National College of Ireland

Identifying and risk-evaluating drifts in Infrastructure as Code (IaC)-managed infrastructures.

MSc Research Project
Cloud Computing

Laura Mendez
Student ID: x23172061

School of Computing
National College of Ireland

Supervisor: Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Laura Mendez
Student ID:	x23172061
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Punit Gupta
Submission Due Date:	12/08/2024
Project Title:	Identifying and risk-evaluating drifts in Infrastructure as Code (IaC)-managed infrastructures.
Word Count:	5651
Page Count:	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Identifying and risk-evaluating drifts in Infrastructure as Code (IaC)-managed infrastructures.

Laura Mendez
x23172061
x23172061@student.ncirl.ie

Abstract

This study investigates the impact of drifts in IaC-managed infrastructures. Using an empirical approach, I created virtual machines in three different clouds (AWS and Azure) using two different IaC tools (Terraform and Pulumi). In each virtual machine, the parameters were changed directly in each cloud console, so drifts between the state of the IaC tool and the real infrastructure were generated, identified, and classified according to the risk. Each case study was analyzed to extract key information about the context of drift, causes, and outcomes in terms of security and operational effects. The findings in this paper showed that drifts are inevitable, but monitoring and observability help when using IaC-managed infrastructures to mitigate risks that could lead to unexpected behaviour.

1 Introduction

Nowadays, the abstraction and virtualisation of devices and computing services is part of a daily routine. This abstraction, known as cloud computing Vaquero et al. (2009), revolutionises how businesses operate, but what does it mean for the IT infrastructure team and how are organisations adapting to this shift?

Infrastructure as Code (IaC) is a practice that includes provisioning and managing computing infrastructure from configuration files rather than doing it directly in a configuration console. The idea was proposed as a strategy to automate the task of creating and maintaining computing infrastructure Morris (2020), making the process efficient, consistent, and scalable using software development practices such as DevSecOps workflow shown in Figure 1.

Nowadays, one of the biggest problems of managing infrastructure as code is that, unlike software development code, cloud infrastructure has many other inputs to consider in order to maintain tracking and consistency, inputs like configuration consoles, other infrastructure management tools, and the dynamic nature of some cloud resources Maayan (2024). Changes made through these other inputs can lead to discrepancies between the declared state in the code and the actual state of the infrastructure. So far, these challenges have been addressed through monitoring, auditing, and manual reconciliation mechanisms.

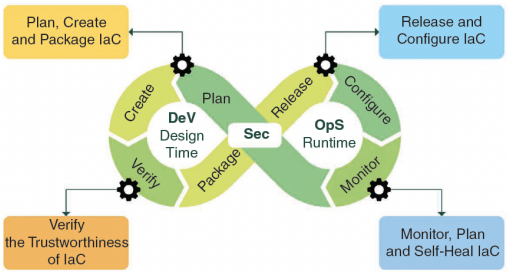


Figure 1: DevSecOps workflow, taken from Alonso et al. (2023)

1.1 Research Question

How can state drifts be identified and risk-evaluated concerning their impact on security, performance, and cost in cloud infrastructures managed by Infrastructure as Code (IaC)?

This paper will cover the following aspects:

Item #1 **Definition of state drifts:** In the context of IaC-managed infrastructures, highlighting the importance of being identified.

Item #2 **Detection methods for state drifts:** Analyzing the strengths and weaknesses of these methods.

Item #3 **Case studies:** Demonstrating a real-world example of state drifts identification using IaC projects.

Limitations:

- The scope of this paper is limited to IaC-managed infrastructures and does not cover traditional infrastructure.
- The risk evaluation framework proposed in this paper is theoretical and will require further validation through practical implementation and testing.
- The case studies are limited to provision only virtual machines; none other cloud resource is considered.

By addressing these aspects, this paper provides an understanding of state drifts in IaC-managed infrastructures and the risks they represent if they are not identified and properly managed.

2 Related Work

This section explores the principal concepts of Infrastructure as Code (IaC), its principles, evolution, challenges, and the methodologies developed to address these challenges; particularly the issue of state drifts. In this review, some state-of-the-art solutions are described and compared, emphasizing the need for risk evaluation when a state drift is detected.

Infrastructure as Code (IaC) helps teams that build and run IT infrastructure to automate their tasks and give more value in less time Morris (2020), it involves provisioning IT infrastructure through configuration scripts, so instead of manually creating and configuring any infrastructure resource, they write a script, run it, and the infrastructure

is automatically provisioned or updated, streamlining the entire process and making it repeatable.

Managing infrastructure using declarative language has been on the radar since 1993 with the inception of CFEngine Burgess (2005), which set the foundation for what today we call Infrastructure as Code. Today, CFEngine evolved to be a Continuous Configuration Automation Tool (CCA) and with the boom of cloud computing, some of their concepts were taken to give way to Cloud Management Tooling such as Azure Management Tools, AWS Cloud Formation, Morpheus, Flexera Cloud Management Platform, Pulumi and Terraform Gartner (2024).

The general principles of using Infrastructure as Code (IaC) include Frank et al. (2017):

- Idempotence (Consistency and Repeatability) - Ensures that the same output is obtained every time a script is executed, which is critical for maintaining consistency and allows reusing the code as many times as needed.
- DevSecOps best practices - Such as version control, static code analysis, automated testing, code review, continuous integration, continuous delivery, security analysis, observability, etc.
- Declarative language - The scripts contain the final desired state of the infrastructure rather than the process to get to that state.

The greatest benefits of using these principles include improved collaboration and governance, robust disaster recovery plans, and efficient automated provisioning.

On paper it is good, but in practice, some challenges have been detected when working collaboratively Nedeltcheva et al. (2023), Falazi et al. (2022), Alonso et al. (2023):

- Limited well-defined IaC code patterns - There is a lack of standardization when talking about infrastructure coding. This limitation results in teams developing their own practices, leading to confusion across different projects and organizations.
- Security and privacy - Configuration files often contain sensitive information such as API keys, passwords, network information, open ports, etc. Handling this information securely can be challenging.
- Portability and interoperability - Even when an IaC tool can handle multi-cloud, the scripts are written for specific cloud providers, making it difficult to transfer configurations between different platforms.
- Market fragmentation - IaC technologies are different in terms of purpose, data format, interfaces, and script language.
- Difficulty in replicating errors - When coding software usually there is a local environment, a testing environment, and a production environment, where errors can be replicated, but with Infrastructure as Code environmental differences or the lack of environments can lead to non-reproducible errors.
- State drifts - State drifts occur when the actual state of the infrastructure diverges from the state defined in the IaC scripts leading to inconsistencies and potential security vulnerabilities.

These challenges have been explored from the coding perspective using DevSecOps principles Petrovic et al. (2022), but what makes IaC different than just code are the other

inputs from where cloud infrastructure can be generated, such as cloud consoles, other IaC projects or even the physical infrastructure. This paper does not focus on the reasons but on the consequences of having different sources that can modify the infrastructure causing state drifts.

