Source

release/4.8.0 ··· | QScanner / **README.md**

 **Saket Sharad** authored `a501eb4090a` 4 days ago

Qualys CLI Scanner (QScanner)

Overview

QScanner is a spectacular command-line utility that transforms the way you scan for vulnerabilities. By providing inline vulnerability reports directly in your command-line interface, it integrates effortlessly into your existing workflow. Designed with enterprise needs in mind, QScanner enhances your security measures without adding any extra footprint to your ecosystem.

With QScanner, you're empowered to maintain robust security while enjoying a more streamlined and efficient scanning process. It's not just a tool, it's a new approach to container image security.

Features

Zero Installation

QScanner is a standalone executable - just download it and start scanning. No installation or deployment is needed, unlike traditional tools that require bulky setup and configurations.

Versatile Image Scanning

- **Local Runtimes:** Scan images from Docker, Containerd, or Podman.
- **Local Archives:** Analyze Docker images or OCI layouts from local files.
- **Remote Registries:** Connect to AWS ECR, Azure Container Registry, JFrog, GHCR, and more.

Immediate Results

Get instant vulnerability reports directly in your console, with output formats like SBOM, JSON, Table, and SARIF. This flexibility allows you to integrate QScanner seamlessly into various tools and workflows.

Fast Performance

Utilizes local caching for quicker scans. With supported storage drivers, data collection is even faster.

Policy Enforcement

QScanner offers centralized policy management through the Qualys Portal without disrupting engineering or DevOps workflows. Security teams can centrally define compliance standards and vulnerability thresholds, ensuring all scans automatically adhere to organizational policies. Developers and DevOps professionals continue using their preferred tools while QScanner enforces these policies in the background. This seamless integration facilitates a shift-left approach, addressing security concerns early without hindering productivity.

Comprehensive Security

QScanner offers robust security features that integrate seamlessly with your existing infrastructure. It leverages a trusted enterprise-grade vulnerability management system that you already trust and use to securely store and manage vulnerabilities.

Refer this to get started.

Table of Contents

Download

Published versions of QScanner are available here. You can get URL of the download script.

```
# Easily configure QScanner version to be used to allow controlled updates
# within your scan pipeline. It's recommended to always use the latest.
$ export QSCANNER_VERSION=v4.6.0

# This will download a script that allows you to download appropriate
# QScanner binary based on OS and architecture of Host machine.
$ wget https://www.qualys.com/qscanner/download/$QSCANNER_VERSION/download_qscanner.sh

# Permission to execute
$ chmod +x download_qscanner.sh

# Determine host OS and architecture and download the corresponding QScanner binary. It will also perform SHA256
checksum validation of the binary.
# QScanner binaries are signed. To validate its signature, you need to provide a public key via `-k <public-key>`
flag. For more details, run `download_qscanner.sh -h`
$ sh download_qscanner.sh

# Binary will be downloaded in specific OS-Arch directory
$ cd linux-amd64/

# To check QScanner version, use:
$ ./qscanner --version
```

General usage

```
./qscanner <global_options> command <command_options> target
```

Prerequisites

OS

QScanner binary is currently available for:

Sl	OS	Architecture	GA (General Availability)
1	Linux	amd64	✅
2	Linux	arm64	✅
3	Windows	amd64	For internal use only
4	Darwin	amd64	✅
5	Darwin	arm64	For internal use only

The `download_qscanner.sh` script automatically figures out your host OS and Architecture and downloads the required qscanner binary.

Write permission

QScanner requires write permission in the following paths:

Sl	Type	Used for	Default path	Can be overriden?
1	Output directory	Output artifacts like report, scan-result, changelist DB, SBOM etc.	`~/qualys/qscanner/data`	Using `--output-dir` flag.
2	Cache directory	To cache artifacts like java-db, qscanner-cache, secret-config etc.	`~/.cache/qualys/qscanner`	Using `--cache-dir` flag.
3	Temp directory	Temporary artifacts required during scan	`/tmp`	Using `TMPDIR` ENV.

Network connectivity

QScanner requires network access for below use cases, please make sure they are accessible from the host where QScanner is running:

- To scan remote images. The registry must be accessible.
- To perform SCA scan (when running in online-scan mode). This is required for downloading java-db which is an SQL database. For more details, refer section on Java Index Database. Below URLs must be accessible:
  - https://ghcr.io
  - https://pkg-containers.githubusercontent.com

If you are running in proxy environment, you can pass proxy details using `--proxy flag`. Example: `--proxy <username>@<password>:<url>:<port>`.

QScanner Blog

Link to QScanner blog.

Quick Start

Obtain client-id and client-secret from Qualys Guard portal.

Using QScanner Binary

Download QScanner binary as described here

```
export QUALYS_CLIENT_ID=<your-client-id>
export QUALYS_CLIENT_SECRET=<your-client-secret>
```

Run QScanner to collect vulnerabilities:

```
$ ./qscanner --pod US3 image centos
```

NOTE:

- List of supported pods can be found here.
- You might need to provide `--skip-verify-tls=true` if you are running in a proxy environment.
- For faster data collection you can make use of:
  - Cache
  - Storage Driver
- Once data collection is done, qscanner polls Qualys backend for fetching vulnerability report. The polling behavior can be controlled using `--network-retry-wait-min`, `-network-retry-wait-max` and `--max-network-retries` flags.

Sample output:

```
$ ./qscanner --pod US3 image debian

   ____    _____
  / __ \  / ___/  _____  ____ _   ____    ____   ___    _____
 / / / /  \__ \  / ___/ / __ `/  / __ \  / __ \ / _ \  / ___/
/ /_/ /  ___/ / / /__  / /_/ /  / / / / / / / //  __/ / /
\___\_\ /____/  \___/  \__,_/  /_/ /_/ /_/ /_/ \___/ /_/

By Qualys | version: 4.2.0-1

2025-03-28T08:20:36.805+0530    INFO    New instance of qscanner-4.5.0-1 started with invocation ID fc90dc02-6d29-
4642-96d0-5090fa5fb467
2025-03-28T08:20:36.808+0530    INFO    Fetching image details
2025-03-28T08:20:36.815+0530    INFO    Image source: docker-overlay2 filesystem
2025-03-28T08:20:36.883+0530    INFO    Starting Metadata scan
2025-03-28T08:20:36.885+0530    INFO    Metadata scan completed in 1.127281ms
2025-03-28T08:20:36.885+0530    INFO    Starting [os sca] scan
2025-03-28T08:20:36.919+0530    INFO    [os sca] scan completed in 33.859004ms
2025-03-28T08:20:36.919+0530    INFO    OS detected: Debian Linux 12.6
2025-03-28T08:20:36.919+0530    INFO    OS package(s) detected: 88
2025-03-28T08:20:36.919+0530    INFO    Technologies detected: 0
2025-03-28T08:20:36.919+0530    INFO    Language package(s) detected: 0
2025-03-28T08:20:36.919+0530    INFO    All scans completed in 35.486895ms
2025-03-28T08:20:36.921+0530    INFO    Scan Result JSON created at
/root/qualys/qscanner/data/2e5b8d3ef33ebde2b2a943fadb241a63de1535a007a7bd185e7037abfff99f63-ScanResult.json
2025-03-28T08:20:36.927+0530    INFO    SPDX document created at
/root/qualys/qscanner/data/2e5b8d3ef33ebde2b2a943fadb241a63de1535a007a7bd185e7037abfff99f63-ScanResult.spdx.json
2025-03-28T08:20:36.992+0530    INFO    Uploading SBOM
/root/qualys/qscanner/data/2e5b8d3ef33ebde2b2a943fadb241a63de1535a007a7bd185e7037abfff99f63-ScanResult.spdx.json.xz
2025-03-28T08:20:38.114+0530    INFO    SBOM uploaded successfully
2025-03-28T08:20:38.114+0530    INFO    Creating changelist
2025-03-28T08:20:38.258+0530    INFO    Compressed Changelist DB created at
/root/qualys/qscanner/data/2e5b8d3ef33ebde2b2a943fadb241a63de1535a007a7bd185e7037abfff99f63-ChangeList.db.xz
2025-03-28T08:20:38.258+0530    INFO    Uploading changelist
/root/qualys/qscanner/data/2e5b8d3ef33ebde2b2a943fadb241a63de1535a007a7bd185e7037abfff99f63-ChangeList.db.xz
2025-03-28T08:20:40.356+0530    INFO    Changelist uploaded successfully
2025-03-28T08:20:40.356+0530    INFO    Skip fetching of policy evaluation result
2025-03-28T08:20:40.356+0530    INFO    Fetching vulnerability report
2025-03-28T08:20:47.474+0530    INFO    Vulnerability report received
2025-03-28T08:20:47.475+0530    INFO    3 vulnerabilities found!
2025-03-28T08:20:47.475+0530    INFO      Severity 5: 0
2025-03-28T08:20:47.475+0530    INFO      Severity 4: 1
2025-03-28T08:20:47.475+0530    INFO      Severity 3: 2
2025-03-28T08:20:47.475+0530    INFO      Severity 2: 0
2025-03-28T08:20:47.475+0530    INFO      Severity 1: 0
Vulnerabilities:
+------------------+---------+-----+----------+----------------+------------------+------------------+------------
--------------------+
|   SOFTWARE NAME  |   QID   | QDS | VULN AGE |     CVE(S)     | INSTALLED VERSION |  FIXED VERSION   |
SUMMARY            |
+------------------+---------+-----+----------+----------------+------------------+------------------+------------
--------------------+
| libc-bin         | 6019286 | 35  | 10 days  | CVE-2025-0395  | 2.36-9+deb12u7   | 2.36-9+deb12u10  | Debian
Security Update for     |
|                  |         |     |          |                |                  |                  | glibc (CVE-
2025-0395)         |
+------------------+         +     +          +                +                  +                  +
+
| libc6:amd64      |         |     |          |                |                  |                  |
|
|                  |         |     |          |                |                  |                  |
|
+------------------+---------+-----+----------+----------------+------------------+------------------+------------
--------------------+
| libgnutls30:amd64 | 6017660 | 30  | 36 days  | CVE-2024-12243 | 3.7.9-2+deb12u3  | 3.7.9-2+deb12u4  | Debian
Security Update for     |
|                  |         |     |          |                |                  |                  | gnutls28
(CVE-2024-12243)     |
+------------------+---------+     +----------+----------------+------------------+------------------+------------
--------------------+
| libtasn1-6:amd64  | 6017622 |     | 43 days  | CVE-2024-12133 | 4.19.0-2         | 4.19.0-2+deb12u1 | Debian
Security Update for     |
|                  |         |     |          |                |                  |                  | libtasn1-6
(CVE-2024-12133)    |
```

```
+-------------------+---------+-----+----------+---------------+-------------------+------------------+-------------
--------------------+
2025-03-28T08:20:47.481+0530    INFO    For more details refer
/root/qualys/qscanner/data/2e5b8d3ef33ebde2b2a943fadb241a63de1535a007a7bd185e7037abfff99f63-Report.sarif.json
2025-03-28T08:20:47.481+0530    INFO    Skipping policy evaluation
```

Using QScanner Image

You can also use QScanner image that is available at Docker Hub.

By default image will spawned with root user which does not need any extra permissions for all mount paths (persistent cache, output_dir, docker socket, storage driver dir) from host.

Default sample command with root user

```
$ docker run --env QUALYS_CLIENT_ID=$QUALYS_CLIENT_ID \
             --env QUALYS_CLIENT_SECRET=$QUALYS_CLIENT_SECRET \
             --volume {local-cache-dir}:{cache-dir} \
             --volume /var/run/docker.sock:/var/run/docker.sock \
             qualys/qscanner:4.2.0-31 image sentry \
             --pod US3 \
             --proxy {proxy-url} \
             --cache local
```

To spawn image with non root user - Customers can specify user while spawning the image.

Default sample command with non root user

```
$ groupadd -r -g 555 $GROUP
$ useradd -r -u 555 -g $GROUP $USER
$ chown -R $USER:$GROUP $HOST_PATH
$ docker run -u 555 --env QUALYS_CLIENT_ID=$QUALYS_CLIENT_ID \
  env QUALYS_CLIENT_SECRET=$QUALYS_CLIENT_SECRET \
  --env HOME=$HOME --volume $HOST_PATH:{cache-dir} \
  --volume $HOST_PATH:{output-dir} \
  art-hq.intranet.qualys.com:5001/qualys/internal/qscanner:4.2.1-4-dev-linux-amd64-1727098790 image centos:7.8.2003
\
  --pod US1 -l debug --cache local
```

> **Note:** For non root user: All mounted host paths should have non root user permission else scan will fail with permission denied error.

Use Cases

Scanning for vulnerabilities.

QScanner detects known vulnerabilities according to the versions of installed packages. The following packages are supported:

- OS packages
- Language specific (SCA) packages

E.g:

```
$ ./qscanner --pod US3 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET image redhat/ubi8:latest
```

Scanning for secrets

QScanner scans any container image and filesystem to detect exposed secrets like passwords, API keys, and tokens. Secret scanning is disabled by default. Refer secret scanning. E.g:

```
$ ./qscanner --pod US3 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --scan-types pkg,secret -
-secret-config-rule my_secret_rules.json  image my_custom_image:latest
```

Scan on image push in AWS ECR

QScanner can be launched to scan for vulnerabilities in images that get pushed into Amazon ECR. The push event can be triggered via Event bridge. Refer this for more details.

Generating SBOM

You can just generate an SBOM for your target container image by running QScanner in `--mode inventory-only`. QScanner supports SBOM in the following formats:

- SPDX JSON
- SPDX TLV
- Cyclone DX

Refer output formats for more details. E.g:

```
$ ./qscanner --mode inventory-only --format spdx image centos
```

CI-CD pipeline

QScanner can be executed with `--mode evaluate-policy` and integrated in CI-CD pipeline. The exit code of QScanner could be used used to pass/fail builds. E.g:

```
$ ./qscanner --pod US3 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --mode get-report --
policy-tags dev,all image openjdk
```

Plug-in vulnerability scanner in the Harbor registry

QScanner can be executed as service which can deployed within Harbor Cluster with `harbor-registry-service` command. This service accepts scan job requests from Harbor Registry (as a **schedule**, **ondemand**, **on push to registry**) scans artifacts, uploads change list db to Qualys Backend, fetches vulnerability report from Qualys Backend and stores the report in Harbor Redis.

Details of Service configuration and deployment please refer - https://stash.intranet.qualys.com/projects/CSSEN/repos/adapter-svc-helm-chart/browse/qualys-harbor?at=refs%2Fheads%2Fdevelop

GitHub action

QScanner can be used to generate vulnerability report in SARIF format which can be consumed by GitHub Actions. Refer report formats for more details. E.g:

```
$ ./qscanner --pod US3 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --report-format
table,sarif image python
```

Refer this demo for QScanner integration in GithHub Actions.

Open Source Dependency

QScanner makes use of sca-data-collector (Qualys' fork of Trivy) to collect SCA packages. Most of the supported SCA languages make use of Trivy analyzers. Refer this for details of how sca-data-collector gets used in QScanner.

Feedback

You can submit feedback on usage here

Demo

v2.0.0

Demo link

02:34 - Overview
05:00 - General convention of CLI usage
05:28 - Supported commands
06:17 - Command Targets
08:35 - Scan modes
09:44 - Mode - Inventory Only
12:48 - Mode - Scan Only
13:40 - Requirements for backend communication
13:55 - Authentication mechanism
15:37 - Mode - Get Report
18:53 - Structure of Vulnerability report
21:00 - Mode - Evaluate Policy
23:45 - Policy
35:25 - CLI workflow based on Scan Mode
37:48 - Demo
44:53 - Report walkthrough
1:04:14 - Collecting inventory without backend interaction

4.0.0

Demo link

4.1.0

Demo link

4.2.0

- Demo link
- QScanner integration with Harbor Service Adapter

4.3.0

SBOM upload

4.3.1

docker-overlayfs and podman-overlay storage drivers

4.4.0

Demo link

4.5.0

Demo link

4.7.0

- Handoff Demo
- Container Scanning
- Repo and Lambda scanning

Supported Commands

QScanner supports following commands:

`image`:

Used to scan a container image. Supported runtimes can be found here. You can also make use of storage drivers for container runtime to avoid expensive image save opearation.

```
# To scan an image available on local runtime or from remote registry
$ ./qscanner --pod US2 image ubuntu

# If image is already pulled on local docker daemon
$ ./qscanner --pod US2 --storage-driver docker-overlay2 image ubuntu
```

`container` (Hidden):

Used to scan a running container. QScanner makes use of the container's filesystem for this. Refer section on storage drivers for more details. QScanner currently supports containers on following runtimes:

- **Docker:** via docker-overlay2 storage driver
- **Podman:** via podman-overlay storage driver

By default QScanner will scan the full filesystem of the given container. To scan only the writable layer, use `--scan-writable-layer-only=true`.

```
$ ./qscanner --mode inventory-only --format json container <container-id>

# To scan only the writable layer
$ ./qscanner --mode inventory-only --format json --scan-writable-layer-only=true container <container-id>
```

`tar`:

To scan an image archive. QScanner supports following formats of image tar:

- Docker archive
- OCI archive

```
$ ./qscanner --pod US2 tar /path/to/image.tar
```

`vmsnapshot` (Hidden):

To scan root directory of Host or mounted VM snapshots. Since root filesystem scanning can be huge, scan will take a lot of time. To perform non-OS scans (SCA, Secret and Malware) in specific directories only, you can use `--include-dirs <scan-type>=<comma separated paths>`. This will help speed up scans by performing such scans in custom paths only. Also, if you wish to allow collection of partial scan results, use `--allow-partial-scan=true` flag. E.g:

```
$ ./qscanner --mode inventory-only --format db,json --shell-commands "uname -a=x86_64" --scan-types
os,sca,secret,malware --include-dirs "sca=my/custom/path1,my/custom/path2" --allow-partial-scan=true vmsnapshot /
```

In the above example, if `my/custom/path1` was scanned and scan timed out during scan of `my/custom/path2`, then qscanner will log this as a warning but still collect the data of `my/custom/path1`.

This is only for internal use, will not be visible in usage help.

`winvmsnapshot` (Hidden):

To scan windows filesystem for vulnerabilities and sca. This accepts multiple volumes of windows at a time which should be comma separated. E.g:

```
qscanner.exe winvmsnapshot D:\,E:,\F:\
```

Supported scan-types are manifest, sca and winregistry.

> **Note:** For scan-type `manifest` and `winregistry`, scanning of a working/live windows system will not work as it will not be able to access the registry file from the same.

This is only for internal use, will not be visible in usage help.

