bridgecrew
BY PRISMA® CLOUD

# The DevSecGuide to Infrastructure as Code

DevSecOps has paved the way for teams to automate security and embed it into the DevOps lifecycle. In this guide, we'll explore the challenges of leveraging DevSecOps to secure the cloud and how infrastructure as code makes it all possible.

# Table of contents

# Introduction to infrastructure as code

Infrastructure as code refers to the technology and processes used to manage and provision infrastructure using code. It enables DevOps processes such as version control, peer reviews, automated testing, tagging, continuous integration, and continuous delivery.

## The rise of IaC

Infrastructure as code—also known as IaC—was first introduced in 2009 by DevOps company Puppet in response to the traditional methods of deploying and managing infrastructure. According to Puppet:

*"The older methods of infrastructure management—manual processes and documentation, brittle single-purpose scripts, and graphical user interface based tools—each had their uses in the past. Today, though, with the perpetual need to scale infrastructure, adoption of ephemeral infrastructure, and greater application system complexity, new ways of keeping things under control are needed."*

Since then, IaC has become the foundation for several companies such as Ansible, Chef, Salt, and others. In more recent years, IaC's popularity has been driven by Terraform, a popular open-source IaC framework by Hashicorp used primarily to define resources in public cloud services. Terraform has made IaC limitlessly customizable and accessible, paving the way for the surrounding IaC ecosystem.

Simultaneously, cloud providers have created their own configuration frameworks to help simplify and automate infrastructure orchestration and management. AWS CloudFormation, Azure Resource Manager (ARM), and Google's Cloud Deployment Manager all make it easier for infrastructure engineers to build repeatable environments.

## How it works

IaC can be either declarative, meaning it defines **what** is going to be provisioned, or imperative, meaning it defines **how** it's going to be provisioned. Terraform and CloudFormation are both examples of declarative frameworks while AWS Cloud Development Kit (CDK) is an example of an imperative IaC framework. Kubernetes is also closely aligned with IaC in that its configuration can be defined in the form of code.

Each specific framework has its own conventions and syntax, but IaC is generally made up of resource declarations, input variables, output values, configuration settings, and other parameters. IaC is most often JSON, HCL, or YAML-based and contains all the configuration needed to spin up your infrastructure, such as compute, networking, storage, security, identity access management (IAM), and more.

```
module "s3_bucket" {
  source = "terraform-aws-modules/s3-bucket/aws"

  bucket = "my-s3-bucket"
  acl    = "private"

  versioning = {
    enabled = true
  }

}
```

A Terraform module that creates a private S3 bucket with versioning enabled.

# Benefits of infrastructure as code

Because IaC uses code to define what's needed to get resources up and running, it enables the ability to automate and scale cloud provisioning with improved repeatability.

## Automation

Today's businesses deploy countless applications daily, and infrastructure needs are constantly changing to meet those demands.

IaC simplifies cloud provisioning by templating all manual configurations into code. By transforming manual infrastructure configurations into machine-readable templates, IaC makes it so that developers don't have to provision and manage infrastructure manually. Instead, it enables engineers to develop, test, and deploy new infrastructure through automated workflows.

## Scalability

IaC makes it easier and more foolproof for teams to configure cloud resources at scale while reducing the risk of misconfiguration without spending unnecessary time and resources. Automation and code configuration make it much easier to deploy cloud services the same way each and every time.

It also makes it easier to de-provision infrastructure when it's not in use, decreasing overall computing costs and maintenance expenses.

## Repeatability

Consistency is key for cloud infrastructure. With IaC, compute, storage, and networking services are deployed the same way every time so that you can maintain consistency across resources and even across multi-cloud environments. That consistency puts human error at a minimum with the potential for complete versioning and logging. The repeatability factor means you can provision more resources with less effort while maintaining high-quality standards, security best practices, and compliance with industry benchmarks.

## Security

Infrastructure as code provides a crucial opportunity for collaboration across teams. By provisioning cloud resources across environments and clouds with a unified, common language, developers and operations can more easily stay on the same page and work together to keep cloud-native applications secure.

