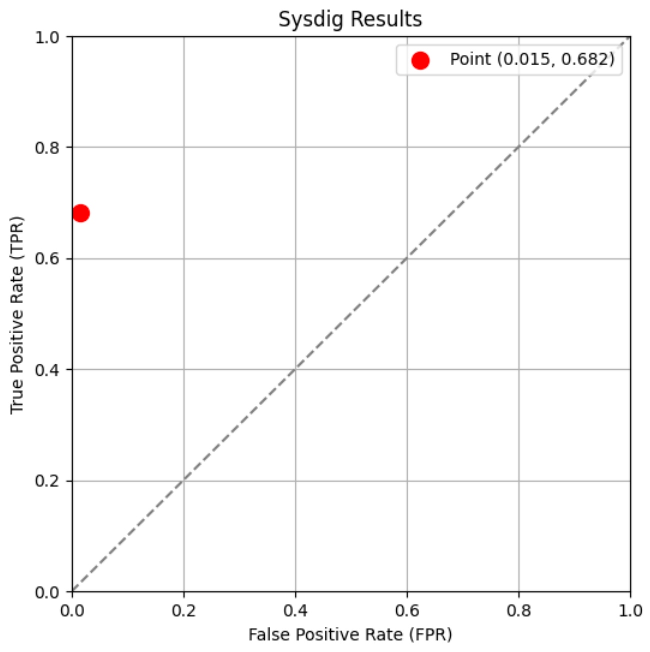


Figure 6.16: Sysdig Results Graph

Comparing with graph 6.15, it is evident that Sysdig's results exhibit a lower level of accuracy in contrast to Checkov's. Nonetheless, the plotted (FPR, TPR) point on graph 6.16 reaffirms an indication of the tool's overall quality, as it remains positioned above the diagonal line, as observed previously.

6.4 Cloud Security Posture Management Results' Analysis

To evaluate Sysdig's Cloud Security Posture Management (CSPM) capabilities, I decided to analyze the configurations of the two cloud platforms that were employed to deploy my Multi-Cloud infrastructure. This assessment was conducted following two widely recognized policies used in enterprises to establish a secure baseline configuration for Azure and AWS environments, respectively: CIS Azure Foundations Benchmark and CIS AWS Foundations Benchmark.

The data that is presented stems from the CSPM reports produced by Sysdig and analogously to the result's analysis carried out in section 6.3, the same measures have been computed and interpreted to assess Sysdig's CSPM efficacy.

Table 6.5 shows the overall result of the True positives, False negatives, False positives, and True negatives classification.

Class	**Number of results**
True positives (TP)	412
False negatives (FN)	4
False positives (FP)	7
True negatives (TN)	719

Table 6.5: CIS Azure and AWS Benchmarks Violations Results.

Table 6.6 presents the values for the *True Positive* and *False Positive* rates obtained from the previous classification.

Metric	**Value**
TP rate	0.99
FP rate	0.01

Table 6.6: Sysdig TPR and FPR values.

Figure 6.17 graphically shows the point with coordinates (FPR, TPR) in a two-dimensional space.

