International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

The Evolution of CDN Management: A Paradigm Shift to Infrastructure as Code for Akamai Property Transformation

Avinash Ibbandi

Senior Manager, Software Engineering, Walmart Inc, Sunnyvale, California, USA

Abstract: ***Infrastructure as Code (IaC) has transformed how organizations manage their infrastructure by automating configurations and deployments. Akamai Property Manager, a key component of Akamai's content delivery network (CDN), can be managed using IaC to streamline property configuration and deployment processes. This journal entry explores the use of Terraform, a popular IaC tool, for automating Akamai Property Manager. The article covers the prerequisites, steps for configuring Terraform, and best practices for version control and CI/CD integration, providing a comprehensive guide for automating Akamai properties.***

Keywords: Property Manager, Akamai Rule Engine, Akamai API Integration, Akamai Terraform Automation, Web Acceleration, Content Caching, Edge Computing, Infrastructure as Code.

1. Introduction

Akamai Property Manager allows users to create and manage edge configurations, rules, and behaviors for content delivery. Managing these properties manually through Akamai's user interface can be cumbersome, especially when managing multiple environments or configurations. Infrastructure as Code (IaC) offers a solution by treating Akamai property configurations as code, automating the setup, and promoting collaboration through version control.

This journal entry explores how to automate Akamai Property Manager using Terraform, a widely adopted IaC tool. It outlines the steps required to automate property configurations, integrate with Akamai's APIs, and implement best practices for version control and CI/CD pipelines.

2. Literature Review

Automating Akamai Property Manager with Terraform situates this practice within infrastructure-as-code (IaC) methodologies and edge service management, where tools like Terraform are essential for codifying, automating, and versioning infrastructure. Studies highlight IaC benefits in improving repeatability, reducing errors, and accelerating deployments. However, scaling IaC introduces challenges, particularly in state management, as large, dynamic environments can strain Terraform state handling and performance issues that are especially relevant in managing Akamai's distributed edge configurations. Although Akamai Property Manager is key for optimizing edge content delivery and security, its Terraform provider does not yet support all native features, limiting comprehensive automation and often requiring a hybrid of manual and automated processes. This constraint introduces potential inconsistencies, and as edge services become increasingly central across industries, robust automation solutions are in high demand. Addressing these limitations calls for improved tooling, provider support, and best practices like code modularization, state management optimization, and fallback strategies for complex edge environments.

3. Methodology & Technical Approach

1) Prerequisites

Before diving into automation, ensure the following prerequisites are met.

- **Install Terraform**: Make sure you have Terraform installed on your local machine. You can download it from the Terraform official site.
- **Akamai Credentials**: You'll need Akamai credentials (client token, client secret, and access token) for API access. These can be generated from your Akamai Control Center under the "Identity and Access Management" section.
- **Akamai Provider for Terraform**: The Akamai provider in Terraform allows you to interact with Akamai's API to manage properties.
- **Akamai Edgegrid Configuration**: Ensure that the ~/.edgerc file is properly set up with your Akamai credentials.

2) Automating Akamai Properties Using Terraform

- Terraform, developed by HashiCorp, is a declarative tool that enables infrastructure provisioning through code. Akamai has a Terraform provider that facilitates property management. Below are the steps to automate Akamai Property Manager using Terraform.
- Install Terraform on your system by downloading it from Terraform's official website. Once installed, set up your Terraform configuration file (main.tf) to interact with the Akamai provider. A typical provider setup looks like this:

Terraform Provider Setup

```
provider "akamai" {
  edgerc = "~/.edgerc"
  section = "default"
}
```

Here, edgerc points to the authentication credentials required for API access, which must be set up before running Terraform commands.

3) Property Configuration

- Using Terraform, you can define your Akamai property in a main.tf file. Below is a sample configuration for a simple Akamai property:
- **Setting Up Terraform for Akamai Terraform Configuration**: Create a directory for your Terraform project, and inside that directory, create a new Terraform configuration file (main.tf).

```
mkdir akamai-property-manager
cd akamai-property-manager
touch main.tf
```

