Automation for Everyone

Today’s Agenda

- Arrivals, Introductions
- Workshop Setup
- Ansible Labs
- Ansible Tower Labs
- Use Cases

Before We Get Started

You will need a system which has an RDP client in order to interact with the lab.

Overview

Today is meant for anyone who has any exposure to Ansible, whether you have used it or not. The format of this discussion is a combination of presentation, demonstration, and hands-on labs.

Your Responsibilities

- Have a discussion. This will be boring if it's just us up here talking for over 4 hours.
- Participate. We are going to cut you loose with Ansible here in just a little while. Have questions. Have opinions.

Hopefully you have your laptop with you. If not, please find a shoulder-surfing buddy. See? Not only can we dig into Ansible but you can make a new friend!

Conventions Used in This Document

Whenever you see a section like this, please pay close attention. We are leading you through a process.

Step 1: Run this command to do something or other

```
ansible windows -m setup
```

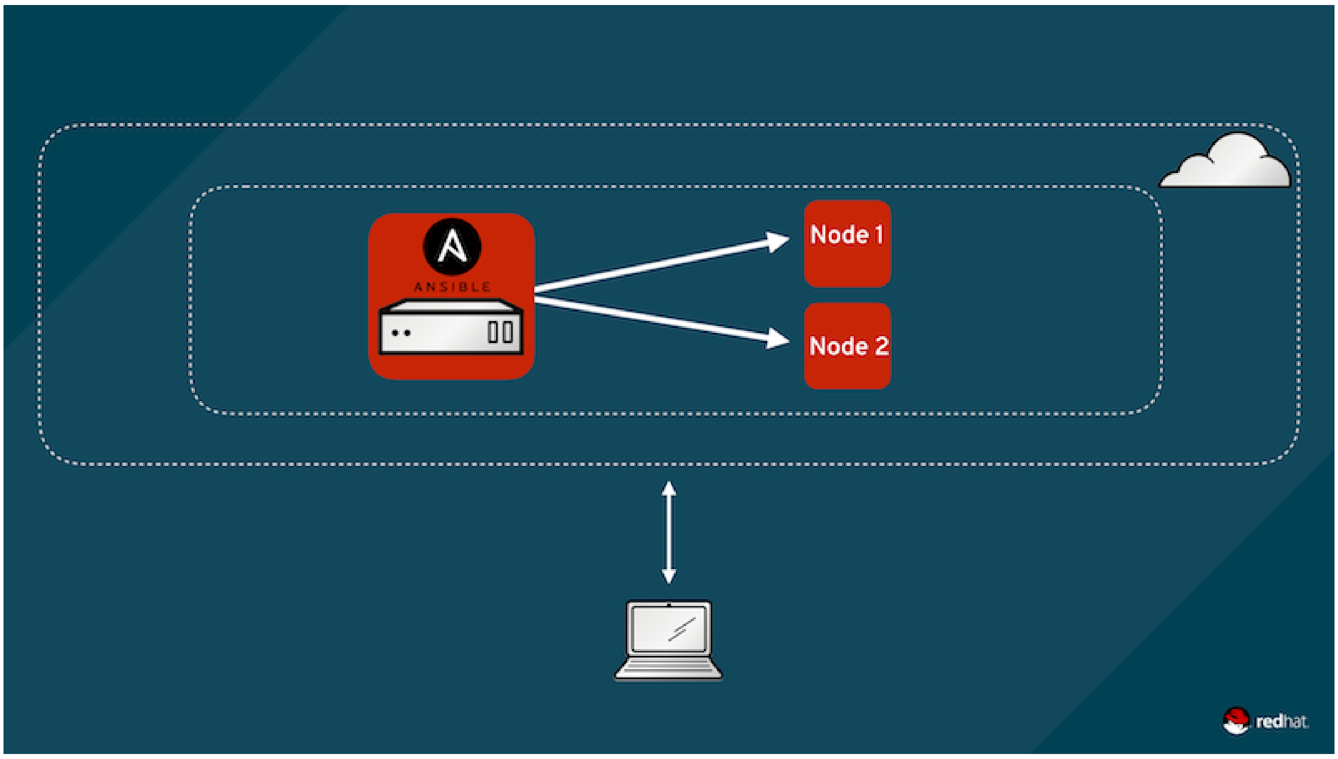
Step 2: Edit this file in this specific way

```
---
- hosts: web
  name: This is a play within a playbook
  become: yes
  vars:
    httpd_packages:
      - httpd
      - mod_wsgi
```

Do not cut and paste text from this document. We really want you to learn ansible during this workshop, and part of that experience is typing out YAML and understanding it’s syntax, spacing, and alignments.

Document Copyright

Workshop Lab Setup



Setup

Today’s workshop infrastructure is being run in Amazon EC2.

Each student will have four instances to use. Connectivity details will be provided by the instructor.

Central Active Directory, DNS, and GitLab environments are available for use.

Workshop Infrastructure

Each student's personal EC2 lab is equipped with the following:

- (1) Control node (Linux host to run ansible plays and install Ansible Tower).
  - Red Hat Enterprise 7.4
- (1) Windows Workstation (to interact with the environment, edit code, etc).
  - Windows 2016
- (2) Windows Endpoint nodes (to execute ansible playbooks upon).
  - Windows 2016
- Ansible Engine 2.4
- Ansible Tower 3.2

Setup Your Environment

Before we can begin doing super-cool automations and such, we've got to get a few basics out of the way. Namely… RDP and SSH access to your EC2 nodes.

Testing your environment

Log into your workstation instance with an RDP client using the details provided by your instructor. You will see a few things pre-installed for use today:

- Putty (SSH to the Ansible Linux host)
- Visual Studio Code (Editor for creating playbooks)
- Git (Source Code Control)
- Firefox

Tower

From your workstation, open up a putty window and connect to 's#tower' where # is your student number. The user will be 'student#', and your password will be provided by the instructor.

> Be sure you can log in to your tower and workstation instances. If you can’t log in start shouting, loudly, and waving your hands! We will test access to the other hosts in the first exercise.

End Result

At this point, everyone should have logged into your tower and workstation nodes. If you haven’t, let us know so we can get you fixed up.

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 Tower 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 /etc/ansible/hosts` 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.

```
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://s#win1/
```

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://s#win1/
```

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
```

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.

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.

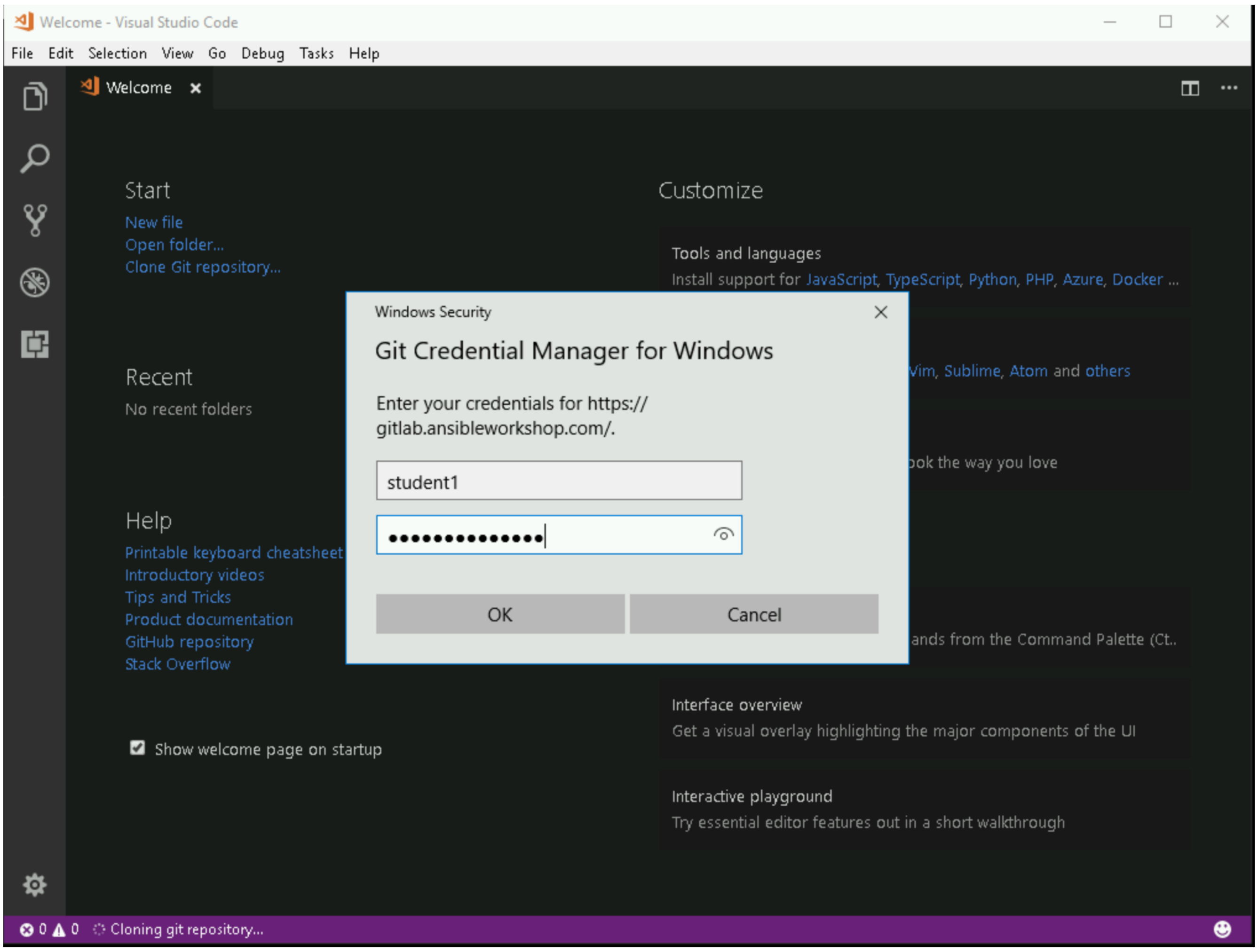
Internet Explorer 11 has some issues with Ansible Tower in the later labs. Due to this we will use Firefox in these examples.

In the main window, click 'Clone git repository'

TODO - Create a 1.1-vscode-gitclone.png!!! Missed this one! image::.png[title="Git Clone",width=1000]

At the top, enter https://gitlab.ansibleworkshop.com/student#/student#-playbooks, where # is your student number. When it prompts you for directory, leave the default (C:\Users\Administrator)

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#/…)


Welcome - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
Welcome
Start
New file
Open folder...
Clone Git repository...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP, Azure, Docker ...
Vim, Sublime, Atom and others
ook the way you love
ands from the Command Palette (Ct..
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try essential editor features out in a short walkthrough
0 0 Cloning git repository...

