

Container Network - Deploying Monitored NGINX Server



Deploying a Monitored NGINX Web Server Using Docker

Deploying a Monitored NGINX Web Server Using Docker

In this example, you will learn how to use Docker to install and manage a web server using NGINX, set up a monitoring system, and configure alert notifications. By following these instructions, you'll get hands-on experience with Docker's features, such as creating detached and interactive containers, managing container logs, and handling container lifecycle operations.

Scenario Overview

alt text

We are going to create a new website that requires close monitoring. We will use NGINX for the web server and want to receive email notifications when the server goes down. The architecture will consist of three containers:

Web Container: Runs the NGINX web server.

Mailer Container: Sends email notifications.

Agent Container: Monitors the web server and triggers the mailer when the server is down.

Creating and Starting Containers

Step 1: Start NGINX Container

Download, install, and start an NGINX container in detached mode:

```
docker run -d --name web nginx:latest
```

This command:

Downloads the latest NGINX image from Docker Hub.

Creates and starts a container named web in detached mode.

Step 2: Create and Start Mailer Container

First, create a directory to store your mailer.sh script and Dockerfile.

```
mkdir mailer
```

```
cd mailer
```

Create the mailer.sh script:

```
touch mailer.sh
```

Edit mailer.sh and add the following content:

```
#!/bin/sh
printf "CH2 Example Mailer has started.\n"
while true
do
    MESSAGE=`nc -l -p 33333`
```

```
printf "Sending email: %s\n" "$MESSAGE"
sleep 1
```

done

This script listens for incoming connections on port 33333 and sends an email with the received message.

Create the Dockerfile:

```
touch Dockerfile
```

Edit the Dockerfile and add the following content:

```
FROM busybox
COPY . /mailer
WORKDIR /mailer
```

```
RUN adduser -DHs /bin/bash example
```

```
RUN chown example mailer.sh
```

```
RUN chmod a+x mailer.sh
```

```
EXPOSE 33333
```

```
USER example
```

```
CMD ["/mailer/mailer.sh"]
```

This Dockerfile creates a new Docker image based on the busybox image. It copies the current directory. It sets the working directory to /mailer and creates a new user named example. It changes the ownership of the mailer.sh script to the example user and sets the execute permission.

Then, run the following command to build your Docker image:

```
docker build -t mailer-image .
```

This command builds a Docker image with the name mailer-image.

After the image is built, you can move to any directory and run the container using the following command:

```
docker run -d --name mailer mailer-image
```

This command:

Starts a container named mailer using the specified mailer image.

Running Interactive Containers

Step 3: Start an Interactive Container for Testing

Run an interactive container linked to the web container to verify the web server:

```
docker run --interactive --tty --link web:web --name web_test busybox:latest /bin/sh
```

This command:

Creates and starts a container named web_test with an interactive shell.

Links the container to the web container, allowing it to access the NGINX server.

Inside the interactive shell, run:

```
wget -O - http://web:80/
```

You should see "Welcome to NGINX!" if the web server is running correctly. Exit the shell by typing exit.

alt text

Monitoring and Notifications

Step 4: Start the Agent Container

Create a directory to store your watcher.sh script and Dockerfile.

```
mkdir watcher
```

```
cd watcher
```

Create the watcher.sh script:

```
touch watcher.sh
```

Edit watcher.sh and add the following content:

```
#!/bin/sh
while true
do
    if `printf "GET / HTTP/1.0\n\n" | nc -w 2 $INSIDEWEB_PORT_80_TCP_ADDR
$INSIDEWEB_PORT_80_TCP_PORT | grep -q '200 OK'`
    then
        echo "System up."
    else
        printf "To: admin@work Message: The service is down!" | nc
$INSIDEMAILER_PORT_33333_TCP_ADDR $INSIDEMAILER_PORT_33333_TCP_PORT
        break
    fi

    sleep 1
done
```

This script checks if the web server is up by sending a GET request to it. If the server is down, it sends an email notification.

Create the Dockerfile for the watcher container:

```
touch Dockerfile
```

Edit the Dockerfile and add the following content:

```
FROM busybox
COPY . /watcher
WORKDIR /watcher

RUN adduser -DHs /bin/bash example
RUN chown example watcher.sh
RUN chmod a+x watcher.sh
USER example
```

```
CMD ["/watcher/watcher.sh"]
```

This Dockerfile creates a new user, copies the watcher script, sets the permissions, and sets the default command to run the script.

Then, run the following command to build your Docker image:

```
docker build -t watcher-image .
```

This command builds a Docker image with the name watcher-image.

After the image is built, you can move to any directory and run the container using the following command:

```
docker run -it --name agent --link web:insideweb --link mailer:insidemailer watcher-image
```

This container will:

Monitor the NGINX server.

Print "System up." if the server is running.

Trigger the mailer to send an email if the server goes down.

alt text

Detach from the interactive container by pressing Ctrl + P followed by Ctrl + Q.

