

## Pied Piper Technical Workshop

Day 1: Cloud Native Applications

#### **Abstract**

This document is provided to assist attendees with completing the appropriate labs to apply the concepts and knowledge learnt throughout the technical workshop program. It is not intended to be used or distributed in isolation and may not contain all required information.

May 2020



#### Revisions

Version	Date	Description
0.1	May 2020	Initial draft

The information in this publication is provided "as is." Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

© 2020 Dell Inc. or its subsidiaries. All Rights Reserved. Dell, EMC, Dell Technologies and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.

Dell believes the information in this document is accurate as of its publication date. The information is subject to change without notice

Student Lab Guide

# **D¢LL**Technologies

## **Table of Contents**

Lab 1: Github	5
Module Objectives:	5
What is Git?	6
How to use .gitignore to avoid committing sensitive data	7
How to use environment variables to store and retrieve sensitive data	8
How to use Github and Github desktop	9
Lab Exercise: Create a new repo and publish a file with your name	10
Lab 2: Deploy an App to PWS	11
Module Objectives:	11
What is Pivotal Web Services (PWS)?	12
Lab Exercise: Clone an example app and deploy to PWS	13
Reflect on what just happened	14
Retrospective: The results	15
Lab 3: Create a Static HTML Website	16
Module Objectives:	16
Lab Exercise: Copy the static webpage provided and setup a new project	17
Retrospective: A basis for an MVP	18
Lab 4: Python and Flask	19
Module Objectives:	19
First a bit about Python & Flask	20
What's changed in Python 3?	21
Lab Exercise: Create a basic Python app to run the Flask web server	22
Retrospective: Applying concepts of Model, Views, Controllers	23
Lab 5: MongoDB	24
Module Objectives:	24
Lab Exercise: Copy the new code files and overwrite existing application files	25
Retrospective: Review the implementation of 'models' into our application	26
Lab 6: ECS	29
Module Objectives:	29
Lab Exercise:	30
Retrospective: Review code and note implementation of Boto3 & Pillow	31
Lab 7: PWS & mLab	34





## Lab 1: Github

Modu	le Ob	jectives:
------	-------	-----------

Learn about git, git tools, Github and Github Desktop GUI
Setup a new personal repo for our project
Use git tools to learn the skills to commit code
Upload a file to Github



#### What is Git?

- Distributed version control system for tracking changes in ANY set of files
  - Created by Linus Torvalds in 2005 for Linux kernel development
- Important to note that Git is an open standard and licensed under the GNU GPL v2
- Here are a few of the popular SaaS based implementations;
  - Github
  - Bitbucket
  - AWS Code Commit
  - Gitlab
  - Etc....



How to use .gitignore to avoid committing sensitive data

- Posting passwords to services like AWS costs \$\$\$\$
- Best Practice: When publishing code use .gitignore to exclude any files containing sensitive data
- Below is an example of using an external file containing credentials and using python code to reference the variables rather than embedding the username and password directly into the main.py application code. You can then use the .gitignore file to exclude the setenv.py file that contains sensitive data.
- This approach is more secure than embedding credentials into application code, but is still not 12-factor compliant and can easily be mistakenly uploaded to a git repository or hosting provider with your source code.

```
setenv.py

user = "joeblogs"
password = "whatever"

.gitignore
setenv.py
```

```
from setenv import user,
password

print "user is: " + user
print "pass is: " + password
```



#### How to use environment variables to store and retrieve sensitive data

- Environment variables are considered 'best practice' when working with sensitive data such as user credentials and environment configuration. See 'The Twelve-Factor App: III. Config'
- If you have a python-based application and want to manage your environment using a separate script that can be ran before the application is deployed, you might have a setup file similar to the below and exclude that file from being uploaded to Github and Pivotal Cloud Foundry.

