SAVONIA

University of Applied Sciences

THESIS – BACHELOR'S DEGREE PROGRAMME
TECHNOLOGY, COMMUNICATION AND TRANSPORT

IAC LANDING ZONE IN AWS

AUTHOR/S Vittorio Frignati

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Internet of Things	
Author(s) Vittorio Frignati	
Title of Thesis IaC Landing Zone in AWS	
Date 30.05.2024	Pages/Number of appendices 33/2
Client Organisation /Partners Nordcloud	

Abstract

In the dynamic real of IT, cloud computing has become the cornerstone for innovation and improved operational efficiency across various industries. As organizations migrate to the Public Cloud, the need for managing increasingly complex infrastructures securely and efficiently had grown. This thesis delves into the deployment of Infrastructure as Code (IaC) via Terraform, focusing on the development of a functional Landing Zone in AWS. This tool aims to ensure a structured environment for deploying, managing, and maintaining cloud resources while aligning with security, compliance, and governance frameworks.

This study explores the how IaC can automate the setup and management of integrations between AWS, Azure, GitHub, and Terraform Cloud, focusing on building on top of an extremely wide and complex environment already deeply rooted in these resources. Particular attention has been paid to how Terraform can be made simple and abstracted, reducing the need for matter experts, and how CI/CD pipelines can be used to support a living product such as the Landing Zone without needing permanent credentials.

The results highlight the role of IaC and the Landing Zone in enhancing security, efficiency, and scalability when deploying AWS Accounts and resources. Automating cloud management via Terraform can significantly decrease the time from ticket to deployment while reducing manual oversight and, therefore, potential errors.

The insights of this thesis aim to contribute to the ongoing conversation on the balance of complexity and efficiency that IaC offsets, deepening the understanding of strategic cloud infrastructure management, and offering a solid starting point for any enterprise looking to build a fully automated Landing Zone in AWS.

Keywords

IaC, Landing Zone, AWS, Cloud engineering, CI/CD, Terraform, Terraform Cloud

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

In the rapidly evolving landscape of digital technology, cloud computing has emerged as a pivotal force driving innovation and efficiency across all sectors of industry. As organizations increasingly migrate to cloud platforms, the need to manage complex infrastructures efficiently and securely has become paramount. This has given rise to the practice of Infrastructure as Code (IaC), a critical development in the management of scalable, reproducible, and secure IT environments. Among the cloud services providers, Amazon Web Services (AWS) stands as the market leader, offering a diverse ecosystem of services.

This thesis will explore the strategy employment of Terraform as an IaC tool, focusing on creating a functioning Landing Zone, a key concept in modern cloud architectures. A Landing Zone provides a structured environment for deploying and managing cloud resources, all the while ensuring that the provisioned are aligned with the organization's security, compliance, and governance framework. IaC will further be explored with regards to how effectively it can be leveraged to orchestrate and automate the setup of complex integrations between multiple platforms and cloud providers in large-scale enterprise applications. We will touch on how the solution was implemented by analysing architecture diagrams, and proposing sample Terraform configurations, GitHub folder structures, and continuous integration and continuous delivery/deployment (CI/CD) pipelines through Terraform Cloud.

We will examine the role of IaC in enhancing operational efficiency, scalability, and security within AWS cloud environments. By automating the setup and management of cloud resources through Terraform-configured Landing Zones, organizations can rapidly deploy consistent and secure infrastructures. These Landing Zones not only facilitate seamless scalability to accommodate growing workloads but also enforce compliance and security standards automatically across all deployments. This approach dramatically reduces manual oversight and potential errors, leading to more robust and efficient cloud operations. The insights derived from this study aim to deepen the understanding of strategic cloud infrastructure management, particularly for enterprises utilizing AWS as their primary cloud service provider.

While some elements of the Landing Zone, such as GitHub or Terraform Cloud, can be replaced with ones of similar use and purpose, it has been designed to solve challenges in the customer's pre-existing environment, thus focusing heavily on managing a large number of AWS accounts via Terraform and deploying resources via Terraform Cloud. The scope will thus limit itself to the customer's environment, however, due to NDA limitations, we will be unable to deep dive into the actual code that powers the various pieces of automation, thus shifting focus to a higher level of analysis.

2 AMAZON WEB SERVICES

According to its website, AWS defines itself as "the world's most comprehensive and broadly adopted cloud, offering over 200 fully featured services from data services globally" (Amazon Web Services). While this definition is accurate, it is too broad to understand what brought about its widespread success. AWS offers pay-as-you-go pricing, enabling small and large businesses to focus on building instead of maintaining. AWS has an extensive global network of data centres, helping to reduce latency, improve performance and scalability, and meet regional compliance and data residency demands. Lastly, AWS is committed to offering secure infrastructure through a broad bag of cloud-native security tools and data encryption at rest and in transit.

On the discussion of pricing for public cloud vs on-premises, the consensus tends to favour the public cloud for both return on investment and overall pricing. However, cloud customers must look beyond the price; Makhlouf (Makhlouf, 2020) finds that:

> Cloud has high asset specificity due to change management costs, meta services and business process reengineering costs, Cloud has a considerable level of uncertainty asking for managing contracts, investing on monitoring and consciously reviewing the legal compliance, and Cloud has high transaction frequency, which compensates the needed investments triggered by uncertainty and asset specificity.

Overall, especially in the case of larger enterprises, the flexibility and the drop in hardware maintenance costs prove to be cost-beneficial. Gitnux finds that "84% of companies that adopt cloud reduced computing costs" and "Companies using cloud computing save 20% annually on infrastructure costs" (Gitnux, 2024).

2.1 AWS Organizations

AWS Organizations is a comprehensive account management service designed to consolidate multiple AWS accounts into a single organisation. This service is the starting point for any enterprise engaging in a multi-account setup, offering essential benefits regarding account centralisation, consolidated billing, compliance, and security. Its primary function is to create a structure where several accounts, which can be thought of as independent resource silos, are grouped together in Organizational Units (OUs). OUs function as logical subdivisions of an AWS Organization, enabling enterprises to manage policies, access, and configurations more effectively across clusters of AWS accounts. (Amazon Web Services).

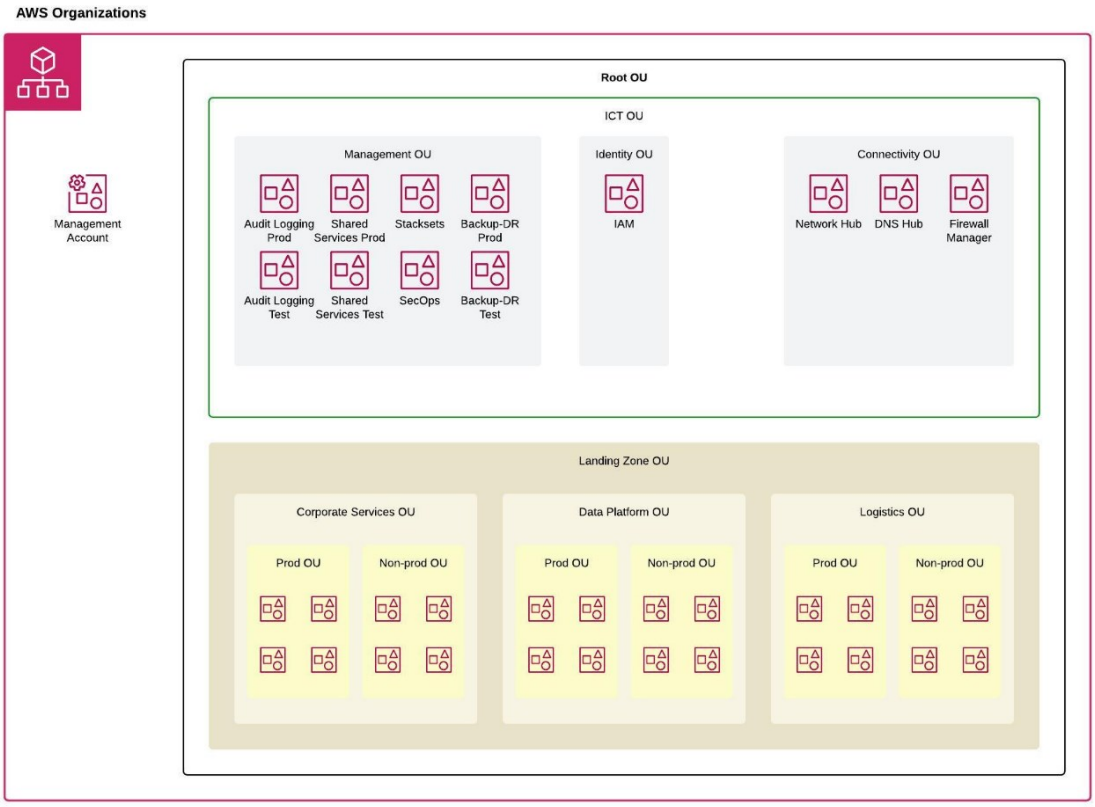

Figure 1. AWS Organization split into OUs. This structure is managed by the Landing Zone repository.

Figure 1 shows how the customer's environment has been segmented into siloed OUs. On the top, one can see the Information and Communications Technology (ICT) OU, a logical grouping of the accounts containing multi-account or organization-wide tools The Cloud Platforms teams owns the ICT OU and is responsible for the Landing Zone, Core Infrastructure, and most cloud native tools utilise across the whole enterprise. This OU is further split into other sub-OUs, grouping accounts by purpose. The AWS Accounts containing logs, various shared tools, backups, security operations, and the AWS delegated administrators for AWS Stacksets are grouped under the Management OU. As of today, the only account under the Identity OU is the Identity and Access management (IAM) account, which is responsible for Single Sign-On (SSO) authentication to the AWS console and is the sole entry point for IAM role-based access to deployments through the pipeline in any of the other accounts. Lastly, the Connectivity OU hold the Network Hub account, which is responsible for connectivity between accounts and towards the public internet through a shared transit gateway; the DNS hub, which holds all the Domain Name System (DNS) records and zones the organisation needs; and the firewall manager, which holds the AWS Shield configuration. On the bottom, there is the Landing Zone OU, a grouping of the various business units, three of which are shown as an example, though the actual number is around ten. Each of them is managed by one or more different teams of the customer's in-house employees.

OUs do not have any inherent property as such, they simply serve as a grouping of the accounts. The Landing Zone OU and ICT OUs are not directly linked, but the accounts contained within are.

Many of the accounts contained in the latter have resources shared from the centralised tools accounts with AWS Resource Access Management (RAM), such as the Transit Gateway in the Network-hub. AWS RAM can share resources with either individual accounts or entire OUs. In any large enterprise is it much easier to utilise the OUs to grant access to multiple AWS accounts at once.

