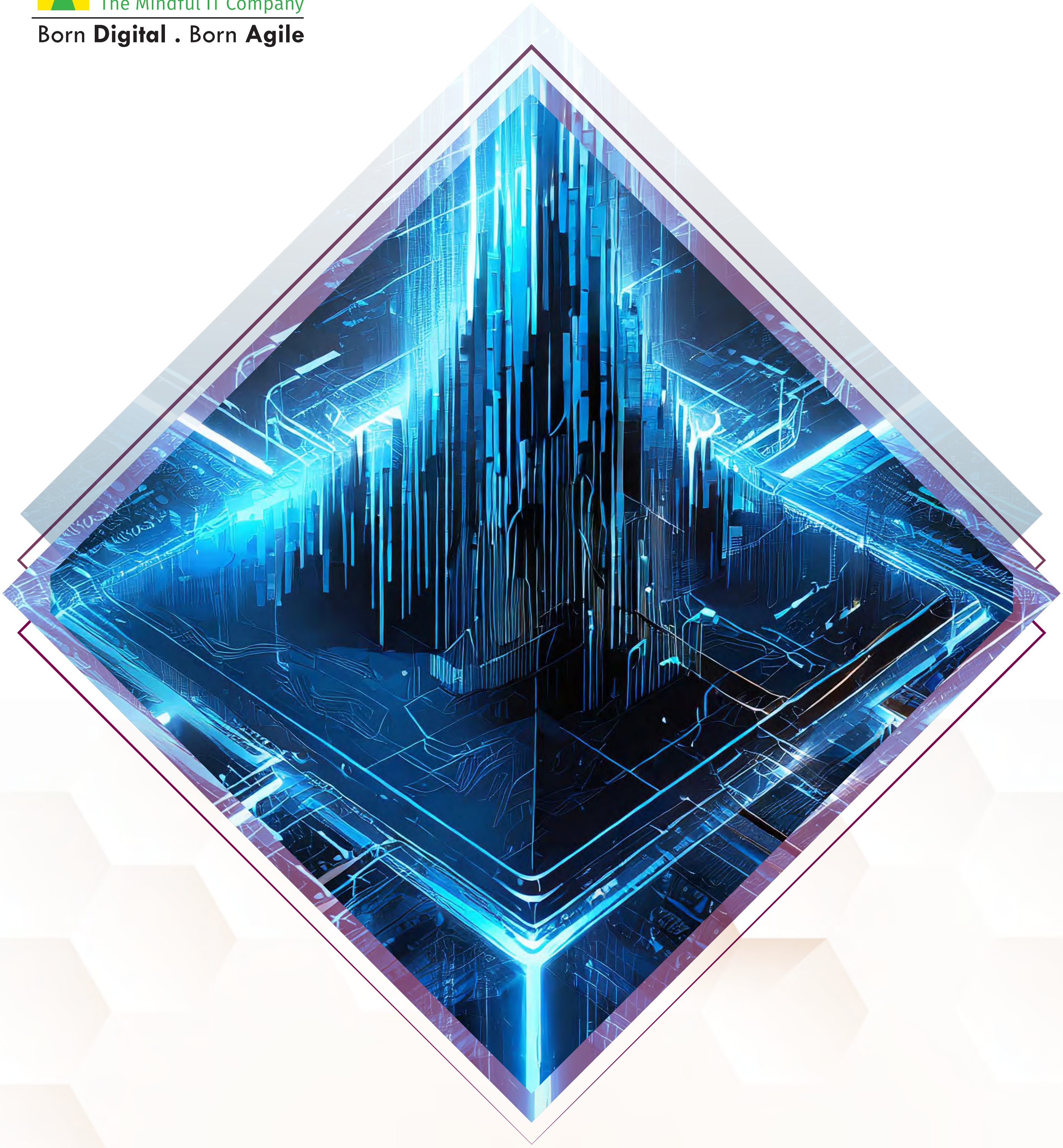
happiest minds
The Mindful IT Company
Born **Digital** . Born **Agile**



PULUMI VS TERRAFORM
INSIGHTS FOR ENHANCED INFRASTRUCTURE

Introduction

In the rapidly evolving domain of cloud computing, Infrastructure as Code (IaC) has emerged as a cornerstone, revolutionizing how we manage and provision IT infrastructure. While several tools have been developed to harness the power of IaC, Terraform by HashiCorp has established itself as a frontrunner, known for its versatile, declarative approach and extensive provider ecosystem. It enables users to define infrastructure in a simple, human-readable language, making it a staple in many organizations' tech stacks.