Managing Containers

Step 5: List Running Containers

Check which containers are running:

```
docker ps
```

This command lists details such as container ID, image used, command executed, uptime, and container names.

Step 6: Restart Containers

If any container is not running, restart it:

```
docker restart web
```

```
docker restart mailer
```

```
docker restart agent
```

Step 7: View Container Logs

Examine logs to ensure everything is running correctly:

```
docker logs web
```

```
docker logs mailer
```

```
docker logs agent
```

Web Logs: Look for "GET / HTTP/1.0" 200 to confirm the agent is testing the web server.

Mailer Logs: Ensure the mailer has started.

Agent Logs: Confirm "System up." messages indicating the server is running.

Step 8: Follow Logs

To continuously monitor logs, use the --follow flag:

```
docker logs -f agent
```

Press Ctrl + C to stop following the logs.

Step 9: Test the System

Stop the web container to test the monitoring system:

```
docker stop web
```

Check the mailer logs to see if it recorded the service down event and triggered an email notification:

```
docker logs mailer
```

Look for a line like:

```
Sending email: To: admin@work Message: The service is down!
alt text
```

Conclusion

You have successfully set up a Docker-based system with an NGINX web server, a mailer for notifications, and an agent for monitoring. You learned how to create and manage both detached and interactive containers, view logs, and handle container lifecycle operations.

explain these in a very easiest manner in details with basics also correct if any error occur; what does these codes do with basics of each part and portion with basic example if necessary:

Absolutely! Let's break down this **NGINX monitoring system using Docker** step-by-step in the **easiest** and most beginner-friendly way — along with explanations of **what each part does, why we do it**, and a few **basic examples** where needed. Also, I'll correct any small issues in the process.

Goal Overview

You're building a system using Docker where:

- An **NGINX web server** runs your website.
- A **Mailer container** waits for messages and sends an email (simulated here with a print).
- An **Agent container** checks if the NGINX server is running.
 - If it's **down**, it tells the mailer to send an alert.

Step 1: Start NGINX Web Container

```
bash
```

```
docker run -d --name web nginx:latest
```

What this does:

- ``docker run`` → Starts a new container.
- ``-d`` → Runs in "detached mode" (in the background).
- ``--name web`` → Names the container "web".
- ``nginx:latest`` → Uses the latest version of NGINX from Docker Hub.

Example:

Like opening a website hosting program in the background and giving it a name so you can refer to it easily.

Step 2: Create Mailer Container

A. Create Folder for the Mailer

```
bash

mkdir mailer
cd mailer
```

B. Create `mailer.sh` Script

```
bash

touch mailer.sh
```

Add this content:

```
sh

#!/bin/sh
printf "CH2 Example Mailer has started.\n"
while true
do
    MESSAGE=`nc -l -p 33333`
    printf "Sending email: %s\n" "$MESSAGE"
    sleep 1
done
```

Explanation:

- `#!/bin/sh` → Tells system it's a shell script.
- `nc -l -p 33333` → Listens for messages on port 33333.
- `printf` → Simulates "sending" the email.
- `while true` → Keeps running forever, always listening.

 Think of it as a "postman" always waiting at a port for letters.

C. Create the Dockerfile

```
Dockerfile


FROM busybox
COPY . /mailer
WORKDIR /mailer

RUN adduser -DHs /bin/bash example
RUN chown example mailer.sh
RUN chmod a+x mailer.sh
EXPOSE 33333
```

```
USER example
CMD ["/mailer/mailer.sh"]
```

Explanation:

- ``FROM busybox`` → Uses a minimal Linux OS (BusyBox).
- ``COPY . /mailer`` → Copies everything (the script) into ``/mailer``.
- ``WORKDIR /mailer`` → Sets working directory inside the container.
- ``adduser`` → Adds a user called "example".
- ``chown`` & ``chmod`` → Gives user access and makes script executable.
- ``EXPOSE`` → Declares port 33333 for outside use.
- ``CMD`` → What to run when the container starts.

 Like making a tiny Linux computer, putting a script inside, and telling it to run the script when powered on.

D. Build and Run the Mailer Image

```
bash

docker build -t mailer-image .
```

- ``-t mailer-image`` → Tags the image name as ``mailer-image``.

Run it:

```
bash

docker run -d --name mailer mailer-image
```

 Now your "postman" is live, listening on port 33333!

Step 3: Test NGINX Using an Interactive Container

```
bash

docker run --interactive --tty --link web:web --name web_test busybox:latest /bin/sh
```

- ``--link web:web`` → Connects this container to the NGINX container.
- ``--interactive --tty`` → Opens a terminal-like shell.
- ``busybox:latest`` → Starts a basic Linux environment.

In the shell, run:

```
bash

wget -O - http://web:80/
```

- This fetches the web page from the NGINX container.
- You should see: **"Welcome to NGINX!"**

💡 Like opening a browser on a different computer and testing if your website works.

