DeepTracy Documentation

Release 0.0.1

Roberto Abdelkader Martínez Pérez

Dec 11, 2020

CONTENTS

1 What's Deeptracy **1**

2 Documentation **3**

3 Contributing **5**

4 License **7**

5 Documentation Index **9**

5.1 Architecture 9
5.2 Quickstart 11
5.3 Configuration File 13
5.4 Deeptracy plugins 15
5.5 Usage 16
5.6 Running Deeptracy 17
5.7 API 18

HTTP Routing Table **23**

CHAPTER

ONE

WHAT'S DEEPTRACY

Deeptracy scans your project dependencies to spot vulnerabilities.

Is a meta tool to analyze the security issues in third party libraries used in your project.

We have created **this project to simplify this process** so you can focus only in the important: your project.

Deeptracy can choose the most suitable security tools for each languages and notify the spotted vulnerabilities in the project dependencies.

CHAPTER

TWO

DOCUMENTATION

You can learn more about Deeptracy in the official documentation.

CHAPTER

THREE

CONTRIBUTING

Any collaboration is welcome!

There're many tasks to do. You can check the Issues and send us a Pull Request.

CHAPTER

FOUR

LICENSE

This project is distributed under Apache License.

CHAPTER

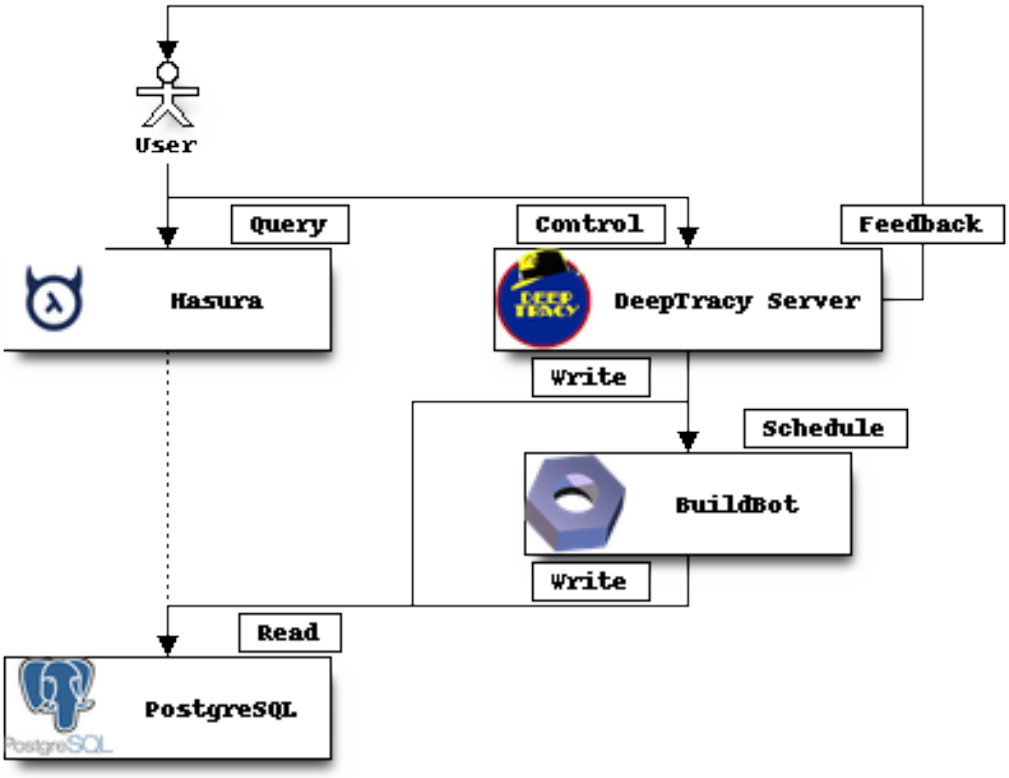
FIVE

DOCUMENTATION INDEX

5.1 Architecture

5.1.1 Components

DeepTracy is composed of several components described in the following diagram:

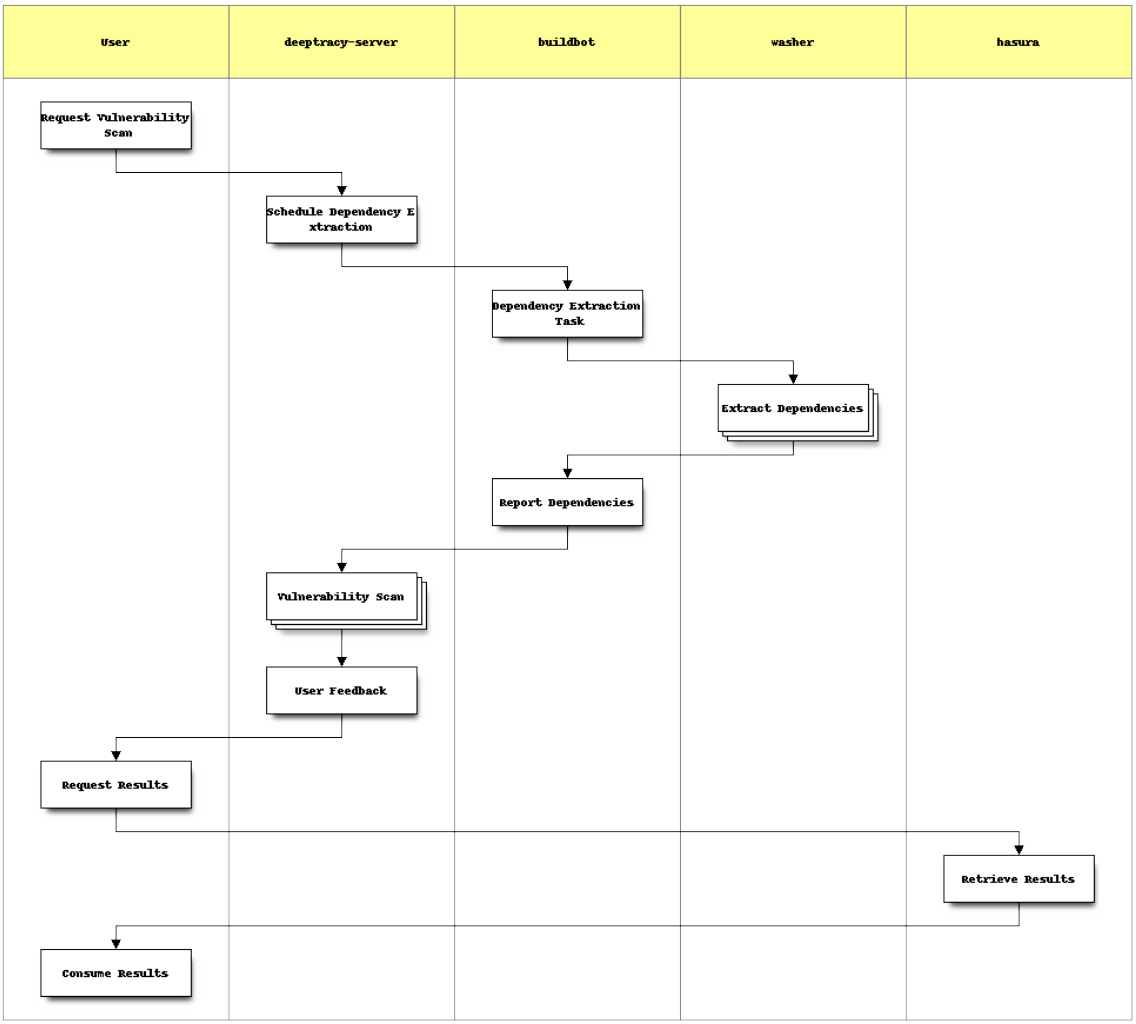


Name	Description
User	A system capable of requesting new vulnerability analysis and retrieving results
BuildBot	Dependency extraction
Hasura	Provides data API for the user
PostgreSQL	Persistence layer
DeepTracy Server	Task orchestration through control API

5.1.2 Interactions

The following activity diagram summarizes the normal interaction among the components of the system.

