Prisma Cloud DevOps Security

docs.paloaltonetworks.com

techDOCS paloalto NETWORKS

Contact Information

Corporate Headquarters:
Palo Alto Networks
3000 Tannery Way
Santa Clara, CA 95054
www.paloaltonetworks.com/company/contact-support

About the Documentation

- To ensure you are viewing the most current version of this document, or to access related documentation, visit the Technical Documentation portal: docs.paloaltonetworks.com.
- To search for a specific topic, go to our search page: docs.paloaltonetworks.com/search.html.
- Have feedback or questions for us? Leave a comment on any page in the portal, or write to us at documentation@paloaltonetworks.com.

Copyright

Last Revised

March 23, 2022

Table of Contents

Prisma Cloud DevOps Security

The Prisma Cloud devOps security capabilities are geared to meet the common goal of delivering releases faster and preventing security lapses by applying a consistent set of checks through the build-to-release process that keep your applications and infrastructure secure.

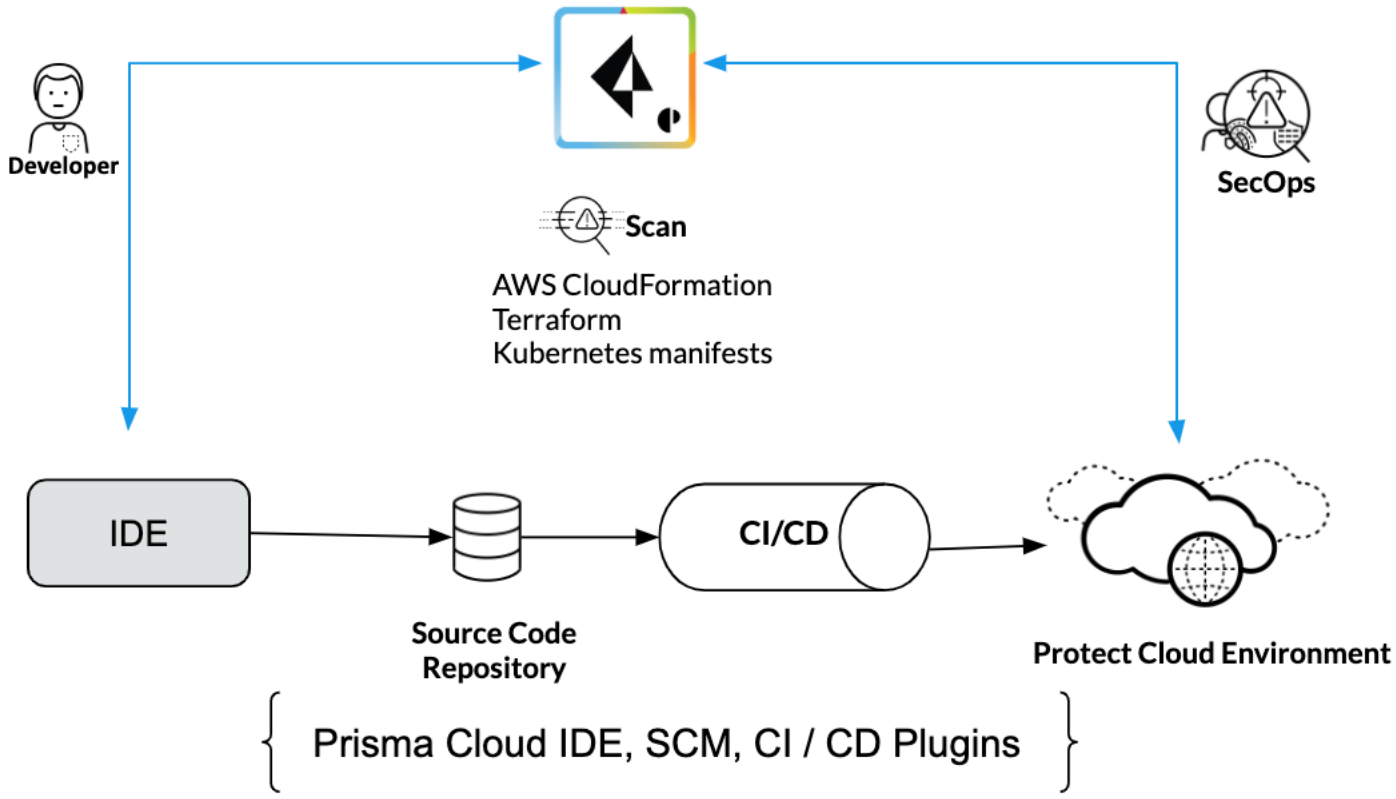
The Prisma Cloud devops security capabilities (IaC scan plugins) are being replaced with a new Code Security module. To enable the Code Security subscription on the Prisma Cloud Administrative Console, refer to the Code Security Administrator's Guide *for details and workflows. The Prisma Cloud devops security capabilities and workflows described in this guide are available to a select set of customers only.*

Secure Your Infrastructure Automation

Prisma Cloud DevOps Security enables DevOps and security teams to identify insecure configurations in Infrastructure-as-Code (IaC) templates and vulnerabilities in container images so that security issues are identified before actual resources are deployed in runtime environments.

To identify potential issues you can scan content in your IaC templates such as AWS CloudFormation Templates (JSON or YAML format), HashiCorp Terraform templates (HCL format), Kubernetes App manifests (JSON or YAML format), or Helm Charts against a list of IaC policies.

Helm Charts are a package manager for Kubernetes manifests.



With a valid Prisma Cloud Enterprise edition license, you can use the IaC scanning and container image scanning functionality in any of the following ways:

- **Plugins/Extensions**—Install and configure the Prisma Cloud Plugins for popular IDEs such as VScode, IntelliJ; Source Control Management systems such as Github ;CI/CD tools such as Jenkins, CircleCI, Azure DevOps. These plugins are designed to easily integrate in to your application development and deployment processes so that you can scan and fix issues in your current workflows without additional tools, thereby reducing the friction and boosting the adoption of better security checks.
- **Prisma Cloud IaC API**—Interact with the Prisma Cloud IaC scanning API endpoint using tools such as Curl, shell scripts, or Postman to scan IaC templates. Prisma Cloud recommends that you use the published plugins/extensions to perform IaC scanning, but you can use the IaC APIs directly for integrating with custom tools or specific use cases. See Use the Prisma Cloud IaC Scan REST API.

- **Twistcli**—Install and scan container images using twistcli. *Twistcli* is a command-line tool supported on Linux, macOS, and Windows, and it requires a Docker Engine to be installed on the machine where you are scanning images for vulnerabilities and malware.

Prisma Cloud Plugins

Prisma Cloud plugins enable you to check your DevOps infrastructure templates for security misconfigurations and scan container images to proactively prevent issues by shifting left.

The plugins or extensions as called on some environments, scan your templates against Prisma Cloud IaC policies to ensure compliance with security best practices before you deploy it into the cloud infrastructure. These plugins enable you to stay secure while being agile because they make it easy to scan your files, review any potential security issues, fix and validate code before you check it in to your source control repository or integrate it in your CI/CD pipeline.

Integration	Category	Marketplace	Documentation
AWS DevOps 	CI/CD	GitHub repository	Use the Prisma Cloud Extension for AWS DevOps
Azure DevOps 	CI/CD	Azure Visual Studio Marketplace	Use the Prisma Cloud Extension for Azure DevOps
Bitbucket 	SCM and CI/CD		Use the Prisma Cloud App for Bitbucket
CircleCI 	CI/CD	Circle CI Orb Registry	Use the Prisma Cloud Plugin for CircleCI
GitHub 	SCM	GitHub Marketplace	Use the Prisma Cloud App for GitHub
GitHub Actions 	CI/CD	GitHub Marketplace	Refer to the documentation in the GitHub Marketplace

Integration	Category	Marketplace	Documentation
GitLab 	SCM and CI/CD	—	Use the Prisma Cloud Extension for GitLab
IntelliJ IDEA 	IDE	Intellij Marketplace	Use the Prisma Cloud Plugin for IntelliJ IDEA
Jenkins 	CI/CD	Get the plugin from the Prisma Cloud administrative console (**Compute> Manage > System > Downloads**)	For scanning container image and serverless functions Use the Prisma Coud plugin for Jenkins
		Jenkins Marketplace	For scanning IaC templates Use the Prisma Cloud IaC Scan Plugin for Jenkins
Visual Studio Code 	IDE	VS Code Marketplace	Use the Prisma Cloud Extension for Visual Studio Code

Prisma Cloud Iac Scan-What's Supported

Template	Version	File Extensions
Terraform	0.11	tf, tf.json, json
	0.12	
	0.13	
CloudFormation (CFT)	<latest>	json, yml, yaml
Kubernetes Manifests	<latest>	json, yml, yaml
Helm Charts	v3	yml, yaml

Set Up Your Prisma Cloud Configuration File for IaC Scan

Prisma Cloud IaC Scan requires a Prisma Cloud configuration file in the repository where your templates are stored. This configuration file can include information about your IaC module structure, runtime variables, and tags that help refine your IaC Scan use. It enables Prisma Cloud IaC scan to support complex module structures and variable formats. This YAML file format is shared across all template types-K8s or Helm Charts, CFT and TF.

Make sure to use a syntax validation tool when you copy and paste content from this page. You can also get the yml file on GitHub

Create this file as **`.prismaCloud/config.yml`** in the root directory of your repository branch.

```
# This is the configuration file for Iac Scan APIv2.

# Specify the template types.The valid values are TF, CFT, K8S
template_type: TF

# For Terraform, it is recommended to provide one of the following values:
 0.11, 0.12 or 0.13.
# For CFT and K8s, this can be omitted.
template_version: 0.12

# For your template_type, fill in the details where applicable.
# variables: Specify any environment variables(TF) or parameters(CFT) as
 key:value pairs under this attribute.
# variable_files: Specify all the custom variable_files under here. It is an
 array of strings.
# variable_files: If it is standard extension(.tfvars for terraform), then
 this can be omitted.
# variable_files: For a custom extension, specify the variable file path
 from the root of the repo.
template_parameters:
  variable_files:
  - './TF/tf12_variable_files/smelks.tfvars'
  - './TF/tf12_nested_variable_files/network/smelks.tfvars'
  variables:
    cidr: 0.0.0.0/0
    acl: public-read-write
    AccessControl: PublicRead

# Define tags to identify your particular scan. You can use these tags to
 search the scan details on the Prisma Cloud.
tags:
  Org: Engineering
  Team: Shift_Left
  Env: Dev
```

IaC Scan Support for Version 1 (Deprecated)

The content of your Prisma Cloud configuration file depends on the IaC Scan support you need. The following show configuration details.

- Configure IaC Scan to Support Terraform
- Configure IaC Scan to Support AWS CloudFormation
- Configure IaC Scan to Support Kubernetes
- Configure Prisma Cloud Tags

Make sure to use a syntax validation tool when you copy and paste content from this page.

Configure IaC Scan to Support Terraform

The following shows the parameters in the Prisma Cloud configuration file that enable you to configure the IaC scan for Terraform 0.11 module with a variable file and/or input variables.

Make sure to use a syntax validation tool when you copy and paste content from this page.

```
     # Specify the template type. Valid values are as follows.
     # - For Terraform: TF
     # - For AWS CloudFormation: CFT
     # - For Kubernetes: K8S

     template_type: TF

     # The valid values for terraform_version are 0.11 or 0.12
     terraform_version: 0.11

     # If terraform_version is 0.11, then terraform_011_parameters is
required.
     # The value for variable_files is an array of custom variable file
names.The path of each file is relative to your repository branch root
directory
     # The value for variable_values is an array of name/value pairs that
identify the input variables your template uses.

     terraform_011_parameters:
     variable_files:
     - scan/rich-value-types/network/variables.tf
     variable_values:
     - name: check
     value: public-read-write
```

The following shows the parameters in the Prisma Cloud configuration file that enable you to configure the IaC scan for Terraform 0.12.

```
     # Specify the template type. Valid values are as follows.
```

```
      # - For Terraform: TF
      # - For AWS CloudFormation: CFT
      # - For Kubernetes: K8S

      template_type: TF

      # Valid values for terraform_version are 0.11 or 0.12.

      terraform_version: 0.12

      # If  terraform_version is 0.12, then terraform_012_parameters is
required.
      # The value of terraform_012_parameters is an array of root_modules.
The value for root_module is relative to your repository branch root
directory.
      # Each root module can have:
      # - variable_files, which is an array of variable file names relative
to your repository branch root directory
      # - variables, which is an array of name/value pairs that identify the
input variables for the module

      terraform_012_parameters:
      - root_module: scan/rich-value-types/
      variables:
      - name: check
      value: public-read-write
      - name: varName2
      value: varValue2
      - root_module: scan/for-expressions/
      variable_files:
      - scan/rich-value-types/expressions/variables.tf
```

Configure IaC Scan to Support AWS CloudFormation

The following shows the parameters in the Prisma Cloud configuration file that enable you to configure the IaC scan for Amazon CloudFormation templates with variables.

```
      # Specify the template type. Valid values are as follows.
      #For Terraform: TF
      # For AWS CloudFormation: CFT
      # For Kubernetes: K8S

      template_type: CFT

      # If template_type value is CFT, setcft_parameters (optional)
      # variable_values is an array of name/value pairs, which identifies
the
      # template variables

      cft_parameters:
      variable_values:
      - name:KeyName
      value: 10
      - name: AMI
      value: ami-45785
```

Configure IaC Scan to Support Kubernetes

The following shows the parameters in the Prisma Cloud configuration file that enable you to configure the IaC scan for Kubernetes.

```
     # Specify the template type. Valid values are as follows.
     # For Terraform: TF
     # For AWS CloudFormation: CFT
     # For Kubernetes: K8S

     template_type: K8S
```

Configure Prisma Cloud Tags

The following shows the parameters in the Prisma Cloud configuration file that enable you to identify Prisma Cloud tags in your template. These tags offer a flexible way to identify and organize your resources in Prisma Cloud.

```
     # Prisma Cloud Tags
     # tags is an array of labels that enable you to organize your
resources
     # with these key/value pairs in Prisma Cloud

     tags:
     - Org:Engineering
     - Team:Shift_Left
```

Use the Prisma Cloud Extension for AWS DevOps

With a Prisma Cloud Enterprise Edition license, you can integrate compliance and vulnerability checks into your AWS continuous integration/continuous deployment (CI/CD) and build environments. This extension enables you to scan Infrastructure-as-Code (IaC) templates like AWS CFT, Terraform templates, and Kubernetes deployment files against Prisma Cloud security policies. It also enables you to use Prisma Cloud Compute to scan container images for vulnerabilities.

The sections below show how to integrate the Prisma Cloud extension with your AWS CodePipeline pipelines and AWS CodeBuild projects.

- Review Prerequisites for the Prisma Cloud Extension for AWS CodePipeline
- Set Up IaC Scanning with AWS CodePipeline
- Set up Container Image Scanning with AWS CodeBuild

Review Prerequisites for the Prisma Cloud Extension for AWS CodePipeline

- A valid Prisma Cloud Enterprise Edition license.
- Prisma Cloud API URL

  The URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. The tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io. Replace app in the URL with api and enter it here. Refer to the Prisma Cloud REST API Reference, which is accessible from the Help Center within the Prisma Cloud web interface for more details.
- Prisma Cloud Access Key

  The access key enables programmatic access to Prisma Cloud. If you do not have a key, you must Create and Manage Access Keys.
- Prisma Cloud Secret Key

  The secret key was generated when you created the access key, and you should have saved it for later use. You cannot view it on the Prisma Cloud administrative console.
- (For container image scanning) Prisma Cloud Compute URL

  The base URL for your Prisma Cloud Compute console. Log in to the Prisma Cloud administrative Console and select **Compute** > **System** > **Downloads** > **Path to Console.**
- (For container image scanning) Prisma Cloud Compute User Credential

  Enter the Prisma Cloud username and password with the CI User role.
- A valid Prisma Cloud Configuration file in your repository root at .prismaCloud/config.yml.
- A valid AWS CodePipeline service role to give AWS CodePipeline access to other resources in your account.
- A multiple-stage pipeline in AWS CodePipeline, where at the source stage the execution collects your template repository and makes it available as input artifacts for subsequent stages.

If your customization uses any AWS commands, then you must install and configure the AWS command line interface.

Set Up IaC Scanning with AWS CodePipeline

You have two options to scan your IaC templates against Prisma Cloud security policies. The recommended approach is to use AWS CodeBuild to scan IaC templates, and the alternative is to use an AWS Lambda function with Python scripts.

You have two options for scanning your IaC templates against Prisma Cloud security policies: AWS CodePipeline which is recommended, or Lambda function which is optional. The Lambda function is compatible with versions one and two of the script, but the limitation is that the repository can't exceed 5MB; if it does then the IaC scan will be blocked.

- Use AWS CodePipeline to Scan IaC Templates
- Use an AWS Lambda Function with Python Scripting

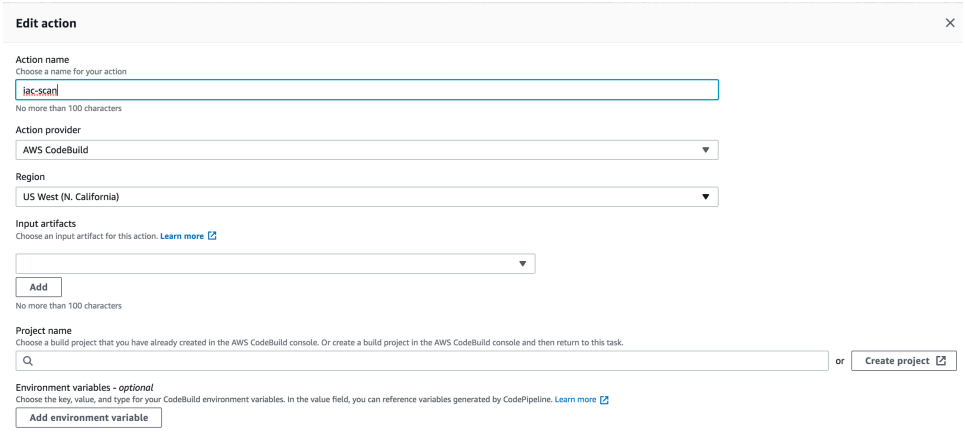
Use AWS CodePipeline to Scan IaC Templates

Add a CodeBuild Action to AWS CodePipeline Stage.

1. Create or edit AWS CodePipeline to add a new stage provider of AWS CodeBuild.

   In the home screen of AWS CodePipeline, select **Edit** > **Edit stage** > **Add action**.

   In the **Edit action** popup window, Enter your **Action name** and select **Action provider** > **AWS CodeBuild.**



2. Add IaC scan parameters as environment variables.

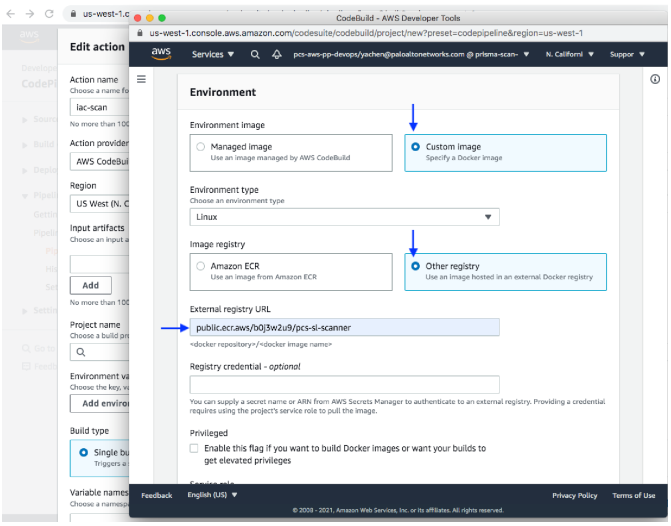
   Add the following IaC scan variables in the environment variables section:

Environment Variable	Value
`prisma_cloud_api_url`	Your Prisma Cloud API URL (e.g. **`https://api.prismacloud.io`**). The exact URL depends on the Prisma Cloud region and cluster of your tenant.
`prisma_cloud_asset_name`	A unique name to identify the scan results, for templates within this repository, on Prisma Cloud.
`prisma_cloud_access_key`	Your Prisma Cloud access key for API access.

Environment Variable	Value
prisma_cloud_secret_key	The secret key that corresponds to your Prisma Cloud access key.
Optional **prisma_cloud_tags**	Prisma Cloud tags enable visibility on the Prisma Cloud administrator console and are also used for build type alert rule scan policies matching. Provide the value for this environment variable as a comma-separated list of tags that you define. An example is: **project:x,owner:y**
Optional **prisma_cloud_repo_dir**	Directory path where you store your IaC templates. The default value is ". ". The mixing of templates is not allowed, therefore each IaC scan must have a single template type.
Optional **prisma_cloud_failure_criteria**	Threshold that should trigger a pipeline failure. Set the High : x, Medium : y, Low : z, Operator: O, where, x,y,z is the number of issues of each severity, and the operator is OR, AND. For example: • To fail the pipeline only when high severity issue is detected, High:1,Medium:1000,Low:1000,Operator:or • To never fail the pipeline, High:1000,Medium:1000,Low:1000,Operator: and

You may choose your own source type other than plain-text.

3. Create or Choose CodeBuild Project with IaC Scanner Docker Image.
   - **Specify the Prisma Cloud IaC scanner docker image**—Click **Create project** in the **Edit action** page. In the CodeBuild project **Environment** setting, select **Custom image**, under **Environment type** select **Linux**, and under **Image registry** select **Other registry**. Enter the Prisma Cloud IaC scanner docker image:`public.ecr.aws/b0j3w2u9/pcs-sl-scanner`

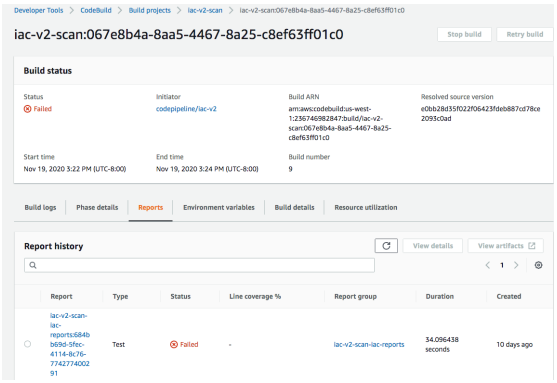


- **Define the CodeBuild Specification (Buildspec)**—The *buildspec* is a YAML file that contains the settings and commands that CodeBuild uses to execute a build.

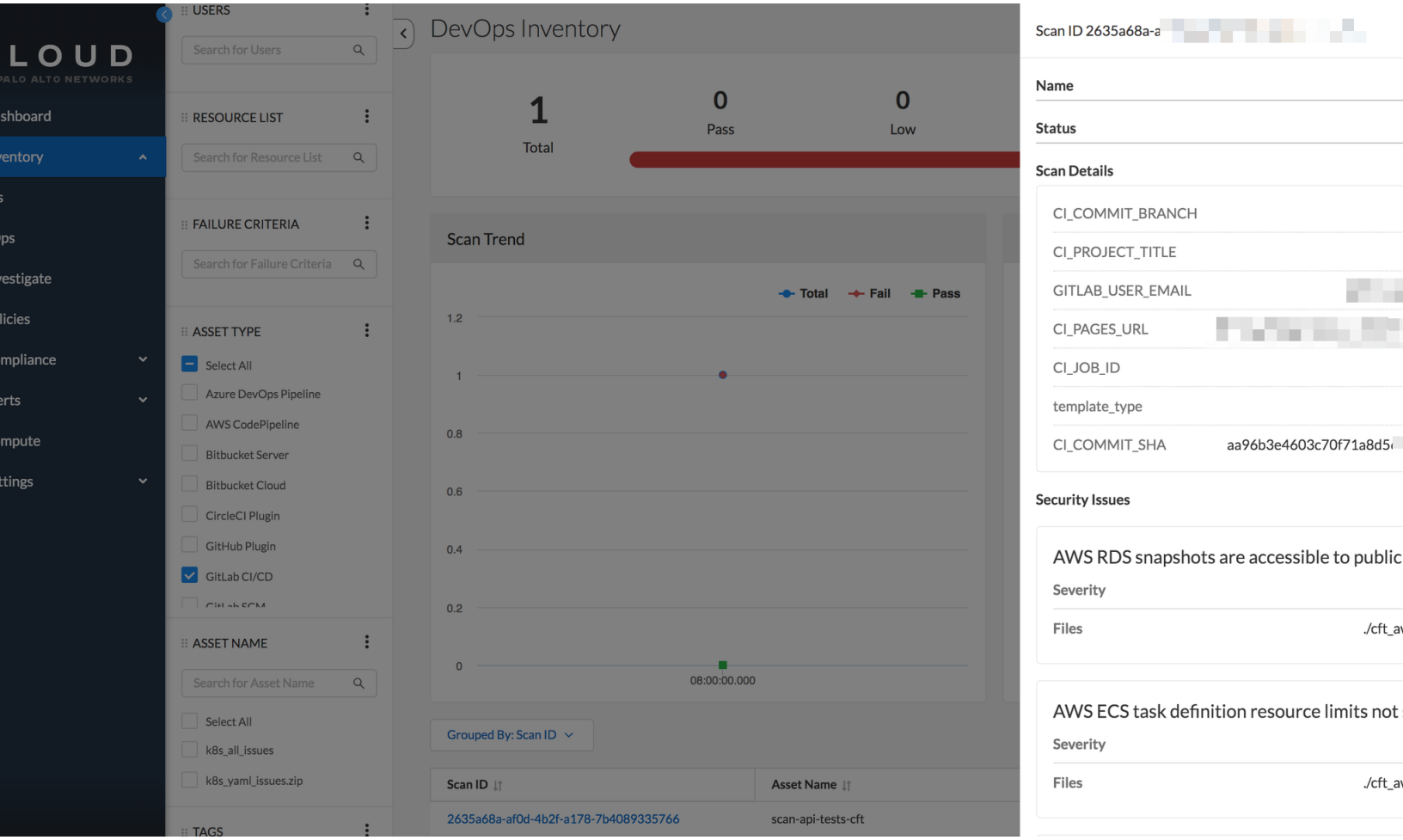
  An example buildspec to run *pcs_iac_repo_scan*:

```
version: 0.2

phases:
  build:
    commands:
      - pcs_iac_repo_scan
reports:
  iac-reports:
    files:
      - report/iac_scan/results.xml
    file-format: JUNITXML
```

- **View Scan Result in CodeBuild Reports**—If reports is defined in the buildspec, then the scan results can be viewed in the Reports tab, otherwise no report will generate. Click on the report link to view detailed violations from the scan result.



The same information appears on the Prisma Cloud **DevOps Inventory** page which you can access by selecting **Inventory** > **DevOPs**.



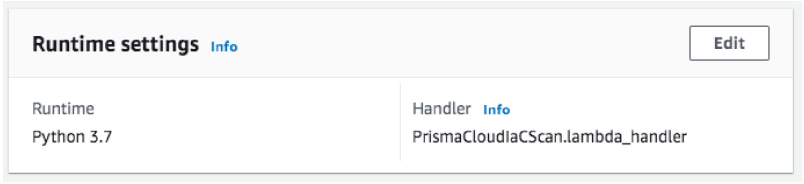
Use an AWS Lambda Function with Python Scripting

You can configure an AWS Lambda function and add it to your pipeline to check your IaC templates against Prisma Cloud security policies.

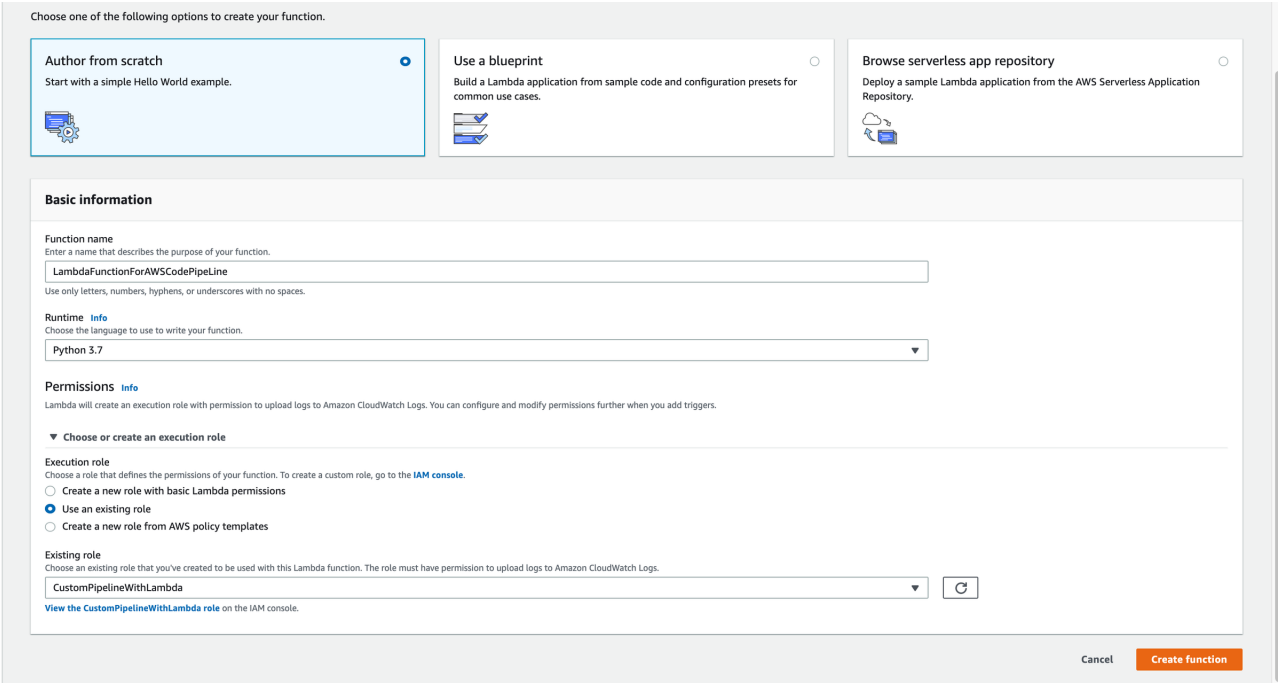
STEP 1 | Create an AWS Lambda Function from the Deployment Package.

Use the AWS command line interface (CLI) or console to create a Lambda function that scans the IaC templates against Prisma Cloud's comprehensive set of security policies.

- Choose a runtime environment of Python 3.6 or 3.7
- Set the **execution role** for Lambda so that it has:
  - Write permission for AWS Code Pipeline
    - *codepipeline:PutJobSuccessResult*
    - *codepipeline:PutJobFailureResult*
- List, Read, and Write permissions for **AWS CloudWatch Logs**
- Read permission for your S3 bucket if i t is your data source
- Use the lambda function with Prisma Cloud Python AWS Lambda deployment package
- Set the Lambda function handler to *PrismaCloudIaCScan.lambda_handler*

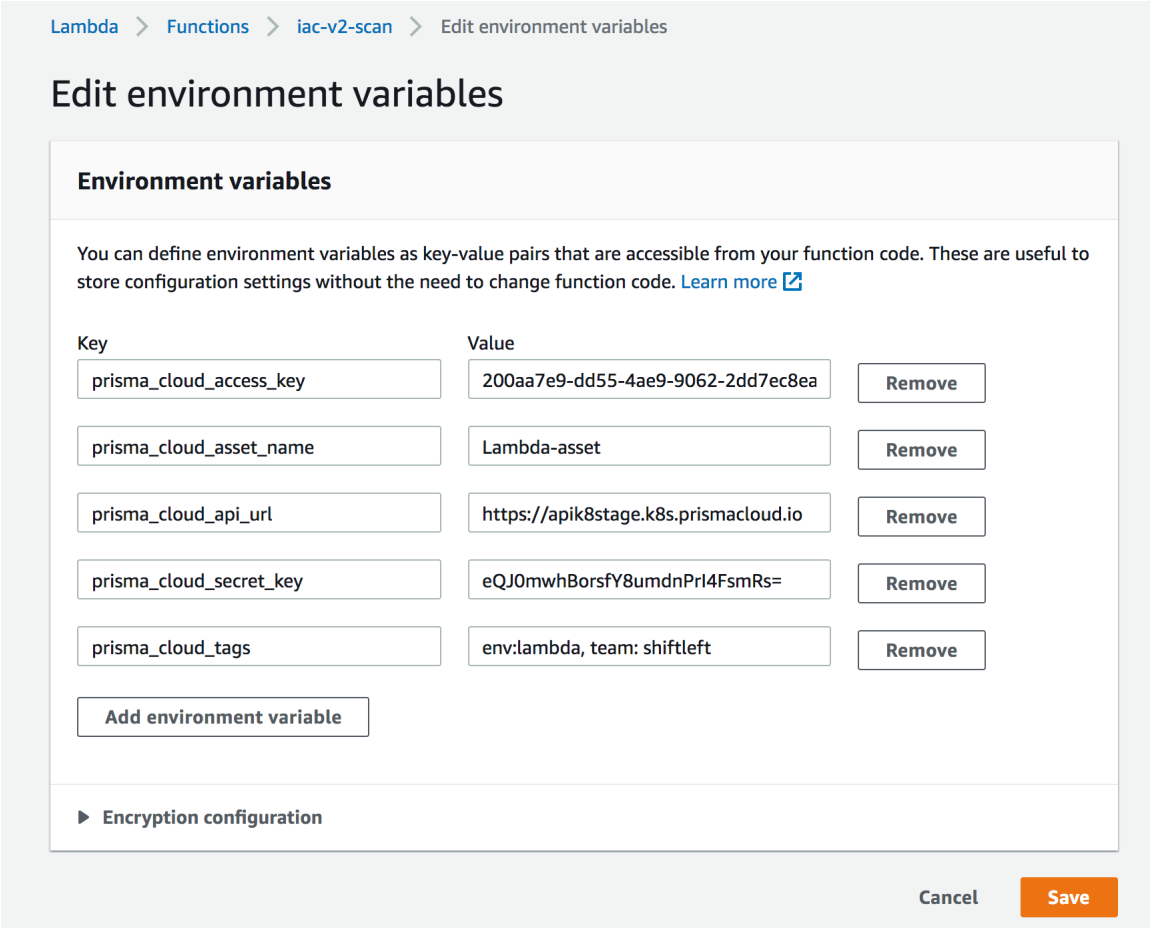


An example for AWS console.



STEP 2 | Configure Lambda Function with Required Environment Variables.

Provide the environment variables in your AWS CLI to create or set the Lambda in your AWS console.



STEP 3 | Adjust the Lambda timeout.

In the basic setting, change the default Lambda timeout from 3 seconds to 15 seconds.

STEP 4 | Add the Lambda Function Action to Your CodePipeline stage.

The following table identifies the fields that have values specific to Prisma Cloud. The value for the **User parameters** is in JSON format and specifies the conditions under which the pipeline job status will fail. In the table, the job will fail if the extension finds one high-severity violation, two medium-severity violations, or five low-severity violations.