**To summarize, IaC saves teams time and resources by:**

- Leveraging automation to make resource provisioning scalable and fast.
- Minimizing human error by making infrastructure deployment consistent and predictable.
- Fostering team collaboration and codifying institutional knowledge that reduces future risk.

# Infrastructure as code risks

For all its flexibility and benefits, IaC comes with some drawbacks that teams should be aware of—especially around security and compliance.

## Adoption gaps

Much like bringing in a new open-source library or SaaS platform, IaC has the ability to improve efficiencies but needs the right level of buy-in and awareness. Because IaC is still relatively new, one of the biggest challenges teams face when adopting IaC is accurately integrating new frameworks with existing infrastructure.

Bringing IaC into your stack may introduce added complexity and confusion as to how and where your resources are provisioned, governed, and secured.

## Immutable drift

With IaC, instead of managing infrastructure in one centralized console, you have another framework running in parallel that should be your source of truth. If changes are made to infrastructure independent of the code provisioning it, you may experience drift between running resources and their underlying IaC configuration.

Without adequate tooling in place, it can also lead to misconfigurations and risk in your environment.

## Security gets left behind

Misconfigurations are the leading source of cloud data breaches. And according to some sources, 99% of organizations' misconfigurations go unnoticed.

Cloud security tooling provides the necessary visibility and monitoring to combat that statistic but provides feedback that can be at odds with IaC.

For teams leveraging IaC, surfacing misconfigurations reactively after they've been merged and deployed can result in friction between teams and additional work created to investigate, prioritize, schedule, and finally, implement a fix.

The best way to combat that challenge is to shift cloud security left so instead of monitoring infrastructure in runtime, you scan for misconfigurations in code. As our research shows, however, addressing security issues at the IaC level has lagged. We'll explore this in the next section.

**To summarize, IaC can cause friction and risk due to:**

- Gaps in how fully IaC is embraced and embedded across environments, teams, and workflows.
- Miscommunication as to how and where infrastructure is governed and policies are enforced.
- Cloud security tooling getting left behind rather than providing proactive, actionable feedback.

# Open source IaC security challenges

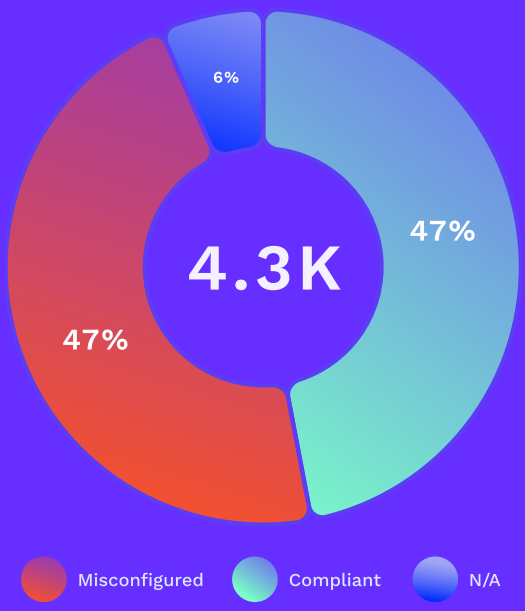
Although IaC enables ready-built, open-source templates or modules, it's important to understand that these components aren't typically shared with a security-first mindset.

The open-source IaC economy is gaining momentum in places like GitHub as well as in purpose-built repositories such as the Terraform Registry and Artifact Hub.

While open source IaC makes it easier and faster for developers to get cloud services up and running, security is often an afterthought.