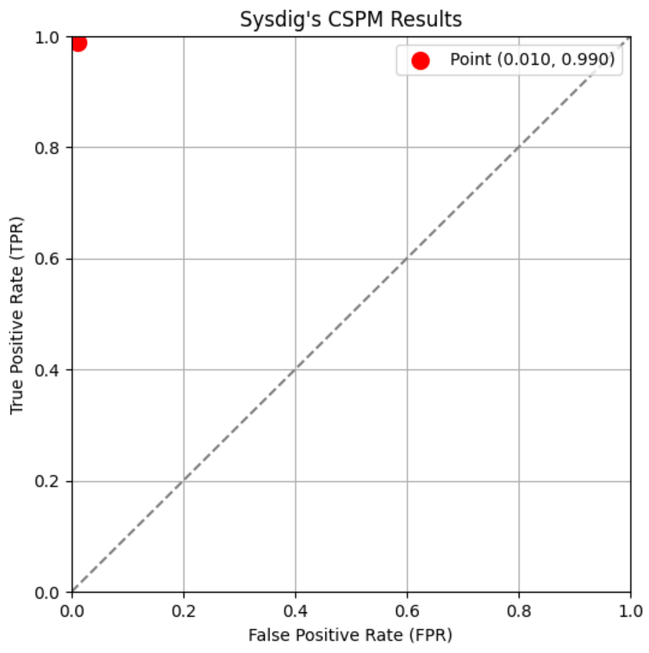
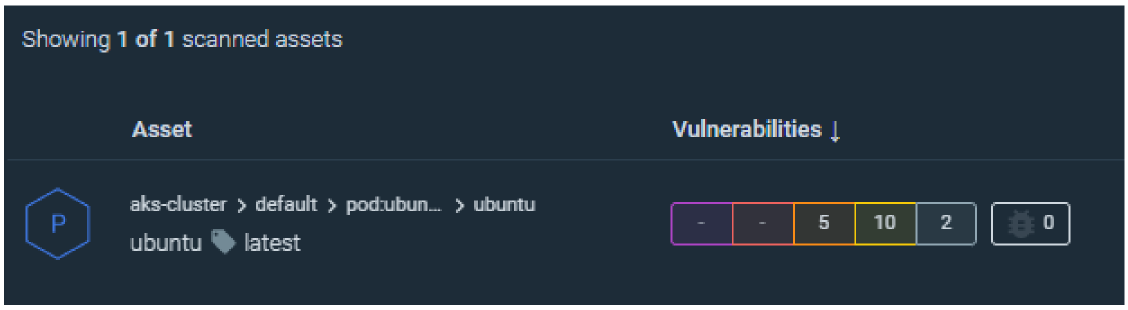


Figure 6.17: Sysdig's CSPM Results Graph

The graph depicted above exhibits nearly perfect *True Positive* and *False Positive* rates (0.99, 0.01 respectively). This last result highlights the quality of the tool in enforcing various policies through the *Policy-as-Code* paradigm, and in identifying misconfigurations present in multiple cloud environments.

6.5 Vulnerability Assessment Capabilities Testing

This test aims to assess Sysdig's capability to identify vulnerabilities in generic cloud-deployed hosts. The scenario involves an Azure Kubernetes Cluster initially subjected to a vulnerability scan. Figure 6.18 displays the vulnerabilities detected within the sole Pod operating in the Kubernetes *default* namespace. Given that this workload consists of an Ubuntu 20.04 machine, the initial scan revealed a limited number of package vulnerabilities, ranging from *informational* to *medium* threat levels.



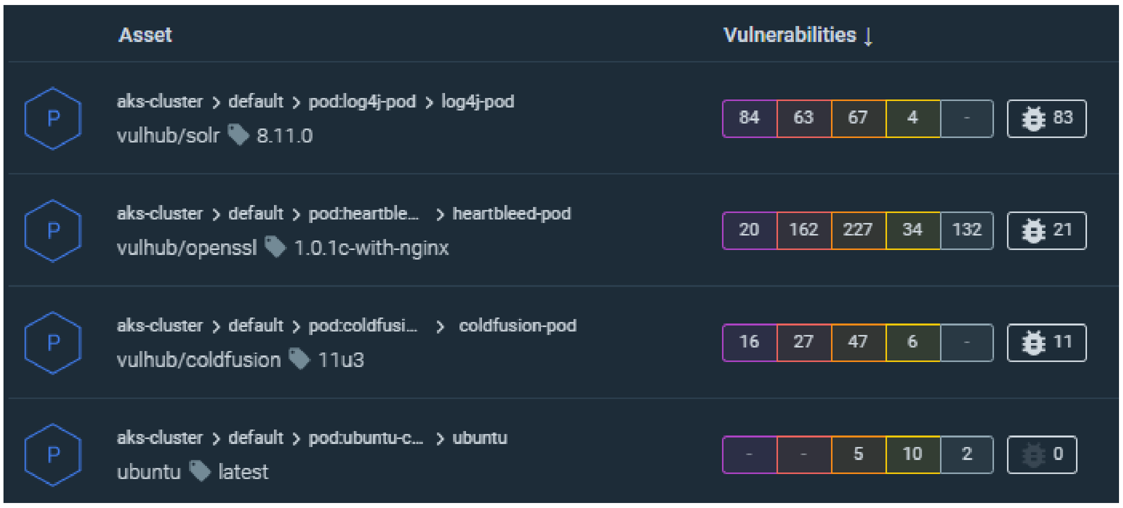
Showing **1 of 1** scanned assets

Asset	Vulnerabilities ↓
aks-cluster > default > pod:ubun... > ubuntu ubuntu latest	- | - | 5 | 10 | 2 | 0