Field	Value
Action provider	`AWS Lambda`
Function name	The function name you used when you created the Lambda function (e.g. `PrismaCloudIaCScan`)
User parameters	Example: `{"FailureCriteria": {"High":1,"Medium":2,"Low":5,"Operator":"or"}}` Valid values for "`Operator`" are "`or`" and "`and`"

An example of the **Edit action** entries is below:

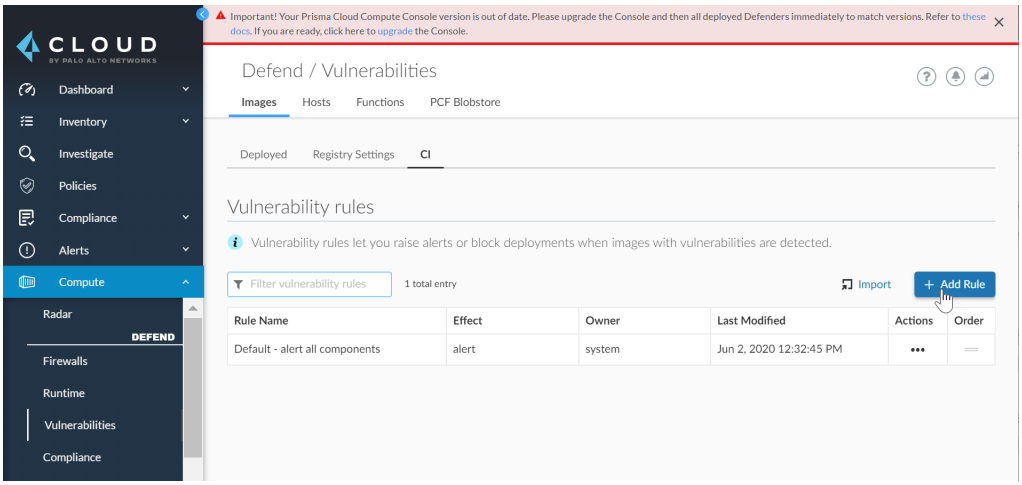
STEP 5 | View the scan result after pipeline execution.

To manually start a pipeline through the AWS console, select **Release change** on the pipeline details page. Select the link to execution details to view the latest CloudWatch logs.

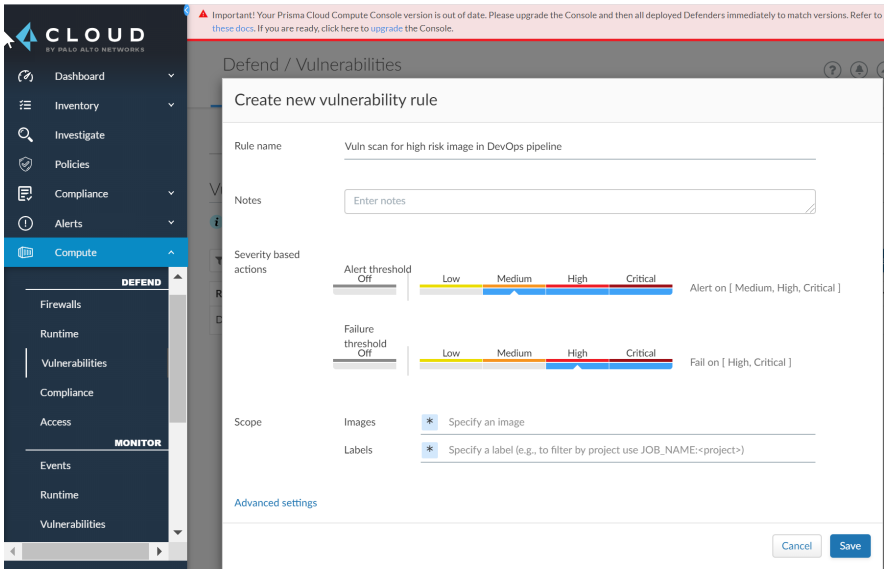
Set up Container Image Scanning with AWS CodeBuild

You can scan container images and serverless functions when you enable twistcli, add a vulnerability scan rule where you define the criteria to fail the build, and set up a task to scan the image or function in the pipeline.

STEP 1 | Select **Compute** > **Defender** > **Vulnerabilities** > **Images** > **CI**.



STEP 2 | Add rule and enter a rule name.



STEP 3 | Specify the **Alert** and **Failure** thresholds.

You can set the vulnerability scan to fail on critical, high, medium, or low severity. The failure threshold must be greater than the alert threshold.

STEP 4 | Specify the **Grace period**.

The grace period is the number of days for which you want. For more information about these settings, see the Prisma Cloud Compute Guide.

STEP 5 | Add a CodeBuild Action to AWS CodePipeline Stage.

Create or edit your AWS CodePipeline to add a new stage action with the provider of **AWS CodeBuild**.

STEP 6 | Configure the buildspec file.

Get the complete image scan Shell script. You will then have two options for using it: you can copy the script into your buildspec commands, or can remote shell in a single command: `curl https://gitlab.com/prismacloud-public/shift-left/extension/-/raw/master/aws-codepipeline/image_scan.sh | bash`

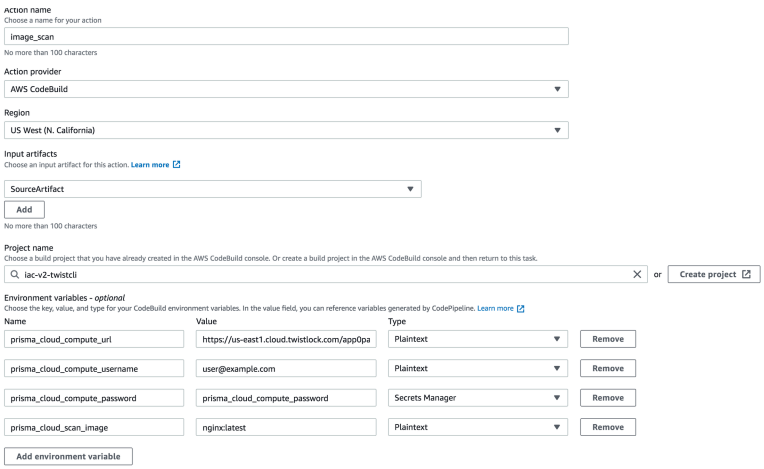
Below is an example buildspec to pull the container image and run the scan steps.

```
version: 0.2

phases:
  build:
    commands:
      # Build or pull the target container image specified in your
 environment variable
      - docker pull $prisma_cloud_scan_image
      # You may also copy the Shell script content for below URL
      - curl https://gitlab.com/prismacloud-public/shift-left/extension/-/
raw/master/aws-codepipeline/image_scan.sh | bash
```

STEP 7 | Set Image Scan CodeBuild Environment Variables.

If you created the buildspec using the above sample, the below CodeBuild environment variables should be added as they're used by CodeBuild.

ENVIRONMENT VARIABLE	DESCRIPTION
prisma_cloud_scan_image	Docker image to be scanned for vulnerabilities
prisma_cloud_compute_username	Prisma Cloud user with the Compute CI User role
prisma_cloud_compute_password	Prisma Cloud user password with the Compute CI User role
prisma_cloud_compute_url	The base URL for your Prisma Cloud Compute console. **Compute** > **System** > **Downloada** > **Path to Console**
prisma_cloud_compute_project	If Prisma Cloud Compute project is used, specify the project name along with the username and password



STEP 8 | View the Scan Results Build Logs.

```
|                |          |      |               |                  |        |             |            | is vulnerable to a Flush+Reload side-channel   |
|                |          |      |               |                  |        |             |            | attack because physical addresses are available to |
|                |          |      |               |                  |        |             |            | ot...                                          |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+
| ·[0mCVE-2016-10228·[0m | ·[93;1mlow·[0m      | ·[0m5.90·[0m | ·[0mglibc·[0m         | ·[0m2.28-10·[0m         | ·[0mopen·[0m   | ·[0m> 3 years·[0m   | ·[0m< 1 hour
·[0mThe iconv program in the GNU C Library (aka·[0m        |
|                |          |      |               |                  |        |             |            | glibc or libc6) 2.25 and earlier, when invoked |
|                |          |      |               |                  |        |             |            | with the -c option, enters an infinite loop when |
|                |          |      |               |                  |        |             |            | processing...                                  |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+
| ·[0mCVE-2020-10029·[0m | ·[93;1mlow·[0m      | ·[0m5.50·[0m | ·[0mglibc·[0m         | ·[0m2.28-10·[0m         | ·[0mopen·[0m   | ·[0m86 days·[0m     | ·[0m< 1 hour
·[0mThe GNU C Library (aka glibc or libc6) before 2.32·[0m |
|                |          |      |               |                  |        |             |            | could overflow an on-stack buffer during range |
|                |          |      |               |                  |        |             |            | reduction if an input to an 80-bit long double |
|                |          |      |               |                  |        |             |            | funct...                                       |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+
| ·[0mCVE-2019-20372·[0m | ·[93;1mlow·[0m      | ·[0m5.30·[0m | ·[0mnginx·[0m         | ·[0m1.17.10-1~buster·[0m | ·[0mopen·[0m   | ·[0m> 4 months·[0m  | ·[0m< 1 hour
·[0mNGINX before 1.17.7, with certain error_page·[0m       |
|                |          |      |               |                  |        |             |            | configurations, allows HTTP request smuggling, as |
|                |          |      |               |                  |        |             |            | demonstrated by the ability of an attacker to read |
|                |          |      |               |                  |        |             |            | unau...                                        |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+
| ·[0mCVE-2019-1551·[0m  | ·[93;1mlow·[0m      | ·[0m5.30·[0m | ·[0mopenssl·[0m       | ·[0m1.1.1d-0+deb10u3·[0m | ·[0mopen·[0m   | ·[0m> 5 months·[0m  | ·[0m< 1 hour
·[0mThere is an overflow bug in the x64_64 Montgomery·[0m  |
|                |          |      |               |                  |        |             |            | squaring procedure used in exponentiation with |
|                |          |      |               |                  |        |             |            | 512-bit moduli. No EC algorithms are affected. |
|                |          |      |               |                  |        |             |            | Analys...                                      |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+
| ·[0mCVE-2018-7169·[0m  | ·[93;1mlow·[0m      | ·[0m5.30·[0m | ·[0mshadow·[0m        | ·[0m1:4.5-1.1·[0m       | ·[0mopen·[0m   | ·[0m> 2 years·[0m   | ·[0m< 1 hour
·[0mAn issue was discovered in shadow 4.5. newgidmap·[0m   |
|                |          |      |               |                  |        |             |            | (in shadow-utils) is setuid and allows an      |
|                |          |      |               |                  |        |             |            | unprivileged user to be placed in a user namespace |
|                |          |      |               |                  |        |             |            | where se...                                    |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+
| ·[0mCVE-2019-19126·[0m | ·[93;1mlow·[0m      | ·[0m3.30·[0m | ·[0mglibc·[0m         | ·[0m2.28-10·[0m         | ·[0mopen·[0m   | ·[0m> 6 months·[0m  | ·[0m< 1 hour
·[0mOn the x86-64 architecture, the GNU C Library·[0m      |
|                |          |      |               |                  |        |             |            | (aka glibc) before 2.31 fails to ignore the    |
|                |          |      |               |                  |        |             |            | LD_PREFER_MAP_32BIT_EXEC environment variable  |
|                |          |      |               |                  |        |             |            | during program...                              |
+----------------+----------+------+---------------+------------------+--------+-------------+------------+------------------------------------------------+

·[1mVulnerabilities found for image nginx:latest: total - 36, critical - 0, high - 2, medium - 0, low - 34·[0m
·[31m·[1mScan failed due to vulnerability policy violations: kunjal-test-rule, 36 vulnerabilities, [low:34 high:2]·[0m·[0m


·[1mCompliance Issues·[0m
+----------+-------------------------------------------------------------------------+
| ·[1mSEVERITY·[0m | ·[1m                          DESCRIPTION                              ·[0m |
+----------+-------------------------------------------------------------------------+
| ·[91;1mhigh·[0m     | ·[0m(CIS_Docker_CE_v1.1.0 - 4.1) Image should be created with a non-root user·[0m |
+----------+-------------------------------------------------------------------------+

·[1mCompliance found for image nginx:latest: total - 1, critical - 0, high - 1, medium - 0, low - 0·[0m
·[1mCompliance threshold check results: PASS·[0m


[Container] 2020/05/29 22:29:56 Command did not exit successfully ./twistcli images scan --details -address $PC_COMPUTE_CONSOLE_URL -u $PC_COMPUTE_USER -p $PC_COMPUTE_
$IMAGE_REPO_NAME:$IMAGE_TAG exit status 1
[Container] 2020/05/29 22:29:56 Phase complete: POST_BUILD State: FAILED
```

Use the Prisma Cloud Extension for Azure DevOps

Use the Prisma Cloud extension to scan IaC templates, container images, and serverless functions in the build or release phase of the Azure DevOps pipeline. After you install this extension from the Azure Visual Studio Marketplace, you can set up the service connections for Prisma Cloud Iac Scan and Prisma Cloud Compute Scan, and then use custom tasks in the build or release pipeline for scanning IaC templates—AWS CloudFormation Templates, Terraform templates (versions of Terraform supported), Kubernetes manifests or app deployment YAML files— container images, or serverless zip files. When you create a custom task, you can specify the build or pipeline failure criteria based on severity of the security issues that the extension identifies.

When you set up the Prisma Cloud extension to scan, you can specify the tags at different stages. Prisma Cloud tags enable visibility on the Prisma Cloud administrator console, and are different from Azure DevOps tags or cloud tags that you may have included within your IaC templates. You can include these tags as **`key:value`** pairs in a comma separated list when you set up the service connection, and within the.prismaCloud/config.yml at the repository-level, or where you define the failure criteria for a Prisma Cloud IaC scan at the task level, and use it as a filter on Prisma Cloud.

- Install and Configure the Prisma Cloud Extensions
- Set up a Custom Task for IaC Scanning
- Set Up Container Image Scanning
- (Required with nested virtualization only) Set Up RASP Defender
- Sample YAML File
- Add Caches for Prisma Cloud Compute Scan
- Generate and Scan the Plan File

Install and Configure the Prisma Cloud Extensions

You need to add the prisma-cloud-config.yml in the root directory of your repository branch, and get the Prisma Cloud extension from the Visual Studio Marketplace, set up service connections to authenticate with Prisma Cloud and start scanning IaC templates, container images, and serverless functions.

STEP 1 | Set up your Azure DevOps organization and pipeline.

If you are just getting started with Azure Pipeline, refer to the Azure documentation.

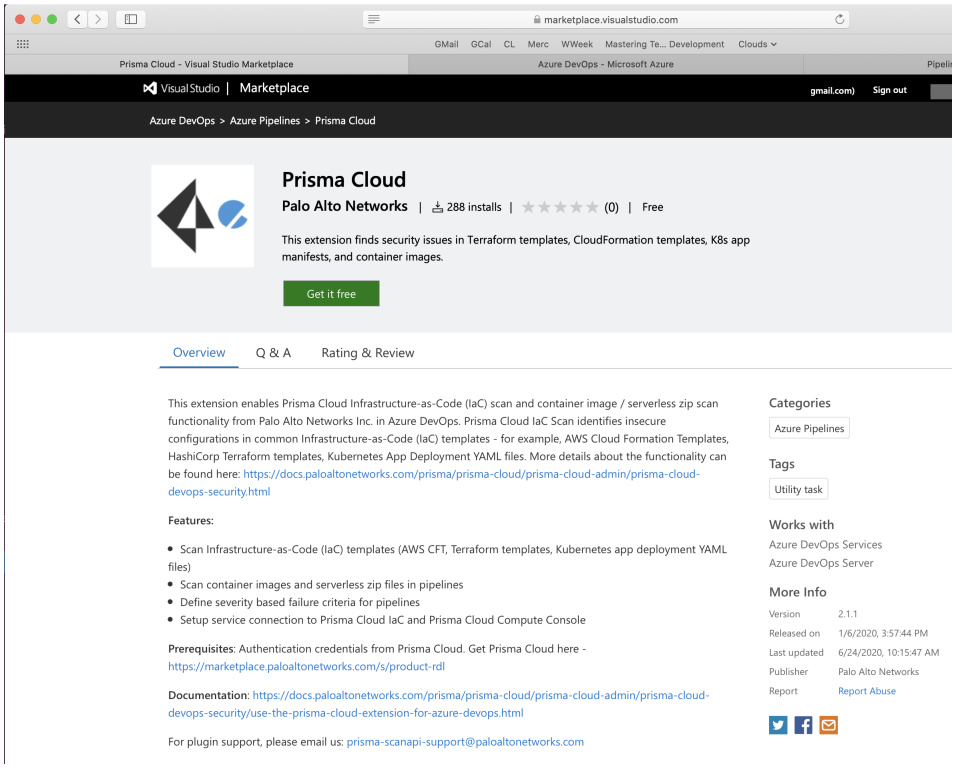
1. Create a project.
2. Create a new pipeline.
3. Select your code repository, configure, and save the pipeline.

STEP 2 | Set Up Your Prisma Cloud Configuration File for IaC Scan file.

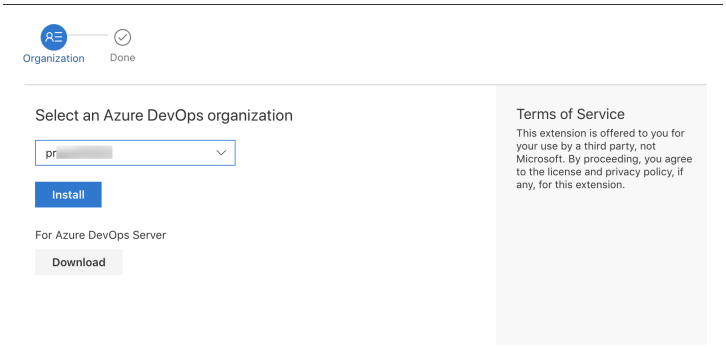
Create the **.prismaCloud/config.yml** and add it to the root directory of your repository branch. The file is required, and it must include the template type, version, and the template specific parameters and tags you use in your environment.

STEP 3 | Install the extension.

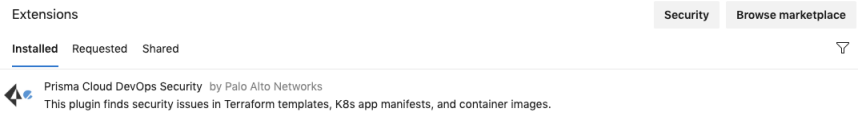
1. Search for **Prisma Cloud** in the Visual Studio Marketplace.



2. **Install** the extension in your Azure DevOps organization.



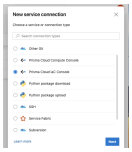
Select **Organization settings** > **Extensions** to verify that the extensions displays in the list of **Installed** extensions.



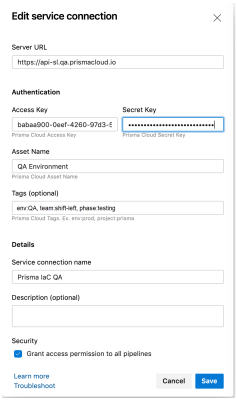
Updating the extension—the expected behavior is that Azure DevOps auto-updates the extension version. However, the tasks are not always properly updated, especially in major releases which can break the pipeline. To avoid such scenarios, it is recommended to uninstall and reinstall the extension.

STEP 4 | Add service connections to authenticate to Prisma Cloud.

You must create a new service connection for each type of scan— one for IaC scanning and one for scanning container image or serverless functions.



1. Select **Project Settings** > **Service Connections** > **New Service Connection** > **Prisma Cloud IaC Console**.
2. Enter the following information for the Prisma Cloud for IaC scanning and save your changes.



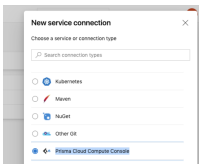
- Enter the Prisma Cloud API URL as **Server URL**.

  The URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. The tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io. Replace **`app`** in the URL with **`api`** and enter it here. Refer to the Prisma Cloud REST API Reference for more details.
- Enter your Prisma Cloud**Access Key**.

  The access key enables programmatic access. If you do not have a key, you must Create and Manage Access Keys.
- Enter your Prisma Cloud**Secret Key**.

  You should have saved this key when you generated it. You cannot view it on the Prisma Cloud web interface.
- Enter an **Asset Name** to identify the repository you want to scan.
- Enter the **Tags** to organize the templates that are scanned with this service connection, for visibility on Prisma Cloud.
- Provide a **Service connection name**.
- Verify that **Grant access permission to all pipelines** is selected and **Save** your changes.

3. Continue to the next step if you want to set up another service connection for container image scanning. If not, go to Set up a Custom Task for IaC Scanning.
4. Select **Project Settings** > **Service Connections** > **New Service Connection** > **Prisma Cloud Compute Console**.
5. Enter the following information for Prisma Cloud Compute Console and save your changes.



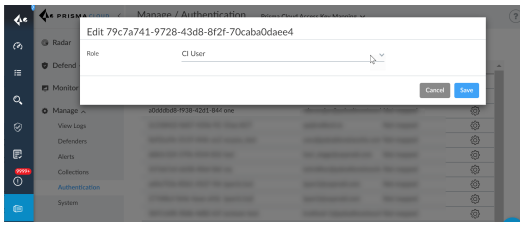
- Server URL.

You need to copy the server URL from the Prisma Cloud interface, **Compute** > **Manage** > **System** > **Downloads** > **Path to Console**. For Prisma Cloud Compute Edition, get the URL from **Manage** > **System** > **Downloads** > **Path to Console**

- Username and password.

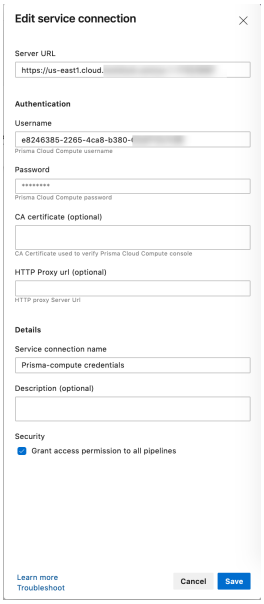
  These credentials are required for the service connection to authenticate with Prisma Cloud. If you are using Prisma Cloud Compute Edition (self-hosted), create a role and enter your username and password.

  If you are using Prisma Cloud Compute, you must first Create Prisma Cloud Roles with the Build and Deploy Security permission group and assign this role to the administrative user so that they can create an access key. The access key is the username and the secret key is your password.



If your password has special characters, make sure to escape any special characters when you enter your password.

- Optional CA certificate, if you are using certificate-based authentication.
- Optional HTTP Proxy URL, if you use a firewall or a proxy to enable access to the internet.
- Add a **Name** for the service connection.



- Verify that **Grant access permission to all pipelines** is selected.

STEP 5 | Continue with Set up a Custom Task for IaC Scanning.

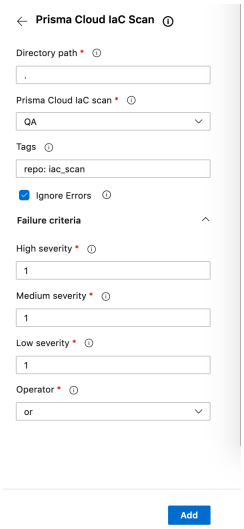
Set up a Custom Task for IaC Scanning

Use the following instructions to add a custom task for IaC scanning and container image and serverless functions scanning in your azure-pipelines.yml. In each task, you can define the pipeline failure criteria based on the severity of the issues that are detected during the scan.

STEP 1 | Under **Pipelines**, select your pipeline and **Edit** to add custom task.

STEP 2 | Add a custom task for IaC scanning.

1. Under **Task**, search for Prisma Cloud IaC Scan you created earlier.



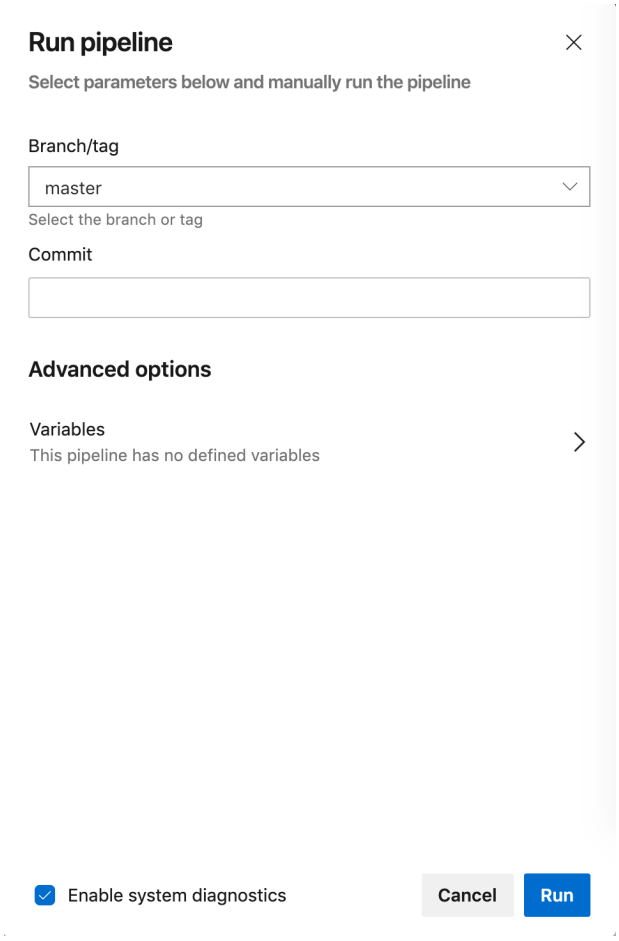
2. Enter the path for the directory you want to scan.

   If you want to scan the entire repository, use dot (`.`) or `$(System.DefaultWorkingDirectory)`.
3. Select the **Service Endpoint**, which is the service connection you created in the previous task.
4. Enter the **Tags** you want to apply to the templates that are being scanned.

   The tags format is `name:value`, and you can add multiple tags that are separated using commas.
5. Specify if you want to ignore API errors.

   By default, errors are not ignored and the pipeline fails when the scan detects API errors or vulnerabilities based on your failure criteria thresholds. Select **Ignore Errors** if you do not want the pipeline to fail on API errors; the pipeline will still fail when your failure criteria is met.
6. Select the **Failure Criteria** for the scan.

   You can set the count for High, Medium, Low severity issues and decide whether you want to use the AND or OR operator to specify your criteria. For example, if you have a very strict threshold and set the failure criteria to 0,0,0 with the OR operator your build will fail if the policy checks detect any issues.
7. **Add** to **yml** file, and **Save** the task.
8. **Enable system diagnostics** and **Run** .



STEP 3 | Run the task.

1. In Azure DevOps, click **Queue** to execute your task on the next available build agent.

   If your task configuration is incomplete, a red status message displays `Some settings need attention` just below **Run your build**.
2. Check the results.

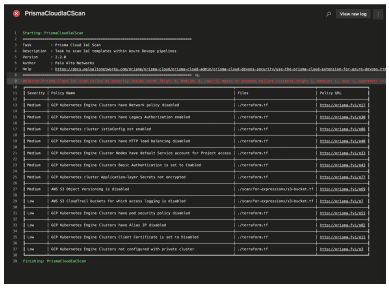
   - If the IaC Scan finds no issues the pipeline task result is successful.

   - If the IaC Scan finds issues but the failure criteria threshold you defined is not met, the job is successful but it displays the list of issues that were detected. For each policy that was violated, click the Policy URL link to review policy details.

     If the failure criteria you defined is more stringent that the default scan threshold, the job will fail and you can review results in the log file.

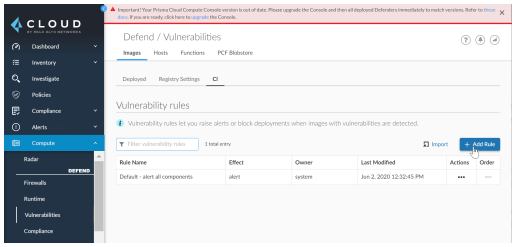


Set Up Container Image Scanning

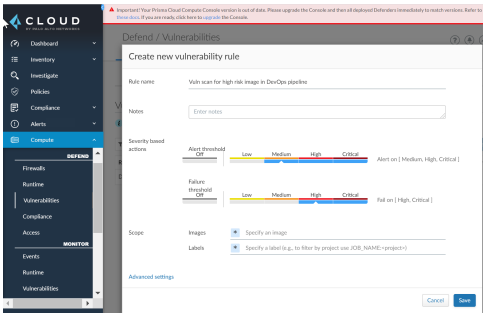
On Windows and Linux OS, you can scan container images and serverless functions when you enable twistcli, add a vulnerability scan rule where you define the criteria to fail the build, and set up a task to scan the image or function in the pipeline.

STEP 1 | Add a vulnerability scan rule on the Prisma Cloud Compute Console.

1. Select **Compute** > **Defender** > **Vulnerabilities** > **Images** > **CI**.



2. **Add Rule** and enter a **Rule name**.



3. Specify the **Alert** and **Failure** thresholds.

   You can set the vulnerability scan to fail on critical, high, medium, low severity. The failure threshold must be equal to or greater than the alert threshold.

4. (Optional) Specify the **Images** to scan.

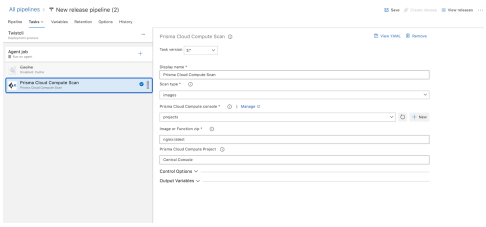
   The image or function zip file name is required later when you add the scan task to the pipeline in Step 3.

5. (Optional) Select **Apply rule when vendor fixes are available**, if you want to scan only for vulnerabilities that have fixes available.

6. Specify the **Grace period**.

   The grace period is the number of days for which you want to ignore a vulnerability. The time frame is measured in days starting at the date from the first vendor publish. For more details on the advanced settings, see the Prisma Cloud Compute guide.

STEP 2 | Add a pipeline task to scan container images using twistcli.

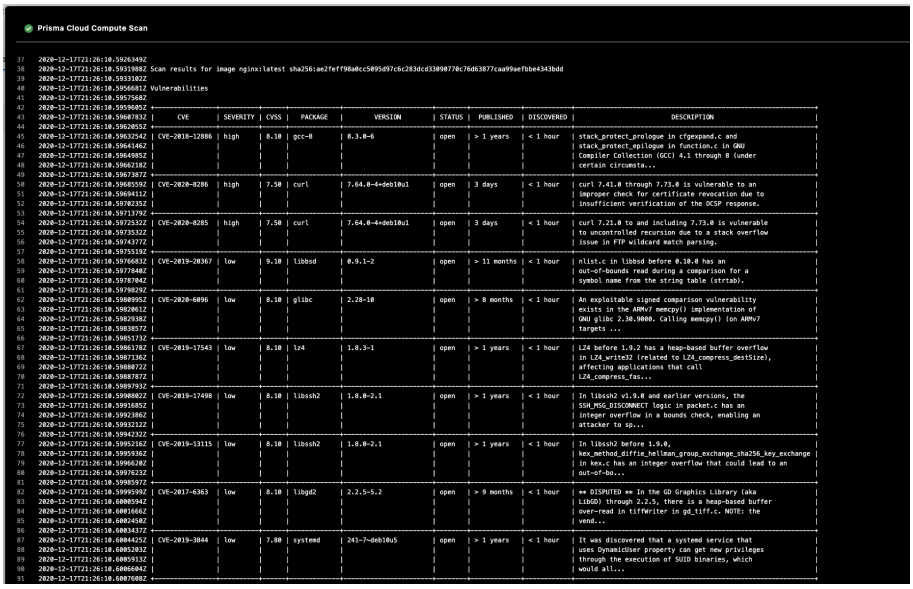


1. Select Pipelines, and **Edit** your pipeline and to add custom task.
2. Search for **Prisma** in the task list and select **Prisma Cloud Compute twistcli scan**.
3. Select the **Scan type**—Images or Serverless.
4. Select the **Prisma Cloud Compute Console** service connection name that you created earlier, from the drop-down.
5. Specify the **Image** name or serverless **Function zip** file name.

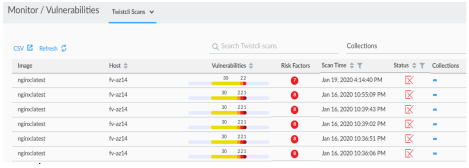
   The image name you enter here must match the name of the image you are building in the pipeline, if it doesn't the scan will fail.
6. Specify the **Project** name if applicable.

STEP 3 | View the results of the scan.

Click on the job, and then select the **prismacloudcomputescan** task to view the CLI output.



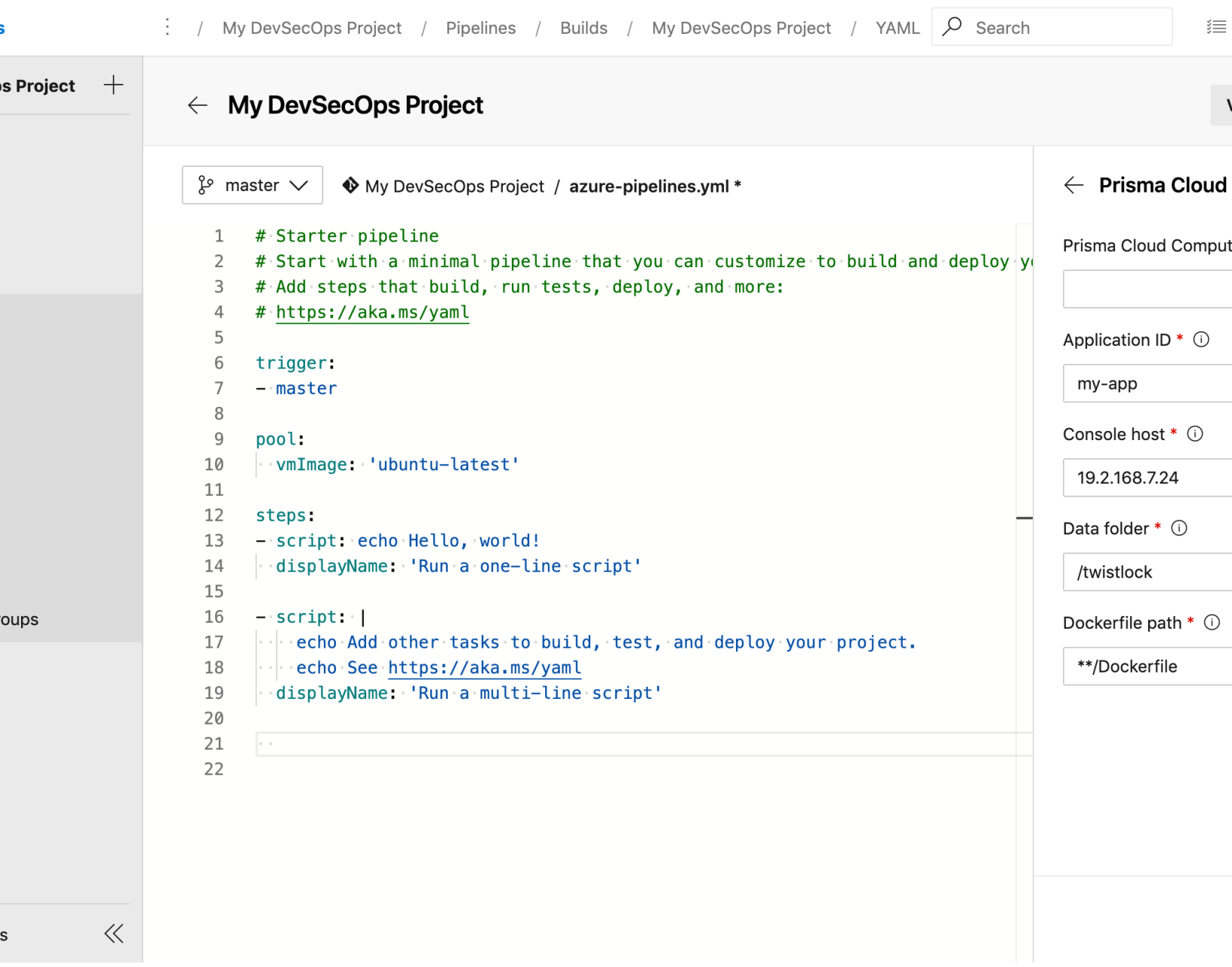
To see results on Prisma Cloud, select **Compute** > **Monitor** > **Vulnerabilities** > **Twistcli Scans**



Set Up RASP Defender

If you are using Docker-in-Docker, where you have a Docker container that itself has Docker installed, and from within the container you use Docker to pull images, build images, run containers, you have to set up RASP Defenders to secure containers at runtime.