2.1 Definition of state drifts

> A drift is often unintentional and happens when undocumented or unapproved changes are made to software, hardware, and operating systems. It can have an impact on system performance and security.
> -*Perforce Software, Inc. (2023)*

Following the workflow in Figure 2 in a simplified way to meet the participants when using Infrastructure as Code. First, the IT infrastructure team declares the desired infrastructure in the configuration files. After testing and approval, these configuration files are applied. This application will generate the infrastructure in the cloud and produce a state file also called *the source of truth*, which describes all the infrastructure generated from the configuration files. You should be able to find all the details of your cloud resources described in the state file.

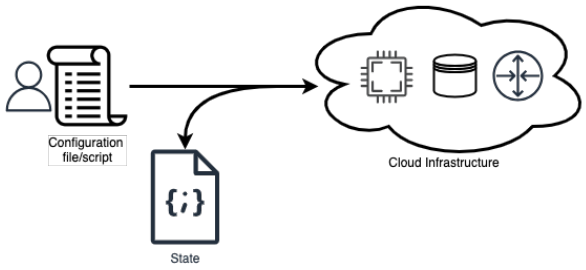


Figure 2: Infrastructure as Code Workflow

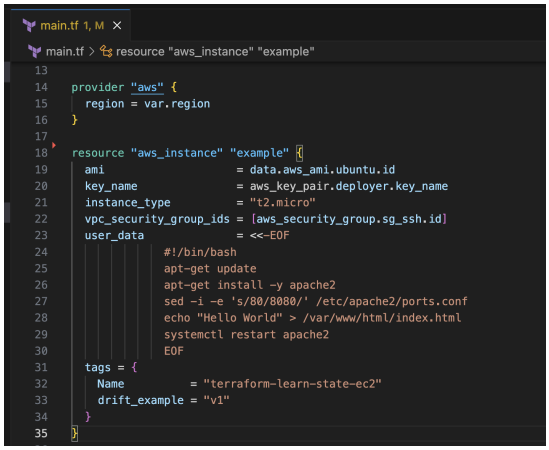
State drifts are any differences between the state file generated by the Cloud Management Tool and the actual Cloud Infrastructure. These discrepancies are generated by changes outside the control of the IaC scripts, leading to inconsistencies Qiu et al. (2023). State drifts can also be referred to as resource drifts or infrastructure drifts since the drift is presented at a resource level in the infrastructure HashiCorp (2024).

It is important to distinguish state drifts from configuration drifts; while state drifts relate to differences in the infrastructure state, configuration drifts are more related to deviations from company policies and intended configuration settings Perforce Software, Inc. (2023).

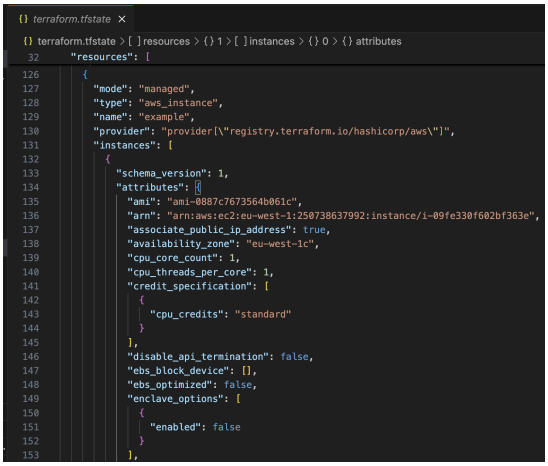
The following example intends to show how a state drift looks like using Terraform to provision an EC2 instance in AWS.

1. First, a Terraform configuration file is written specifying the desired state of an EC2 instance, including the instance type, AMI, and other necessary parameters (see Figure 3a). The Terraform configuration file for this example was taken from `https://github.com/hashicorp/learn-terraform-drift-management.git`
2. Once applied, Terraform generates the infrastructure in AWS and creates the state file that captures the current state of the provisioned EC2 instance (see Figure 3b).
3. Over time, changes are made directly in the AWS console, in this example someone modified the instance type (see Figure 3c), and the scripts and the state file were not updated.

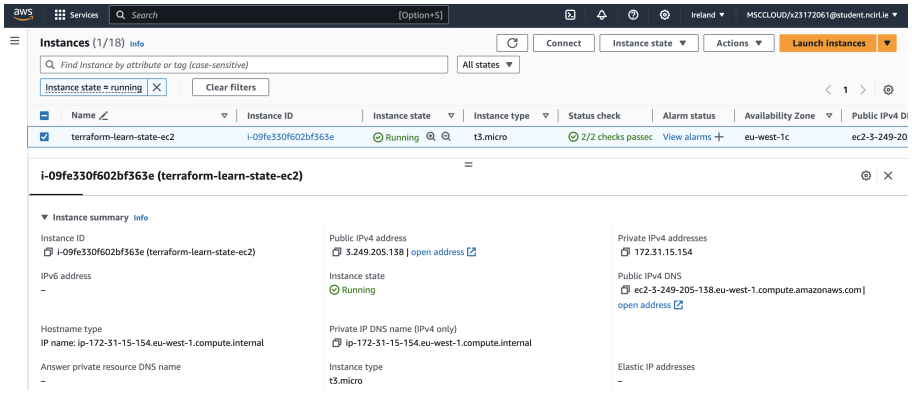
   At this point, the difference between the Terraform state file and the cloud infrastructure in AWS is considered a state drift.



(a) Terraform Configuration File Showing Initial Definition of EC2 Instance



(b) Terraform State File Showing Last Known State of EC2 Instance



(c) AWS Cloud Console Showing Modified EC2 Instance

Figure 3: Example - how a state drift looks

What if nobody notices the drift, what are the consequences? So far the drift seems inoffensive given that the instance type may not represent a big risk in this particular example, but what if the modified instance type would be a u-12tb1.112xlarge that up today costs $109.20 per hour Amazon Web Services (2024a)? Then this drift could be a big problem.

State drifts in Infrastructure as Code (IaC) managed infrastructures can result in losing control over the desired infrastructure, leading to different consequences such as Mikkelsen et al. (2019):

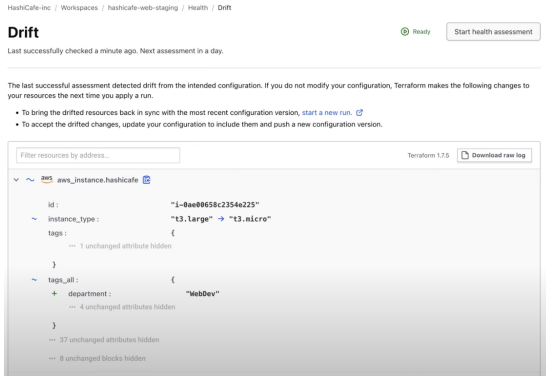
- Vulnerabilities in security.
- Unpredictable system behavior, loss of control.
- SLA non-compliance and increase in the mean time to recovery (MTTR).
- Penalties, fines, and harm to reputation due to failing in compliance.
- Extra expenses.

