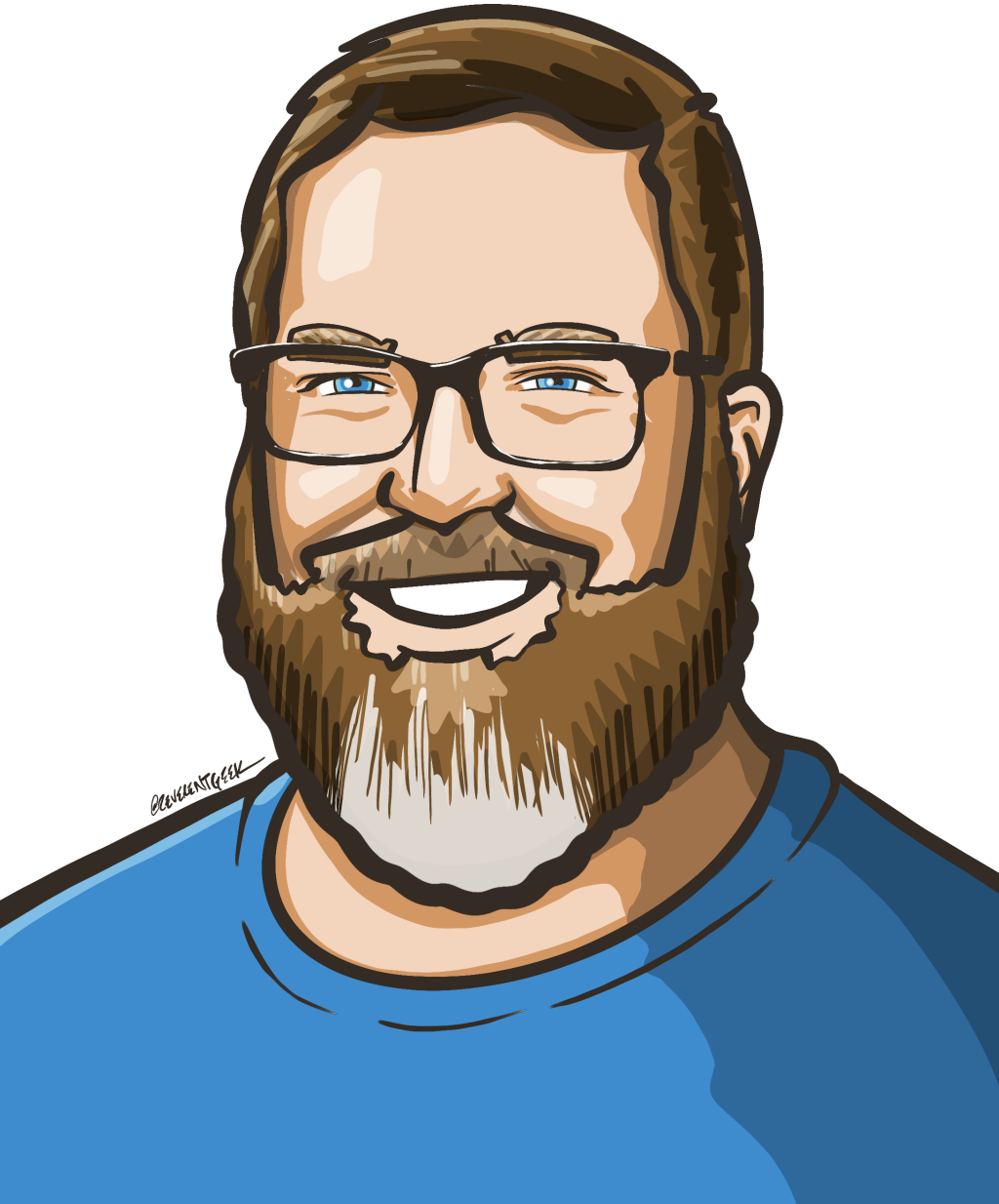


Securely Deploying Infrastructure As Code

Chris Ayers





Chris Ayers

Principal Software Engineer
Azure EngOps AzRel
Microsoft

 BlueSky: [@chris-ayers.com](https://chris-ayers.com)

 LinkedIn: - [chris-l-ayers](#)

 Blog: <https://chris-ayers.com/>

 GitHub: [Codebytes](#)

 Mastodon: [@Chrisayers@hachyderm.io](https://hachyderm.io/@Chrisayers)

