QA Consulting

QACDEVPUP – DEPLOYMENT PROJECT

DEPLOYING DOCKER CONTAINERS TO THE CLOUD

Team JPEJS

Jessica Taylor-Ashley

Pawan Uppadey

Elijah Zadok

James Awobayiku

Syeb Chowdhury

Contents

Project Overview

This project utilises all of the skills gained throughout the DevOps course. You are tasked with using Ansible to configure your Docker containers which should be deployed on to Amazon Web Services.

In teams, you will use Ansible to set up your Docker containers that host a Continuous Integration Pipeline. You are responsible for container design, this includes which tools you will include in your pipeline. Some have been listed as a guide below, but it is for you to decide which tools to use.

These Docker containers should be deployed into an AWS Environment, so that you're CI Pipeline is live and hosted in the cloud. You need to consider how this works and then implement the solution.

Suggested tools to be included within containers include;

- *A CI Tool*
- *A Code Review Tool*
- *A Build Management Tool*
- *A Monitoring Tool*
- *An Artifact Repository*
- *A Project Tracker*

All of your work must be uploaded to a Source Code Repository. This should include script files, playbooks, diagrams, Dockerfiles and anything else that you have used to build your project. You should be aware of any commands you have executed so you can explain how you achieved what you have.

Tools used

- *Continuous Integration Tool*

  Jenkins

- *Deployment Tool*

  amazon web services

- *Version control system*

  git

- *Project management Tool*

  Trello

- *A build management Tool*

  Apache maven

- *Team collaboration Tool*

