DejaCode

nexB Inc.

May 10, 2024

GETTING STARTED

Welcome to the very start of your DejaCode journey!

CHAPTER

ONE

INSTALLATION

Welcome to the **DejaCode** installation guide! This guide explains how to install DejaCode on various platforms. Please follow the instructions carefully for a smooth installation experience.

The **recommended DejaCode installation** method is to *Run with Docker*, which is the easiest and ensures all features work with minimal configuration. This installation works on all operating systems.

Alternatively, if you prefer, you can install DejaCode locally as a development server with some limitations. In this case, you may refer to the *Local development installation* section for more details.

1.1 Run with Docker

1.1.1 1. Get Docker

Start by downloading and **installing Docker on your platform**. Refer to Docker's documentation for the best installation path for your system: .

1.1.2 2. Build the image

DejaCode comes with the necessary `Dockerfile` and `docker-compose.yml` files to create the Docker images and to manage the services (database, cache, webserver).

Warning: For **Windows** users, before cloning the repository, ensure that your git `autocrlf` configuration is set to `false`:

```
git config --global core.autocrlf false
```

Clone the DejaCode repository

Open your terminal and clone the DejaCode repo with the following command:

```
git clone --depth=1 https://github.com/nexB/dejacode.git
```

Build the Docker image

Create an **environment file**, and **build the Docker image** with:

```
cd dejacode && make envfile
docker compose build
```

1.1.3 3. Run the app

To **run the DejaCode images as containers**, use the following command:

```
docker compose up -d
```

1.1.4 4. Create an application user

To create a superuser for the application, use the following command:

```
make createsuperuser
```

Follow the prompt instructions, providing the required information:

- **Username**: Choose a unique username.
- **Email Address**: Provide a valid email address.
- **Strong Password**: Create a password following security guidelines.

Use these credentials to access the application.

Congratulations!

Congratulations, you are now ready to use DejaCode.

Open a web browser and visit to **access the web UI**.

You can sign-in with your user credentials generated above.

You can move onto the Tutorials section starting with the *Tutorial 1 - Your first Product*.

Important: DejaCode will utilize all available CPUs according to your Docker configuration, ensuring faster processing.

Make sure to allocate enough memory to support each CPU process.

A good rule of thumb is to allocate **1 GB of memory per CPU**. For example, with Docker configured for 8 CPUs, allocate a minimum of 8 GB of memory.

1.1.5 5. Dataspace setup and AboutCode integrations

Upon the initialization of the DejaCode application, the `nexB` reference *Dataspace* is created with a **default set of data**, including license and organization libraries.

Additionally, **AboutCode integrations are pre-configured** to connect to **public instances** of the following AboutCode applications:

- **ScanCode.io**: Facilitates package scanning. Refer to *Enable package scanning with your ScanCode.io server*.
- **PurlDB**: Provides access to a database of scanned packages. Refer to *Enable PurlDB service*.
- **VulnerableCodeDB**: Enables access to a database containing information on package vulnerabilities. Refer to *Enable VulnerableCodeDB service*.

Warning: In the scenario of **deploying DejaCode as an enterprise service** within your organization, it is **strongly recommended to review these configurations**. Consideration should be given to **running your own instances** of these applications to ensure that **sensitive or private data** is not inadvertently submitted to public services. This strategic approach helps to safeguard organizational data and privacy during package scanning and vulnerability assessments.

1.2 Hardware requirements

The minimum hardware/system requirements for running DejaCode as an enterprise server are:

Item	Minimum
Processor	Modern X86 64 bit Multi Core, with at least **4 physical cores**
Memory	**64 GB** or more (ECC preferred)
Disk	2 * 500GB SDD in RAID mirror setup (enterprise disk preferred)
OS	**Ubuntu 22.04 LTS 64-bit** server clean installation

1.3 Local development installation

Note: This section is designed for those interested in actively contributing to the development and enhancement of DejaCode. After setting up DejaCode, please refer to our Contributing chapter for comprehensive instructions on submitting code changes.

1.3.1 Supported Platforms

DejaCode has been tested and is supported on the following operating systems:

1. **Debian-based** Linux distributions
2. **macOS** 10.14 and up

Warning: On **Windows** DejaCode can **only** be *Run with Docker*.

1.3.2 Pre-installation Checklist

Before you install DejaCode, make sure you have the following prerequisites:

1. **Python: versions 3.12** found at https://www.python.org/downloads/
2. **Git**: most recent release available at https://git-scm.com/
3. **PostgreSQL**: release 16 or later found at https://www.postgresql.org/ or https://postgresapp.com/ on macOS

1.3.3 Clone and Configure

1. Clone the DejaCode GitHub repository:

```
git clone https://github.com/nexB/dejacode.git && cd dejacode
```

2. Install the dependencies:

```
make dev
```

3. Create an environment file:

```
make envfile
```

1.3.4 Database

PostgreSQL is the preferred database backend. To set up the database user, database, and table, run:

```
make postgresdb
```

1.3.5 Tests

You can validate your DejaCode installation by running the test suite:

```
make test
```

1.3.6 Run the App

Start the local web server using:

```
make run
```

Then, open your web browser and visit http://localhost:8000/ to access the web application.

Warning: This setup is **not suitable for deployments** and is **only supported for local development**. It is highly recommended to use the *Run with Docker* setup to ensure the availability of all the features.

CHAPTER

TWO

DATASPACE

The Dataspace serves as a pivotal mechanism within DejaCode, facilitating the segregation of data for each organization while maintaining a unified storage structure in the same database, schema, or table. Within a given installation, multiple "Dataspace" organizations can be defined, but there exists only one reference.

This concept is a crucial element employed across DejaCode to effectively separate **reference data provided by nexB** from the data utilized in a specific DJE installation. Essentially, it introduces the notion of a "tenant" within a DJE installation, enabling the isolation of organization-specific and/or private records. This segregation supports both multi-tenancy and the coexistence of nexB-provided reference data and organization-specific or customized data.

The key purposes of this separation include:

1. **Orderly and Simplified Data Updates**: Facilitates smooth and streamlined updates from nexB reference data, ensuring efficient data synchronization and exchange across Dataspaces.
2. **Dataspace-Specific Customizations**: Allows for customization of Dataspace-specific data, such as configurations for license tags or specific preferences, tailoring the installation to the unique needs of each organization.
3. **Support for Multi-Tenancy**: Enables the sharing of the same DJE instance among different organizations, each operating within its distinct Dataspace, promoting multi-tenancy while maintaining data segregation.

In summary, the Dataspace concept in DejaCode plays a pivotal role in maintaining data integrity, enabling efficient updates, accommodating customization, and supporting multi-tenancy for a diverse range of organizations within a DJE installation.

2.1 Setting up your License Library

After you have created your own Dataspace, you should copy the Reference License Library into it.

To load the License Library data in your Dataspace, use the following command:

```
dejacode clonedataset nexB <YOUR_DATASPACE_NAME> <SUPERUSER_USERNAME> --license_library
```

This `clonedataset` command may take several hours. Once it has completed, you can review the results by signing on to the application as a superuser in your own Dataspace.

Select the **Licenses** option in the navigation header to see the user view of the Licenses.

Select the **Administration > Licenses** option from the dropdown list under your user name to access the **Browse Licenses** administrator form. In this mode, you can select a license to review its details and modify it as appropriate for your organization.

2.2 Managing Users in your Dataspace

As part of your DejaCode installation process you should have created a Superuser in the **nexB Reference Dataspace**; you want to keep that User for DevOps purposes in your ongoing maintenance of DejaCode.

Most of the Users that you need to define should be assigned to your own Dataspace. While you are initially setting up your own Dataspace, you can define those user participants in your project team as Superusers in order to give them access to the various entities that need to be reviewed and defined to meet the requirements of your organization. Please note, as before, that you should also check the Staff status field when you define a Superuser to enable access to the Administrative part of DejaCode.

When you define a User, take note of specific essential fields:

- **Email notification:** Generally you want to leave this field unchecked. When checked, the User will receive an email notification for every update to the database in your dataspace, and it is very unlikely that you will want that.
- **Staff status:** Be sure to check this field for any User that needs to add and/or update the data in your Dataspace.
- **Superuser status:** Check this field to enable the User to perform all system and data administration tasks in DejaCode. This status is especially helpful while you are in Dataspace setup mode to ensure that members of the project team have the DejaCode access that they need. Note that you will generally leave this field unchecked for most of your Users.

2.3 User Permission Groups

The **Permission Groups** are defined to support the most likely roles that your User Community will perform. You can get details about the application tasks available to each one by clicking on the **(permission details)** link.

Generally, the two most important and useful Permission Groups for you to use are the following:

- **Legal:** The Legal Users are primarily responsible for setting policy on your Licenses and Components, and communicating those details to your overall User Community through DejaCode. Be sure to also check **Staff status** when you assign the Legal Permission Group to a User. You may also assign this group to members of your senior management team.
- **Engineering:** The Engineering Users primarily use DejaCode to read Component and License information, including the policies that you have set, and also to create new Components and/or copy them from Reference Data. If you want to restrict such Users from actually performing data updates in DejaCode, simply make sure that **Staff status** is not checked; otherwise, you can give them access to creating and updating the Components that they discover as part of their ongoing software development process.

Note: All Users can see the data in the User Views of the application. You can control the **Tab Visibility** of each application object by Permission Group from your Dataspace definition.

2.4 Defining your Usage Policies

The Reference licenses are intentionally not defined with Usage Policy assignments, since a Usage Policy assignment is specific to your organization.

To get started:

- Navigate to the **Administration** dashboard and select the **Usage policies** option.
- After your initial installation, the results on your **Browse Usage policies** form will probably be empty.
- You can kickstart the process of defining policies by copying Usage Policies defined in Reference Data to your Dataspace.
- Click on the **View Reference Data** button, to view the sample Usage Policies defined in Reference Data.
- You can copy them one by one with the "Copy to my Dataspace" link, or
- Click on the checkbox at the top of the list to select all of the Reference Data Usage Policies, and then select the **Copy the selected objects** option from the dropdown menu at the bottom left of the page and click the **Go** button.
- On the following screens, accept all defaults and click the **Copy** button in the lower right hand corner of the page.

