PRODUCTS & TECHNOLOGY

Why Policy as Code?

JAN 12 2018 ARMON DADGAR

HashiCorp advocates for "infrastructure as code" approaches to managing infrastructure. We have talked about it publicly and published about it in our Tao of HashiCorp. At HashiConf 2017, we announced Sentinel, a framework for "policy as code". The same coding practices that are applied to infrastructure can be very effective in enforcing and managing policies. Codifying policy removes the need for ticketing queues, without sacrificing enforcement.

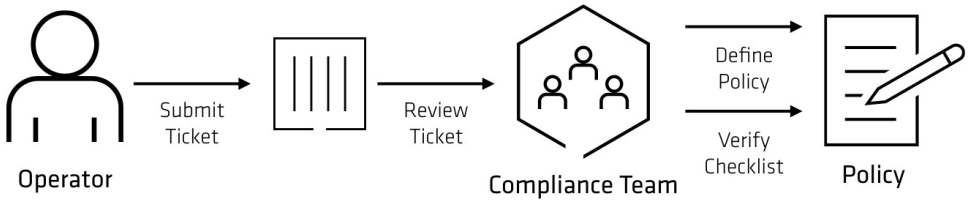
Operators adopt tools like HashiCorp Terraform to provide a simple workflow for managing infrastructure. Users write configurations and run a few commands to test and apply changes. Organizations with governance requirements typically interrupt this workflow with ticketing queues to enforce policies.

What is Policy?

Policy is a nebulous term and can span many different categories:

- **Compliance Policies.** These policies ensure compliance with external standards such as PCI-DSS, SOC, or GDPR. External industry working groups or government agencies establish and mandate these standards.
- **Security Policies.** Security policies adopted internally protect data privacy and infrastructure integrity. For example, ensuring only certain applications run on public networks or expose specific ports to the Internet.
- **Operational Excellence.** These policies prevent service outages or degradation. For example, a policy may mandate at least two service instances, or validation of new configurations.

Historically, most policies are defined and enforced the same way. Policy authors translate the business requirements into a Word document with specific rules that should be enforced. Enforcement of policy is based on a ticketing workflow, where tickets are filed against a compliance or security team. For each ticket, the policy document is used to verify the ticket which is either approved or denied.



This workflow is slow, error prone, and makes it difficult to scale either the number of policies or the number of tickets being reviewed. End to end automation is difficult since the ticketing workflow is asynchronous and slow, and the implementation is often different for each system.

What is Policy as Code?

Our driving mission has been infrastructure as code, the idea that infrastructure can be codified and automated with tools like Terraform and Nomad. This approach allows configuration to be tested, peer reviewed, versioned, automated, and re-used much like application code. Applying this approach to policy seemed natural and brings the same benefits.

Treating policy as code requires a way to specify policies and a mechanism to enforce them. Traditionally, policies were defined in Word documents and enforced manually. Instead, Sentinel provides a simple policy-oriented language to write policies, and integrates with our tools like Terraform Enterprise and Nomad Enterprise to enforce them. Below is an example Sentinel policy that ensures staging is deployed to the "us-east-1" region, while production is deployed to "us-west-1":

```
import "tfplan"

valid_regions = {"production": "us-west-1", "staging": "us-east-1"}
env_region = valid_regions[tfplan.variables.env]

is_not_aws = func(type) {
        return type is not "aws"
}

// Check the provider alias region matches the environment region
validate_aws_region = func(provider) {
        return all provider.alias as a {
                a.config.region is env_region
        }
}

main = rule {
        all tfplan.config.providers as type, provider {
                is_not_aws(type) or validate_aws_region(provider)
        }
}
```

The goal of the language is to be simple to read and write by individuals with a limited background in programming. This motivated us to write a custom language rather than adopt a general purpose programming language.

