Instalação Redmine Docker Bitnami

Link: https://github.com/bitnami/containers/tree/main/bitnami/redmine

Bitnami package for Redmine

What is Redmine?

> Redmine is an open source management application. It includes a tracking issue system, Gantt charts for a visual view of projects and deadlines, and supports SCM integration for version control.

Overview of Redmine Trademarks: This software listing is packaged by Bitnami. The respective trademarks mentioned in the offering are owned by the respective companies, and use of them does not imply any affiliation or endorsement.

TL;DR

```
docker run --name redmine bitnami/redmine:latest
```

Warning: This quick setup is only intended for development environments. You are encouraged to change the insecure default credentials and check out the available configuration options in the Environment Variables section for a more secure deployment.

Why use Bitnami Images?

- Bitnami closely tracks upstream source changes and promptly publishes new versions of this image using our automated systems.
- With Bitnami images the latest bug fixes and features are available as soon as possible.
- Bitnami containers, virtual machines and cloud images use the same components and configuration approach - making it easy to switch between formats based on your project needs.
- All our images are based on **minideb** -a minimalist Debian based container image that gives you a small base container image and the familiarity of a leading Linux distribution- or **scratch** -an explicitly empty image-.
- All Bitnami images available in Docker Hub are signed with Notation. Check this post to know how to verify the integrity of the images.
- Bitnami container images are released on a regular basis with the latest distribution packages available.

Looking to use Redmine in production? Try VMware Tanzu Application Catalog, the commercial edition of the Bitnami catalog.

How to deploy Redmine in Kubernetes?

Deploying Bitnami applications as Helm Charts is the easiest way to get started with our applications on Kubernetes. Read more about the installation in the Bitnami Redmine Chart GitHub repository.

Bitnami containers can be used with Kubeapps for deployment and management of Helm Charts in clusters.

Only latest stable or LTS branch maintained in the free Bitnami catalog

Starting December 10th 2024, only the latest stable or LTS branch of any container will receive updates in the free Bitnami catalog. To access up-to-date releases for all upstream-supported

branches, consider upgrading to Bitnami Premium. Previous versions already released will not be deleted. They are still available to pull from DockerHub.

Please check the Bitnami Premium page in our partner Arrow Electronics for more information.

Supported tags and respective `Dockerfile` links

Learn more about the Bitnami tagging policy and the difference between rolling tags and immutable tags in our documentation page.

You can see the equivalence between the different tags by taking a look at the `tags-info.yaml` file present in the branch folder, i.e `bitnami/ASSET/BRANCH/DISTRO/tags-info.yaml`.

Subscribe to project updates by watching the bitnami/containers GitHub repo.

Get this image

The recommended way to get the Bitnami Redmine Docker Image is to pull the prebuilt image from the Docker Hub Registry.

```
docker pull bitnami/redmine:latest
```

To use a specific version, you can pull a versioned tag. You can view the list of available versions in the Docker Hub Registry.

```
docker pull bitnami/redmine:[TAG]
```

If you wish, you can also build the image yourself by cloning the repository, changing to the directory containing the Dockerfile and executing the `docker build` command. Remember to replace the `APP`, `VERSION` and `OPERATING-SYSTEM` path placeholders in the example command below with the correct values.

```
git clone https://github.com/bitnami/containers.git
cd bitnami/APP/VERSION/OPERATING-SYSTEM
docker build -t bitnami/APP:latest .
```

How to use this image

Redmine requires access to a MySQL, MariaDB or PostgreSQL database to store information. We'll use the Bitnami Docker Image for MariaDB for the database requirements.

Using the Docker Command Line

Step 1: Create a network

```
docker network create redmine-network
```

Step 2: Create a volume for MariaDB persistence and create a MariaDB container

```
$ docker volume create --name mariadb_data
docker run -d --name mariadb \
  --env ALLOW_EMPTY_PASSWORD=yes \
  --env MARIADB_USER=bn_redmine \
  --env MARIADB_PASSWORD=bitnami \
  --env MARIADB_DATABASE=bitnami_redmine \
  --network redmine-network \
  --volume mariadb_data:/bitnami/mariadb \
  bitnami/mariadb:latest
```

Step 3: Create volumes for Redmine persistence and launch the container

```
$ docker volume create --name redmine_data
docker run -d --name redmine \
  -p 8080:8080 -p 8443:8443 \
  --env ALLOW_EMPTY_PASSWORD=yes \
  --env REDMINE_DATABASE_USER=bn_redmine \
  --env REDMINE_DATABASE_PASSWORD=bitnami \
  --env REDMINE_DATABASE_NAME=bitnami_redmine \
  --network redmine-network \
  --volume redmine_data:/bitnami/redmine \
  bitnami/redmine:latest
```

Access your application at `http://your-ip/`

Run the application using Docker Compose

```
curl -sSL https://raw.githubusercontent.com/bitnami/containers/main/bitnami/redmine/docker-compo
docker-compose up -d
```

Please be aware this file has not undergone internal testing. Consequently, we advise its use exclusively for development or testing purposes. For production-ready deployments, we highly recommend utilizing its associated Bitnami Helm chart.