When you return to the **Browse Usage policies** page, you can see the results of your copy action. Note that the **Show all** button resets the view all records from your own Dataspace, and the **View Reference Data** button resets the view to see the originally installed data.

At this point you can decide to work with the sample Usage Policies that you copied, or you can modify them to customize them, create new ones and/or delete those that you do not need.

Note: The techniques that you used to **Copy Reference Data** to your own Dataspace are the same that you can use from most of the **Browse forms** in DejaCode.

2.5 Assigning Usage Policies to licenses

There are more than **2,000 Licenses** in your License list from Reference Data, so you will want to develop a strategy that works for your business requirements to assign Usage Policies efficiently. The two basic techniques are:

1. One at a time: edit a license and assign it a policy from the dropdown field on the **Change License** form.
2. Mass update: Select a group of licenses on the **Browse License** form and use Mass Update to assign a Usage Policy to that group of licenses.

As an example of the first technique, locate and select the license with the key of **apache-2.0** in your list. Your organization probably already has a policy regarding this very common license; you may simply allow engineering to use components under that license or you may require additional review of how they are actually using those components or you may have concerns about specific clauses in that license, depending on your business requirements. Based on those considerations, you should be able to select a **Usage Policy** from the dropdown list on that field. At this point you may also choose to enter text into the **Guidance** field, which is reserved for comments unique to your organization, as well as the **Guidance URL** field which may point to a web page (usually one internal to your organization) that provides additional extended guidance. Both of the Guidance fields are optional, and you can always return to them at a later time. When you have completed your updates, click the **Save** button at the bottom right corner of the page to save your changes.

As an example of the second technique, let us assume that your legal group does not require any review of the usage of components under a **Public Domain license**. You can set a **Usage Policy** for all of those licenses at once:

- Set a filter on the **Browse Licenses** page, and select the **Public Domain** choice under the **Category** filter.
- Use the **Select All** checkbox in the upper left corner of the list to select all the licenses in the **Public Domain** category,
- then select the **Mass update** option from the dropdown list at the bottom of the page, and click the **Go** button.

The application will present a form that shows the field updates that you can apply to all of the selected Licenses.

- Select the **Usage Policy** field using its checkbox, and then select a Usage Policy from the dropdown.
- Click the **Update records** button in the lower right hand of the form to save this Usage Policy assignment.

Note: The techniques that you used to **Mass Update** licenses in your Dataspace are the same that you can use from most of the **Browse forms** in DejaCode.

2.6 Reviewing your Dataspace settings

The presentation of your License Usage Policies and selected license attributes to your user community is controlled by a number of flags in your Dataspace definition.

From the **Administration dashboard**, select **Dataspaces** and open your Dataspace definition.

There are several options grouped in sections such as:

- **Attribution Package Information** used when generating Product Attribution notices,
- **User Interface Settings** to control some aspects of the user interface,
- **Application Process Settings** .

These are initially set to the recommended default settings when you install.

To complete your initial "Usage Policies" configuration, make sure that the **Show usage policy in license library view** option is checked.

If you make any changes, be sure to save them by clicking the **Save** button at the bottom of the form.

To see the results of your **Usage Policy assignments**, click the **Licenses** option at the top of any page to return to the user view of the **License Library**. The icon of any Usage Policy that you assigned to a License will be displayed in its own column on the License list.

Note: You can get **additional information** and help for each field on this form (and any administrative form in DejaCode) by clicking the **Show/Hide help** button at the top of the page.

2.7 Using your Component Catalog

After you have setup the License Library in your own Dataspace, and have defined your Usage Policies, you are ready to start working with **Components** and **Packages**. There are multiple ways to discover and copy the components that interest you from Reference Data; here are a few ways to do that:

1. Search for a specific component or package in the User View of the Reference Data, select the one you want, and copy it to your Dataspace.

2. Search and/or filter Reference Data components or packages using the Administrator's Browse Components or Browse Packages pages, select the ones you need, and Copy the selected entries to your Dataspace, either one by one or many at once.

As an example of the first technique, click on the **Components** option to see the User View of the components in your Dataspace. You can use this view to search your own Dataspace, or you can click on the **View Reference Data** button to search for new components that you need.

For example, if you enter `aboutcode` in the search field near the top of the form, you will see at least two versions of the component "AboutCode toolkit" in Reference Data. Click on the `+` sign to expand the list to see all the versions. You can open version `3.0.2` to see if it is the component that you want.

Simply click the **Copy to my Dataspace** button, and on the following screens, accept all defaults and click the **Make the Copy** button in the lower right hand corner of the form.

To review and possibly edit the copied component, click on its name on the page presented by the application, which will take you to the **Change Component form**. You can scroll down to see the Usage Policy field on that component, and if you accepted and/or checked the Dataspace option to **Set usage policy on new component from licenses** the Usage Policy will already be assigned. In our AboutCode toolkit example, this is based on your Usage Policy on the `Apache 2.0 (apache-2.0)` license.

You can review and optionally modify any of the fields on the component and Save your changes. The copied component now appears in the Components User View, ready for your user community to see.

As an example of the second technique, go to the **Browse Components** form in the Administrator side of the application, and click the **View Reference Data** button. Enter the value `name^angular` in the search field (which means: find all components with a name that begins with "angular") and press Return.

DejaCode will show you a list of various components that meet your search criteria. Identify the ones that you want and check the selection boxes. From the dropdown list in the lower right corner of the form, select **Copy the selected objects** and click the Go button.

On the next page, you may see a message if any of the selected components already exist in your own Dataspace; optionally, you can check any of those to update those components from Reference Data. Click the **Make the Copy and Update** button to continue, and DejaCode shows you the results of your action on the next page.

Note: The techniques that you used to **Copy Reference Data** to your own Dataspace are the same that you can use from most of the **Browse forms** in DejaCode.

2.8 Assigning Usage Policies to Components

As you add new components to your Dataspace, you will want to develop a strategy that works for you business requirements to assign Usage Policies efficiently.

The basic techniques to use in your own Dataspace are:

1. Edit a component and assign it a policy from the dropdown field on the **Change Component** form.
2. Select a group of components on the **Browse License** form and use **Mass Update** to assign a policy to that group of components.
3. Select a group of components on the **Browse License** form and use the **Set usage policy from licenses** option in the dropdown list in the lower right hand corner of the form and follow the prompts to complete that action.

2.9 Enable package scanning with your ScanCode.io server

DejaCode integration with a ScanCode.io server enables you to take advantage of the detailed Package metadata that ScanCode can provide for publicly available software.

You can:

- Simply provide a Download URL for the Package to initiate Package creation, data collection, and scanning in DejaCode.
- Initiate scanning on an existing Package in your DejaCode database.
- View formatted scan results on the Scan tab of the DejaCode Package user view.
- Move specific results returned from a scan to your Package definition.
- Download the scan results to a JSON-formatted file to integrate with other analysis and reporting tools.

2.9.1 Install and configure ScanCode.io

Warning: If you plan to run ScanCode.io **on the same server** (virtual or physical) as the DejaCode instance, **ensure that the host machine has sufficient resources** to handle both applications.

Also, you will have to **provide custom network ports for the ScanCode.io** application as the ports 80 and 443 will be already used by the DejaCode application.

See https://scancodeio.readthedocs.io/en/latest/installation.html#use-alternative-http-ports Finally, it's essential to specify the following **ScanCode.io URL** instead of using `http://localhost:[port]`:

- On **macOS and Windows**: `http://host.docker.internal:[port]`
- On **Linux**: `http://172.17.0.1:[port]`

1. Install a ScanCode.io server following instructions at https://scancodeio.readthedocs.io/en/latest/installation.html

   For production use, the **minimum system requirements for ScanCode.io** are:

Item	Minimum
Processor	Modern X86 64 bit Multi Core, with at least **8 physical cores**
Memory	**64GB** or more (ECC preferred)
Disk	**2x500GB SDD** in RAID mirror setup (enterprise disk preferred).

2. Set the ScanCode.io Server URL in your Dataspace Configuration:

- Access your DejaCode web application **Administration dashboard**.
- Navigate to the **Dataspaces** section and select your Dataspace name.
- Within the **Application Process Settings** section, enable the **Enable package scanning** option.
- Update the values for the **ScanCode.io URL** field located in the **Configuration** panel at the bottom of the form.

Warning: If running ScanCode.io on the same docker host as DejaCode, you will have to use `http://host.docker.internal:[port]` as the **ScanCode.io URL**.

- Click the **Save** button.

You can now access the **Scans** section from the **Tools** menu and initiate package scans from this view.

Tip: To validate the setup of the integration, navigate to the top right menu dropdown in the DejaCode header, and select **Integrations Status**.

2.9.2 Secure ScanCode.io enforcing Authentication

In the context of deploying ScanCode.io as an enterprise solution, enforcing **authentication** measures is **critical**. This section provides guidance on **activating ScanCode.io authentication** and **generating an API key**. This key is essential for enabling **secure interactions** between DejaCode and ScanCode.io.

1. Follow the instructions at Scancode.io Application Settings to enable the ScanCode.io authentication system.
2. After making the settings change, restart the ScanCode.io services using the following commands:

```
docker compose restart web worker
```

3. Generate an API key for authentication by your DejaCode instance. Refer to CLI Create User for detailed instructions.
4. Set the ScanCode.io API key in your Dataspace Configuration:

- Access your DejaCode web application **Administration dashboard**.
- Navigate to the **Dataspaces** section and select your Dataspace name.
- Update the value for the **ScanCode.io API key** field located in the **Configuration** panel at the bottom of the form.
- Click the **Save** button.

