Table of Contents

Networking

Post Installation

- Broker Client parameters (any SCM flows only)

Changelog

- 2025-09-02
- Changed
- 2025-05-28
- Added
- Removed
- 2025-01-09
- Changed
- Removed
- Fixed
- 2024-10-28
- Added
- Fixed
- Changed
- 2024-08-19
- Added
- Changed
- Removed
- Fixed
- 2024-06-21
- Added
- Fixed
- Changed
- 2024-04-15
- Fixed
- Changed
- 2024-03-14
- Fixed
- Changed
- Added
- 2024-02-08
- Added
- Fixed
- Changed
- Removed
- 2024-01-26
- Added
- Fixed

- Changed
  - 2024-01-16
- Fixed
- Changed
  - 2024-01-11
- Changed
  - 2023-12-07
- Fixed
  - 2023-12-07
- Added
- Changed
  - 2023-11-27
- Fixed
  - 2023-11-23
- Changed
- Removed
- Fixed
  - 2023-11-10
- Changed
  - 2023-11-08
- Changed
- Fixed
  - 2023-10-17
- Changed
  - 2023-10-17
- Added
- Changed

- v2.6.1
  - 2023-09-20
  - Changed
- v2.6.0
  - 2023-09-18
  - Added
  - Changed
  - Fixed
  - Removed
- v2.5.0
  - 2023-09-04
  - Added

    - Changed
    - Fixed
- v2.4.2
    - 2023-08-17
    - Added
        - Changed
    - Deprecated
- v2.4.1
    - 2023-07-14
    - Added
- v2.4.0
    - 2023-07-13
    - Added
- v2.3.0
    - 2023-07-12
    - Added
- v2.2.3
    - 2023-06-15
    - Added
- v2.2.2
    - 2023-06-13
    - Fixed
- v2.2.1
    - 2023-06-09
    - Added
    - Fixed
- v2.2.0
    - 2023-05-12
    - Added
    - Removed
- v2.0.0
    - 2023-04-20
    - Added
    - Removed
    - Changed

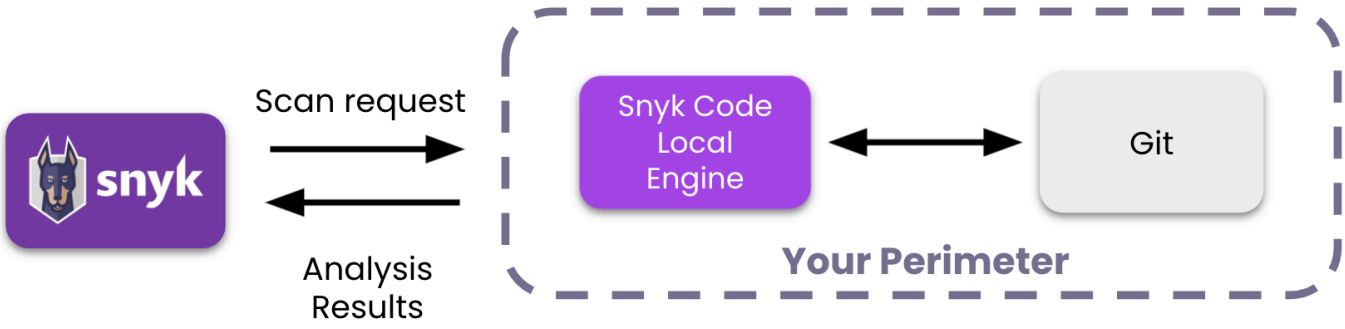
Snyk Code Local Engine

Welcome to the Snyk Code Local Engine documentation.

This documentation will guide you through the process of:

- Becoming familiar with Snyk Code Local Engine,
- Installing Snyk Code Local Engine on your infrastructure, and
- Testing your installation of Snyk Code Local Engine

Snyk Code Local Engine is a fully contained version of the Snyk Code Engine, meaning users do not need to upload code to the internet.



When using Snyk Code Local Engine, no source code leaves the perimeter of your organization. All analysis is performed internally, with only analysis results being sent to Snyk for review through the Snyk Web UI.

To install Snyk Code Local Engine, start with Installation Prerequisites. Some activities will require Snyk to make changes on Snyk systems to activate your installation of Snyk Code Local Engine.

For any questions or additional support, please reach out to your Snyk Representative.

Installation

Prerequisites

Before proceeding to install Snyk Code Local Engine, ensure you have:

- Image Pull Secrets (these are provided to you by Snyk), and
- Snyk Broker Settings (to connect to your private Source Code Management system)
- An installation of Helm `3.8.0` or higher (`NOTE: we only support install and upgrade via helm`).
- Access to your target Kubernetes Cluster to deploy the Snyk Code Local Engine Helm Chart

Deployment Requirements

Refer to Snyk Code Local Engine System Requirements for the most up-to-date requirements.

Shared Cluster Deployments

Snyk Code Local Engine can be installed on a shared Kubernetes Cluster. This does not change resource requirements, and comes with additional concerns. Review Advanced Deployment Models - Shared Cluster to understand the impact.

Network Requirements

Snyk Code Local Engine will communicate with Snyk to:

- Trigger Code Analyses
- Check Snyk Code Local Engine settings for Snyk CLI
- Manage inbound webhooks from SCM

Depending on selected functionality. The domains Snyk Code Local Engine requires outbound connectivity (supporting WebSockets) to are `*.snyk.io`. All outbound connections are secured with HTTPS/TLS encryption.

Outbound Proxy

Connections to Snyk domains from Snyk Code Local Engine may be routed through a proxy. Review Outbound Proxy for further information.

Basic Deployments

Snyk Code Local Engine leverages Helm for installation. Follow these steps to obtain the Snyk Code Local Engine Helm Chart and perform basic configuration.

Pulling the Snyk Code Local Engine Helm Chart

Snyk Code Local Engine Helm Charts are hosted on DockerHub, and can be obtained by:

```
$ helm registry login -u <username> registry-1.docker.io
Password:
Login suceeded
```

Where `username` and `password` are those provided by Snyk.

Once authenticated, pull the Snyk Code Local Engine Helm Chart:

```
helm pull oci://registry-1.docker.io/snyk/snyk-code-local-engine <--version x.y.z>
```

Optionally provide a version, otherwise Helm will fetch the latest version of the Chart.

Configuring the Snyk Code Local Engine Helm Chart

Configuration of Snyk Code Local Engine to suit your environment and requirements is carried out by modifying the `values-customer-settings.yaml` file provided with this documentation.

This YAML file will be used to keep settings between Snyk Code Local Engine upgrades, and may contain sensitive information.

Pulling Snyk Code Local Engine Docker Images

Snyk Code Local Engine uses a mixture of Snyk and third party Docker images. Add your `imagePullSecret` information to the `values-customer-settings.yaml` file:

```
global:
  imagePullSecret:
    credentials:
      username: <username>
      password: <password>
```

Helm will create a Kubernetes Secret with these credentials to allow your cluster to pull Snyk images from DockerHub.

Enabling Required Services

Snyk Code Local Engine consists of multiple services which provide functionality for different integration methods and features.

To enable and deploy the services required for your desired usage set the tag values to `true` in `values-customer-settings.yaml`:

```yaml
tags:
  scm: false # set to true to deploy services required for SCM integration
  scmPrCheck: false # set to true to deploy services required for SCM PR check functionality
  cli: false # set to true to deploy services required for CLI integration
  ide: false # set to true to deploy services required for supported IDE plugins
```

Enabling CLI, IDEs or PR Checks

Snyk Code Local Engine needs to be accessible from outside the cluster: - for CLI and IDE analysis, and/or - if leveraging Legacy or Universal Broker with Local Configuration to enable PR checks.

By default, no ingress is created.

Enabling the built-in Ingress

Set the following in `values-customer-settings.yaml`:

```yaml
global:
  ingress:
    enabled: true
  localEngineUrl: "http(s)://your-scle-domain-or-ip-here(:1234)" # needed for Legacy or Universal Broker with Local Configuration
```

Where the value for `global.localEngineUrl` resolves to the Kubernetes ingress used for Snyk Code Local Engine.

