Research on Security Challenges in Cloud Environments and Solutions based on the "Security-as-Code" Approach

Oleksandr Vakhula[1], Ivan Opirskyy[1], and Olha Mykhaylova[1]

[1] *Lviv Polytechnic National University, 5 Knyaz Roman str., Lviv, 79013, Ukraine*

Abstract

"Security as code" is an approach to security organization in cloud environments, which is based on the method of integrating security controls, policies, and best practices directly into the software development and deployment processes. The integration process includes the transformation of security requirements and configurations into software code, which in turn is considered an integral part of the full software development life cycle. By embedding security measures into code, scripts, templates, and automated workflows, an organization ensures that there are well-defined security controls that will be consistently enforced across all operational phases of software creation (development, testing, implementation, and support). This article examines the main problems of building security in cloud environments and their causes, also considers the components and principles of the "Security as Code" approach, implementation examples with an explanation, of the advantages of this approach, as well as the role of DevSecOps. This article aims to help readers understand the importance of the security-as-code approach as one of the most effective methods for managing security in cloud environments. As cloud environments continue to evolve and proliferate, and threats become more sophisticated, the Security as Code approach represents a core strategy for proactively protecting digital assets. This publication serves as a guide to understanding, implementing, and benefiting from a security-as-code approach, providing insight into the future cloud security landscape and the critical role of automation and integration in addressing today's security challenges. To support the research, an extensive review of literature and articles providing information on the Security as Code approach and its application was conducted.

Keywords

Security as code, Infrastructure as code, DevSecOps, DevOps, cloud environments, cloud service provider, software development cycle, cloud security threats, shift-left security approach.

1. Introduction

In cloud computing, which is constantly evolving and combining flexibility and innovation, the importance of robust security measures cannot be overstated. As organizations continue to harness the transformative potential of cloud technologies, the need to protect digital assets from a growing spectrum of threats becomes not just a priority but a strategic imperative [1–2].

"Security as Code", born at the intersection of cybersecurity and software development, represents a paradigm shift in how organizations conceptualize, implement, and maintain their security strategies in cloud environments. This approach encapsulates the fusion of security principles into code, creating a proactive, automated, and integrated security ecosystem seamlessly aligned with modern development methodologies [3–4].

The authors examine the fundamental security challenges faced by consumers of

CPITS-2023-II: Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2023, Kyiv, Ukraine
EMAIL Oleksandr.p.vakhula@lpnu.ua (O. Vakahula); ivan.r.opirskyi@lpnu.ua (I. Opirskyy); mykhaulovaolga1@gmail.com (O. Mykhaylova)
ORCID: 0009-0008-5367-3344 (O. Vakahula); 0000-0002-8461-8996 (I. Opirskyy); 0000-0002-3086-3160 (O. Mykhaylova)

cloud services. Root causes include a lack of understanding of the shared responsibility model, which is foundational; the dynamic and scalable nature of the environment, unlike traditional on-premises infrastructure; inadequate visibility of resources or "shadow IT;" underestimating risks associated with APIs; the complexity of navigating data in a distributed environment, including sensitive data; manual configuration settings and the high likelihood of errors due to human factors; the complexity of Identity and Access Management (IAM) services; multi-cloud and hybrid environments; and the shortage of qualified cloud security professionals—demand outweighs supply [5].

This publication aims to highlight the aforementioned problems, actuality, and their root causes, and to explore the "Security as Code" approach, which can help to solve part of it and mitigate risks related. A lot of articles point out that DevOps practitioners degrade the priority of security since the regard security is the biggest hurdle to rapid application development considering traditional security methods do not fit the pipeline and are an inhibitor to DevOps agility [6].

Traditionally, security measures are typically addressed after the development team has completed the product. This approach often results in a backlog of challenging bugs to fix. The project manager may think, "If we implement all these fixes, we'll be delayed, and the company won't be pleased. Let's put it off until the next iteration" [7–8].

As an illustration, consider a scenario where a product manager wants to grant customers access to certain data without requiring any form of authentication. In the past, the security team has consistently rejected such requests. However, with the implementation of DevSecOps, the response shifts to, "Yes, you can provide this access, but it must be done in a secure manner" [9]. In many instances, in pursuit of business agility and velocity, essential security aspects are overlooked in operational applications. Security is often relegated to the final check, conducted after the application is fully developed. In practical terms, ensuring security with each iteration can be a considerable challenge, both in terms of time and financial resources, unless it is deliberately incorporated into the early stages of the DevOps workflow [10].

The latest research in this field shows, that, Security as Code is the driving force behind future application security. According to O'Reilly, Security as Code is a way to build Security by mapping how code and infrastructure change to DevOps tools and workflows and finding places to add security controls, tests, and ports without cost or delay. Developers can define the infrastructure using a programming language with Infrastructure as Code. You need to do the same to bring security up to DevOps speed [11].

All of the above allows us to assert the relevance of the issue and calls for proposals on its resolution. In this publication, we will focus on the active security approach as a form of security code that can be considered preventive security control.

2. Challenges in Organizing Security in Cloud Environments

2.1. Shared Responsibility Model, Leading to Confusion over Security Responsibilities

The problem of cloud providers following a shared responsibility model, leading to confusion over who is responsible for securing what, is a crucial aspect of cloud security that organizations must address. In cloud computing, the shared responsibility model is a widely accepted framework that defines the division of security responsibilities between the Cloud Service Provider (CSP) and the cloud customer (organization using the cloud services). The exact responsibilities assigned to each party can vary depending on the type of cloud service, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS).

