cloudrail

A Practical Guide to

Continuous Compliance for Your Cloud Infrastructure

A 5-step journey to achieve continuous compliance without sacrificing speed.

A Practical Guide to Continuous Compliance for Your Cloud Infrastructure

Abstract

Continuous infrastructure automation extends the agile and DevOps practices used in software development to infrastructure engineering. Think of it as CI/CD for infrastructure that enables you to rapidly and safely deploy your cloud environments in a seamless and repeatable process. Gartner predicts 70% of organizations worldwide will implement continuous infrastructure automation to improve business agility by 2025. This is a 5-step journey to help you get started with this initiative to achieve continuous compliance without sacrificing speed.

1. Start with infrastructure automation by adopting Infrastructure-as-Code (IaC) enabling developers to become more autonomous and agile.
2. Add IaC to the CI/CD pipeline so you can start treating infrastructure as software projects for continuous integration and delivery.
3. Automate security validations in the pipeline to catch security violations and prevent security and compliance issues before deployment.
4. Analyze IaC with the live cloud environment to detect configuration drift and hidden issues.
5. Enforce policy to prevent security issues from making it to your environment.

The ebook team and contributors

Ryder Damen
Ulrica de Fort-Menares
Charles Kim
Yoni Leitersdorf
Carlos Mora
Oscar Quintas
Devassy Tharakan

Introduction

With the new digital era, every product is being transformed through the addition of a software stack. To cope with the explosion of digitized products, organizations are making security a priority to ensure users have a trustworthy and positive experience. Recently, the industry has seen a surge in “product security programs” to cope with the new challenges. Organizations are realizing product security is critical to business sustainability and growth.

What is Product Security?

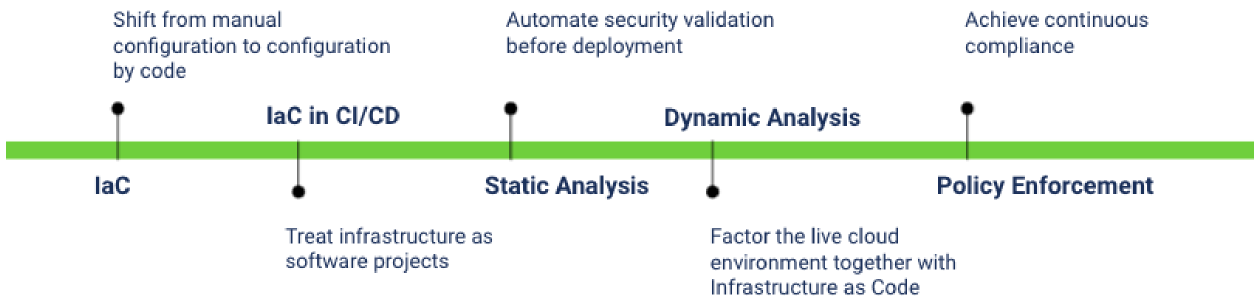
Product security is about securing the entire product environment with a focus on designing security into the product from the beginning. Product security should be factored into all stages of product development, including requirements, design, development, product testing, release, and post-deployment.

Application Security is different from product security. One of the key aspects of traditional Application Security is about securing application code through testing throughout the software development life cycle. With Software-as-a-Service, it’s important to consider the entire stack including cloud infrastructure, not just the release of artifacts. Once the product is released, it’s also necessary to manage ongoing product security operations. In some way, product security is a superset of application security, infrastructure security, and security operations.

In this ebook, we will present a framework to help product security professionals implement infrastructure continuous compliance in the cloud. The framework applies the same disciplines and quality gates that are used to manage applications, to cloud infrastructure. The goal is to ensure the cloud environment stays secure at all times throughout the development life cycle and post-deployment. Automation, version control, security integration testing, and deployment pipelines are all in scope.

Infrastructure Continuous Compliance Framework

This is a five-step journey to make security a priority for your cloud infrastructure without compromising speed.


