SECTION 07

Class Project

THE PROJECT MAP

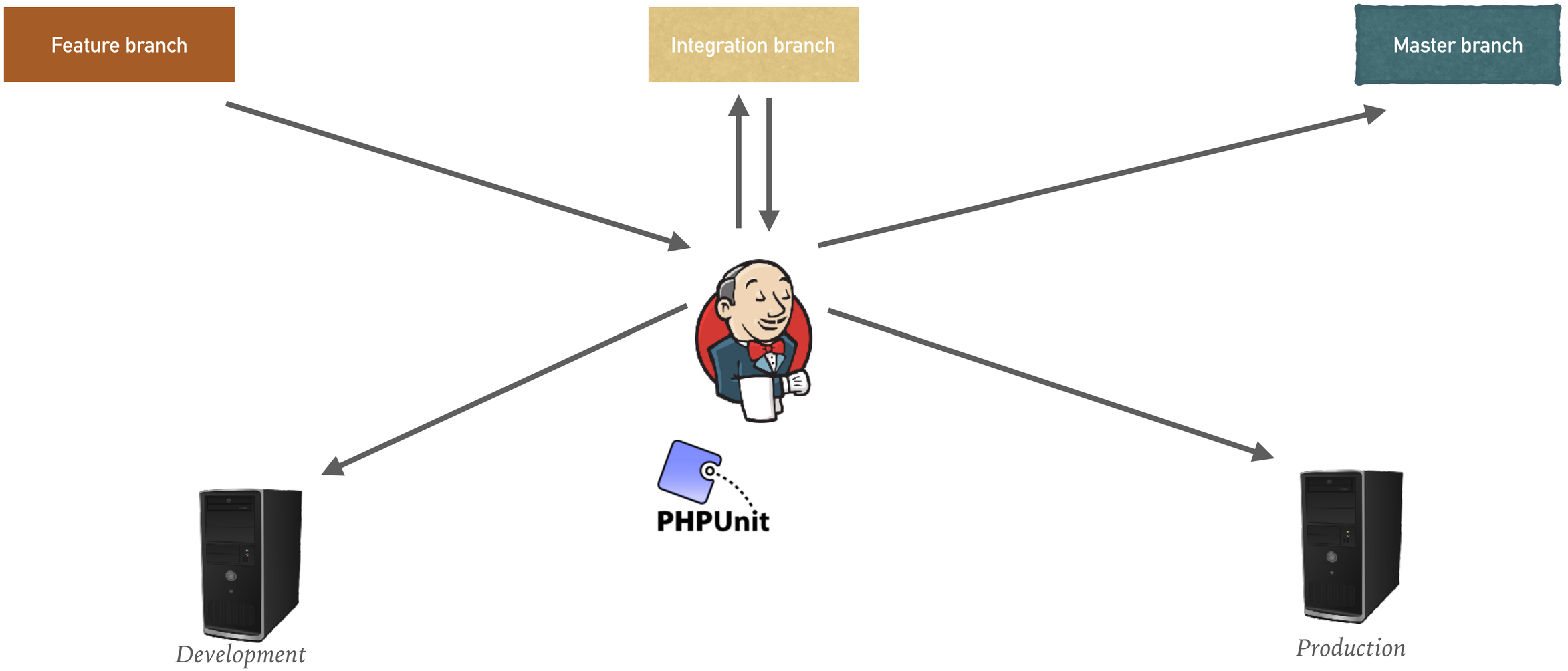
➤ The project is a web-based booking application. It contains the following components:

Environment	Number of machines	Purpose
Client	1	Mimics the developer machine used to write the application code. Also used for running Ansible against the other environments.
Testing	2	The application server hosting the web application and the database server hosting the backend database
Production	2	The application server, the database server.

