**DevOps Implementation**

* **Drone** - Drone is a Continuous Integration platform built on container technology.
* **Docker** - Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.
* **Ansible**: Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.
* **Blue-green deployment:** Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with the live environment serving all production traffic. This technique can eliminate downtime due to application deployment.

**Flow**

Our workflow is:

* + Developer pushes changes to master/feature in GIT
  + GIT notifies Drone through the webhook configured
  + Drone pulls master/feature, runs unit tests, builds a Docker image, pushes the image to the Docker Hub and then uses SSH to do a rolling deployment to servers.
  + Continuous Testing, Blue-Green deployment and Continuous Monitoring can be included as a part of the process.

**Sample Application – Django**

Compose Section:

1. Include all the dependencies that our build has.
2. Say, MySQL database for the sample Django application. Any other dependencies of the project can be included in this section.

Build Section:

1. Drone connects the containers in build to the compose.
2. Unit tests are executed and as we are using Python 3.5, we tell Drone to pull down the Python 3.5 image from the Docker Hub.
3. Environment variables for the username and password to use to connect to the linked MySQL database container from the above compose section.

**.drone.yml**

compose:

database:

image: mysql

environment:

- MYSQL\_USER=admin

- MYSQL\_PASSWORD=admin

build:

image: python:3.5

environment:

- DB\_USER=admin

- DB\_PASSWORD=admin

commands:

- pip install -r requirements.txt

- python manage.py test

Publish Action:

1. The Docker image is built and tagged with both production as well as the build number ("$$BUILD\_NUMBER").
2. This is useful in cases of build failures and in roll back situations.
3. Docker Hub credentials can be encrypted and stored in .drone.sec file, so that we can reference them using variables.

publish:

docker:

username: $$DOCKER\_USERNAME

password: $$DOCKER\_PASSWORD

email: $$DOCKER\_EMAIL

repo: django/{artifact\_id}

tag:

- production

- "$$BUILD\_NUMBER"

file: Dockerfile

insecure: false

when:

branch: master

Deploy Action:

1. Drone SSH’s into our servers one at a time to execute a shell script.
2. There is a sleep time of 5 seconds between deploying to each server to make sure the load balancer has allotted the servers properly.

**deploy**:

ssh:

host:

- $$DOCKER\_HOST\_1

- $$DOCKER\_HOST\_2

user: $$SSH\_USER

port: 22

sleep: 5

commands:

- sh deploy-app.sh

when:

branch: master

ssh:

host:

- $$DOCKER\_HOST\_1

- $$DOCKER\_HOST\_2

user: $$SSH\_USER

port: 22

sleep: 5

commands:

- sh deploy-app.sh

when:

branch: master

**deploy-app.sh**

#!/bin/bash

set -e

docker pull docker-registry/image-name:latest

docker stop image-name

docker rm image-name

docker run –name django\_app [OPTIONS] docker-registry/django\_app [COMMAND] [ARG...]

/django\_apprepo

|

|---- .drone.yml

|

|---- /.drone

| |

| |---- compose.sh

| |---- build.sh

| |---- publish.sh

| |---- deploy.sh

| |

| |----

| |---- CODE..