`repo` (Hidden):

To scan code repository. This command is intended to scan pre-build artifacts. For list of supported files for each SCA languages, refer this section. Following sources are supported:

1. **Local:** Locally available source code.
2. **Archive:** Source code archived as zip, tar, etc.
3. **Remote:** Remote code repository like GitHub. To provide credentials or private repositories, you can make use of `--registry-username`, `--registry-password` and `--registry-token` flags. Refer this for more details on how to use these flags.

```
./qscanner --pod US2 --registry-username <username> --registry-token <token> repo https://github.com/jaygajera17/E-commerce-project-springBoot.git

# To exclude certain directories or files from scanning, you could specify patterns. For instance, in my Go project,
# I don't wish to scan unit test source code and files that are used by them. To avoid scanning in these paths, I can
# make use of exclude-dirs and exclude-files flags that accept comma separated glob patterns
 ./qscanner --pod US2 --registry-username <username> --registry-token <token> --scan-types pkg,secret --exclude-dirs **/testdata --exclude-files **/*_test.go  repo https://stash.intranet.qualys.com/scm/cssen/qscanner.git
```

`rootfs` (Hidden):

To scan root filesystem. This command is intended to scan post-build artifacts. For list of supported files for each SCA languages, refer this section.

```
# To scan host machine for OS vulnerabilities
$ ./qscanner --pod US2 --scan-types os --shell-commands 'uname -a=$(uname -a)' rootfs /

# Default scans (pkg+fileinsight) by excluding certain paths
$ ./qscanner --pod US2 --scan-types pkg,fileinsight --exclude-dirs /run,/var/lib --shell-commands 'uname -a=$(uname -a)' rootfs /

# If we are not performing OS scan, we don't require uname input
# JAR scanning
$ ./qscanner --pod US2 --shell-commands --scan-types sca rootfs /path/to/my/test.jar

# Binary scanning
$ ./qscanner --pod US2 --shell-commands --scan-types sca rootfs /usr/local/go/bin/go

# It can also be used to scan an unpacked container image filesystem
$ docker export $(docker create maven:3.8.3) | tar -C /tmp/maven-rootfs -xvf -
$ ./qscanner --pod US2 --shell-commands "uname -a=$UNAME_OUTPUT" rootfs /tmp/maven-rootfs
```

`lambda` (Hidden):

Used to scan an AWS Lambda. QScanner downloads the Lambda zip and performs scan on the extracted path. For this, it requires that the user has permissions to access AWS Lambda (`AWSLambda_FullAccess`) and ECR (`ECRfullacess`) IAM role. This command is intended to scan post-build artifacts. QScanner supports the following deployments:

- AWS Lambda Zip
- AWS Lambda Image

For credentials, you can either set the below environment variables or configure it using AWS CLI that creates config and credentials in `~/.aws/config` and `~/.aws/credentials` respectively. Region is mandatory because AWS Lambdas are tied to specific region.

```
# Set required AWS environment variables.
$ export AWS_ACCESS_KEY_ID=<access-key-id>
$ export AWS_SECRET_ACCESS_KEY=<secret-key>
$ export AWS_REGION=us-west-2

# Launch QScanner by providing the function name or the lambda ID (ARN)
$ ./qscanner --pod US2 lambda arn:aws:lambda:us-west-2:362990800442:function:QualysECSFargateImageScanningLambda
```

`configure`:

To create configuration file for qscanner. See here for more details.

`clear-local-cache`:

To clear local cache database. Local cache is created when running QScanner with `--cache local` option. For more details, check caching. The cache directory also holds the java-db (Java Index Database) which is is used when scanning JAR files. This is basically an SQlite DB that stores parsed indexes (containing `ArtifactID`, `GroupID`, `Version` and `sha1` for JAR files) from maven repository.

`tdatacollectorversion` (Hidden):

Show version of sca-data-collector being used. This is basically Qualys' fork of trivy that is used for data collection- i.e, gathering OS, SCA and Secrets from different artifacts. This is a hidden command, i.e, it won't show in QScanner help.

`show-scanner-version-hash`:

Lists hash of analyzer versions used for each scan-type. You can use this to determine whether data collection logic for respective scan-types has changed or not. E.g:

```
$ ./qscanner show-scanner-version-hash
compliance  : b9250ecbec6392b1a3676d204ecc194304090539
fileinsight : 644ac3ed45e8d5010ed9426e0f9771e060ca38ba
malware     : 2f7f1259f15cf746d64eef3d2a1a1f25656e8aa0
manifest    : 3cbd3d5b6f1b288bb66aa777982853e5db7acbf2
metadata    : 7457032858b723e69d0654cb823241103911207c
os          : b611524cb930925d956ae32dbe77a7b330301256
sca         : b00822014ece46997584aaafae9c130ba81da1c8
secret      : 1da2f668238e2df301b76100618f2f943e55963b
winregistry : 87c14abfd84240dbffe7c8f1d3b1f95db088e07a
```

`show-msft-sca-scanner-version` (Hidden):

This is a hidden command that has been introduced for MSFT use case (Linux and Windows Scan Utility). This will show a hard-coded value that represents SCA scanner version. Please refer `pkg/util/constants/versions.go` for more details on this.

`show-registered-analyzers` (Hidden):

This is a hidden command that lets developers see the supported analyzers for each scan-type. E.g:

```
$ ./qscanner show-registered-analyzers
Registered Analyzers:
+----+--------------------------+---------+---------------+
| SL |         ANALYZER         | VERSION |     TYPE      |
+----+--------------------------+---------+---------------+
| 1  | compliance-global        | 1       | compliance    |
| 2  | fileinsight              | 2       | fileinsight   |
| 3  | fileinsight-global       | 1       | fileinsight   |
| 4  | system-file-filter       | 3       | fileinsight   |
| 5  | unpackaged               | 1       | fileinsight   |
| 6  | malware-global           | 1.0.0-0 | malware       |
| 7  | manifest-global          | 1       | manifest      |
| 8  | metadata-global          | 1       | metadata      |
| 9  | alma                     | 1       | os            |
| 10 | alpine                   | 1       | os            |
| 11 | amazon                   | 1       | os            |
| 12 | apk                      | 4       | os            |
| 13 | bottlerocket-pkg-manager | 2       | os            |
```

```
| 14 | cbl-mariner              | 1       | os            |
| 15 | centos                   | 1       | os            |
| 16 | debian                   | 1       | os            |
| 17 | dpkg                     | 7       | os            |
| 18 | dpkg-license             | 1       | os            |
| 19 | fedora                   | 1       | os            |
| 20 | gentoo                   | 1       | os            |
| 21 | oracle                   | 1       | os            |
| 22 | os-global                | 1       | os            |
| 23 | os-release               | 1       | os            |
| 24 | pacman                   | 1       | os            |
| 25 | photon                   | 1       | os            |
| 26 | portage                  | 1       | os            |
| 27 | redhat                   | 1       | os            |
| 28 | rocky                    | 1       | os            |
| 29 | rpm                      | 4       | os            |
| 30 | rpmqa                    | 2       | os            |
| 31 | suse                     | 1       | os            |
| 32 | system-file-filter       | 3       | os            |
| 33 | ubuntu                   | 1       | os            |
| 34 | unpackaged               | 1       | os            |
| 35 | bundler                  | 1       | sca (Ruby)    |
| 36 | cargo                    | 1       | sca (Rust)    |
| 37 | composer                 | 1       | sca (PHP)     |
| 38 | composer-vendor          | 1       | sca (PHP)     |
| 39 | conda-environment        | 2       | sca (Python)  |
| 40 | conda-pkg                | 1       | sca (Python)  |
| 41 | dotnet-core              | 2       | sca (.Net)    |
| 42 | dotnet-runtime           | 1       | sca (.Net)    |
| 43 | gemspec                  | 1       | sca (Ruby)    |
| 44 | gobinary                 | 1       | sca (Go)      |
| 45 | gomod                    | 2       | sca (Go)      |
| 46 | gradle-lockfile          | 2       | sca (Java)    |
| 47 | jar                      | 1       | sca (Java)    |
| 48 | java-class               | 1       | sca (Java)    |
| 49 | node-pkg                 | 1       | sca (Node.js) |
| 50 | npm                      | 1       | sca (Node.js) |
| 51 | nuget                    | 3       | sca (.Net)    |
| 52 | packages-props           | 1       | sca (.Net)    |
| 53 | pip                      | 1       | sca (Python)  |
| 54 | pipenv                   | 1       | sca (Python)  |
| 55 | pnpm                     | 2       | sca (Node.js) |
| 56 | poetry                   | 1       | sca (Python)  |
| 57 | pom                      | 1       | sca (Java)    |
| 58 | pubspec-lock             | 2       | sca (Dart)    |
| 59 | python-egg               | 1       | sca (Python)  |
| 60 | python-pkg               | 2       | sca (Python)  |
| 61 | rustbinary               | 1       | sca (Rust)    |
| 62 | sbt-lockfile             | 1       | sca (Java)    |
| 63 | sca-global               | 1       | sca           |
| 64 | system-file-filter       | 3       | sca           |
| 65 | unpackaged               | 1       | sca           |
| 66 | yarn                     | 2       | sca (Node.js) |
| 67 | secret                   | 1       | secret        |
| 68 | secret-global            | 1       | secret        |
| 69 | system-file-filter       | 3       | secret        |
| 70 | unpackaged               | 1       | secret        |
| 71 | system-file-filter       | 3       | winregistry   |
| 72 | unpackaged               | 1       | winregistry   |
| 73 | winregistry              | 1       | winregistry   |
| 74 | winregistry-global       | 1       | winregistry   |
+----+--------------------------+---------+---------------+
```

`harbor-registry-service`:

This is a hidden command that has been introduced for spawning QScanner as a service which can be registered as Harbor Adapter to support scanning of images from Harbor registery. Please refer `https://stash.intranet.qualys.com/projects/CSSEN/repos/adapter-svc-helm-chart/browse/qualys-harbor` for more details on this.

`list-feature-toggles` (Hidden):

This is a hidden command that lists all the feature toggles. Fore more details, refer this.

`show-scan-metadata`:

Use this command to show the details of scan configuration without triggering scan. This command could be useful for consumers like Sensor to fetch scan configuration hash based on the given options. QScanner will show scan-metadata on console as well as a separate JSON file `<output-dir>/<invocation-id>-ScanMetadata.json`. Check this for more details.

```
# ./qscanner show-scan-metadata --scan-types pkg,secret,fileinsight,compliance --offline-scan=true
   ____    _____
  / __ \  / ___/  _____  ____ _   ____    ____   ___    _____
 / / / /  \__ \  / ___/ / __ `/  / __ \  / __ \ / _ \  / ___/
/ /_/ /  ___/ / / /__  / /_/ /  / / / / / / / //  __/ / /
\___\_\ /____/  \___/  \__,_/  /_/ /_/ /_/ /_/ \___/ /_/

By Qualys | version: dev-0

2025-11-17T11:46:48.566+0530    INFO    New instance of qscanner-dev-0 started with invocation ID eb488592-3cd7-
4c39-964d-a99ba0fc8936
{
    "ScanStartTime": 1763369208384592920,
    "ScanEndTime": 1763369208421286409,
    "ScanOptions": {
        "ScanTypes": [
            "os",
            "sca",
            "secret",
            "fileinsight",
            "compliance"
        ],
        "OutputFormats": [
            "db",
            "json",
            "spdx"
        ],
        "ReportFormats": [
            "table",
            "sarif"
        ],
        "ExcludeDirs": [],
        "ExcludeFiles": [],
        "IncludeDirs": {},
        "ScanTargetInfo": {},
        "ShellCommands": {},
        "ExitIfOSNotFound": false,
        "ExitIfOSPkgNotFound": false,
        "SecretConfigFile": "/root/.cache/qualys/qscanner/secret_config.json",
        "SecretConfigRefreshDuration": 86400000000000,
        "ForceSecretConfigRefresh": false,
        "OfflineScan": true,
        "SkipJavaDbUpdate": false,
        "JavaDbUpdateIntervalInDays": 15,
        "ScanTimeout": 300000000000,
        "Mode": 30,
        "Tags": [],
        "CacheType": "local",
        "CacheCleanupDurationThreshold": 3888000000000000,
        "CacheCleanupFrequency": 604800000000000,
        "EnableCacheCleanup": true,
        "DisabledSCALanguages": [],
        "InputCacheFile": "",
        "LimitResourceUsage": false,
        "MultiVolume": false,
        "MultiPartition": false,
        "StorageDriver": "none",
        "RegistryUsername": "",
        "RegistryPassword": "",
        "RegistryToken": "",
```

```
        "AllowPartialScan": false,
        "Platform": "",
        "ManifestPath": "",
        "DetectionPriority": "precise",
        "QDSThreshold": -1,
        "FileInsightConfigFile": "",
        "WinRegistryConfigFile": "",
        "MaskEnvVariables": false,
        "ScanWritableLayerOnly": false,
        "CollectBuildPipelineMetadata": false
    },
    "ScannerVersionHash": {
        "compliance": "b9250ecbec6392b1a3676d204ecc194304090539",
        "fileinsight": "50921895b8e7ead4d059190f4fb83f5b2d5d8712",
        "metadata": "7457032858b723e69d0654cb823241103911207c",
        "os": "ed04f50394d766d61213cf37cfc9273ccbc7ad11",
        "sca": "b00822014ece46997584aaafae9c130ba81da1c8",
        "secret": "1da2f668238e2df301b76100618f2f943e55963b"
    },
    "ScanConfigHash": {
        "compliance": "623761753452f71d893b3aa20fcb2f4805e0179d00f852ac04a670ab43dd971a",
        "fileinsight": "0016ef92b83c6cc01e5893a2d6c1d5a46d16f285af8eea0d53a34cfcb101f16d",
        "metadata": "4851d675fd439cc9709beda967098059bbd30376dde0191a3d53ae96681d3bb0",
        "os": "c7ac61e31e8682b6b3775c2d6b36be799cb896f2b649c0023bb7ac158277341d",
        "sca": "7feaf0ea3dd10d075903e94e1f75c6de808cc9ca00a33d0818d9b090147a1e0b",
        "secret": "5b8c0f5d6164693bd209f8d9a9e749389ed4d31ce7ce33ef272bdf933fec80f9"
    }
}
2025-11-17T11:46:48.572+0530    INFO    Scan metadata generated at /root/qualys/qscanner/data/eb488592-3cd7-4c39-964d-a99ba0fc8936-ScanMetadata.json
```


To view general help use:
`qscanner --help`

For command specific options:
`qscanner <command> --help`

Modes

QScanner can run in different modes depending on use case. Mode is controlled using the `--mode` flag. Each mode is incremental in nature, i.e., the subsequent modes are super set of all the operations that gets performed in the lower mode. Below are the supported modes:

1. Data collection

Option: `--mode inventory-only`.

In this mode QScanner will just perform data collection based on scan types that have been specified by the user. No data is sent to Qualys Backend and subsequently vuln evaluation is not performed.

QScanner will perform static scanning of the target to collect different types of information based on scan types that have been specified by the user. Scans to perform can be specified via `--scan-types` flag and can have multiple values like `os`, `sca`, `secret` etc. For more details, check section on Supported Scan Types.

Analogy: This is how Linux and Windows ScanUtility for MSFT works today. They just generate the ChangelistDB which then is separately provided to DPAAS for vuln evaluation.

Based on scan types, QScanner will create different Scanners that gather data from the specified target. If you just want to collect the inventory data, use `--mode inventory-only`. A JSON of the inventory will be created at the path specified by `--output-dir`. If any mode other than `inventory-only` is used, a changelist DB will also be created. See output formats for all supported output formats for inventory. There are several configurable parameters related to data collection. Please refer to the list of options supported by each command.

When running in `inventory-only` mode, if any of the scanner fails, QScanner will **NOT** exit with failure. Instead, it will simply log this as an error and continue with other scans. This behavior is helpful when QScanner is used as a data collector by different consumers. In that case, it makes sense to let QScanner complete all scans even if any particular scanner has failed. Consumers can check the scan status in `ScansPerformed` section of the `ScanResult.json`.

To make use of cache for faster data collection, refer caching. To limit CPU and memory utilization, use `--limit-resource-usage` option. This will help conserve system resources but will take more time. Refer this for more details.

2. Uploading data to backend

Option: `--mode scan-only`.

In this mode QScanner will perform data collection and upload the SBOM to Qualys backend. User will only be able to view the vulnerability on the UI. No console report will be available.

Analogy: This is how Sensor works today. It just scans and uploads SBOM to backend.

QScanner will perform LZMA compression on the Changelist DB and upload it in fragments. Internal users can modify fragment size using the `--fragment-size <fragment size in KB>` flag. The default is 399 KB. Depending on the options provided to `--scan-types`, multiple artifacts can be sent to backend like SBOM, Secret result etc. Scan results will be available in Qualys UI. For backend communication it is **mandatory** to pass certain parameters to the CLI:

- `--pod`: This is the POD that will be used for communicating with Qualys backend. For e.g: US1, EU1, IN1, CA1, AE1, UK1, AU1, KSA1, ENG-POD01, ENG-POD24 etc. More details on this here.
- `--client-id`: Provide client-id for authorization. The preferred way of providing this is via the `QUALYS_CLIENT_ID` environment variable. Preference will be given to the value passed via flag.
- `--client-secret`: Provide client-secret for authorization. The preferred way of providing this is via the `QUALYS_CLIENT_SECRET` environment variable. Preference will be given to the value passed via flag.

Apart from this, you might also need to configure the following depending on your environment:

- `--proxy <proxy_url>`: Use the given proxy. If not specified, QScanner will use the proxy server configured on your machine.
- `--skip-verify-tls`: If enabled, secure TLS verification will be skipped.
- `--cert-path <certificate_path>`: If a certificate path is provided, it will be used. If this option is provided, `skip-verify-tls` will be ignored.

3. Fetching report

Option: `--mode get-report` (Default).

In this mode QScanner will perform data collection, upload the SBOM to Qualys backend, retrieve the vuln report from backend and show it on console in a tabular format. Additionally, a detailed SARIF file is also generated. This report contains details of QIDs detected, CVE IDs associated with that QID, severity, software name, installed version and version in which its fix is available. See report formats for more details on report formats.

4. Evaluating policy

Option: `--mode evaluate-policy`.

QScanner can be used in a CI-CD pipeline as well. In such use cases it can make use of policy evaluation feature. Based on policies that are configured in the backend, QScanner can pass or fail the scan result of a target. The policy evaluation result will be available in the generated report. You can specify comma separated tags via `--policy-tags` flag to make use of Policy rules based on tag.

It can have 3 possibilities:

- `ALLOW`: Evaluation against the configured policies indicates success. QScanner will terminate with exit code `EXIT_SUCCESS`.
- `AUDIT`: None of the policies were evaluated, this will cause QScanner to terminate with exit code `EXIT_POLICY_EVALUATION_AUDIT`.
- `DENY`: Evaluation against the configured policies indicates that one or more rules failed. QScanner will terminate with exit code `EXIT_POLICY_EVALUATION_DENY`.

To see values of all possible exit codes, see this.

> **Note:** Backend combines vulnerability reports of same image SHA. In a scenario where instance 1 of QScanner scanned **Image A** with `--scan-types os,sca` and instance 2 of QScanner scanned **Image A** with `--scan-types os`, the vulnerability

> report will contain both os and sca data. This will apply to policy evaluation as well. Instance 2 of QScanner will get policy evaluation based on both os and sca.

