MA
UNIT 3
Chapter 2:
Delivering Continuously:

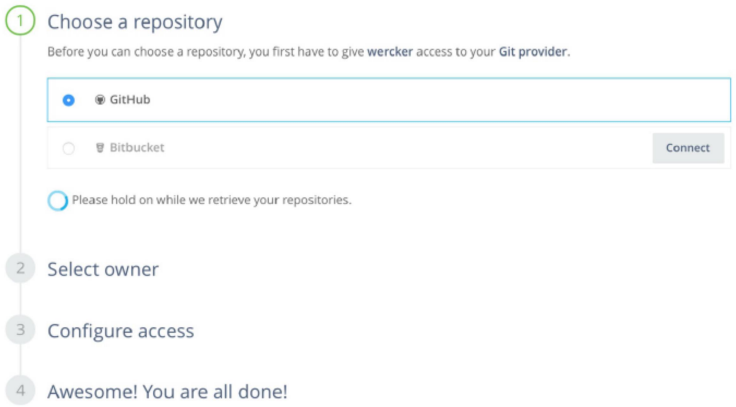
Q1) Explain the concept of Docker.

- It is a container tool that utilizes Linux kernel features like cgroups and namespaces to isolate network, file, and memory resources without incurring the burden of a full, heavyweight virtual machine.
- The beauty of Docker is that it works in all of the environments without changing the container format.
- Docker gives us the ability to create an immutable release artifact that will run anywhere, regardless of the target environment.
- An immutable release means that we can test a Docker image in a lower environment like development or QA and have reasonable confidence that it will perform exactly the same way in production.
-
- **Installing Docker:**
- Make sure you check the documentation to ensure you're looking at the newest installation instructions before performing the install.
- The Docker app comes with a nice icon that sits in your menu bar and automatically manages your environment to allow terminal/shell access.
- Since Docker relies on features specific to the Linux kernel, you're really starting up a VirtualBox virtual machine that emulates those Linux kernel features in order to start a Docker server daemon.
- docker images: This command lists the Docker images you have stored in your local repository.
-
- [
- **Install Docker Desktop on Windows**
- 1. Double-click Docker Desktop Installer.exe to run the installer.
- If you haven't already downloaded the installer (Docker Desktop Installer.exe), you can get it from Docker Hub. Follow the instructions on the installation wizard to accept the license, authorize the installer, and proceed with the install.
- 2. When prompted, authorize the Docker Desktop Installer with your system password during the install process. Privileged access is needed to install networking components, links to the Docker apps, and manage the Hyper-V VMs.
- 3. Click Finish on the setup complete dialog and launch the Docker Desktop application.
-]
-
- **Running Docker Images:**
- Docker lets you manually pull images into your local cache from a remote repository like docker hub. However, if you issue a docker run command and you haven't already cached that image, you'll see it download in the terminal.
- you need to map the port from the inside of the container to the outside port so you can open up a browser from your desktop:
- $ docker run -p 8080:8080 dotnetcoreservices/hello-world
- maps port 8080 inside the Docker image to port 8080 outside the Docker image.
- Docker provides network isolation, so unless you explicitly allow traffic from outside a container to be routed inside the container, the isolation will function just like a firewall.

- $ docker ps (use this to check status of running application in docker)
- To kill a running Docker process, just find the container ID from the docker ps output and pass it to docker kill:
- $ docker kill 61a68ffc3851.

Q2) Explain the continuous integration with Wercker.

- **1.Building Services with Wercker:**
- **Features:**
- Free trials, easy to use, tight integration with Docker and support for spinning up multiple attached Docker images for integration testing are outstanding.
- **three basic steps:**
- 1. Create an application in Wercker using the website.
- 2. Add a wercker.yml file to your application's codebase.
- 3. Choose how to package and where to deploy successful builds.
- We need to sign up for an account.
- Once you've got an account and you're logged in, click the Create link in the top menu.



- The wizard will prompt you to choose a GitHub repository as the source for your build.
- It will then ask you whether you want the owner of this application build to be your personal account or an organization to which you belong.
- Once you've created the application, you need to add a wercker.yml file to the repository.
- This file contains most of the metadata used to describe and configure your automatic build.

- **2. Installing the Wercker CLI:**
- Your code is added to a Docker image specified in your wercker.yml file, and then you choose what gets executed and how.
- To run Wercker builds locally, you'll need the Wercker CLI.
- The Wercker Command Line Interface (CLI) is an application that replicates the behaviour of Wercker in your local development environment, allowing you to build and test your applications and CI/CD flows locally.
- use Homebrew to install the Wercker CLI:
- $ brew tap wercker/wercker
- $ brew install wercker-cli
- If you've installed the CLI properly, you should be able to ask the CLI for the version:
- $ wercker version

- **3.Adding the wercker.yml Configuration File:**
- create a wercker.yml file to define how you want your application built and deployed.

