Day 12 — Automating Docker Image Builds & Container Deployment with Shell Scripting!

If you're diving into Docker and automation, mastering simple shell scripts can turbocharge your DevOps workflow!

Today, I'm sharing a **robust shell script** that:

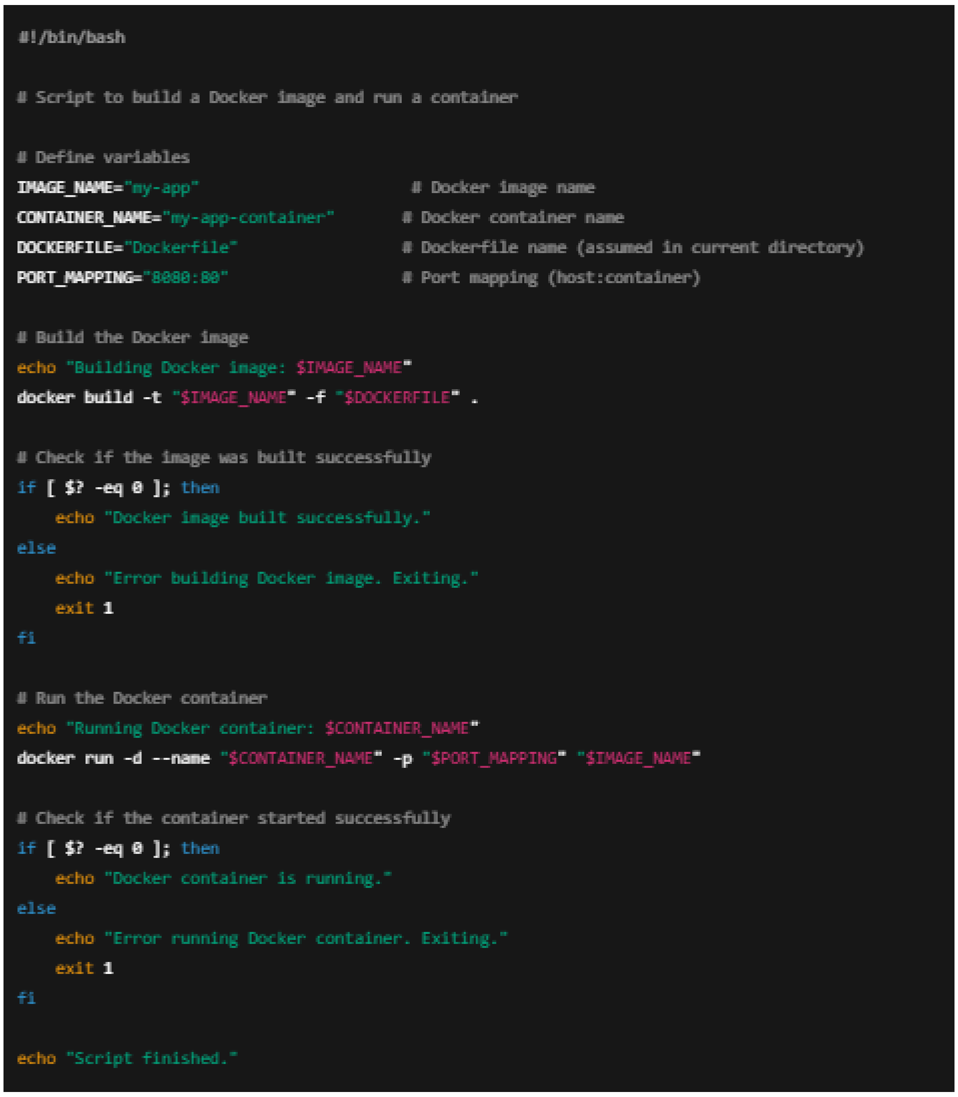
- Builds a Docker image from a Dockerfile
- Automatically runs a container with your desired settings
- Validates each step and handles errors gracefully

Why automate this?

- Manually building and running containers every time is repetitive and error-prone.
- A script saves time, reduces mistakes, and can be easily integrated into CI/CD pipelines.
- It ensures your Docker containers are always started with the right parameters.

I have explored multiple approaches to writing Docker shell scripts, each designed to streamline the process of building and running containers efficiently. These scripts offer various techniques to automate container creation, handle configurations, and optimize deployment workflows. By experimenting with different scripting methods, I aim to enhance flexibility, improve error handling, and make container management more seamless. Let me know if you'd like to review or refine any specific approach!