Note: This conceptual diagram describe the type of interactions but not how they are performed. In other words, this diagram does not describe if the interactions are synchronous nor asynchronous.



Name	Description
Request Vulnerability Scan	User request to schedule a vulnerability scan over a source repository
Schedule Dependency Extraction	Ask buildbot to perform the dependency extraction process in the given repository/commit
Dependency Extraction Task	Use washer docker containers to extract dependencies
Extract Dependencies	Launch docker containers with the appropiate environments and extract project(s) dependencies
Report Dependencies	Report dependency list to DeepTracy Server
Vulnerability Scan	Scan for vulnerabilities on the retrieved dependencies using vulnerability providers
User Feedback	The provided webhook is called back to acknowledge the user that the scan is finished
Request Results	Using GraphQL© query language the user request the scan information
Retrieve Results	Results are queried and retrieved from the database
Consume Results	:)

5.2 Quickstart

All system components are provided as Docker containers and a docker-compose configuration exists to assist the user on launching a testing environment.

5.2.1 Launching the Testing Environment

The testing environment is managed through the *Makefile* present in the project root directory.

```
$ make
```

Executing the previous code in a Linux shell will setup a fresh testing environment.

Warning: The previous command will remove any existing data.

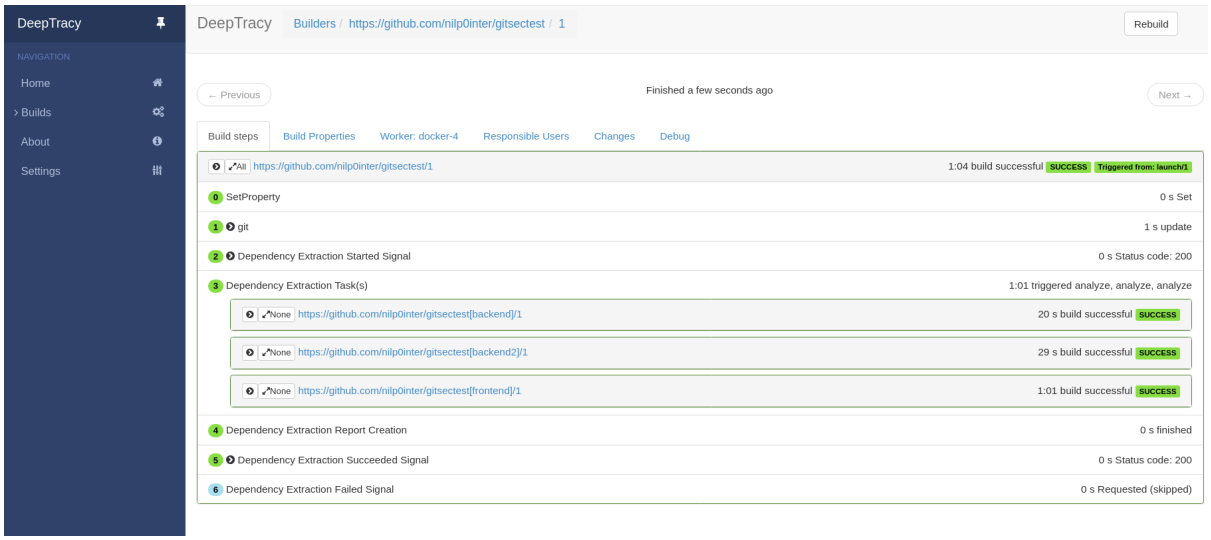
5.2.2 Testing Environment Services

The following table contains the list of open services on the Testing Environment.

