bridgecrew
BY PRISMA® CLOUD

State of Open Source Terraform Security

We analyzed the open source Terraform Registry to gauge its current security and compliance posture.

Introduction

Infrastructure-as-code adoption is soaring as more and more practitioners experience its benefits:

- It leverages automation and scalability to make compute resource provisioning simpler than ever.
- It minimizes human error by making infrastructure deployment consistent and predictable.
- It codifies institutional knowledge that reduces future risk.
- Most importantly, all of those benefits add up to save time and resources for teams.

Infrastructure-as-code, sometimes referred to as infrastructure code or abbreviated as IaC, is used to automate infrastructure deployment, scaling, and management through the use of machine-readable configuration files.

Learn more about IaC >

As with any emerging technology, however, the power of IaC comes with its own "supply chain" challenges. Because IaC is built first and foremost to provision functional infrastructure resources, that oftentimes means that security isn't prioritized as much as it should be. Additionally, because it's a relatively new technology and methodology, most code-level static analysis and traditional cloud security tools don't support infrastructure-as-code, leaving gaps in coverage and unmitigated risk.

At Bridgecrew, it's our mission to help teams close that gap, and the first step in solving a problem is knowing you have one. That's why we're publishing this research, and why we hope you're reading it.

This report addresses the current state of IaC security and compliance, starting with Terraform. In it, we cover:

- The overall compliance of public Terraform Registry modules used to build resources for AWS, Azure, and Google Cloud
- The most common misconfigurations categories, why they're common, and how to prevent them
- Trends in the most popularly downloaded modules
- Recommendations for approaching IaC security

Terraform is a popular open-source declarative infrastructure-as-code framework used primarily to define resources in public cloud services. HashiCorp is the company behind Terraform along with commercial versions of it. In this report, we'll be referring specifically to the open-source framework.

Learn about Terraform >

We hope this research and guidance gives you insight into the state of IaC security as a whole and into your own IaC security posture.

About the data

The data in this report was collected by scanning the public Terraform Registry using Checkov, our open source infrastructure-as-code static analysis tool.

DATA SOURCE

In this research, we analyzed modules within the public and open source **Terraform Registry**.

Terraform modules come pre-built with standardized configurations and variable templates, which we have cross-checked against common industry standards. All modules are public under common open source licenses and can be found in other variations in downstream projects utilizing them to build more complex environments.

This research is based on over 2.6K scanned Registry modules used to build resources in AWS, Azure, and Google Cloud.

Keep in mind that because the Terraform community is an active and growing community, the Registry is constantly growing, and modules are constantly being updated. The data in this report is up-to-date as of June 15, 2020, 4 am PDT.

Maintained by HashiCorp, cloud providers, and community contributors, the **Terraform Registry** is made up of over 3K modules. Modules are self-contained packages of Terraform configurations that are managed as a group.

Check out the Registry >

DATA ANALYSIS

To scan the Registry, we utilized our open-source infrastructure-as-code scanning tool **Checkov**.

Checkov has over 300 compliance and security checks across AWS, Azure, and Google Cloud.

Although difficult to create a universal benchmark of industry security standards to cover all the major cloud providers, we leverage cloud security policies as defined by the Center of Internet Security (CIS), AWS Foundations, SOC2, PCI, and additional best practices to create Checkov's checks.

Checkov is Bridgecrew's open source scanning tool for infrastructure-as-code. Released in December 2019, Checkov now has over 600K downloads and supports scanning for Terraform, CloudFormation, Kubernetes, Azure Resource Manager, and Serverless frameworks.

Learn more about Checkov >

Key findings

After analyzing 2.6K Terraform Registry modules used to build resources on AWS, Azure, and Google Cloud, this is what we found:

1. Nearly 1 in 2 modules within the Terraform Registry is misconfigured.

2. Q2 2020 saw the biggest quarter-over-quarter growth, as well as the highest ratio of misconfigured to compliant modules yet.

3. Since 2017, misconfigured modules have been downloaded over 15M times. 8 out of the 10 most popular modules are misconfigured.

4. Modules used to provision AWS resources make up the vast majority of the Terraform Registry and are the most likely to be misconfigured.

5. The most common misconfigurations are in the Backup and Recovery, Logging, and Encryption categories.

Terraform module compliance

Terraform has become one of the leading provisioning frameworks for multi-cloud development. There has been a steady growth of modules both from the Terraform team and from community contributors.

As you can see in **Figure 1**, there has been a dramatic rise in module contributions within the Terraform Registry since 2019—especially in **Q2 2020**. Unfortunately, as you can see in **Figure 2**, many of those contributions contain misconfigurations.

Q2 2020 growth within the Terraform Registry seemingly coincides with the global COVID-19 pandemic. This finding is consistent with several accounts that **development work has seen an uptick** since the global lockdown.

Figure 1: Monthly Terraform Registry growth since 2017

The Terraform Registry has seen, on average, 35% quarter-over-quarter growth. Q2 2020 saw the biggest jump in contributions with 83% quarter-over-quarter growth.