AWS Organizations also offers a feature to consolidate billing across the entire enterprise. This simplifies the view and management of spending by centralising billing and usage information into one singular view. It enables the management of complex budgets more effectively by allowing organisations to track and optimise their cloud spend across any number of accounts without needing third-party tools. (Amazon Web Services).

2.2 Security in AWS

In the context of AWS, security is managed on two levels: within the confines of individual AWS accounts and across the overarching AWS Organization. In terms of security, this thesis will focus largely on the overarching tools. Individual accounts, generally, are handed out to development teams, and they are responsible for securing the application's infrastructure. AWS operates on the shared responsibility model, meaning AWS is responsible for securing the hardware, software, networking, and facilities where the various services are hosted while the user is in charge of maintaining, configuring, and accessing said services securely. (Amazon Web Services, 2023).

The following is a list of the significant security measures put in place in the customer's environment:

- Service Control Policies
- AWS CloudTrail
- AWS Config

Service Control Policies (SCPs) are the foremost tool for managing security and compliance across multiple AWS accounts; up to five may be attached to an individual AWS account, though this is generally discouraged due to the complexity it would create on a large scale, an AWS OU, or the entire AWS Organization. SCPs are JSON policies that limit what actions may be taken by whitelisting some actions or by blacklisting others, effectively functioning as guardrails that apply to all users and roles in scope. (Amazon Web Services).

Here are the six most common SCPs:

- Deny leave the organisation
  This ensures no rogue account is allowed to leave the AWS Organization and its control.
- Deny creation of IAM Roles & Users with specific permissions.
  This ensures that no user can raise their permissions above what they have been granted.
- Deny access to AWS in certain regions.
  This ensures that no rogue resources are created in regions out of scope.
- Require EC2 instances to use a specific type.
  This ensures that no VMs that are too big and expensive are created, e.g., in a Dev account.

- Deny resource sharing outside of the organisation.
  This ensures that no external accounts may access resources without control.
- Require certain tags on resources.
  Tags are the best way to identify the purpose, source, and user of a certain resource. They are essential in billing, management, and applying changes to these resources, and they are often the only way to distinguish one resource of a type from another.

SCPs are crucial for adhering to the National Institute of Standards and Technology (NIST) Cybersecurity Framework 2.0 (National Institute of Standards and Technology, 2024), specifically its core components of access control, data protection, incident response, and continuous monitoring and improvement. By implementing strict guardrails on user actions and permissions, SCPs ensure compliance by consistently applying security controls across all AWS accounts. These policies can restrict unauthorized access and actions and bolster data security by limiting where data can be stored and ensuring its protection. Additionally, SCPs support effective incident response by preserving critical settings and enhance ongoing security measures through enforced policy compliance. This proactive approach to managing access and permissions is in line with the protective and preventive measures advocated by NIST Cybersecurity Framework 2.0.

AWS CloudTrail plays a vital role in security and compliance monitoring across an organisation's AWS accounts. Centralising CloudTrail logs across multiple accounts in AWS Organizations simplifies management, enhances security, and supports the creation of a complete and compliant audit trail. When logs are centralised, CloudTrail consolidates the visibility of all user activities, API usage, and system actions across the entire organisation, enabling rapid response to incidents and anomalous behaviours.

Following the centralisation of logs, it naturally follows to create backups of such crucial data in a separate, secure account. This practice safeguards the logs from accidental or malicious deletion and ensures they are available for forensic purposes in case of a security breach. Backup accounts and the logs' storage mediums should always be configured with strict access controls, minimising the risk of unauthorised access and the alteration of log data. Logs are typically grouped into a cloud object storage, encrypted server-side, and accessible only with read-only permissions. Most often, in AWS, the storage medium of choice is an AWS Simple Storage System (S3) bucket, as shown in green in Figure 2.

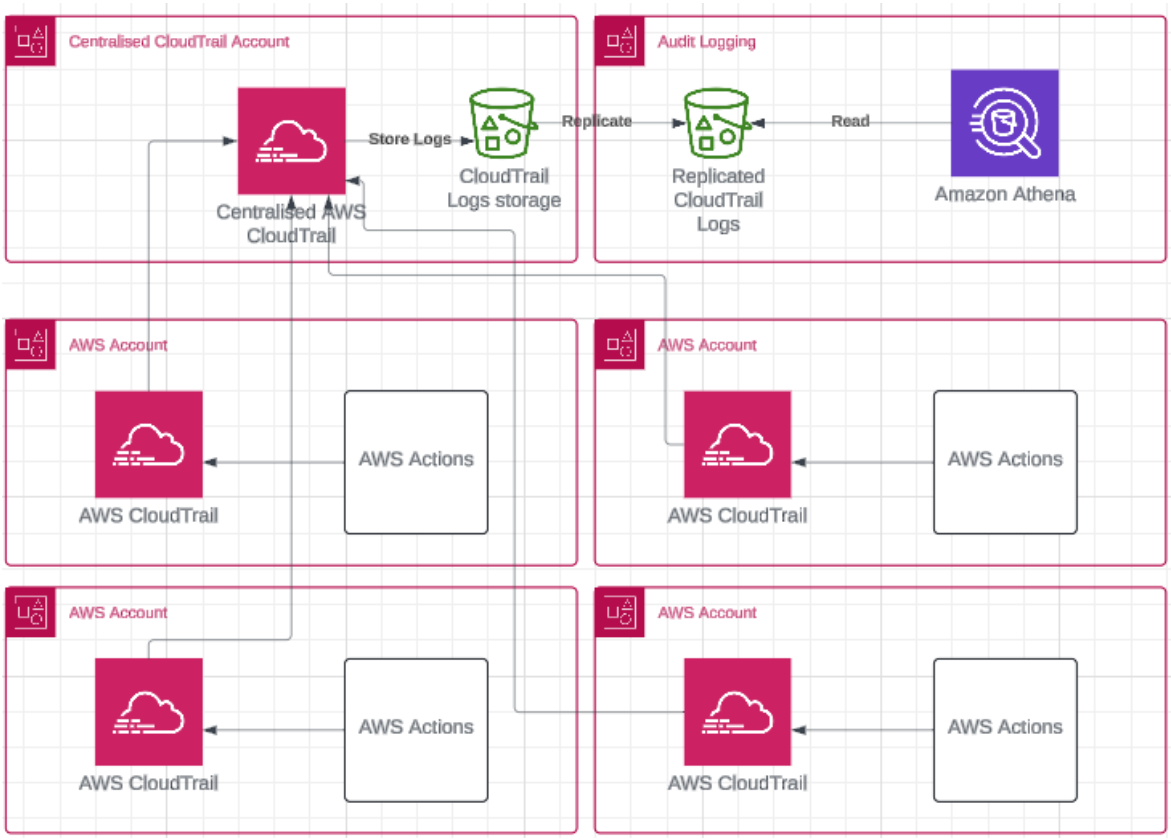

Figure 2. AWS centralised CloudTrail with log storage and replication.

Figure 2 illustrates a sample AWS CloudTrail log management system, showing the integration and centralisation of logging mechanisms across multiple AWS accounts. At the bottom of the diagram, four generic AWS accounts, which could easily be any other number, are configured to generate and manage their CloudTrail logs. On the top, the diagram presents the two specialised AWS accounts taking part in this integration. On the top left, the centralised AWS CloudTrail account is tasked with aggregating the logs from all four accounts below. It acts as a centralised repository where all logs are pushed to a designated S3 bucket. On the top right, the audit logging account holds a replicated version of the S3 bucket from the centralised CloudTrail account.

AWS Config is another powerful tool in AWS Organizations that enhances security by providing a detailed view of the configurations of AWS resources existing in the enterprise. By continuously monitoring and recording configurations their configurations, AWS Config can identify resources that are not compliant with the organisation's settings. If misconfigured resources or unauthorised changes are detected, the tool can send a notification to the designated users or trigger a previously set-up automatic remediation.

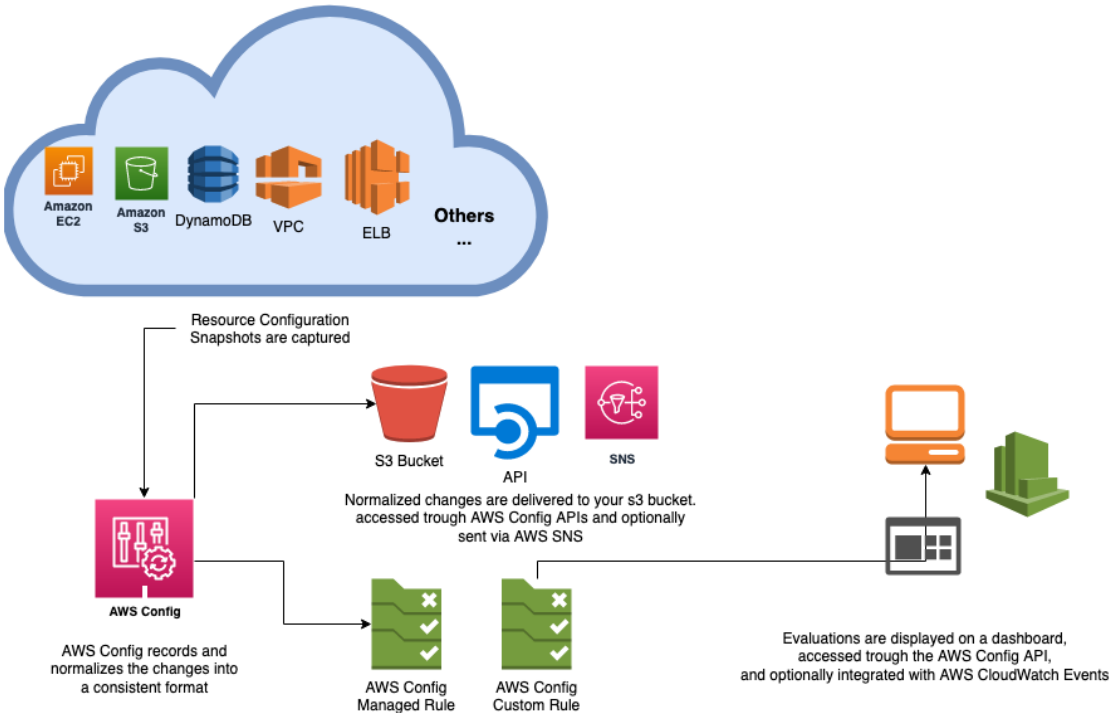

Figure 3. Workflow of AWS Config Managed Rule (Silva & Snehal, 2020).