 ~~Twitter: @Chris_L_Ayers~~

<https://chris-ayers.com>

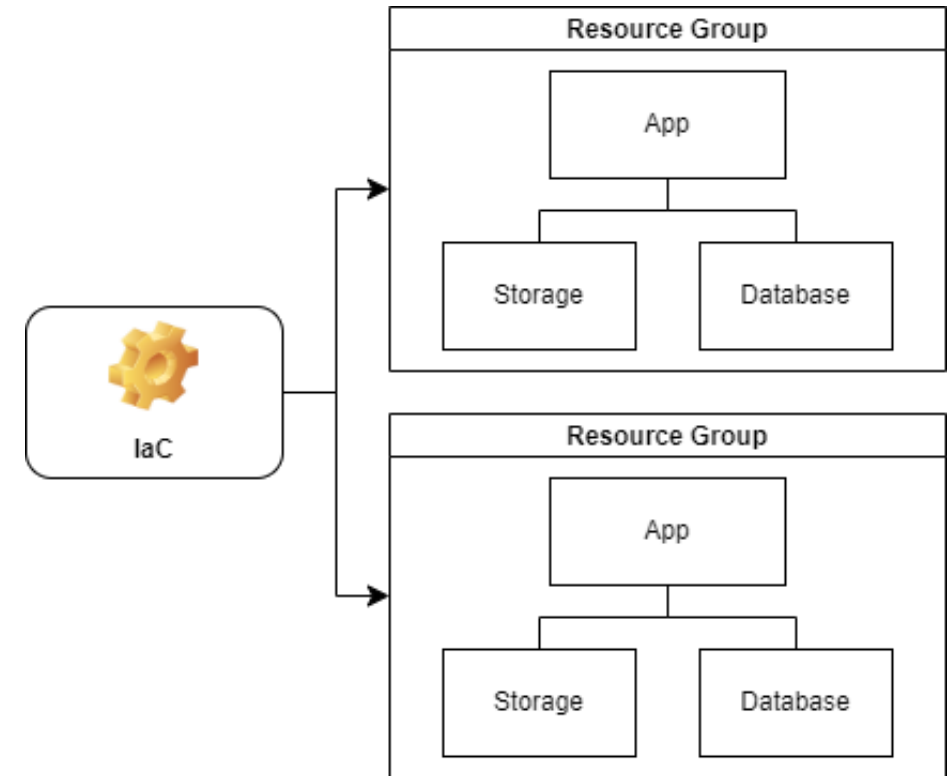


Agenda

- **What is Infrastructure as Code (IaC)?**
- **Security Tooling**
 - Rules & Customization
 - Workflow
- **Integration**
 - Precommit hook
 - VSCode Integration
 - GitHub Actions

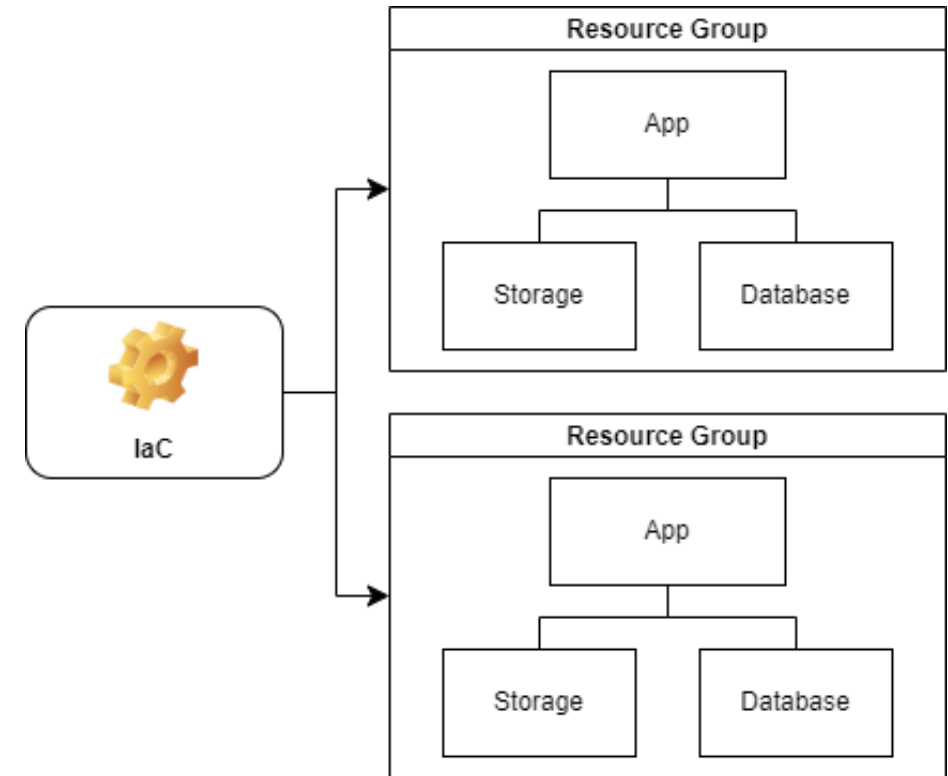
What is Infrastructure as Code (IaC)?