However, the IaC landscape is dynamic, with newer tools bringing innovative approaches and capabilities. Among these, Pulumi stands out as a modern IaC tool that challenges conventional boundaries. Pulumi differentiates itself by allowing developers to define infrastructure using general-purpose programming languages such as Python, TypeScript, JavaScript, C#, Go, F#, Java, and YAML. This approach not only resonates with software engineers' existing skill sets but also opens doors to more sophisticated, programmable infrastructure management.

This white paper aims to delve into the advantages of Pulumi over Terraform, shedding light on why engineers and organizations should consider it as a viable and potentially superior alternative. We will explore how Pulumi's modern approach to IaC can lead to enhanced productivi ty, greater flexibility, and better integration with existing development practices. By comparing these two leading tools, we can derive valuable insights to make informed decisions about adopting the right IaC solutions to meet the evolving infrastructure needs.

Infrastructure as Code (IaC)

Infrastructure as code, or IaC, provides and manages infrastructure through code instead of manually doing it. Infrastructure is described as code that enables users to efficiently distribute and update configurations while maintaining the infrastructure's desired state. This implies that reproducible infrastructure setups are possible.

"The enabling idea of Infrastructure as code is that the systems and devices that are used to run software can be treated as if they, themselves, are software".

Benefits of IAC

- Higher Speed
- Scalability
- Consistency
- Security
- Accountability
- Disaster Management
- Enhanced Efficiency