Figure 3 shows various AWS services, shown in the cloud, are having their configurations checked by AWS Config, which then stores these records into an S3 bucket. AWS Config measures these records against rules, a rule is a set of instructions to which a resource configuration must adhere, and, if configured, takes automated action to remediate deviations. If, for example, an S3 bucket had a policy allowing non encrypted data to be stored within, AWS Config would automatically update it to prevent unencrypted data to be stored there.

2.3 Authentication in AWS

When establishing authentication in AWS, it is essential to distinguish between machine users and human users. In both cases, assigning a specific Identity and Access Management (IAM) role based on the principle of least privilege remains a recommended practice. By limiting permissions only to the required resources, this method ensures that no unauthorised actions may be taken, thus minimising security risks and improving control over the factor of human error. IAM Authentication can be through IAM Users, IAM Roles, or a combination of the two; users have a pair of permanent credentials used to access the AWS API and AWS console, while roles provide only temporary credentials, usually lasting up to 12 hours. The former should only be used in specific use cases where a role would not be a suitable solution.

Human users utilise the IAM Identity Center, an AWS native tool that grants SSO permission to anyone with credentials. This service uses IAM roles behind the scenes, authenticated through an Identity Provider, in the customer's case, Azure Entra ID.

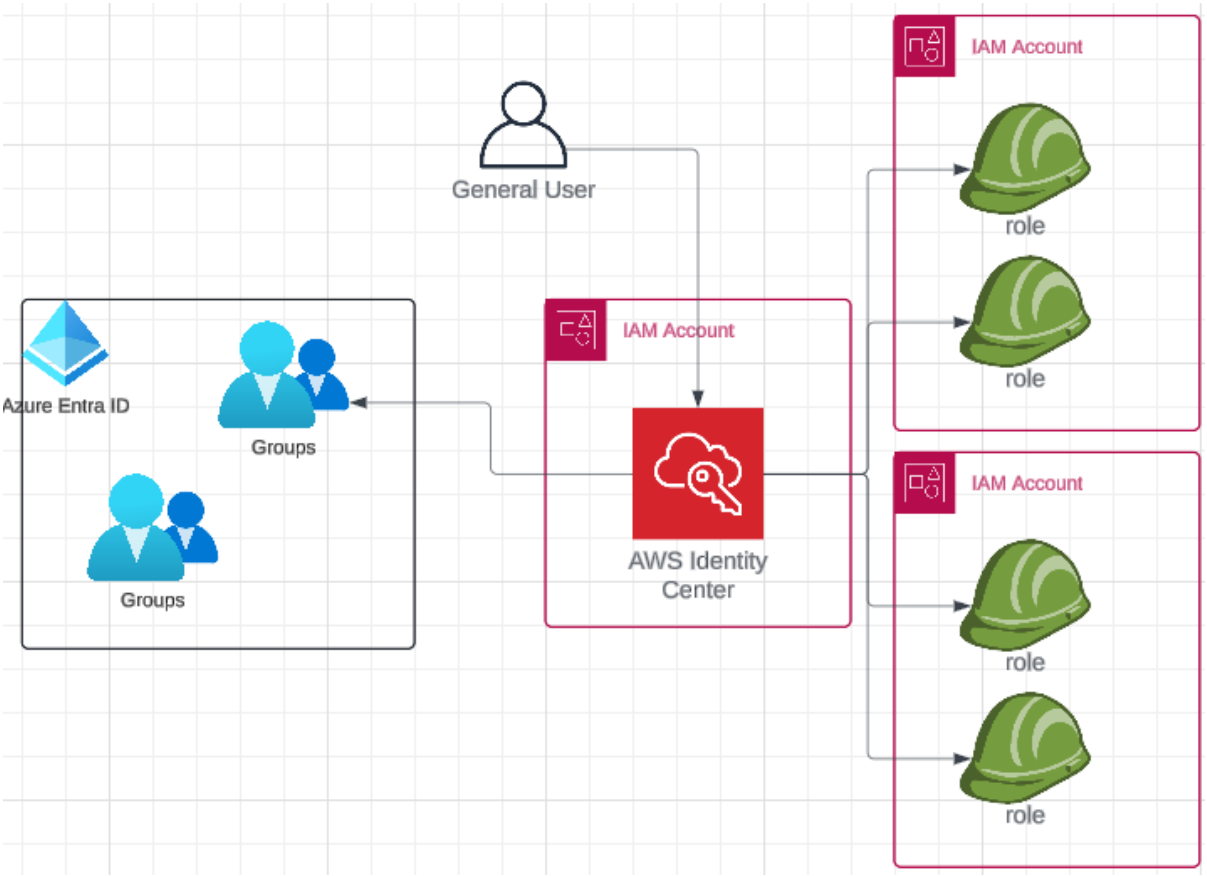

Figure 4. AWS Identity Center (Formerly AWS Single Sign-On) with Entra ID as IdP.

In Figure 4, the flow of access is visible: a human user will access the AWS Identity Center login page, which usually has a customised URL, they will be redirected to authenticate against the organisation's Azure Entra ID, then, based on which group their Entra user belongs to, temporary credentials will be granted to the pre-configured AWS Accounts or resources. This eases IAM management by centralising permissions for human users at one single point. It ensures no permanent credentials are stored in AWS, as it is seamlessly integrated with Azure Entra ID with SAML 2.0.

A custom-built solution has been developed for machine users; it involves integrations between AWS, the public cloud of choice, GitHub, the version control system (VCS) and automation medium, and Terraform Cloud, the pipeline for major resource deployments.

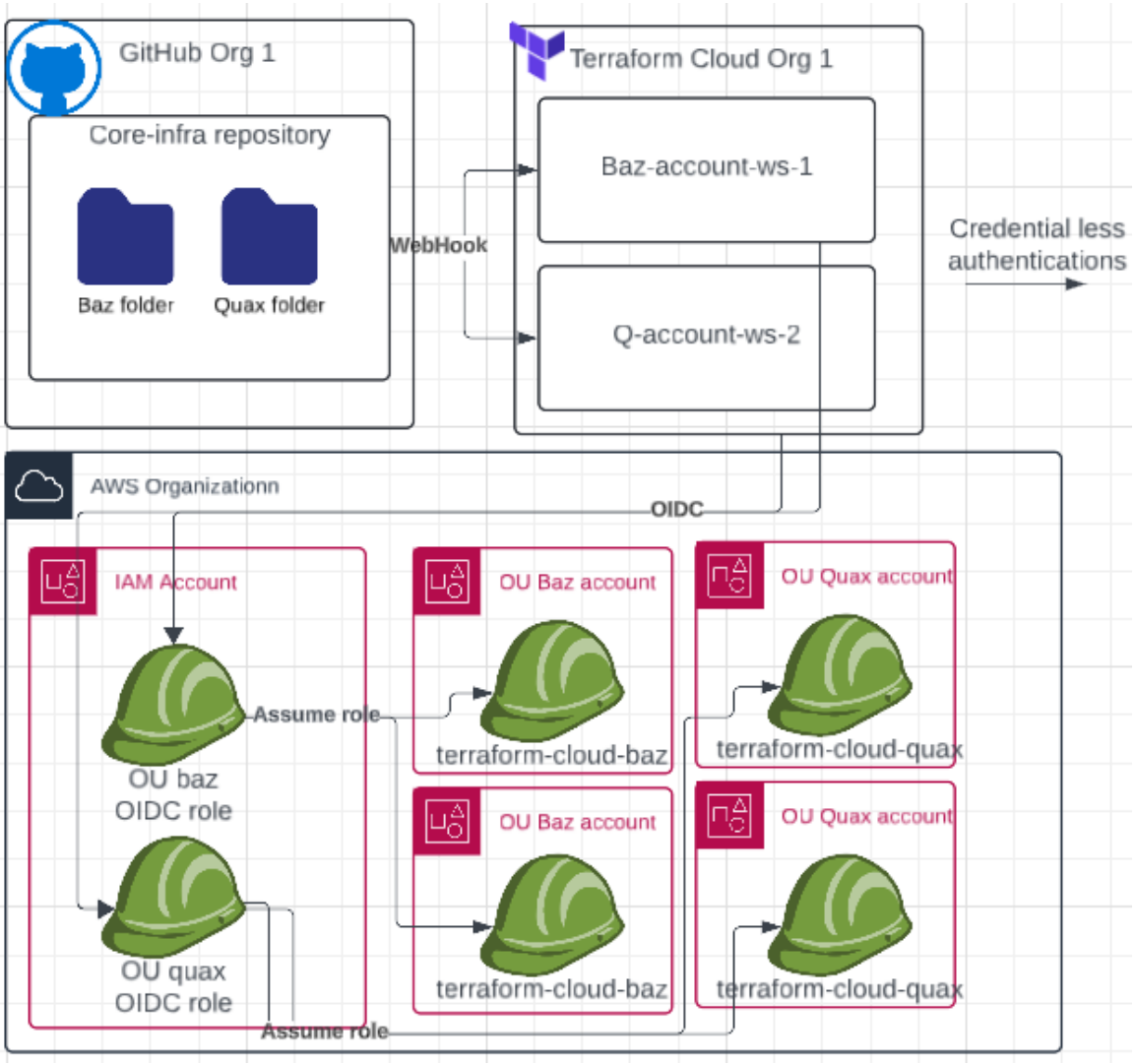

Figure 5. Authentication flow for code to resources pipeline.

As shown in Figure 5, the authentication flow for pipelines begins from a GitHub repository; in the example, one can see the core-infra repository, which is triggered by a human user pushing code and opening a pull request (PR). A PR opening will trigger the webhook between GitHub and Terraform Cloud; said webhook works on a payload delivery basis, meaning specific configured actions will trigger the linked Terraform Cloud workspaces and initiate a plan. The Terraform Cloud plans require being authenticated against AWS to check the status of existing resources, resolve interdependencies, and create new resources. As shown in the diagram, this authentication is done through OpenID Connect (OIDC). The Terraform Cloud workspace holds an AWS role Amazon Resource Name (ARN), stored in a variable available at the environment level, and a Boolean value instructing it to proceed with OIDC authentication, as shown in Figure 5.

Workspace variables (2)

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set precedence.

Key	Value	Category	
TFC_AWS_PROVIDER_AUTH	true	env	•••
TFC_AWS_RUN_ROLE_ARN	arn:aws:iam::012345678910:role/terraform-cloud	env	•••

Figure 6. Terraform Cloud workspace variables configuration.

