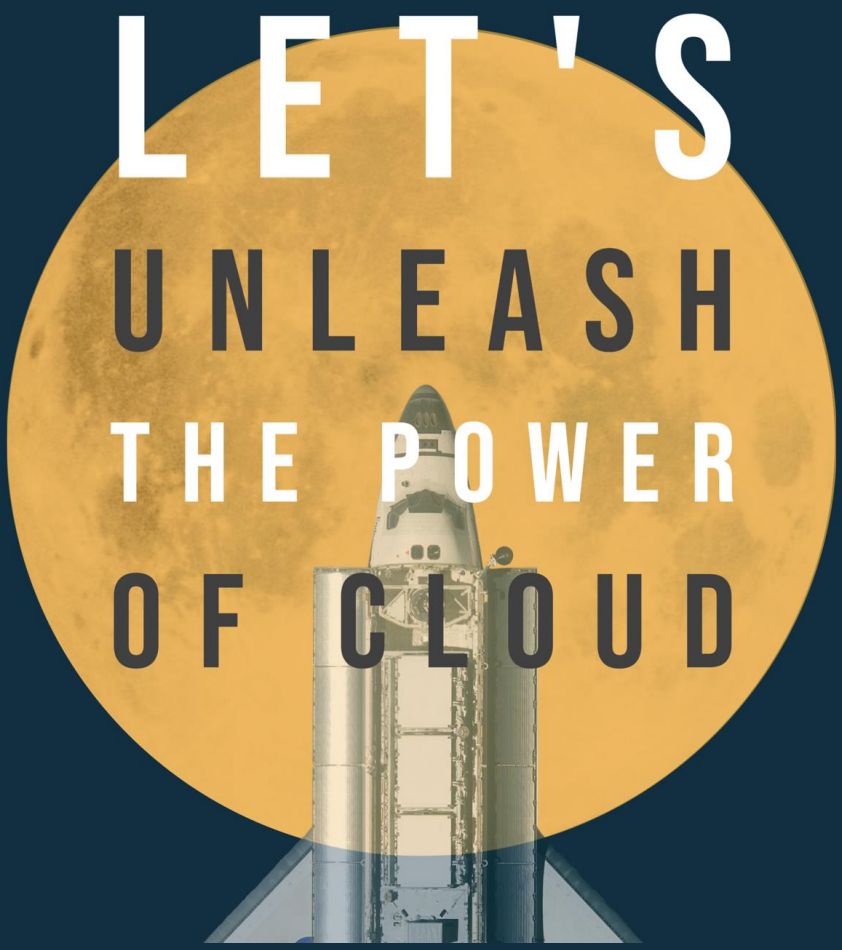
TechNative




# DOING DEVOPS TOGETHER

a Whitepaper about TechNative's approach and principles running DevOps in partnership with our clients

**By:** Bas Anneveld, Mandy van Os, Pim Snel

**Date:** January 16 2024

**Handbook:** TNT0431b

**Version:** 3

# Contents

**1 Introduction** **3**
1.1 Audience . . . 3

**2 TechNative's approach and principles** **3**
2.1 Infrastructure as Code (IaC) . . . 3
2.2 Ownership of source code . . . 3
2.3 Cloud Native and Serverless . . . 4
2.4 AWS Multi-account setup . . . 4
2.5 EC2 OS Provisioning . . . 5
2.6 EC2 OS Patching . . . 5
2.7 Container Image Provisioning . . . 5
2.8 Cloud Orchestration . . . 6
2.9 Container Orchestration . . . 6
2.10 Orchestration to Run and Manage Web Apps . . . 6
2.10.1 AWS Beanstalk . . . 6
2.11 Development and Test Processes . . . 7
2.11.1 Local Development: Pre-commit . . . 7
2.11.2 Local Development: Unit Tests . . . 7
2.11.3 Local Development: Code Coverage . . . 7
2.11.4 Pull Request: First Build tests . . . 7
2.11.5 Build / Staging Phase: Integration / E2E Tests . . . 8
2.11.6 Staging / Production Phase: Performance Tests . . . 8
2.11.7 Production Phase: Smoke Test / Auto Roll-back . . . 8
2.12 Continious Integration & Delivery . . . 8
2.13 Continuous Security Monitoring . . . 9
2.13.1 Prowler . . . 9
2.13.2 Deepfence Secretscanner . . . 9
2.14 Policy as Code . . . 9
2.15 Documentation . . . 10
2.16 Review Merge and Apply IaC . . . 10
2.16.1 The workflow step by step . . . 11
2.16.2 Branches . . . 12
2.16.3 Development Branches . . . 12
2.16.4 Reviewers . . . 12
2.16.5 Merge en Apply . . . 12