Update the Dockerfile and embed the RASP defender as part of the Azure DevOps build.



1. Select Pipelines, and **Edit** your pipeline and to add custom task
2. Search for **Prisma** in the task list and select **Prisma Cloud Compute embed RASP**.
3. Select the **Scan type**—Images or Serverless.
4. Select the **Service connection** you created earlier for Prisma Cloud Compute Console.
5. Provide a unique **Application ID** for the RASP defender.

   For example, <your company>-<app>
6. Enter the **Console Host**, which is the DNS name or IP address of your Prisma Cloud Compute Console.
7. Specify the **Data Folder**, which is the read-write directory in the container file system.

   For example, /twistlock/.
8. Enter the **Dockerfile path** of the container image to which you want to add the RASP defender.

Sample YAML File

The following is a sample azure-pipeline.yml when you enable both the Prisma Cloud IaC scan and Prisma Cloud Compute scan. This file autogenerates is referenced below as an example.

```
 # Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy
 your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml
trigger:
 branches:
    include:
    - master
pool:
  vmImage: 'ubuntu-latest'
steps:
- task: Palo-Alto-Networks.build-release-task.custom-build-release-
task.prisma-cloud-compute-scan@1
  displayName: 'Prisma Cloud Compute Scan'
  inputs:
    twistlockService: 'NewEnv Connection'
    artifact: 'nginx:latest'
- task: Prisma Cloud IaC Scan@1
  inputs:
    Path: 'repo'
    prismaCloudService: 'Prisma Cloud Scan'
    High: '0'
    Medium: '0'
    Low: '0'
    Operator: 'or'
- script: |
    echo Add other tasks to build, test, and deploy your project.
    echo See https://aka.ms/yaml
 displayName: 'Run a multi-line script'
```

Add Caches for Prisma Cloud Compute Scan

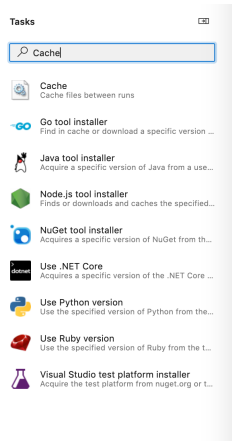
If no Cache Task is implemented in Prisma Cloud Compute Scan, then the task downloads the twistcli binary with every run. This can be time consuming as the same pipeline is executed every time. We can add cache task which will download the twistcli binary only when either of these is true:

- The binary is not already present.
- When a different version of the binary is required.

Note: Caches are only available for yaml pipelines and not for classic pipelines.

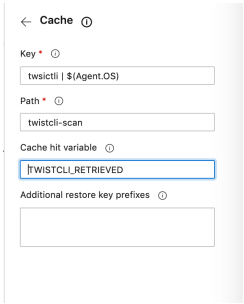
STEP 1 | Add Cache Task to your Prisma Cloud Compute Scan Pipeline.

In your pipeline search for Cache Task.



STEP 2 | Enter the details for your Cache.

- **Key**: a unique value to identify and retrieve the cache value later.
- **Path**: enter **`twistcli-scan`**.
- **Cache-hit-variable-enter**: **`TWISTCLI_RETRIEVED`**.



STEP 3 | Click Add to add this task to your pipeline.

STEP 4 | Add your task for Prisma Cloud Compute Scan.

```
steps:
Settings
- task: Cache@2
  inputs:
    key: 'twsictli | $(Agent.OS)'
    path: 'twistcli-scan'
    cacheHitVar: 'TWISTCLI_RETRIEVED'

Settings
- task: CmdLine@2
  inputs:
    script: |
      echo Write your commands here

      echo Hello world

      cd twistcli-scan
      ls

Settings
- task: prisma-cloud-compute-scan@3
  inputs:
    scanType: 'images'
    twistlockService: 'with-proxy'
    artifact: 'nginx:latest'
```

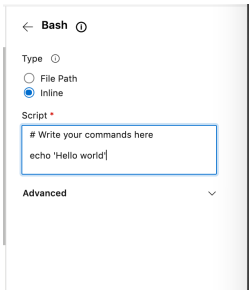
The first time the pipeline runs, it will download the twistcli binary; afterwards, it will first look into the cache and only download the binary if required.

Generate and Scan the Plan File

Use the plan file to scan your repositories. For example, your current repository refers to templates in remote repositories.

STEP 1 | Create the Azure DevOps pipeline for your repository.

Add a Bash task and choose **Inline** or **Script** based on your environment.



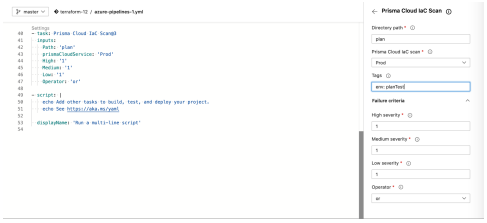
STEP 2 | Configure your plan file.

The script should have commands for downloading Terraform and generating a plan file of your repository.

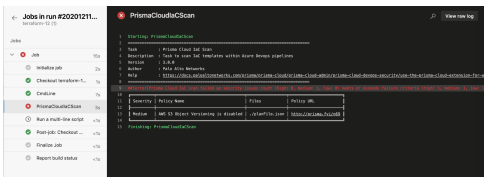
For example, in the following script, the plan file for scan/for-expressions folder is being generated using Terraform 0.13. The generated plan file is in JSON format which was placed under the **plan** folder, and .prismaCloud is copied to the same folder:

```
Settings
- task: CmdLine@2
  inputs:
    script: |
      curl -O -L https://releases.hashicorp.com/terraform/0.13.2/terraform_0.13.2_linux_amd64.zip
      unzip terraform_0.13.2_linux_amd64.zip
      rm terraform_0.13.2_linux_amd64.zip
      chmod +x terraform

      export AWS_ACCESS_KEY_ID=AKIATOHZYIW7Y2
      export AWS_SECRET_ACCESS_KEY=hKLAUSjmIm1ZWu/++wh3
      cd scan/for-expressions
      ../../terraform init
      ../../terraform plan -out=planFile
      mkdir ../../plan
      ../../terraform show -json planFile > ../../plan/planFile.json

      echo "Check if plan file got created at right place/n"
      ls -la ../../plan/

      echo "Going back to the root/n"
      cd ../..
      echo "copy .prismaCloud folder to plan folder/n"
      cp -r .prismaCloud plan/
```

STEP 3 | Add Prisma Cloud Scan IaC task and your folder path in **Directory path**.



STEP 4 | Run the pipeline to view the results.



Use the Prisma Cloud App for Bitbucket

Use the Prisma Cloud app to scan IaC templates during the code review and merge phase of your development process to provide feedback where you need it. This capability allows you to shift left, and deploy secure code with confidence.

- Use the Prisma Cloud App for Bitbucket Server

Use the Prisma Cloud App for Bitbucket Server

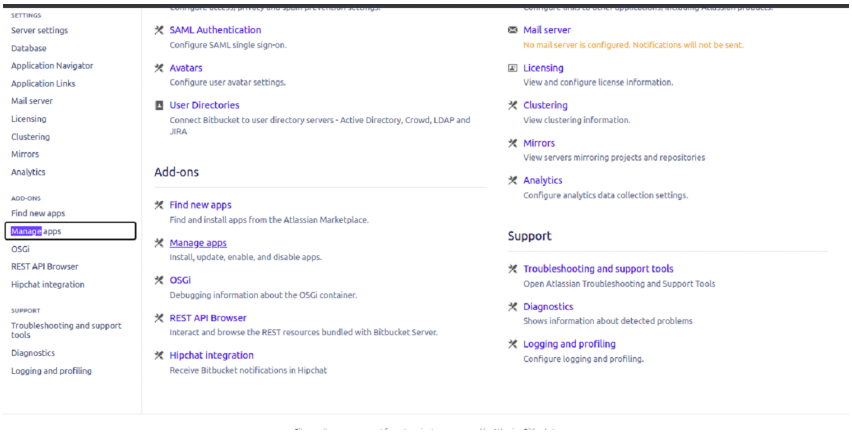
The Prisma Cloud ™ app for Bitbucket Server allows you to perform IaC scans on Bitbuckets pull requests and check them against Prisma Cloud's comprehensive set of policies. The plugin performs a full repository scan for the branch that the pull request was made on. If policy violations exceed the specified threshold, the pull request is blocked and a comment containing the security issues is displayed. The results generate within Bitbucket as a report and provides you with visibility into the security of your Bitbucket workflow.

STEP 1 | Install the Prisma Cloud app for Bitbucket Server.

You must have administrator privileges to install apps on Bitbucket server.

1. Launch a browser and go to the following URL: **`http://localhost:7990`**.

   If you used a different port number then replace *7990* with that number.
2. Enter your login credentials and authenticate.
3. Select **Admin** > **Manage apps** > **Find new apps.**



4. Enter **`prisma cloud`**.
5. **Install** and restart your Bitbucket server.

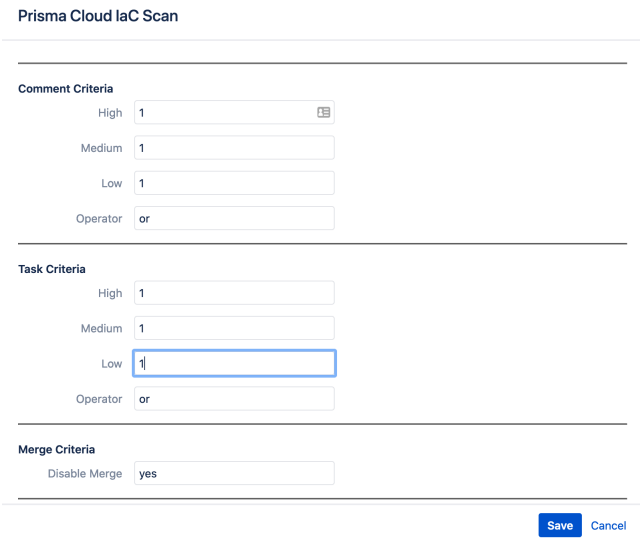
STEP 2 | Specify the settings for the Bitbucket app.

You configure the Bitbucket app settings to connect it to your instance of Prisma Cloud.

1. Select **Users** > **Projects** > **Repositories**, and then select the repository you want to scan.
2. Select the gear ⚙ icon and then **Prisma Cloud Settings**.

   - **Prisma Cloud API URL**—The URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. If the tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io, replace **`app`** in the URL with **`api`**. Refer to the Prisma Cloud REST API Reference for more details.

- **Prisma Cloud Access Key**—Enables programmatic access to Prisma Cloud. To create an access key select **Settings** > **Access Keys** > **Add New.**
- **Prisma Cloud Secret Key**—You should have saved this secret key when you generated it. You cannot view it on the Prisma Cloud web interface.
- **Prisma Cloud SCM Asset Name**—The name of the *assetName* you want to scan. For example, *bitbucket-build-test-1*.
- **Prisma Cloud SCM Tag**—The key-value pairs separated by commas which allows the build to be searched in the DevOps inventory UI of Prisma Cloud. Examples of valid tags; **`env:dev`**, **`tag:value`**, **`team:team-one`**.
- **Template Type**—A template is a configuration management tool that is used to provision resources in the cloud. The templates supported are Terraform, AWS CloudFormation, and Kubernetes Templates. Enter the templates abbreviations as values, for example **`TF`**, **`CFT`**, and **`K8S`**.
- **Template Version** (Optional)—This field is only applicable if you entered **`TF`** in the Template Type field. See what versions of Terraform are supported.The value you enter will be a hint as the system will attempt to determine the correct version number. If the version number can't be detected, then the system will use the value you entered.



3. **Save** the settings and click **Test Connection** to confirm that your login credentials work.

STEP 3 | Configure the Failure Threshold.

1. Specify the number of issues for each severity.

   Select **Enabled** next to **Prisma Cloud IaC Scan** to define the number of issues by severity.

   Set the High: *x*, Medium: *y*, Low : *z*, Operator: *O*, and Merge: *m* values. The variables *x*, *y*, and *z* are the number of issues of each severity, *O* represents the logical operators OR and AND, and *m*, represents YES or NO for the merge request.

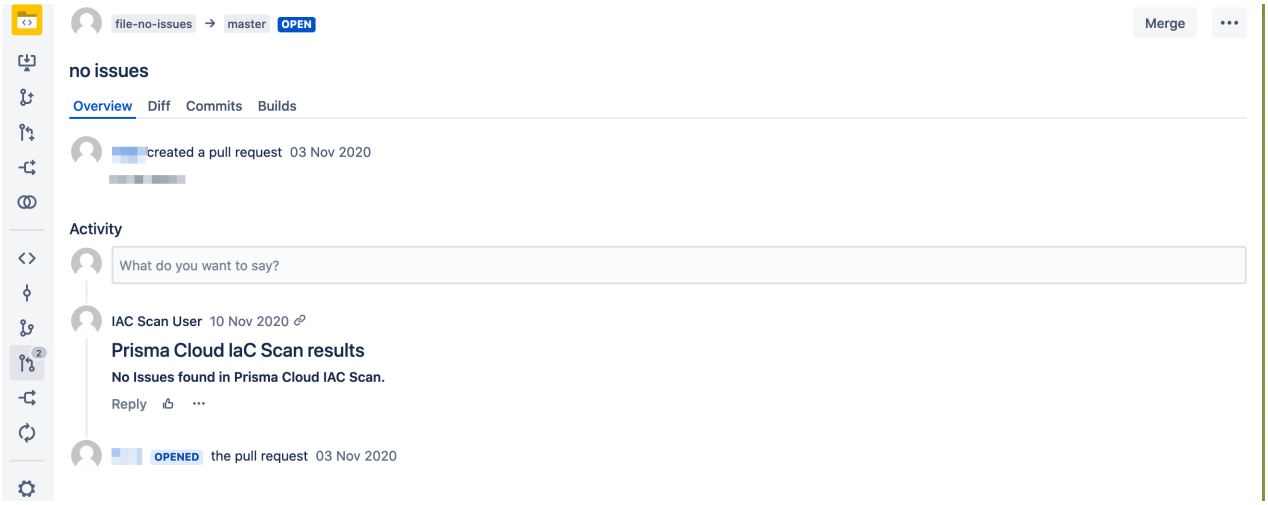
   For example:

   - To fail the pull request for any security issue detected—*High* : 0, *Medium* : 0, *Low* : 0, *Operator*: OR, *Merge*: no.
   - To never fail the pull request—*High* : 1000, *Medium* : 1000, *Low* : 1000, *Operator*: AND, *Merge*: yes.

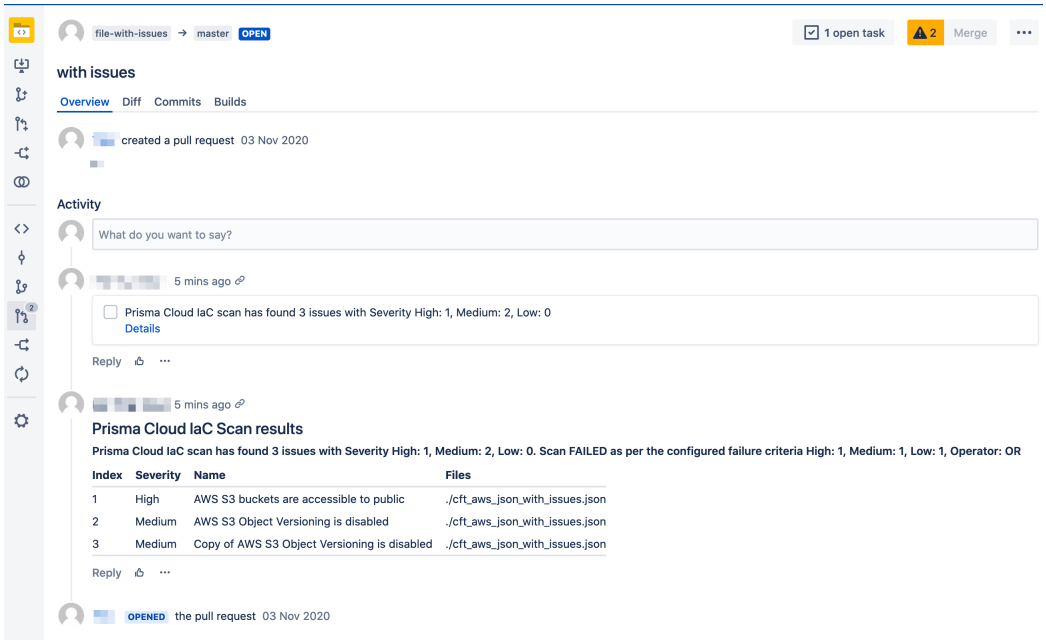
   Comment Criteria and **Task Criteria** are two ways to show the report of the IaC scan to the end user. If the number of issues matches the values specified in the **Comment Criteria** then a comment will be created, and if the values matches a **Task Criteria** then a task will be created.
2. Generate a pull request.

Create a pull request by selecting the pull request icon. Enter the details for your pull request.

An example of a pull request with a comment:



An example of a pull request with tasks:



The two tasks must be resolved in order for the pull request to be merged into the main branch.

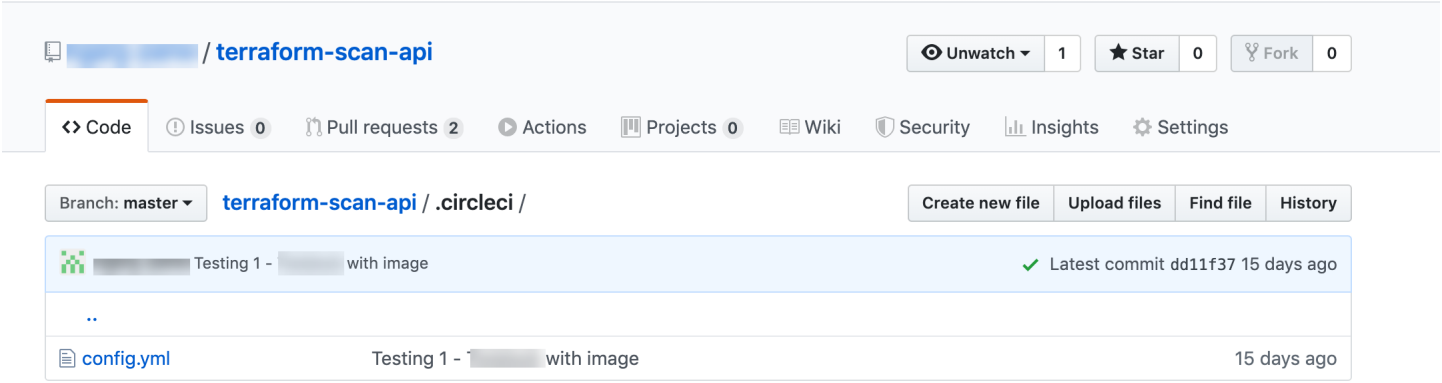
Use the Prisma Cloud Plugin for CircleCI

Use the Prisma Cloud orb for CircleCI to scan IaC templates and container images during CircleCI pipelines. In order to use Prisma Cloud IaC scan functionality, you need to have a connection to a Prisma Cloud API server and have details for that connection specified as environment variables. Similarly, in order to perform container image vulnerability scans, you need a connection to the Prisma Cloud Compute console. When you create a custom task to embed this functionality in your CircleCI pipeline, you can specify the build or pipeline failure criteria based on the severity of the security issues that are identified.

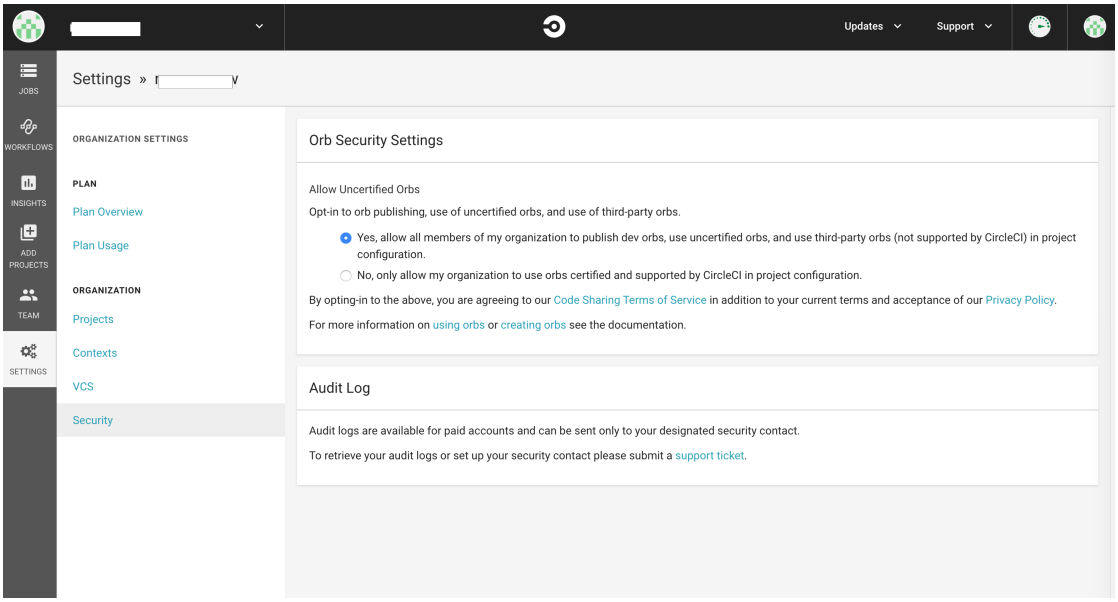
STEP 1 | Verify the prerequisites.

- Verify the .circleci/config.yml is in your project root directory.

  CircleCI uses this file each time it runs a build.



- Set CircleCI org permissions to allow orbs that are not part of the Certified and Partner list. Your CircleCI org admin can opt in to use uncertified third-party orbs by navigating to **Settings** > **Security** and selecting the opt-in setting.



- For IaC scan, get the details for enabling authentication to Prisma Cloud.
  - **Prisma Cloud API URL**.

The URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. The tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io. Replace **`app`** in the URL with **`api`** and enter it here. Refer to the Prisma Cloud REST API Reference, which is accessible from the Help Center within the Prisma Cloud web interface for more details.

- **Access Key**.

  The access key enables programmatic access to Prisma Cloud. If you do not have a key, you must Create and Manage Access Keys.

- **Secret Key**.

  You should have saved this secret key when you generated it. You cannot view it on the Prisma Cloud web interface.

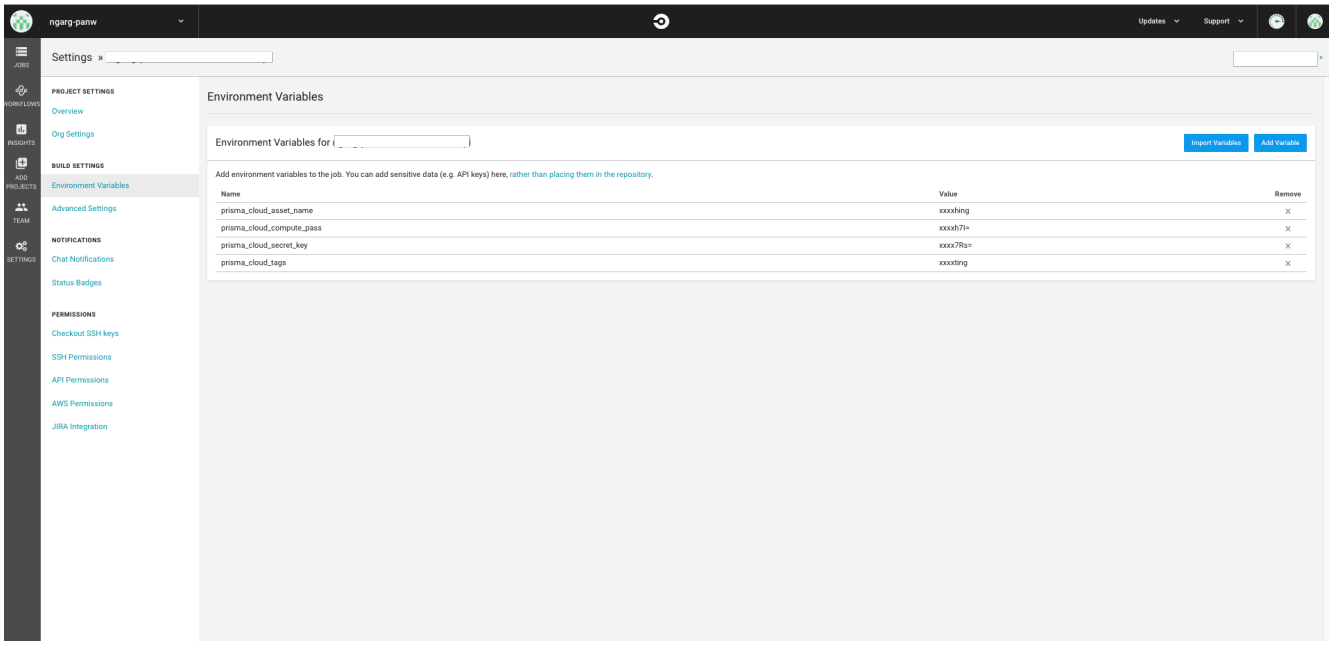
- For image scan, get the details for authenticating to the Prisma Cloud Compute.
  - Prisma Cloud Compute URL.

    You need to copy the URL from the Prisma Cloud interface, **Manage** > **Defenders** > **Deploy**.
  - Prisma Cloud Compute username and password.

STEP 2 | Add the environment variables for enabling authentication to Prisma Cloud.

On CircleCI, you must add the environment variables as name value pairs. The following table lists the environment variables, and the figure below shows an example of environment variable settings for IaC scans.

Name	Value	Notes
prisma_cloud_asset_name	CircleCI server. Examples: creditapp_server, ConsumerBU_server	Used to track specific results in Prisma Cloud. For IaC scan only. Required.
prisma_cloud_secret_key	Prisma Cloud secret key	See Create and Manage Access Keys for details about the secret key. For IaC scan only. Required.
prisma_cloud_compute_pass	Prisma Cloud Compute password	The Prisma Cloud Compute users's password.
prisma_cloud_tags	Comma-separated list of key/value pairs. Examples: project:x, owner:mr.y, compliance:pci	Used for visibility in Prisma Cloud UI. For IaC scan only. Optional.



1. For IaC scan, enter the **Name** and **Value** for the Prisma Cloud secret key.

   The default name is **`prisma_cloud_secret_key`**. If you enter a different name, make sure to match this name in the config.yml file in the next step.
2. For IaC scan, enter the **Name** and **Value** for your asset name.

   Prisma Cloud uses **`prisma_cloud_asset_name`** to track specific scan results.
3. For IaC scan, enter the **Name** and **Value** for all the **`prisma_cloud_tags`** you want to define.

   The **Value** is a comma-separated list of key/value pairs for tags that you want to define. Use an equals sign to assign tag values to tag keys. Tags are optional but enable visibility in the Prisma Cloud UI.
4. For image scan, enter the **Name** and **Value** for the Prisma Cloud Compute password.

   The default name is **`prisma_cloud_compute_pass`**. If you enter a different name, make sure to match this name in the config.yml file in the next step.

STEP 3 | Add the Prisma Cloud configuration file.

The Prisma Cloud configuration file supports various IaC scan features. To add this file, create a subdirectory and file **`.prismacloud/config.yml`** in the root folder of your project or repository branch. See Set Up Your Prisma Cloud Configuration File for IaC Scan for details. Note that this file is different from **`.circleci/config.yml`**, and subsequent references to config.yml in these steps indicate the **`.circleci/config.yml`** file.

STEP 4 | Add the Prisma Cloud orb to the config.yml.

1. Modify the config.yml to include the orb named prisma_cloud/devops_security for IaC and image scanning.

   Note that, through the orb, you can customize the IaC scan failure criteria and the vulnerability thresholds for image scanning based on your security needs.

   More details about the Prisma Cloud orb are available at Prisma Cloud Orb Quick Start Guide.

   The following table lists the parameters you can specify to customize the Prisma Cloud IaC scan job in your orb.

Parameter	Description	Req	Default	Type
access_key	Prisma Cloud access key	no	$prisma_cloud_access_key	String
secret_key	Prisma Cloud secret key	no	prisma_cloud_secret_key	Environment variable
prisma_cloud_api_url	Prisma Cloud server URL	no	$prisma_cloud_console_url	String
terraform_variable_filenames	Comma-separated list of file names containing Terraform variables	no	''	String
templates_directory_path	Directory path where IaC templates are stored. Note: The total size of the IaC templates in this directory cannot exceed 9 MB.	no	.	String
faiure_criteria_high_severity	Provides failure threshold for high severity security issues	no	0	Integer
failure_criteria_medium_severity	Provides failure threshold for medium severity security issues	no	0	Integer

Parameter	Description	Req	Default	Type
failure_criteria_low_severity	Provides failure threshold for low severity security issues	no	0	Integer
failure_criteria_operator	Provides operator for high, medium, low severity failure thresholds	no	or	String
tags	Comma-separated list of tags for your task. Used for visibility in Prisma Cloud UI. For IaC scan only. Optional.	no	''	String

The following table lists the parameters you can specify to customize the Prisma Cloud Compute container image scanning job in your orb.

Parameter	Description	Require	Default	Type
prisma_cloud_compute_user	The Prisma Cloud Compute user with the CI User role	no	$prisma_cloud_compute_user	String
prisma_cloud_compute_pass	The Prisma Cloud Compute user's password	no	prisma_cloud_compute_pass	Environment variable
prisma_cloud_compute_url	The base URL for the console -- e.g. http://console.<abc>.com:8083	no	$prisma_cloud_compute_url	String

Parameter	Description	Require	Default	Type
	-- without a trailing /			
workspace_name	Name of workspace to “docker save” the image-tar into so it can be scanned by orb	no	workspace	String
image_tar	The name of the image tar file stored in the workspace	no	image.tar	String
image	The name of the image to scan -- myimage or myorg/ myimage or myorg/ myimage:latest	yes		String

The following script is an example that shows you how to add the details required to set up both IaC and container image scanning.

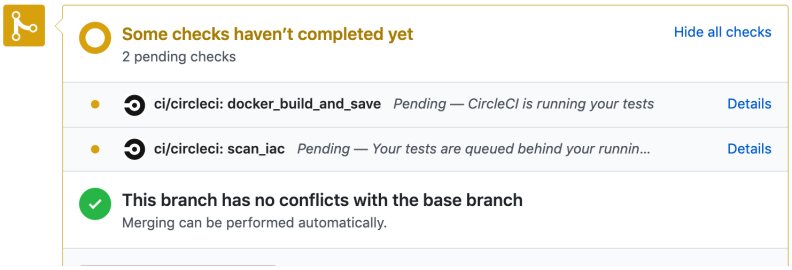
*Make sure that you have defined the secret key (**`prisma_cloud_secret_key`** for IaC scan) and the Prisma Cloud compute password (**`prisma_cloud_compute_pass`** for container image scanning), each as an environment variable.*

```
version: 2.1orbs:  scan: prisma_cloud/devops_security@2.0.0jobs:
 docker_safe_build:    executor: pcs/compute    steps:      - checkout
      - run: 'docker pull nginx'      - pcs/scan_image:
 prisma_cloud_compute_url: https://us-east1.cloud.twistlock.com/
console123          prisma_cloud_scan_image: nginxworkflows:
 scan:    jobs:      - pcs/scan_iac:          prisma_cloud_api_url:
 <prisma cloud api url>          prisma_cloud_repo_dir: ./scan
       prisma_cloud_asset_name: $CIRCLE_PROJECT_REPONAME
 prisma_cloud_tags: "svc:github.com"      - docker_safe_build
```

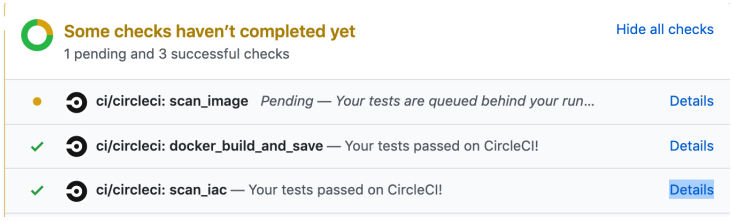
2. Check the scan results.

   After you update the config.yml, whenever a PR is created, the Prisma Cloud orb checks for any potential issues. The build is a Success or Failure depending on whether the number of the of issues detected is lower than or more than the specified threshold.

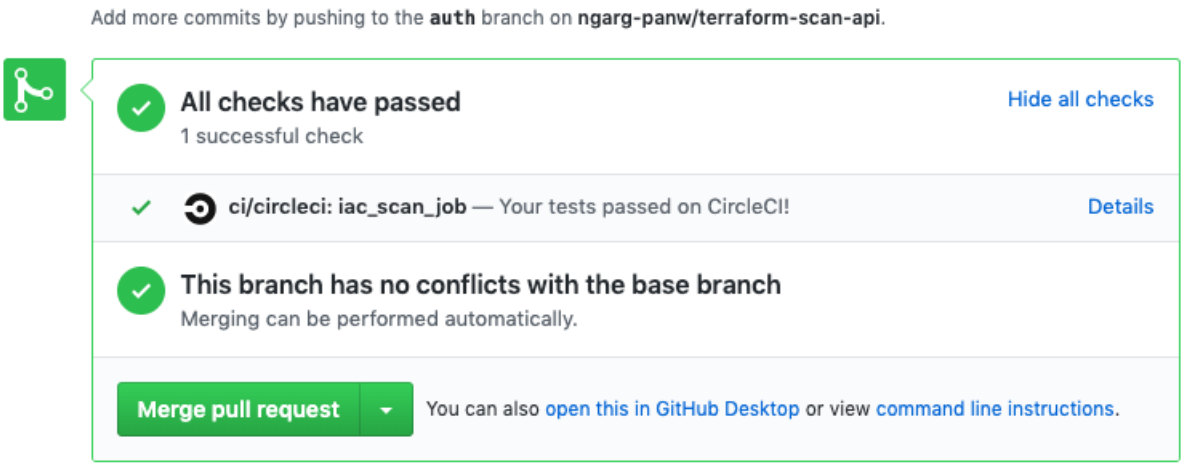
   When the scan starts, you can view the status:



In the following image you can view the status of the checks. The IaC scan reports as successful, while the image scan has completed the prerequisite check and is pending completion.