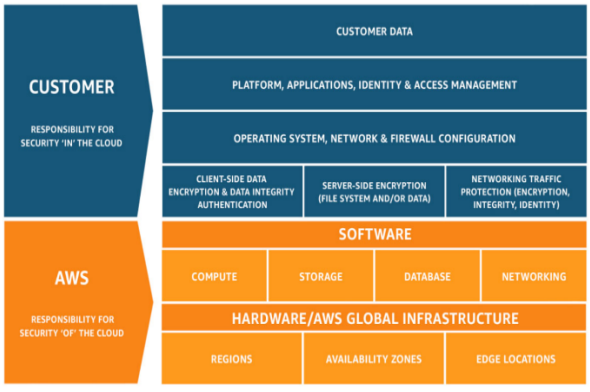

Figure 1: AWS Shared Responsibility Model [12]

Cloud providers, like AWS, Azure, and Google Cloud, are responsible for the security of the underlying cloud infrastructure, including the physical data centers, networking, and the hypervisor layer. They also typically provide security features and controls related to the overall cloud platform's integrity and availability.

On the other hand, cloud customers (organizations) are responsible for securing their data, applications, configurations, and access controls within the cloud environment. This includes securing virtual machines, containers, databases, and any other resources they deploy on the cloud platform. Customers are also responsible for managing user access and permissions, implementing encryption, and configuring security settings specific to their cloud resources.

The challenge arises when there is a lack of clarity or understanding about where the CSP's responsibilities end and the customer's responsibilities begin.

2.2. Lack of Visibility—Inadequate Insight into Cloud Environments

The cloud environment, by its nature, is complex and consists of numerous services, components, containers, and microservices distributed across different regions. This distributed multi-component structure creates a vast attack surface, making it extremely important to maintain complete visibility of all assets within the cloud ecosystem.

Traditional security tools designed for on-premises environments find it challenging to adapt to the dynamic cloud landscape. The traditional concept of a "perimeter" lacks clear boundaries, complicating the monitoring and protection of interactions between various components.

The lack of visibility leads to "blind spots" where security teams cannot effectively monitor and detect events in cloud resources. Configuration errors, anomalous behavior, unauthorized access, and potential breaches can go unnoticed, putting confidential data and critical business services at risk.

Detecting incidents, indicators of compromise, identifying the root cause of an incident (the so-called "patient zero"), tracking the spread, and containment become challenging without comprehensive monitoring of the entire cloud ecosystem.

Compliance with standards is a crucial requirement for organizations. The absence of visibility complicates the ability to demonstrate compliance with standards to auditors and regulatory bodies, potentially resulting in fines and reputational damage.

2.3. Complex IAM—Ensuring Comprehensive Identity and Access Management Across Multiple Cloud Services

Modern cloud environments encompass a wide range of services, each with its own set of user accounts, access control mechanisms, and authorization systems. These services can cover infrastructure resources, applications, databases, and more, and are often provided by various cloud providers.

Each cloud provider typically maintains its repository of identity data, which stores user information, account details, and access policies. This diversity of identity data repositories creates what is known as a user identity data silo and complicates the task of unified identity management across all providers and services.

In multi-cloud environments, users and applications often require interaction between different services. Managing access and permissions necessary for these interactions can quickly become complex, leading to errors, misconfigurations, and security gaps.

The vast number of permissions and roles that need to be defined, managed, and reviewed increases the likelihood of errors and oversight.

Complex access management scenarios amplify security risks. Users may be granted excessive permissions or incorrect configurations may inadvertently provide unauthorized access to confidential data. These vulnerabilities can be exploited by malicious actors to gain unauthorized access.

2.4. Security Configuration Management—Navigating the Complexity of Consistent Security Configurations in Cloud Services

As organizations transition to cloud environments, the management of security configurations becomes a paramount concern. Cloud services offer unprecedented flexibility and agility, allowing resources to be provisioned, modified, and decommissioned rapidly. However, this dynamic nature introduces a significant challenge: ensuring consistent and robust security configurations across the multitude of services, instances, and platforms that constitute a modern cloud ecosystem.

Cloud environments are designed for agility, with resources being created, scaled, and terminated dynamically. This dynamism accelerates development and deployment but complicates the task of maintaining consistent security settings.

In the cloud, security misconfigurations are a leading cause of data breaches and cyber incidents. A single misconfigured security group, firewall rule, or access policy can expose sensitive resources to unauthorized access.

Modern cloud environments offer a bunch of services, each with its security controls, access mechanisms, and configuration options. Securing virtual machines, databases, serverless functions, and containers requires mastering different configurations.

Multi-cloud and hybrid cloud strategies often involve services spread across different cloud providers, regions, and accounts. Ensuring consistent security configurations across this scale is a formidable task.

As cloud resources evolve, security configurations can drift away from best practices or organizational policies. Manual interventions and updates can lead to deviations from desired security settings.

Meeting regulatory requirements and industry standards demands consistent security configurations. Failing to maintain these configurations can result in compliance violations and legal consequences.

2.5. API Security—Safeguarding Cloud Environments from Vulnerabilities in APIs

Cloud services heavily rely on Application Programming Interfaces (APIs), which can be vulnerable to attacks. Very often, security engineers underestimate this vector.

Application Programming Interfaces serve as a crucial link facilitating interactions between cloud services. This technology allows developers to access cloud resources, manipulate data, and execute functions remotely. While this streamlined interaction enhances efficiency, it also exposes APIs to potential security risks.

Because APIs facilitate communication between various components, they can become entry points for attackers. Weaknesses in API design, implementation, or authentication can be exploited for unauthorized access, injection attacks, or data breaches.

Common vulnerabilities that can harm APIs in cloud environments include:

Injection Attacks: Insufficient input data validation can lead to injection attacks where malicious code or commands are inserted into the input.

Broken Authentication: Weak authentication mechanisms or improper session management can allow unauthorized access to APIs.

Insecure Deserialization: Mishandling serialized data can result in remote code execution.

Inadequate Authorization: Flaws in access control mechanisms can permit users to perform actions they are not authorized for.

Exposure of Sensitive Data: Mishandling of data or improper encryption can lead to the leakage of sensitive information.

In multi-cloud and hybrid cloud environments, third-party developer APIs further complicate the security landscape. Organizations often rely on external APIs for specialized services, expanding the attack surface.

2.6. Data Protection and Compliance Challenges Arising from Dispersed Cloud Data

Cloud environments offer flexibility, allowing organizations to distribute data among different services, regions, and even multiple cloud providers. Data can be stored in databases, file systems, object stores, and more, encompassing a wide spectrum of cloud resources.