2.10 Enable PurlDB service

DejaCode integration with the **PurlDB** service enables user access to the PurlDB option from the Tools menu, which presents a list of PurlDB data mined and scanned automatically from multiple public sources. Users can view PurlDB details and can create DejaCode Package definitions using those details, and DejaCode also presents a new PurlDB tab when viewing the details of a Package with matching key values. This integration also enhances the **Global Search** feature to extend the search scope beyond the standard DejaCode objects (Packages, Components, Licenses, Owners) and perform an asynchronous query of the PurlDB to find relevant data.

You can:

- Browse and search from a list of over **21 millions Packages**.
- Get extra information on your local Packages from the **"PurlDB" tab**.
- **Create local Packages automatically** from entries found in the PurlDB.
- Enhance the **Global search** results with Packages from the PurlDB.
- Check for **new Package versions** from your Products inventory

2.10.1 PurlDB service

A public instance of **PurlDB** is available at https://public.purldb.io/api/packages/

Alternatively, you have the option to run your instance of PurlDB by following the documentation provided at https://purldb.readthedocs.io/

2.10.2 Set the PurlDB Server URL and API key in your Dataspace Configuration

- Access your DejaCode web application **Administration dashboard**.
- Navigate to the **Dataspaces** section and select your Dataspace name.
- Within the **Application Process Settings** section, enable the **Enable PurlDB access** option.
- Update the values for the **PurlDB URL** and **PurlDB API key** fields located in the **Configuration** panel at the bottom of the form.
- Click the **Save** button.

You can now access the **PurlDB** section from the **Tools** menu and browse package from this view.

2.11 Enable VulnerableCodeDB service

DejaCode integration with the **VulnerableCodeDB** service authorizes DejaCode to access the VulnerableCodeDB using a Package URL (purl) to determine if there are any **reported vulnerabilities for a specific Package** and return the Vulnerability ID and related URLs to a **Vulnerabilities tab** in the **Package details** user view.

DejaCode displays a Vulnerability icon next to the Package identifier in the user view list, and also in any Product Inventory list using that Package.

Users can view the VulnerableCodeDB details of an affected Package and use the links to access publicly available reports (e.g. CVE, CPE, GHSA, DSA), discussions, and status updates regarding the vulnerabilities.

You can:

- Explore the Vulnerabilities that affect a Package.
- Review and edit your Product Package assignments to record your analysis, the actions you have taken, and the current status of your usage of that Package.

2.11.1 VulnerableCodeDB service

A public instance of **VulnerableCodeDB** is available at https://public.vulnerablecode.io/api/

Alternatively, you have the option to run your instance of VulnerableCodeDB by following the documentation provided at https://vulnerablecode.readthedocs.io/

2.11.2 Set the VulnerableCodeDB Server URL and API key in your Dataspace Configuration

- Access your DejaCode web application **Administration dashboard**.
- Navigate to the **Dataspaces** section and select your Dataspace name.
- Within the **Application Process Settings** section, enable the **Enable VulnerableCodeDB access** option.
- Update the values for the **VulnerableCode URL** and **VulnerableCode API key** fields located in the **Configuration** panel at the bottom of the form.
- Click the **Save** button.

You can now see Vulnerabilities in the Packages user view. The availability of the services can be checked by clicking on your user name in the top right corner of the app, then "Status > Integrations Status".

CHAPTER

THREE

APPLICATION SETTINGS

3.1 Settings file

DejaCode is configured with environment variables stored in a `.env` file.

The `.env` file is created at the root of the DejaCode codebase during its installation. You can configure your preferences using the following settings in the `.env` file.

Note: DejaCode is based on the Django web framework and its settings system. The list of settings available in Django is documented at Django Settings.

Tip: Settings specific to DejaCode are all prefixed with `DEJACODE_`.

Restarting the services is required following any changes to .env:

```
docker compose restart web worker
```

3.1.1 DATABASE

The database can be configured using the following settings:

```
DEJACODE_DB_HOST=localhost
DEJACODE_DB_NAME=dejacode_db
DEJACODE_DB_USER=user
DEJACODE_DB_PASSWORD=password
DEJACODE_DB_PORT=5432
```

3.1.2 ALLOWED_HOSTS

A list of strings representing the host/domain names that this application can serve.

To enable this setting you need to have a proper host and domain name configured for your DejaCode installation.

This setting is a security measure to prevent an attacker from poisoning caches and password reset emails with links to malicious hosts by submitting requests with a fake HTTP Host header, which is possible even under many seemingly-safe webserver configurations.

Values in this list can be fully qualified names (e.g. 'www.example.com'), in which case they will be matched against the request's Host header exactly (case-insensitive, not including port).

A value beginning with a period can be used as a subdomain wildcard: '.example.com' will match example.com, www.example.com, and any other subdomain of example.com. A value of '*' will match anything; in this case you are responsible to provide your own validation of the Host header.

```
ALLOWED_HOSTS=*
```

3.1.3 EMAIL

This settings enables the email notification feature in DejaCode. If set, the provided username, password and email/SMTP server details are used to send email notifications to your DejaCode users.

```
# The SMTP user used for authentication on your SMTP server.
EMAIL_HOST_USER=''
# Password to use for the SMTP server defined in EMAIL_HOST.
# Can be empty on non-secured, test servers.
EMAIL_HOST_PASSWORD=''
# The SMTP server host to use to send emails.
EMAIL_HOST=''
# Port to use for the SMTP server defined in EMAIL_HOST.
EMAIL_PORT=587
# Default "FROM" email address to use when sending email notifications
DEFAULT_FROM_EMAIL=''
# Whether to use a TLS (secure) connection when talking to the SMTP server
# You should always use a secure connection.
EMAIL_USE_TLS=True
```

3.1.4 SITE_URL

The base URL of this DejaCode installation. This setting is required to build URLs that reference objects in the application. It is also used when including URLs in email notifications.

```
SITE_URL=http://www.yourdomain.com/
```

3.1.5 DEJACODE_SUPPORT_EMAIL

An optional email address to reach the support team of this instance. When defined, it will be displayed in various views and emails related to account registration, activation, and password reset.

```
DEJACODE_SUPPORT_EMAIL=support@dejacode.com
```

3.1.6 ANONYMOUS_USERS_DATASPACE

One Dataspace can be designed as accessible to anyone in a view-only mode. Set this with an existing Dataspace name to enable view-only access to anonymous, no logged-in users.

```
ANONYMOUS_USERS_DATASPACE=DATASPACE_NAME
```

3.1.7 REFERENCE_DATASPACE

An administrative User in the Reference Dataspace can see and copy data from every Dataspace; otherwise, the User can only see data from his/her assigned Dataspace and copy from the Reference Dataspace. An administrative User in the Reference Dataspace can also maintain User definitions for all Dataspaces.

The default Reference Dataspace is always 'nexB' unless the following setting is set to another existing Dataspace. If set to an empty value or a non-existent Dataspace, 'nexB' will be considered the Reference Dataspace.

Caution: be careful when changing this setting as you may no longer have access to nexB-provided reference data.

```
REFERENCE_DATASPACE=nexB
```

3.1.8 SESSION

You can control whether the DejaCode session framework uses web browser-lifetime sessions vs. persistent sessions with the `SESSION_EXPIRE_AT_BROWSER_CLOSE` setting. If `SESSION_EXPIRE_AT_BROWSER_CLOSE` is set to True, DejaCode cookies will expire as soon as a user closes his or her web browser. Use this if you want the user to have to log-in every time they open a browser.

```
SESSION_EXPIRE_AT_BROWSER_CLOSE=True
```

The `SESSION_COOKIE_AGE` setting is the maximum age of DejaCode session cookies, in seconds. The DejaCode user session will expire if the user is "inactive" in the application for longer than this value.

```
# 1 hour, in seconds.
SESSION_COOKIE_AGE=3600
```

3.1.9 DEJACODE_LOG_LEVEL

By default, only a minimum of logging messages is displayed in the console, mostly to provide some progress about pipeline run execution.

Default: `INFO`

The `DEBUG` value can be provided to this setting to see all DejaCode debug messages to help track down configuration issues for example. This mode can be enabled globally through the `.env` file:

```
DEJACODE_LOG_LEVEL=DEBUG
```

3.1.10 CLAMD_ENABLED

When enabled, DejaCode will perform virus scanning on any and all files that a user attempts to import in the various places where data imports are supported. A file with a detected virus will be blocked from upload, and DejaCode will present a pertinent error message to the user when this occurs.

To enable anti-virus scan on file upload, set the `CLAMD_ENABLED` setting to True.

```
CLAMD_ENABLED=True
```

3.1.11 TIME_ZONE

A string representing the time zone for the current ScanCode.io installation. By default the `US/Pacific` time zone is used:

```
TIME_ZONE=US/Pacific
```

Note: You can view a detailed list of time zones here.

3.2 AboutCode integrations

To **integrate DejaCode with other applications within the AboutCode stack**, you have the flexibility to configure and set up integrations using the following application settings.

It's important to understand that employing application settings will make these integrations **globally accessible across all Dataspaces** within your DejaCode instance.

Alternatively, if you wish to tailor the availability of these features to a specific Dataspace, you can define and set those values directly within the *Dataspace* configuration. This can be done through the Dataspace admin UI, allowing you to scope the availability of these integrations exclusively to the designated Dataspace.

3.2.1 SCANCODEIO

Provide the URL and API key of your ScanCode.io instance.

```
SCANCODEIO_URL=https://your_scancodeio.url/
SCANCODEIO_API_KEY=insert_your_api_key_here
```

Note: You have the option to define and set those settings directly on your Dataspace. For detailed instructions, refer to *Enable package scanning with your ScanCode.io server*.

3.2.2 PURLDB

Provide the URL and API key of your PurlDB instance.

```
PURLDB_URL=https://your-purldb.url/
PURLDB_API_KEY=insert_your_api_key_here
```

Note: You have the option to define and set those settings directly on your Dataspace. For detailed instructions, refer to *Enable PurlDB service*.