If you detect any issue in the `docker-compose.yaml` file, feel free to report it or contribute with a fix by following our Contributing Guidelines.

Persisting your application

If you remove the container all your data will be lost, and the next time you run the image the database will be reinitialized. To avoid this loss of data, you should mount a volume that will persist even after the container is removed.

For persistence you should mount a directory at the `/bitnami/redmine` path. If the mounted directory is empty, it will be initialized on the first run. Additionally you should mount a volume for persistence of the MariaDB data.

The above examples define the Docker volumes named `mariadb_data` and `redmine_data`. The Redmine application state will persist as long as volumes are not removed.

To avoid inadvertent removal of volumes, you can mount host directories as data volumes. Alternatively you can make use of volume plugins to host the volume data.

Mount host directories as data volumes with Docker Compose

This requires a minor change to the `docker-compose.yml` file present in this repository:

```
  mariadb:
    ...
    volumes:
```

```
-      - 'mariadb_data:/bitnami/mariadb'
+      - /path/to/mariadb-persistence:/bitnami/mariadb
   ...
   redmine:
     ...
     volumes:
-      - 'redmine_data:/bitnami/redmine'
+      - /path/to/redmine-persistence:/bitnami/redmine
   ...
-volumes:
-  mariadb_data:
-    driver: local
-  redmine_data:
-    driver: local
```

> NOTE: As this is a non-root container, the mounted files and directories must have the proper permissions for the UID `1001`.

Mount host directories as data volumes using the Docker command line

Step 1: Create a network (if it does not exist)

```
docker network create redmine-network
```

Step 2. Create a MariaDB container with host volume

```
docker run -d --name mariadb \
  --env ALLOW_EMPTY_PASSWORD=yes \
  --env MARIADB_USER=bn_redmine \
  --env MARIADB_PASSWORD=bitnami \
  --env MARIADB_DATABASE=bitnami_redmine \
  --network redmine-network \
  --volume /path/to/mariadb-persistence:/bitnami/mariadb \
  bitnami/mariadb:latest
```

Step 3. Create the Redmine container with host volumes

```
docker run -d --name redmine \
  -p 8080:8080 -p 8443:8443 \
  --env ALLOW_EMPTY_PASSWORD=yes \
```

```
  --env REDMINE_DATABASE_USER=bn_redmine \
  --env REDMINE_DATABASE_PASSWORD=bitnami \
  --env REDMINE_DATABASE_NAME=bitnami_redmine \
  --network redmine-network \
  --volume /path/to/redmine-persistence:/bitnami/redmine \
  bitnami/redmine:latest
```

Configuration

Environment variables

Customizable environment variables

Name	Description	Default Value
`REDMINE_DATA_TO_PERSIST`	Files to persist relative to the Redmine installation directory. To provide multiple values, separate them with a whitespace.	`${REDMINE_CONF_DIR}/configuration.yml ${REDMINE_CONF_DIR}/database.yml files plugins public/plugin_assets`
`REDMINE_PORT_NUMBER`	Port number in which Redmine will run.	`3000`
`REDMINE_ENV`	Redmine environment mode. Allowed values: *development*, *production*, *test* .	`production`
`REDMINE_LANGUAGE`	Redmine site default language.	`en`
`REDMINE_REST_API_ENABLED`	Whether to allow REST API calls to Redmine.	`0`
`REDMINE_LOAD_DEFAULT_DATA`	Whether to generate default data for Redmine.	`yes`
`REDMINE_SKIP_BOOTSTRAP`	Whether to perform initial bootstrapping for the application.	`nil`
`REDMINE_QUEUE_ADAPTER`	Active job queue adapter. You may need to install additional dependencies if you select a value other than "async" or "inline".	`inline`
`REDMINE_USERNAME`	Redmine user name.	`user`
`REDMINE_PASSWORD`	Redmine user password.	`bitnami1`
`REDMINE_EMAIL`	Redmine user e-mail address.	`user@example.com`