Effective data protection requires encryption both at rest and during transmission and processing. However, different cloud services may employ various encryption methods, key management techniques, and security levels. Managing encryption in these services can be complex.

Managing access control and permissions for decentralized data is a challenging task. Improperly configured access control can lead to unauthorized access, data leaks, and compliance violations.

In multi-cloud environments where data can be stored on various cloud platforms, compliance with regulatory standards becomes even more challenging.

Compliance with data residency and jurisdiction rules poses a complex challenge. Ensuring data storage and processing within the legal boundaries of relevant regulations can be problematic when data is distributed across cloud services with different geographical locations.

2.7. Multi-Cloud and Hybrid Environments—Navigating Complex Security Management Across Diverse Platforms

Multi-cloud and hybrid environments, where multiple cloud providers are used, each with different services, interfaces, and security paradigms, multiply the complexity of security management.

Each cloud platform can become a silo of security practices, making it challenging to maintain consistency in security policies, access controls, and threat detection mechanisms.

Effective security management often requires specialized knowledge for each of the cloud providers. Teams must understand the nuances of security features and configurations for each platform.

Consistent threat detection and response processes in multi-cloud environments pose a challenge for security teams. Different monitoring tools and mechanisms complicate the standardization of threat detection procedures and incident response.

In hybrid environments, where data moves between on-premises infrastructure and multiple cloud platforms, data protection and secure data transfer become even more complex due to a lack of complete visibility.

2.8. Lack of Cloud Security Expertise—Confronting the Challenge of Insufficient Cloud Security Knowledge

The rapid evolution of cloud computing has revolutionized the way organizations operate, but it has also exposed a critical challenge: the scarcity of cloud security expertise. As businesses transition to cloud environments, they often find themselves grappling with the complexities of securing these dynamic and distributed systems. The shortage of skilled professionals who possess the necessary cloud security knowledge presents a significant obstacle to achieving robust cloud security practices.

Cloud security is a specialized domain that demands an understanding of both traditional cybersecurity principles and the unique intricacies of cloud platforms. Rapid technological advancements continually reshape the threat landscape, necessitating constant learning and adaptation.

Cloud environments encompass an array of services, each with its own security controls, configurations, and best practices. Securing virtual machines, containers, serverless functions, and data stores requires expertise that spans a wide spectrum of cloud services.

The demand for cloud security experts outpaces the available talent pool. Organizations struggle to find and retain professionals with the necessary skills to architect, implement, and manage robust cloud security measures.

In the absence of cloud security expertise, misconfigurations become a common risk. Poorly configured security settings can inadvertently expose sensitive data, increase

attack surfaces, and compromise the overall security posture.

Effective threat detection and incident response in cloud environments require specialized knowledge. Identifying and responding to cloud-specific threats and vulnerabilities requires understanding the nuances of cloud operations.

Different cloud providers offer distinct security features, tools, and practices. Cloud security experts must navigate these nuances to implement consistent security measures across diverse platforms.

3. "Security as a Code" Approach for Cloud Environments

Considering all the problems mentioned above, which can sometimes be a hindrance to organizations migrating to the cloud,— "Security as code" (SaC has been the most effective approach to securing cloud workloads with speed and agility. At this point, most cloud leaders agree that Infrastructure as Code (IaC) allows them to automate the building of systems in the cloud without error-prone manual configuration. SaC takes this one step further by defining cybersecurity policies and standards programmatically, so they can be referenced automatically in the configuration scripts used to provision cloud systems and systems running in the cloud can be compared with security policies to prevent "drift" [13]. To successfully implement the "Security as Code" approach, we need a comprehensive cloud strategy that also works as code. The fundamental idea is that we cannot secure something using the "Security as Code" approach if it's not implemented as code.

Most consumers of cloud services agree that "Infrastructure as Code" (IaC) allows for the rapid deployment of services in the cloud without manual configuration and, consequently, errors. "Security as Code" takes this approach further by defining security policies, standards, and best practices programmatically so that they can be used by default in configuration scripts used to set up cloud services and systems. IT departments can transition from the eternal balance between business flexibility and security to the realization that these elements can be combined to provide an adequate level of both without sacrificing either.

Let's consider a simplified example (Fig. 2): organizational policies contain a list of required security controls. Controls are broken down into rules, which are transformed into code that is understandable by a Centralized Compliance Check service. Later, rules are grouped into policies organized hierarchically and defined by an inheritance structure. The Centralized Compliance Check service serves, as a conditional gate where infrastructure code is checked for compliance with the resources that are supposed to be deployed according to the specified policies [14–15].

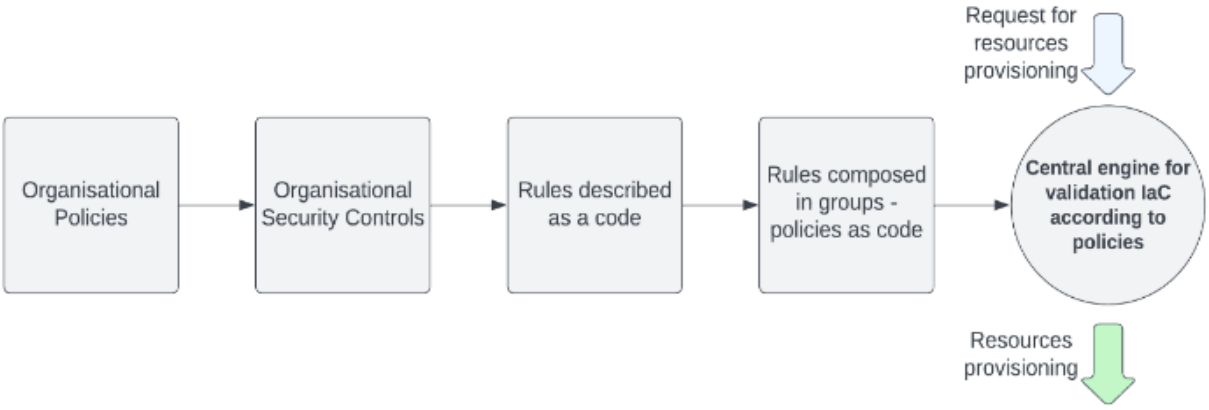

Figure 2: Simplified scheme of SaC concept

For example, if an organization sets a policy that dictates personal data or payment card data in storage must be encrypted, this policy will be declared as one of the rules that are automatically triggered when DevSecOps deploys cloud resources. A code that violates the policy is automatically rejected. Examples of policies could also include requirements such as container or virtual machine deployment images must come from trusted registries, mandatory database backup, resource replication across two availability

zones, mandatory disk encryption for virtual machines, tagging and naming conventions for resources, and so on [15].

