

## Exercise 1.0 - Running Ad-hoc commands

# Section 1: Ad-hoc commands

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.

## Step 0:

SSH into your **control** host and define your inventory. Inventories are crucial to Ansible as they define remote machines on which you wish to run commands or your playbook(s). For this workshop we have already created a base inventory file to save you the effort. Use `cat ~/lightbulb/lessons/lab_inventory/student#-instances.txt` to view the file. Notice there is a section of vars to be assigned to particular groups. Then there are individual hosts within groups including authentication details.



At this point we are just using basic authentication to a local Administrator. Also, the password is clear text in the inventory. Later in this workshop we will show how to use AD authentication and securely store credentials.

## Step 1:

Let's start with something really basic - pinging a host. The **win\_ping** module makes sure our windows hosts are responsive. This is not a traditional 'ping', but actually verifying connectivity to the host.

Ansible uses WinRM to interact with Windows hosts. This has already been setup in our lab. However, in other environments a script to enable this is provided here: (<https://github.com/ansible/ansible/blob/devel/examples/scripts/ConfigureRemotingForAnsible.ps1>)

```
ansible windows -m win_ping
```

## Step 2:

Now let's see how we can run a PowerShell command and view the output using the **win\_shell** module.

```
ansible windows -m win_shell -a "Get-Service"  
ansible windows -m win_shell -a "Get-Process"
```

## Step 3:

Take a look at your windows nodes configuration. The **setup** module displays ansible facts (and a lot of them) about an endpoint.

```
ansible windows -m setup
```

## Step 4:

Now, let's install IIS using the **win\_feature** module. It may take a moment to complete.

```
ansible windows -m win_feature -a "name=Web-Server state=present"
```

## Step 5:

OK, IIS is installed now so let's be certain it is started using the **service** module. Test connectivity with curl to see a base IIS page (Make certain to replace # with your user number).

```
ansible windows -m win_service -a "name=W3Svc state=started"
curl http://student#-node1/
```

## Step 6:

Finally, let's clean up after ourselves. First, stop the httpd service. You can verify again the the curl just hangs. (ctrl-c to break out of the command)

```
ansible windows -m win_service -a "name=W3Svc state=stopped"
curl http://student#-node1/
```

## Step 7:

Next, remove the IIS feature and reboot.

```
ansible windows -m win_feature -a "name=Web-Server state=absent"
ansible windows -m win_reboot
```

You can run **win\_ping** module as you did earlier to verify the systems are back online.



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

```
ansible windows --module-name win_ping
```

is the same as running

```
ansible windows -m win_ping
```

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

## Exercise 1.1 - Writing Your First playbook

Now that you've gotten a sense of how ansible works, we are going to write our first ansible **playbook**. The playbook is where you can take some of those ad-hoc commands you just ran and put them into a repeatable set of **plays** and **tasks**.

A playbook can have multiple plays and a play can have one or multiple tasks. The goal of a **play** is to map a group of hosts. The goal of a **task** is to implement modules against those hosts.

For our first playbook, we are only going to write one play and two tasks.

### Overview

Starting at this task we are going to work from our workstation host. We will use Visual Studio Code as our editor. In addition, we will use GitLab for source code control. This will allow us to minimize development work on the linux command line. Other editors or source code solutions can be used, but this will show the general workflow.

Alternatively you can login to gitlab and use built-in text editor to commit directly. This can be useful when you have slow connection to workstation instance. Check with your instructor if you have any questions.

## Section 1: Creating a Directory Structure and Files for your Playbook

There is a [best practice](#) on the preferred directory structures for playbooks. We strongly encourage you to read and understand these practices as you develop your Ansible ninja skills. That said, our playbook today is very basic and creating a complex structure will just confuse things.

Instead, we are going to create a very simple directory structure for our playbook, and add just a couple of files to it.

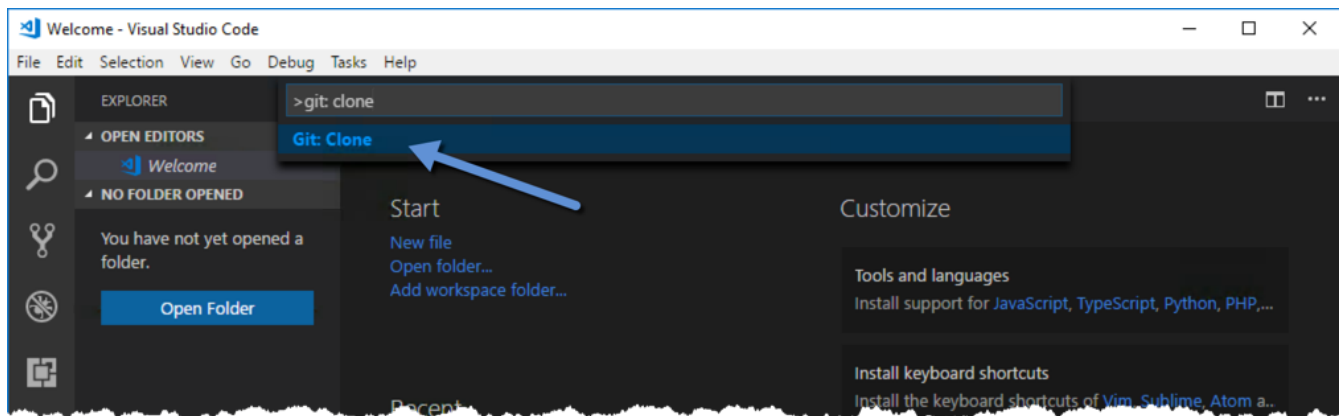
### Step 1: Open Visual Studio Code and Clone an empty git repository

When you open Visual Studio Code, a 'Welcome' screen will load. When prompted, choose **Firefox** as your default browser (and don't import anything). You can close the browser for now.



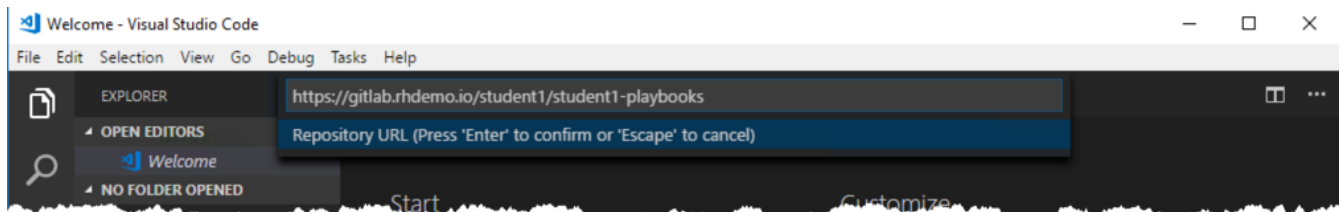
Internet Explorer 11 has some issues with Ansible Tower in the later labs. Due to this we will use Firefox in these examples.

In the main window, press Ctrl + Shift + P to bring up the action search window. Type Git: Clone and select.



#### Git Clone

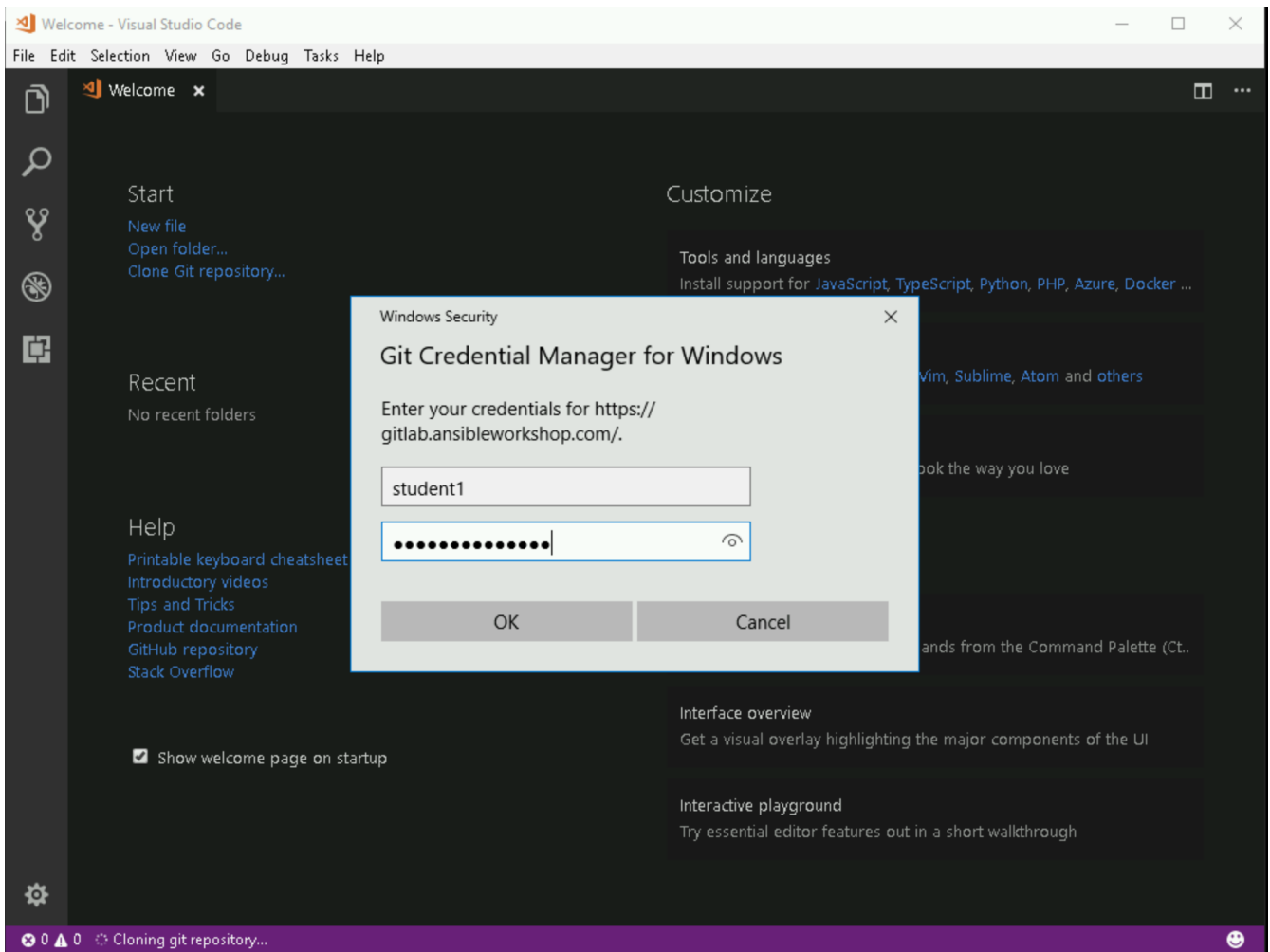
At the top, enter <https://gitlab.rhdemo.io/student#/student#-playbooks>, where # is your student number.