Windows Security

Git Credential Manager for Windows

Enter your credentials for https://gitlab.ansibleworkshop.com/.

student1

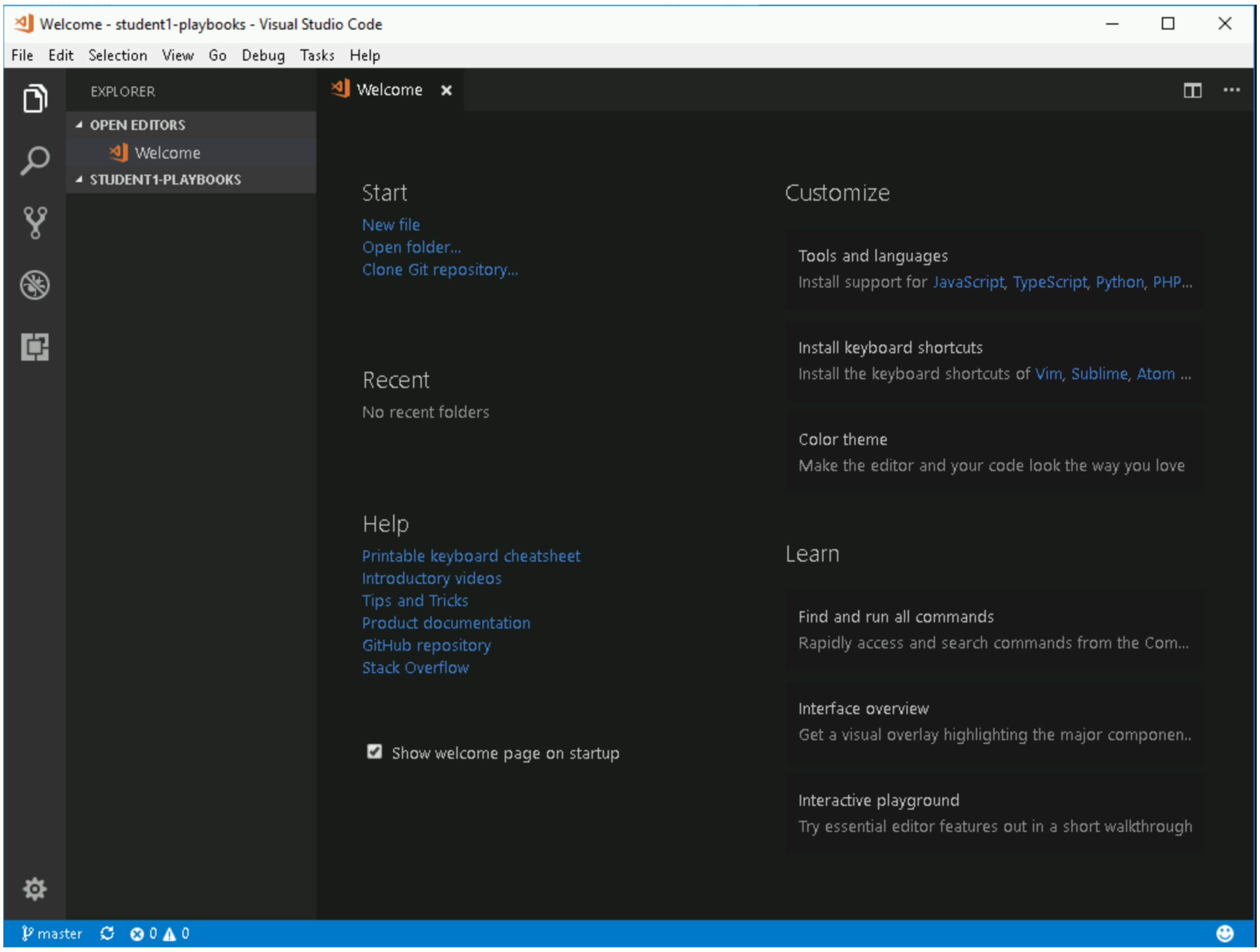
••••••••••••••

OK Cancel

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.


Welcome - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
Welcome
STUDENT1-PLAYBOOKS
Welcome
Start
New file
Open folder...
Clone Git repository...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP...
Install keyboard shortcuts
Install the keyboard shortcuts of Vim, Sublime, Atom ...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Com...
Interface overview
Get a visual overlay highlighting the major componen..
Interactive playground
Try essential editor features out in a short walkthrough
master 0 0

Student Playbooks Repo

Step 2: Create a directory and called **iis_basic** and a file called `install_iis.yml`

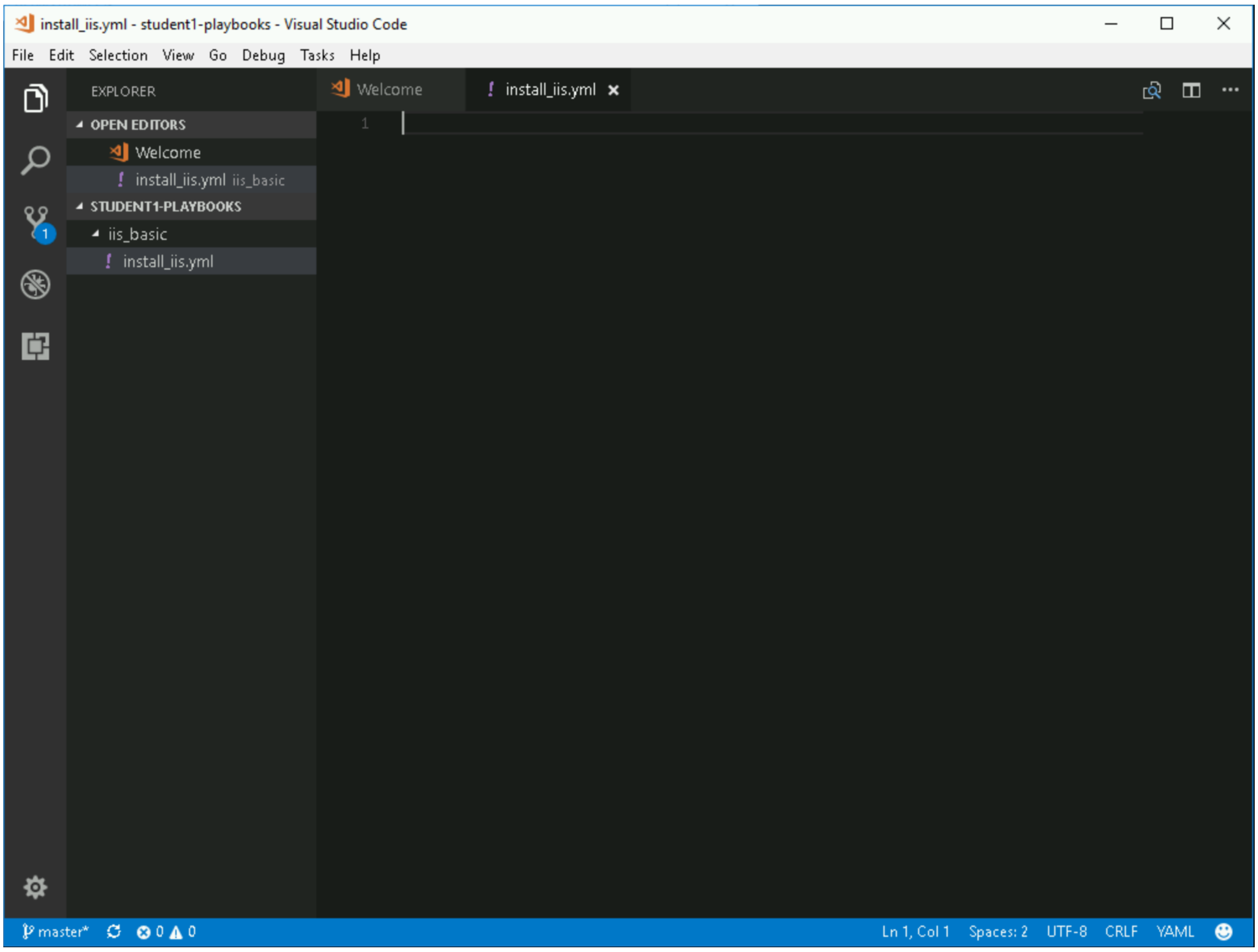
Hover over the 'student#-playbooks' section and click on the 'New Folder' button

Type iis_basic and 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 install_iis.yml and hit 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

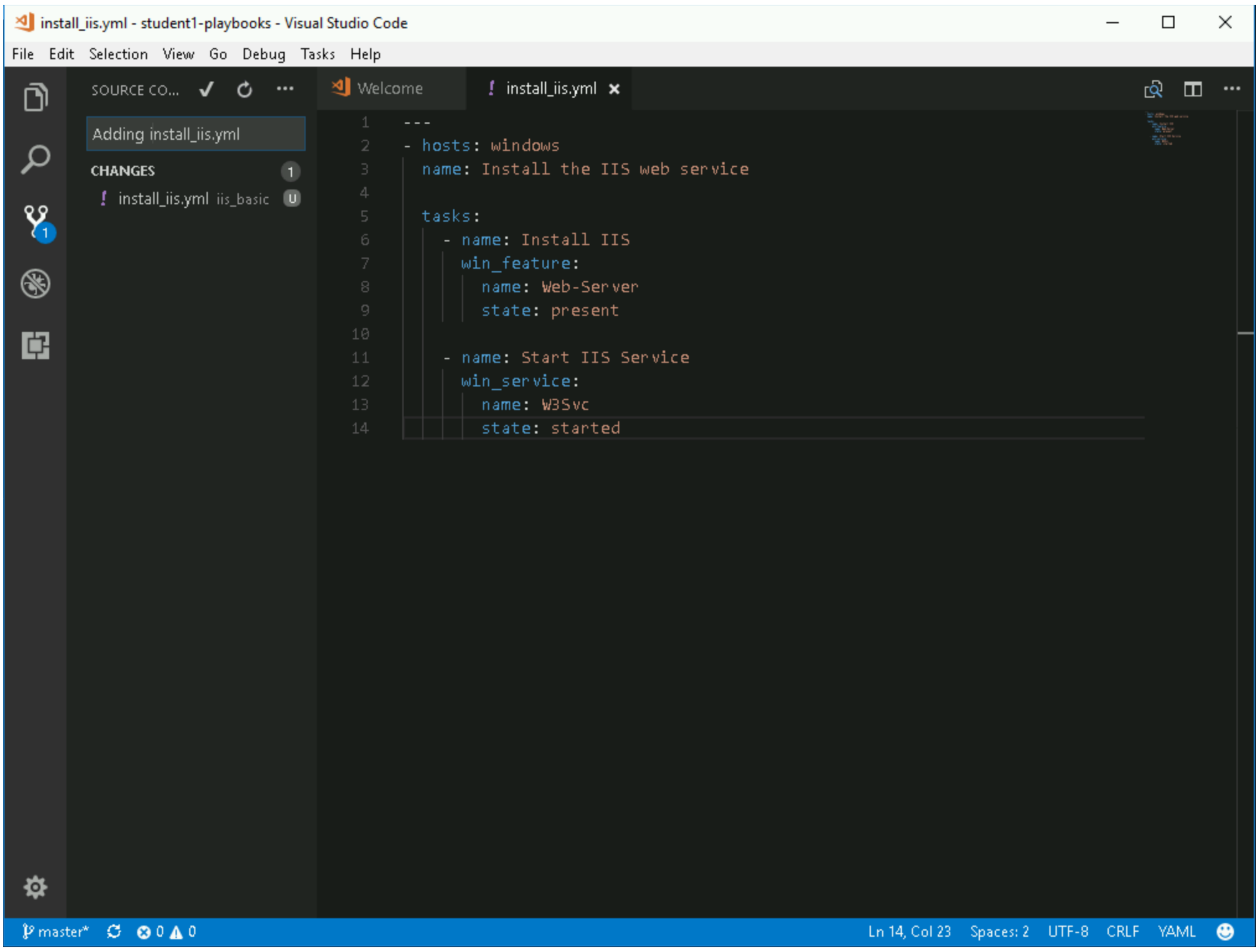
Now that you've completed writing your playbook, it would be a shame not to keep it.