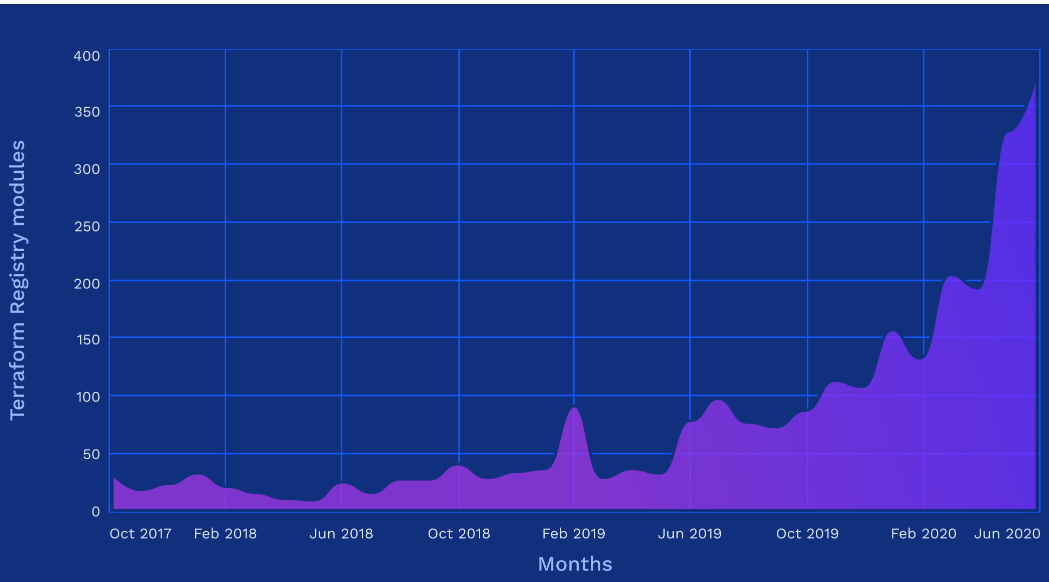
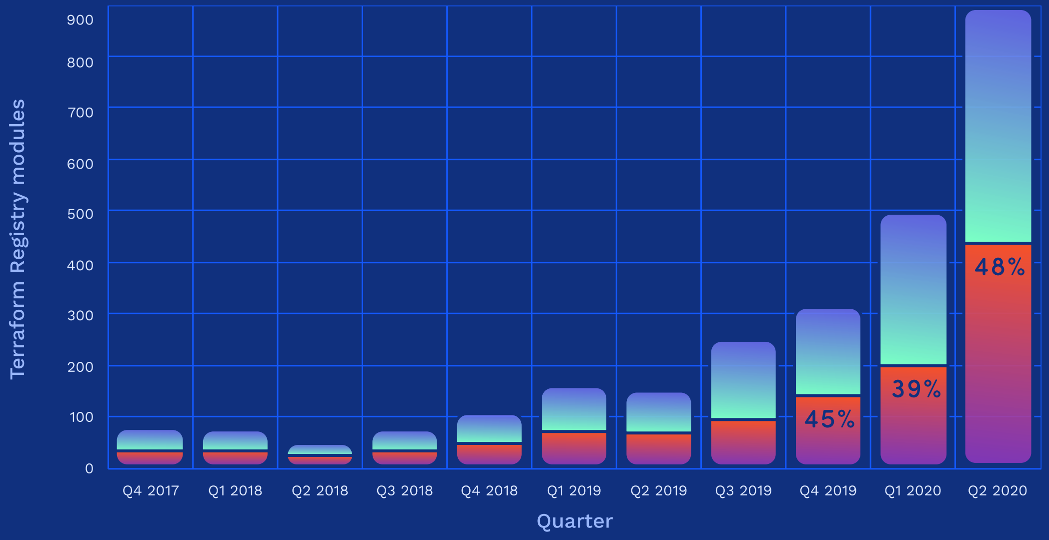


Figure 2: Quarterly Terraform Registry growth by compliance

The ratio of misconfigured modules to compliant modules has fluctuated between 38% and 45%. Q2 2020 saw a slight increase, with 48% containing misconfigurations.



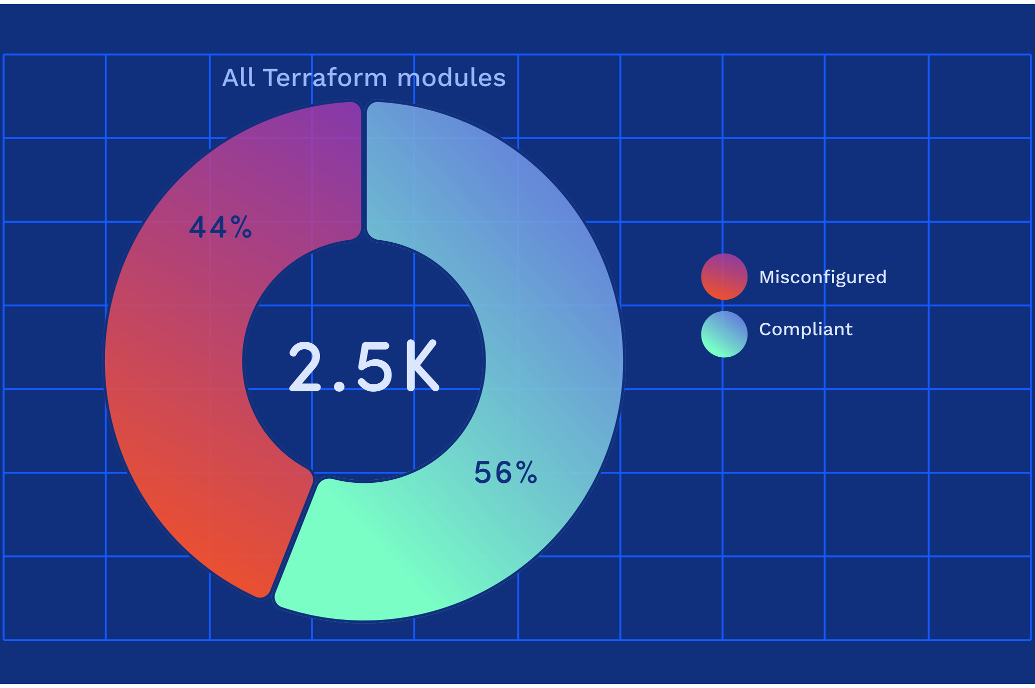
Terraform module compliance (continued)

To get a sense of the high-level state of Terraform compliance and security, we scanned the data set for known misconfigurations and risk.

Figure 3: Terraform Registry Modules by compliance

Out of all the Terraform modules used to build resources in AWS, Azure, and Google Cloud, 44% contain at least one misconfiguration.

Note: Modules that do not have a compliant or non-compliant state are not included in this figure.