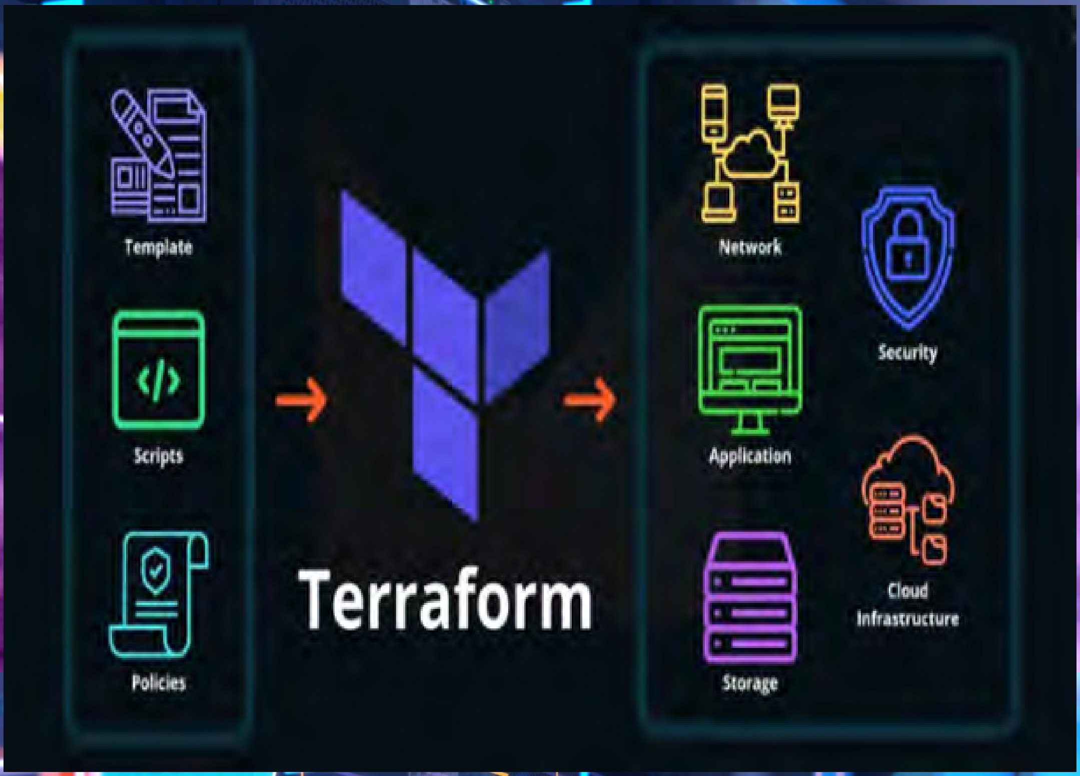
IAC Tools

- Terraform
- Pulumi
- AWS CloudFormation
- Google CDM
- ARM Templates

Terraform

Terraform, introduced by HashiCorp in 2014, has rapidly ascended to become the leading tool in the Infrastructure as Code (IaC) landscape. Its emergence marked a significant shift in how organizations approach infrastructure automation and management. Today, Terraform is extensively used by a multitude of companies worldwide, ranging from small startups to large enterprises, owing to its powerful and flexible infrastructure provisioning capabilities.

Renowned for its declarative configuration approach, Terraform utilizes the HashiCorp Configuration Language (HCL) to enable clear and concise infrastructure definitions. This approach has made Terraform exceptionally popular among organizations seeking to implement consistent, reproducible, and scalable infrastructure management practices.



Terraform no longer an Open-Source

On August 10, 2023, HashiCorp announced a change of license for its products, including Terraform. After 9 years of Terraform being open source under the MPL v2 license, it was to move under a non-open source BSL v1.1 license, starting from the next (1.6) version.

Usage limitations for BSL License

All non-product uses are permitted. All production uses are allowed other than hoisting or embedding the software.

Impact of License Change

DevOps product companies: These companies might face additional licensing costs or restrictions. It affects product development roadmaps to ensure compatibility under new license terms.

SaaS Companies: License change could significantly impact these companies' own infrastructure and automation flows.

Disadvantages of Terraform

Learning Curve	• Terraform's unique syntax, the HashiCorp Configuration Language (HCL), while powerful, can be a hurdle for newcomers.Users often need to invest time to become proficient in HCL.
State Management	• Terraform manages state locally by default, which can lead to challenges in collaborative environments. • While remote state backends are available, setting up and managing these, especially in large teams or projects, can add complexity.
Secret Management	• Secrets are stored in a separate product (Vault). There is no way to encrypt them in the state file.
Testing and Validation	• Terraform does not provide built-in testing capabilities, requiring users to rely on external testing frameworks and tools.
Infrastructure Reuse and Modularity	• Constraint abilities, as can reuse only Terraform modules.
Adopt Existing Resources	• No code generation capabilities while adopting existing resources.
Dynamic Support	• Terraform does not have Dynamic providers, hence the challenge of managing resources or services that do not have an existing Terraform provider.
Error Messages and Debug	• The error messages generated are often cryptic, hence challenging to debug.
Third Party Integration	• Some advanced features and functionality may require integrating third party tool or services, adding complexity to overall setup.

Terraform has long stood as a leading tool in the Infrastructure as Code landscape, praised for its versatility and robust provider ecosystem. However, its recent transition from an open-source model to a Business Source License (BSL) introduces new considerations for users, particularly around licensing restrictions and commercial usage. Coupled with its inherent disadvantages such as its learning curve and state management complexities, this licensing change may prompt organizations and developers to re-evaluate their IaC strategies. As the landscape of cloud infrastructure continues to evolve, it becomes increasingly important for these stakeholders to stay adaptable, balancing the strengths of tools like Terraform with emerging needs and alternatives in the market.

Pulumi

EMPOWERING CLOUD INFRASTRUCTURE WITH OPEN-SOURCE INNOVATION

Pulumi is an open-source infrastructure as code platform that helps teams tame the cloud's complexity using the world's most popular programming languages (TypeScript, Go, .NET, Python, and Java) and markup languages (YAML, CUE). Pulumi is a newer, development friendly tool. It can be used for a wide range of cloud-based infrastructure deployments. As a modern IaC, Pulumi leverages existing programming languages and their native ecosystems to interact with cloud resources.

