Instalação WordPress Docker (Bitnami)

Link: https://github.com/bitnami/containers/tree/main/bitnami/wordpress#how-to-use-this-image

Bitnami package for WordPress

What is WordPress?

> WordPress is the world's most popular blogging and content management platform. Powerful yet simple, everyone from students to global corporations use it to build beautiful, functional websites.

Overview of WordPress

TL;DR

```
docker run --name wordpress bitnami/wordpress: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 WordPress in production? Try VMware Tanzu Application Catalog, the commercial edition of the Bitnami catalog.

How to deploy WordPress 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 WordPress Chart GitHub repository.

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

Why use a non-root container?

Non-root container images add an extra layer of security and are generally recommended for production environments. However, because they run as a non-root user, privileged tasks are typically off-limits. Learn more about non-root containers in our docs.

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 WordPress Docker Image is to pull the prebuilt image from the Docker Hub Registry.

```
docker pull bitnami/wordpress: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/wordpress:[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

WordPress requires access to a MySQL or MariaDB 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 wordpress-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_wordpress \
  --env MARIADB_PASSWORD=bitnami \
  --env MARIADB_DATABASE=bitnami_wordpress \
  --network wordpress-network \
  --volume mariadb_data:/bitnami/mariadb \
  bitnami/mariadb:latest
```

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

```
$ docker volume create --name wordpress_data
docker run -d --name wordpress \
  -p 8080:8080 -p 8443:8443 \
  --env ALLOW_EMPTY_PASSWORD=yes \
  --env WORDPRESS_DATABASE_USER=bn_wordpress \
  --env WORDPRESS_DATABASE_PASSWORD=bitnami \
  --env WORDPRESS_DATABASE_NAME=bitnami_wordpress \
  --network wordpress-network \
  --volume wordpress_data:/bitnami/wordpress \
  bitnami/wordpress:latest
```

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

Run the application using Docker Compose

```
curl -sSL https://raw.githubusercontent.com/bitnami/containers/main/bitnami/wordpress/docker-com
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/wordpress` 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 `wordpress_data`. The WordPress 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
   ...
   wordpress:
     ...
     volumes:
-      - 'wordpress_data:/bitnami/wordpress'
+      - /path/to/wordpress-persistence:/bitnami/wordpress
   ...
-volumes:
-  mariadb_data:
-    driver: local
```

```
-  wordpress_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 wordpress-network
```

Step 2. Create a MariaDB container with host volume

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

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

Step 3. Create the WordPress container with host volumes