Policies can be sourced from standards, regulations, best practices, and recommendations, including external institutions such as:

- Cloud Security Alliance (CSA)
- Center for Internet Security (CIS)
- NIST
- GDPR
- HIPAA
- PCI DSS
- SOC2
- Internal
- Others.

In most cases, these requirements and recommendations can be described as code, which can serve as preventive, detective, and reactive controls.

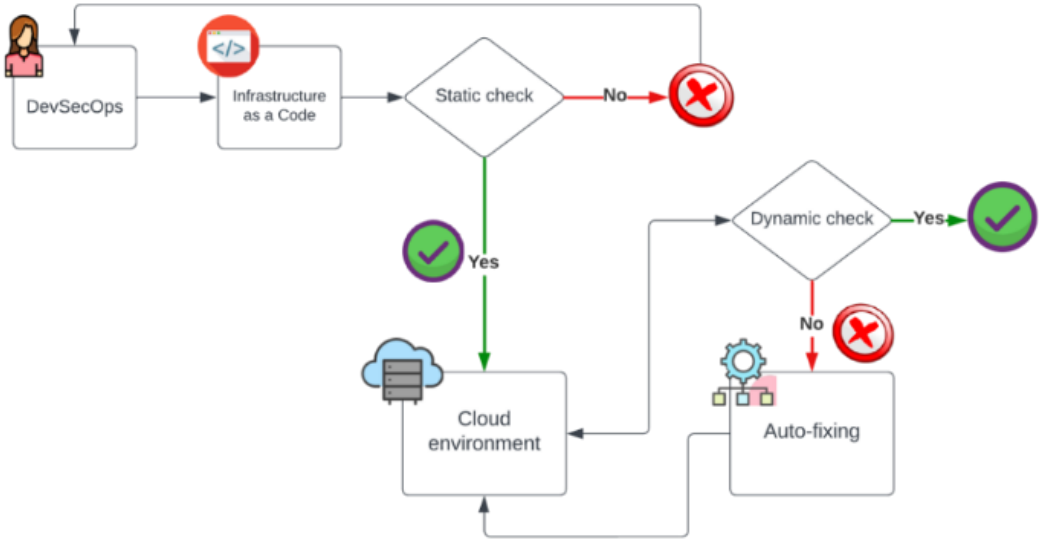

Figure 3: Process of static and dynamic validation according to policy

IaC is a prerequisite preceding the static policy compliance check. IaC can be implemented using tools like CloudFormation for AWS, Deployment management for GCP, or Resource Manager for Azure, and for a more universal solution, Terraform or Pulumi. Static policy checks should be integrated into the infrastructure code's CI/CD pipeline and adhere to GitOps best practices to avoid the installation of erroneous configurations and to correct inconsistencies at an early stage.

Detective control involves checking for inconsistencies in resource changes caused by uncontrolled factors such as manual changes or the establishment of a process that does not adhere to IaC standards. Dynamic policy checking provides real-time scanning of infrastructure to confirm its current state. Reactive control is performed according to detected non-compliance events and ensures automatic correction using serverless functions.

The component of the Centralized Policy Compliance Verification Service can be implemented using Open Policy Agent (OPA) or Regula, both of which are open-source software. In the Cloud Native Computing Foundation (CNCF), OPA was adopted as an incubating project in April 2019 and then moved to the Graduated maturity level on January 29, 2021. It provides a unified framework for policy enforcement across the stack. OPA allows you to decouple policy decisions from your services, APIs, and microservices and manage policies separately from your application code. OPA can be used in API management to declaratively define and enforce policy at multiple layers [16–17].

OPA can work with JSON files and perform static Infrastructure as Code checks, aligning with preventive control practices.

Regarding the tool for dynamically checking the current state's policy compliance for already running cloud resources, Cloud Custodian can be used. It is an open-source product that serves as both a detective and, if needed, a reactive control. This tool is built in Python, agentless, and can be deployed as a serverless function, with rules described in YAML format [18].

4. Policy examples based on CIS Amazon Web Services Foundations Benchmark v2.0.0

All CIS Benchmarks focus on technical configuration settings used to maintain and/or increase the security of the addressed technology, and they should be used in conjunction with other essential cyber hygiene tasks like:

- Monitoring the base operating system for vulnerabilities and quickly updating with the latest security patches.
- Monitoring applications and libraries for vulnerabilities and quickly updating with the latest security patches.

In the end, the CIS Benchmarks are designed as a key component of a comprehensive cybersecurity program.

This document provides prescriptive guidance for configuring security options for a subset of Amazon Web Services with an emphasis on foundational, testable, and architecture-agnostic settings [19].

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 1.16. Ensure IAM policies that allow full "*:*" administrative privileges are not attached

```
   package
terraform.aws_iam_admin_policies
  import input.tfplan
  deny[msg] {
      resource = tfplan.resources[_]
      resource["type"] ==
"aws_iam_policy"  # Adjust the resource
type as per your Terraform
configuration.
  hasFullAdminPrivileges(resource["values
"]["name"]["new"])
      msg = sprintf("IAM policy '%v'
allows full administrative privileges
and should not be attached.",
[resource["values"]["name"]["new"]])
  }
  hasFullAdminPrivileges(policyName) {
      # Define a list of administrative
privileges you want to deny.
      administrativePrivileges := ["*:*"]
      resource_policy :=
data.aws_iam_policy_document[resource["v
alues"]["policy"]["new"]]
      statements :=
resource_policy["Statement"]
      some i, statement := statements {
          statement.Action ==
administrativePrivileges[_]
          statement.Effect == "Allow"
          policyName ==
resource["values"]["name"]["new"]
      }
  }
  default allow = true
```

The policy imports the input.tfplan input, which represents the Terraform plan.