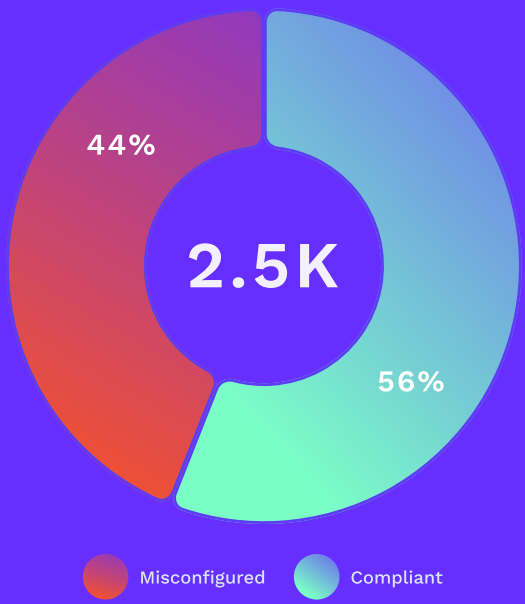
Our research shows that around half of all open-source Terraform modules within the Terraform Registry and Helm charts within Artifact Hub contained misconfigurations. This research highlights the gap in how and where cloud security is being addressed. It also shows how much security has lagged and how big of an impact DevSecOps can have for securing cloud infrastructure.

Scanned Helm charts by compliance




Read more about these findings in our Open Source Helm Security research.

Scanned Terraform modules by compliance




Read more about these findings in our State of Open Source Terraform Security Report.

# DevSecOps challenges with IaC

Without the right approach, strategy, and tooling adopting DevSecOps to secure the cloud can end up creating more bottlenecks and friction between teams.

It's no secret that the motivations of developers and security are often at odds. DevOps wants to move fast and work iteratively while security's out-of-sprint, reactive feedback gets in the way.

DevSecOps aims to bring security into the fold of DevOps to avoid misconfigurations or weak implementations that can leave cloud-native applications vulnerable. Without the right approach, incorporating a DevSecOps strategy comes with its own list of challenges—especially when it comes to securing the cloud.

There are two main reasons for this. First, the infrastructure landscape is rapidly changing, and keeping up with these changes requires experience and expertise that may not always be available within the organization. Second, many existing processes and tools were not designed with cloud security in mind and can therefore create bottlenecks for cloud-native engineering and security teams.

## Inconsistent governance

DevSecOps in the cloud is difficult because of the lack of processes for developing scalable infrastructure. IaC is an answer to that call, alleviating the performance and cost-related challenges of deploying infrastructure at scale.

Policy governance is of the utmost importance with IaC, as it creates additional permutations of cloud infrastructure managed by different teams with varying workflows. These workflows create layers of complexity and ambiguity as to where and when policies are being enforced. At best, that ambiguity creates redundancy. At worst, it can lead to risk.

**How does this unfold in the wild?**

- Engineering assumes that security and compliance policies are being enforced at the cloud level, using cloud provider tools or other security solutions.
- Security is unaware of all the provisioning frameworks in place and whether or not the resources they are deploying have the proper policies in place.

Having a workflow to close this gap is key, or else cloud resources deployed with misconfigured code will result in additional engineering work.

### Tip: Fix issues at the source

If a misconfiguration only gets fixed in runtime when IaC is in use, there is a 70% chance of it coming back. Fixing issues at the source prevents misconfigurations from resurfacing down the line.

## Skills and access gaps

Automated tools are great at surfacing everything from critical security issues down to informational best practice violations. But even the latest and greatest tool cannot substitute for reliable cloud DevSecOps processes and workflows.

That's because misconfigurations identified by a tool still need to be understood in context, prioritized, and remediated. Getting security and engineering together to do that isn't always straightforward. The problem is that security and compliance typically don't have the necessary access to code repos or cloud consoles to implement fixes. Even if they do, they may lack the know-how and full application context to fix cloud misconfigurations themselves. On the flip side, engineering lacks the security context and understanding to prioritize fixes accurately. Even if they did have the knowlege, having the time to actually implement the fix after they've switched contexts is also a challenge.

**Here's an example of how automation might exacerbate the very bottleneck it is trying to alleviate:**

- Security opens a ticket, assigns it to a team lead who maybe is or isn't working on the team for which the issue is related.
- After some Jira roulette, that remediation gets slotted into some numbered sprint.
- Meanwhile, ten more "P0" tickets have been created.
- With automated scanning in place, those tickets easily snowball into hundreds more and the whole process is repeated ad nauseam.

