KICS:   
Keeping Infrastructure as Code Secure

short line

# By - Kavan Dalwadi

# 

About

Architecture

Demo

Queries

Result

CI Integration

Versions - <https://github.com/Checkmarx/kics/releases>

# 

# What is KICS?

## KICS finds security vulnerabilities, compliance issues, and infrastructure misconfigurations early in the development cycle of your infrastructure-as-code.

KICS scans and detects issues in following Infrastructure as Code solutions:

Terraform, Kubernetes, Docker, AWS CloudFormation, Ansible, Helm, Google Deployment Manager, AWS SAM, Microsoft ARM, and OpenAPI 3.0 specifications.

KICS is 100% open source and is written in Golang using Open Policy Agent (OPA), and Rego as a query language.

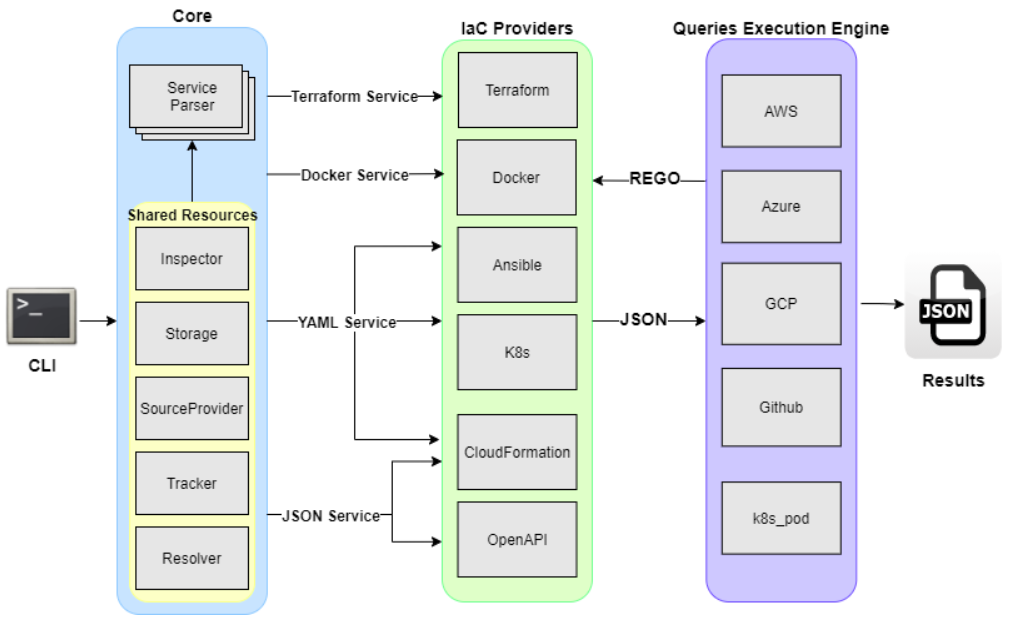
KICS 1.5 is the most advanced KICS version available. This version includes features such as:

* Enabling users to discover and better manage the security posture of runtime environments pertaining to their deployed cloud resources
* Scanning support for AWS Serverless Application Model (AWS SAM)
* Support for generating reports in AWS Security Finding Format (ASFF)
* More comprehensive support for scanning infrastructure deployed when using Google Cloud Deployment Manager

# 

# KICS Architecture

KICS has a pluggable architecture with an extensible pipeline of parsing IaC languages, which allows an easy integration of new IaC languages and queries.



* Command Line Interface => Provides CLI input to KICS.
* Parser => responsible for parsing input IaC files (terraform and others)
* IaC Providers => Converts IaC language into normalized JSON
* Queries Execution Engine => applies REGO queries against normalized JSON
* Security Queries => pre-built REGO queries for each security and misconfiguration
* Writer => Writes results into JSON format

# Demo -

Installing KICS as docker container -

1. Pull KICS image:

docker pull checkmarx/kics:latest

1. Run container and mount files/folder path as volume for scanning:

docker run -t -v "{path\_to\_host\_folder\_to\_scan}":/path checkmarx/kics scan -p "/path" -o "/path/"

Ref - <https://docs.kics.io/latest/getting-started/>

Output Result - <https://docs.kics.io/latest/results/#descriptions>

# Queries -

The anatomy of a query is straightforward. It builds up a policy and defines the result.

The policy builds the pattern that breaks the security of the infrastructure code and which the query is looking for.

The result defines the specific data used to present the vulnerability in the infrastructure code.

Each query has a metadata.json companion file with all the relevant information about the vulnerability, including the severity, category and its description.

Filesystem-wise, KICS queries are organized per IaC tool and grouped under the provider. Per each query created, it is mandatory the creation of test cases with, at least, one negative and positive case and a JSON file with data about the expected results.

Summarizing, the following is the expected file tree for a query:

- <technology>

|- <provider>

| |- <queryfolder>

| | |- test

| | | |- positive<.ext>

| | | |- negative<.ext>

| | | |- positive\_expected\_result.json

| | |- metadata.json

| | |- query.rego

Ref - <https://docs.kics.io/latest/queries/all-queries/>

# CI Integration -

For CI, we can create a job using a kics image to scan the code in a pipeline. Then it can check vulnerability for our code and store the output in artifacts.

GitLab-CI e.g -

image:

name: checkmarx/kics:latest

entrypoint: [""]

stages:

- kics

kics-scan:

stage: kics

script:

- kics scan --no-progress -p ${PWD} -o ${PWD} --report-formats json --output-name kics-results

artifacts:

name: kics-results.json

paths:

- kics-results.json

when: always

Ref - <https://docs.kics.io/latest/integrations/>

short dash