Version : 01 of Shell Script



```bash
#!/bin/bash

# Script to build a Docker image and run a container

# Define variables
IMAGE_NAME="my-app"                    # Docker image name
CONTAINER_NAME="my-app-container"     # Docker container name
DOCKERFILE="Dockerfile"               # Dockerfile name (assumed in current directory)
PORT_MAPPING="8080:80"                # Port mapping (host:container)

# Build the Docker image
echo "Building Docker image: $IMAGE_NAME"
docker build -t "$IMAGE_NAME" -f "$DOCKERFILE" .

# Check if the image was built successfully
if [ $? -eq 0 ]; then
    echo "Docker image built successfully."
else
    echo "Error building Docker image. Exiting."
    exit 1
fi

# Run the Docker container
echo "Running Docker container: $CONTAINER_NAME"
docker run -d --name "$CONTAINER_NAME" -p "$PORT_MAPPING" "$IMAGE_NAME"

# Check if the container started successfully
if [ $? -eq 0 ]; then
    echo "Docker container is running."
else
    echo "Error running Docker container. Exiting."
    exit 1
fi

echo "Script finished."
```

Explanation:

- ✓ #!/bin/bash: Shebang line, specifying that the script should be executed with Bash.
- ✓ Variables:
  - ➢ IMAGE_NAME: The name you want to give to your Docker image.
  - ➢ CONTAINER_NAME: The name you want to assign to the running container.
  - ➢ DOCKERFILE_PATH: The location of the Dockerfile used to build the image.

    - PORT_MAPPING: Specifies how ports on your host machine are mapped to ports inside the container.
- ✓ docker build: This command builds a Docker image from a Dockerfile.
    - -t $IMAGE_NAME: Tags the image with the specified name.
    - -f $DOCKERFILE_PATH: Specifies the path to the Dockerfile.

Simple version of code

docker build -t my_app -f Dockerfile .

-t Image_Name → *We represent the Image with Tag, the built image with a name (Image_Name). This makes it easier to refer to the image later**.***

-f File_Name → *Specifies the name of the Dockerfile (File_Name)*

. (dot at the end) → *meaning Docker should use the current directory as the location for files needed during the build**.***

Condition Explanation

$?→ holds the exit status of the last command executed(**docker build -t my_app -f Dockerfile .**)

If the previous command (assumed to be a Docker image build command) ran successfully ($? -eq 0), it prints "Docker image built successfully!".

Otherwise, it prints "Error building Docker image. Exiting." and exits the script with status 1 (indicating failure).

Simple version of code of If block

```
if[$? == 0]; then
    echo “Docker image built successfully”
else
     echo "Error building Docker image. Exiting."
     exit 1
if
```

- ✓ docker run: This command runs a Docker container from a Docker image.
  - ➢ -d: Runs the container in detached mode (in the background).
  - ➢ --name $CONTAINER_NAME: Assigns a name to the container.
  - ➢ -p $PORT_MAPPING: Maps ports between the host and the container

Port Mapping :

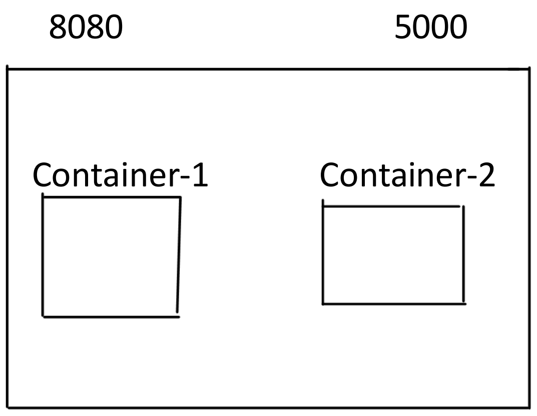
80 → the container port and 8080 → Host port

In real time we can create a container, by default the ports were bind to containers.

→ Containers are isolated by default, meaning their services are not accessible from the host.