#### setenv.py

```
import os
os.environ["user"]="joeblogs"
os.environ["pass"]="whatever"
```

#### .gitignore and .cfignore

```
setenv.py
```

• The main application would retrieve the credentials from the environment as shown below;

#### main.py

```
import os

user = os.environ.get('user')
pass = os.environ.get('pass')
```



#### How to use Github and Github desktop

- If you do not know how to use Github, please complete the following tutorial https://guides.github.com/activities/hello-world/
- If you do not know how to use Github Desktop GUI client, please complete the following tutorial https://help.github.com/desktop/guides/getting-started-with-github-desktop/



Lab Exercise: Create a new repo and publish a file with your name

Complete the below steps to demonstrate your understanding of the tools and concepts required for the remaining lab exercises;

- 1. Create a new private repo on Github using your personal account
- 2. Create a text file with your name as the filename
- 3. Add the content of the text file with the words "Hello Piper class of 2020!"
- 4. Commit your file to the above repo using the Github Desktop GUI tool and publish it to Github



## Lab 2: Deploy an App to PWS

Module Objectives
-------------------

Learn about PWS & cf cli tools
Confirm cf cli is installed and working
Login to PWS via cf cli
Push a demo app to PWS and confirm we have the skills and tools required to publish a web app to the PWS PaaS platform



#### What is Pivotal Web Services (PWS)?

- PWS is a hosted environment of Pivotal Cloud Foundry. PWS is hosted on AWS in the US-East region. PWS utilizes two availability zones for redundancy.
- Your free trial org includes 2GB of memory quota and \$87 of Trial credit. A trial org expires after one year or when you have used up all of your Trial credits.



Important note: Please create an organisation before continuing with the lab



Lab Exercise: Clone an example app and deploy to PWS

Complete the below steps to demonstrate your understanding of the tools and concepts required for the remaining lab exercises;

1. Clone the following repository;

https://github.com/cloudfoundry-samples/cf-sample-app-spring.git

- 2. Open a command prompt and navigate to the directory cloned above
  - > cd <path of cloned repo's>\cf-sample-app-spring
- Login to PWS with your account creds
  - > cf login -a https://api.run.pivotal.io
- 4. Push the app to PWS
  - > cf push
- 5. Test app by opening in browser

Use the URL shown in the 'routes:' section of the output displayed in your command prompt.

```
name: cf-demo
requested state:
routes: cf-demo-responsible-wombat.cfapps.io
last uploaded: stack: cflinuxfs3
buildpacks: https://github.com/cloudfoundry/java-buildpack.git

type: web
instances: 1/1
memory usage: 768M

Start command: JAVA_OPTS="-agentpath:$PWD/.java-buildpack/open_jdk_jre/bin/jvmkill-1.16.0_RELEASE=printHeapHistogram=1
-Djava.io.tmpdir=$TMPDIR -XX:ActiveProcessorCount=$(nproc)
-Djava.io.tmpdir=$TMPDIR -XX:ActiveProcessorCount=$(nproc)
-Djava.io.tmpdir=$TMPDIR -XX:ActiveProcessorCount=$(nproc)
-Djava.ext.dir=$$PWD/.java-buildpack/open_jdk_jre/bin/java-buildpack/open_jdk_jre/lib/ext
-Djava.security.properties=$PWD/.java-buildpack/java_security/java.security $JAVA_OPTS" &&
CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_jre/bin/java-buildpack-memory-calculator-3.13.0_RELEASE
-totMemory-$\frac{\text{$MEMORY}}{\text{$MEMORY}} \frac{\text{$MEMORY}}{\text{$MEMORY}} \frac{\text{$MEMO
```

e.g. https://cf-demo-responsible-wombat.cfapps.io (your URL will be different)



#### Reflect on what just happened

Take a look at the application name, number of instances, memory allocation, routes and build-pack details.

These are all provided by the **manifest.yml** file as application configuration options.