Infrastructure as code (IaC) is a way to manage and provision infrastructure resources using configuration files and automation tools.



Why use Infrastructure as Code (IaC)?

The goal of IaC is to make it easier to deploy and manage infrastructure in a repeatable, reliable way, and to reduce the risk of errors caused by manual configuration.



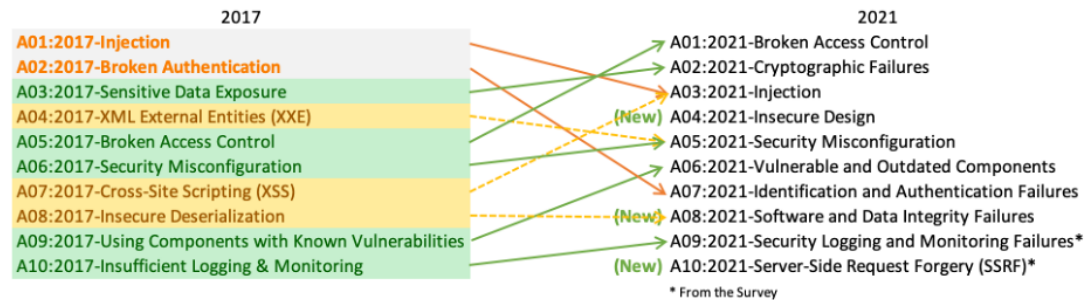


Security and Compliance

There have been multiple breaches and attacks due to misconfiguration. Vulnerabilities can be a simple omitted property.

OWASP Top 10

A05:2021 – Security Misconfiguration



Kubernetes OWASP Top Ten

OWASP Kubernetes Top Ten

[Main](#)