Shift from manual configuration to configuration by code
IaC in CI/CD
Automate security validation before deployment
Dynamic Analysis
Achieve continuous compliance
IaC
Static Analysis
Policy Enforcement
Treat infrastructure as software projects
Factor the live cloud environment together with Infrastructure as Code

Step 1: Start with automating infrastructure deployment using Infrastructure-as-Code.
Step 2: Treat infrastructure as software projects by integrating IaC with the CI/CD pipeline.
Step 3: Automate infrastructure security scanning before deployment with Static Analysis as unit testing.
Step 4: Factor the cloud environment together with IaC to catch configuration drifts and hidden issues.
Step 5: Implement policies to achieve continuous compliance.

In the following chapters, we will discuss each stage in more detail. We will explain the rationale for each stage, discuss the challenges you will likely encounter, and share how you can mitigate these challenges. We'll suggest metrics to measure success for each stage of the journey and include an actionable checklist for implementations.

Chapter 1: Move From Manual Provisioning to Infrastructure Automation

Why Automation?

As you scale your cloud environments, keeping your infrastructure manageable can be quite a challenge. Organizations are rapidly adopting Infrastructure as Code (IaC) for faster and more scalable deployments. IaC is the concept of managing and provisioning your infrastructure the same way you do for code. Instead of manually spinning up an environment from a console, you provision your entire infrastructure with code. IaC is more than just infrastructure automation - it applies traditional software practices to ensure that code can be easily reused to redeploy multiple resources. Repeatable infrastructure is easier to manage as your organization grows.

Increase Availability

Not only does infrastructure automation reduce the error rate associated with manual provisioning of resources, it enables you to seamlessly run parallel infrastructures. One of the challenges in deploying new software to production is minimizing downtime. The blue-green deployment approach maintains two identical production environments, enabling you to release new software with minimal downtime. For example, a feature can be tested in the blue environment while the green environment is live. Once the feature is working in the blue environment, you can safely promote the blue environment to face live users. Infrastructure automation is the key enabler for the blue-green deployment approach.

Automated Documentation

Automated documentation is another benefit of IaC. Automation provides a level of documentation for IT to be able to support and extend operations. With automated documentation, code itself will document the state of the cloud environment. For example, in FDA (Food and Drug Administration) regulated environments, the physical location of the server where data is stored (e.g. rack number) must be identified. With a public cloud and Infrastructure-as-a-Service, this policy is not realistic. Instead, you can leverage IaC as documentation to identify your existing resources. Gone are the days of having to maintain tedious - often outdated - documentation. IaC is always an accurate documentation of its live environment, as long as changes are not made out of the IaC process (see drift detection further below).

Productivity Gain

Organizations that achieve the automation and self-service ideals of DevOps are able to deploy applications faster than ever before. Imagine a developer is building a new service that requires a new lambda function to be created in AWS. Before committing the new code, the developer needs to test the code in a production-like environment. Pre-IaC days, the developer typically had to get in line to submit an infrastructure change request and wait for the change to happen. With IaC, developers can themselves make the infrastructure change. By leveraging a Git repository that contains declarative descriptions of infrastructure reflecting the production environment, a developer can launch a cloned environment. They can test the new code using nearly an exact replica of the latest production environment within a matter of minutes. This means a developer has full control, owning all application and infrastructure stacks. Essentially, IaC enables developers to achieve the goals for becoming more autonomous and agile. This level of efficiency makes agile delivery a reality.

Automation Challenges

While automation adds a lot of value to the IT environment, there are some challenges that should be considered, especially at the beginning of that journey.

1. Lack of coding skills
   While you don't need to be an expert at coding, the learning curve for IaC can be an investment if you don't already have experience with JSON, Hashicorp Configuration Languages (HCL), YAML, Ruby, etc. The shortage of these skill sets can hamper your IaC initiatives.