It uses a deny rule to check each IAM policy resource in the Terraform plan. If the policy contains any statements that allow full administrative privileges (specified as "*:*"), it generates a denial message.

The hasFullAdminPrivileges function checks if the IAM policy document contains any statements that allow *:* (full administrative privileges).

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 -2.1.1. Ensure S3 Bucket Policy is set to deny HTTP requests

```
package
terraform.aws_s3_bucket_policy_validatio
n
import input.tfplan
deny[msg] {
    resource = tfplan.resources[_]
    resource["type"] ==
"aws_s3_bucket_policy"
    not
hasDenyHttpStatement(resource["values"][
"policy"]["new"])
    msg = sprintf("S3 Bucket policy '%v'
does not deny HTTP requests and should
be denied.",
[resource["values"]["bucket"]])
}
hasDenyHttpStatement(policyDoc) {
    statements := policyDoc["Statement"]
    some i, statement := statements {
        statement.Effect == "Deny"
        statement.Action ==
"s3:GetObject"

containsHttpCondition(statement.Conditio
n)
    }
}
containsHttpCondition(condition) {
    keys := keys(condition)
    "IpAddress" in keys
    condition["IpAddress"] ==
{"aws:SourceIp": "HTTP request IP
address"}
}
default allow = true
```

It checks each S3 Bucket Policy resource in the Terraform plan. If the policy does not contain a Deny statement that denies HTTP requests, it generates a denial message.

The hasDenyHttpStatement function checks if the policy document contains a Deny statement that specifically denies HTTP requests for s3:GetObject actions.

The containsHttpCondition function checks if the Deny statement contains a condition that involves an HTTP request IP address.

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 2.2.1. Ensure EBS Volume Encryption is Enabled in all Regions

```
package
terraform.aws_ebs_volume_encryption
import input.tfplan
deny[msg] {
    resource = tfplan.resources[_]
    resource["type"] == "aws_ebs_volume"
# Adjust the resource type as per your
Terraform configuration.
    not isEBSEncrypted(resource)
    msg = sprintf("EBS volume encryption
is not enabled in all regions in the
Terraform configuration.")
}
isEBSEncrypted(resource) {
    encryption_enabled :=
resource["values"]["encrypted"]["new"]
    encryption_enabled == true
}
default allow = true
```

The policy imports the input.tfplan input, which represents the Terraform plan.

It uses a deny rule to check each AWS EBS volume resource in the Terraform plan. If the encrypted attribute is not set to true (i.e., EBS volume encryption is not enabled), it generates a denial message.

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 2.3.1. Ensure that encryption-at-rest is enabled for RDS Instances

```
import input.tfplan
deny[msg] {
    resource = tfplan.resources[_]
    resource["type"] ==
"aws_db_instance"
    not isEncryptionEnabled(resource)
    msg = sprintf("RDS instance %s is
not configured with encryption at
rest.", [resource["name"]])
}
isEncryptionEnabled(resource) {
    # Modify this rule to match the
naming convention of your encryption
attribute.
    attribute_exists :=
resource["values"]["storage_encrypted"]
    attribute_value :=
resource["values"]["storage_encrypted"][
"new"]
    attribute_value == true
}
default allow = false
```

The policy imports the input.tfplan input, which represents the Terraform plan.

It uses a deny rule to check each AWS RDS instance resource in the Terraform plan. If the storage_encrypted attribute is not set to true (i.e., encryption at rest is not enabled), it generates a denial message.

In AWS, storage_encrypted is typically used to enable encryption at rest.

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 3.1. Ensure CloudTrail is enabled in all regions

```
package terraform.aws_cloudtrail
import input.tfplan
deny[msg] {
    resource = tfplan.resources[_]
    resource["type"] == "aws_cloudtrail"
    not isCloudTrailEnabled(resource)
    msg = sprintf("AWS CloudTrail is not
enabled in all regions in the Terraform
configuration.")
}
isCloudTrailEnabled(resource) {
    # Modify this rule to match the
naming convention of your CloudTrail
attributes.
    attribute_exists :=
resource["values"]["is_multi_region_trai
l"]
    attribute_value :=
resource["values"]["is_multi_region_trai
l"]["new"]
    attribute_value == true
}
default allow = true
```

The policy imports the input.tfplan input, which represents the Terraform plan.

It uses a deny rule to check each AWS CloudTrail resource in the Terraform plan. If

the is_multi_region_trail attribute is not set to true (i.e., CloudTrail is not configured to be enabled in all regions), it generates a denial message.

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 5.2. Ensure no security groups allow ingress from 0.0.0.0/0 to remote server administration ports.

```
package
terraform.aws_security_group_validation
import input.tfplan
deny[msg] {
    resource = tfplan.resources[_]
    resource["type"] ==
"aws_security_group_rule"
isRemoteAdminPort(resource["values"]["fr
om_port"])
isEverywhereAllowed(resource["values"]["
cidr_blocks"])
    msg = sprintf("Security group rule
allows ingress from 0.0.0.0/0 to remote
server administration ports: %v",
[resource["values"]["from_port"]])
}
isRemoteAdminPort(port) {
    port == 22  // Add more remote server
administration ports as needed (e.g.,
3389 for RDP)
}

isEverywhereAllowed(blocks) {
    "0.0.0.0/0" in blocks
}
default allow = true
```

This policy uses the input.tfplan input, which represents the Terraform plan.