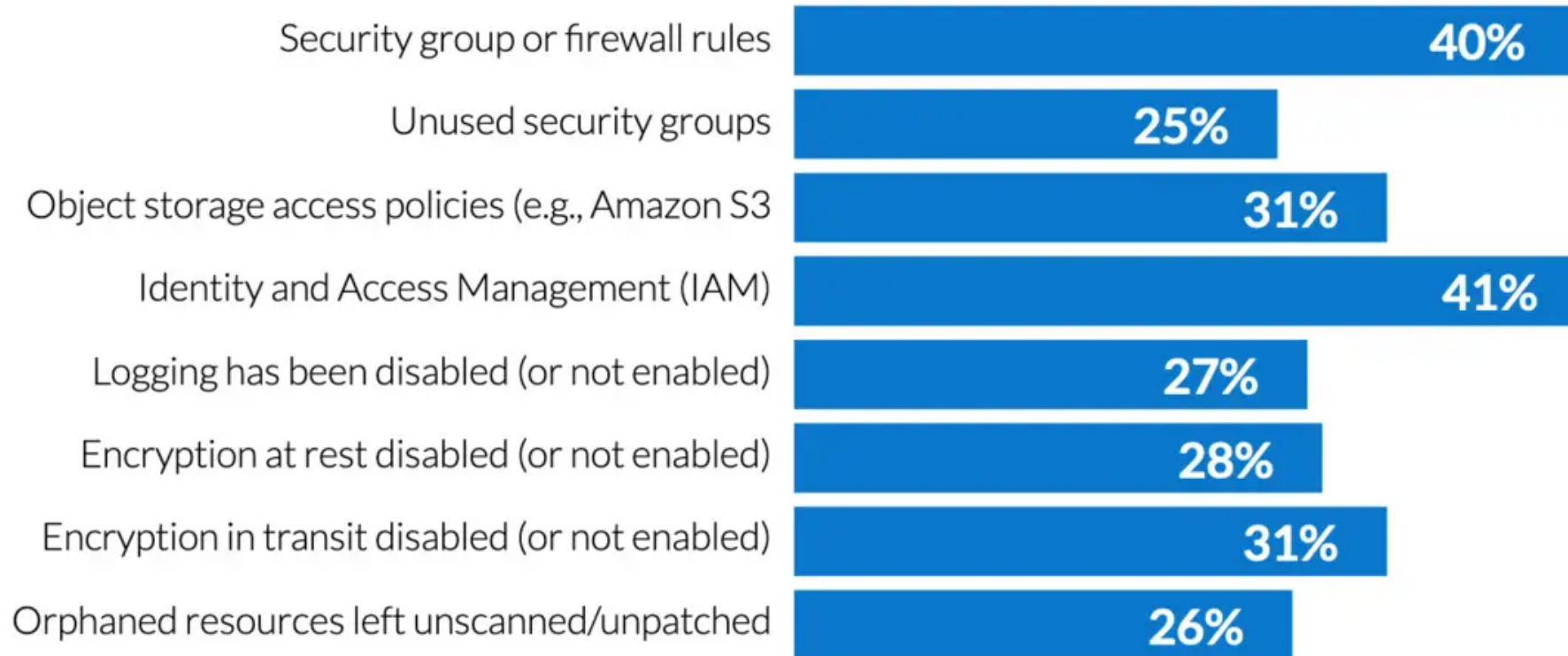
About the Kubernetes Top 10

When adopting [Kubernetes](#), we introduce new risks to our applications and infrastructure. The *OWASP Kubernetes Top 10* is aimed at helping security practitioners, system administrators, and software developers prioritize risks around the Kubernetes ecosystem. The Top Ten is a prioritized list of these risks backed by data collected from organizations varying in maturity and complexity.

- K00:2022 [Welcome to the Kubernetes Security Top Ten](#)
- K01:2022 [Insecure Workload Configurations](#)
- K02:2022 [Supply Chain Vulnerabilities](#)
- K03:2022 [Overly Permissive RBAC Configurations](#)
- K04:2022 [Lack of Centralized Policy Enforcement](#)
- K05:2022 [Inadequate Logging and Monitoring](#)
- K06:2022 [Broken Authentication Mechanisms](#)
- K07:2022 [Missing Network Segmentation Controls](#)
- K08:2022 [Secrets Management Failures](#)
- K09:2022 [Misconfigured Cluster Components](#)
- K10:2022 [Outdated and Vulnerable Kubernetes Components](#)

Cloud Misconfiguration

Common cloud misconfigurations experienced



Source: The State of Cloud Security 2021 Report

Shift Left on Security

Save time and money

We can't just do security in production after everything is built, we need to go into solutions with security baked in.

Security Tooling

Running security testing tools against infrastructure as code (IaC) is a way to ensure that the infrastructure being provisioned is secure and compliant with best practices.

Why run Security Tooling?

-  To catch security issues early
-  To ensure compliance
-  To improve the security of your infrastructure
-  To save time and effort by shifting Left

Security Tooling: SAST vs DAST

Static Application Security Testing (**SAST**)

SAST is a method of debugging that is performed by looking at the source code of an application without actually executing the application.

Dynamic Application Security Testing (**DAST**)

DAST, on the other hand, involves testing an application while it is running, in order to identify potential security vulnerabilities.

Security Tooling - Terraform

Each of these tools does similar things and are SAST (Static Analysis Security Tooling).
With Terraform you can analyze in a few ways.

- HCL files
- Terraform Plan

Security Tooling - OSS

There are many open-source tools as well as commercial solutions. We can integrate these tools in our local environments as well as our pipelines to secure things earlier.

Feature	tfsec	terrascan	checkov
CI/CD	Yes	Yes	Yes
Rules	100+	100+	100+
Custom Rules	Yes	Yes	Yes
Rule Language	json, yaml, rego	rego	python

Rule customization

- Ignoring rules
- Overriding rules
- Add custom rules



tfsec

tfsec is a static analysis security scanner for your Terraform code supported by Aquasecurity.

Designed to run locally and in your CI pipelines, developer-friendly output and fully documented checks.

- OPA/Rego Policies
- VS Code Extension
- GitHub Actions

Terrascan

Terrascan has support for Terraform, Azure, GCP, AWS, Kubernetes (manifests, Helm, Kustomize), Docker and even GitHub. Supported by Tenable and now integrated into nessus.