2. Aversion to Change
   You should be prepared for objections like "We don't make changes to infrastructure often, so it would be easier and cheaper to manually spin up the resources from the console."

3. Talent retention
   There is a cut-throat competition to attract the best talent in the market. For organizations that do have an automation initiative, their first challenge is to attract the right talent. The next challenge is to uplift the rest of the organization who come from operations backgrounds with limited or no development experience. How do you make automation a new culture? How do you retain the new talent?

Automation Best Practices

1. Promote a DevOps Culture Internally
   Infrastructure automation serves as a starting point for DevOps adoption, allowing teams to release features more frequently and more reliably. By promoting a DevOps culture, it can help attract diverse talent.

2. Management support can make this happen
   The key to success is getting management support. Automation can be a huge undertaking and the first automation tasks are going to take time. With help from management, creating an automation culture is notably easier.

3. Acceleration through Outsourcing
   Consider outsourcing Infrastructure-as-Code services for the initial implementation. This provides you with an opportunity to familiarize yourself with the processes and tools. The consulting professionals can train your staff.

Automation Metrics

Metrics are important tools that help you focus your teams and resources on the automation tasks. Consider leveraging these Key Performance Indicators for your automation initiatives.

- [ ] Number of workloads migrated from manual provisioning to automated deployment
- [ ] Number of deployments per day with automation
- [ ] Lead time to deploy infrastructure - measure the time it takes to provision a server by filling out a form. Compare the time between manual process and automated deployment

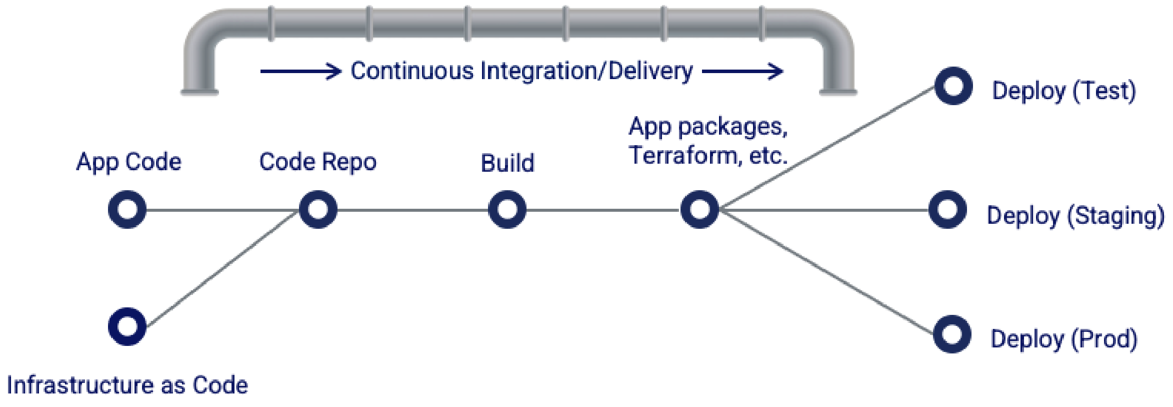
Automation Checklist

- [ ] Determine the IaC language(s) that your company will adopt
- [ ] Identify skill gaps
- [ ] Identify training requirements
- [ ] Ensure your ticketing system supports the new automation workflow
- [ ] Identity and prioritize the workloads for automation

Chapter 2: Operate by Merge Requests with IaC in the CI/CD

With infrastructure automation in place and represented as code, you can take the software-oriented approach to infrastructure. IaC offers the opportunity to introduce continuous integration and continuous deployment in your development process.

Continuing with the example in Chapter 1, after the new service is successfully tested in an isolated environment, the developer is ready to commit the code with a merge/pull request (PR). Instead of managing the infrastructure change in its own silo, it makes sense that the updated infrastructure definition is version-controlled and stored in Git alongside the version with the new application code. This naturally provides a detailed audit trail for changes. Version control is also very helpful for restoring any potential changes or diagnosing any issues.



