## Overview

Containerizing an application means updating it's behavior to work well in a containerized environment. In this lab, we will modify a small, "legacy" application to work well as a container. Then we will test it running in docker.

[Overview](#_4l6y9k9n7exo)

[Instructions](#_hxr977vi6sdt)

[Task 0: Create simple application](#_hszxfr2ntf23)

[Step 0: Make new directory](#_z1ilshtapmqx)

[Step 1: Write app.py](#_2efpjn8nvm1c)

[Step 2: Create config file](#_c0i533prk1n7)

[Step 3: Run the application](#_2c4oq5a723qv)

[Task 1: "Containerize" App](#_uxscfrnn8986)

[Step 1: Change logging to standard output](#_y2xjad3y77zo)

[Step 2: Read config directly from environment variables](#_stcldscuqytw)

[Step 3: Run and test the application](#_y8iqsoq879p7)

[Task 2: Dockerize the Application](#_gjg0saj5jfxe)

[Step 1: Create Dockerfile](#_qwyzyj22iu7v)

[Step 2: Build image](#_ysv4078yt1m6)

[Step 3: Run the app](#_xni8u8rpim9l)

[Step 4: View the app logs](#_w2jjp8uqt6s9)

[Step 5: Cleanup](#_yjc2aibns43x)

## 

## Instructions

Read this lab like a book, all text is there for a reason!

"→" denotes an action you must take

Use your favorite editor to edit files within the console. I suggest VI, nano, or emacs.

|  |
| --- |
| White boxes with black text denote commands and file contents |

|  |
| --- |
| Black boxes with green text denote example output |

### Task 0: Create simple application

#### Step 0: Make new directory

→

|  |
| --- |
| mkdir ~/containerize  cd ~/containerize |

#### Step 1: Write app.py

→ app.py

|  |
| --- |
| import os  import configparser  from flask import Flask, redirect, request, url\_for  import logging  logging.basicConfig(filename='app.log', filemode='w', format='%(name)s - %(levelname)s - %(message)s', level=logging.DEBUG)  config = configparser.ConfigParser()  has\_config = config.read('vars.ini')  signatures = []  app = Flask(\_\_name\_\_)  # configurations  font = config['style']['DISPLAY\_FONT']  font\_color = config['style']['DISPLAY\_COLOR']  environment = config['debug']['ENVIRONMENT']  @app.route('/', methods=['GET'])  def index():  html = """  Signatures: <br />  <font face="%(font)s" color="%(color)s">  %(messages)s  </font>  <br /> <br />  <form action="/signatures" method="post">  Sign the Guestbook: <input type="text" name="message"><br>  <input type="submit" value="Sign">  </form>  <br />  <br />  Debug Info: <br />  ENVIRONMENT is %(environment)s  """  messages\_html = "<br />".join(signatures)  return html % {"font": font, "color": font\_color, "messages": messages\_html, "environment": environment}  @app.route('/signatures', methods=['POST'])  def write():  message = request.form.get('message')  signatures.append(message)  return redirect(url\_for('index'))  if \_\_name\_\_ == "\_\_main\_\_":  app.run(host='0.0.0.0', port=8080) |

#### Step 2: Create config file

→ vars.ini

|  |
| --- |
| [debug]  ENVIRONMENT=DEV  [style]  DISPLAY\_FONT=arial  DISPLAY\_COLOR=red |

#### Step 3: Run the application

→ Ensure flask dependency

|  |
| --- |
| pip3 install --user Flask |

→ Run the app

|  |
| --- |
| python3 app.py |

→ Go to [http://localhost:8080](http://localhost:8080/) in browser

→ Sign the guestbook

→ Type cntl-C to kill the app

→ View logs in app.log file

|  |
| --- |
| cat app.log |

### Task 1: "Containerize" App

#### Step 1: Change logging to standard output

→ Update the code to no longer log to a file

See the docs for the logging module <https://docs.python.org/3/library/logging.html>

#### Step 2: Read config directly from environment variables

→ Remove the need for /opt/guestbook/vars.ini

\*hint\* You should no longer need the configparser module. Check out the os module for reading environment variables <https://docs.python.org/3/library/os.html>

Name your environment variables whatever you please, just ensure they match what you provide later on.

#### Step 3: Run and test the application

→ Supply some environment variables and ensure they take effect

|  |
| --- |
| DISPLAY\_COLOR=green DISPLAY\_FONT=arial ENVIRONMENT=Prod python3 app.py |

### Task 2: Dockerize the Application

#### Step 1: Create Dockerfile

This is a similar dockerfile to the previous lab, since this is also a python flask application. Let's add some additional metadata to our image.

[Dockerfile Reference](https://docs.docker.com/engine/reference/builder/)

→ Create Dockerfile

|  |
| --- |
| FROM python:3.8-alpine  # TODO Set metadata for who maintains this image  COPY \* /app/  RUN pip3 install Flask  EXPOSE 8080  #TODO Set default values for env variables  #TODO \*bonus\* add a health check that tells docker the app is running properly  # TODO have the app run as a non-root user  CMD python /app/app.py |

#### Step 2: Build image

→ Build the image

|  |
| --- |
| docker build -t guestbook . |

#### Step 3: Run the app

→ Run the container

|  |
| --- |
| docker run --name guestbook -e DISPLAY\_COLOR="green" -e DISPLAY\_FONT="arial" -e ENVIRONMENT="Prod" -p 8080:8080 -d guestbook |

→ Test the app in browser

#### Step 4: View the app logs

We should see the output of our logs if we correctly output to stdout.

→ View the logs of our container

|  |
| --- |
| docker logs guestbook |

|  |
| --- |
| \* Serving Flask app "app" (lazy loading)  \* Environment: production  WARNING: Do not use the development server in a production environment.  Use a production WSGI server instead.  \* Debug mode: off  \* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)  172.17.0.1 - - [04/May/2019 20:06:33] "GET / HTTP/1.1" 200 - |

#### Step 5: Cleanup

→ Remove the container

|  |
| --- |
| docker stop guestbook && docker rm guestbook |