It checks each AWS Security Group Rule resource in the Terraform plan. If the rule allows ingress from 0.0.0.0/0 (anywhere) to remote server administration ports (e.g., SSH on port 22), it generates a denial message.

The isRemoteAdminPort function checks if the rule's from_port matches a remote server administration port (e.g., 22 for SSH). You can add more ports as needed.

The isEverywhereAllowed function checks if 0.0.0.0/0 is present in the cidr_blocks of the rule, indicating that it allows ingress from anywhere.

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

CIS Amazon Web Services Foundations Benchmark v2.0.0 - 06-28-2023 - 4.9. Ensure AWS Config configuration changes are monitored.

```
package terraform.aws_config_monitoring
import input.tfplan
deny[msg] {
    resource = tfplan.resources[_]
    resource["provider"]               ==
"provider[\"aws\"]"
    resource["type"]                   ==
"aws_config_configuration_recorder"

    not hasConfigMonitoring(resource)
    msg     =     sprintf("AWS     Config
configuration     changes     must     be
monitored.")
}
hasConfigMonitoring(recorder) {
recorder["values"]["recording_group"][0]
["all_supported"] == true
}
default allow = true
```

It checks each AWS Config Configuration Recorder resource in the Terraform plan. If the recorder is not monitoring all supported resource types (all_supported set to true), it generates a denial message.

The hasConfigMonitoring function checks if the Configuration Recorder has all_supported set to true, indicating that it's monitoring all supported resource types.

The default allow = true statement at the end of the policy allows all other resources not matched by the deny rule.

Rego, however, is a language that works very differently than most and can be quite unintuitive at first glance. It's more similar to SQL than to common imperative languages like Python. This means that the learning curve can be quite steep. Moreover, copy-paste development will very often not help you understand Rego—and authoring complicated policies—better [20].

5. DevSecOps role in Implementation of "Security as a Code" Approach

DevSecOps is the evolution of the DevOps philosophy, which integrates security into the software development and deployment process from its early stages. The role of DevSecOps in the "Security as Code" paradigm is pivotal, as it ensures that security concerns are embedded throughout the entire software

development lifecycle, providing a proactive and holistic approach to cloud security.

Let's consider the fundamental principles of DevSecOps methodologies and how they intersect with the "Security as Code" approach. To aid in understanding, we'll use a graphical representation of the software development lifecycle with security controls highlighted.

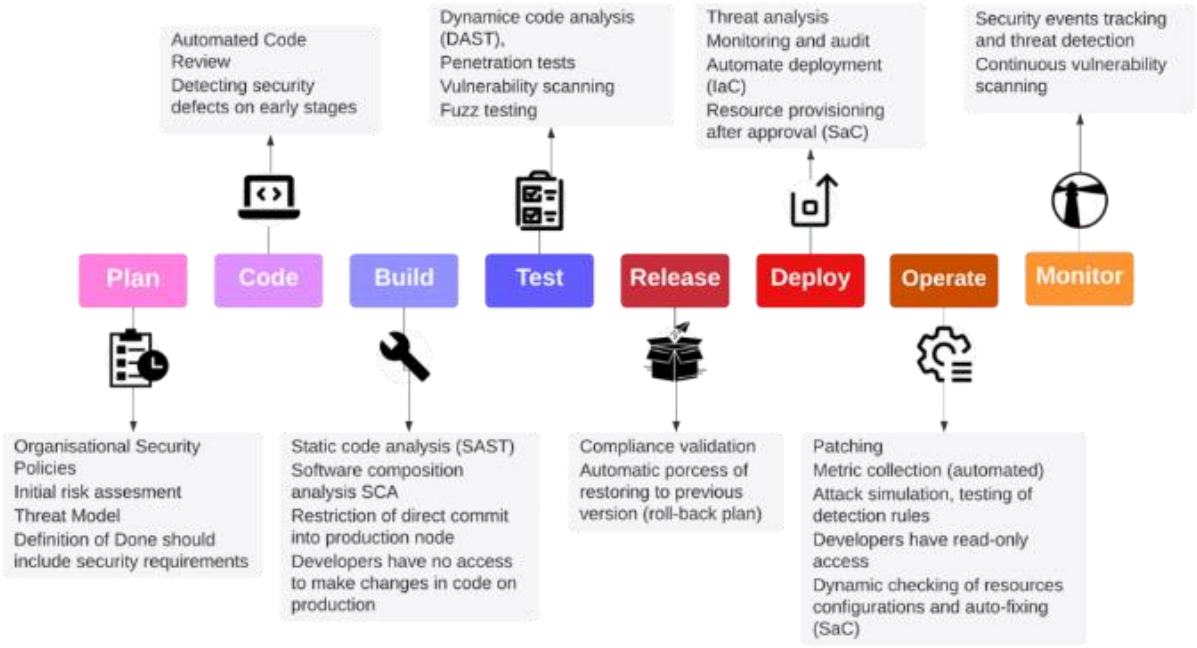

Figure 4: A software development cycle with security controls, some of which can be implemented using the "Security as Code" approach

Let's review the popular DevSecOps methodology Shift-Left principle. The principle of Shift-Left in DevSecOps practices means that security integration should occur at the early stages of development. "Security as Code" precisely facilitates such inclusion of controls, reducing the risk of deploying unprotected configurations [21]

Let's dive deeper, and answer on question—why Shift-Left security, before the advent of agile development practices and cloud computing, developers would request infrastructure from IT and receive a server weeks or months later. Over the past two decades, IT has shifted left. Today development infrastructure is fully automated and operates on a self service basis:

Developers can provision resources to public clouds such as AWS, GCP, or Azure without involving operations or IT staff:

- Continuous integration and continuous deployment (CI/CD) processes automatically set up testing, staging, and production environments in the cloud or on-premises and tear them down when they are no longer needed.
- Infrastructure-as-Code (IaC) is widely used to deploy environments declaratively, using tools like Amazon CloudFormation and Terraform.
- Kubernetes is everywhere, enabling organizations to provision containerized workloads dynamically using automated, adaptive processes.