Improve Efficiency

By integrating IaC into the CI/CD pipeline, infrastructure changes can be templatized and tracked, along with the application source code. In an outdated process, a developer submitted their infrastructure change request via ticketing systems with typically vague descriptions of the request. A member of the IT team would then need to process the ticket and send it for a revision process by a review team. Once all aspects had been reviewed, the ticket was approved. Eventually, the infrastructure change was applied by the operation. With a CI/CD integration, the change request is done via a Pull Request (PR) where all changes are reflected. The vague description disappears, replaced instead by a declarative description of infrastructure changes represented as code. By automatically triggering infrastructure deployment upon the merge of the PR, it also ensures only changes approved will be applied.

Increase Controls

Instead of allowing users to deploy cloud infrastructure directly from their workstations, check your IaC into a code repository such as GitHub and leverage a CI/CD pipeline to control deployments. In addition to keeping a centralized source of truth, it has the added benefit of limiting the number of administrators required in your cloud account. Using this mechanism also enables easier rollback of changes, by simply rolling back the code changes created in the PR.

Improve Quality

With infrastructure treated like software, infrastructure changes can be tested early in the process. Once an infrastructure change is deployed, system tests are run automatically to ensure that the infrastructure change hasn't broken the application or introduced performance degradation. Adding IaC to the continuous integration workflow enables deployment of infrastructure alongside your application software in the same pipeline.

Managing your infrastructure just like your application code with Git and CI/CD tools can increase the rate at which you produce quality code. This new trend is known as GitOps. Developers can use Git and merge pull requests to manage both infrastructure provisioning and software development.

Challenges integrating with the CI/CD Pipeline

1. CI/CD pipelines for application development are well understood. However, CI/CD pipelines for infrastructure is a new concept.

2. How can we best avoid delays when using Pull Requests for infrastructure changes? What test automation is necessary to identify the infrastructure change?

3. Is a feature branches approach or a trunk-based approach the best strategy for our organization?

CI/CD Integration Best Practices

1. DevOps Outsourcing
   Improve time to market and tap into experienced DevOps talent. Without the time or expertise to manage a complicated application infrastructure, outsourcing is a valuable practice.

CI/CD Integration Success Metrics

- ☐ Number of issued/approved/rejected PRs per workload
- ☐ Mean time between PR creation and PR approval

CI/CD Integration Checklist

- ☐ Identify and document the new change management process. This should include how it works, roles, and responsibilities
- ☐ Communicate the new change management process and provide new process training where applicable

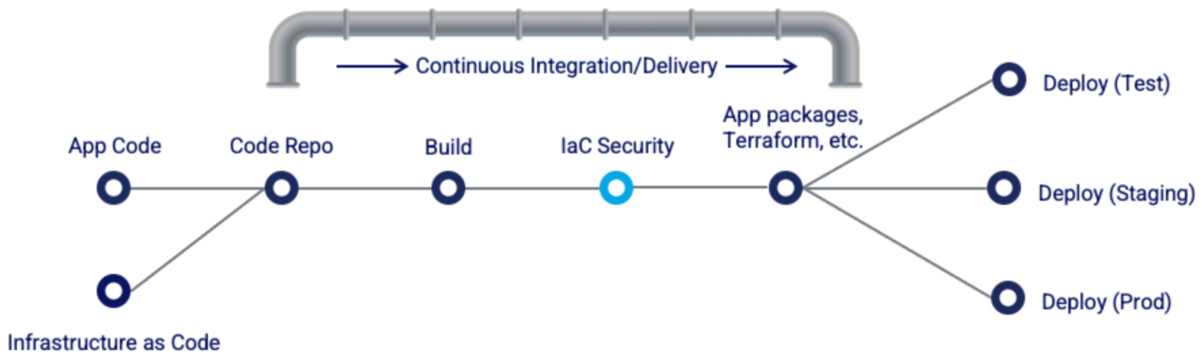
Chapter 3: Catch Security Issues in the Pipeline with IaC Static Analysis

While the GitOps concept promises faster and more frequent deployments, the last thing you want is to be slowed down by legacy security programs. How often has a release at your organization been stopped because it failed the security gate towards the end of the release process? All too often, security testing is tacked on at the end of the delivery process. Consequently, developers spend significant time and energy investigating these security issues, causing delays to the release. Uncovering issues late into the cycle creates unnecessary stress and frustrates developers.