2.17 TerraForm Code Organization . . . . . 13
2.17.1 Account Infrastructure project . . . . . 13
2.17.2 Terraform Module Project . . . . . 13
2.18 Building Release Workflows . . . . . 14
2.19 Workload health KPIs . . . . . 14

**3 TechNative Standard Base** **15**
3.1 Well Architected Review . . . . . 15
3.2 TechNative Landing Zone . . . . . 16
3.2.1 Encryption at rest KMS . . . . . 17
3.2.2 AWS Multi-account structure . . . . . 17
3.2.3 Extended CloudTrail Setup . . . . . 18
3.2.4 Cross account access through assume role . . . . . 18
3.2.5 Pre-configured AWS resources maintained by TechNative through Infrastructure as Code (IaC). . . . . 18
3.2.6 TechNative's Auto Inventory System . . . . . 19
3.3 TechNative's Advanced Cloud Observability System . . . . . 19

**4 References** **19**

**5 Changelog** **19**
5.1 Version 3 - 2024-01-16 . . . . . 19
5.2 Version 2 - 2023-10-23 . . . . . 19
5.3 Version 1 - 2023-07-23 . . . . . 20

# 1 Introduction

**We are TechNative, and we are here to unleash the power of your cloud.**

At TechNative we enable and guide you to build, optimize, and manage your tech stack with the best tools and services. You will get access to real expertise and FinOps methodologies needed to make your applications more scalable and reliable. By working with us, you will effectively improve your cloud, platform, performance and operational excellence.

We're a team of consultants, hands-on engineers & FinOps experts bridging the gap between business, finance and technology. Because we're 'techies' by heart, we know what 'good' looks like. And while knowledge is more important than certifications, just for easy of mind, all our engineers are certified.

## 1.1 Audience

We've written this white paper for our clients to inspire, to share our DevOps approach, and to share best practices that we've learned in the many years of experience we have with cloud engineering.

# 2 TechNative's approach and principles

## 2.1 Infrastructure as Code (IaC)

TechNative implements hosting infrastructure for clients by documenting all hosting resources in source code, Infrastructure as Code or IaC. We use Terraform and Python for this purpose. This enables version control and advanced CI/CD of the hosting infrastructure.

## 2.2 Ownership of source code

The applied source code of the delivered infrastructure is the property of the client. All modules used are the property of the respective authors. We are proud of the Terraform and Python modules we develop and publish them on GitHub under the Apache Licence.


## 2.3 Cloud Native and Serverless

If the circumstances allow it, TechNative will always recommend using Cloud Native components instead of installing software on an EC2 or container. These benefits almost always apply for cloud native components;

- They are cheaper.
- Costs are more transparent.
- Implementation is easier.
- Maintenance is easier.
- Automatically scalable.

Technative advocates using as many serverless components as possible. You pay for what you consume, serverless systems are easy to use and are scalable.

## 2.4 AWS Multi-account setup

On lift and shift basis, TechNative recommends clients to use a multi-account setup strategy as cloud architecture.

The benefits of a multi cloud strategy are significant;

- Group workloads based on business purpose and ownership.
- Apply distinct security controls by environment.
- Constrain access to sensitive data.
- Limit scope of impact from adverse events.
- Support multiple IT operating models.
- Manage costs.
- Distribute AWS Service Quotas and API request rate limits.


## 2.5 EC2 OS Provisioning

TechNative uses EC2 provisioning solutions who fit the customer's needs. Mostly this is Ansible, Salt or Puppet. We also use Nix and NixOS for provisioning EC2 Linux servers quite a lot.

## 2.6 EC2 OS Patching

When we take control over EC2 and the operating system is supported, we will install SSM agent as standard for automatic scanning and patching of security updates. SSM is a part of AWS System Manager. AWS System Manager reports are integrated into TechNative's Advanced Cloud Observability System, which is a part of TechNative's Standard Base.