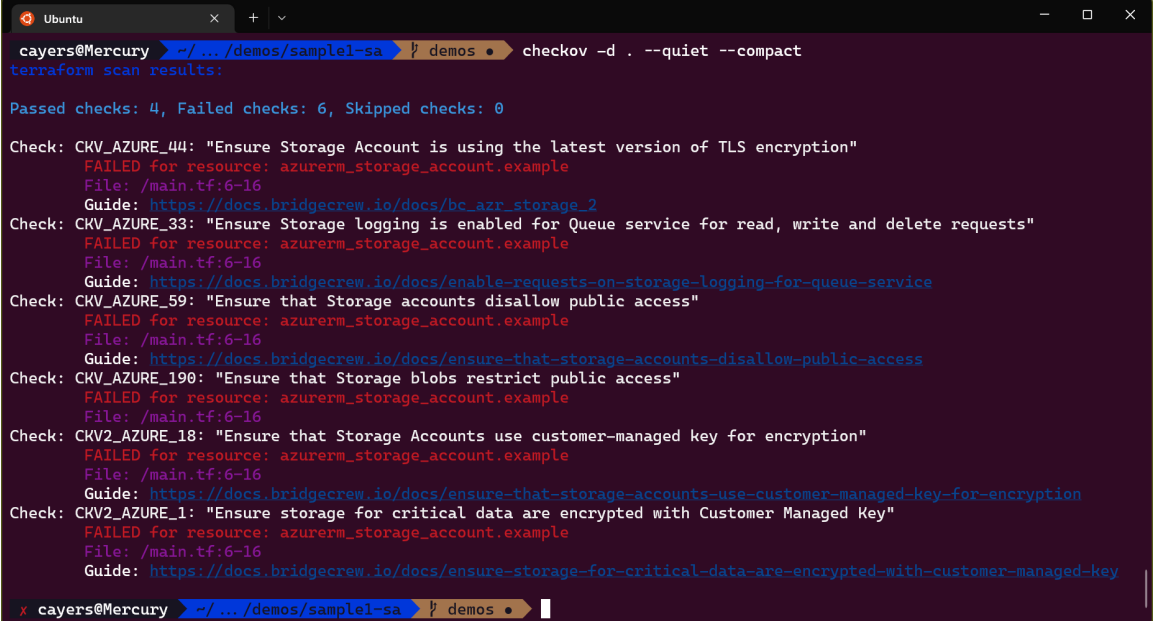
Terrascan has a large number of built in policies as well as support for custom OPA/Rego Policies.

```
Violation Details -
  Description : Enabling S3 versioning will enable easy recovery from both unintended user actions, like deletes and overwrites
  File       : modules/m4/modules/m4a/main.tf
  Line      : 20
  Severity   : HIGH
  Rule Name  : s3Versioning
  Rule ID    : AWS.S3Bucket.IAM.High.0370
  Resource Name : bucket4a
  Resource Type : aws_s3_bucket
  Category   : IAM
-----
Skipped Violations -
  Description : Enabling S3 versioning will enable easy recovery from both unintended user actions, like deletes and overwrites
  File       : modules/m1/main.tf
  Line      : 20
  Severity   : HIGH
  Skip Comment : Rule can be skipped
  Rule Name  : s3Versioning
  Rule ID    : AWS.S3Bucket.IAM.High.0370
  Resource Name : bucket
  Resource Type : aws_s3_bucket
  Category   : IAM
-----
Scan Summary -
  File/Folder : /Users/apple/go/src/github.com/patilpankaj212/terrascan/pkg/iac-providers/terraform/v14/testdata/deep-modules
  IaC Type    : terraform
  Scanned At  : 2021-01-19 17:21:53.503036 +0000 UTC
  Policies Validated : 552
  Violated Policies : 1
  Low             : 0
  Medium          : 0
  High            : 1
```

Checkov

Checkov is another tool that lets us do scanning and compliance.

Checkov is by BridgeCrew and python based. Checkov, like terrascan, supports Terraform, Azure, GCP, AWS, Kubernetes (manifests, Helm, Kustomize) and Docker.

