// Tutorial //

How to Automatically Update Docker Container Images with Watchtower on Ubuntu 22.04

Published on May 24, 2022

Docker Container Ubuntu 22.04

By Tony Tran

Introduction

Every Docker container requires a Docker image as its base. Docker images may exist locally or be pulled remotely from Docker repositories. These repositories can store every iteration of a Docker image over time, as long as they share the same name and are differentiated by tags. Docker repositories are subsequently stored in a Docker registry, which holds collections of repositories. Docker Hub is the most prominent registry, being the official registry hosted by the Docker team.

Base Docker images are updated throughout the development process, but by default these updates must be manually pulled and applied to each running container. Watchtower automates this process of detecting updates for your Docker container's base image. Specifically, it watches a specified Docker image repository for new Docker image pushes. This repository can be either public or private, and hosted on the Docker Hub registry or your own image registry.

By pushing a new Docker image to your repository, Watchtower will automatically trigger a chain of events to update your running container's base Docker image. When Watchtower detects a new push, it

will pull the new base image, gracefully shutdown your running container, and start it back up. Your restarted container will be in the same state it initialized with, but with the new base Docker image applied.

In this tutorial, you will use Watchtower with both Docker's `run` command and Docker Compose to automatically update a Docker image. Both methods are functionally the same in creating a container running Watchtower, then pointing it towards a container you wish to keep automatically updated. Additionally, a monitor-only solution coupled with email notifications is given for situations where automatic updates are not an option.

Prerequisites

To complete this tutorial, you will need:

- An Ubuntu 22.04 server, set up according to our initial server setup guide for Ubuntu 22.04, with a non-**root** user with `sudo` privileges and a firewall enabled.
- Docker installed on your server, following **Steps 1 and 2** of how to install and use Docker on Ubuntu 22.04.
    - If you plan on doing a test update with your own custom Docker image on Docker Hub, you will need a Docker Hub account. Be familiar with pushing changes to a Docker Hub repository, and create a custom `ubuntu-nodejs` Docker image in your Docker Hub by following **Steps 5 through 8** of how to install and use Docker on Ubuntu 22.04.

- Docker Compose installed on your server, following **Step 1** of how to install and use Docker Compose on Ubuntu 22.04.
- If you plan to set up monitor-only notifications through Gmail, you will need a Gmail account with 2-Step Verification enabled, then you will have to create an application-specific App Password for Gmail. If you are not familiar with this process, check out how to use Google's SMTP server.

Step 1 — Watching an Externally Maintained Docker Image Using Docker's Command

Before you start using Watchtower, you need a target container for it to watch. This target container can be a custom image from your own repository, or a publicly available image maintained by a third party. Watching a third party's Docker image using Watchtower will provide you with automatic updates whenever a new update is available, however you are beholden to the third party's update schedule.

First, create a container using the official base Docker image, which is externally maintained by Canonical. Docker containers only exist as long as a process is running within it, otherwise it terminates itself to save resources. To bypass this for testing purposes, the example commands throughout this tutorial include `sleep infinity` to keep containers alive and ready to receive updates.

Usually a `docker run` call will take over your terminal and prevent further command input while the container is active. To avoid this, include the `-d` flag to run the container in detached mode. Create your container with `ubuntu-container` as the container name:

```
docker run -d \
  --name ubuntu-container \
  ubuntu \
```

Copy

```
  sleep infinity
```

Now that you have a container to watch for base image updates, you can start up Watchtower. Watchtower does not require a conventional installation through `apt`. Like the Docker containers it watches, you install and run Watchtower within a Docker container.

Use a `docker run` command similar to the previous one, but with the official Watchtower image which is named `containrrr/watchtower`. All Watchtower calls must include the `-v` flag for mounting `docker.sock` as the connection from within your Docker container to the server it is running on. This allows Watchtower to interact with the Docker API in order to monitor your running containers.

Watchtower automatically detects the base image of the containers it watches. Start your `watchtower` container and pass `ubuntu-container` as the container name to watch for updates:

```
docker run -d \
  --name watchtower \
  -v /var/run/docker.sock:/var/run/docker.sock \
  containrrr/watchtower \
  ubuntu-container
```

Copy

Docker will pull the `containrrr/watchtower` image from Docker Hub after failing to find it locally on your server. It then creates and starts the container, returning output similar to this:

```
Output
Unable to find image 'containrrr/watchtower:latest' locally
latest: Pulling from containrrr/watchtower
1045b2f97fda: Pull complete
35a104a262d3: Pull complete
1a0671483169: Pull complete
Digest: sha256:bbf9794a691b59ed2ed3089fec53844f14ada249ee5e372ff0e595b73f4e9ab3
Status: Downloaded newer image for containrrr/watchtower:latest
b6d1b765b2b8480357f246d3bcc3f422ff492b3deabb84cb0ab2909e2d63b9d3
```