## 2.7 Container Image Provisioning

TechNative has adopted a multi-platform approach, incorporating Docker, Podman, and NixOS for defining container images. These tools allow TechNative to create and manage lightweight, portable, and isolated environments for running applications.

## 2.8 Cloud Orchestration

When possible, TechNative will choose Terraform + CI/CD for cloud orchestration. We developed a complete set of boilerplates and modules to achieve efficient and effective deployment workflows for infrastructure and applications.

If the customer or the situation prefers another orchestration system, our team fully adapts to that orchestration system. We have experience with Ansible, Salt, and Puppet.

## 2.9 Container Orchestration

TechNative does not have a preference for container orchestration. We support everything our customers run.

We work a lot with Kubernetes, ECS, EKS, and Mesos.

## 2.10 Orchestration to Run and Manage Web Apps

### 2.10.1 AWS Beanstalk

Sometimes clients would like to use AWS Beanstalk for hosting their PHP application stack. We have experienced that AWS Beanstalk is an excellent service for deploying services and TechNative can implement it as IaC. However, we recommend directly implementing the application stack in containers using AWS Elastic Container Service.

AWS Beanstalk is an entry-level service and considered a black box. The operating systems are from Amazon and unfortunately are not very up to date. Compared to a custom container implementation, Beanstalk is relatively expensive.

Finally, the advantages of containers on ECS are;

- Can integrate into any possible pipeline configuration.
- Deployment time can be improved through optimization.
- Any possible software version can be used.
- There are multiple auto-scaling strategies available.

## 2.11 Development and Test Processes

TechNative helps set up or streamline development and testing processes. There are several factors that determine how these processes are implemented. As a result, each development and testing process is unique. However, TechNative works with standard templates and components to design or optimize a process for your organization that meets industry standards. This allows you to achieve an agile and efficient development process in which your products are automatically and continuously validated against the highest quality standards.

In different steps in the pipeline, we incorporate standard or custom checks that prevent developers from deploying poorly written, unsafe, or malfunctioning software.

### 2.11.1 Local Development: Pre-commit

By using `pre-commit`, we help developers detect various types of errors before they commit their changes in Git. Standard checks we implement:

- Linting of configuratie files
- Code Smell detectors (wrong code style, duplicate code, wrong programming patterns)
- Offline Secrets Detection
- Commit Message Style (is all information needed and correctly tagged)

### 2.11.2 Local Development: Unit Tests

If necessary, we help set up Unit Tests, including setting up standard scripts or Makefiles. We create examples tailored to the requirements.

### 2.11.3 Local Development: Code Coverage

A seperate procedure needs to garantee that (unit) tests have covered the complete, or a minimun percentage of the source code. TechNative helps implementing a code coverage framework that fits in the chosen development framework.

Code Coverage is tied together in the scripts that fire the Unit Tests.

### 2.11.4 Pull Request: First Build tests

In most cases a Pull Request a.k.a Merge Request triggers the pipeline to start a runner which builds the product and runs several test steps. The pre-commit rules and unit tests will be executed again, as well as the Code Coverage.

The pipeline jobs posts a summary of the build and test results in the Pull Request Conversation.

#### 2.11.5 Build / Staging Phase: Integration / E2E Tests

Depending on the set up integration tests are implemented to ensure compoments work together as they should in a real life situation.

TechNative has a lot of experience and is a strong advocate for using End to End tests as integration tests. After a representative version of the software stack is deployed, automated tests are executed as if they were performed by an end user. A robot script will perform a series of steps that a real user would also perform. The entire software stack is thus put to work and checked for correct functioning.

E2E tests can also be combined with Code Coverage.

#### 2.11.6 Staging / Production Phase: Performance Tests

Performance tests can be implemented by default to ensure the system meets the performance requirements. TechNative implements Performance tests using external propes or internal E2E tests.

#### 2.11.7 Production Phase: Smoke Test / Auto Roll-back

When deploying changes to production, the end user is confronted with a new version of the system. This is the domain where business continuity is of the highest significance, and this continuity must not be compromised in any way.

To ensure that a production deployment is successful, smoke tests are implemented. These are quick tests that demonstrate that everything is functioning as it should. In the event of a failure in any of these tests, an automatic roll back to the previous version or a standby system is executed.