3.2.3 VULNERABLECODE

You can either run your own instance of VulnerableCode or connect to the public one https://public.vulnerablecode.io/.

Note: Providing an API key is optional when using the public VulnerableCode instance.

```
VULNERABLECODE_URL=https://public.vulnerablecode.io/
VULNERABLECODE_API_KEY=insert_your_api_key_here
```

Note: You have the option to define and set those settings directly on your Dataspace. For detailed instructions, refer to *Enable VulnerableCodeDB service*.

3.3 LDAP Integration

3.3.1 AUTHENTICATION_BACKEND

This setting enables users to authenticate against an LDAP server.

To enable the LDAP authentication, set the following value for the `AUTHENTICATION_BACKENDS` setting.

```
AUTHENTICATION_BACKENDS=dje.ldap_backend.DejaCodeLDAPBackend
```

An alternative setup is to allow the authentication in the system first using LDAP, and then using a DejaCode user account if the authentication through LDAP was not successful. For example, this can be useful if the LDAP server is down.

```
AUTHENTICATION_BACKENDS=dje.ldap_backend.DejaCodeLDAPBackend,django.contrib.auth.
↪backends.ModelBackend
```

3.3.2 SERVER_URI

The URI of the LDAP server.

```
AUTH_LDAP_SERVER_URI=ldap://ldap.server.com:389
```

3.3.3 START_TLS

By default, LDAP connections are unencrypted. If you need a secure connection to the LDAP server, you can either use an `ldaps://` URI or enable the StartTLS extension.

To enable StartTLS, set `AUTH_LDAP_START_TLS` to True.

```
AUTH_LDAP_START_TLS=True
```

3.3.4 BIND

`AUTH_LDAP_BIND_DN` and `AUTH_LDAP_BIND_PASSWORD` should be set with the distinguished name, and password to use when binding to the LDAP server.

Note: Use empty strings (the default) for an anonymous bind.

```
AUTH_LDAP_BIND_DN=""
AUTH_LDAP_BIND_PASSWORD=""
```

3.3.5 USER_DN

The following setting is required to locate a user in the LDAP directory. The filter parameter should contain the placeholder %(user)s for the username. It must return exactly one result for authentication to succeed.

```
AUTH_LDAP_USER_DN="ou=users,dc=example,dc=com"
AUTH_LDAP_USER_FILTERSTR="(uid=%(user)s)"
```

3.3.6 AUTOCREATE_USER

When `AUTH_LDAP_AUTOCREATE_USER` is True (default), a new DejaCode user will be created in the database with the minimum permission (a read-only user).

Enabling this setting also requires a valid dataspace name for the `AUTH_LDAP_DATASPACE` setting. New DejaCode users created on the first LDAP authentication will be located in this Dataspace.

```
AUTH_LDAP_AUTOCREATE_USER=True
AUTH_LDAP_DATASPACE=your_dataspace
```

Note: Set `AUTH_LDAP_AUTOCREATE_USER` to False in order to limit authentication to users that already exist in the database only, in which case new users must be manually created by a DejaCode administrator using the application.

```
AUTH_LDAP_AUTOCREATE_USER=False
```

3.3.7 USER_ATTR_MAP

AUTH_LDAP_USER_ATTR_MAP is used to copy LDAP directory information into DejaCode user objects, at creation time (see AUTH_LDAP_AUTOCREATE_USER) or during updates (see AUTH_LDAP_ALWAYS_UPDATE_USER). This dictionary maps DejaCode user fields to (case-insensitive) LDAP attribute names.

```
AUTH_LDAP_USER_ATTR_MAP=first_name=givenName,last_name=sn,email=mail
```

3.3.8 ALWAYS_UPDATE_USER

By default, all mapped user fields will be updated each time the user logs in. To disable this, set AUTH_LDAP_ALWAYS_UPDATE_USER to False.

```
AUTH_LDAP_ALWAYS_UPDATE_USER=False
```

3.3.9 Group permissions

User's LDAP group memberships can be used with the DejaCode group permissions system.

The LDAP groups that a user belongs to will be mapped with existing DejaCode groups using the Group name attribute. The permissions defined for each of the mapped DejaCode groups will be loaded for the LDAP user.

To enable and configure DejaCode to use LDAP groups you need to enable LDAP as explained above and also do these additional tasks:

- In the reference nexB Dataspace, create the DejaCode groups and associated permissions through the DejaCode admin interface. From the Admin dashboard: Administration > Groups.
- Configure DejaCode settings to enable LDAP groups retrieval by adding these lines to your DejaCode settings file. Set the proper AUTH_LDAP_GROUP_SEARCH values matching for your LDAP configuration.

```
AUTH_LDAP_FIND_GROUP_PERMS=True
AUTH_LDAP_GROUP_DN="ou=groups,dc=example,dc=com"
AUTH_LDAP_GROUP_FILTERSTR="(objectClass=groupOfNames)"
```

3.4 Configuration examples

3.4.1 Configuration 1

- LDAP as the only way to log-in DejaCode.
- Unencrypted connections with the LDAP server.
- Anonymous bind to the LDAP server.
- Users need to be manually created in DejaCode by an administrator first.
- No mapping for users attributes is defined
- Users field values in the database are not updated at authentication time.

- Users are located using the `uid` attribute with the `ou=users,dc=example,dc=com` distinguished name.

```
AUTHENTICATION_BACKENDS=dje.ldap_backend.DejaCodeLDAPBackend
AUTH_LDAP_SERVER_URI=ldap://ldap.server.com:389
AUTH_LDAP_USER_DN="ou=users,dc=example,dc=com"
AUTH_LDAP_USER_FILTERSTR="(uid=%(user)s)"
AUTH_LDAP_AUTOCREATE_USER=False
AUTH_LDAP_ALWAYS_UPDATE_USER=False
```

3.4.2 Configuration 2

- LDAP as the first way to log-in, and then using a DejaCode user account if the authentication through LDAP was not successful.
- Encrypted connections with the LDAP server.
- Binding to the LDAP server using `cn=admin,ou=users,dc=example,dc=com` for the distinguished name and `pw` the password.
- Users are located using the `cn` attribute with the `ou=users,dc=example,dc=com` distinguished name.
- Users will be automatically created or updated. New users will be located in the "nexB" dataspace.
- Users attributes will be mapped according to the `AUTH_LDAP_USER_ATTR_MAP` values.

```
AUTHENTICATION_BACKENDSdje.ldap_backend.DejaCodeLDAPBackend,django.contrib.auth.backends.
↪ModelBackend
AUTH_LDAP_SERVER_URI=ldaps://ldap.server.com:636
AUTH_LDAP_BIND_DN=cn=admin,ou=users,dc=example,dc=com
AUTH_LDAP_BIND_PASSWORD=pw
AUTH_LDAP_USER_DN="ou=users,dc=example,dc=com"
AUTH_LDAP_USER_FILTERSTR="(cn=%(user)s)"
AUTH_LDAP_AUTOCREATE_USER=True
AUTH_LDAP_DATASPACE=nexB
AUTH_LDAP_ALWAYS_UPDATE_USER=True
AUTH_LDAP_USER_ATTR_MAP=first_name=givenName,last_name=sn,email=mail
```

CHAPTER

FOUR

FAQS

You can't find what you're looking for? Below you'll find answers to a few of our frequently asked questions.

4.1 How can I download SBOM for my products?

CycloneDX SBOM and SPDX document can be downloaded from the "Share" menu of the product details view in the web UI or from dedicated endpoint URLs of the REST API.

Refer to *How To 3 - Downloading SBOM for Your Products* for more details.

CHAPTER

FIVE

TUTORIAL 1 - YOUR FIRST PRODUCT

Sign into DejaCode.

5.1 Create a Product

1. Select *Products* from the main menu bar.

2. Click the green *Add Product* button. Enter the values that you know, you can refer to *Product model* for details about each fields.

3. Set a **name**, and click the *Add Product* button at the bottom of the form.

Note: You are ready to assign Inventory objects to your Product!

5.2 Load a Software Bill of Materials (SBOM) to your Product

You have the flexibility to employ either your CycloneDX, SPDX, or AboutFile Software Bill of Materials (SBOMs).

Alternatively, you can conveniently download one of the provided examples from the following GitHub repository.

On the Product details page, from the *Scan* dropdown, select *Load Packages from SBOMs*:

- Click the *Choose File* button on the **SBOM file or zip archive** field.
- Select your SBOM (.cdx.json or .spdx.json) and click the *Open* button.
- Check the *Update existing packages with discovered packages data* option.
- Click the *Load Packages* button.

DejaCode presents the *Imports* tab. Refresh your screen from the browser to see the status of your import.

View your import results in the *Inventory tab*.

Note: Continue assigning packages to your Product as required.

5.3 Assign Packages to your Product

From the *Manage* dropdown, select *Packages*:

- Click the *Add Package to Product* button.
- Enter the start of a **package identifier**, for example `diagrams` and select package `diagrams-0.12.0.tar.gz`. DejaCode gets the license `mit` from the package definition.
- Click the *Save* button.

You can see the results by selecting the *Inventory tab*.

Select *Packages* from the main menu bar.

- Locate one or more packages to be used in your Product.
- Use the checkbox on the left to select your package(s).
- Select the `Product` option from the *Add to* dropdown.
- Select your product from the dropdown list.
- Click the *Add to Product* button.

View your results in the *Inventory tab*.

Note: Continue assigning packages to your Product as required.

5.4 Review your progress

Click the *Attribution* button:

- Accept all the default attribution configuration settings.
- Scroll down and click the *Generate Attribution*.
- Explore the attribution document that DejaCode presents to you.
- Save the document to your local file system using your browser File Save command.

Select *Reports* from the *Tools* dropdown:

- Select an appropriate report such as `2-Product Package Analysis`.
- Enter your product Name and Version and click *Rerun Report*.
- Explore the results that DejaCode presents to you.
- Export the report to your local file system using the *Export* button.

5.5 Check for New Versions of your Product Packages

Select *Products* from the main menu bar.