Therefore, detecting state drifts is crucial for maintaining the integrity and reliability on Infrastructure as Code (IaC) workflow.

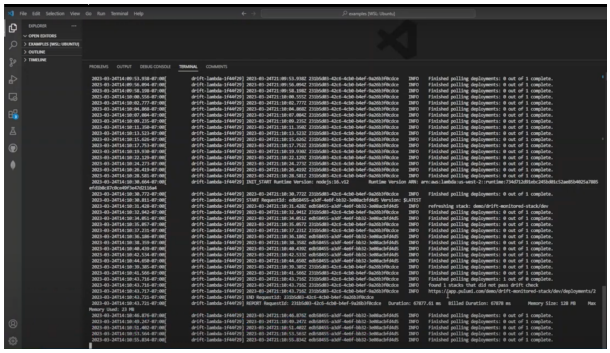
2.2 Detection methods for state drifts

The main approach to detecting state drifts involves comparisons between the state defined in the IaC configuration files and the actual infrastructure. Tools such as Terraform, AWS Config, and Ansible offer self-built functionalities to facilitate this process. These tools typically provide the option to compare the current state with the desired state and highlight any discrepancies.

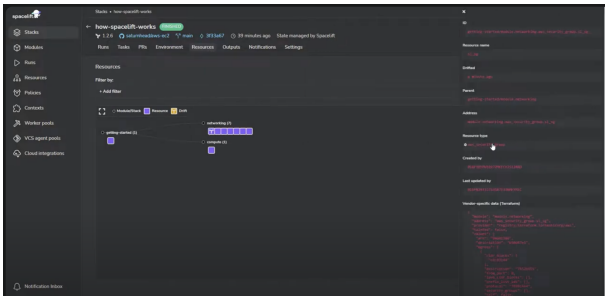
For example, Terraform's plan command generates an execution plan that shows the changes required to align the actual state with the configuration file, and it also has a drift managing module shown in Figure 4a. This solution has other functionalities, such as remediating the detected drift and sending notifications when a drift is detected.



(a) Example of Terraform Drift Detection taken from HashiCorp Youtube Channel[1]



(b) Example of Pulumi Drift Detection taken from PulumiTV Youtube Channel[2]



(c) Example of Spacelift Drift Detection taken from Spacelift Youtube Channel[3]

Figure 4: Examples of Drift Detection

Similarly, AWS Config Amazon Web Services (2024b) provides an inventory of the resources in the AWS account. This solution helps maintaining compliance with the desired state, any deviation triggers alerts allowing fast remediation of state drifts.

There is also a Pulumi program Pulumi (2022) in which a Lambda is periodically activated to refresh the Pulumi state using Pulumi Deployments API. This solution is based on the monitoring of a log where every drift-check execution is printed, if a drift is detected, the log announces which execution does not pass the check, an example of this log can be seen in Figure 4b.

[1]Automatic drift detection in Terraform Cloud. URL: `https://www.youtube.com/watch?v=FHoPzQQJw_Y`

[2]Drift Detection — Modern Infrastructure. URL: `https://www.youtube.com/watch?v=-K90I1F6tfs`

[3]Drift Detection. URL: `https://www.youtube.com/watch?v=4inDSpTEZ54`

So far, the most complete solution is the one developed by Spacelift Dinu and Fontaine (2024) where a complete state drift detection automation has been developed. This solution includes from manual reconciliation to automated jobs programmed to reconcile when a drift is detected Spacelift (2024). In Figure 4cthe Spacelift web console can be observed, showing graphically the drift identified with the detailed description on the right column.

The projects that have been discussed in this section, their singularity, and drawbacks are summarized in Table 1.

Table 1: Drift detection existing projects

Project	**Cloud**	**Singularity**	**Drawbacks**
Spacelift	Multi-cloud	Automatic remediation solution	Private workers needed, no risk evaluation
Terraform	Multi-cloud	Remediation options	Manual application, no risk evaluation
Pulumi	Multi-cloud	Open source	Log-based solution, no risk evaluation
AWS Config	AWS	Embedded on the AWS Cloud Platform	Single-cloud, no risk evaluation

Despite the advances described in this section, there are still challenges associated with state drift detection, such as the complexity of managing multi-cloud environments, the managing of continuous monitoring, the potential for false positives, and the unknown real risk and impact of drifts even when they are detected.

Given that poor collaboration is often the cause of state drifts, much literature recommends following best practices already used for software development, including continuous integration/continuous deployment (CI/CD) pipelines to ensure that drift detection is an ongoing process.

3 Methodology

A qualitative approach is performed by systematically analyzing drift scenarios using IaC tools, specifically Terraform and Pulumi, for the provisioning and management of virtual machines across AWS and Azure cloud providers. This approach ensures a comprehensive understanding of the impact of state drifts in IaC-managed environments.

The general methodology involves the following steps.

1. Data collection for analysis purposes:
    - Generating input files for provisioning virtual machines (scripts based on verified scripts taken from official IaC tool repositories with an *Apache-2.0 license.*). The final collection is formed by combining IaC tools and cloud providers:
        - Terraform - AWS [1]
        - Terraform - Azure [2]
        - Pulumi(Python) - AWS [3]

[1] `https://github.com/terraform-aws-modules/terraform-aws-ec2-instance`
[2] `https://github.com/Azure/terraform-azurerm-virtual-machine`
[3] `https://github.com/pulumi/templates/tree/master/vm-aws-python`

    - Pulumi(Python) - Azure [4]
  - Collecting state files generated by applying the scripts.
  - Using CLI tools to get the actual state of the infrastructure.

2. Analysis of properties in collected data:

   - Analyzing the properties shared among the sources, comparing configurations, and identifying differences.
   - Selecting properties to work with.
   - Evaluating the risk of each property when drifted and considering buffer of options.

3. Core solution development (API):

   - Building a REST API exposing the core functionality of the solution, that can be summarized as:
     - Saving the source files and cloud provider connectivity data.
     - Using the project ID as input to fetch the source files and cloud provider connectivity data.
     - Implementing functionality to identify state drift between the actual infrastructure, the state file (desired infrastructure), and the configuration file.
     - Giving a risk evaluation for the differences identified, based on the resulting table from the analysis.

4. User interface development:

   - Building a Web Application to allow final users to consume the final solution, the main interactions to develop are:
     - Collecting source files and cloud provider connectivity data, to save it as a project.
     - Displaying the summary report for the infrastructure project.
     - Showing detailed information on each drifted attribute identified.