Now add in the well-documented cybersecurity skills shortage, and the result is security and compliance responsibilities get put on the plates of developers who are often not adequately prepared.

Whether there are security resources at hand or not, it is unrealistic to expect developers to have comprehensive knowledge of all possible configuration issues they may face when building cloud infrastructure. This makes it easy for misconfigurations to be deployed.

Unaddressed misconfigurations aren't just risky; they become exponentially more costly to address the further away from developers' workstations they get.

**To summarize, DevSecOps can end up causing more friction if there is:**

- Unclear and inconsistent policy governance between infrastructure code and deployed cloud resources.
- Skills and access gaps when it comes to identifying and fixing misconfigurations that may lead to cloud risk.

# Tips for embracing cloud DevSecOps

So far we've addressed potential challenges for approaching DevSecOps in the cloud. These are the three things you can do to overcome those challenges.

## Automate everything

Your engineering teams are likely already automating much of your software testing, from unit testing to dependency scanning. Cloud security should be no different. Even the most seasoned security experts can't be expected to stay up to date with every single security best practice across every cloud provider, service layer, orchestration platform, resource, etc., and apply it back to their specific architecture.

Automated scanning is the only way to comprehensively and programmatically apply security and compliance guardrails without spending excessive time and resources combing through documentation. Without actionable feedback, automation can end up creating more friction and work for engineering. That's why finding the proper tooling—whether commercial solutions or open source tools such as Checkov—to automate both finding and fixing issues is imperative.

## Leverage existing processes

Surfacing even the most actionable feedback at the wrong time in the wrong place can be noisy and unproductive. The only way to make DevSecOps in the cloud easily adopted is by embedding it into the tools and processes developers depend on every day. Luckily for us, the key to embracing automation and security-as-code is already likely in place.

Where and how you enforce guardrails will depend on several factors, including your current suite of tools, organizational goals, and security maturity.

## Secure in code

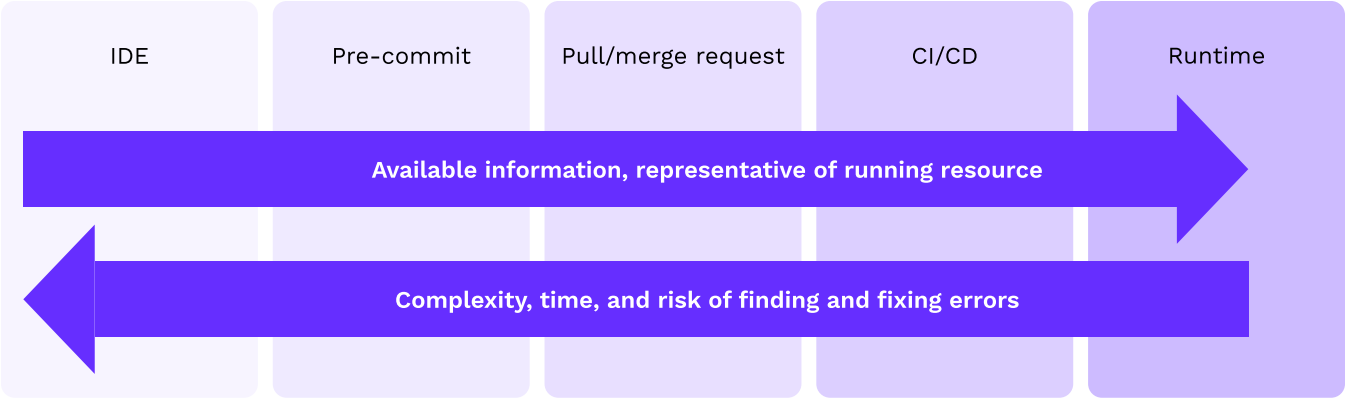
Much of the DevSecOps friction comes from the historically reactive security mindset, which relies on monitoring for issues rather than preventing them in the first place. With IaC, you can flip that mindset into a proactive approach. Whether you call it policy-as-code or security-as-code, translating governance automation into a common language equips both security practitioners and DevOps engineers with timely feedback and actionable solutions.