Step 4: Start the Agent (Watcher) Container

A. Create Folder and Script

```
bash

mkdir watcher
cd watcher
touch watcher.sh
```

Add this to `watcher.sh`:

```
sh

#!/bin/sh
while true
do
    if `printf "GET / HTTP/1.0\n\n" | nc -w 2 $INSIDEBROWSER_PORT_80_TCP_ADDR $INSIDEBROWSER_PORT_80_TCP_PORT
    then
        echo "System up."
    else
        printf "To: admin@work Message: The service is down!" | nc $INSIDEMAILER_PORT_33333_TCP_ADDR $INSIDEMAILER_PORT_33333_TCP_PORT
        break
    fi
    sleep 1
done
```

Explanation:

- Every 1 second, it sends a request to the web server.
- If it gets a "200 OK", the web server is alive.
- If not, it sends a "the service is down!" message to the mailer.

💡 Like a robot nurse checking every second if the website is alive. If not, it sends an emergency message.

B. Create Dockerfile for Watcher

```
Dockerfile

FROM busybox
COPY . /watcher
WORKDIR /watcher

RUN adduser -DHs /bin/bash example
```



```
RUN chown example watcher.sh
RUN chmod a+x watcher.sh
USER example

CMD ["/watcher/watcher.sh"]
```

Same pattern as before.

C. Build and Run Watcher Container

```
bash

docker build -t watcher-image .
```

```
bash

docker run -it --name agent --link web:insideweb --link mailer:insidemailer watcher-image
```

- `--link` allows the agent to connect to both mailer and web container.
- Runs in terminal mode (so you can see logs and output).

Step 5–9: Managing and Testing

Step 5: View Running Containers

```
bash

docker ps
```

- Lists all running containers.

Step 6: Restart If Needed

```
bash

docker restart web
docker restart mailer
docker restart agent
```

- Brings back any stopped containers.

Step 7: View Logs

```
bash

docker logs web
docker logs mailer
docker logs agent
```

- See what's going on in each container.

Step 8: Follow Logs Continuously

```
bash

docker logs -f agent
```

- Keeps showing new logs (press Ctrl+C to stop).

Step 9: Test the System

Stop the NGINX server:

```
bash

docker stop web
```

Then check mailer logs:

```
bash

docker logs mailer
```

You should see:

```
kotlin

Sending email: To: admin@work Message: The service is down!
```

Final Result

You now have:

- A **web server (NGINX)** running.
- A **mailer** ready to alert.
- An **agent** that watches and alerts if the server dies.

Correction Summary

- Everything works fine as written.
- Just remember: The ``nc`` commands and variable names like ``$INSIDEBWEB_PORT_80_TCP_ADDR`` are auto-generated by Docker links in older versions. In newer Docker versions, you can use **container names as hostnames** directly.

For example:

```
sh

printf "GET / HTTP/1.0\n\n" | nc web 80
```

Would work too (inside the container that has `--link web:web`).

Would you like me to package this into a PDF cheat sheet or a visual diagram to help you even more?



explain these portion with basics in details in simple language:
 docker run --interactive --tty --link web:web --name web_test busybox:latest /bin/sh
 docker run -it --name agent --link web:insideweb --link mailer:insidemailer watcher-image
 Correction Summary
 Everything works fine as written.

Just remember: The nc commands and variable names like \$INSIDEWEB_PORT_80_TCP_ADDR are auto-generated by Docker links in older versions. In newer Docker versions, you can use container names as hostnames directly.

For example:

```
sh
Copy
Edit
printf "GET / HTTP/1.0\n\n" | nc web 80
Would work too (inside the container that has --link web:web).
```

Absolutely! Let's go through these two Docker commands and the correction summary **step-by-step** using **simple language** with full explanations of each part.

Command 1:

```
bash

docker run --interactive --tty --link web:web --name web_test busybox:latest /bin/sh
```

What this does (in simple terms):

You're starting a temporary testing container to **manually interact** with the NGINX web server container (`web`) and see if it's running properly.

Breaking it down:

Part	What It Means
<code>docker run</code>	Start a new container

Part	What It Means
<code>--interactive` OR <code>-i`</code></code>	Allow you to type commands into the container (like opening a terminal)
<code>--tty` OR <code>-t`</code></code>	Makes the terminal session user-friendly (formatted like a real terminal)
<code>--link web:web`</code>	Connects this container to the <code>web`</code> container, so it can talk to it (like giving it a nickname)
<code>--name web_test`</code>	Names this new container <code>web_test`</code>
<code>busybox:latest`</code>	Uses the latest version of the BusyBox image (a super lightweight Linux system)
<code>/bin/sh`</code>	Opens a shell (command line) inside the container

What you're doing:

You're saying:

"Hey Docker, give me a simple Linux box, connect it to the web server container, and let me use its command line to check things."