#### Git Clone URL

When it prompts you for directory, leave the default (C:\Users\student#)

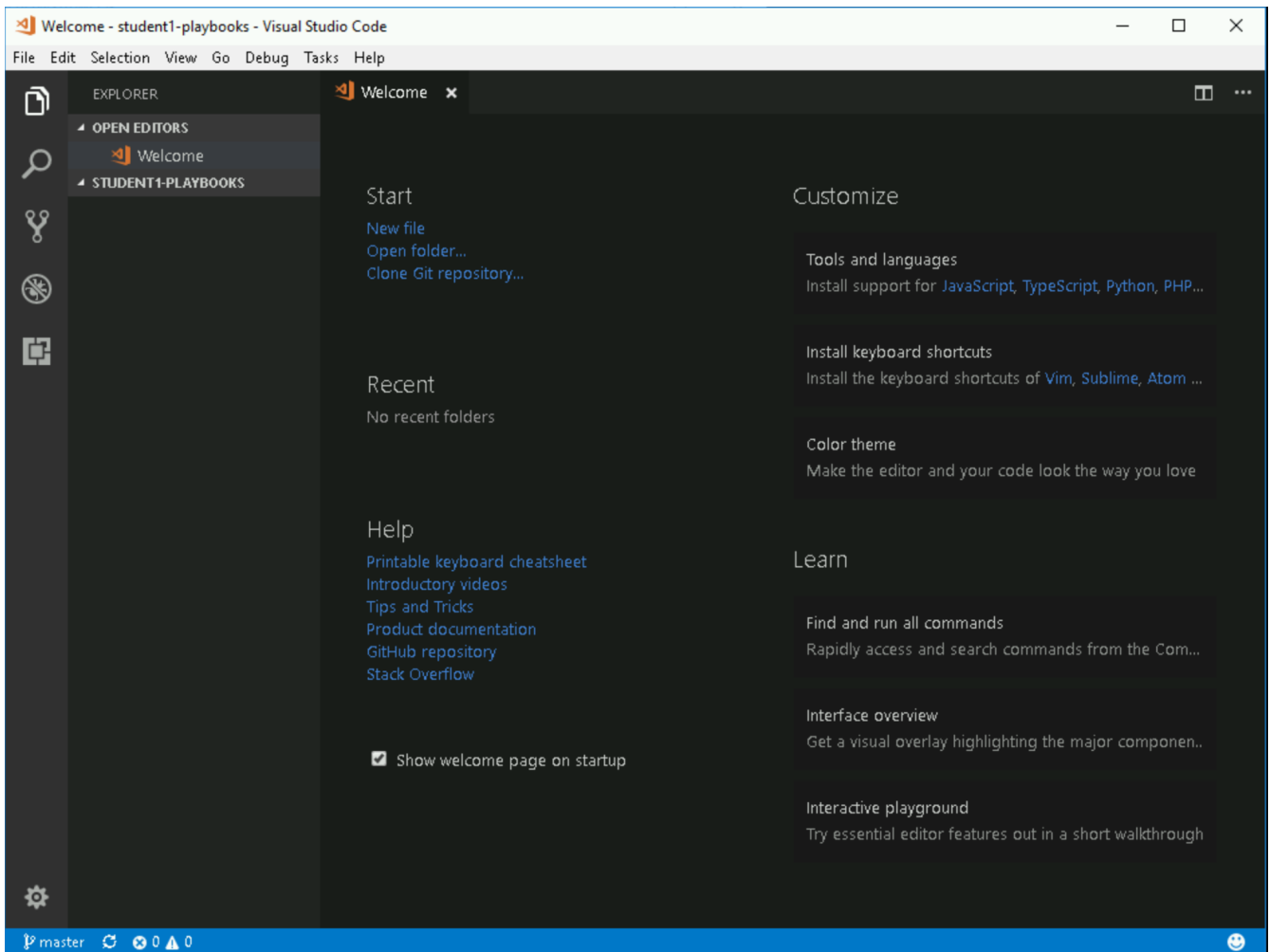
You will see the progress spin in the bottom left bar. Eventually you will be prompted for your gitlab user/password. (Watch your menu bar for this to pop up) Use your AD credentials (student#/...)



### *Git Clone Password*

Once synced, click the **Open Repository** button at the top.

At this point in the Explorer accordion you should have a 'student#-playbooks' section with no files.



*Student Playbooks Repo*

**Step 2:** Create a directory and called **iis\_basic** and a file called **install\_iis.yml**

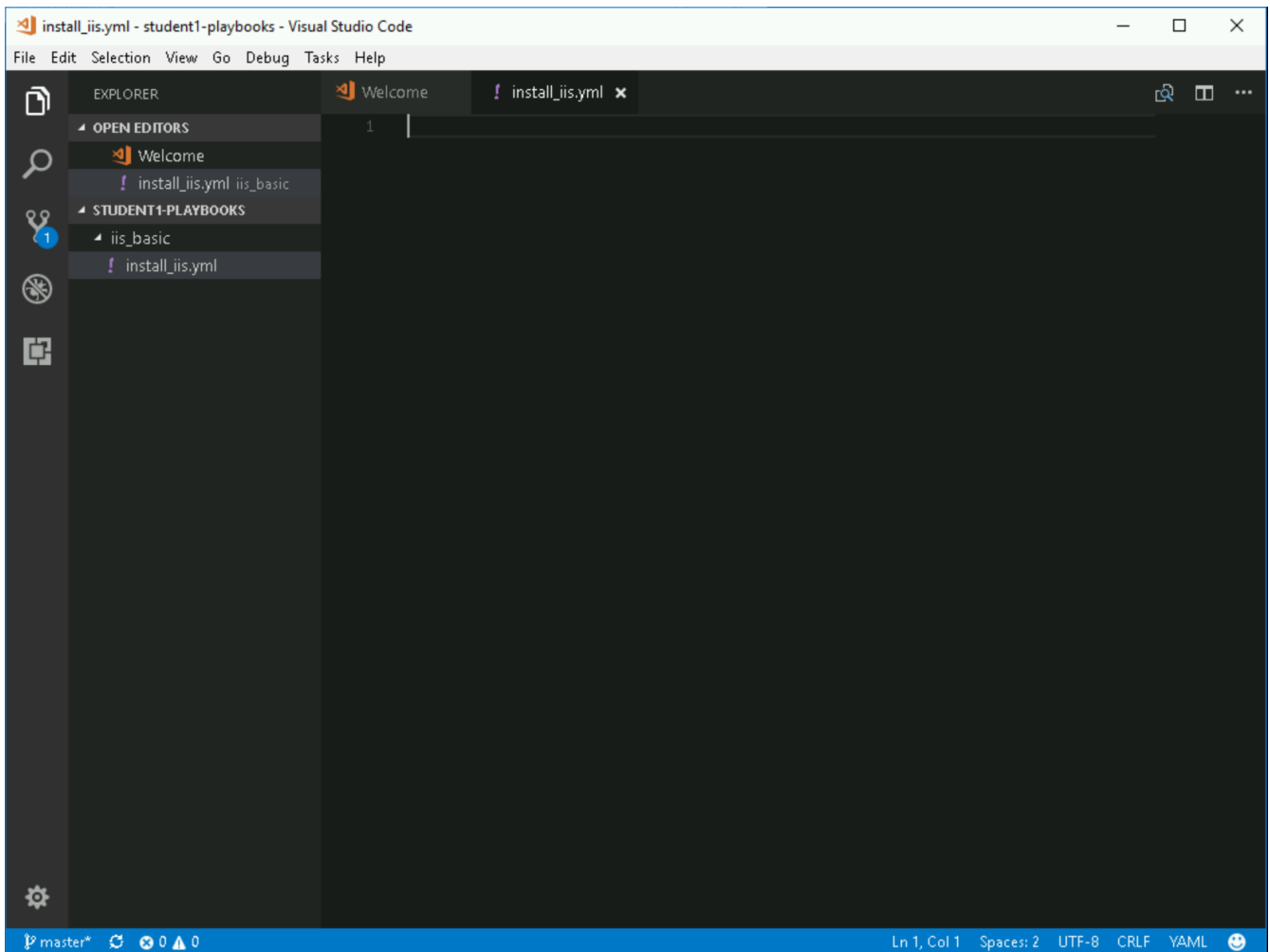
Hover over the **student#-playbooks** section and click on the **New Folder** button

Type **iis\_basic** and press enter. Then click on that folder so it is selected.

Hover over the **student#-playbooks** section again and click on the **New File** button.

Type **install\_iis.yml** and press enter.

You should now have an editor open in the right pane that can be used for creating your playbook.



*Empty install\_iis.yml*

## Section 2: Defining Your Play

Now that you are editing `install_iis.yml`, let's begin by defining the play and then understanding what each line accomplishes

```
---  
- hosts: windows  
  name: Install the IIS web service
```

- `---` Defines the beginning of YAML
- `hosts: windows` Defines the host group in your inventory on which this play will run against
- `name: Install the IIS web service` This describes our play

## 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 **n** `name`.

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 IIS
    win_feature:
      name: Web-Server
      state: present

  - name: Start IIS Service
    win_service:
      name: W3Svc
      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

```
win_feature:
  name: Web-Server
  state: present
```

- These three lines are calling the Ansible module **win\_feature** to install the IIS Web Server. [Click here](#) to see all options for the win\_feature module.

```
win_service:
  name: W3Svc
  state: started
```

- The next few lines are using the ansible module **win\_service** to start the IIS service. The win\_service module is the preferred way of controlling services on remote hosts. [Click here](#) to learn more about the **win\_service** module.

## Section 4: Saving your Playbook

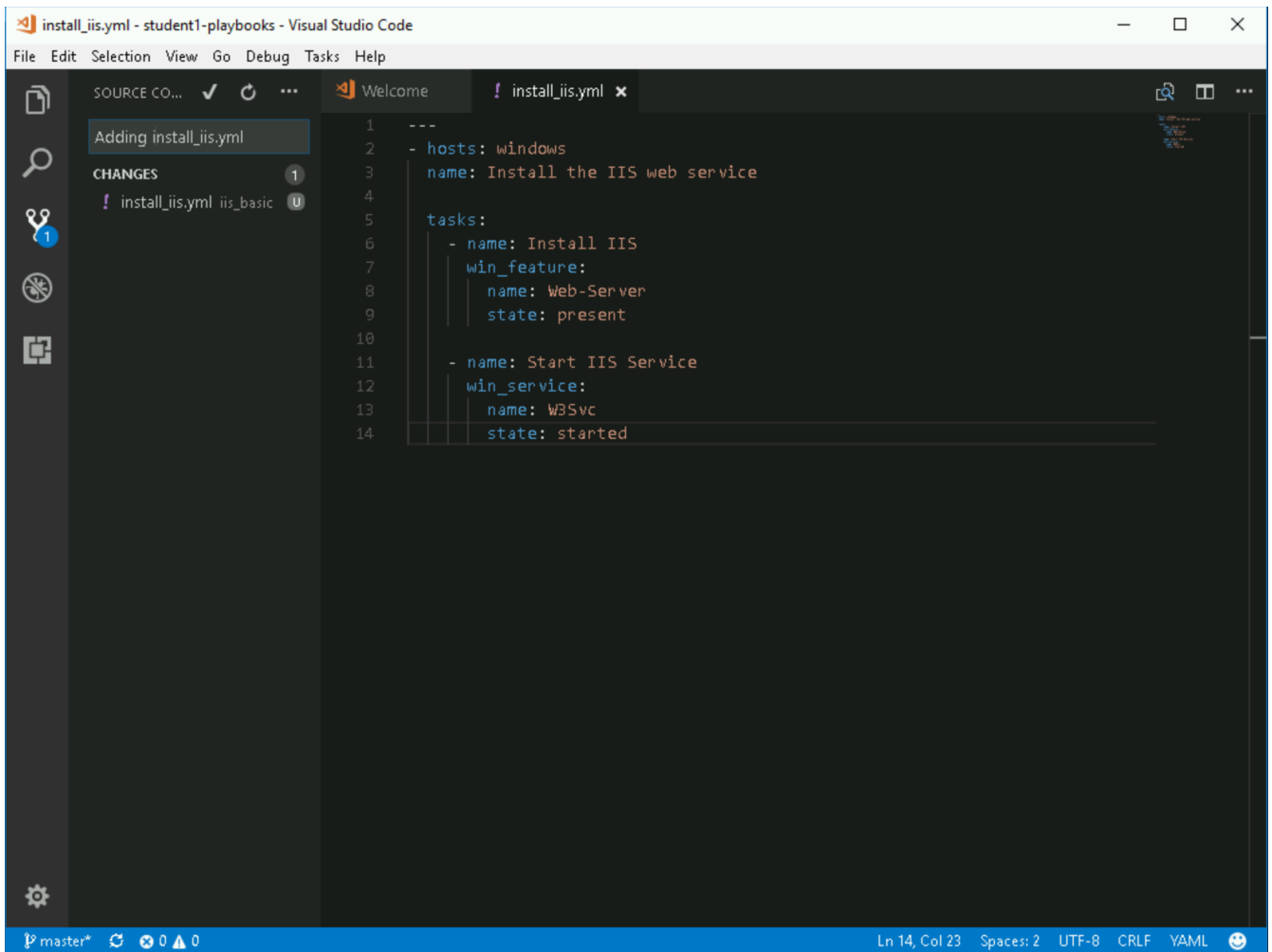
Now that you've completed writing your playbook, it would be a shame not to keep it.