Enforcing policies at the code layer ensures that cloud security is consistently applied and enables it to scale throughout the environment over time. It will also end up saving you resources in the long run, as it can cost 100x more to fix a software flaw in production than in code. It's also true, however, that the earlier in the lifecycle you are, the less information is available about what infrastructure will look like and, thus, what misconfigurations may be present.

That's why it's crucial to address infrastructure security throughout each phase of the DevOps lifecycle.

# Cloud security in the DevOps lifecycle

Although there's no one-size-fits-all solution for every organization, the key to encouraging the adoption of cloud DevSecOps is to enforce guardrails early in IaC and surface actionable feedback throughout the DevOps lifecycle.




## IDE scanning

Raising flags as soon as possible is the epitome of shift-left security, and what better place to surface feedback than the time and place developers are actually writing code?

To shift cloud security as far "left" as possible (besides the design and plan phases), embedding guardrails into your Integrated Development Environment (IDE) is the best way to go. This likely will come in the form of a plugin or extension, such as the Checkov VS Code extension. Because it minimizes context switching, IDE scanning is the cheapest and most foolproof way to identify issues.

## Pre-commit hooks

Pre-commit unit and integration testing is a generally accepted best practice. Now, pre-commit IaC security scanning should be as well.

Scanning IaC locally for misconfigurations is the best way to address errors in the safety of your own workspace before code gets integrated into a shared repository. Another major benefit of pre-commit scanning is not having to waste time failing builds (and delaying your teammates' builds) or tripping the wire on pull request checks.

Whether you're taking the feedback and implementing changes in the moment, or you're using IDE suggestions as an educational tool for writing secure infrastructure, there are very few downsides. The only drawback of local scanning is that the onus is on the developer to actually run the scans and make the changes.

With the right balance of passive analysis and actionable feedback, you can become a better security advocate without an outsized amount of effort.

## Pull/merge request checks

For teams who live and die by their version control systems (VCSs), embedding security into sanctioned code review processes has many benefits. Depending on what triggers your CI/CD builds, this approach can be used with, or instead of, scanning via CI/CD.

Your VCS may also afford some unique controls for implementing guardrails. All three major platforms (e.g., GitHub, GitLab, and Bitbucket) offer versioning, branching, and embedded code review, and authorization controls that allow developers to test without compromising running production systems and continue making code changes before merging.

### Tip: Add security guardrails

You may want to set up your code review settings to block merges when checks fail, further ensuring that misconfigured IaC doesn't make its way to your master branch.

## CI/CD jobs

CI/CD pipelines are critical in compiling infrastructure and testing compiled code before it's deployed. Scanning this level of abstraction is important to surface misconfigurations in the core resources, variables, and dependent modules about to be provisioned.

The other benefit of embedding infrastructure security into the CI/CD pipeline is that it's automated and can be fully customized for your workflow. You can determine what kinds of checks fail builds and surface feedback directly in your CI/CD provider. It's also a collaborative process that your entire team is using to review, reject, and approve changes.

Regardless of whether you're scanning in CI/CD or your VCS, this level of control promotes collaboration and accessibility to developers.

## Security in runtime

Keep in mind that developer-first security doesn't preclude "traditional" cloud security methods of monitoring running cloud resources for security and compliance misconfigurations.

Even if you're fully covered by IaC, manual changes can still happen, resulting in unintentional drift between code and running cloud resources. Cloud drift management can provide insight into those resources to determine potentially risky gaps.

Relying solely on build-time findings without attributing them to actual states in runtime could result in configuration clashes. Because runtime scanning follows the actual states of configurations, it is the only viable way of evaluating configuration changes over time when managing configuration in multiple methods. It's also a great way to satisfy compliance audits that require continuous change-control auditing and tracing.