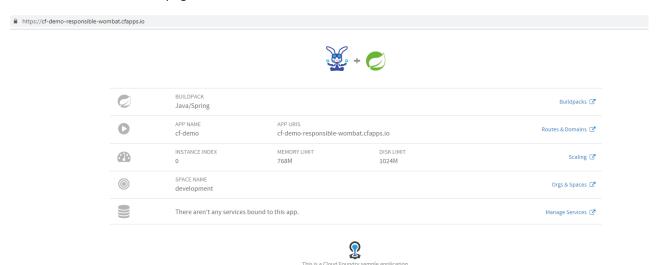
**Random-route: true** is a Cloud Foundry function that randomly generates a unique DNS based on selecting 2 words from a provided dictionary. This is used only in testing to ensure that DNS routes are not conflicting when deploying the same application many times.

**buildpacks:** are used to define the required libraries and runtimes to setup an environment for the application to run successfully. If you do not define a buildpack, Cloud Foundry will try to determine a default buildpack based on the language of the code provided.



Retrospective: The results

If your app was successfully deployed and you were able to open the correct URL in a web browser, you should see the below webpage;





## Lab 3: Create a Static HTML Website

### Module Objectives:

Learn a little bit about HTML & CSS
Create a basic website that will be used as a starting point for our application
Review the code and become familiar with the layout and design
Test the functionality of the default website
Save and upload to Github



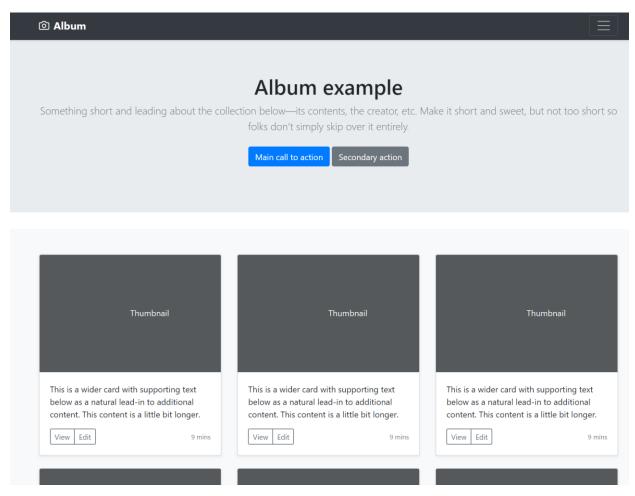
Lab Exercise: Copy the static webpage provided and setup a new project

- 1. Create a new repo on GitHub to store your project call it whatever you want
- 2. Clone the following repo and copy the file into your new working folder as created above <a href="https://github.com/theocrithary/Piper-2020/tree/master/Day%201/Lab%2003%20-%20CSS">https://github.com/theocrithary/Piper-2020/tree/master/Day%201/Lab%2003%20-%20CSS</a>
- 3. Open the 'default.html' file in a browser to test its 'look and feel'
- 4. Open the 'default.html' file in the Atom text editor provided and review the code
- 5. Commit the file to your personal repo and push to GitHub



Retrospective: A basis for an MVP

You now have a HTML web page that can be used as a template design for our new application.



If you opened the default.html file in the Atom text editor, you might have noticed the below lines of code in the <head> section at the top;

These are the Bootstrap responsive design and external CSS functionality that provides the look and feel of the webpage as discussed in the theory module earlier.

Try commenting out these lines of code and reload the page in the browser to see what happens.



## Lab 4: Python and Flask

M	od	u	le	Ol	bi	e	cti	ves	5

Learn about Python & Flask
Write a Python app using Flask web server to run the 'default.html' website we created earlier
Review the code and become familiar with the Python code
Test the functionality of the web app
Save and upload to Github



#### First a bit about Python & Flask

- Python is consistently in the top 4 languages YoY
- Python is embedded in many IAC tools (e.g. Ansible)
- Python is easy to implement with Raspberry Pi based IoT projects
- Python is becoming a leader in AI/ML applications
- Python is rich in community support
- Flask is a lightweight web server that easily runs with Python

May 2020	May 2019	Change	Programming Language	Ratings	Change
1	2	^	С	17.07%	+2.82%
2	1	•	Java	16.28%	+0.28%
3	4	^	Python	9.12%	+1.29%
4	3	<b>~</b>	C++	6.13%	-1.97%
5	6	^	C#	4.29%	+0.30%
6	5	•	Visual Basic	4.18%	-1.01%
7	7		JavaScript	2.68%	-0.01%
8	9	^	PHP	2.49%	-0.00%
9	8	<b>~</b>	SQL	2.09%	-0.47%
10	21	*	R	1.85%	+0.90%