This shift has tremendously improved development productivity and velocity, but also raises serious security concerns. In this fast paced environment, there is little time for post-development security reviews of new software versions or analysis of cloud infrastructure configurations. Even when problems are discovered, there is little time for remediation before the next development sprint begins.

DevOps organizations realized that they must also shift security left to avoid introducing more security risks than security and operations teams can manage. This movement is known as DevSecOps, and uses a variety of tools and technologies to close the gap and enable rapid, automated security assessment as part of the CI/CD pipeline [22].

Automated compliance checks in DevSecOps imply maximum automation and the elimination of manual components in configurations, aligning well with the “Security as Code” approach. Automated security checks and scanning can be easily integrated into continuous integration and continuous deployment (CI/CD) pipelines (Fig. 4). This ensures that code and infrastructure are evaluated for security compliance at each stage of development.

A collaborative approach in DevSecOps involves cooperation between development, operations, and security teams. In the context of “Security as Code”, this collaboration ensures that all teams understand and adhere to security requirements. Security experts guide defining policies, while developers implement these policies in code.

Code review and analysis are continuous processes in DevSecOps. In the “Security as Code” paradigm, this process extends beyond functional code and encompasses security-related code. Automated code analysis tools can help identify security vulnerabilities and compliance violations.

Continuous monitoring is a fundamental aspect of DevSecOps, involving ongoing monitoring of applications and infrastructure. Using the “Security as Code” approach, you can monitor the cloud environment for security policy and configuration deviations. Automated monitoring tools can rapidly identify deviations from established security standards and remediate them to the appropriate level.

DevSecOps should have incident response tools for rapid security incident response. Implementing the “Security as Code” approach allows for the automation of incident response concerning deviations from established practices and policies. The ability to react quickly is critically important.

The synergy between DevSecOps methodologies and the “Security as Code” approach creates a reliable security foundation for cloud environments. It aligns security with the principles of automation, collaboration, and continuous improvement, enabling organizations to actively address security challenges in a dynamic cloud landscape [23].

6. Fundamental principles of the "Security as Code" approach

We can highlight the following fundamental technological principles for SaC:

- **Automation**

“Security as Code” relies on automation for the consistent and scalable implementation of security policies. This includes automating the deployment of security controls, vulnerability detection, and issue remediation.

- **Version Control**

“Security as Code” should be treated as software code and managed within a version control system. This ensures a clear history of changes, facilitates collaboration among teams, and allows for testing changes in a controlled environment before production.

- **Reusability**

“Security as Code” should be modular and designed for reusability. This enables different teams to use and share standardized security control components and configurations, reducing the time and effort required for security implementation.

- **Open Standards**

“Security as Code” should be built upon open standards. This provides a more flexible and vendor-agnostic approach, reducing dependence on specific providers and allowing teams to choose the best solutions for various use cases [24].

Also, there are key organizational principles for achieving success in the implementation of SaC:

- **Establishing Clear Ownership and Accountability**

The initial principle underscores the importance of emphasizing ownership and accountability within an organization. This involves creating an internal framework to govern roles, responsibilities, and permissions. For example, determining who can author policies and for which aspects of the cloud infrastructure is vital.

- **Develop and Administer Codified Controls**

The second principle revolves around the creation and management of control objectives tailored to address specific, identified use cases. Crafting policy content that is detailed enough to meet established cloud control standards is essential. Additionally, it involves

efficiently managing an ever-expanding inventory of codified security assets.

- **Implement Cloud Security Controls Thoroughly**

The third and final principle encompasses the widespread application of security measures and safeguards wherever feasible. Employ APIs to embed security mechanisms into source code management tools, CI/CD pipelines, and runtime environments. Continuously perform audits on cloud services and workloads to assess their security, resilience, and adherence to regulatory requirements. Furthermore, establish a unified framework to enhance visibility, control, and collaboration across multi-cloud environments.

All the principles mentioned above, technological and organizational, can help avoid mistakes in the initial phases of SaC implementation and are indispensable for establishing a strong, adaptable, and agile Security-as-Code program to address the ever-evolving demands of public clouds [25].

7. Advantages of the "Security as Code" Approach

The first advantage is speed. To fully realize the business benefits of the cloud, security teams must move at a pace they are not accustomed to in on-premises environments. Manual security control configurations create friction that slows down progress and questions the overall value of the cloud for the business.

The second advantage is risk reduction. Local security control tools simply do not account for the nuances of the cloud. Cloud security requires its components to evolve throughout the entire development lifecycle. The only way to achieve this level of integration is through "Security as Code".

This approach fosters business growth. Security and compliance requirements are becoming increasingly important for company products and services. In this regard, "Security as Code" not only accelerates time-to-market but also expands opportunities for product innovation and creativity without compromising security.

Improved collaboration and morale—as development teams transitioned to more agile workflows more quickly, it created a certain gap with security teams that often operated under older methodologies. When this approach is applied, teams work in sync and have a shared understanding because they essentially speak the same language of code.

Increased visibility and transparency—with the "Security as Code" approach, security teams clearly understand which policies are applied and actively work with them.

8. Summary

In the ever-evolving world of cloud computing, where flexibility and innovation are paramount, the importance of robust security practices cannot be overstated. As organizations embark on digital transformation journeys and migrate their infrastructures to the cloud, the significance of a dynamic and adaptable approach to security becomes critical. The concept of 'Security as Code" is introduced, a revolutionary concept that not only aligns with the requirements of modern cloud environments but also transforms the fundamentals of cybersecurity. This publication has shown that "Security as Code" is more than just a trendy term; it is a transformational strategy that blends security principles with software development practices. By treating security policies, controls, and best practices as code, organizations gain the ability to automate, integrate, and enforce security measures throughout the entire lifecycle of cloud resources. One of the key findings of our research may be that "Security as Code" is more than just a technological shift; it represents an evolutionary leap. Teams comprising developers, operators, and security experts come together with a shared goal of safeguarding digital assets. Through automated testing, continuous monitoring, and iterative improvements, these teams not only close vulnerabilities but also promote a culture of transparent security. Organizations across various sectors have experienced improvements in security, streamlined compliance adherence, and accelerated incident response times. The concept has proven effective in various cloud environments, from startups to enterprises, providing a standardized environment that aligns with the dynamics of cloud infrastructure. "Security as Code" is a resilient

strategy capable of adapting to new threats and technologies.