Figure 6.18: Initial Pods Vulnerabilities Scan

After collecting the results of the initial vulnerability assessment conducted by Sysdig's agents, I subsequently deployed a set of new workloads within the Azure Kubernetes Cluster. These workloads included a large number of vulnerable packages, with Docker images obtained from **Vulhub** [22] Github repository, which provides a wide range of pre-built vulnerable environments based on Docker Compose. Specifically, the newly deployed Pods were known to contain critical vulnerabilities such as *Log4J* [17], *Coldfusion* [15] and *Heartbleed* [16]

Figure 6.19, presents the results of an additional vulnerability assessment on the same Kubernetes Cluster. Compared to the previous results, it's evident that the three vulnerable pods are now listed among the compromised workloads, with an extensive number of high and critical issues detected by Sysdig's agents. These findings further validate Sysdig CNAPP's capabilities in detecting vulnerabilities of heterogeneous workloads, complementing its demonstrated proficiency in Cloud Security Posture Management.



	Asset	Vulnerabilities ↓					
P	aks-cluster > default > pod:log4j-pod > log4j-pod vulhub/solr 8.11.0	84	63	67	4	-	83
P	aks-cluster > default > pod:heartble... > heartbleed-pod vulhub/openssl 1.0.1c-with-nginx	20	162	227	34	132	21
P	aks-cluster > default > pod:coldfusi... > coldfusion-pod vulhub/coldfusion 11u3	16	27	47	6	-	11
P	aks-cluster > default > pod:ubuntu-c... > ubuntu ubuntu latest	-	-	5	10	2	0

Figure 6.19: Final Pods Vulnerabilities Scan

6.6 Runtime Threat Detection Stress Test

This final test assesses Sysdig's Runtime Threat Detection capabilities within the context of detecting multiple rule violations occurring within a limited timeframe. The analysis involves executing potentially malicious actions on resources deployed in Azure and AWS environments, including Virtual Machines, Clusters, and Virtual Storage. Additionally, some of these actions were also performed on workloads running inside the Azure Kubernetes Cluster, as previously discussed in Section 4.4.2.

6.6.1 Experiment Setup

To set up this experiment, I initially defined a set of violations that will be intentionally executed across various workloads. These violations were required to align with the policies enforced by the Sysdig Runtime Threat Detection Engine. Listed below are the selected rules, along with the number of times they were violated during the execution of this experiment, and categorized by the environments in which they will be enforced.

Rule (Azure)	**Number of violations**
Allow RDP Access to VM	1
Allow SSH Access to VM	1
Create Azure Storage Account Accessible from the Internet	5
Create Azure Storage Account HTTP Accessible	5
Creation of a Blob within a storage account	5
Azure Storage Account Deleted	5
Azure Storage Container Deleted	5

Table 6.7: Falco Rules for Azure and Number of violations that were performed

Rule (AWS)	**Number of violations**
Create Access Key For User	5
Create IAM Policy that allows all	5
Create RDS with Public Access	5
Grant All Users Access to S3 Bucket	5
CloudTrail Trail Deleted	5

Table 6.8: Falco Rules for AWS and Number of violations that were performed

Rule (Kubernetes)	**Number of violations**
Attach/Exec to Pod	20
Create Privileged Pod	10
Create ClusterRoleBinding with cluster-admin	10
Delete Pod	10
Create Deployment	10
Delete Namespace	1

Table 6.9: Falco Rules for Kubernetes and Number of violations that were performed

6.6.2 Experiment Automation

Executing all the previously defined violations manually would have been both time-consuming and disruptive to the experiment's repeatability. To address this, I chose to develop a Bash script that utilizes the Azure CLI, AWS CLI, and Kubectl to automate CRUD (Create, Read, Update, Delete) operations across the three platforms.