Depending on your Kubernetes Cluster you may wish to use a custom Ingress Controller and/or a custom Ingress template.

***JetBrains IDE limitation:** Set your SCLE organization ID in the JetBrains IDE settings and restart the IDE *before* carrying out any scans. This ensures scans are directed to SCLE.*

Setting the Snyk Region

By default, Snyk Code Local Engine will connect to the US Snyk Region.

If using an alternative region, replace any occurrence of `snyk.io` to the geographic alternative in this documentation.

To use an alternative region, configure the following in `values-customer-settings.yaml`:

EU Tenant

```
broker-client:
  brokerServerUrl: "https://broker.eu.snyk.io"
  brokerDispatcherUrl: "https://api.eu.snyk.io"
deeproxy:
  verificationEndpoint: "https://api.eu.snyk.io/v1/validate/token/snyk-to-deepcode-proxy-validation"
```

AU Tenant

```
broker-client:
  brokerServerUrl: "https://broker.au.snyk.io"
  brokerDispatcherUrl: "https://api.au.snyk.io"
deeproxy:
  verificationEndpoint: "https://api.au.snyk.io/v1/validate/token/snyk-to-deepcode-proxy-validation"
```

Note: See Data Residency At Snyk for other information (CLI, IDE, Web URL).

Installing Snyk Code Local Engine with Helm

The following should be used to install Snyk Code Local Engine for the first time on a Kubernetes cluster:

```
helm upgrade --wait <DEPLOYMENT_NAME> ./snyk-code-local-engine-<version>.tgz -i -f values-customer-settings.yaml -n <NAMESPACE>
```

Where:

- `<DEPLOYMENT_NAME>` will identify the Helm Chart controlled resources on the Kubernetes Cluster,
- `-f values-customer-settings.yaml` passes any values from this YAML file to the Helm Chart, and
- `-n <NAMESPACE>` is optional - use this if installing Snyk Code Local Engine to a Shared Cluster. If not specified, the `default` namespace is used.

If a re-install is needed (i.e a resource that cannot be patched is changed), first remove the installation and then install again.

Upgrading Snyk Code Local Engine with Helm

As Snyk Code Local Engine is stateless, upgrades may be performed without considering any data loss. The same command may be used to perform an upgrade:

```
helm upgrade --wait <DEPLOYMENT_NAME> ./snyk-code-local-engine-<version>.tgz -i -f values-customer-settings.yaml -n <NAMESPACE>
```

Versions within the same minor or patch increment are safe to install without removal. Upgrades from a previous major version should be a clean installation (remove and install).

Version Change	Upgrade/Clean Install	Process
`x.y.1` to `x.y.2`	`Upgrade`	Perform the `helm upgrade ...` command
`x.1.z` to `x.2.z`	`Upgrade`	Perform the `helm upgrade ...` command
`1.y.z` to `2.y.z`	`Clean Install`	First remove the installation and perform the `helm upgrade ...` command

Removing Snyk Code Local Engine

Snyk Code Local Engine may be removed with:

```
helm delete <DEPLOYMENT_NAME> -n <NAMESPACE>
```

This removes all components deployed by the Helm Chart. The `--purge` option may be used to delete all history if required.

Private Registry

Use the following steps in order to work with your private registry:

1. Enable the required analysis flow(s) in `customer-values-settings` (or via `helm --set ...)` under the `tags` key
2. Get the list of images and tags to pull by executing the following command: `helm template ./snyk-code-local-engine-<version>.tgz -f values-customer-settings.yaml | yq '.. | .image? | select(.)' | sort | uniq`
3. Push the images to your registry in the following format:
   - Any `snyk` images should reside within the `snyk` namespace
   - `minio`, `redis` images should reside within the `bitnamilegacy` namespace
   - `envoy` image should reside within the `envoyproxy` namespace

```
snyk/
 ├─── broker/
 ├─── code-pr-check-service/
 ├─── deepcode-suggest_v6/
 ├─── deeproxy/
 ├─── files-bundle-store/
 ├─── sast-analysis-api/
 ├─── scm-bundle-store/
 ├─── service-health-aggregator/
envoyproxy/
 ├─── envoy/
bitnamilegacy/
 ├─── minio/
 ├─── redis/
```

4. Set the `global.imageRegistry` value in `values-customer-settings.yaml` to your private registry. Include any additional path segments that your custom registry may require: for example, `global.imageRegistry: my.private.registry.io/parent/path` will pull snyk images from `my.private.registry.io/parent/path/snyk` and so on.

Note: If your private registry requires authentication, provide credentials as described in `values-customer-settings.yaml`.

What Next?

- Check the health of a running Snyk Code Local Engine installation
- Set up any networking for your installation
- Configure your installation with Snyk

Configuring Connections

Configuring Snyk Broker

Snyk Broker is deployed when using one or more SCM integrations and/or enabling PR Checks with Snyk Code Local Engine.

Snyk Broker is responsible for:

- Managing webhooks from one or more SCM(s) towards Snyk Code Local Engine
- Integrating with one or more private SCM(s) when importing a project via the Snyk UI

Each Brokered connection is identified by a unique Broker Token (a `uuidv4` string) that allows the Snyk Platform to direct requests to the Snyk Code Local Engine installation.

> Note: Each integration ***must*** be configured by a Snyk representative if newly created to ensure connectivity with Snyk Code Local Engine. If *cloning* an integration configured for Snyk Code Local Engine, further configuration by Snyk is not necessary. If integrations are created and used without configuration by a Snyk representative, you risk sending source code to the SaaS Snyk Platform.

Operating Modes

There are three operating modes for the Snyk Broker:

Mode	Available From	Description	Limitations	Recommended Use Case
Universal Mode with Remote Configuration	`v.2.12.0`	Stores Broker configuration on the Snyk Platform. Supports many integrations across many deployments. Broker tokens are no longer persisted within static configuration.	Requires setup via the Snyk Platform before use.	High scale requirements, more complex authorization scenarios.
Universal Mode with Local Configuration	`v2.8.0`	Stores Broker within a ConfigMap. Stores many integrations across one deployment.	Must handle Broker tokens and configuration data manually. Not recommended to scale above 20 integrations.	Moderate scale requirements, simple authorization scenarios.

Mode	Available From	Description	Limitations	Recommended Use Case
Legacy Mode	`v2.0.0`	Stores Broker configuration within a ConfigMap. Supports one integration.	No support for GitHub Server App. No support for multiple SCM integrations. Must handle Broker tokens and configuration data.	One SCM integration, small scale requirements, simple authorization scenarios.

Select the operating mode most applicable to your use case.

Universal Mode with Remote Configuration

Starting with Snyk Code Local Engine `v2.12.0`, the Broker supports running in Universal Mode with Remote Configuration. For more details on the Universal Broker and its configuration, see https://docs.snyk.io/enterprise-setup/snyk-broker/universal-broker.

Configuration data is stored on the Snyk Platform and fetched by the Broker Client at startup, rather than being specified directly in the values file.

Snyk Code Local Engine supports multiple Universal Broker Deployments for increased SCM integration scale.

Note: Mixed mode (local and remote configuration) is not supported. All connections must use either Local or Remote Configuration.

Prerequisites

- *Required*: Snyk Code Local Engine `v2.12.0` or higher.
- *Recommended*: Review public documentation for the Universal Broker to become familiar with new concepts (Deployments, Connections, Credential References).
- *Recommended*: The `snyk-broker-config` CLI tool. (Changes can also be made via API, but this documentation focuses on the `snyk-broker-config` tool).

Requirements

- Use the appropriate regional Snyk API endpoint when prompted for the Broker Client URL.
- All webhooks for PR Checks must transit the Snyk Platform. Existing Snyk Code Local Engine webhooks may need recreation via the Snyk UI.

Configuration

Universal Broker Snyk App

The Universal Broker authenticates with the Snyk Platform via an OAuth2 exchange. When creating the first Universal Broker integration, the CLI will support the installation of the Universal Broker Snyk App against a Snyk Organization.

Capture and securely store the returned Client Id and Client Secret. These values cannot be retrieved again - if lost, the Universal Broker Snyk App must be uninstalled and re-created, or the Client Secret must be rotated.