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)

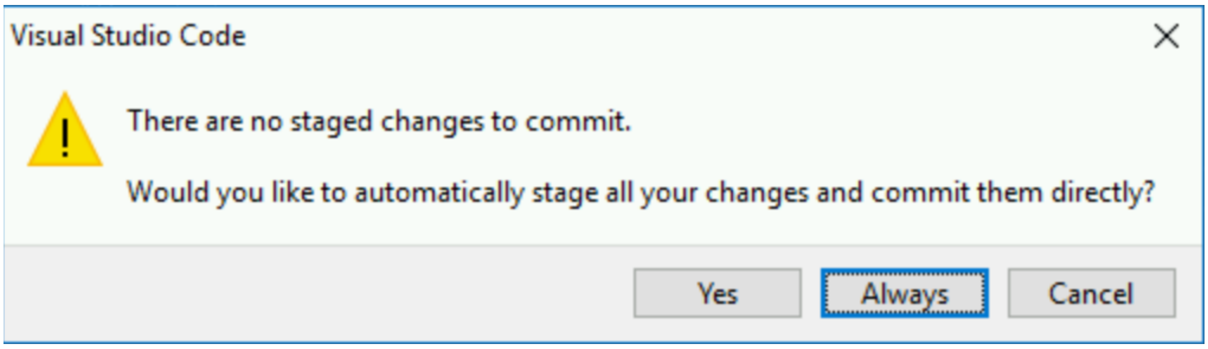
Type in a commit message such as 'Adding install_iis.yml' and click the check box above to commit.


install_iis.yml - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
SOURCE CO...
Adding install_iis.yml
CHANGES 1
install_iis.yml iis_basic U
Welcome
install_iis.yml
master*
0 0
Ln 14, Col 23 Spaces: 2 UTF-8 CRLF YAML

```
---
- 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
```

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.



Visual Studio Code

There are no staged changes to commit.

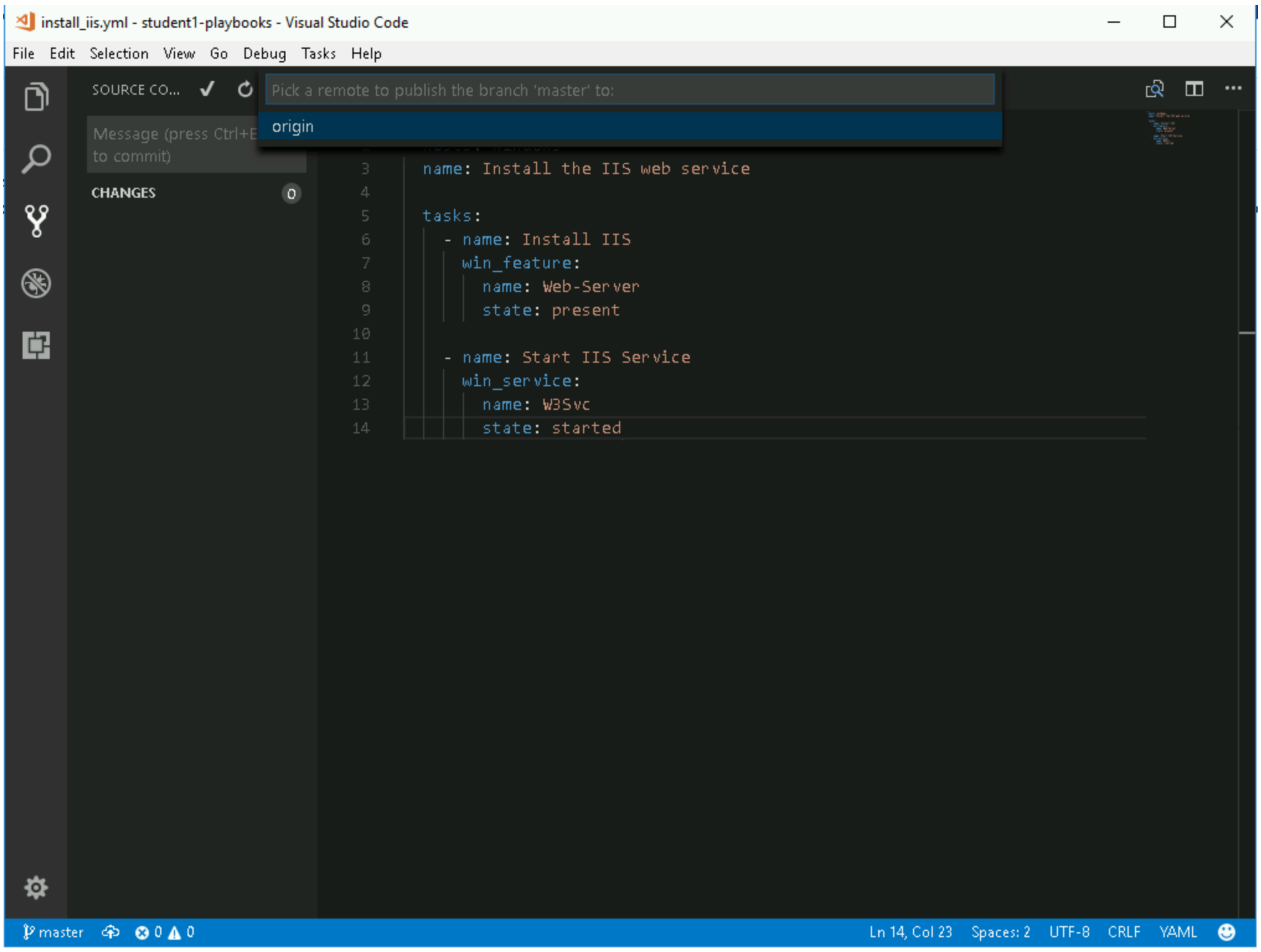
Would you like to automatically stage all your changes and commit them directly?

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`)



install_iis.yml - student1-playbooks - Visual Studio Code

Pick a remote to publish the branch 'master' to:

origin

```
  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
```

File Edit Selection View Go Debug Tasks Help
SOURCE CO...
Message (press Ctrl+E to commit)
CHANGES 0
master 0 0
Ln 14, Col 23 Spaces: 2 UTF-8 CRLF YAML

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.ansibleworkshop.com`. 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 s#tower 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.ansibleworkshop.com/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, lets take a few moments to understand some options.

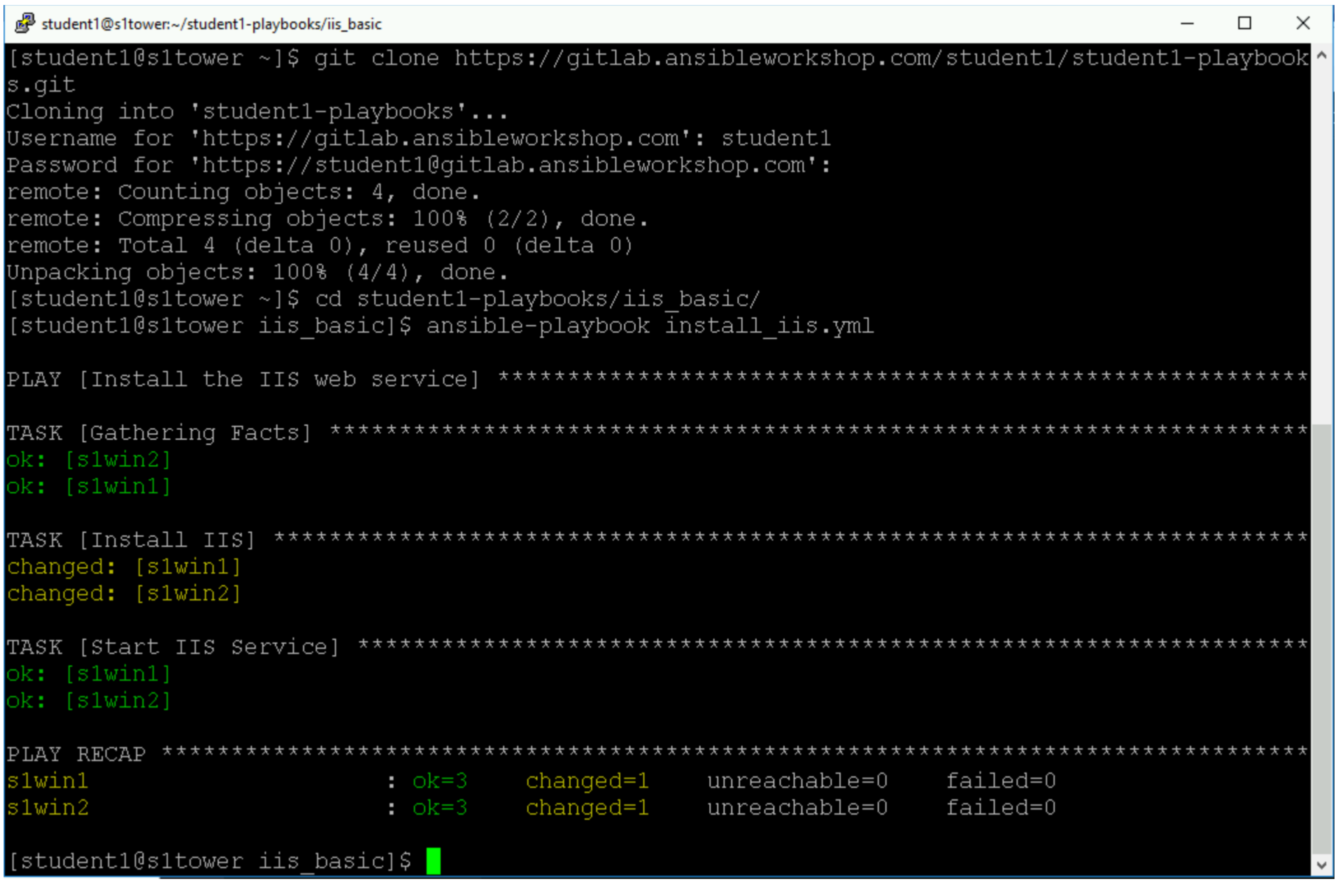
- **-i** This option could be used to specify the inventory file you wish to use. We are using the default (/etc/ansible/hosts)
- **-v** Altough not used here, this increases verbosity. Try running your playbook a second time using `-v` or `-vv` to increase the verbosity