THE TOOLS THAT ARE USED

- Vagrant for infrastructure provisioning
- Ansible for configuration management
- Git for version control, GitHub as a central repository
- Jenkins as a CI/CD tool

THE PROJECT BRANCHES



FIRST STEP: THE INFRASTRUCTURE

- The client machine is a Centos 7 Vagrant machine with the following components:
  - Ansible
  - Git
- The application server is a Centos 7 Vagrant machine with the following components:
  - Apache
  - PHP 7
  - Git
  - Swift application (the CodeIgniter project used)
- The database server is a Centos 7 Vagrant machine with the following:
  - MySQL
- The CI/CD server is an Ubuntu Vagrant machine containing the following:
  - Jenkins with the following plugins:
    - SSH host
    - GitHub
  - PHPUnit testing tool
  - Selenium Python web driver
- A testing environment will contain a clone of the application and database machines to be used by Jenkins

PROVISION THE MACHINES

- Remove the existing CodeIgniter installation as it will be replaced with the developed one.
- Install Git
- Replace the code for deploying CodeIgniter with this code:

```
- name: Deploy the swift application
      git:
        repo: https://github.com/abohmeed/swift
        dest: /var/www/html/CodeIgniter-3.1.5/
```

- Add the following to database.yml:

```
- name: Import bookings.sql
      mysql_db:
        state: import
        name: all
        target: /tmp/bookings.sql
```

- Run ansible from the client machine to re-configure the database and the application machines:

```
ansible-playbook /vagrant/application.yml
ansible-playbook /vagrant/database.yml
```

PROVISIONING THE JENKINS MACHINE

- Add the following to the end of the Vagrant file (before the last `end` keyword):

```
config.vm.define "jenkins" do |jenkins|
    jenkins.vm.box = "bento/ubuntu-16.10"
    jenkins.vm.hostname = "jenkins"
    jenkins.vm.network "private_network", ip: "192.168.33.40"
    jenkins.vm.provision "ansible" do |ansible|
      ansible.playbook = "jenkins.yml"
    end
  end
```

- The jenkins.yml playbook is displayed next

THE JENKINS.YML PLAYBOOK

```
➤ ---
  - hosts: jenkins
    become: yes
   tasks:
      - name: Install the repo key
        apt_key:
          url: https://pkg.jenkins.io/debian/jenkins-ci.org.key
          state: present
      - name: Add Jenkins repository
        apt_repository:
          repo: 'deb http://pkg.jenkins.io/debian-stable binary/'
          state: present
      - name: Install Jenkins
        apt:
          name: jenkins
          state: present
          update_cache: yes
      - name: Start and enable the service
        service:
        name=jenkins state=started enabled=yes
      - name: Install required packages
        apt: name={{ item }} state=present
        with_items:
          - python-pip
          - git
          - phpunit
      - name: Install Selenium testing framework
        pip:
          name: selenium
```

REQUIRED PLUGINS

- Once Jenkins is up and running, you need to make sure the following plugins are installed:
  - GitHub
  - SSH
  - SSH Agent

UNDERSTANDING UNIT TESTING

- Jenkins can work with a lot of testing frameworks. You can even create your own testing tools and integrate them with Jenkins.
- For the integration tests, are going to use PHPUnit testing tool. It's simple and easy to install. For CodeIgniter, I've used the implementation provided by this URL https://github.com/kenjis/ci-phpunit-test. Just create a directory tests in application directory of CodeIgniter and drop the files there. Now you can use CodeIgniter with PHPUnit.
- Let's write a very simple test to ensure that the home page will respond with code 200, which is an HTTP code that indicates a normal response.
- In the application/tests/controllers directory edit Welcome_test.php and add the following:
  class Welcome_test extends TestCase

```
{
  public function test_index()
    {
        $this->request('GET', '/welcome');
        $this->assertResponseCode(200);
    }
}
```

- Now, clone the application on your Jenkins server just to test whether or not the testing work. In the tests directory run phpunit. You'll see that everything is fine.
- Now on the client machine make a destructive change to the controller and push your changes.
- In your Jenkins machine, pull the changed code and write phpunit again. Observe how the test fails.