Universal Broker Deployments

Each Universal Broker Deployment is a logical group of Connections (where a Connection is a unique link between a specific SCM type and a Credential Reference).

Per Deployment: - Up to 25 Connections are supported - A new Broker StatefulSet is deployed

This permits scaling to *n* Deployments, supporting *n * 25* Connections, where each Deployment may consume up to `4vCPU` and `1Gi Memory` (if running each Deployment in High Availability Mode).

Creating New Connections

Follow steps outlined in public documentation to create one or more Universal Broker Connections.

When specifying a value for the Broker Client URL, the suggested default of `api.snyk.io` (or your regional equivalent) *must* be used. Targeting the Snyk Code Local Engine ingress is not supported.

Note: If leveraging the `github-server-app` integration, the default mount path for GitHub Server App certificates for the Universal Broker is `/etc/connections`. Review Configuring GitHub Server App for more information.

Migrating from Local Configuration

Follow the public documentation to migrate each connection to Remote Configuration.

Once migration is complete, follow steps in Helm to configure Universal Broker. The `universalBrokerConnections` block is *ignored* in Remote Configuration mode - any remaining configuration will have no effect.

Helm

Before deploying changes, migrate existing Connections or create new Connections for use with Snyk Code Local Engine.

Configure the Universal Broker with remote configuration using these values:

Value	Description
`broker-client.skipRemoteConfig`	Set to `true` to fetch configuration from the Snyk Platform.
`broker-client.brokerPlatformCredentials.clientId`	The `clientId` generated when installing the Universal Broker Snyk App. If `credentialSecretName` is specified, this value is ignored.

Value	Description
`broker-client.brokerPlatformCredentials.clientSecret`	The `clientSecret` generated when installing the Universal Broker Snyk App. If `credentialSecretName` is specified, this value is ignored.
`broker-client.brokerPlatformCredentials.credentialSecretName`	The name of a secret that exposes `clientId` and `clientSecret` - refer to the Platform Secret documentation for the expected format.
`broker-client.brokerDeployments[deployment-name].deploymentId`	The `deploymentId` generated when creating the Universal Broker Deployment
`broker-client.brokerDeployments[deployment-name].brokerPlatformRef`	Link a `brokerPlatformCredentials` object to this Deployment. This tells Universal Broker which OAuth2 credentials to use when fetching configuration for this Deployment.
`broker-client.brokerDeployments[deployment-name].credentialReferences`	A map of `<environment_variable_name>:<secret_value>` - each value should match that defined within a Credential Reference.
`broker-client.brokerDeployments[deployment-name].credentialReferencesSecretName`	The name of a secret that contains `DEPLOYMENT_ID` and one or more `<environment_variable_name><:secret_value>` pairs - refer to the Credential Reference Secret documentation for the expected format.

Example

```yaml
broker-client:
  skipRemoteConfig: false # necessary to fetch configuration from the Snyk Platform
  brokerType: universal
  brokerPlatformCredentials:
    defaultClient: # this client uses inline credentials
      clientId: "<client-id>"
      clientSecret: "<client-secret>"
    myClientUsingAnExternalSecret: # this client reads an existing secret for credentials
      credentialSecretName: <an-existing-secret>

  brokerDeployments:
    my-first-deployment: # this deployment defines secret data inline
      deploymentId: <your-deployment-id>
      brokerPlatformRef: defaultClient
      credentialReferences:
        MY_GITHUB_TOKEN: <gh_pat>
        MY_GITLAB_TOKEN: <gl_pat>

    my-other-deployment: # this deployment reads an existing secret for the deployment id
and credential references
      brokerPlatformRef: defaultClient
      credentialReferencesSecretName: <an-existing-secret>
```

See Secret Management for more information on creating external secrets for the Universal Broker.

Limitations

Since configuration is stored on the Snyk Platform, the Broker cannot dynamically determine `NO_PROXY` values. If using an outbound proxy, specify all required hosts in `global.proxy.noProxyHosts`:

```yaml
global:
  proxy:
    noProxyHosts: my.ghe.io,my.gitlab.io
```

Universal Mode with Local Configuration

Note: Snyk recommends migrating to leveraging the Universal Mode with Remote Configuration.

Starting with Snyk Code Local Engine `v2.8.0`, the Broker can be configured in `Universal` mode to support:

- Multiple SCM types
- Multiple instances of a single SCM
- Multiple Azure Repos Organizations
- The GitHub Server App integration

Specify each connection in the `universalBrokerConnections` map, including its specific configuration (auth, token, host).

Example defining a `github-enterprise` connection:

```
    broker-client:
        brokerType: universal
        universalBrokerConnections:
            myGithubEnterpriseConnection:
                type: github-enterprise
                auth: <github-enterprise-pat>
                brokerToken: <broker-token>
                githubHost: <your.private.ghe.instance.tld>
            <another-connection>
            <another-connection>
```

`myGithubEnterpriseConnection` can be any descriptive alphanumeric string.

Universal Broker Connection Types

Define each connection object under `broker-client.universalBrokerConnections` based on the SCM type.

GitHub Enterprise

```
myGithubEnterpriseConnection:
  type: github-enterprise
  brokerToken: <broker-token>
  githubHost: <your-ghe-host>
  auth: <your-ghe-pat>
```

GitHub Server App

At least one `githubServerApp` must be defined under `broker-client.githubServerApps`. See Configuring Github Server App for details.

```
myGitHubServerApp:
  type: github-server-app
  brokerToken: <broker-token>
  githubAppInstallationId: <your-gh-app-install-id>
  githubServerAppName: <your-gh-server-app-name>
```

GitHub

```
myGithubConnection:
  type: github
  brokerToken: <broker-token>
  auth: <your-gh-pat>
```

BitBucket Server

```
myBitbucket:
  type: bitbucket-server
  brokerToken: <broker-token>
  auth: <your-bb-password>
  bitbucketUsername: <your-bb-user>
  bitbucketHost: <your-bb-host>
```

BitBucket Server Bearer Auth

```
myBitbucketBearerAuth:
  type: bitbucket-server-bearer-auth
  brokerToken: <broker-token>
  auth: <your-bb-pat>
  bitbucketHost: <your-bb-host>
```

GitLab

```
myGitlab:
  type: gitlab
  brokerToken: <broker-token>
  auth: <your-gl-token>
  gitlabHost: <your-gl-host>
```

Azure Repos

```
myAzureRepos:
  type: azure-repos
  brokerToken: <broker-token>
  azureReposOrg: <your-azure-org>
  azureReposHost: <your-azure-host>
  auth: <your-azure-repos-token>
```

Multiple Instances of an SCM type

Recommended when multiple Snyk Organisations use SCLE with differently scoped SCM access tokens. Neither credential pooling or High Availability mode are implemented here.

Name connections based on Snyk Organisations or SCM identifiers:

```
broker-client:
  brokerType: universal
  universalBrokerConnections:
    myProductOrganisation:
      type: github-enterprise
      auth: <github-enterprise-pat>
      brokerToken: <broker-token>
      githubHost: <your.private.ghe.instance.tld>
    mySecurityOrganisation:
      type: github-enterprise
      auth: <another-github-enterprise-pat>
      brokerToken: <another-broker-token>
      gitHubHost: <your.private.ghe.instance.tld>
```

Multiple Azure Repos Organisations

Define multiple `azure-repos` connections, one for each Azure Repos Organisation. You can reuse the Azure Repos PAT if permissions allow, or create separate tokens per organisation. Name connections based on the Azure Repos Organisation:

```
    broker-client:
        brokerType: universal
        universalBrokerConnections:
            azureReposOrgName:
                type: azure-repos
                auth: <azure-repos-pat>
                azureReposOrg: <org-name>
                azureReposHost: <your.azure-repos.host.tld>
                brokerToken: <broker-token>
            anotherAzureReposOrgName:
                type: azure-repos
                auth: <azure-repos-pat>
                azureReposOrg: <another-org-name>
                azureReposHost: <your.azure-repos.host.tld>
                brokerToken: <another-broker-token>
            ...
```

Note: Helm supports YAML anchors for variable reuse:

```
    broker-client:
        brokerType: universal
        universalBrokerConnections:
            azureReposOrgName:
                type: azure-repos
                auth: &azureReposAuth <azure-repos-pat>
                azureReposHost: &azureReposHost <your.azure-repos.host.tld>
                ...
            anotherAzureReposOrgName:
                type: azure-repos
                auth: *azureReposAuth
                azureReposHost: *azureReposHost
```

High Availability Mode

Starting with Snyk Code Local Engine `v2.8.1`, High Availability (HA) Mode for the Snyk Broker is available.