The ability to merge code is enabled only when the result is successful.



Click **Details** to view more information. When any of the checks are unsuccessful, the results are generated as a JUNIT report by default, so it shows up in the **TESTS** tab.

Dashboard > Project > Branch > Workflow > Job

All Pipelines > iac-tf12-dev > master > scan > pcs/scan_iac (37)

pcs/scan_iac Failed CONCURRENCY LIMIT

Duration / Finished	Queued	Executor	Branch	Commit	Author & Message
41s / 6d ago	32s	Docker Medium	master	2589701	Update config.yml

STEPS TESTS 5 ARTIFACTS

5 tests failed out of 5 in iac_scan

AWS S3 buckets are accessible to public - High

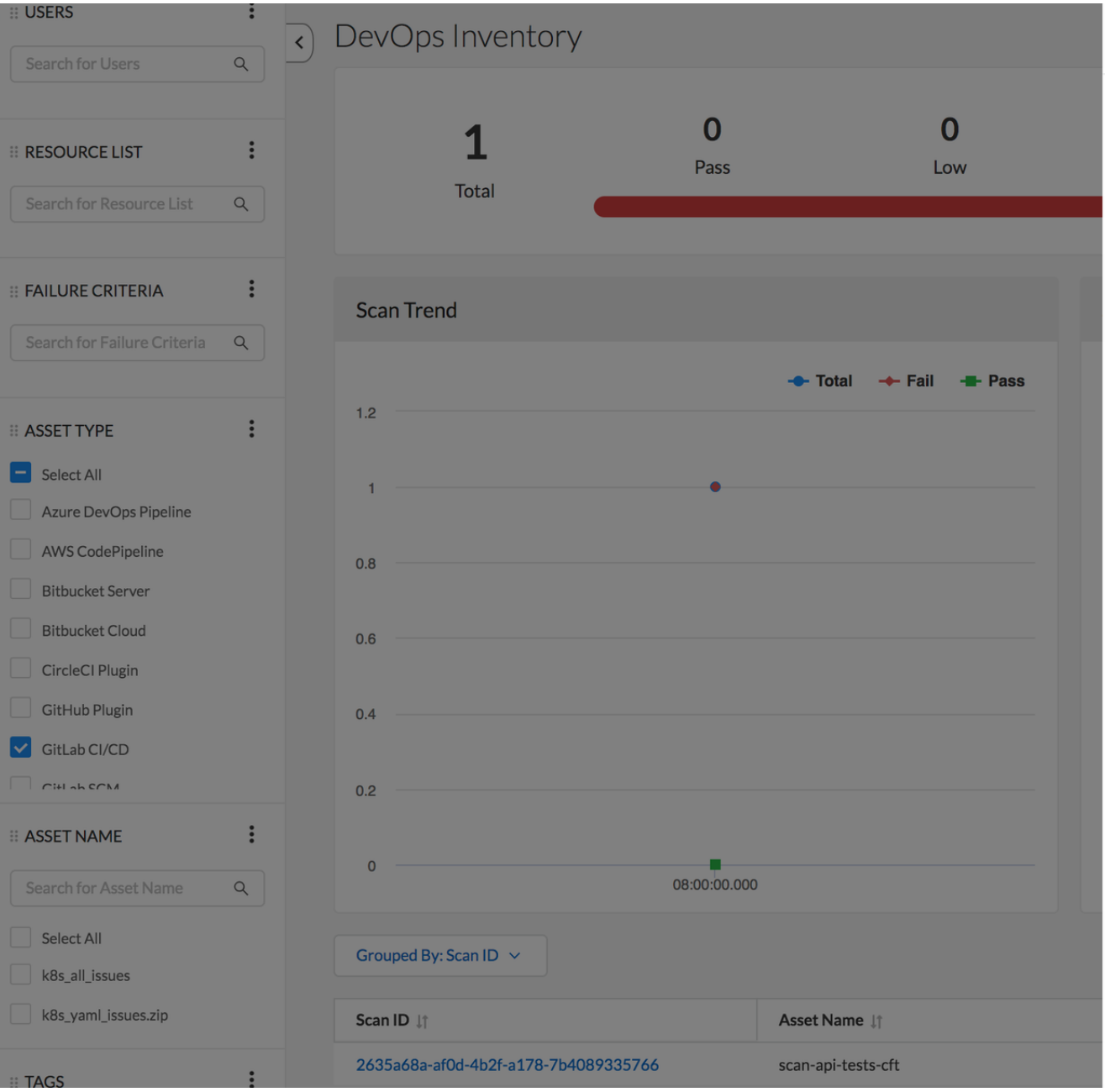
terraform12novariablemodule/main.tf


```
{
 "severity": "high",
 "name": "AWS S3 buckets are accessible to public",
 "rule": "$.resource.aws_s3_bucket exists and ($.resource.aws_s3_bucket[*].acl anyEqual public-read-write or $.resource.aws_s3_bucket[*].acl
public-read)",
 "desc": "This policy identifies S3 buckets which are publicly accessible. Amazon S3 allows customers to store or retrieve any type of conter
in the web. Often, customers have legitimate reasons to expose the S3 bucket to public, for example, to host website content. However, these
contain highly sensitive enterprise data which if left open to public may result in sensitive data leaks.",
 "systemDefault": false,
 "files": [
  "./terraform12novariablemodule/main.tf"
 ],
 "policyId": "630d3779-d932-4fbf-9cce-6e8d793c6916",
 "docUrl": "http://prisma.fyi/p71"
}
```

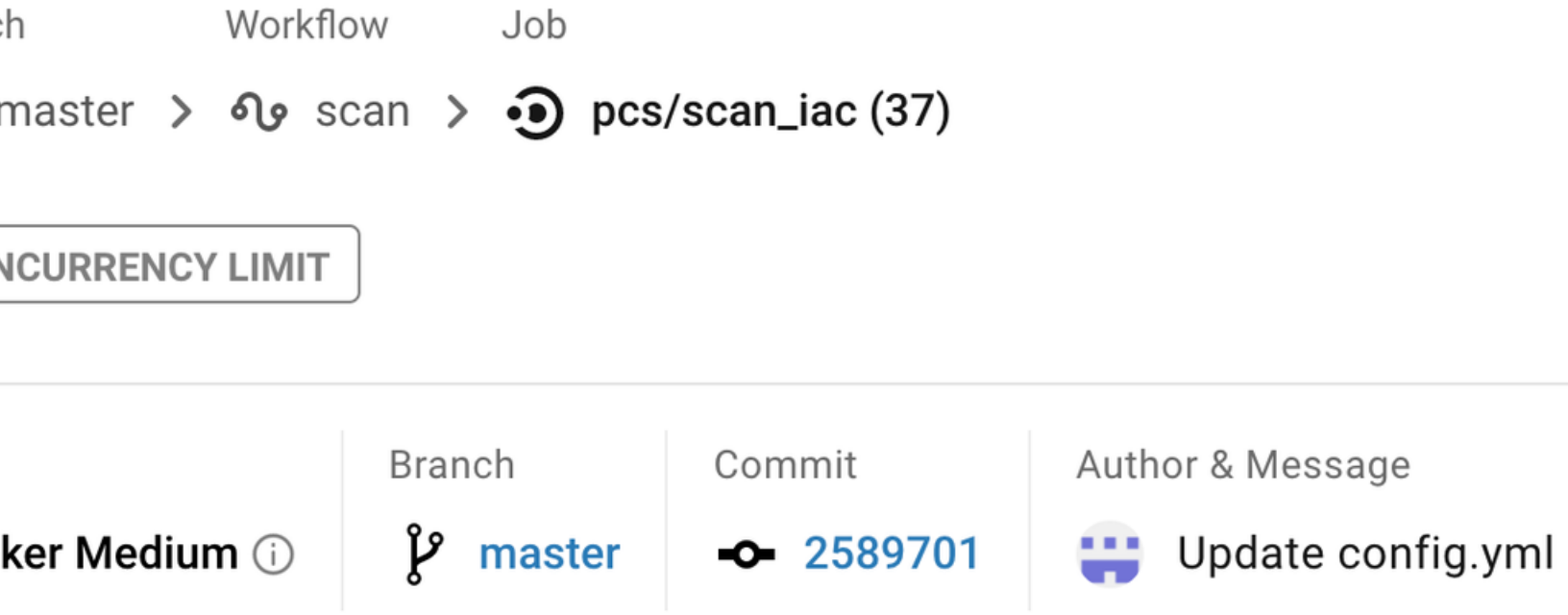

AWS S3 Object Versioning is disabled - Medium

terraform12novariablemodule/main.tf

The same information is also available in the Prisma Cloud DevOps Inventory page:



If *csv* is included in the report list parameter, then results.csv is uploaded to CircleCI. You can download the violation results from the ARTIFACTS tab.



Index	Severity	Name	Files
1	low	AWS IAM policy attached to users	main.tf
2	medium	AWS CloudTrail logs are not encrypted using Customer Master Keys	cloudtrail.tf.json;cloudtrail.tf
3	medium	AWS S3 object versioning is disabled	cloudtrail.tf.json;cloudtrail.tf
4	medium	AWS Access logging not enabled on S3 buckets	cloudtrail.tf.json;cloudtrail.tf
5	medium	AWS IAM password policy does not have a minimum of 14 characters	iampassword.tf
6	medium	AWS IAM password policy allows password reuse	iampassword.tf
7	medium	AWS VPC allows unauthorized peering	vpcpeering.tf
8	medium	AWS VPC NACL allow egress traffic from blocked ports	SG.tf
9	medium	AWS VPC NACL allows traffic from blocked ports	SG.tf

The following is an example of IaC scan results when the scan is successful and you have no detected security issues.

```
  adding: scan/ (stored 0%)
  adding: scan/eks.tf.json (deflated 63%)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100  1213    0    86  100  1127     84   1102  0:00:01  0:00:01 --:--:--  1102
Good job! Prisma Cloud did not detect any issues.
```

The following is an example of IaC scan results when the result was successful but with issues.

```
dex   Severity  Name                                                          Files
      --------  ----                                                          -----
      medium    AWS Access logging not enabled on S3 buckets                  cloudtrail.tf.json;cloudtrail.tf
      medium    AWS CloudTrail logs are not encrypted using Customer Master Keys  cloudtrail.tf.json;cloudtrail.tf
      medium    AWS IAM password policy allows password reuse                 iampassword.tf
      medium    AWS IAM password policy does not have a minimum of 14 characters  iampassword.tf
      medium    AWS S3 object versioning is disabled                          cloudtrail.tf.json;cloudtrail.tf
      medium    AWS VPC allows unauthorized peering                           vpcpeering.tf
      medium    AWS VPC NACL allow egress traffic from blocked ports          SG.tf
      medium    AWS VPC NACL allows traffic from blocked ports                SG.tf
      low       AWS IAM policy attached to users                              main.tf
isma Cloud IaC Scan has been successful as security issues count (high: "0", medium: "", low: "1") does not exceed the failure crit
m:2, low:3, operator:and)
```

The following is an example of IaC scan results that returned a failure because of the number and type of security issues it found.

```
ndex   Severity  Name                                                    Files
       --------  ----                                                    -----
       medium    AWS Access logging not enabled on S3 buckets            cloudtrail.tf.json;cloudtrail.tf
       medium    AWS CloudTrail logs are not encrypted using Customer Master Keys  cloudtrail.tf.json;cloudtrail.tf
       medium    AWS IAM password policy allows password reuse           iampassword.tf
       medium    AWS IAM password policy does not have a minimum of 14 characters  iampassword.tf
       medium    AWS S3 object versioning is disabled                    cloudtrail.tf.json;cloudtrail.tf
       medium    AWS VPC allows unauthorized peering                     vpcpeering.tf
       medium    AWS VPC NACL allow egress traffic from blocked ports    SG.tf
       medium    AWS VPC NACL allows traffic from blocked ports          SG.tf
0      low       AWS IAM policy attached to users                        main.tf
risma Cloud IaC scan has failed as security issues count (high: "0", medium: "8", low: "1") meets or exceeds the failure criteria (
ow:3, operator:or)

xited with code exit status 1
```

The following shows an example of container image scan results that failed because the IaC scan found security issues in the image.

```
ilities
-------
        ID                  CVE              Package          Version             Severity    Status    CVSS
        --                  ---              -------          -------             --------    ------    ----
test    f7bb5701a33c0e57    CVE-2019-2201    libjpeg-turbo    1:1.5.2-2           critical    open      9.3
test    f7bb5701a33c0e57    CVE-2019-18224   libidn2          2.0.5-1             high        open      7.5
test    f7bb5701a33c0e57    CVE-2018-12886   gcc-8            8.3.0-6             high        open      8.1
test    f7bb5701a33c0e57    CVE-2019-17546   tiff             4.0.10-4            medium      open      6.8
test    f7bb5701a33c0e57    CVE-2019-12290   libidn2          2.0.5-1             medium      open      5
test    f7bb5701a33c0e57    CVE-2019-19126   glibc            2.28-10             low         open      2.1
test    f7bb5701a33c0e57    CVE-2017-16932   libxml2          2.9.4+dfsg1-7       low         open      7.5
test    f7bb5701a33c0e57    CVE-2018-14404   libxml2          2.9.4+dfsg1-7       low         open      7.5
test    f7bb5701a33c0e57    CVE-2017-18258   libxml2          2.9.4+dfsg1-7       low         open      6.5
test    f7bb5701a33c0e57    CVE-2018-14567   libxml2          2.9.4+dfsg1-7       low         open      6.5
test    f7bb5701a33c0e57    CVE-2019-19956   libxml2          2.9.4+dfsg1-7       low         open      5
test    f7bb5701a33c0e57    CVE-2017-17942   tiff             4.0.10-4            low         open      8.8
test    f7bb5701a33c0e57    CVE-2019-14973   tiff             4.0.10-4            low         open      6.5
test    f7bb5701a33c0e57    CVE-2016-9318    libxml2          2.9.4+dfsg1-7       low         open      7.8
test    f7bb5701a33c0e57    CVE-2019-14855   gnupg2           2.2.12-1+deb10u1    low         open      0
test    f7bb5701a33c0e57    CVE-2016-10228   glibc            2.28-10             low         open      5.9
test    f7bb5701a33c0e57    CVE-2019-18276   bash             5.0-4               low         open      10
test    f7bb5701a33c0e57    CVE-2019-17543   lz4              1.8.3-1             low         open      6.8
test    f7bb5701a33c0e57    CVE-2019-1551    openssl          1.1.1d-0+deb10u2    low         open      5
test    f7bb5701a33c0e57    CVE-2019-15847   gcc-8            8.3.0-6             low         open      7.5
test    f7bb5701a33c0e57    CVE-2016-2781    coreutils        8.30-3              low         open      6.5
test    f7bb5701a33c0e57    CVE-2019-17371   libpng1.6        1.6.36-6            low         open      4.3
test    f7bb5701a33c0e57    CVE-2019-13627   libgcrypt20      1.8.4-5             low         open      6.8
test    f7bb5701a33c0e57    CVE-2019-12904   libgcrypt20      1.8.4-5             low         open      5.9
test    f7bb5701a33c0e57    CVE-2019-3844    systemd          241-7~deb10u2       low         open      7.8
test    f7bb5701a33c0e57    CVE-2019-3843    systemd          241-7~deb10u2       low         open      7.8
test    f7bb5701a33c0e57    CVE-2018-20839   systemd          241-7~deb10u2       low         open      9.8
test    f7bb5701a33c0e57    CVE-2018-21029   systemd          241-7~deb10u2       low         open      7.5
test    f7bb5701a33c0e57    CVE-2018-7169    shadow           1:4.5-1.1           low         open      5.3
test    f7bb5701a33c0e57    CVE-2018-1152    libjpeg-turbo    1:1.5.2-2           low         open      6.5
test    f7bb5701a33c0e57    CVE-2018-14498   libjpeg-turbo    1:1.5.2-2           low         open      6.5
test    f7bb5701a33c0e57    CVE-2013-0337    nginx            1.17.6-1~buster     low         open      7.5
ility threshold check results: FAIL

ce
--
        ID                  Severity    Description
        --                  --------    -----------
test    f7bb5701a33c0e57    high        (CIS_Docker_CE_v1.1.0 - 4.1) Image should be created with a non-root user

ith code exit status 1
```

Use the Prisma Cloud Plugin for IntelliJ IDEA

With the Prisma Cloud Enterprise edition license, you can install the IntelliJ IDEA plugin that enables you to check Infrastructure-as-Code (IaC) templates and deployment files against Prisma Cloud IaC policies, within your integrated development environment (IDE). The following steps show how simple it is to install and check your IaC templates and files for potential security misconfigurations.

If you were using version 1.2 or earlier of the Prisma Cloud plugin for IntelliJ IDEA, you must update the plugin to version 3.0.0 or later. Use the instructions in this section to set up the plugin with the updated Prisma Cloud API URL and enter the credentials that are required to authenticate to Prisma Cloud.

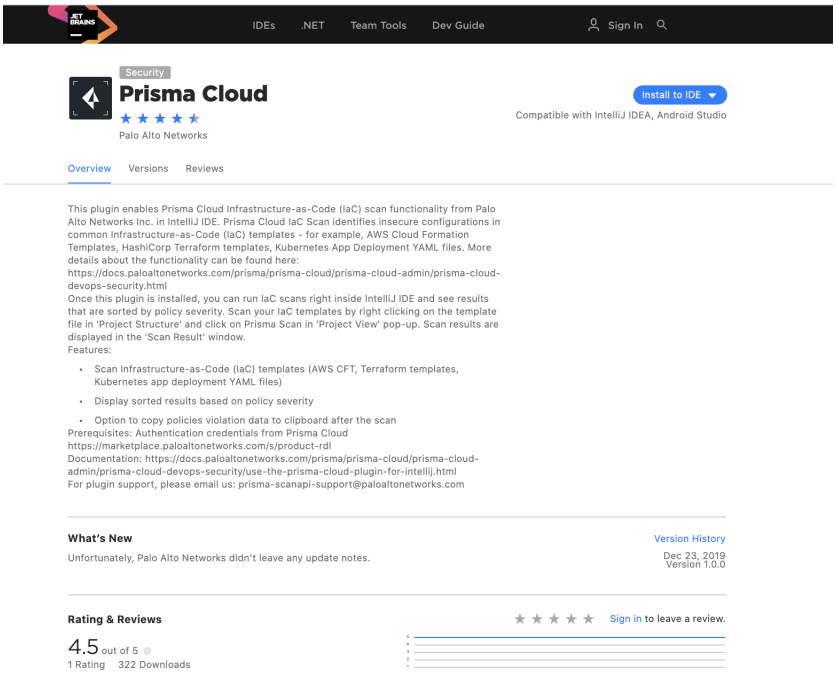
1. Install the Prisma Cloud Plugin for IntelliJ
2. Configure the Prisma Cloud Plugin for IntelliJ
3. Scan Using the Prisma Cloud Plugin for IntelliJ

Install the Prisma Cloud Plugin for IntelliJ

The Prisma Cloud plugin supports IntelliJ IDEA version 2016.2 and above.

STEP 1 | In IntelliJ IDEA, select **File** > **Settings** > **Plugins** (on macOS, select **Preferences** > **Plugins**).

STEP 2 | On the **Plugins** page, select **Marketplace** and search for **Prisma Cloud**.



STEP 3 | **Install** the plugin.

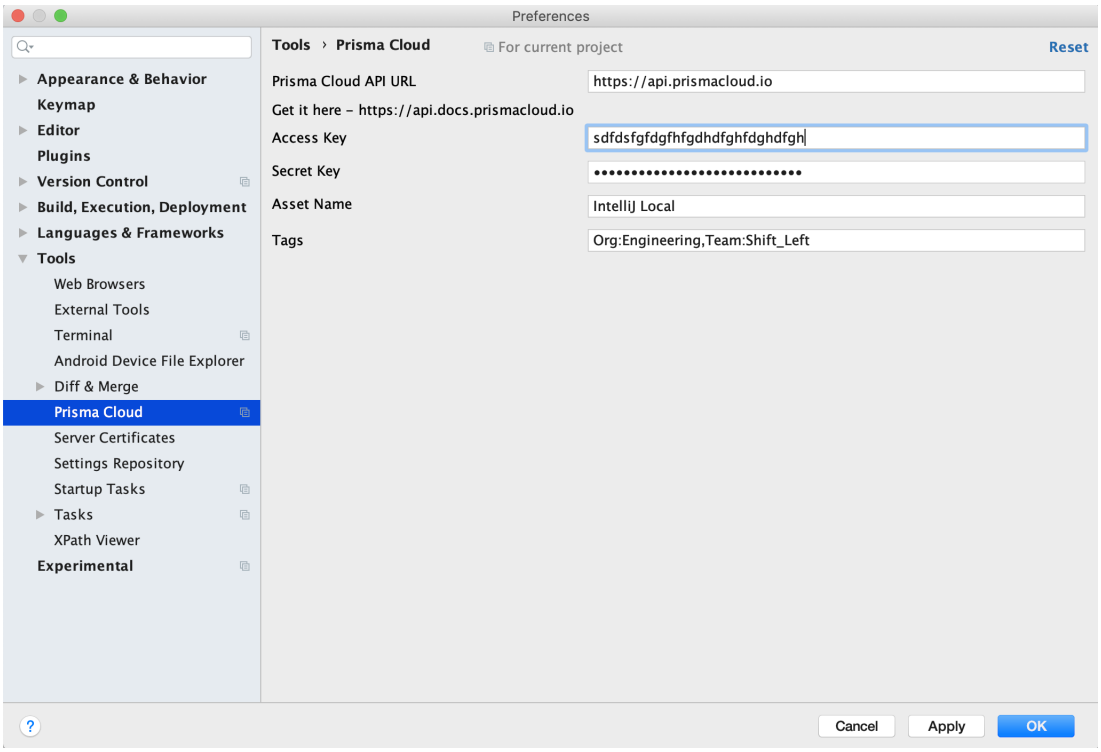
Restart the IDE and verify that the Prisma Cloud plugin displays in the list of **Installed** plugins.

Configure the Prisma Cloud Plugin for IntelliJ

After you install the plugin, you must provide the Prisma Cloud API URL and Prisma Cloud access key information to authenticate and start scanning you IaC templates. If your access key changes, you'll need to update the access key information in this configuration.

STEP 1 | In IntelliJ IDEA, select **Settings** > **Tools** > **Prisma Cloud Plugin** (on macOS, select **Preferences** > **Tools** > **Prisma Cloud Plugin**).

STEP 2 | Enter the following information to set up the plugin.



- **Prisma Cloud API URL**.

  The URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. The tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io. Replace **`app`** in the URL with **`api`** and enter it here. Refer to the Prisma Cloud REST API Reference, which is accessible from the Help Center within the Prisma Cloud web interface for more details.
- **Access Key**.

  The access key enables programmatic access to Prisma Cloud. If you do not have a key, you must Create and Manage Access Keys.
- **Secret Key**.

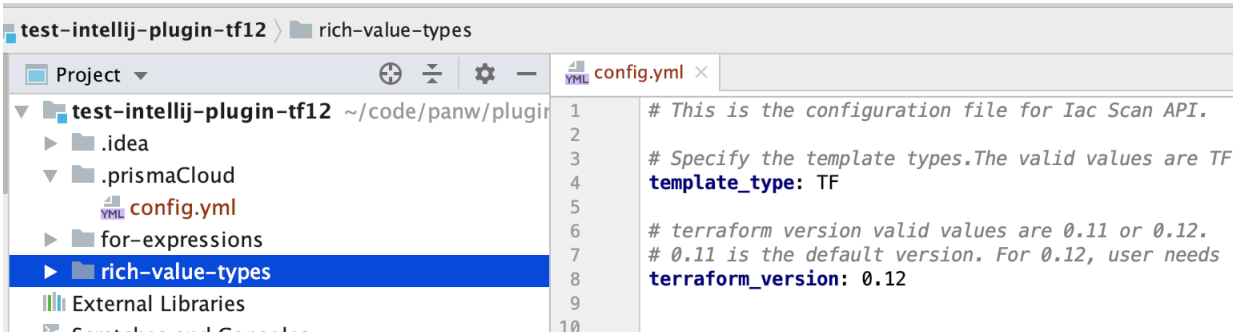
  You should have saved this secret key when you generated it. You cannot view it on the Prisma Cloud web interface.
- **Asset Name**

  Enter an asset name to identify the repository you want to scan.
- **Tags**.

  Define tags to organize the templates that are scanned with this service connection, for visibility on Prisma Cloud.

STEP 3 | Add the Prisma Cloud configuration file.

The Prisma Cloud configuration file supports IaC scanning of complex module structures and variable formats. To add this file, create a subdirectory and file .prismaCloud/config.yml in the root folder of your project or repository branch. See Set Up Your Prisma Cloud Configuration File for IaC Scan for details.



Scan Using the Prisma Cloud Plugin for IntelliJ

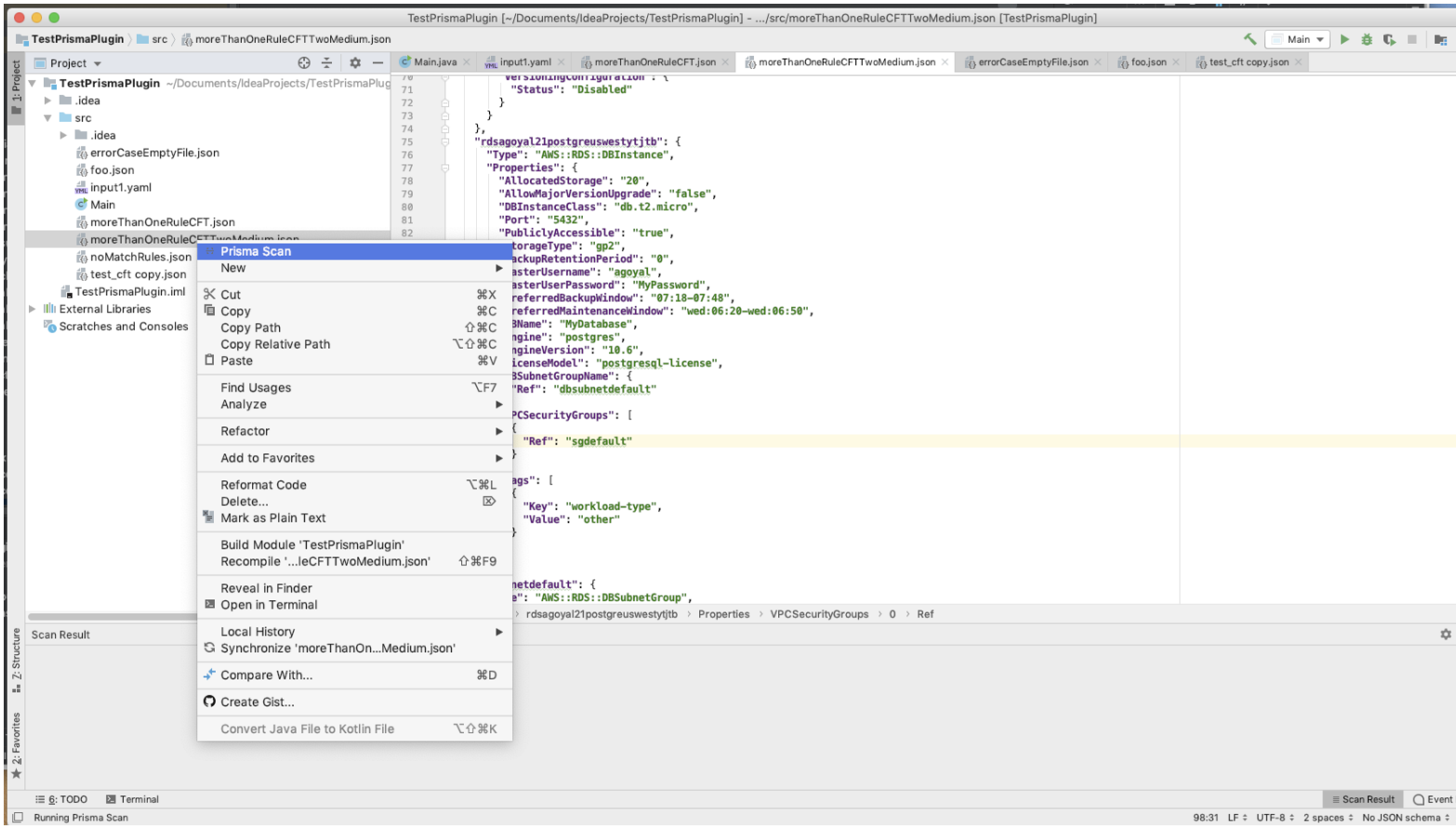
Now, you are ready to scan your templates and view the results before you check it in to the repository or pipeline.

You must have a Prisma Cloud Enterprise edition license and valid credentials to scan IaC templates.

STEP 1 | Scan the files for insecure configurations.

Right-click to scan your template file or folder in the IDEA **Project** window and select **Prisma Scan**.

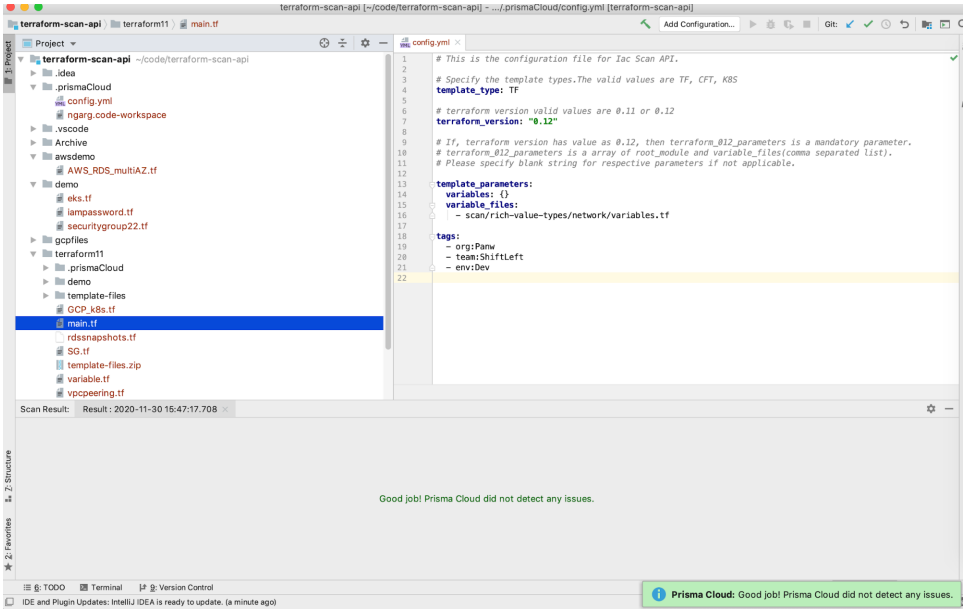
If you are using Helm Charts then you must right-click on the directory containing Chart.yml; do not click on Chart.yml itself.



STEP 2 | View the results of the scan in the **Scan Result** tool window.

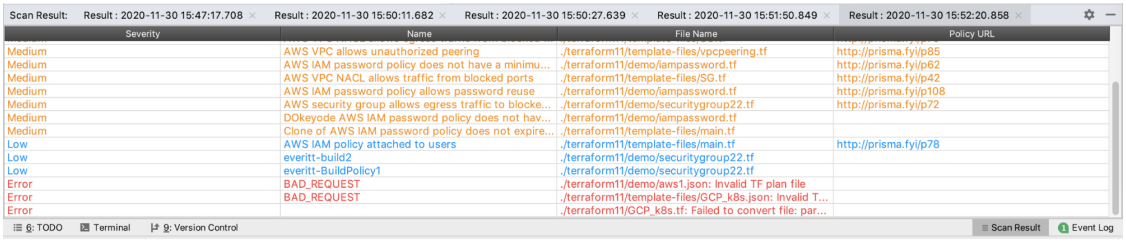
The title of the Scan Result window includes the date and time of the scan. For each scan, a new scan result window is added. The tab situated farthest to the right displays the results of the latest scan.

If the scan detects no potential issues, the message displays as follows:



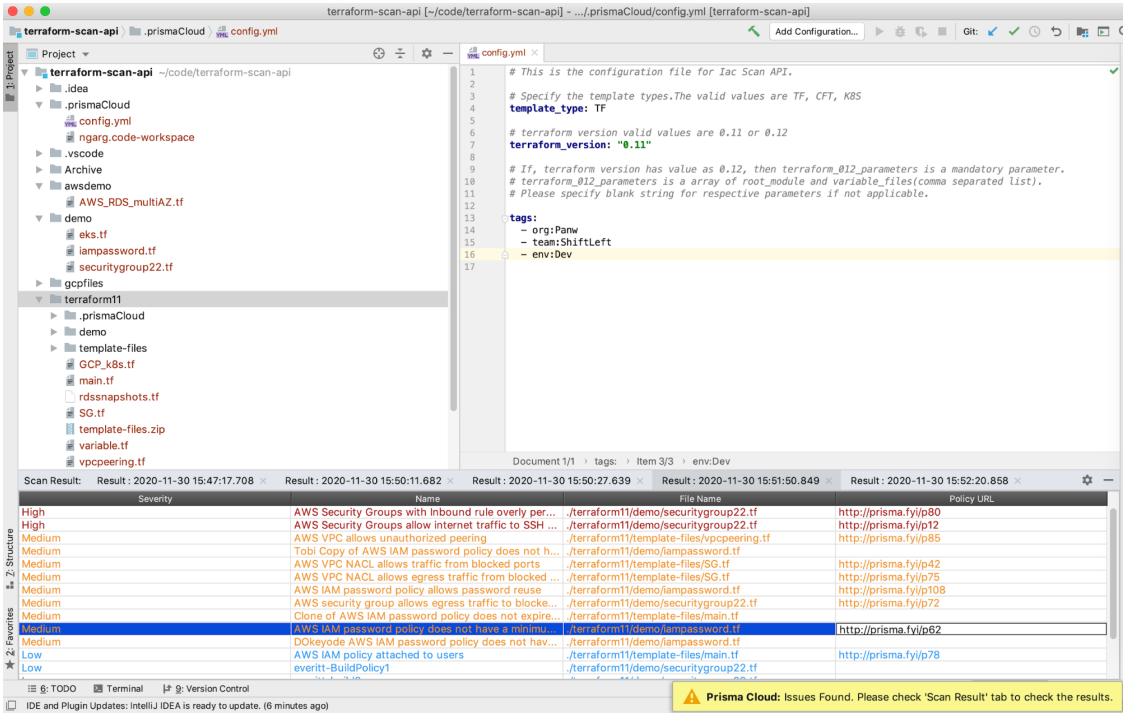
If the scan detects any policy violations, the scan result displays the following details for each violation.