  slack

- *A monitoring Tool*

  ZABBIX

- *Web server*

  Apache Tomcat

- *An Artifact Repository*

  Nexus

- *Configuration Management Tool*

  ANSIBLE

- *A project Tracker*

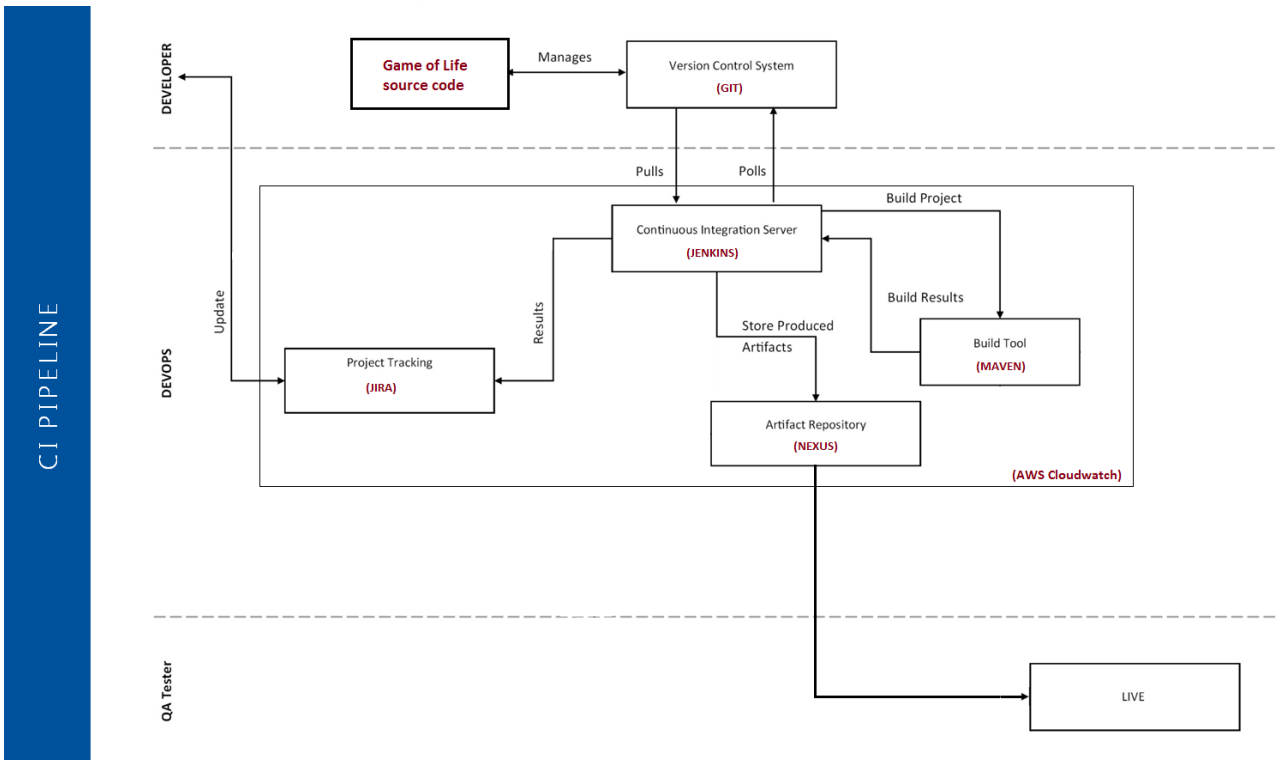
  JIRA

- *Container*

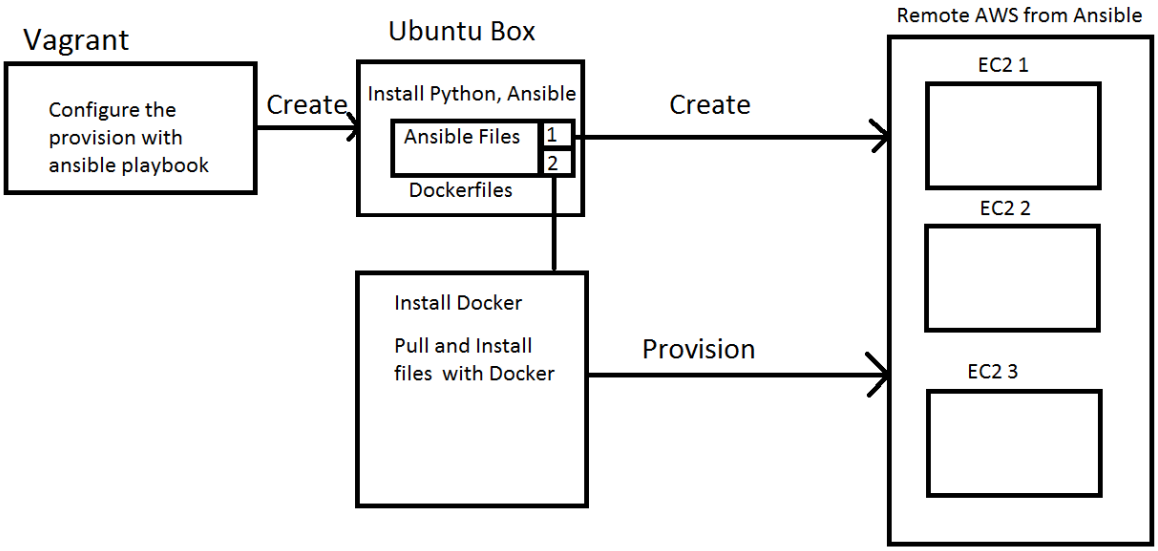
  docker

Project plan

Continuous Integration Pipeline



System architecture



Running Game of Life

Python version running on Terminal

Clone the git depository

git clone https://github.com/domoritz/gameoflife-python.git

Install python

sudo apt-get install python-minimal

Run the game

python game_of_life.py

Run the test

python test.py

JS running on HTML

Install apache2 server

sudo apt-get install apache2

Clone the git project

git clone https://github.com/pmav/game-of-life.git

Move the project to apache directory

sudo mv /game-of-life /* /var/www/html

Navigate to your IP address, it should be running on port 80

Game-of-life by wakaleo

Fork the project to use for the project

https://github.com/pawnu/game-of-life

Provision with required services

Java, maven, Git

These services are installed on the base infrastructure so it can be shared with docker containers.

```
---
        - name: install java, maven and git
          hosts: all
          remote_user: ubuntu
          become: yes
          tasks:
           - name: update package manager
             apt: update_cache=yes
           - name: Install Java
             apt: name=openjdk-8-jdk state=present
           - name: Install git
             apt: name=git state=present
           - name: Install Maven
             apt: name=maven state=present
```