A terminal window titled 'Ubuntu' showing the execution of the 'checkov' command. The command is 'checkov -d . --quiet --compact' run from the directory '/.../demos/sample1-sa'. The output shows 'terraform scan results:' followed by a summary: 'Passed checks: 4, Failed checks: 6, Skipped checks: 0'. Below this, six individual checks are listed, all of which failed. Each failure entry includes the check ID, a description, the resource it failed for ('azurerm_storage_account.example'), the file path ('/main.tf:6-16'), and a guide URL. The checks are: CKV_AZURE_44 (TLS encryption), CKV_AZURE_33 (Storage logging for Queue service), CKV_AZURE_59 (Storage accounts disallow public access), CKV_AZURE_190 (Storage blobs restrict public access), CKV2_AZURE_18 (Storage Accounts use customer-managed key for encryption), and CKV2_AZURE_1 (Storage for critical data are encrypted with Customer Managed Key).

```
cayers@Mercury > /.../demos/sample1-sa > demos • checkov -d . --quiet --compact
terraform scan results:

Passed checks: 4, Failed checks: 6, Skipped checks: 0

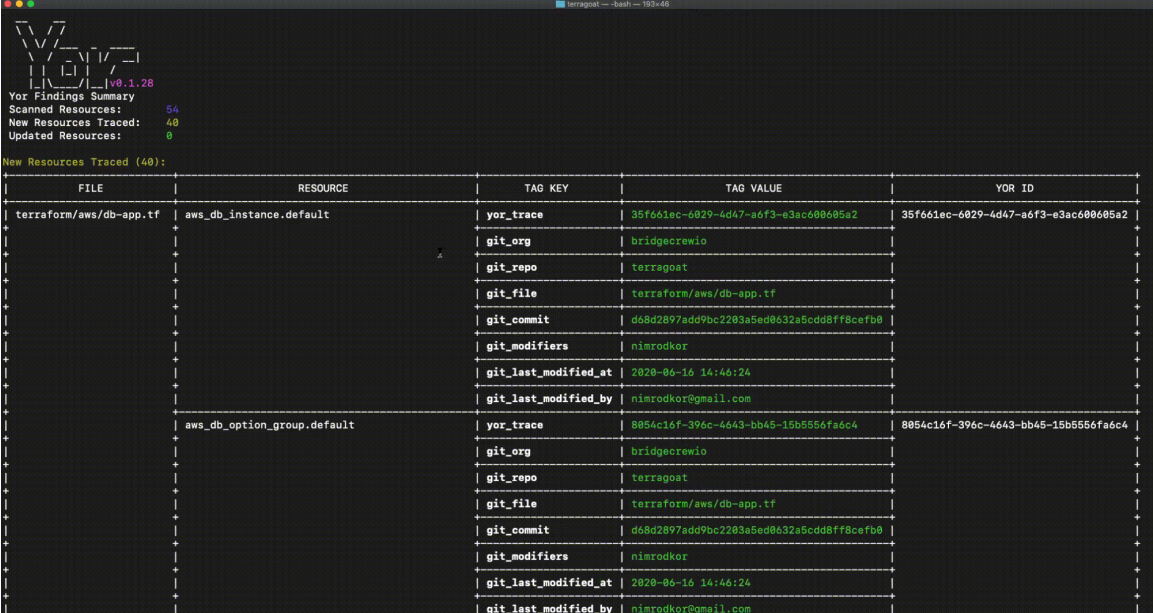
Check: CKV_AZURE_44: "Ensure Storage Account is using the latest version of TLS encryption"
  FAILED for resource: azurerm_storage_account.example
  File: /main.tf:6-16
  Guide: https://docs.bridgecrew.io/docs/bc_azr_storage_2
Check: CKV_AZURE_33: "Ensure Storage logging is enabled for Queue service for read, write and delete requests"
  FAILED for resource: azurerm_storage_account.example
  File: /main.tf:6-16
  Guide: https://docs.bridgecrew.io/docs/enable-requests-on-storage-logging-for-queue-service
Check: CKV_AZURE_59: "Ensure that Storage accounts disallow public access"
  FAILED for resource: azurerm_storage_account.example
  File: /main.tf:6-16
  Guide: https://docs.bridgecrew.io/docs/ensure-that-storage-accounts-disallow-public-access
Check: CKV_AZURE_190: "Ensure that Storage blobs restrict public access"
  FAILED for resource: azurerm_storage_account.example
  File: /main.tf:6-16
Check: CKV2_AZURE_18: "Ensure that Storage Accounts use customer-managed key for encryption"
  FAILED for resource: azurerm_storage_account.example
  File: /main.tf:6-16
  Guide: https://docs.bridgecrew.io/docs/ensure-that-storage-accounts-use-customer-managed-key-for-encryption
Check: CKV2_AZURE_1: "Ensure storage for critical data are encrypted with Customer Managed Key"
  FAILED for resource: azurerm_storage_account.example
  File: /main.tf:6-16
  Guide: https://docs.bridgecrew.io/docs/ensure-storage-for-critical-data-are-encrypted-with-customer-managed-key

x cayers@Mercury > /.../demos/sample1-sa > demos •
```

Bonus Tool - Yor

<https://yor.io/>

- **Tagging:** Automates tagging in IaC for better resource traceability.
- **Context:** Provides enriched data on resource deployment.
- **CI/CD Integration:** Facilitates automated tagging at scale.



