Instalação Minio docker bitnami

Link: https://hub.docker.com/r/bitnami/minio

What is Bitnami Object Storage based on MinIO®?

> MinIO® is an object storage server, compatible with Amazon S3 cloud storage service, mainly used for storing unstructured data (such as photos, videos, log files, etc.).

Overview of Bitnami Object Storage based on MinIO® Disclaimer: All software products, projects and company names are trademark(TM) or registered(R) trademarks of their respective holders, and use of them does not imply any affiliation or endorsement. This software is licensed to you subject to one or more open source licenses and VMware provides the software on an AS-IS basis. MinIO(R) is a registered trademark of the MinIO Inc. in the US and other countries. Bitnami is not affiliated, associated, authorized, endorsed by, or in any way officially connected with MinIO Inc. MinIO(R) is licensed under GNU AGPL v3.0.

TL;DR

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

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 Bitnami Object Storage based on MinIO® in production? Try VMware Tanzu Application Catalog, the commercial edition of the Bitnami catalog.

How to deploy MinIO(R) 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 MinIO(R) Chart GitHub repository.

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.

Only the latest stable branch maintained in the free Bitnami catalog

Starting December 10th, 2024, only the latest stable branch of each container image will receive updates in the free Bitnami catalog. To access up-to-date releases for all upstream-supported branches (e.g., LTS), consider upgrading to Bitnami Premium. Previously released versions will not be deleted and will remain available for pulling 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 MinIO(R) Docker Image is to pull the prebuilt image from the Docker Hub Registry.

```
docker pull bitnami/minio: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/minio:[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 .
```

Persisting your database

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/minio/data` path.

```
docker run --name minio \
    --publish 9000:9000 \
    --publish 9001:9001 \
    --volume /path/to/minio-persistence:/bitnami/minio/data \
    bitnami/minio:latest
```

or by modifying the `docker-compose.yml` file present in this repository:

```
services:
  minio:
  ...
    volumes:
      - /path/to/minio-persistence:/bitnami/minio/data
  ...
```

You can also mount a volume to a custom path inside the container, provided that you run the container using the `MINIO_DATA_DIR` environment variable.

```
docker run --name minio \
    --publish 9000:9000 \
    --publish 9001:9001 \
    --volume /path/to/minio-persistence:/custom/path/within/container \
    --env MINIO_DATA_DIR=/custom/path/within/container \
    bitnami/minio:latest
```

or by modifying the `docker-compose.yml` file present in this repository:

```
services:
  minio:
  ...
    volumes:
      - /path/to/minio-persistence:/custom/path/within/container
  ...
    environment:
      - MINIO_DATA_DIR=/custom/path/within/container
  ...
```

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

Connecting to other containers

Using Docker container networking, a MinIO(R) server running inside a container can easily be accessed by your application containers.

Containers attached to the same network can communicate with each other using the container name as the hostname.

Using the Command Line

In this example, we will create a MinIO(R) client container that will connect to the server container that is running on the same docker network as the client.

Step 1: Create a network

```
docker network create app-tier --driver bridge
```

Step 2: Launch the MinIO(R) server container

Use the `--network app-tier` argument to the `docker run` command to attach the MinIO(R) container to the `app-tier` network.

```
docker run -d --name minio-server \
    --env MINIO_ROOT_USER="minio-root-user" \
    --env MINIO_ROOT_PASSWORD="minio-root-password" \
    --network app-tier \
    bitnami/minio:latest
```

Step 3: Launch your MinIO(R) Client container

Finally we create a new container instance to launch the MinIO(R) client and connect to the server created in the previous step. In this example, we create a new bucket in the MinIO(R) storage server:

```
docker run -it --rm --name minio-client \
    --env MINIO_SERVER_HOST="minio-server" \
    --env MINIO_SERVER_ACCESS_KEY="minio-access-key" \
    --env MINIO_SERVER_SECRET_KEY="minio-secret-key" \
    --network app-tier \
    bitnami/minio-client \
    mb minio/my-bucket
```

Using a Docker Compose file

When not specified, Docker Compose automatically sets up a new network and attaches all deployed services to that network. However, we will explicitly define a new `bridge` network named `app-tier`. In this example we assume that you want to connect to the MinIO(R) server from your own custom application image which is identified in the following snippet by the service name `myapp`.

```
version: '2'

networks:
  app-tier:
    driver: bridge

services:
  minio:
```

```
    image: bitnami/minio:latest
    ports:
      - 9000:9000
      - 9001:9001
    environment:
      - MINIO_ROOT_USER=minio-root-user
      - MINIO_ROOT_PASSWORD=minio-root-password
    networks:
      - app-tier
  myapp:
    image: YOUR_APPLICATION_IMAGE
    networks:
      - app-tier
    environment:
      - MINIO_SERVER_ACCESS_KEY=minio-access-key
      - MINIO_SERVER_SECRET_KEY=minio-secret-key
```