```
docker run -d --name wordpress \
  -p 8080:8080 -p 8443:8443 \
  --env ALLOW_EMPTY_PASSWORD=yes \
  --env WORDPRESS_DATABASE_USER=bn_wordpress \
  --env WORDPRESS_DATABASE_PASSWORD=bitnami \
  --env WORDPRESS_DATABASE_NAME=bitnami_wordpress \
  --network wordpress-network \
  --volume /path/to/wordpress-persistence:/bitnami/wordpress \
  bitnami/wordpress:latest
```

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

Configuration

Environment variables

Customizable environment variables

Name	Description	Default Value
`WORDPRESS_DATA_TO_PERSIST`	Files to persist relative to the WordPress installation directory. To provide multiple values, separate them with a whitespace.	`wp-config.php wp-content`
`WORDPRESS_ENABLE_HTTPS`	Whether to enable HTTPS for WordPress by default.	`no`
`WORDPRESS_BLOG_NAME`	WordPress blog name.	`"User's blog"`
`WORDPRESS_SCHEME`	Scheme to generate application URLs. Deprecated by `WORDPRESS_ENABLE_HTTPS`.	`http`
`WORDPRESS_HTACCESS_OVERRIDE_NONE`	Set the Apache `AllowOverride` variable to `None`. All the default directives will be loaded from `/opt/bitnami/wordpress/wordpress-htaccess.conf`.	`yes`
`WORDPRESS_ENABLE_HTACCESS_PERSISTENCE`	Persist the custom changes of the htaccess. It depends on the value of `WORDPRESS_HTACCESS_OVERRIDE_NONE`, when `yes` it will persist `/opt/bitnami/wordpress/wordpress-htaccess.conf` if `no` it will persist `/opt/bitnami/wordpress/.htaccess`.	`no`
`WORDPRESS_RESET_DATA_PERMISSIONS`	Force resetting ownership/permissions on persisted data when initializing, otherwise it assumes the ownership/permissions are correct. Ignored when running as non-root.	`no`
`WORDPRESS_TABLE_PREFIX`	Table prefix to use in WordPress.	`wp_`
`WORDPRESS_PLUGINS`	List of WordPress plugins to install and activate, separated via commas. Can also be set to `all` to activate all currently installed plugins, or `none` to skip.	`none`
`WORDPRESS_EXTRA_INSTALL_ARGS`	Extra flags to append to the WordPress 'wp core install' command call.	`nil`
`WORDPRESS_EXTRA_CLI_ARGS`	Extra flags to append to all WP-CLI command calls.	`nil`
`WORDPRESS_EXTRA_WP_CONFIG_CONTENT`	Extra configuration to append to wp-config.php during install.	`nil`
`WORDPRESS_SKIP_BOOTSTRAP`	Whether to perform initial bootstrapping for the application.	`no`

Name	Description	Default Value
`WORDPRESS_AUTO_UPDATE_LEVEL`	Level of auto-updates to allow for the WordPress core installation. Valid values: `major`, `minor`, `none`.	`none`
`WORDPRESS_AUTH_KEY`	Value of the AUTH_KEY	`nil`
`WORDPRESS_SECURE_AUTH_KEY`	Value of the SECURE_AUTH_KEY	`nil`
`WORDPRESS_LOGGED_IN_KEY`	Value of the LOGGED_IN_KEY	`nil`
`WORDPRESS_NONCE_KEY`	Value of the NONCE_KEY	`nil`
`WORDPRESS_AUTH_SALT`	Value of the AUTH_SALT	`nil`
`WORDPRESS_SECURE_AUTH_SALT`	Value of the SECURE_AUTH_SALT	`nil`
`WORDPRESS_LOGGED_IN_SALT`	Value of the LOGGED_IN_SALT	`nil`
`WORDPRESS_NONCE_SALT`	Value of the NONCE_SALT	`nil`
`WORDPRESS_ENABLE_REVERSE_PROXY`	Enable WordPress support for reverse proxy headers	`no`
`WORDPRESS_ENABLE_XML_RPC`	Enable the WordPress XML-RPC endpoint	`no`
`WORDPRESS_USERNAME`	WordPress user name.	`user`
`WORDPRESS_PASSWORD`	WordPress user password.	`bitnami`
`WORDPRESS_EMAIL`	WordPress user e-mail address.	`user@example.com`
`WORDPRESS_FIRST_NAME`	WordPress user first name.	`UserName`
`WORDPRESS_LAST_NAME`	WordPress user last name.	`LastName`
`WORDPRESS_ENABLE_MULTISITE`	Enable WordPress Multisite configuration.	`no`