- **Contents of wercker.yml:**
- box: microsoft/dotnet:1.1.1-sdk
- no-response-timeout: 10
- **build:**
- steps:
- - script:
- name: restore
- code: |
- dotnet restore
- - script:
- name: build
- code: |
- dotnet build
- - script:
- name: publish
- code: |
- dotnet publish -o publish
- - script:
- name: copy binary
- code: |
- cp -r . $WERCKER_OUTPUT_DIR/app
- cd $WERCKER_OUTPUT_DIR/app
- **deploy:**
- steps:
- - internal/docker-push:
- username: $USERNAME
- password: $PASSWORD
- repository: dotnetcoreservices/hello-world
- registry: https://registry.hub.docker.com
- entrypoint: "/pipeline/source/app/docker_entrypoint.sh"
-
- **4. We then run the following commands inside this container:**
- 1. dotnet restore : to restore or download dependencies for the .NET application.
- 2. dotnet build to compile the application.
- 3. dotnet publish to compile and then create a published, "ready to execute" output directory.
-
- The easiest way to run a Wercker build is to simply commit code.
- Once Wercker is configured, your build should start only a few seconds after you push.
-
- **5. The next step after that is to see how the application builds using the Wercker pipeline:**
- We usually have a script with our applications
- that looks like this to invoke the Wercker build command:
- rm -rf _builds _steps _projects
- wercker build --git-domain github.com \
- --git-owner microservices-aspnetcore \
- --git-repository hello-world
- rm -rf _builds _steps _projects

Q3) Continuous Integration with Circle CI

- CircleCI offers control at a slightly lower level.
- If you go to http://circleci.com you can sign up for free with a new account or log in using your GitHub account.
- You can start with one of the available build images and then supply a configuration file telling CircleCI how to build your app.
- Here's a look at the circle.yml file for the "hello world" project:
- **machine:**
- pre:
- - sudo sh -c 'echo "deb [arch=amd64] https://aptmo.
- trafficmanager.net/repos/
- dotnet-release/ trusty
- main" > /etc/apt/sources.list.d/dotnetdev.list'
- - sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --
- **recv-keys**
- 417A0893
- - sudo apt-get update
- - sudo apt-get install dotnet-dev-1.0.1
- **compile:**
- **override:**
- - dotnet restore
- - dotnet build
- - dotnet publish -o publish
- **test:**
- override:
- - echo "no tests"
-
- The key difference between this build and Wercker is that instead of being able to run the build inside an arbitrary Docker image that already has .NET Core installed on it, here we have to use tools like apt-get to install the .NET tools.
- You may notice that the list of shell commands executed in the pre phase of the machine configuration is exactly the same set of steps listed on Microsoft's website to install .NET Core on an Ubuntu machine.
- That's basically what we're doing—installing .NET Core on the Ubuntu build runner provided for us by CircleCI.

Q4) Deploying to Docker Hub.

- Once you have a Wercker (or CircleCI) build that is producing a Docker image and all your tests are passing, you can configure it to deploy the artifact anywhere you like.
- For now, we're going to deploy to docker hub.
- In the wercker.yml file there is a deploy section that, when executed, will deploy the build artifact as a docker hub image.
- We use Wercker environment variables so that we can store our docker hub username and password securely and not check sensitive information into source control.
- deploy:
- steps:
- - internal/docker-push:
- username: $USERNAME
- password: $PASSWORD
- repository: dotnetcoreservices/hello-world

- registry: https://registry.hub.docker.com
- entrypoint: "/pipeline/source/app/docker_entrypoint.sh"
- Assuming our docker hub credentials are correct and the Wercker environment variables are set up properly, this will push the build output to docker hub and make the image available for pulling and executing on anyone's machine—including our own target environments.
- After we successfully build, we then deploy the artifact by executing the deploy step in the wercker.yml file.
- The docker hub section of this pipeline is easily created by clicking the "+" button in the GUI and giving the name of the YAML section for deployment (in our case it's deploy).

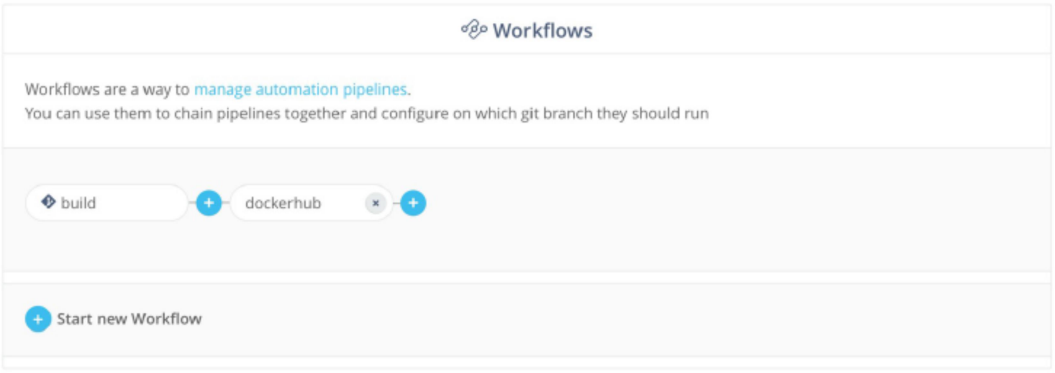


Figure 2-3. Deployment pipelines in Wercker

-
- Note: YAML—short for "YAML Ain't Markup Language"—is a human-readable, data-oriented markup language used to parse wercker.yml files.
- YAML is a whitespace-indented language, meaning that indentation is used to denote structure. Items of the same indentation are considered to be siblings, while more or less indentation denotes child and parent relationships, respectively.