The screenshot shows the Yor tool interface in a terminal window. At the top, there's a logo and version information (v0.1.28). Below that, a 'Yor Findings Summary' is displayed with the following statistics:

- Scanned Resources: 54
- New Resources Traced: 40
- Updated Resources: 0

Below the summary, a section titled 'New Resources Traced (40):' displays a table of traced resources. The table has five columns: FILE, RESOURCE, TAG KEY, TAG VALUE, and YOR ID. Two resources are shown as examples:

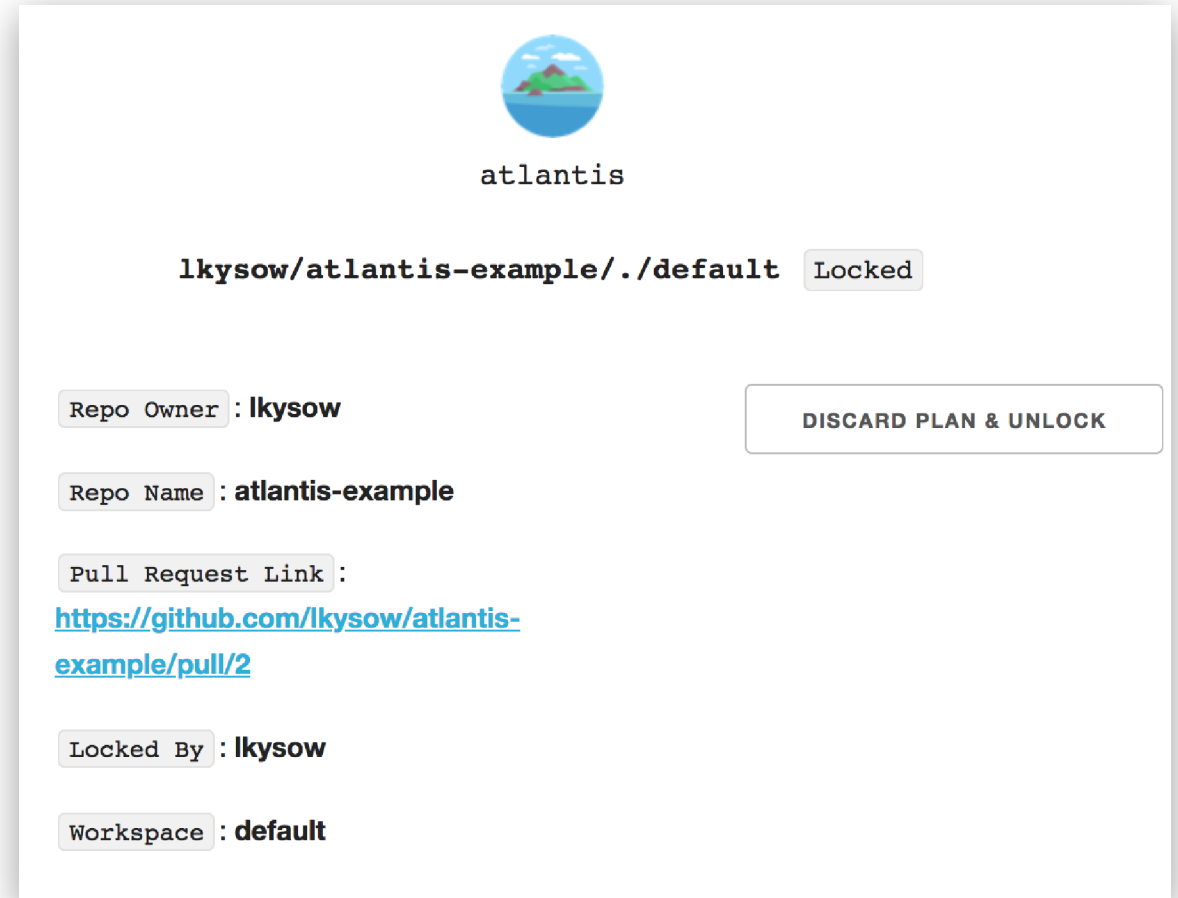
FILE	RESOURCE	TAG KEY	TAG VALUE	YOR ID
terraform/aws/db-app.tf	aws_db_instance.default	yor_trace	35f661ec-6029-4d47-a6f3-e3ac600605a2	35f661ec-6029-4d47-a6f3-e3ac600605a2
		git_org	bridgecrewio	
		git_repo	terrageat	
		git_file	terraform/aws/db-app.tf	
		git_commit	d68d2897add9bc2203a5ed0632a5cdd8ff8cefb8	
		git_modifiers	nimrodkor	
		git_last_modified_at	2020-06-16 14:46:24	
		git_last_modified_by	nimrodkor@gmail.com	
	aws_db_option_group.default	yor_trace	8054c16f-396c-4643-bb45-15b5556fa6c4	8054c16f-396c-4643-bb45-15b5556fa6c4
		git_org	bridgecrewio	
		git_repo	terrageat	
		git_file	terraform/aws/db-app.tf	
		git_commit	d68d2897add9bc2203a5ed0632a5cdd8ff8cefb8	
		git_modifiers	nimrodkor	
		git_last_modified_at	2020-06-16 14:46:24	
		git_last_modified_by	nimrodkor@gmail.com	

Bonus Tool - Atlantis

<https://www.runatlantis.io/>

Terraform PR Automation
Handles planning, locking, and
applying

<https://chris-ayers.com>



You Can Use Multiple Tools

Defense in Depth

Because these tools are independent and all scan the raw HCL or interpreted HCL, you can get different rules and potentially better compliance.