The fact that nearly one in two modules in the Terraform Registry contain misconfigurations confirms our hypothesis that infrastructure-as-code security is not yet a top priority. It also tells us a few things about the IaC and Terraform landscapes:

- **Terraform is thriving.** The community around Terraform draws individual contributors, as well as organized development groups to build and contribute modules. On the one hand, they increase the usability of Terraform as a cloud provisioning language, but on the other, they expose more end-users to modules that aren't built according to industry benchmarks.
- **Industry standards are improving.** Newly issued services include better requirements and definitions around proper usage. Cloud providers are investing more in educating the market on how to best use their services, and downstream module developers have to realize it's their responsibility to write new modules based on those standards.

Check category compliance

We found that the majority of **checks fail** because the modules' optional arguments for enhancing data security and traceability were not defined.

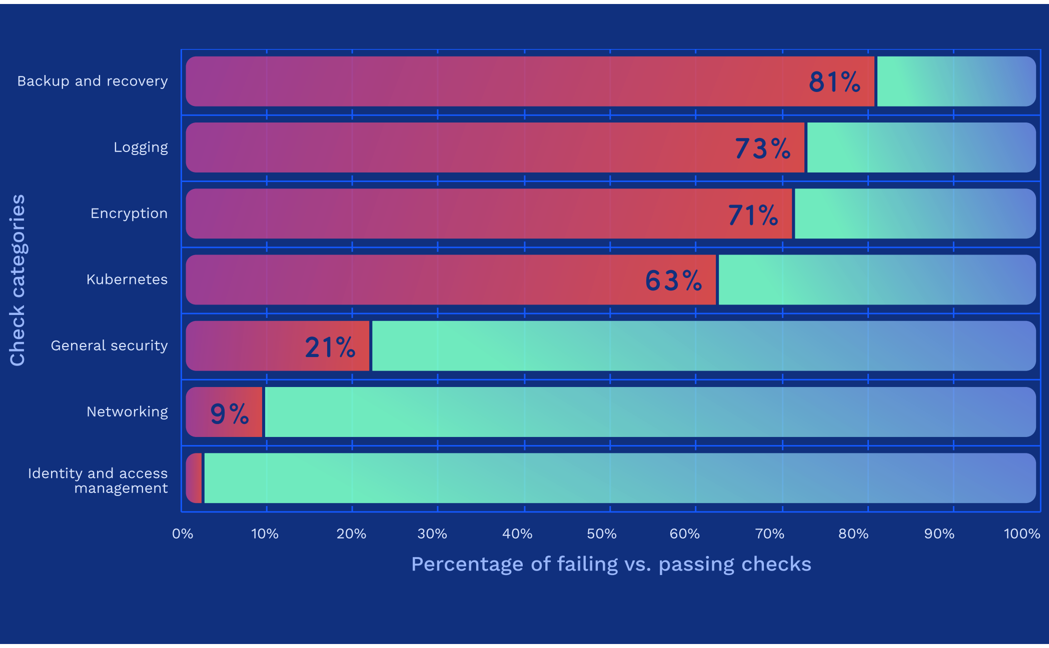
In addition to looking at the nature of how checks fail, we looked at why they failed. In **Figure 4**, we grouped checks by category, which refers to the broad classifications defined by CIS. These categories are intended to be applicable across cloud providers.

To gather this data, we analyzed the resources within each Terraform Registry module against hundreds of predefined checks within Checkov. **Checks fail** for a variety of reasons and fall into a handful of categories.

See all Checkov checks >

Figure 4: Categories by failing vs. passing checks

The three most common failing check categories are Backup and Recovery (81%), Audit Logging (73%), and Encryption (71%).



LOGGING

Modules that lack defined **logging** options can have negative governance consequences and can hinder investigations of suspected security incidents. Granular auditing, offered by most popular services, is a great way to ensure that fine-grained activities—such as S3 access logs—are stored for forensic and governance purposes.

We speculate that logging checks often fail because it's assumed that centralized logging is defined on the account level rather than in individual provisioned resources. We've also seen a lack of awareness in the ability to define logging at the infrastructure code level. Alternatively, because storing logs is an additional cost, teams may intentionally avoid using full audit logging capabilities.

Logging misconfigurations are modules with the ability to define stricter audit logging settings but do not use that configuration by default.

Check category compliance (continued)

BACKUP AND RECOVERY

Using standard **backup and recovery** protocols is the first line of defense against any disruption of service. Specifically, for data-related workloads, the addition of near-term backup options ensured that systems are promptly restored to their functional states.

As with logging-related misconfigurations, similar suggestions around centralized management and cost savings could explain why these are not utilized by default by the public modules.

Backup and recovery misconfigurations are modules that are missing required parameters that utilize default backup procedures.

ENCRYPTION

Defining either **encryption** at rest or in transit is required by foundational best practices requirements.

Applying native encryption helps ensure that data exposed publicly is not accessible without a decryption mechanism. For organizations managing multiple data storage technologies with public interfaces, encryption is the optimal method to protect data against unintended external access. With embedded encryption costs drastically reduced over the past few years, it remains a question as to why most modules do not include this configuration by default.

Encryption-related misconfigurations are modules that do not, by default, enable encryption at rest or in transit.

ADDITIONAL OBSERVATIONS

- Across the board, Identity and Access Management (IAM) is mostly properly configured across public modules. This may be because IAM misconfiguration detection in build-time is quite nuanced and complex. There are a few basic checks to ensure that MFA is enabled, and password policies are met, but IAM is an area that is notoriously hard to automate and gauge in build-time.
- Networking is also a mostly compliant category with just 22% of checks failing. We were happy to see strong compliance in the Networking category as these misconfigurations can potentially lead to more risk than other categories. Networking or compute instances exposed on the Internet can be hacked by malicious actors that can attempt to use them to infiltrate into privately hosted cloud networks and, from there, obtain access to sensitive data.
- General Security, which contains universal configurations usually tied up to the account level, is also fairly secure, with just 25% of checks failing.