By default, one Snyk Broker replica is deployed. To increase the replica count, set:

```
broker-client:
    replicaCount: <integer: 2, 3, 4>
    highAvailabilityMode:
        enabled: true
```

The maximum number of Snyk Broker replicas is `4`. Enable HA mode for Kubernetes clusters with unstable nodes or for high-traffic SCLE installations.

See HA Mode public documentation for details.

If using a non-US Snyk Region, set `brokerDispatcherUrl` to your regional equivalent (see Setting the Snyk Region).

Single SCM Mode

Note: This mode is deprecated, and should not be used. Snyk recommends migrating to Universal Mode with Remote Configuration.

Starting with Snyk Code Local Engine `v2.0.0`, specify `.broker-client.brokerType` and related authentication details for your SCM in `values-customer-settings.yaml`.

An example is shown below:

```
broker-client:
  brokerType: github-enterprise
  brokerToken: <broker-token>
  githubToken: <github-enterprise-pat>
  githubHost: <your.private.ghe.instance.tld>
```

The following SCMs are not supported in Single SCM Mode:

- `github-server-app`

Configuring GitHub Server App

Using the GitHub Server App integration (public docs) with Snyk Code Local Engine (SCLE) requires additional setup.

Overview of steps:

For SCLE v2.12.0 and above:

1. Create a Custom GitHub App
2. Create Brokered Integrations
3. Configure Snyk Code Local Engine
4. Provide GitHub Server App Key Material

For SCLE v2.11.0 and below:

1. Create a Custom GitHub App
2. Create Brokered Integrations
3. Configure Snyk Code Local Engine

Before starting, ensure you have:

- Permissions to create a GitHub App on your GitHub Enterprise instance.
- A valid Snyk API token for a User or Service Account with `View Organization`, `View Integrations`, and `Edit Integrations` permissions.
- If using Remote Configuration, access to the Snyk Platform as a Tenant Admin.

After completing the configuration, update or install Snyk Code Local Engine. The new GitHub Server App integrations will then be available.

Create a Custom GitHub App

Follow the Snyk documentation to create a GitHub App for Universal Broker. Do not use the `GitHub Server App` tile in the Snyk UI for this setup.

Capture the following information during creation:

- A GitHub App Client ID
- A GitHub App ID
- A GitHub App Private Key
- One or more GitHub App Installation IDs

Snyk Code Local Engine v2.12.0+

These steps apply when using Universal Broker in Remote Configuration mode (`broker-client.skipRemoteConfig: false`) with SCLE `v2.12.0` or higher.

Create Brokered Integrations

Follow the Snyk documentation to create the Universal Broker connection for your GitHub Server App, noting these SCLE specifics:

- When creating a `github-server-app` connection with the `snyk-broker-config` tool, you will be prompted for the private key mount path. The SCLE Helm Chart defaults to mounting keys under `/etc/connections` .

Configure Snyk Code Local Engine

Using the details (Credential References, Connections, Deployments) from the `snyk-broker-config` tool, update `values-customer-settings.yaml` :

- Add `brokerPlatformCredentials` entries for the `clientId` and `clientSecret` from the Universal Broker Snyk App installation.
- Add `brokerDeployments` entries to define Broker Client groups.
- Add `githubServerApps` entries for each unique GitHub App Private Key.

Example `values-customer-settings.yaml` for one GitHub Server App used by a Universal Broker Deployment:

```
broker-client:
  skipRemoteConfig: false
  type: universal
  githubServerAppsPrivateKeyMountPath: /etc/connections #optionally override the mount path
  githubServerApps:
    myGHSA: #define a GitHub App called myGHSA, creating a key at
/etc/connections/myghsa.pem
      githubAppPrivateKey: |-
        <your private key
        as a multiline
        string>
      githubAppPrivateKeyName: "" # Optionally override the key name
  brokerPlatformCredentials: # Renamed from brokerPlatformRef for clarity based on previous
doc
    defaultClient:
      clientId: <your-client-id>
      clientSecret: <your-client-secret>
  brokerDeployments:
    myDeployment:
      deploymentId: <your-deployment-id>
      brokerPlatformRef: defaultClient # References the key under brokerPlatformCredentials
```

Providing GitHub Server App Key Material

Option 1: Private Keys in `values-customer-settings.yaml`

The mounted key name defaults to the `githubServerApp` entry name:

```
broker-client:
  type: universal
  githubServerApps:
    myGHSA:
      githubAppPrivateKey: <your-private-key-as-a-multiline-string>
```

The above example provides the key material at `/etc/connections/myghsa.pem`.

Optionally, override the mount path and/or key name:

```
broker-client:
  type: universal
  githubServerAppsPrivateKeyMountPath: /a/custom/path # Must not be relative
  githubServerApps:
    myGHSA:
      githubAppPrivateKey: <your-private-key-as-a-multiline-string>
      githubAppPrivateKeyName: <my-custom-name> # Must not end with .pem
```

Option 2: Private Keys from External Secrets

Provide the name of an external Kubernetes secret for each `githubServerApp`. The secret must contain keys named `<certificate-name>.pem`:

```
kind: Secret
metadata:
  name: my-ghsa-secrets
type: Opaque
data:
  key.pem: <key-material>
  another-key.pem: <key-material>
```

```
broker-client:
  type: universal
  githubServerApps:
    myGHSA:
      githubAppPrivateKeySecretName: my-ghsa-secrets
```

This example mounts both keys from `my-ghsa-secrets` under `/etc/connections`. `githubAppPrivateKeyName` is ignored (use the key names in the secret). `githubServerAppsPrivateKeyMountPath` can still be set if needed.

Snyk Code Local Engine v2.11.0 and v2.10.0

These steps apply when using Local Configuration mode with SCLE `v2.11.0` or `v2.10.0`.

Create Brokered Integrations

Create a GitHub Server App integration once per Snyk Organisation using the Snyk API (replace `api.snyk.io` with your regional endpoint):

```
curl --location 'https://api.snyk.io/v1/org/:orgId/integrations' \
  --header "Content-Type: application/json; charset=utf-8" \
  --header "Authorization Token: token <your-snyk-api-token>" \
  --data '{
    "type": "github-server-app",
    "broker": {
      "enabled": true
      }
    }'
```

A successful response includes a `brokerToken`. Each `brokerToken` corresponds to one entry in the `universalBrokerConnections` map in your values file, linked to one GitHub App Installation ID.

A new tile will appear in your Snyk Org under "Integrations".

Configure Snyk Code Local Engine

In `values-customer-settings.yaml`:

- Define your GitHub App details (Client ID, App ID, Private Key) under `broker-client.githubServerApps`.
- Create `github-server-app` entries under `universalBrokerConnections`. Each entry needs a `brokerToken` and `githubAppInstallationId`, and must reference a named app defined in `githubServerApps` via `githubServerAppName`.

After configuration, update or install Snyk Code Local Engine.