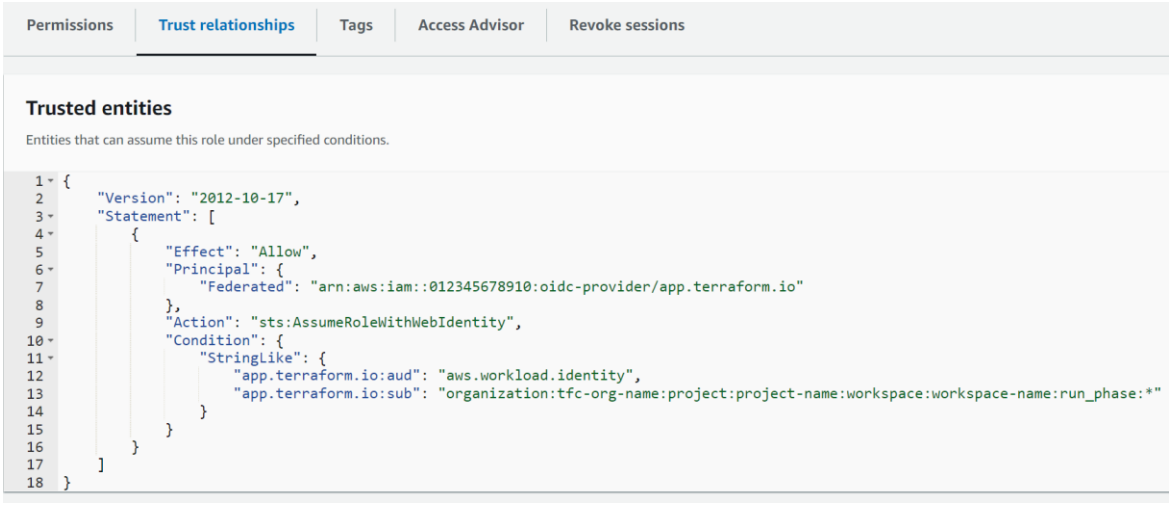

Figure 7. AWS Role trust relationship configuration.

For the TFC workspace to be able to authenticate, AWS needs to be configured correctly. Following the instructions by HashiCorp, the company behind Terraform (TF) and Terraform Cloud, the next step is to create an OIDC provider in AWS (HashiCorp). According to the definition on AWS' documentation page (Amazon Web Services):

> *IAM OIDC identity providers* are entities in IAM that describe an external identity provider (IdP) service that supports the OpenID Connect (OIDC) standard, such as Google or Salesforce. You use an IAM OIDC identity provider when you want to establish trust between an OIDC-compatible IdP and your AWS account.

Once the provider has been configured, the AWS role should be granted a Trust relationship policy akin to the one in Figure 6, where the "Federated" value, on line 7, should be one's OIDC provider's ARN, and the "sub" value, on line 13, should have its value adapted to one's Terraform Cloud organisation. The created AWS role should follow best practices and have as few permissions as possible. Ideally, it should only be able to assume another role in the account designated for resource deployment.

Once all the above has been configured correctly, upon a triggered run, the Terraform Cloud workspace will assume the created AWS role in the target account, in the customer's case, the IAM account, as shown in Figure 4. Lastly, Terraform, meaning the codebase, should be configured to do

one more hop via AWS Assume Role at the provider declaration level, as shown in the Figure 8.

```
provider "aws" {
  region = var.region
  assume_role {
    role_arn     = "arn:aws:iam::109876543210:role/terraform-cloud"
    session_name = "TerraformCloud"
  }
}
```

Figure 8. AWS Provider configuration in Terraform.

This final piece of code is essential in that it allows the customer's organisation to more closely follow the principle of least privilege, ensuring that authentication has a single point of access, the IAM account, inside of which the first assumed role has no permissions to modify resources or configurations while retaining a certain degree of flexibility in the target account. Multiple provider declarations are possible in the same codebase, but the reason this hop is possible is thanks to the Terraform Cloud workspace assuming the first role while being initialised and the second role while planning.

3 TERRAFORM & TERRAFORM CLOUD

3.1 Infrastructure as code

Historically, IT infrastructure was managed manually, and system administrators physically set up hardware and managed and maintained software. This process was time-consuming and, most importantly, prone to human error, thus leading to inconsistent configurations and hard-to-identify errors. As the complexity of systems grew, particularly with the virtualisation of cloud computing, new issues started arising, first among them "configuration drift", where different environments became dissimilar over time despite having been supposedly configured in the same way, leading to the well-known "it works on my machine" syndrome. Scripting was the first solution, and while a great fit at the time, it had its limitations, often being environment-specific and requiring significant maintenance.

IaC is a refined form of scripting where a standardised language, code, is used to deploy the same exact infrastructure to multiple environments. Since it is, as mentioned, code, there are two main ways to approach the problem: imperative and declarative. The former provides the computer with explicit instructions on how to perform a task, while the latter abstracts the instruction by focusing on the outcome rather than the procedure.

3.2 Terraform

Terraform, a coding language defined in terms of a syntax called HCL (HashiCorp), is part of the declarative family. Developed by HashiCorp, it is one of the most prominent IaC tools available. Its success is primarily due to its ability to describe the desired end state of infrastructure through code, meaning that the user does not need to know how the resources are provisioned, only the desired end result. Behind the scenes, Terraform builds a ”resource dependency graph” (HashiCorp), which lets Terraform know what the best order of operations is to ensure all interdependencies are considered.

Terraform was born open source and has thus cultivated a strong community that contributes to its development and constant improvement. Thanks to this, the provider ecosystem has broadened, and Terraform supports a vast number of clouds, such as AWS, Azure, Google Cloud, and Alibaba Cloud; it also supports several platforms, such as GitHub, Terraform Cloud, Kubernetes, Helm, and many more. Any number of the above-mentioned can be used simultaneously, making it an excellent tool in multi-cloud environments. One might also develop their own custom-made provider and manage any resource through Terraform.

3.3 Terraform Modules

In Terraform and Infrastructure as Code (IaC) approaches, modularity and reusability are key principles that significantly enhance consistency and simplify maintenance. By employing modular designs, Terraform allows developers to create reusable components, known as modules, that can be deployed across different environments. This practice ensures uniform configurations, further pre-

venting the inconsistencies and drifts caused by many different configurations and streamlines the maintenance process. When updates or changes are required, modifying a single module can propagate these changes across all deployments where the module is utilized.

In our case, the customer has an ever-growing list of maintained modules, the majority of which are developed, maintained, and updated by the Cloud Platform team. It is worth noting that the Landing Zone codebase is also a module.

3.4 Terraform in the customer's environment

While several Infrastructure as Code (IaC) tools are available native to public clouds—such as AWS's SDK and CloudFormation and Azure's Bicep—the customer elected to standardise nearly every piece of infrastructure through Terraform. This choice centres Terraform as the primary source of truth across multiple platforms, including GitHub, AWS, Azure, Entra ID, and Terraform Cloud. Native tools like AWS's SDK and CloudFormation, or Azure's Bicep, were found to be too limiting and added unnecessary complexity to an already challenging environment by creating the need to adopt, learn, and become proficient in multiple tools. Terraform was run through a simple wrapper service through GitHub before moving to Terraform Cloud as a pipeline, so the company has a history of use, further solidifying it as the go-to tool.

3.5 Terraform Cloud

Terraform Cloud is a managed service offered by HashiCorp designed to promote efficient and secure infrastructure management. It stands out in the world of infrastructure as a code (IaC) tool. This service eliminates a significant portion of the operational overhead generated by manually run Terraform by maintaining the infrastructure necessary for state management and resource deployments. One of the critical advantages of Terraform Cloud is its enhanced collaboration features. The platform is specifically designed to support team-based workflows, providing a secure and consistent environment where users can collaborate on infrastructure projects. Terraform Cloud further enhances ease of use by natively supporting stare-sharing between workspaces, thus enabling modular infrastructure setups where outputs from one workspace can be used as inputs in another. This capability promotes simple dependency management and reusability across projects.

Security is another critical aspect of Terraform Cloud, with integrated features that include strong access controls, secret management, a private data registry, and Sentinel. These capabilities enable users to define precise access permissions and securely store sensitive information such as API keys or passwords. Moreover, Terraform Cloud trivialises remote operations by executing plans and applying them in a consistent remote environment. This approach ensures that Terraform executions do not depend on individual user environments, leading to more predictable outcomes and nullifying the common problem of inconsistencies between local and shared environments.

Scalability is yet another significant benefit of Terraform Cloud. The platform is designed to scale effortlessly with the needs of a business, accommodating everything from small teams new to infrastructure automation to large enterprises managing complex multi-cloud environments. The limitations imposed on the previously utilised self-hosted Terraform wrapper were major factors in the customer's switch to Terraform Cloud. The ECS cluster would often run out of memory and crash when over-utilised, and it was not suitable to handle an ever-growing enterprise with tens of concurrent run requests. Terraform Cloud seamlessly integrates with GitHub, automating the process of triggering Terraform runs based on VCS events. This integration is pivotal in enhancing continuous integration/continuous deployment (CI/CD) workflows.

3.6 Terraform Cloud in the customer's environment

As touched on above, Terraform Cloud has cemented itself as an essential tool for the customer. As shown in the Figure 9, most of its pipelines are run and maintained by Terraform Cloud workspaces arranged in three organisations.

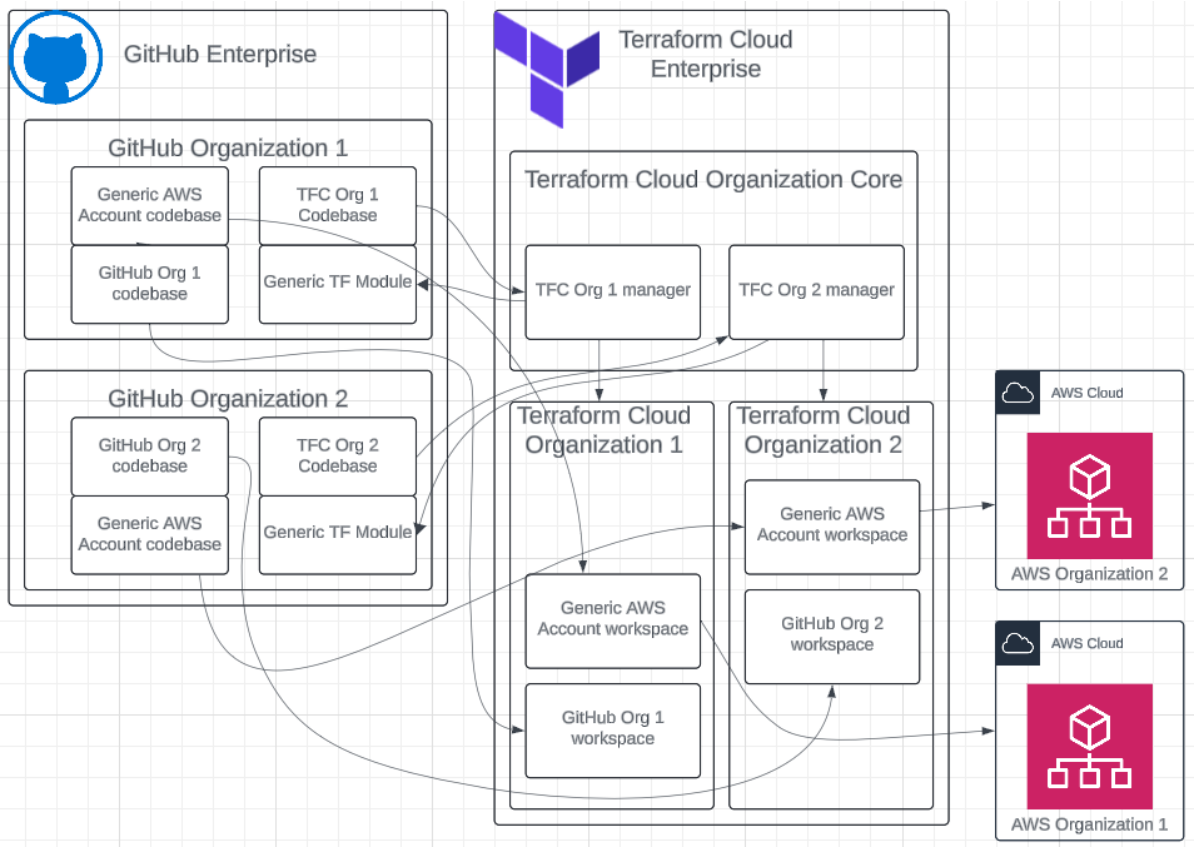

Figure 9. Integrations between GitHub, Terraform Cloud, and AWS.