Compliance across cloud providers

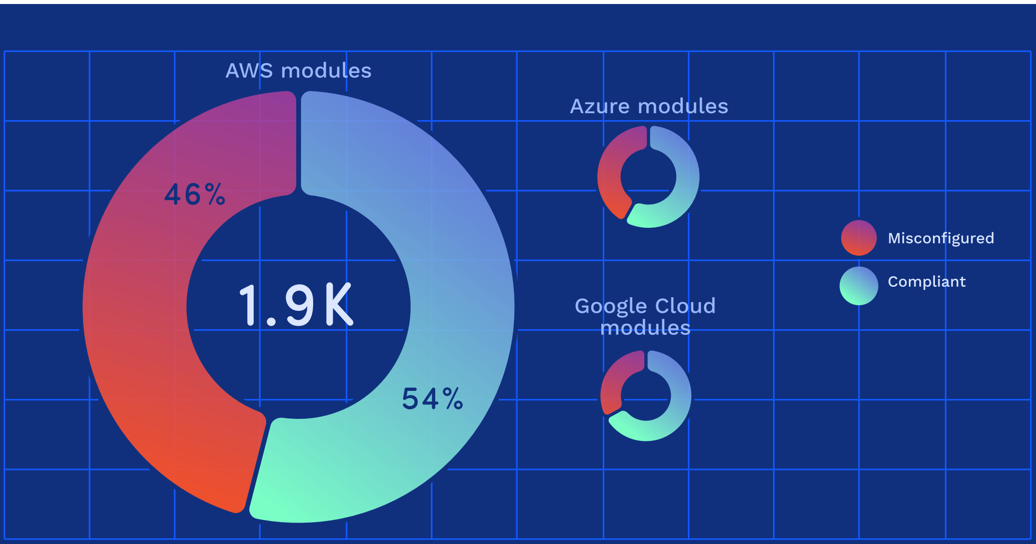
The Terraform Registry is made up of modules that build cloud resources for specific cloud providers. Checkov is also equipped with **provider-specific checks**.

To add provider-specific context to our findings, we analyzed modules across the three major cloud providers—AWS, Azure, and Google Cloud—for overall compliance and interesting findings in specific categories.

Keep in mind that while we've devoted considerable **provider-specific checks**, AWS was our first cloud provider and may have slightly more coverage. With that in mind, let's look at the percentage of configurations across categories and cloud providers.

Figure 5: Module volume by cloud providers and compliance

AWS modules make up 76% of the Terraform Registry. AWS has the highest ratio of misconfigured to compliant modules (46%) while Google Cloud has the lowest (33%).



As you can tell by **Figure 5**, AWS modules make up the vast majority of the Registry, which is not entirely surprising. AWS is the biggest cloud provider leading in global coverage, product portfolio, and user base. Additionally, the Terraform community has embraced AWS and its services since the beginning, while Azure and Google Cloud are starting to make gains.

Check categories across cloud providers

Best practices and configuration settings vary slightly from one provider to another, so looking at categories across them all doesn't represent the nuanced cloud security landscape. To narrow in on misconfigured categories within each cloud provider, we started by looking at the most prevalent check categories across providers, as shown in **Figure 6**.

Next, we looked at the misconfiguration ratio for each in **Figure 7** and, based on those findings, dug into misconfiguration categories specifically relevant to each cloud provider.

Figure 6: Distribution of check categories for each cloud provider

The most frequently occurring check category for AWS is IAM. For Azure and Google Cloud, the largest check category is Networking.

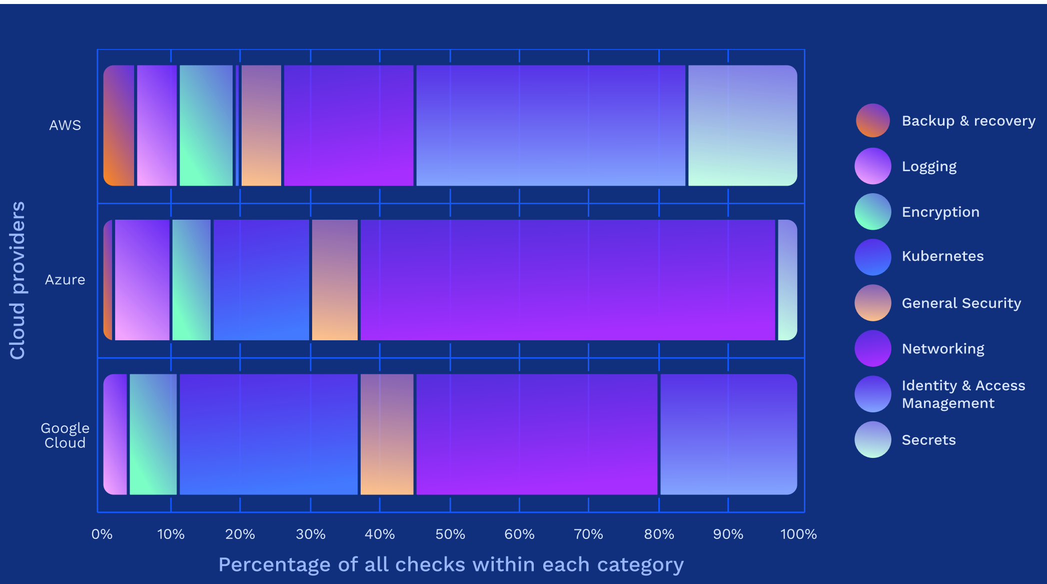
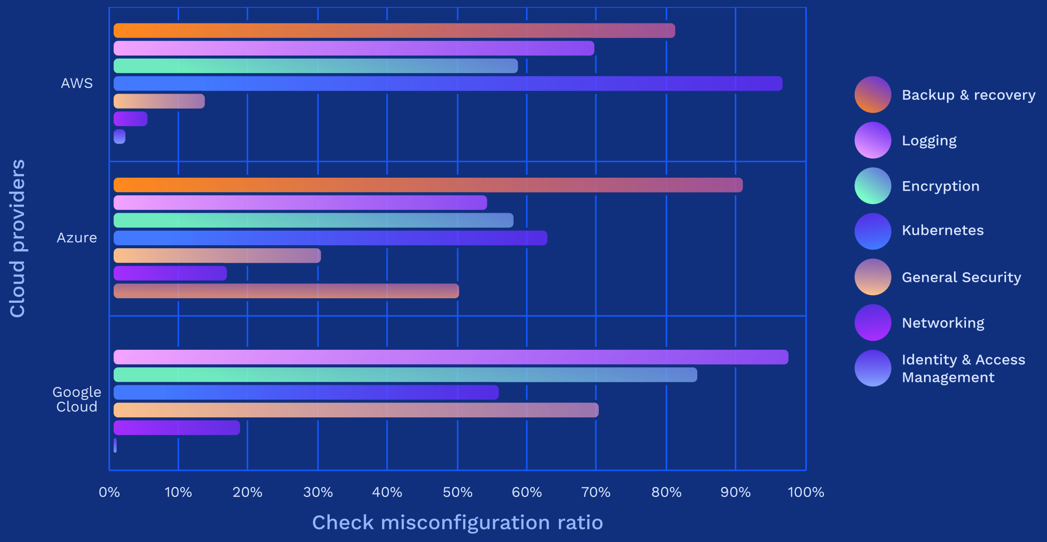