hosts: all should contain the AWS instances ip addresses to provision to

Jira and Jenkins

Install docker dependency docker-py with pip

```
           - name: update package manager
             apt: update_cache=yes
           - name: Install pip
             apt: name=python-pip state=present
           - name: Install docker
             shell: "wget -qO- https://get.docker.com/ | sh"
           - name: Install docker py
             pip: name=docker-py state=present
```

JIRA and Jenkins both use port 8080 inside container. We can point outside port 8080 to JIRA and 8081 to Jenkins.

outsideport:containerport

```
           - name: Run JIRA
             docker_container:
               name: jira
               image: cptactionhank/atlassian-jira
               state: started
               restart: yes
               ports:
```

```
      - "8080:8080"
  - name: Run jenkins
    docker_container:
      name: jenkins
      image: jenkins
      state: started
      ports:
       - "8081:8080"
       - "50000:50000"
```

Nexus

Create a data volume container for nexus with busybox image

```
  - name: Create nexus data container
    docker_container:
      name: nexus-data
      image: busybox
      volumes:
       - /data
  - name: Run Nexus
```

Run the nexus image with the volume mounted

```
  - name: Run Nexus
    docker_container:
      name: nexus
      image: sonatype/nexus
      pull: yes
      state: started
      ports:
       - "8082:8081"
      volumes_from:
       - nexus-data
```

Zabbix

This image makes a container that includes zabbix server, frontend and mysql and runs Zabbix Server and Zabbix Web UI on a CentOS 6.5 base.

```
  - name: Install zabbix from berngp
    docker_container:
      name: zabbix
      image: berngp/docker-zabbix
      state: started
      ports:
       - "10051:10051"
       - "10052:10052"
       - "8082:80"
```

```
- "2812:2812"
```

Create AWS instances using ansible

```
#!/bin/bash
sudo add-apt-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get -y install ansible
ansible --version
#using python3 as we're using python3 interpreter for ansible
sudo apt install python3-pip
#boto is required for ansible
sudo pip install boto
sudo pip install --upgrade pip
sudo git clone https://github.com/pawnu/DevOps-Exercise-Book.git
sudo scp /home/ubuntu/DevOps-Exercise-Book/Project/* /etc/ansible
sudo chown -R ubuntu /home/ubuntu/.ansible
cd /etc/ansible/
#create localhost for installing docker and services in the master
cat<<EOT>> /etc/ansible/hosts
[hosts]
localhost ansible_connection=local
[webserver]
EOT
#requires sudo to create and retry the yml file cause of boto errors
sudo ansible-playbook -i hosts jira.yml
#install java,git,maven to all agents
ansible-playbook -i hosts javamavengit.yml
#add the keypair used to access the slave machines to environment
exec ssh-agent bash
#ssh-add ~/.ssh/AWSJJSEPK.pem
# ansible-playbook -i '35.176.201.215, ' nexus.yml -e 'ansible_python_in
```

ansible-playbook -i '35.176.201.215, ' nexus.yml -e 'ansible_python_interpreter=/usr/bin/python3'

In master VM install ansible

```
sudo add-apt-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get -y install ansible
```

Install boto as it's required to run ansible playbook

sudo apt install python-pip
sudo pip install boto
sudo pip install --upgrade pip

Edit the **hosts** file in **/etc/ansible** and create localhost and webserver hosts

```
[local]
localhost ansible_connection=local

[webserver]
35.176.36.60 ansible_python_interpreter=/usr/bin/python3
```

The ipaddress of AWS instances should have ansible interpreter point to python3 as it will look for python 2.7

