For building webapp: Application runs on Java 8 on Apache TomCat 8

1) For building webapp to parse “https://boards.greenhouse.io/definiens#.WNCM4lUrLX4”, I’ve used Java with Spring framework. Spring handles request mapping the URLs such that requests are directed to the designated function.
2) Maven has been used to handle all the dependencies (external jars/libraries). I did this in order to avoid bloating up the application, and to avoid adding/downloading the jars manually
3) For parsing HTML page, I used JSoup java library. Since, the common thing in all the job postings was that they were all in the form of links, I parsed HTML page for links. I stored all the links in a List. I searched for presence of keyword “DevOps”, in the returned list.
4) The webapp can be accessed at IP:8080/DefiniensJobSearchApp/devops. It returns message “DevOps Position Available” or “Sorry! No DevOps Roles Found” depending upon presence of position

For running webapp:

1) Maven was configured to package the application in the form of a .war file, which is stored in targets directory
2) In case, any changes are made to the application code, the application will have to be built again, which can be performed using command: maven clean install. This would re-generate the .war file
3) Application can be further tested for other webpages by changing url in file Controller.java inside package com.nivedita.springrest (DefiniensJobSearchApp\src\main\java\com\nivedita\springrest\Controller.java)

Provisioning:

AWS Config

For the basic setup to start provisioning, I created an Ubuntu 14.04 LTS server VM on AWS and installed Ansible on it. This would serve as my controller VM that I’ll use to provision other VM instances. I have a trial account of AWS that I used for this purpose. I created a security group with an inbound rule allowing traffic to this VM only from my IP. This was done to prevent access from any un-trusted machine. In order to access AWS services programmatically through scripts I installed AWS CLI on this machine. This machine would act as controller, for provisioning other VMs on the cloud. It requires Python 2.6.2+ to run. I added my aws access key and id in .aws/config.

For “to-be-provisioned” VMs, I created an IAM policy to give zero access to any of the resources in AWS and attached it to a role. New VM instances will be provisioned with this policy, this has been done keeping in mind the scenario that if these instances gets compromised, attacker should not have the permission to make changes to our AWS account. The policy document contains the rule to deny access to any of the resources. This was configured from AWS web console to contain rule:

```
 {
"Action": "*",
"Effect": "Deny",
```

```
"Resource": "*"
 }
```

Ansible Config: The main reason for using Ansible is that firstly, I am a bit familiar with it. Secondly, it works with minimum setup. So, it saved me from installing a config management tool on every VM instance that is going to be provisioned.

A quick overview of what I've explained in next few points:
I first set up some variables that will apply to all tasks in playbook, then I launch Ubuntu 14.04 LTS server instance and create a user on it, after which I install Java 8 and Apache tomcat 8 on newly created VM instance, followed by installing webapp .WAR file in the server. At the end of the script, we have a VM instance running DevOps job search webapp.

1) Group_vars/all.yml contains some group level variables. I'm basically setting the values of some variables that will apply to any task in the playbook, unless they are modified at the task level. It configures variables such as security group, instance type, region, default ami, iam_profile.

2) The starting point is webapp.yml which launches an AWS instance and applies various roles on this instance.

3) immutable/roles/launch/tasks/main.yml contains task to first launch a new instance. It gets values for variables such as region, key_pair, group, image, etc. from group_vars. After launching the instance, it is added to a host group and waits for the instance to boot

4) immutable/roles/base-setup/tasks/main.yml creates a user "npaul" in newly launched instance, configures sudo privilege for this user, ensures that apt cache is updated and installs some packages such as git etc.

5) /immutable/roles/oracle-java/tasks/main.yml installs java 8 on the VM instance. Since default package available for Java in Ubuntu 14.04 LTS server is Java 7 and my webapp uses Java 8, I took reference from https://github.com/malk/ansible-java8-oracle/blob/master/tasks/main.yml on how to install Java 8 with ansible on Ubuntu 14.04

6) /immutable/roles/tomcat-ubuntu/tasks/main.yml installs apache-tomcat 8 on VM instance. It first creates a directory to install tomcat packages, configures it with required permissions. Tomcat related packages and libraries are then downloaded from package archive. Tomcat also requires certain users such as admin to be configured first, these configurations are also performed in this task. These packages are installed after they are downloaded successfully. Then, certain environment variables such as CATALINA_OPTS and JAVA_OPTS required by tomcat are set up. For this part as well, I took reference from internet

7) immutable/roles/webapp-setup/tasks/main.yml just copies the webapp WAR file from files folder to webapp folder inside tomcat8 and restarts the server

8) ansible.cfg in the root of my ansible playbook directory contains settings for ssh connection to remote hosts (provisioned VMs)

The whole code is fully functional: the webapp as well as the Ansible part

TO-DO: In order to allow only SSH access and web server ports I thought of using Ansible ec2 group module: http://docs.ansible.com/ansible/ec2_group_module.html. Another way could have been: create a new security group from AWS web console 'new-sec-group' that allows only TCP protocol on port 22 and 8080 from any host,and just add that security group to the VM while provisioning using Ansible. This can be done as:

In roles/launch/tasks/main.yml
group: "{{ security_groups }}"

In group_vars/all.yml
security_groups: ['new-sec-group'])

However, I did not have the time to do this.