Service	URL	Type
graphql-engine	http://localhost:8080/console	Web Application
graphql-engine	http://localhost:8080/v1alpha/graphql	Web Application
deeptracy-server	http://localhost:8088	REST API
deeptracy-buildbot	http://localhost:8010	Web Application

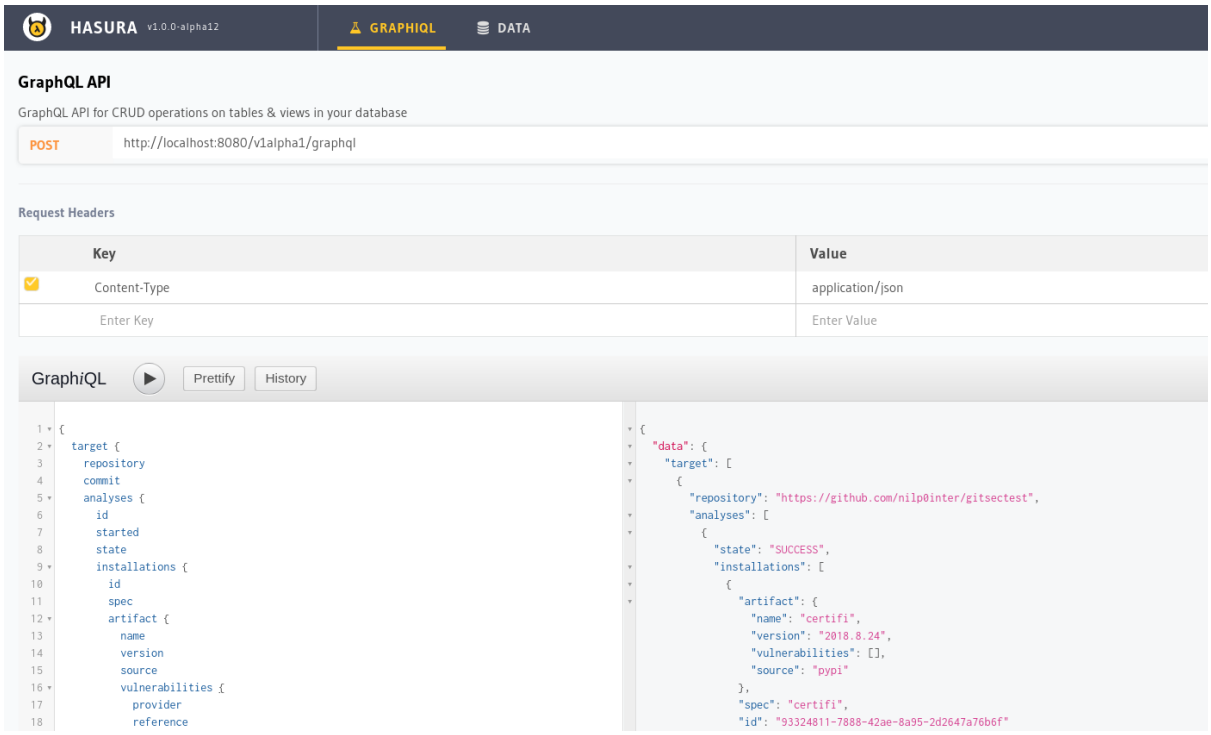
Buildbot

Buildbot console provides a convenient way of debugging the status of the Dependency Extraction Phase of Deeptracy.



Hasura

Hasura Console helps the user on composing GraphQL queries.



5.2.3 Makefile Targets

The main *Makefile* provides some convenient targets.

image

Generate the main docker image used by *DeepTracy Server* and *buildbot*.

start

Bring up the testing environment using docker-compose.

stop

Bring down the testing environment using docker-compose.

down

Destroy the testing environment using docker-compose and remove all data.

logs

Print the testing environment log files to stdout.

status

Shows the status of the different components of the testing environment.

plugins

Build all plugin docker images.

5.3 Configuration File

DeepTracy's analysis is driven by a YAML configuration file.

5.3.1 Load Strategy

This configuration file can be load from several places:

- Analyzed respository root directory.
- On the POST analysis request (not implemented yet).
- Automatically generated from the analyzed repository using heuristics (not implemented yet).

5.3.2 File Format

projects key

A list of *projects* to be scanned.

projects:<name>:type

The name of the buildbot-washer docker image to be used for this scan.

projects:<name>:strategy

The name of the buildbot-washer task to be executed by the image.

Note: This value depends on the *type*

---

projects:<name>:unimportant

Boolean value. If *true* the scan will continue even if this particular scan fails.

projects:<name>:config

Object containing the particular configuration of the strategy.