- **Create Property Configuration via Terraform Code**: In main.tf, you can start defining a **basic property** in Akamai Property Manager. This includes rules for cache control, redirects, and any other edge optimizations. Below is the terraform code configure property manager on Akamai.
- **Ensure that the syntax and indentation are correct before executing the code.**

```
data "akamai_property_rules_builder" "property_rule_default" {
rules_v2024_02_12 {
name      = "default"
is_secure = false
# comments = "Rules are evaluated from top to bottom and the last matching rule wins."
behavior {
origin {
              cache_key_hostname             = "REQUEST_HOST_HEADER"
              compress                       = true
              custom_valid_cn_values         = ["{{Origin Hostname}}", "{{Forward Host Header}}", ]
              enable_true_client_ip          = true
              forward_host_header            = "ORIGIN_HOSTNAME"
              hostname                       = var.origin_hostname
              http_port                      = 80
              https_port                     = 443
              ip_version                     = "IPV4"
              origin_certificate             = ""
              origin_certs_to_honor          = "STANDARD_CERTIFICATE_AUTHORITIES"
              origin_sni                     = true
              origin_type                    = "CUSTOMER"
              ports                          = ""
              standard_certificate_authorities = ["akamai-permissive", ]
              true_client_ip_client_setting    = false
              true_client_ip_header          = "True-Client-IP"
              verification_mode              = "CUSTOM"
            }
          }
          behavior {
            cp_code {
              value {
                description = var.cpcode_name
                id          = akamai_cp_code.cp_code.id
                name        = var.cpcode_name
                products    = [var.product_id, ]
              }
            }
          }
          behavior {
            caching {
              behavior       = "NO_STORE"
            }
          }
          children = [
            data.akamai_property_rules_builder.property_rule_origins.json,
            data.akamai_property_rules_builder.property_rule_redirects.json,
            data.akamai_property_rules_builder.property_rule_caching_rules.json,
            data.akamai_property_rules_builder.property_rule_denied_by_ip.json,
          ]
        }
      }
```

```
data "akamai_property_rules_builder" "property_rule_origins" {
 rules_v2024_02_12 {
  name                = "Origins"
  comments            = "Control the routings to different origin servers."
  criteria_must_satisfy = "all"
  children = [
   data.akamai_property_rules_builder.property_rule_routing_based_on_path.json,
  ]
 }
}
data "akamai_property_rules_builder" "property_rule_redirects" {
 rules_v2024_02_12 {
  name                = "Redirects"
  comments            = "Control the redirects."
  criteria_must_satisfy = "all"
  children = [
   data.akamai_property_rules_builder.property_rule_redirect_based_on_path.json,
  ]
 }
}
data "akamai_property_rules_builder" "property_rule_caching_rules" {
 rules_v2024_02_12 {
  name                = "Caching Rules"
  comments            = "Control the settings related to caching content at the edge and in the browser."
  criteria_must_satisfy = "all"
  children = [
   data.akamai_property_rules_builder.property_rule_css_and_java_script.json,
  ]
 }
}
data "akamai_property_rules_builder" "property_rule_denied_by_ip" {
 rules_v2024_02_12 {
  name                = "Denied By IP"
  comments            = "Control the IP whitelist."
  criteria_must_satisfy = "all"
  criterion {
   client_ip {
    match_operator      = "IS_NOT_ONE_OF"
    use_headers         = false
    values              = ["127.0.0.1", ]
   }
  }
  behavior {
   deny_access {
    reason  = "deny-by-ip"
    enabled = true
   }
  }
 }
}
data "akamai_property_rules_builder" "property_rule_routing_based_on_path" {
 rules_v2024_02_12 {
  name                = "Routing Based on Path"
  comments            = "Route request to a origin based on path match."
  criteria_must_satisfy = "all"
  criterion {
   path {
    match_case_sensitive = false
    match_operator      = "MATCHES_ONE_OF"
    normalize           = false
    values              = ["/sourcepath", ]
   }
  }
```

```
    behavior {
     origin {
      cache_key_hostname             = "REQUEST_HOST_HEADER"
      compress                       = true
      custom_valid_cn_values         = ["{{Origin Hostname}}", "{{Forward Host Header}}", ]
      enable_true_client_ip          = true
      forward_host_header            = "ORIGIN_HOSTNAME"
      hostname                       = var.origin_hostname
      http_port                      = 80
      https_port                     = 443
      ip_version                     = "IPV4"
      origin_certificate             = ""
      origin_certs_to_honor          = "STANDARD_CERTIFICATE_AUTHORITIES"
      origin_sni                     = true
      origin_type                    = "CUSTOMER"
      ports                          = ""
      standard_certificate_authorities = ["akamai-permissive", ]
      true_client_ip_client_setting  = false
      true_client_ip_header          = "True-Client-IP"
      verification_mode              = "CUSTOM"
     }
    }
   }
}
data "akamai_property_rules_builder" "property_rule_redirect_based_on_path" {
 rules_v2024_02_12 {
   name              = "Redirect Based on Path"
   comments          = "301/302 Redirect based on path match."
   criteria_must_satisfy = "all"
   criterion {
    path {
     match_case_sensitive = false
     match_operator       = "MATCHES_ONE_OF"
     normalize            = false
     values               = ["/sourcepath", ]
    }
   }
   behavior {
    redirect {
     destination_hostname   = "SAME_AS_REQUEST"
     destination_path       = "OTHER"
     destination_protocol   = "HTTPS"
     mobile_default_choice  = "DEFAULT"
     query_string           = "APPEND"
     response_code          = 301
     destination_path_other = "/destinationpath"
    }
   }
  }
}
data "akamai_property_rules_builder" "property_rule_css_and_java_script" {
 rules_v2024_02_12 {
   name              = "CSS and JavaScript"
   comments          = "Override the default caching behavior for CSS and JavaScript."
   criteria_must_satisfy = "any"
   criterion {
    file_extension {
     match_case_sensitive = false
     match_operator       = "IS_ONE_OF"
     values               = ["css", "js", ]
    }
   }
   behavior {
```

```
    caching {
      behavior       = "MAX_AGE"
      must_revalidate = false
      ttl            = "2h"
    }
  }
 }
}
```