> **IMPORTANT**:
>
> 1. Please update the **YOUR_APPLICATION_IMAGE_** placeholder in the above snippet with your application image
> 2. In your application container, use the hostname `minio` to connect to the MinIO(R) server. Use the environment variables `MINIO_SERVER_ACCESS_KEY` and `MINIO_SERVER_SECRET_KEY` to configure the credentials to access the MinIO(R) server.
> 3. Make sure that the environment variables `MINIO_ROOT_PASSWORD` and `MINIO_SERVER_SECRET_KEY` meet the 8 character minimum length requirement enforced by MinIO(R).

Launch the containers using:

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

Configuration

Environment variables

Customizable environment variables

Name	Description	Default Value

`MINIO_DATA_DIR`	MinIO directory for data.	`/bitnami/minio/data`
`MINIO_API_PORT_NUMBER`	MinIO API port number.	`9000`
`MINIO_BROWSER`	Enable / disable the embedded MinIO Console.	`off`
`MINIO_CONSOLE_PORT_NUMBER`	MinIO Console port number.	`9001`
`MINIO_SCHEME`	MinIO web scheme.	`http`
`MINIO_SKIP_CLIENT`	Skip MinIO client configuration.	`no`
`MINIO_DISTRIBUTED_MODE_ENABLED`	Enable MinIO distributed mode.	`no`
`MINIO_DEFAULT_BUCKETS`	MinIO default buckets.	`nil`
`MINIO_STARTUP_TIMEOUT`	MinIO startup timeout.	`10`
`MINIO_SERVER_URL`	MinIO server external URL.	`$MINIO_SCHEME://localhost:$MINIO_API_PORT_NUMBER`
`MINIO_APACHE_CONSOLE_HTTP_PORT_NUMBER`	MinIO Console UI HTTP port, exposed via Apache with basic authentication.	`80`
`MINIO_APACHE_CONSOLE_HTTPS_PORT_NUMBER`	MinIO Console UI HTTPS port, exposed via Apache with basic authentication.	`443`
`MINIO_APACHE_API_HTTP_PORT_NUMBER`	MinIO API HTTP port, exposed via Apache with basic authentication.	`9000`
`MINIO_APACHE_API_HTTPS_PORT_NUMBER`	MinIO API HTTPS port, exposed via Apache with basic authentication.	`9443`
`MINIO_FORCE_NEW_KEYS`	Force recreating MinIO keys.	`no`
`MINIO_ROOT_USER`	MinIO root user name.	`minio`
`MINIO_ROOT_PASSWORD`	Password for MinIO root user.	`miniosecret`

Read-only environment variables

Name	Description	Value
`MINIO_BASE_DIR`	MinIO installation directory.	`${BITNAMI_ROOT_DIR}/minio`
`MINIO_BIN_DIR`	MinIO directory for binaries.	`${MINIO_BASE_DIR}/bin`
`MINIO_CERTS_DIR`	MinIO directory for TLS certificates.	`/certs`
`MINIO_LOGS_DIR`	MinIO directory for log files.	`${MINIO_BASE_DIR}/log`
`MINIO_TMP_DIR`	MinIO directory for log files.	`${MINIO_BASE_DIR}/tmp`
`MINIO_SECRETS_DIR`	MinIO directory for credentials.	`${MINIO_BASE_DIR}/secrets`
`MINIO_LOG_FILE`	MinIO log file.	`${MINIO_LOGS_DIR}/minio.log`
`MINIO_PID_FILE`	MinIO PID file.	`${MINIO_TMP_DIR}/minio.pid`
`MINIO_DAEMON_USER`	MinIO system user.	`minio`

Name	Description	Value
`MINIO_DAEMON_GROUP`	MinIO system group.	`minio`

Additionally, MinIO can be configured via environment variables as detailed at MinIO(R) documentation.

