HW2: Deploy to Cloud with CI/CD Pipeline

Welcome to the second half of the project assignment for NYU CSCI-GA. 2820-001 DevOps and Agile Methodologies. This assignment is worth **20** points toward you total grade and has a duration of 6 weeks and must be completed in 3 Sprints each having a duration of 2 weeks. The due dates for the sprints are the following Milestones:

- Sprint 2: Milestone is due November 4th 2020 (in two weeks)
- Sprint 3: Milestone is due November 18th 2020 (in four weeks)
- Sprint 4: Final Milestone is due **December 9th 2020** (in seven weeks)

I will continue to accept submissions until Friday December 11th but they will incur a **10%** grade penalty (i.e., you will start with 18 points instead of 20). No submissions will be accepted after the second deadline. Notice that the last sprint is 3 weeks in duration due to the Thanksgiving Holiday in the US.

Create a New Sprint Milestone

Start by cleaning up from your previous Sprint. Move all Stories that are Done to Closed and then mark the Sprint Milestone as Closed. Create a new Sprint Milestone and begin adding Stories for each of the items in this homework. Conduct scrum meetings as in the previous homework for Backlog Refinement, Sprint Planning. Daily Stand-Ups, etc. Make sure that you follow good Social Coding and Agile practices by working in feature branches, assign stories to yourself and set them to In Progress while working on them and then issue Pull Requests and move them to Done when complete and Close them after your Sprint Review. Check that your Burndown chart reflects this progress because I will be checking the burndown during the sprints. We will conduct a mini **Sprint Review** at the beginning of class **November 4, 18,** and **December 9.** This is when you will present your status and demo any features to me. Remember the purpose of a sprint is to have running code at the end that you can demo some business value.

Business Scenario for Homework 2

Congratulations on competing a local version of your RESTful service but it's time to let the world see your work. The business has decided to deploy it to the public Cloud. Also, in an effort to gain better business agility, your company realizes that manually deploying your application is not scalable. To achieve continuous delivery you need to really embrace DevOps and create automated Testing integrated with an automated CI/CD DevOps Pipeline in the Cloud.

In order to be successful, the following requirements from the business must be satisfied:

1. Add Continuous Integration to every Pull Request
2. Deploy to the public Cloud
3. Implement Automated BDD Testing
4. Refactor using Flask-RESTPlus
5. Add Swagger Docs
6. Build an automated CI/CD DevOps Pipeline

You may satisfy these requirements in any order that you want. You may want to set up a simple 2 stage Pipeline first to make deploying easier and then go back and add the last 2 stages later. It's up to you.

(Optional)

Reorganize the Agile Development Squads

Oh... one more thing. Your company has also reorganized to better adapt to changing business conditions. This means that your agile development squad will encounter some personnel changes. ;-)

Two members from each team have been reassigned to another team. In the true spirit of being agile, we must adapt. The GitHub repository owner and IBM Cloud Organization owner (or one other member if this is the same person) will stay on the current squad. The other two members will join another squad. I will post the new squad membership to NYU Classes.

Luckily, you have all been practicing Social Coding and using a standard set of DevOps tools, and following the Scrum process. Other then getting use to the new code base and team members, you should hit the ground running by cloning your new team's repository, running `vagrant up` and you should be ready to assign an Issue to yourself and start working. This will help everyone learn the entire codebase better.

Requirement 1: Add Continuous Integration

In this requirement you will add Continuous Integration (CI) to your repository. Every Pull Requests should run both your TDD test cases from homework 1, and eventually your BDD test cases that you will create in requirement 3. The Pull Request will tell you if all of the tests have passed and you should NOT accept a Pull Request with tests that are failing. This will ensure that the master branch is always deployable. Use Travis CI to run your tests on every Pull Requests.