- Severity: Low, Medium or High.
- Name of the violation.
- File names in which violations are found.
- Policy URL for details.



The end of the table shows the files which were not scanned with the corresponding error message. By default, the results are sorted by severity. You can also sort the Scan Result using the policy name.

STEP 3 | The following examples show scan results for templates in which vulnerabilities were found.



Use the Prisma Cloud App for GitHub

This Prisma Cloud app for GitHub enables you to scan IaC templates to check them against security policies when you open a pull request. For each pull request, you can define the pass criteria and view the scan results directly on GitHub. When the defined criteria are not met, the pull request fails and you can view all the checks that failed. In addition, the Prisma Cloud app creates an issue and adds the scan results as comments, so that you can fix all the issues reported before the changes are merged to the repository.

Use this app for scanning files in a private GitHub repository that has enabled restricted access. Be sure to create a read-only role on Prisma Cloud and to generate a secret key and access key for a user. You will need to provide these credentials to authenticate to Prisma Cloud for API access for the scanning capabilities.

1. Set up the Prisma Cloud App Files for GitHub
2. Install the Prisma Cloud App for GitHub

Recent versions of the app capture the Prisma Cloud credentials as part of the installation process and no longer require the credentials to be hard-coded in configuration file .github/prisma-cloud-config.yml. Existing customers should remove the credentials from this file after the app upgrade.

Set up the Prisma Cloud App Files for GitHub

To set up for IaC scans for a repository, you need to create IaC scan configuration files. These files enable you to control the behavior of your scans to meet your needs for that repository. For example, depending on the thresholds you defined in these files, the Prisma Cloud app will perform checks that allow or fail requests to merge or commit changes.

Creating these files before you install the Prisma Cloud app for GitHub enables the app installation itself to run a full IaC scan of selected repositories as part of the installation.

The two new files are:

- The Prisma Cloud configuration file .prismaCloud/config.yml

  This file identifies the templates types you wish to scan.
- .github/prisma-cloud-config.yml

  This file includes the criteria that defines whether or not you allow the commit for the pull request.

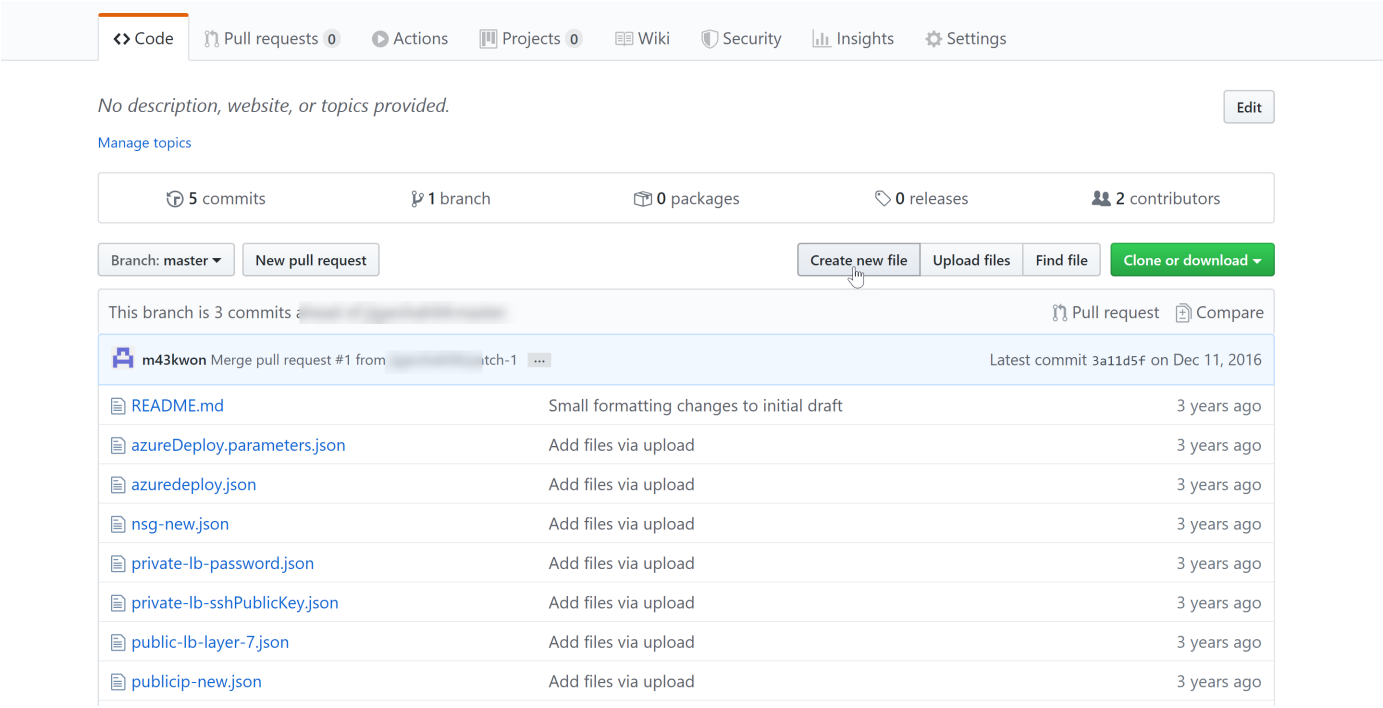
STEP 1 | Set Up Your Prisma Cloud Configuration File for IaC Scan.

Create the .prismaCloud/config.yml in the root directory of your repository branch. This file is required, and it must include the template type, version, and the template specific parameters and tags you use in your environment.

STEP 2 | Create the prisma-cloud-config.yml file to support the ability to scan IaC templates.

1. Select **Create new file**.

   Add a new folder called .github, and name the file prisma-cloud-config.yml. The path should be ***<your repository name>/.github/prisma-cloud-config.yml***.



2. Copy the template for this new file.

   Copy and paste the following contents into .github/prisma-cloud-config.yml.

```
# Please update with the respective environment values and commit
# to master branch under the .github folder before performing scans

# Define the failure criteria for creating checks. If the criteria
# matches a check will be created. The template for the checks can
# becustomized in the "/.github/prisma-template-for-scan-results"
# file.
failure_criteria_for_creating_checks:
  high: 1
  medium: 1
  low: 1
  operator: or

# Define the failure criteria for creating issues. If the criteria
# matches an issue will be created. The template for issues can be
# customized in the "/.github/prisma-template-for-scan-results"
# file.
failure_criteria_for_creating_issues:
  high: 1
  medium: 1
  low: 1
  operator: or

# Define github asset name
github_asset_name: "Github Asset Dev"

# Define tags
tags:
- phase:testing
- env:QA
```

3. Define the parameter values in prisma-cloud-config.yml.

   The parameters in prisma-cloud-config.yml define the failure criteria for pull requests. You can set **`failure_criteria_for_creating_checks`** to define the number and severity of security policy check failures that need to occur to trigger a merge request failure. The syntax for the **`failure_criteria_for_creating_checks`** value is as follows.

```
high: x
medium: y
low: z
operator: op
```

   In the syntax above, x is a count of high-severity policy check failure, y is a count of medium-severity policy check failures, and z is a count of low-severity policy check failures. The **`operator`** value determines what combination of High/Medium/Low counts should result in a merge request failure. The default for each count is 0. The value for operator, op, can be either OR or AND. The default is OR. Some examples of settings for **`failure_critieria_for_creatings_checks`** are as follows.

   - The setting below would result in a failed merge request security check for any detected policy check failure

```
high: 0
medium: 0
low: 0
operator: OR
```

   - The setting below would result in merge requests never failing a security check.

```
high: 1000
medium: 1000
low: 1000
operator: AND
```

   You can also use failure_criteria_for_creating_issues to define the number and severity of security policy check failures that need to occur to trigger creation of a GitHub issue, during a pull request. The syntax of the variable value is the same as that for failure_criteria_for_creating_checks. The value includes **`high`**, **`medium`**, and **`low`** counts and includes an **`operator`** whose possible values are **`AND`** and **`OR`**.

   Prisma Cloud uses the asset name to track results. Some example names are creditapp_server and ConsumerBU_server.

   Prisma Cloud tags enable visibility on the Prisma Cloud administrator console.

Install the Prisma Cloud App for GitHub

You must set up the app to authenticate to Prisma Cloud, and you can optionally customize the scan settings.

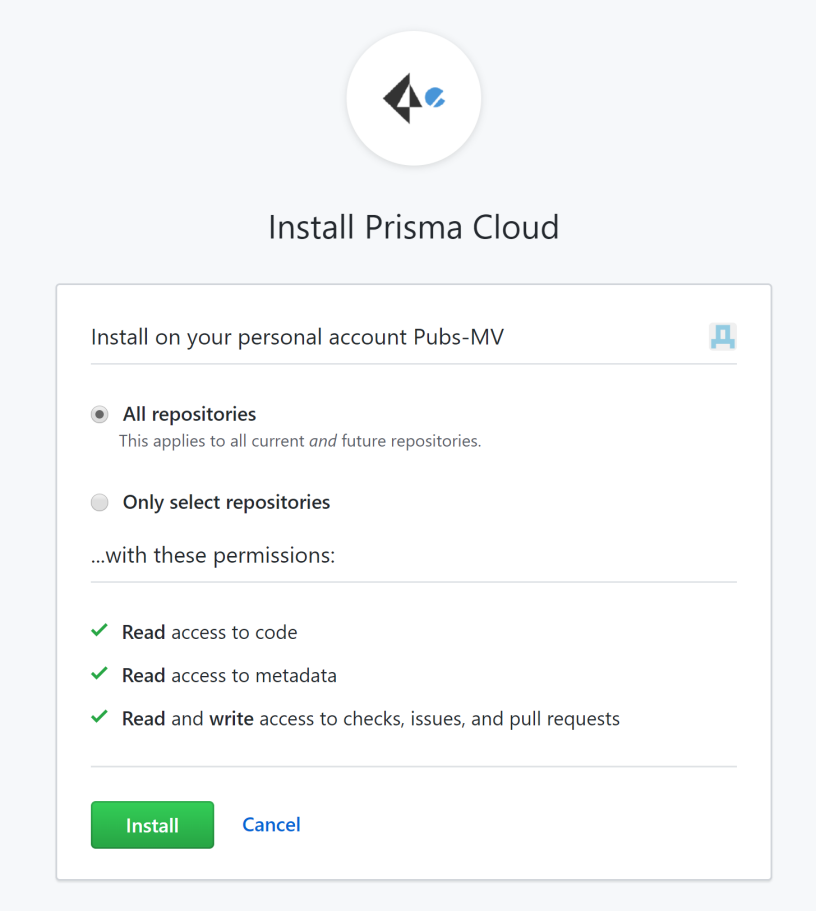
STEP 1 | Search for Prisma Cloud on the GitHub marketplace.

STEP 2 | Select **Settings** > **Integrations &services** > **Add service** and add **Prisma Cloud**.

This app requires the following permissions:

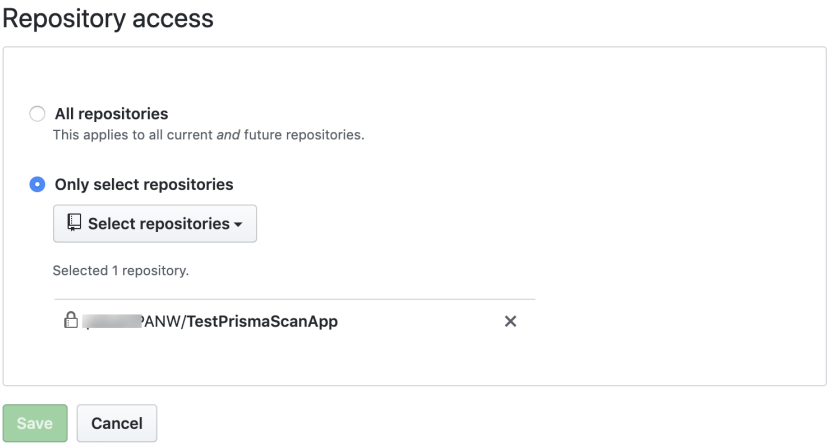
- Read access to code, to perform scan on template files.

- Read/write access to check for issues and open pull requests.
- Read access to metadata.



STEP 3 | Specify where you want to install the app.

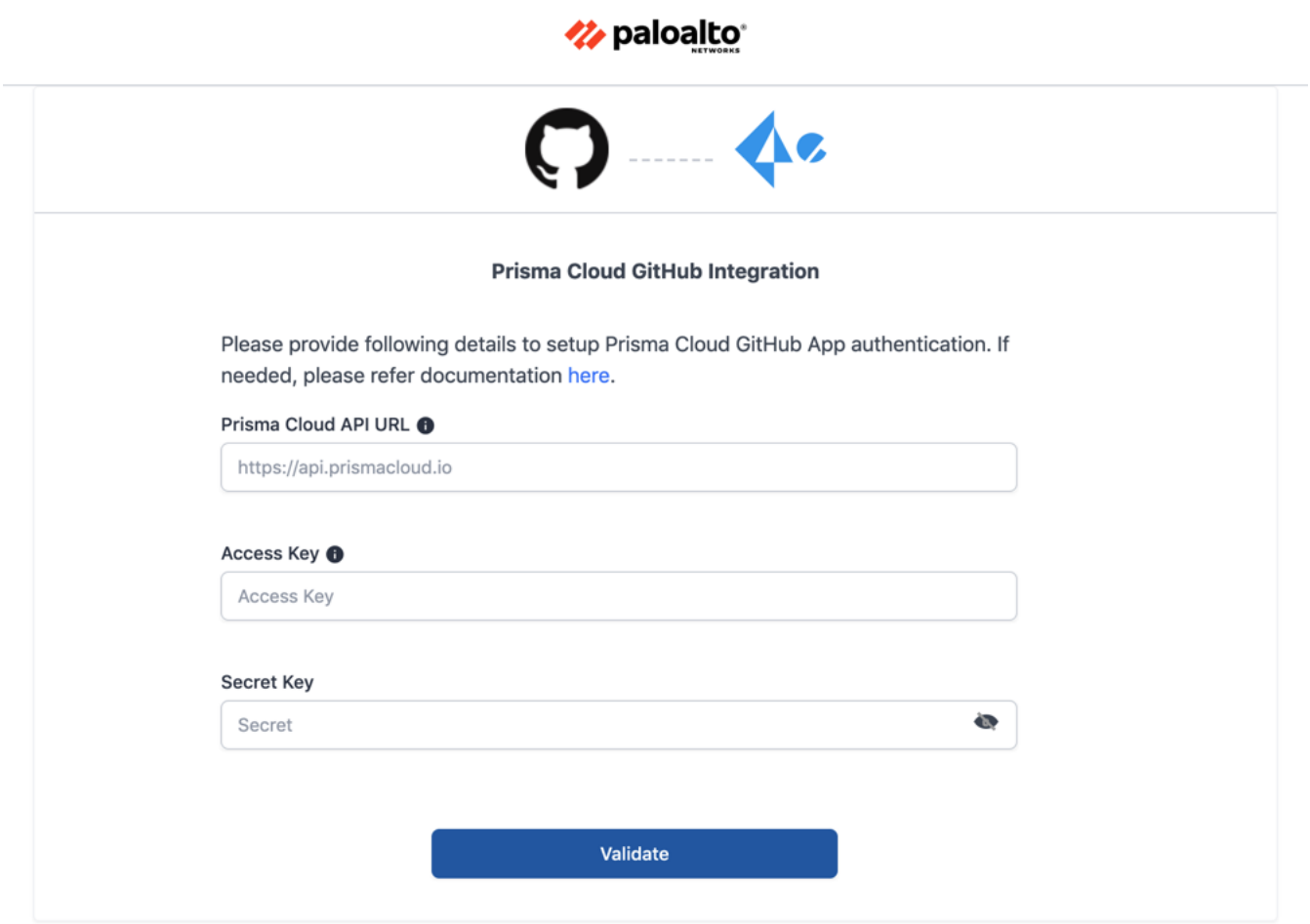
You can choose to install the Prisma Cloud app for GitHub on all repositories or only on selected repositories. You can change this setting later to include more repositories for scanning.



STEP 4 | Specify the Prisma Cloud API URL, Prisma Cloud access key ID, and corresponding secret key to use for the integration.

The Prisma Cloud API URL you specify depends on the region and cluster of your Prisma Cloud tenant. For example, if your Prisma Cloud admin console URL is https://app.prismacloud.io, then your Prisma Cloud API URL is https://api.prismacloud.io. See the Prisma Cloud REST API Reference for a list of Prisma Cloud API URLs.

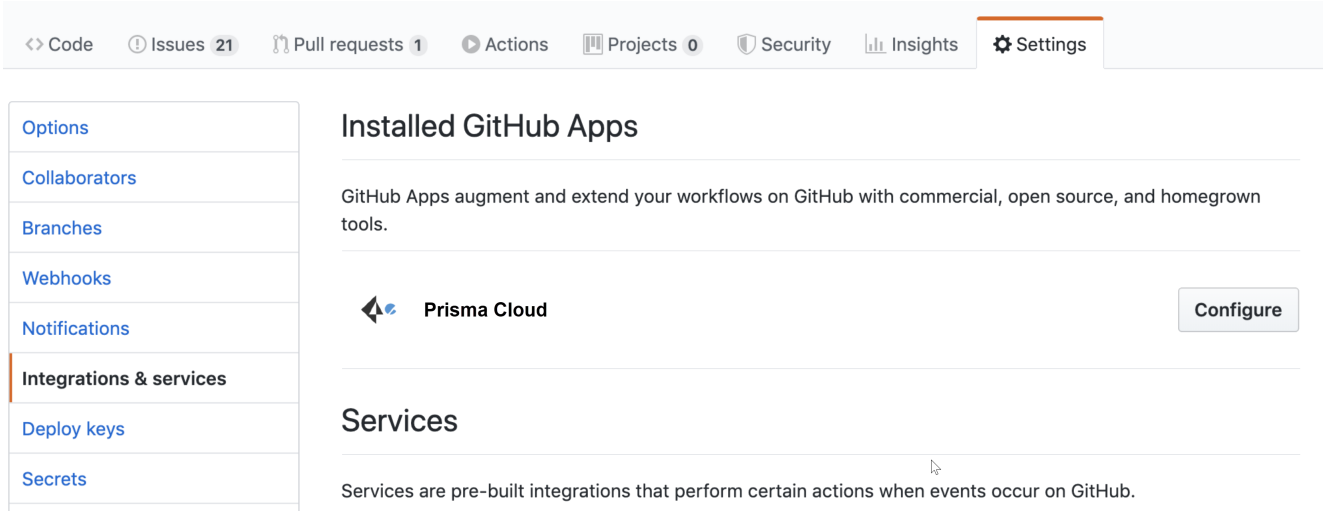
See Create and Manage Access Keys for details about Prisma Cloud access keys.



Once you've entered your settings, select **Validate**. If the settings are valid, a **Save** button appears, which enables you to save your settings.

STEP 5 | To add other repositories or to modify the configuration, you can select **Settings** > **Integrations & services** > **Prisma Cloud** to **Configure** the app.

Whenever you use this option to add repositories, the addition will result in an IaC scan of the repository if all the configuration files for the Prisma Cloud app are set up.



Use the Prisma Cloud Extension for GitLab

Use the Prisma Cloud extension to scan IaC templates in the build or release phase of the GitLab CI/CD pipeline or SCM when you create or merge a request. Container image or serverless function scanning is not available with this plugin, currently.

- Use the Prisma Cloud Extension for the GitLab CI/CD Pipeline
- Use the Prisma Cloud Extension for GitLab SCM

Use the Prisma Cloud Extension for the GitLab CI/CD Pipeline

You can use the Prisma Cloud extension for GitLab CI/CD to scan IaC templates to check against Prisma Cloud policies or to scan container images to check for vulnerabilities.

To scan IaC templates in the build or release phase of the GitLab CI/CD pipeline, you need to configure the Prisma Cloud extension. To start, include the Prisma Cloud IaC extension template file in your .gitlab-ci.yml file, then configure the extension using GitLab variables and the config.yml file. When you use this extension to then scan your templates in the pipeline, the failure thresholds you specify are used to pass or fail the check based on the severity of the security issues that are detected. When the scan is successful, the code can be merged. If the scan is unsuccessful, the security issues must be fixed in order to merge code changes.

The list of inputs that are required for scanning IaC templates in the build or release phase of the GitLab CI/CD pipeline are:

- Connection settings as environment variables to enable communication between the Prisma Cloud API server and your GitLab repository.
- The .gitlab-ci.yml at the root level in your repository.

  For any commit or push to your repository, this file start jobs on GitLab runners according to the contents of the file. You must have a shared runner or a project-specific/custom runner for the job to run successfully.
- config.yml file at the root-level within the project under the .prismaCloud directory.

  The path for this file must be .prismaCloud/config.yml. Prisma Cloud requires this configuration file to learn about your IaC module structure, runtime variables, and tags so that it can scan the IaC templates in your repository.

To scan container images for vulnerabilities, the way to set up the Prisma Cloud extension is very similar to IaC scan. You need to first connect to Prisma Cloud Compute using environment variables, and then the container image scanning is available as a command that you invoke through a pipeline job.

The list of inputs that are required for scanning container images are:

- Connection settings as environment variables to enable communication between the Prisma Cloud Compute console and your GitLab repository.
- The .gitlab-ci.yml at the root level in your repository.
- The image to scan.

  You can either provide the image details as an environment variables or directly scan an image using a twistcli command in the pipeline job.

To set up the Prisma Cloud GitLab extension:

- Configure the Prisma Cloud Extension for GitLab CI/CD
- Set Up a Custom GitLab Pipeline Job for IaC Scan

- Set Up a Custom GitLab Pipeline Job for Container Image Scan

If you have been using the Prisma Cloud GitLab extension v1, there are no updates to the environment variables. However, the .gitlab-ci.yml file has been simplified as described in Set Up a Custom GitLab Pipeline Job for IaC Scan.

Configure the Prisma Cloud Extension for GitLab CI/CD

The table below summarizes the environment variables you must configure for your GitLab project.

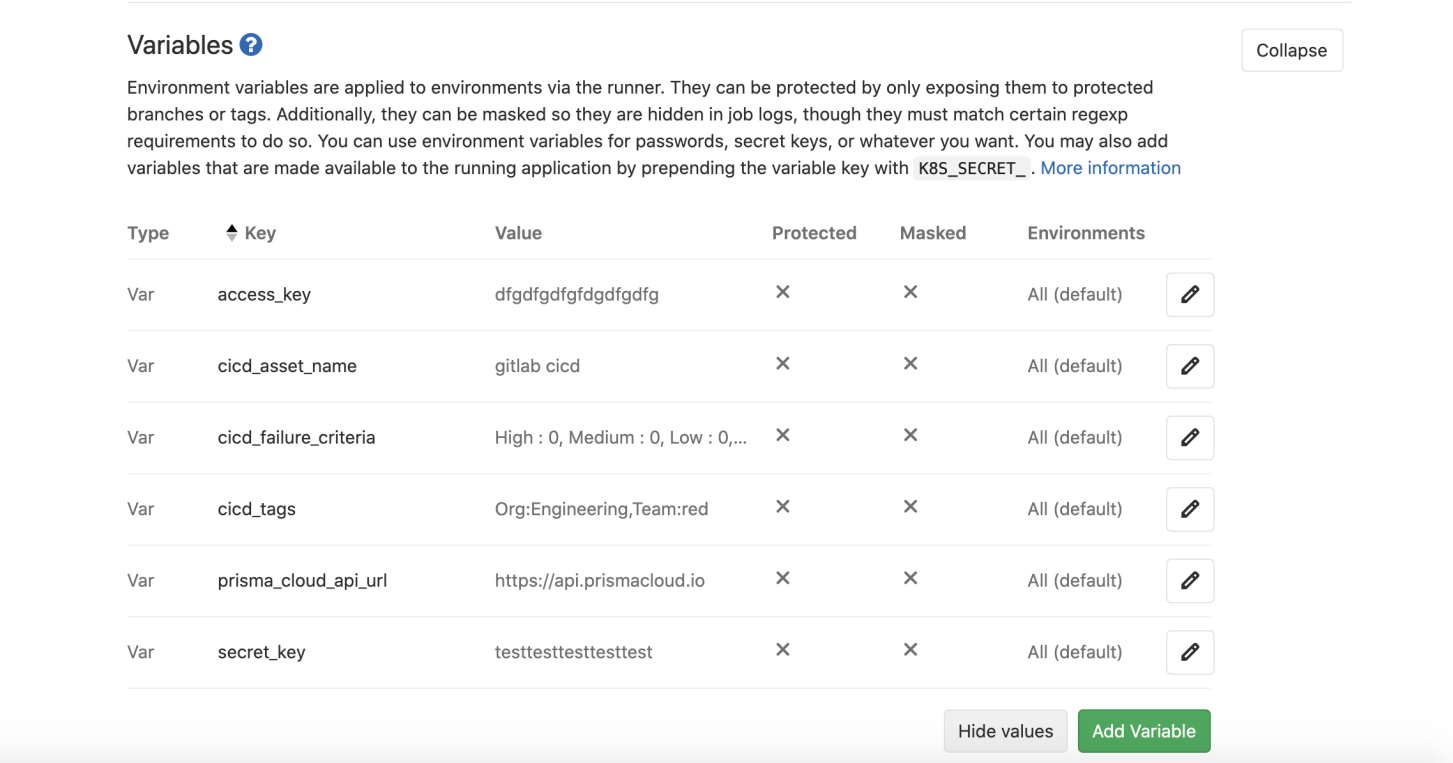
For	Key	Description
IaC Template Scan	prisma_cloud_api_url	Prisma Cloud base API URL. The API URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. If the tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io, replace **`app`** in the URL with **`api`**. Refer to the Prisma Cloud REST API Reference, for more details.
	prisma_cloud_access_key	Prisma Cloud access key for API access. If you do not have a key, see Create and Manage Access Keys.
	prisma_cloud_secret_key	Secret key that corresponds to the Prisma Cloud access key
	prisma_cloud_cicd_asset_name	GitLab server name
	prisma_cloud_repo_dir	Template repository directory on the GitLab runner. You must have your IaC files in this directory. You cannot mix different template types (CFT, Terraform etc. in the single scan)
	prisma_cloud_cicd_failure_criteria	(Optional) String that defines criteria that should trigger a pipeline failure. Set the High : x, Medium : y, Low : z, Operator: O, where, x,y,z is the number of issues of each severity, and the operator is OR, AND. For example: • To fail the pipeline only when high severity issue is detected, **`High:1,Medium:1000,Low:1000,Operator:`**

For	Key	Description
		• To never fail the pipeline, **`High:1000,Medium:1000,Low:1000,Operato`** **`and`** *You do not have to define the failure threshold because the default failure criteria is set as follows—High: 1, Medium: 1, Low: 1, Operator: or, so any failure will trigger failure of the pipeline.*
	prisma_cloud_cicd_tags	(Optional) Prisma Cloud tags are different from GitLab tags or cloud tags that you may have included within your IaC templates. Prisma Cloud tags enable visibility on the Prisma Cloud administrator console. For example, **`prisma_cloud_cicd_tag:project:x,owner:mr`** *You cannot include a comma in the values. For example, the following is invalid:* `tags=key1:val1,val2` *key1 and val1 will be kept while val2 will be discarded.*
Container Image Scan	prisma_cloud_compute_url	Base URL for the Prisma Cloud Compute console. For example http://console<example>.com:8083.
	prisma_cloud_compute_username	Prisma Cloud Compute user with the CI User role. You must set up a user who belongs to the Build and Deploy Security permission group, see Create Prisma Cloud Roles.
	prisma_cloud_compute_password	Prisma Cloud Compute user password (for container image scan)

For	Key	Description
	prisma_cloud_scan_image	Target image ID, or name:tag to scan, or image tar file name. You can use regular expressions when specifying image names, such as nginx*.

STEP 1 | Set up the connection to the Prisma Cloud API.

1. Add the connection settings as environment variables to **Project** > **Settings** > **CICD** > **Variables**.



You can also provide the GitLab variables in the .gitlab-ci.yml under variables, instead of adding it in the project settings.

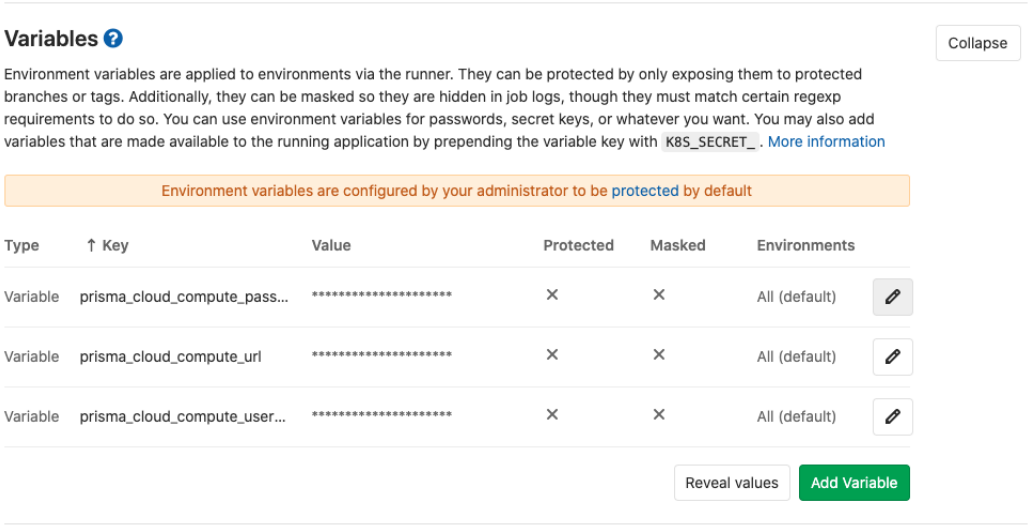
2. Set the Prisma Cloud API URL as the value for the **`prisma_cloud_api_url`** environment variable.
3. Add your Prisma Cloud access key as the value for the **`prisma_cloud_access_key`** environment variable.

   The access key enables programmatic access.
4. Add your GitLab server name as the value for the **`prisma_cloud_cicd_asset_name`** environment variable.

   On Prisma Cloud, the asset name is used to track results. Some examples names are - creditapp_server, ConsumerBU_server. etc

STEP 2 | Set up environment variables for container image scans.

Set up the following environment variables only if you want to run container image scans. As with the environment variables that support IaC scans, you navigate to **Project** > **Settings** > **CICD** > **Variables** to add new environment variables.



1. Add **`prisma_cloud_compute_url`**, whose value is the base URL for your Prisma Cloud Compute console.
2. Add **`prisma_cloud_compute_username`**, whose value is the Prisma Cloud Compute user with a CI user role.
3. Add **`prisma_cloud_compute_password`**, whose value is the password for the Prisma Cloud Compute user.
4. Add **`prisma_cloud_scan_image`**, whose value is the details to identify the image.

STEP 3 | Set Up Your Prisma Cloud Configuration File for IaC Scan.

Create the **.prismaCloud/config.yml** file and add it to the root directory of your repository branch. The file is required, and it must include the template type, version, and the template specific parameters and tags you use in your environment.

Set Up a Custom GitLab Pipeline Job for IaC Scan

STEP 1 | Add the Prisma Cloud GitLab extension to the GitLab CI configuration.

The GitLab CI configuration is stored in the .gitlab-ci.yml file. Add the following line to this file—

```
include:
  - remote: 'https://gitlab.com/prismacloud-public/shift-left/extension/-/raw/master/.pcs.gitlab-ci.yml'
```

Refer to the GitLab documentation to learn about the gitlab-ci.yml file.

The IaC templates in this directory must not exceed the 300MB size limit.

STEP 2 | Extend pipeline job from .pcs_iac_scan.

```
prisma-cloud-scan:
  stage: build
  extends:
    - .pcs_iac_scan
```

STEP 3 | Modify IaC scan variable values.

You can define the variable again in the gitlab-ci.yml file to override the value for any variable specified in the project settings.

```
prisma-cloud-scan:
  stage: build
  extends:
    - .pcs_iac_scan
  variables:
    prisma_cloud_asset_name: ${CI_PROJECT_NAME}
    prisma_cloud_cicd_tags: "project:scan-api-test"
    prisma_cloud_repo_dir: ${CI_PROJECT_DIR}/tests/templateFiles/k8s
```

A sample gitlab.ci.yml is show below:

```
include:
  - remote: 'https://gitlab.com/prismacloud-public/shift-left/extension/-/
raw/master/.pcs.gitlab-ci.yml'

stages:
  - build

prisma-cloud-scan:
  stage: build
  extends: .pcs_iac_scan
  variables:
    prisma_cloud_api_url: https://api.prismacloud.io
    prisma_cloud_access_key: ${access_key_app2slqa}
    prisma_cloud_secret_key: ${secret_key_app2slqa}
    prisma_cloud_asset_name: ${CI_PROJECT_NAME}
    prisma_cloud_cicd_tags: "project:scan-api-test"
    prisma_cloud_repo_dir: ${CI_PROJECT_DIR}/tests/templateFiles/k8s
```

STEP 4 | View IaC scan results.

The Prisma Cloud IaC scan uses the failure criteria you defined in the **`prisma_cloud_cicd_failure_criteria`** environment variable to pass or fail a scan. When it detects a security issue, it generates an artifact. The scan report is in Junit compatible format and you can view the results on your GitLab pipeline and on the DevOps Inventory on Prisma Cloud.

- View the scan results on the **tests** tab on your GitLab pipeline.

  List of job results.

