Epic of Gilgamesh – workbook

# Introduction:

## How is workbook built:

Both guidebook and the workbook are ased on “Bloom’s Taxonomy” for learning practices, you may find more info on the WIKI page.

Bloom’s taxonomy is structured from 3 main domains:

* The cognitive domain (knowledge based) – it’s our focus
* The affective domain
* The psychomotor domain (action based)

The cognitive domain:

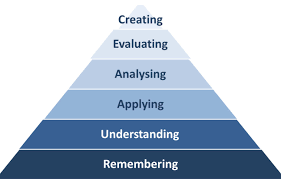


Figure : Bloom's Taxonomy - cognitive domain

## Exercises types:

### Walkthrough practices

Those practices are the simplest ones, the purpose of those, is let the student some familiarity with both tools and terms used in the class. Usually will take no longer than 1 hour to finish.

### Technical tasks

Those practices are the more complex, there is no guide of step by step. Usually you will get a set of goals that your artifact will need to achieve. The purpose of those practices is let the student more knowledge and confidence with applying the knowledge used on the tools and terms. Although diagrams and architecture papers aren’t must, but it is highly advised to create ones.

### Analyzing scenarios

### Building whole solutions

# Linux Administration

## Linux operations:

Linux custom services:

## Script writing:

## Linux Provisioning using Vagrant:

#### Technical Task 1:

# CI\CD World

## Code Building:

## Jenkins:

### Jenkins pipelines:

#### Technical Task 1: Simple CI script (maven):

Compose a Jenkins file pipeline with following goals:

1. Source code located at: <https://github.com/zivkashtan/course.git>
2. Please see the following notes:
   1. The pipeline code should not include passwords hard-codded
   2. Consider using the *script{}* block
3. Result war will be stored on the Nexus server
4. No need to create docker container
5. No modifications are allowed in the *pox.xml* file
6. Create a readme.md file with following info:
   1. What Jenkins plugin did you used?
   2. What is the code flow you have created?

#### Technical Task 3: Simple CI script (maven,docker,reading pom file):

Compose a Jenkins file pipeline with following goals:

1. Source code located at: <https://github.com/zivkashtan/course.git>
2. Please see the following notes:
   1. The pipeline code should not include passwords hard-codded
   2. Consider using the *script{}* block
3. Result war will be stored on the Nexus server, with the build number as the artifact version number (if in the pom.xml file written 1.0.0 and the build number is 13, than the version uploaded into the nexus will be 1.0.13)
4. No need to create docker container
5. No modifications are allowed in the *pox.xml* file
6. Create a readme.md file with following info:
   1. What Jenkins plugin did you used?
   2. What is the code flow you have created?

#### Technical Task 4: Simple CI script (maven,docker,reading pom file):

Compose a Jenkins file pipeline with following goals:

1. Source code located at: <https://github.com/zivkashtan/course.git>
2. Please see the following notes:
   1. The pipeline code should not include passwords hard-codded
   2. Consider using the *script{}* block
3. Result docker image will be stored on the Nexus server, with the build number as the artifact version number (if in the pom.xml file written 1.0.0 and the build number is 13, than the version uploaded into the nexus will be 1.0.13)
4. If there is no Dockerfile file, than the pipeline will create one.
5. No need to create docker container
6. No modifications are allowed in the *pox.xml* file
7. Create a readme.md file with following info:
   1. What Jenkins plugin did you used?
   2. What is the code flow you have created?

#### Technical Task 5: Simple CI script (maven,docker,conditions):

Compose a Jenkins file pipeline with following goals:

1. Source code located at: <https://github.com/zivkashtan/course.git>
2. The job will have following parameters:
   1. To fail on test failures (Boolean)
   2. To create a Docker container (Boolean)
   3. To upload war to Nexus (Boolean)
3. Please see the following notes:
   1. The pipeline code should not include passwords hard-codded
   2. Consider using the *script{}* block
4. Create a readme.md file with following info:
   1. What Jenkins plugin did you used?
   2. What is the code flow you have created?

#### Technical Task 6: Simple CI script (maven,docker,conditions):

Compose a Jenkins file pipeline with following goals:

1. Source code located at: <https://github.com/zivkashtan/course.git>
2. The job will have following parameters:
   1. To fail on test failures (Boolean)
   2. To create a Docker container (Boolean)
   3. To upload war to Nexus (Boolean)
3. Please see the following notes:
   1. The pipeline code should not include passwords hard-codded
   2. Consider using the *script{}* block
4. Create a readme.md file with following info:
   1. What Jenkins plugin did you used?
   2. What is the code flow you have created?