The following code extracts provide examples of the operations that were automated to conduct the final runtime threat detection stress test.

Bash function to create an Azure Storage Account

```
function createStorageAccountContainer {
    containerName="containertest"

    az storage container create \
        --name $containerName \
        --account-name $storageAccountName \
        --account-key $storageAccountKey \
        --public-access blob

    echo "$(date +%r) Storage Account Container Created (Blob)" \
          >>operations_timestamps.md
    echo "Azure Storage Account Container '$containerName'
       created."
}
```

Bash function to create an Access Key for an AWS User

```
function createAndDeleteAWSAccessKey {
    IAM_USER_NAME=$1

    create_key_output=$(aws iam create-access-key
        --user-name "$IAM_USER_NAME")

    # Check the exit code to see if the command was successful
    if [ $? -eq 0 ]; then
    echo "$(date +%r) User Access Key Created" >>
        operations_timestamps.md
    echo "Access key created successfully for user
        $IAM_USER_NAME."
    else
        echo "Error creating access key for user $IAM_USER_NAME."
        echo "Program exits createAndDeleteAWSAccessKey: DELETE"
        exit 1
    fi

    ACCESS_KEY_ID=$(echo "$create_key_output" |grep -o
        '"AccessKeyId": "[^"]*' |awk -F'"' '{print $4}')

    sleep 15

    aws iam delete-access-key --user-name "$IAM_USER_NAME"
        --access-key-id "$ACCESS_KEY_ID"

    # Check the exit code to see if the command was successful
    if [ $? -eq 0 ]; then
    echo "Access key $ACCESS_KEY_ID deleted successfully for
        user $IAM_USER_NAME."
    else
        echo "Error deleting access key $ACCESS_KEY_ID for user
            $IAM_USER_NAME."
        echo "Program exits createAndDeleteAWSAccessKey: DELETE"
        exit 1
    fi
}
```

Bash function to deploy a Kubernetes Privileged Pod

```
function deployPrivilegedPod {
    kubectl apply -f manifests/privileged-pod.yml -n
        "$K8S_NAMESPACE"
    echo "$(date +%r) Privileged Pod Launched" >>
        operations_timestamps.md
    echo "Privileged Pod deployed successfully"


    kubectl wait pod/privileged-pod -n "$K8S_NAMESPACE"
        --for=condition=Running --timeout=40s
    echo "Privileged Pod is Running"

    kubectl delete pod privileged-pod -n "$K8S_NAMESPACE"
    echo "$(date +%r) Privileged Pod Deleted" >>
        operations_timestamps.md
    echo "Privileged Pod deleted successfully"
}
```

6.6.3 Experiment Results

The metrics that have been collected and evaluated to establish the outcome of the experiment and the efficacy of Sysdig's Runtime Threat Detection are:

- The number of violations detected by Sysdig, as opposed to the total number of violations executed.
- The minimum, maximum, and average times between the execution of a violation and its detection.

The following three tables present the results of the experiment, with each table dedicated to documenting violations in either Azure, AWS, or Kubernetes environment. These results were obtained after executing the previously mentioned automatic script and collecting all the relevant violations detected by Sysdig. Subsequently, there will be an interpretation and a comparison of the results.

Azure Runtime Threat Detection	
Total Violations Detected	20 out of 27
Min Detection Time	02:38 minutes
Max Detection Time	06:31 minutes
Average Detection Time	04:55 minutes

Table 6.10: Azure Runtime Threat Detection Results

AWS Runtime Threat Detection	
Total Violations Detected	25 out of 25
Min Detection Time	00:55 minutes
Max Detection Time	07:05 minutes
Average Detection Time	04:30 minutes

Table 6.11: AWS Runtime Threat Detection Results

Kubernetes Runtime Threat Detection	
Total Violations Detected	57 out of 61
Min Detection Time	00:02 minutes
Max Detection Time	01:55 minutes
Average Detection Time	01:30 minutes