Figure 9 is another detail of the bigger picture, presented in chunks to ease comprehension. The GitHub enterprise is visible on the left, holding two organisations, each responsible for one of the customer's semi-independent divisions. Each organisation stores the codebase for its configuration, the corresponding Terraform Cloud organisation's configuration, and all AWS accounts in its division. Configurations for GitHub and Terraform Cloud include users, groups/teams, permissions, repositories/workspace, and access and permissions to the aforementioned. The codebase for the AWS ac-

counts includes every resource deployed in said account; the customer is very keen on maintaining Terraform and IaC as the single source of truth.

Terraform Cloud workspaces are grouped by project, logical groupings based on development teams, business divisions, or purpose. Access to Terraform Cloud follows the industry standard of least privilege, ensuring no team has access to another team's workspaces, thus reducing the amount of affected resources, applications, and systems, commonly known as "blast radius", and improving security. Human access to Terraform Cloud is credential-less SSO backed by Azure Entra ID as the IdP.

4 LANDING ZONE AND CORE INFRA

The term "Landing Zone" has two meanings: it encompasses all the overarching tools used to manage, secure, and maintain the AWS organisation and surrounding infrastructure, but it also refers to the GitHub repository responsible for the creation and management of AWS accounts, AWS Organisational Units, and AWS Stacksets. This section will discuss the operational procedures and various automations in more detail.

4.1 Landing Zone repository structure

Figure 10 shows how the Landing Zone repository structure is separated into four folders: "accounts", responsible for the YAML configuration of all AWS accounts existing in the organisation, "organizational_units", holding the configuration of the AWS organisation and Ous; and "stackfiles" and "stacksets", respectively containing the AWS CloudFormation configuration files, and the YAML configuration for their deployment.

```
environments
└── prod
    ├── accounts
    │   ├── business-unit1
    │   │   ├── account1.yaml
    │   │   └── ...
    │   └── ...
    ├── organizational_units
    │   └── organisation.yaml
    ├── stackfiles
    │   ├── stackfile1.yaml
    │   └── ...
    └── stacksets
        ├── stackset1.yaml
        └── ...
```

Figure 10. Landing Zone folder structure (Friend).

4.2 Landing Zone Stacksets deployment

AWS Stacksets is a powerful feature within AWS CloudFormation that enables the deployment of stacks across multiple accounts in a single operation. When paired with AWS Organisation, it becomes an invaluable tool. Whenever a new account joins an OU targeted by a Stackset, either by being created or added, the stacks attached to that OU cascade automatically and are deployed in the account. This is pivotal because it allows greenfield environments with pre-configured resources such as IAM roles for CI/CD pipelines, Defender for Cloud configurations, and more. The IAM For example, the IAM roles in the generic AWS accounts in Figure 3ated by a Stackset.

4.3 Landing Zone account creation

The Terraform codebase takes the YAML configuration files as inputs, renders them usable objects, and performs the designated operations based on the Landing Zone module. The configuration for AWS OUs is not particularly interesting since it is a simple single-file YAML configuration; the AWS account YAML, on the other hand, is more interesting, with it being a more complex object.

```yaml
super-service:
  serviceName: super-service
  global_tags:
    all_accounts_tag: tag_value
    another_one: another_one
  environments:
    prod:
      parent_ou_name: best-ou
      tags:
        account_specific_tag: tag_value
        another_acc_tag_key: another_acc_tag_value
    test:
      parent_ou_name: best-ou-test
      tags:
        account_specific_tag: tag_value_but_test
        another_acc_tag_key: another_acc_tag_value
```

Figure 11. AWS Account YAML Configuration.

Figure 11 shows the YAML configuration for an AWS account. This example file will add two brand new accounts to the AWS OUs specified, in this example, "best-ou" for prod and "best-ou-test" for test. The account names are generated by combining the "serviceName" and the environment, ensuring uniqueness and ease of identification. The file also specifies a list of tags to be given to the individual account, some of which are common to both, listed under "global_tags", while others are unique to each account, specified at the environment level under "tags". Any number of configuration files and folders can exist in the top-level "accounts" folder. The Landing Zone will be able to process them regardless. The customer's environment currently holds over 600 managed accounts.

4.4 AWS Account preparation

After an AWS Account has been created, some settings and options that AWS Stacksets cannot configure need to be added to maintain compliance with the industry's best practices. Such changes are managed by the "bootstrapper" AWS Lambda automation. This Lambda resides in the Management account and utilises an AWS IAM role to perform the necessary actions.

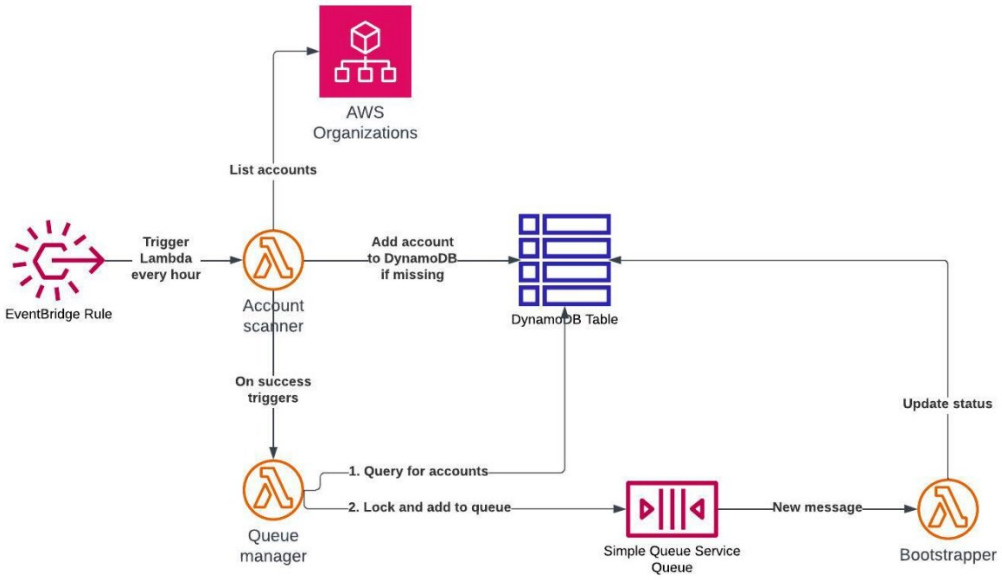

Figure 12. Bootstrapper Lambda design.

Figure 12 shows the execution flow of the automation. Every hour, an AWS EventBridge rule will trigger, prompting the "Account Scanner" Lambda to check for newly joined accounts; if any are found, they are added to a DynamoDB table and marked as not yet fully configured and ready to use -boostrapped- and the next Lambda, the "Queue Manager" will be triggered. The "Queue Manager" will check against the DynamoDB table to see if any accounts should be bootstrapped. Should any be detected, information concerning them is sent in a JSON payload to a Simple Queue Service (SNS) Queue. New messages in the SNS queue trigger the final Lambda, the "Boostrapper". This final piece of automation will perform the list of actions in Figure 13 and, upon successful completion, will mark them as "done" in the DynamoDB table. A DynamoDB table entry with all actions performed is considered bootstrapped and will no longer be picked up by the first Lambda.

3. `aws_account_bootstrapper`

This Lambda function does the bootstrapping by conducting the following steps:

1. Setting the account's IAM password policy (The policy parameters are stored in the SSM Parameter Store on the management account under `/organizations/password-policy`)
2. Setting the account's alias name
3. Setting ECS resource ARN format
4. Deleting all Default VPCs in all regions
5. Setting up and configuring AWS Config
6. Checking Enterprise Support Plan (creating support ticket if needed to enroll the account)
7. Configuring AWS Shield Advanced parameters for the new account

Figure 13. Actions performed by the "Boostrapper" Lambda function.

The Terraform code responsible for provisioning the resources outlined is maintained in the "Core-infra" repository and deployed through Terraform Cloud. Conversely, the code for the Lambda functions is managed and deployed via a GitHub Action from a distinct repository named "app-lambda-bootstrapper". This deliberate segregation of infrastructure and application code minimises the impact of potential errors—commonly referred to as reducing the "blast radius". This separation is significant because, typically, development teams tend not to be well-versed in infrastructure management. By maintaining this separation, the organisation ensures that changes in application code do not inadvertently affect the underlying infrastructure, enhancing overall system stability and security.

4.5 Core Infrastructure

By "Core Infrastructure," we mean the standard set of resources that most developers will request upon account creation. Since these are closely related to the overarching structure discussed thus far, the platform team is responsible for management, configuration, deployment, and maintenance. The available AWS resources included in the package are as follows:

- Virtual Private Network (VPC)
- Multi-tiered subnets following the data segregation standards.
- Access to the public internet or VPCs in other accounts through the shared Transit Gateway.
- DNS Zone delegations in Route 53.
- Shared DNS resolvers.
- AWS Key Management Service (KMS) keys.
- OIDC roles for application deployments through GitHub Actions.
- Ad-hoc resources that fall outside of the development teams' responsibilities.

4.6 Core Infra repository structure

The folder structure of the core infrastructure repository is different from most others, featuring a many subfolders, each of which is linked to a Terraform Cloud workspace.