Examples

Example 1:

```
$ ./qscanner image celery --log-level debug --mode inventory-only --pod US1 --client-id
$QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --output-dir /root/qscanner/data/
```

This will collect OS packages of *celery* image and create an LZMA compressed ChangeList DB file at `/root/qscanner/data/`. You can use `unxz` command to uncompress the changelist.

Example 2:

```
$ ./qscanner image pypy --scan-types os,sca --log-level debug --mode scan-only --gateway-url
https://gateway.p19.eng.sjc01.qualys.com --client-id $QUALYS_CLIENT_ID --client-secret
$QUALYS_CLIENT_SECRET
```

This will collect OS and SCA packages of *pypy* image, create an LZMA compressed ChangeList.db and upload it to the Qualys backend. You can view the scan result of this image on Qualys UI.

Example 3:

```
$ ./qscanner vmsnapshot /mnt/sda/sda1 --shell-commands "uname -a = x86_64" --scan-target-info
"provider_name=AWS" --scan-target-info "asset_uuid=ed52331a-6be6-4039-b592-0991984d8ece" --mode
inventory-only
```

This will collect OS and SCA packages for the VM snapshot mounted at `/mnt/vm_snapshot` and create an LZMA compressed ChangeList DB.

Supported Scan Types

You can use `--scan-types <scan-types>` flag to specify scans that should be performed. Options are `os`, `sca`, `secret`, `malware`, `fileinsight`, `compliance`, `manifest` and `winregistry`. If multiple scan-types are provided, all of them will be done.

Metadata

Metadata scan is always performed. You don't need to specify an option on enable this. This scan collects the following information:

Scan Metadata

1. `ScanStartTime`: Scan start time. Unix timestamp in nano-seconds. Please note that this is the timestamp of beginning of actual data collection. It does not considers the time required for preparation of the scan artifact like saving image tar, etc.
2. `ScanEndTime`: Scan end time. Unix timestamp in nano-seconds.
3. `ScanOptions`: Options used by qscanner to perform the scan. These options are provided to QScanner via flags or ENV variables. All the supported flags have a default value. If the flag is not explicitly provided, the default values get used.
4. `ScannerVersionHash`: This is a map of scans performed and their corresponding version hashes. This remains same for a specific version of QScanner. This result is similar to the output that is received by running `show-scanner-version-hash` command.
5. `ScanConfigHash`: This is a map of scans that were performed and the hash of scan configuration. This hash is generated by considering above 2 factors. This indicates whether the collection for the scan type for the given scan options has changed. This parameter can be used by consumers to determine cache keys. For more details on factors contributing to this hash, refer `pkg/analyzer/cachekey.go`.

This information is always collected when scanning any artifact. You can also generate it without actually launching a scan by running `show-scan-metadata` command.

Host Metadata

Information about the host on which QScanner is running, like Hostname, Host IP etc.

Target Metadata

Details about the target (image, filesystem etc.) getting scanned:

- OS
- Architecture
- Filesystem Metadata
- Image Metadata:
    - Image ID
    - Repo Tags
    - Repo Digests
    - Layer Inventory
    - Inspect Info

Build Pipeline Metadata

Information about the Build pipeline in which QScanner is executing. QScanner relies on ENV variables of the pipeline to extract the following details:

Sl	Parameters	GitLab	GitHub	Jenkins
1	Repository	`CI_REPOSITORY_URL`	`GITHUB_REPOSITORY`	`JOB_URL`
2	Branch	`CI_COMMIT_BRANCH`	`GITHUB_REF_NAME`	`BRANCH_NAME`
3	Commit ID	`CI_COMMIT_SHA`	`GITHUB_SHA`	`GIT_COMMIT`
4	Commit Author	`CI_COMMIT_AUTHOR`	`GITHUB_TRIGGERING_ACTOR`	`GIT_COMMITTER_NAME`

Custom Pipeline Metadata

TBD

Collection of CI pipeline information is enabled by default. To disable it, use `--collect-build-pipeline-metadata=false`.

OS

Option: `--scan-types os`.

OS scan broadly performs:

1. **OS fingerprinting:** Information about target OS. The OS name collected is in the format that Qualys' signature evaluation expects.
2. **Collecting OS packages:** List of OS installed packages.
3. **System Installed Files:** Collects list of files that were installed by OS packages.

Supported OSes

QScanner can scan targets based on the following Operating Systems:

- AlmaLinux
- Alpine Linux
- Amazon Linux
- Arch Linux
- AWS Bottlerocket Linux
- BellSoft Alpaquita Linux

- CBL-Mariner Linux
- CentOS
- Chainguard
- Debian
- Fedora
- Gentoo Linux
- Google Distroless (Debian Linux)
- Microsoft Azure Linux
- OpenSUSE
- Oracle Enterprise Linux
- Photon
- Red Hat Enterprise Linux Server
- Rocky Linux
- SUSE Linux Enterprise Server
- Ubuntu
- Wolfi

Supported Package managers

QScanner supports collection of OS installed packages from the following package managers:

- APK
- Bottlerocket Inventory file
- DPKG
- Portage (Gentoo Linux)
- RPM
- Pacman

SCA

Option: `--scan-types sca`.

SCA stands for Software Composition Analysis. With SCA scan it is possible to scan for vulnerabilities in the application dependencies.

Pre and Post Build artifacts

The files analyzed vary depending on the target. This is because QScanner primarily categorizes targets into two groups:

- **Pre-Build:** If the target is a pre-build project, like a code repository, QScanner will analyze files used for building, such as lock files, pom.xml, go.mod, etc.
- **Post-Build:** When the target is a post-build artifact, like a container image, QScanner will analyze installed package metadata like .gemspec, JARs, binary files, etc.

Why the differentiation?

If we were to scan both sets of files, there would be several problems:

1. **Duplication of results:** For example, it's a bit difficult to identify which JAR file was generated from which pom.xml, so the two results would overlap. This applies not only to JARs but also to Go and Rust binaries.
2. **Slower scan execution:** Scanning JAR files and the like takes time, so scanning both pom.xml and JARs reduces the efficiency of the scan. Even though it would have been sufficient to scan the pom.xml, it ends up scanning the JAR.

For these reasons, we divide the scan targets depending on the target. For detailed list of SCA packages analyzed for each target, refer this.

Target	Pre-Build	Post-Build
`image`	-	✅
`container`	-	✅

Target	Pre-Build	Post-Build
`tar`	-	✅
`vmsnapshot`	-	✅
`winvmsnapshot`	-	✅
`lambda`	-	✅
`repo`	✅	-
`rootfs`	-	✅

Supported Languages

QScanner supports collection of following language (SCA) based packages:

Sl	Language	File	Image, Container, Lambda, VM Snapshot	Repository
1	Ruby	Gemfile.lock	-	✅
		gemspec	✅	-
2	Rust	Cargo.lock	✅	✅
		Binaries built with cargo auditable	✅	-
3	PHP	composer.lock	-	✅
		installed.json	✅	✅
4	Java	JAR/WAR/PAR/EAR	✅	-
		pom.xml	-	✅
		gradle.lockfile	-	✅
		.sbt.lock	-	✅
5	Go	Binaries built with go	✅	-
		go.mod	-	✅
6	Python	Pipfile.lock	-	✅
		poetry.lock	-	✅
		requirements.txt	-	✅
		egg package	✅	-
		wheel package	✅	-
		conda package	✅	-
		conda environment (environment.yaml)	-	✅
7	.NET	packages.lock.json	✅	✅
		packages.config	✅	✅
		deps.json	✅	✅
		packages.props	✅	✅
		App.runtimeconfig.json	✅	✅
8	Node.js	package-lock.json	-	✅

Sl	Language	File	Image, Container, Lambda, VM Snapshot	Repository
		yarn.lock	-	✅
		pnpm-lock.yaml	-	✅
		package.json	✅	-
9	Dart	pubspec.lock	-	✅

To disable collection of packages for certain languages, use `--disable-sca-languages <languages_to_disable>`. The values to this flag are case insensitive. For e.g, below command will not collect Ruby, .NET and Node.js packages from the given image.

```
$ ./qscanner image sentry --disable-sca-languages ruby,.net,Node.js
```

Java Index Database (java-db)

Java Index database is an SQlite DB that stores ArtifactID, GroupID, Version and SHA1 for JAR files. This data is created by parsing all indices from Maven repository. It is created and maintained by Trivy and distributed via GitHub Container registry (GHCR). It is required for scanning JAR files. To understand why this is required, refer. In summary:

- If Package name is detected in `pom.properties` file of JAR, there is no dependency on java-db.
- Else, we figure out package by parsing `MANIFEST.mf` or SHA1 digest of JAR file by validating its existence in java-db (central MAVEN repository).

QScanner runs in online scan mode by default. In this, it downloads java-db from https://ghcr.io periodically. This database is used when scanning JAR files so that QScanner can identify the groupId, artifactId, and version of JAR files. QScanner tries to download new java-db after every 5 days (configurable via `--java-db-update-interval`). This gets downloaded in `java-db` directory within the QScanner cache directory `$USER_CACHE_DIR/qualys/qscanner/` or the path specified by `--cache-dir` flag. The default cache directory is `/root/.cache/qualys/qscanner/`.

> **Warning:** If java-db download fails, qscanner will fallback to `offline-scan`.

> **Warning:** Please note, when using the `clear-local-cache` command, it will clear the entire cache directory which includes local cache database and the java index database.

Supported Java DB repositories

In addition to ghcr, there are additional repositories where this java-db gets published. List of supported repositories are:

- `ghcr`: ghcr.io/aquasecurity/trivy-java-db
- `aws`: public.ecr.aws/aquasecurity/trivy-java-db
- `docker`: docker.io/aquasec/trivy-java-db

By default, only ghcr is used. You can override this behavior using `QSCANNER_ENABLED_JAVADB_REPOS` env variable. E.g:

```
export QSCANNER_ENABLED_JAVADB_REPOS=ghcr,aws,docker
```

This will allow QScanner to attempt download of java-db from all the provided repositories in case of failure. Please note, the order defined here is the order in which QScanner will try the repository to download java-db.

Offline Scan

It is possible to avoid downloading java-db and run in `--offline-scan=true`. Use this parameter to disable internet access for the SCA scan and run the scan in offline mode instead. It is recommended to run scan in online mode. Quality of software package enumeration for Java substantially degrades when the scan is run in offline mode. This can affect accuracy of the vulnerability posture of the image.

```
./qscanner image maven \
    --mode inventory-only \
```

```
    --format json \
    --offline-scan=true
```

Air gapped environment

Java-db can be used in an air gapped environment by using `--skip-java-db-update=true` flag and making it available in the QScanner cache directory manually. This way, QScanner won't check for updated java-db and will continue using the already available file.

```
# -------------------------
# Step 1: Oras installation
# -------------------------
# Oras is a CLI tool that is used to download files from OCI registry. Install oras CLI (reference:
https://oras.land/docs/installation/)
export VERSION="1.2.0"
curl -LO "https://github.com/oras-project/oras/releases/download/v${VERSION}/oras_${VERSION}_linux_amd64.tar.gz"
mkdir -p oras-install/
tar -zxf oras_${VERSION}_*.tar.gz -C oras-install/
sudo mv oras-install/oras /usr/local/bin/
rm -rf oras_${VERSION}_*.tar.gz oras-install/


# ----------------------------------------
# Step 2: Make java-db available to qscanner
# ----------------------------------------
# Download Java Index Database
oras pull ghcr.io/aquasecurity/trivy-java-db:1

# Move the extracted files to java-db directory in QScanner's cache directory.
mkdir -p /root/.cache/qualys/qscanner/java-db
tar -zxf javadb.tar.gz -C /root/.cache/qualys/qscanner/java-db
rm -rf javadb.tar.gz


# ----------------------------------------------------
# Step 3: Run QScanner with skip-java-db-update=true
# ----------------------------------------------------
./qscanner image maven \
    --mode inventory-only \
    --format json \
    --skip-java-db-update=true \
    --offline-scan=false
```

Secret

Option: `--scan-types secret`.

Secret scanning can be used to collect secrets based on a rule file. This file can be specified via `--secret-config-file <file_path>`. If the config file is not provided or does not contain any configurations, QScanner will try to download secret-config from Qualys Backend.

- QScanner will download secret-config from backend and store it in the cache directory (`{QSCANNER_CACHE_DIR}/secret_config.json`)
- On subsequent runs, it will make use of the cached rule file.
- This file will remain cached by default for 24 hours (configurable via `--secret-config-refresh-duration`) after which QScanner will attempt to check backend if there are any changes to the rules file.
- If no changes, then it will continue using the existing `secret_config.json`.
- If rules have updated (decided on the basis of content hash), then new secret-config file will be downloaded.
- To forcefully refresh the secret-config, use `--force-refresh-secret-config=true` flag.
- To use a custom secret-config rules file, use `--secret-config-file <path-to-rule-file>`. If this file is provided, QScanner will not attempt to download secret-config from Qualys backend.

Secret Config

Secret config file is basically a file that contains rules that define what should be treated as a secret. It contains the following parameters:

- **id:** Unique identifier for each rule.
- **category:** Rule category.
- **title:** User friendly name of the rule.
- **regex:** Actual regex that is matched against the file content to categorize it as secret. This is encoded in base64 format.
- **severity:** Severity of secret detected using this rule.
- **keywords:** These keywords are matched against the content of the file before evaluating the regex. If the file does not has any of the mentioned keywords, then this rule is not evaluated. If no keywords are provided, the rule will be evaluated against the regex.
- **show-secret-content:** By default, if a secret is detected, only its line number and filename is shown in the secret results. If the matched content of a rule is required to be revealed, then this field can be set to `true`.

Sample secret-config-file:


```
{
    "rules": [
        {
            "id": "5ec62a0c-c95f-11ed-afa1-0242ac120002",
            "category": "AWS",
            "title": "AWS Account Id",
            "regex":
"KD9pKShefFxzKylbIiddPyhhd3MpP18/YWNjb3VudF8/KGlkKT9bIiddP1xzKig6fD0+fD0pXHMqWyInXT8oP1A8c2VjcmV0PlswLTldezR9XC0/WzAtOV17
            "severity": "HIGH",
            "keywords": [
                "account"
            ],
            "show-secret-content": true
        },
        {
            "id": "d3fa61b2-c95f-11ed-afa1-0242ac120002",
            "category": "Github",
            "title": "GitHub Personal Access Token",
            "regex": "Z2hwX1swLTlhLXpBLVpdezM2fQ==",
            "severity": "CRITICAL",
            "keywords": [
                "ghp_"
            ],
            "show-secret-content": false
        }
    ]
}
```


Sample secret result:

Below is an example of `SecretResults` in the `ScanResult.json` file:


```
{
    "SchemaVersion": 5,
        "QScannerVersion": "4.2.0-1",
        "ScansPerformed": [
        {
            "ScanType": "metadata",
            "ScanDuration": 123206,
            "Status": "SUCCESS"
        },
        {
            "ScanType": "secret",
            "ScanDuration": 6112274,
            "Status": "SUCCESS"
        }
        ],
        "Metadata": {...}
        "SecretResults": [
        {
            "Target": "/root/my_credentials.txt",
                "Secrets": [
                {
                    "RuleID": "5ec62a0c-c95f-11ed-afa1-0242ac120002",
                    "Category": "AWS",
                    "Severity": "HIGH",
                    "Title": "AWS Account Id",
```

```
                    "StartLine": 7,
                    "StartColumn": 18,
                    "EndLine": 7,
                    "Match": "1234-5678-9123",
                    "Layer": {
                        "DiffID": "sha256:003e192cfb421ec7e8378543d1b6c829ed2b6ffb215195219eca1f95c46468f3"
                    }
                },
                {
                    "RuleID": "d3fa61b2-c95f-11ed-afa1-0242ac120002",
                    "Category": "Github",
                    "Severity": "CRITICAL",
                    "Title": "GitHub Personal Access Token",
                    "StartLine": 5,
                    "StartColumn": 20,
                    "EndLine": 5,
                    "Layer": {
                        "DiffID": "sha256:003e192cfb421ec7e8378543d1b6c829ed2b6ffb215195219eca1f95c46468f3"
                    }
                }
            ]
        }
        ]
    ...
}
```


To generate secret results in a separate file, you can specify `--format secret-result`. This will generate an LZMA compressed JSON file with just the `SecretResults` object. Refer this for more details.

NOTE: If you wish to show matched secret content for all detected secrets, you can set ENV `SHOW_SECRET_CONTENT=yes`. Please note, this is for internal use only and must not be exposed to customers.

Malware

Option: `--scan-types malware`.

Collect data about executable files that can be analyzed by backend and check if they are malicious or not. QScanner internally spawns **MLExtractor** binary to collect meta information (feature vector) about executable files. This feature vector is consumed by the *BlueHexagon* backend to classify binaries as *malicious*, *benign* or *unknown*.

Malware scan is dependent on **MLExtractor**. This is a data collector provided by *BlueHexagon*. Currently this binary is only available for *linux-amd64*. So this means that Malware scanning is only supported for *linux-amd64* binary of QScanner. For other platforms, if malware scan is performed, it will fail with below error:

```
Malware Scanner failed: Failed to create MLExtractor: MLExtractor binary not found.
```

File Insight

Option: `--scan-types fileinsight`.

Collect information (file path, size, permission, etc.) about files (based on enabled rules) encountered in the scan target. We have capability to collect following information depending upon the matched rule configuration:

- File path
- Permission
- File size in bytes
- Origin layer (applicable for image artifacts only)
- MIME type
- MD5 digest
- SHA1 digest
- SHA256 digest
- Capture groups
- Executable Info (available for binaries only)

- Name (ELF and PE)
- Version (ELF and PE)
- PURL (ELF and PE)
- Imported Libraries (ELF and PE)
- Product Name (PE only)
- Product Version (PE only)
- Company Name (PE only)
- Copyright (PE only)
- Description (PE only)
- Publisher (PE only)

Note: `fileinsight` scanner will not collect files that came as part of OS installation, i.e, it filters out system installed files from the final result. If you wish to collect **all** the files, then run only `fileinsight` scan (without other scan-types). E.g:

```
./qscanner --mode inventory-only --format json --scan-types fileinsight image python
```

Alternatively, you could use `--detection-priority comprehensive`. This will not filter out files installed by OS. E.g:

```
./qscanner --mode inventory-only --format json --scan-types pkg,fileinsight image python --detection-priority
comprehensive
```

Built-in Rules

Following are the built-in rules that are supported by `fileinsight` scan-type. For more details on these rules, refer `pkg/analyzer/fileinsight/builtin_config.go`.

Sl	Rule Name	Type	Enabled?	Purpose
1	`yum_repos`	Non-Binary	✅	Content of all files under `/etc/yum.repos.d/*.repo`
2	`dnf_modules`	Non-Binary	✅	Content of all files under `/etc/dnf/modules.d/*.module`
3	`content_manifests`	Non-Binary	✅	Content of all files under `/root/buildinfo/content_manifests/*.json`
4	`els_release`	Non-Binary	✅	Content of `/etc/els-release` file.
	`all_executables`	Binary	❌	Details of all executables/binaries.
	`bash_binary`	Binary	❌	Details of `**/bash` ELF.
	`busybox_binary`	Binary	❌	Details of `**/busybox` ELF.
	`chrome_binary`	Binary	❌	Details of `**/chrome` ELF.
	`curl_binary`	Binary	❌	Details of `**/curl` ELF.
	`firefox_binary`	Binary	❌	Details of `**/firefox.*` ELF.
	`nginx_controller_binary`	Binary	❌	Details of `**/nginx-ingress-controller` ELF.
	`openssl_binary`	Binary	❌	Details of `**/openssl` ELF.