5. Evaluation:

   - The primary goal is to replicate the drift scenarios to evaluate the accuracy and effectiveness of the developed solution. The objective is to evaluate whether the solution can detect and evaluate drifts in a manner consistent with previous studies.
     - Setting up test environments across AWS and Azure using the same IaC scripts used in the earlier phases of the methodology.
     - Making manual changes to the infrastructure that deviate from the original state.
     - Running the developed system to detect these drifts and evaluate the risks associated with them.

[4] `https://github.com/pulumi/templates/blob/master/vm-azure-python`

– Comparing the outcome against the known introduced changes to verify accuracy and compare overall information provided with the technologies mentioned in section 2.2.

4 Design Specification

4.1 Model Description

The entire project is divided into three main stages: Data Normalisation Stage, Drift Detection Stage, and Risk Evaluation Stage.

4.1.1 Data Normalisation Stage

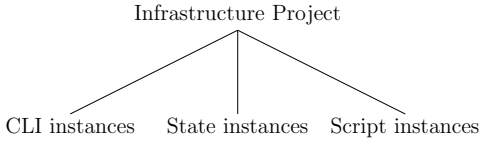
The solution proposed in this paper requires three inputs: the configuration file, the state file, and the actual architecture file, so this stage aims to collect these files and normalise the attributes among the three of them.

The configuration file and the state file must be provided by the final user of the solution, the state file must be the one obtained by applying the configuration file provided.

The actual architecture files are dynamically updated using CLI tools to obtain the current state of the infrastructure, the connection data to use the CLI tool must also be provided by the final user of the solution. Data provided for the user is stored under an infrastructure-project-ID, and using encrypted storage for security reasons.

Lastly, all the collected files have different structures and attribute names, so this stage takes the source files and normalise them into a unified format following structure in Figure 5

Figure 5: Drift Detection Project Collection Data Structure



4.1.2 Drift Detection Stage

The methods described in section 2.2 use the Terraform and Pulimi refresh functionality to identify drifts; here, an external connection directly to the cloud is used to avoid IaC tool bias functionality. They also do not identify missing instances.

This stage analyses the collected data using the outcome of the Data Normalisation Stage for the algorithm to identify drifts, such algorithm is designed as follows:

1. Identify the common instances between the CLI instances and the State instances.
    - The instances in the CLI instances but not in the State instances are marked as missing instances.
2. For each common instance:
    (a) Identify the common attributes between the CLI attributes and the State attributes.

(b) Verify that the attribute values match.

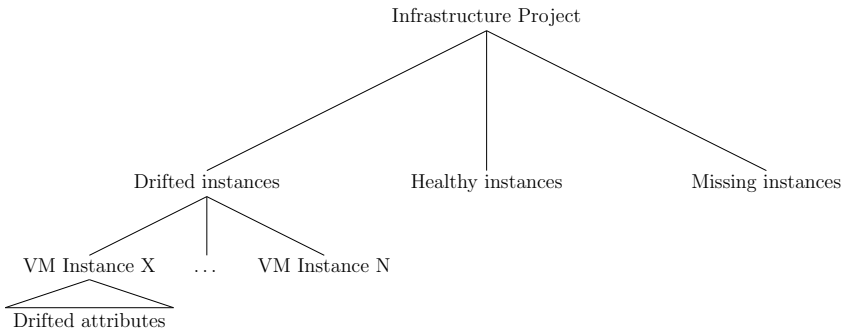
- If the values do not match, it is marked as a drifted attribute

(c) Evaluate whether the attribute exists in the script attributes.

- If the attribute does not exist, it is marked as a drifted attribute
- If the attribute exists, but its value does not match with the CLI and State value, it is marked as a drifted attribute

The data structure of the solution at this point should look like Figure 6 where each drifted attribute may contain its CLI value, state value, and script value.

Figure 6: Drift Detection Project Output Data Structure



4.1.3 Risk Evaluation Stage

This stage evaluates the risk associated with each drifted attribute identified in the Drift Detection Stage. The model evaluates each drift based on predefined risk tables and assigns a risk level based on the deviation found. This helps the end user prioritize remediation efforts.

Although the existing methods described in section 2.2 have remediation and automation features, none has a risk evaluation of the drift identified.

The risk evaluation model is applied on each drifted attribute taken as input the values from the different sources (CLI tool, state file, and script file) and follows these steps to assess the risk:

1. Normalization: If the attribute value is a number, a simple normalization will be applied to obtain values between 0 and 100.
2. Ranking: A High, Medium, or Low label is assigned to the attribute according to the deviation between the state and CLI value. This deviation is based on predefined risk tables.
3. Identification of default values: this scenario is identified when the scrip value is empty but the state and CLI values match.
4. Identification of outdated values: this scenario is identified when the state and CLI values match but the scrip value is different, which means the provided state file is not the last version generated by the provided script file.
5. Scoring: The complete infrastructure project is evaluated according to the total High, Medium, and Low risk identified, giving an overall percentage of trustworthiness.

4.2 Components

The solution is composed of two components: the API Module, where the core solution is developed, and the User Interface Module, where the results are shown to the final user. These components ensure that the system is not only functional but also accessible and user-friendly.

4.2.1 API Module

The API Module is designed to provide a programmable interface for storing the input data, accessing and managing the state drift analysis results, and facilitating integration with external systems or tools.

The endpoints required are the following. Create a new project, get the project list, check the health of a project, and get the details of a drifted project.

4.2.2 User Interface Module

The User Interface (UI) Module is designed to provide a user-friendly platform for interacting with the API. This module must allow users to easily check the health of their IaC projects. In the Web Interface, the user can create projects, initiate drift detection, view the results, and get the details needed to prioritise remediation activities.

5 Implementation

The Implementation section details the final stage of the proposed solution, highlighting the outputs produced, the tools and frameworks used, and the overall process that translates the specifications described in section 4 into a functional product.

The novelty of this project lies in its approach to risk evaluation for identified drifts and the trustworthiness assessment of the entire IaC-managed infrastructure. This study developed a solution that integrates both drift detection and risk evaluation. For drift detection, using CLI tools for AWS and Azure, as well as state files for Terraform and Pulumi. This method contrasts with existing projects, which are typically native to Terraform or Pulumi and do not specify their source for comparison, making their methods less transparent.

The main loophole identified in current projects is the absence of risk evaluation for detected drifts. Existing solutions focus on detecting drifts and suggesting remediation, without evaluating any risks that might be associated. This project overcomes this gap by providing a risk evaluation for each identified drift, it includes the reason why the drift was identified showing the original value, the desired value, and the current value. Additionally, it takes the given risk evaluations to calculate an overall trustworthiness percentage for the entire IaC-managed infrastructure, offering an overview of the reliability of the IaC.

5.1 Outputs Produced

The final implementation involved integrating all modules and producing various outputs, including transformed data, developed code, and generated reports. This section outlines the outputs produced during the overall solution.

5.1.1 Models

The two models developed as the final solution enable the identification of state drifts in virtual machine instances and evaluate their associated risks.