Pulumi supports all AWS services and stays up-to-date with all AWS features

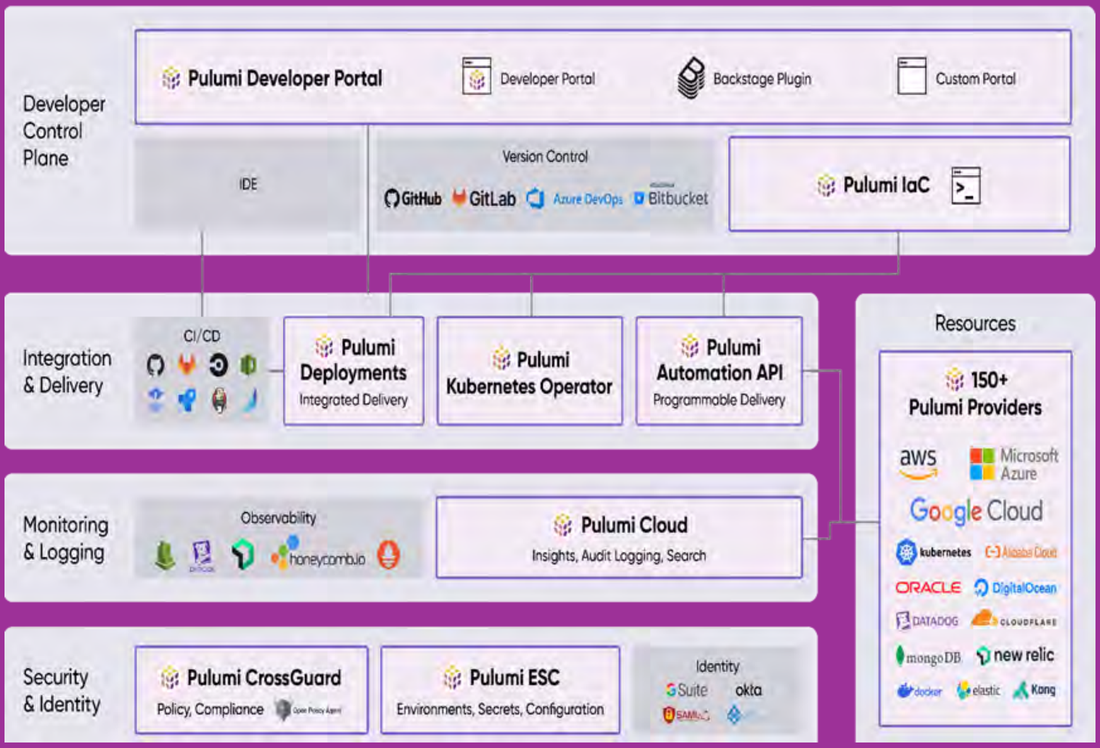
Pulumi supports all Google Cloud APIs and stays up-to-date with all Google Cloud features.

The Azure Native provider is always up-to-date and covers 100% of the resources in Azure Resource Manager (ARM).

Pulumi has first class support for all popular kubernetes tools

Pulumi For Platform Teams

Pulumi's platform-in-a-box approach provide a comprehensive and integrated solution for IaC management. This enables developers to quickly provision approved infrastructure, boosting productivity with pre-configured architectures, automated testing, and deployment adhering to organizational standards



Pulumi's Platform-In-Box Approach

Pulumi Development Portal

Distribute standard private templates through an out-of-the-box Service Catalog experience, which developers can browse and deploy from using the Pulumi Cloud console.

Pulumi IaC

- Utilize open-source IaC in TypeScript/JavaScript, Python, Go, C#, Java, and YAML.
- Build and distribute reusable infrastructure components for 150+ cloud & SaaS providers, supporting modern and cloud-native architectures.

Pulumi Deployments

- Centrally orchestrate automated deployment workflows with `git push to deploy`, UI triggers, and API.
- Advanced capabilities like ephemeral environments and extensibility for drift detection, TTL, blue/green, and more. Integrate with CI/CD, VCS, and more using the API.

Pulumi Backstage Plugin

- Enable developers to browse, deploy, and monitor Pulumi infrastructure deployments from an existing Backstage portal.
- Use the plugin to integrate Backstage with Pulumi Developer Portal, where your private infrastructure templates are hosted.

Pulumi Automation API

Automation API is a programmatic interface for Pulumi CLI, allowing you to embed infrastructure automation into application code that runs on your servers.