NOTE: In case of busybox image, this binary will likely be present as a hard link. QScanner doesn't support scanning of hard/soft links. Use overlay for this.

Sample output

```
$ ./output/linux-amd64/qscanner --mode inventory-only -f json,table  --scan-types fileinsight --cache none image
registry.k8s.io/ingress-nginx/controller:v0.34.0 --platform linux/arm/v7
   ____    _____
```

```
  / __ \  / ___/  _____  ____ _   ____    ____   ___    _____
 / / / /  \__ \  / ___/ / __ `/  / __ \  / __ \ / _ \  / ___/
/ /_/ /  ___/ / / /__  / /_/ /  / / / / / / / //  __/ / /
\___\_\ /____/  \___/  \__,_/  /_/ /_/ /_/ /_/ \___/ /_/

By Qualys | version: dev-0

2025-07-16T09:52:42.389+0530    INFO    New instance of qscanner-dev-0 started with invocation ID b1ce391c-b8d6-
459c-b43a-7ce2e4b9ffc3
2025-07-16T09:52:42.391+0530    INFO    Fetching image details
2025-07-16T09:52:43.613+0530    INFO    Image source: remote
2025-07-16T09:52:43.614+0530    INFO    Starting Metadata scan
2025-07-16T09:52:43.615+0530    INFO    Metadata scan completed in 861.743µs
2025-07-16T09:52:43.615+0530    INFO    Starting [fileinsight] scan
2025-07-16T09:52:49.023+0530    INFO    [fileinsight] scan completed in 5.407835361s
2025-07-16T09:52:49.023+0530    INFO    Files detected: 5
2025-07-16T09:52:49.023+0530    INFO    All scans completed in 5.409093717s

Binaries:
+----+-------------------------------------------------------------------------+
| SL |                                  PURL                                   |
+----+-------------------------------------------------------------------------+
| 1  | pkg:generic/bash@5.0.11#bin/bash                                        |
| 2  | pkg:generic/busybox@1.31.1#bin/busybox                                  |
| 3  | pkg:generic/nginx-ingress-controller@v0.34.0#nginx-ingress-controller   |
| 4  | pkg:generic/curl@7.67.0#usr/bin/curl                                    |
| 5  | pkg:generic/openssl@1.1.1g#usr/bin/openssl                              |
+----+-------------------------------------------------------------------------+
2025-07-16T09:52:49.026+0530    INFO    Scan Result JSON created at
/root/qualys/qscanner/data/37d6625e4710556c69ee1177523314d70937adf5c3b253a80051b0f4e67701be-ScanResult.json
2025-07-16T09:52:49.026+0530    INFO    Skip fetching of report
2025-07-16T09:52:49.026+0530    INFO    Skipping policy evaluation
```

FileInsight Config

FileInsight scan is governed by the config file specified using `--fileinsight-config-file <path/to/config/file>`. It is composed of Rules that define what files to scan and which all attributes to extract from it. If this config file is not provided, QScanner will use **built-in** config for fileinsight scan.

Structure of rule

- `id`: UUID for this rule. Each rule must have unique non-empty value for ID.
- `name`: Display name for the rule for identifiying what this rule actually does. This must be non-empty.
- `disabled`: If true, this rule will not be applied. Default is false.
- `filter`: Filters are used to decide which files to scan. Following filters are supported:
  - `path_filter`: Specify a glob pattern here to filter file based on full-path.
  - `type_filter`: Specify type of file to scan. Options are:
    1. `any`: All files - text, binary, image, obj etc. **[Default]**
    2. `executable`: To scan only ELF (application/x-sharedlib and application/x-executable) and PE (application/vnd.microsoft.portable-executable) files.
    3. `non-executable`: To scan only non-executables.
- `matcher`: Matchers are used to collect data from content of the file. It uses regexes and stores data as named capature groups. Following matchers are supported:
  1. `path_matchers`: Add capture groups on filepath. Multiple path matchers can be provided for a single rule. If there is a match, others are not evaluated.
  2. `content_matchers`: Add capture groups on file content. Multiple content matchers can be provided for a single rule. If there is a match, others are not evaluated.
- `attributes`: This defines the attributes that needs to be collected for this file. If nothing is specified or `all` is provided, all supported attributes are collected. For list of supported attributes, refer `pkg/analyzer/fileinsight/attribute.go`.

Matchers

PathMatchers and ContentMatchers are used for collecting named capture groups based on file path and file content respectively. In case of ELF, QScanner used the **.rodata** section of the ELF for matching against the provided rules. For ELF, `name`, `version` and `release` are special named capture groups that are used by the scanner to identify the binary version (e.g. busybox@1.37.0). Notable points:

- Allowed characters for named capture groups- alpha-numerics and underscore `[A-Za-z0-9_]`
- Multiple path and content matchers can be provided. Whichever matches first, is used.
- If there is an overlap in the capture group name between path matcher and content matcher, latter gets preference.
- If multiple rules use same capture group name for the same file, then the rule evaluated later will replace the existing one. Ideally, we should ensure that such rules don't exist in the config.

To know more about advanced regex, refer this.

Special Capture Groups

When scanning ELF binaries, following are special capture groups used to identify binary name and version:

- `name`: Used for `ExecutableInfo.Name`. If `name` is not found in capture group, filename is used as a fallback.
- `version`: Used for `ExecutableInfo.Version`.
- `release`: Optional. If found for a rule, it is used to populate version with format `{$VERSION}-{$RELEASE}`.

For ELF, the content matcher will match only if it has a non-empty `name` and `version`.

Examples

Example 1: Use of content matcher to identify version of busybox binary

```
- id: "d3a577aa-2b7f-40d3-8168-1c86d8c47d17"
  name: "busybox-binary"
  filter:
    path_filter: '**/busybox'
    type_filter: executable
  matcher:
    path_matchers: []
    content_matchers:
        - '(?m)BusyBox\s+v(?P<version>[0-9]+\.[0-9]+\.[0-9]+)'
  attributes:
    - sha256digest
  disabled: false
```

ScanResult.json output:

```
{
    "FilePath": "bin/busybox",
    "MimeType": "application/x-sharedlib",
    "MD5Digest": "md5:6a389635384d88e391307a24fe1b63cd",
    "SHA1Digest": "sha1:e80a903c5b91a0481185caa4e07348f7f0aebdac",
    "SHA256Digest": "sha256:58d22501495a1db93e859f6e406460ed5e179d7afc1197d19e16f58bf8562c34",
    "Permission": "-rwxr-xr-x",
    "Size": 808712,
    "Layer": {
        "Digest": "sha256:fe07684b16b82247c3539ed86a65ff37a76138ec25d380bd80c869a1a4c73236",
        "DiffID": "sha256:fd2758d7a50e2b78d275ee7d1c218489f2439084449d895fa17eede6c61ab2c4"
    },
    "ExecutableInfo": {
        "Name": "busybox",
        "Version": "1.37.0",
        "PURL": "pkg:generic/busybox@1.37.0#bin/busybox",
        "ImportedLibs": [
            "libc.musl-x86_64.so.1"
        ]
    }
}
```

Example 2: Use of path matcher to identify binary name and content matcher for version of firefox binary

```
- id: "39e15755-10b4-4498-90a3-e2a22097a139"
  name: "firefox-binary"
```

```
  filter:
    path_filter: '**/firefox-bin*'
    type_filter: executable
  matcher:
    path_matchers:
        - '(?P<name>firefox).*'
    content_matchers:
        - 'version=(?P<version>[0-9a-z].*)&buildid=.*'
  attributes:
    - sha256digest
  disabled: false
```

Example 3: Use of multiple content matchers to identify java version. Whichever matches first, gets used

```
- id: "6a58690d-a0f4-4e82-8eab-2ebe6e375b6f"
  name: "java-binary"
  filter:
    path_filter: '**/java'
    type_filter: executable
  matcher:
    path_matchers: []
    content_matchers:
        - '(?m)\x00openjdk\x00java\x00(?P<release>[0-9]+[.0-9]*)\x00(?P<version>[0-9]+[^\x00]+)\x00'
        - '(?m)\x00(?P<release>[0-9]+[.0-9]*)\x00+(?P<version>[0-9]+[^\x00]+)\x00+openjdk\x00java'
        - '(?m)\x00(?P<version>[0-9]+\.[0-9]+\.[0-9]+(\+[0-9]+)?([-._a-zA-Z0-9]+)?)\x00'
        - '(?m)\x00(?P<version>[0-9]+[.0-9]+[.0-9]+\+[0-9]+-jvmci-[0-9]+[.0-9]+-b[0-9]+)\x00'
  attributes:
    - sha256digest
  disabled: false
```

Example 4: Use of wildcard path filter to collect all binaries

```
- id: "3629d77c-482e-421d-abb4-9351ef0fd637"
  name: "all executables"
  filter:
    path_filter: '**/*'
    type_filter: executable
  matcher:
    path_matchers: []
    content_matchers: []
  attributes:
    - sha256digest
  disabled: false
```

Example 5: Use of content matcher for collecting enabled repo info from JSON file

```
- id: "e77a442c-f2df-4ccc-ad23-52da79a81643"
  name: "enabled repo redhat"
  filter:
    path_filter: '**/buildinfo/content-sets.json'
    type_filter: non-executable
  matcher:
    path_matchers: []
    content_matchers:
        - '(?m)"(?P<enabled_repo_redhat>.*-rpms.*)"'
  attributes:
    - all
  disabled: false
```

ScanResult.json output:


```
{
    "FilePath": "usr/share/buildinfo/content-sets.json",
    "MimeType": "application/json",
    "MD5Digest": "md5:48e9c210d8dea46d3912086fb31cd15f",
    "SHA1Digest": "sha1:b63053847649771daac37f71f7287f170aae6ea8",
    "SHA256Digest": "sha256:ad8075312973670fb1f9ca519a4b2fab425565424a1da129aef3dd2889fdc25d",
    "Permission": "-rw-r--r--",
    "Size": 939,
    "Layer": {
        "Digest": "sha256:04422592e50ada8800bdc6f50f1859a16e94171406eb2b8935f109d5eca03060",
```

```
        "DiffID": "sha256:93cd51feeb40d97d594fb52e25579ac270de074e63a905ec38daa65fd9415a1d"
    },
    "CaptureGroups": {
        "enabled_repo_redhat": [
            "rhel-8-for-aarch64-appstream-rpms",
            "rhel-8-for-aarch64-appstream-source-rpms",
            "rhel-8-for-aarch64-baseos-rpms",
            "rhel-8-for-aarch64-baseos-source-rpms",
            "rhel-8-for-ppc64le-appstream-rpms",
            "rhel-8-for-ppc64le-appstream-source-rpms",
            "rhel-8-for-ppc64le-baseos-rpms",
            "rhel-8-for-ppc64le-baseos-source-rpms",
            "rhel-8-for-s390x-appstream-rpms",
            "rhel-8-for-s390x-appstream-source-rpms",
            "rhel-8-for-s390x-baseos-rpms",
            "rhel-8-for-s390x-baseos-source-rpms",
            "rhel-8-for-x86_64-appstream-rpms",
            "rhel-8-for-x86_64-appstream-source-rpms",
            "rhel-8-for-x86_64-baseos-rpms",
            "rhel-8-for-x86_64-baseos-source-rpms"
        ]
    }
}
```


Example 6: Use of content matcher for collecting entire file content for appstream info

```
- id: "8fadb831-87b6-4f3d-93f1-b854e0869c0d"
  name: "appstream info"
  filter:
    path_filter: 'etc/dnf/modules.d/*'
    type_filter: non-executable
  matcher:
    path_matchers: []
    content_matchers:
        - '(?m)(?P<appstream_info>.*)'
  attributes:
    - all
  disabled: false
```

ScanResult.json output:


```
{
    "FilePath": "etc/dnf/modules.d/perl-DBD-SQLite.mod",
    "MimeType": "text/plain; charset=utf-8",
    "MD5Digest": "md5:8e40d3af9258431af2e554c94774bc5f",
    "SHA1Digest": "sha1:f0960f3e631b571316d17100f1f81a77b3e3bc6c",
    "SHA256Digest": "sha256:b43503888037e68518a8f25e1a10d9de9ef8a7758740f8953391469ff71feefa",
    "Permission": "-rw-r--r--",
    "Size": 76,
    "Layer": {},
    "CaptureGroups": {
        "appstream_info": [
            "[perl-DBD-SQLite]",
            "name=perl-DBD-SQLite",
            "stream=1.58",
            "profiles=",
            "state=enabled",
            "",
            ""
        ]
    }
},
```


Compliance

Option: `--scan-types compliance`.

Compliance scanning will perform scan based on built-in controls. For full list of supported controls, refer `pkg/datacollector/compliance/controls.go`. The scan target will be evaluated against all supported controls to return a posture- `PASS`, `FAIL` or `NOT EVALUATED`. Compliance scan will also check for use of secrets in image layers. For this, QScanner will use the secret-config provided via `--secret-config-file` or will download it from backend. More details here.

Manifest

Option: `--scan-types manifest`.

Manifest scanning will perform scan based provided manifest file with `--manifest-file`. The scan is supported for Windows targets `winvmsnapshot` and `image`. The image scan needs to be performed with `--storage-driver docker-windowsfilter` only whenever manifest scan is used. This scan will gather data from Windows registry and Windows file system for vulnerability analysis.

WinRegistry

Option: `--scan-types winregistry`.

WinRegistry scanning allows the collection of Windows registry keys and values based on a specified configuration file. This file can be provided using the `--winregistry-config-file <file_path>` option. If the config file is not supplied, QScanner will default to using standard registry keys and will gather `OS` details.

Winregistry Config

Winregistry config file is basically a file that contains keys and values that define what should be to collect from windows registry hive file. It contains the following parameters:

- **key:** Windows registry full key path as like in manifest db.
- **recursion-depth:** Upto this depdth child will be traverse.
- **collect-values:** If also like to collect values then this field should be `true`.
- **value-name-regex:** Regex that is matched against the key value name. This is encoded in base64 format.
- **value-type-regex:** Regex that is matched against the key value type. This is encoded in base64 format.
- **max-bytes-to-collect:** Upto this length, collect the data, other data discarded.

Sample winregistry-config-file:

```
{
    "windows-registry-keys": [
        {
            "key": "HKLM\\SOFTWARE\\MICROSOFT\\WINDOWS NT\\CURRENTVERSION",
            "recursion-depth": 0,
            "collect-values": true,
            "value-name-regex": "UkVMRUFTRUlE",
            "value-type-regex": "XjEkfF4yJHxeMyR8XjQkfF43JHxeMTEk",
                  "max-bytes-to-collect": 4096
        },
        {
            "key": "HKLM\\SOFTWARE\\MICROSOFT\\WINDOWS NT\\CURRENTVERSION",
            "recursion-depth": 0,
            "collect-values": true,
            "value-name-regex": "UFJPRFVDVE5BTUU=",
            "value-type-regex": "XjEkfF4yJHxeMyR8XjQkfF43JHxeMTEk",
                  "max-bytes-to-collect": 4096
        }
    ]
}
```

Detection Behavior

When QScanner performs any scan along with OS scan, it keeps track of files that were installed by OS package manager. It uses this list of files to filter out the final scan result (SCA, secret, fileinsight etc.) so that such files are not reported. This is done to avoid false positives. If you still want to report such files, you can perform scan with `--detection-priority comprehensive`. This flag lets you Specify the detection priority:

- `precise` : Prioritizes precise by minimizing false positives (default)
- `comprehensive` : Aims to detect more security findings at the cost of potential false positives.

Supported Container Runtimes

QScanner will look for the specified image in a series of locations. By default, it will first look in the local Docker Engine, then Containerd, Podman, and finally remote registry.

Docker

QScanner tries to look for the specified image in your local Docker Engine. It will be skipped if Docker Engine is not running locally. QScanner expects the docker socket at `unix:///var/run/docker.sock`. You can override this path using `DOCKER_HOST` environment variable. For e.g:

```
export DOCKER_HOST=unix:///some/other/path/to/docker/socket/docker.sock
```

ContainerD

QScanner tries to look for the specified image in your local containerd. User will have to specify the full image name with registry and tag in order for Qscanner to search locally and scan it on containerd for example, `docker.io/library/golang:latest` or the repo digest of the image `golang@sha256:70031844b8c225351d0bb63e2c383f80db85d92ba894e3da7e13bcf80efa9a37`. It will be skipped if containerd is not running locally. If your containerd socket is not the default path (`//run/containerd/containerd.sock`), you can override it via `CONTAINERD_ADDRESS` environment variable.

```
$ export CONTAINERD_ADDRESS=/run/k3s/containerd/containerd.sock
$ ./qscanner image docker.io/library/golang:latest
```

If your scan targets are images in a namespace other than containerd's default namespace (`default`), you can override it via `CONTAINERD_NAMESPACE`.

```
$ export CONTAINERD_NAMESPACE=k8s.io
$ ./qscanner image docker.io/library/golang:latest
```

Podman

Scan your image in Podman (>=2.0) running locally. The remote Podman is not supported. Before performing QScanner commands, you must enable the `podman.sock` systemd service on your machine. For more details, see here.

```
$ systemctl --user enable --now podman.socket
```

QScanner will look for `podman.sock` in `$PODMAN_SOCKET_PATH`. By default, this is `/run/podman/podman.sock`. You can find out this path by running `podman info` command. To override this path, you can use `PODMAN_SOCKET_PATH` env variable. On Mac you can use podman machine inspect to get podman socket path. E.g:

```
# To use /my/custom/socket/path/podman/podman-machine.sock
$ export PODMAN_SOCKET_PATH=/my/custom/socket/path/podman/podman-machine.sock
```

CRI-O

Currently CRI-O runtime is only supported when using `--storage-driver crio-overlay`. Refer this for more details.

Storage Drivers

QScanner tries to save the image and create a tar out of it. This operation can be very time consuming if size of the image is large. If you have a container runtime installed and the image is pulled locally, you can avoid image save by making use of the runtime's underlying filesystem. For this, you can use `--storage-driver <storage-driver-to-use>` option.

Following storage drivers are supported:

No storage driver

Option: `--storage-driver none`.
Don't use any storage driver. Perform actual image save for scanning. To be used when the container runtime uses a storage driver that is not supported by QScanner.