Example `values-customer-settings.yaml` for one GitHub App connected to two GitHub installations, each linked to a different Snyk Organisation:

```
broker-client:
  type: universal
  githubServerApps:
    myGHSA: #define a GitHub App called myGHSA
      githubAppClientId: <your-client-id>
      githubAppId: <your-app-id>
      githubAppPrivateKey: |-
        <your private key
        as a multiline
        string>
      githubAppPrivateKeyName: "" # Optionally override the key name
  universalBrokerConnections:
    myGHSAInstallOrgA: # Descriptive name for connection
      type: github-server-app
      brokerToken: <broker-token-for-org-A> # Broker Token from Snyk Organisation A
      githubAppInstallationId: <installation-id-A>
      githubServerAppName: myGHSA # Matches the app defined above
    myGHSAInstallOrgB: # Descriptive name for connection
      type: github-server-app
      brokerToken: <broker-token-for-org-B> # Broker Token from Snyk Organisation B
      githubAppInstallationId: <installation-id-B>
      githubServerAppName: myGHSA # Matches the app defined above
```

Advanced Deployments

Shared Cluster

Using a shared Kubernetes cluster allows Snyk Code Local Engine to be installed alongside pre-existing workloads. The following considerations are important:

- Snyk Code Local Engine has significant resource requirements and may negatively impact other workloads already running on the cluster if capacity is low. Snyk Code Local Engine may be unable to fully schedule. See Snyk Code Local Engine Requirements to ensure your cluster has adequate resources available.
- By default, Kubernetes schedules pods for Snyk Code Local Engine on any available node within the cluster. See Using `tolerations`, `affinity` or `nodeSelector` for advanced resource targeting.
- By default, Helm installs to the `default` namespace. To change this, provide a namespace with `-n <namespace>` to Helm when installing. If the namespace does not exist, provide the `--create-namespace` flag as well (assuming your user is privleged to create namespaces).

Using `tolerations`, `affinity` or `nodeSelector`

This section will build on information given in Kubernetes documentation

To target some/all of the workloads to particular nodes, specify the some/all of the mechanisms as follows (example values are provided, adjust these to match your own node labels/other running workloads):

```
service_name:
    nodeSelector: #matches a node with label `NodeType` and value `large`
        NodeType: large
    tolerations: #matches if a node has taint `bigNode` with value `"true"`. The taint
blocks other workloads that do not have this toleration.
        - key: "bigNode"
          operator: "Equal"
          value: "true"
          effect: "NoSchedule"
    affinity: #schedules only on nodes with `kubernetes.io/os:linux`
        nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
            - key: kubernetes.io/os
                operator: In
                values:
                - linux
```

This approach is recommended for the heaviest workloads in Snyk Code Local Engine (`bundle`, `suggest`) to avoid the considerations mentioned in Shared Cluster.

Making these changes in `values-customer-settings.yaml` could look like:

```
...
suggest:
    ...
    nodeSelector:
        NodeType: large
    ...
```

Secret Management

Snyk Code Local Engine makes use of Kubernetes Secrets to authenticate access to internal components. By default, these secrets are managed by the Helm Chart.

The inbuilt secret lifecycle is:

1. On installation, create Opaque secrets using the `randAlphaNum` Helm function to generate values for required keys which are then base64 encoded and persisted to the deployment.
2. On upgrade, Helm will attempt to lookup the value(s) of a pre-existing secret:
    a. If the lookup succeeds (a value exists on the cluster), Helm will not re-generate the secret.
    b. If the lookup fails, Helm will re-generate the secret.

To use an external secret management solution with Snyk Code Local Engine, the following steps may be followed:

External Secrets

First create secrets for consumption by Snyk Code Local Engine. Then instruct Snyk Code Local Engine to consume these secrets, disabling the inbuilt secret generation mechanism.

Creating Secrets

Create secrets in the namespace Snyk Code Local Engine will be deployed to. Secrets are expressed with required key(s) and the format of the value(s) required. Any regex strings are provided to indicate the value (if any) that *would* be automatically generated by Snyk Code Local Engine if using the internal secret mechanism:

Session Secret

Function: Used for internal session authentication

Key	Format	Internal Secret Autogenerated Value
`SESSION_SECRET`	Alphanumeric string	`[a-zA-Z0-9]{64}`

Redis Secrets

Function: Used to secure access to any Redis deployments

Key	Format	Internal Secret Autogenerated Value
`REDIS_PASSWORD`	Alphanumeric string	`[a-zA-Z0-9]{32}`

S3 Secret

Function: Used to secure access to S3 deployments

Key	Format	Internal Secret Autogenerated Value
`root-user`	Alphanumeric string	`admin`
`root-password`	Alphanumeric string	`[a-zA-Z0-9]{16}`
`region`	Alphanumeric string	`us-east-1`
`name`	Alphanumeric string	`scm-bundle-store-s3`

JWT Secret

Function: Used for encrypting operations on objects stored in the object store

Key	Format	Internal Secret Autogenerated Value
`jwtSecretKey`	Alphanumeric string	`[a-zA-Z0-9]{16}`

Universal Broker Secrets

Supported as of Snyk Code Local Engine `2.12.0`.

PLATFORM SECRET

Function: Used to store Universal Broker Client ID and Client Secret for authentication towards the Snyk Platform

Key	Format	Internal Secret Autogenerated Value
`CLIENT_ID`	Alphanumeric String	``
`CLIENT_SECRET`	Alphanumeric String	``

CREDENTIAL REFERENCE SECRET

Function: Used to specify secret values for any Credential References required by this Universal Broker Deployment

Key	Format	Internal Secret Autogenerated Value
`DEPLOYMENT_ID`	Alphanumeric String	``

Key	Format	Internal Secret Autogenerated Value
`ENV_VAR_NAME`	Alphanumeric string	``

Consuming Secrets

Update the `values-customer-settings.yaml` file to match the following:

```yaml
global:
  localEngine:
    sessionSecretName: << the name of the secret created >>
    redisSecretName: << the name of the secret created >>
    s3SecretName: << the name of the secret created >>
    jwtSecretName: << the name of the secret created >>

broker-client:
  skipRemoteConfig: true
  brokerType: universal
  brokerPlatformCredentials:
    myClient:
      credentialSecretName: << the name of the secret created >>
  brokerDeployments:
    myDeployment:
      brokerPlatformRef: myClient
      credentialReferencesSecretName: << the name of the secret created >>
```

Networking

Core Networking

Re-using the Ingress definition

Follow this documentation to re-use this definition with your own Ingress Controller.

Pre-requisites

- [REQUIRED]: Obtain the Ingress Class Name for your clusters Ingress Controller.
- [RECOMMENDED]: Set up a DNS record for Snyk Code Local Engine.

Updating the inbuilt Ingress definition

Set the following in `values-customer-settings.yaml`:

```
global:
  ingress:
    enabled: true
    ingressClassName: <<name-of-ingress-class>>
    host: <<fdqn-dns-record>>
```

Where:

- `ingressClassName` is the name of the IngressClass definition on your cluster
- `host` is an optional DNS record resolving to your Ingress, which will be used by the Ingress Controller.

Custom Ingress

The inbuilt Ingress definition is used by default to automatically provision routes to the appropriate Snyk Code Local Engine services. If you wish to recreate the Ingress for any other reason, follow these steps.

Pre-requisites

All the `inbound` requests should be forwarded to `internal-proxy` service, port `10000`.

The `internal-proxy` Service is created by default, and will expose its respective backend pod(s) via `ClusterIP`.

Disabling the built-in Ingress

Set the following in `values-customer-settings.yaml`:

```yaml
global:
  ingress:
    enabled: false
```

Enabling TLS

Note: If using a custom ingress, these steps may not apply.

Adding a Certificate

By default, Snyk Code Local Engine is not secured by TLS. To enable TLS on the provided Ingress:

- a DNS record must be created,
- a certificate generated,
- the following changes made in the `values-customer-settings.yaml` file:

```yaml
global:
  ...
  localEngineUrl: "https://<HOST>"
  ...
  ingress:
    ...
    host: "<HOST>"
    tls:
      enabled: true
      secret:
        ...
        key: |
          <KEY>
        cert: |
          <CERT>
```

Where `<HOST>` is the domain used when the TLS `<CERT>` and `<KEY>` were generated.

Read more about securing an ingress with TLS here and TLS secrets in general.

Self Signed Certificates

If using a self-signed certificate to secure Snyk Code Local Engine, append the `--insecure` option to any `snyk` commands, or provide the CA to `snyk` by setting `NODE_EXTRA_CA_CERTS`. This help article provides further information and support.

Outbound Proxy Support

Note: Snyk Code Local Engine does not support proxied connections to an external SCM. The following section is for proxying outbound traffic to Snyk domains only.