CREATE THE REQUIRED BRANCHES

- We need to create two branches to test changes to our code: the feature branch to add new features and the integration branch to integrate those features after testing them.
- On client machine, clone the application by running the following command:
  git clone https://github.com/abohmeed/swift
- Inside the application directory, create the feature branch by typing the following commands:

```
git checkout -b feature
git push --set-upstream origin feature
```

- Do the same for the integration branch

```
git checkout -b integration
git push --set-upstream origin integration
```

CONTINUOUS TESTING

- Create a new job called swift-feature
- In Source Code Management select Git
- Add the repository URL: https://github.com/abohmeed/swift
- Choose the feature branch
- In the Build triggers click on Poll SCM and specify the desired schedule
- In the Build are select a new build step and choose shell script. Add the following:
```
cd application/tests/
phpunit
```
- In the post build actions choose publish Unit test result and add the following application/tests/build/logs/junit.xml
- Save the job
- Test it by making changes and pushing them to GitHub and running the job.

CONTINUOUS INTEGRATION

- The integration job will always run after the feature job runs.
- It will only take an action if the feature job succeeds (the tests pass).
- The job will merge the changes done to the feature branch to the integration branch and push those changes to GitHub.
- Add a descriptive name for the job. Let's say feature_integration
- In Source Code management specify that it is a Git repository
- Add the repository URL https://github.com/abohmeed/swift and this time add also the credentials as they will be needed to push the changes back.
- Specify */feature as the branch to track
- In Build triggers check "Build after other projects are built" and add the feature_test (or whatever name you chose for the first job). Make sure that Trigger only if build is stable is checked.
- In the Build section add the following steps (Execute shell):
  - Step 1: cd application/tests
    phpunit
  - Step 2: git checkout testfeature
    git pull
    git checkout integration
    git merge testfeature
- In the post build actions, choose Git publisher and check on "Push only if build succeeds". Choose the integration branch as the remote branch and origin as the remote name.
- Save the job and try to make changes to the code on the client machine, push them to GitHub on the feature branch and observe Jenkins behavior when the job is triggered.

CONTINUOUS DELIVERY

- The second job will work after the previous one is complete and will actually deploy the changes to the development environment.
- Before we can do that we need to create an SSH site to define the machine where the code will be delivered for further testing. In Manage Jenkins choose configure Jenkins and go to SSH remote sites.
- Add the details of the SSH machine including the credentials and the remote location.
- Now create a new job that is copied from feature_integration but with the following changes:
  - Build after feature_integration
  - Build execute shell will contain only the testing phase
  - The second build step will be execute SSH script on remote hosts. Choose the SSH site already configured and add the following commands:

```
cd /var/www/html/CodeIgniter-3.1.5/
git checkout integration
git pull
```

CONTINUOUS MASTER INTEGRATION

- Now that the code has been tested when committed and tested before integration, let's deploy it to the development machine.
- In the Vagrantfile, duplicate the code used to create the web machine and change the IP address accordingly. This will create a clone of the web application machine for our testing purposes.
- Add a new job in Jenkins called feature_master
- Configure the job to be a clone of the feature_integration.
- This will make the job take all the properties of the feature_integration one. We need to make the following changes:
  - The branch to track will be */integration
  - Build after: feature_integration
  - The first build step will remain as is, the second will contain the following:

```
git checkout integration
git pull
git checkout master
git merge integration
```

  - The post build actions will publish to master

CONTINUOUS DEPLOYMENT

- This job will work after the previous one is complete and will actually deploy the changes to the production environment.
- Before we can do that we need to create an SSH site to define the machine where the code will be delivered for further testing. In Manage Jenkins choose configure Jenkins and go to SSH remote sites.
- Add the details of the SSH machine including the credentials and the remote location.
- Now create a new job that is copied from feature_integration but with the following changes:
  - Build after feature_delivery
  - Build execute shell will contain only the testing phase
  - The second build step will be execute SSH script on remote hosts. Choose the SSH site already configured and add the following commands:

```
cd /var/www/html/CodeIgniter-3.1.5/
git checkout master
git pull
```