→ Port mapping lets traffic from your ***host machine*** *reach the container.*

Host Port -----------------Mapping----------------→ Container Port



Host {our machine has different ports}

Note : once the host port is bind with one container, we cannot bind the other container to same host

Condition Explanation

Again, $? is checked to verify if the docker run command was successful.

```
docker run -d  --my-app-container -p 8080: 80 my-app
```

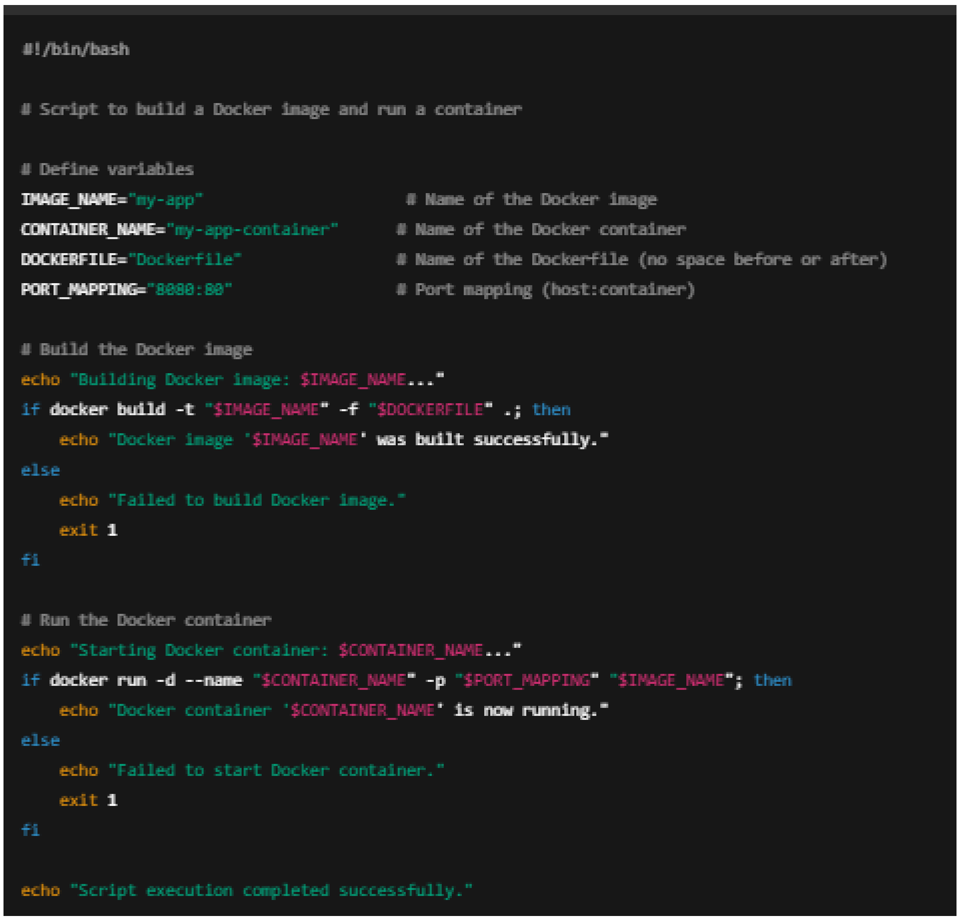
If successful ($? -eq 0), it prints "Docker container is running!".

Otherwise, it prints "Error running Docker container. Exiting." and exits with status 1.

Simple version of code of If block

```
if[$? == 0]; then
      echo "Docker container is running!"
else
       echo "Error running Docker container. Exiting."
       exit 1
if
```

Version : 02 of Shell Script



```bash
#!/bin/bash

# Script to build a Docker image and run a container

# Define variables
IMAGE_NAME="my-app"                    # Name of the Docker image
CONTAINER_NAME="my-app-container"     # Name of the Docker container
DOCKERFILE="Dockerfile"               # Name of the Dockerfile (no space before or after)
PORT_MAPPING="8080:80"                # Port mapping (host:container)

# Build the Docker image
echo "Building Docker image: $IMAGE_NAME..."
if docker build -t "$IMAGE_NAME" -f "$DOCKERFILE" .; then
    echo "Docker image '$IMAGE_NAME' was built successfully."
else
    echo "Failed to build Docker image."
    exit 1
fi

# Run the Docker container
echo "Starting Docker container: $CONTAINER_NAME..."
if docker run -d --name "$CONTAINER_NAME" -p "$PORT_MAPPING" "$IMAGE_NAME"; then
    echo "Docker container '$CONTAINER_NAME' is now running."
else
    echo "Failed to start Docker container."
    exit 1
fi

echo "Script execution completed successfully."
```

- ✓ Building the Docker Image
    - ➢ The script prints a message indicating the start of the Docker image build process.
    - ➢ It executes the docker build command with the -t flag to assign a name to the image.
    - ➢ The -f flag specifies the path to the Dockerfile.
    - ➢ The . at the end tells Docker to use the current directory as the build context.