Docker overlay2

Option: `--storage-driver docker-overlay2`.
Use docker overlay2 filesystem.

```
# Set docker root path as shown below if docker root directory path is not default("/var/lib/docker)
$ export DOCKER_ROOT_DIR=$CUSTOM_DOCKER_ROOT_DIR
$ ./qscanner --pod US2 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --storage-driver docker-overlay2 image centos:latest
```

Docker overlayfs (Containerd Snapshotter)

Option: `--storage-driver docker-overlayfs`.
Use docker overlay (containerd snapshotter) file system for docker. If the storage driver shown in the output of docker info command is `docker overlayfs` then you have to provide `docker-overlayfs` in `--storage-driver` command line parameter of qscanner. In case of docker-overlayfs, the default root directory is `/var/lib/containerd`, you can override this default root directory using the `DOCKER_ROOT_DIR` environment variable. Please note : The docker overlay support for Qscanner is added by taking into consideration the docker engine version 27.3.1 as containerd snapshotter is an experimental feature. If `--storage-driver docker-overlayfs` option is used with Qscanner on earlier docker engine versions it may not behave as we are expecting it to behave. It may have changed behaviour in newer versions as it's an experimental feature.

```
# To use custom docker socket
export DOCKER_HOST=unix:///some/other/path/to/docker/socket/docker.sock

# To use custom root directory.
$ export DOCKER_ROOT_DIR=$CUSTOM_DOCKER_ROOT_DIR
$ ./qscanner --pod US2 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --storage-driver docker-overlayfs image centos:latest
```

Containerd overlayfs

Option: `--storage-driver containerd-overlayfs`.
Use overlay file system for containerd. In case of `containerd-overlayfs`, you can override the default (`/var/lib/containerd`) root directory using the `CONTAINERD_ROOT_DIR` environment variable.

```
$ export CONTAINERD_ROOT_DIR=$CUSTOM_CONTAINERD_ROOT_DIR

# # To use non-default containerd socket: /my/custom/runtime/dir/containerd/containerd.sock
$ export CONTAINERD_ADDRESS=/my/custom/runtime/dir/containerd/containerd.sock
$ ./qscanner --pod US2 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --storage-driver containerd-overlayfs image docker.io/library/golang:latest
```

Please note: User will have to specify the full image name with registry and tag in order for Qscanner to search locally and scan it on containerd for example, `docker.io/library/golang:latest` or the repo digest of the image `golang@sha256:70031844b8c225351d0bb63e2c383f80db85d92ba894e3da7e13bcf80efa9a37`.

Containerd GCFS

Option: `--storage-driver containerd-gcfs`.

Use GCFS snapshotter for ContainerD. When "Image streaming" is enabled on the GKE cluster, it uses a different snapshotter for storing the image file system.

CRI-O overlay

Option: `--storage-driver crio-overlay`.
Use overlay file system for CRI-O. In case of `crio-overlay`, you can override default installation config using `/etc/crio/crio.conf`. By default socket location is `/var/run/crio/crio.sock`. We refer this crio.socket path for creating client, you can override this default path by using `XDG_RUNTIME_DIR` environment variable. For custom Root Directory please set `CRIO_ROOT_DIR` in env else it will refer to default path `/var/lib/containers/storage`

```
$ export CRIO_ROOT_DIR=$CUSTOM_CRIO_ROOT_DIR

# To use non-default CRI-O socket: /my/custom/runtime/dir/crio/crio.sock
$ export XDG_RUNTIME_DIR=/my/custom/runtime/dir
$ ./qscanner --pod US2 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --storage-driver crio-overlay image centos:latest
```

Podman overlay

Option: `--storage-driver podman-overlay`.
Use overlay file system for podman. If podman is not installed at default location please set `PODMAN_SOCKET_PATH=<podman-path>` in env else it will refer to default `/run/podman/podman.sock` location. For custom Root Directory please set `PODMAN_ROOT_DIR` in env else it will refer to default path `/var/lib/containers/storage`

```
$ export PODMAN_ROOT_DIR=$CUSTOM_PODMAN_ROOT_DIR

# To use non-default podman socket: /my/custom/runtime/dir/podman/podman.sock
$ export PODMAN_SOCKET_PATH=/my/custom/runtime/dir/podman/podman.sock
$ ./qscanner --pod US2 --client-id $QUALYS_CLIENT_ID --client-secret $QUALYS_CLIENT_SECRET --storage-driver podman-overlay image centos:latest
```

Docker windowsfilter

Option: `--storage-driver docker-windowsfilter`.
Use docker windowsfilter filesystem

Remote Registries

QScanner can download images from a private registry (like JFrog, Harbor etc.) without the need for installing Docker or any other 3rd party tools. This makes it easy to run within a CI process. In this, QScanner will download the image layers from the registry (similar to Crane tool) and perform scan on them.

QScanner supports registries that comply with the following specifications:

- Docker Registry HTTP API V2
- OCI Distribution Specification

When scanning images from remote registries with Qscanner by giving imagename@digest, the ScanResult will not include RepoTag information.

By default, QScanner will attempt scanning of image with platform linux/amd64. If you want to specify different platform, you can use `--platform <os>/<arch>`. E.g: `--platform linux/arm64`.

For private registries, you can provide registry credentials using the following flags:

- `registry-username`: Username of registry for the target image. ENV variable for this - `QSCANNER_REGISTRY_USERNAME`.
- `registry-password`: Password of registry for the target image. ENV variable for this - `QSCANNER_REGISTRY_PASSWORD`. Instead of password, you can alternatively provide the registry token using `--registry-token` flag.

- `registry-token`: This token gets used as a Bearer authentication. Think of how github access token is used. ENV variable for this - `QSCANNER_REGISTRY_TOKEN`.

It is recommended to provide credentials using their corresponding environment variables. Example:

```
$ export QUALYS_CLIENT_ID=<client-id>
$ export QUALYS_CLIENT_SECRET=<client-secret>
$ export QSCANNER_REGISTRY_USERNAME=<registry-username>
$ export QSCANNER_REGISTRY_PASSWORD=<registry-password>
$ ./qscanner --pod US2 image qualysjfrog20dec2023.jfrog.io/tony/rails:latest
```

Cloud Registries

QScanner can pull image from the cloud registries and scan it. This can be done in hosts without a container runtime.

AWS ECR (Elastic Container Registry)

QScanner uses AWS SDK. You don't need to install aws CLI tool. You can use AWS CLI's ENV Vars. QScanner will load AWS credentials from `~/.aws/credentials` (if available). If this file is not present, environment variables `AWS_SECRET_ACCESS_KEY` and `AWS_ACCESS_KEY_ID` needs to be set. Permission to pull images from ECR should be provided.

```
# Set required AWS environment variables
export AWS_ACCESS_KEY_ID=<access-key-id>
export AWS_SECRET_ACCESS_KEY=<secret-key>

# Run QScanner
./qscanner --pod US2 image <image-uri>:<tag>
```

In use case where we make use of Event Bridge, QScanner should be running in a lambda that gets triggered via Event Bridge. This lambda should have IAM role assigned to it so that it has permissions to access the ECR (`ECRfullacess`). When this role is assigned to the lambda running QScanner, environment variables `AWS_SECRET_ACCESS_KEY` and `AWS_ACCESS_KEY_ID` would be automatically configured. QScanner will then be able to pull images from ECR.

Please refer this QWiki that demonstrates how QScanner can be used in AWS pipelines.

ACR (Azure Container Registry)

QScanner uses Azure SDK for Go. You don't need to install `az` command. Service principal must have the `AcrPull` permissions.

```
# Create service principal
export SP_DATA=$(az ad sp create-for-rbac --name QScannerTest --role AcrPull --scope
"/subscriptions/<subscription_id>/resourceGroups/<resource_group>/providers/Microsoft.ContainerRegistry/registries/<regist

# Set required Azure environment variables. Make sure QSCANNER_REGISTRY_USERNAME is not set.
export AZURE_CLIENT_ID=$(echo $SP_DATA | jq -r .appId)
export AZURE_CLIENT_SECRET=$(echo $SP_DATA | jq -r .password)
export AZURE_TENANT_ID=$(echo $SP_DATA | jq -r .tenant)

# Run QScanner
./qscanner --pod US2 image <project>.azurecr.io/<image>:<tag>
```

Scanning multi-arch images

QScanner supports scanning of images built on multiple architectures. Below are the formats in which image can be provided:

1. `<index-digest>` + `--platform`
2. `<name>:<tag>` + `--platform`
3. `<name>:<tag>@<index-digest>` + `--platform`
4. `<name>:<tag>@<manifest-digest>` (platform flag not required)

Special case

When a multi-arch image is pulled, let's say using `docker pull <name>@<manifest-digest>` (without using platform flag), output of `docker images` shows "none" in image's tag. In this case we should use option **#4** (as mentioned above) to scan it.

Usage of `--platform` flag

User have to specify the platform in `<os>/<architecture>/<variant>` format. User will have to provide all the values that are applicable for the target image in order as mentioned above. OS and architecture are mandatory parameters.

Depending on architecture of the image, a default variant is used automatically. User need not provide all the 3 components (os, arch & variant). For more details please refer: https://github.com/containerd/containerd/blob/v1.4.3/platforms/platforms.go#L63

E.g: All these are valid inputs to `--platform` flag.

- `linux/arm64/v8`
- `linux/arm64`
- `linux/amd64/v2`
- `linux/amd64`

Note: If `--platform` flag is not mentioned or the value is empty then Qscanner will use OS and Architecture value of the host and perform scanning based on that.

Applicability of `--platform` flag

QScanner supports scanning of images built on multiple architectures using `--platform` flag in following conditions and targets as of now:

	Target	Platform flag applicable?	Comment
	remote images	✅	
Image archive	oci-archive	-	`index.json` file of OCI-archive does not contain any information about multiple supported platforms.
	docker-archive	-	`manifest.json` file of Docker-archive does not contain information about multiple supported platforms.
Runtime	docker	-	Docker client APIs do not support the usage of the platform flag with respect to image information fetching.
	containerd	✅	
	podman	-	Podman API supports only building of multi-arch image. Ref
	crio	-	`crictl` APIs don't have support for specifying platform.
Runtime with Storage Driver	docker-overlayfs	✅	
	docker-overlay2	-	Same as mentioned in comments for Docker runtime
	containerd-overlayfs	✅	
	podman-overlay	-	Same as mentioned in comments for Podman runtime
	crio-overlay	-	Same as mentiomed in comments for CRIO runtime

Scanning VM Snapshots

QScanner can be used to scan mounted VM snapshots. To scan it, simply run qscanner with `vmsnapshot` or `winvmsnapshot` command and provide the path to the mount point.

Linux snapshots

```
$ ./qscanner --mode inventory-only --format json vmsnapshot /mnt/i-00688adb059b0911d/volume-a
```

Windows snapshots.

```
$ ./qscanner.exe --mode inventory-only --format json winvmsnapshot <Volume1>:\,<Volume2>:\
```

Meta data about target VM

Certain parameters that are required for ChangeList evaluation are not available when we do a filesystem or vmsnapshot scan. To provide such parameters `--shell-commands` and `--scan-target-info` flags are provided.

Linux

Entries provided via the `--shell-commands` and `--scan-target-info` flag are inserted into the *UnixCommand* and *AgentInfo* table of ChangeList DB respectively. If changelist DB output format (`db`) is not used, then `--shell-commands` and `--scan-target-info` flag is not required.

```
$ ./qscanner vmsnapshot /mnt/i-00688adb059b0911d/volume-a \
        --mode inventory-only \
        --format json,db \
        --exit-if-os-not-found=true \
        --log-level debug \
        --shell-commands "uname -a=x86_64"\
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "agent_ip=10.10.1.164" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal"
```

Windows

Entries provided via the `--shell-commands` flag are inserted into the *Metadata* table of Changelist DB and entries provided via the `--scan-target-info` flag are inserted into the *ClientInfo* and *ProviderMetadata* table of ChangeList DB. If entries provided via `--shell-commands` contains partition details, it won't be filled in the table instead it will be used in the scanning process. If changelist DB output format (`db`) is not used, then `--scan-target-info` flag is not required.

```
> qscanner.exe winvmsnapshot D:\,E:\,F:\ \
        --mode inventory-only \
        --scan-types manifest \
        --format json,db \
        --log-level debug \
        --shell-commands "signature_version=VULNSIGS-2.6.273-8" \
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "LocalIPv4=10.10.1.164" \
        --scan-target-info "LocalIPv6=fe80::e452:9f2d:b7ee:93bb" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal" \
        --shell-commands "partition:D=dabb3f9d-c253-56c3-92ff-6c62837fc8cb;" \
        --shell-commands "partition:E=dabb3f9d-c253-44c3-92ff-6c62837fc8cc;" \
        --shell-commands "partition:F=dabb3f9d-ca14-44c3-92ff-6c62813fc8cb;Windows;"
```

Scan Status

When qscanner is running in inventory-only mode, it will **not** fail with non-zero exit code if any of the scan fails. The inventory file (db, json etc.) will always be generated (depending on options provided to `--format` flag). To check the status of individual scans, consumers can refer the `ScansPerformed` section of `ScanResult.json` file.

By default on Linux, if OS is not detected or OS based packages are not found, it is not treated as an error. To treat unknown or unsupported OS as error, you can specify `--exit-if-os-not-found=true`. If you wish to fail scan if OS packages are not detected, use `--exit-if-os-pkg-not-found=true`. OS scan status will be updated in `ScansPerformed` section depending on values of these flags.

Handling multi-volume and multi-partition snapshots

Linux

If there are multiple partitions within the mounted snapshot, each of those partitions should be scanned separately. In such cases, use `--multi-partition=true` flag. E.g:

```
/mnt/i-00688adb059b0911d
|--- vol-a
   |----- partition-0
   +----- partition-1
```

Similarly if multiple volumes are mounted and each contains multiple partitions, use `--multi-volume=true` flag. E.g:

```
/mnt/i-00688adb059b0911d
|--- vol-a
|  |----- partition-0
|  +----- partition-1
+--- vol-b
   |----- partition-0
   +----- partition-1
```

For both the above examples, `/mnt/i-00688adb059b0911d` should be provided to qscanner for vmsnapshot scan.

Windows

For Windows, Multi-Volume and Multi-Partition is handled in a flat structure. The winvmsnapshot command accepting mutiple volumes suffice the requirement. However for target volumes mounted of the scanner instance, the volume name can be different then the original volume. For scan-type `manifest`, the reported paths are absolute paths. To retrieve the correct target path on the target instance, `--shell-commands` with partition details are to be used. The details on the partitions to be passed as input to qscanner can be retrieved by running a powershell scrip located on `scripts\pre_scan.ps1`. The output of this script will contain all the necessary details to identify the target volume such as partition GUID, Volume Label and any Mount path if present.

```
--shell-commands "partition:D=dabb3f9d-c253-56c3-92ff-6c62837fc8cb;" \
--shell-commands "partition:E=dabb3f9d-c253-44c3-92ff-6c62837fc8cc;" \
--shell-commands "partition:F=dabb3f9d-ca14-44c3-92ff-6c62813fc8cb;Windows;"
```

For example above, `D,E,F` are the volumes belonging to the same instance, whose original target path could be `C,D,E`.

Optimized OS scan

Starting from QScanner v4.1.0, OS scan in vmsnapshot command has been optimized. With this optimization, full filesystem traversal will not be performed for OS scans. This will result in drastic performance improvement in OS scan without any dependency on size of VM snapshot.

Restricted scanning

Non-OS scans are very time consuming, particularly if the size of the VM snapshot is large. This is because such scans iterate the entire file system. To have the flexibility of scanning only specific directories for non-OS scans (i.e, SCA, Secret and Malware), `--include-dirs` flag can be used. This flag accepts multiple paths separated by comma. These paths should be relative to mounted root path.

Please note, regardless of directories provided in include-dirs flag, OS scan will be performed by QScanner in all relevant paths. So, include-dirs is only applicable for SCA, Secret and Malware scans. If nothing is provided in include-dirs, full snapshot will be scanned.

If `multi-partition` or `multi-volume` flags are also specified for Linux, include dirs will consider path relative to partition and volume respectively.

Partial scanning

When `--allow-partial-scan=true` is provided, scans will not error out on timeout. Instead it will return the collected data for whatever paths have been scanned. This flag can be effectively used in conjunction with `--include-dirs` to scan based on a priority of paths.

```
$ ./qscanner vmsnapshot /mnt/i-00688adb059b0911d/volume-a \
            --mode inventory-only \
            --format json \
            --scan-types os,sca,secret,malware \
            --secret-config-file secret-rules.json \
            --include-dirs "sca=root/go/bin,root/go/src/project-X,root/workdir"  \
            --include-dirs "secret=root/go/src/project-X,root/workdir"  \
            --include-dirs "malware=root/go/bin"  \
            --multi-partition=true \
            --scan-timeout 1m \
            --allow-partial-scan=true
```

In the above example, if SCA scan times out after scanning the 1st path mentioned in include-dirs (i.e during 2nd scan of 2nd path), then SCA scan will not fail. Instead it will show warning that SCA scan of 2nd path could not be done and the data collected for 1st path will be reported. Additionally, the `ScansPerformed` section of the `ScanResult.json` will indicate details of which all paths were scanned successfully and which paths failed. Please note, this `PartialScanInfo` object will not be added if `allow-partial-scan=false`. E.g:


```
"ScansPerformed": [
        {
            "ScanType": "metadata",
            "ScanDuration": 38562,
            "Status": "SUCCESS"
        },
        {
            "ScanType": "os",
            "ScanDuration": 2454040822,
            "Status": "SUCCESS"
        },
        {
            "ScanType": "sca",
            "ScanDuration": 3936451052,
            "Status": "SUCCESS"
        },
        {
            "ScanType": "secret",
            "ScanDuration": 53669653210,
            "Status": "SUCCESS",
            "PartialScanInfo": {
                "/root/go/bin": true,
                "/root/go/src/project-X": false,
                "/root/workdir": false
            }
        },
        {
            "ScanType": "malware",
            "ScanDuration": 110692,
            "Status": "SUCCESS",
            "PartialScanInfo": {
                "/root/go/bin": false,
                "/root/go/src/project-X": false,
                "/root/workdir": false
            }
        }
    ],
```


> **Note:** Partial scan is not applicable for OS and Manifest scans.

Usage Examples

Below are use cases of how QScanner could be used by Snapshot Based Assessment to scan VM snapshots for OS and/or SCA.

OS-only scan

Provide `--scan-types os` and appropriate value of `--multi-partition` and `--multi-volume` depending on how the VM snapshot has been mounted.

```
$ ./qscanner vmsnapshot /mnt/i-00688adb059b0911d \
        --mode inventory-only \
        --format json,db \
        --log-level debug \
        --scan-types os \
        --multi-volume=true \
        --multi-partition=true \
        --exit-if-os-not-found=true \
        --shell-commands "uname -a=x86_64" \
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "agent_ip=10.10.1.164" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal"
```

SCA-only scan

Linux

Provide `--scan-types os,sca`. For SCA-only, OS scan is also required because downstream modules need OS and Platform info which is not possible without OS scan. To avoid overhead of parsing package manager and increasing size of Changelist DB for InstalledSoftware table, certain analyzers resposible for collecting OS packages need to be disabled.