A MinIO(R) Client (`mc`) is also shipped on this image that can be used to perform administrative tasks as described at the MinIO(R) Client documentation. In the example below, the client is used to obtain the server info:

```
docker run --name minio -d bitnami/minio:latest
docker exec minio mc admin info local
```

or using Docker Compose:

```
curl -sSL https://raw.githubusercontent.com/bitnami/containers/main/bitnami/minio/docker-compose.yml >
docker-compose.yml
docker-compose up -d
docker-compose exec minio mc admin info local
```

Creating default buckets

You can create a series of buckets in the MinIO(R) server during the initialization of the container by setting the environment variable `MINIO_DEFAULT_BUCKETS` as shown below (policy is optional):

```
docker run --name minio \
    --publish 9000:9000 \
    --publish 9001:9001 \
    --env MINIO_DEFAULT_BUCKETS='my-first-bucket:policy,my-second-bucket' \
    bitnami/minio:latest
```

or by modifying the `docker-compose.yml` file present in this repository:

```
services:
  minio:
  ...
    environment:
      - MINIO_DEFAULT_BUCKETS=my-first-bucket:policy,my-second-bucket
  ...
```

Securing access to MinIO(R) server with TLS

You can secure the access to MinIO(R) server with TLS as detailed at MinIO(R) documentation.

This image expects the variable `MINIO_SCHEME` set to `https` and certificates to be mounted at the `/certs` directory. You can put your key and certificate files on a local directory and mount it in the container as shown below:

```
docker run --name minio \
    --publish 9000:9000 \
    --publish 9001:9001 \
    --volume /path/to/certs:/certs \
    --env MINIO_SCHEME=https
    bitnami/minio:latest
```

or by modifying the `docker-compose.yml` file present in this repository:

```
services:
  minio:
  ...
    environment:
    ...
      - MINIO_SCHEME=https
    ...
    volumes:
      - /path/to/certs:/certs
  ...
```

Setting up MinIO(R) in Distributed Mode

You can configure MinIO(R) in Distributed Mode to setup a highly-available storage system. To do so, the environment variables below **must** be set on each node:

- `MINIO_DISTRIBUTED_MODE_ENABLED`: Set it to 'yes' to enable Distributed Mode.
- `MINIO_DISTRIBUTED_NODES`: List of MinIO(R) nodes hosts. Available separators are ' ', ',' and ';'.
- `MINIO_ROOT_USER`: MinIO(R) server root user. Must be common on every node.
- `MINIO_ROOT_PASSWORD`: MinIO(R) server root password. Must be common on every node.

You can use the Docker Compose below to create an 4-node distributed MinIO(R) setup:

```
version: '2'


services:
```

```
  minio1:
    image: bitnami/minio:latest
    environment:
      - MINIO_ROOT_USER=minio-root-user
      - MINIO_ROOT_PASSWORD=minio-root-password
      - MINIO_DISTRIBUTED_MODE_ENABLED=yes
      - MINIO_DISTRIBUTED_NODES=minio1,minio2,minio3,minio4
      - MINIO_SKIP_CLIENT=yes
  minio2:
    image: bitnami/minio:latest
    environment:
      - MINIO_ROOT_USER=minio-root-user
      - MINIO_ROOT_PASSWORD=minio-root-password
      - MINIO_DISTRIBUTED_MODE_ENABLED=yes
      - MINIO_DISTRIBUTED_NODES=minio1,minio2,minio3,minio4
      - MINIO_SKIP_CLIENT=yes
  minio3:
    image: bitnami/minio:latest
    environment:
      - MINIO_ROOT_USER=minio-root-user
      - MINIO_ROOT_PASSWORD=minio-root-password
      - MINIO_DISTRIBUTED_MODE_ENABLED=yes
      - MINIO_DISTRIBUTED_NODES=minio1,minio2,minio3,minio4
      - MINIO_SKIP_CLIENT=yes
  minio4:
    image: bitnami/minio:latest
    environment:
      - MINIO_ROOT_USER=minio-root-user
      - MINIO_ROOT_PASSWORD=minio-root-password
      - MINIO_DISTRIBUTED_MODE_ENABLED=yes
      - MINIO_DISTRIBUTED_NODES=minio1,minio2,minio3,minio4
      - MINIO_SKIP_CLIENT=yes
```