Pipeline DAG Jobs 4 Failed Jobs 2 **Tests 78**

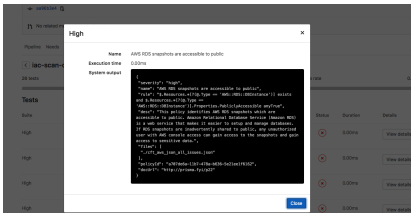
Summary

78 tests | 77 failures | 1 errors | 0% success rate | 0.00ms

Jobs

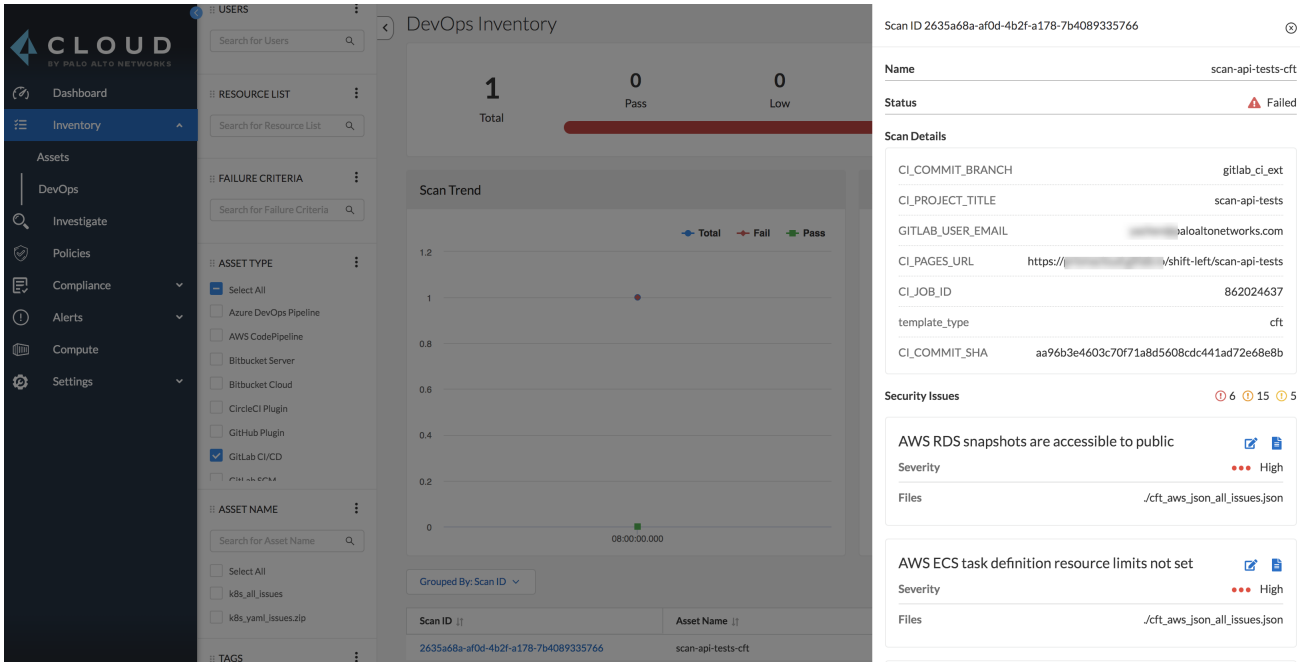
Job	Duration	Failed	Errors	Skipped	Passed	Total
iac-scan-cft	0.00ms	26	0	0	0	26
iac-scan-tf	0.00ms	0	0	0	0	0
iac-scan-k8s	0.00ms	11	0	0	0	11
image-scan	0.00ms	40	1	0	0	41

- IaC Scan result with details for a specific job.



- View the Prisma Cloud IaC scan results on the Prisma Cloud administrator console.

  Inventory > **DevOps**, select the job and view the details.



Set Up a Custom GitLab Pipeline Job for Container Image Scan

You can add container image scans to the extension you have already configured for IaC scans, or you can configure the extension to perform just container image scans.

STEP 1 | Add the Prisma Cloud GitLab extension to the GitLab CI configuration.

The GitLab CI configuration is stored in the .gitlab-ci.yml file. Add the following line to this file—

```
include:
  - remote: 'https://gitlab.com/prismacloud-public/shift-left/extension/-/
raw/master/.pcs.gitlab-ci.yml'
```

STEP 2 | Extend pipeline job from .pcs_compute_scan.

```
prisma-cloud-compute-scan:
  stage: build
  extends:
    - .pcs_compute_scan
```

STEP 3 | Refine IaC scan variables.

You can override the value for any variable specified in the project settings by defining the variable again in the gitlab-ci.yml file. Verify that the prisma_cloud_scan_image is in the runner docker before you execute the image scan script ./image_scan.sh

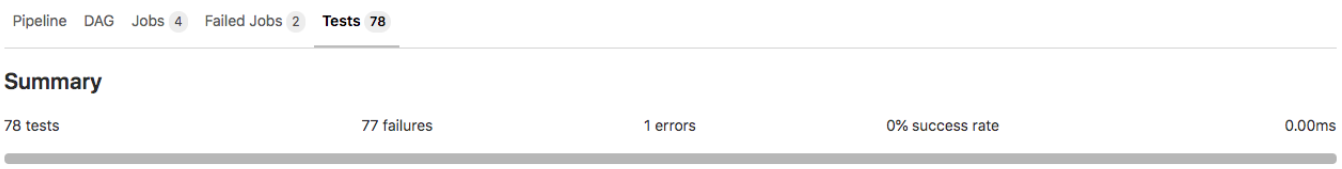
```
prisma-cloud-compute-scan:
  stage: build
  extends: .pcs_compute_scan
  variables:
    prisma_cloud_scan_image: nginx:latest
  script:
    # build or pull your docker image first
    - docker pull $prisma_cloud_scan_image
    # trigger the pre-defind image scan script located in currennt
 directory
    - ./image_scan.sh
```

STEP 4 | View container scan results.

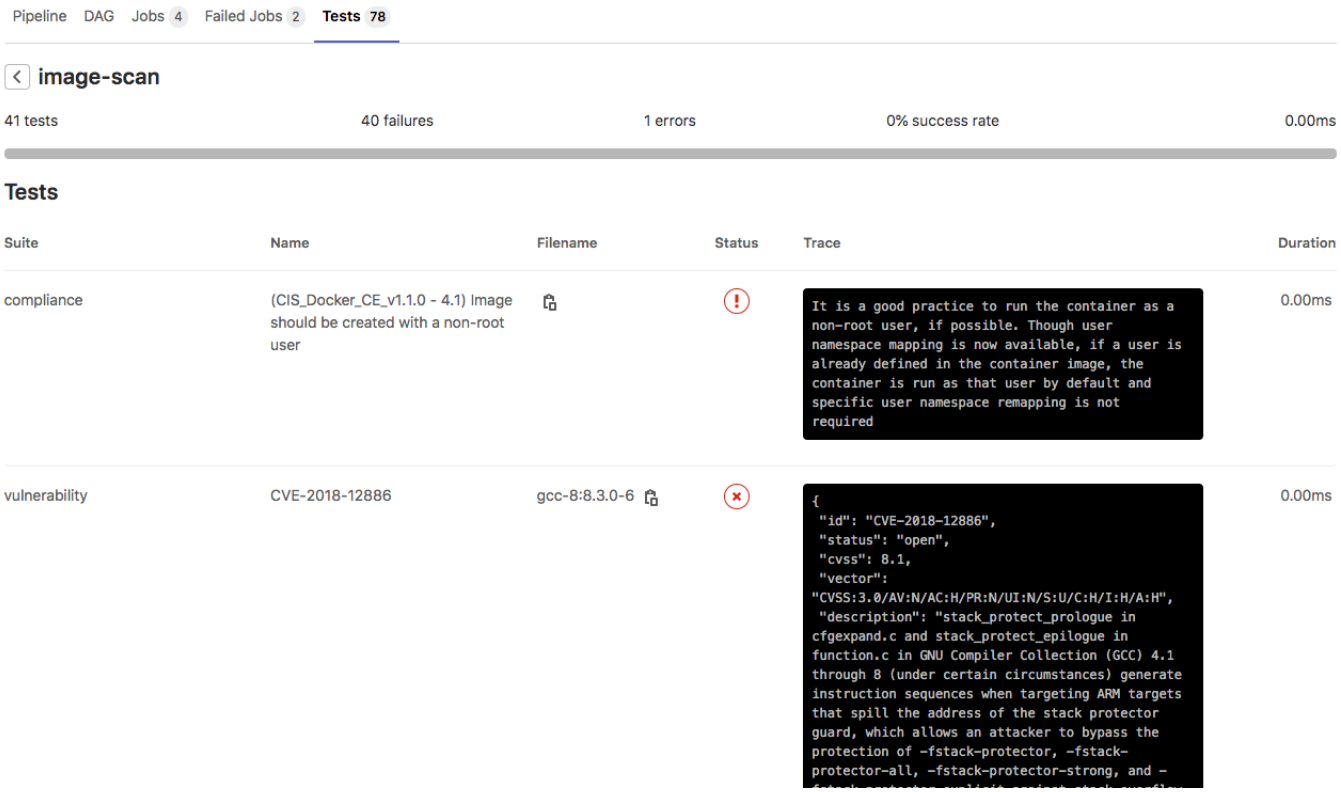
The Prisma Cloud container image scan uses the failure criteria you defined in the **prisma_cloud_cicd_failure_criteria** environment variable to pass or fail a scan. When it detects a security issue, it generates an artifact. The scan report is in Junit compatible format and you can view the results on your GitLab pipeline and on the DevOps Inventory on Prisma Cloud.

- View the scan results on the **tests** tab on your GitLab pipeline.

  List of job results.



- Container Scan result with details for a specific job.



Use the Prisma Cloud Extension for GitLab SCM

Use the Prisma Cloud extension to scan IaC templates when you create or update a merge request. You can define failure criteria for each GitLab project and view the scan results directly in the GitLab user interface. In addition, the Prisma Cloud extension can create GitLab issues that report details from IaC scans for checks against security policies. This ability enables you to fix all the reported issues before your changes are merged into the repository.

The sections below describe how to set up the Prisma Cloud extension and how to use it.

- Configure the Prisma Cloud Extension for GitLab SCM

- Run an IaC Scan in a Merge Request

If you have been using the Prisma Cloud GitLab extension v1, there are no updates to the environment variables. For backward compatibility, the GitLab user access token that you provided in v1 is still supported. You must however update the webhook URL to ***`https://{api_server}/iac/v2/gitlab/webhook`*** *. See Step 3 for details.*

Configure the Prisma Cloud Extension for GitLab SCM

The Prisma Cloud Extension for GitLab SCM uses a Webhook integration to scan your IaC templates. It uses a config.yml file where you specify the template type, template specific parameters, and the tags you use in your environment. The Prisma Cloud Extension for GitLab SCM does not require a software installation.

The following table summarizes the environment variables you must configure on your GitLab project to enable IaC scans.

If you want to run IaC scans for both GitLab SCM and GitLab CICD in a single project, you can set environment variables for both in your project settings.

STEP 1 | Create an access token on GitLab.

1. Select **User settings** > **access tokens**
2. Use a bot account or service account to create the token.

   The account requires project Maintainer, Owner, or Administrator role to ensure the Prisma Cloud extension can read the environment variables. The permissions required are— **`api`**, **`read_user`**, and **`read_repository`**, so that the webhook can send the appropriate data to the Prisma Cloud IaC service to perform the checks against security policies.

STEP 2 | Generate Prisma Cloud GitLab SCM Webhook token.

Make a curl request or browse to the Prisma Cloud GitLab SCM API endpoint to get the webhook token with the following parameters:

- prismacloud_api_server—Prisma Cloud base API URL. The API URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed.

  If the tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io, replace **`app`** in the URL with **`api`**.

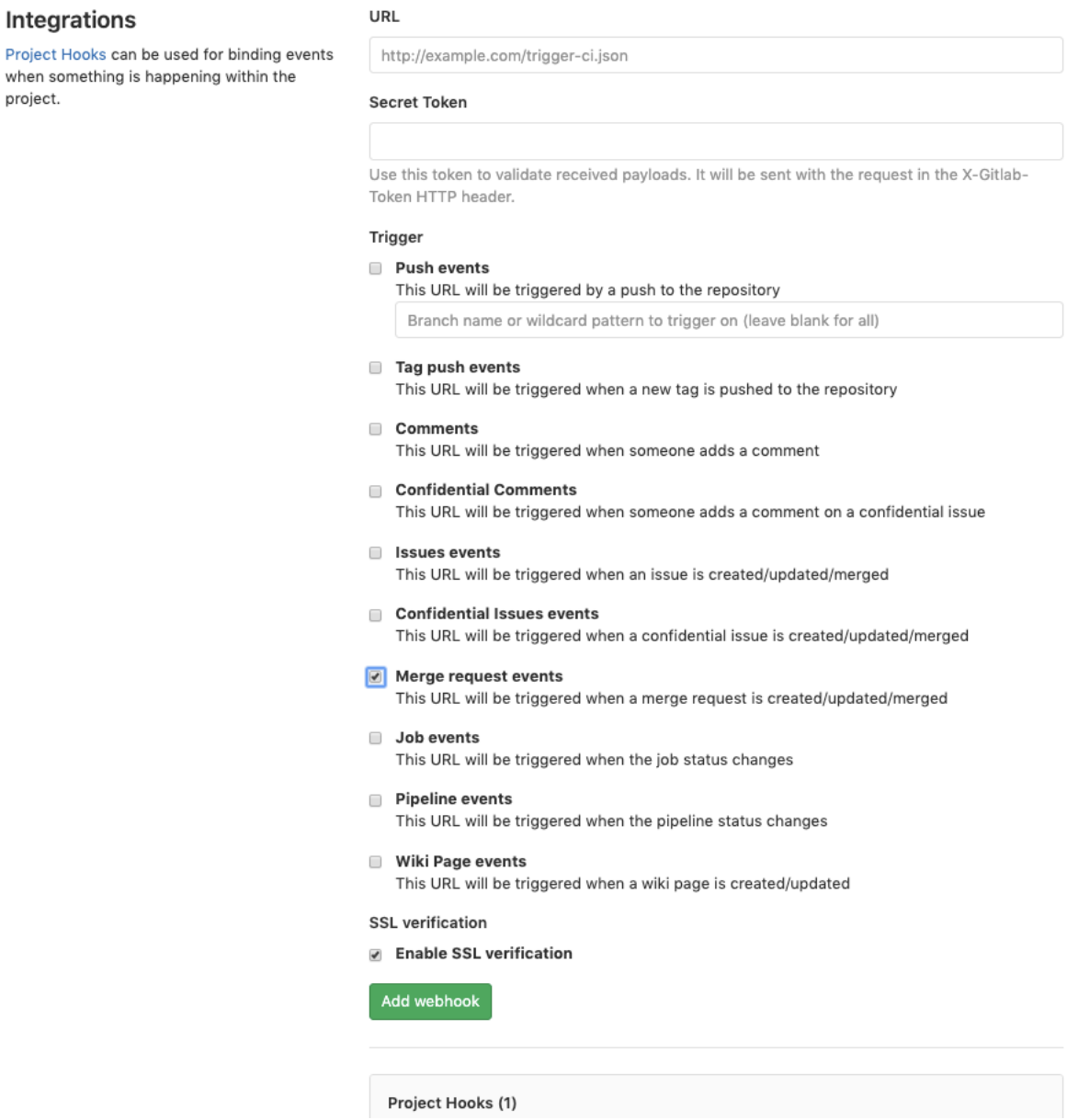
  Refer to the Prisma Cloud REST API Reference, for more details.
- access_key—Prisma Cloud access key for API access. If you do not have a key, see Create and Manage Access Keys.
- secret_key—Secret key that corresponds to the Prisma Cloud access key.
- project_repo—The name of the GitLab project that you want to scan using the Prisma Cloud SCM extension.
- gitlab_token—The token you generated in the previous step.

For example: **`curl -u {access_key}:{secret_key} --request GET \ 'https://{prismacloud_api_server}/iac/v2/gitlab/token?project={project_repo}&access_token={gitlab_token}'`**

```
{"token":"HfwMaNS8zBNlszyxOAS2Q-
QQYAEkRCntUWH2E90gldK49XkkibRu4FZVap2GImkjLo4cFAADm62aBQDzdYqg4Po7w9IDDTgy=="}
```

STEP 3 | Set up a webhook to perform the IaC scan during merge request operations.

1. Navigate to **Project** > **Settings** > **Webhooks**



2. Specify the webhook URL in the **URL** field.

   The webhook URL includes the Prisma Cloud API server URL and a set of optional query parameters. The syntax is: **`https://{api_server}/iac/v2/gitlab/webhook?{parameters}`**

api_server	Prisma Cloud API server. For example: api.prismacloud.io	
asset_name	The IaC scan asset name you choose.	The GitLab project name
template_type	The asset template type. Allowed values include cft, k8s and tf.	The template_type defined in **`.prismaCloud/config.yml`**

failure_mr_criteria	This the number and severity of security policy check failures that need to occur to trigger a merge request failure. The syntax for the value is as follows. `High:x,Medium:y,Low:z,Operator:op` In the syntax above, x is a count of high-severity policy check failure, y is a count of medium-severity policy check failures, and z is a count of low-severity policy check failures. The Operator value determines what combination of High/Medium/Low counts should result in a merge request failure. The default for each count is 0. The value for Operator, op, can be either OR or AND. The default is OR.	`High:1,Medium:1,Low:1,Operator:`
failure_issue_criteria	This the number and severity of security policy check failures that need to occur to trigger GitLab issue creation. The criteria should be equal or more restricted than MR failure criteria to avoid too many trivial issues created.	`High:100,Medium:100,Low:100,Ope`
tags	Prisma Cloud tags are different from GitLab tags or cloud tags that you might have included within your IaC templates. Prisma Cloud tags enable visibility on the Prisma Cloud administrator console. Provide the value for this environment variable as a comma-separated list of tags that you define. An example is: project:x,owner:y	None

3. Provide the Prisma Cloud GitLab SCM Webhook token that you generated earlier in the **Secret Token** field.
4. Select **Merge request events** as the trigger.
5. Select **Enable SSL verification**.
6. Select the **Add webhook** button to add the webhook you just configured.

   Your newly added webhook should appear under the **Project Hooks** list on the same page.

STEP 4 | Set Up Your Prisma Cloud Configuration File for IaC Scan file.

Create the **.prismaCloud/config.yml** and add it to the root directory of your repository branch. The file is required, and it must include the template type, version, and the template specific parameters and tags you use in your environment.

Run an IaC Scan in a Merge Request

When you create, update, or reopen a merge request with added or modified files, this set up will trigger a merge request event to invoke a Prisma Cloud IaC scan for all files in the merge request. The scan does not include deleted files.

You can see the results of the IaC scan through a comment on the merge request. If the scan results meets or exceeds the failure criteria set in the environment variable **`prisma_cloud_scm_failure_mr_criteria`**, then the results will show that the security check failed.

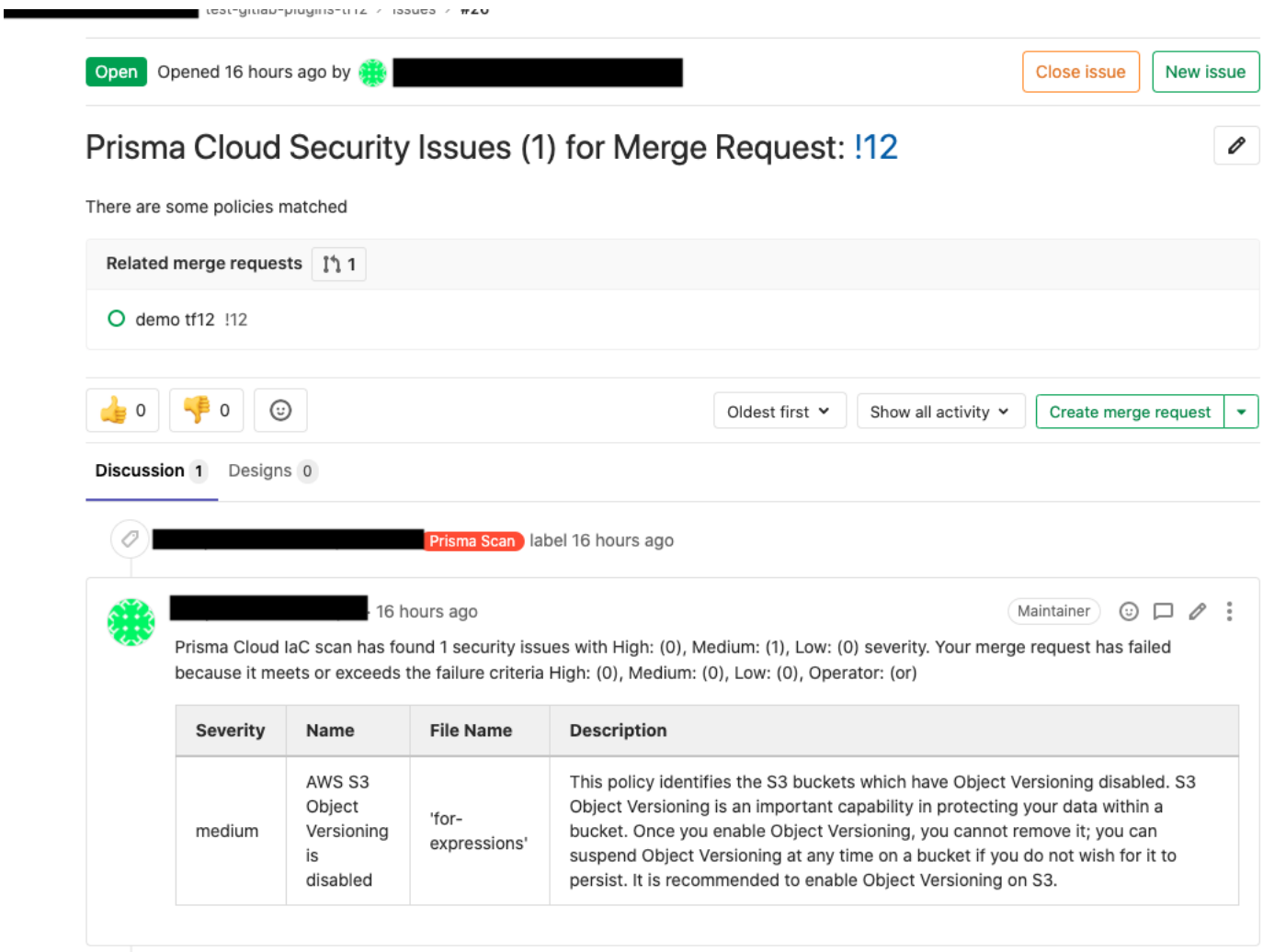
The following shows the result of an IaC scan for a merge request. In this example, the IaC scan resulted in some security policy check failures. Since the number and severity of the failures did not meet the failure

criteria set in the environment variable **`prisma_cloud_scm_failure_mr_criteria`**, the security check passed, and the merge request succeeded.

Prisma Cloud IaC scan has found 2 security issues with High: (0), Medium: (1), Low: (1) severity. Your merge request has succeeded because it does not exceed the failure criteria High: (5), Medium: (5), Low: (5), Operator: (or)

Severity	Name	File Name	Description
medium	AWS S3 Object Versioning is disabled	'test.tf'	This policy identifies the S3 buckets which have Object Versioning disabled. S3 Object Versioning is an important capability in protecting your data within a bucket. Once you enable Object Versioning, you cannot remove it; you can suspend Object Versioning at any time on a bucket if you do not wish for it to persist. It is recommended to enable Object Versioning on S3.
low	AWS S3 CloudTrail buckets for which access logging is disabled	'test.tf'	This policy identifies S3 CloudTrail buckets for which access is disabled. S3 Bucket access logging generates access records for each request made to your S3 bucket. An access log record contains information such as the request type, the resources specified in the request worked, and the time and date the request was processed. It is recommended that bucket access logging be enabled on the CloudTrail S3 bucket.

The following is an example of output that occurs when the failure criteria in the environment variable **`prisma_cloud_scm_failure_issue_criteria`** is met or exceeded.



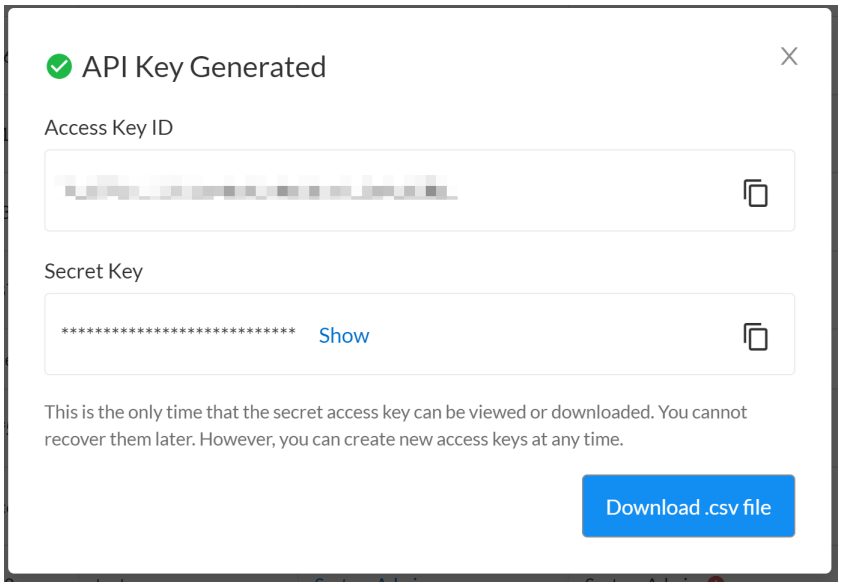
Use the Prisma Cloud IaC Scan Plugin for Jenkins

Use the Prisma™ Cloud IaC Scan Plugin to perform Infrastructure as Code (IaC) scanning during Jenkins builds. To use Prisma Cloud IaC scan functionality, you need to have a connection to a Prisma Cloud api server and the login credentials. The Prisma Cloud IaC scan plugin scans the templates for misconfigurations, and if an issue is detected then you will be able to see the issues generated as a report within Jenkins.

STEP 1 | Verify the prerequisites.

You must have administrative privileges in Jenkins to install the Prisma Cloud IaC scan plugin.

1. Launch your browser to point to the location of your Jenkins server. For example, the default URL is http://localhost:8080. Replace **`8080`** with a custom port you used.
2. Enter your **`username`** and **`password`** and click **Sign in**.
3. Generate your access key and secret key in Prisma Cloud.
   - **Access Key**—Enables programmatic access to Prisma Cloud. To create an access key select **Settings** > **Access Keys** > **Add New.**
   - **Secret Key**—You should have saved this secret key when you generated it. You cannot view it on the Prisma Cloud web interface.



4. Set up your Prisma Cloud configuration file for IaC Scan

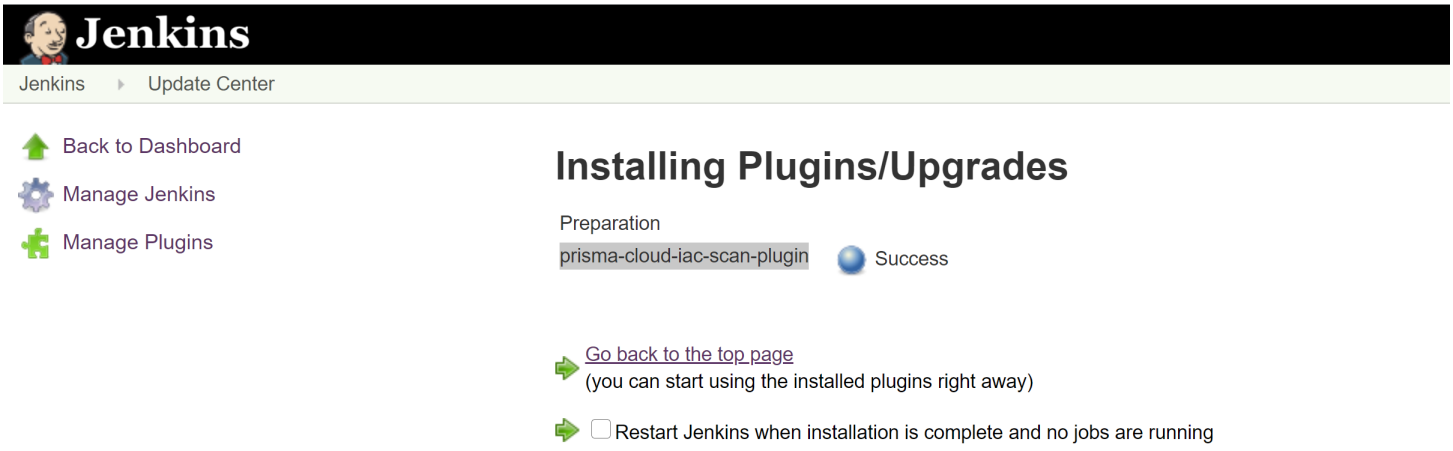
   Create the .prismaCloud/config.yml file and add it to the root directory of your source code in your repository. The config.yml file is required, and it currently supports template_parameters like variables, and the tags you use in your environment.

STEP 2 | Install the Prisma Cloud IaC Scan plugin.

Use the plugin manager in Jenkins to install the Prisma Cloud IaC scan plugin for scanning your templates and images across all Source Code Management repositories connected to Jenkins.

1. Log in to Jenkins.
2. Select **Manage Jenkins** > **Manage Plugins** > **Available**.
3. Enter **`prisma cloud`** to find Prisma Cloud iac scan on the Jenkins marketplace.

4. Install the plugin.
5. Select **Restart Jenkins** when installation is complete and no jobs are running.



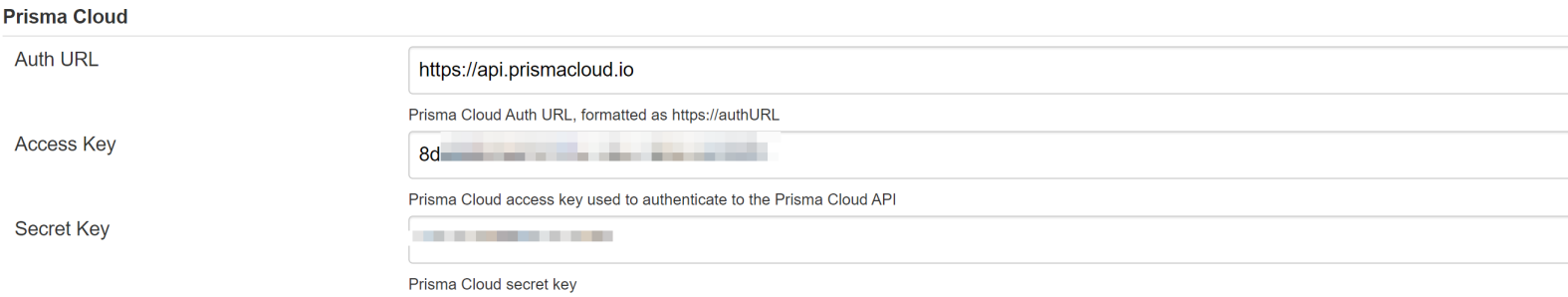
6. Select **Manage Jenkins** > **Manage Plugins** > **Installed** to verify that the Jenkins plugin is available for use.

STEP 3 | Connect Jenkins to your Prisma Cloud server.

The API URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed.

If the tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io, replace **`app`** in the URL with **`api`**. Refer to the Prisma Cloud REST API Reference, for more details.

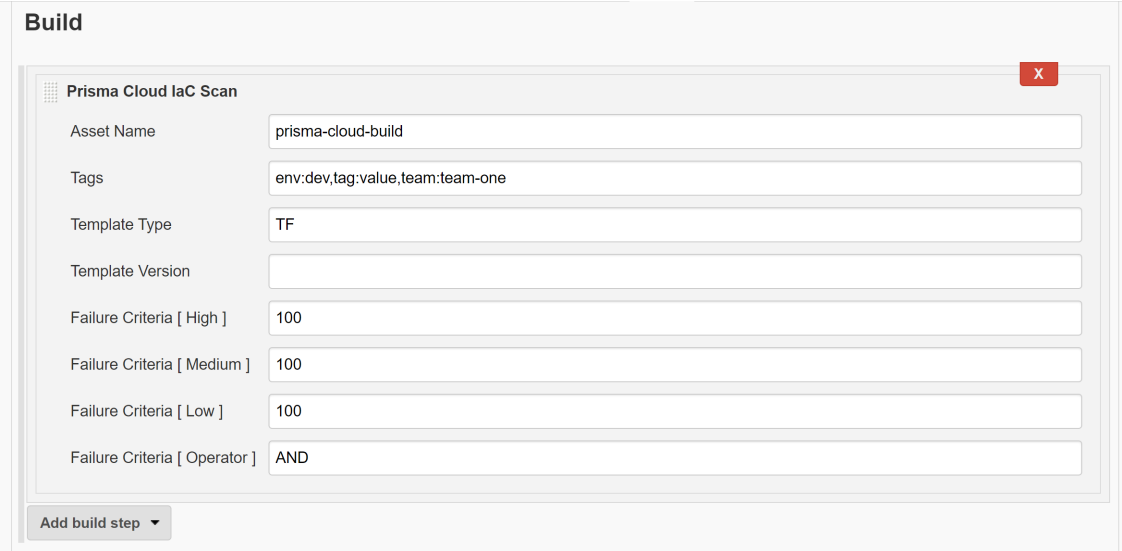
1. In Jenkins select **Manage Jenkins** > **Configure System.** Enter the credentials for **`Auth URL`**, **`Access Key`**, and **`Secret Key`**.



2. Click **Test Connection** to authenticate into Prisma Cloud and after successful connection the message **`Successfully authenticated with server`** will display.

STEP 4 | Add a build step in Jenkins

After you connect Prisma Cloud IaC scan to Prisma Cloud add a build step.



1. Select **New Item** to add a name for the item that you want to build.
2. Save your item by clicking **OK**.
3. Select **Configure** and navigate to **`Build`**. Specify the following fields except Template Version which is optional:

   - **Asset Name**—The registered asset name that will appear as the resource name in the Prisma Cloud Devops inventory. The character limit for asset name should not exceed 255 characters. For example, *prisma-cloud-build* is a valid asset name.
   - **Tags**—The key-value pairs separated by commas which allows the build to be searched in the DevOps inventory UI of Prisma Cloud. Examples of valid tags; **`env:dev`**, **`tag:value`**, **`team:team-one`**.
   - **Template Type**—A template is a configuration management tool that is used to provision resources in the cloud. The templates supported are Terraform, AWS CloudFormation, and Kubernetes Templates. Enter the templates abbreviations as values, for example **`TF`**, **`CFT`**, and **`K8S`**.
   - **Template Version (Optional)**—This field is only applicable if you entered **`TF`** in the Template Type field. Examples of valid values are 0.11, 0.12, or 0.13. The value you enter will be a hint as the system will attempt to determine the correct version number, otherwise the system will use the value you entered.
   - **Set up the failure criteria for the Prisma Cloud IaC scan**

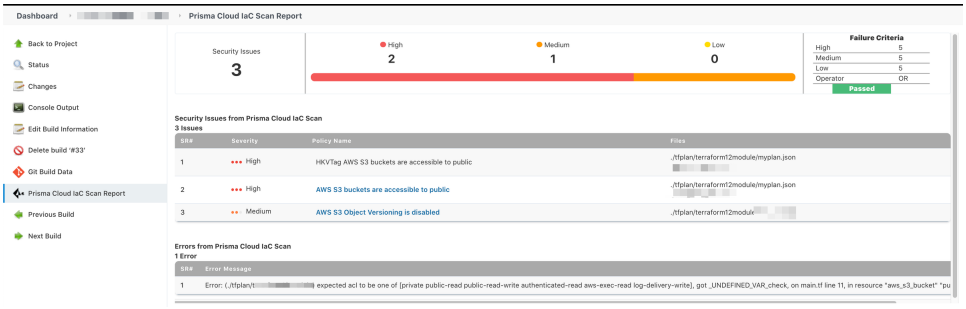
     Define the number of issues by severity in **Prisma Cloud IaC scan** plugin. Set the **`High`**: x, **`Medium`**: y, **`Low`**: z, Operator: O, where, x,y, and z are the number of issues of each severity, and the operator is **`OR`**, **`AND`**.