> **Note:** For SCA, connectivity to ghcr.io is required. This is mainly required to resolve Java dependencies. To scan in offline mode, you can specify `--offline-scan=true`. Please note, the quality of software package enumeration for Java substantially degrades when the scan is run in Offline mode. This can affect the accuracy of the vulnerability posture of the image. Hence, it is recommended to run the scan in Online mode.

```
# Disable analyzers responsible for collecting OS pacakges.
$ export QSCANNER_DISABLED_ANALYZERS=rpm,rpmqa,dpkg,dpkg-license,apk,bottlerocket-pkg-manager,portage,pacman

# Run qscanner
$ ./qscanner vmsnapshot /mnt/i-00688adb059b0911d \
        --mode inventory-only \
        --format json,db \
        --log-level debug \
        --scan-types os,sca \
        --multi-volume=true \
        --multi-partition=true \
        --exit-if-os-not-found=true \
        --shell-commands "uname -a=x86_64" \
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "agent_ip=10.10.1.164" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal"
```

Windows

Provide `--scan-types sca,winregistry`. For SCA-only, winregistry scan is also required because downstream modules need OS details which is not possible without winregistry scan.

> **Note:** For SCA, `signature_version` and `partition` details are not required. The reported paths will be with respect to the provided volumes.

```
# Run qscanner
> qscanner.exe winvmsnapshot D:\,E:\ \
        --mode inventory-only \
        --format json,db \
        --log-level debug \
        --scan-types sca,winregistry \
```

```
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "LocalIPv4=10.10.1.164" \
        --scan-target-info "LocalIPv6=fe80::e452:9f2d:b7ee:93bb" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal" \
        --exclude-dirs "D:\Windows" \
        --include-dirs "winregistry=Windows\System32" \
```

Manifest-only scan

Provide `--scan-types manifest` and appropriate value of `--manifest-file` with required volumes to scan depending on how the VM snapshot has been mounted.

```
> qscanner.exe winvmsnapshot D:\,E:\ \
        --format json,db \
        --log-level debug \
        --scan-types manifest \
        --manifest-file VULNSIGS-VM-2.6.300-2.manifest.db \
        --shell-commands "signature_version=VULNSIGS-2.6.300-2" \
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "LocalIPv4=10.10.1.164" \
        --scan-target-info "LocalIPv6=fe80::e452:9f2d:b7ee:93bb" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal" \
        --shell-commands "partition:D=dabb3f9d-c253-56c3-92ff-6c62837fc8cb;" \
        --shell-commands "partition:E=dabb3f9d-c253-44c3-92ff-6c62837fc8cc;" \
        --shell-commands "partition:F=dabb3f9d-ca14-44c3-92ff-6c62813fc8cb;Windows;"
```

Combined OS+SCA scan

In future, if OS+SCA is supported by downstream modules, then QScanner will not require any code changes. Just unset `QSCANNER_DISABLED_ANALYZERS` and run same command as above.

```
# Run qscanner
$ ./qscanner vmsnapshot /mnt/i-00688adb059b0911d \
        --mode inventory-only \
        --format json,db \
        --log-level debug \
        --scan-types os,sca \
        --multi-volume=true \
        --multi-partition=true \
        --exit-if-os-not-found=true \
        --shell-commands "uname -a=x86_64" \
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "agent_ip=10.10.1.164" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal"
```

Combined Manifest+SCA scan for Windows

In future, if Manifest+SCA is supported by downstream modules, then QScanner will not require any code changes. The combined scan will not require `winregistry` scan as OS details will be covered by manifest scan itself.

```
# Run qscanner
> qscanner.exe winvmsnapshot D:\,E:\ \
        --mode inventory-only \
        --format json,db \
        --log-level debug \
        --scan-types manifest,sca \
        --manifest-file VULNSIGS-VM-2.6.300-2.manifest.db \
        --shell-commands "signature_version=VULNSIGS-2.6.300-2" \
        --scan-target-info "provider_name=AWS" \
        --scan-target-info "instance_id=i-00688adb059b0911d" \
        --scan-target-info "LocalIPv4=10.10.1.164" \
        --scan-target-info "LocalIPv6=fe80::e452:9f2d:b7ee:93bb" \
        --scan-target-info "hostname=ip-10-10-1-164.ec2.internal" \
        --shell-commands "partition:D=dabb3f9d-c253-56c3-92ff-6c62837fc8cb;" \
        --shell-commands "partition:E=dabb3f9d-c253-44c3-92ff-6c62837fc8cc;"
```

Output Formats

There are different formats in which the inventory can be generated:

1. **Table** - This will be generated if `--format table` is used. This will list down PURLs of all OS and SCA packages in tabular format on console.
2. **Changelist DB** - This will be generated if `--format db` is used or any mode other than `inventory-only` is provided. QScanner will generate an LZMA compressed db file. This can be uncompressed using `unxz` Linux utility.
3. **JSON** - This will be generated if `--format json` is used. This is enabled by default.
4. **SPDX JSON** - Inventory generated in SPDX JSON format. If we are running QScanner in non-inventory-only mode, then this SPDX JSON will be uploaded to the backend. You can download this SBOM from UI.To generate this, use `--format spdx`. QScanner currently supports SPDX schema version v2.3.
5. **SPDX TLV** - Inventory generated in SPDX Tag-value format. To generate this, use `--format spdx-tlv`. QScanner currently supports SPDX schema version v2.3.
6. **Cyclonedx** - Inventory generated in CycloneDX format. To generate this, use `--format cyclonedx`. QScanner currently supports CycloneDX schema version v1.6.
7. **Secret Result** - Secret results generated in a separate LZMA compressed JSON file. To generate this, use `--format secret-result`. This option will not be shown in CLI help as it is for internal use only.

If multiple options are provided to `--format` flag, inventory will be generated in all those formats. All the output files will be generated in the output directory. The default output directory is `$USER_HOME_DIR/qualys/qscanner/data/`. This can be overridden by `--output-dir` flag.

Report Formats

Report is the artifact that gets generated after the evaluation of scan result. For instance, once scan is completed, changelist DB gets generated and is uploaded to backend for signature evaluation. QScanner then fetches the vulnerability report and/or policy evaluation result from backend. It will then generate report in the following formats:

Tabular Report

A concise tabular report will be shown on console. The table shows vulnerability detected in the scanned image. It will not be shown if QScanner is running in quiet mode (`--quiet` flag). It has following information:

- Vulnerable Software name
- Detected QID
- Qualys Detection Score (QDS)
- Vulnerability age
- CVE IDs corresponding to this QID
- Version of vulnerable software
- Version of software in which this vulnerability has been fixed (if fix is available, else it will show a "-")
- Summary of the vulnerability

Vulnerability Filtering

To filter vulnerabilities based on QDS you can use `--qds-threshold` flag.

- The filtering will only affect the output of tabular report. SARIF and JSON reports remain unchanged.
- This is just so that a developer or freemium user can filter out large no. of vulns from console tabular output to focus on high priority vulnerable packages.
- This is just formatting/filtering, it has no functional impact on policy evaluation.
- This flag is applicable even in case of `get-report` mode (where centralized policy evaluation does not come into picture).
- In case of `evaluate-policy` mode, policy evaluation is governed by central policy. No changes to that. But the CLI user can still filter the vulns on the tabular report.

SARIF Report

The tabular report gives only a glimpse of most relevant information. If you wish to get more details, the detailed report should be referred. This report file is in SARIF format.

SARIF stands for Security Assertion Results Format. It is an open standard file format for exchanging security analysis results. SARIF files are self-contained and can be used by a variety of tools and platforms. The SARIF format is supported by a growing number of security analysis tools like Trivy, CodeQL, Fortify SCA etc. The tabular report format is not usable in some of the CI/CD tools such as Github. In order to integrate to multiple CI/CD tools and security hubs, QScanner generates a detailed report in SARIF format.

This report includes information about vulnerabilities like:

- **QID-** Qualys vulnerability ID
- **CVE-IDs-** List of CVEs associated with this QID.
- **CVSSInfo-** CVSS scores- baseScore, temporalScore and accessVector.
- **CVSS3Info-** CVSS3 scores- baseScore and temporalScore.
- **Severity-**
- **CustomerSeverity-**
- **Risk-**
- **Softwares-** List of vulnerable softwares with name, path, installed version, fixed version and layer in which it was found.
- **Category-** "OEL", "SCA", "Debian", "Security Policy" etc.
- **FirstFound-** Timestamp of when this QID was first detected.
- **LastFound-** Timestamp of when this QID was last detected. This would change on each scan.
- **Published-** Timestamp of when this QID was published by Qualys.
- **VulnerabilityAge-** Duration since this QID was published by Qualys.
- **TypeDetected-** "CONFIRMED" etc.
- **ThreatIntel-** Knowledge base information of QIDs indicating activeAttacks, denialOfService, easyExploit, zeroDay, etc.
- **Title-** Short description of the vulnerability.

QScanner generates SARIF report in the output directory (specified by `--output-dir` flag) with name `<image_id>-Report.sarif.json`. Please refer this QWiki to get more understanding on the report generated by QScanner

This report is in complaince with Github Actions. With this QScanner can be integrated in github pipeline.

JSON Report

Detailed JSON report which includes inventory as well. This gets generated in the output directory (specified by `--output-dir` flag) with name `<image_id>-Report.json`.

Gitlab Report

GitLab is a web-based platform that helps software development teams manage their projects, code, and security. It's used for DevOps, DevSecOps, and GitOps. This format is as per Gitlab standards so that the qscanner report can be integrated with Gitlab pipeline. This gets generated depending upon scan types in the output directory (specified by `--output-dir` flag) with name `<image_id>-gitlab_vuln_report.json` when passed `--report-format gitlab` as by default. If scan type selected is secrets then it will also generate `<image_id>-gitlab_secret_report.json`.

Backend Communication

QScanner communicates with Qualys Backend via Gateway for various operations, like uploading SBOM, fetching vulnerability report etc. For this you need to provide the POD identifier using `--pod` flag. List of supported PODs can be found here. For full list of supported endpoints, refer `pkg/urlbuilder/urlbuilder.go`. When using Engineering Pods, you might also have to provide `--skip-verify-tls=true` flag.

Additionally, for authentication you need to provide `--client-id` and `--client-secret`. Preferred way is to use `QUALYS_CLIENT_ID` and `QUALYS_CLIENT_SECRET` ENV variables respectively. QScanner will generate an OAUTH (JWT) token using this that will have a validity of 4 hours. This gets used as a bearer token in each backend communication. The generated token gets cached at `~/.cache/qualys/qscanner/creds.json` so that QScanner can re-use it on subsequent runs (till its expiry). To disable the storing of creds in file, use `--cache-auth-token=false`.

NOTE: To maintain backwards compatibility, the `--access-token` flag and its corresponding `QUALYS_ACCESS_TOKEN` env variable will still be available. In this case, this will be directly used as bearer token for each backend communication.

QScanner also supports retries for all backend communications. The retries use exponential back-offs on each failed attempt. So we would have default retries after 5s, 10s, 20s, 40s, 80s, 120s ... (10 times). The exact behavior is specific to each endpoint but at a high level, below

parameters allow some of the customizations:

- `--max-network-retries`: Maximum number of retries to be performed in case of retryable server errors like response codes 429, 500, 502, 503 etc. For certain endpoints QScanner will retry on other response codes as well, e.g 404 for fetching vuln report and 424 for fetching policy evaluation result. If 0 is provided, no retries will be performed for any backend communication.
- `--network-retry-wait-min`: Minimum duration to wait before attempting for 1st retry. Please note, certain communications have an extra wait time before attempting for 1st retry. Please note, certain communications have an extra wait time before attempting for 1st retry, like fetch-vuln-report waits for 15 seconds before trying to fetch the report.
- `--network-retry-wait-max`: Maximum interval possible between 2 consecutive retries.

Pod Identifiers

Production pods (Reference: https://www.qualys.com/platform-identification/)

- `US1`: "https://gateway.qg1.apps.qualys.com"
- `US2`: "https://gateway.qg2.apps.qualys.com"
- `US3`: "https://gateway.qg3.apps.qualys.com"
- `US4`: "https://gateway.qg4.apps.qualys.com"
- `EU1`: "https://gateway.qg1.apps.qualys.eu"
- `EU2`: "https://gateway.qg2.apps.qualys.eu"
- `EU3`: "https://gateway.qg3.apps.qualys.it"
- `IN1`: "https://gateway.qg1.apps.qualys.in"
- `CA1`: "https://gateway.qg1.apps.qualys.ca"
- `AE1`: "https://gateway.qg1.apps.qualys.ae"
- `UK1`: "https://gateway.qg1.apps.qualys.co.uk"
- `AU1`: "https://gateway.qg1.apps.qualys.com.au"
- `KSA1`: "https://gateway.qg1.apps.qualysksa.com"

Engineering pods

- `ENG-POD01`: "https://gateway.p01.eng.sjc01.qualys.com"
- `ENG-POD03`: "https://gateway.p03.eng.sjc01.qualys.com"
- `ENG-POD04`: "https://gateway.p04.eng.sjc01.qualys.com"
- `ENG-POD05`: "https://gateway.p05.eng.sjc01.qualys.com"
- `ENG-POD09`: "https://gateway.p09.eng.sjc01.qualys.com"
- `ENG-POD11`: "https://gateway.p11.eng.sjc01.qualys.com"
- `ENG-POD12`: "https://gateway.p12.eng.sjc01.qualys.com"
- `ENG-POD13`: "https://gateway.p13.eng.sjc01.qualys.com"
- `ENG-POD17`: "https://gateway.p17.eng.sjc01.qualys.com"
- `ENG-POD18`: "https://gateway.p18.eng.sjc01.qualys.com"
- `ENG-POD19`: "https://gateway.p19.eng.sjc01.qualys.com"
- `ENG-POD24`: "https://gateway.p24.eng.sjc01.qualys.com"
- `ENG-POD29`: "https://gateway.p29.eng.sjc01.qualys.com"
- `ENG-POD35`: "https://gateway.p35.eng.sjc01.qualys.com"
- `ENG-POD37`: "https://gateway.p37.eng.sjc01.qualys.com"
- `ENG-POD39`: "https://gateway.p39.eng.sjc01.qualys.com"
- `OPS-POD01`: "https://gateway.p01.ops.sjc01.qualys.com"

If you are not sure about your POD identifier, you can also provide the full gateway URL via `--gateway-url` flag. E.g: `--gateway-url https://gateway.p24.eng.sjc01.qualys.com`.

Configuration

You can create custom configuration files using the `configure` command. This will create a configuration file that can be used to pass parameters and options for the QScanner easily. This will help you to store commonly used configurations in a file and re-use them without having to provide them to QScanner on each run. To create a custom configuration file, use:

```
qscanner configure [file path] [FLAGS]
```

Examples

Example 1:

```
$ ./qscanner configure --skip-verify-tls=true --mode inventory-only
```

This will create `/root/.config/qualys/qscanner/config.json` configuration file which can be used like:

```
$ ./qscanner image ubuntu
```

If `--config-file` parameter is not specified, CLI will load configuration from default config location `/root/.config/qualys/qscanner/config.json` (if it exists).

Example 2:

```
$ ./qscanner configure /root/myconfig.json --gateway-url https://gateway.p19.eng.sjc01.qualys.com/ --skip-verify-tls=true
```

This will create `myconfig.json` at `/root/` which will have custom values for `gateway-url` and `skip-verify-tls` flags and default values for all other flags. To use this configuration file:

```
$ ./qscanner image ubuntu --config-file /root/myconfig.json
```

Logging

QScanner supports following log levels:

- debug
- info (default)
- error
- warn
- fatal

This can be set by using `--log-level <level-string>`. E.g: To enable debug logging, use `--log-level debug`.

Additional options

By default, logs will be shown on console. To disable console logging, use `--quiet` flag. To write logs to file, use `--file-logging`. This will append logs to `qscanner.log` file present at the path specified by `--output-dir`.

Example use case:

To run qscanner in silent mode but at the same time generate log file, use both `--quiet` and `--file-logging` flags.

sca-data-collector logs

QScanner uses Qualys' fork of trivy (called *sca-data-collector*) for data collection. To enable logs from *sca-data-collector*, set `ENABLE_TDATACOLLECTOR_LOGS` environment variable before running qscanner. Log level used for qscanner will be applicable for *sca-data-collector* logs as well. If you want to get detailed logs, enable *sca-data-collector* logging and use `debug` log level for qscanner. E.g:

```
$ export ENABLE_TDATACOLLECTOR_LOGS=yes
$ ./qscanner --log-level debug --pod US2 image pypy
```

Caching

You can speed up data collection by using different types of cache. QScanner currently supports caching only for images. When cache is used, qscanner will check if data of a particular layer is already present. If found, it will simply fetch it from the cache and use it.

QScanner maintains cache data on a per-layer per-scan-type basis. It also takes care of the parameters that might change the scan data even if they are for the same layer and scan-type. For e.g, if SCA scan was performed for layer-1 before and a new scan has now been triggered with offline scan, the scan data is expected to change. QScanner knows this, so it triggers fresh SCA scan for layer-1.

Following types of cache are supported:

1. Local cache
2. Remote cache (-TBD-)
3. Input cache
4. No-Op cache

Use `--cache <cache-type>` flag. Following options are supported:

- `local` : Use local cache. QScanner creates a local cache database in `$USER_CACHE_DIR/qualys/qscanner/`. This path can be overridden by using `--cache-dir` flag.
- `remote` : Use remote cache. This will fetch cache information from Qualys backend. (not supported right now)
- `none` : Don't use any cache.

Default is `none`.

Input cache (used via sensor) is supported by a different flag:
`--input-cache-file <cache-file-path>` : QScanner loads cache from given cache-file path. As of today, this only contains list of cached layers for Malware scan.

Local Cache cleanup

QScanner maintains the local cache database at `$USER_CACHE_DIR/qualys/qscanner/` or the path specified by `--cache-dir` flag. This cache database will keep growing as and when new images are being scanned. To manage the growth of the cache database there are few options.

Clear entire cache

The entire QScanner cache directory can be cleared by using the `clear-local-cache` command. If a custom cache directory is being used, this needs to be executed with `cache-dir` flag. After this operation, subsequent QScanner run will not have any cache data. If it is executed with `--cache local`, this cache database will be created again.

```
$ ./qscanner clear-local-cache --cache-dir /my/custom/cache/path/
```

> **Note:** Please note, using `clear-local-cache` command will also clear the Java Index Database. Please refer this for more details on it.

Clear old unused cache entries

QScanner also maintains *last-accessed* field for each cache entry. It uses this field to determine old and unused cache entries that could be deleted. To let QScanner cleanup such cache entries you can use `--enable-cache-cleanup=true` flag. Below flags control the cleanup strategy:

- `--enable-cache-cleanup`: If true, QScanner will perform cleanup of old unused cache entries.
- `--cache-cleanup-duration-threshold duration`: If a cache entry is not used for this duration, QScanner will remove it from the cache database on next cleanup operation. This will be ignored if `--enable-cache-cleanup` is false. Default value is *45 days*. This means, QScanner will remove all cache entries that have not been accessed from last 45 days.
- `--cache-cleanup-frequency duration`: This is the duration after which QScanner will attempt to perform cleanup of old unused local cache entries. This ensures that cache cleanup process is not attempted on each run. This will be ignored if `--enable-cache-cleanup` is false. Default value is *7 days*. This means, QScanner will try to find out old unused cache entries once in every 7 days. Once such entries are identified based on value of `--cache-cleanup-duration-threshold` flag, they will be removed from the cache database.