Name	Description	Default Value
`REDMINE_FIRST_NAME`	Redmine user first name.	`UserName`
`REDMINE_LAST_NAME`	Redmine user last name.	`LastName`
`REDMINE_SMTP_HOST`	Redmine SMTP server host.	`nil`
`REDMINE_SMTP_PORT_NUMBER`	Redmine SMTP server port number.	`nil`
`REDMINE_SMTP_USER`	Redmine SMTP server user.	`nil`
`REDMINE_SMTP_DOMAIN`	Redmine SMTP domain. USER@ part from SMTP_USER is used when not defined.	`nil`
`REDMINE_SMTP_PASSWORD`	Redmine SMTP server user password.	`nil`
`REDMINE_SMTP_PROTOCOL`	Redmine SMTP server protocol to use.	`nil`
`REDMINE_SMTP_AUTH`	Redmine SMTP server protocol to use. Allowed values: *login*, *plain*, *cram_md5*.	`login`
`REDMINE_SMTP_OPENSSL_VERIFY_MODE`	SMTP sets the level of verification for the SSL certificate presented by the server. Allowed values: *none*, *peer*.	`peer`
`REDMINE_SMTP_CA_FILE`	Path to the SMTP CA file.	`/etc/ssl/certs/ca-certificates.crt`
`REDMINE_DATABASE_TYPE`	Database type to be used for the Redmine installation. Allowed values: *mariadb*, *postgresql*.	`mariadb`
`REDMINE_DATABASE_HOST`	Database server host.	`$REDMINE_DEFAULT_DATABASE_HOST`
`REDMINE_DATABASE_PORT_NUMBER`	Database server port.	`3306`
`REDMINE_DATABASE_NAME`	Database name.	`bitnami_redmine`
`REDMINE_DATABASE_USER`	Database user name.	`bn_redmine`
`REDMINE_DATABASE_PASSWORD`	Database user password.	`nil`

Read-only environment variables

Name	Description	Value
`REDMINE_BASE_DIR`	Redmine installation directory.	`${BITNAMI_ROOT_DIR}/redmine`
`REDMINE_CONF_DIR`	Redmine directory for configuration files.	`${REDMINE_BASE_DIR}/config`
`REDMINE_VOLUME_DIR`	Redmine directory for mounted configuration files.	`${BITNAMI_VOLUME_DIR}/redmine`
`REDMINE_DAEMON_USER`	Redmine system user.	`redmine`
`REDMINE_DAEMON_GROUP`	Redmine system group.	`redmine`

Name	Description	Value
`REDMINE_DEFAULT_DATABASE_HOST`	Default database server host.	`mariadb`

When you start the Redmine image, you can adjust the configuration of the instance by passing one or more environment variables either on the docker-compose file or on the `docker run` command line. If you want to add a new environment variable:

- For docker-compose add the variable name and value under the application section in the `docker-compose.yml` file present in this repository:

```
redmine:
  ...
  environment:

  ...
```

- For manual execution add a `--env` option with each variable and value:

```
$ docker run -d --name redmine -p 80:8080 -p 443:8443 \
                                --env                  REDMINE_PAS
                                                --network
                                        --volume            /path/to/redmine-persister
  bitnami/redmine:latest
```

Examples

SMTP configuration using a Gmail account

This would be an example of SMTP configuration using a Gmail account:

- Modify the `docker-compose.yml` file present in this repository:

```
  redmine:
    ...
    environment:

  ...
```

- For manual execution:

```
$ docker run -d --name redmine -p 80:8080 -p 443:8443 \
                                        --env REDMINE_DATABASE_
                                    --env REDMINE_DATABASE_NAME=bi
                                        --env REDMINE_SMTP_HOS
                                                --env
                                    --env REDMINE_SMTP_USER=your_em
                                       --env REDMINE_SMTP_PASSWORD
                                                --network
                                    --volume /path/to/redmine-persiste
  bitnami/redmine:latest
```

Connect Redmine container to an existing database

The Bitnami Redmine container supports connecting the Redmine application to an external database. This would be an example of using an external database for Redmine.

- Modify the `docker-compose.yml` file present in this repository:

```
   redmine:
     ...
     environment:
-
+




-
+
     ...
```

- For manual execution:

```
$ docker run -d --name redmine\
                        -p 8080:8080 -p 8
                                    --network
                                --env REDMINE_DATABASE_HO
                                 --env REDMINE_DATABASE_P
                                  --env REDMINE_DATABASE_
                                --env REDMINE_DATABASE_US
                            --env REDMINE_DATABASE_PASSWORD=redmi
                                  --volume redmine_data:/
  bitnami/redmine:latest
```

In case the database already contains data from a previous Redmine installation, you need to set the variable `REDMINE_SKIP_BOOTSTRAP` to `yes`. Otherwise, the container would execute the installation wizard and could modify the existing data in the database. Note that, when setting `REDMINE_SKIP_BOOTSTRAP` to `yes`, values for environment variables such as `REDMINE_USERNAME`, `REDMINE_PASSWORD` or `REDMINE_EMAIL` will be ignored.