Go to **/etc/ansible/** and create yml file for creating AWS instances

```
- name: Provision an EC2 Instance
  hosts: local
  connection: local
  gather_facts: no
  tags: provisioning
```

```
# Necessary Variables for creating/provisioning the EC2 Instance
    vars:
      instance_type: t2.micro
      security_group: TeamJJSEPKGroup
      image: ami-f1d7c395
      keypair: AWSJJSEPK
      region: eu-west-2
      count: 1
      tagname: Name=TeamJJSEP
```

*Note: **Name: TeamJJSEP** will give error as we're using the = format to create ec2 instance using local_action module*

```
  # Task that will be used to Launch/Create an EC2 Instance
  tasks:

    - name: Create a security group
      local_action:
        module: ec2_group
        aws_access_key: [redacted]
        aws_secret_key: [redacted]
        name: "{{ security_group }}"
        description: Security Group for webserver Servers
        region: "{{ region }}"
        rules:
          - proto: tcp
            from_port: 22
            to_port: 22
            cidr_ip: 0.0.0.0/0
          - proto: tcp
            from_port: 80
            to_port: 80
            cidr_ip: 0.0.0.0/0
          - proto: tcp
            from_port: 443
            to_port: 443
            cidr_ip: 0.0.0.0/0
        rules_egress:
          - proto: all
            cidr_ip: 0.0.0.0/0
      register: basic_firewall
```

```
- name: Launch the new EC2 Instance
  local_action: ec2
                aws_access_key=[redacted]
                aws_secret_key=[redacted]
                group={{ security_group }}
                instance_type={{ instance_type}}
                image={{ image }}
                wait=true
                region={{ region }}
                keypair={{ keypair }}
                count={{count}}
  register: ec2
```

```
- name: Add the newly created EC2 instance(s) to the local host group (located inside the directory)
  local_action: lineinfile
                dest="./hosts"
                regexp={{ item.public_ip }}
                insertafter="[webserver]" line={{ item.public_ip }}
  with_items: "{{ ec2.instances }}"
```

```
- name: Wait for SSH to come up
  local_action: wait_for
                host={{ item.public_ip }}
                port=22
                state=started
  with_items: "{{ ec2.instances }}"
```

Run the ansible-playbook to create AWS instances

sudo ansible-playbook –i hosts ec2create.yml

Copy the pem file from AWS to the master and put it in ssh-agent

```
ubuntu@ip-10-0-0-229:/etc/ansible$ exec ssh-agent bash
ubuntu@ip-10-0-0-229:/etc/ansible$ ssh-add ~/.ssh/AWSJJSEPK.pem
Identity added: /home/ubuntu/.ssh/AWSJJSEPK.pem (/home/ubuntu/.ssh/AWSJJSEPK.pem)
```

```
sudo chown -R ubuntu /home/ubuntu/.ansible
```

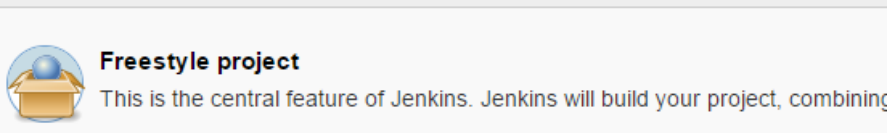
Configuring services to run the pipeline