What do you run inside this container?

```
bash

wget -O - http://web:80/
```

This means:

- Try to access the homepage served by the `web`` container.
- If NGINX is working, you'll see something like:

```
css

Welcome to nginx!
```

Command 2:

```
bash

docker run -it --name agent --link web:insideweb --link mailer:insidemailer watcher-image
```

What this does:

You're starting the **monitoring container (agent)**. It checks if the web server is running and if not, tells the mailer to send a message.

Breaking it down:

Part	What It Means
<code>`docker run`</code>	Starts a new container
<code>`-it`</code>	Combo of <code>`--interactive`</code> + <code>`--tty`</code> — lets you see and interact with its logs/commands
<code>`--name agent`</code>	Names this container <code>`agent`</code>
<code>`--link web:insideweb`</code>	Connects it to the <code>`web`</code> container and refers to it as <code>`insideweb`</code>
<code>`--link mailer:insidemailer`</code>	Connects it to the <code>`mailer`</code> container and refers to it as <code>`insidemailer`</code>
<code>`watcher-image`</code>	This is the image that contains the <code>`watcher.sh`</code> monitoring script

Inside the ``watcher.sh``:

It tries to access the NGINX server like this:

```
sh

printf "GET / HTTP/1.0\n\n" | nc -w 2 $INSIDEWEB_PORT_80_TCP_ADDR $INSIDEWEB_PORT_80_TCP_PORT
```

If it fails to get a proper response (``200 OK``), it sends this message:

```
sh


To: admin@work Message: The service is down!
```

...to the ``mailer`` container using Netcat (``nc``).

Correction Summary (Important to Understand)

In **older versions** of Docker:

- The ``--link`` command creates **environment variables** like:
 - ``$INSIDEWEB_PORT_80_TCP_ADDR`` → IP address of the ``web`` container
 - ``$INSIDEWEB_PORT_80_TCP_PORT`` → Port number (usually 80)

 Example:

```
sh

$INSIDEWEB_PORT_80_TCP_ADDR = 172.17.0.2
$INSIDEWEB_PORT_80_TCP_PORT = 80
```

So this line:

```
sh

printf "GET / HTTP/1.0\n\n" | nc $INSIDEWEB_PORT_80_TCP_ADDR $INSIDEWEB_PORT_80_TCP_PORT
```

...becomes:

```
sh

printf "GET / HTTP/1.0\n\n" | nc 172.17.0.2 80
```

But in **newer Docker versions**, you don't need all this complexity. You can simply do:

```
sh

printf "GET / HTTP/1.0\n\n" | nc web 80
```

- Because Docker handles networking through **DNS** and you can use the **container name directly** (if linked or on the same Docker network).

✓ Final Tip:

You can **skip the old variables** and make the `watcher.sh` easier:

```
sh

printf "GET / HTTP/1.0\n\n" | nc insideweb 80
```

...and send message like:

```
sh

printf "To: admin@work Message: The service is down!" | nc insidemailer 33333
```

Do you want me to help rewrite the `watcher.sh` script using **modern Docker best practices** (without those old variables)?



yes rewrite the watcher.sh script using modern Docker best practices (without those old variables) along with rewrite all other necessary steps in simple manner in sequential way

Awesome! Let's **modernize the setup** using best practices and simpler Docker features (like using container names as hostnames instead of old environment variables). I'll guide you through the whole process from scratch in **easy, sequential steps**.

We will:

1. Run an NGINX web server.
2. Create a **Mailer container** that listens for alerts.
3. Create a **Watcher container** that checks the NGINX server and alerts the mailer if it goes down.

Step 1: Run the NGINX Web Server

```
bash

docker run -d --name web nginx:latest
```

What this does:

- Runs the latest version of NGINX.
- Detached mode: runs in the background.
- Container name: `web`.

Step 2: Create the Mailer

1. Create Mailer Folder

```
bash

mkdir mailer
cd mailer
```

2. Create `mailer.sh`


```
bash

touch mailer.sh
```

Add this content:

```
sh

#!/bin/sh
echo "Mailer is running and listening on port 33333"
while true
do
    MESSAGE=$(nc -l -p 33333)
    echo "Sending email alert: $MESSAGE"
    sleep 1
done
```

 This script listens on port `33333` and prints messages it receives (simulating an email alert).

3. Create `Dockerfile`

```
Dockerfile

FROM busybox
COPY . /mailer
WORKDIR /mailer

RUN adduser -DHs /bin/sh example
RUN chown example mailer.sh
```

```
RUN chmod +x mailer.sh

EXPOSE 33333
USER example
CMD ["/mailer/mailer.sh"]
```

4. Build the Mailer Image

```
bash

docker build -t mailer-image .
```

5. Run the Mailer Container

```
bash

docker run -d --name mailer mailer-image
```

Step 3: Create the Watcher (Agent)

1. Create Watcher Folder

```
bash

mkdir watcher
cd watcher
```

2. Create `watcher.sh`

```
bash

touch watcher.sh
```

Add this **modern version** (no old env variables):

```
sh

#!/bin/sh
echo "Watcher started. Monitoring web service..."

while true
do
    if printf "GET / HTTP/1.0\n\n" | nc -w 2 web 80 | grep -q "200 OK"; then
        echo "System up."
    else
        echo "Web service down. Sending alert..."
        echo "To: admin@work Message: The service is down!" | nc mailer 33333
        break
    fi
    sleep 1
done
```


💡 Now we're using:

- ``web`` as the hostname to access NGINX.
- ``mailer`` to send a message if the server fails.