Pulumi CrossGuard

Utilize compliance-ready policies for any cloud to enhance compliance and use remediation policies to automatically correct configuration violations like auto-tagging, removing Internet access, and enabling storage encryption.

Pulumi Cloud

- Maintain control and tracking over deployed infrastructure, with complete history of updates and audit logs easily viewable from a console.
- Enhance security with RBAC, identity provider integrations, SSO, and more.

Pulumi ESC

- Centrally store and manage secrets and configuration from different providers.
- It provides a unified, secure location for all your configuration while managing developer access centrally.

Pulumi's Platform-In-Box Approach

- Boost developer productivity
- Enforce standards & compliance
- Increase visibility & observability
- IDP solutions out-of-box
- Collaborate accross DevSecOps

Why Pulumi Over Terraform

Feature	Terraform	Pulumi
Language Support	• Terraform utilizes its domain-specific language (HCL), which is specifically designed for infrastructure configuration. • Developers need to familiarize with a new language.	• Pulumi embraces a wide range of programming languages. It includes Python, JavaScript, TypeScript, Go, and .NET, allowing developers to leverage their existing coding skills and preferences. • This versatility enables teams to choose the language that best suits their needs and integrate infrastructure provisioning seamlessly into their existing workflows.
IDE Support	• Though Terraform is compatible with most IDEs, you need to download external plugins.	• With Pulumi, you can tap into decades of innovation with great IDEs. • The IDEs automatically provide code completion, strong typing, error squiggles, rich resource documentation, and more
Infrastructure Reuse and Modularity	• With Terraform we can reuse only the modules. • Terraform uses HCL which requires you to build proprietary modules and Go-based providers to build modular and reusable infrastructure.	• With Pulumi, you can reuse functions, classes, and packages. • Pulumi also has a built-in component model that lets you abstract and encapsulate complexity with higher-level abstractions. • Pulumi also provides Pulumi Packages, which allows you to author components in one language and make the component accessible in all the other languages that Pulumi supports.
Testing and Validation	• Terraform does not have built-in testing capabilities. However, external testing tools and frameworks can be used alongside Terraform to validate infrastructure code.	• Pulumi provides native support for testing infrastructure code using familiar testing frameworks and tools for the chosen programming language. • Pulumi provides unit tests, property tests and integration tests.

Feature	Terraform	Pulumi
State Management	• Terraform stores the states locally in a file, that can lead to collaboration issues like conflicts when multiple team members working on same infrastructure. • The local file can be vulnerable to security risks such as data loss, corruption, and unauthorized access	• Pulumi uses Pulumi Cloud to manage states. • Pulumi automatically maintains and stores the state securely and is scalable, eliminating the need for manual state file management. • Additionally, Pulumi supports fine-grained resource-level diffs, which can improve deployment efficiency by selectively updating only the necessary resources.
Modes of Execution	• Terraform uses its CL tool to create infrastructure.	• Pulumi Supports CLI to execute commands. • Pulumi also provides two APIs by which you can execute Pulumi commands. • First, the Automation API allows you to provision, update, and destroy infrastructure through Pulumi directly in your application code. • Second, the REST API allows you to query and interact with state information, history, stack tags when using the Managed Pulumi Cloud.
Secrets Management	• Terraform stores all secrets in plain text in the state file, making them visible to anyone who can access the file.	• Pulumi encrypts all secrets both during transmission and while stored, so that they are not viewable in plain text and can only be accessed with the encryption key. • Pulumi also provides an extensible encryption facility that allows you to elect to use your own keys managed by a third-party solution.
Policy as Code	• Terraform provides policy as code through its Sentinel product, which is closed source and limited to Terraform Enterprise and Terraform Cloud. • Sentinel also requires the use of a proprietary HashiCorp Sentinel Language.	• Pulumi, however, provides policy as code through CrossGuard which acts as programmable guardrails to enforce security, best practices, and cost across all infrastructure. • CrossGuard is open source, free to use, and lets you write rules in Python, JavaScript, or Open Policy Agent (OPA) Rego.

