Pre-Deployment Policy Compliance

Abhishek Tiwari

Citation: *A. Tiwari*, ”Pre-Deployment Policy Compliance”, Abhishek Tiwari, 2023. doi:10.59350/ngt0n-5dm13

Published on: July 23, 2023

The ability to deploy applications quickly and efficiently is crucial for organizations. Agile methodologies and continuous integration/continuous deployment (CI/CD) pipelines have become the norm, enabling rapid release cycles and frequent updates. However, amidst the drive for speed, ensuring policy compliance is often overlooked, leading to potential security vulnerabilities and compliance risks. Pre-deployment policy compliance, supported by policy as code frameworks such as Sentinel, Open Policy Agent (OPA), Conftest, etc. is an emerging approach that enables organizations to ensure secure deployments while adhering to policies and regulatory requirements. In this article, we will explore into the significance of pre-deployment policy compliance, explore the role of policy as code frameworks, and showcase real-life examples of successful implementations.

Understanding Pre-Deployment Policy Compliance

Pre-deployment policy compliance involves verifying that software applications adhere to specific policies, regulations, and guidelines before they are deployed into production environments. These policies encompass a wide range of concerns, including security best practices, data protection measures, industry regulations, privacy laws, accessibility requirements, and internal organizational policies.

In traditional software development practices, compliance checks often occur late in the development lifecycle, closer to deployment. However, this approach carries risks, as non-compliance discovered at that stage may result in costly rework and potential security breaches.

To address these challenges, the concept of "Shift Left" has gained prominence, emphasizing the early integration of compliance checks in the software development process. By adopting a proactive approach, organizations can identify and address compliance issues early, reducing the likelihood of non-compliant software being deployed.

Policy as Code Frameworks: Enabling Pre-Deployment Policy Compliance

Policy as code frameworks play a critical role in facilitating pre-deployment policy compliance. They allow organizations to codify policies into machine-readable formats, which are then automatically validated against code, configurations, and deployment artifacts. By integrating policy as code frameworks into CI/CD pipelines, compliance checks are seamlessly integrated into the development process.

Here are some policy as code frameworks used for pre-deployment policy compliance:

Sentinel: Developed by HashiCorp, Sentinel is a policy as code framework that integrates with various HashiCorp products, including Terraform, Vault, and Consul. It allows organizations to define policies in a domain-specific language (DSL) tailored to infrastructure-as-code (IAC) configurations.

For instance, Sentinel policies can enforce rules such as “All AWS S3 buckets must have server-side encryption enabled” or “Only specific IAM roles can access certain resources.”

Open Policy Agent (OPA): OPA is an open-source policy as code framework that provides a general-purpose policy engine. It supports multiple query languages, including Rego, which allows organizations to express policies declaratively. OPA can be integrated with a wide range of tools and systems, making it highly versatile. For example, OPA can be used to enforce policies related to Kubernetes deployments, API access control, and more.

Conftest: Conftest is a tool designed to enforce policies on configuration files. It integrates with various configuration management tools like Kubernetes, Helm, and Terraform. Organizations can use Conftest to ensure that configuration files adhere to specific policy rules before deployment. For example, Conftest can validate that Kubernetes deployments have the required resource limits defined or that Terraform configurations adhere to organizational naming conventions.

terraform-compliance: As the name suggests, terraform-compliance focuses on enforcing compliance on Terraform configurations. It allows organizations to define compliance rules using natural language constructs or BDD like specifications. For instance, terraform-compliance can enforce that all AWS EC2 instances have specific tags defined or that all Google Cloud resources have encryption enabled.

Checkov: Checkov is a static code analysis tool. Checkov scans IAC files for misconfigurations that may lead to security or compliance problems., offering 750+ predefined policies to detect security and compliance issues. Custom policies can also be created and contributed.

Benefits of Pre-Deployment Policy Compliance with Policy as Code Frameworks

Policy as code frameworks automate compliance checks, reducing the need for manual validation. This ensures that compliance checks are consistently applied to all code changes and configurations.

By integrating policy as code frameworks into CI/CD pipelines, compliance checks are shifted left, enabling early identification and resolution of policy violations. This reduces the risk of non-compliant code being deployed to production.

Policy as code frameworks allow organizations to continuously monitor compliance throughout the development lifecycle. As code changes are made, policies are reevaluated, ensuring ongoing adherence to policies.

Early detection and resolution of compliance issues lead to faster deployments. Organizations can confidently deploy code knowing that it adheres to all relevant policies, reducing the time required for regulatory reviews.

Real-Life Examples of Pre-Deployment Policy Compliance

Let's explore real-life examples of how organizations have successfully implemented pre-deployment policy compliance using various policy as code frameworks:

Sentinel for Infrastructure as Code

A cloud service provider uses Terraform for infrastructure provisioning. To ensure adherence to security and compliance requirements, they employ Sentinel policies to validate Terraform configurations.

Sentinel policies are written to enforce rules such as:

- Requiring encryption for all Amazon S3 buckets and Azure Blob Storage containers.
- Restricting access to specific AWS services based on defined IAM roles.
- Ensuring that security groups restrict inbound traffic to approved ports.