Jenkins

Getting the dependencies

Create a new item

Select Freestyle project

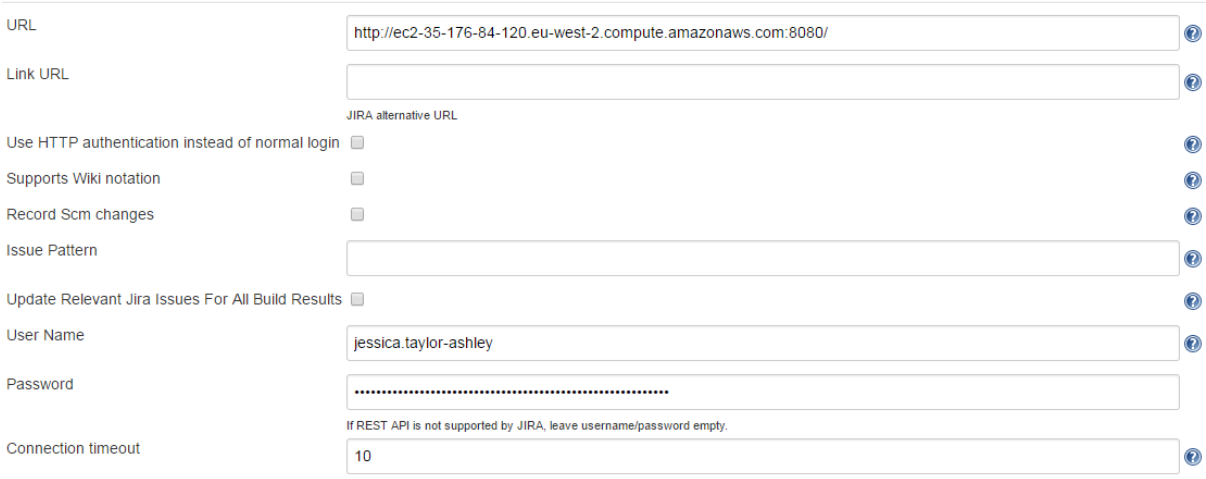


Go to **manage jenkins – manage plugin** and install the following:

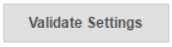
- Git plugin
- Jira plugin
- Nexus artifact uploader
- Deploy to container plugin

Go to **manage Jenkins – global tool configuration** to setup the tools.
Oracle credential is required to install from java.sun.com.

Go to **manage jenkins – configure system** and on section JIRA

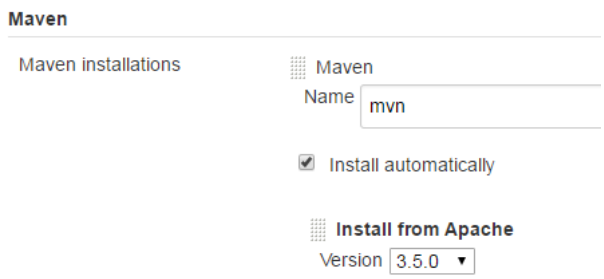


Click on **validate settings** to set up JIRA login for jenkins



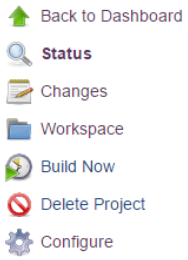
To setup JIRA for external access with Jenkins,
Go to **System - General Configuration – edit settings**
Change **External user management** to **ON**

Install maven automatically.



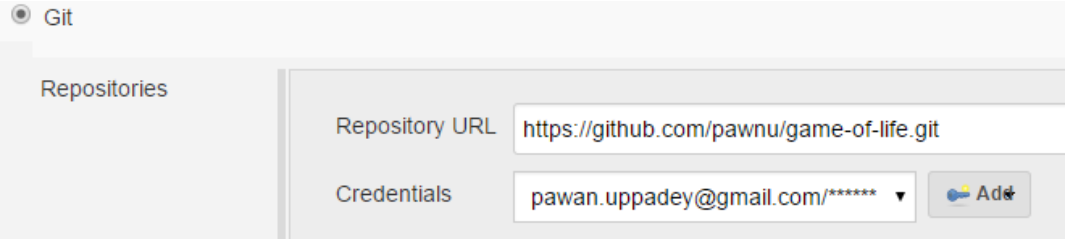
Setting up project

In the project go to configure

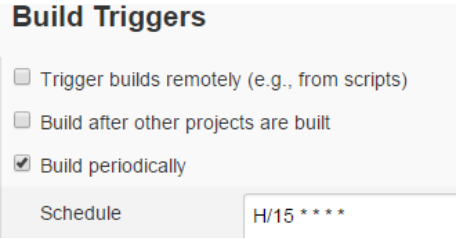


In source code management, select git.

Add repository URL for your project and credentials.



Select the trigger to build the project



Use version of maven to run the project

GroupID repositories should be by default included in nexus and repository releases as well.