Feature	Terraform	Pulumi
Provider support	• Terraform supports across multiple IaaS, SaaS, and PaaS providers.	• Pulumi has deep support for cloud native technologies like Kubernetes and supports advanced deployment scenarios that cannot be expressed with Terraform. • Pulumi supports over 60 of the leading cloud providers and modern cloud SaaS offerings. • Pulumi also has native providers for AWS, Azure, Google, and Kubernetes that provide same-day support for every new release.
Cloud native support	• Terraform offers support for the Kubernetes core API and Helm but has generic support for CRDs, meaning no compile-time type-checking or auto-complete.	• Pulumi supports the cloud native ecosystem. This includes a native Kubernetes provider with 100% Kubernetes API coverage in all languages, including compile-time type-checking. • Pulumi also includes Helm support, strongly typed Custom Resource Definitions (CRDs), deploying Kubernetes YAML or customize templates, as well as a YAML-to-Pulumi conversion tool that can translate any Kubernetes YAML into your desired language. • Pulumi also offers playbooks with built-in best practices for production cluster deployments for AWS EKS, Azure AKS, and Google GKE.
Dynamic provider support	• No support. • Terraform does not have a direct equivalent to Dynamic Providers and would require writing complex and proprietary modules to build custom resources with CRUD operations.	• Pulumi provides dynamic providers that allow you to extend your system by creating new kinds of custom resources by directly coding CRUD operations for the new resource in your Pulumi program. • This can be used to support new resource types in addition to performing complex integrations like database migrations, configuration management for virtual machines, and more, all orchestrated alongside your IaC workflows.

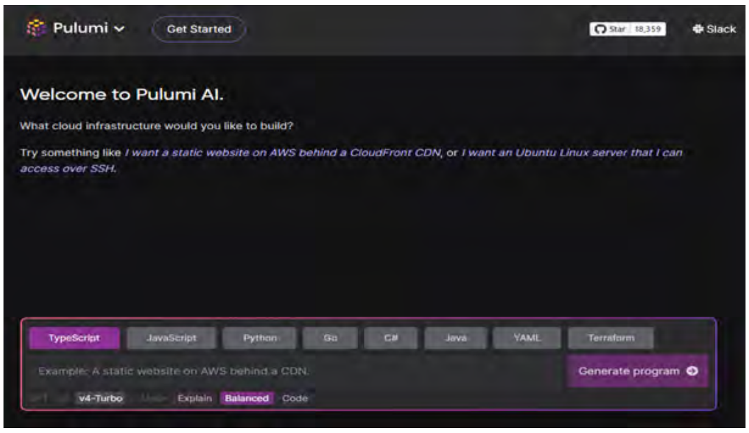
Feature	Terraform	Pulumi
Import code from other IaC tools	• No support.	• Pulumi allows you to convert templates by Terraform HCL, Kubernetes YAML, and Azure ARM into Pulumi programs. • This preserves existing program structure, which may be important if you carefully designed your existing infrastructure as code layout in terms of names, modules, and configurability.
Embedded with Application Code	• No support.	• Pulumi can embed Pulumi programs directly into your application code through the Automation API, a programmatic interface for running Pulumi programs without the Pulumi CLI. • The Automation API is a strongly typed and safe way to use Pulumi in embedded contexts such as web servers without having to shell out to a CLI.
Audit Capabilities	• No support.	• Pulumi provides audit logs that enable you to track the activity of users within an organization. • Audit logs capture the UNIX timestamp of the event, the user who invoked the action, the event that took place, and the source IP of the call the user made. • These logs are available to organizations with an Enterprise level subscription. • The logs are immutable and record all user actions.
Adopt Existing Resources	• Terraform supports importing existing resources. • Terraform only supports importing state but requires you to hand-author the HCL.	• Pulumi also supports importing existing resources. • Pulumi also allows you to generate code in your language of choice from the existing state.
Aliases	• Terraform supports the notion of resource renaming and reparenting but Terraform does not currently support declaratively changing the underlying type of a resource or moving it to another workspace.	• Aliases help facilitate refactoring by allowing you to modify certain properties of a resource without risk of replacing it. • With an alias, you can change the logical name of a given resource, change its parent (i.e., move it from one component to another), change its underlying resource type, or even move it to an entirely different project or stack.