Click 'File' from the menu and then click 'Save'

And that should do it. You should now have a fully written playbook called **install\_iis.yml**.

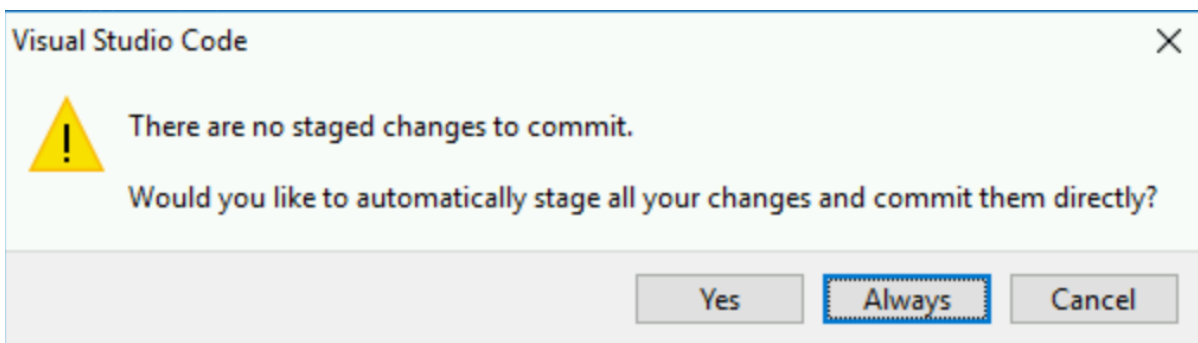
**But wait!!!** We haven't committed our changes from our local copy to source code. Click on the Source Code icon as shown below (It is the middle on the far left of the page that has the blue circle with # 1 in it)

Type in a commit message such as **Adding install\_iis.yml** and click the check box above to commit.



*Git Commit install\_iis.yml*

This will prompt to ask if you want to stage the changes. Click on 'Always' and you won't be prompted again.



*Stage Commits Always*

Now you need to push the committed changes to your repository.

On the bottom left blue bar, click the cloud with the up arrow on it to publish changes.

Next in the top of the window it will prompt you to pick a remote to publish to. Choose the default (**origin**)

The screenshot shows the Visual Studio Code interface with a file named `install_iis.yml` open. The file contains an Ansible playbook with the following content:

```
3  name: Install the IIS web service
4
5  tasks:
6    - name: Install IIS
7      win_feature:
8        name: Web-Server
9        state: present
10
11    - name: Start IIS Service
12      win_service:
13        name: W3Svc
14        state: started
```

A Git push dialog is open, asking to pick a remote to publish the branch 'master' to. The 'origin' remote is selected. The dialog also shows a commit message field and a 'Commit' button.

### Git Push Origin

This may take as long as 30 seconds to push. If you're interested in validating the code is in git, you can connect to gitlab to verify. Open **Firefox** and connect to <https://gitlab.rhdemo.io>. Login with your AD user (student#) and password and you should see your repo.

You are ready to automate!



Ansible (well, YAML really) can be a bit particular about formatting especially around indentation/spacing. When you all get back to the office, read up on this [YAML Syntax](#) a bit more and it will save you some headaches later. In the meantime, your completed playbook should look like this. Take note of the spacing and alignment.

```
---
- hosts: windows
  name: Install the IIS web service

  tasks:
    - name: Install IIS
      win_feature:
        name: Web-Server
        state: present

    - name: Start IIS Service
      win_service:
        name: W3Svc
        state: started
```

# Exercise 1.2 - Running Your Playbook

## Section 1: Running the Playbook

We are now going to run you're brand spankin' new playbook on your two windows nodes. To do this, you are going to use the **ansible-playbook** command.

### Step 1:

Use putty to login to your student#-control host as student#

Clone the playbooks from source code and then execute the playbook. Be certain to use your student number below. Also, due to our self-signed cert we need to first disable ssl verification

```
git config --global http.sslVerify false

# Setup to cache git credentials in memory for a day
git config --global credential.helper cache
git config --global credential.helper 'cache --timeout=86400'

git clone https://gitlab.rhdemo.io/student#/student#-playbooks.git
# Username is student# and your associated password.
cd student#-playbooks/iis_basic
ansible-playbook install_iis.yml
```



If you have already cloned the repository, you can run a 'git pull' within the directory to get the latest version.

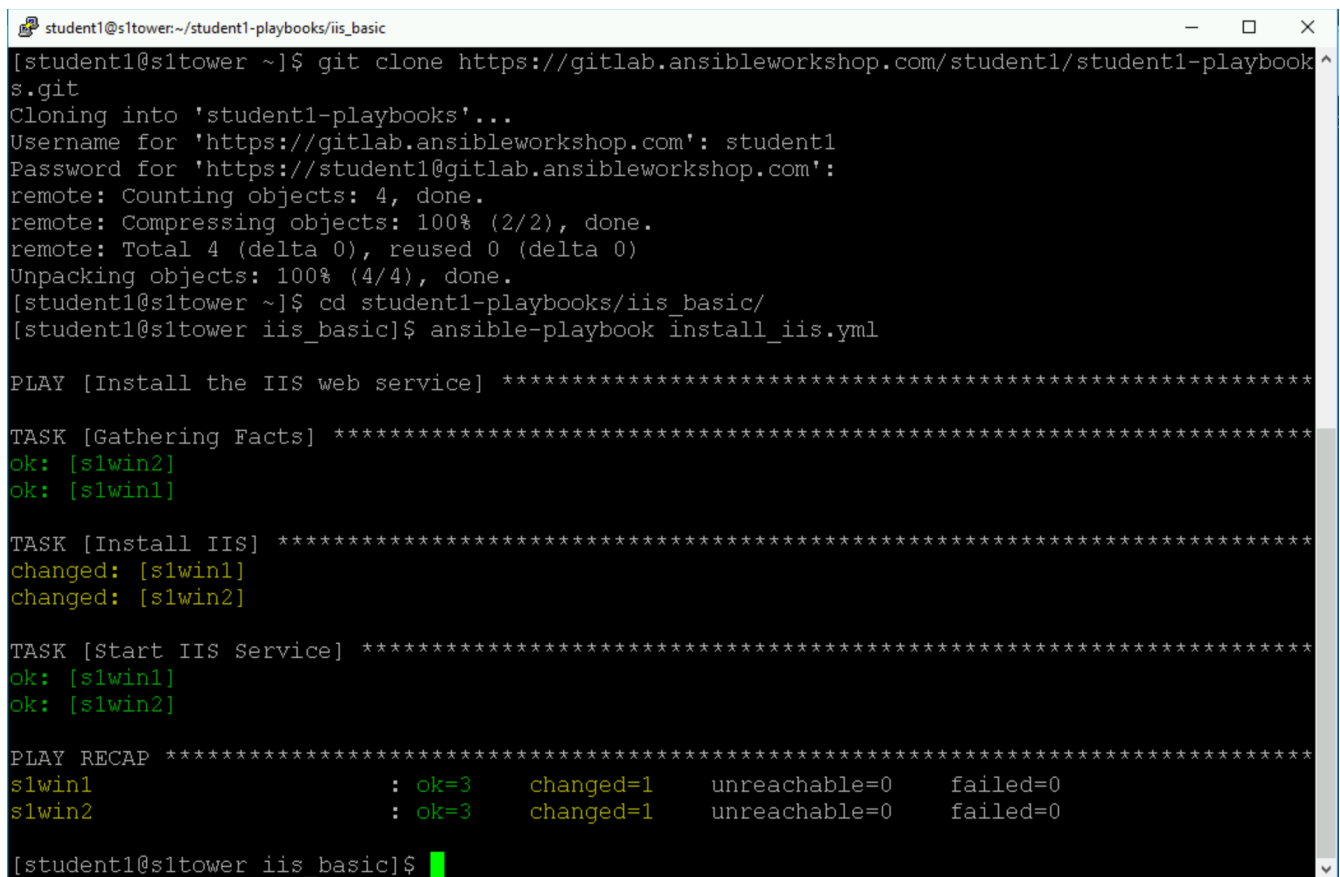
However, before you go ahead and run that command, let's take a few moments to understand some options.

- **-i** This option could be used to specify the inventory file you wish to use. We are using the default (/etc/ansible/hosts)
- **-v** Although not used here, this increases verbosity. Try running your playbook a second time using **-v** or **-vv** to increase the verbosity
- **--syntax-check** If you run into any issues with your playbook running properly; you know, from that copy/pasting that you didn't do because we said "*don't do that*"; you could use this option to help find those issues like so...

```
ansible-playbook install_iis.yml --syntax-check
```

OK, go ahead and run your playbook as specified in **Step 1**

In standard output, you should see something that looks very similar to the following:



```
student1@sltower:~/student1-playbooks/iis_basic
[student1@sltower ~]$ git clone https://gitlab.ansibleworkshop.com/student1/student1-playbook
s.git
Cloning into 'student1-playbooks'...
Username for 'https://gitlab.ansibleworkshop.com': student1
Password for 'https://student1@gitlab.ansibleworkshop.com':
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
[student1@sltower ~]$ cd student1-playbooks/iis_basic/
[student1@sltower iis_basic]$ ansible-playbook install_iis.yml

PLAY [Install the IIS web service] *****

TASK [Gathering Facts] *****
ok: [slwin2]
ok: [slwin1]

TASK [Install IIS] *****
changed: [slwin1]
changed: [slwin2]

TASK [Start IIS Service] *****
ok: [slwin1]
ok: [slwin2]

PLAY RECAP *****
slwin1      : ok=3    changed=1    unreachable=0    failed=0
slwin2      : ok=3    changed=1    unreachable=0    failed=0

[student1@sltower iis_basic]$
```

*install\_iis playbook stdout*

Notice that the play and each task is named so that you can see what is being done and to which node it is being done to. You also may notice a task in there that you didn't write; <cough> **setup** <cough>. This is because the **setup** module runs by default. To turn it off, you can specify **gather\_facts: false** in your play definition like this:

```
---
- hosts: windows
  name: Install the IIS web server
  gather_facts: false
```

## Step 2:

### Remove IIS

OK, for the next several minutes or as much time as we can afford, we want to experiment a little. We would like you to reverse what you've done, i.e. stop and uninstall iis on your web nodes. So, go ahead and make a copy of your playbook in Visual Studio Code named 'remove\_iis.yml', edit and commit the changes, and run as previous. For this exercise we aren't going to show you line by line, but we will give you a few hints.