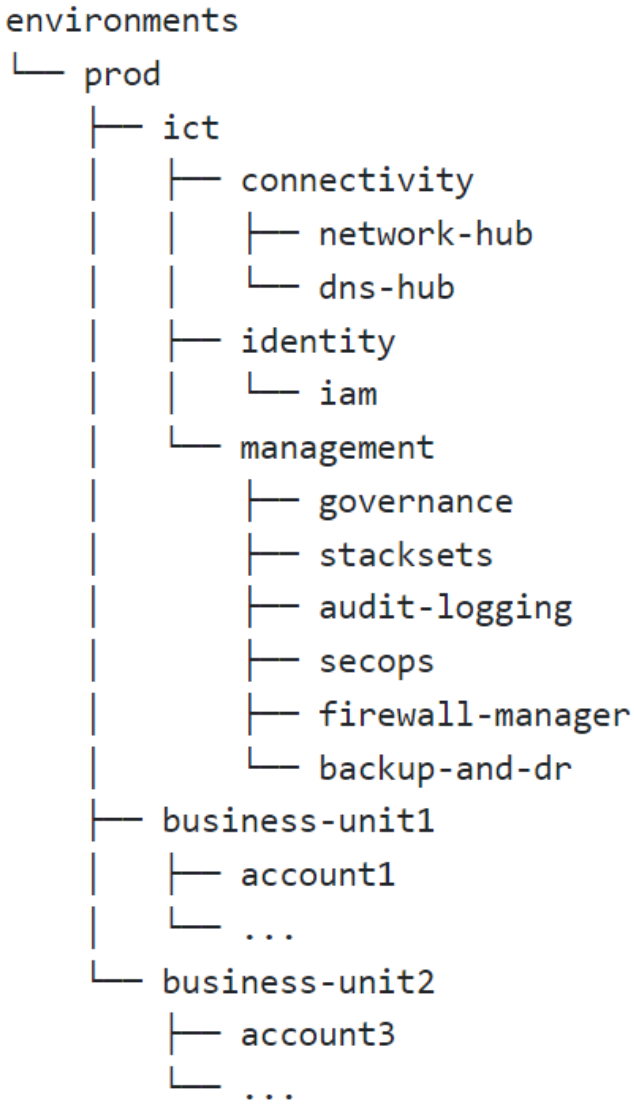

Figure 14. Core Infrastructure folder structure (Friend).

As shown in Figure 14, each account is structured with its own folder containing Terraform code and YAML configuration files. Unlike the Landing Zone's configuration, this repository uses a webhook that triggers different workspaces based on the path of the file edited. This design allows changes to specific folders to trigger plans in corresponding Terraform Cloud workspaces (Figure 15 & Figure 16), affecting different Terraform states. This approach centralises the codebase while siloing resources to isolate the impact of changes and enhance security management. This also takes care of the diverging environments (Infrastructure as code) by having the same code deployed in each application environment with, if needed, different parameters configured in the YAML files. The ICT folder represents a particular group of accounts that contains the AWS resources for deploying and using the overarching tools mentioned thus far. The generic "business-unit" folders contain standard core infrastructure, as described earlier.

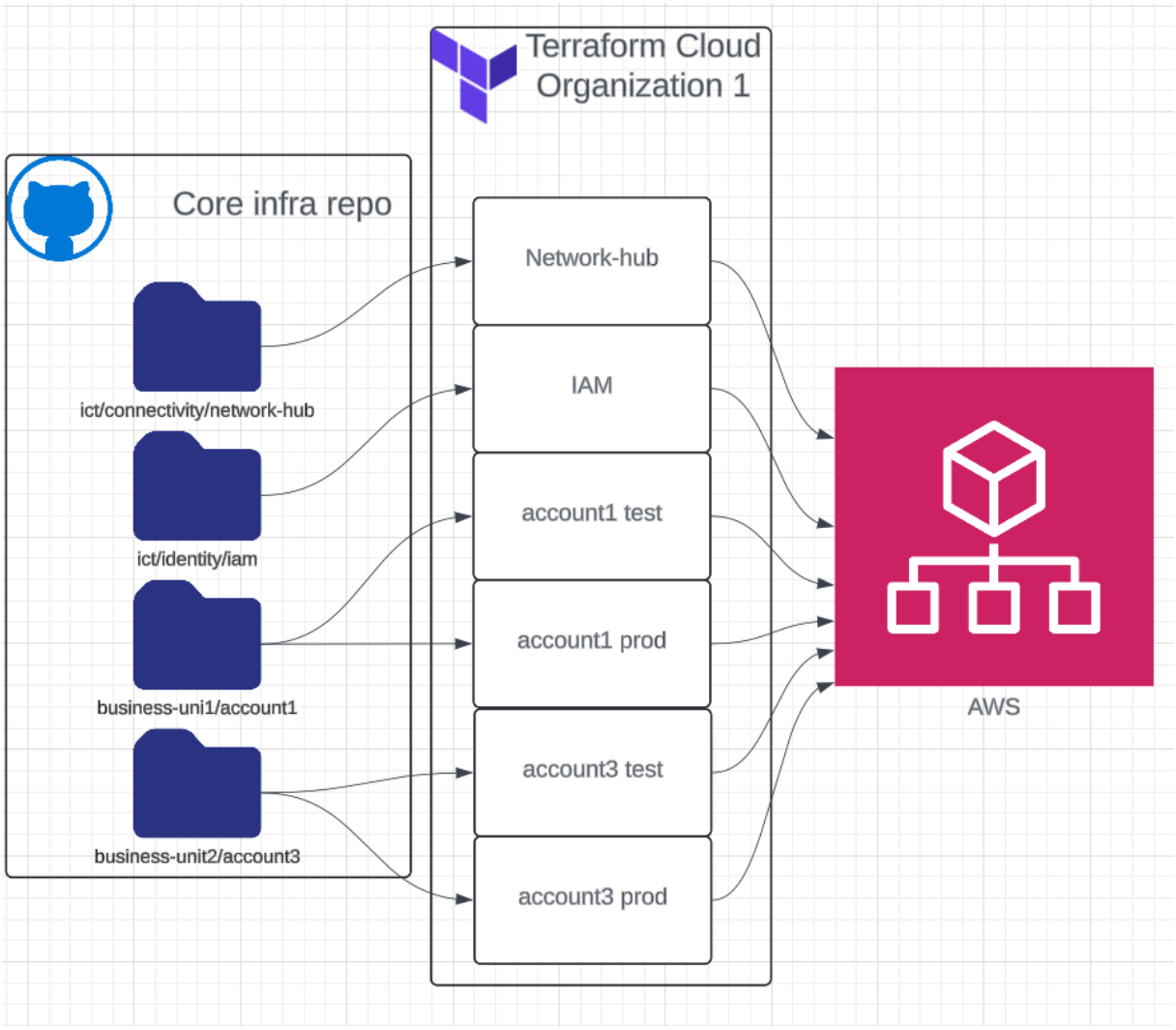

Figure 15. Core Infrastructure repository folders triggering individual workspaces in Terraform Cloud.

Figure 16. Detail of multiple triggered Terraform Cloud workspaces from one PR in GitHub.

4.7 Generic core infrastructure

Continuing with the practice of having human-readable configuration files in YAML, the module is designed to take a complex yet structured object as input (Figure 17) and, based on it, create the necessary resources.

```yaml
service_name: super-service
route53_hosted_zones:
  - prod
  - super-prod
networks:
  vpc:
    tags:
      yaml: isCool
    cidr_block: 10.1.2.0/22
    enable_dns_hostnames: true
    enable_dns_support: true
    internet_gateway: true
    tgw:
      transit_gateway_environment: super-prod
      transit_gateway_add_routes: true
    subnets:
      public:
        is_public: true
        cidr_blocks:
          a: 10.1.2.0/26
          b: 10.1.2.64/26
          c: 10.1.2.128/26
      private-t1:
        tgw_access: true
        cidr_blocks:
          a: 10.1.3.0/25
          b: 10.1.3.128/25
          c: 10.1.4.0/25
```

Figure 17. Sample YAML configuration for a two-tier VPC with Transit Gateway access and two Route 53 hosted zones.

The code is designed to be dynamic and flexible enough to be identical in each deployment, with the sole differences being in the YAML configuration files, such as the one in Figure 17.

4.8 Network Hub centralised internet breakout

In large enterprises, the management of operational expenses is crucial as costs can quickly rack up. Previously, the client maintained three Network Address Translation (NAT) gateways, one per availability zone, in each account that required access to the public internet. This arrangement was extremely expensive, with potential cost savings of tens of thousands of dollars per month. Thus, a revised infrastructure solution was developed. The new setup involves using a shared transit gateway (TGW) that routes traffic to the internet, with a next-generation firewall positioned in front of it.