Table 6.12: Kubernetes Runtime Threat Detection Results

In Table 6.10, 6.11, and 6.12, the results illustrate a significant contrast in the *Average Detection Time* for violations in Kubernetes environments and Azure and AWS Cloud Service Providers (CSPs). This discrepancy may be attributed to the difference in the scale and complexity of resources monitored in each environment. A Kubernetes Cluster inherently exhibits lower complexity compared to an entire Cloud Service Provider (CSP). As a result, Sysdig's Kubernetes agent is tailored to efficiently monitor this reduced set of resources, unlike other Sysdig agents, which are responsible for monitoring an entire CSP account. In terms of the number of detected violations, only AWS Sysdig's agent successfully identified and accurately transmitted all triggered events to Sysdig Secure's dashboard for display. In contrast, Kubernetes and Azure agents detected 93% and 74% of events, respectively. The relatively lower performance of Azure Runtime Threat Detection may be attributed to the integration capabilities of Azure Sysdig's agent with Microsoft's CSP, which Sysdig periodically enhances with the release of new agent versions. In conclusion, the high detection rate, with 102 out of 113 triggered events being successfully identified, along with the low average time between the occurrence of a violation and its detection, reaffirms the efficacy of adopting a tool like Sysdig Secure to ensure fast detection of runtime attacks across multiple cloud environments.

Chapter 7

Conclusions and Future Works

This thesis demonstrates the implementation of security measures across multiple dimensions within a Proof-of-Concept Multi-Cloud Infrastructure, adopting a Cloud Native Application Protection Platform (CNAPP) and embracing the Policy-as-Code paradigm. These security measures include aspects such as Cloud Security Posture Management, Cloud Infrastructure Entitlements Management, Cloud Workload Protection, Infrastructure-as-Code Security, and Runtime Threat Detection.

Initially, this work provided an introduction to fundamental theoretical concepts, including Cloud Computing, Cloud Governance, and key principles of Cloud/Multi-Cloud Security. This was followed by a literature review of Security-as-a-Service solutions proposed by researchers to safeguard cloud-based assets. Chapter 3 delved into the Policy-as-Code paradigm, offering a theoretical perspective on the various Cloud Security measures mentioned above, alongside an exploration of the DevSecOps approach in the context of Infrastructure-as-Code Security.

Chapter 4 thoroughly covers the realization of the Proof-of-Concept Multi-Cloud infrastructure, providing a comprehensive description of the integrations in the domains of Identity and Access Management, Secrets Management, and Observability. This Use-Case infrastructure was seamlessly integrated with a Cloud Native Application Protection Platform to enhance security across various dimensions.

Utilizing Sysdig Secure CNAPP was essential in uncovering misconfigurations and vulnerabilities in cloud workloads, including issues in IaC templates, package vulnerabilities on hosts, and runtime threats within cloud providers' platforms and Kubernetes clusters. Without this tool, these issues might have otherwise remained unnoticed.

Finally, a validation of the Multi-Cloud infrastructure's features and an evaluation of Checkov and Sysdig Secure capabilities was carried out.

For Multi-Cloud Infrastructure validation, I leveraged three specific use case scenarios to confirm the functionality of the integrations mentioned earlier.

The evaluation of IaC Security with Checkov and Sysdig involved analyzing the misconfigurations detected by the two tools, which initially assessed a set of Terraform and Kubernetes template files. Subsequently, I verified the accuracy of these misconfiguration reports generated by the two tools, by determining which identified misconfigurations were valid and which were not.

Similar to the evaluation of IaC Security, Sysdig's Cloud Security Posture Management was also assessed by analyzing the compliance issues identified by the tool and verifying their validity, distinguishing between those that were genuine and those that were not, while enforcing separate CIS benchmarks for both Azure and AWS platforms.

To conclude the evaluation of Sysdig, two additional experiments were conducted: The first experiment aimed to validate Sysdig's capability to detect the installation of vulnerable packages within generic hosts. The second experiment assessed Sysdig's Runtime Threat Detection in a scenario where an attack targeted multiple resources.

These tests confirmed the successful operation of the deployed Multi-Cloud infrastructure and demonstrated the effectiveness of the Checkov IaC Security tool and Sysdig CNAPP in detecting a wide range of threats with the potential to disrupt enterprise cloud environments.