Note: Some of the key-value pairs in this object are specific to the *type* while others may be common to all of them.

---

projects:<name>:config:path

The location to be scanned relative to the project root.

5.3.3 Configuration Example

```
projects:
    backend:
        type: deeptracy-python:2.7
        strategy: requirement_file
        unimportant: false
        config:
            path: src/backend
            requirement: requirements.txt
    backend2:
        type: deeptracy-mvn:3.5-jdk-9
        strategy: mvn_dependencytree
        unimportant: false
        config:
            path: src/backend2
    frontend:
        type: deeptracy-node:8
        strategy: npm_install
        unimportant: false
```

(continues on next page)

(continued from previous page)

```
        config:
            path: src/frontend
```

5.4 Deeptracy plugins

The dependency extraction proccess is carried out by Buildbot. It leverages on a plugin architecture in which separated components, the plugins, provide different ways, called tasks, of doing de extraction. Each plugin gives support for a particular programming language.

5.4.1 Available plugins

Currently Deeptracy offers several plugins to do dependency extraction that give support to the main programming languages.

Dependencycheck

This plugin is intended for java projects, and uses the OWASP Dependencycheck utility (version 4.0.2) to do the dependency extraction.

It publishes one task 'dependency_check'.

Maven

A set of plugins intended for java projects that use Maven to do the dependency extraction, each plugin gives support for a specific version of Maven.

Each plugin publishes one task 'mvn_dependencytree'

Npm

A set of plugins intended for javascript projects that use Npm to do the dependency extraction, each plugin gives support for a specific version of Npm.

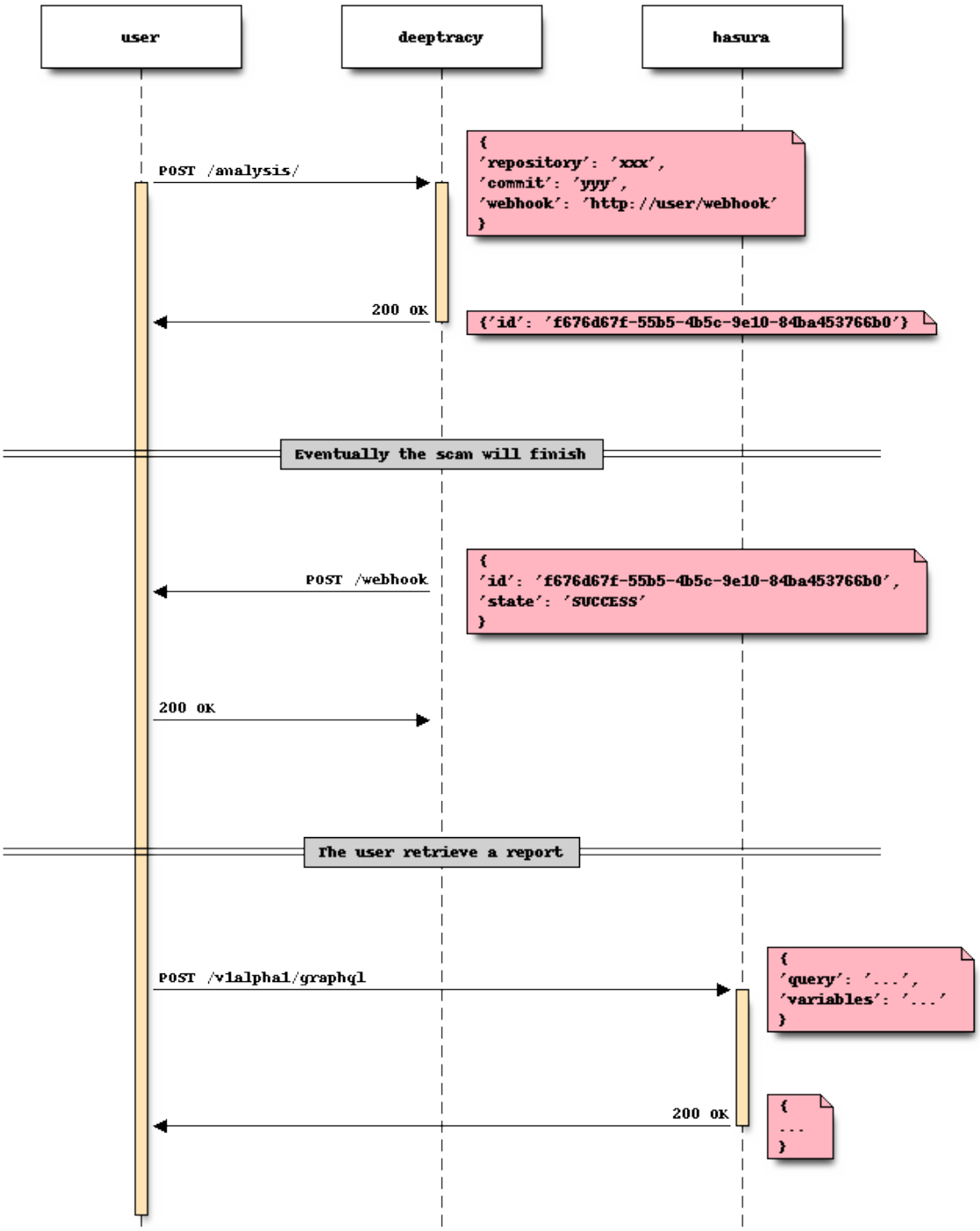
Each plugin publishes one task 'npm_install'