Testability

One of the other benefits of Sentinel is that it also has a full testing framework. Consider the previous example, we could write a test case to validate its correctness using simple json syntax:

```
"mock": {
    "tfplan": {
      "variables": {
```

```
        "env": "production"
      },
      "config": {
        "providers": {
            "aws": {
              "alias": [
                {"config": {"region": "us-west-1"}}
              ]
            }
        }
      }
    }
  },

  "test": {
      "main": true
  }
}
```

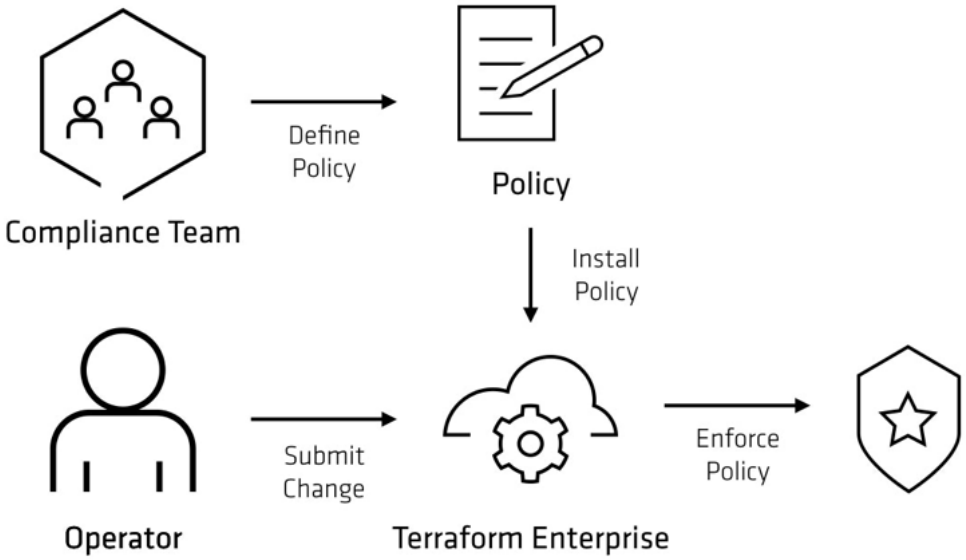
We can then use the Sentinel command line tool to check our policy:

```
$ sentinel test -verbose
PASS - staging.sentinel
  PASS - test/staging/pass_not_aws.json
    trace:
      TRUE - staging.sentinel:15:1 - Rule "main"
        TRUE - staging.sentinel:17:3 - is_not_aws(k)
  PASS - test/staging/pass.json
    trace:
      TRUE - staging.sentinel:15:1 - Rule "main"
        FALSE - staging.sentinel:17:3 - is_not_aws(k)
        TRUE - staging.sentinel:17:20 - validate_region(v)
  PASS - test/staging/fail_aws_invalid_region.json
    trace:
      FALSE - staging.sentinel:15:1 - Rule "main"
 FALSE - staging.sentinel:17:3 - is_not_aws(k)
        FALSE - staging.sentinel:17:20 - validate_region(v)
```