Click the **Product name** of the Product you are defining to open it.

From the *Manage* dropdown, select *Check for new Package versions*: New Package Versions are displayed on the *Inventory* tab. You can click on new versions and add them to DejaCode from the PurlDB.

5.6 Assign Catalog Components to your Product

Select *Products* from the main menu bar.

Click the **Product name** of the Product you are defining to open it.

From the *Manage* dropdown, select *Components*:

- Click the *Add Component to Product* button.
- Enter the start of a **Component**, for example `log` and select a version of component `Apache Log4J`. DejaCode gets the license `apache-2.0` from the component definition.
- Click the *Save* button.

You can see the results by selecting the *Inventory tab*.

Select *Components* from the main menu bar.

- Locate one or more components to be used in your Product.
- Use the checkbox on the left to select your package(s).
- Select the `Product` option from the *Add to* dropdown.
- Select your product from the dropdown list.
- Click the *Add to Product* button.

View your results in the *Inventory tab*.

Note: Continue assigning components to your Product as required.

5.7 Review your impact

Click the *Attribution* button:

- Accept all the default attribution configuration settings.
- Scroll down and click the *Generate Attribution*.
- Explore the attribution document that DejaCode presents to you.
- Save the document to your local file system using your browser File Save command.

Select *Reports* from the *Tools* dropdown:

- Select an appropriate report such as `2-Product Component Analysis`.
- Enter your product Name and Version and click *Rerun Report*.

- Explore the results that DejaCode presents to you.
- Export the report to your local file system using the *Export* button.

5.8 Assign Custom Components to your Product

Select *Products* from the main menu bar.

Click the **Product name** of the Product you are defining to open it.

From the *Manage* dropdown, select *Add custom Component*: Enter the data fields that define your custom Component. * Click the *Save* button. Your results are displayed on the *Inventory tab*.

Click the *Attribution* button:

- Accept all the default attribution configuration settings.
- Scroll down and click the *Generate Attribution*.
- Explore the attribution document that DejaCode presents to you.
- Save the document to your local file system using your browser File Save command.

Select *Reports* from the *Tools* dropdown:

- Select an appropriate report such as `2-Product Custom Component Analysis`.
- Enter your product Name and Version and click *Rerun Report*.
- Explore the results that DejaCode presents to you.
- Export the report to your local file system using the *Export* button.

5.9 Review the Licenses that Impact your Product

Select *Products* from the main menu bar.

Click the **Product name** of the Product you are defining to open it.

From the *Manage* dropdown, select *License Summary*: Your Product Licenses are displayed on the *License summary form*. DejaCode displays the **Usage Policy** and all the **Items** for each **License**. Export the **License summary** by clicking the button *Export as CSV*.

5.10 Assign Everything Else to your Product

Continue refining and reviewing your product.

In *Tutorial 2 - Working with Packages*, we'll explore Packages in greater detail!

CHAPTER

SIX

TUTORIAL 2 - WORKING WITH PACKAGES

Sign into DejaCode.

6.1 Add a Package from a Download URL

Select *Packages* from the main menu bar.

Get the download URL of a publicly available package; for example, select one of the releases at https://github.com/cli/cli/tags and copy its download URL.

Click the green *Add Package* button and paste the download URL that you copied into the text box. Click the green *Add* button.

DejaCode presents the new Package.

- Refresh your browser page to see the *Scan* tab.
- Explore the scan results.
- Select values to apply to the new Package definition, such as one or more licenses, a Copyright statement, and a Primary langauge.
- Click the *Set values to Package* button.
- Review, and optionally modify, the values in the modal dialog.
- Click the *Set values* button.
- Review the updated Package definition.
- Click the *Edit icon* (the pencil) near the top of the form.
- Make suitable changes to the Package definition; for example, enter text in the **Description** field.
- Click the *Update Package* button.
- Review the updated Package definition.

6.2 Create a Package using the data entry form

Select *Packages* from the main menu bar.

Select the *Add Package form* option from the green *Add Package* dropdown. Enter the values that you know, you can refer to *Package model* for details about each fields.

Note: A Filename or a Package URL (type + name) is required. The other fields are optional.

Keep the checked default to "Automatically collect the SHA1, MD5, and Size using the Download URL and apply them to the package definition."

Click the *Add Package* button.

Note: DejaCode validates your entry, creates the package, applies automatic updates, and submits the a scan request to ScanCode.io if you have that enabled.

Review and edit your new package.

6.3 Import a Package to DejaCode from the PurlDB

Select the *PurlDB* option from the main menu bar *Tools* dropdown.

Use the *Filters* button to enter filtering or sorting criteria. For example, select `Release date (descending)` to see recent data. Click on the **Identifier** of an interesting package.

Review the data defined for the package. Click the green *Create Package* button.

Review and edit the new DejaCode Package definition. Click the *Add Package* button. DejaCode validates the entry, creates the package, and applies automatic updates.

Review and edit your new package.

6.4 Import Package Definitions from a CSV

Select *Packages* from the main menu bar.

Select the *Import packages* option from the green *Add Package* dropdown.

Click the *Download immport template* button. DejaCode uses your browser to download a file named `pakcage_import_template.csv`. Open that file in a spreadsheet editor (such as Excel) and save it with a meaningful name that describes the data you intend to import.

Enter the values for one or more packages into the CSV. You can get additional help for each field by clicking the *Show/hide Supported Columns* option on the Import form in DejaCode. Save your package import CSV.

Use the *Browse...* button near the bottom of the Import form to select your CSV and click the blue *Upload* button to import the data to DejaCode.

DejaCode validates your data. Review the results and correct your CSV as needed. Click the blue *Import* button to create the new packages in DejaCode.

Review the results of the Import presented by DejaCode. Optionally edit and update the new package(s). Optionally add the new package(s) to a Product using the *Add to Product* button.

6.5 Improve Package Data by Scanning

Select *Packages* from the main menu bar.

Identify and select a Package that needs to be improved. Click the *Scan* button on the Package details form.

Optionally follow the progress of the Scan by selecting the *Scans* option from the *Tools* dropdown on the main menu bar.

Open or refresh the Package form when the Scan is completed. Review the results on the Scan tab, select data to apply to the Package definition, modify that data as needed, and click the *Set values* button to save the updates.

Optionally click the *Download Scan data* button at the bottom of the *Scan* tab to export a JSON-formatted file with the detailed scan results. You can view that file in a readable format using a browser such as Firefox.

Continue refining and reviewing your packages.

In *Tutorial 3 - Working with Reports*, we'll go further!

CHAPTER

SEVEN

TUTORIAL 3 - WORKING WITH REPORTS

Sign into DejaCode.

7.1 Create a Reporting Query

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Queries* option in the *Reporting* panel.
- Click the *Add query* button in the upper right section of the form.
- Enter a **Name**. For this example, enter `Quarterly License Activity`.
- Enter a **Description**. For this example, enter `Licenses Created in the 'past_90_days' OR Licenses Modified in the 'past_90_days'.`
- Select an **Object type**. For this example, choose `license_library | license`.
- Select an **Operator**. For this example, choose `or`.
- Click the *Add another filter* option in the *Filters* panel.

First Filter:

- Select `created_date` for the **Field Name**.
- Select `Greater than or equal to` for the **Lookup**.
- Enter `past_90_days` for the **Value**.
- Check **Runtime Parameter** to enable it.

Second Filter:

- Select `last_modified_date` for the **Field Name**.
- Select `Greater than or equal to` for the **Lookup**.
- Enter `past_90_days` for the **Value**.
- Check **Runtime Parameter** to enable it.

Note: In addition to specific date values, DejaCode recognizes special values for date field lookups: `today`, `past_7_days`, `past_30_days`, `past_90_days`

Click the *Add another order field* option in the *Order Fields* panel.

First Order Field:

- Select `last_modified_date` for the **Field Name**.
- Select `descending` for **Sort**.

Second Order Field:

- Select `key` for the **Field Name**.
- Select `ascending` for **Sort**.

Click the *Save and continue editing* button in the lower right section of the form.

- Click the blue link in the message `See the nnn licenses in changelist`.
- DejaCode presents the selected data on the `Browse Licenses` form.
- Select any license to access the `Change license` form.
- Click the *History* button in the upper right section of the form.
- Review the license history.
- Click the License Name in the upper part of the form to return to the main License form.
- Click the *Return to list* button in the lower left section of the form.
- On the `Browse Licenses` form, click the *Show all* button in the upper left.
- Click the *Filter* dropdown in the upper right and click the `All` value in the `Reporting query` field, and select any Query, noting that the Query you just created is in the list of Queries. Review the results.

7.2 Create a Column Template

- Select the *Administration* in the upper right section of the form.
- Select the *Column templates* option in the *Reporting* panel.
- Click the *Add column template* button in the upper right section of the form.
- Enter a **Name**. For this example, enter `Quarterly License Activity`.
- Enter a **Description**. For this example, enter `License fields that pertain to analysis of recently created or modified licenses.`
- Select an **Object type**. For this example, choose `license_library | license`.
- Click the *Add another column template assigned field* option in the *Column Template Assigned Fields* panel.

First Column Template Assigned Field:

- Select `key` for the **Field Name**.
- Enter `License Key` for the **Display name**.

Add additional Assigned Fields as follows:

- `short_name`: `License Short Name`
- `category > label`: `Category`
- `last_modified_by > username`: `Modified by`
- `last_modified_date`: `Date modified`
- `where_used`: `Where used`
- any additional fields that interest you.

- Click the *Save and continue editing* button in the lower right section of the form.
- You can optionally change the order of the fields using the `Move item` icon. in the right hand section of each Assigned Field.
- Click the *Save* button in the lower right section of the form.

Note: You are now ready to use your Column Template in a DejaCode Report.