Nexus Details

Nexus Version	NEXUS2
Protocol	HTTP
Nexus URL	52.56.121.131:8081/nexus
Credentials	admin/****** Add
GroupId	repositories
Version	v1
Repository	releases
Artifacts	Artifact
ArtifactId	gameoflife
Type	war
Classifier	
File	/var/jenkins_home/workspace/GameofLife/gameoflife-web/target/gameoflife.war

Add

Publish the test report and archive the artifacts in their respective target directory.

Post-build Actions

Publish JUnit test result report

Test report XMLs	**/target/surefire-reports/*.xml

Fileset 'includes' setting that specifies the generated raw XML report files, such a

☐ Retain long standard output/error

Health report amplification factor	1.0

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Additional test report features	Add
Allow empty results	☐ Do not fail the build on empty test results

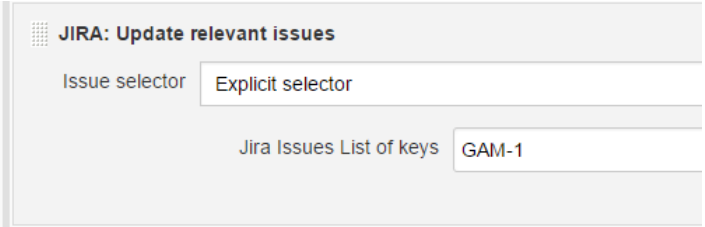
Archive the artifacts

Files to archive	**/target/*.war

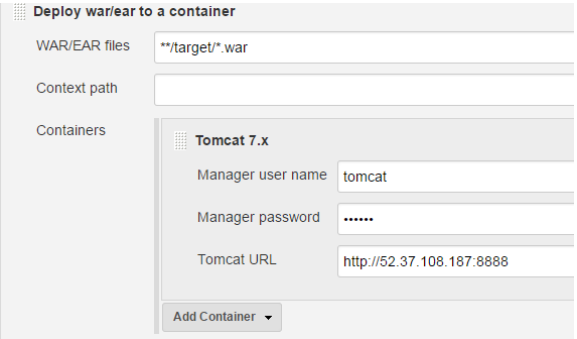
Result:

```
Uploading artifact gameoflife.war started....
GroupId: repositories
ArtifactId: gameoflife
Classifier:
Type: war
Version: v1
File: gameoflife.war
Repository:releases
Uploading: http://52.56.121.131:8081/nexus/content/repositories/releases/repositories/gameoflife/v1/gameoflife-v1.war
10 % completed (319 kB / 3.2 MB).
20 % completed (639 kB / 3.2 MB).
30 % completed (958 kB / 3.2 MB).
40 % completed (1.3 MB / 3.2 MB).
50 % completed (1.6 MB / 3.2 MB).
60 % completed (1.9 MB / 3.2 MB).
70 % completed (2.2 MB / 3.2 MB).
80 % completed (2.6 MB / 3.2 MB).
90 % completed (2.9 MB / 3.2 MB).
100 % completed (3.2 MB / 3.2 MB).
Uploaded: http://52.56.121.131:8081/nexus/content/repositories/releases/repositories/gameoflife/v1/gameoflife-v1.war (3.2 MB at 6.3 MB/s)
```

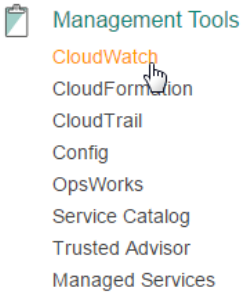
On post build action, add update relevant issues