### 2.12 Continious Integration & Delivery

TechNative does not have a preference for CI/CD systems. We work a lot with CodePipeline, GitLab CI, GitHub WorkFlows, Jenkins, and Bitbucket Pipelines.

If needed, we can assist with writing tests in various languages and frameworks.

## 2.13 Continuous Security Monitoring

### 2.13.1 Prowler

TechNative uses Prowler to automate scanning of our customers AWS accounts for configuration errors and security issues. Our cloud engineers periodically analyse the reports to determine why recommendations are made.

The standard focus during analysis of Prowler reports includes;

- The least privileges in security groups.
- The least privileges in iam roles and policies.
- Encryption of data in rest.
- MFA authentication of root and user accounts.


### 2.13.2 Deepfence Secretscanner

TechNative uses SecretScanner by deepfence by default in pipelines for scanning for secrets in code repositories.

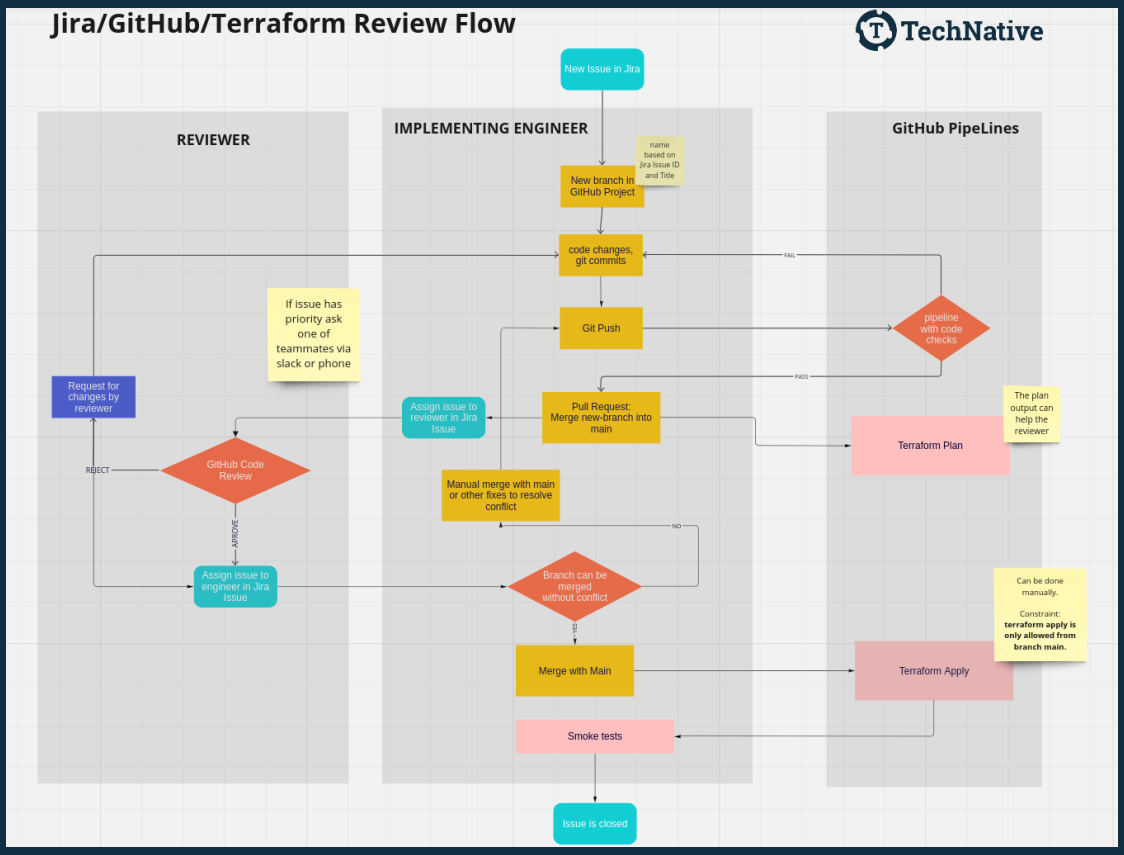
## 2.14 Policy as Code

TechNative uses the Regula open policy engine to check policies for misconfigurations and security errors. We integrate Regula by default into the CI/CD pipeline. We use the standard set of Regula rules supported with a set of rules that we have developed ourselves. It sometimes happens that we add rules or modify existing rules together with the customer.