4) Applying the Configuration

After configuring the property, you can apply the Terraform configuration using the following commands:

- Initialize Terraform: **terraform init**
  This command initializes the working directory, setting up the required Terraform provider.
- Plan the Changes: **terraform plan**
  This step allows you to preview the changes Terraform will make when applying the configuration, ensuring that everything is correct.
- Apply the Changes: **terraform apply**
  Terraform will now create or modify the Akamai property as defined in the configuration.

5) Integrating Akamai CLI for Automation

In addition to Terraform, Akamai provides a CLI that can be used for additional automation tasks such as activating properties, validating configurations, and managing edge behaviors. The Akamai CLI can complement IaC by handling post-deployment tasks.

Install Akamai CLI using: **brew install akamai**

To activate a property, for example, you can use the following command:

- akamai property activate --network staging --property-name example-property
- This command activates the property on the staging network, ready for testing before pushing it to production.

6) Version Control and CI/CD

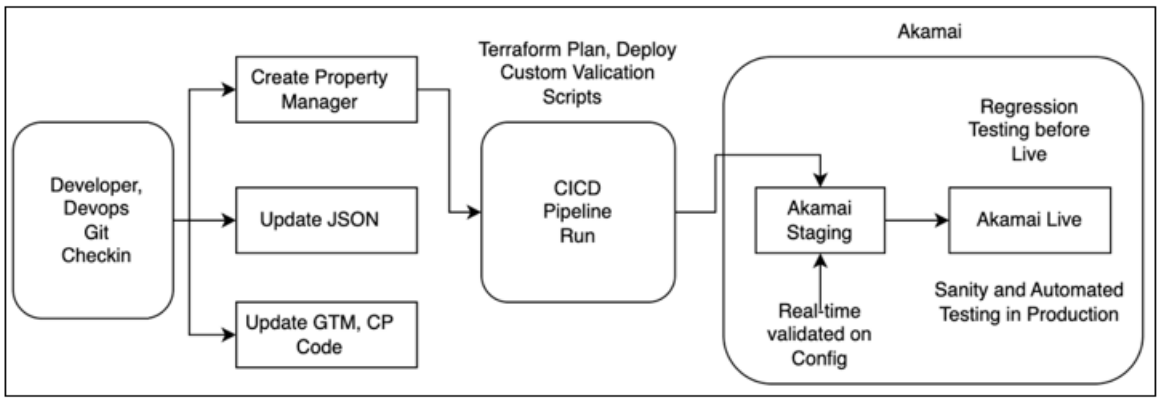
- Managing Akamai properties as code enables collaboration, version control, and repeatability. Storing your Terraform configuration files in a version control system like Git ensures that every change is tracked, allowing for seamless collaboration among teams.
- Furthermore, integrating Terraform with CI/CD pipelines, such as Jenkins, GitLab CI, or GitHub Actions, allows for automated deployments and continuous delivery of Akamai configurations. This automation enhances deployment reliability, reduces manual errors, and ensures consistency across environments.

CI/CD Example Pipeline

Here's an example of how a CI/CD pipeline might look in GitLab CI to automate Akamai Property Manager:

```
stages:
  - deploy
deploy_to_akamai:
  stage: deploy
  script:
    - terraform init
    - terraform apply -auto-approve
  only:
    - main
```

This configuration ensures that changes to the main branch automatically trigger the Terraform deployment, applying updates to Akamai properties.