1. Drift Detection Model

   The Drift Detection Model is developed to identify discrepancies between the actual state of the infrastructure and the desired state as specified in the configuration files and state files. This model performs the steps in Algorithm 1.

Algorithm 1 Drift Detection Algorithm

```
Input: cloud, tool, state_file, script_file, cloud_connectivity_data
Output: healthy_instances, drifted_instances, missing_instances
Step 1: Retrieve actual infrastructure using CLI
raw_cli_instances ← cli_connection(cloud_connectivity_data)
Step 2: Normalize instances according to the cloud provider
cli_instances ← normalise(cloud, raw_cli_instances)
state_instances ← normalise(cloud, state_file)
script_instances ← normalise(cloud, script_file)
Step 3: Get common instances and missing instances
common_instances ← cli_instances and state_instances
missing_instances ← cli_instances - common_instances
Step 4: Compare attributes and get drifted instances
drifted_instances ← []
for each instance in common_instances do
    cli_instance ← cli_instances[instance]
    state_instance ← state_instances[instance]
    is_drifted ← false
    for each attribute in cli_instance.attributes do
        if cli_instance[attribute] ≠ state_instance[attribute] then
            is_drifted ← true
        end if
    end for
    if is_drifted then
        add instance to drifted_instances
    end if
end for
Step 5: Get healthy instances
healthy_instances ← common_instances - drifted_instances
return healthy_instances, drifted_instances, missing_instances
```

   The model retrieves the actual infrastructure using CLI tools, compares them against instances specified in the state file, and the instances not found in the state file, and marks them as missing instances. For each common instance, the model identifies attributes that do not match between the actual state and the desired state. These discrepancies are flagged as drifts. Drifted instances are those with one or more drifted attributes; if no drifted attributes, then is a healthy instance. The output of this model is the list of healthy, drifted, and missing instances, along with detailed information on the drifted attributes.

2. Risk Evaluation Model

   The drifts are classified into high, medium, or low risk based on the distance of the new value from the original value, using the hierarchy table and a simple slope equation specified in Equation 1.

$$\text{Change rate} = \text{Round}\left(\frac{|\text{New Value} - \text{Original Value}|}{\max(\text{Values}) - \min(\text{Values})}\right) \tag{1}$$

Then the risk is classified[5] according to Table 2

Change Rate (%)	**Risk Level**
1% – 10%	Low
11% – 50%	Medium
51% – 100%	High

Table 2: Risk Level Based on Change Rate

Once all the attributes have a risk evaluation then the complete project infrastructure can be evaluated to indicate how much the Infrastructure as Code (IaC) can be trusted, and prioritize remediation tasks.

The trustworthiness calculation uses a weight-scoring algorithm, which evaluates the overall infrastructure reliability. This algorithm assigns different weights to each risk level (low, medium, and high) and normalizes the results to produce a trustworthiness score. The detailed calculation process is based on Algorithm 2

Algorithm 2 Calculate Trustworthiness

Input: low_risk_count, medium_risk_count, high_risk_count, unknown_risk_count, coverage
Output: trustworthiness
Step 1: Initialize total_count
total_count ← low_risk_count + medium_risk_count + high_risk_count + unknown_risk_count
Step 2: Define risk weights
low_risk_weight ← 1
medium_risk_weight ← 3
high_risk_weight ← 5
Step 3: Calculate the weighted sum of risks
weighted_sum ← (low_risk_count × low_risk_weight) + (medium_risk_count × medium_risk_weight) + (high_risk_count × high_risk_weight) + (unknown_risk_count × high_risk_weight)
Step 4: Normalize the total risk score
max_possible_weighted_sum ← total_count × high_risk_weight
normalized_risk_score ← (weighted_sum / max_possible_weighted_sum) × 100
Step 5: Calculate trustworthiness
trustworthiness ← (100 - normalized_risk_score) × coverage
return trustworthiness

5.1.2 Code Developed

Two coding projects[6] developed for this solution were an API with the core solution and a User Interface:

[5]If the attribute is not in the hierarchy table, it is classified as Unknown and counted as a high-risk weight in the trustworthiness calculation.

[6]The complete code developed can be found in the repositories (permission must be granted for access):

- API: `https://github.com/laurusik/drift_detector_api.git`
- User Interface: `https://github.com/laurusik/drift_detector_user_interface.git`

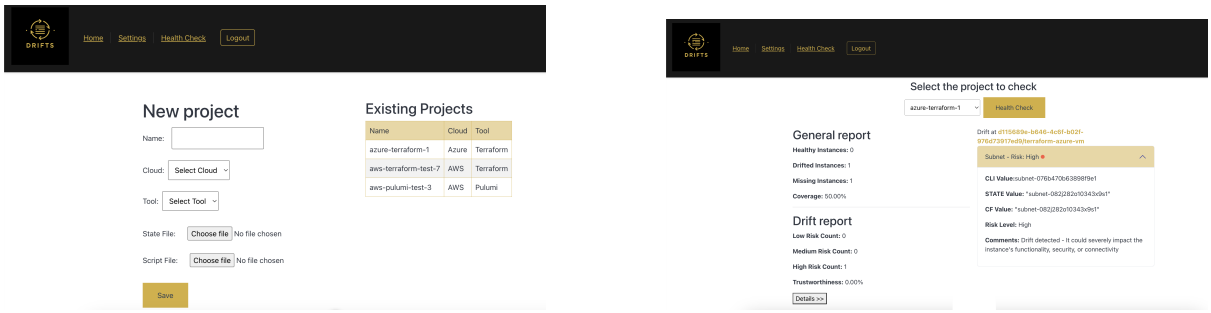
1. API

   The developed API is the main component that facilitates the interaction between the processing modules and the user interface. It provides endpoints for data storage, drift detection, and result summarising, and also incorporates security measures to protect sensitive information. The available endpoints are:

   - **Data Storage Endpoint**: Accepts the source files (state and configuration files) and the cloud provider connectivity data. Validates and stores these data securely by implementing encrypted storage.
   - **Drift Details Endpoint**: Triggers the drift detection process for an infrastructure project and returns the detailed identified drifts with the risk assigned.
   - **Report Endpoint**: Fetches the summarized results of the drift analysis and risk evaluation.

2. User Interface

   The user interface (UI) was developed to provide an intuitive and interactive Web Portal for users to interact with the system. It allows users to upload source files, initiate drift detection, and view drift reports. The UI is built using a responsive and user-friendly experience having the following features:



(a) Settings View

(b) Health Check View

Figure 7: User Interfaces

   - Views
     - **Home**: Displays an overview of the purpose of the Web Portal.
     - **Settings**: Allows users to create projects, upload the necessary source files, and provide the cloud connectivity details - Figure 7a.
     - **Health Check**: Displays the report results allowing the user to get more details about the identified drifts - Figure 7b.
   - Security
     - **Authentication**: Implements secure user authentication and authorization mechanisms to protect user data and ensure privacy.