python

A set of plugins intended for Python project, each plugin gives support for a specific version of Python.

Each plugin publishes two task: - 'requirement_file', for doing the dependency extraction by analyzing the project's requirements.txt file. - 'pip_install', for doing the dependency extraction by using the pip utility.

5.5 Usage



The sequence diagram above shows the typical usage workflow.

5.6 Running Deeptracy

Deeptracy service is composed of several pieces, a docker-compose project has been created in the compose directory in order to ease deployment and tests. These are the files and their purpose:

- .env Contains the environmental variable values needed to authenticate against a database server.
- deeptracy-config.env Contains the environmental variable values to configure the deeptracy server instance.
- docker-compose.yml Starts all the containers needed for the service. POSTGRES_HOST environment variable must be provided (in command line or ,env file) in order to provide the database to the containers if not using the database compose file.
- docker-compose-database.yml Starts a container with a postgresql database and configures the deeptracy containers to connect to this instance.
- Dockerfile.hasuracli and hasura directory Used for configuring GraphQL engine against Deeptracy's database.

5.6.1 Deploy with internal database

In order to start a fully containerized environment run:

```
> docker-compose -f docker-compose.yml -f docker-compose-database.yml up
```

5.6.2 Deploy with external database

If you want to run against an existing database server run:

```
> docker-compose -f docker-compose.yml -e POSTGRES_HOST=somehost up
```

5.6.3 Docker images

Each component of the Deeptracy server has been published as a container in the BBVALabs' organization at Docker Hub. Each container can be configured by using environmental variables:

Buildbot

The following variables are used to configure the Buildbot server container:

- DOCKER_HOST (default="unix://var/run/docker.sock") For container management.
- WORKER_IMAGE_AUTOPULL default=True) Pull needed images.
- WORKER_INSTANCES (default=16) Number of instances to start.
- WORKER_IMAGE_WHITELIST (default=*) Comma separated list of allowed image shell-like patterns.
- BUILDBOT_MQ_URL (default=None) MQ endpoint if used.
- BUILDBOT_MQ_REALM (default="buildbot") MQ realm if MQ is used.
- BUILDBOT_MQ_DEBUG (default=False) Activate MQ debug.
- BUILDBOT_WORKER_PORT (default=9989) TCP port used by buildbot workers.
- BUILDBOT_WEB_URL (default="http://localhost:8010/") URL of Buildbot's web UI.
- BUILDBOT_WEB_PORT (default=8010) Port in which Buildbot web UI is listening.
- BUILDBOT_WEB_HOST (default="localhost") Host in which Buildbot web UI is listening.