#### Technical Task 7: Simple CI script (npm,docker,conditions):

Compose a Jenkins file pipeline with following goals:

1. Source code located at: <https://github.com/DanMaxic/nodejs-practice>
2. Note, this is a node.js project!
3. The job will have following parameters:
   1. To create a Docker container (Boolean)
   2. To perform deploy to a server
4. Please see the following notes:
   1. The pipeline code should not include passwords hard-codded
   2. Consider using the *script{}* block
5. Create a readme.md file with following info:
   1. What Jenkins plugin did you used?
   2. What is the code flow you have created?

### Advanced Pipelines:

### Advanced CI\CD practices:

# OPS World

## Application Operations

Application logging management

Application Monitoring

Application Performance Management (metrics)

## Application Orchestration

Metrics

## Infrastructure Designing

AWS for DevOps

## Infrastructure Orchestration

### Containers & Docker CI with Docker

### Ansible

#### Introduction:

#### Walkthrough 1: ad-hoc operations

**For our first exercise, we are going to run some ad-hoc commands to help you get a feel for how Ansible works. Ansible Ad-Hoc commands enable you to perform tasks on remote nodes without having to write a playbook. They are very useful when you simply need to do one or two things quickly and often, to many remote nodes.**

Like many Linux commands, ansible allows for long-form options as well as short-form. For example:

ansible web --module-name ping

is the same as running

ansible all -i inventory -m ping

We are going to be using the short-form options throughout this workshop

##### Step 0: environment setup:

run the vagrant project located at: **resources/vagrants/ansible-lab**

* Validate lab is up and running

##### **Step 1: Let’s start with something really basic - pinging a host. The ping module makes sure our web hosts are responsive.**

ansible web -m ping

##### **Step 2: Now let’s see how we can run a good ol' fashioned Linux command and format the output using the command module.**

ansible all -i inventory -m ping -m command -a "uptime" -o

##### **Step 3: Take a look at your web node’s configuration. The setup module displays ansible facts (and a lot of them) about an endpoint.**

ansible all -i inventory -m setup

##### **Step 4: Now, let’s install Apache using the yum module**

ansible all -i inventory -m yum -a "name=httpd state=present" -b

##### **Step 5: OK, Apache is installed now so let’s start it up using the service module**

ansible web -i inventory -m service -a "name=httpd state=started" -b

##### **Step 6: Finally, let’s clean up after ourselves. First, stop the httpd service**

ansible web -i inventory -m service -a "name=httpd state=stopped" -b

##### **Step 7: Next, remove the Apache package**

ansible web -i inventory -m yum -a "name=httpd state=absent" -b

#### Walkthrough 2: writing and running playbooks

##### Step 0: environment setup:

run the vagrant project located at: **resources/vagrants/ansible-lab**

Validate lab is up and running

##### Step 1: Create a directory called apache\_basic in your home directory and change directories into it

mkdir ~/apache\_basic

cd ~/apache\_basic

##### Step 2: Define your inventory.

Inventories are crucial to Ansible as they define remote machines on which you wish to run your playbook(s). Use vi or vim to create a file called hosts. Then, add the appropriate definitions which can be viewed in ~/lightbulb/lessons/lab\_inventory/{{studentX}}-instances.txt. (And yes, I suppose you could copy the file, but we’d rather you type it in so you can become familiar with the syntax)

[web]

node-1 ansible\_host=<IP Address of your node-1>

node-2 ansible\_host=<IP Address of your node-2>

##### Step 3: Use vi or vim to create a file called install\_apache.yml

##### Section 2 - Defining Your Play

Now that you are editing install\_apache.yml, let’s begin by defining the play and then understanding what each line accomplishes

---

- hosts: web

name: Install the apache web service

become: yes

--- Defines the beginning of YAML

hosts: web Defines the host group in your inventory on which this play will run against

name: Install the apache web service This describes our play

become: yes Enables user privilege escalation. The default is sudo, but su, pbrun, and several others are also supported.