MinIO(R) also supports ellipsis syntax (`{1..n}`) to list the MinIO(R) node hosts, where `n` is the number of nodes. This syntax is also valid to use multiple drives (`{1..m}`) on each MinIO(R) node, where `n` is the number of drives per node. You can use the Docker Compose below to create an 2-node distributed MinIO(R) setup with 2 drives per node:

```
version: '2'
services:
```

```
  minio-0:
    image: bitnami/minio:latest
    volumes:
      - minio_0_data_0:/bitnami/minio/data-0
      - minio_0_data_1:/bitnami/minio/data-1
    environment:
      - MINIO_ROOT_USER=minio
      - MINIO_ROOT_PASSWORD=miniosecret
      - MINIO_DISTRIBUTED_MODE_ENABLED=yes
      - MINIO_DISTRIBUTED_NODES=minio-{0...1}/bitnami/minio/data-{0...1}
  minio-1:
    image: bitnami/minio:latest
    volumes:
      - minio_1_data_0:/bitnami/minio/data-0
      - minio_1_data_1:/bitnami/minio/data-1
    environment:
      - MINIO_ROOT_USER=minio
      - MINIO_ROOT_PASSWORD=miniosecret
      - MINIO_DISTRIBUTED_MODE_ENABLED=yes
      - MINIO_DISTRIBUTED_NODES=minio-{0...1}/bitnami/minio/data-{0...1}
volumes:
  minio_0_data_0:
    driver: local
  minio_0_data_1:
    driver: local
  minio_1_data_0:
    driver: local
  minio_1_data_1:
    driver: local
```

Find more information about the Distributed Mode in the MinIO(R) documentation.

Reconfiguring Keys on container restarts

MinIO(R) configures the access & secret key during the 1st initialization based on the `MINIO_ROOT_USER` and `MINIO_ROOT_PASSWORD` environment variables, respetively.

When using persistence, MinIO(R) will reuse the data configured during the 1st initialization by default, ignoring whatever values are set on these environment variables. You can force MinIO(R) to reconfigure the keys based on the environment variables by setting the `MINIO_FORCE_NEW_KEYS` environment variable to `yes`:

```
docker run --name minio \
    --publish 9000:9000 \
    --publish 9001:9001 \
    --env MINIO_FORCE_NEW_KEYS="yes" \
    --env MINIO_ROOT_USER="new-minio-root-user" \
    --env MINIO_ROOT_PASSWORD="new-minio-root-password" \
    --volume /path/to/minio-persistence:/bitnami/minio/data \
    bitnami/minio:latest
```

Logging

The Bitnami MinIO(R) Docker image sends the container logs to the `stdout`. To view the logs:

```
docker logs minio
```

or using Docker Compose:

```
docker-compose logs minio
```

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.

HTTP log trace

To enable HTTP log trace, you can set the environment variable `MINIO_HTTP_TRACE` to redirect the logs to a specific file as detailed at MinIO(R) documentation.

When setting this environment variable to `/opt/bitnami/minio/log/minio.log`, the logs will be sent to the `stdout`.

```
docker run --name minio \
    --publish 9000:9000 \
    --publish 9001:9001 \
    --env MINIO_HTTP_TRACE=/opt/bitnami/minio/log/minio.log \
    bitnami/minio:latest
```

or by modifying the `docker-compose.yml` file present in this repository:

```
services:
  minio:
  ...
```

```
  environment:
    - MINIO_HTTP_TRACE=/opt/bitnami/minio/log/minio.log
...
```

Maintenance

Upgrade this image

Bitnami provides up-to-date versions of MinIO(R), including security patches, soon after they are made upstream. We recommend that you follow these steps to upgrade your container.

Step 1: Get the updated image

```
docker pull bitnami/minio:latest
```

or if you're using Docker Compose, update the value of the image property to `bitnami/minio:latest`.

Step 2: Stop and backup the currently running container

Stop the currently running container using the command

```
docker stop minio
```

or using Docker Compose:

```
docker-compose stop minio
```

Next, take a snapshot of the persistent volume `/path/to/minio-persistence` using:

```
rsync -a /path/to/minio-persistence /path/to/minio-persistence.bkp.$(date +%Y%m%d-%H.%M.%S)
```

Step 3: Remove the currently running container

```
docker rm -v minio
```

or using Docker Compose:

```
docker-compose rm -v minio
```

Step 4: Run the new image

Re-create your container from the new image.

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

or using Docker Compose:

```
docker-compose up minio
```

Using `docker-compose.yaml`

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.

Contributing

We'd love for you to contribute to this Docker image. 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 include the following information in your issue:

- Host OS and version
- Docker version (`docker version`)
- Output of `docker info`
- Version of this contain

Note: the README for this container is longer than the DockerHub length limit of 25000, so it has been trimmed. The full README can be found at

https://github.com/bitnami/containers/blob/main/bitnami/minio/README.md

---

Revision #1
Created 15 June 2025 20:55:05 by Administrador
Updated 15 June 2025 20:59:12 by Administrador