`WORDPRESS_MULTISITE_NETWORK_TYPE`	WordPress Multisite network type to enable. Valid values: `subfolder`, `subdirectory`, `subdomain`.	`subdomain`
`WORDPRESS_MULTISITE_EXTERNAL_HTTP_PORT_NUMBER`	External HTTP port for WordPress Multisite.	`80`
`WORDPRESS_MULTISITE_EXTERNAL_HTTPS_PORT_NUMBER`	External HTTPS port for WordPress Multisite.	`443`
`WORDPRESS_MULTISITE_HOST`	WordPress hostname/address. Only used for Multisite installations.	`nil`

Name	Description	Default Value
`WORDPRESS_MULTISITE_ENABLE_NIP_IO_REDIRECTION`	Whether to enable IP address redirection to nip.io wildcard DNS when enabling WordPress Multisite. This is only supported when running on an IP address with subdomain network type.	`no`
`WORDPRESS_MULTISITE_FILEUPLOAD_MAXK`	Maximum upload file size allowed for WordPress Multisite uploads, in kilobytes.	`81920`
`WORDPRESS_SMTP_HOST`	WordPress SMTP server host.	`nil`
`WORDPRESS_SMTP_PORT_NUMBER`	WordPress SMTP server port number.	`nil`
`WORDPRESS_SMTP_USER`	WordPress SMTP server user.	`nil`
`WORDPRESS_SMTP_FROM_EMAIL`	WordPress SMTP from email.	`${WORDPRESS_SMTP_USER}`
`WORDPRESS_SMTP_FROM_NAME`	WordPress SMTP from name.	`${WORDPRESS_FIRST_NAME} ${WORDPRESS_LAST_NAME}`
`WORDPRESS_SMTP_PASSWORD`	WordPress SMTP server user password.	`nil`
`WORDPRESS_SMTP_PROTOCOL`	WordPress SMTP server protocol to use.	`nil`
`WORDPRESS_DATABASE_HOST`	Database server host.	`$WORDPRESS_DEFAULT_DATABASE_HOST`
`WORDPRESS_DATABASE_PORT_NUMBER`	Database server port.	`3306`
`WORDPRESS_DATABASE_NAME`	Database name.	`bitnami_wordpress`
`WORDPRESS_DATABASE_USER`	Database user name.	`bn_wordpress`
`WORDPRESS_DATABASE_PASSWORD`	Database user password.	`nil`
`WORDPRESS_ENABLE_DATABASE_SSL`	Whether to enable SSL for database connections.	`no`
`WORDPRESS_VERIFY_DATABASE_SSL`	Whether to verify the database SSL certificate when SSL is enabled for database connections.	`yes`
`WORDPRESS_DATABASE_SSL_CERT_FILE`	Path to the database client certificate file.	`nil`
`WORDPRESS_DATABASE_SSL_KEY_FILE`	Path to the database client certificate key file.	`nil`
`WORDPRESS_DATABASE_SSL_CA_FILE`	Path to the database server CA bundle file.	`nil`
`WORDPRESS_OVERRIDE_DATABASE_SETTINGS`	Override the database settings in persistence.	`no`

Read-only environment variables

Name	Description	Value
`WORDPRESS_BASE_DIR`	WordPress installation directory.	`${BITNAMI_ROOT_DIR}/wordpress`
`WORDPRESS_CONF_FILE`	Configuration file for WordPress.	`${WORDPRESS_BASE_DIR}/wp-config.php`
`WP_CLI_BASE_DIR`	WP-CLI installation directory.	`${BITNAMI_ROOT_DIR}/wp-cli`
`WP_CLI_BIN_DIR`	WP-CLI directory for binary files.	`${WP_CLI_BASE_DIR}/bin`
`WP_CLI_CONF_DIR`	WP-CLI directory for configuration files.	`${WP_CLI_BASE_DIR}/conf`
`WP_CLI_CONF_FILE`	Configuration file for WP-CLI.	`${WP_CLI_CONF_DIR}/wp-cli.yml`
`WORDPRESS_VOLUME_DIR`	WordPress directory for mounted configuration files.	`${BITNAMI_VOLUME_DIR}/wordpress`
`WORDPRESS_DEFAULT_DATABASE_HOST`	Default database server host.	`mariadb`
`PHP_DEFAULT_MEMORY_LIMIT`	Default PHP memory limit.	`512M`
`PHP_DEFAULT_POST_MAX_SIZE`	Default PHP post_max_size.	`80M`
`PHP_DEFAULT_UPLOAD_MAX_FILESIZE`	Default PHP upload_max_size.	`80M`
`WP_CLI_DAEMON_USER`	WP-CLI system user.	`daemon`
`WP_CLI_DAEMON_GROUP`	WP-CLI system group.	`daemon`

When you start the WordPress 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. Please note that some variables are only considered when the container is started for the first time. 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:

```
wordpress:
  ...
  environment:

  ...
```

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

```
$ docker run -d --name wordpress -p 80:8080 -p 443:8443 \
                                        --env                   WORDPRESS_PASS
                                                --network
                                --volume                  /path/to/wordpress-persisten
  bitnami/wordpress: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:

```
  wordpress:
    ...
    environment:








  ...
```

- For manual execution:

```
$ docker run -d --name wordpress -p 80:8080 -p 443:8443 \
  --env WORDPRESS_DATABASE_USEI
  --env WORDPRESS_DATABASE_NAME=bitnam
  --env WORDPRESS_SMTP_HOST=
  --env WOR
  --env WORDPRESS_SMTP_USER=your_ema
  --env WORDPRESS_SMTP_PASSWORD=
  --network
  --volume /path/to/wordpress-persisten
  bitnami/wordpress:latest
```

Connect WordPress container to an existing database

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

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

```
    wordpress:
      ...
      environment:
-
+
```

```
-
+
      ...
```

- For manual execution:

```
$ docker run -d --name wordpress\
    -p 8080:8080 -p 8
    --network
  --env WORDPRESS_DATABASE_HOS
   --env WORDPRESS_DATABASE_PO
  --env WORDPRESS_DATABASE_NAM
 --env WORDPRESS_DATABASE_USER=w
  --env WORDPRESS_DATABASE_PASSWORD=wordpress_
  --volume wordpress_data:/bit
  bitnami/wordpress:latest
```

In case the database already contains data from a previous WordPress installation, you need to set the variable `WORDPRESS_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 `WORDPRESS_SKIP_BOOTSTRAP` to `yes`, values for environment variables such as `WORDPRESS_USERNAME`, `WORDPRESS_PASSWORD` or `WORDPRESS_EMAIL` will be ignored. Make sure that, in this imported database, the table prefix matches the one set in `WORDPRESS_TABLE_PREFIX`.

WP-CLI tool

The Bitnami WordPress container includes the command line interface **wp-cli** that can help you to manage and interact with your WP sites. To run this tool, please note you need use the proper system user, **daemon**.

This would be an example of using **wp-cli** to display the help menu:

- Using `docker-compose` command:

```
docker-compose exec wordpress wp help
```

- Using `docker` command:

```
docker exec wordpress wp help
```

Find more information about parameters available in the tool in the official documentation.

Logging

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

```
docker logs wordpress
```

Or using Docker Compose:

```
docker-compose logs wordpress
```

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.

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 wordpress
```

Or using Docker Compose:

```
docker-compose stop wordpress
```

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/wordpress-backups:/backups --volumes-from wordpress busybox \
  cp -a /bitnami/wordpress /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 WordPress container:

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

Upgrade this image

Bitnami provides up-to-date versions of MariaDB and WordPress, 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 WordPress container. For the MariaDB upgrade see https://github.com/bitnami/containers/tree/main/bitnami/mariadb#upgrade-this-image

The `bitnami/wordpress: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/wordpress:latest`. However it is recommended to use tagged versions.

Step 1: Get the updated image

```
docker pull bitnami/wordpress:latest
```

Step 2: Stop the running container

Stop the currently running container using the command

```
docker-compose stop wordpress
```

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 wordpress
```

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
```

Customize this image

The Bitnami WordPress Docker image is designed to be extended so it can be used as the base image for your custom web applications.

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 ports used by Apache for HTTP and HTTPS, by setting the environment variables `APACHE_HTTP_PORT_NUMBER` and `APACHE_HTTPS_PORT_NUMBER` respectively.
- Adding custom virtual hosts.
- Replacing the 'httpd.conf' file.
- Using custom SSL certificates.

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/wordpress
## Put your customizations below
...
```

Here is an example of extending the image with the following modifications:

- Install the `vim` editor
- Modify the Apache configuration file
- Modify the ports used by Apache

```
FROM bitnami/wordpress

## Change user to perform privileged actions
USER 0
## Install 'vim'
RUN install_packages vim
## Revert to the original non-root user
USER 1001

## Enable mod_ratelimit module
RUN sed -i -r 's/#LoadModule ratelimit_module/LoadModule ratelimit_module/' /opt/bitnami/apache/

## Modify the ports used by Apache by default
# It is also possible to change these environment variables at runtime
ENV APACHE_HTTP_PORT_NUMBER=8181
ENV APACHE_HTTPS_PORT_NUMBER=8143
EXPOSE 8181 8143
```

Based on the extended image, you can update the `docker-compose.yml` file present in this repository to add other features:

```
   wordpress:
-    image: bitnami/wordpress:latest
+    build: .
     ports:
-      - '80:8080'
-      - '443:8443'
+      - '80:8181'
+      - '443:8143'
     environment:
+      - PHP_MEMORY_LIMIT=512m
     ...
```

Notable Changes

6.4.1-debian-11-r5

- The XML-RCP endpoint has been disabled by default. Users can manually activate via the new `WORDPRESS_ENABLE_XML_RPC` environment variable.