Figure 7: Misconfiguration ratio by category for each cloud provider

The most frequently failing checks are Logging within Google Cloud modules and Kubernetes within AWS modules. The most frequently passing checks are IAM within AWS and Google Cloud modules.

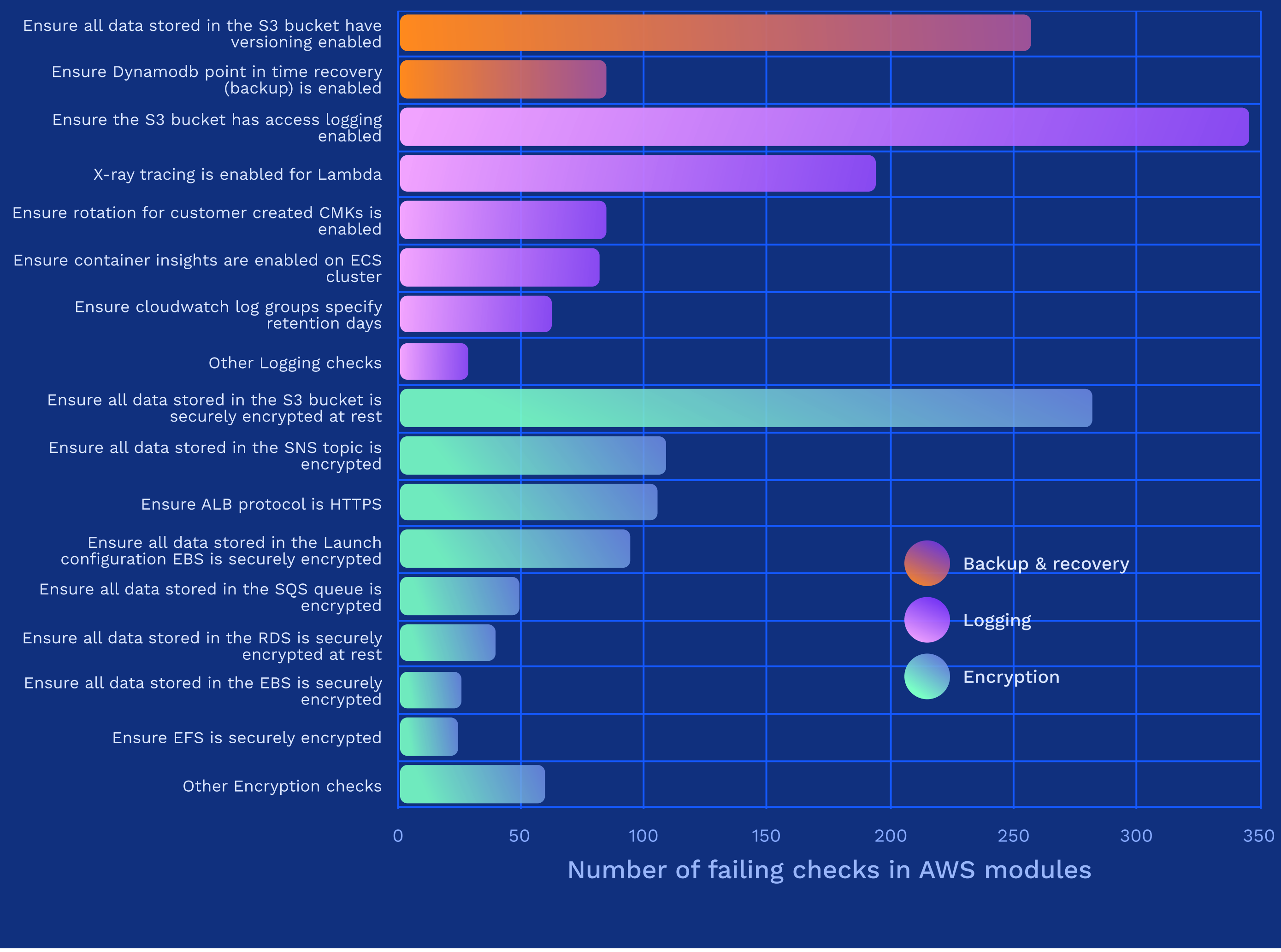


Check categories across cloud providers (continued)

AWS

For AWS modules, over half of the checks in Backup and recovery, Logging, and Encryption categories failed. The main reason these checks fail so commonly is that they are among the most common optional fields in most Terraform modules. We analyzed these categories in **Figure 8** to see if anything interesting popped out. The top three most common failing checks in these categories within AWS modules have to do with S3 bucket logging, versioning, and encryption.