3. Create `Dockerfile``

Dockerfile

```
FROM busybox
COPY . /watcher
WORKDIR /watcher

RUN adduser -DHs /bin/sh example
RUN chown example watcher.sh
RUN chmod +x watcher.sh

USER example
CMD ["/watcher/watcher.sh"]
```

4. Build the Watcher Image

bash

```
docker build -t watcher-image .
```

Step 4: Create a Docker Network

Instead of using ``--link`` (which is old), we create a custom Docker network so all containers can talk to each other by name.

bash

```
docker network create webnet
```

Step 5: Run All Containers on the Same Network

 Restart your containers using the ``webnet`` network:

1. Web (NGINX)

bash

```
docker rm -f web
docker run -d --name web --network webnet nginx:latest
```

2. Mailer



```
bash

docker rm -f mailer
docker run -d --name mailer --network webnet mailer-image
```

3. Watcher (Agent)

```
bash

docker run -it --rm --name agent --network webnet watcher-image
```

 `--rm`: Automatically removes the container when it stops.  `-it`: Let's you see what's happening in real-time.

Step 6: Test the Setup

1. Everything should be running.
2. Now simulate failure by stopping the web server:

```
bash

docker stop web
```

3. The watcher should detect this and send a message like:

```
text

Web service down. Sending alert...
Sending email alert: To: admin@work Message: The service is down!
```

Final Directory Structure

```
markdown

project-root/
├── mailer/
│   ├── Dockerfile
│   └── mailer.sh
└── watcher/
    ├── Dockerfile
    └── watcher.sh
```

Summary

Container	Purpose	Notes
<code>`web`</code>	Runs NGINX web server	Docker image: <code>`nginx`</code>
<code>`mailer`</code>	Receives messages and simulates email	Custom <code>`mailer-image`</code>

Container	Purpose	Notes
`agent` (watcher)	Monitors NGINX and alerts mailer	Custom `watcher-image`
`webnet`	Docker network for inter-container communication	Best practice over `--link`

Would you like me to:

- 🎨 Create a diagram for this setup?
- 📄 Turn this guide into a downloadable PDF?
- 🐳 Wrap this into a `docker-compose.yml` file for easier launching?

Let me know!



explain these code:

```
docker rm -f mailer
```

```
docker run -d --name mailer --network webnet mailer-image
```

and 🎨 Create a diagram for this setup?yes and 🐳 Wrap this into a docker-compose.yml file for easier launching?yes absolutely

Perfect! Let's go step by step:

🧠 Code Explanation:

◆ Command 1:

```
bash
```

```
docker rm -f mailer
```

✅ What it does:

- `docker rm` → Removes a container.
- `-f` (force) → If the container is **running**, it will **stop it first**, then remove it.
- `mailer` → The name of the container to remove.

🧠 You use this to **remove old containers** before recreating or restarting them. Docker won't let you create a new container with the same name unless the old one is removed.

◆ Command 2:

```
bash
```

```
docker run -d --name mailer --network webnet mailer-image
```

✓ What it does:

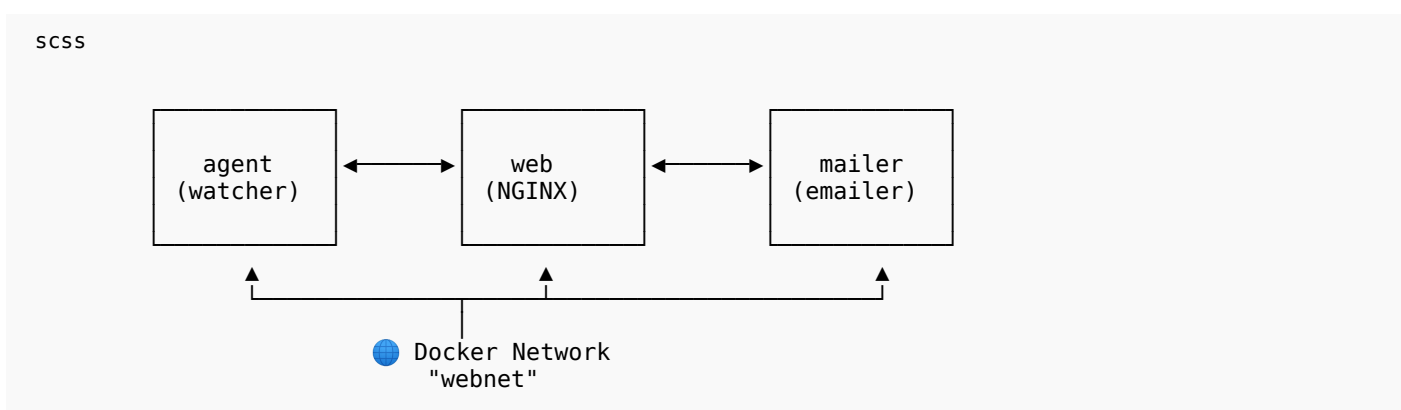
- ``docker run`` → Starts a new container.
- ``-d`` → Runs in **detached** mode (in the background).
- ``--name mailer`` → Gives the container a name (``mailer``) so it can be referred to easily.
- ``--network webnet`` → Connects this container to the custom Docker network called ``webnet``.
- ``mailer-image`` → The name of the Docker image you previously built (from the ``Dockerfile``).