  --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@s1tower:~/student1-playbooks/iis_basic

```
[student1@s1tower ~]$ 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@s1tower ~]$ cd student1-playbooks/iis_basic/
[student1@s1tower iis_basic]$ ansible-playbook install_iis.yml

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

TASK [Gathering Facts] *********************************************************************
ok: [s1win2]
ok: [s1win1]

TASK [Install IIS] *************************************************************************
changed: [s1win1]
changed: [s1win2]

TASK [Start IIS Service] *******************************************************************
ok: [s1win1]
ok: [s1win2]

PLAY RECAP *********************************************************************************
s1win1                     : ok=3    changed=1    unreachable=0    failed=0
s1win2                     : ok=3    changed=1    unreachable=0    failed=0

[student1@s1tower 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 if 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 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

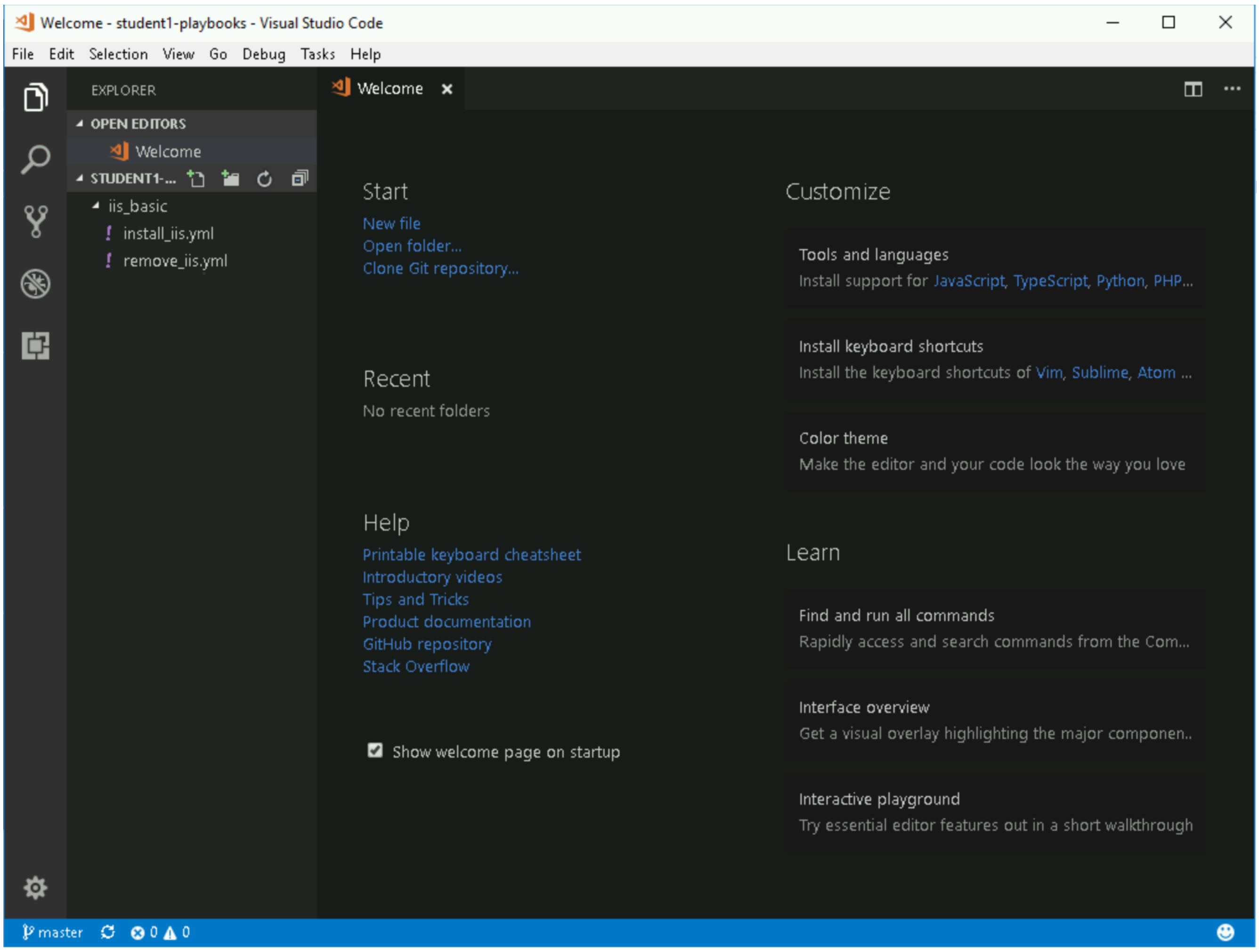
To begin, we are going to create a new playbook, but it should look very familiar to the one you created in exercise 1.2

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.


Welcome - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
Welcome
STUDENT1-...
iis_basic
install_iis.yml
remove_iis.yml
Welcome
Start
New file
Open folder...
Clone Git repository...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP...
Install keyboard shortcuts
Install the keyboard shortcuts of Vim, Sublime, Atom ...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Com...
Interface overview
Get a visual overlay highlighting the major componen..
Interactive playground
Try essential editor features out in a short walkthrough
master
0 0

Student Playbooks

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

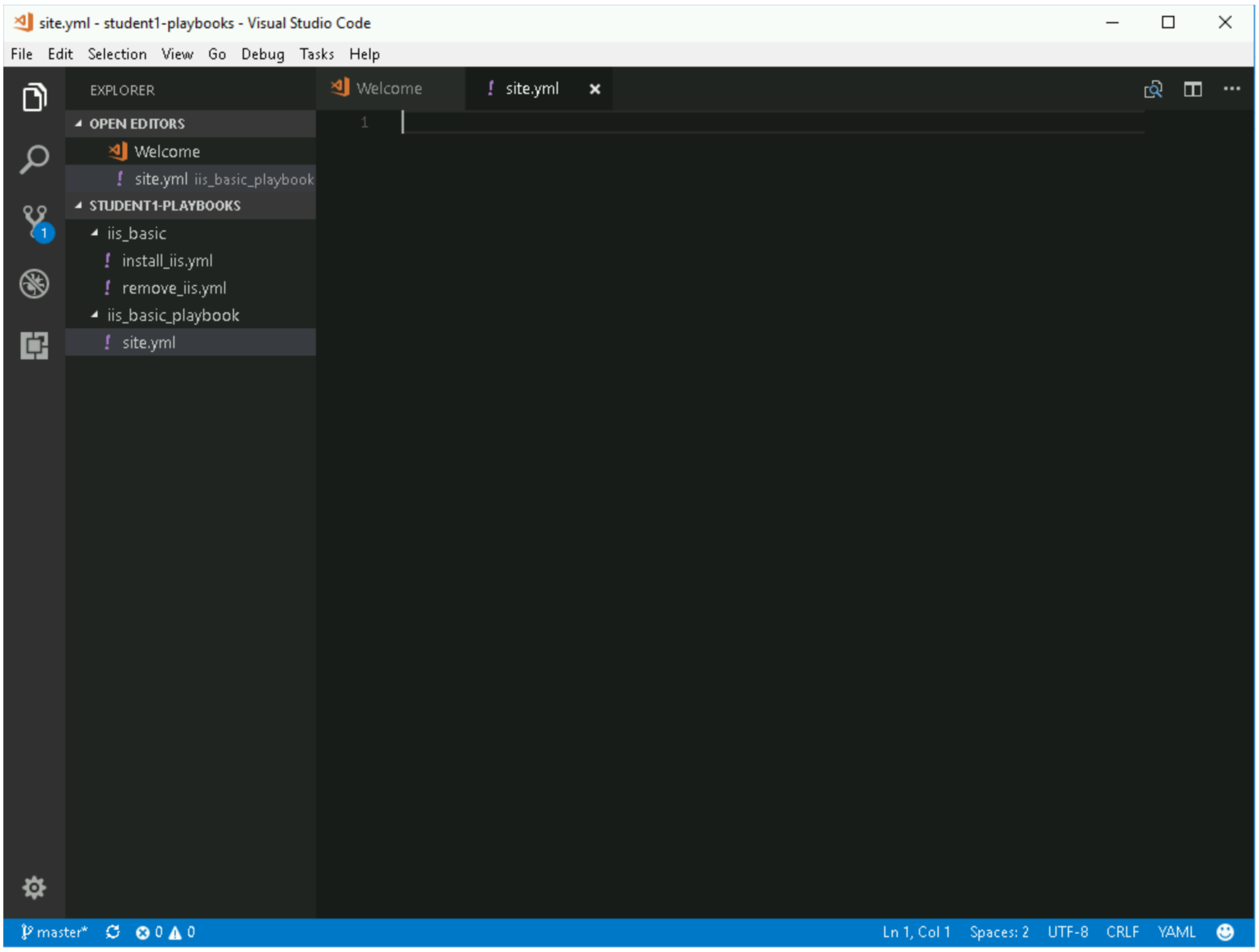
Hover over the 'student#-playbooks' section and click on the 'New Folder' button

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 hit enter.

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


site.yml - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
Welcome
site.yml iis_basic_playbook
STUDENT1-PLAYBOOKS
iis_basic
install_iis.yml
remove_iis.yml
iis_basic_playbook
site.yml
Welcome
site.yml
master*
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF YAML

Empty site.yml

Step 2:

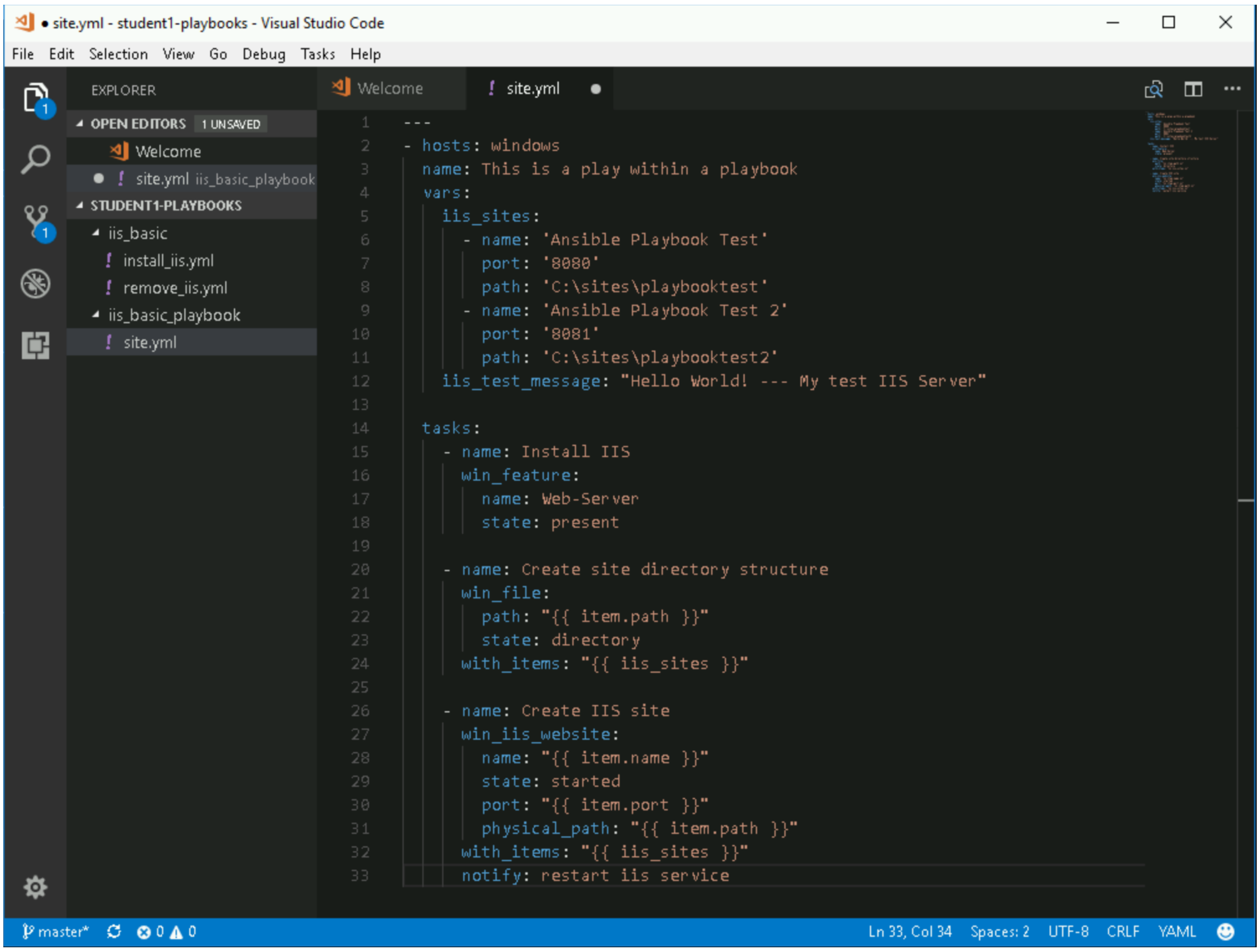
Add a play definition and some variables to your playbook. These include addtional packages your playbook will install on your web servers, plus some web server specific configurations.