9. Conclusion

As a result of this research, it can be concluded that the "Security as Code" approach, when implemented correctly, can significantly mitigate the risks posed by the aforementioned challenges, which represent the most significant threat to valuable information assets and resources.

This publication provides us with a direction for further research aimed at enhancing the effectiveness of this method. It also explores the expansion of its application to a wider range of services offered by cloud providers and investigates the feasibility and practicality of its application in environments such as multi-cloud or hybrid setups.

References

[1] S. Yevseiev, et al., Modeling of Security Systems for Critical Infrastructure Facilities, Technology Center (2022). doi: 10.15587/978-617-7319-57-2.

[2] S. Vasylyshyn, et al., A Model of Decoy System Based on Dynamic Attributes for Cybercrime Investigation, Eastern-European J. Enterp. Technol. 1(9) (121) (2023) 6–20. doi:10.15587/1729-4061.2023.273363.

[3] V. Grechaninov, et al., Decentralized Access Demarcation System Construction in Situational Center Network, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3188, no. 2 (2022) 197–206.

[4] P. Anakhov, et al., Increasing the Functional Network Stability in the Depression Zone of the Hydroelectric Power Station Reservoir, in: Workshop on Emerging Technology Trends on the Smart Industry and the Internet of Things, vol. 3149 (2022) 169–176.

[5] V. Grechaninov, et al., Formation of Dependability and Cyber Protection Model in Information Systems of Situational Center, in: Workshop on Emerging Technology Trends on the Smart Industry and the Internet of Things, vol. 3149 (2022) 107–117.

[6] Z. Xin, et al., Revisit Security in the Era of DevOps: An Evidence Based Inquiy Into DevSecOps Industry, IET Softw. 17(4) (2023) 435–454. doi: 10.1049/sfw2.12132.

[7] V. Buriachok, V. Sokolov, P. Skladannyi, Security Rating Metrics for Distributed Wireless Systems, in: Workshop of the 8th International Conference on "Mathematics. Information Technologies. Education": Modern Machine Learning Technologies and Data Science, vol. 2386 (2019) 222–233.

[8] V. Buhas, et al., Using Machine Learning Techniques to Increase the Effectiveness of Cybersecurity, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 3188, no. 2 (2021) 273–281.

[9] R. Kumar, R. Goyal, Modeling Continuous Security: A Conceptual Model for Automated DevSecOps Using Open-Source Software Over Cloud (ADOC), Comput. Secur. 97 (2020). doi: 10.1016/j.cose.2020.101967.

[10] K. Carter, Francois Raynaud on DevSecOps, IEEE Software 34(5) (2017) 93–96. doi: 10.1109/ms.2017.3571578.

[11] S. Das, Security as Code, 1st Edition, O'Reilly Media (2023).

[12] Amazon Web Service Documentation, Shared Responsibility Model. URL: https://aws.amazon.com/compliance/shared-responsibility-model/?nc1=h_ls

[13] C. Adtani, et al., Security as Code: The Best (and Maybe Only) Path to Securing Cloud Applications and Systems (2022). URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/security-as-code-the-best-and-maybe-only-path-to-securing-cloud-applications-and-systems

[14] R. Ferreira, Policy Design in the Age of Digital Adoption: Explore how PolicyOps can drive Policy as Code adoption in an organization's digital transformation, 1st Edition (2022).

[15] X. Zhang (2021). Cloud Governance and Compliance on AWS With Policy as Code (2011). URL: https://aws.amazon.com/ru/blogs/opensource/cloud-governance -

and-compliance-on-aws-with-policy-as-code/
[16] S. Chevre, A. Soormally, 6 Open Source Projects to Boost Your Cloud-Native API Management Game (2023). URL: https://www.cncf.io/blog/2023/05/24/6-open-source-projects-to-boost-your-cloud-native-api-management-game/
[17] T. Sandall, Open Policy Agent Graduates in the Cloud Native Computing Foundation (2021). URL: https://blog.openpolicyagent.org/open-policy-agent-graduates-in-the-cloud-native-computing-foundation-f00145202a99
[18] X. Zhang, Compliance as Code and Auto-Remediation with Cloud Custodian (2020). URL: https://aws.amazon.com/blogs/opensource/compliance-as-code-and-auto-remediation-with-cloud-custodian/
[19] C. Spiess, et al., CIS Amazon Web Services Foundations Benchmark v2.0.0 (2023). URL: https://www.scribd.com/document/664903767/CIS-Amazon-Web-Services-Foundations-Benchmark-v2-0-0
[20] J. Martin, Introduction ro Open Policy Agent (OPA) Rego Language (2022). URL: https://spacelift.io/blog/open-policy-agent-rego
[21] B. Lee, Using Open Policy Agent (OPA) to Apply Policy-as-Code to Infrastructure-as-Code (2022). URL: https://cloudsecurityalliance.org/blog/2020/04/02/using-open-policy-agent-opa-to-apply-policy-as-code-to-infrastructure-as-code/
[22] S. Gunja, Shift Left vs Shift Right: A DevOps Mystery Solved (2023). URL: https://www.dynatrace.com/news/blog/what-is-shift-left-and-what-is-shift-right
[23] G. Wilson, DevSecOps A Leasder's Guide to Producing Secure Software Without Compromising Flow Feedback and Continuous Improvement (2020).
[24] Written by Mike Tyson of the Cloud Security as Code(SaC): How to Implement and Why Use it? (2023). URL: https://blog.brainboard.co/security-as-code-3d06e0d4cd80
[25] T. Karam, Securing DevOps: The ABCs of Security-as-Code (2022). URL: https://cloudsecurityalliance.org/blog/2022/01/19/securing-devops-the-abcs-of-security-as-code/