7.3 Create a DejaCode Report

- Select the *Administration* in the upper right section of the form.
- Select the *Reports* option in the *Reporting* panel.
- Click the *Add report* button in the upper right section of the form.
- Enter a **Name**. For this example, enter `Quarterly License Activity`.
- Enter a **Description**. For this example, enter `Licenses Created in the 'past_90_days' OR Licenses Modified in the 'past_90_days'.`
- Select `Quarterly License Activity` from the **Query** dropdown.
- Select `Quarterly License Activity` from the **Column template** dropdown.
- Check **User available** to enable it.
- Accept the defaulted text in the **Report context** field.
- Click the *Save and continue editing* button in the lower right section of the form.
- Click the *View* button in the upper right section of the form.

Review the Report results:

- Click the link icon to the left of a License Key to view that License.
- Export the Report results to an `xlsx` formatted file.
- Modify the value of any Field Parameter and click the **Rerun Report** button.
- Experiment with various Export formats and Field Parameter values.

Select the *Reports* option from the main menu bar *Tools* dropdown.

Select other Reports to run and review.

7.4 Manage Your Report Collection

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Reports* option in the *Reporting* panel.
- In the search field on the right, enter `name:activity` and press Return.
- Use the checkbox in the first column to select one or more reports, including your new report.
- Select `Mass update` from the dropdown in the lower left section of the form and click the **Go** button.
- Check **Update** on the `Group` row.

- Enter `Activity` in the **New value** field.
- Click the *Update records* button in the lower right section of the form.
- Review the results of your updates on the `Browse Reports` form.
- Select the *Reports* option from the main menu bar *Tools* dropdown.

Note: The selected reports are now grouped together under your `Group` label.

You can return to the `Browse Reports` form at any time to review and update the `Group` assignments to meet your requirements.

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Reports* option in the *Reporting* panel.
- On the `Browse Reports` form, click the *View Reference Data* button in the upper left section of the form.
- Use the checkbox in the first column to select one or more reports that interest you.
- Select `Copy the selected objects` from the dropdown in the lower left section of the form and click the *Go* button.
- Follow the prompts on the following forms to complete your Copy action.
- Review and edit the copied reports in your own Dataspace.

Continue refining and reviewing your reports.

In Tutorial 4, we'll go further!

CHAPTER

EIGHT

HOW TO 1 - MANAGE YOUR USAGE POLICIES

Sign into DejaCode.

Note: Sign in as a User with full administrative permissions.

8.1 Review and Maintain your Usage Policies

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Usage Policies* option in the *Policy* panel.
- Review the Usage Policies currently defined for your dataspace.
- Optionally click the *View Reference Data* button in the upper left section of the form. Review the Reference Data Usage Policies. Optionally, you can click the *Copy to my Dataspace* button to the left of any entry in order to add or update a Usage Policy to your own Dataspace from Reference Data. You can click the *View My Data* button to return to your own Usage Policies.

Examine the Details:

- Open a Usage Policy or add a new one.
- The **Field Name** should concisely express a Usage Policy defined by your organization.
- The **Object type** identifies the kind of object governed by the Usage Policy. Consider, for example, that your Usage Policy list may vary somewhat for Licenses as opposed to specific Components.
- The **Guidelines** are intended to explain your organization's definition of a Usage Policy and can also provide detailed requirements and instructions for compliance.
- The **Icon** associated with a Usage Policy should be one from the available icons at https://fontawesome.com/icons?d=gallery&m=free
- The **Color** should be a valid HTML color code (e.g. #FFFFFF) to apply to your icon.
- The **Compliance Alert** indicates how the usage of a DejaCode object (license, component, package, etc.) complies with your organization's policies. Value choices are: "Pass" (or empty, the default value), "Warning" (should be reviewed), and "Error" (fails compliance policy guidelines).
- The **Associated product relation status** enables you to specify the product relation status to use automatically when a component or package with an assigned usage policy is added to a product, overriding the general default defined in the product relation status table, which supports the list of values that you can select. By defining this association, you can save a lot of time and effort when you are reviewing a product inventory by concentrating your attention on the exceptions.

- In the *Associated Policies* section, you can define a default value for DejaCode to apply automatically on an associated object. This is especially pertinent for automatically applying a Usage Policy to a Component or Package when you assign a License to one of those objects.

Click the *Save* button in the lower right section of the form. Review your progress in the Usage Policies list.

8.2 Assign your Usage Policies to Licenses

- Select the *Licenses* option from the dropdown beneath your User name.

Filter Licenses as Needed

- Use the **Filter** dropdown in the upper right to restrict the amount of data that you process to a manageable list. For example, you can:
- Filter to see all Licenses where **Usage Policy** is empty.
- Filter to see Licenses in a **Category** , **License profile** , or **License style** .
- Select a **Reporting query** to perform more complex filtering.

Perform Mass Updates to Set License Usage Policies

- Use the checkboxes on the left side of the form to select Licenses for update.
- Select the **Mass update** option from the dropdown in the lower left and click the **Go** button.
- Check the **Usage policy** field and choose the Policy to apply to the selected Licenses.
- Click the *Update records* button in the lower right.

Continue this process to assign Usage Policies to all of your Licenses. You can also assign a Usage Policy to a single License on the Change License form.

8.3 Assign your Usage Policies to Components

- Select the *Components* option from the dropdown beneath your User name.

Filter Components as Needed

- Use the **Filter** dropdown in the upper right to restrict the amount of data that you process to a manageable list. For example, you can:
- Filter to see all Components where **Usage Policy** is empty.
- Select a **Reporting query** to perform more complex filtering.

Set Component Usage Policies from Licenses

- Use the checkboxes on the left side of the form to select Components for update.
- Select the **Set usage policy from licenses** option from the dropdown in the lower left and click the **Go** button.
- Use the checkboxes on the right to select Components to update.
- Click the **Set policies** button in the lower right to apply updates.
- Continue this proess to assign Usage Policies to all of your Components.
- Note that you can also edit any Component to specify a Usage Policy different from its primary License.

Use the same process to set Package Usage Policies from Licenses.

8.4 The Importance of Package and Component Usage Policy Assignments

Note that when you are familiar with the way that product teams actually use a package or component, you may want to set the usage policy on those items to reflect that usage. For example, if you know that a copyleft-licensed item is always used unmodified, or as a library, or only as a non-deployed/non-distributed tool, you can avoid the effort of reviewing each product assignment of that item by setting the usage policy to indicate that it is approved for product usage. A similar logic applies to packages or components with complex license expressions; you can confirm that your usage of that item is only going to execute the code in a certain way and set the item usage policy to reflect that.

8.5 Make Usage Policies Visible to your Users

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Dataspaces* option in the *Administration* panel.
- Select your Dataspace to open it and edit the details.
- In the *User Interface Settings* section, check the **Show usage policy in user views** in order to include the usage policy in user views that show licenses, components or packages.
- Save your work.

8.6 Review Usage Policy Impact

- Open the User View List of Licenses, Components, or Packages to see the Usage Policy Icon associated with objects that have Usage Policy assigned. If you open one of these objects to see the details view, there is also a Usage Policy tab that shows more extensive information about the Policy, including your Guidelines.
- Open a Product and select the Inventory tab. In addition to the Usage Policy Icons, you will also see that Items with a Compliance Alert are highlighted with yellow for a warning and red for an error, as you defined on the associated Usage Policies.
- You can add the Usage Policy Label as a field to your Column Templates in order to see them on your Reports, and to include those values when you export the Report Results to your preferred output file format for distribution to your team.

Continue refining and reviewing your Usage Policies.

8.7 Export License Policy Definitions

You can export a list of your License Keys along with associated Usage Policy details to a YAML-formatted file. This file can be used by other tools such as the open source ScanCode Toolkit (scancode-toolkit).

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Usage Policies* option in the *Policy* panel.
- Click the *Export License Policies as YAML* button in the upper right section of the form.
- View or edit the exported **license_policies.yml** file in your preferred text editor.
- Use the file as an input option to ScanCode Toolkit to enhance the output results.

CHAPTER

NINE

HOW TO 2 - TAKE ADVANTAGE OF REFERENCE DATA

Sign into DejaCode.

Note: Sign in as a User with full administrative permissions.

9.1 Explore and Copy Reference Data

In this example, you are researching ways to get the most out of your DejaCode data by creating Queries and Reports. One way to get started on this is to explore the Queries that are already defined in Reference Data. This is a good way to get better acquainted with methods for taking advantage of the various Query features.

- Select the *Dashboard* option from the dropdown beneath your User name.
- Select the *Queries* option in the *Reporting* panel.
- Review the Queries currently defined for your dataspace.
- Optionally click the *View Reference Data* button in the upper left section of the form. Review the Reference Data Queries. Optionally, you can click the *Copy to my Dataspace* button to the left of any entry in order to add or update a Query to your own Dataspace from Reference Data.
- Alternatively you can click to select multiple Queries on the Browse form, and then you can use the *Copy selected objects* choice in the dropdown in the lower left part of the form. Follow the prompts to copy or update the Reference Queries to your own Dataspace.

  You can follow a similar procedure to copy Column Templates and Reports from Reference Data to your own Dataspace.

9.2 Check for Updates to a Specific Object in Reference Data

In the DejaCode user view of your own Dataspace, when you are viewing the details of an Owner, a License, a Package, or a Component, you can click on a button in the upper part of the form called *Check for Updates* which will prompt DejaCode to present you with a form either telling you that the two objects are exactly the same or that you can select specific fields to apply updated values from Reference Data to your own Dataspace. Simply follow the prompts to make those changes.

9.3 Check for Recently Updated Objects in Reference Data

As and administrative user, you can use the dropdown list in the upper-right corner of DejaCode to select Components, Packages, Licenses or Owners to navigate to the Administrator's Browse forms for those objects. From the Browse form, you can click the *View Reference Data* button in the upper left section of the form to prompt DejaCode to show you the current contents of Reference Data. Note that the objects are by default sorted by Date Modified descending, so you can easily see the most recent updates. You can also search and filter the results to find groups of objects that interest you, and you can follow a procedure similar to the one described above for Queries to copy or update Reference Data objects to your own Dataspace.