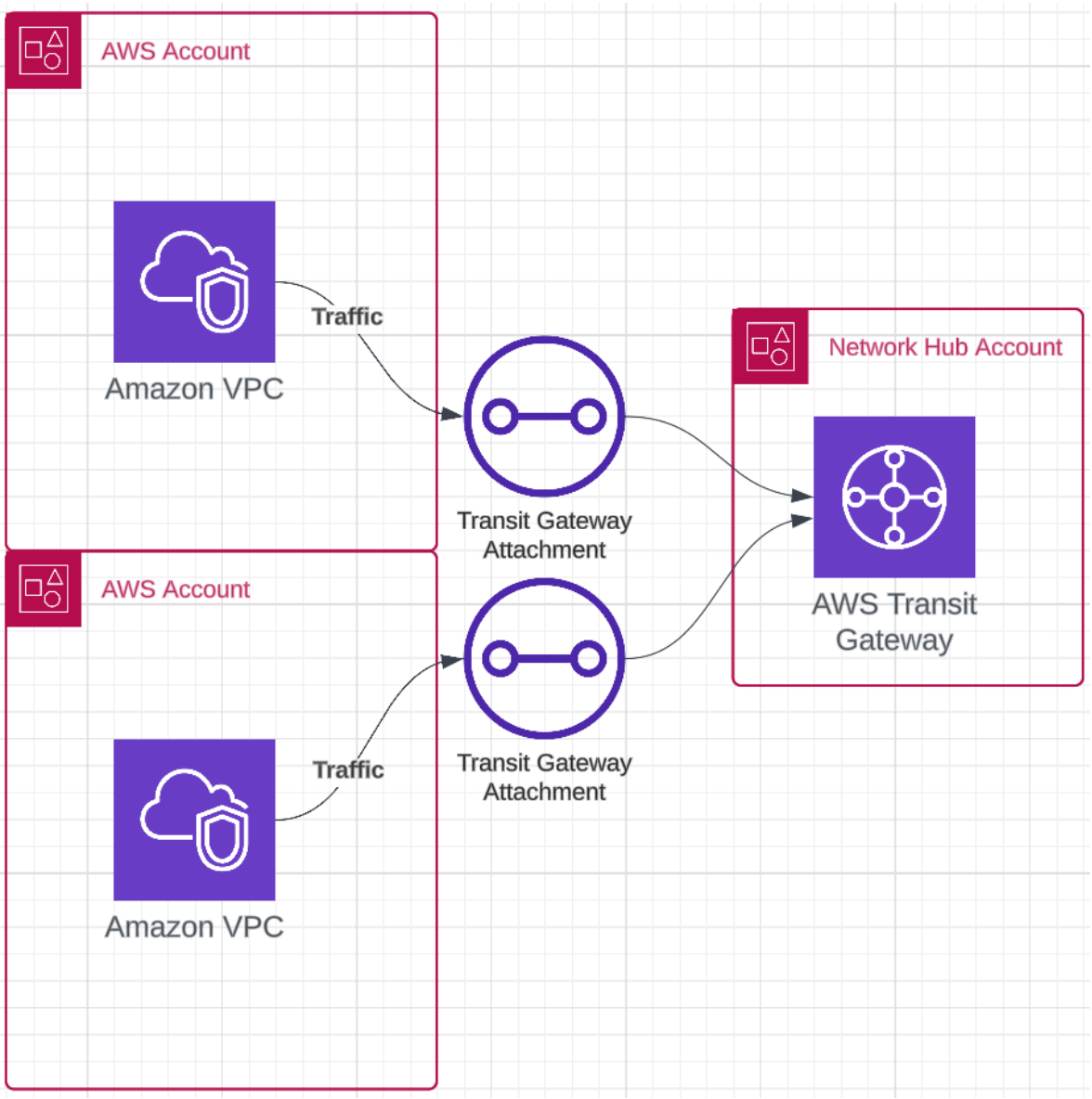

Figure 18. Traffic from spoke AWS accounts to the Network-Hub account.

Figure 18 shows how any number of AWS accounts, two shown, can send their traffic to the shared TGW via Transit gateway attachments. A TGW attachment is a handshake between the TGW and the VPC, allowing the latter to send traffic to the former.

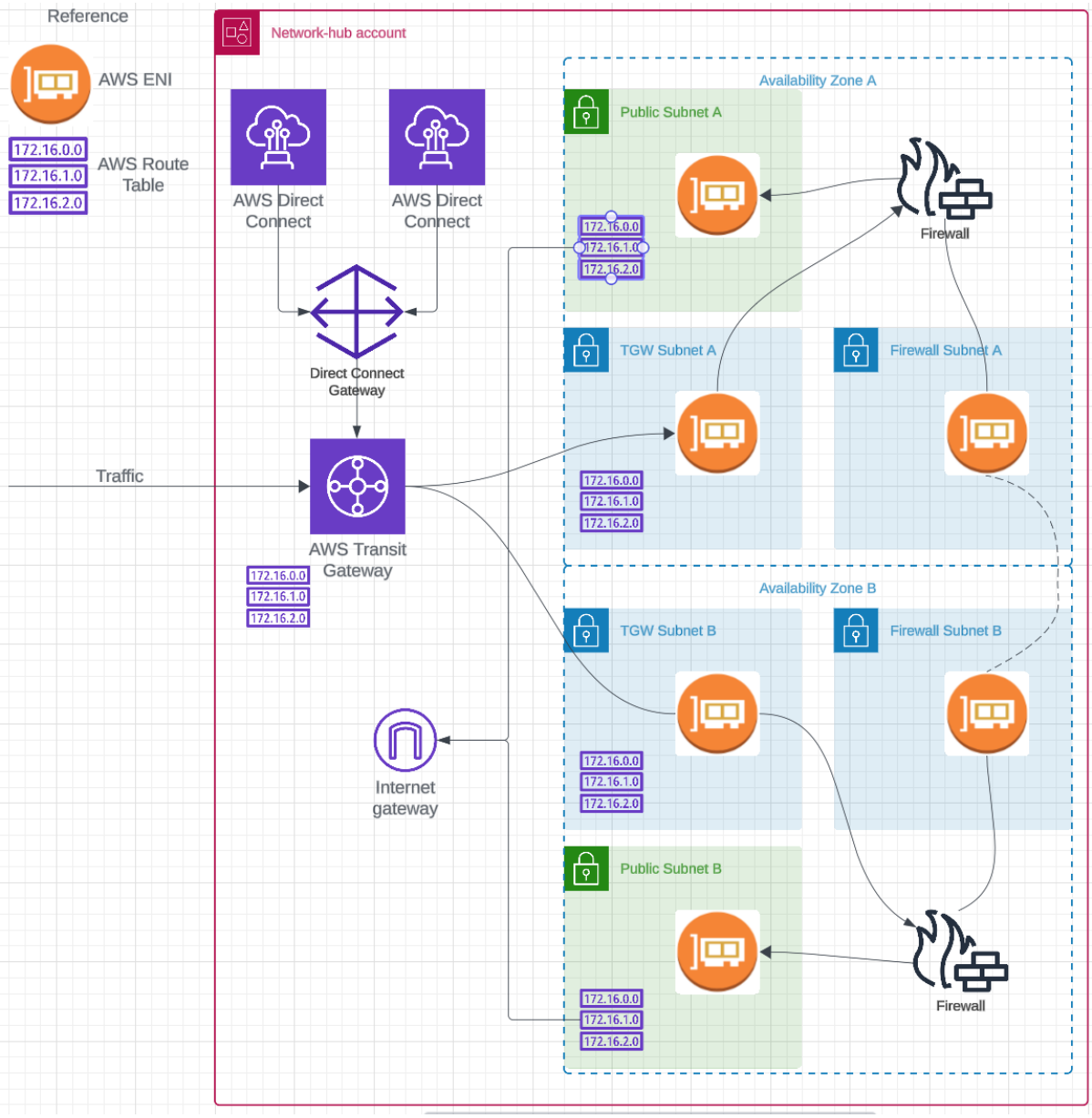

Figure 19. Anonymised internet breakout setup in Network-Hub account, with Elastic Network Interfaces (ENI) shown.

The traffic flow was updated so that all outbound traffic is flowing through an AWS Transit Gateway (TGW). The TGW is shared to each account that needs it through AWS RAM, thus allowing them to send traffic to it. Once traffic has reached the TGW it is redirected to the TGW Subnets in the Network Hub account, as shown in Figure 18, from there it finds a route to the Firewall Subnets where a Next Generation Firewall (NGFW) inspects it and, if approved, sends it to the Public Subnets, from which it is able to reach the public internet through the Internet Gateway.

This solution significantly reduces costs because both AWS NAT gateways and Transit Gateways (TGWs) charge based on uptime and data usage. Sharing one TGW across multiple accounts is far cheaper than using three NAT gateways per account, especially since the volume of traffic remained consistent and its costs are the same whether you use NATs or a TGW. This setup also improved security with the NGFW's stateful packet inspection, integrated intrusion prevention, application awareness and control, and threat intelligence (Cisco).

5 CONCLUSIONS

5.1 Practical implications

The exploration of infrastructure as code in this thesis underlines its transformative impact on modern cloud infrastructure management and deployment. By automating and templating the provisioning of resources, IaC introduces significant enhancements in efficiency, consistency, reliability, and agility. Fully building a Landing Zone through IaC has drastically reduced the time frame between ticket and development, allowing what once took days to be carried out in minutes. Such speed is sure to accelerate development cycles and support dynamically changing business needs.

From a security perspective, controlling all infrastructure through peer-reviewed code reduces the risk associated with human error, malicious actors, and environmental drift. It also integrates essential compliance and security measures by embedding best practices and policies into tangible code. The ability to version control and effortlessly audit configurations strengthens governance and compliance and ensures consistent and repeatable deployments of identical infrastructure, thus becoming an essential pillar in disaster recovery operations and security.

The scalability facilitated by IaC, and particularly by a multi-cloud enabled tool such as Terraform, proves to be invaluable for any organisation looking to expand their infrastructure seamlessly across multiple platforms and environments without excessive overhead.

Ease of maintenance and deployment, however, come at a price: development, added technical complexity and specialised personnel costs. While IaC is a fantastic tool, it is an additional tool for a professional to learn and familiarize themselves with. Proper development of new features takes time since it should be carefully thought out and planned. It is also worth noting that interfacing with a third-party tool, instead of the target cloud directly, can, occasionally, cause some errors and frustrations.

Overall, this thesis finds that the benefits far outweigh the drawbacks of Terraform as an IaC tool. Expertise and careful development are a small price to pay for the automation, scalability, and multi-cloud compatibility that Terraform provides, particularly in large enterprises where complexity would be far worse without an IaC tool.

5.2 Future plans

It is crucial to understand that the IT landscape is constantly evolving, and it is essential to stay up to date with the latest technological advancements. For this reason, the tools mentioned earlier should be considered "living tools," as they are extensively used and updated to keep pace with the ever-changing environment. Weekly discussions are held with the customer on what to tackle next in the challenge to stay one step ahead of the curve.

5.2.1 Comprehensive Landing Zone automation – from ticket to product

Before the introduction of the Landing Zone, the customer's account creation process was done through "click-ops," a humorous term referring to doing everything by hand, clicking buttons and filling boxes. This alone was a significant undertaking, and the expected timeframe was several business days. After that, more custom-made scripts had to be used, and, finally, a poorly templated Terraform code was manually applied. This did the job, but it was inefficient and prone to human errors. I have personally witnessed and corrected several, and it was expensive. The modern solution of the Landing Zone readies an account and the needed environments in, at most, six simple steps:

- Account creation + automated Lambda configuration.
- Entra ID groups creation (if required).
- Terraform Cloud workspaces creation.
- GitHub repository creation.
- Core Infra deployment.
- AWS SSO Assignment creation.

While this is already a significant achievement, and it reduced the estimated timeframe to an hour or two, our goal is to reduce the number of steps to one: reviewing and approving the ticket request (Figure 20).