5.1.3 Data

There are two main outputs in this solution, the report and the drift details, here, is the description of the data transformation for each output.

1. Data Normalization and Risk Evaluation

   The configuration files, state files, and actual architecture files are transformed into a unified object at running time. This normalisation is applied at the attribute level so that all attributes can be compared, and evaluated to assign a risk level.

Data Normalization Output Example


```
{
    "drifted_instances": {
        "vmb951a7d0": {
            "instance_id": "vmb951a7d0",
            "attributes": {
                "Subnet_Id": {
                    "key": "Subnet_Id",
                    "cli_value": "subnet-0e78ac25b5142713d",
                    "state_value": "subnet-076b470b63898f9e1",
                    "conf_value": "subnet-076b470b63898f9e1",
                    "risk_level": "High",
                    "comments": "Drift detected - It could severely
                        impact the instance's functionality,
                       security, or connectivity"
                }
            }
            // ... other attributes
        }
        // ... other drifted instances
    },
    "healthy_instances": {
        // ... healthy instances data
    },
    "missing_instances": {
        // ... missing instances data
    }
}
```


2. Infrastructure Project Report

   The report gives an overall health summary of the infrastructure project. It includes the total count of missing, drifted, and healthy instances, calculating the coverage of the actual infrastructure built by IaC. Additionally, the report calculates the trustworthiness of the Infrastructure as Code (IaC)-managed infrastructure based on the types of risk detected. Here is an example of a report output.

Infrastructure Project Report Output Data Example


```
{
    "drifted_instances_count": 1,
    "healthy_instances_count": 2,
    "missing_instances_count": 1,
    "coverage": "75.00",
    "low_risk_count": 3,
    "medium_risk_count": 1,
    "high_risk_count": 1,
    "trustworthiness": "42.00"
}
```

5.2 Tools and Frameworks

The complete list of tools and frameworks used for this solution can be found in Table 3. The category and purpose of each tool or framework are described to understand the application of each tool in the context of this solution. The specific methodology step it supports is also mentioned.

Table 3: Tools and Frameworks

Tool or framework	Category	Purpose	Methodology step
Terraform and Pulumi	Infrastructure-as-Code (IaC) Tools	Provision and management of infrastructure for AWS and Azure	Analysis and Evaluation
AWS and Azure virtual machine instances	Cloud Infrastructure	Provide secure, resizable computing in the cloud, offering a big range of processor, storage, networking, and OS options	Analysis and Evaluation
Python + Flask	General-Purpose Programming Language	Development of the API to handle requests and responses for the main solution	Development of the core solution
MySQL	Database	Provide encryption storage to ensure security on sensitive data storage	Development of the core solution
Vue + Vite	Web Development Framework	Development of an interactive user interface to display drift analysis results	User interface development
Keycloak	Security: Identity and Access Management Tool	Provision authentication to the user interface.	User interface development

6 Evaluation

The Evaluation section provides a comprehensive analysis of the results and insights of the solution proposed. The most relevant results are discussed and identified as supporting the research question.

To evaluate the effectiveness of the developed solution, there were carried 4 case studies using IaC scripts to provide virtual machines across AWS and Azure following the scenarios described in Table 4

This distribution allows to show how drifts in different IaC tools across AWS and Azure can be detected. The intention is to show how these tools operate in different contexts, rather than comparing the same scenario across both platforms.

Additionally, using the NCI cloud resources allows to demonstrate a scenario with low coverage in AWS but not in Azure. This difference provides valuable insights into the challenges and considerations when managing IaC infrastructures.

During the initial phase of the research, native AWS (AWS CloudFormation) and Azure (Azure Resource Manager) IaC tools were considered, as they came up while writing the related work. However, they were not used in this evaluation because the focus of the study is multi-cloud environments, then, only tools like Terraform and Pulumi that offer multi-cloud compatibility were chosen. Native tools are limited to their respective cloud environments, which would not provide a cross-platform analysis.

Table 4: Evaluation Scenarios

Scenario ID	Cloud	IaC Tool	Purpose
1	Azure	Terraform	Showing a full coverage NON-drifted infrastructure project
2	Azure	Pulumi	Showing a full-coverage high-trustworthiness (low-risk) infrastructure
3	AWS	Terraform	Showing a low-coverage medium-trustworthiness (medium-risk) infrastructure
4	AWS	Pulumi	Showing a low-coverage low-trustworthiness (high-risk) infrastructure

6.1 Scenario 1

Showing a fully covered, NON-drifted infrastructure project, this case was chosen to provide a baseline comparison against projects where drifts are present. The main purpose is to show that the solution has the ability to identify a healthy infrastructure.

Cloud: Azure **IaC Tool:** Terraform

Steps: Followed steps to replicate this scenario

1. Apply the IaC script to provision a Virtual Machine
2. Set up the Azure-Terraform project in the solution
3. Run the health check solution

Results: Expected and obtained results are summarised in Table 5, the supportive evidences can be found in Figure 8

Table 5: Comparison of Expected and Obtained Results for Scenario 1

Metric	Expected Result	Obtained Result
Healthy instances	1	1
Coverage	100%	100%
Drifts Detected	None	None
Trustworthiness	100%	Not Displayed

This successful baseline scenario indicates that any drift found after is indeed generated by a change made in the infrastructure outside of the IaC workflow. The healthy instance found confirms that no unauthorised changes have been made and that all components match the desired state. The trustworthiness metric was expected to be 100%, but it was not displayed as by design the trustworthiness is only calculated when a drift is detected because the maximum trustworthiness that a project can reach is the percentage of the entire infrastructure covered, in this scenario is 100%.

6.2 Scenario 2

Showing a half-coverage high trustworthiness (low risk) infrastructure, this case was chosen to show a low-risk identification and the behaviour of the solution when the drift happens by adding features in the cloud provider console.

In this scenario, it is considered that the instance created in section 6.1 has been destroyed and would not affect the trustworthiness value in this scenario.