9.4 Preserving Specific Field Values in Your Dataspace

When you copy Reference Data that already exists in your own Dataspace, you should note that DejaCode allows you to exclude certain fields when you take that action. An important example where this is important to you is the **Usage Policy** field, since that is a value you set to comply with your own business requirements. By default, DejaCode excludes **Usage Policy** when updating an object to your Dataspace from Reference Data, and you may also select other fields to preserve as well.

CHAPTER

TEN

HOW TO 3 - DOWNLOADING SBOM FOR YOUR PRODUCTS

You can obtain both **CycloneDX and SPDX Software Bill of Materials (SBOM)** documents either through the web user interface (UI) or via the REST API endpoints.

10.1 Web User Interface

1. Navigate to the product details view.
2. Click on the “Share” menu.
3. Download the desired SBOM format from the available options.

10.2 REST API Endpoints

You can programmatically fetch the SBOMs using the following dedicated endpoint URLs of the REST API:

- CycloneDX: `/api/v2/products/{uuid}/cyclonedx_sbom/`
- SPDX: `/api/v2/products/{uuid}/spdx_document/`

Replace `{uuid}` with the unique identifier of your product.

You can also provide your prefered CycloneDX spec version using the `spec_version` query argument such as: `/api/v2/products/{uuid}/cyclonedx_sbom/?spec_version=1.6`

CHAPTER

ELEVEN

MODELS REFERENCE

11.1 DejaCode Modules

DejaCode includes the following modules:

- **Product Portfolio**: Record and maintain software inventories for your products.
- **Component and Packages Catalog**: Identify the origin, licensing terms and relationships of open source and other software components by consulting the catalog. Communicate your company usage policy for components to your users, and provide them with detailed guidance.
- **License Library**: Understand software licensing terms with the nexB library of open source and proprietary licenses. Communicate your company usage policy for licenses to your users, and provide them with detailed guidance.
- **Reporting**: Create your own reports, from your queries and column templates, to explore, analyze and export your DejaCode application data.
- **Workflow Requests**: Create your own request templates to enable your users to submit requests regarding your products, components, licenses and their policies, and to track the progress of each request.
- **API**: Use the DejaCode API to integrate with your other data sources and applications.

11.2 Product model

- **Name** - Your product name. Required field (the only required field).
- **Version** - Your product version. Recommended field. For example, `4.1.3`
- **Owner** - Your owner name. Recommended field. If you have defined an Owner for your organization, enter it here. You can enter `Unspecified` and update it later.
- **License expression** - The license that applies to your product. If you have defined a license for your organization products, enter it here. You can enter `commercial-license` or `proprietary-license` and update it later.
- **Copyright** - Your product copyright statement. Recommended field. For example, `Copyright (c) Starship LLC`
- **Notice text** - Your product notice. For example, `Licensed by Starship LLC`
- **Description** - A concise description of your product.
- **Keywords** - Use the autocomplete feature to enter and select keywords. For example, `Framework, Web Service`
- **Primary language** - Use the autocomplete feature to enter and select a primary language. For example, `Python`

- **Homepage URL** - The URL of the home page for your product. This must be a valid URL.
- **Contact** - Contact name for your product.
- **Active** - Leave this checked (True) for an Active product.
- **Configuration status** - Keep the `New` default value.
- **Release date** - Use the date picker to specify your product release date.

11.3 Package model

- **Filename** - The exact filename of the package.
- **Download URL** - The download URL for obtaining the package.
- **Package URL Type** - A short code to identify the type of this package. For example: gem for a Rubygem, docker for a container, pypi for a Python Wheel or Egg, maven for a Maven Jar, deb for a Debian package, etc.
- **Package URL Namespace** - Package name prefix, such as Maven groupid, Docker image owner, GitHub user or organization, etc.
- **Package URL Name** - Name of the package.
- **Package URL Version** - Version of the package.
- **Package URL Qualifiers** - Extra qualifying data.
- **Package URL Subpath** - Extra subpath within a package, relative to the package root.
- **License expression** - The license expression that applies to the package. You can enter a placeholder such as `other-permissive` and update it later.
- **Copyright** - The package copyright statement. Recommended field. For example, `Copyright (c) Starship LLC`
- **Notice text** - The notice provided by the package authors. For (a very simple) example, `Licensed by Starship LLC under Apache 2.0`
- **Holder** - The name(s) of the copyright holder(s) of a package, as documented in the code
- **Author** - The name(s) of the author(s) of a package as documented in the code.
- **Description** - Free form description, preferably as provided by the author(s).
- **Homepage URL** - The homepage URL of the project responsible for the package. This must be a valid URL.
- **Primary language** - Use the autocomplete feature to enter and select a a primary language. For example, `Python`

CHAPTER

TWELVE

REFERENCE 1 - DECLARED LICENSE EXPRESSION AND LICENSE CLARITY SCORING

When you scan a Package from DejaCode, you can view the Scan Results in a *Scan* tab on the Package details user view. DejaCode presents a selection of scan details with an emphasis on license detection. You can also download the complete *Scan Results* in .json format.

12.1 License Summary Fields

In DejaCode, the Scan tab of the Package details user view shows new license clarity scoring fields and new summary fields.

The new summary fields are:

- *declared_license_expression*
- *declared_holder*
- *primary_language*
- *other_license_expressions*
- *other_holders*
- *other_languages*

You can set the values from *declared_license_expression*, *declared_holder*, and *primary_language* to the package definition in DejaCode.

12.2 Declared License Expression

Declared License Expression is the primary license expression as determined from the declaration(s) of the authors of the package.

Note that the term declared_license_expression is used equivalently for the concept of a primary license expression in order to align with community usage, such as SPDX.

Here is how ScanCode determines the value for a declared license expression, holder and primary language of a package when it scans a codebase:

- Look at the root of a codebase to see if there are any package manifest files that have origin information.
- If there is package data available, collect the license expression, holder, and package language and use that information as the declared license expression, declared holder, and primary language.

- If there are multiple package manifests at the codebase root, then concatenate all of the license expressions and holders together and use those concatenated values to construct the declared license expression and declared holder
- If there is no package data, then collect license and holder information from key files (such as LICENSE, NOTICE, README, COPYING, ADDITIONAL_LICENSE_INFO). Try to find the primary license from the licenses referenced by the key files. If unable to determine a single license that is the primary, then concatenate all of the detected license expressions from key files together and use that as a conjunctive declared license expression. Concatenate all of the detected holders from key files together as the declared holder.

Note that a count of how many times a license identifier occurs in a codebase does NOT necessarily identify a license that appears in the declared (primary) license expression due to the typical inclusion of multiple third-party libraries that may have varying standards for license declaration. It is possible that the declared license expression constructed by this process may not appear literally in the codebase.

12.3 License Clarity Scoring

License Clarity License Clarity is a set of criteria that indicate how clearly, comprehensively and accurately a software project has defined and communicated the licensing that applies to the project software. Note that this is not an indication of the license clarity of any software dependencies.

Score The license clarity score is a value from 0-100 calculated by combining the weighted values determined for each of the scoring elements: Declared license, Identification precision, License texts, Declared copyright, Ambiguous compound licensing, and Conflicting license categories.

Declared license When true, indicates that the software package licensing is documented at top-level or well-known locations (key files) in the software project, typically in a package manifest, NOTICE, LICENSE, COPYING or README file. Scoring Weight = 40.

Identification precision Identification precision indicates how well the license statement(s) of the software identify known licenses that can be designated by precise keys (identifiers) as provided in a publicly available license list, such as the ScanCode LicenseDB, the SPDX license list, the OSI license list, or a URL pointing to a specific license text in a project or organization website, Scoring Weight = 40.

License texts License texts are provided to support the declared license expression in files such as a package manifest, NOTICE, LICENSE, COPYING or README. Scoring Weight = 10.

Declared copyright When true, indicates that the software package copyright is documented at top-level or well-known locations (key files) in the software project, typically in a package manifest, NOTICE, LICENSE, COPYING or README file. Scoring Weight = 10.

Ambiguous compound licensing When true, indicates that the software has a license declaration that makes it difficult to construct a reliable license expression, such as in the case of multiple licenses where the conjunctive versus disjunctive relationship is not well defined. Scoring Weight = -10 (note negative weight).

Conflicting license categories When true, indicates the declared license expression of the software is in the permissive category, but that other potentially conflicting categories, such as copyleft and proprietary, have been detected in lower level code. Scoring Weight = -20 (note negative weight).

Note: Refer to *Tutorial 2 - Working with Packages* for package creation and maintenance procedures.

CHAPTER

THIRTEEN

REFERENCE 2 - UNDERSTAND THE PACKAGE URL (PURL) IN DEJACODE

A Package URL (purl) provides an exact, unique identification of a software object that also includes information about its origin.

The Package URL is an open source community specification defined here: https://github.com/package-url/purl-spec

The Package URL is now a widely accepted standard for identifying software objects, critical to the usefulness of SBOMs (Software Bills of Materials) and for integration with vulnerability tracking services.

13.1 Package URL Elements

Package URL (purl) DejaCode dynamically derives a Package URL (purl) value as a string that combines all of the Package URL elements. The Package URL is displayed in multiple contexts in the DejaCode UI, and is also available as a Column Template field in DejaCode Reports.

Package URL elements are:

- *Type*
- *Namespace*
- *Name*
- *Version*
- *Qualifiers*
- *Subpath*

Type The Package URL Type is a short code to identify the type of this package. For example: gem for a Rubygem, docker for a container, pypi for a Python Wheel or Egg, maven for a Maven Jar, deb for a Debian package, etc.

Namespace The Package URL Namespace is a Package name prefix, such as Maven groupid, Docker image owner, GitHub user or organization, etc.

Name The Package URL Name is the Name of the package.

Version The Package URL Version is the Version of the package.

Qualifiers Package URL Qualifiers provide extra qualifying data for a package such as the name of an OS, architecture, distro, etc.