**A well-rounded cloud DevSecOps program should include:**

- Making tools accessible to developers for early, inexpensive feedback.
- Enforcing guardrails collectively within your VCS or CI/CD pipeline.
- Getting continuous visibility into runtime resources to address drift resulting in manual configuration changes.

# Conclusion

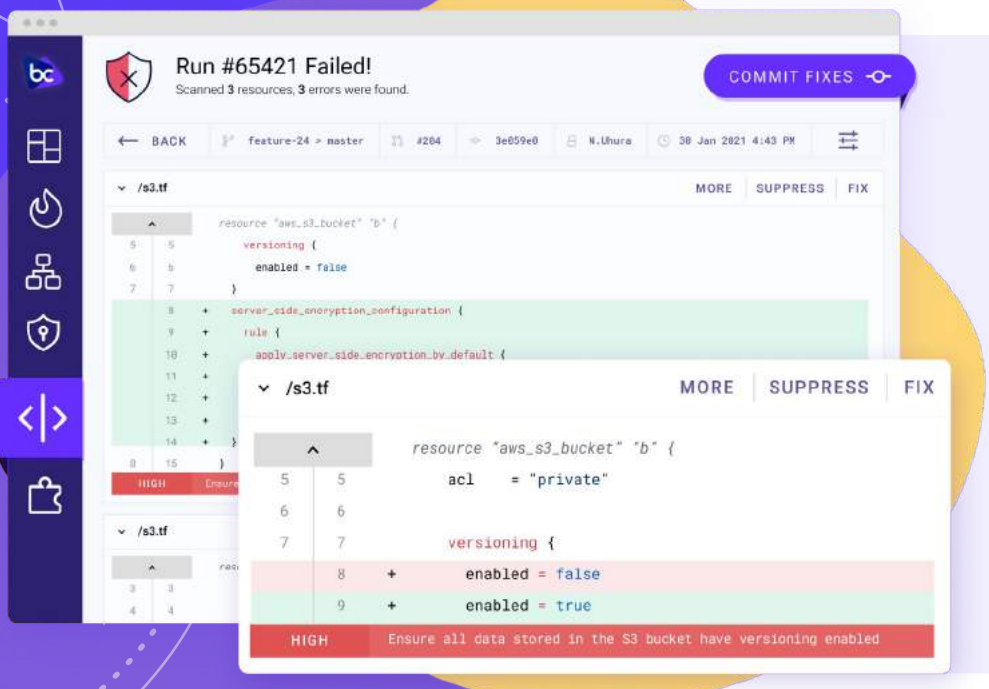
Infrastructure as code makes deploying and managing infrastructure more efficient and is crucial to enabling cloud DevSecOps. Although it brings its own set of challenges, IaC can be leveraged to secure your infrastructure.

By embedding IaC scanning and security-as-code fixes throughout the DevOps lifecycle, you can embrace a more modern approach to cloud security.

As is true with adopting any new technology, IaC can introduce new complexity and risk—especially when several frameworks are in use across teams. Because it can also run in parallel with manual cloud orchestration, implementing IaC without full adoption and visibility can lead to gaps as to how resources are provisioned and, most importantly—secured.

The benefits of IaC generally outweigh the costs, allowing for automation, scalability, and repeatability in cloud provisioning and management.

Here at Bridgecrew, we also see IaC as the key to bridging the gap between infrastructure engineering, DevOps, and security. By leveraging IaC's benefits, teams can automate cloud security and embed it into the DevOps lifecycle. Our codified cloud security platform enables such collaboration by integrating IaC scanning and security-as-code fixes with the tools and workflows developers already use.




**Get started with Bridgecrew to:**

- Find and fix misconfigurations in cloud resources and IaC.
- Enforce hundreds of built-in policies across security and compliance benchmarks.
- Embed guardrails via IDE plugins, pre-commit hooks, and native VCS and CI/CD integrations.

GET STARTED FOR FREE