Cloud: Azure **IaC Tool:** Pulumi

Steps: Followed steps to replicate this scenario

1. Apply the IaC script to provision a Virtual Machine
2. Set up the Azure-Pulumi project in the solution
3. Manually create a drift by adding a Tag directly in the Azure console.
4. Run the health check solution

Results: Expected and obtained results are summarised in Table 6, the supportive evidences can be found in Figure 9

Table 6: Comparison of Expected and Obtained Results for Scenario 2

Metric	Expected Result	Obtained Result
Missing Instances	0	0
Coverage	100%	100%
Drifts Detected	1-Low	1-Low
Trustworthiness	80%	80%

Figure 8: Scenario 1 Result

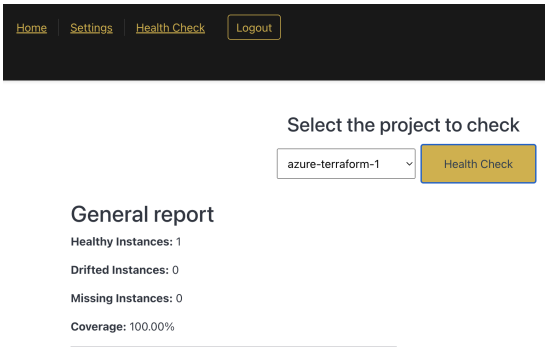
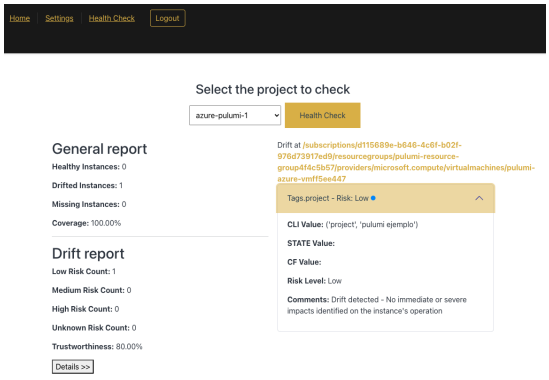


Figure 9: Scenario 2 Result



This scenario shows 100% accuracy in finding added configurations to the ones defined by the IaC project. A low-risk drift typically involves minor changes, but even when the impact is evaluated as low risk, the identification of these drifts is key to planning remediation work not only in the infrastructure but also in the teamwork and the workflow and procedures that should be followed.

6.3 Scenario 3

Showing a low-covered infrastructure, which means that most of the instances are missing from the Infrastructure as Code (IaC) project evaluated. The drift found will exemplify a medium risk detected by modifying a property set by the configuration scripts.

Cloud: AWS **IaC Tool:** Pulumi

Steps: Followed steps to replicate this scenario

1. Apply the IaC script to provision an EC2 instance
2. Set up the AWS-Terraform project in the solution
3. Manually create a drift modifying the instance type from t2.micro to **t3.large**.
4. Run the health check solution

Results: Expected[7] and obtained results are summarised in Table 7, the supportive evidences can be found in Figure 10

Table 7: Comparison of Expected and Obtained Results for Scenario 3

Metric	Expected Result	Obtained Result
Missing instances	> 10	76
Coverage	low	1.3%
Drifts Detected	1-Medium	1-Medium, 1-Low, 1-Unknown
Trustworthiness	1.04%	0.52%

This is one example of a worst-case scenario where only 1.3% of the infrastructure is managed by an IaC tool, this means that the highest trustworthiness that can be expected is 1.3% and any drift found will affect the final result. The low coverage in this scenario shows the importance of having a proper IaC project where all the instances and changes can be tracked in a unique process. The accuracy of drift identification is 33.3% as only one drift was expected and 3 were identified, reducing even more the trustworthiness.

6.4 Scenario 4

Having the same case as section 6.3, changing the same attribute but a higher distance from the original value to have a higher risk evaluated.

Cloud: AWS **IaC Tool:** Pulumi

Steps: Followed steps to replicate this scenario

1. Apply the IaC script to provision an EC2 instance
2. Set up the AWS-Terraform project in the solution
3. Manually create a drift modifying the instance type from t2.micro to **i3.large**.
4. Run the health check solution

Results: Expected[7] and obtained results are summarised in Table 8, the supportive evidences can be found in Figure 11

[7] As using NCI cloud resources, cannot be specified an exact number of missing instances nor coverage

Table 8: Comparison of Expected and Obtained Results for Scenario 4

Metric	Expected Result	Obtained Result
Missing instances	> 10	75
Coverage	low	1.32%
Drifts Detected	1-High	1-High, 1-Low, 1-Unknown
Trustworthiness	0%	0.35%

The result of this scenario aims to highlight the impact of a drift when the new value is further away from the original value. Having as a baseline the results in section 6.3 where the impact of changing t2.micro to t3.large was evaluated as medium risk, then changing t2.micro to i3.large implicates a major impact on potential performance and cost, that is why it is evaluated as high risk.

Figure 10: Scenario 3 Result

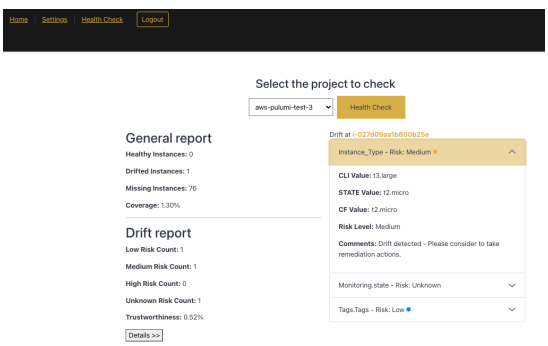
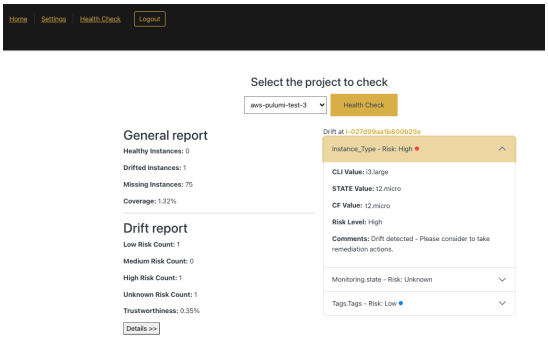


Figure 11: Scenario 4 Result



6.5 Discussion

The evaluation scenarios presented demonstrate the robustness of the developed solution in detecting and assessing infrastructure drifts across different cloud environments and IaC tools. The key findings include:

- **Drift Detection**: The solution consistently detected drifts across all scenarios where drifts were expected.
- **Trustworthiness**: The trustworthiness scores reflected the health of the infrastructure, with higher scores correlating with fewer or less-risk drifts.
- **IaC Coverage**: Scenarios 3 and 4 highlighted the critical role of IaC coverage in maintaining infrastructure integrity. Low coverage not only reduces trustworthiness but also increases the probability of untracked changes.
- **Risk Evaluation**: The comparison between Scenarios 3 and 4 illustrates how the risk level of a drift, directly influences the overall trustworthiness.

While the experiments demonstrated the overall effectiveness of the solution, they also highlighted areas where the design could be improved:

- **Drift Detection**: The solution successfully identified drifts in the infrastructure suggesting a robust model. Yet in scenarios 3 and 4 there were identified more drifts than expected indicating the Drift Detection Model may need an adjustment handling nested attributes.