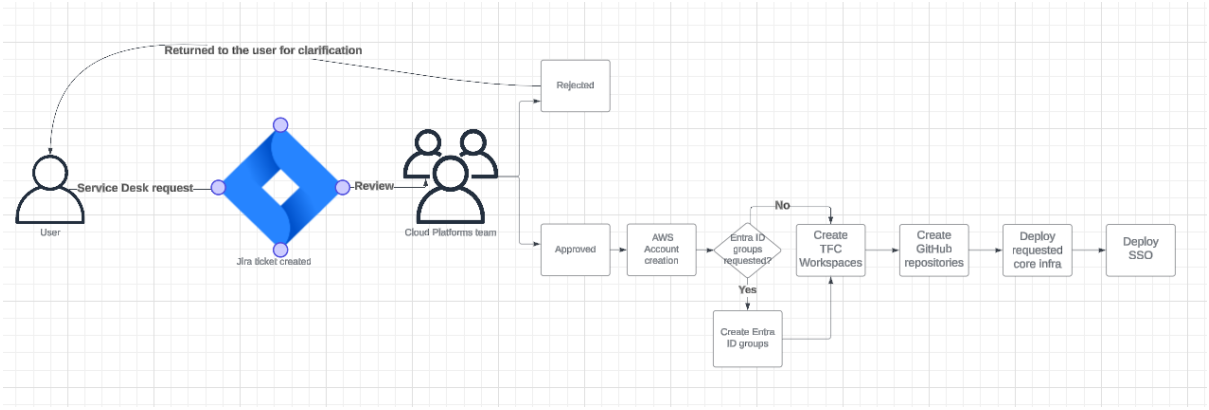

Figure 20. Conceptualised future automated workflow.

5.2.2 Automated Contacts for health notifications

Up until six months ago, Stacksets were used to deploy an AWS Simple Notification Service (SNS) topic connected to AWS EventBridge configured to send out a notification whenever a new AWS health notification was issued. The "governance" account functions now as the delegated administrator for health notifications, meaning they are already centralised and funnelled into one account. A similar AWS EventBridge to SNS topic configuration has been set up there, pushing notifications to an internal tool. This tool is configured with a database of relevant contacts to receive the necessary information to remediate the matter at hand, whenever necessary (Figure 21).

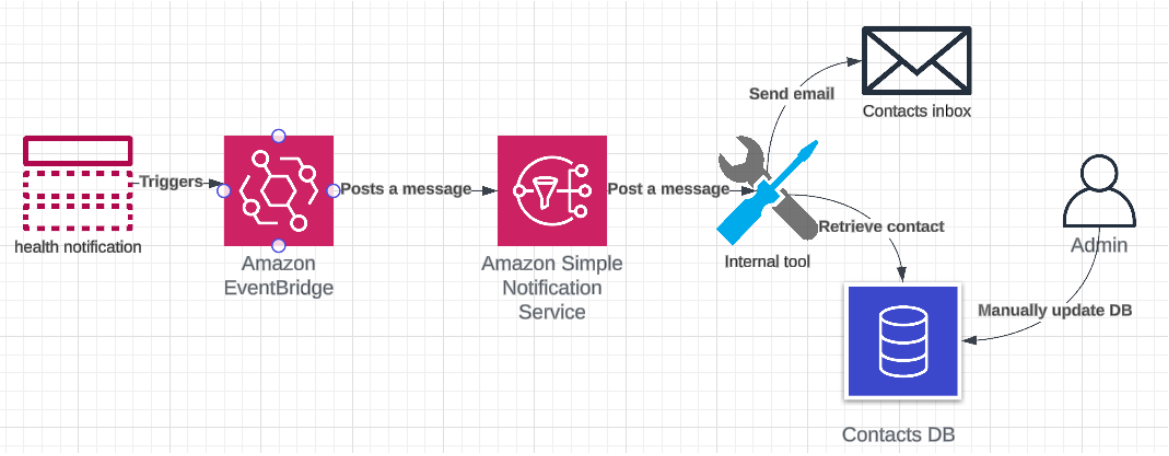

Figure 21. Current health notification delivery process showing manual database maintenance.

Manual work is required to onboard newly created accounts onto the tool and to update the database when the responsible people change. Ideally, this should be removed; there should be one centralised place where account information is stored: AWS Account tags. These are managed in the Landing Zone, so one could implement automation to trigger a Lambda to update the database upon the account tags changing (Figure 22). This could be further automated by having the "Bootstrapper" Lambda (Figure 12) also be able to trigger it.

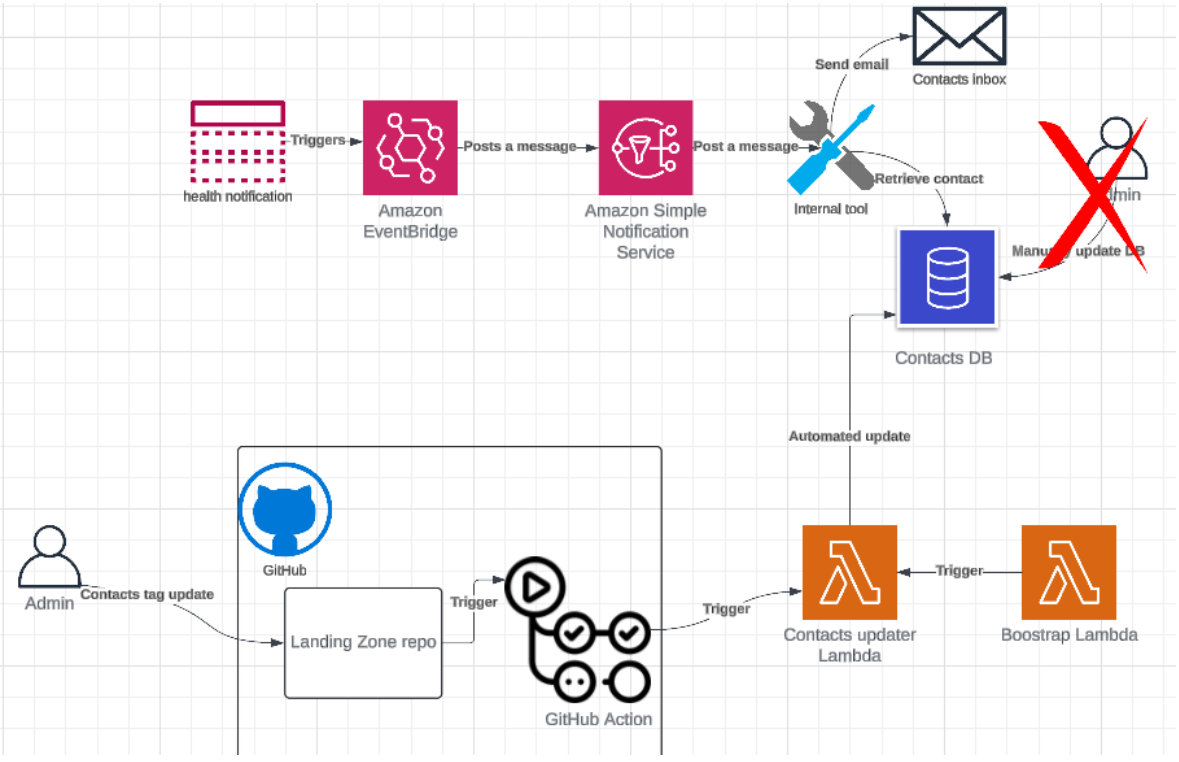

Figure 22. Automated onboarding and contacts update.

REFERENCES

Amazon Web Services. (2023, December 6). Security Pillar. *AWS Well-architected Framework*. Retrieved from https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf

Amazon Web Services. (n.d.). AWS IAM documentation. *AWS Docs*. Retrieved April 27, 2024, from https://docs.aws.amazon.com/iam/

Amazon Web Services. (n.d.). AWS Organizations. *AWS Docs*. Retrieved April 25, 2024, from https://docs.aws.amazon.com/organizations/latest/userguide/orgs_introduction.html

Amazon Web Services. (n.d.). AWS SDKs. Retrieved April 27, 2024, from https://aws.amazon.com/developer/tools/

Amazon Web Services. (n.d.). Consolidated billing for AWS Organizations. Retrieved May 29, 2024, from https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/consolidated-billing.html

Amazon Web Services. (n.d.). Creating a member account in your organization. *AWS Docs*. Retrieved April 27, 2024, from https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_accounts_create.html

Amazon Web Services. (n.d.). OIDC Providers. *AWS Docs*. Retrieved April 24, 2024, from https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_create_oidc.html

Amazon Web Services. (n.d.). Service Control Policies. *AWS Docs*. Retrieved April 23, 2024, from https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scps.html

Amazon Web Services. (n.d.). What is AWS. Retrieved April 25, 2024, from https://aws.amazon.com/what-is-aws/

Amazon Web Services. (n.d.). What is IAM Identity Center. *AWS Docs*. Retrieved April 24, 2024, from https://docs.aws.amazon.com/singlesignon/latest/userguide/what-is.html

Cisco. (n.d.). What is a next-generation firewall. Retrieved April 26, 2024, from https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-next-generation-firewall.html

Friend, N. (n.d.). tree.nathanfriend.io. Retrieved April 27, 2024, from tree.nathanfriend.io

GitHub. (n.d.). GitHub Actions. *GitHub Docs*. Retrieved April 26, 2024, from https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Gitnux. (2024). Gitnux market data report . Retrieved April 21, 2024, from https://gitnux.org/cloud-cost-savings-statistics/

HashiCorp. (n.d.). Dynamic provider credentials in AWS. Retrieved April 25, 2024, from https://developer.hashicorp.com/terraform/cloud-docs/workspaces/dynamic-providercredentials/aws-configuration

HashiCorp. (n.d.). Provider configuration. Retrieved April 25, 2024, from https://developer.hashicorp.com/terraform/language/providers/configuration

HashiCorp. (n.d.). Resource Graph. Retrieved April 27, 2024, from https://developer.hashicorp.com/terraform/internals/graph

HashiCorp. (n.d.). Terraform. Retrieved April 24, 2024, from https://developer.hashicorp.com/terraform/language/syntax/configuration

HashiCorp. (n.d.). Terraform Registry. Retrieved April 27, 2024, from https://registry.terraform.io/providers/hashicorp/aws/latest/docs

HashiCorp. (n.d.). Terraform shared state. Retrieved April 27, 2024, from https://developer.hashicorp.com/terraform/cloud-docs/workspaces/state

Makhlouf, R. (2020, January 13). Cloudy transaction costs: a dive into cloud computing economics. *Journal of Cloud Computing*, 9. Retrieved April 24, 2024, from https://doi.org/10.1186/s13677-019-0149-4

Microsoft Azure. (n.d.). Claims mapping policy type. Retrieved April 26, 2024, from https://learn.microsoft.com/enus/entra/identity-platform/reference-claims-mapping-policy-type

Microsoft Azure. (n.d.). Configure optional claims. Retrieved April 25, 2024, from https://learn.microsoft.com/en-us

National Institute of Standards and Technology. (2024, February 26). The NIST Cybersecurity Framework (CSF) 2.0. Retrieved April 24, 2024, from https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf

Silva, A., & Snehal, N. (2020, September 18). Using AWS Config for security analysis and resource administration. *AWS Blog*.

Artificial intelligence has been used in the work as follows:

ChatGPT 2023. OpenAI. GPT-3.5. Accessed for language check, April 2024. https://chat.openai.com