You can also hit a Signal to Noise problem.

Workflow Options

- Pre-Commit Hooks
- IDE Integration
- CI/CD Integration

Pre-Commit Hooks

Pre-commit Hooks run before code gets committed to a git repo.
You do it yourself or use the [Pre-Commit Framework](#).



IDE Integration

- Extensions for VSCode
 - [tfsec](#)
 - [checkov](#)
- DevContainer features
 - [tfsec](#)
 - [terrascan](#)
 - [checkov](#)

Pipeline integration

- GitHub Marketplace Extensions
 - [tfsec action](#)
 - [Run tfsec PR commenter](#)
 - [Terrascan IaC scanner](#)
 - [Checkov GitHub Action](#)

DEMOS

Backend providers

- [AzureRM](#)
- [Terraform on Azure Docs](#)
- Docs for other providers

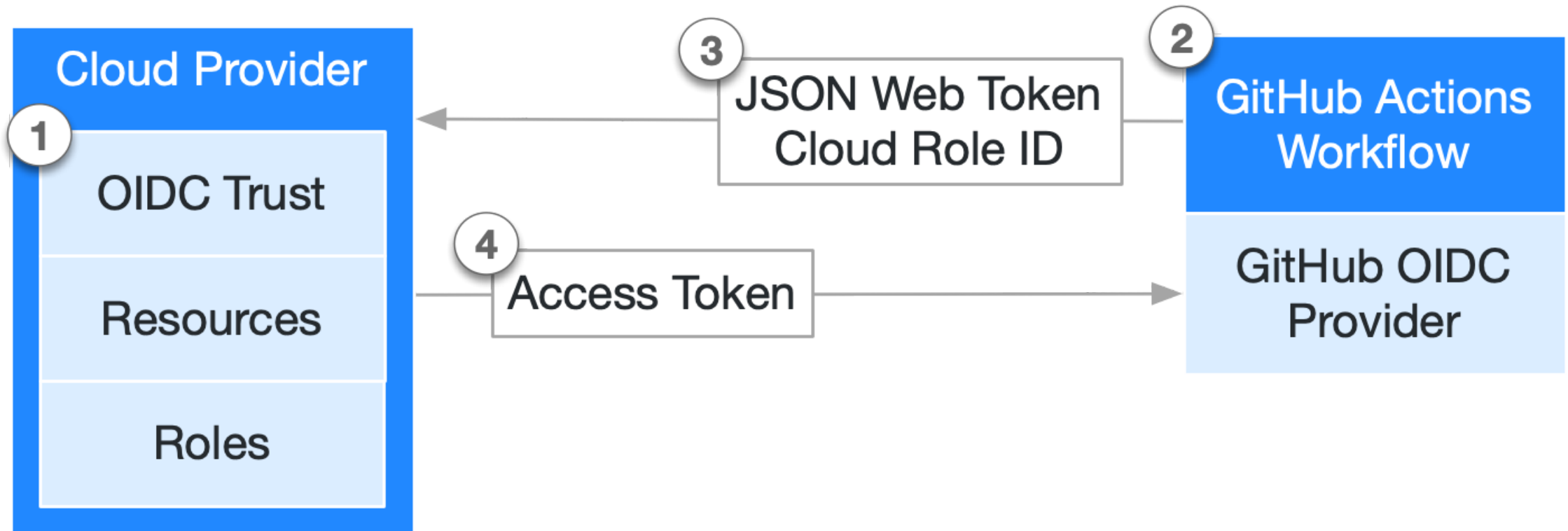
Overriding backend provider configuration

- Check docs
- Use environment vars

Open ID Connect (OIDC) Auth

No more passwords

Auth is claims based on repo, environment, branch..



Demos

Questions



Resources

GitHub Repo

<https://github.com/codebytes/secure-terraform-on-azure>

Blog

<https://chris-ayers.com>

Contact

 BlueSky: [@chris-ayers.com](https://chris-ayers.com)

 LinkedIn: - [chris-l-ayers](#)

 Blog: <https://chris-ayers.com/>

 GitHub: [Codebytes](#)

 Mastodon:

@Chrisayers@hachyderm.io

 ~~Twitter: @Chris_L_Ayers~~