- **Risk Evaluation**: The solution confirms its capability to detect low, medium, and high-risk drifts. However, there is a fourth category of unknown risk, which means that the Risk Evaluation Model must consider more cases and drift scenarios.
- **Metrics**: The coverage and trustworthiness scores are the data that any infrastructure manager is looking for. The trustworthiness calculation gives very useful information about the overall health of the infrastructure. However, the linear reduction, particularly in high-risk scenarios, suggests that the scoring algorithm could need refinement to avoid disproportionate penalties such as in scenario 4 where only one high-risk identified drift reduced trustworthiness to 0%.
- **Methodology**: Using CLI tools to fetch the actual infrastructures is the most direct way to go to the source, but during the construction of the solution, identifying and unifying the common attributes among all the source files across the different IaC tools and cloud formats closed the development in terms of including any other cloud to IaC tool beside the ones defined from the begging. This supports the decisions made by the solutions presented in section 2.2 not using file comparison that will imply high maintenance in long-term scenarios.

7 Conclusion and Future Work

This paper aimed to answer the question: How can state drifts be identified and risk-evaluated concerning their impact on security, performance, and cost in cloud infrastructures managed by Infrastructure as Code (IaC)?

To achieve that objective, the study developed and tested a solution that automates drift detection and risk evaluation of state drifts in different cloud environments (AWS and Azure) using IaC tools (Terraform and Pulumi). Four case studies were conducted, each focusing on different aspects of drift detection, risk evaluation impact, and infrastructure coverage. The findings were:

- Drift Detection: The solution detected state drifts across different cloud providers and IaC tools.
- Risk Evaluation: The risk evaluation model provided valuable insights for the detected drifts that can work as guidance for infrastructure management.
- IaC Coverage Importance: Having a high coverage using IaC tools to manage cloud infrastructure is key to keeping track of changes and maintaining trustworthiness in the projects.

In conclusion, this solution supports that drift detection and risk evaluation in IaC-managed environments improve infrastructure management practices, particularly in multi-cloud environments. However, it has limitations: the scope was limited to virtual machines, and the risk evaluation framework requires further validation in more diverse and complex infrastructure scenarios. Lastly, trustworthiness scores in high-risk scenarios suggest the need to adjust the calculation algorithm.

7.1 Future Work

There are several areas where future work could be done:

- Extending Beyond Virtual Machines: Cover a broader range of cloud resources. This would provide a more detailed infrastructure health.
- Extending cloud providers: Cover cloud providers different from AWS and Azure. This would enhance the multi-cloud environment's usability.
- Extending IaC tools: Cover IaC tools different from Terraform and Pulumi. This would provide a different methodology that may involve less attribute normalisation.
- Enhance drift detection model: Go deep into nested attributes that may involve a finer but important risk evaluation.
- Enhance risk evaluation: Redefine the risk evaluation model to consider context using machine learning techniques allowing dynamically adjusting risk scores.
- Exploring Potential for Commercialisation: Commercialization could make the solution accessible to a wider audience commercialising the solution as a SaaS (Software as a Service) product.

References

Alonso, J., Piliszek, R. and Cankar, M. (2023). Embracing iac through the devsecops philosophy: Concepts, challenges, and a reference framework, *IEEE Software* **40**(1): 56–62.

Amazon Web Services (2024a). *Amazon EC2 On-Demand Pricing*. Accessed: 2024-07-26.
URL: *https://aws.amazon.com/ec2/pricing/on-demand/*

Amazon Web Services (2024b). *AWS Config Developer Guide*. Accessed: 2024-07-20.
URL: *https://docs.aws.amazon.com/config/latest/developerguide/cloudformation-stack-drift-detection-check.html*

Burgess, M. (2005). A tiny overview of cfengine: Convergent maintenance agent, *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO* **2005**: 4. P.O. Box 4, St. Olavs Plass, Oslo 0230, Norway. Email: mark@iu.hio.no.

Dinu, F. and Fontaine, J.-M. (2024). Infrastructure drift detection and how to fix it with iac tools. Accessed: 2024-07-24.
URL: *https://spacelift.io/blog/drift-detection*

Falazi, G., Breitenbücher, U., Leymann, F., Stötzner, M., Ntentos, E., Zdun, U., Becker, M. and Heldwein, E. (2022). On unifying the compliance management of applications based on iac automation, *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pp. 226–229.

Frank, F., Alfke, M., Franceschi, A., Pastor, J. S. and Uphillis, T. (2017). *Puppet: Mastering Infrastructure Automation*, Packt Publishing, Birmingham, UK.
URL: *https://research.ebsco.com/linkprocessor/plink?id=d3abd1f7-b67b-354f-a0d3-68a51e02536b*

Gartner (2024). Cloud management tooling. Accessed: 2024-07-17.
URL: *https://www.gartner.com/reviews/market/cloud-management-tooling*

HashiCorp (2024). Detect resource drift in terraform state. Accessed: 2024-07-17.
URL: *https://developer.hashicorp.com/terraform/tutorials/state/resource-drift*

Maayan, G. D. (2024). Configuration as code: Trends and predictions for 2024, `https://devops.com/configuration-as-code-trends-and-predictions-for-2024/`. Accessed: 2024-02-14.

Mikkelsen, A., Grønli, T.-M. and Kazman, R. (2019). Immutable infrastructure calls for immutable architecture, *Hawaii International Conference on System Sciences*.
URL: *https://api.semanticscholar.org/CorpusID:102351693*

Morris, K. (2020). *Infrastructure as Code*, O'Reilly Media, Inc. ISBN-13: 978-1-0981-1467-1.

Nedeltcheva, G. N., Xiang, B., Niculut, L. and Benedetto, D. (2023). Challenges towards modeling and generating infrastructure-as-code, *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23 Companion, Association for Computing Machinery, New York, NY, USA, p. 189–193.
URL: *https://doi.org/10.1145/3578245.3584937*

Perforce Software, Inc. (2023). Configuration drift, `https://www.puppet.com/blog/configuration-drift`. Accessed: 2024-07-20.

Petrovic, N., Cankar, M. and Luzar, A. (2022). Automated approach to iac code inspection using python-based devsecops tool, *2022 30th Telecommunications Forum (TELFOR)*, IEEE, Belgrade, Serbia, pp. 1–4.
URL: *https://ieeexplore.ieee.org/document/9983681/*

Pulumi (2022). Drift detection. Accessed: 2024-07-24.
URL: *https://github.com/pulumi/deploy-demos/tree/main/pulumi-programs/drift-detection*

Qiu, Y., Kon, P. T. J., Xing, J., Huang, Y., Liu, H., Wang, X., Huang, P., Chowdhury, M. and Chen, A. (2023). Simplifying cloud management with cloudless computing, *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, Association for Computing Machinery, New York, NY, USA, p. 95–101.
URL: *https://doi.org/10.1145/3626111.3628206*

Spacelift (2024). *Drift Detection in Spacelift Stacks*. Accessed: 2024-07-24.
URL: *https://docs.spacelift.io/concepts/stack/drift-detection*

Vaquero, L. M., Rodero-Merino, L., Caceres, J. and Lindner, M. (2009). A break in the clouds: towards a cloud definition, *SIGCOMM Comput. Commun. Rev.* **39**(1): 50–55.
URL: *https://doi.org/10.1145/1496091.1496100*