Components in Snyk Code Local Engine will make TLS-secured outbound connections to Snyk domains (`app.snyk.io`, `api.snyk.io` and `broker.snyk.io` for the default US tenant) to:

- Authenticate users of the CLI and IDE.
- Make analysis results available on the Snyk Web UI.

If outbound connections to the internet require a proxy, the following may be configured in the `values-customer-settings.yaml` file:

```
global:
  ...
  proxy:
    configMapName: 'proxy-configmap' #default, do not change
    enabled: true
    url: <proxy-url>
    tlsRejectUnauthorized: [false|true]
    usePrivateCaCert: [false|true]
  ...
  privateCaCert:
    enabled: [false|true]
    cert: |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
    certMountPath: "/etc/config"
```

Where:

- `global.proxy.enabled` is set to true to use an external proxy,
- `global.proxy.url` is the proxy URL, of form `http(s)://<user>:<password>@<uri>:<port>`,
- `global.proxy.tlsRejectUnauthorized` should be set to `true` *only if* your proxy requires a certificate and one cannot be provided,
- `global.proxy.usePrivateCaCert` should be set to `true` if providing a certificate for the external proxy (see `global.privateCaCert`),
- `global.privateCaCert.cert` should be set to the full certificate chain required to trust your proxy. This should be a multi-line string in `PEM` format. This must be specified if `global.proxy.usePrivateCaCert` is `true`.

Further Proxy Configuration

- If your proxy uses whitelisting, ensure the appropriate domain for your Snyk tenant are added - for example, `app.snyk.io`, `api.snyk.io` and `broker.snyk.io` (if using broker) for the default US tenant.
- If your proxy certificate and/or authentication information changes, update the value(s) above and re-apply Helm. Any running pods that use the proxy will be recreated with the new proxy configuration.

Private Certificate Authority Support for SCM

If the SCM used with Snyk Code Local Engine requires clients to trust a private/self-signed certificate, the following may be configured in the `values-customer-settings.yaml` file:

```
global:
  privateCaCert:
    enabled: true
    cert: |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

Where:

- `global.privateCaCert.enabled` is set to true to provide a certificate to components that interact with an SCM,
- `global.privateCaCert.cert` should be set to the full certificate chain required to trust your SCM. This should be a multi-line string in `PEM` format.

Private Certificate Authority Support for both SCM and Proxy

To enforce certificate trust towards both SCM and Proxy, provide one or more CA certificates to the `global.privateCaCert.cert` key:

```
global:
  ...
  proxy:
    enabled: true
    url: <proxy-url>
    tlsRejectUnauthorized: false
    usePrivateCaCert: true
  ...
  privateCaCert:
    enabled: true
    cert: |
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
      -----BEGIN CERTIFICATE-----
      ...
      -----END CERTIFICATE-----
```

These certificates will be concatenated to a single file and mounted by all services that interact with the outbound proxy, the SCM, or both.

Post Installation

Snyk Code Local Engine Health

Snyk Code Local Engine exposes useful endpoints to interrogate the status of your installation.

`/` or `/status`

Query either `/` or `/status` to report on the status of your installation:

```
curl http[s]://<LOCAL_ENGINE_URL>/
```

or:

```
curl http[s]://<LOCAL_ENGINE_URL>/status
```

NOTE: The response will be cached by default for 60 seconds.

Healthy Response

The following message is returned if all pods and services are running:

```
{
    "message": "Snyk Code is healthy!  ",
    "ok": true,
    "version": "<version_of_snyk_code_local_engine>"
}
```

Unhealthy Response

A JSON object is returned with details of the unhealthy workloads. If response code `503` is returned, the `service-health-aggregator` pod may be unable to start correctly.

`/broker/healthcheck`

The `/broker/healthcheck` endpoint ensures the `broker-client` pod is able to communicate with Snyk correctly.

```
curl http(s)://<LOCAL_ENGINE_URL>/broker/healthcheck
```

Healthy Response

```
{
  "ok": true,
  "websocketConnectionOpen": true,
  "brokerServerUrl": "https://broker.snyk.io",
  "version": "<version_of_broker>",
  "transport": "websocket"
}
```

`/api/healthcheck`

The `/api/healthcheck` endpoint tests the health of the `sast-analysis-api` pod.

```
curl http[s]://<LOCAL_ENGINE_URL>/api/healthcheck
```

Healthy Response

```
{ "gitSha": "sha", "ok": true }
```

Snyk Configuration

Note: At this point, you should have a running installation of Snyk Code Local Engine. As IP addresses may be dynamic/change, it is highly recommended to set up a DNS record.

If using either the Snyk CLI or PR Checks, Snyk requires one of the following once a Snyk Code Local Engine installation is complete:

- The DNS name of the ingress of the cluster, or
- The IP address of the ingress of the cluster

This allows the Snyk CLI/IDE plugins to function correctly.

To specify the DNS name or IP address of the ingress:

- Add the following to `values-customer-settings.yaml`:

```
global:
  ...
  localEngineUrl: <either your assigned DNS record or your IP address - for example, http://local-engine.domain or http:/10.170.1.40>
```

- Execute `helm upgrade --wait <DEPLOYMENT_NAME> snyk-code-local-engine-<VERSION>.tgz -i -f values-customer-settings.yaml`

If an IP address was previously provided to your Snyk representative, please update us with your new DNS record.

Enabling Snyk Code Local Engine for Snyk CLI/IDE

Once enabled, the Snyk Code Local Engine URL/IP address will be shown in `Settings`, under the `Snyk Code` section.

Configure the Snyk CLI

If using a particular Snyk Organization for Snyk Code Local Engine, ensure that either:

- The `--org=<LOCAL_ENGINE_ORG_NAME>` flag is set when using the Snyk CLI, or
- `snyk config set org=<LOCAL_ENGINE_ORG_NAME>` is executed

This ensures Snyk Code analysis requests are directed to Snyk Code Local Engine.

To confirm, check the output of `snyk code test -d`:

```
...
snyk-code ---> API request log  => HTTP GET http://<LOCAL_ENGINE_URL_OR_IP>/api/filters
...
```

Updating Snyk Code Local Engine - Snyk Configuration

If either of these change:

- The DNS name assigned to the ingress of the cluster, or
- If not using DNS records, the IP address assigned to the ingress of the cluster, or
- TLS encryption is enabled/disabled,

Snyk must be provided with the updated DNS name/IP address.

What Next?

Once your cluster is configured, Snyk Code Local Engine is ready for use. Results will appear on the Snyk Web UI after setting up PR Checks or importing Projects.

Helm Parameters

The available Helm Parameters are listed below.

Parameters

Enable Snyk Code Services

Name	Description	Value
`tags.scm`	Enable services required for SCM Imports	`false`
`tags.scmPrCheck`	Enable services required for SCM PR Checks	`false`
`tags.cli`	Enable services required for CLI	`false`
`tags.ide`	Enable services required for IDE	`false`

Global Parameters

Name	Description	Value
`global.imagePullSecret.enabled`	Disable if an Image Pull Secret is not required	`true`
`global.imagePullSecret.name`	The name of the Image Pull Secret to be created	`snyk-code-local-engine-pull-secret`
`global.imagePullSecret.credentials.username`	A Username to authenticate against a Docker registry	`""`
`global.imagePullSecret.credentials.password`	A Password to authenticate against a Docker registry	`""`

Name	Description	Value
`global.imagePullSecrets`	Set to '[]' if your image registry does not require authentication, or '[`<existing-secret-name>`] if re-using credentials that already exist on your Kubernetes cluster	`["snyk-code-local-engine-pull-secret"]`
`global.imageRegistry`	Optionally define a private image registry address if using non-default image registries (hostname/port only, no protocol)	`""`
`global.localEngineUrl`	The full URL including schema that points to the cluster ingress. Required for CLI/PR Checks.	`""`
`global.proxy.enabled`	Set to true to enable outbound proxy support	`false`
`global.proxy.url`	Proxy URL, including schema: http[s]://username:password@proxy:port	`""`
`global.proxy.tlsRejectUnauthorized`	Set to true to trust any and all certificates presented by the proxy	`false`
`global.proxy.usePrivateCaCert`	Set to true to enable private CA support for the proxy - see global.privateCaCert	`false`
`global.proxy.noProxyHosts`	Provide a comma separated list of hostnames. Required when Broker Client is running in universal mode, a proxy is used, *and* when `broker-client.skipRemoteConfig` is `false`.	`""`
`global.privateCaCert.enabled`	Set to true to enable trust of private CA certificate(s) towards the SCM and/or proxy	`false`
`global.privateCaCert.cert`	Multiline string containing any/all certificates for connections to the SCM and/or proxy in PEM format ()	`""`
`global.privateCaCert.certMountPath`	A fully qualified path to mount the certificate	`/etc/config`
`global.ingress.enabled`	Set to true to create an Ingress for CLI/PR Checks	`false`

Name	Description	Value
`global.ingress.host`	Optionally define the host associated with this ingress - otherwise leave blank	`""`
`global.ingress.ingressClassName`	Optionally define the Ingress Class for this ingress - otherwise leave blank	`""`
`global.ingress.annotations`	Optionally define any annotations to add to the ingress - otherwise leave blank	`{}`
`global.ingress.tls.enabled`	Set to true to enable TLS on the in-built ingress	`false`
`global.ingress.tls.secret.name`	Either specify the name of a pre-existing Kubernetes secret containing TLS secrets, or leave blank to create a new secret	`""`
`global.ingress.tls.secret.key`	The TLS key for TLS encryption, in PEM format	`""`
`global.ingress.tls.secret.cert`	The TLS certificate for TLS encryption, in PEM format	`""`
`global.localEngine.redisSecretName`	Optionally specify the name of a pre-existing Kubernetes secret containing Redis authentication data	`""`
`global.localEngine.sessionSecretName`	Optionally specify the name of a pre-existing Kubernetes secret containing session secret data	`""`
`global.localEngine.s3SecretName`	Optionally specify the name of a pre-existing Kubernetes secret containing S3 authentication data	`""`
`global.localEngine.jwtSecretName`	Optionally specify the name of a pre-existing Kubernetes secret containing S3 encryption data	`""`

Deeproxy parameters (EU/AU Snyk Tenants only)

Name	Description	Value
`deeproxy.verificationEndpoint`	Optionally specify the geographically-appropriate endpoint. Set to US tenant by default	`https://api.snyk.io/v1/validate/token/snyk-to-deepcode-proxy-validation`

Broker Client parameters (any SCM flows only)

Name	Description	Value
`broker-client.brokerToken`	Broker Token is a value from Snyk. Get this from the Snyk integration settings page or your Snyk Representative	`""`
`broker-client.brokerType`	Define the SCM Broker will connect to	`""`
`broker-client.universalBrokerConnections`	A map of Universal Broker client connections. Ignored if `broker-client.skipRemoteConfig` is `false`.	`{}`
`broker-client.skipRemoteConfig`	Set to `false` to fetch configuration from the Snyk Platform	`true`
`broker-client.brokerPlatformCredentials`	A map of Universal Broker platform credentials, containing clientId and clientSecret. Ignored if `broker-client.skipRemoteConfig` is `true`.	`{}`
`broker-client.brokerDeployments`	A map of Universal Broker deployments, containing a brokerPlatformRef, a deploymentId, and one or more Credential References. Ignored if `broker-client.skipRemoteConfig` is `true`.	`{}`
`broker-client.githubHost`	GHE URL - Ex: your.ghe.domain.com (do not prepend HTTPS) - For GHE Cloud use api.github.com	`""`
`broker-client.githubApi`	GHE API Address - do not prepend HTTPS	`""`
`broker-client.githubGraphQl`	GHE Graph QL Address - do not prepend HTTPS	`""`

Name	Description	Value
`broker-client.githubToken`	Github Token - for GitHub or GitHub Enterprise	`""`
`broker-client.bitbucketUsername`	Bitbucket Server Username	`""`
`broker-client.bitbucketPassword`	Bitbucket Server Password	`""`
`broker-client.bitbucketHost`	Bitbucket Server Host URL - do not prepend HTTPS	`""`
`broker-client.bitbucketToken`	Bitbucket Server Personal Access Token	`""`
`broker-client.gitlabHost`	GitLab URL - do not prepend HTTPS	`""`
`broker-client.gitlabToken`	GitLab Token	`""`
`broker-client.azureReposOrg`	Azure Repos Organization	`""`
`broker-client.azureReposHost`	Azure Repos Hostname - do not prepend HTTPS	`""`
`broker-client.azureReposToken`	Azure Repos Token	`""`
`broker-client.codeSnippet.enabled`	Set to enable viewing of analysis results in the Snyk UI. Caution - will allow Snyk servers to fetch source code files	`false`

Name	Description	Value
`broker-client.largeManifestFileRule.enabled`	Set to enable in order to be able to fetch large manifest files (> 1Mb). Available only for 'github' and 'github-enterprise' broker types/connections	`false`
`broker-client.brokerServerUrl`	Modify if using Snyk Private Cloud/a different Snyk tenant.	`https://broker.snyk.io`
`broker-client.highAvailabilityMode.enabled`	Set to true for High Availability Broker. Will take effect with 2, 3 or 4 replicas of Broker	`false`
`broker-client.brokerDispatcherUrl`	Modify if using Snyk Private Cloud/a different Snyk tenant. Required if `highAvailabilityMode.enabled=true`.	`https://api.snyk.io`
`broker-client.replicaCount`	The number of Broker Clients to run. Set if using High Availability Broker (maximum 4 replicas)	`1`

Changelog

All notable changes to Code Local Engine project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

v2.12.1

2025-09-02

Changed

- Updated `broker-client` to 4.218.2
- Updated `redis` dependencies to 7.4.0-debian-12-r0
- Updated `minio` dependencies to 2025.7.23-debian-12-r3
- Snyk services updated
- Bitnami images are now pulled from the `bitnamilegacy` repository as a consequence of this announcement: https://github.com/bitnami/charts/issues/35164

v2.12.0

2025-05-28

Added

- Snyk Code Local Engine supports the Universal Broker with remote configuration. This requires existing Legacy Broker connections to be migrated to connections represented on the Snyk Platform. Further detail is contained within documentation.

Removed

- The `code-pr-check-service` is removed.

v2.11.0

2025-01-09

Changed

- Snyk Code rules updated

- Updated `broker-client` to 4.203.4
- Update routing to handle Code PR checks in `sast-analysis-api` instead of `code-pr-check-service`. The `code-pr-check-service` has been made redundant and is pending removal.

Removed

- The `broker-client` no longer supports body logging (`broker-client.logEnableBody`). This value is deprecated, and has no effect if specified.

Fixed

- `scm-bundle-store` and `minio` images upgraded to resolve a vulnerability in the Golang SSH package. This vulnerability was not reachable in SCLE.

v2.10.0

2024-10-28

Added

- `broker-client` supports the GitHub Server App integration type - this functionality is in Closed Beta. Discuss with your Snyk representative for enablement.
- New documentation to support configuration of GitHub Server App connections with Snyk Code Local Engine

Fixed

- Corrected `github-com` to `github` when specifying a Universal Broker connection to `github.com`
