Terraform static code analysis

What is Terraform Static Code Analysis?

Terraform Static code analysis is a method of debugging the IAC source code by examining it before it is applied on cloud provider. The analysis is done by analysing the code with the set of predefined rules.

Tools Used to perform Static Code analysis on Terraform IAC:

We use three tools to perform Code analysis, they are

1. CHECKOV
2. TFLINT
3. Terraform Validate

Tools Configuration and Usage:

1. CHECKOV:
   1. Overview
      1. CHECKOV is a static code analysis tool for scanning Infrastructure as Code (IAC) files for misconfiguration that may lead to security or compliance problems.
      2. CHECKOV is mainly helpful in finding security and compliance problems on our IAC
      3. It contains more than 750 predefined policies to check for common Misconfiguration and security or compliance problems.
      4. It also provides us option to create custom rules using python which can be added along with the test.
      5. It uses a common CLI to manage and analyse Infrastructure As Code (IAC) scan results across platforms such as Terraform, Cloud Formation etc.
   2. Installing Checkov:
      1. **>> pip install checkov**
      2. Once installed we can check the checkov version using
         1. >> checkov –version
   3. Code Analysis Syntax using Chekov
      1. **>> checkov –file <IAC File name> --output <output type>**
      2. Example:
         1. **>> checkov –file main.tf –output json**
   4. Analysis with SonarQube:
      1. The checkov analysis report can be generated in Checkstyle format using below command
         1. **Checkov –file main.tf –output junit >> checkov\_report.xml**
      2. This check style XML file can be imported into Sonarqube and we can extend our analysis with SonarQube Quality Gates
   5. CHECKOV Reference: <https://www.checkov.io/1.Welcome/Quick%20Start.html>
2. TFLINT
   1. Overview:
      1. TFLINT is also a static code analysis tool for scanning Infrastructure as Code files for Misconfigurations.
      2. How TFLINT differs from CHECKOV is that TFLINT takes Existing environment into consideration while performing static code analysis.
      3. For example, we have a code to create some resources in Cloud provider and it uses others resources, TFLINT checks that the all-existing resources listed in the code already exist in our environment or not and perform deep analysis. (Note: Deep Scan needs to be enabled for this analysis on TFLINT)
      4. This will help us avoid most of the configuration related bugs.
   2. Installing TFLINT:
      1. No Installation is Needed
      2. Download the TFLINT Binary
      3. Create an environment path points to the Binary folder, so that the system will find the program when we call TFLINT.
      4. TFLINT Reference: <https://github.com/terraform-linters/tflint.git>
   3. Configuring Cloud provider Rules:
      1. In-order to make use of TFLINT full potential we need to integrate the rules set for corresponding cloud provider and need to initialize it.
      2. TFLINT Cloud Provider Rules Resources Link:
         1. The Rules for various cloud providers can be found in below Git profile
            1. <https://github.com/terraform-linters>
      3. We need to create a configuration file called “.tflint.hcl” and we need to define the cloud providers details inside the file.
      4. Each cloud-provider-rules repo contains detailed readme file on creating configuration file for corresponding Cloud provider.
      5. One the file is created we need to initialize the cloud provider using below command
         1. **>> tflint –init**
   4. Code Analysis Using tflint:
      1. Make sure that the TFLINT library path is added to the Environment variable path and we can able to access TFLINT from terminal
      2. To perform code analysis
         1. **>> tflint <filename> --format=junit**
      3. Example
         1. **>> tflint main.tf –format=junit**
   5. Analysis with SonarQube:
      1. The TFLINT analysis report can be generated in Check style format using below command

**>> tflint main.tf –format=junit >> tflint\_report.xml**

* + 1. This check style XML file can be imported into Sonarqube and we can extend our analysis with SonarQube Quality Gates