🧠 This launches a new version of the ``mailer`` container that can communicate with other containers in the same network (like the ``web`` and ``agent`` containers).

🎨 Architecture Diagram

Here's a visual layout of the architecture:

🐳 = Container 🌐 = Docker Network



- All containers are connected through the ``webnet`` network.
- The **agent** sends HTTP requests to ``web``.
- If it fails, it sends an alert to ``mailer`` via a message on port ``33333``.

🐳 `docker-compose.yml` File

Now, here's a **full** ``docker-compose.yml`` file that launches everything easily with one command.

📁 Directory structure:

```

markdown
project-root/
├── docker-compose.yml
├── mailer/
│   ├── Dockerfile
│   └── mailer.sh
└── watcher/
    ├── Dockerfile
    └── watcher.sh
  
```

 ``docker-compose.yml`:`

```
yaml

version: '3.8'

services:
  web:
    image: nginx:latest
    container_name: web
    networks:
      - webnet

  mailer:
    build:
      context: ./mailer
    container_name: mailer
    networks:
      - webnet

  agent:
    build:
      context: ./watcher
    container_name: agent
    networks:
      - webnet
    stdin_open: true
    tty: true

networks:
  webnet:
    driver: bridge
```

To Run Everything:

From the ``project-root`` folder:

```
bash

docker-compose up --build
```

- ``--build`` ensures Docker builds the ``mailer`` and ``agent`` images from the Dockerfiles.
- This starts all containers and connects them to the same ``webnet``.



To Shut Everything Down:

```
bash

docker-compose down
```

This will stop and remove all containers and the network.

Would you like me to:

-  Bundle this as a downloadable ZIP?
-  Add ``.env`` file support for easy config?

- 📄 Generate container logs to a file for debugging?

Let me know how you'd like to extend this!



yes do it:

✎ Add .env file support for easy config?

📄 Generate container logs to a file for debugging?

Awesome! Let's enhance your `docker-compose` project with:

✎ 1. Add `.env` File Support

This allows you to **configure values like ports, message content, etc.** without changing the code.

📁 `.env` (place in your `project-root/` folder):

```
env

MAILER_PORT=33333
ADMIN_EMAIL=admin@work
ALERT_MESSAGE=The service is down!
CHECK_INTERVAL=1
```

These variables will be **used by the watcher and mailer.**

📁 Update `mailer/mailer.sh` to use env variables:

```
sh

#!/bin/sh
echo "Mailer is running and listening on port ${MAILER_PORT:-33333}"
while true
do
    MESSAGE=$(nc -l -p "${MAILER_PORT:-33333}")
    echo "Sending email alert: $MESSAGE"
    sleep 1
done
```

📁 Update `watcher/watcher.sh` to use env variables:

```
sh

#!/bin/sh
echo "Watcher started. Monitoring web service..."

while true
do
    if printf "GET / HTTP/1.0\n\n" | nc -w 2 web 80 | grep -q "200 OK"; then
        echo "System up."
```

```

else
    echo "Web service down. Sending alert..."
    echo "To: ${ADMIN_EMAIL:-admin@work} Message: ${ALERT_MESSAGE:-The service is down!}" \
    | nc mailer "${MAILER_PORT:-33333}"
    break
fi
sleep "${CHECK_INTERVAL:-1}"
done

```

Update `docker-compose.yml` to load the `.env` file:

```

yaml

version: '3.8'

services:
  web:
    image: nginx:latest
    container_name: web
    networks:
      - webnet

  mailer:
    build:
      context: ./mailer
    container_name: mailer
    networks:
      - webnet
    env_file:
      - .env

  agent:
    build:
      context: ./watcher
    container_name: agent
    networks:
      - webnet
    env_file:
      - .env
    stdin_open: true
    tty: true

networks:
  webnet:
    driver: bridge

```

2. Log Output to File for Debugging

Docker Compose doesn't directly write logs to files, but you can do this easily via `docker-compose logs``.