Simple version of code

docker build -t my_app -f Dockerfile .

-t Image_Name → *We represent the Image with Tag, the built image with a name (Image_Name). This makes it easier to refer to the image later.*

-f File_Name → *Specifies the name of the Dockerfile (File_Name)*

. (dot at the end) → *meaning Docker should use the current directory as the location for files needed during the build.*

- ➢ The script uses if command; then ... else ... fi to check if the build was successful.
- ➢ If successful, it prints confirmation. Otherwise, it provides an error message and stops execution using exit 1.

Simple version of code of If block

```
if [docker build -t my_app -f Dockerfile .]; then

        echo "Docker image built successfully"

else

        echo "Error building Docker image. Exiting."

        exit 1

if
```

- ✓ Running the Docker Container
    - ➢ The script announces the start of the container.
- ✓ It runs the docker run command:
    - ➢ -d ensures the container runs in the background.
    - ➢ --name assigns the specified name to the container.
    - ➢ -p maps ports between the host and the container.
    - ➢ The image name is passed as the source for the container.
- ✓ If the container starts correctly, it confirms success. If not, an error message is shown, and the script exits.

Condition Explanation

docker run -d --my-app-container -p 8080: 80 my-app

If successful (docker run -d --my-app-container -p 8080: 80 my-app), it prints "Docker container is running!".

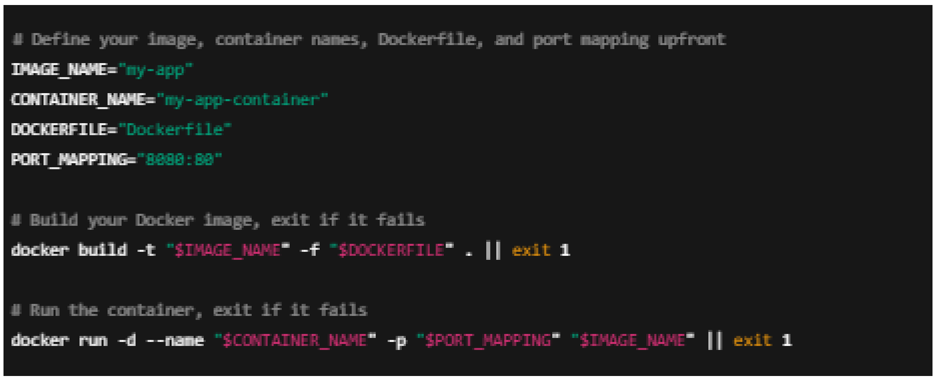
Otherwise, it prints "Error running Docker container. Exiting." and exits with status 1.

Simple version of code of If block

```
    if[docker run -d  --my-app-container -p 8080: 80 my-app]; then
          echo "Docker container is running!"
   else
         echo "Error running Docker container. Exiting."
         exit 1
  if
```

- ✓ Final Confirmation
    - ➢ A final message is printed to indicate that all steps have been completed successfully.

Version : 03 of Shell Script

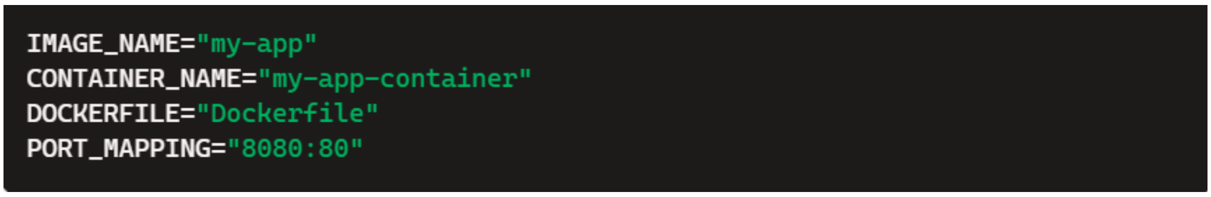


```
# Define your image, container names, Dockerfile, and port mapping upfront
IMAGE_NAME="my-app"
CONTAINER_NAME="my-app-container"
DOCKERFILE="Dockerfile"
PORT_MAPPING="8080:80"

# Build your Docker image, exit if it fails
docker build -t "$IMAGE_NAME" -f "$DOCKERFILE" . || exit 1

# Run the container, exit if it fails
docker run -d --name "$CONTAINER_NAME" -p "$PORT_MAPPING" "$IMAGE_NAME" || exit 1
```

Explanation:

#!/bin/bash: Shebang line, specifying that the script should be executed with Bash.