     For example:

     - To fail the pipeline for any security issue detected—High : 0, Medium : 0, Low : 0, Operator: OR
     - To never fail the pipeline—High : 1000, Medium : 1000, Low : 1000, Operator: AND

STEP 5 | Run an IaC Scan on your build.

1. Select **Build Now** to generate your build.
2. Refresh Jenkins so that **Prisma Cloud IaC Scan Report** will appear in the left pane.
3. Select the build and then select **Prisma Cloud IaC Scan Report** to view the report.



Use the Prisma Cloud Extension for Visual Studio Code

With the Prisma Cloud Enterprise edition license, you can install the Prisma Cloud extension for Visual Studio (VS) Code to detect issues in your Infrastructure-as-Code (IaC) templates and deployment files against Prisma Cloud security policies early in the software development process, directly within your VS Code editor. The following steps show how simple it is to install and check your templates and files for potential security misconfigurations.

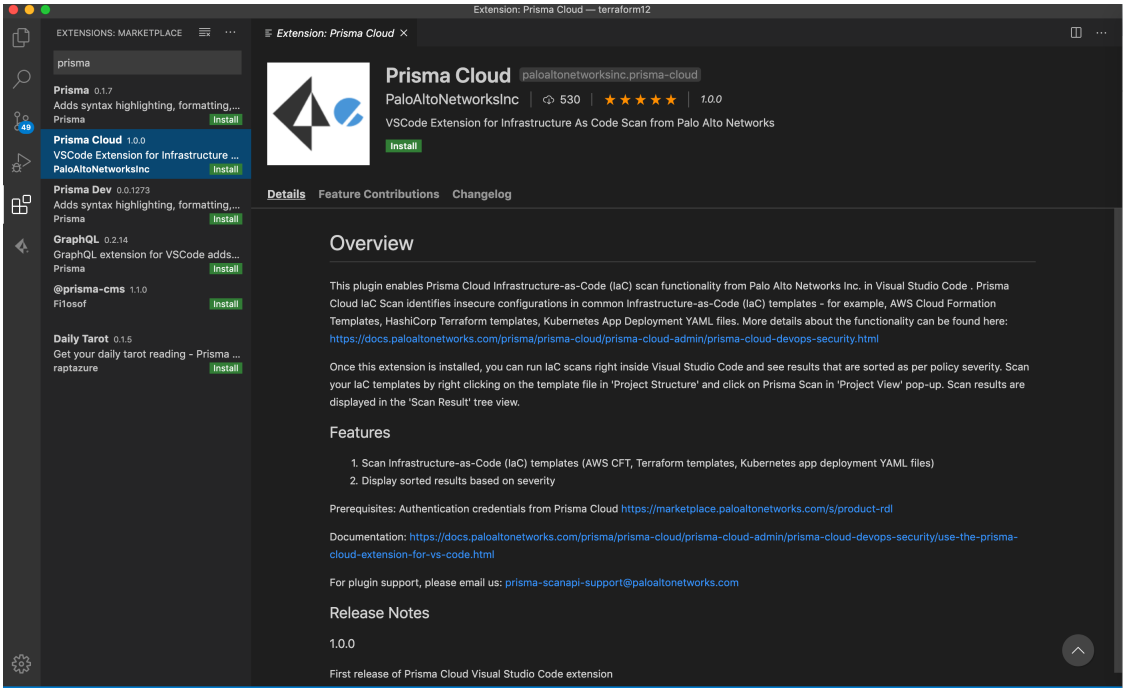
1. Install Prisma Cloud Extension for Visual Studio Code
2. Configure the Prisma Cloud Extension for VS Code
3. Scan Using the Prisma Cloud VS Code Extension

Install Prisma Cloud Extension for Visual Studio Code

The Prisma Cloud extension supports VS Code version 1.36.0 and later.

STEP 1 | In VS Code, navigate to **Extensions**.

STEP 2 | Enter `Prisma Cloud` in search.



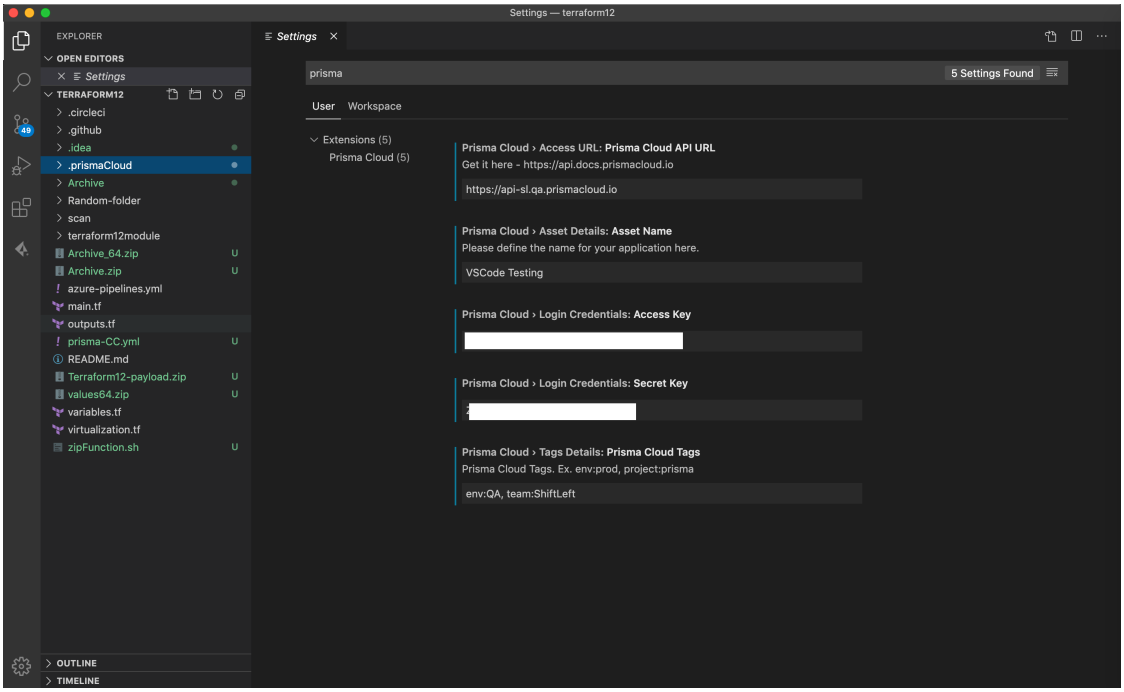
STEP 3 | **Install** the extension.

Configure the Prisma Cloud Extension for VS Code

Before you can use the Prisma Cloud extension for VS Code, you'll need to configure the extension to include your API access key, secret key, and Prisma Cloud API URL. If your access keys change, you must update the details in the extension settings.

STEP 1 | In VS Code, navigate to **Settings** > **Extensions** > **Prisma Cloud**.

STEP 2 | Enter the following information for the Prisma Cloud extension:



- **Prisma Cloud API URL**.

  The URL for Prisma Cloud varies depending on the region and cluster on which your tenant is deployed. The tenant provisioned for you is, for example, https://app2.prismacloud.io or https://app.eu.prismacloud.io. Replace **`app`** in the URL with **`api`** and enter it here. Refer to the Prisma Cloud REST API Reference, which is accessible from the Help Center within the Prisma Cloud web interface for more details.
- **Access Key**.

  The Prisma Cloud access key enables programmatic access. If you do not have a key, you must Create and Manage Access Keys.
- **Secret Key**.

  You should have saved this key when you generated your Prisma Cloud access key and corresponding secret key. You cannot view the secret key on the Prisma Cloud web interface.
- **Asset Name**

  Give your VSCode instance an asset name. You can choose an arbitrary name. Prisma Cloud uses the asset name to track results. Some examples of names are appteam_vscode or johndoe_vscode.
- **Prisma Cloud Tags**

  Prisma Cloud tags arre different from cloud tags that you may have included within your IaC templates. Prisma Cloud tags enable visibility in the Prisma Cloud administrator console.

  Provide the values as a comma-separated list of tags. in the Prisma Cloud Tags field. An example list is: **`owner:johndoe, team:creditapp, env:dev`**.

STEP 3 | Set Up Your Prisma Cloud Configuration File for IaC Scan

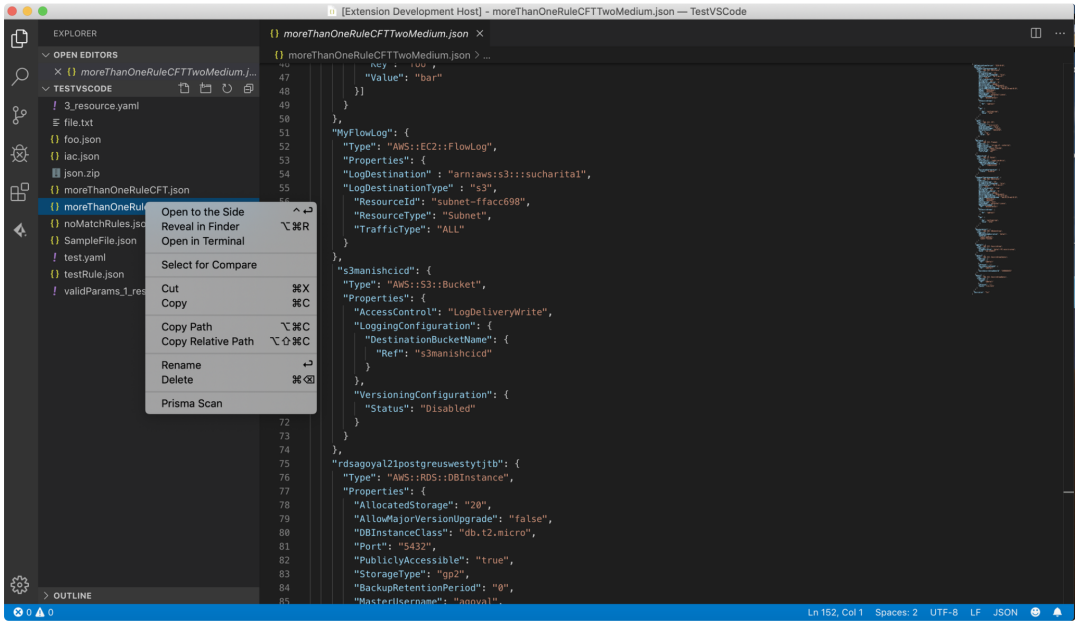
Create the .prismaCloud/config.yml file and add it to the root directory of your repository branch. The file is required, and it must include the template type, version, and the template specific parameters and tags you use in your environment.

Scan Using the Prisma Cloud VS Code Extension

Now, you are ready to scan your templates and view the results within the VS Code editor.

STEP 1 | Scan a file or folder.

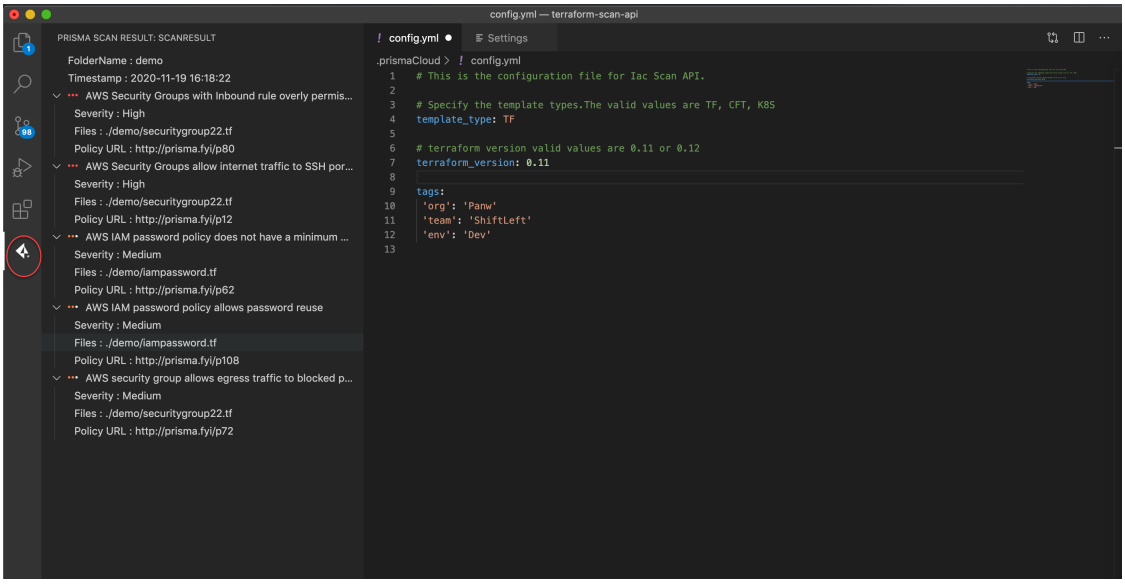
Right-click on your template file in the VS Code Explorer and select **Prisma Scan** to check your template against Prisma Cloud IaC policies.



If you are performing the scan using Helm Charts, then right-click on the directory containing Chart.yml; do not click on the Chart.yml file. To select a root directory in VSCode click Open Folder, click the directory in VSCode, right-click on some open space, and then select Prisma Scan. If the project contains multiple directories then in VSCode click Open Folder, navigate to the directory that contains /charts, right-click on some open space, and then select Prisma Scan.

STEP 2 | View the scan results.

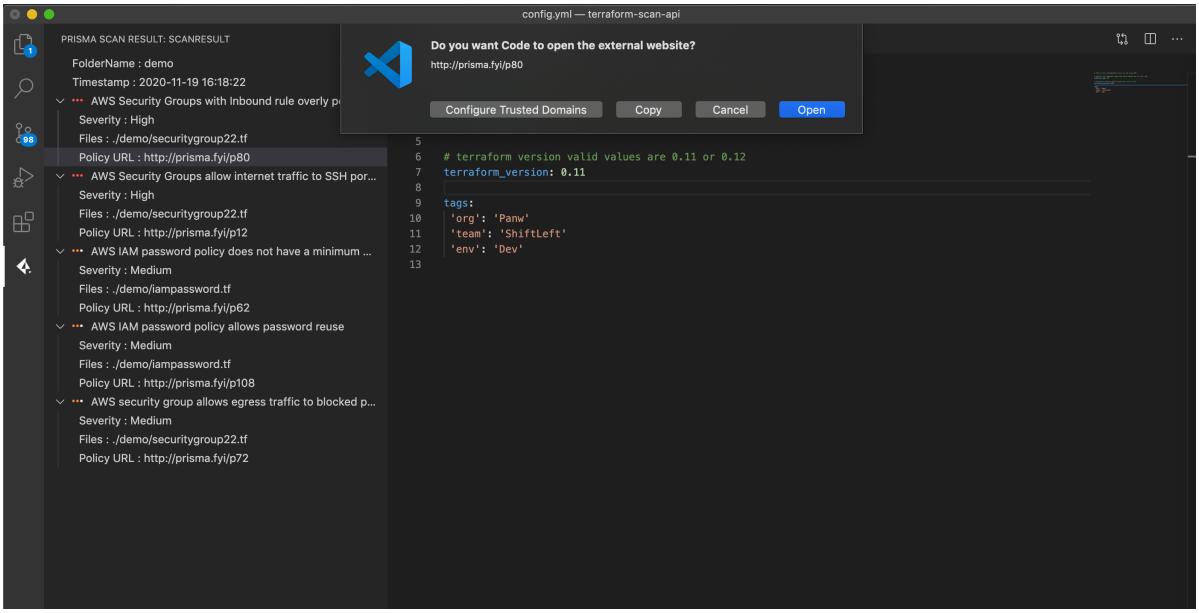
Select the **Prisma Cloud** icon on the Activity Bar.



The results of the check will appear in the **Prisma Cloud Result** window. If the extension discovers any policy violations, the **Prisma Cloud Result** window sorts the results by severity and displays the following details for each violation:

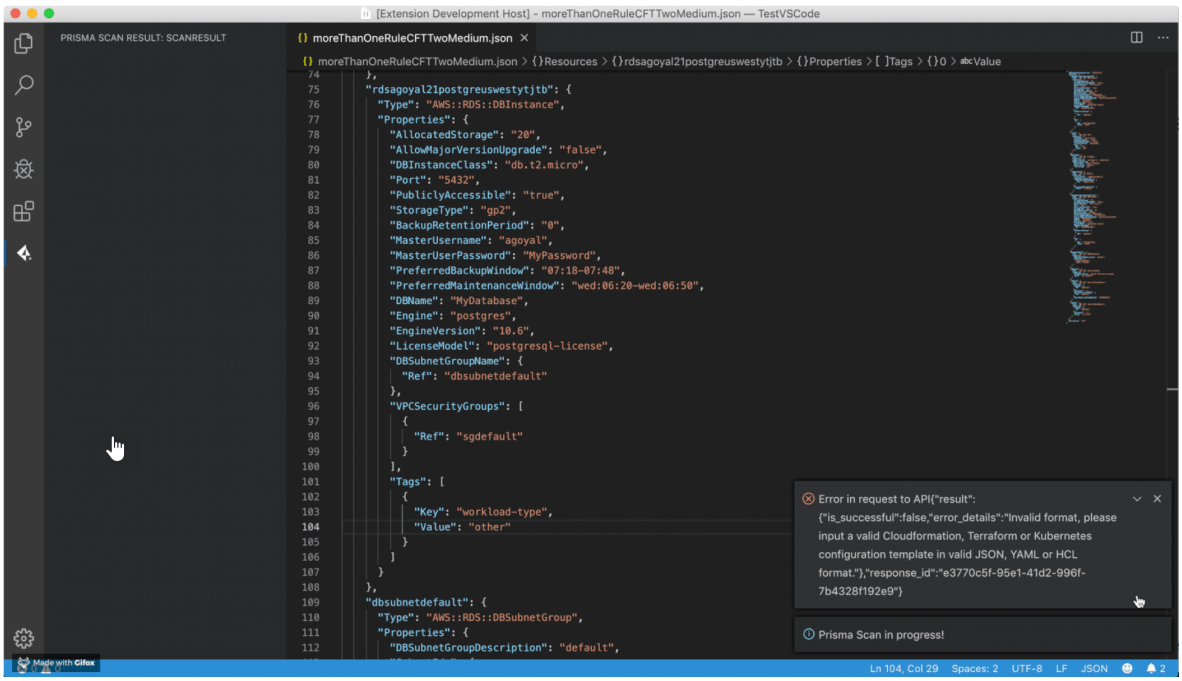
- Name of the violated policy
- Severity of the violation
- Names of the module or files that have issues
- Timestamp of the scan

For more details on a matched policy, select **Policy URL** for its detailed documentation.



When you scan a different template, the result window refreshes to display the latest scan results.

In case of an error condition, the error will appear at the bottom right of VSCode and in the scan result window.



Use the Prisma Cloud IaC Scan REST API

The IaC Scan API is deprecated and will be removed in a future release.

Prisma Cloud provides a REST API that enables you to scan IaC templates to test them against Prisma Cloud security policies. With this API, you can initiate IaC scans asynchronously and integrate your scan results with Prisma Cloud.

The Prisma Cloud IaC scan service supports the following:

- Terraform templates
- Terraform plan files in JSON format
- CloudFormation templates
- Kubernetes app manifests

While you should take advantage of the IaC scan plugins that are available, you have the option to use the IaC scan API directly.

Prerequisites

Before you make your first IaC Scan API V2 request, complete the following prerequisites:

- Know the base API URL for your Prisma Cloud tenant, which will serve as the base URL of all your IaC scan API V2 requests.

  Your Prisma Cloud API base URL depends on the region and cluster of your Prisma Cloud tenant. For example, if your Prisma Cloud admin console URL is **`https://app.prismacloud.io`**, then your Prisma Cloud API base URL is **`https://api.prismacloud.io`**. See Prisma Cloud API URLs for a list of valid Prisma Cloud API URLs.
- Get the Prisma Cloud JSON web token (JWT) token for authentication.

  Your Prisma Cloud administrator normally assigns an API access key, which you can use with a Prisma Cloud login API request to obtain a JWT token. Include the JWT token in the header of your IaC scan API V2 requests. See Access the Prisma Cloud REST API for details.
- Know the **`Content-Type`** header, which is in the table below.

  The following table shows the required headers for all the IaC scan API V2 requests.

Request Header	Value
`x-redlock-auth`	Your JWT token as described above
`Content-Type`	**`application/vnd.api+json`**

If you want to scan Terraform plan files, an additional prerequisite is to use the Terraform plan and Terraform show commands to convert your Terraform modules to JSON-formatted plan files.

Use the IaC Scan API V2 to Scan Templates

The basic steps to scan your templates are:

- Create an IaC scan asset in Prisma Cloud.
- Upload the templates to be scanned. The IaC Scan service can process files that are up to 300MB in size.

- Start a job to perform a scan of your uploaded templates.

After you submit a job, you can check your job status and view the results.

The IaC scan API V2 is JSON:API compliant.

The examples below show requests to run and manage an asynchronous IaC scan job in Prisma Cloud.

STEP 1 | Create an IaC scan asset in Prisma Cloud.

The following endpoint enables you to create an IaC scan asset. An IaC scan asset represents a collection of one or more templates whose contents have been or will be scanned by the Prisma Cloud scan service to check against Prisma Cloud IaC policies.

Method	Endpoint URL
POST	https://<Prisma Cloud API base URL>/iac/v2/scans

The following curl request creates an IaC scan asset.

```
curl -X POST 'https://<Prisma Cloud API URL>/iac/v2/scans' \
    --header 'x-redlock-auth: <JWT Token>' \
    --header 'Content-Type: application/vnd.api+json' \
    --data-raw '
{
  "data": {
    "type": "async-scan",
    "attributes": {
      "assetName": "my-asset",
      "assetType": "IaC-API",
      "tags": {
        "env": "dev"
      },
      "scanAttributes": {
        "projectName": "my-project"
      },
      "failureCriteria": {
        "high": 1,
        "medium": 10,
        "low": 30,
        "operator": "or"
      }
    }
  }
}'
```

The request body parameters both provide metadata about your scan and define failure criteria for an IaC scan job. Set *data.type* to **`async-scan`**. The parameters in this example includes the following attributes:

- *data.attributes.assetName*: Can be a project name or any identifier you want to attach to the scan. Some examples are a CI/CD project name or a Git repository name.
- *data.attributes.assetType*: Identifies where the IaC scan is being done. For the IaC scan API V2, the standard value is IaC-API. You are allowed to use other values, though. For example, if you are requesting a scan through a GitHub action and using the IaC scan API V2, you can set the

data.attributes.assetType to **`GitHub`**. See the request body of the Add Scan Asset for a list of supported asset types.

- *data.attributes.tags*: Can be used to set up role-based access control and policies. Prisma Cloud tags are different from cloud tags that you might have included in your IaC templates. An example tag list is **`"owner:johndoe", "team:creditapp", "env:dev"`**.
- *data.attributes.scanAttributes*: Any additional details that you want to send as key/value pairs with each scan. Some examples are Git pull request numbers or specific build numbers inside a CI/CD project.
- *data.attributes.failureCriteria*: Enables you to evaluate scan results against set failure criteria to obtain failed or passed verdicts. You can set the count for high, medium, and low severity issues and use **`and`** or **`or`** operators to refine your criteria. The endpoint checks each severity violation number separately against scan results and applies the operator to each evaluation. The scan triggers a failure if the number of violations is greater than or equal to the *data.attributes.failureCriteria* values. For example, if you want a scan to fail when it detects any type of issue, you can set the failure criteria to **`"high": 1`**, **`"medium": 1`**, **`"low": 1`** and **`"operator": "or"`**. An example of *data.attributes.failureCriteria* values that cause the endpoint to run checks in warning mode is **`"high": 1000`**, **`"medium": 1000`**, **`"low": 1000`**, **`"operator": and`**.

See Add Scan Asset for additional request body parameters.

The following example shows the response of a successful request. In this example, the *data.id* value is a *scanID* that you will use in subsequent requests to manage your scan job. The *data.links.url* value is a pre-signed URL that you will use to upload the files you want scanned.

```
{
  "data": {
    "id": "12345678-5717-4562-b3fc-2c963f66afa6",
    "links": {
      "url": "https://s3.amazonaws.com/s3sign2-bucket-hchq3nwuo8ns/s3-
sign-demo.json?X-Amz-Security-Token=FQ…&X-Amz-Credential=ASIAJF3BXG…&X-
Amz-Date=20170125T044127Z&X-Amz-Expires=60&X-Amz-Signature=24db0"
    }
  }
}
```

STEP 2 | Upload the templates to be scanned.

Prisma Cloud uses a pre-signed URL that gives you temporary access to upload your files. As was mentioned in the previous step, the pre-signed URL is the *data.links.url* from the response for a request to add a scan asset. The following curl command uploads a file.

```
curl -v -X PUT '<pre-signed URL>' -T <path of file to be uploaded>
```

You can upload a single file or upload multiple files as a zip archive.

Some example templates and plan files are available at Prisma Cloud IaC samples on GitHub; you can use one of these files to experiment with an upload request.

STEP 3 | Start a job to perform a scan of your uploaded templates.

The following endpoint enables you to start an asynchronous job to perform a scan of your uploaded file.

Method	Endpoint URL
POST	https://<Prisma Cloud API base URL>/iac/v2/scans/{scanID}

The path parameter *scanID* is the *data.id* from the response of your earlier request to create a scan asset.

The following curl command starts a job that scans the file you've already uploaded.

```
curl -X POST 'https://<Prisma Cloud API URL>/iac/v2/
scans/12345678-5717-4562-b3fc-2c963f66afa6' \
    --header 'x-redlock-auth: <JWT Token>' \
    --header 'Content-Type: application/vnd.api+json' \
    --data-raw '
{
  "data": {
    "id": "12345678-5717-4562-b3fc-2c963f66afa6",
    "attributes": {
      "templateType": "tf"
    }
  }
}'
```

The *data.id* in the request body parameter is the same as the path parameter *scanID* and is optional.

Optional request parameters are available for more complex scenarios, such as:

- Runtime variables for templates.
- Variable files to use if the endpoint cannot automatically detect the files.
- A list of files to which the scan will be limited.
- A list of folders to which the scan will be limited. With this specification, the IaC scan service will scan the content of the listed folders, including files and sub-folders.

See Initiate Scan Job for details about the endpoint and request parameters.

STEP 4 | Query your job status.

The following endpoint enables you to query the status of your asynchronous IaC scan job.

Method	Endpoint URL
GET	https://<Prisma Cloud API base URL>/iac/v2/{scanID}/status

The *scanID* path parameter is the *data.id* in the response object from your request to create a scan asset and which you also used as a path parameter to start the scan job. The following is an example of a curl command to request your job status.

```
curl -X GET 'https://<Prisma Cloud API URL>/iac/v2/
scans/12345678-5717-4562-b3fc-2c963f66afa6/status' \
--header 'x-redlock-auth: <JWT Token>' \
--header 'Content-Type: application/vnd.api+json'
```

The following example shows the response of a successful request.

```
{
  "data": {
    "id": 12345678-5717-4562-b3fc-2c963f66afa6",
    "attributes": {
      "status": "processing"
```

```
    }
  }
}
```

STEP 5 | Request IaC scan results.

The following endpoint enables you to request your IaC scan results after the scan job is done. The results include details about potential violations the scan service discovered while checking files against Prisma Cloud policies.

Method	Endpoint URL
GET	https://<Prisma Cloud API base URL>/iac/v2/scans/{scanID}/results

The *scanID* path parameter is the *data.id* from the response object of your request to create a scan asset and which you also used as a path parameter to start the scan job. The following is an example of a curl command to request your job results.

```
curl -X GET 'https://<Prisma Cloud API URL>/iac/v2/
scans/12345678-5717-4562-b3fc-2c963f66afa6/results' \
--header 'x-redlock-auth: <JWT Token>' \
--header 'Content-Type: application/vnd.api+json'
```

The response includes information such as:

- Which Prisma Cloud policies were violated
- Which file(s) violated the policies
- The severity of the violations
- A link to more information about the policies

The following example shows the response of a successful request.

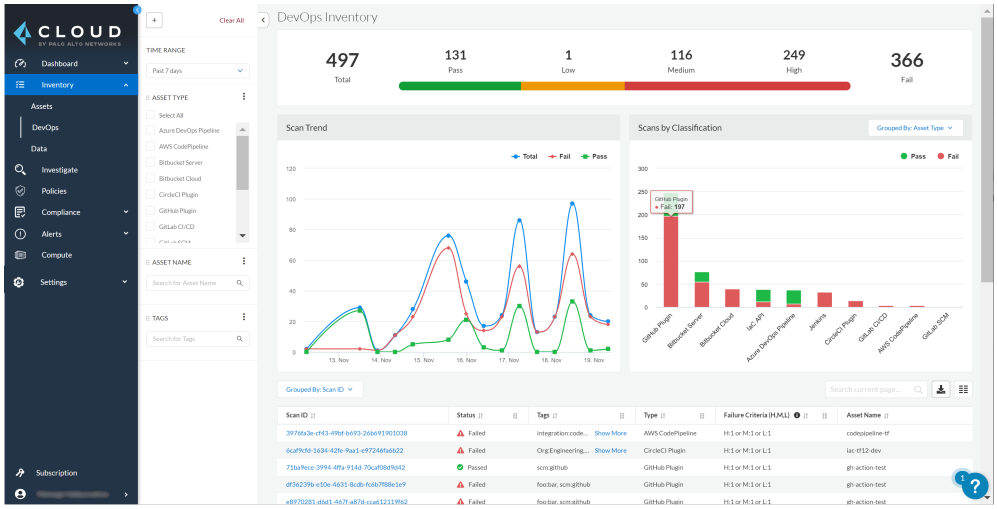
```
{
  "meta": {
    "matchedPoliciesSummary": {
      "high": 1,
      "medium": 0,
      "low": 0
    },
    "errorDetails": []
  },
  "data": [
    {
      "id": "12345678-5717-4562-b3fc-2c963f66afa6",
      "attributes": {
        "severity": "high",
        "name": "AWS Security Groups allow internet traffic to SSH port
 (22)",
        "rule": "$.resource[*].aws_security_group exists and
 ($.resource.aws_security_group[*].ingress[?(@.protocol == 'tcp'&&
 @.from_port<23 && @.to_port>21 )].cidr_blocks contains 0.0.0.0/0",
        "desc": "This policy identifies AWS Security Groups which do allow
 inbound traffic on SSH port (22) from public internet",
        "files": [
          "./main.tf"
```

```
        ],
        "policyId": "617b9138-584b-4e8e-ad15-7fbabafbed1a",
        "docUrl": "https://docs.paloaltonetworks.com/prisma/prisma-cloud/
prisma-cloud-policy-reference/configuration-policies/configuration-
policies-build-phase/amazon-web-services-configuration-policies/
policy_617b9138-584b-4e8e-ad15-7fbabafbed1a.html"
      }
    }
  ]
}
```

Prisma Cloud DevOps Inventory

The DevOps Inventory dashboard (**InventoryDevOps**) provides a snapshot of the scan results across the different Prisma Cloud Plugins that you are using to manage security misconfigurations natively in your development environments before you deploy assets, infrastructure, or code in production.

You can view scan results for the past seven days, by default, from different *asset types* — IDEs, SCMs, CI/CD pipelines—such as Azure DevOps pipeline or IntelliJ IDE. The interactive dashboard provides filters to change the scope of data displayed, so that you can analyze information you want to view in greater detail.



At a glance the DevOps Inventory dashboard has four sections:

- **Scan Summary** - Shows the overall summary of the scans triggered by Prisma Cloud plugins, the API directly or twistcli. The overall number is split by count to display the templates that passed without any security violations and the templates that failed the scan. For the failure case, you can view the count of the violations sorted for low, medium, and high severity policies.
- **Scan Trend**—Depicts the overall health of how your processes are improving or worsening over time. The green, blue and red trend lines are overlaid to visually display the pass and failed templates scan results against the total number of templates that have been scanned. The trends depict the overall security posture of IaC scans and how they are performing over time so you can identify sudden surges with failed policy checks or sustained improvements with passing policy checks.
- **Scans By Classification**—Bar graph grouped by asset name (default), asset type, or resource list that depicts the ratio of passed to failed resources. This interactive graph allows you to drill into the passed and failed resources for details on the corresponding templates that passed or failed policy checks; you can click and drag a section of the chart to zoom in further.
- **Tabular data**— The table enables you to group the results by scan ID (default), asset name, asset type, or resource list and then drill down to view granular information the filtered results, and download it as a CSV file.

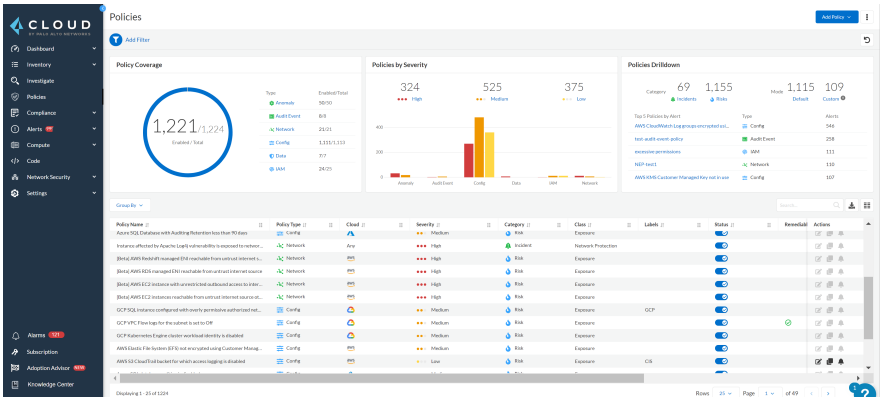
  Each row displays the IaC scan name with details on the tags, scan status, and the failure criteria. The links is each column help you explore and gain the additional context to fix the policy violations that were identified in the scan. For example, you can view the errors reported for a specific template type and resolve them early in the development lifecycle.

Create a Custom Configuration Policy for Build-Time Checks

Configuration policies monitor your resource configurations for potential policy violations. Configuration policies on Prisma Cloud can be of two sub-types—Build and Run—to enable a layered approach. Build policies enable you to check for security misconfigurations in the IaC templates and ensure that these issues do not make their way into production. The Run policies monitor resources and check for potential issues once these cloud resources are deployed. The policies used for scanning IaC templates use a JSON query instead of RQL.

Create a Configuration Policy

Use these instructions to add a custom configuration policy, for checking resources in the build phase of your application lifecycle.