Figure 8: Top failing logging, backup, and encryption checks found in AWS modules



Check categories across cloud providers (continued)

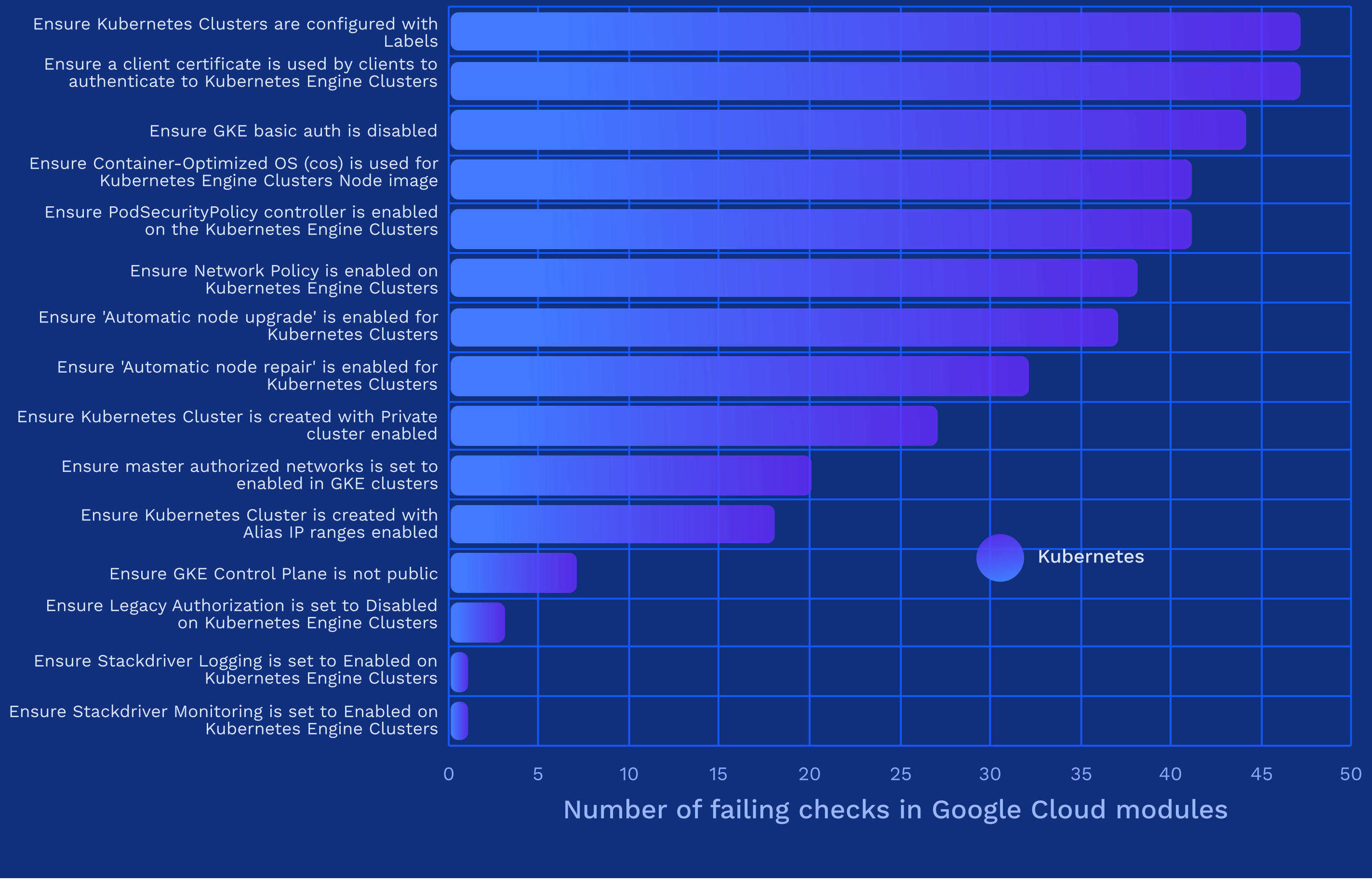
GOOGLE CLOUD

Across the board, Google Cloud modules are the most compliant. IAM checks were almost entirely compliant, with nearly all 500+ passing. Across almost every other category, there's room for improvement. Most notably, nearly all 100+ Logging checks failed, and over 75% of the ~200 Encryption checks failed.

We were interested in taking a look at Kubernetes checks within Google Cloud, as there has been a concerted effort to ensure GKE is preconfigured with CIS benchmarks in mind.

Despite that effort, we found many recurring gaps within the 700+ checks. As you can see in **Figure 9**, most of the failing checks are related to missing authentication hardening and lack of Network Policy and PodSecurityPolicy controllers.

Figure 9: Top failing Kubernetes checks found in Google Cloud modules



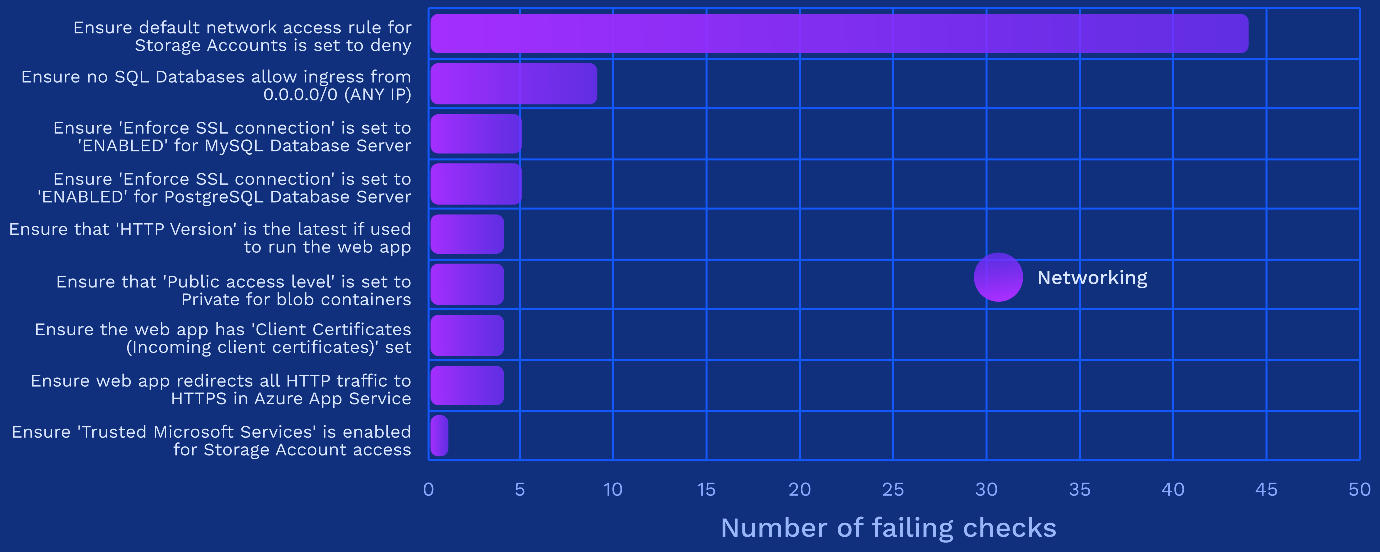
Check categories across cloud providers (continued)