- BUILDBOT_DB_URL (default="sqlite://") Database used by Buildbot to store its state.
- DEEPTRACY_SERVER_CONFIG (default=None) Defaults to use in repository analysis.
- DEEPTRACY_WORKER_IMAGE (default="bbvalabsci/gitsec-worker") Image used to clone repository and parse deeptracy.yml file for repository configuration.
- DEEPTRACY_BACKEND_URL (default=None) URL of Deeptracy server to use.

Deeptracy

The following variables are used to configure the Deeptracy server container:

- POSTGRES_HOST (default=None) Database server name.
- POSTGRES_DB (default='deeptracy') Database name.
- POSTGRES_USER (default=None) Database username.
- POSTGRES_PASSWORD (default=None) Database password.
- REDIS_HOST (default=None) Redis' listening address.
- REDIS_PORT (default=6379) Redis' listening port.
- REDIS_DB (default=0) Redis' listening .
- BUILDBOT_API (default='http://deeptracy-buildbot:8010') Buildbot's URL.
- PATTON_HOST (default='patton.owaspmadrid.org:8000') Patton's host and port.
- SAFETY_API_KEY (default=None)
- BOTTLE_MEMFILE_MAX (default=2048)
- MAX_ANALYSIS_INTERVAL (default=86400)
- HOST (default='localhost') Server's listening address.
- PORT (default=8088) Server's listening port.
- DEBUG (default=False) Activate server debug mode.

By default the ports exposed by each server are:

- 8010 Buildbot server.
- 8080 GraphQL engine.
- 8088 Deeptracy server.
- 9989 Buildbot worker.

5.7 API

DeepTracy API is divided in two parts: a *control* API and a *query* API.

5.7.1 Control API

The control API is provided by **DeepTracy Server** service and allows the user to manage the scanning tasks with a minimal REST API.

POST /analysis/
Create and start a new analysis for the given repository & commit.

Requires a JSON object with the following parameters:

- repository: The repository.
- commit: The commit.
- webhook (optional): Webhook to notify to when the analysis finish.

Example:

```
{"repository": "https://github.com/nilp0inter/gitsectest",
 "commit": "fdd09edd73f3fe87ea4265eeddb95935c7d25a51",
 "webhook": "http://myapp.com/analysis-finished"}
```

Returns a JSON object containing the id of the created analysis.

```
{"id": "b6e98743-7830-4aef-adf6-6a0b022f778a"}
```

PUT /analysis/ (*analysis_id*) **/extraction/started**
Signal from buildbot that the extraction phase for an analysis has started.

Note: Internal API

PUT /analysis/ (*analysis_id*) **/extraction/succeeded**
Dependency extraction phase succeeded.

Must contain a JSON object with the number of tasks spawned in the server (requests made to */dependencies* and */vulnerabilities* endpoints.

Example result:

```
{'task_count': <int>}
```

Note: Internal API

PUT /analysis/ (*analysis_id*) **/extraction/failed**
Dependency extraction phase failed.

Note: Internal API

POST /analysis/ (*analysis_id*) **/**
*execution_id***/dependencies** Installation data from buildbot.

Requires a JSON list of objects with the following keys:

- installer: The system used to install the dependency.
- **spec: The full specification used by the user to request the** package.
- source: Entity providing the artifact.
- name: The real package name.
- version: The installed version of the package.

Note: Internal API

POST /analysis/ (*analysis_id*) **/** *execution_id***/vulnerabilities** Vulnerability data from buildbot.

Requires a JSON list of objects with the following keys:

- provider: Name of the system providing the vulnerability information.
- reference: Provider unique identifier of the vulnerability.
- details: Extended JSON metadata.
- **installation: JSON object containing:**
  - installer: The system used to install the dependency.
  - **spec: The full specification used by the user to request the** package.
  - source: Entity providing the artifact.
  - name: The real package name.
  - version: The installed version of the package.

Note: Internal API

5.7.2 Query API

The query API is provided by **Hasura** service allowing the user retrieve any report structure she wants using GraphQL language.

The complete reference manual for Hasura can be found here.

The following example illustrate how to request all the vulnerabilities for a given analysis:

GET /v1alpha1/graphql
Perform the given query and return a JSON object with the results.