- If your first task in the playbook was to install Web-Server feature and the second task was to start the service, which order do you think those tasks should be in now?
- If **started** makes sure a service is started, then what option ensures it is stopped?
- If **present** makes sure a feature is installed, then what option ensures it is removed? Er... starts with an **ab**, ends with a **sent**

Feel free to browse the help pages to see a list of all options.

- [Ansible win\\_feature module](#)
- [Ansible win\\_service module](#)

## Exercise 1.3 - Using Variables, Loops, and Handlers

Previous exercises showed you the basics of Ansible Core. In the next few exercises, we are going to teach some more advanced ansible skills that will add flexibility and power to your playbooks.

Ansible exists to make tasks simple and repeatable. We also know that not all systems are exactly alike and often require some slight change to the way an Ansible playbook is run. Enter variables.

Variables are how we deal with differences between your systems, allowing you to account for a change in port, IP address or directory.

Loops enable us to repeat the same task over and over again. For example, lets say you want to start multiple services, install several features, or create multiple directories. By using an ansible loop, you can do that in a single task.

Handlers are the way in which we restart services. Did you just deploy a new config file, install a new package? If so, you may need to restart a service for those changes to take effect. We do that with a handler.

For a full understanding of variables, loops, and handlers; check out our Ansible documentation on these subjects.

[Ansible Variables](#)

[Ansible Loops](#)

[Ansible Handlers](#)

## Section 1: Running the Playbook

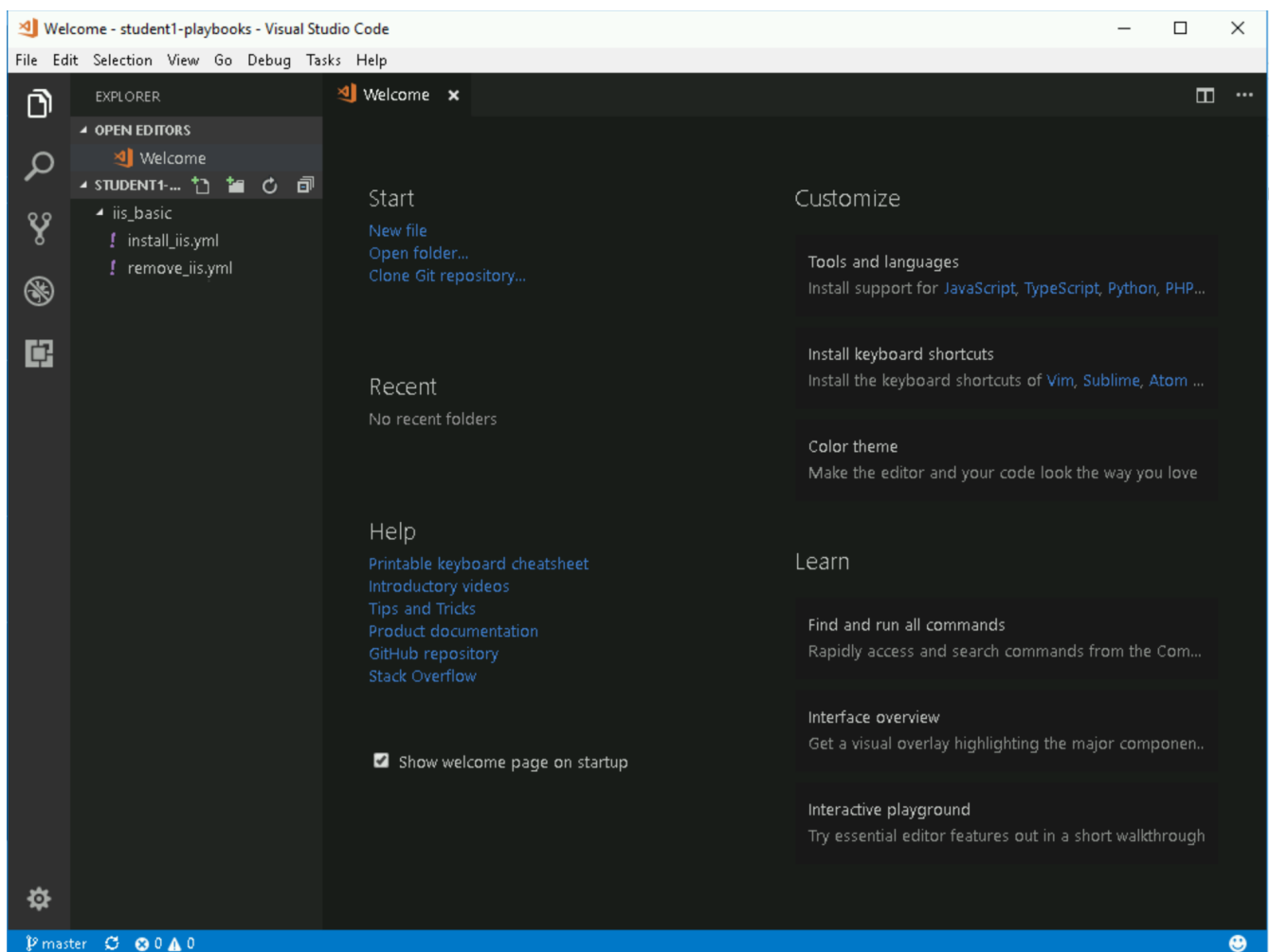
To begin, we are going to create a new playbook, but it should look very familiar to the one you created in exercise 1.2

We are now going to run you're brand spankin' new playbook on your two web nodes. To do this, you are going to use the `ansible-playbook` command.

### Step 1:

Within Visual Studio Code, create a new directory in your git repo and create a site.yml file.

In the Explorer accordion you should have a 'student#-playbooks' section where you previously made iis\_basic.



#### Student Playbooks

Create a folder called `iis_basic_playbook` and a file called `site.yml`

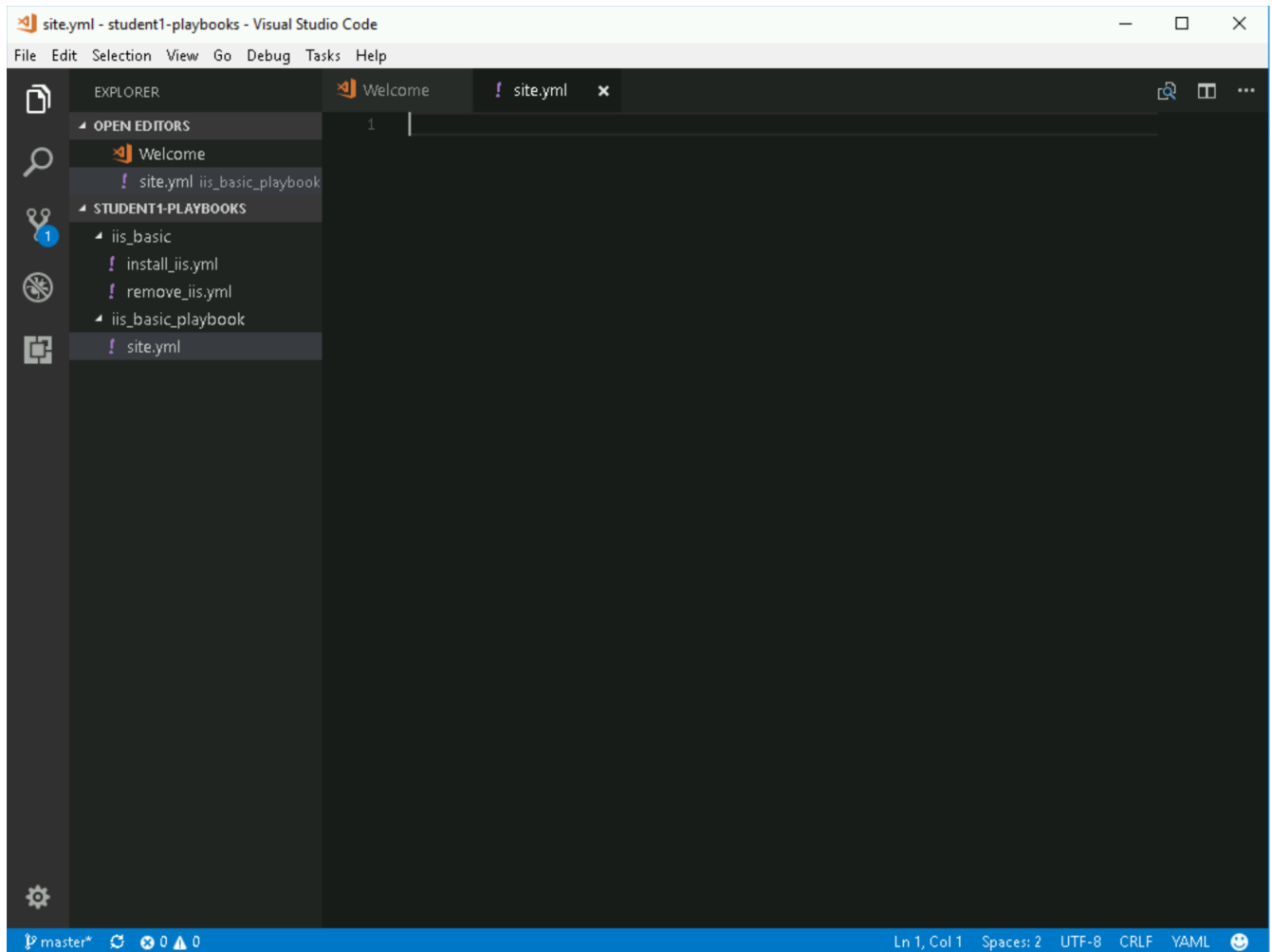
Hover over the 'student#-playbooks' section and click on the 'New Folder' button

Type `iis_basic_playbook` and hit enter. Then click on that folder so it is selected.

Hover over the **student#-playbooks** section again and click on the **New File** button.

Type **site.yml** and press enter.

You should now have an editor open in the right pane that can be used for creating your playbook.