5.7.1-debian-10-r21

- The size of the container image has been decreased.
- The configuration logic is now based on Bash scripts in the *rootfs/* folder.
- Multisite support was added via `WORDPRESS_ENABLE_MULTISITE` and related environment variables.
- Plugins can be installed and activated on the first deployment via `WORDPRESS_PLUGINS`.
- Added support for limiting auto-updates to WordPress core via `WORDPRESS_AUTO_UPDATE_LEVEL`. In addition, auto-updates have been disabled by default. To update WordPress core, we recommend to swap the container image version for your deployment instead of using the built-in update functionality.
- This image now supports connecting to MySQL and MariaDB databases securely via SSL.

5.3.2-debian-10-r30

- The WordPress container has been migrated to a "non-root" user approach. Previously the container ran as the `root` user and the Apache daemon was started as the `daemon` user. From now on, both the container and the Apache daemon run as user `1001`. You can revert this behavior by changing `USER 1001` to `USER root` in the Dockerfile.
- Consequences:
  - The HTTP/HTTPS ports exposed by the container are now `8080/8443` instead of `80/443`.
  - Backwards compatibility is not guaranteed when data is persisted using docker or docker-compose. We highly recommend migrating the WP site by exporting its content, and importing it on a new WordPress container. In the links below you'll find some alternatives:
    - Migrate WordPress using All-in-One WP Migration plugin
    - Migrate WordPress using VaultPress
  - No writing permissions will be granted on `wp-config.php` by default.

5.2.1-debian-9-r9 and 5.2.1-ol-7-r9

- This image has been adapted so it's easier to customize. See the Customize this image section for more information.
- The Apache configuration volume (`/bitnami/apache`) has been deprecated, and support for this feature will be dropped in the near future. Until then, the container will enable the Apache configuration from that volume if it exists. By default, and if the configuration

volume does not exist, the configuration files will be regenerated each time the container is created. Users wanting to apply custom Apache configuration files are advised to mount a volume for the configuration at `/opt/bitnami/apache/conf`, or mount specific configuration files individually.
- The PHP configuration volume (`/bitnami/php`) has been deprecated, and support for this feature will be dropped in the near future. Until then, the container will enable the PHP configuration from that volume if it exists. By default, and if the configuration volume does not exist, the configuration files will be regenerated each time the container is created. Users wanting to apply custom PHP configuration files are advised to mount a volume for the configuration at `/opt/bitnami/php/conf`, or mount specific configuration files individually.
- Enabling custom Apache certificates by placing them at `/opt/bitnami/apache/certs` has been deprecated, and support for this functionality will be dropped in the near future. Users wanting to enable custom certificates are advised to mount their certificate files on top of the preconfigured ones at `/certs`.

5.1.1-r28, 5.1.1-rhel-7-r31 and 5.1.1-ol-7-r30

- Users reported that they wanted to import their WordPress database from other installations. Now, in order to cover this use case, the variable `WORDPRESS_SKIP_INSTALL` can be set to avoid the container launch the WordPress installation wizard.

5.0.3-r20

- For performance and security reasons, Apache will set the `AllowOverride` directive to `None` by default. This means that, instead of using `.htaccess` files, all the default directives will be moved to the `/opt/bitnami/wordpress/wordpress-htaccess.conf` file. The only downside of this is the compatibility with certain plugins, which would require changes in that file (you would need to mount a modified version of `wordpress-htaccess.conf` compatible with these plugins). If you want to have the default `.htaccess` behavior, set the `WORDPRESS_HTACCESS_OVERRIDE_NONE` env var to `no`.

5.0.0-r0

- **wp-cli** tool is included in the Docker image. Find it at **/opt/bitnami/wp-cli/bin/wp**.

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 © 2024 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 2024-09-07 01:58:22 UTC by Administrador
Updated 2024-09-07 02:01:35 UTC by Administrador