```
---
- 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
```


● site.yml - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS 1 UNSAVED
Welcome
site.yml iis_basic_playbook
STUDENT1-PLAYBOOKS
iis_basic
install_iis.yml
remove_iis.yml
iis_basic_playbook
site.yml
Welcome
site.yml
master* 0 0
Ln 33, Col 34 Spaces: 2 UTF-8 CRLF YAML

```yaml
---
- 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
```

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 on the 'New Folder' button

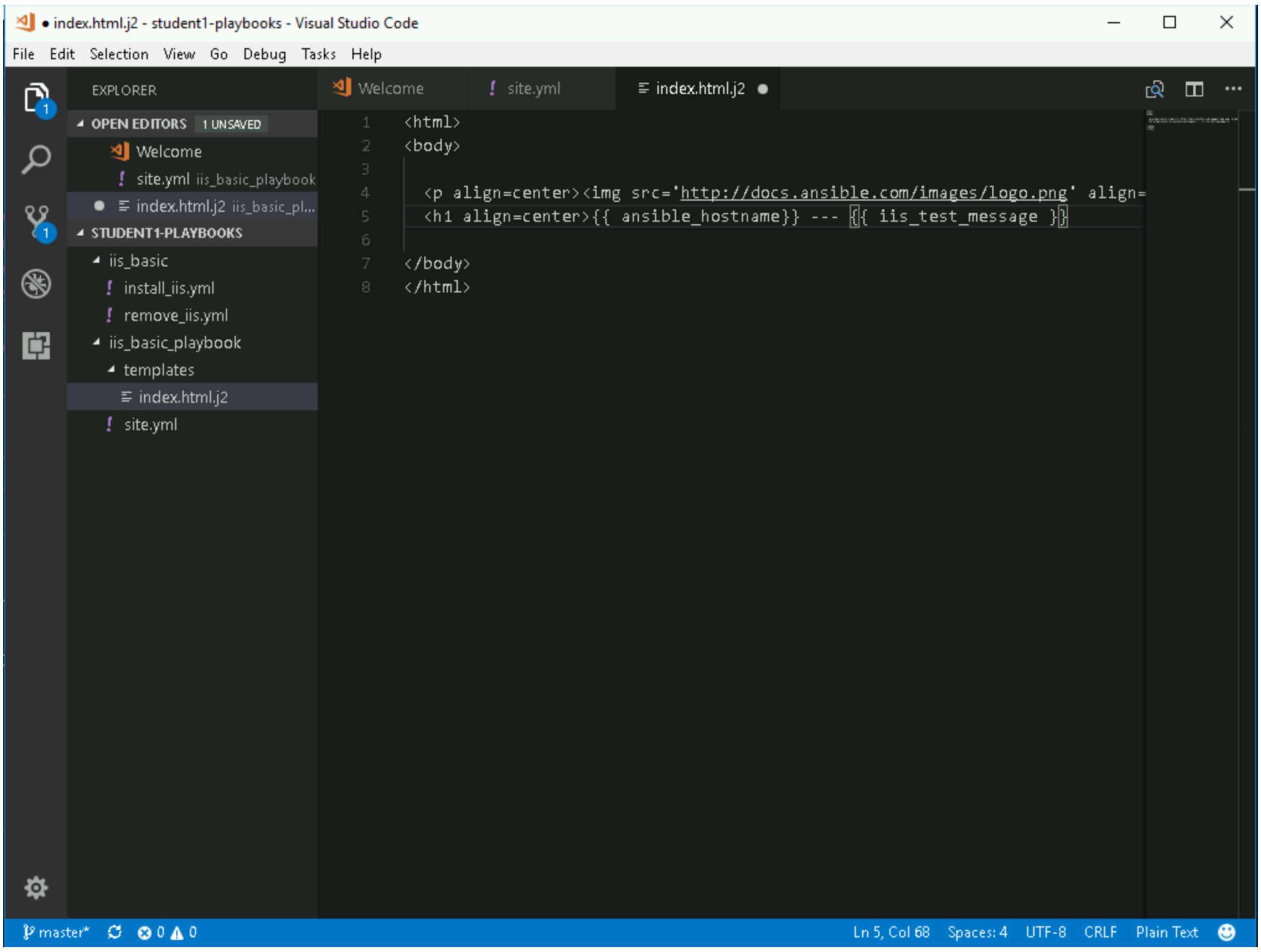
Type templates and hit enter. Then click on that folder so it is selected.

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

Type index.html.j2 and hit 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.j2 - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS 1 UNSAVED
Welcome
site.yml iis_basic_playbook
index.html.j2 iis_basic_pl...
STUDENT1-PLAYBOOKS
iis_basic
install_iis.yml
remove_iis.yml
iis_basic_playbook
templates
index.html.j2
site.yml
Welcome
site.yml
index.html.j2
master* 0 0
Ln 5, Col 68 Spaces: 4 UTF-8 CRLF Plain Text

```
<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

[1.3 vscode secondhalf site.yml] | *1.3-vscode-secondhalf-site.yml.png*

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.

[1.3 vscode commit site.yml] | *1.3-vscode-commit-site.yml.png*

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 line counts and 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 s#tower 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.ansibleworkshop.com/student#/student#-playbooks.git
cd iis_basic_playbook
```

Since you already have an inventory file, /etc/ansible/hosts, we will re-use it.

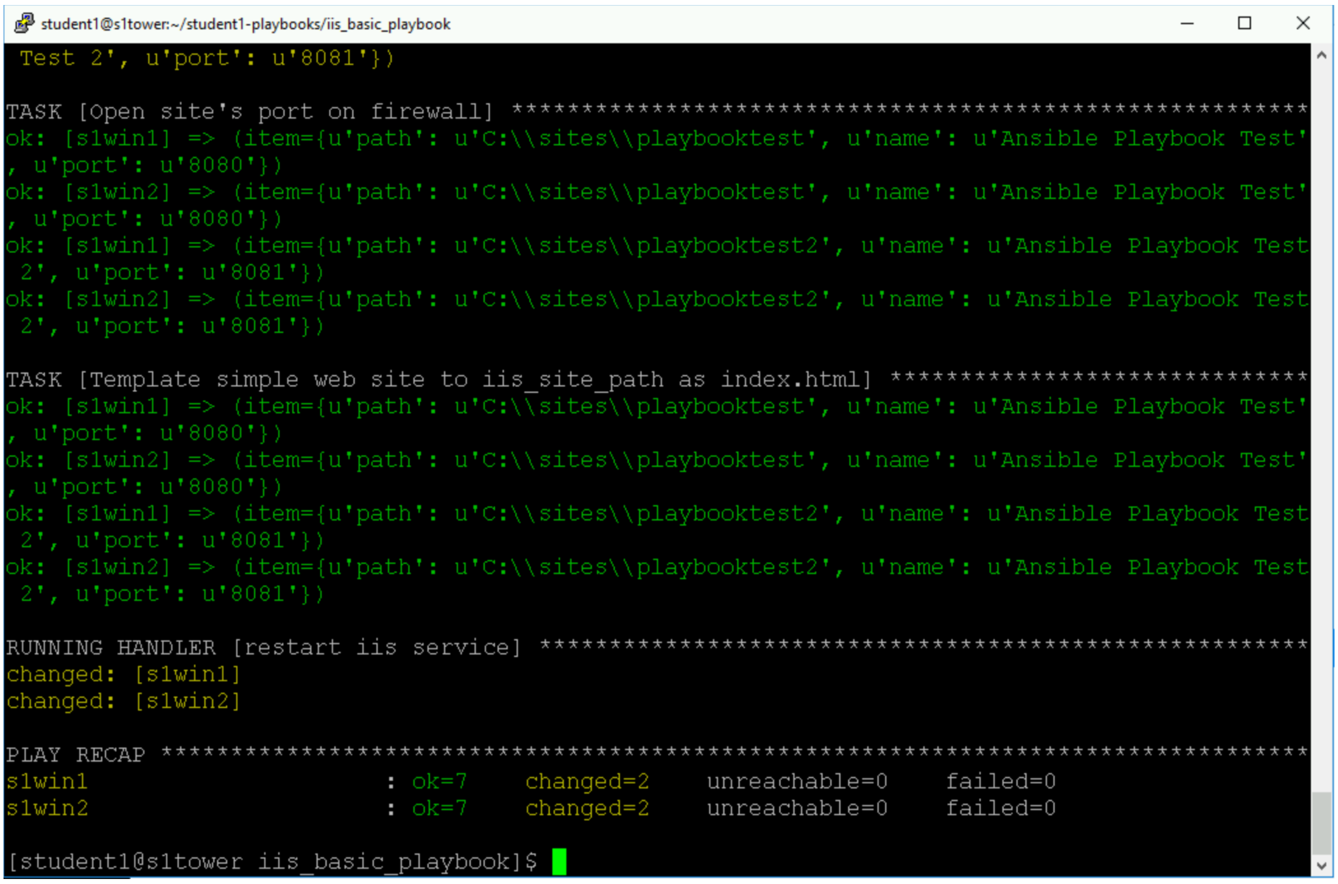
Step 2:

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.



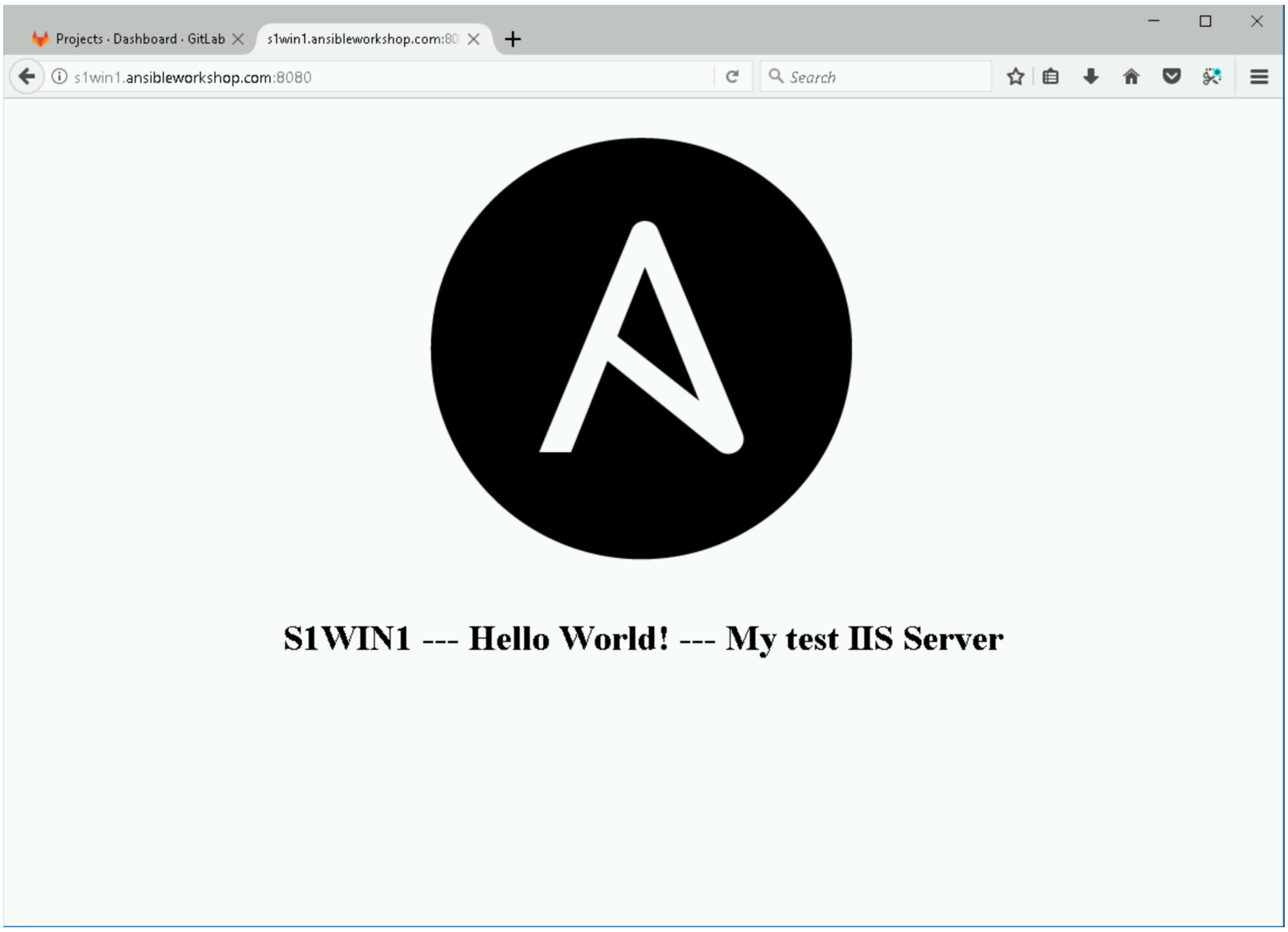
```
student1@s1tower:~/student1-playbooks/iis_basic_playbook
 Test 2', u'port': u'8081'})