*Empty site.yml*

## Step 2:

Add a play definition and some variables to your playbook. These include additional packages your playbook will install on your web servers, plus some web server specific configurations.



```

---
- hosts: windows
  name: This is a play within a playbook
  vars:
    iis_sites:
      - name: 'Ansible Playbook Test'
        port: '8080'
        path: 'C:\sites\playbooktest'
      - name: 'Ansible Playbook Test 2'
        port: '8081'
        path: 'C:\sites\playbooktest2'
    iis_test_message: "Hello World!  My test IIS Server"

```

## Step 3:

Add a new task called **install IIS**.

```

tasks:
  - name: Install IIS
    win_feature:
      name: Web-Server
      state: present

  - name: Create site directory structure
    win_file:
      path: "{{ item.path }}"
      state: directory
    with_items: "{{ iis_sites }}"

  - name: Create IIS site
    win_iis_website:
      name: "{{ item.name }}"
      state: started
      port: "{{ item.port }}"
      physical_path: "{{ item.path }}"
    with_items: "{{ iis_sites }}"
    notify: restart iis service

```

```
1 ---
2 - hosts: windows
3   name: This is a play within a playbook
4   vars:
5     iis_sites:
6       - name: 'Ansible Playbook Test'
7         port: '8080'
8         path: 'C:\sites\playbooktest'
9       - name: 'Ansible Playbook Test 2'
10        port: '8081'
11        path: 'C:\sites\playbooktest2'
12     iis_test_message: "Hello World! --- My test IIS Server"
13
14   tasks:
15     - name: Install IIS
16       win_feature:
17         name: Web-Server
18         state: present
19
20     - name: Create site directory structure
21       win_file:
22         path: "{{ item.path }}"
23         state: directory
24         with_items: "{{ iis_sites }}"
25
26     - name: Create IIS site
27       win_iis_website:
28         name: "{{ item.name }}"
29         state: started
30         port: "{{ item.port }}"
31         physical_path: "{{ item.path }}"
32         with_items: "{{ iis_sites }}"
33       notify: restart iis service
```

site.yml part 1

### What the Helsinki is happening here!?



- **vars:** You've told Ansible the next thing it sees will be a variable name
- **iis\_sites** You are defining a list-type variable called iis\_sites. What follows is a list of each site with it's related variables
- **file:** This module is used to create, modify, delete files, directories, and symlinks.
- **{{ item }}** You are telling Ansible that this will expand into a list item. Each item has several variables like **name**, **port**, and **path**.
- **with\_items: "{{ iis\_sites }}"** This is your loop which is instructing Ansible to perform this task on every **item** in **iis\_sites**
- **notify: restart iis service** This statement is a **handler**, so we'll come back to it in Section 3.

## Section 2: Opening Firewall and Deploying Files

When you need to do pretty much anything with files and directories, use one of the [Ansible Files](#)

modules. We already used the `win_file` module to create our directory. Next we'll leverage the `win_template` modules to create a dynamic file using variables.

After that, you will define a task to start the start the apache service.

## Step 1:

Create a **templates** directory in your project directory and create a template as follows:

Ensure your **iis\_basic\_playbook** folder is highlighted and then hover over the **student#-playbooks** section and click the **New Folder** button.

Type **templates** and hit enter. Then click on that folder so it is selected.

Hover over the **student#-playbooks** section again and click the **New File** button.

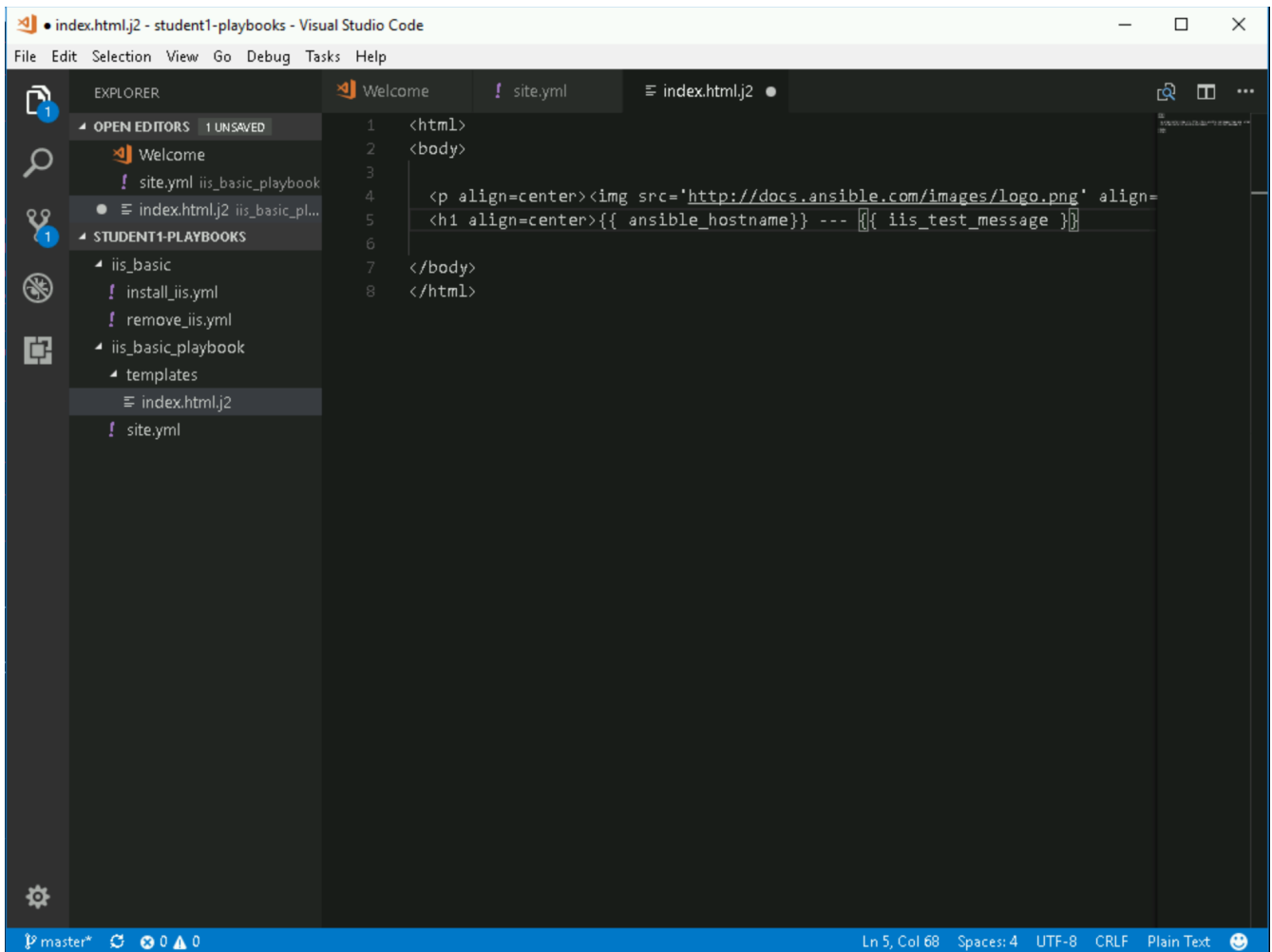
Type **index.html.j2** and press enter.

You should now have an editor open in the right pane that can be used for creating your template. Enter the following details:

```
<html>
<body>

  <p align=center><img src='http://docs.ansible.com/images/logo.png' align=>
  <h1 align=center>{{ ansible_hostname }} --- {{ iis_test_message }}

</body>
</html>
```



*index.html template*

## Step 2:

Add to your playbook, **site.yml**, opening your firewall ports and writing the template. Use single quotes for win\_template in order to not escape the forward slash.

```

- name: Open port for site on the firewall
  win_firewall_rule:
    name: "iisport{{ item.port }}"
    enable: yes
    state: present
    localport: "{{ item.port }}"
    action: Allow
    direction: In
    protocol: Tcp
    force: true
    with_items: "{{ iis_sites }}"

- name: Template simple web site to iis_site_path as index.html
  win_template:
    src: 'index.html.j2'
    dest: '{{ item.path }}\index.html'
    with_items: "{{ iis_sites }}"

```

#### So... what did I just write?



- **win\_firewall\_rule:** This module is used to create, modify, and update firewall rules. Note in the case of AWS there are also security group rules which may impact communication. We've opened these for the ports in this example.
- **win\_template:** This module specifies that a jinja2 template is being used and deployed.
- **jinja-who?** - Not to be confused with 2013's blockbuster "Ninja II - Shadow of a Tear", [jinja2](#) is used in Ansible to transform data inside a template expression, i.e. filters.

## Section 3: Defining and Using Handlers

There are any number of reasons we often need to restart a service/process including the deployment of a configuration file, installing a new package, etc. There are really two parts to this Section; adding a handler to the playbook and calling the handler after the a task. We will start with the former.

### Step 1:

Define a handler.

```
handlers:
  - name: restart iis service
    win_service:
      name: W3Svc
      state: restarted
      start_mode: auto
```



### You can't have a former if you don't mention the latter

- **handler:** This is telling the **play** that the **tasks:** are over, and now we are defining **handlers:**. Everything below that looks the same as any other task, i.e. you give it a name, a module, and the options for that module. This is the definition of a handler.
- **notify: restart iis service** ...and here is your latter. Finally! The **notify** statement is the invocation of a handler by name. Quite the reveal, we know. You already noticed that you've added a **notify** statement to the **win\_iis\_website** task, now you know why.

## Section 4: Commit and Review

Your new, improved playbook is done! But remember we still need to commit the changes to source code control.

Click **File** → **Save All** to save the files you've written

```
25 |  
26 |  
27 | - name: Create IIS site  
28 |   win_iis_website:  
29 |     name: "{{ item.name }}"  
30 |     state: started  
31 |     port: "{{ item.port }}"  
32 |     physical_path: "{{ item.path }}"  
33 |   with_items: "{{ iis_sites }}"  
34 |   notify: restart iis service  
35 |  
36 | - name: Open site's port on firewall  
37 |   win_firewall_rule:  
38 |     name: "iisport{{ item.port }}"  
39 |     enable: yes  
40 |     state: present  
41 |     localport: "{{ item.port }}"  
42 |     action: Allow  
43 |     direction: In  
44 |     protocol: Tcp  
45 |     force: true  
46 |   with_items: "{{ iis_sites }}"  
47 |  
48 | - name: Template simple web site to iis_site_path as index.html  
49 |   win_template:  
50 |     src: 'index.html.j2'  
51 |     dest: '{{ item.path }}\index.html'  
52 |     with_items: "{{ iis_sites }}"  
53 | handlers:  
54 |   - name: restart iis service  
55 |     win_service:  
56 |       name: W3Svc  
57 |       state: restarted  
58 |       start_mode: auto
```

### site.yml part 2

Click on the Source Code icon, type in a commit message such as **Adding basic playbook**, and click the check box above.

```

25
26
27 - name: Create IIS site
28   win_iis_website:
29     name: "{{ item.name }}"
30     state: started
31     port: "{{ item.port }}"
32     physical_path: "{{ item.path }}"
33   with_items: "{{ iis_sites }}"
34   notify: restart iis service
35
36 - name: Open site's port on firewall
37   win_firewall_rule:
38     name: "iisport{{ item.port }}"
39     enable: yes
40     state: present
41     localport: "{{ item.port }}"
42     action: Allow
43     direction: In
44     protocol: Tcp
45     force: true
46   with_items: "{{ iis_sites }}"
47
48 - name: Template simple web site to iis_site_path as index.html
49   win_template:
50     src: "index.html.j2"
51     dest: "{{ item.path }}\index.html"
52   with_items: "{{ iis_sites }}"
53 handlers:
54   - name: restart iis service
55     win_service:
56       name: WBSvc
57       state: restarted
58       start_mode: auto

```

*Commit site.yml*

Sync to gitlab by clicking the arrows on the lower left blue bar. When prompted, click **OK** to push and pull commits.

It should take 20-30 seconds to finish the commit. The blue bar should stop rotating and indicate 0 problems...