Pulumi AI Support

Pulumi AI is the AI assistant that can create cloud infrastructure using Pulumi.

User can use natural language prompts to generate Pulumi infrastructure-as-code programs in any language.

It builds on the power of Large Language Models (LLMs) and GPT to dramatically reduce the time it takes to discover, learn, and use new cloud infrastructure APIs



Welcome to Pulumi AI.

What cloud infrastructure would you like to build?

Try something like *I want a static website on AWS behind a CloudFront CDN*, or *I want an Ubuntu Linux server that I can access over SSH.*

Pulumi Code vs Terraform Code

A sample code for provisioning AWS EC2 instance

Terraform

```
# Define provider and region
provider "aws" {
  access_key = "YOUR_ACCESS_KEY"
  secret_access_key = "YOUR_SECRET_ACCESS_KEY"
  region = "us-west-2"
}
# Create an EC2 instance
resource "aws_instance" "example_instance" {
  ami           = "ami-0c94855ba95c71c99"  # Amazon Linux
2 AMI ID
  instance_type = "t2.micro"
  key_name      = "my_key_pair"
  tags = {
    Name = "Example Instance"
  }
}
```

Pulumi (in TypeScript)

```
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";
const stackConfig = new pulumi.Config();
const region = stackConfig.require("aws:region"); // Set
your desired AWS region in the Pulumi configuration
const instance = new aws.ec2.Instance("example-instance",
{
    ami: "ami-0c94855ba95c71c99", // Specify the desired
AMI ID
    instanceType: "t2.micro", // Set the desired instance
type
    tags: {
        Name: "example-instance",
    },
});

export const instanceId = instance.id;
```

Pulumi's Support for Terraform

With Pulumi, it is possible to consume both local and remote Terraform state, which can be useful if you are transitioning to Pulumi or if different teams within your organization have different tool preferences. Pulumi can adapt any Terraform Provider for use with Pulumi, enabling management of any infrastructure supported by the Terraform Providers ecosystem using Pulumi programs. By using the state reference support, for instance, you can create higher-level infrastructure in Pulumi that utilizes the VPC information provided by Terraform, such as the VPC ID and Subnet IDs, making the integration between Pulumi and Terraform effortless.

Summary

EMPOWERING TRANSFORMATION-INSIGHTS FROM TOP COMPANIES USING PULUMI

Drawing from in-depth case studies of 20 leading companies, the following analysis offers a detailed look at Pulumi's transformative impact in the realm of IaC. The experiences of these industry leading companies underscore Pulumi's value as a strategic tool for modern cloud infrastructure management.

Few leading companies selected for analysis.

ATLASSIAN | fauna | Mercedes-Benz Research & Development North America, Inc.

LEARNING MACHINE | CISESS Cooperative Institute for Satellite Earth System Studies | snowflake

Why Are Top Companies choosing Pulumi?

The top 5 reasons why companies chose Pulumi are as following.

01

Language Familiarity and Flexibility

Due to Pulumi's support for general-purpose languages like Python, TypeScript and Go to align with existing developer skills, reduce the learning curve and integrate seamlessly with standard development practices.

02

Flexibility, Customization and Enhanced Infrastructure

Due to Pulumi's flexible approach to infrastructure as code, that allows for the tailored solutions that fit specific business needs, enabling more sophisticated infrastructure management.

03

Multi Cloud Support

Due to Pulumi's ability to manage resources across different cloud providers efficiently.

04

DeveloperEmpowerment

Due to Pulumi's "developer-first" approach, empowering developers to take full control of the infrastructure, leading to faster innovation and deployment.

05

Security Features and Compliance

Due to Pulumi's integration of robust security features and compliance mechanisms.

Key Benefits Realized by Organizations

Significant Boost in Deployment Speed

Companies reported an average of 70% improvement in deployment times, translating to faster and more efficient operations

Significant Cost Reductions

On average, organizations using Pulumi observed a 65% decrease in operational costs, highlighting Pulumi's role in driving economic efficiency.

Boosted Productivity

A notable 60% increase in developer productivity was a common theme, underscoring Pulumi's effectiveness in optimizing workflow efficiencies.