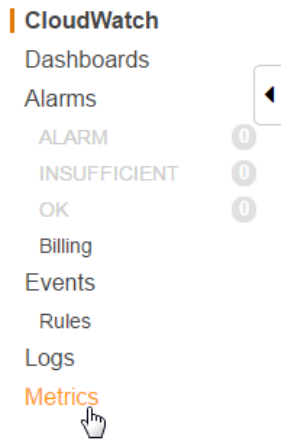


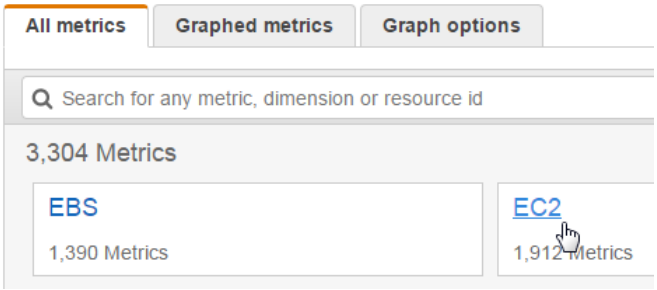
Deploy war file to tomcat 7.0



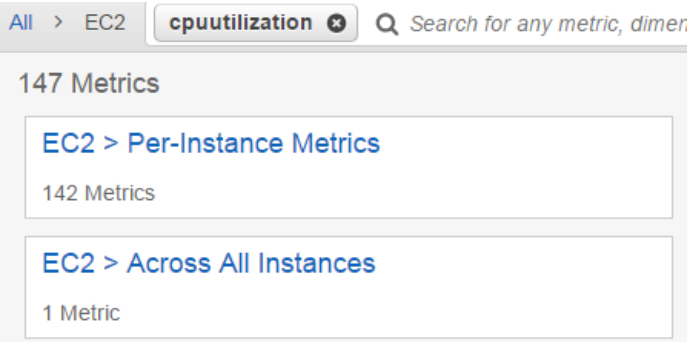
Monitoring the instances with AWS CloudWatch

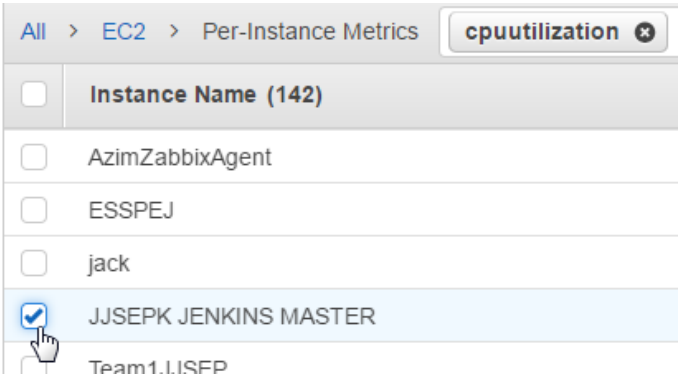


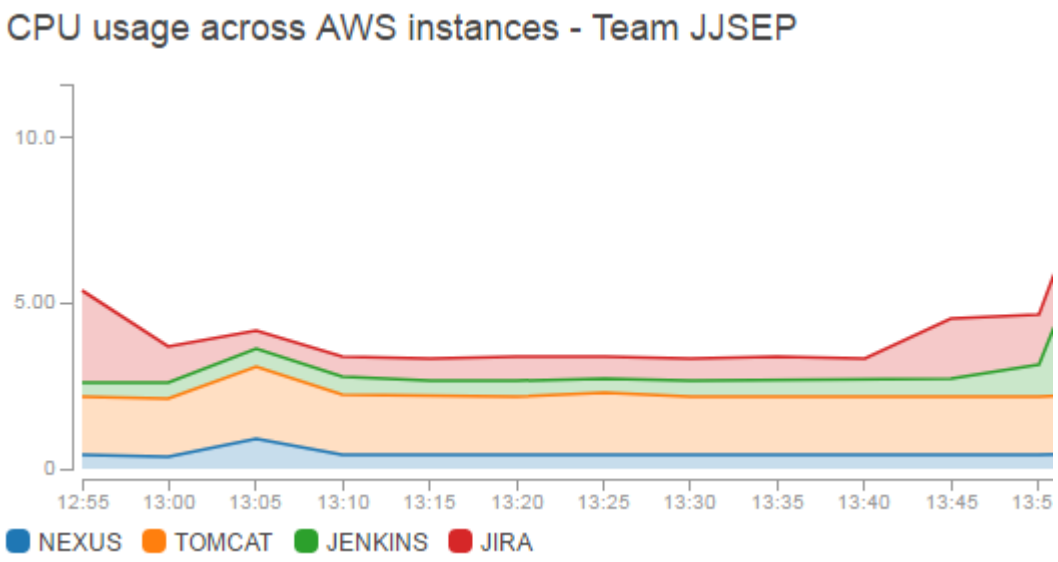




Search for cpuutilization and select the instances.