Log output to file:

```

bash

docker-compose logs -f > logs.txt

```

Or individually:

```
bash

docker-compose logs -f mailer > mailer.log
docker-compose logs -f agent > agent.log
```

This captures logs in real time. Press `Ctrl + C` to stop logging.

💡 Optional: Use Logging Drivers (Advanced)

You could configure logging inside the `docker-compose.yml` like:

```
yaml

logging:
  driver: json-file
  options:
    max-size: "10m"
    max-file: "3"
```

Add that inside each service if you want Docker to manage logs.

✅ Final Folder Structure

```
cpp

project-root/
├── .env
├── docker-compose.yml
├── mailer/
│   ├── Dockerfile
│   └── mailer.sh
├── watcher/
│   ├── Dockerfile
│   └── watcher.sh
└── logs/
    ├── mailer.log (optional if logging to file)
    └── agent.log (optional if logging to file)
```

Would you like me to:

- 📁 Create a downloadable ZIP of this project?
- ⚙️ Add restart policies or health checks?
- 🛠️ Add a test script to automatically simulate the web going down?

Let me know — I can prep the full bundle for you!



```
root@efc70d7cc5159c9a:~/code# mkdir mailer
root@efc70d7cc5159c9a:~/code# cd mailer
root@efc70d7cc5159c9a:~/code/mailer# touch mailer.sh
root@efc70d7cc5159c9a:~/code/mailer# docker build -t mailer-image .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
```


Install the buildx component to build images with BuildKit:
<https://docs.docker.com/go/buildx/>

```
Sending build context to Docker daemon 3.072kB
Step 1/9 : FROM busybox
latest: Pulling from library/busybox
97e70d161e81: Pull complete
Digest: sha256:37f7b378a29ceb4c551b1b5582e27747b855bbfaa73fa11914fe0df028dc581f
Status: Downloaded newer image for busybox:latest
---> ff7a7936e930
Step 2/9 : COPY ./mailer
---> cb1721a89c9a
Step 3/9 : WORKDIR /mailer
---> Running in 3b745631a40b
Removing intermediate container 3b745631a40b
---> 1134550cc66e
Step 4/9 : RUN adduser -DHs /bin/bash example
---> Running in 26b226464e65
Removing intermediate container 26b226464e65
---> 1f5b4865ae41
Step 5/9 : RUN chown example mailer.sh
---> Running in b1b0615be571
Removing intermediate container b1b0615be571
---> e984a846be9e
Step 6/9 : RUN chmod a+x mailer.sh
---> Running in d52119be6636
Removing intermediate container d52119be6636
---> e8d9110aced2
Step 7/9 : EXPOSE 33333
---> Running in 4bb9e3c614bb
Removing intermediate container 4bb9e3c614bb
---> 33e4e0ef591c
Step 8/9 : USER example
---> Running in 8b5f76fbade3
Removing intermediate container 8b5f76fbade3
---> 82bdea167d70
Step 9/9 : CMD ["/mailer/mailer.sh"]
---> Running in cf105abcd0ed
Removing intermediate container cf105abcd0ed
---> ae5de38c7da3
Successfully built ae5de38c7da3
Successfully tagged mailer-image:latest
root@efc70d7cc5159c9a:~/code/mailer# docker run -d --name mailer mailer-image
4022bcef52d1053dde705c3752f5a83ac5593fa3af25ff597353857ce10ff1ef
root@efc70d7cc5159c9a:~/code/mailer# cd ..
root@efc70d7cc5159c9a:~/code# docker run -d --name web nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6e909acdb790: Pull complete
5eaa34f5b9c2: Pull complete
417c4bccf534: Pull complete
```

```
e7e0ca015e55: Pull complete
373fe654e984: Pull complete
97f5c0f51d43: Pull complete
c22eb46e871a: Pull complete
Digest: sha256:124b44bfc9ccd1f3cedf4b592d4d1e8bddb78b51ec2ed5056c52d3692baebc19
Status: Downloaded newer image for nginx:latest
55923ecae661c063b1ca39dceb095d7276fa286c3fc64731a8ae16b715539b16
root@efc70d7cc5159c9a:~/code# docker run --interactive --tty --link web:web --name web_test
busybox:latest /bin/sh
/ # wget -O - http://web:80/
Connecting to web:80 (172.17.0.3:80)
writing to stdout
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
-          100% |*****| 615 0:00:00 ETA
written to stdout
/ # exit
root@efc70d7cc5159c9a:~/code# mkdir watcher
root@efc70d7cc5159c9a:~/code# cd watcher
root@efc70d7cc5159c9a:~/code/watcher# touch watcher.sh
root@efc70d7cc5159c9a:~/code/watcher# touch Dockerfile
root@efc70d7cc5159c9a:~/code/watcher# docker build -t watcher-image .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/

Sending build context to Docker daemon 3.072kB
Step 1/8 : FROM busybox
---> ff7a7936e930
Step 2/8 : COPY . /watcher
```



```
root@efc70d7cc5159c9a:~/code/watcher# cd ..
root@efc70d7cc5159c9a:~/code# docker stop web
web
root@efc70d7cc5159c9a:~/code# docker logs mailer
CH2 Example Mailer has started.
root@efc70d7cc5159c9a:~/code#
```