To successfully satisfy this requirement you will need Stories that:

- Create a free account on Travis CI (travis-ci.org) for your GitHub organization.
- Create a `.travis.yml` file and configure Travis CI to run test cases every time someone pushes to the master branch or submits a Pull Request.
- Set up Travis CI to monitor your GitHub repository for push and pull requests.
- Add a Travis CI badge to your `README.md` file so that everyone will know if the build is broken.
- All the while maintaining 95%+ code coverage
- Run your test cases to make sure that your tests work before creating pull requests (because if you don't the pull request will!)
- (optional) Integrate your Pull Requests with your Slack channel so that any time someone makes a pull request, the channel is notified so someone can approve it.

What I am look for is that your Pull Requests run your unit tests and no pull request gets merged that will break the `master` branch.

This part is worth **2 points**.

Requirement 2: Deploy to the Cloud

In this requirement you will deploy your application to the cloud. In particular the Cloud Foundry platform in IBM Cloud. For your persistence (e.g., PostreSQL, MySQL, etc.) you will need to bind the equivalent IBM Cloud services to your application. You also want to take advantage of the

horizontal scaling capabilities of the cloud so be sure to deploy at least 2 instances and not just an individual instance.

To successfully satisfy this requirement you will need to:

- Someone on the team must must invite other members of your team to their IBM Cloud account. I will provide instructions on NYU Classes that will tell you specifically how to do this so that Cloud Foundry works. (`00-invite-to-ibm-cloud.pdf`)
- Create a Python Flask Cloud Foundry application in IBM Cloud - US South.
- Naming is very important so please follow the following format:
- Use the singular version of your resource name (e.g., orders becomes `order`) prefixed by `nyu-` and suffixed by `-service-f20`. For example: if you are creating the /orders API then please name your IBM Cloud service `nyu-order-service-f20` (The full URL will then be `nyu-order-service-f20.us-south.cf.appdomain.cloud`)
- Update your Vagrantfile to download and install the IBM Cloud CLI (*hint: an example of this is in the `lab-bluemix-cf` repo*)
- Create the necessary Cloud Foundry metadata files for deployment (`manifest.yml`, `Procfile`, `runtime.txt`) file so that IBM Cloud will know how to deploy your application.
- Modify your application to get the database credentials from the `VCAP_SERVICES` environment variable if you are using an IBM Cloud database like Cloudant or IBM DB2, or `DATABASE_URI` if you are using a 3rd party database like PostgreSQL (from ElephantSQL.com). (hint: see `models.py` in `lab-bluemix-cf` for an example)
- Add `honcho` to your requirements.txt file and make sure the the command: `honcho start` works with your `Procfile`.
- Use `ibmcloud cf push` to push your application to IBM Cloud
- Go to the URL of your service and make sure that it works

What I am looking for here is a working service that is running on IBM Cloud. I also need to be able to start your application in your vagrant VM with the command: `honcho start` as learned in class.

This part is worth **3 points**.

Requirement 3: Implement Automated BDD Testing

In this requirement we will need a Behavior Driven Development test suite that runs integration tests using Selenium. This will be the gate in our DevOps Pipeline to deploy to Production. This part will require a Web UI to test. A sample UI will be included in the GitHub example from class (`lab-flask-bdd`). Feel free to modify it for your resource or build you own if you have the UI skills.

To successfully satisfy this requirement you will need to:

- Create a simple single page UI for your application
- Create a BDD `.feature` file and accompanying `steps.py` files to test your RESTful API from the outside in.
- Write the feature using `Gherkin` syntax that is understood by the behave tool
- There should be at least 7 Scenarios in the feature file, one for each of Create, Read, Update, Delete, List, Query, and Action
- Run these using `behave` until all scenarios are green (passing)
- Use Selenium to conduct your UI testing. There should be no direct connection to your service from your BDD tests. It should interact with the web interface only!
- Write Stories (as Github Issues) to plan the work needed to create the test cases and add them to your Sprint Backlog.
- Move these Stories through the ZenHub pipeline as you work on them just like you did in homework 1

What I am looking for is a demonstration of your understanding of how write good feature scenarios that you can talk about with your customer. It is important that these be expressed in terms that the stakeholder can understand which means the syntax should be customer friendly and not look like technical jargon.