Challenges

Wrong GameofLife project

Initially wrong version of gameoflife was used. It didn't have pom files that managed dependencies and used maven to build the project. A pom file had to be built but it was difficult to get the project running.

We used the correct version of Game of Life with pom file which allowed Jenkins to build the project as well as run tests.

Host file connection requirement

```
[local]
localhost ansible_connection=local

[webserver]
35.176.36.60 ansible_python_interpreter=/usr/bin/python3
```

Without specifying **ansible_connection=local** the localhost wasn't being recognised as local connection

Without specifying **ansible_python_interpreter=/usr/bin/python3**, the host wouldn't work as it was looking for python2.7

Adding instance name to aws instance

```
- name: Add tag to Instance(s)
  local_action: ec2_tag resource="{{ item.id }}" region="{{ region }}" state=present
  with_items: "{{ ec2.instances }}"
  args:
    tags:
      Name: TeamJJSEP
```

The problem with the tag **Name: TeamJJSEP** was not consistent with the **local_action** following arguments e.g. **resource=”” and region=””**

The inner arguments **Name: TeamJJSEP** had to be **Name=TeamJJSEP**

Assigning ports to different services in same agent

The port assignment used was **8080:8080** for Jenkins and **8082:8082** for Jira. This didn’t work as by default JIRA was listening on port 8080 inside its own container.

After drawing a diagram to construct the assignment logic, we figured out that the container has its own port system and we could assign external traffic coming into 8082 port of AWS agent to port 8080 of JIRA container.

JIRA intense memory requirement

Although JIRA was up and running on port 8082 of our agent, it was causing the AWS instance to freeze up and crash.

We used **docker logs jira** to look at logs and found there is Out of memory error.

The t2.micro instance created to run JIRA didn’t have the spec to run JIRA so we upgraded the instance to a level up.

Zabbix multiple dependencies

It proved to be difficult to get Zabbix running. It required Zabbix server to run which required a database with username and password for zabbix to use as well as a database created for zabbix to use. This meant creating mysql and zabbix server container and linking them together with volumes. After that a web interface had to be installed and linked to zabbix server and the database.

After that a zabbix agent had to be installed on machines where monitoring was required.

Rather than doing these, we used a version of zabbix that would create all these from docker hub.

Running ansible-playbook as sudo

Setting up ssh keypair with agent and master was done with user Ubuntu and the environment variable for Ubuntu was populated with the keys. Using sudo will make the user root which doesn’t have the required authentication keys and this gave the unreachable error.

```
fatal: [35.176.159.180]: UNREACHABLE! => {"changed": false, "msg": "Failed to
a ssh: Permission denied (publickey).\r\n", "unreachable": true}
```

Pip version mismatch

Since python3 was already instealled by default on Ubuntu 16 and ansible interpreter used was switched to python3, the pip version also had to be for python 3.

```
[webserver]
35.176.36.60 ansible_python_interpreter=/usr/bin/python3
```

```
  - name: Install pip
    apt: name=python3-pip state=present
```

References

http://docs.ansible.com/ansible/ec2_module.html

http://docs.ansible.com/ansible/guide_rolling_upgrade.html

https://aws.amazon.com/blogs/apn/getting-started-with-ansible-and-dynamic-amazon-ec2-inventory-management/

https://community.hortonworks.com/articles/86924/using-ansible-to-deploy-instances-on-aws.html

https://code.tutsplus.com/tutorials/automate-all-the-things-with-ansible-part-one--cms-25931

http://www.jiayul.me/hacking/2016/07/24/using-ansible-to-provision-aws-ec2-instances-with-docker.html

https://github.com/dkanbier/docker-zabbix-server

https://hub.docker.com/r/berngp/docker-zabbix/