Enhanced Time-to-Market

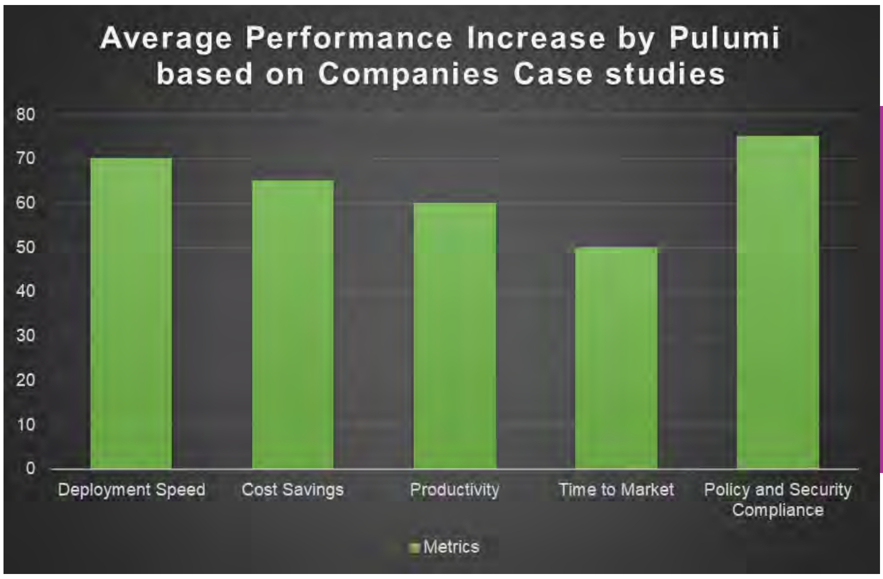
Companies experienced a 50% improvement in their time-to-market for new features and products, a critical factor in maintaining competitive advantage.

Robust Policy as Code &Security Compliance

A standout benefit of 75% improvement in policy and security compliance with Pulumi, showcases Pulumi's superior capability in ensuring robust security standards and adherence to regulatory compliance.

Conclusion

EMBRACING THE FUTURE OF INFRASTRUCTURE MANAGEMENT WITH PULUMI



"Pulumi is not just a tool, it's a transformative force for modern organizations. It empowers entire teams – bridging the gap between developers, operations, and infrastructure. By unifying the language of infrastructure with the language of software development, Pulumi enables a cohesive, efficient, and innovative approach to infrastructure management, driving businesses towards new heights of agility and operational excellence."

As we navigate the ever-evolving landscape of Infrastructure as Code (IaC), it becomes increasingly clear that the choice of tools is not just about managing infrastructure; it's about shaping the future of technology deployment and management. This white paper has journeyed through the foundational concepts of IaC, evaluated the established presence of Terraform, and illuminated the rising significance of Pulumi in this domain.

This white paper reveals that while Terraform has set a high standard in the IaC field with its robust features and widespread adoption, Pulumi emerges as a formidable contender, especially for teams seeking advanced programming capabilities and tighter integration with software development practices. The unique advantages of Pulumi, including its use of familiar programming languages, sophisticated infrastructure management capabilities, and real-time feedback, position it as a tool that not only addresses current needs but also anticipates future challenges in cloud infrastructure management.
As we stand at this technological crossroads, it's imperative for engineers, architects, and IT decision-makers to carefully consider their options. The decision to choose between Terraform and Pulumi should be guided by specific project requirements, team skill sets, and the long-term strategic goals of the organization. The future of IaC is dynamic and promising, and tools like Pulumi are at the forefront of this innovation, offering new possibilities and efficiencies.

The comparison between Terraform and Pulumi is not just a technical evaluation; it is a glimpse into the future of how we build, manage, and evolve our digital infrastructures in an increasingly cloud-centric world.

Author

Anusha Sasindran
Test Lead, PDES

Anusha is a seasoned Test Lead at Happiest Minds Technologies, bringing over 12 years of extensive experience in quality assurance. Her expertise lies in both manual and automated testing, primarily on JavaScript and Java frameworks. Throughout her career, Anusha has worked with a diverse range of clients across various domains and technologies, consistently ensuring the highest quality of products. She is passionate about technology and an avid learner who is always keen on exploring new advancements in the field.