Limit Resource Usage

QScanner by default is optimized to improve scan speed by utilizing the availabe resources (CPU and RAM). To perform scans using fewer resources, you can provide `--limit-resource-usage=true`. This will obviously take longer to complete the scans. The default value

of `--limit-resource-usage=false`.

At a high level, following parameters are controlled using this flag:

Parameter	`--limit-resource-usage=false`	`--limit-resource-usage=true`
Number of entities(layers/file paths) getting scanned concurrently.	`WORKER_THREADS_HIGH`	`WORKER_THREADS_LOW`
Number of analyzers running concurrently.	`WORKER_THREADS_HIGH`	`WORKER_THREADS_LOW`
Number of JAR files getting analyzed concurrently.	`WORKER_THREADS_HIGH`	`WORKER_THREADS_LOW`

Where,

- `WORKER_THREADS_HIGH = 5`
- `WORKER_THREADS_LOW = 1`

Ref: `pkg/util/constants/constants.go`

Example

With `--limit-resource-usage=false`

```
2025-10-17T12:45:17.759+0530    INFO    New instance of qscanner-dev-0 started with invocation ID 03537c43-39fc-
465a-9dbd-80ad06250bda
2025-10-17T12:45:17.760+0530    INFO    Starting Performance monitoring with interval 1s
2025-10-17T12:45:17.761+0530    INFO    Fetching image details
2025-10-17T12:45:17.763+0530    INFO    Perf: CPU Cores: 4, Total Memory: 7.5 GiB, PID: 2362323
2025-10-17T12:45:17.770+0530    INFO    Perf: St: S, CPU: 33.78%, Mem: 1.12%, OSThrd: 9, GoThrd: 9, CtxSw: 6, rd: 0
B, wr: 0 B, usr: 0.19s, sys: 0.07s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:18.612+0530    INFO    Image source: remote
2025-10-17T12:45:18.773+0530    INFO    Perf: St: S, CPU: 19.17%, Mem: 1.21%, OSThrd: 9, GoThrd: 13, CtxSw: 6, rd: 0
B, wr: 0 B, usr: 0.25s, sys: 0.10s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:18.982+0530    INFO    Starting Metadata scan
2025-10-17T12:45:18.983+0530    INFO    Metadata scan completed in 795.101µs
2025-10-17T12:45:18.983+0530    INFO    Starting [os sca] scan
2025-10-17T12:45:19.776+0530    INFO    Perf: St: S, CPU: 41.08%, Mem: 1.68%, OSThrd: 9, GoThrd: 28, CtxSw: 6, rd: 0
B, wr: 8.7 MiB, usr: 0.88s, sys: 0.27s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:20.778+0530    INFO    Perf: St: S, CPU: 94.25%, Mem: 2.05%, OSThrd: 10, GoThrd: 22, CtxSw: 6, rd:
0 B, wr: 8.7 MiB, usr: 2.93s, sys: 0.63s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:21.780+0530    INFO    Perf: St: S, CPU: 124.70%, Mem: 1.91%, OSThrd: 10, GoThrd: 22, CtxSw: 6, rd:
0 B, wr: 8.8 MiB, usr: 4.96s, sys: 1.00s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:22.782+0530    INFO    Perf: St: S, CPU: 143.03%, Mem: 1.71%, OSThrd: 10, GoThrd: 22, CtxSw: 6, rd:
0 B, wr: 8.8 MiB, usr: 6.95s, sys: 1.33s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:23.784+0530    INFO    Perf: St: S, CPU: 156.11%, Mem: 1.78%, OSThrd: 10, GoThrd: 22, CtxSw: 6, rd:
0 B, wr: 8.8 MiB, usr: 8.93s, sys: 1.66s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:24.786+0530    INFO    Perf: St: S, CPU: 154.52%, Mem: 1.51%, OSThrd: 10, GoThrd: 21, CtxSw: 6, rd:
0 B, wr: 9.1 MiB, usr: 10.21s, sys: 1.83s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:25.787+0530    INFO    Perf: St: S, CPU: 153.63%, Mem: 1.55%, OSThrd: 10, GoThrd: 21, CtxSw: 6, rd:
0 B, wr: 9.1 MiB, usr: 11.41s, sys: 2.09s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:26.790+0530    INFO    Perf: St: S, CPU: 153.13%, Mem: 1.48%, OSThrd: 10, GoThrd: 21, CtxSw: 6, rd:
0 B, wr: 9.1 MiB, usr: 12.70s, sys: 2.29s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:27.791+0530    INFO    Perf: St: S, CPU: 151.24%, Mem: 1.46%, OSThrd: 10, GoThrd: 21, CtxSw: 6, rd:
0 B, wr: 9.1 MiB, usr: 13.85s, sys: 2.47s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:28.575+0530    INFO    [os sca] scan completed in 9.591970986s
2025-10-17T12:45:28.579+0530    INFO    OS detected: Red Hat Enterprise Linux Server 7.9
2025-10-17T12:45:28.579+0530    INFO    OS package(s) detected: 335
2025-10-17T12:45:28.579+0530    INFO    Technologies detected: 0
2025-10-17T12:45:28.579+0530    INFO    Language package(s) detected: 0
2025-10-17T12:45:28.579+0530    INFO    All scans completed in 9.597417021s
2025-10-17T12:45:28.586+0530    INFO    Scan Result JSON created at
command1_output/efe11e13cb2d7a214f1b939e8be6b482ee695c0a87f14de5cfc35f364df056d5-ScanResult.json
2025-10-17T12:45:28.599+0530    INFO    SPDX document created at
command1_output/efe11e13cb2d7a214f1b939e8be6b482ee695c0a87f14de5cfc35f364df056d5-ScanResult.spdx.json
2025-10-17T12:45:28.599+0530    INFO    Skip SBOM upload
2025-10-17T12:45:28.599+0530    INFO    Creating changelist
2025-10-17T12:45:28.793+0530    INFO    Perf: St: S, CPU: 150.09%, Mem: 2.50%, OSThrd: 10, GoThrd: 16, CtxSw: 6, rd:
0 B, wr: 28 MiB, usr: 14.80s, sys: 2.90s, io: 0.00s, idle: 0.00s
```

```
2025-10-17T12:45:28.807+0530    INFO    Compressed Changelist DB created at
command1_output/efe11e13cb2d7a214f1b939e8be6b482ee695c0a87f14de5cfc35f364df056d5-ChangeList.db.xz
2025-10-17T12:45:28.807+0530    INFO    Skip changelist upload
2025-10-17T12:45:28.807+0530    INFO    Skip fetching of report
2025-10-17T12:45:28.807+0530    INFO    Skipping policy evaluation
2025-10-17T12:45:29.794+0530    INFO    Performance monitoring stopped
```

With `--limit-resource-usage=true`

```
2025-10-17T12:45:29.965+0530    INFO    New instance of qscanner-dev-0 started with invocation ID 41be8e35-f66f-
4604-a91c-f43c225de100
2025-10-17T12:45:29.965+0530    INFO    Starting Performance monitoring with interval 1s
2025-10-17T12:45:29.966+0530    INFO    Fetching image details
2025-10-17T12:45:29.966+0530    INFO    Perf: CPU Cores: 4, Total Memory: 7.5 GiB, PID: 2362362
2025-10-17T12:45:29.971+0530    INFO    Perf: St: S, CPU: 19.58%, Mem: 1.14%, OSThrd: 9, GoThrd: 11, CtxSw: 4, rd: 0
B, wr: 0 B, usr: 0.14s, sys: 0.05s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:30.974+0530    INFO    Perf: St: S, CPU: 12.16%, Mem: 1.23%, OSThrd: 10, GoThrd: 13, CtxSw: 4, rd:
0 B, wr: 0 B, usr: 0.18s, sys: 0.06s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:31.337+0530    INFO    Image source: remote
2025-10-17T12:45:31.582+0530    INFO    Starting Metadata scan
2025-10-17T12:45:31.582+0530    INFO    Metadata scan completed in 318.154µs
2025-10-17T12:45:31.582+0530    INFO    Starting [os sca] scan
2025-10-17T12:45:31.976+0530    INFO    Perf: St: S, CPU: 9.74%, Mem: 1.26%, OSThrd: 10, GoThrd: 18, CtxSw: 4, rd: 0
B, wr: 0 B, usr: 0.21s, sys: 0.08s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:32.979+0530    INFO    Perf: St: S, CPU: 38.21%, Mem: 1.39%, OSThrd: 11, GoThrd: 18, CtxSw: 4, rd:
8.7 MiB, wr: 18 MiB, usr: 1.26s, sys: 0.26s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:33.980+0530    INFO    Perf: St: S, CPU: 54.62%, Mem: 1.41%, OSThrd: 11, GoThrd: 18, CtxSw: 4, rd:
8.7 MiB, wr: 48 MiB, usr: 2.29s, sys: 0.43s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:34.983+0530    INFO    Perf: St: S, CPU: 65.02%, Mem: 1.41%, OSThrd: 11, GoThrd: 19, CtxSw: 4, rd:
8.7 MiB, wr: 78 MiB, usr: 3.30s, sys: 0.59s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:35.986+0530    INFO    Perf: St: S, CPU: 71.72%, Mem: 1.41%, OSThrd: 11, GoThrd: 18, CtxSw: 4, rd:
8.7 MiB, wr: 94 MiB, usr: 4.28s, sys: 0.73s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:36.987+0530    INFO    Perf: St: S, CPU: 72.37%, Mem: 1.40%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 94 MiB, usr: 4.98s, sys: 0.81s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:37.989+0530    INFO    Perf: St: S, CPU: 76.21%, Mem: 1.38%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 120 MiB, usr: 5.92s, sys: 0.94s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:38.991+0530    INFO    Perf: St: S, CPU: 78.97%, Mem: 1.41%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 167 MiB, usr: 6.81s, sys: 1.08s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:39.993+0530    INFO    Perf: St: S, CPU: 81.41%, Mem: 1.39%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 216 MiB, usr: 7.72s, sys: 1.23s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:40.995+0530    INFO    Perf: St: S, CPU: 83.95%, Mem: 1.38%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 259 MiB, usr: 8.64s, sys: 1.43s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:41.998+0530    INFO    Perf: St: S, CPU: 85.94%, Mem: 1.38%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 281 MiB, usr: 9.62s, sys: 1.55s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:43.000+0530    INFO    Perf: St: S, CPU: 87.36%, Mem: 1.36%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 321 MiB, usr: 10.53s, sys: 1.70s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:44.003+0530    INFO    Perf: St: S, CPU: 89.52%, Mem: 1.36%, OSThrd: 11, GoThrd: 17, CtxSw: 4, rd:
8.7 MiB, wr: 349 MiB, usr: 11.57s, sys: 1.87s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:45.005+0530    INFO    Perf: St: S, CPU: 92.35%, Mem: 1.52%, OSThrd: 11, GoThrd: 19, CtxSw: 4, rd:
26 MiB, wr: 367 MiB, usr: 12.72s, sys: 2.06s, io: 0.00s, idle: 0.00s
2025-10-17T12:45:45.175+0530    INFO    [os sca] scan completed in 13.592772315s
2025-10-17T12:45:45.179+0530    INFO    OS detected: Red Hat Enterprise Linux Server 7.9
2025-10-17T12:45:45.179+0530    INFO    OS package(s) detected: 335
2025-10-17T12:45:45.179+0530    INFO    Technologies detected: 0
2025-10-17T12:45:45.179+0530    INFO    Language package(s) detected: 0
2025-10-17T12:45:45.179+0530    INFO    All scans completed in 13.597543524s
2025-10-17T12:45:45.186+0530    INFO    Scan Result JSON created at
command2_output/efe11e13cb2d7a214f1b939e8be6b482ee695c0a87f14de5cfc35f364df056d5-ScanResult.json
2025-10-17T12:45:45.207+0530    INFO    SPDX document created at
command2_output/efe11e13cb2d7a214f1b939e8be6b482ee695c0a87f14de5cfc35f364df056d5-ScanResult.spdx.json
2025-10-17T12:45:45.207+0530    INFO    Skip SBOM upload
2025-10-17T12:45:45.207+0530    INFO    Creating changelist
2025-10-17T12:45:45.400+0530    INFO    Compressed Changelist DB created at
command2_output/efe11e13cb2d7a214f1b939e8be6b482ee695c0a87f14de5cfc35f364df056d5-ChangeList.db.xz
2025-10-17T12:45:45.400+0530    INFO    Skip changelist upload
2025-10-17T12:45:45.400+0530    INFO    Skip fetching of report
2025-10-17T12:45:45.400+0530    INFO    Skipping policy evaluation
2025-10-17T12:45:46.006+0530    INFO    Performance monitoring stopped
```

Performance Monitoring

CPU and Memory profiling

Profiling is useful for identifying expensive or frequently called sections of code. The Go runtime provides profiling data in the format expected by the pprof visualization tool. Currently following profiles are supported:

- **cpu:** CPU profile determines where a program spends its time while actively consuming CPU cycles (as opposed to while sleeping or waiting for I/O).
- **heap:** Heap profile reports memory allocation samples; used to monitor current and historical memory usage, and to check for memory leaks.

To run CPU and memory profiling you can provide `--run-profiler=true` flag. This is a hidden flag and is intended for internal use only. This will generate `<invocation-uuid>-cpu.prof` and `<invocation-uuid>-mem.prof` in the output directory. These can be analyzed using go tool and then running `top` command or `tree` command for more details. You can run `help` for more details of available options.

Usage Example

```
$ go tool pprof /root/qualys/qscanner/data/profiling/60c1f9ea-fc09-4271-872c-76dc45f55de7-cpu.prof
File: qscanner
Type: inuse_space
Time: Mar 8, 2024 at 11:56am (IST)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top
(pprof) top
Showing nodes accounting for 1830ms, 73.49% of 2490ms total
Dropped 130 nodes (cum <= 12.45ms)
Showing top 10 nodes out of 203
      flat  flat%   sum%        cum   cum%
    1220ms 49.00% 49.00%     1220ms 49.00%  runtime/internal/syscall.Syscall6
     250ms 10.04% 59.04%      250ms 10.04%  runtime.memmove
      90ms  3.61% 62.65%      250ms 10.04%  runtime.scanobject
      50ms  2.01% 64.66%       50ms  2.01%  runtime.madvise
      50ms  2.01% 66.67%       50ms  2.01%  runtime.memclrNoHeapPointers
      40ms  1.61% 68.27%       60ms  2.41%  regexp.(*Regexp).tryBacktrack
      40ms  1.61% 69.88%       40ms  1.61%  runtime.heapBits.nextFast (inline)
      30ms  1.20% 71.08%       40ms  1.61%  bytes.indexFunc
      30ms  1.20% 72.29%       30ms  1.20%  indexbytebody
      30ms  1.20% 73.49%       70ms  2.81%  runtime.findObject
(pprof) tree
Showing nodes accounting for 2.17s, 87.15% of 2.49s total
Dropped 130 nodes (cum <= 0.01s)
Showing top 80 nodes out of 203
----------------------------------------------------------+-------------
      flat  flat%   sum%        cum   cum%   calls calls% + context
----------------------------------------------------------+-------------
                                             0.38s 31.15% |
github.com/aquasecurity/trivy/pkg/fanal/image/daemon.DockerImage.imageOpener.func6
                                             0.30s 24.59% |   archive/tar.(*regFileReader).Read
                                             0.28s 22.95% |   bufio.(*Reader).Read
                                             0.12s  9.84% |   github.com/aquasecurity/trivy/pkg/fanal/analyzer.
(*CompositeFS).CopyFileToTemp
                                             0.06s  4.92% |   github.com/aquasecurity/trivy/pkg/fanal/walker.
(*cachedFile).Clean
                                             0.02s  1.64% |   github.com/aquasecurity/trivy/pkg/fanal/analyzer.
(*CompositeFS).Cleanup
                                             0.01s  0.82% |
github.com/aquasecurity/trivy/pkg/fanal/analyzer/pkg/dpkg.dpkgAnalyzer.PostAnalyze.WalkDir.func3
     1.22s 49.00% 49.00%      1.22s 49.00%                | runtime/internal/syscall.Syscall6
----------------------------------------------------------+-------------
                                             0.23s 92.00% |   github.com/aquasecurity/trivy/pkg/fanal/walker.
(*cachedFile).Open.func1
                                             0.01s  4.00% |   io.ReadAtLeast
     0.25s 10.04% 59.04%      0.25s 10.04%                | runtime.memmove
----------------------------------------------------------+-------------
                                             0.13s 52.00% |   runtime.mallocgc
                                             0.12s 48.00% |   runtime.gcDrain
```

```
     0.09s  3.61% 62.65%      0.25s 10.04%                | runtime.scanobject
                                           0.07s 28.00% |   runtime.findObject
                                           0.04s 16.00% |   runtime.heapBits.nextFast (inline)
                                           0.03s 12.00% |   runtime.greyobject
                                           0.02s  8.00% |   runtime.heapBitsForAddr
----------------------------------------------------------+-------------
                                           0.02s 40.00% |   runtime.mallocgc
     0.05s  2.01% 64.66%      0.05s  2.01%                | runtime.madvise
----------------------------------------------------------+-------------
...
<snip>
...
```

Performance statistics

To log performance related statistics, you can specify the `--show-perf-stat=true` flag. This will start logging CPU and memory statistics as info level logs every 1 second. This interval can be controlled using `--perf-stat-interval <interval>` flag.

Below stats will be shown one time:

1. CPU cores

Sum of all the available CPU sockets, cores and hardware threads.

2. Total Memory

Total available RAM.

3. PID

Process ID of running qscanner instance.

Following information will be shown periodically:

1. State

One of the following characters, indicating process state:

- **R:** Running
- **S:** Interruptible sleep. During process execution, it might come across a portion of its code where it needs to request external resources. Mainly, the request for these resources is IO-based such as to read a file from disk or make a network request. Since the process couldn't proceed without the resources, it would stall and do nothing. In events like these, they should give up their CPU cycles to other tasks that are ready to run, and hence they go into a sleeping state.
- **D:** Waiting in uninterruptible disk sleep.
- **Z:** Zombie. Between the time when the process terminates and the parent releases the child process, the child enters into what is referred to as a Zombie state. A process can remain in a Zombie state if the parent process should die before it has a chance to release the process slot of the child process.
- **T:** Stopped by job control signal.
- **t:** Stopped by debugger during trace.
- **W:** Paging (only before Linux 2.6.0)
- **X:** Dead (from Linux 2.6.0 onward)
- **x:** Dead (Linux 2.6.33 to 3.13 only)
- **K:** Wakekill (Linux 2.6.33 to 3.13 only)
- **W:** Waking (Linux 2.6.33 to 3.13 only)
- **P:** Parked (Linux 3.9 to 3.13 only)
- **I:** Idle (Linux 4.14 onward)

2. CPU

Total CPU % being used by qscanner.