## 2.15 Documentation

Continuity of customer services is our highest priority, which is why every cloud engineer should be replaceable (in theory). We achieve this by providing our new employees with a comprehensive training and ensuring that every system managed by TechNative is fully documented according to our standards. Documentation ensures that everyone in our Managed Cloud Service team knows what to do in case of any issues.

## 2.16 Review Merge and Apply IaC




To make sure our codebase and the state of infrastructure we manage, keeps in sync, we created a few guidelines. These guidelines inform our teams on how to merge, review and apply the IaC source code. If we follow these guidelines, our codebase and state stays clean, making it easier to implement changes, it makes us develop more efficiently, so we can deliver faster with better quality.

Follow these guidelines if you need to merge a pull request together with `terraform apply` to make your changes effective in the infrastructure.

### 2.16.1 The workflow step by step

The Jira/GitHub/TerraForm Workflow diagram describes all steps in different areas. These areas represent the user roles: `reviewer`, `implementing engineer` and GitHub Pipelines.

This is a textual description of the complete process from change request to deployment.

1. A change is requested, a ticket in Jira is created.
2. At refinement, the definition of done is defined.
3. The `implementing engineer` is assigned to implement the changes.
4. The `implementing engineer` sets Jira ticket state to `progress`.
5. The `implementing engineer` creates a feature branch from the code base of subject.
6. The `implementing engineer` implements the code changes and commits and pushes to GitHub.
7. The `implementing engineer` creates a Pull Request of his feature branch against `main`.

#### Request review

1. The `implementing engineer` assigns his Jira-ticket to a `reviewer`, sets ticket state to `review`.
2. The `implementing engineer` assigns requests a review in the PR.
3. The `reviewer` reviews the code changes.

#### At reject

1. The `reviewer` describes request for changes in PR-review.
2. The `implementing engineer` makes changes and starts again with **request review.**

#### At approval

1. The `reviewer` set `approve` in the GitHub PR.
2. The `reviewer` assigns Jira ticket to the `implementing engineer`, set ticket status to ‘merge’.
3. The `implementing engineer` merges PR with `main`.
4. The `implementing engineer` deletes his PR-source-branch.
5. The `implementing engineer` checkouts main, pulls, and runs `terraform apply`.

### 2.16.2 Branches

We only have two kinds of branches: The `Main` branch and temporary development branches.

**Main** The stable branch is the `main` branch, and this branch should normally represent the infrastructure state. When a PR is merged, the state is behind the code. At this point, it is important that a `terraform apply` follows ASAP. The `implementing engineer` is responsible for this `terraform apply` action.

To guarantee that main and state are in sync. `terraform apply` should always be executes from within the `main` branch.

Engineers develop code in feature branches. It should be prevented to commit and push directly into the main branch.

### 2.16.3 Development Branches

A development branch is a temporary branch which contain changes to be implemented to fulfil the requirements of a Jira ticket. When a Pull Request is merged, the development branch should be deleted. Feature branches and Hotfix branches are both development branches. The branchname should include the Jira Ticker number and a short description of the changes. E.g. `TLDA-999-Joe-access-to-playground-999`

### 2.16.4 Reviewers

The managed service team has a pool of engineers who are experienced enough to review a PR. When changes have high priority and need to be merged and applied as soon as possible, reviewers should be able to react quickly. The team should be flexible in when situations like this occur.

### 2.16.5 Merge en Apply

When a reviewer has approved the PR, the implementing engineer can merge it into main. After merging, it should be applied to terraform as soon as possible. Currently, we handle this manually. But in the future, we might automate this action with a git-pipeline. The implementing engineer should always live check if `terraform apply` action has finished successfully. When problems occur, the engineer is responsible for fixing these problems ASAP.

## 2.17 TerraForm Code Organization

This section describes the code organization of Account Infrastructure Projects and Terraform Module Projects. Both types of projects are written in mostly in Terraform HCL.

### 2.17.1 Account Infrastructure project

An account infrastructure project describes all resources in an AWS account. An account belongs to TechNative or to one of TechNative customers.

The project repository is named with the convention `[company]-awsaccount-[aws-accound-id]-[account-description]`. All in lowercase characters.