Don't run it just yet, we'll do that in our next exercise. For now, let's take a second look to make sure everything looks the way you intended. If not, now is the time for us to fix it up. The figure below shows the complete playbook. Please note the spacing.

```

---
- hosts: windows
  name: This is a play within a playbook
  vars:
    iis_sites:
      - name: 'Ansible Playbook Test'
        port: '8080'
        path: 'C:\sites\playbooktest'
      - name: 'Ansible Playbook Test 2'
        port: '8081'
        path: 'C:\sites\playbooktest2'
    iis_test_message: "Hello World!  My test IIS Server"

```



#### tasks:

- name: Install IIS  
win\_feature:  
  name: Web-Server  
  state: present
- name: Create site directory structure  
win\_file:  
  path: "{{ item.path }}"  
  state: directory  
with\_items: "{{ iis\_sites }}"
- name: Create IIS site  
win\_iis\_website:  
  name: "{{ item.name }}"  
  state: started  
  port: "{{ item.port }}"  
  physical\_path: "{{ item.path }}"  
with\_items: "{{ iis\_sites }}"  
notify: restart iis service
- name: Open port for site on the firewall  
win\_firewall\_rule:  
  name: "iisport{{ item.port }}"  
  enable: yes  
  state: present  
  localport: "{{ item.port }}"  
  action: Allow  
  direction: In  
  protocol: Tcp  
  force: true  
with\_items: "{{ iis\_sites }}"
- name: Template simple web site to iis\_site\_path as index.html  
win\_template:  
  src: 'index.html.j2'  
  dest: "{{ item.path }}\index.html"  
with\_items: "{{ iis\_sites }}"

#### handlers:

- name: restart iis service  
win\_service:  
  name: W3Svc  
  state: restarted  
  start\_mode: auto

# Exercise 1.4 - Running the apache-basic-playbook

Congratulations! You just wrote a playbook that incorporates some key Ansible concepts that you use in most if not all of your future playbooks. Before you get too excited though, we should probably make sure it actually runs.

So, lets do that now.

## Section 1 - Running your new iis playbook

### Step 1:

Use putty to login to your student#-control host as student#

Pull the latest playbooks from source code and then execute the playbook. Be certain to use your student number below.

```
cd ~/student#-playbooks
git pull
# Or if you don't have the directory:
# git clone https://gitlab.rhdemo.io/student#/student#-playbooks.git
cd iis_basic_playbook
```



Since you already have an inventory file, `~/lightbulb/lessons/lab_inventory/student#-instances.txt`, we will re-use it.

### Step 2:

Run your playbook

```
ansible-playbook site.yml
```

## Section 2: Review

If successful, you should see standard output that looks very similar to the following. If not, just let us know. We'll help get things fixed up.

```
student1@sltower:~/student1-playbooks/iis_basic_playbook
Test 2', u'port': u'8081'}}

TASK [Open site's port on firewall] *****
ok: [slwin1] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook Test', u'port': u'8080'})
ok: [slwin2] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook Test', u'port': u'8080'})
ok: [slwin1] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test 2', u'port': u'8081'})
ok: [slwin2] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test 2', u'port': u'8081'})

TASK [Template simple web site to iis_site_path as index.html] *****
ok: [slwin1] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook Test', u'port': u'8080'})
ok: [slwin2] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook Test', u'port': u'8080'})
ok: [slwin1] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test 2', u'port': u'8081'})
ok: [slwin2] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test 2', u'port': u'8081'})

RUNNING HANDLER [restart iis service] *****
changed: [slwin1]
changed: [slwin2]

PLAY RECAP *****
slwin1      : ok=7    changed=2    unreachable=0    failed=0
slwin2      : ok=7    changed=2    unreachable=0    failed=0

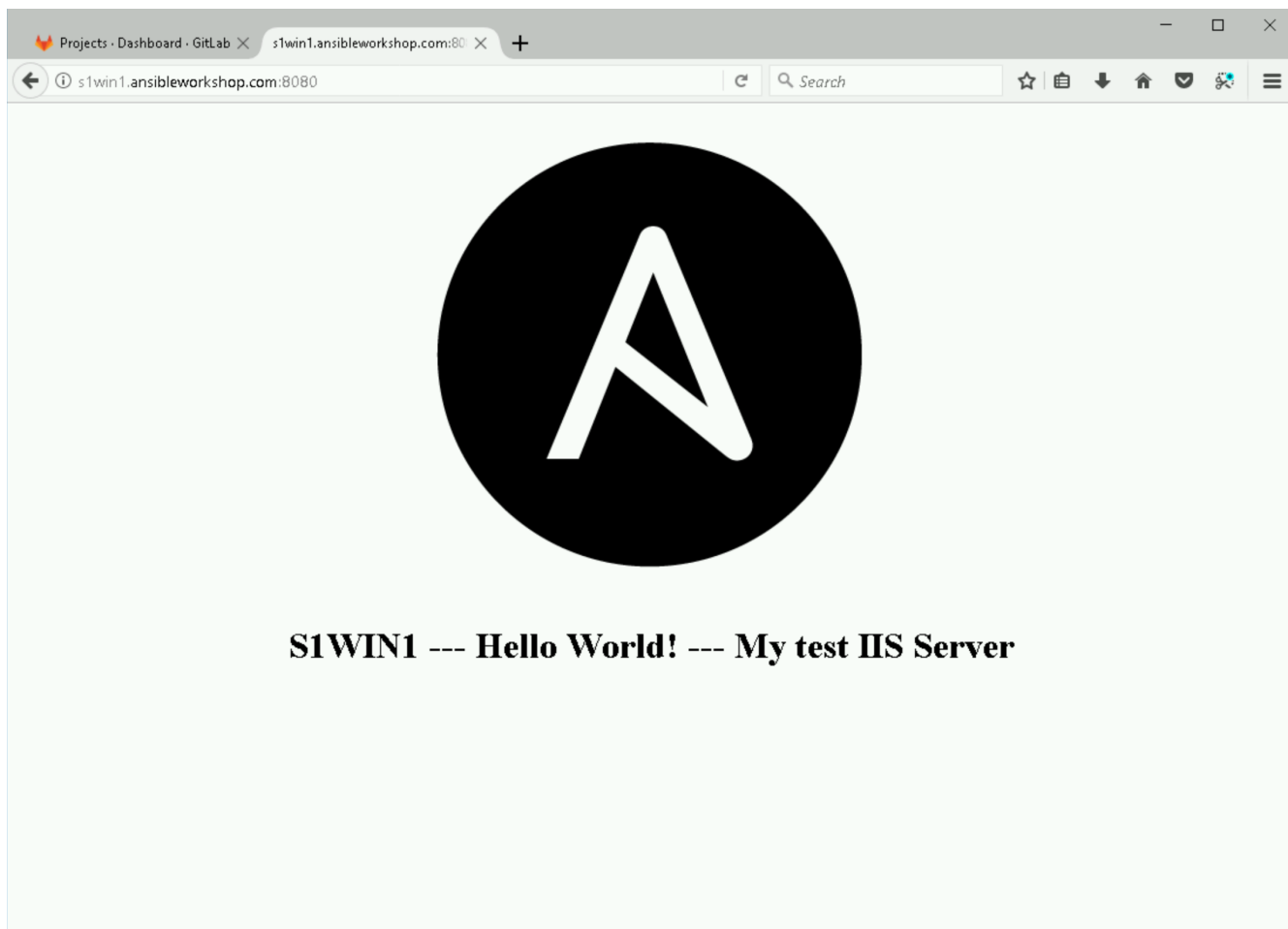
[student1@sltower iis_basic_playbook]$
```

*site.yml stdout*

Your output may vary from above based on whether you removed IIS previously or not.

Once completed, on your workstation you can open firefox and test a connection to one of your servers (student#-node1 or student#-node2):

- <http://student#-node1.rhdemo.io:8080/>
- <http://student#-node2.rhdemo.io:8081/>



*IIS Templated Site*



DNS is only available in your workstation environment, so you will not be able to connect remotely from your own machine

## Exercise 1.5 - Roles: Making your playbooks reusable

While it is possible to write a playbook in one file as we've done throughout this workshop, eventually you'll want to reuse files and start to organize things.

Ansible Roles is the way we do this. When you create a role, you deconstruct your playbook into parts and those parts sit in a directory structure. "Wha?? You mean that seemingly useless [best practice](#) you mentioned in exercise 1.1?". Yep, that one.

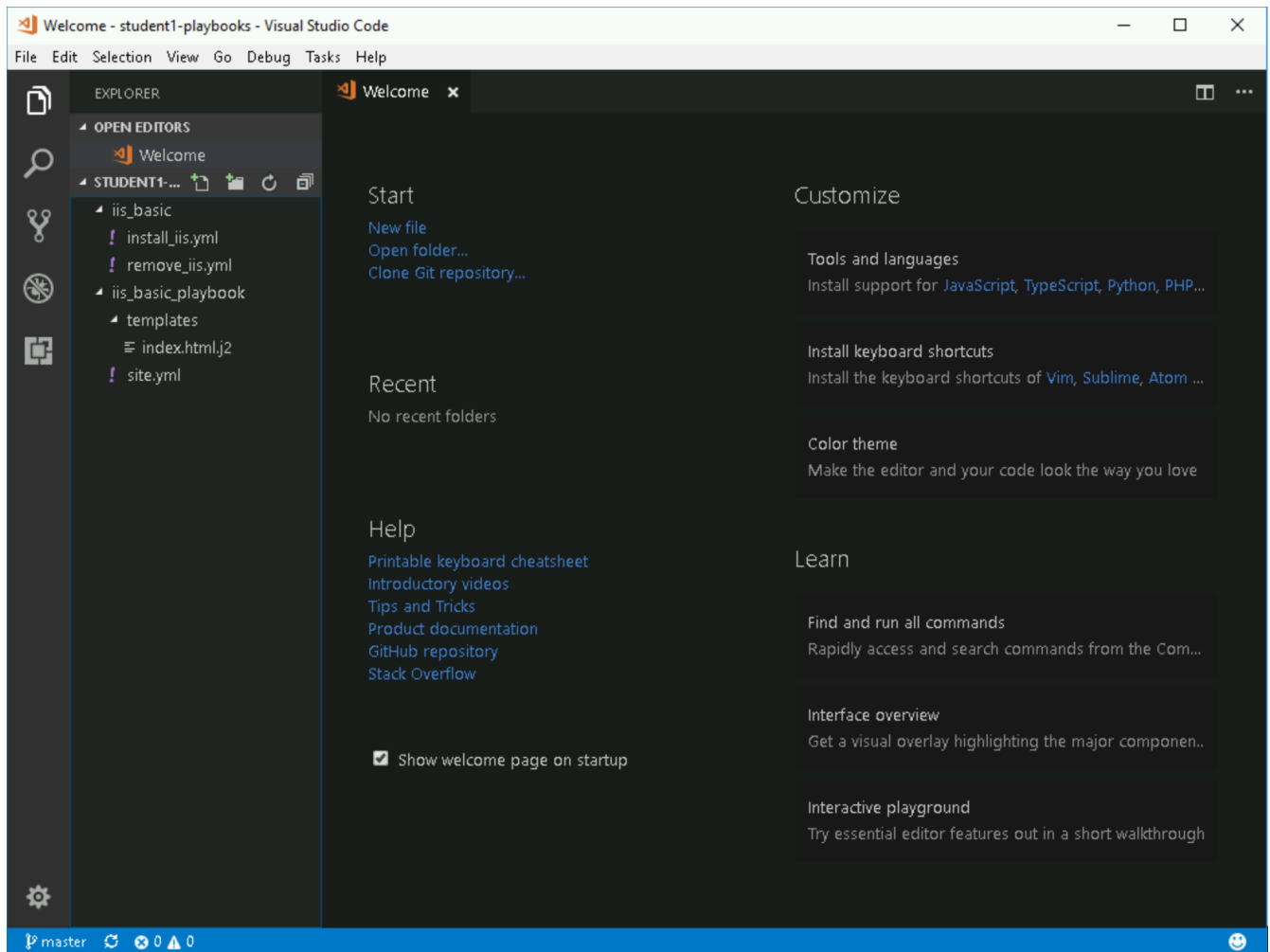
For this exercise, you are going to take the playbook you just wrote and refactor it into a role.