TASK [Open site's port on firewall] ************************************************************
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\\playbooktest', u'name': u'Ansible Playbook Test'
, 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 [Template simple web site to iis_site_path as index.html] ******************************
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\\playbooktest', u'name': u'Ansible Playbook Test'
, 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'})

RUNNING HANDLER [restart iis service] **********************************************************
changed: [s1win1]
changed: [s1win2]

PLAY RECAP ****************************************************************************************
s1win1                     : ok=7    changed=2    unreachable=0    failed=0
s1win2                     : ok=7    changed=2    unreachable=0    failed=0

[student1@s1tower 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 (s#win1 or s#win2):

- http://s#win1.ansibleworkshop.com:8080/
- http://s#win1.ansibleworkshop.com:8081/


Projects · Dashboard · GitLab
s1win1.ansibleworkshop.com:80
s1win1.ansibleworkshop.com:8080
Search

S1WIN1 --- Hello World! --- My test IIS Server

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.

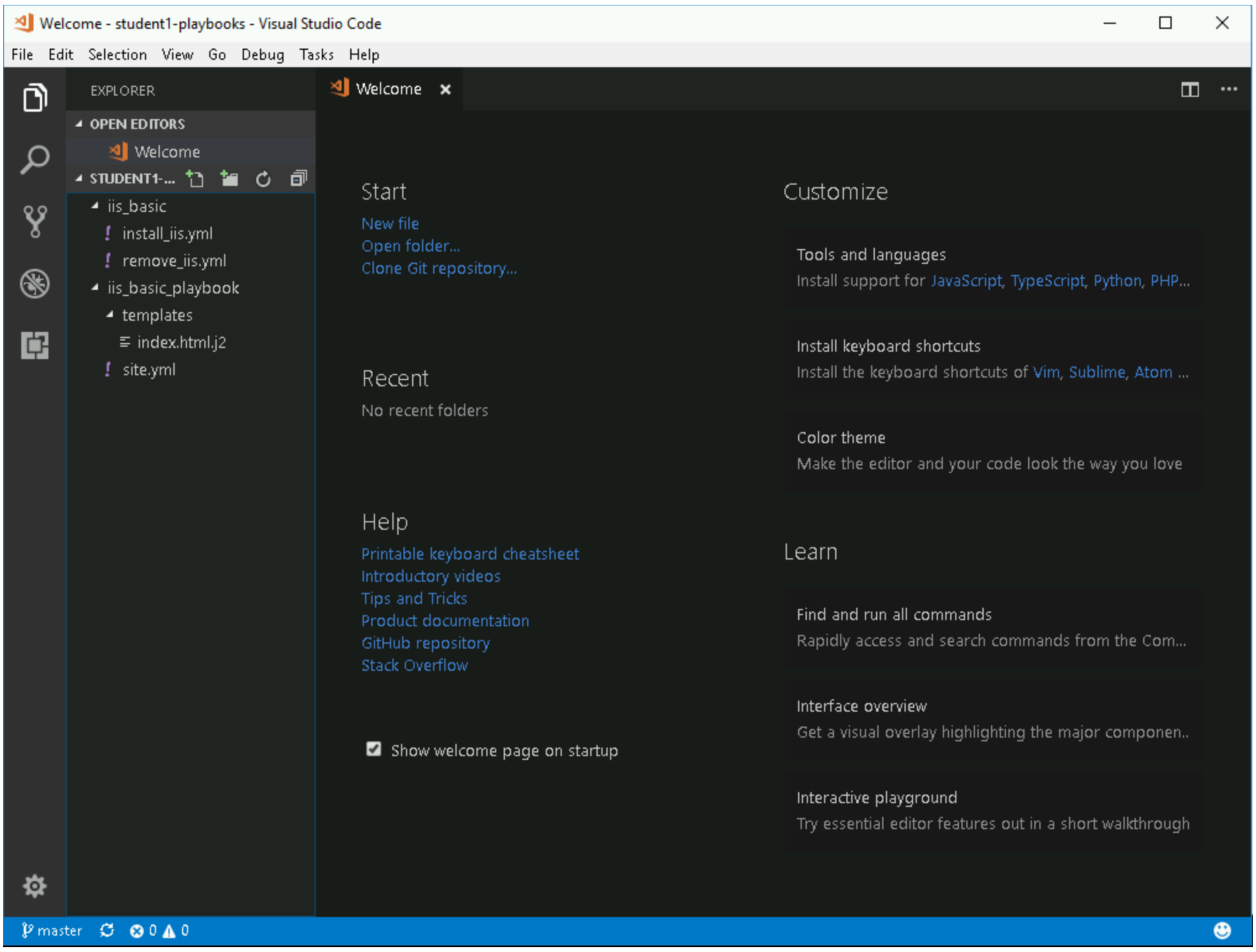
For this exercise, you are going to take the playbook you just wrote and refactor it into a role.

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.


Welcome - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
Welcome
STUDENT1-...
iis_basic
install_iis.yml
remove_iis.yml
iis_basic_playbook
templates
index.html.j2
site.yml
Start
New file
Open folder...
Clone Git repository...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP...
Install keyboard shortcuts
Install the keyboard shortcuts of Vim, Sublime, Atom ...
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Com...
Interface overview
Get a visual overlay highlighting the major componen..
Interactive playground
Try essential editor features out in a short walkthrough
master 0 0

iis_basic_playbook

Select the **iis_basic_playbook** folder.

Create a directory called **roles** by right-clicking on iis_basic_playbook and selecting 'New Folder'

Now right-click on **roles** and create a new folder underneath called `iis_simple`.

Step 2:

Within 'iis_simple' create new folders as follows:

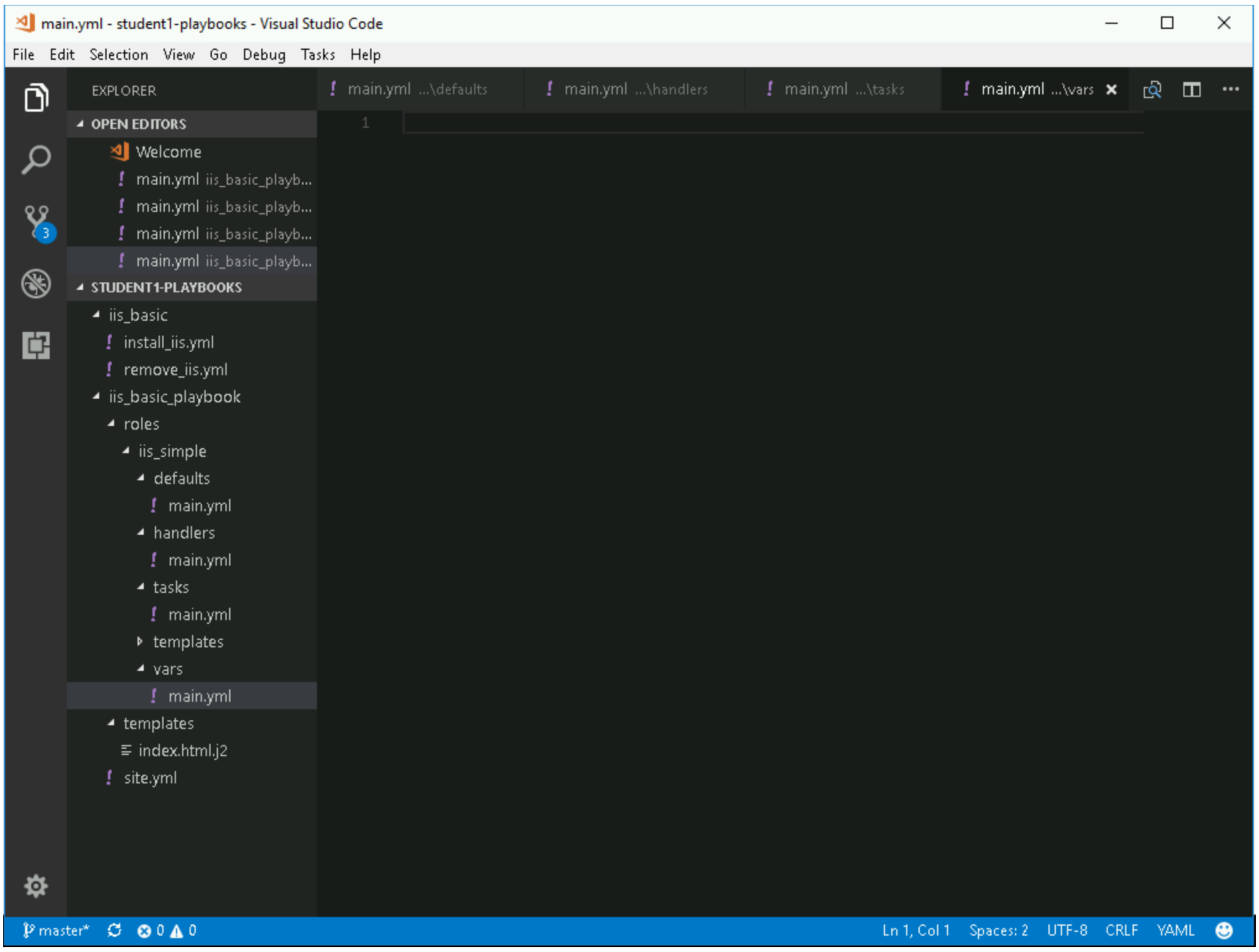
- defaults
- vars

- 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:



```
OPEN EDITORS
  Welcome
  main.yml iis_basic_playb...
  main.yml iis_basic_playb...
  main.yml iis_basic_playb...
  main.yml iis_basic_playb...
STUDENT1-PLAYBOOKS
  iis_basic
    install_iis.yml
    remove_iis.yml
  iis_basic_playbook
    roles
      iis_simple
        defaults
          main.yml
        handlers
          main.yml
        tasks
          main.yml
        templates
        vars
          main.yml
    templates
      index.html.j2
    site.yml
```

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`.

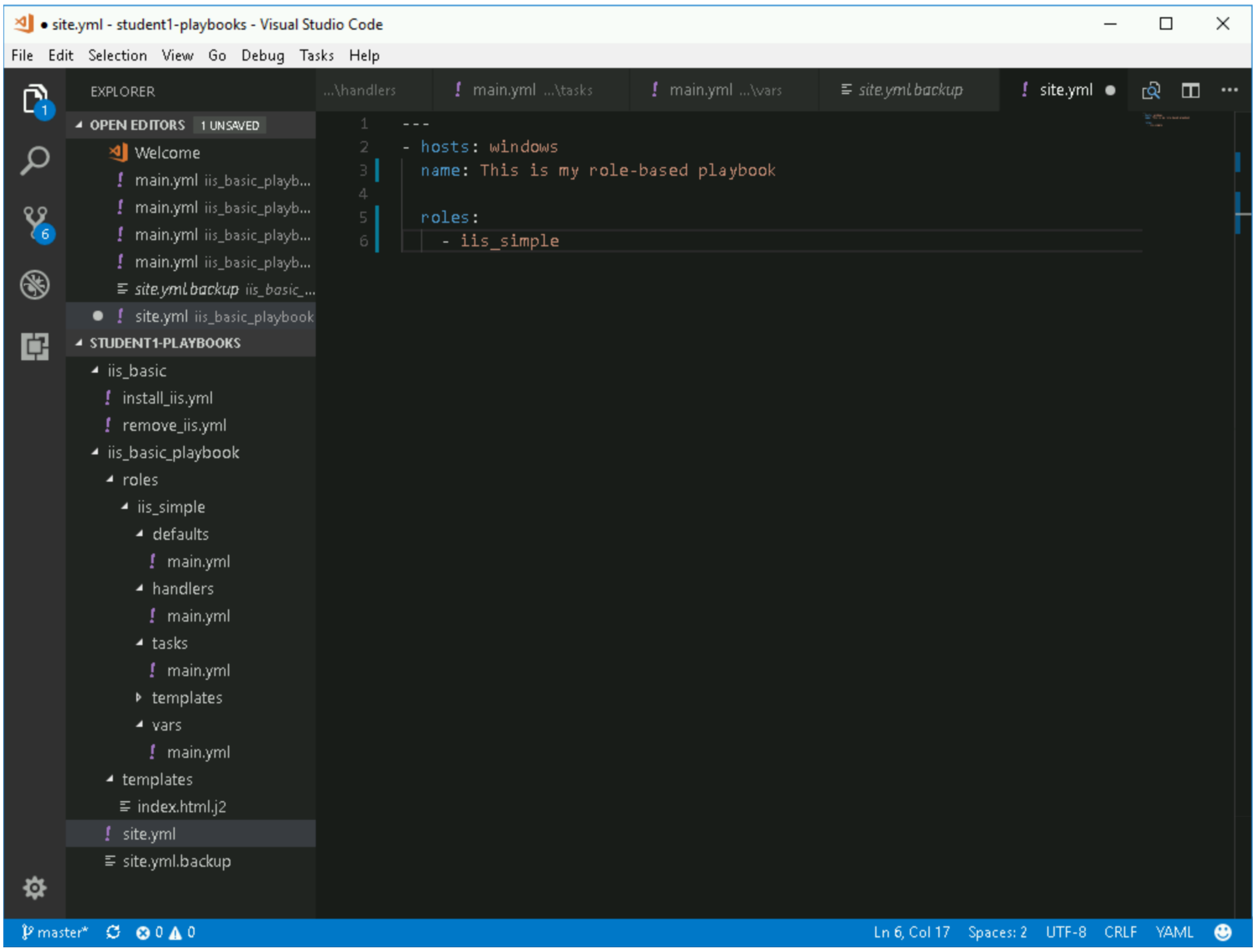
Navigate to your `iis_basic_playbook` folder, right click on `site.yml`, click `rename`, and call it `site.yml.backup`

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
```


● site.yml - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS 1 UNSAVED
Welcome
main.yml iis_basic_playb...
main.yml iis_basic_playb...
main.yml iis_basic_playb...
main.yml iis_basic_playb...
site.yml.backup iis_basic_...
site.yml iis_basic_playbook
STUDENT1-PLAYBOOKS
iis_basic
install_iis.yml
remove_iis.yml
iis_basic_playbook
roles
iis_simple
defaults
main.yml
handlers
main.yml
tasks
main.yml
templates
vars
main.yml
templates
index.html.j2
site.yml
site.yml.backup
master* 0 0
Ln 6, Col 17 Spaces: 2 UTF-8 CRLF YAML

```
---
- 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 on `roles\iis_simple\templates` and create a new file called `index.html.j2` with the following content:

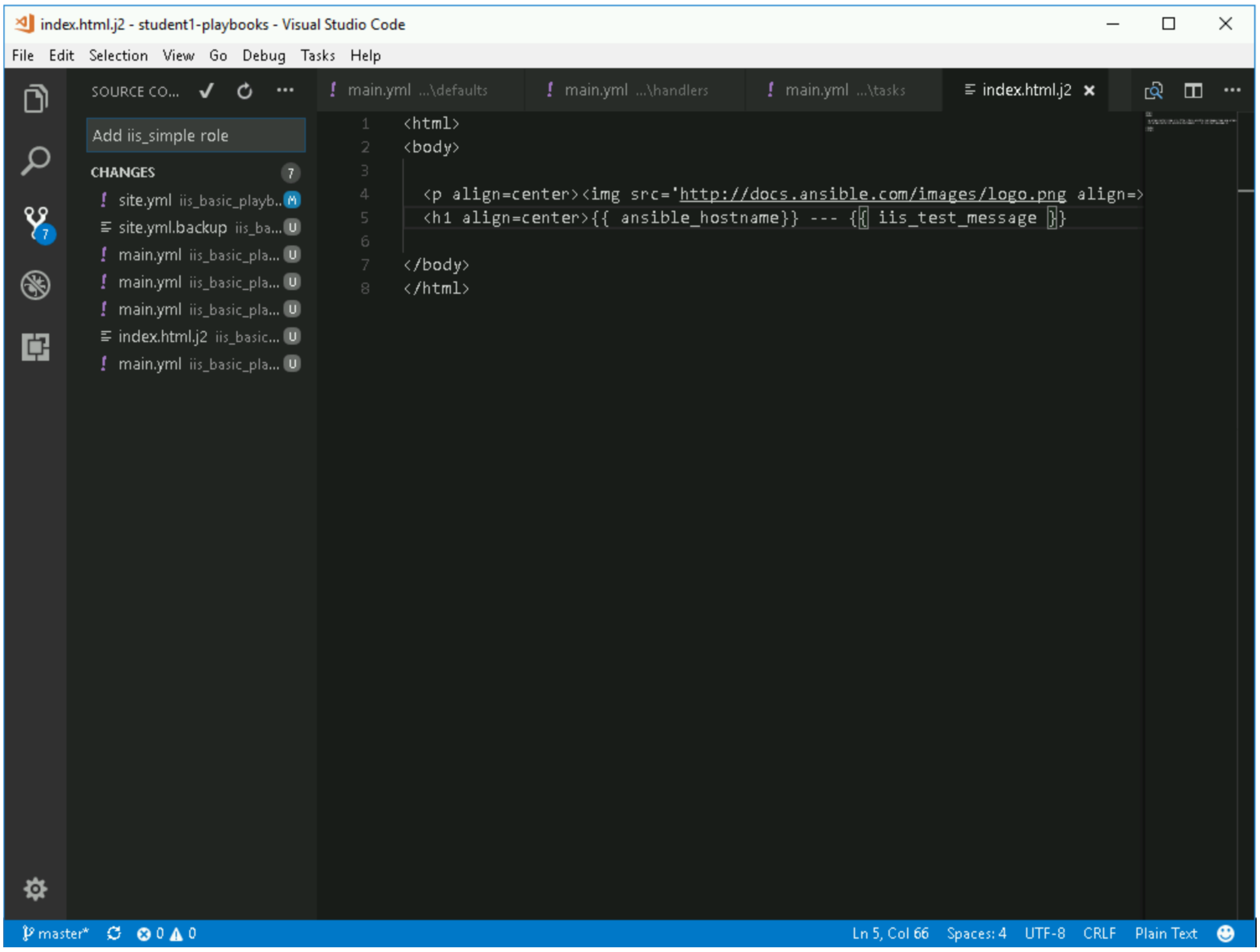
```
<html>
<body>

  <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 on File → Save All to ensure all your files are saved.

Click on the Source Code icon as shown below.

Type in a commit message like `Add iis_simple role` and click the check box above.


index.html.j2 - student1-playbooks - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
SOURCE CO...
main.yml ...\defaults
main.yml ...\handlers
main.yml ...\tasks
index.html.j2
CHANGES 7
site.yml iis_basic_playb.. M
site.yml.backup iis_ba... U
main.yml iis_basic_pla... U
main.yml iis_basic_pla... U
main.yml iis_basic_pla... U
index.html.j2 iis_basic... U
main.yml iis_basic_pla... U
master* 0 0
Ln 5, Col 66 Spaces: 4 UTF-8 CRLF Plain Text

Add iis_simple role