Examples;

```
technative-awsaccount-082378161014-gitbackup
codeseed-awsaccount-166375656655-management
```

**Code structure** The root of an account-repository contains all relevant Terraform code. The terraform code should be as compact as possible. Simple resources can be described. Complex stacks of resources should be defined in modules outside the account project.

Example;

```
/README.md
/backend.tf
/main.tf
/outputs.tf
/provider.tf
```

### 2.17.2 Terraform Module Project

A Terraform module project is a git-repository with terraform code configuring resources that belong together and can be used to implement a specific topic.

A TerraForm module can be imported from another project or from an account project. Using modules for specialized tasks keeps the code understandable, maintainable and DRY.

Most of the time, we use or create TerraForm AWS modules. AWS here refers to the terraform-provider. There are more types of modules though. The terraform-null-url-parser module for example helps to parse URLs. The `null` in its name refers to the fact that this module does not need any terraform-provider.

**Creating a new Terraform AWS module** At TechNative our modules are in most cases, Terraform AWS modules. That is why we have created a template repsitory to make life more easy.

To start a new Terraform AWS Module, open the terraform-aws-module-template and follow the instructions at the top of the `README.md`

## 2.18 Building Release Workflows

As a DevOps partner, TechNative assists in setting up release workflows. We follow a standard procedure consisting the following steps.

- Identifying stakeholders.
- Identifying responsibilities.
- Identifying quality assurance procedure.
- Identifying other steps.
- Automating quality assurance control steps.
- Integrating CI/CD.
- Documenting in the form of run books.

## 2.19 Workload health KPIs

TechNative works with CloudWatch or Grafana for the implementation of Health Dashboards. The TechNative observability module, part of the TechNative Standard Base, can be connected to CloudWatch to forward generated alerts to any standard alerting tool such as PagerDuty or OpsGenie.

For Cloud Native services like AWS RDS, we follow the limits recommended by AWS in combination with the specs of the specific service to configure alert rules.

Custom services, such as apps, containers, and EC2 instances, are calibrated according to the specific needs.

Cost limits are also configured in dashboards and alert rules in consultation with the customer.

For the customer's platforms or applications, a summary is also configured as a "core vitals" dashboard.

# 3 TechNative Standard Base


TechNative Standard Base is a standard set of tools and modules that our customers can implement and use without any licence costs. Through the TechNative Standard Base, we are able to efficiently and effectively support our customers. The TechNative Standard Base consists of the following components.

## 3.1 Well Architected Review

When boarding new clients, TechNative starts with a Well Architected Review.

The AWS Well Architected Review is a high overview set of foundational questions based on AWS architectural standards for designing and operating secure, reliable, efficient, cost-effective and sustainable workloads in the AWS Cloud.

The WAR is based on the 6 pillars of the Well Architected Framework;

- Operational excellence;
- Security;
- Reliability;
- Performance Efficiency;
- Cost Optimization;
- and Environmental Impact.

TechNative is trained to use the WAR tool provided by AWS, which allows us to break down

these pillars into questions in which TechNative can conduct deep-dive interviews, collect data and analyse the current infrastructure environment in order to provide guidance for future enhancements to the environment.

The purpose of a Well Architected Review is to assess the client's, existing environments and to identify and categorize risks based on the outcome of AWS Well architected tool.

Results are consolidated in the WAR Report. TechNative delivers a risk assessment of these findings in addition to the outcome of the AWS report.

## 3.2 TechNative Landing Zone


The TechNative Landing Zone (TLZ) for AWS is an AWS template environment which enables our clients to quickly deploy new accounts in their AWS organization with advanced features enabled out of the box. All our clients can use TLZ for AWS without additional costs.

TLZ for AWS includes...

- multi-account environments;
- encryption at rest using KMS;
- extended CloudTrail setup enabled by default;
- default roles and policies;
- AWS Control Tower compatible;
- SSO compatible;
- TechNative Auto Inventory System.

Let's dive a bit deeper on some of these features to get a better picture of the TechNative landing zone.

### 3.2.1 Encryption at rest KMS

The standard functionality of the TLZ and our Terraform modules is the use of KMS keys to encrypt data at rest.