AZURE

For five of the seven categories across Azure modules, over 50% of checks failed. However, it's important to note that the sample size of Azure modules is much smaller than the other providers—the only categories with 100+ checks are Kubernetes and Networking.

That said, we found some interesting findings when looking at failing checks within the ~500 Networking checks. As you can see in **Figure 10**, most Networking checks failed primarily due to a lack of port restrictions in the various database modules Azure has to offer. Restricting port access to those absolutely required by those databases would drastically improve these modules from a security and compliance standpoint.

Figure 10: Top failing Networking checks found in Azure modules



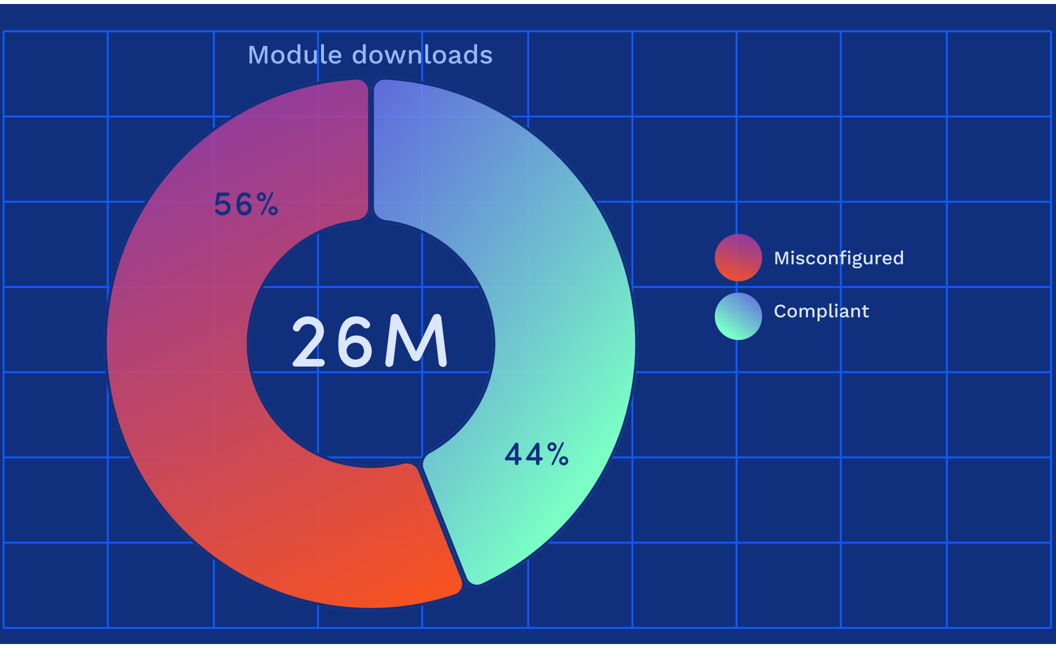
Compliance of Terraform modules in use

Examining Registry modules in isolation gives us a solid baseline for the overall state of Terraform security, but it doesn't capture the real-life implications for developers and teams using these modules in their own applications every day.

To better understand the potential impact of misconfigurations, we extrapolated our previous findings across download data and analyzed the most popular downloaded modules for trends.

Figure 11: Misconfiguration ratio of module downloads

When looking at Registry module downloads, we found that 56%—which amounts to over 15M downloads—contained misconfigured resources.