7.1 Future Works

First, it is worth considering the finalization of a Multi-Cloud Identity and Access Management strategy, which was previously hindered by certain Azure Active Directory limitations. This objective can be achieved by selecting an Identity Provider that facilitates the centralization of user identities across cloud platforms. One potential candidate for this role is Azure Active Directory, which would replace Keycloak and serve as the unified Identity Provider for the entire Multi-Cloud infrastructure.

Furthermore, considering Sysdig's compatibility with on-premise hosts, expanding this Multi-Cloud infrastructure to a Hybrid Cloud setup could provide opportunities for implementing additional integrations and evaluating CNAPP's capabilities in a more complex environment.

Finally, exploring agentless workload protection techniques shows potential for simplifying workloads' deployment, reducing the need for maintaining numerous agents, and lowering overall cloud-related costs for companies.

Bibliography

[1] Mohamed Almorsy, John C. Grundy, and Ingo Müller. «An Analysis of the Cloud Computing Security Problem». In: *CoRR* abs/1609.01107 (2016). arXiv: `1609.01107`. URL: `http://arxiv.org/abs/1609.01107`.

[2] Amazon. *Amazon Encryption Service.* http://docs.aws.amazon.com/AmazonS3/latest/dev side-encryption.html.

[3] Mattia Caracciolo. *Policy as Code, How to Automate Cloud Compliance Verification with Open Source Tools.* https://webthesis.biblio.polito.it/26908/.

[4] Checkpoint. *Checkpoint 2022 Cloud Security Report.* https://pages.checkpoint.com/2022-cloud-security-report.html.

[5] Maurizio Colombo et al. «Data Protection as a Service in the Multi-Cloud Environment». In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD).* 2019, pp. 81–85. DOI: `10.1109/CLOUD.2019.00025`.

[6] Gartner. *Gartner CNAPP Rating.* https://www.gartner.com/reviews/market/cloud-native-application-protection-platforms.

[7] Gartner. *Gartner Multi-Cloud Usage.* https://www.gartner.com/en/conferences/apac/infra operations-cloud-india/featured-topics/cloud.

[8] Google. *Google Cloud Storage.* http://googlecloudplatform.blogspot.co.uk/2013/08/google-cloud-storage-now-provides.html.

[9] Google. *Google Online Boutique Microservices Demo.* https://github.com/GoogleCloudPla demo.

[10] HashiCorp. *HCP Vault Auto-Unseal using Azure Key Vault.* https://developer.hashicorp.co unseal/autounseal-azure-keyvault.

[11] HashiCorp. *HCP Vault Centralized Secrets Management.* https://www.hashicorp.com/resou the-cloud-operating-model-security.

[12] HashiCorp. *HCP Vault Integrations.* https://www.hashicorp.com/blog/hashicorp-vault-surpasses-100-integrations-with-75-partners.

[13] IBM. *Cloud Security.* https://www.ibm.com/topics/cloud-security.

[14] Palo Alto Networks. *Policy as Code.* https://www.paloaltonetworks.com/cyberpedia/what-is-policy-as-code.

[15] NIST. *Coldfusion CVE-2023-29300.* https://nvd.nist.gov/vuln/detail/CVE-2023-29300.

[16] NIST. *Heartbleed CVE-2024-0160.* https://nvd.nist.gov/vuln/detail/cve-2014-0160.

[17] NIST. *Log4J CVE-2021-44228.* https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

[18] Oracle. *Oracle Multi-Cloud.* https://www.oracle.com/lu/news/announcement/98-percent-enterprises-adopted-multicloud-strategy-2023-02-09/.

[19] Pramod S. Pawar et al. «Security-as-a-Service in Multi-cloud and Federated Cloud Environments». In: *Trust Management IX*. Ed. by Christian Damsgaard Jensen et al. Cham: Springer International Publishing, 2015, pp. 251–261. ISBN: 978-3-319-18491-3.

[20] Statista. *AWS Origins.* https://www.statista.com/topics/4418/amazon-web-services.

[21] TrendMicro. *Misconfigurations in Cloud Environments.* https://www.trendmicro.com/vinfo
and-cloud/the-most-common-cloud-misconfigurations-that-could-lead-to-security-breaches.

[22] Vulhub. *Vulhub Pre-Built Vulnerable Environments Based on Docker-Compose.* https://github.com/vulhub/vulhub.