Context switching is expensive for developers who constantly do focused work. Bothering developers with security issues that happened a few sprints ago leads to greater lost time. Much like unit testing, providing immediate feedback, so security issues can be addressed within the same sprint, is immensely valuable. Catching issues late in the development cycle is expensive and painful to fix.

Early Security Testing with Static Analysis

Incorporating security testing early in the development process is a much more effective strategy. IaC tests can be employed continuously to evaluate security impacts of changes and quickly provide feedback to developers to avoid context switching.



Introducing static analysis for IaC to the CI/CD pipeline is a great way to strike the right balance between governance and speed by providing guardrails for developers, thereby keeping developers from unintentionally creating security issues in the cloud. Developers no longer need to get in line for security reviews. Instead, IaC will be automatically evaluated for security impacts. Security controls are integrated into the infrastructure development lifecycle. For the first time, the security team has early visibility to the cloud security posture throughout the development cycle.

Challenges with Static Analysis

Businesses want more features faster. There is a general perception among developers that security often slows down a release. If security is a bottleneck, it's a non-starter. A core tenet of DevOps is to increase feature velocity. The goal of the security integration is to deploy secure cloud environments at the speed of the business.

Challenges to consider:

1. Lack of Alignment

The friction between security teams and development teams is ongoing in many organizations. Developers have extremely demanding schedules and are often frustrated with the security gates slowing down the delivery pipeline. The security teams are typically risk-averse and are often misrepresented as inhibitors to innovation.

2. Talent shortages in DevOps and Security

Shortages of cybersecurity professionals have been a real problem. According to the 2020 ISC2 Security Workforce Study, the global cybersecurity workforce needs to grow 89% to effectively defend organizations' critical assets. Similarly, skill shortages are a common problem for DevOps professionals. Unfortunately, the talent shortage is here to stay.

3. Getting buy in

Creating a security mindset to understand the importance of early security checks and avoid delaying such security reviews requires a cultural change.

4. False Positives

Static Analysis is notorious for being noisy with many false positives. A noisy security tool can be counterproductive if it inadvertently stops pipelines, leading to developer frustration.

Static Analysis Best Practices

1. ### Focus on visibility initially

   When first implementing Static Analysis for your infrastructure, security scanning should not impact the pipeline. Security violations should be for information purposes initially. Getting visibility of the security posture should be the first goal and this needs to be clearly communicated to all stakeholders.

2. ### Empathy and Collaboration

   You should always put yourself in another person's shoes. It is important to communicate clearly with each other and set strong boundaries. The security folks should collaborate with the development team to introduce security controls at a pace that makes sense to them. The development team should have a partner on the security side of the house. It is always good to reflect back on a decision. Together, you can jointly make a decision that makes sense for both sides. You can take corrective actions if a decision made has a negative impact.

3. ### Establish shared goals

   Establish shared goals between the development team and security teams. Often the development and security teams have conflicting goals. Alignment from the top and across the organization is often needed here.

Static Analysis Success Metrics

- [ ] Number of issues potentially impacting the pipelines
- [ ] Number of critical issues prevented before deployment
- [ ] Rate of deployments per week has increased
- [ ] How many services have been validated
- [ ] Number of policies incorporated
- [ ] Number of false positives
- [ ] Time takes to deal with false positives (and how)
- [ ] How long does it take to fix issues
- [ ] Number of issues detected trend
- [ ] Is the feedback from the static analysis tool quick enough?