3. Mem

Percent of total available memory being used by qscanner.

4. OSThrd

Total number of OS threads running at this instant. These are operating system-level entities responsible for executing code. Each thread has its own fixed-size stack, typically several megabytes in size. Threads are scheduled and managed by the operating system kernel.

5. GoThrd

Total number of go-routines running at this instant. Go routines integrate deeply with Go's runtime. They are nonpreemptive and rely on the Go runtime to observe their behavior and automatically suspen them when they block.

6. CtxSw

There are 2 types of Context switches:

1. **Voluntary-** Voluntary context switch occurs when your application is blocked in a system call and the kernel decide to give it's time slice to another process.
2. **Non-voluntary-** Non-voluntary context switch occurs when your application has used all the timeslice the scheduler has attributed to it (the kernel try to pretend that each application has the whole computer for themselves, and can use as much CPU they want, but has to switch from one to another so that the user has the illusion that they are all running in parallel).

The logs will only show the non-voluntary context switches.

7. rd

Size of data which this process caused to be fetched from the storage layer.

8. wr

Size of data which this process caused to be sent to the storage layer.

9. sys

Seconds spent in system mode.

10. usr

Seconds spent in user mode.

11. io

Seconds spent waiting for I/O operation.

12. idle

Seconds for which this process was idle.

13. other

Sum of CPU seconds spent in nice, niced guest, steal, irq and softirq.

- **nice:** Seconds spent in user mode with low priority.
- **guestnice:** Seconds spent spent running a niced guest (virtual CPU for guest operating systems under the control of the Linux kernel).
- **steal:** Seconds spent in other operating systems when running in a virtualized environment.
- **irq:** Seconds spent in serving interrupt requests.
- **softirq:** Seconds spent in serving soft interrupt requests.

Feature Toggles

What is feature toggle?

Feature toggles (also known as feature flags or feature flipping) is a technique in development that allows you to enable or disable a feature without deploying any codes. They help developers to add new features rapidly and safely. Some of the use cases where feature toggles might be helpful:

- Enable/disable features based on some conditions.
- Pain of maintaining long-lived feature branches.
- Feature is ready but QA bandwidth not available to test it before delivery. This can happen if there is scope change after feature development due to any unforeseen circumstances.
- Feature is ready but integration testing not done with backend. This has happened several times when backend dependent tasks are not completed in sync with qscanner side development.

Category

There are 2 broad categories of Feature Toggles:

Static Feature Toggle

- Non-configurable. These are created during build time and reside in code.
- For list of currently available static feature toggles, refer `pkg/featuretoggle/static_feature_toggle_list.go`.
- These could be used to allow us to commit untested features (perhaps because of QA bandwidth issue or backend integration issues etc.) without having the overhead of maintaining separate feature branches.
- These should be only configured (enabled/disabled) during build time.
- They should only be enabled before releasing it to customers. Which means, those features should have been fully tested.

Dynamic Feature Toggle

- Configurable. State of these toggles can be changed at run-time.
- These configuration reside in a separate `~/.config/qualys/qscanner/feature-toggle.yaml`. For list of currently available dynamic feature toggles, refer `pkg/featuretoggle/feature_toggle.yaml`.
- These could be used to control specific behavior of QScanner, perhaps from Qualys backend (not customers themselves). This could include rolling out new features in a controlled way, perhaps A/B testing.
- At this point, I've added code for processing this YAML from `~/.config/qualys/qscanner/feature_toggle.yaml`. But this file could come from our backend in future.
- Dynamic feature toggles could also be used by QA to enable a feature just so that it could be tested, but not yet release it (for whatever reason).
- For customers, this config file would be absent and subsequently all the features controlled by dynamic feature toggle will be disabled.

Dynamic Feature toggle usage

Following steps can be followed to configure features controlled by dynamic feature toggles:

1. List of currently available dynamic feature toggles are available in `pkg/featuretoggle/feature_toggle.yaml`.
2. Copy the YAML and place it in `~/.config/qualys/qscanner/feature-toggle.yaml`.
3. Enable/disable the feature that you want to test.
4. To check status of all the feature toggles, run `./qscanner list-feature-toggles`.
5. Verify the state of the feature toggle that you wish to test.
6. Launch QScanner.

Types

Based on intended usage and longevity, feature toggles can be of following types:

1. **Release:** Used to control whether a feature is enabled or not. These are short-lived and should be removed after the feature is released.
2. **Operational:** Used to control a specific workflow.
3. **Experimental:** These are typically used for testing purposes and are typically longer-lived flags that can be removed once we no longer need to collect testing data.

Listing

You can list all the available feature toggles by using the `list-feature-toggles` command. Please note, this is a hidden command and is intended to be used internally only.

```
$ ./qscanner list-feature-toggles
Feature Toggles:
+----+----------------------------------------+----------------+----------+---------+------------------------------
--+
| SL |                 NAME                   |      TYPE      | CATEGORY | ENABLED |          DESCRIPTION
|
+----+----------------------------------------+----------------+----------+---------+------------------------------
--+
| 1  | docker-overlay-for-non-inventory-modes | ft_release     | static   | No      | QScanner only supports
```

```
|
|    |                                        |                 |          |         | docker-overlayfs storage
driver|
|    |                                        |                 |          |         | for inventory-only mode.
|
|    |                                        |                 |          |         | Enable this toggle to allow
|
|    |                                        |                 |          |         | non-inventory operation as
|
|    |                                        |                 |          |         | well for this storage driver.
|
|    |                                        |                 |          |         | Please note: docker-overlayfs
|
|    |                                        |                 |          |         | is an experimental feature by
|
|    |                                        |                 |          |         | Docker. This should only be
|
|    |                                        |                 |          |         | enabled when this feature is
|
|    |                                        |                 |          |         | available publicly.
|
+----+----------------------------------------+-----------------+----------+---------+-----------------------------
--+
| 2  | secret-config-download                 | ft_release      | static   | No      | QScanner will attempt to
|
|    |                                        |                 |          |         | download secret-config file
|
|    |                                        |                 |          |         | from backend.
|
+----+----------------------------------------+-----------------+----------+---------+-----------------------------
--+
| 3  | secret-payload-file                    | ft_experimental | dynamic  | No      | QScanner will generate
|
|    |                                        |                 |          |         | 'secret_payload.json' in
|
|    |                                        |                 |          |         | current directory before
|
|    |                                        |                 |          |         | uploading it to backend. This
|
|    |                                        |                 |          |         | can be used for testing to
|
|    |                                        |                 |          |         | evaluate the secret result
|
|    |                                        |                 |          |         | payload generated by
QScanner. |
|    |                                        |                 |          |         |  Please note, this file
|
|    |                                        |                 |          |         | is different than the file
|
|    |                                        |                 |          |         | generated when using '--
format |
|    |                                        |                 |          |         | secret-result'.
|
+----+----------------------------------------+-----------------+----------+---------+-----------------------------
--+
| 4  | secret-result-upload                   | ft_release      | static   | No      | QScanner will upload secret
|
|    |                                        |                 |          |         | scan result to backend.
|
+----+----------------------------------------+-----------------+----------+---------+-----------------------------
--+
```

Further Reading

- Motivation
- Best Practices

Other Options

Hidden Flags

Below is a list of hidden flags. These flags will not be shown on the QScanner help. These are mostly intended for developer's use.

- `--fragment-size <fragment_size>`: Maximum size (KB) of payload for communication with gateway. Default: 390
- `--secret-config-file <file_path>`: Specify the path to config file for secret detection rules. When backend communication is implemented in QScanner, this flag will not be hidden.
- `--input-cache-file <file_path>`: Specify path of the file containing input cache information. This is currently used by Sensor.
- `--run-profiler`: To enable CPU and Memory profiling. Refer this for more details on how to run profiler.

Environment Variables

General

- `QUALYS_ACCESS_TOKEN` : This is used to pass access token. It is recommended to avoid `--access-token` flag and use this environment variable.
- `QUALYS_CLIENT_ID` : This is used to client-id. It is recommended to avoid `--client-id` flag and use this environment variable.
- `QUALYS_CLIENT_SECRET` : This is used to client-secret. It is recommended to avoid `--client-secret` flag and use this environment variable.
- `QSCANNER_REGISTRY_USERNAME` : This is used to pass registry username.
- `QSCANNER_REGISTRY_PASSWORD` : This is used to pass registry password. It is recommended to avoid `--registry-password` flag and use this environment variable.
- `QSCANNER_REGISTRY_TOKEN` : This is used to pass registry token. It is recommended to avoid `--registry-token` flag and use this environment variable.
- `CONTAINERD_ROOT_DIR` : Set this environment variable with containerd root directory. If containerd root directory path is default(/var/lib/containerd) then no need to set this environment variable.
- `CONTAINERD_NAMESPACE` : If namespace of target scan image is not default then set this environment variable with namespace of target scan image.
- `CONTAINERD_ADDRESS` : Set this environment variable with containerd.sock file path. If containerd.sock file path is default(`//run/containerd/containerd.sock`) then no need to set this environment variable.
- `DOCKER_ROOT_DIR` : Set this environment variable with docker root directory path. If docker root directory path is default(/var/lib/docker) then no need to set this environment variable.
- `QSCANNER_ENABLED_JAVADB_REPOS` : Use this to provide an ordered list of repositories from where to attempt download of java-db. For more details, refer this.
- `TMPDIR` : Use this ENV variable to override default temp location that QScanner uses to create various temporary artifacts. The default is `/tmp`.

Internal Use Only

- `QSCANNER_DISABLED_ANALYZERS` : This environment variable can be used to disable specific analyzers of QScanner. This is meant to be used internally only. If you wish to disable specific languages from SCA scanning, specify it using `--disable-sca-languages` flag. Refer this for more details. To view a list of all the registered analyzers, you can use below command:
- `QSCANNER_ENABLE_JAVA_CLASS_ANALYZER` : Set this to `yes` to enable java-class analyzer. By default, this analyzer is disabled. It is mainly used by Sensor to detect log4j vulnerabilities in OS-only scan.
- `ML_BIN_PATH` : This is used to override the default path to MLExtractor artifacts. By default, QScanner will look for MLExtractor artifacts in `MLExtractor/` directory.
- `ENABLE_TDATACOLLECTOR_LOGS`: Set this to `yes` to enable logging of sca-data-collector. By default, its logging is disabled.
- `SHOW_SECRET_CONTENT`: Set this to `yes` to show match line of detected secrets. By default, match lines will not be shown.
- `DISABLE_QSCANNER_BANNER`: Set this to `yes` to disable QScanner banner.
- `DISABLE_SBOM_UPLOAD`: Set this to `yes` to disable SBOM upload to the CS backend.
- `SHOW_RESPONSE_BODY_IN_BACKEND_ERROR`: Set this to `yes` to show response payload in case of errors in backend communication. Please note, enabling this might reveal sensitive information of Infra. This has only been provided for internal debug usage.
- `ENABLE_CASE_INSENSITIVE_PATHS`: Set this to `yes` to have case-insensitive path comparision. By default, the comparision will be case-sensitive.
- `QSCANNER_ADD_UNIX_COMMAND_ENTRIES_IN_JSON` : Set this to `yes` to populate `UnixCommandEntries` entries in ScanResult JSON. The `Command` and `StdOutput` objects will be base64 encoded. This ENV has been provided only for Sensor/SU so that it does not need to imitate the logic that is provided by `fauxgen` package of QScanner.

Deploying Harbor Service

As a part of Harbor Scanner Adapter feature for deployment purpose we have created docker image supporting both arm and amd platform. Which can also be executed as a container for scanning images apart from Harbor Registry.

Clustered Deployment

This is when Harbor is deployed as clustered setup. Please refer https://stash.intranet.qualys.com/projects/CSSEN/repos/adapter-svc-helm-chart/browse/qualys-harbor?at=release%2F1.0.0.

This has all the details around deployment in Kubernetes environment through Helm Chart.

Standalone Docker Deployment

This is when Harbor is deployed as standalone docker setup where each Harbor Service corresponds to docker container.

Sample command to spawn QScanner image as docker container within Harbor. E.g:

```
docker run --name qualys-scanner \
    --network=harbor_harbor
    -v /home/ubuntu/qualys-scanner/values.yaml:/usr/share/qscanner/values.yaml \
    -v /home/ubuntu/accesstoken:/usr/share/qscanner/accesstoken/token \
    -p 5000:5000 \
    art-hq.intranet.qualys.com:5001/qualys/internal/qscanner:4.2.0-202-dev-linux-amd64-1723099998 \
    harbor-registry-service --file-logging=true --log-level=debug
```

Here we need to mount `values.yaml` within container at this location `/usr/share/qscanner/values.yaml` and need to mount accesstoken file at `/usr/share/qscanner/accesstoken/token`.

This command has 2 volume mounts, exposed port followed by QScanner command and flags

Deployment pipeline

https://jenkins.intranet.qualys.com/job/CS-Sensor/job/adapter-svc-helm-chart/job/develop/ -> for Helm Chart

https://jenkins.intranet.qualys.com/job/CS-Sensor/job/QScanner/job/develop/ -> for QScanner image artifacts

Also these images are pushed to
`art-hq.intranet.qualys.com:5001/qualys/internal/qscanner:4.2.1-220-dev-linux-amd64-1726226480` and
`art-hq.intranet.qualys.com:5001/qualys/internal/qscanner:4.2.1-220-dev-linux-arm64-1726226480`.

This QScanner image is not Harbor feature dependent. This can be used as standalone image please refer this.

Exit codes

General:

Exit Code	Description
0	`EXIT_SUCCESS`
1	`EXIT_GENERIC_ERROR`
2	`EXIT_INVALID_PARAMETER`
3	`EXIT_LOGGER_INIT_FAILED`

Scan Artifact related:

Exit Code	Description
5	`EXIT_FILESYSTEM_ARTIFACT_FAILED`
6	`EXIT_IMAGE_ARTIFACT_FAILED`
7	`EXIT_IMAGE_ARCHIVE_ARTIFACT_FAILED`
8	`EXIT_IMAGE_STORAGE_DRIVER_ARTIFACT_FAILED`
9	`EXIT_CONTAINER_ARTIFACT_FAILED`
10	`EXIT_OTHER_ARTIFACT_FAILED`

Scan related:

Exit Code	Description
11	`EXIT_METADATA_SCAN_FAILED`
12	`EXIT_OS_SCAN_FAILED`
13	`EXIT_SCA_SCAN_FAILED`
14	`EXIT_SECRET_SCAN_FAILED`
15	`EXIT_OS_NOT_FOUND`
16	`EXIT_MALWARE_SCAN_FAILED`
17	`EXIT_OS_NOT_SUPPORTED`
18	`EXIT_FILE_INSIGHT_SCAN_FAILED`
19	`EXIT_COMPLIANCE_SCAN_FAILED`
20	`EXIT_MANIFEST_SCAN_FAILED`
21	`EXIT_WINREGISTRY_SCAN_FAILED`

Output handler related:

Exit Code	Description
30	`EXIT_JSON_RESULT_HANDLER_FAILED`
31	`EXIT_CHANGELIST_CREATION_FAILED`
32	`EXIT_CHANGELIST_COMPRESSION_FAILED`
33	`EXIT_CHANGELIST_UPLOAD_FAILED`
34	`EXIT_SPDX_HANDLER_FAILED`
35	`EXIT_CDX_HANDLER_FAILED`
36	`EXIT_SBOM_COMPRESSION_FAILED`
37	`EXIT_SBOM_UPLOAD_FAILED`
38	`EXIT_SECRET_RESULT_CREATION_FAILED`
39	`EXIT_SECRET_RESULT_UPLOAD_FAILED`

Reporting related:

Exit Code	Description
40	`EXIT_FAILED_TO_GET_VULN_REPORT`
41	`EXIT_FAILED_TO_GET_POLICY_EVALUATION_RESULT`
42	`EXIT_POLICY_EVALUATION_DENY`
43	`EXIT_POLICY_EVALUATION_AUDIT`

Frequently Asked Questions

Q1. I am getting error *Failed to create scan target: Gateway URL (--gateway-url or --pod) and Access token (--access-token) are required* on running below command:

```
$ ./qscanner image <image_name>
```

A. By default QScanner runs in *get-report* mode. In this mode it will try to communicate with Qualys backend to fetch vulnerability report. To allow this, you need to specify the POD that you wish to use and the access token for your account.

```
$ export QUALYS_ACCESS_TOKEN=<your-access-token>
$ ./qscanner --pod US2 image <image_name>
```

For more details refer this

Q2. I provided both access token and pod. I still get *tls: failed to verify certificate: x509* error:

```
ERROR Changelist Output Handler failed: Failed to upload fragment #1: Failed to make HTTP request: Post
"https://gateway.p24.eng.sjc01.qualys.com/cms/cli/v1.0/asset/image/7a7b2e234b158c1e01eab04f851fc4b1a33296dbaa68c57d11815e
43d9-4e6a-b7e0-161b7621efdf/fragment/1/finalize": tls: failed to verify certificate: x509: certificate signed by
unknown authority
```

A. This can happen when you are running in a proxy environment. Specify `--skip-verify-tls=true` flag to bypass secure TLS verfication.

Q3. QScanner execution is stuck. Nothing happens.
A. Make sure that you have internet connectivity. If you are running in a proxy environment, make sure the `https_proxy` environment variable is configured.

Q4. Can we specify file name for the scan result that gets generated by QScanner?
A. You cannot specify custom name for each file generated by QScanner. QScanner generates inventory (details about packages, layer mappings, metadata etc.) in a file called `<prefix>_ScanResult.json` in the path specified by `--output-dir`. ScanResult.json is not the only output file generated by QScanner. It can also generate Changelist DB, Report, other output formats of inventory etc. depending upon the options provided. This is the reason that `--output-dir` option is provided. This will let you keep all artifacts generated by one instance of QScanner in one place. E.g:

```
$ ./qscanner --pod US1 --output-dir /tmp/pythonData/ image python
```

Above command will create the following:

- `/tmp/pythonData/<image_id>_ChangeList.db.xz`
- `/tmp/pythonData/<image_id>_ScanResult.json`
- `/tmp/pythonData/<image_id>_Report.sarif.json`

Q5. It is very cumbersome to provide the complete gateway URL to QScanner on each run. Is there any easy way?
A. Starting with QScanner v4.0.0, we support a new flag `--pod`. This will let you specify the POD to be used easily. For more details check here

Q6. Vulnerability report is generated in a tabular format on console. How can I use it conveniently by some automation program?
A. Vulnerability report is shown in a table on console. For details you can refer the full JSON report generated at `<output_dir>/<prefix>_Report.sarif.json`.

Q7. I get below error when running QScanner, what is wrong?

```
ERROR: Logging module is not initialized
Failed to initialize configuration: neither $XDG_CONFIG_HOME nor $HOME are defined
```

A. Please ensure that home directory is configured on the host that is running QScanner. Either `$HOME` or `XDG_CONFIG_HOME` should be configured. E.g:

```
export HOME=/home/newuser
```