Check that both your containers are running:

```
docker ps
```

Copy

Your output may be slightly different, but this output will list both containers:

```
Output
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS
```

```
b6d1b765b2b8   containrrr/watchtower   "/watchtower custom_…"   2 minutes ago   Up 2 minutes
56ac4341d662   ubuntu                  "sleep infinity"         6 minutes ago   Up 5 minutes
```

When an update is pushed to the official `ubuntu` Docker image repository by the Canonical team, your `watchtower` container will automatically detects the change, and initiates a graceful shutdown of `ubuntu-container`. `watchtower` then updates the base image of `ubuntu-container`, then starts it back up. By default, `watchtower` checks for these updates every 24 hours.

If you would like to verify this update process in real-time, you must do a test push with an update to your own custom image on Docker Hub, which will be demonstrated in **Step 4**.

To stop using Watchtower, stop your `watchtower` container with Docker's `stop` command. The `watchtower` container is the same as any other Docker container, and all standard Docker commands are applicable. Stop both of your containers with this command:

```
docker stop watchtower ubuntu-container
```

Copy

Remove the containers completely by using the Docker's `rm` command after stopping them:

```
docker rm watchtower ubuntu-container
```

Copy

With this you can now start, stop, and remove Watchtower according to your needs.

Step 2 — Setting Up Watchtower in a Docker Compose File

Containers for Watchtower can be created with Docker's `run` command and through Docker Compose. Translating the previous commands into a Docker Compose file provides a maintainable, centralized location for your Watchtower settings.

> **Note:** The complete Docker Compose file written in the course of this tutorial is given in **Step 5** for your reference.

First, make a directory for your Watchtower project and then navigate into it:

```
mkdir ~/watchtower
cd ~/watchtower
```

Copy

Create a new YAML file named `docker-compose.yml` using `nano` or your preferred text editor:

```
nano docker-compose.yml
```

Copy

Define the same `ubuntu` image as before, this time in Docker Compose format using `services`. The Docker image, container name, and given command will be exactly the same as those from the previous step that you defined with the `run` command. Insert the following into `docker-compose.yml`:

docker-compose.yml

```
version: "3"
services:
  ubuntu:
    image: ubuntu
    container_name: ubuntu-container
    command: sleep infinity
```

Next, add a second service which will handle creating your `watchtower` container. Again, these settings are exact translations from the settings you defined in the `run` command in Step 1:

docker-compose.yml

```
. . .

  watchtower:
    image: containrrr/watchtower
    container_name: watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    command: ubuntu-container
```

Save and exit your file. If you used `nano`, you can do this by pressing `CTRL+O`, `ENTER`, then `CTRL+X`. Now you can start your containers using `docker compose up`. Add the `-d` flag to prevent Docker from taking over your terminal:

```
docker compose up -d
```

You now have the exact same setup from Step 1 now translated into Docker Compose. Next is adding watching multiple custom containers with Watchtower.

Step 3 — Watching Multiple Containers Including Custom Images

You currently have two services each running a container. The `watchtower` service runs a container named `watchtower` which keeps track of a single container named `ubuntu-container` managed by

the `ubuntu` service.

Assuming you followed **Steps 5 through 8** of the prerequisite How To Install and Use Docker on Ubuntu 22.04, you have a custom image named `ubuntu-nodejs` in your Docker Hub repository based on Docker's official `ubuntu` image, but modified to have Node.js installed. This custom image and repository will be the basis for testing a real-time push and automatic update with `watchtower`.

To prepare for this, create two custom containers named `test-container` and `edit-container` running the same `ubuntu-nodejs` Docker image. Open your `docker-compose.yml` file:

```
nano docker-compose.yml
```

Copy

Add the services that will create these containers, substituting `sammy` with your Docker Hub username:

docker-compose.yml

```
. . .

  custom-test:
    image:  sammy /ubuntu-nodejs
    container_name: test-container
    command: sleep infinity

  custom-edit:
    image:  sammy /ubuntu-nodejs
    container_name: edit-container
    command: sleep infinity

. . .
```

Copy

There are two ways for Watchtower to watch multiple containers. By default `watchtower` will watch all active containers if container names are not passed through a command. However, watching all containers is not always desirable. To watch only specific containers, you can list multiple container names separated by a space.

Add `test-container` to the list of containers tracked by `watchtower`, making sure to add a space:

docker-compose.yml

```
. . .

  watchtower:
    image: containrrr/watchtower
    container_name: watchtower
    volumes:
```

Copy

```
      - /var/run/docker.sock:/var/run/docker.sock
    command: ubuntu-container  test-container
```

Now `watchtower` tracks `ubuntu-container` and `test-container`, but not `edit-container`, since you explicitly defined a list that excludes it.