- Removed the `ephemeral-storage` limit for `scm-bundle-store`. This prevents pods that were previously being replaced by Kubernetes when exceeding `ephemeral-storage` limits remaining until cluster-level garbage collection occurs, keeping health checks accurate.

Changed

- Snyk Code rules updated
- Updated `broker-client` to 4.196.7

v2.9.0

2024-08-19

Added

- Included `.snyk` ignore file for Minio
- Add Snyk Code Local Engine documentation as a pdf

Changed

- `broker-client` now deploys as a statefulset to increase stability when running in HA mode
- Updated `scm-bundle-store` to use Minio instead of MongoDB for backend storage
- `global.localEngine.mongodbSecretName` is deprecated, replaced by `global.localEngine.s3SecretName` and `global.localEngine.jwtSecretName`.
- Updated `broker-client` to 4.193.4

Removed

- References to the MongoDB image are removed

Fixed

- Removed the deprecated form of `deeproxy.verificationEndpoint` from `values-customer-settings.yaml`, using `api.snyk.io` instead
- Added `broker-client.brokerDispatcherUrl` to `values-customer-settings.yaml` for High Availability Broker

v2.8.2

2024-06-21

Added

- Support for Personal Access Token in Bitbucket Server

Fixed

- Resolved an ingress deployment failure when TLS secret name is provided. The fix ensures that Snyk Code Local Engine can now correctly use a pre-existing TLS secret for the certificate and key material when specified
- Corrected Broker behaviour when encountering non-ASCII characters in payloads
- Resolves some C++ analyses reporting all issues on line 1

Changed

- Updated `broker-client` to 4.190.3
- Update Snyk Code services with latest rulesets

v2.8.1

2024-04-15

Fixed

- Updated values-customer-settings.yaml file with new settings for Universal Broker

Changed

- Updated MongoDB image to Debian 12 version
- Updated ignores for MongoDB
- Updated `broker-client` to 4.181.1

v2.8.0

2024-03-14

Fixed

- Removed local analysis queue debug endpoints
- Corrected a filter for the GitLab Snyk Broker that would cause some requests to fail

Changed

- Updated the list of images to remove the standalone `mongodb` image, which is no longer required
- Updated `broker-client` to v4.179.3

Added

- Support for multiple SCMs/instances of SCMs via Broker in Universal mode
- Updated snyk ignore file for Redis

v2.7.11

2024-02-08

Added

- Added versioned snyk ignore (.snyk) files. These files detail any vulnerabilities within Snyk Code Local Engine that are either unreachable or otherwise not valid.

Fixed

- Resolved a Helm validation bug for Broker

Changed

- Snyk Code rules updated
- Updated `broker-client` to v4.174.1

Removed

- `scm-bundle-store.server.useTokenAuth` is now deprecated - repository detection should ensure the presence of required headers for self-hosted Azure DevOps/TFS servers. This value now has no effect.

v2.7.10

2024-01-26

Added

- `scm-bundle-store.server.useTokenAuth` for compatibility with self-hosted Azure DevOps Server

Fixed

- Corrected documentation for using EU or AU Snyk tenants with Snyk Code Local Engine
- Resolved a bug that caused git requests to Azure DevOps Server to fail

Changed

- Snyk Code rules updated
- Introduced additional validation rules for EU or AU Snyk tenant usage
- Updates the default Snyk API domain from `https://snyk.io` to `https://api.snyk.io`
- Updated the `broker-client` to v4.172.6

v2.7.9

2024-01-16

Fixed

- Resolved a bug preventing cleanup jobs from running successfully

Changed

- Updates to Snyk images for updated rulesets
- Updated the `broker-client` to v4.172.2

v2.7.8

2024-01-11

Changed

- Documentation updated to include alternative tenant setup
- Updates to Snyk images for new rulesets
- Updated the `broker-client` to v4.171.9

v2.7.7

2023-12-07

Fixed

- Snyk Code rules updated
- Suggest services updated JDK to remove vulnerabilities
- Documentation updated to remove "Overview" section and streamline introduction to Snyk Code Local Engine

v2.7.6

2023-12-07

Added

- CronJobs to clean up older/expired data in MongoDB

Changed

- Minimum Kubernetes version is now 1.21

v2.7.5

2023-11-27

Fixed

- Fixed non-reachable vulnerabilities in the scm-bundle-store and mongodb components.

v2.7.4

2023-11-23

Changed

- Corrected the list of images under the Private Registry section

Removed

- The `scm-meld` component is no longer required and has been removed

Fixed

- Resolved an "Unauthorized" failure during IDE and CLI scans occurring after a proxy CA certificate change. The fix ensures that Snyk Code Local Engine properly picks up the new configuration when redeployed.

v2.7.3

2023-11-10

Changed

- The `/status` endpoint is now also presented on `/` for better compatibility with Load Balancer health checks

v2.7.2

2023-11-08

Changed

- Updated the `broker-client` to v4.169.2
- Updates to latest Snyk Code rules
- Updates to Snyk Code services

Fixed

- Fixed outbound CA support for SCM when a proxy is not utilized

v2.7.1

2023-10-17

Changed

- Updated the architectural diagram with `suggest-sticky` component

v2.7.0

2023-10-17

Added

- Introduced a new `largeManifestFileRule` value, gives the option to add rule for fetching large manifest file. Avaliable for Github and Github Enterprise only.
- Caching mechanism for IDE scans by the new `suggest-sticky` component.

Changed

- Updated architecture diagram with new internal-proxy connectivity

v2.6.1

2023-09-20

Changed

- Updated Ingress template to include the `host` key if specified
- Updated documentation for JetBrains IDE

v2.6.0

2023-09-18

Added

- `internal-proxy` component (based on `envoy`) replaces routes previously defined by the Ingress resource

Changed

- Updated the `broker-client` to v4.163.0
- Ingress resource simplified - defies one route (`/`) and removes the need for request rewriting/regex capture groups

Fixed

- Updated documentation for proxy and custom Certificate Authority support for better clarity
- Specify the `brokerServerUrl` by default in the `values-customer-settings.yaml` file

Removed

- Any references to the previously-used MongoDB Sharded cluster in documentation
- The pre-packaged NGINX Ingress Controller is removed. Functionality is handled internally by the `internal-proxy` component

v2.5.0

2023-09-04

Added

- Allows python projects that use poetry to be scanned by Snyk Open Source through the broker

Changed

- Updated the `broker-client` to v4.161.0
- Updated Snyk Code services for latest analysis rules
- Changed database infrastructure for `scm-bundle-store` from a sharded MongoDB cluster to a single MongoDB instance

Fixed

- Fixed an upgrade/stability issue with MongoDB by migrating to a single MongoDB instance

v2.4.2

2023-08-17

Added

- Snyk Code Local Engine now supports custom CAs towards SCMs via `global.privateCaCert.*` values.
- A subset of available Helm values are listed in documentation
- A subset of available Helm values are subject to input validation
- The IDE has been added to the Architecture diagram

Changed

- NGINX Ingress documentation has been updated to better reflect usage and deployment options

Deprecated

- The `global.proxy.cert` and `global.proxy.useCustomCert` values are both *deprecated*.

v2.4.1

2023-07-14

Added

- The inbuilt NGINX Ingress Controller is now disabled by default, and is separate from the Ingress resource. This enables customers to re-use their own instance of NGINX Ingress Controller without manually manipulating the Chart.
  - To enable the NGINX Ingress Controller, set `global.ingressController.enabled: true`.

v2.4.0

2023-07-13

Added

- Support for custom image registries:
  - Authenticated/unauthenticated private registries
  - Custom image pull secrets

v2.3.0

2023-07-12

Added

- IDE Scans for VSCode v1.21 and higher

v2.2.3

2023-06-15

Added

- Update of scm-meld to support custom CA override
- Update of files-bundle-store to improve CPU usage, and concurrency

v2.2.2

2023-06-13

Fixed

- Suggest has been upgraded with some key bug fixes:
  - Better queueing mechanism to reduce stuck analyses
  - Introduced better analyses timeout mechanisms
  - Suggest runs as non-root

v2.2.1

2023-06-09

Added

- Migrates additional services to run as non-root

Fixed

- Inconsistency when deploying Local Engine to a custom namespace
- Webhook creation for PR checks

v2.2.0

2023-05-12

Added

- Modular service deployment, only deploy the services needed for the intended use case
- PR check functionality
- Ability to configure self managed secrets
- Partial standardisation of service base images (more to follow)
- Partial migration of services not to run as root anymore (more to follow)
- Updates core Snyk Code services to include new rule sets

Removed

- We removed CRDs and ClusterRoles - no more cluster-wide access needed.

v2.0.0

2023-04-20

Added

- Includes the “new” Snyk Code stack, giving customers parity between Snyk SaaS and Local Engine environments.
- We host the Helm Chart on Dockerhub - customers can pull the Helm Chart with the same credentials for v1.Documentation and the values-customer-settings.yaml are still shared manually with the customer.

Removed

- We removed CRDs and ClusterRoles - no more cluster-wide access needed.

Changed

- This release does not include PR Checks. CLI scans/ Imports are currently supported.
- This release does not include pulling images from a custom registry.
- This release does not include centralised logging.