Let's begin with seeing how your iis-basic-playbook will break down into a role...

# Section 1: Create directory structure for your new role

## Step 1:

In Visual Studio Code, navigate to explorer and your `student#-playbooks` project where you previously made `iis_basic_playbook`.



`iis_basic_playbook`

Select the **iis\_basic\_playbook** folder.

Create a directory called **roles** by right-clicking **iis\_basic\_playbook** and selecting **New Folder**.

Now right-click on **roles** and create a new folder called **iis\_simple**.

## Step 2:

Within **iis\_simple** create new folders as follows:

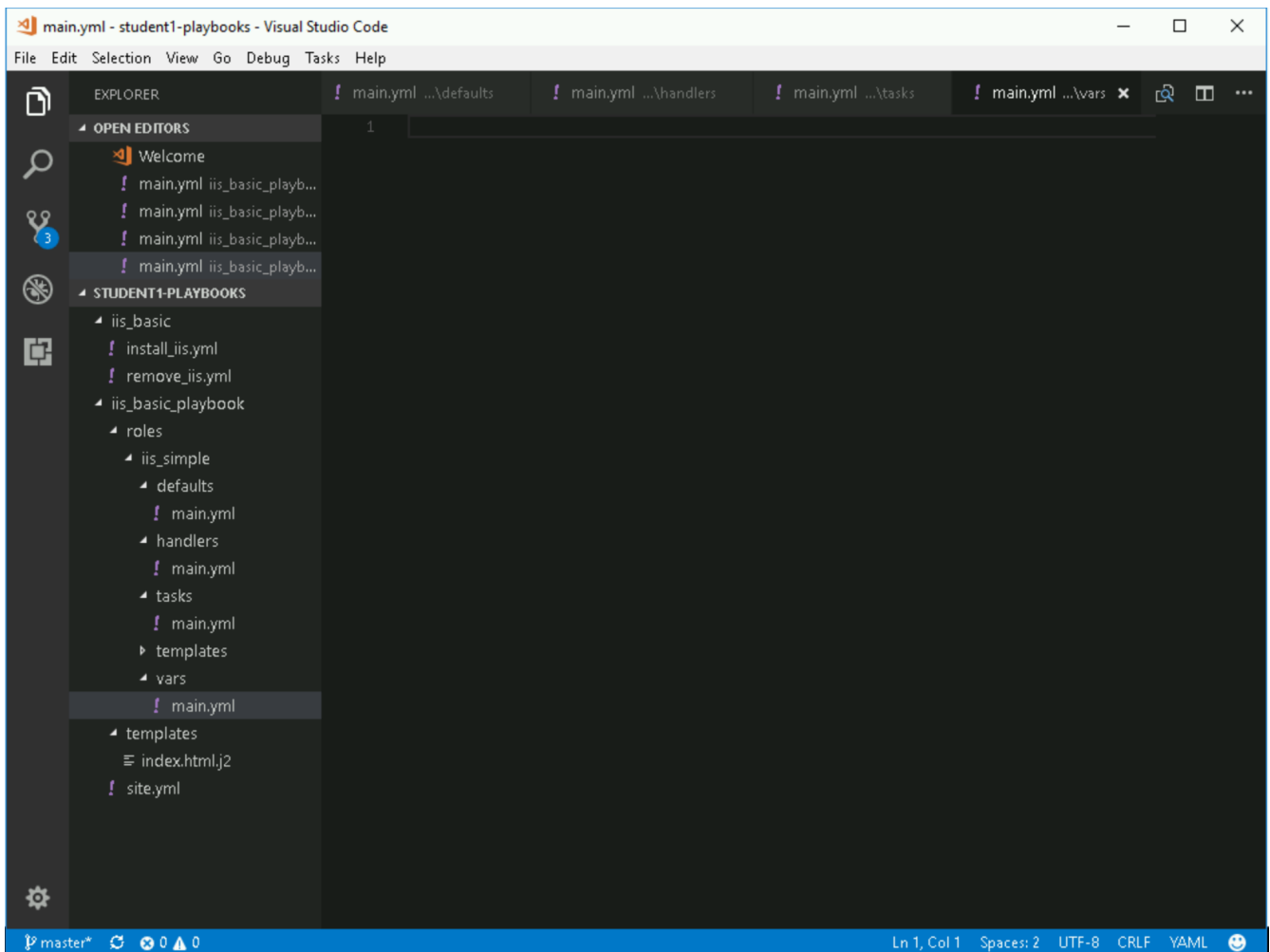
- defaults
- vars

- handlers
- tasks
- templates

## Step 3:

Within each of these new folders, right-click and create **New File**. Create a file called **main.yml** in each of these folders. Do not do this under templates as we will create individual template files. This is your basic role structure and main.yml will be the default file that the role will use for each section.

The finished structure will look like this:



*Role Structure*

## Section 2: Breaking Your `site.yml` Playbook into the Newly Created `iis_simple` Role

In this section, we will separate out the major parts of your playbook including `vars:`, `tasks:`, `template:`, and `handlers:`.

## Step 1:

Make a backup copy of **site.yml**, then create a new **site.yml**.

Navigate to your **iis\_basic\_playbook** folder, right click **site.yml**, click **rename**, and name it as **site.yml.backup**.

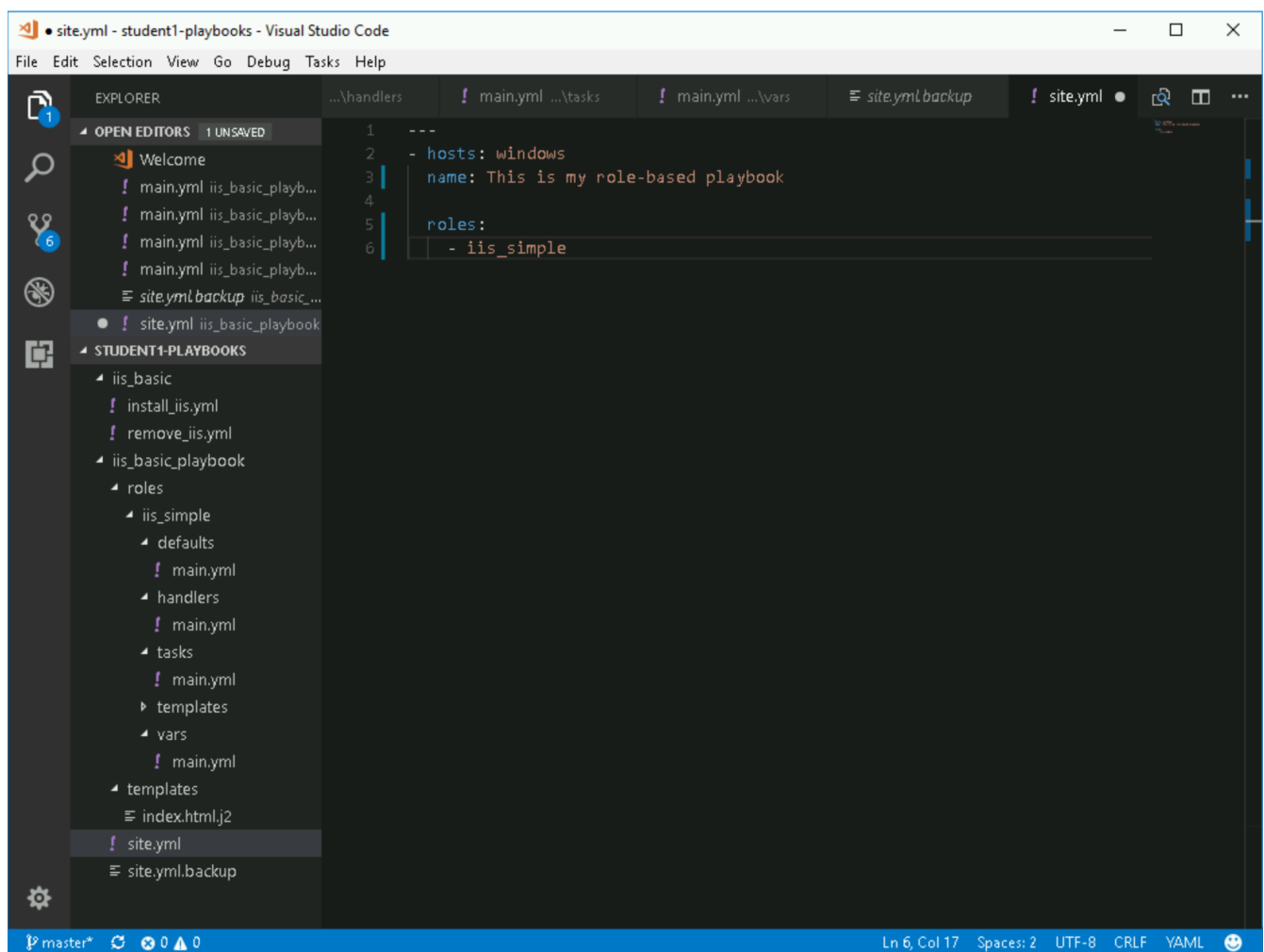
Create a blank new file called **site.yml** in the same folder.

## Step 2:

Update **site.yml** to look like to only call your role. It should look like below:

```
---
- hosts: windows
  name: This is my role-based playbook

roles:
  - iis_simple
```



*New site.yml*

## Step 3:

Add a default variable to your role. Edit the **roles\iis\_simple\defaults\main.yml** as follows:

```
---
# defaults file for iis_simple
iis_sites:
  - name: 'Ansible Playbook Test'
    port: '8080'
    path: 'C:\sites\playbooktest'
  - name: 'Ansible Playbook Test 2'
    port: '8081'
    path: 'C:\sites\playbooktest2'
```

## Step 4:

Add some role-specific variables to your role in **roles\iis\_simple\vars\main.yml**.

```
---
# vars file for iis_simple
iis_test_message: "Hello World!  My test IIS Server"
```

**Hey, wait just a minute there buster... did you just have us put variables in two seperate places?**

Yes... yes we did. Variables can live in quite a few places. Just to name a few:

- vars directory
- defaults directory
- group\_vars directory
- In the playbook under the **vars:** section
- In any file which can be specified on the command line using the **--extra\_vars** option
- On a boat, in a moat, with a goat (*disclaimer: this is a complete lie*)



Bottom line, you need to read up on [variable precedence](#) to understand both where to define variables and which locations take precedence. In this exercise, we are using role defaults to define a couple of variables and these are the most malleable. After that, we defined some variables in **/vars** which have a higher precedence than defaults and can't be overridden as a default variable.

