Handout: Automatic Container Updates

Modern **CI/CD workflows** — short for Continuous Integration and Continuous Delivery/Deployment — automatically build, test, and publish new container images whenever you push code (e.g., via GitHub Actions). These images are typically stored in a repository on a remote container registry, such as Docker Hub.
But unless explicitly handled, your **production server** — where containers are running (e.g., on a virtual machine or cloud instance) — remains unaware of the new image and continues using the old one.

Step	Performed By	Where
Code is pushed	Developer / GitHub user	Developer machine / GitHub
CI builds and pushes image	GitHub Actions	GitHub → Docker Hub
Container is updated	🛑 Manually (default)	Production server

This final manual step introduces delays and adds operational overhead — for example, a security patch may be available in the image, but production containers won't benefit until someone logs into the server, pulls the new image, and restarts the container manually.

Watchtower is a lightweight container that monitors your running containers and automatically updates them when a new version of the image becomes available.

1. Using Watchtower to Automatically Update Containers

Watchtower is a third-party open-source tool developed by the community (not part of Docker itself). It runs as a **separate container** on your production server:

1. Periodically checks for updates (default: every 24h, configurable via `--interval`).
2. Compares the current image of each container with the latest version in the registry (e.g., Docker Hub) → Watchtower doesn't just look at the tag (e.g., `:latest`), but compares the **image digest** — a unique hash that reflects the actual image contents
3. If a newer image is found:
   - Stops and removes the outdated container
   - Pulls the updated image from the registry
   - Recreates and starts a new container using the same configuration

You can start Watchtower using the following command:

```
docker run -d \
  --name watchtower \
  -v /var/run/docker.sock:/var/run/docker.sock \
  containrrr/watchtower \
  --interval 900 \  # every 15 minutes
```

```
    <container-name>
```

- `-v /var/run/docker.sock:/var/run/docker.sock`: Grants Watchtower access to inspect, stop, remove, and recreate containers
- `<container-name>`: List one or more container names to monitor; omit this to monitor all running containers
- `--interval`: Controls how often Watchtower checks for updates (in seconds); default is `86400` (24h)

Watchtower performs a **stop–remove–start** cycle for updates. It does **not** support rolling updates or traffic handover. For zero-downtime upgrades or high-availability deployments, consider using an orchestrator like **Kubernetes**.

2. Using Watchtower with Private Docker Hub Repositories

If your images are stored in a private repository on Docker Hub, Watchtower needs access credentials to pull updates. This requires a Docker-compatible `config.json` file containing your Docker Hub username and a personal access token, encoded in Base64.

2.1. Create a Personal Access Token

Go to your Docker Hub account settings, navigate to **Personal access tokens**, and generate a new token. Copy it — it will only be shown once.

2.2. Encode Your Credentials

Use this command to encode your Docker Hub credentials in the required Base64 format:

```
    echo -n '<username>:<personal-access-token>' | base64
```

You'll get an output like: `dXNlcm5hbWU6dG9rZW4xMjM=`.

Base64 is not encryption — it's a reversible encoding used to represent credentials in a format Docker expects. Anyone with access to the encoded string can decode it, so treat it like a password and keep it private.

2.3. Create a `config.json` File

On your production server (or wherever Watchtower runs), create a `config.json` file like this:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "<base64-encoded-credentials>"
    }
  }
}
```

Example:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "dXNlcm5hbWU6dG9rZW4xMjM="
    }
  }
}
```

2.4. Mount the Auth File into Watchtower

Run Watchtower and mount the config file:

```
docker run -d \
  --name watchtower \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v "${PWD}/config.json":/config.json:ro \
  containrrr/watchtower \
  --interval 900
```

Adjust the path if your `config.json` is stored elsewhere.

3. Using Watchtower with Docker Compose and Labels

For convenience and portability, you can run Watchtower as part of your `compose.yaml` setup. This allows you to configure it alongside your other services and manage everything with `docker compose up`.
To avoid accidentally restarting every container on your system, you can enable **label-based filtering**. With this feature, Watchtower will only update containers that explicitly opt in via a special label.
Example:

```yaml
services:
  watchtower:
    image: containrrr/watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ./config.json:/config.json:ro
    environment:
      - WATCHTOWER_LABEL_ENABLE=true
      - WATCHTOWER_POLL_INTERVAL=15s

  frontend:
    image: mycompany/frontend:latest
    labels:
      - "com.centurylinklabs.watchtower.enable=true"
```

- `WATCHTOWER_LABEL_ENABLE=true`
  Enables label-based tracking. Only containers with the required label will be monitored and updated.
- `WATCHTOWER_POLL_INTERVAL=15s` Controls how often Watchtower checks for updates (in this case, every 15 seconds).
- `com.centurylinklabs.watchtower.enable=true`
  Add this label to any service you want Watchtower to watch and update.

Label-based tracking is the recommended approach in multi-service setups. It avoids unintended updates and gives you fine-grained control over what gets automatically refreshed.