Additionally, by default Watchtower polls your repository for image updates every 86400 seconds, or 24 hours. Since your testing requires immediate feedback, lower this number to a 30 second polling interval.

docker-compose.yml

```
. . .                                                                    Copy

  watchtower:
    image: containrrr/watchtower
    container_name: watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    command:  --interval 30  ubuntu-container test-container
```

Save and close your file. Apply your changes by calling `docker compose up` again. This time, however, pass the `--force-recreate` flag to recreate your containers with your updated `docker-compose.yml` file:

```
docker compose up -d --force-recreate                                    Copy
```

Next, you can push an update to verify a real-time automatic update with Watchtower.

Step 4 — Performing a Test Update with a Custom Image on Docker Hub

To test the automatic update functionality, you will no longer directly manipulate `test-container`. Instead, you will be performing edits on the `edit-container` custom container, which uses the same `ubuntu-nodejs` Docker base image. Even if these edits are done on a completely separate container, pushing your edits to the `ubuntu-nodejs` repo that both containers share will trigger `watchtower` to automatically update `test-container`.

First, make a change within the container itself to verify that `watchtower` can successfully perform an update. Any change will do, but one that would be easily noticeable would be to create a test file. The following command does this with an `echo` command piped into a `tee` command to create a file named `test.txt` in your user's home directory. The only contents of this file is the line `This was updated`. For these commands to work, though, they must be run within the container.

To gain access inside `edit-container`, use Docker's `exec` command paired with the `-it` flag. `exec` requires an executable program to be passed instead of a raw command, so you must

pass `sh -c` to evoke a shell through which you pass your commands:

```
docker exec -it edit-container sh -c "echo 'This was updated' | tee ~/test.txt"
```

Copy

As a control for your test, check that this file does not exist in `test-container` before you commit changes to your Docker image repository. The following command is similar to the previous one, but it uses `cat` to try to read the file instead of `echo` and `tee` to create it:

```
docker exec -it test-container sh -c "cat ~/test.txt"
```

Copy

Since this file does not exist yet in `test-container`, this is outputted:

```
Output
cat: /root/test.txt: No such file or directory
```

Now that you have verified that `test-container` does not contain the test file, it's time to commit the change to your Docker repository. Committing the update from `edit-container` to the Docker image repository will trigger an update for `test-container` through `watchtower`, without your manual input.

Commit the change, substituting your Docker Hub username in place of `sammy`:

```
docker commit -m "added test file" -a " sammy " edit-container  sammy /ubuntu-nodejs
```

Copy

Next, log in to your Docker Hub account with your Docker Hub username, which will then prompt you to input your Docker Hub password:

```
docker login -u  sammy
```

Copy

Complete your commit:

```
docker push  sammy /ubuntu-nodejs
```

Copy

Your `ubuntu-nodejs` base Docker image that is used by both `test-container` and `edit-container` is now updated. Since you only edited `edit-container` directly, it's time to verify whether `watchtower` has automatically updated `test-container` with those same changes.

Keep in mind that your `watchtower` instance is currently set to have a polling time of 30 seconds. The update and restart itself will take time, so wait approximately one minute for the update to complete.

After sufficient time has passed, run the same command as before to check for the existence of your test file in `test-container`:

```
docker exec -it test-container sh -c "cat ~/test.txt"
```

Copy

This time there won't be an error :

```
Output
This was updated
```

After creating `test-container`, you never manually edited its contents. Instead, `watchtower` has completed an automatic update based on your updates to the base image of `ubuntu-nodejs` which was edited through `edit-container`.

Step 5 — Enabling Monitor-Only Mode with Email Notifications (Optional)

There are situations where fully automatic base image updates are not desirable. Every update requires a container restart that is inherently disruptive. Instead, you can use Watchtower to only watch for changes, without the automatic shutdown and application of updates. You will only get an email notification that updates are available, then you choose when to manually apply those updates.

Open your `docker-compose.yml` file:

```
nano docker-compose.yml
```

Copy

Set up monitoring by adding the `WATCHTOWER_MONITOR_ONLY` setting, which is an environment variable used by Watchtower:

docker-compose.yml

```
. . .

  watchtower:
    image: containrrr/watchtower
    container_name: watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
     environment:
       WATCHTOWER_MONITOR_ONLY: 'true'
    command: --interval 30 ubuntu-container edit-container
```

Copy

Whenever the base image for `ubuntu-container` or `edit-container` has a new update available, Watchtower will send you a notification. This requires setting up Watchtower notifications, which can come in multiple formats including email, Slack, and Microsoft Teams. Here you will set up email notifications using Gmail as an SMTP server.