Static Analysis Integration Checklist

- ☐ Select the right tool. A few questions to ask:
    - ☐ Have you collected the requirements from your stakeholders?
    - ☐ How does the tool work with your team's workflow?
    - ☐ What is the acceptable execution time for static analysis?
    - ☐ Open-source tools or commercial products?
    - ☐ Do you already have a roadmap to help you pick the right tool that meets your long term requirements?
- ☐ Determine where you would insert a scan that would be the least painful for your team
- ☐ Provide guidance to identify the criticality of the security issues.
- ☐ Define your bug fix policy including expected time to fix critical security issues.
- ☐ Provide guidelines to identify false positives
- ☐ Establish the process to deal with false positives

Chapter 4: Evaluate Security Issues with IaC Dynamic Analysis

While static analysis for IaC is a big leap forward for developers and the security team, it does present some new challenges. Static analysis is notorious for generating many false positives and can inadvertently disrupt a developer's workflow. In the world of application security, organizations have faced similar challenges with noise. This has forced the evolution from SAST (Static Application Security Testing) to DAST (Dynamic Application Security Testing) and IAST (Interactive Application Security Testing). A similar shift is imminent in the IaC security arena.

Dynamic Analysis evaluates the IaC together with the live cloud environment in order to predict security issues before deployment. For example, most static analysis tools will alert if a security group is too permissive allowing any ingress traffic or a security group permits SSH traffic from the public internet. What if that security group is not used?? Or the group is used, but the attached machine resides in a subnet where the network access control list (NACL) disallows port 22? Dynamic analysis analyzes the IaC along with the live environment, and understands that the NACL blocks SSH traffic. The security group, while ill-configured, is not a real problem to deal with at this time. Dynamic analysis keeps noise levels down.

Hidden Security Issues

Besides being noisy, many security issues only manifest themselves through analysis within the deployed environment. As a result, static analysis is unable to detect many cloud security issues. Imagine you have created an RDS database, placing it in a subnet that has internet access, but without a public IP address. In such a case, the database is not publicly accessible.

```
resource "aws_db_instance" "test" {
 ... (usual DB parameters) ...
 db_subnet_group_name = aws_db_subnet_group.db.name
 vpc_security_group_ids = [ aws_security_group.db.id]
 publicly_accessible = false
}
```

Now imagine someone else, working with the same cloud account, is intending to deploy a new EC2 instance separately on the same subnet. This EC2 instance would be publicly accessible.

```
// This instance can potentially be used to hop into the DB
resource "aws_instance" "public_ins" {
  ami = "ami-0130bec6e5047f596"
  instance_type = "t3.nano"
  associate_public_ip_address = true
  vpc_security_group_ids = [aws_security_group.publicly_accessible_sg.id]
  subnet_id = "subnet-samesubnetasdbwascreatedonabove"

}
```

Each Terraform code is not necessarily a cloud security concern by themselves. However, the user creating the RDS database can potentially have their database exposed to outside access by the other deployment. The owner of the EC2 instance Terraform file inadvertently introduced a new cloud security concern to the RDS instance, and the EC2 owner had no idea. Worse yet, the owner could have been a third party.

This is another reason for dynamic analysis. The only way to catch this type of hidden issue is to concurrently evaluate the proposed plan with the live cloud environment. Otherwise, it is very easy to overlook potentially serious security concerns.

Infrastructure Drift Detection

Another reason for dynamic analysis is cloud infrastructure drift. Infrastructure drift is change that occurs in a cloud environment that differs from the originally deployed method. For example, if you provision a resource using Terraform, and you then make a change using the web console, that is considered a drift. Without a tightly controlled change management process, drift can be a source of security issues resulting in potential blind spots. Imagine you are in the middle of an audit. Your policy is to use IaC as the single source of truth. When an auditor asks you how you manage your IAM, you proudly present your Terraform code without realizing a drift has occurred, causing non-compliance. No doubt that one of the biggest challenges in an IaC managed infrastructure is to spot discrepancies as they happen. To effectively operationalize your cloud environment with IaC, it’s necessary to ensure the IaC in the Git repository is the “single source of truth”.