Source: <a href="https://www.tiobe.com/tiobe-index/">https://www.tiobe.com/tiobe-index/</a>



Source: https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019



#### What's changed in Python 3?

Today's Python community largely supports Python 3 code and it is rare to find new code written in Python 2 anymore. However, if you do come across older code written in Python 2, it is important to understand the changes and what is required to port the code to Python 3.

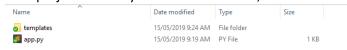
- Print function
- Integer division
- Strings have changed
- Xrange is deprecated
- Error and exception handling has changed
- Banker's rounding

Python 2	Python 3
print 'Hello, World!'	print ('Hello, World!')
print 3 / 2 1	print (3 / 2) 1.5
Strings are ASCII	Strings are Unicode (utf-8)
	NameError: not defined
raise <u>IOError</u> , "file error"	raise <u>IOError</u> ("file error")
except NameError, err:	except NameError as err:
round(16.5) 17.0	round(16.5) 16



#### Lab Exercise: Create a basic Python app to run the Flask web server

- 1. Create a new directory in your project folder called 'templates'
- 2. Move your static HTML page created in the previous exercise to this new folder
- 3. Download the files from <u>Lab 04 Python & Flask</u> directory of the repository and place into your working project folder
- 4. Your project directory should look like this;



- 5. Open a command prompt and navigate to your project directory
- 6. Run the following command
  - > python app.py

```
C:\Users\CRITHT\OneDrive - Dell Technologies\Documents\Sites\Piper-Bootcamp-2019\Lab 04>python app.py

* 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://0.0.0.0:5000/ (Press CTRL+C to quit)
```

7. Open a browser with the following URL

http://localhost:5000



#### Retrospective: Applying concepts of Model, Views, Controllers

Review the app.py code to see how views and controllers are used to implement a template with a default route controller;

Also take note of the port on the last line of code that determines which port the web server will listen on. For testing purposes we are using port 5000, but this can easily be changed to port 80 at any time.



## Lab 5: MongoDB

## Module Objectives:

Setup local MongoDB server & install pymongo module (if not already)
Use Compass UI to connect to DB and test it's functionality
Edit our Python app we created earlier to add dynamic functionality with MongoDB
Review the code and become familiar with the new Python code
Test the functionality of the web app
Save and upload to Github



Lab Exercise: Copy the new code files and overwrite existing application files

- 1. Copy the following files from Lab 05 MongoDB into your project directory
  - /templates
    - default.html
    - submit-photo.html
  - app.py
  - models.py
- 2. Open a command prompt, navigate to your project directory and run the app
  - > python app.py
- 3. Open a browser with the following URL and test uploading an image

http://localhost:5000

4. Use the MongoDB UI (Compass) to check that the title, comments and filename fields are written to the database collection 'photos'

```
_id:ObjectId("5ce2473e2aad45b6ed7fcc1e")
title:"Paris"
comments:"Moulin Rouge"
photo:"Paris_Moulin_Rouge_1920x1080.jpg"
thumb:"Paris_Moulin_Rouge_1920x1080-thumb.jpg"
```



#### Retrospective: Review the implementation of 'models' into our application

Review the models.py code to understand how the form data is being collected and inserted into the MongoDB database.