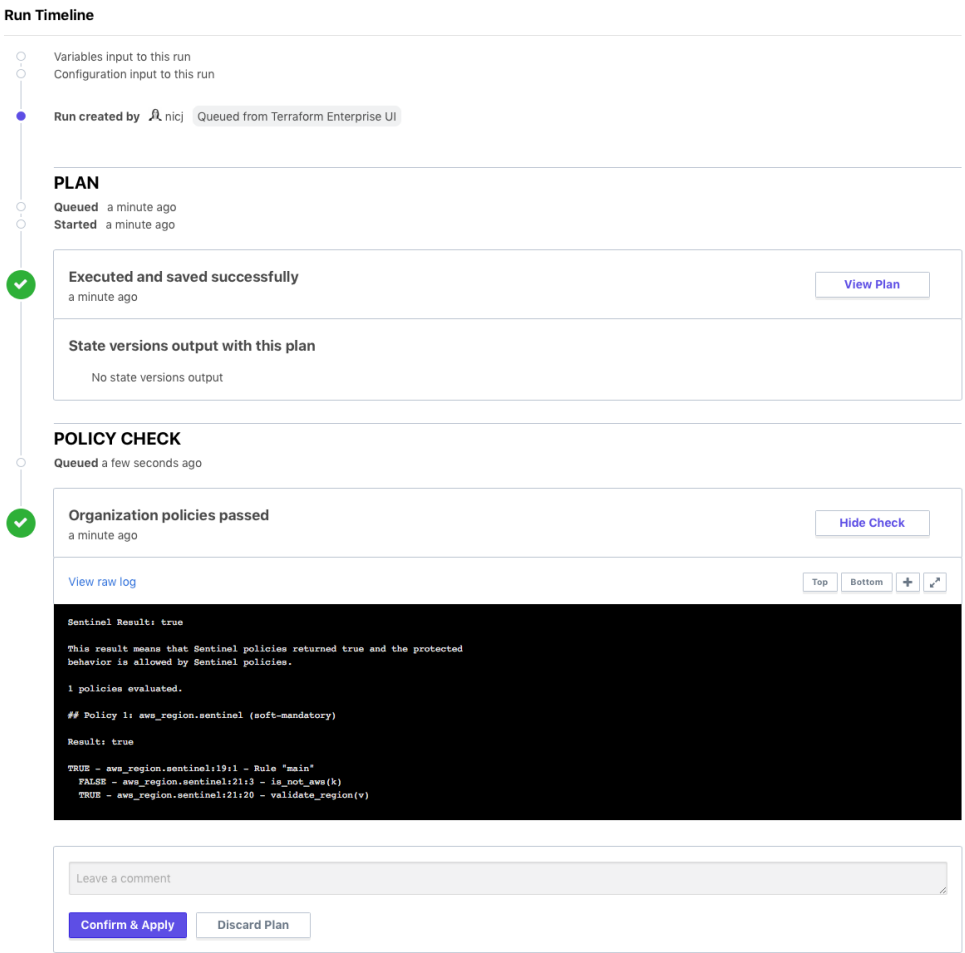
Using this approach we can construct multiple test cases for pass and fail scenarios, allowing us to easily validate our policy without having to execute it. This also allows us to change our policy and ensure regressions are not introduced.

Executing Policy

Once written, the policy can be enforced automatically without manual review. This enforcement is best done in the change path, so that policy violations can be prevented rather than detected. Detecting policy violations requires a separate remediation workflow and is more cumbersome than preventing the problem entirely. Here is an example enforcement workflow with Terraform Enterprise:



This workflow decouples the policy definition and enforcement, keeping a fast feedback loop for operators. Using automation avoids the human error inherent to a manual review process, and allows a large number of policies or changes as well.



Conclusion

Using a ticketing queue and manual review for policy enforcement can bottleneck changes.

Policy as code uses codified policies and automated enforcement. This approach extends on infrastructure as code, which brought similar benefits. Sentinel is a framework for policy as code that we introduced and have integrated into Terraform Enterprise, Vault Enterprise, Nomad Enterprise, and Consul Enterprise. Avoiding a ticketing workflow allows organizations to provide more self-service capabilities and end-to-end automation, minimizing the friction for developers and operators.

Learn more about Sentinel: https://www.hashicorp.com/sentinel.

Sentinel

More blog posts like this one

JUNE 23 2022 | PRODUCTS & TECHNOLOGY

New on HCP: Boundary, Waypoint, Drift Detection, and Azure Support

HashiCorp Cloud Platform has added several new capabilities, including managed services for HashiCorp Boundary and Waypoint, and Drift Detection for Terraform Cloud.

JUNE 21 2022 | PRODUCTS & TECHNOLOGY

Terraform Cloud Adds Drift Detection for Infrastructure Management

Drift Detection for Terraform Cloud continuously checks infrastructure state to detect and notify operators of any changes, minimizing risk, downtime, and costs.

JUNE 20 2022 | PRODUCTS & TECHNOLOGY

The 3 Phases of Infrastructure Automation

From adoption to standardization to operating and optimizing at scale, t... evolution of infrastructure automation is critical to modern hybrid and multi-cloud environments.

Business Email address

Subscribe to Newsletter

I agree to HashiCorp's Privacy Policy.*

INFRASTRUCTURE
- Terraform
- Packer

NETWORKING
- Consul

SECURITY
- Vault
- Boundary

APPLICATIONS
- Nomad
- Waypoint
- Vagrant

RESOURCES
- Blog
- Tutorials
- Community
- Events
- Integrations
- Library
- Partners
- Podcast
- Support
- Training

COMPANY
- About Us
- Jobs WE'RE HIRING
- Press Center
- Investors
- Brand
- Contact Us

System Status Cookie Manager Terms of Use Security Privacy Trademark Policy Trade Controls

stdin: is not a tty