Challenges with Dynamic Analysis

1. Configuration Drift
   Any changes performed outside of the approved IaC change management process have great impacts, either in daily operations or in security. Sometimes changes are inevitable, but it's important to be aware of them.

2. Security Concerns
   The need to scan a production environment with customer information can pose security concerns when it comes to access to sensitive data.

3. Time it takes to scan may be an issue for very large deployments.

Dynamic Analysis Best Practices

1. Follow the principle of least privilege when assigning permissions to tools that require access to a live environment.

2. Drift detection
   When handling drift detection, a manual approach may be acceptable initially. In this case, the event is manually reported via a ticketing system. An IaC engineer then must update the corresponding IaC templates to reflect the drift. The best approach is to automate the process, detecting drifts, and evaluating security violations automatically.

3. Apply techniques to obfuscate sensitive data.

Dynamic Analysis Success Metrics

- [ ] Number of critical issues prevented before deployment
- [ ] Deployment trend - Is the # of deployments growing
- [ ] How many services have been validated
- [ ] Number of policies incorporated
- [ ] How to fix detected security issues
- [ ] How long does it take to fix issues
- [ ] Number of issues detected trend
- [ ] Number of drifts
- [ ] Mean time to detect in live environment
- [ ] Number of false positives

- [ ] Time takes to deal with false positives
- [ ] Time it takes to complete the dynamic analysis

Dynamic Analysis Integration Checklist

- [ ] Tool selection
- [ ] Verify the permissions
- [ ] Identify how quickly you want to determine configuration drifts?
- [ ] Tools to identify configuration drifts

Chapter 5: Enforce Policy to Achieve Continuous Compliance

By embracing IaC, collaborating through merge/pull requests, and implementing CI/CD, you are effectively in a position to deliver a faster and more predictable product. By introducing security evaluations with static analysis early on, you gain visibility into your cloud posture throughout the development lifecycle. With dynamic analysis, you operationalize your IaC managed infrastructure. You've essentially taken a 'security by design' approach to managing your cloud. With this, you are now in a position to enforce policy as code as part of DevOps workflows. This means stopping the pipeline and alerting stakeholders when a security issue important to your organization is detected. This way, you can prevent security issues from ever making it into your production environment and reduce the possibility of security breaches.

Challenges with Policy Enforcement

1. How to ensure the right guardrails are in place?
2. How to make sure you are enabling the necessary security controls and not too many?
3. How to ensure policy enforcement does not slow down development too much?

Policy Enforcement Best Practices

1. Collaborate with stakeholders to identify security policies you want to enforce.
2. Resolve security issues before enforcing security policies to minimize impact to the pipeline.
3. Only apply enforcement to new resources and not existing resources. This means that existing resources with certain violations will not be impacted and can be resolved separately, to avoid slowing down adoption of policy enforcement.

Policy Enforcement Success Metrics

- [ ] Number of policies enforced
- [ ] Number of times the pipeline was stopped & trend
- [ ] Deployments trend is growing
- [ ] How many services have been validated
- [ ] How long does it take to address security issues

Policy Enforcement Checklist

- [ ] Identify stakeholders who may be impacted by the new policy enforcement process
- [ ] Acceptance from stakeholders
- [ ] How will the new policy enforcement impact existing resources?
- [ ] Communicate the new security policies

Conclusion

Modernizing your Infrastructure

As you transition to cloud-native development and cloud modernization efforts, you have the opportunity to implement security by design without sacrificing speed. We hope you find this roadmap useful in guiding your organization's digital transformation. This security-centric journey is a new paradigm that is moving toward a preventive cloud security strategy. By incorporating security early in the development lifecycle, you can now take appropriate preventive steps to remediate misconfigurations and security risks before deployment, while achieving continuous compliance.