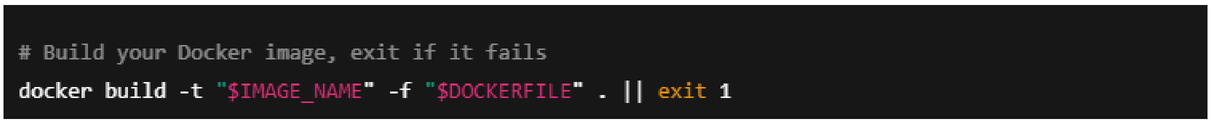
Variable :



```
IMAGE_NAME="my-app"
CONTAINER_NAME="my-app-container"
DOCKERFILE="Dockerfile"
PORT_MAPPING="8080:80"
```

- ✓ **IMAGE_NAME="my-app"**
  - ➢ Sets the name for the Docker image that will be created.
  - ➢ Instead of hardcoding "my-app" throughout the script, using a variable makes it easier to modify in the future.
- ✓ **CONTAINER_NAME="my-app-container"**
  - ➢ Defines the name of the running container.
  - ➢ Using a specific name avoids random container names.
- ✓ **DOCKERFILE="Dockerfile"**
  - ➢ Specifies the Dockerfile to use for building the image.
  - ➢ If the file had a different name (e.g., Dockerfile.dev), you could update the variable here instead of changing multiple lines in the script.

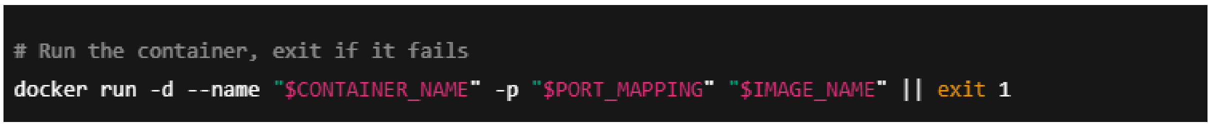
Docker build :



```
# Build your Docker image, exit if it fails
docker build -t "$IMAGE_NAME" -f "$DOCKERFILE" . || exit 1
```

- ✓ -t "$IMAGE_NAME"
    - ➢ Tags the image with the name my-app.
- ✓ -f "$DOCKERFILE"
    - ➢ Specifies which Dockerfile to use.
- ✓ . **(dot at the end)**
    - ➢ Defines the "build context," meaning Docker will look in the current directory for files needed during the build.
- ✓ || exit 1
    - ➢ If the build fails, the script immediately exits with an error code (1).

Docker Run :



```
# Run the container, exit if it fails
docker run -d --name "$CONTAINER_NAME" -p "$PORT_MAPPING" "$IMAGE_NAME" || exit 1
```

- ✓ -d
    - ➢ Run container in detach mode means background
- ✓ --name "$CONTAINER_NAME"
    - ➢ Specifies the name of container we were using.
- ✓ -p "$PORT_MAPPING"

Port Mapping :

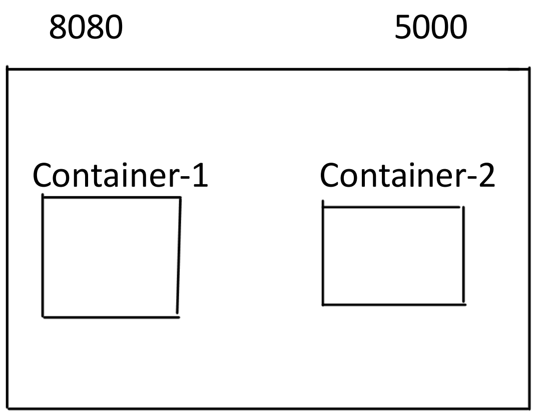
80 → the container port and 8080 → Host port

In real time we can create a container, by default the ports were bind to containers.

→ *Containers are isolated by default, meaning their services are not accessible from the host.*

→ *Port mapping lets traffic from your* ***host machine*** *reach the container.*

Host Port -----------------Mapping----------------→ Container Port



Host {our machine has different ports}

Note : once the host port is bind with one container, we cannot bind the other container to same host

- ✓ $IMAGE_NAME
  - ➢ Specifies name of Image we were used.
- ✓ || exit 1
  - ➢ If the build fails, the script immediately exits with an error code (1).