By leveraging Sentinel to validate Terraform configurations, the cloud service provider guarantees that their infrastructure deployments meet the organization's stringent security standards.

Open Policy Agent (OPA) for Kubernetes Policies

A technology company manages a Kubernetes cluster to deploy microservices. To enforce best practices and security measures, they implement OPA as an admission controller for Kubernetes.

OPA policies are written in Rego to enforce rules such as:

- Ensuring all images come from a trusted registry
- Ensuring that pods are assigned to specific namespaces with appropriate labels.
- Enforcing resource limits and requests for CPU and memory usage.

With OPA as an admission controller, the technology company enforces policy compliance for all Kubernetes resources, safeguarding against unauthorized deployments and potential security risks.

```
package kubernetes.admission

import future.keywords

deny contains msg if {
    input.request.kind.kind == "Pod"
    some container in input.request.object.spec.containers
    image := container.image
    not startswith(image, "docker.com/")
    msg := sprintf("image '%s' comes from untrusted registry", [image])
}
```

Conftest for Configuration Compliance

A DevOps team manages a Kubernetes cluster and Helm charts for deploying microservices. To ensure consistent configurations across deployments, they utilize Conftest to validate Helm charts and Kubernetes manifests.

Conftest policies are written to enforce rules such as:

- Verifying that all Kubernetes deployments have resource limits defined to prevent resource over-utilization.
- Enforcing the use of specific labels and annotations for service discovery and monitoring.
- Validating that Helm charts use the correct release names and namespaces.

By integrating Conftest into their CI/CD pipeline, the DevOps team ensures that all deployments adhere to predefined configuration policies, promoting consistency and reducing the likelihood of configuration-related issues.

Checkov for Terraform Compliance

A cloud-native startup leverages Terraform to provision cloud infrastructure on multiple cloud platforms. To maintain security and compliance across their infrastructure code, they integrate Checkov into their CI/CD pipeline.

Checkov provides out-of-the-box checks for Terraform configurations, such as:

- Ensuring that S3 buckets have server-side encryption enabled.
- Verifying that security groups have restrictive inbound rules to minimize exposure.
- Ensuring that new RDS services spun-up are encrypted at rest

By running Checkov during their CI/CD process, the startup gains confidence that their Terraform configurations comply with industry best practices and organizational policies.

```
from checkov.common.models.enums import CheckResult, CheckCategories
from checkov.terraform.checks.resource.base_resource_check import
   BaseResourceCheck


class RDSEncryption(BaseResourceCheck):
    def __init__(self) -> None:
        name = "Ensure all data stored in the RDS is securely encrypted at
            rest"
        id = "CKV_AWS_16"
        supported_resources = ("aws_db_instance",)
        categories = (CheckCategories.ENCRYPTION,)
        super().__init__(name=name, id=id, categories=categories,
           supported_resources=supported_resources)


    def scan_resource_conf(self, conf: dict[str, list[Any]]) ->
       CheckResult:
    """
        Looks for encryption configuration at aws_db_instance:
        https://www.terraform.io/docs/providers/aws/d/db_instance.html
    :param conf: aws_db_instance configuration
    :return: <CheckResult>
    """
    if 'storage_encrypted' in conf.keys():
        key = conf['storage_encrypted'][0]
        if key:
            return CheckResult.PASSED
    return CheckResult.FAILED
```

terraform-compliance for AWS Compliance

A financial services company manages a complex AWS infrastructure. To ensure compliance with internal security policies and AWS best practices, they utilize terraform-compliance as part of their deployment process.

terraform-compliance policies are written to enforce requirements such as:

- Requiring all AWS EC2 instances to be launched within a specified VPC.
- Verifying that AWS RDS instances have backup and retention policies in place.
- Well-known insecure protocol exposure on Public Network for ingress traffic

By integrating terraform-compliance into their CI/CD pipeline, the financial services company ensures that their AWS infrastructure adheres to predefined compliance standards.

```
Scenario Outline: Well-known insecure protocol exposure on Public Network
   for ingress traffic
    Given I have AWS Security Group defined
    When it has ingress
    Then it must have ingress
    Then it must not have <proto> protocol and port <portNumber> for
       0.0.0.0/0
   Examples:
     | ProtocolName | proto | portNumber |
     | HTTP         | tcp   | 443        |
     | Telnet       | tcp   | 23         |
     | SSH          | tcp   | 22         |
     | MySQL        | tcp   | 3306       |
     | MSSQL        | tcp   | 1443       |
     | NetBIOS      | tcp   | 139        |
     | RDP          | tcp   | 3389       |
     | Jenkins Slave| tcp   | 50000      |
```

Conclusion

Pre-deployment policy compliance, along with the adoption of policy as code frameworks has become essential for ensuring secure and compliant software deployments. By integrating compliance checks early in the development process, organizations can mitigate security risks, reduce rework, and demonstrate adherence to regulations and policies. As the software development landscape continues to evolve, pre-deployment policy compliance will remain a key strategy for organizations aiming to deliver secure, compliant, and high-quality software solutions to meet the demands of modern development practices.