Advantages;

- **Enhanced Security;** By using KMS keys, data is protected through encryption, providing an additional layer of security for sensitive information.
- **Compliance with Data Regulations;** Encrypting data at rest is often a requirement for compliance with data protection regulations, such as the General Data Protection Regulation (GDPR). Using KMS keys ensures adherence to these regulations.
- **Ease of Implementation;** The TLZ and Terraform modules provide a streamlined and straightforward way to implement data encryption using KMS keys. This simplifies the process for developers and reduces the potential for errors.
- **Scalability and Flexibility;** As the TLZ and Terraform modules are designed to be easily integrated into existing systems, they allow scalability and flexibility. This means that businesses can adapt their encryption methods to suit their evolving needs without significant disruption.
- **Centralized Key Management;** KMS keys facilitate centralized key management, allowing administrators to control and manage encryption keys from a single location. This helps to streamline administrative tasks and ensure consistent encryption practices across the organization.
- **Seamless integration with AWS Services;** Since KMS is an AWS service, utilizing KMS keys seamlessly integrates with other AWS services. This enables seamless encryption and decryption across various AWS resources, providing a cohesive and secure data ecosystem.

### 3.2.2 AWS Multi-account structure

A multi-account structure offers benefits such as;

- **Grouping workloads;** Separating workloads based on business purpose or ownership.
- **Security controls per environment;** - determine who can do what per environment through role based access control policies.
- **Blast radius;** Containment of workloads/environments on an infrastructure level, having a smaller blast radius when something goes unplanned.
- **Cost isolation;** Multiple accounts help separate items at a billing level across business units, functional teams, or individual users.

- **Data isolation;** Separating data stores in different accounts limits the number of people that can access and manage that data. This contains exposure to highly private data and helps with GDPR compliance.

#### 3.2.3 Extended CloudTrail Setup

AWS CloudTrail monitors and records all your account activity across your AWS infrastructure. This gives you control over storage, analysis, and remediation actions. TechNative has adopted AWS CloudTrail as a default in accounts deployed by TLZ for AWS. TechNative has created Terraform modules which implements an extended CloudTrail setup, including log-trails for account policies, organization policies and S3 policies.

Using TechNative's extended CloudTrail Setup helps us and our clients with debug issues to resolve these quicker and better.

#### 3.2.4 Cross account access through assume role

To prevent AWS administrators to manage multiple user accounts for multiple AWS environments. We leverage on the use of IAM roles to assume a role for cross account access, AWS Cloud trail for tracking users and API usage and role based access control to manage authorization. By using the power of IAM roles, we can;

- Manage team member's access to multiple accounts.
- Cross-account access to be managed under a central IAM account.
- Less prone to security risks as IAM management becomes centralized.

AWS also offers various information on cross account access and the benefits of using roles and centralizing user management. AWS, Cross Account Access

#### 3.2.5 Pre-configured AWS resources maintained by TechNative through Infrastructure as Code (IaC).

During our projects, we noticed that many organizations tend to use a common set of AWS resources on their AWS platform. To prevent re-inventing the wheel with every project, we decided to leverage the power of Terraform as our Infrastructure as Code tool to document and create a clear and lightweight abstraction to configure your AWS resources. Terraform allows infrastructure to be expressed as code in a simple, human-readable language called HCL (HashiCorp Configuration Language). It reads configuration files and provides an execution plan of changes, which can be reviewed for safety and then applied and provisioned.

#### 3.2.6 TechNative's Auto Inventory System

Part of TLZ is TechNative's Auto Inventory System. This system inventories all resources in an AWS Account and reports changes through the TechNative's Advanced Observability System. Newly deployed AWS Accounts in an organization are automatically included in the inventory scan.

### 3.3 TechNative's Advanced Cloud Observability System


Part of TechNative's Standard Base is our advanced modular Cloud Observability System. The system consists of a basic implementation combined with some default monitoring and alarm definitions. As a client, your team can add your own monitoring definitions based on your workload.

The system leverages on AWS CloudWatch, as this gives us complete visibility. In the integrated Messaging Hub, connections can be made with various alarm management systems or messaging apps, such as OpsGenie, PagerDuty or Slack.

## 4 References

References like PRIAC-001 refer to the AWS DevOps Self Assessment requirements.

## 5 Changelog

### 5.1 Version 3 - 2024-01-16

- Website version

### 5.2 Version 2 - 2023-10-23

- Added section about Deepfence SecretScanner

### 5.3 Version 1 - 2023-07-23

- initial publication