Example request

```
POST /v1alpha1/graphql HTTP/1.1
Content-Type: application/json

  {
    "query": "{\n    analysis (where: {id: {_eq: \"904a2117-1da1-4c9c-a3d5-
↪b03262f53d97\"}}){\n      state\n      installations {\n      \tspec\n
↪ artifact {\n            name\n          \tversion\n
↪vulnerabilities {\n              provider\n              reference\n
↪     details\n            }\n        }\n      }\n    }\n}\n",
    "variables": null
  }
```

Example response

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Fri, 24 Aug 2018 11:21:03 GMT
Server: Warp/3.2.22
Access-Control-Allow-Origin: http://localhost:8080
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,POST,PUT,PATCH,DELETE,OPTIONS
```

(continues on next page)

(continued from previous page)

```
Content-Type: application/json; charset=utf-8

  {
    "data": {
      "target": [
        {
          "repository": "https://github.com/nilp0inter/gitsectest",
          "analyses": [
            {
              "state": "SUCCESS",
              "installations": [
                {
                  "artifact": {
                    "name": "Django",
                    "version": "1.4.4",
                    "vulnerabilities": [
                      {
                        "reference": "CVE-2013-1443",
                        "details": {
                          "cve": "CVE-2013-1443",
                          "score": 5,
                          "summary": "The authentication framework (django.
↪contrib.auth) in Django 1.4.x before 1.4.8, 1.5.x before 1.5.4, and 1.6.x␣
↪before 1.6 beta 4 allows remote attackers to cause a denial of service (CPU␣
↪consumption) via a long password which is then hashed."
                        },
                        "provider": "patton"
                      }
                    ],
                    "source": "pypi"
                  },
                  "spec": "django",
                  "id": "c4cc6c32-de4e-457d-a1b0-14adeeeaeec4"
                },
                {
                  "artifact": {
                    "name": "org.springframework.boot:spring-boot-starter-web",
                    "version": "1.1.5.RELEASE",
                    "vulnerabilities": [],
                    "source": "central.maven.org"
                  },
                  "spec": "org.springframework.boot:spring-boot-starter-
↪web:jar:1.1.5.RELEASE:compile",
                  "id": "b0ea360d-60a4-4817-a4f3-978f44bd2d95"
                },
                {
                  "artifact": {
                    "name": "y18n",
                    "version": "3.2.1",
                    "vulnerabilities": [],
                    "source": "https://registry.npmjs.org/y18n/-/y18n-3.2.1.tgz
↪"
                  },
                  "spec": "y18n@^3.2.1",
                  "id": "fce4863f-db14-4702-849d-0315d324c2e2"
                }
              ],
              "id": "4b200a05-f514-40fc-94b5-12ec5dbe5985",
              "started": "2018-08-24T12:08:02.852095"
            }
          ],
```

(continues on next page)

(continued from previous page)

```
            "commit": "a5a01ca69ac99c793ec5af1bbc190f518d8fc412"
          }
        ]
      }
    }
```

Query Parameters

- **query** – GraphQL query
- **variables** – List of variables to be used within the GraphQL query.

An example request using *curl*.

```
$ curl 'http://localhost:8080/v1alpha1/graphql' \
       -H 'Content-Type: application/json' \
       --data-binary '
         {"query":"
            {
              analysis (where: {id: {_eq: \"904a2117-1da1-4c9c-a3d5-b03262f53d97\"}
↪}){
                state
                installations {
                  spec
                  artifact {
                    name
                    version
                    vulnerabilities {
                      provider
                      reference
                      details
                    }
                  }
                }
              }
            }",
          "variables":null}'
```

HTTP ROUTING TABLE

/analysis

POST /analysis/, 19
POST /analysis/(analysis_id)/(execution_id)/dependencies, 19
POST /analysis/(analysis_id)/(execution_id)/vulnerabilities, 20
PUT /analysis/(analysis_id)/extraction/failed, 19
PUT /analysis/(analysis_id)/extraction/started, 19
PUT /analysis/(analysis_id)/extraction/succeeded, 19

/v1alpha1

GET /v1alpha1/graphql, 20