Subpath The Package URL Subpath is an extra subpath within a package, relative to the package root.

DejaCode automatically combines the various Package URL elements to display the complete Package URL string. You can also reference that in a Package-based Column Template in your DejaCode reports.

13.2 Setting the Package URL in DejaCode

When you create a new Package in DejaCode, the application automatically derives the Package URL elements when you save it from the data your provide, primarily from the Download URL value.

For existing Packages in DejaCode that do not have the Package URL set, you can use the Administrative Browse Packages form to select those Packages and use the *Set Package URL "purl" from the Download URL command* to prompt DejaCode to calculate and set a Package URL using the available Package data.

13.3 Package Vulnerability Tracking

In DejaCode, there is a Dataspace option to "Enable VulnerableCodeDB access" that authorizes DejaCode to access the VulnerableCodeDB using a Package URL (purl) to determine if there are any reported vulnerabilities for a specific Package and return the Vulnerability ID and related URLs to a Vulnerabilities tab in the Package details user view. DejaCode displays a Vulnerability icon next to the Package identifier in the user view list, and also in any Product Inventory list using that Package.

You can view the VulnerableCodeDB details of an affected Package and use the links to access publicly available reports (e.g. CVE, CPE, GHSA, DSA), discussions, and status updates regarding the vulnerabilities.

Your system administrator can configure DejaCode to provide the necessary credentials to access a VulnerableCodeDB.

For more information about the open source VulnerableCode project, see https://github.com/nexB/vulnerablecode

---

Note: Refer to *Tutorial 2 - Working with Packages* for package creation and maintenance procedures.

CHAPTER

FOURTEEN

CONTRIBUTING TO DEJACODE

Thank you so much for being so interested in contributing to DejaCode. We are always on the lookout for enthusiastic contributors like you who can make our project better, and we're willing to lend a helping hand if you have any questions or need guidance along the way. That being said, here are a few resources to help you get started.

Note: By contributing to the DejaCode project, you agree:

- To the Developer Certificate of Origin.
- That your Contributions to DejaCode are licensed under Apache-2.0.

14.1 Do Your Homework

Before adding a contribution or create a new issue, take a look at the project's README, read through our documentation, and browse existing issues, to develop some understanding of the project and confirm whether a given issue/feature has previously been discussed.

14.2 Ways to Contribute

Contributing to the codebase is not the only way to add value to DejaCode or join our community. Below are some examples to get involved:

14.2.1 First Timers

You are here to help, but you're a new contributor! No worries, we always welcome newcomer contributors. We maintain some good first issues and encourage new contributors to work on those issues for a smooth start.

Warning: "Can I work on this issue?"

You do not need our permission to work on an open issue. A good start is to present your understanding of the problem/bug and how you would fix it. Providing some code using a pull request will come handy, but being able to explain a solution is always a good start.

Make sure to read through this page and follow the recommendations.

> **Warning: "Is this issue is open?"**
>
> Unless closed, yes it is open.

14.2.2 Report Issues

- Report a new bug; just remember to be as specific as possible.
- Create a new issue to request a feature, submit a feedback, or ask a question.
- Look into existing bugs, try to reproduce the issue on your side, and discuss solutions in the comments.

Note: Make sure to check existing issues, to confirm whether a given issue or a question has previously been discussed.

14.2.3 Code Contributions

Code is contributed to the codebase using **pull requests**. A pull request should always be attached to an existing issue. When there is no existing issues, start by creating one to discuss potential solutions and implementation details before writing any code.

We use several conventions to ensure code quality regarding format, testing, and attribution.

Make sure to follow those conventions before submitting your code:

1. **Code validation**

   We use PEP8 conventions. A command is available to automatically format your code:

   ```
   make valid
   ```

2. **Unit tests**

   We write tests, a lot of tests, thousands of tests. When fixing bugs or adding new features, you should add tests too. You can run the test suite with:

   ```
   make test
   ```

3. **Commit messages and Developer Certificate of Origin**

   Follow the instructions at Writing good Commit Messages and check some examples.

   You must include a "Signed-off-by" to your commit messages:

   ```
   Signed-off-by: Philippe Ombredanne <pombredanne@nexb.com>
   ```

4. **Your code is now ready to be pushed as a PR**

Note: Pull requests that are not passing the automated integration tests are unlikely to be reviewed. Focus on making all the "Checks" to pass before asking for a code review.

14.2.4 Documentation Improvements

Documentation is a critical aspect of any project that is usually neglected or overlooked. We value any suggestions to improve DejaCode documentation.

Tip: Our documentation is treated like code. Make sure to check our writing guidelines to help guide new users.

14.2.5 Other Ways

You want to contribute to other aspects of the DejaCode project, and you can't find what you're looking for! You can always discuss new topics, ask questions, and interact with us and other community members on Gitter.

14.3 Helpful Resources

- Review our comprehensive guide for more details on how to add quality contributions to our codebase and documentation
- Check this free resource on how to contribute to an open source project on github
- Follow this wiki page on how to write good commit messages
- Pro Git book
- How to write a good bug report

CHAPTER

FIFTEEN

RELEASE NOTES

Version 5.1.0-dev

- Upgrade Python version to 3.12 and Django to 5.0.x https://github.com/nexB/dejacode/issues/50
- Replace Celery by RQ for async job queue and worker. https://github.com/nexB/dejacode/issues/6
- Add support for CycloneDX spec version "1.6". In the UI and API, older spe version such as "1.4" and "1.5" are also available as download. https://github.com/nexB/dejacode/pull/79
- Lookup in PurlDB by purl in Add Package form. When a Package URL is available in the context of the "Add Package" form, for example when using a link from the Vulnerabilities tab, data is fetched from the PurlDB to initialize the form. https://github.com/nexB/dejacode/issues/47
- If you select two versions of the same Product in the Product list, or two different Products, and click the Compare button, you can now download the results of the comparison to a .xlsx file, making it easy to share the information with your colleagues. https://github.com/nexB/dejacode/issues/7
- Add dark theme support in UI. https://github.com/nexB/dejacode/issues/25
- Add "Load Packages from SBOMs", "Import scan results", and "Pull ScanCode.io project data" feature as Product action in the REST API. https://github.com/nexB/dejacode/issues/59
- Add REST API endpoints to download SBOMs as CycloneDX and SPDX. https://github.com/nexB/dejacode/issues/60
- Refactor the "Import manifest" feature as "Load SBOMs". https://github.com/nexB/dejacode/issues/61
- Add support to import packages from manifest. https://github.com/nexB/dejacode/issues/65
- Add a vulnerability link to the VulnerableCode app in the Vulnerability tab. https://github.com/nexB/dejacode/issues/4
- Add a DEJACODE_SUPPORT_EMAIL setting for support email address customization. https://github.com/nexB/dejacode/issues/76
- Show the individual PURL fields in the Package details view. https://github.com/nexB/dejacode/issues/83
- Fix the logout link of the admin app. https://github.com/nexB/dejacode/issues/89
- Display full commit in the version displayed in the UI https://github.com/nexB/dejacode/issues/88
- Refine the Product comparison logic for Packages. The type and namespace fields are now used along the name field to match similar Packages (excluding the version). https://github.com/nexB/dejacode/issues/113

Version 5.0.1

- Improve the stability of the "Check for new Package versions" feature. https://github.com/nexB/dejacode/issues/17
- Improve the support for SourgeForge download URLs. https://github.com/nexB/dejacode/issues/26

Version 5.0.0

Initial release.

CHAPTER

SIXTEEN

DOCUMENT MAINTENANCE

16.1 Document Software Setup

The DejaCode User Guide is built using Sphinx. See http://www.sphinx-doc.org/en/master/index.html

Individual document files are in reStructuredText format. See http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

You create, build, and preview DejaCode documentation on your local machine.

You commit your updates to the DejaCode repository on GitHub.

16.2 Clone DejaCode

To get started, create or identify a working directory on your local machine.

Open that directory and execute the following command in a terminal session:

```
git clone https://github.com/nexB/dejacode.git
```

That will create a /dejacode directory in your working directory. Now you can install the dependencies in a virtualenv:

```
cd dejacode
python3.12 -m venv .
source bin/activate
```

Now you can build the HTML documents locally:

```
make html
```

Assuming that your Sphinx installation was successful, Sphinx should build a local instance of the documentation .html files:

```
open docs/build/html/index.html
```

You now have a local build of the DejaCode documents.

16.3 Format and style

Use the following tags to highlight elements of the documentation:

- Button/Link: *Click here*
- Value: `value`
- Field: **Field Name**

16.4 Improve DejaCode Documents

Before you begin creating and modifying DejaCode documents, be sure that you understand the basics of reStructured-Text as explained at http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

Ensure that you have the latest DejaCode files:

```
git pull
git status
```

Use your favorite text editor to create and modify .rst files to make your documentation improvements.

Review your work:

```
make html
open docs/build/html/index.html
```

16.5 Share DejaCode Document Improvements

Follow standard git procedures to upload your new and modified files. The following commands are examples:

```
git status
git add source/index.rst
git add source/how-to-scan.rst
git status
git commit -m "New how-to document that explains how to scan"
git status
git push
git status
```

To be continued . . .

CHAPTER

SEVENTEEN

TUTORIAL DOCUMENTS

Tutorial documents provide specific instructions to help you get started.

- *Tutorial 1 - Your first Product*
- *Tutorial 2 - Working with Packages*
- *Tutorial 3 - Working with Reports*

CHAPTER

EIGHTEEN

HOW-TO DOCUMENTS

How-To documents explain how to accomplish specific tasks.

- *How To 1 - Manage your Usage Policies*
- *How To 2 - Take Advantage of Reference Data*

CHAPTER

NINETEEN

REFERENCE DOCUMENTS

Reference documents describe application concepts in depth.

- *Reference 1 - Declared License Expression and License Clarity Scoring*
- *Reference 2 - Understand the Package URL (purl) in DejaCode*

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- modindex
- search