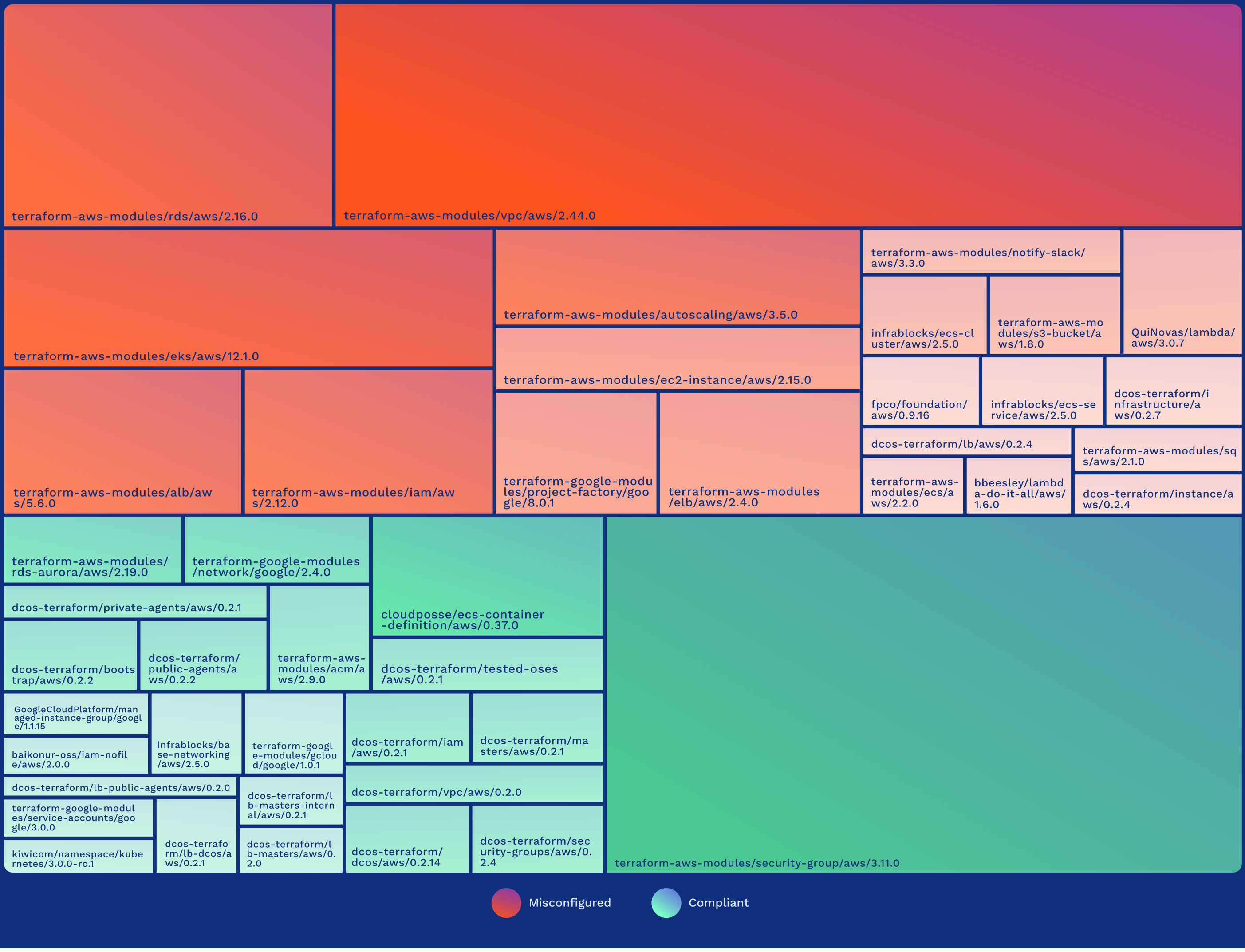
The fact that the misconfiguration ratio for downloaded modules is higher than that of modules overall suggests that many of the more popular modules contain misconfigured resources.

At this point, we should be reminded that many of these misconfigurations do not inherently represent risk. Misconfigurations' eventual impact on organizations' overall cloud security or compliance posture is affected by various factors, including the nature of resources and their intended use, as well as additional protective layers put in place at the cloud provider level.

Compliance of Terraform modules in use (continued)

Lastly, we analyzed the most popularly downloaded modules within the Registry by compliance.

Figure 12: Terraform modules with over 100K downloads by compliance



Our hypothesis that misconfigured modules are in higher circulation is confirmed by **Figure 12**. Out of just the top 10 most downloaded modules, 8 contain misconfigurations. Analyzing the most popularly downloaded modules revealed a few interesting findings:

- The most downloaded configured modules include popular networking modules in AWS and Google Cloud. Among them, compliant and hardened modules for Security Groups and VPC settings were prevalent.
- The most downloaded misconfigured modules include some very popular and widely used services such as AWS RDS, AWS EKS, AWS ALB, and AWS IAM.

Recommendations

At Bridgecrew, we believe that maintaining compliance as early as possible not only hardens your infrastructure but is also a much more cost-effective method to protect against risk. By defining configurations upstream, you can ensure configurations are prevented as part of every code review and avoid relying on other teams to fix them downstream.

Here are a few ways you can start addressing cloud misconfigurations before they become risks:

ENFORCE CONSISTENT POLICIES IN FAMILIAR WORKSPACES

There are multiple techniques to prevent misconfigurations from ever getting to production.

- For individual contributors, start by introducing a pre-commit hook scanner or an IaC linter to capture misconfigurations during routine code authoring.
- For smaller teams, a lightweight CI/CD job can catch most common misconfigurations and revert them to their code owner.
- For larger teams, we recommend looking at developing a service catalog model and pre-approve specific modules after testing them against static analysis.

FIX MISCONFIGURATIONS FAST AND EARLY

Like vulnerabilities, infrastructure misconfigurations can quickly pile-up and create unworkable backlogs. Performing binary, non-intrusive code fixes during build-time can help ensure you're meeting compliance standards.

- Start by enforcing consistent backup and log auditing policies that can be added using a single argument in most frameworks.
- For more complex problems such as network or IAM rightsizing, take a holistic approach to ensure complex logic is always managed in IaC and goes through standard code review processes.
- To improve data security posture by applying end-to-end encryption, we recommend migrating existing non-IaC workloads into pre-vetted modules that always explicitly define encryption at rest and in transit.

IMPROVE "DAY-TWO" VISIBILITY INTO CONFIGURATION CHANGES

Infrastructure provisioning is an ongoing process. With day-one activities usually revolving around getting configurations planned and applied as expected, day-two is a continuous effort to ensure changes are properly logged and versioned in CI/CD automation workflows. Having eyes and ears from commit, through build, and all the way to deployment makes it much easier to trace errors and revert back to known good states.

Conclusion

Cloud-native technology is enabling developers to innovate and build faster than ever before. At Bridgecrew, we'd like to see every developer empowered with the knowledge, skills, and tooling to be able to ship secure products into the cloud. Bridgecrew was founded as part of a community effort to share collective knowledge and make security more accessible to developers. We are hopeful that with this information in hand, developers and practitioners will be empowered to play a bigger part in securing the cloud infrastructure they build.

Bridgecrew provides teams both big and small, with an all-in-one platform to find, fix, and prevent cloud security issues and violations. With support for AWS, Azure, Google Cloud, Terraform, CloudFormation, Kubernetes, and Serverless, Bridgecrew is transforming the way teams approach cloud security by automating, codifying, and streamlining it.

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