##### Section 3: Adding Tasks to Your Play

Now that we’ve defined your play, let’s add some tasks to get some things done. Align (vertically) the t in task with the b become. Yes, it does actually matter. In fact, you should make sure all of your playbook statements are aligned in the way shown here. If you want to see the entire playbook for reference, skip to the bottom of this exercise.

tasks:

- name: install apache

yum:

name: httpd

state: present

- name: start httpd

service:

name: httpd

state: started

tasks: This denotes that one or more tasks are about to be defined

name: Each task requires a name which will print to standard output when you run your playbook. Therefore, give your tasks a name that is short, sweet, and to the point

yum:

name: httpd

state: present

These three lines are calling the Ansible module yum to install httpd. Click here to see all options for the yum module.

service:

name: httpd

state: started

The next few lines are using the ansible module service to start the httpd service. The service module is the preferred way of controlling services on remote hosts. Click here to learn more about the service module.

##### Section 4: Review

Now that you’ve completed writing your playbook, it would be a shame not to keep it.

Use the write/quit method in vi or vim to save your playbook, i.e. Esc :wq!

And that should do it. You should now have a fully written playbook called install\_apache.yml. You are ready to automate!

#### Technical Task 1: node exporter deployment

Here you will need to perform ansible deployment of component called “node exporter”. This component responsible to export all machine related performance metrics, and expose them into a dedicated port, for future consuming.

Installation script for the node exporter:

|  |
| --- |
| *#!/usr/bin/env bash*  *NODE\_EXPORTER\_VER="0.17.1"*  *useradd --no-create-home --shell /bin/false node\_exporter*  *curl -LO https://github.com/prometheus/node\_exporter/releases/download/v${NODE\_EXPORTER\_VER}/node\_exporter-${NODE\_EXPORTER\_VER}.linux-amd64.tar.gz*  *tar xvf node\_exporter-\*.linux-amd64.tar.gz*  *cp node\_exporter-\*.linux-amd64/node\_exporter /usr/local/bin*  *chown node\_exporter:node\_exporter /usr/local/bin/node\_exporter*  *rm -rf node\_exporter-\*.linux-amd64.tar.gz node\_exporter-\*.linux-amd64*  *cat << EOF > /etc/systemd/system/node\_exporter.service*  *[Unit]*  *Description=Node Exporter*  *Wants=network-online.target*  *After=network-online.target*  *[Service]*  *User=node\_exporter*  *Group=node\_exporter*  *Type=simple*  *ExecStart=/usr/local/bin/node\_exporter*  *[Install]*  *WantedBy=multi-user.target*  *EOF*  *systemctl daemon-reload*  *systemctl enable node\_exporter*  *systemctl start node\_exporter*  *#systemctl status node\_exporter* |

Your tasks are the following:

1. Create a new “ansible project”
2. Create and compose the inventory file
3. Create the deployment script
4. Create the playbook yaml file
5. Create a “readme.md” file with following:
   1. What is the command used to evaluate the service installed properly on all servers?
   2. What Ansible modules you used during the development, and what attributes you have set to them?
   3. What is the structure of the “Ansible Project” you used?

#### Technical Task 2: Nginx Deployment

Here is the following html page:

|  |
| --- |
| *<!DOCTYPE html>*  *<html lang="en">*  *<head>*  *<meta charset="UTF-8">*  *<title>nginx play</title>*  *</head>*  *<body>*  *hello world!*  *</body>*  *</html>* |

tasks are the following:

1. Create a new “ansible project”
2. Create and compose the inventory file
3. Create the deployment script
4. Create the playbook yaml file
5. Create the HTML page
6. Create a “readme.md” file with following:
   1. What is the command used to evaluate the service installed properly on all servers?
   2. What Ansible modules you used during the development, and what attributes you have set to them?
   3. What is the structure of the “Ansible Project” you used (use the command *tree*)?

#### Technical Task 2: war deployment

Get a war file (from CI practices), and follow the tasks:

tasks are the following:

1. Create a new “ansible project”
2. Create and compose the inventory file
3. Create the deployment script
4. Create the playbook yaml file
5. Create a “readme.md” file with following:
   1. What is the command used to evaluate the service installed properly on all servers?
   2. What Ansible modules you used during the development, and what attributes you have set to them?
   3. What is the structure of the “Ansible Project” you used (use the command *tree*)?
   4. How did you pass the war file from your workstations to the target servers? What method did you used?

### Kubernetes

### Terraform

### Other tools and utilities