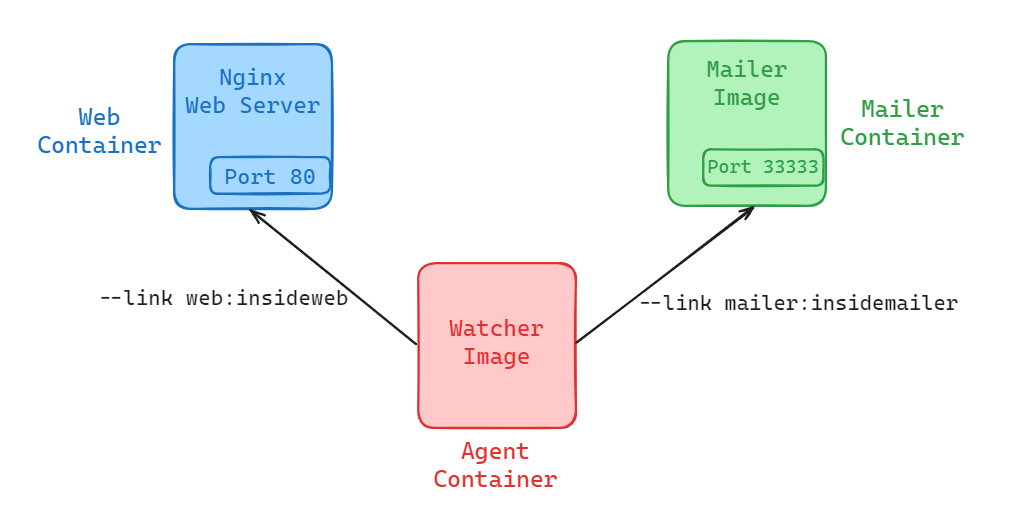
**Deploying a Monitored NGINX Web Server Using Docker**

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**

****

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:

1. **Web Container**: Runs the NGINX web server.
2. **Mailer Container**: Sends email notifications.
3. **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

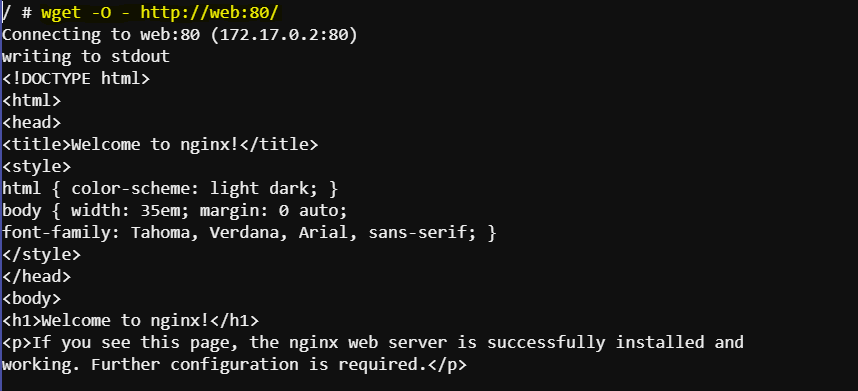
**This command:**

* 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.



**Monitoring and Notifications**

**Step 4: Start the Agent Container**

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

mkdir watcher

cd watcher

**Edit watcher.sh and add the following content:**

[[ extra file a ase watcher.sh]]

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.

**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 .

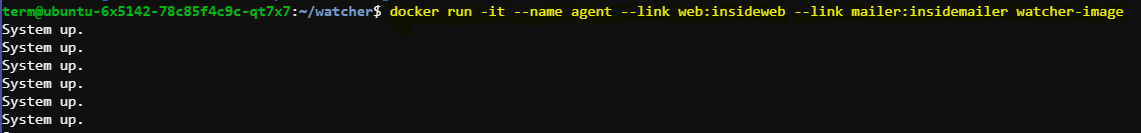
This command builds a Docker image with the name watcher-image.

**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.



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

**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:**

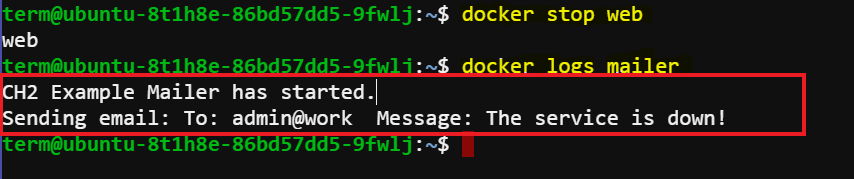
docker stop web

**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!**

****

**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.