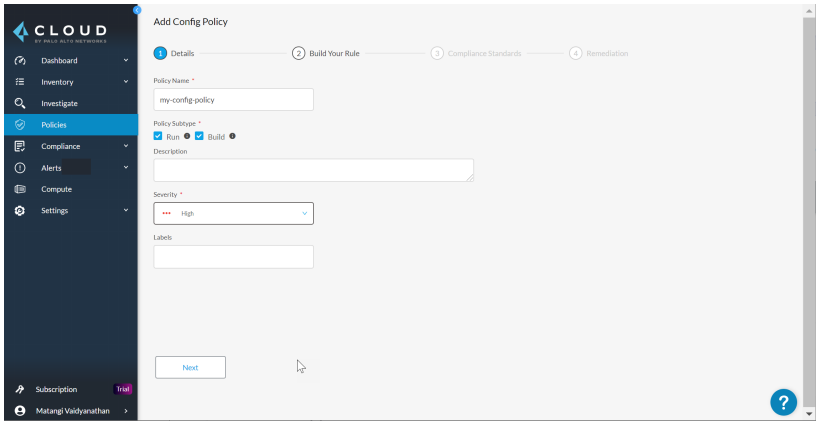
STEP 1 | Select **Policies** and click **Add Policy** > **Config**.

STEP 2 | Enter a **Policy Name**.

You can optionally add a **Description** and **Labels**.

STEP 3 | Select the **Build** policy subtype to scan code repositories and IaC templates—Terraform, CloudFormation, Kubernetes manifest and click **Next**.

You can choose one or both the policy subtypes options. The **Run** subtype enables you to scan cloud resources that are already deployed on a supported cloud platform.

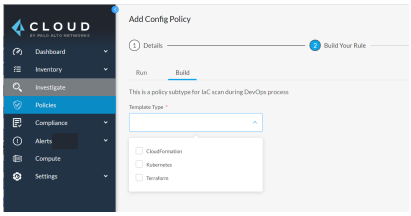


STEP 4 | Select the **Severity** for the policy and click **Next**.

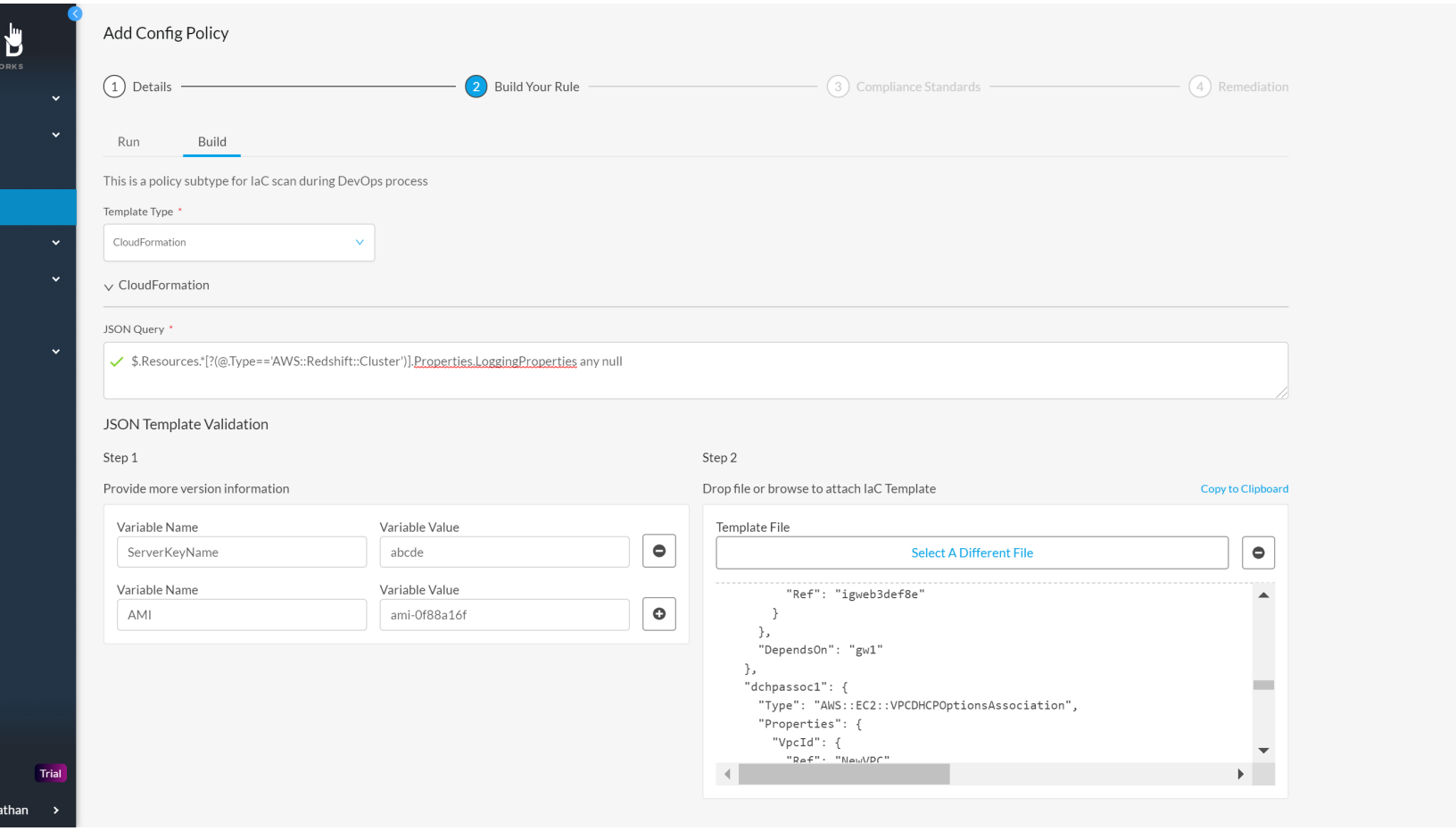
STEP 5 | Build the query to define the match criteria for your**Build** phase policy.

1. Select the **Template Type** you want to scan—CloudFormation, Kubernetes, or Terraform. You can add one or more types.

   For scanning Terraform templates, you must select the Cloud Type and the Terraform version. See what versions of Terraform are supported.



2. Add the JSON query that specifies the properties or objects for which you want to apply policy checks. For more information see Add a JSON Query for Build Policy Subtype and Prisma Cloud IAC Scan Policy Operators.



   If you choose to upload a template in the next step, the query you entered above is validated against the template. Each time you modify the query or upload a new template, the JSON query is re-validated.

3. (Optional) Upload a file to validate the JSON query.

   The JSON Template Validation is optional. You can upload a single file or a .zip file. The supported file formats are HCL,YAML, JSON. The uploaded file is converted to JSON and displayed on-screen.

In addition, you can include a variable name and value to pass to the sample file and verify that the build rule works before you save the policy. For example, if you want to check whether EC2 instances include tags to identify the owner, the variables enable you to quickly validate against the sample template you attached.

STEP 6 | Add the compliance standards to your policy.

1. Choose the compliance **Standard**, **Requirement**, and **Section**.
2. Click **+** to add more standards as required and click **Next**.

STEP 7 | Enter details in the remediation section, include remediation instructions on a policy violation.

Build phase policies do not support remediation CLI. You can however add the instructions for manually fixing the issue in the **Recommendation for Remediation**.

STEP 8 | **Submit** your changes.

Add a JSON Query for Build Policy Subtype

Policy rules to Secure Your Infrastructure Automation are written in JSON. So, you need to first convert your template file—CFT, Kubernetes or Terraform templates— to JSON and then parse the JSON structure to write the JSON query that correctly identifies the parameter and criteria on which you want to be alerted. See Build Policy Query Examples below.

STEP 1 | Convert your template file to JSON.

You can also use any online tool to convert your CFT, Kubernetes or Terraform templates to JSON. In this workflow, you use the following sample Terraform template and upload the template file to Prisma Cloud.

1. Get your template file.

   For example:

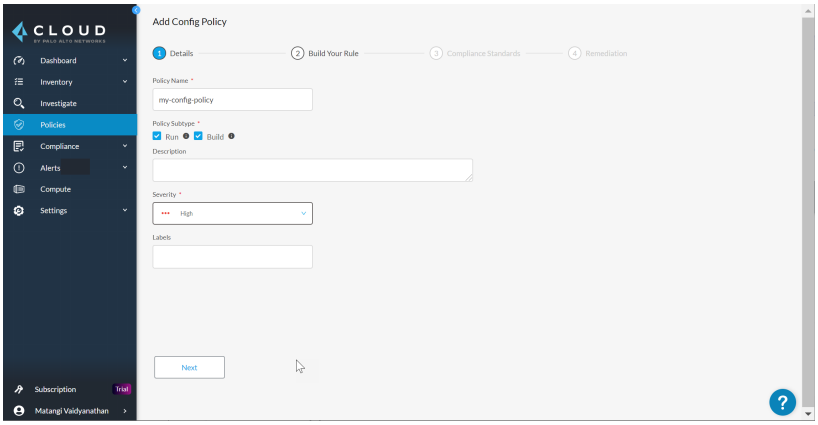
```
Resources:
   myTrail:
     DependsOn:
       - BucketPolicy
       - TopicPolicy
     Type: AWS::CloudTrail::Trail
     Properties:
       S3BucketName:
         Ref: S3Bucket
       SnsTopicName:
         Fn::GetAtt:
           - Topic
           - TopicName
       IsLogging: true
       IsMultiRegionTrail: true
  myTrail2:
     DependsOn:
       - BucketPolicy
       - TopicPolicy
     Type: AWS::CloudTrail::Trail
     Properties:
       S3BucketName:
         Ref: S3Bucket
       SnsTopicName:
         Fn::GetAtt:
           - Topic
```

```
            - TopicName
        IsLogging: true
        IsMultiRegionTrail: true
```

2. Select **Policies** and click **New Policy** > **Config**.
3. Enter a **Policy Name**.

   You can optionally add a **Description** and **Labels**.
4. Select the **Build** policy subtype and click **Next**.

   The build subtype enables you to scan IaC templates that are used to deploy cloud resources. For Run policy subtype, see Create a Configuration Policy.

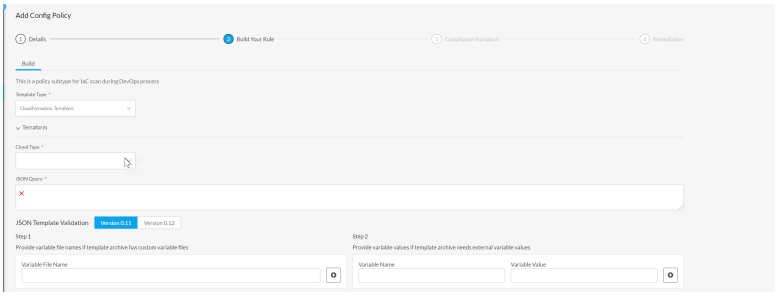


5. Select the **Severity** for the policy and click **Next**.
6. Build the query to define the match criteria for your policy.

   If your policy is for both **Run** and **Build** checks and you have added the RQL query, your cloud type for the build rule is automatically selected. It is based on the cloud type referenced in the RQL query. Otherwise, you must select the template type and the cloud type.

   1. Select the **Template Type** you want to scan—CloudFormation, Kubernetes, or Terraform. The supported files types are HCL, YAML, JSON.

      For scanning Terraform templates, you must select the Cloud Type and the Terraform version. See what versions of Terraform are supported.. For the other templates, you do not need to select the cloud type.



   2. Upload the template file for conversion to JSON in the JSON Template Validation section.

      In order to upload a file you must add a **JSON query** string. You can enter `object exists` to enable the ability to attach a file.

      Then, upload a single file or a .zip file.

      The sample template above when converted to JSON looks like this:

```
{
```

```
    "Resources": {
        "myTrail": {
            "Type": "AWS::CloudTrail::Trail",
            "Properties": {
                "S3BucketName": {
                    "Ref": "S3Bucket"
                },
                "IsLogging": true,
                "IsMultiRegionTrail": true
            }
        },
        "myTrail2": {
            "Type": "AWS::CloudTrail::Trail",
            "Properties": {
                "S3BucketName": {
                    "Ref": "S3Bucket"
                },
                "IsLogging": true,
                "IsMultiRegionTrail": true
            }
        }
    }
}
```

STEP 2 | Use JSON path validators to write your **JSON query** in the policy.

Use the following guidelines to parse the file structure and write the JSON query that specifies the properties or objects for which you want to apply policy checks:

- *$ - symbol refers to the root object or element.*
- *@ – symbol refers to the current object or element.*
- *. – operator is the dot-child operator, which you use to denote a child element of the current element.*
- *[ ] – is the subscript operator, which you use to denote a child element of the current element (by name or index).*
- ** – operator is a wildcard, that returns all objects or elements without regard of the name.*
- *? ( ) – to query all items that meet a certain criteria.*

For validating the JSON path, you can use validators such as https://jsonpath.com/ or others available on the internet. Each time you modify the query or upload a new template, Prisma Cloud revalidates your JSON query.

Refer to the list of Prisma Cloud IAC Scan Policy Operators to define the operators for the match.

If for example, you require that all AWS accounts across your organization have enabled AWS CloudTrail, you can check for violations where AWS CloudTrail is not enabled.

- **`$.Resources.*[?(@.Type=='AWS::CloudTrail::Trail')]`**, enables filtering the Resources whose type is **`AWS::CloudTrail::Trail`**
- and **`$.Resources.*[?(@.Type=='AWS::CloudTrail::Trail')].Properties.IsMultiRegionTrail 'any null'`** or **`$.Resources.*[?(@.Type=='AWS::CloudTrail::Trail')].Properties.IsMultiRegionTrail anyFalse`** enables you to further filter for the value specified for the MultiRegionalTrail parameter.

In this case, you are looking to identify a security issue when the value is missing or set to false. So, the match works as follows:If the Type: “AWS::CloudTrail::Trail”, then:

Parameter	Value	Outcome
IsMultiRegionalTrail	missing	Because the default is false, the rule will match. It means there a security issue detected.
IsMultiRegionalTrail	false	The rule will match. It means there is a security issue detected.
IsMultiRegionalTrail	true	The rule will not match. It means there is no security issue detected.

STEP 3 | Choose your next steps:

Continue to Step 6if you want to add compliance standards to the policy rule or to 5.b. Otherwise, **Save** the policy rule.

Build Policy Query Examples

The following section shows you an example of a Terraform template and an AWS CloudFormation Template (CFT).

- Sample JSON file (after being converted from Terraform)
  1. View the file contents.

```
      {
"data": [
  {
    "azurerm_client_config": [
      {
        "current": [
          {}
        ]
      }
    ]
  }
],
"provider": [
  {
    "azurerm": [
      {
        "features": [
          {
            "key_vault": [
              {
                "purge_soft_delete_on_destroy": true
              }
            ]
```

```
            }
          ]
        }
      ]
    }
  ],
  "resource": [
    {
      "azurerm_resource_group": [
        {
          "example": [
            {
              "location": "West US",
              "name": "resourceGroup1"
            }
          ]
        }
      ]
    },
    {
      "azurerm_key_vault": [
        {
          "example": [
            {
              "access_policy": [
                {
                  "key_permissions": [
                    "list"
                  ],
                  "object_id": "11111111-2222-3333-4444-555555555555",
                  "secret_permissions": [
                    "list"
                  ],
                  "storage_permissions": [
                    "get"
                  ],
                  "tenant_id": "2111-3333-4445-555"
                }
              ],
              "enabled_for_disk_encryption": true,
              "location": "azurerm_resource_group.example.location",
              "name": "testvault",
              "network_acls": [
                {
                  "bypass": "AzureServices",
                  "default_action": "Deny"
                }
              ],
   "purge_protection_enabled": false,
              "resource_group_name": "def",
              "sku_name": "standard",
              "soft_delete_enabled": true,
              "tags": [
                {
                  "environment": "Testing"
                }
              ],
              "tenant_id": "2111-3333-4445-555"
            }
          ]
        }
      ]
```

```
    }
  ]
}
```

2. Define the match criteria for the policy.

   This following query checks that the template for an Object ID match. It checks whether the object ID of the user or service principal in Azure Active Directory that is granted permissions matches your organizational policy.

```
          $.resource[*].azurerm_key_vault.*[*].*.access_policy exists
and $.resource[*].azurerm_key_vault.*[*].*.access_policy[*].object_id
== "11111111-2222-3333-4444-555555555555"
```

- Original CFT file that defines the security groups and the ports that allow ingress traffic.

1.

```
          AWSTemplateFormatVersion: '2010-09-09'
Parameters:
  testDescription:
    Description: Tests for blocked ports negative case in AWS Security
 Groups
    Type: String
Resources:
  myELB:
    Type: AWS::ElasticLoadBalancing::LoadBalancer
    Properties:
      AvailabilityZones:
      - eu-west-1a
      Listeners:
      - LoadBalancerPort: '80'
        InstancePort: '80'
        Protocol: HTTP
  myELBIngressGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: ELB ingress group
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
        SourceSecurityGroupOwnerId:
          Fn::GetAtt:
          - myELB
          - SourceSecurityGroup.OwnerAlias
        SourceSecurityGroupName:
          Fn::GetAtt:
          - myELB
          - SourceSecurityGroup.GroupName
  myELBIngressGroup2:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: ELB ingress group
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
```

```
        CidrIp6: "::/0"
        SourceSecurityGroupOwnerId:
          Fn::GetAtt:
          - myELB
          - SourceSecurityGroup.OwnerAlias
        SourceSecurityGroupName:
          Fn::GetAtt:
          - myELB
          - SourceSecurityGroup.GroupName
```

2. Sample JSON file (after being converted from AWS CFT)

```
          {
   "AWSTemplateFormatVersion": "2010-09-09",
   "Parameters": {
   "testDescription": {
     "Description": "Tests for blocked ports negative case in AWS
Security Groups",
     "Type": "String"
   }
 },
   "Resources": {
       "myELB": {
           "Type": "AWS::ElasticLoadBalancing::LoadBalancer",
           "Properties": {
               "AvailabilityZones": [
                   "eu-west-1a"
               ],
               "Listeners": [
                   {
                       "LoadBalancerPort": "80",
                       "InstancePort": "80",
                       "Protocol": "HTTP"
                   }
               ]
           }
       },
       "myELBIngressGroup": {
           "Type": "AWS::EC2::SecurityGroup",
           "Properties": {
               "GroupDescription": "ELB ingress group",
               "SecurityGroupIngress": [
                   {
                       "IpProtocol": "tcp",
                       "FromPort": 22,
                       "ToPort": 22,
                       "CidrIp": "0.0.0.0/0",
                       "SourceSecurityGroupOwnerId": {
                           "Fn::GetAtt": [
                               "myELB",
                               "SourceSecurityGroup.OwnerAlias"
                           ]
                       },
                       "SourceSecurityGroupName": {
                           "Fn::GetAtt": [
                               "myELB",
     "SourceSecurityGroup.GroupName"
                           ]
                       }
                   }
```

```
                ]
            }
        },
        "myELBIngressGroup2": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "ELB ingress group",
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "22",
                        "ToPort": "22",
                        "CidrIp6": "::/0",
                        "SourceSecurityGroupOwnerId": {
                            "Fn::GetAtt": [
                                "myELB",
                                "SourceSecurityGroup.OwnerAlias"
                            ]
                        },
                        "SourceSecurityGroupName": {
                            "Fn::GetAtt": [
                                "myELB",
                                "SourceSecurityGroup.GroupName"
                            ]
                        }
          }
                ]
            }
        }
    }
}
```

3. Define the policy match

   Check for any IPv4 or IPv6 CIDR range that allows unrestricted access for ingress traffic on port 22.

   ```
   $.Resources.*[?(@.Type ==
   'AWS::EC2::SecurityGroup')].Properties.SecurityGroupIngress[?(@.IpProtocol
   == 'tcp' && @.FromPort == '22' && @.ToPort == '22' && @.CidrIp
   == '0.0.0.0/0')] size greater than 0 or $.Resources.*[?(@.Type ==
   'AWS::EC2::SecurityGroup')].Properties.SecurityGroupIngress[?(@.IpProtocol
   == 'tcp' && @.FromPort == '22' && @.ToPort == '22' && @.CidrIp6 ==
   '::/0')] size greater than 0
   ```

   This rule will match as follows:

Parameter	Value	Outcome
IpProtocol	tcp	The rule will match.
FromPort	22	Security issue found.
ToPort	22	
CidrIp	0.0.0.0/0	
IpProtocol	tcp	The rule will match.
FromPort	22	Security issue found.
ToPort	22	

Parameter	Value	Outcome
CidrIp6	::/0	
IpProtocol FromPort ToPort CidrIp6	ftp 23 23 0.0.0.0/0 or ::/0	The rule will not match. No security issue found.
IpProtocol FromPort ToPort CidrIp6	tcp 22 22 4.0.0.0/0 or ::/16	The rule will not match because an IP address restriction is in place. No security issue found.

- The JSON query on Prisma Cloud does not support the following cases:

  Embedded policies or similar structures that are represented as a JSON string instead of JSON elements.

  Example: If your resource template is like this:

```
  resource "aws_iam_role" "nat" {
  name = "${local.infra}-nat"
  path = "/"

  assume_role_policy =<<EOF
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Principal": {
              "Service": "ec2.amazonaws.com"
            },
            "Effect": "Allow",
            "Sid": ""
        }
    ]
}
EOF
}
```

  When converted to JSON, the **`assume_role_policy`** is a json string within json. While Prisma Cloud can fetch the complete policy, it does not support the ability to filter the parameters inside it.

```
   {
            "aws_iam_role": [
                {
                    "nat": [
                        {
                            "path": "/",
                            "name": "${local.infra}-nat",
```

```
                                "assume_role_policy": "{\n    \"Version
\": \"2008-10-17\",\n    \"Statement\": [\n  {\n  \"Action\":
 \"sts:AssumeRole\",\n    \"Principal\": {\n  \"Service\":
 \"ec2.amazonaws.com\"\n },\n  \"Effect\": \"Allow\",\n  \"Sid\":
 \"\"\n }\n]\n}\n"
                            }
                        ]
                    }
                ]
            },
```

- Filtering with multiple criteria or with literal * matching.

  Example: An IAM policy that allows full administrative permissions, that is access to all AWS actions and resources, is a policy that contains a statement with

  "Effect": "Allow" for "Action": "*" over "Resource": "*", i.e. "Statement": [{ "Effect": "Allow", "Action": "*", "Resource": "*" }].

  The following JSON equivalent for the policy above is not supported on Prisma Cloud:

```
$.Resources.*[?
(@.Type=='AWS::IAM::Policy')].Properties.PolicyDocument.Statement[?(
  @.Effect == 'Allow'  && @.Action == '*' && @.Resource == '*' )] exists
```

Prisma Cloud IAC Scan Policy Operators

For Prisma Cloud DevOps Security, you can create configuration policies to scan your Infrastructure as Code (IaC) templates that are used to deploy cloud resources. The policies used for scanning IaC templates use a JSON query instead of RQL. The following list of operators are available for use in a JSON query, when you Add a JSON Query for Build Policy Subtype and specify the properties or objects for which you want to apply policy checks.

Operator	Usage Examples
'greater than' | ' > '	`$.spec.template.spec.containers[*].securityContext.runAsUs greater than 1`
'less than' | ' < '	`$.spec.template.spec.containers[*].securityContext.runAsUs less than 9999`
' equals ' | '=='	`$.resource[*].aws_iam_account_password_policy[*].*[*].pass equals 0`
'does not equal'	`$.resource[*].aws_vpc_peering_connection[*].*[*].peer_vpc does not equal $.resource[*].aws_vpc_peering_connection[*].*[*].vpc_id`

Operator	Usage Examples
'starts with' | 'startsWith'	`$.resource[*].aws_eks_cluster[*].*[*].version starts with 1.9.`
'does not start with' | '!startsWith'	`$.resource[*].aws_eks_cluster[*].*[*].version does not start with 1.9.`
'ends with' | 'endsWith'	`$.data[*].google_iam_policy[*].*[*].binding[?( @.role=='roles/editor' || @.role=='roles/owner' )].member does not end with ".gserviceaccount.com"`
'does not end with' | ' !endsWith'	`$.data[*].google_iam_policy[*].*[*].binding[?( @.role=='roles/editor' || @.role=='roles/owner' )].member does not end with ".gserviceaccount.com"`
'contains' 'includes one'	`$.Resources.*[?(@.Type == 'AWS::SQS::Queue')].Properties.KmsMasterKeyId contains alias/aws/sqs`
'does not contain' | '!contains'	`$.Resources.*[?(@.Type == 'AWS::SQS::Queue')].Properties.KmsMasterKeyId does not contain alias/aws/sqs`
'is empty' | 'isEmpty'	`$.resource.*.google_container_cluster.*.*.*.master_auth.* is empty`
'is not empty' | ' !isEmpty'	`$.resource.*.google_container_cluster.*.*.*.master_auth.* is not empty`
'any empty' | 'anyEmpty'	`$.Resources.*[?( @.Type == 'AWS::CloudTrail::Trail' )].Properties.KMSKeyId none empty`
'none empty' | 'noneEmpty'	`$.Resources.*[?( @.Type == 'AWS::CloudTrail::Trail' )].Properties.KMSKeyId none empty`

Operator	Usage Examples
'all empty' | 'allEmpty'	`$.Resources.*[?( @.Type == 'AWS::CloudTrail::Trail' )].Properties.KMSKeyId all empty`
'any null' | 'anyNull'	`$.Resources.*[? (@.Type=='AWS::S3::Bucket')].Properties.LoggingConfigurati any null`
'exists'	`$.Resources.*[?(@.Type == 'AWS::Elasticsearch::Domain')].Properties.VPCOptions exists`
'does not exist' | '!exists'	`$.Resources.*[?(@.Type == 'AWS::Elasticsearch::Domain')].Properties.VPCOptions does not exist`
'any start with' | 'anyStartWith'	`$.resource[*].aws_eks_cluster[*].*[*].version any start with 1.9.`
'none start with' | 'noneStartWith'	`$.resource[*].aws_eks_cluster[*].*[*].version none start with 1.9.`
'all start with' | 'allStartWith'	`$.resource[*].google_container_node_pool.*[*].*.node_confi all start with cos`
'any end with' | 'anyEndWith'	`$.data[*].google_iam_policy[*].*[*].binding[? ( @.role=='roles/editor' || @.role=='roles/ owner' )].members any end with ".gserviceaccount.com"`
'none end with' | 'noneEndWith'	`$.data[*].google_iam_policy[*].*[*].binding[? ( @.role=='roles/editor' || @.role=='roles/ owner' )].members none end with ".gserviceaccount.com"`
'all end with' | 'allEndWith'	`$.data[*].google_iam_policy[*].*[*].binding[? ( @.role=='roles/editor' || @.role=='roles/ owner' )].members all end with ".gserviceaccount.com"`

Operator	Usage Examples
'any equal' | 'anyEqual'	`$.Resources.*[? (@.Type=='AWS::S3::Bucket')].Properties.AccessControl any equal PublicReadWrite`
'none equal' | 'noneEqual'	`$.Resources.*[? (@.Type=='AWS::S3::Bucket')].Properties.AccessControl none equal PublicReadWrite`
'all equal' | 'allEqual'	`$.resource.*.azurerm_storage_account[*].*[*].network_rules all equal "AzureServices"`
' size equals ' | ' size == '	`$.resource.*.azurerm_monitor_log_profile[*].*[*].retentio size equals 0`
' size does not equal ' | ' size != '	`$.resource.*.azurerm_monitor_log_profile[*].*[*].retentio size does not equal 0`
' size greater than ' | ' size > '	`$.Resources.*[?(@.Type == 'AWS::EC2::SecurityGroup')].Properties. SecurityGroupIngress[?(@.IpProtocol == 'tcp' && @.FromPort == '22' && @.ToPort == '22' && @.CidrIp == '0.0.0.0/0')] size greater than 0`
' size less than ' | ' size < '	`$.resource.*.azurerm_monitor_log_ profile[*].*[*].retention_policy size less than 10`
'length equals' | 'length =='	
'length does not equal' | 'length !='	
'length greater than' | 'length >'	
'length less than' | 'length <'	
'any true' | 'anyTrue'	`$.resource[*].aws_db_event_subscription[*].*[? (@.source_type=='db-security-group')].enabled any true`
'none true' | 'noneTrue'	`$.resource[*].aws_db_event_subscription[*].*[? (@.source_type=='db-security-group')].enabled none true`

Operator	Usage Examples
' all true' | ' allTrue'	`$.resource[*].aws_db_event_subscription[*].*[?(@.source_type=='db-security-group')].enabled all true`
'any false' | 'anyFalse'	`$.resource[*].aws_db_event_subscription[*].*[?(@.source_type=='db-security-group')].enabled any false`
none false' | 'noneFalse'	`$.resource[*].aws_db_event_subscription[*].*[?(@.source_type=='db-security-group')].enabled none false`
' all false' | ' allFalse'	`$.resource[*].aws_db_event_subscription[*].*[?(@.source_type=='db-security-group')].enabled all false`
'is true' | 'isTrue'	`$.resource[*].aws_db_event_subscription[*].*[?(@.source_type=='db-security-group')].enabled is true`
'is false' | 'isFalse' 'is type' | 'isType' 'is not type' | '!isType'	`$.resource[*].aws_db_event_subscription[*].*[?(@.source_type=='db-security-group')].enabled is false`
'is member of' | 'isMemberOf'	`$spec.containers[*].securityContext.capabilities.add[*] is member of (FSETID, SETUID, SETGID,SYS)`
'is not member of' | '!isMemberOf'	`$spec.containers[*].securityContext.capabilities.add[*] is not member of (FSETID, SETUID, SETGID,SYS)`
IDENTIFIER '[]' | IDENTIFIER '[*]' | IDENTIFIER '[' INT ']' | IDENTIFIER '[?(' query_expr ')]' | '[*]'	`$resource[*].google_compute_subnetwork[*] $.Resources.*[?(@.Type=='AWS::S3::Bucket')].Properties`

Operator	Usage Examples
query_expr ('&&' query_expr)+ | query_expr ('||' query_expr)+ | '@.' IDENTIFIER | '@.' IDENTIFIER '>' INT | '@.' IDENTIFIER '<' INT | '@.length-' INT | '@.' IDENTIFIER '==' ('true' | 'false') | '@.' IDENTIFIER '!=' ('true' | 'false') | '@.' IDENTIFIER '==' INT | '@.' IDENTIFIER '==' RS_STRING | '@.' IDENTIFIER '!=' INT | '@.' IDENTIFIER '!=' RS_STRING | '@.' IDENTIFIER '==' SNGL_QUOTE (NUMERIC_VALUE | INT) SNGL_QUOTE | '@.' IDENTIFIER '!=' SNGL_QUOTE (NUMERIC_VALUE | INT) SNGL_QUOTE | '@.' IDENTIFIER '==' WILDCARD	`$.resource[*].aws_network_acl.*[*].*.egress[?(@.protocol == 'tcp' && @.from_port == '22' && @.to_port == '22')].action==allow`

Create an Alert Rule for Build-Time Checks

Alert rules for build-time checks enable you to detect issues early in your production process. When you implement security practices and automated checks at the very beginning of the production cycle, you can reduce risk and compliance violations later in the asset or code management lifecycle.

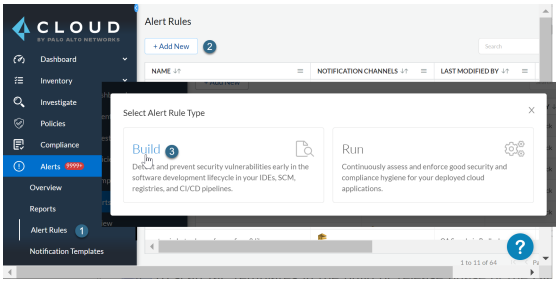
When you create a *build* alert rule, you select the configuration policies to which the rule applies and the corresponding set of tags for which you want to detect issues. When you set up the Prisma Cloud plugins, and specify the same tags in the config.yml file, the alert rule is matched with the tags to determine which policies you want to scan against and detect violations.

This workflow is for the Prisma Cloud devops security capabilities that is being replaced with a new Code Security module. The Code Security capabilities do not require an alert rule.

STEP 1 | Select **Alerts** > **Alert Rules** and **+Add New** alert.

Alert rules for build-time checks do not generate alerts on Prisma Cloud. You can view the scan results and trends for the IaC templates used in your processes and set up guardrails earlier in your business operations.

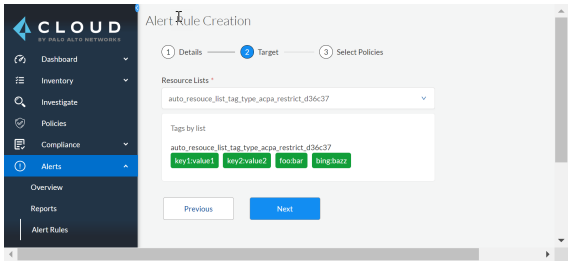
STEP 2 | Select **Build**.



STEP 3 | Enter an **Alert Rule Name** and, optionally, a **Description** to communicate the purpose of the rule and then click **Next**.

STEP 4 | Select the **Resource List** to which you want this alert rule to apply and then click **Next**.

The tags (key: value pairs) that you have defined within the resource list display.The policies that you select in this alert rule are used to scan IaC templates that match the tags listed in the attached resource list. To set one up, see Add a Resource List on Prisma Cloud.



- *When the IaC templates match the tags attached with the rule, the templates are scanned against the policies you select in the next step. Otherwise, the templates are scanned against all Prisma Cloud Configuration Policies of Subtype Build to detect any security issues. The scan used the failure criteria you defined when setting up the respective Prisma Cloud plugin, app, or extension to determine the scan result.*

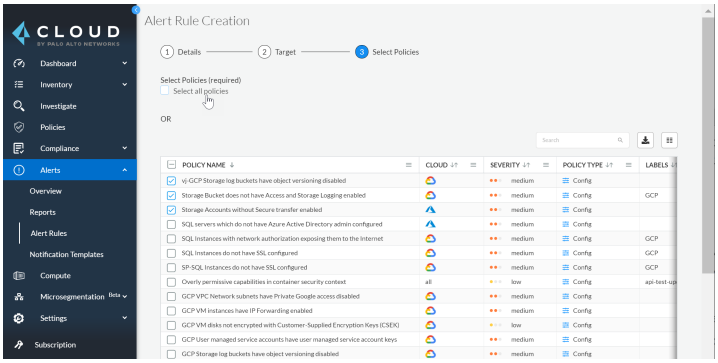
- *If you add multiple resource lists to an alert rule, each IaC template must match on all the tags includes in the resource lists.*

  If you have multiple alert rules that match on the tags in an IaC template, the templates are scanned against all policies included in the rules.

STEP 5 | Select the policies you want this alert rule to scan.

Either **Select All Policies** or select the specific policies that you want to scan.To help you find the specific group of policies for which you want this rule to alert

- **Filter Results**—Enter a search term to filter the list of policies to those with specific keywords.
- **Column Picker**—Select ( ) to modify which columns to display.
- **Sort**—Click the corresponding **Sort** icon ( ) to sort on a specific column.



STEP 6 | **Save** the alert rule.

STEP 7 | To view the scan results for policy violations on IaC templates that are scanned against these rules, select **Inventory** > **DevOps**.