## Step 5:

Create your role handler in **roles\iis\_simple\handlers\main.yml**.



```
---  
# handlers file for iis_simple  
- name: restart iis service  
  win_service:  
    name: W3Svc  
    state: restarted  
    start_mode: auto
```

## Step 6:

Add tasks to your role in **roles\iis\_simple\tasks\main.yml**.

```

---
# tasks file for iis_simple

- name: Install IIS
  win_feature:
    name: Web-Server
    state: present

- name: Create site directory structure
  win_file:
    path: "{{ item.path }}"
    state: directory
  with_items: "{{ iis_sites }}"

- name: Create IIS site
  win_iis_website:
    name: "{{ item.name }}"
    state: started
    port: "{{ item.port }}"
    physical_path: "{{ item.path }}"
  with_items: "{{ iis_sites }}"
  notify: restart iis service

- name: Open port for site on the firewall
  win_firewall_rule:
    name: "iisport{{ item.port }}"
    enable: yes
    state: present
    localport: "{{ item.port }}"
    action: Allow
    direction: In
    protocol: Tcp
    force: true
  with_items: "{{ iis_sites }}"

- name: Template simple web site to iis_site_path as index.html
  win_template:
    src: 'index.html.j2'
    dest: '{{ item.path }}\index.html'
  with_items: "{{ iis_sites }}"

```

## Step 7:

Add your **index.html** template.

Right-click **roles\iis\_simple\templates** and create a new file called **index.html.j2** with the following content:

```
<html>
<body>

  <p align=center><img src='http://docs.ansible.com/images/logo.png' align=>
  <h1 align=center>{{ ansible_hostname }} --- {{ iis_test_message }}

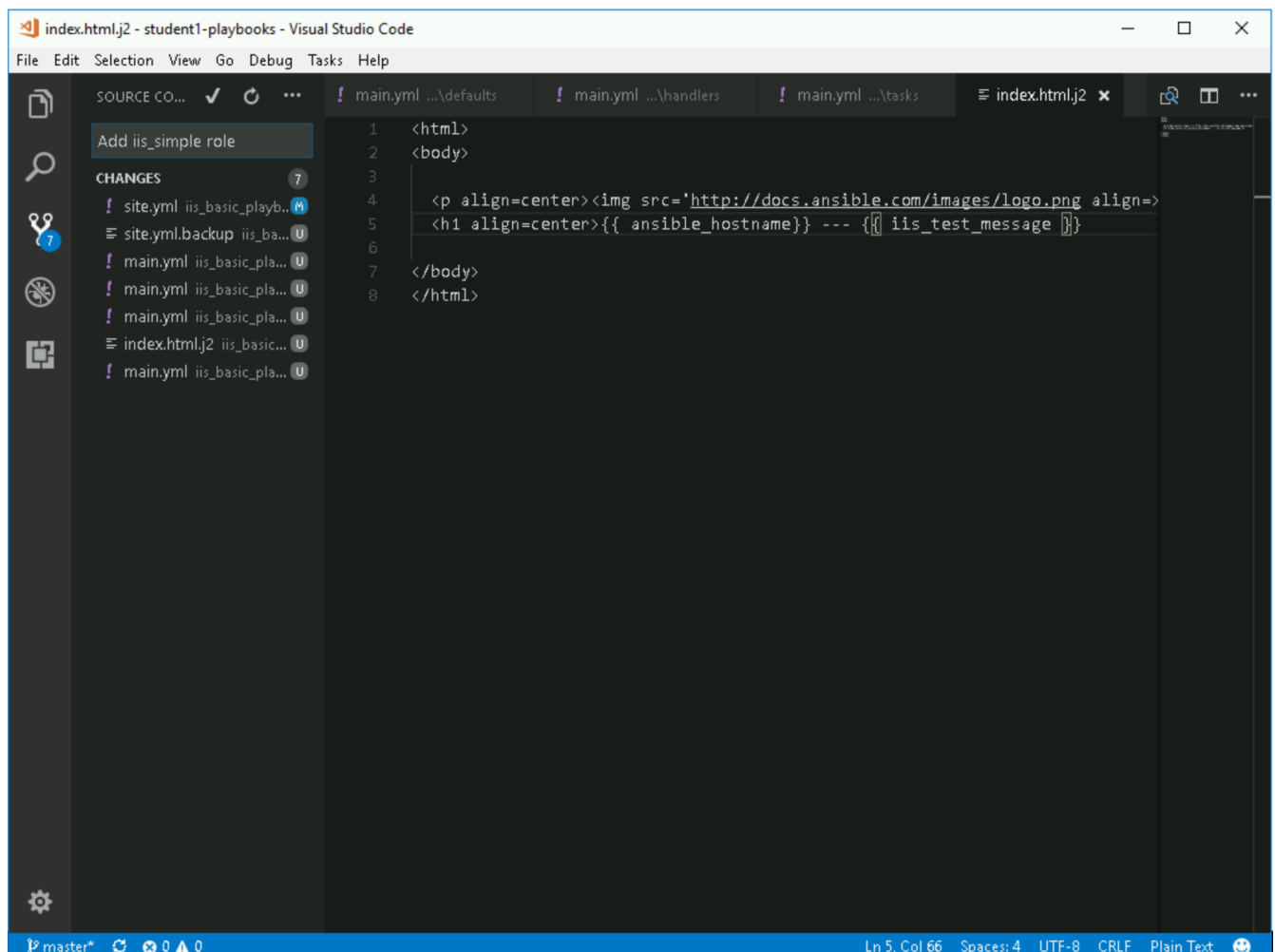
</body>
</html>
```

## Step 8: Commit

Click **File** → **Save All** to ensure all your files are saved.

Click the Source Code icon as shown below.

Type in a commit message like **Add iis\_simple role** and click the check box above.



*Commit iis\_simple\_role*

Click the **synchronize changes** button on the blue bar at the bottom left (and click **OK**). This should again return with no problems.

## Section 3: Running your new role-based playbook

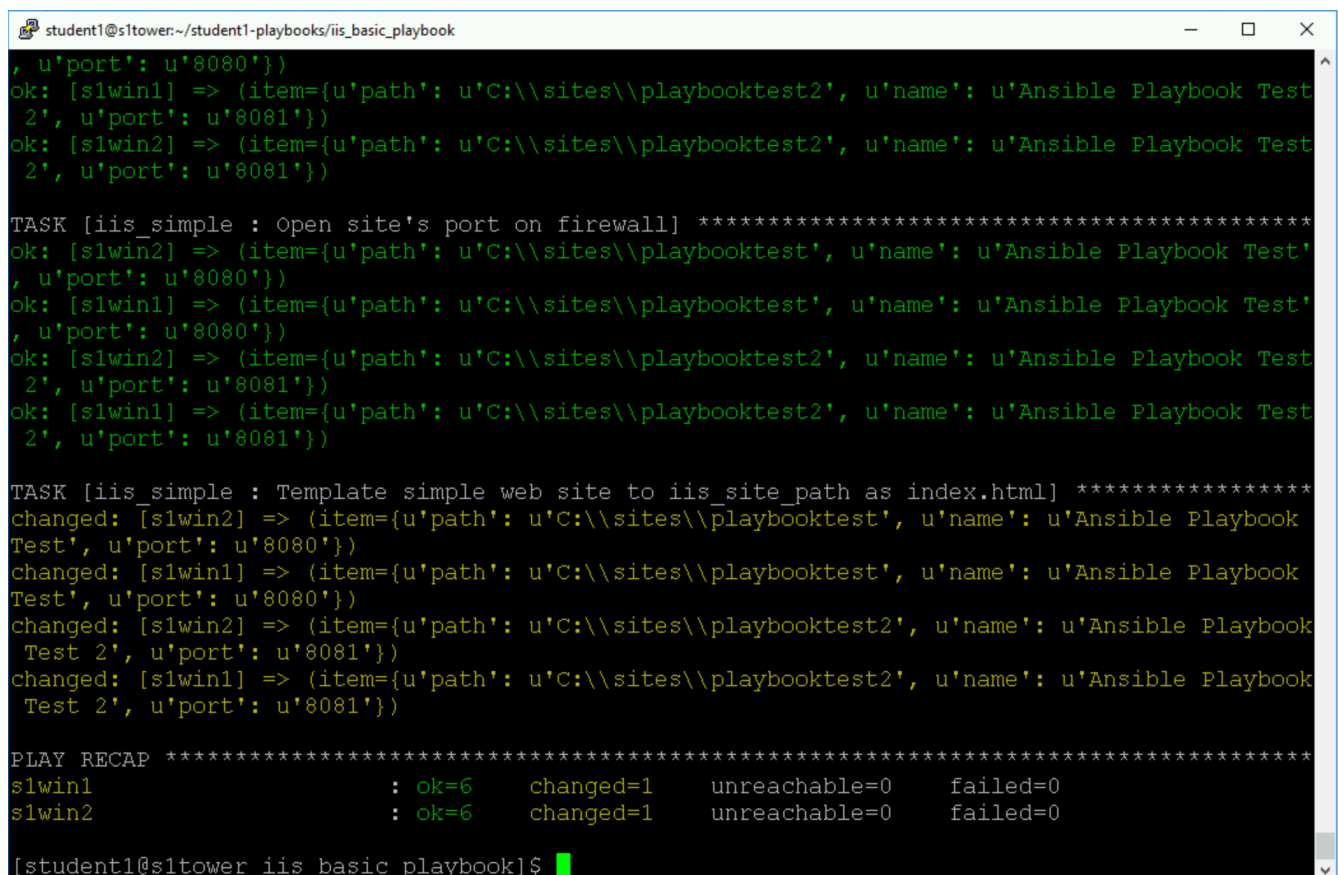
Now that you've successfully separated your original playbook into a role, let's run it and see how it works.

### Step 1:

Run the playbook on your tower host.

```
cd ~/student#-playbooks
git pull
cd iis_basic_playbook
ansible-playbook site.yml
```

If successful, your standard output should look similar to the figure below. Note that most of the tasks return **"ok"** because we've previously configured the servers and services are already running.



```
student1@sttower: ~/student1-playbooks/iis_basic_playbook
, u'port': u'8080'})
ok: [s1win1] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test
2', u'port': u'8081'})
ok: [s1win2] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test
2', u'port': u'8081'})

TASK [iis_simple : Open site's port on firewall] *****
ok: [s1win2] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook Test'
, u'port': u'8080'})
ok: [s1win1] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook Test'
, u'port': u'8080'})
ok: [s1win2] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test
2', u'port': u'8081'})
ok: [s1win1] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook Test
2', u'port': u'8081'})

TASK [iis_simple : Template simple web site to iis_site_path as index.html] *****
changed: [s1win2] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook
Test', u'port': u'8080'})
changed: [s1win1] => (item={u'path': u'C:\\sites\\playbooktest', u'name': u'Ansible Playbook
Test', u'port': u'8080'})
changed: [s1win2] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook
Test 2', u'port': u'8081'})
changed: [s1win1] => (item={u'path': u'C:\\sites\\playbooktest2', u'name': u'Ansible Playbook
Test 2', u'port': u'8081'})

PLAY RECAP *****
s1win1                : ok=6    changed=1    unreachable=0    failed=0
s1win2                : ok=6    changed=1    unreachable=0    failed=0

[student1@sttower iis_basic_playbook]$
```

*Role site.yml stdout*

## Section 4: Review

You should now have a completed playbook, **site.yml** with a single role called **iis\_simple**. The

advantage of structuring your playbook into roles is that you can now add reusability to your playbooks as well as simplifying changes to variables, tasks, templates, etc. [Ansible Galaxy](#) is a good repository of roles for use or reference.