Logging

The Bitnami Redmine Docker image sends the container logs to `stdout`. To view the logs:

```
docker logs redmine
```

Or using Docker Compose:

```
docker-compose logs redmine
```

You can configure the containers logging driver using the `--log-driver` option if you wish to consume the container logs differently. In the default configuration docker uses the `json-file` driver.

Customize this image

The Bitnami Redmine Docker image is designed to be extended.

Extend this image

Before extending this image, please note there are certain configuration settings you can modify using the original image:

- Settings that can be adapted using environment variables. For instance, you can change the port used by Redmine by setting the environment variable `REDMINE_PORT_NUMBER`.
- You can mount your custom scripts under `/docker-entrypoint-init.d` directory. These scripts will be executed in alphabetical order when the container during the 1st container bootstrap.

If your desired customizations cannot be covered using the methods mentioned above, extend the image. To do so, create your own image using a Dockerfile with the format below:

```
FROM bitnami/redmine
### Put your customizations below
...
```

Here is an example of extending to install custom plugins:

```
FROM bitnami/redmine
```

```
### Install custom plugins
RUN cd /opt/bitnami/redmine && \
    git clone https://github.com/user_name/name_of_the_plugin.git plugins/name_of_the_plugin &&
    bundle config set frozen false && bundle install && bundle config set frozen true
```

Maintenance

Backing up your container

To backup your data, configuration and logs, follow these simple steps:

Step 1: Stop the currently running container

```
docker stop redmine
```

Or using Docker Compose:

```
docker-compose stop redmine
```

Step 2: Run the backup command

We need to mount two volumes in a container we will use to create the backup: a directory on your host to store the backup in, and the volumes from the container we just stopped so we can access the data.

```
docker run --rm -v /path/to/redmine-backups:/backups --volumes-from redmine busybox \
  cp -a /bitnami/redmine /backups/latest
```

Restoring a backup

Restoring a backup is as simple as mounting the backup as volumes in the containers.

For the MariaDB database container:

```
 $ docker run -d --name mariadb \
   ...
-  --volume /path/to/mariadb-persistence:/bitnami/mariadb \
+  --volume /path/to/mariadb-backups/latest:/bitnami/mariadb \
```

```
   bitnami/mariadb:latest
```

For the Redmine container:

```
 $ docker run -d --name redmine \
   ...
-  --volume /path/to/redmine-persistence:/bitnami/redmine \
+  --volume /path/to/redmine-backups/latest:/bitnami/redmine \
   bitnami/redmine:latest
```

Upgrade this image

Bitnami provides up-to-date versions of MariaDB and Redmine, including security patches, soon after they are made upstream. We recommend that you follow these steps to upgrade your container. We will cover here the upgrade of the Redmine container. For the MariaDB upgrade see:

https://github.com/bitnami/containers/tree/main/bitnami/mariadb#upgrade-this-image

The `bitnami/redmine:latest` tag always points to the most recent release. To get the most recent release you can simple repull the `latest` tag from the Docker Hub with `docker pull bitnami/redmine:latest`. However it is recommended to use tagged versions.

Step 1: Get the updated image

```
docker pull bitnami/redmine:latest
```

Step 2: Stop the running container

Stop the currently running container using the command

```
docker-compose stop redmine
```

Step 3: Take a snapshot of the application state

Follow the steps in Backing up your container to take a snapshot of the current application state.

Step 4: Remove the currently running container

Remove the currently running container by executing the following command:

```
docker-compose rm -v redmine
```

Step 5: Run the new image

Update the image tag in `docker-compose.yml` and re-create your container with the new image:

```
docker-compose up -d
```

Notable Changes

4.2.1-debian-10-r70

- The size of the container image has been decreased.
- The configuration logic is now based on Bash scripts in the *rootfs/* folder.
- It is now possible to use an already populated Redmine database from another installation. In order to do this, use the environment variable `REDMINE_SKIP_BOOTSTRAP`, which forces the container not to run the initial Redmine setup wizard.
- The following environment variables have been deprecated. They will continue to work as before, but support for these may be removed in a future update:
  - `REDMINE_DB_POSTGRES` in favor of `REDMINE_DATABASE_HOST`. When used, `REDMINE_DATABASE_TYPE=postgresql` will also be set.
  - `REDMINE_DB_MYSQL`, in favor of `REDMINE_DATABASE_HOST`. Whenused, `REDMINE_DATABASE_TYPE=mariadb` will also be set.

Contributing

We'd love for you to contribute to this container. You can request new features by creating an issue or submitting a pull request with your contribution.

Issues

If you encountered a problem running this container, you can file an issue. For us to provide better support, be sure to fill the issue template.

License

Copyright © 2025 Broadcom. The term "Broadcom" refers to Broadcom Inc. and/or its subsidiaries.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Revision #1
Created 2025-04-21 14:15:18 UTC by Administrador
Updated 2025-04-21 14:22:35 UTC by Administrador