This part is worth **5 points.**

Requirement 4: Refactor using Flask-RESTPlus

Since you have been practicing Test Driven Development and have at least 95% test coverage you should be confident about refactoring your code. We want to move from a basic Flask application to one that uses **Flask-RESTPlus**. This will allows you to provide Swagger documentation with very little effort in requirement 5. The `lab-flask-restplus-swagger` repository is the example code that you should use convert to Flask-RESTPlus which we will cover in class.

To successfully satisfy this requirement you will need to:

- Refactor your micro service to use Flask-RESTPlus following the file layout outlined in `lab-flask-restplus-swagger`.
- Make sure that all HTTP errors return json error messages and not html.
- Run your test cases and make sure that your Flask-RESTPlus implementation still works like the original.

What I am look for is good use of Flask-RESTPlus and that he same test cases should still work as expected. I will throw bad data at your service and I expect to not see 500 errors (400 errors are fine). ;-)

This part is worth **3 points**.

Requirement 5: Add Swagger Docs

Once requirement 4 has been met it should be easy to add Swagger documentation to your REST API.

To successfully satisfy this requirement you will need to:

- Add Swagger annotations to your service to document your REST API using Flask-RESTPlus as shown in class.
- The documentation API must be available at: `/apidocs/index.html`
- Add Swagger data models to document your API payloads
- Ensure that you Swagger docs work correctly and specify your data model so that others know what to send to your API

What I am look for is good use of Flask-RESTPlus annotations and that he same test cases should still work as expected. I also expect working and accurate Swagger docs.

This part is worth **2 points**.

Requirement 6: Build an automated CI/CD DevOps Pipeline

In this requirement you will add a DevOps Pipeline to deploy your service to the IBM Cloud automatically. It should also run your TDD unit tests and your BDD integration tests. You will set up a DevOps Pipeline that will have 4 stages (Build, Deploy, Test, Production). The Build stage will also run TDD (unit) tests and then call the Deploy Stage to deploy to a Development instance. Then the Integration Test Stage will run you BDD tests against the running development instance, and if all of the tests pass, it will deploy to a Production instance. Your Dev and Prod instances should use different database instances if the free IBM Cloud account allows this (some services you can only have one of with the free trial).

To successfully satisfy this requirement you will need to:

- Set up the DevOps Continuous Delivery Pipeline in IBM Cloud to build and deploy your project when the master branch changes.
- Connect this DevOps pipeline to the GitHub repository for your project via a webhook.
- Your Pipeline must have 4 stages: Build, Dev, Test, Prod
- Add a Unit Test Job to the Build Stage of the pipeline to run your units tests with `nosetests`
- Add a Dev Deployment Stage to the pipeline to deploy to your Dev space if the unit tests pass
- Add a Integration Test Stage to the pipeline to run your integration tests with `behave` against the Dev deployment
- Create a new Space in your Cloud Foundry account called: `prod`
- Add a Production Deploy Stage to the pipeline to deploy to the production space if all of the integration tests pass.
- Write Stories (as Github Issues) to plan the work needed to create the necessary integrations and add them to your Backlog.

- Move these Stories through the ZenHub pipeline as you work on them just like you did in homework 1

I'm looking for an operational pipeline with 4 stages that is trigger by a commit to the master branch on your GitHub repo that runs the TDD unit tests, and if successful deploys to your Dev space, then runs the BDD integration tests against the service running in dev, and if successful deploys the service to the Prod space.

This part is worth **5 points**.

Final Submission:

For the final submission on the December 9, 2019, please submit:

1. The URL of your service running on IBM Cloud in dev
2. The URL of your service running on IBM Cloud in prod
3. The URL of your DevOps Pipeline in IBM Cloud

Post any questions to our Slack `#homework` channel.