When automating Akamai configurations, especially using tools like Terraform or APIs, following best practices ensures smooth deployments, minimizes errors, and maximizes efficiency. Below are the key best practices for automating Akamai configurations:

4. Best Practices

1) Use Infrastructure as Code (IaC) Tools

- **Terraform Automation:** Automate Akamai configurations like Property Manager and EdgeWorkers with Terraform for repeatability, version control, and consistency.
- **Version Control:** Store Terraform code in Git for tracking, collaboration, and rollback.

2) **Modularize Code**
   - **Reusable Modules:** Divide Terraform code into modules (e.g., caching, security policies) to simplify updates and scalability.
   - **Use Variables:** Externalize environment-specific values with Terraform variables for flexible configuration management.
3) **Ensure Idempotency**
   - **Automate Consistently:** Avoid manual overrides, ensuring automation produces consistent results.
   - **Test in Staging:** Deploy configurations in staging before production to catch issues early.
4) **Adopt GitOps**
   - **CI/CD Pipelines:** Integrate with CI/CD pipelines (e.g., Jenkins) to test and validate configurations before deployment.
   - **Approval Gates:** Require approvals for sensitive updates to ensure impact control.
5) **Versioning and Rollbacks**
   - **Akamai Versioning:** Use Akamai's API for automatic version control.
   - **State Management:** Manage Terraform state files securely using remote backends like AWS S3.
6) **Monitoring and Alerts**
   - **Real-time Monitoring:** Use Akamai's APIs with observability tools for insights on performance and security.
   - **Set Alerts**: Enable alerts for key metrics and implement synthetic testing to validate changes.
7) **Security Best Practices**
   - **Automate Security** Settings: Use Terraform to automate WAF, bot management, and DDoS settings.
   - **Rotate API Keys:** Regularly rotate API keys and enforce HTTPS for secure content delivery.
8) **Edge Logic Automation**
   - **EdgeWorkers:** Automate EdgeWorker deployments and version control to streamline edge processing.
9) **Caching and Purging**
   - **Automate Cache Purging:** Use Akamai's Fast Purge API for timely cache invalidation and apply appropriate TTLs for content.
10) **Multi-environment Strategy**
   - **Separate Environments:** Automate each environment separately to ensure consistency and reduce errors.
11) **Documentation**
   - **Document Configurations:** Keep detailed documentation of configurations and CI/CD workflows for easy reference.
12) **Regular Audits**
   - **Audit Configurations:** Periodically audit configurations to ensure they meet performance and security standards. Review metrics to optimize performance.
   - **Review Usage Metrics**: Regularly review CDN usage, cache efficiency, and traffic patterns to ensure that automation is delivering optimal performance.

5. Discussion

Akamai automation through Terraform simplifies the process of managing and provisioning Akamai properties, enabling DevOps teams to achieve infrastructure-as-code (IaC) practices. By leveraging Terraforms declarative configuration language, users can define Akamai resources such as property configurations, behaviors, and rulesets in version-controlled code. This brings consistency, repeatability, and the ability to track changes over time, which is needed for large-scale deployments. Automation using Terraform also eliminates the need for manual configuration within Akamai Property Manager, reducing human error and speeding up the deployment process. Terraforms state management ensures that the current configuration is always in sync with the defined code, allowing teams to maintain precise control over their Akamai edge platform configurations.

In addition to streamlining the configuration process, terraform automation provides the flexibility to integrate with CI/CD pipelines, further enhancing the operational efficiency of managing Akamai properties. Through automated pipelines, changes to Akamai configurations can be validated, tested, and applied seamlessly without manual intervention. This approach aligns well with the broader DevOps philosophy of continuous integration and continuous delivery, allowing organizations to deploy updates and improvements quickly and with reduced risk. Furthermore, leveraging Terraforms modularity, teams can create reusable components, ensuring that common Akamai configuration patterns are standardized across the organization, driving both efficiency and consistency in edge service management.

6. Conclusion

Automating Akamai Property Manager with Infrastructure as Code (IaC) streamlines property management, reduces manual effort, and enhances scalability. By using tools like Terraform and Akamai CLI, organizations can ensure all configurations are version controlled and integrated with CI/CD pipelines. This journal outlines a guide for implementing IaC for Akamai properties, emphasizing efficient large-scale configuration management, minimal downtime, and fast updates.

While Terraform offers substantial benefits for managing Akamai properties, challenges such as configuration complexity, limited provider features, and large state file performance issues remain. Overcoming these through modular design, manual interventions for unsupported features, and efficient state management enables teams to harness Terraform effectively, reducing errors and accelerating deployments.

References

[1] Terraform Documentation: https://www.terraform.io/docs
[2] Akamai Developer: https://developer.akamai.com
[3] Akamai CLI: https://techdocs.akamai.com/cli/docs.