> **Note:** While sending email through the Gmail SMTP server is a well documented use case, you should review the process and security implications outlined in how to use Google's SMTP server. It is recommended that you use a new, separate Gmail account for these purposes. Alternatively, you can install and configure Postfix as a send-only SMTP server.

Open your `docker-compose.yml` file:

```
nano docker-compose.yml
```

Copy

To enable email notification functionality for your `watchtower` container, you need to add several environmental variables:

- `WATCHTOWER_NOTIFICATION_EMAIL_FROM` : the email address the emails will be sent from.
- `WATCHTOWER_NOTIFICATION_EMAIL_TO` : the email address to receive the notification emails.
- `WATCHTOWER_NOTIFICATION_EMAIL_SERVER` : the SMTP server. In this case, Gmail is used.
- `WATCHTOWER_NOTIFICATION_EMAIL_SERVER_PORT` : the port used to connect to the SMTP server to send the email. Port `587` is used for TLS encryption.
- `WATCHTOWER_NOTIFICATION_EMAIL_SERVER_USER` : the username for your Gmail credentials.
- `WATCHTOWER_NOTIFICATION_EMAIL_SERVER_PASSWORD` : the App Password for your Gmail credentials.

docker-compose.yml

```
. . .

    environment:
      WATCHTOWER_MONITOR_ONLY: 'true'
      WATCHTOWER_NOTIFICATIONS: email
      WATCHTOWER_NOTIFICATION_EMAIL_FROM:  your_gmail
      WATCHTOWER_NOTIFICATION_EMAIL_TO:  recipient_email
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER: smtp.gmail.com
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER_PORT: 587
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER_USER:  your_gmail
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER_PASSWORD:  gmail_app_password

. . .
```

Copy

Save and close your file. Your completed `docker-compose.yml` file will contain:

docker-compose.yml

```
version: "3"
services:
  ubuntu:
    image: ubuntu
    container_name: ubuntu-container
    command: sleep infinity

  custom-test:
    image: ubuntu-nodejs
    container_name: test-container
    command: sleep infinity

  custom-edit:
    image: ubuntu-nodejs
    container_name: edit-container
    command: sleep infinity

  watchtower:
    image: containrrr/watchtower
    container_name: watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      WATCHTOWER_MONITOR_ONLY: 'true'
      WATCHTOWER_NOTIFICATIONS: email
      WATCHTOWER_NOTIFICATION_EMAIL_FROM:  your_gmail
      WATCHTOWER_NOTIFICATION_EMAIL_TO:  recipient_email
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER: smtp.gmail.com
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER_PORT: 587
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER_USER:  your_gmail
      WATCHTOWER_NOTIFICATION_EMAIL_SERVER_PASSWORD:  gmail_app_password
    command: --interval 30 ubuntu-container test-container edit-container
```

Copy

Next, recreate your containers to apply your changes:

```
docker compose up -d --force-recreate
```

Copy

To test this, repeat the commands from the previous step. Instead of making an actual change, you can commit your current image as new:

```
docker commit -m "testing notifications" -a " sammy " edit-container  sammy /ubuntu-n
```

Copy

Push your changes to the repository:

```
docker push  sammy /ubuntu-nodejs
```

Copy

The email notification will not be immediate, and may take a few minutes depending on the email server's traffic conditions. You will receive an email similar to this one:

Watchtower updates on a41c05df8cba

```
Watchtower 1.4.0
Using notifications: smtp
Only checking containers with name "test-container"
Scheduling first run: 2022-05-10 12:32:08 +0000 UTC
Note that the first check will be performed in 23 hours, 59 minutes, 59 seconds
```

The email notification will indicate an update is available for you to manually apply. You can initiate a manual update with your `watchtower` container with the `--rm` and `--run-once` flags.
Normally, `watchtower` is constantly running as a daemon, watching the containers you assign it. You can tell Watchtower to instead start up, apply updates to the target container, then remove itself.

Here's an example for starting the `watchtower` container, but have it run only once:

```
docker run --rm\
  -v /var/run/docker.sock:/var/run/docker.sock \
  containrrr/watchtower \
  --run-once \
  edit-container
```

Copy

Once your `watchtower` container is finished with its one-time update, it will remove itself.

Conclusion

Your containers are now set up to automatically update whenever you push a new Docker image to your image repository, or when an external maintainer updates an image you're watching. Watchtower runs in Docker containers that behave like any other Docker container, so you can use the same techniques.

To learn more about how to work with Docker containers, check out this guide on how to share data between Docker containers on Ubuntu 22.04. For more detailed information on how to remove Docker containers, check out this guide on how to remove Docker images, containers, and volumes.