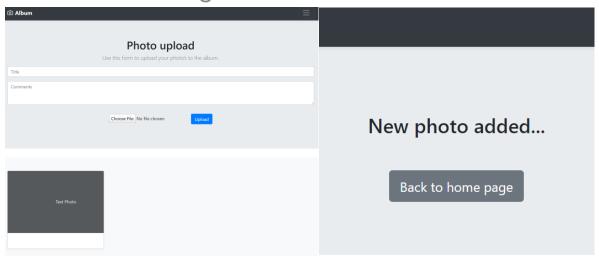
```
models.py
import os
from pymongo import MongoClient
from werkzeug.utils import secure_filename
client = MongoClient('127.0.0.1:27017')
DB_NAME = "mongodb" ##### Make sure this matches the name of your MongoDB database ######
db = client[DB_NAME]
def get_photos():
   return db.photos.find({})
def insert_photo(request):
    title = request.form['title']
    comments = request.form['comments']
    filename = secure_filename(request.files['photo'].filename)
    thumbfile = filename.rsplit(".",1)[0] + "-thumb.jpg"
    db.photos.insert_one({'title':title, 'comments':comments, 'photo':filename, 'thumb':thumbfile})
```

Also take note of the additional route added to the app.py code that provides a new application function to confirm the upload and provide the user with a message before returning to the main page.

## **D¢LL**Technologies

```
import os
from flask import Flask, render_template, request
from models import get_photos, insert_photo
app = Flask(__name__)
@app.route('/')
def home():
   album_photos = get_photos()
   return render_template('default.html',album_photos=album_photos,url="home")
@app.route('/upload', methods=['POST'])
def upload():
   insert_photo(request)
   return render_template('submit-photo.html')
if __name__ == "__main__":
    app.run(debug=False, host='0.0.0.0', port=int(os.getenv('PORT', '5000')), threade
```

# **D¢LL**Technologies



At this point, we still do not have anywhere to store the images and are only inserting the name of the image into the database. This is why you do not see the actual image on the web page yet.

In our next lab we will add some object storage to upload the users images and use the database to store the URL link to the images.



## Lab 6: ECS

Module Obj	ectives:
------------	----------

Setup object storage with ECS test drive
Use S3 Browser to connect to ECS and create a bucket
Edit our Python app we created earlier to add object storage functionality with S3 compliant Python module
Review the code and become familiar with the Python code
Test the functionality of the web app
Save and upload to Github



#### Lab Exercise:

- 1. Copy the following files from "Lab 06 ECS" into your project directory
  - /templates
    - default.html
    - photo.html
    - submit-photo.html
  - app.py
  - models.py
  - config.py (rename the file from the provided config-example.py file)
  - /uploads (create this as a new directory)
  - .gitignore
- 2. Retrieve your ECS credentials from the following URL; <a href="https://portal.ecstestdrive.com/">https://portal.ecstestdrive.com/</a>
- 3. Edit the config.py file with your credentials

#### AWS S3

- 4. Install Boto3 & Pillow (if not already installed)
  - > pip install boto3
  - > pip install pillow
- 5. Run your app
  - > python app.py
- 6. Test your app and upload an image;

http://localhost:5000



#### Retrospective: Review code and note implementation of Boto3 & Pillow

Take a look at the app.py code and notice the additional route used for displaying a full resolution image on a separate page and the additional call to the upload\_photo function provided by the models.py file.

```
import os
from flask import Flask, render_template, request
from models import get_photos, insert_photo, upload_photo
app = Flask(__name__)
@app.route('/')
def home():
    album_photos = get_photos()
    return render_template('default.html',album_photos=album_photos,url="home")
@app.route('/upload', methods=['POST'])
def upload():
    insert_photo(request)
    upload_photo(request.files['photo'])
    return render_template('submit-photo.html') # Return a page to inform the user of
@app.route('/photo/<path:photo>')
def photo(photo):
    return render_template('photo.html',photo=photo)
if __name__ == "__main__
    app.run(debug=False, host='0.0.0.0', port=int(os.getenv('PORT', '5000')), threaded=True)
```

Open the models.py file and note the additional upload\_photos function that uses the Boto3 module to upload the image into ECS test drive using the standards based S3 protocol and the Pillow module that converts the full resolution image into a smaller thumbnail image to be used in the gallery page of our application.