Great job following through the full process! 🎉 You're **almost done**, and everything seems to be working smoothly — your ``web``, ``mailer``, and ``agent`` containers are functioning correctly. However, there's **one small issue** causing the mailer to not show the alert after stopping the web server:

⚠️ Issue: No alert message received by ``mailer``

Even though the agent says "System up." and stops the web container, your mailer log only shows:

```
nginx

CH2 Example Mailer has started.
```

That means the message like:

```
kotlin

Sending email: To: admin@work Message: The service is down!
```

was not printed — indicating the ``agent`` either:

- **didn't actually detect failure**, or
- **failed to connect to ``mailer`` properly** when web went down.

✅✅ Here's what you should do next (fix + improvements):

✅ Step 1: ✅ Update ``watcher.sh`` to use container names directly (modern way)

Instead of using old `--link`` aliases like ``insideweb`` and ``insidemailer``, update your watcher script like this:

watcher/watcher.sh

```
sh

#!/bin/sh
echo "Watcher started. Monitoring web service..."

while true
do
    if printf "GET / HTTP/1.0\n\n" | nc -w 2 web 80 | grep -q "200 OK"; then
        echo "System up."
    else
        echo "Web service down. Sending alert..."
        echo "To: admin@work Message: The service is down!" | nc mailer 33333
```

```

        break
    fi
    sleep 1
done

```

Then **rebuild** your watcher image:

```

bash

cd watcher
docker build -t watcher-image .

```

✅ Step 2: ✅ Create a Docker network

This will allow containers to **talk to each other using just their names** (which is best practice).

```

bash

docker network create webnet

```

✅ Step 3: ✅ Restart your containers using the new network


 Remove the old containers (if still running):

```

bash

docker rm -f web mailer agent

```


 Start them using the `webnet` network:

```

bash

docker run -d --name web --network webnet nginx:latest
docker run -d --name mailer --network webnet mailer-image
docker run -it --rm --name agent --network webnet watcher-image

```

 Now the `agent` container will be able to reach `web` and `mailer` using just their names: `web` and `mailer`.

✅ Step 4: ✅ Test the alert system

1. **Everything running?** Confirm:

```

bash


docker ps

```

2. **Stop the web server:**


```
bash

docker stop web
```

3.  You should now see the message printed in the agent output:

```
bash

Web service down. Sending alert...
```

4.  And in the mailer logs:

```
bash

Sending email alert: To: admin@work Message: The service is down!
```

BONUS: Save Logs to a File (for debugging)

To save logs from mailer and agent:

```
bash

docker logs mailer > mailer.log
docker logs agent > agent.log
```




Or follow in real-time:

```
bash

docker logs -f mailer
```

Next Steps: Generate Project Bundle?

Would you like me to now:

-  Generate a clean `.env` version of this?
-  Create a final `docker-compose.yml` with this network logic?
-  Zip up the full folder with `mailer/`, `watcher/`, `.env`, and `docker-compose.yml`?

Let me know — I can prep and send you the full ZIP bundle ready to run!



what is the work of nc in these case; explain it in simple manner

Great question! Let's break it down **very simply**:

What is `nc`?

`nc` stands for **Netcat**.


It's a tiny tool that lets you **send and receive data** over the network using ports. Think of it as a **walkie-talkie for computers**.

In your project, `nc` is used in two roles:


✓ 1. In `mailer.sh` — Listening for messages

```
sh
MESSAGE=$(nc -l -p 33333)
```

- `-l` = listen mode
- `-p 33333` = on port 33333

 This means:

“Wait here and listen on port 33333. When someone sends me a message, grab it and save it into `MESSAGE`.”

So the **mailer** is like a post office waiting for a letter .

✓ 2. In `watcher.sh` — Sending a message

```
sh
echo "To: admin@work Message: The service is down!" | nc mailer 33333
```

This means:

“Send this alert message to the `mailer` container on port 33333.”

So the **watcher (agent)** is like a person mailing a letter  when it sees the website is down.

Simple Analogy:

Imagine:

- `mailer` is a **radio operator** always listening on channel `33333`.
 - `watcher` is a **person with a walkie-talkie** who says:

“Hey, the web server is down!”
 - Netcat (`nc`) is just the **walkie-talkie system** used to send/receive that message.
-

✓ Summary

Script	How `nc` is used	Meaning
`mailer.sh`	`nc -l -p 33333`	Wait for a message on port 33333
`watcher.sh`	`nc mailer 33333`	Send a message to the `mailer` on port 33333

This allows **communication between containers** using simple network messages — no email servers, no web sockets, just plain text over TCP/IP using `nc`.

Would you like me to replace `nc` with a more modern Python or Node.js version in the future? Or is this minimal setup perfect for your learning stage?