## **D&LL**Technologies

```
def upload_photo(file):
   ecs_endpoint_url = ecs_test_drive['ecs_endpoint_url']
   ecs_access_key_id = ecs_test_drive['ecs_access_key_id']
   ecs_secret_key = ecs_test_drive['ecs_secret_key']
   ecs_bucket_name = ecs_test_drive['ecs_bucket_name']
   session = boto3.resource(service_name='s3', aws_access_key_id=ecs_access_key_id, aws_secret_access_key=e
       imgraw = Image.open(f)
       img = imgraw.convert("RGB")
       img.thumbnail(size)
       thumbfile = filename.rsplit(".",1)[0] + "-thumb.jpg"
       img.save("uploads/" + thumbfile,"JPEG")
       img.close()
   img = None
   session.Object(ecs_bucket_name, filename).put(Body=open("uploads/" + filename, 'rb'), ACL='public-read')
   session.Object(ecs_bucket_name, thumbfile).put(Body=open("uploads/" + thumbfile, 'rb'), ACL='public-read
   os.remove("uploads/" + filename)
   os.remove("uploads/" + thumbfile)
```

The credentials are kept as an external file and the .gitignore file ensures that the credentials are not uploaded accidently to your Github repository.

```
config-example.py x

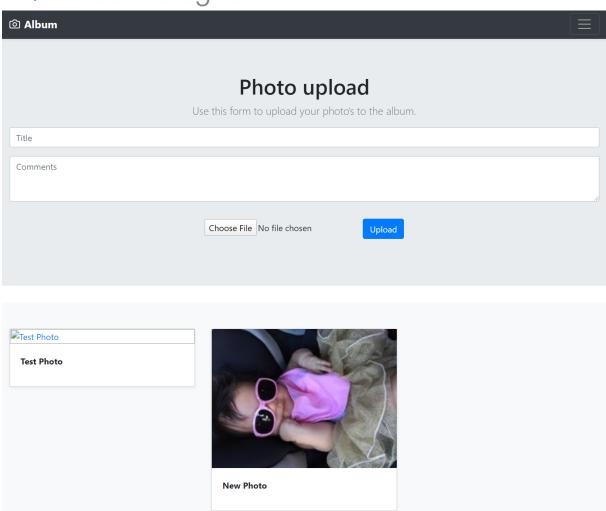
ecs_test_drive = {
    'ecs_endpoint_url' : 'https://object.ecstestdrive.com',
    'ecs_access_key_id' : '1234-your-unique-number-5678@ecstestdrive.emc.com',
    'ecs_secret_key' : 'your-long-secret-key-from-ECS-testdrive-portal',
    'ecs_bucket_name' : 'photo-album'
}

.gitignore

1 config.py
2 __pycache__
```

When you test your new app, it should link your image and display the thumbnail similar to the below;

# **D¢LL**Technologies





### Lab 7: PWS & mLab

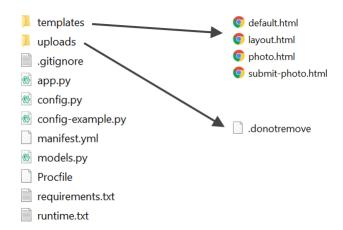
### Module Objectives:

Upload app to PWS PaaS platform
Create a MongoDB as a Service instance
Edit our Python app we created earlier to leverage mLab service
Review the code and become familiar with the Python code
Test the functionality of the web app
Save and upload to Github



#### Lab Exercise (Part A): Create the pre-requisite platform files and upload the app to PWS

Copy all files from "Lab 07 – PWS & mLab" folder and place into your working directory
 Overwrite the existing files and confirm the folder looks like this;



- 2. Open a command prompt and navigate to your project
  - > cd <path of your project>
- 3. Login to PWS with your account creds
  - > cf login -a https://api.run.pivotal.io
- 4. Push the app to PWS
  - > cf push
- 5. Your app will fail to start and crash...... Why?

```
Waiting for app to start...

Start unsuccessful

TIP: use 'cf logs tc-photo-album --recent' for more information

FAILED
```



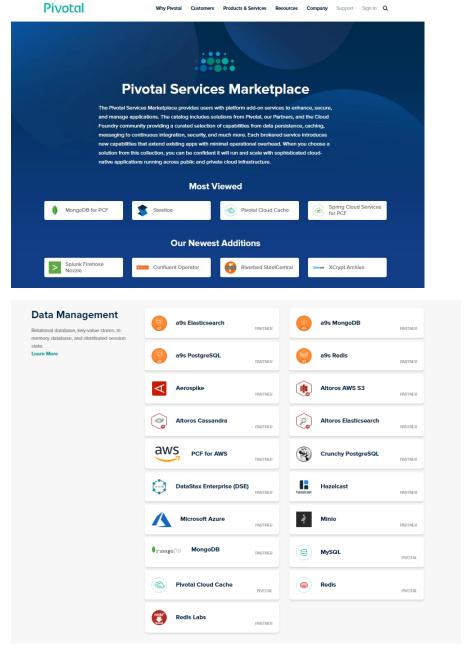
#### Pivotal Services Marketplace

If you haven't already guessed, the reason the application failed in the previous lab is because the code is expecting to connect to a MongoDB database that we have not provisioned into the PWS environment yet.

The Pivotal Services Marketplace is available for both the private deployment of Pivotal Cloud Foundry and the public hosted service provided by PWS.

Some of the packaged services that have been developed to be deployed at found at;

https://pivotal.io/platform/services-marketplace



Student Lab Guide



#### Pivotal Web Services Marketplace

The services that are available in PWS are a subset of the marketplace and are targeted at some of the most common services requested by customers.

To view all available services, login to <a href="https://console.run.pivotal.io/">https://console.run.pivotal.io/</a> and browse to the services section of your organization.

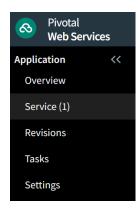
For our application, we will deploy a MongoDB service provided by mLab as a hosted multi-tenanted database instance.

Continue to the next part of the lab (Part B) for instructions to deploy and bind the mLab service to our application.

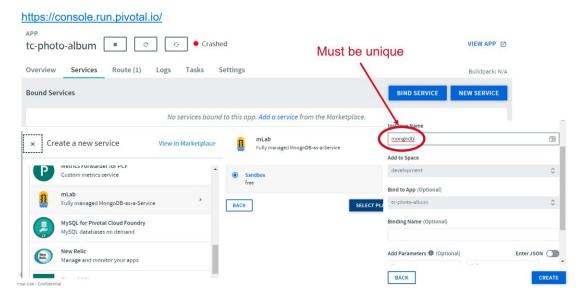


Lab Exercise (Part B): Add the external MongoDB service from the PWS Marketplace

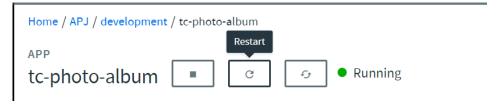
- 6. Login to the PWS console at <a href="https://console.run.pivotal.io">https://console.run.pivotal.io</a>
- 7. Find your deployed application and click on the 'Service' menu option on the left



8. Click on the 'NEW SERVICE' button and create an mLab instance to bind to your app



- 9. Use the default options for everything except the Instance Name, which needs to be unique
- 10. Restart your app from the console if required



11. Open the app

Student Lab Guide



#### 12. Test functionality and upload an image

Overview	Service (1)	Route (1)
Routes		
https://tc-photo-album.cfapps.io ☑		



Retrospective: Review code and take note of the 'VCAP\_SERVICES' section in models.py

The application is now running in a resilient PaaS platform with the ability to scale on demand with autoscaling or manually with admin intervention.

Take note of the models.py application code to retrieve the environment variables with the MongoDB credentials setup in the previous Lab Part B section.

```
if 'VCAP_SERVICES' in os.environ:
    VCAP_SERVICES = json.loads(os.environ['VCAP_SERVICES'])
    MONGOCRED = VCAP_SERVICES["mlab"][0]["credentials"]
    client = MongoClient(MONGOCRED["uri"] + "?retryWrites=false")
    DB_NAME = str(MONGOCRED["uri"].split("/")[-1])
```

Take note of the platform specific files required for the PaaS environment to configure and setup the containers with the right pre-requisites for the application to run.