```
<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>
```

Commit iis_simple_role

Click on 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

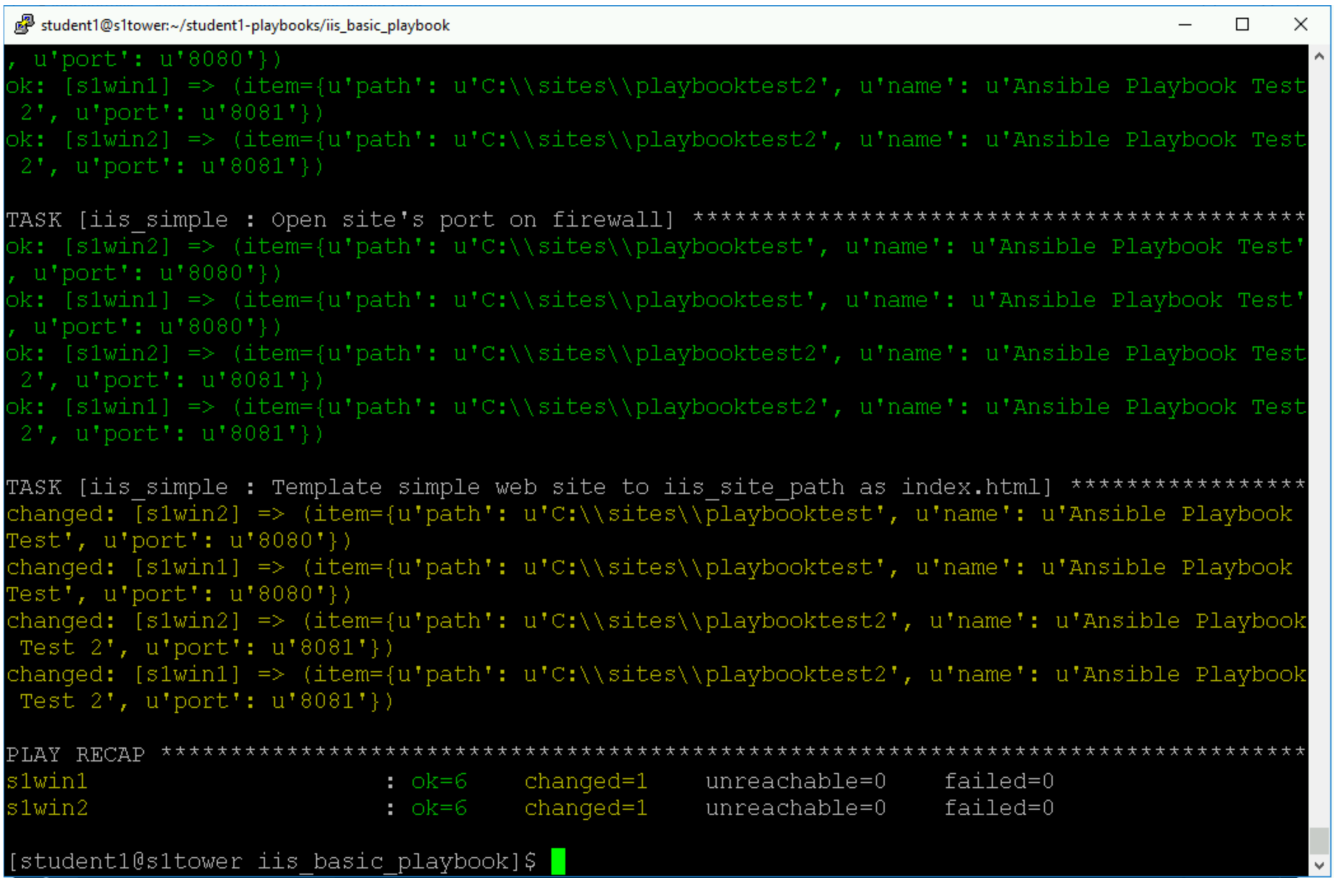
Now that you've successfully separated your original playbook into a role, let's run it and see how it works.

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@s1tower:~/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@s1tower 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.

Exercise 2.0 - Installing Ansible Tower

In this exercise, we are going to get Ansible Tower installed on your control node

Installing Ansible Tower

Step 1:

Use putty to SSH to your Tower node. Change directories to /tmp

```
cd /tmp
```

Step 2:

Download the latest Ansible Tower package

```
curl -O http://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-
latest.tar.gz
```

Step 3:

Untar and unzip the package file

```
tar xvfz /tmp/ansible-tower-setup-latest.tar.gz
```

Step 4:

Change directories into the ansible tower package

```
cd /tmp/ansible-tower-setup-*
```

Step 5:

Using an editor of your choice, open the inventory file

```
vi inventory
```

Step 6:

Fill a few variables out in an inventory file: `admin_password, pg_password, rabbitmq_password`

A few hints for `vi` if it's not your thing.

- Move around the file with your arrow keys

- a enters into add mode
- esc exits add mode
- x when not in add mode will delete a character
- esc followed by :wq! will save the file
    - Just doing :q! will exit without saving

```
[tower]
localhost ansible_connection=local

[database]

[all:vars]
admin_password=\'ansibleWS\'

pg_host=\''
pg_port=\''

pg_database=\'awx'
pg_username=\'awx'
pg_password=\'ansibleWS\'

rabbitmq_port=5672
rabbitmq_vhost=tower
rabbitmq_username=tower
rabbitmq_password=\'ansibleWS'
rabbitmq_cookie=cookiemonster

# Needs to be true for fqdns and ip addresses
rabbitmq_use_long_name=false
```

Step 7:

Run the Ansible Tower setup script

```
sudo ./setup.sh
```

Step 7 will take approx. 10-15 minutes to complete. This uses ansible to install Tower. This may be a good time to take a break.

Step 8:

As our git has a self signed cert, we have one more configuration step. We need to enable tower to not verify SSL for repo syncs. To do this, execute the following commands:

```
sudo su -
echo "AWX_TASK_ENV['GIT_SSL_NO_VERIFY'] = 'True'" >> /etc/tower/settings.py
ansible-tower-service restart
exit
```

End Result

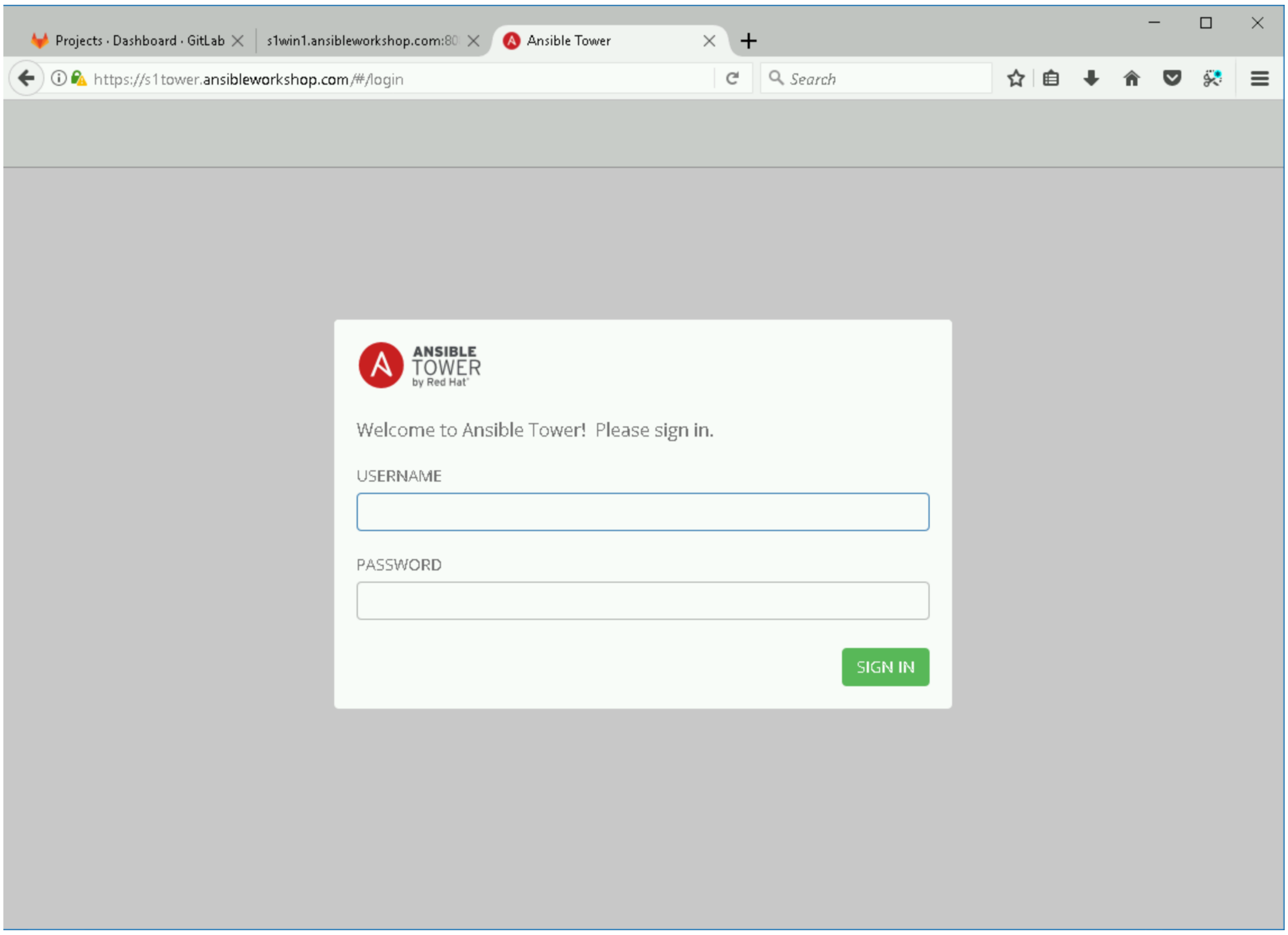
At this point, your Ansible Tower installation should be complete. You can access Tower from the browser on your s#workstation host at:

```
https://s#tower.ansibleworkshop.com
```

As we're using a self-signed cert you will have to ignore the security warning and `Add` this site as trusted.

Ensuring Installation Success

You know you were successful if you are able to browse to your Ansible Tower's url and get something like this


Projects · Dashboard · GitLab
s1win1.ansibleworkshop.com:80
Ansible Tower
https://s1tower.ansibleworkshop.com/#/login
Search
ANSIBLE TOWER by Red Hat
Welcome to Ansible Tower! Please sign in.
USERNAME
PASSWORD
SIGN IN

Ansible Tower Login Screen

Exercise 2.1 - Configuring Ansible Tower

In this exercise, we are going to configure Tower so that we can run a playbook.

Configuring Ansible Tower

There are a number of contructs in the Ansible Tower UI that enable multi-tenancy, notifications, scheduling, etc. However, we are only going to focus on a few of the key contructs that are required for this workshop today.

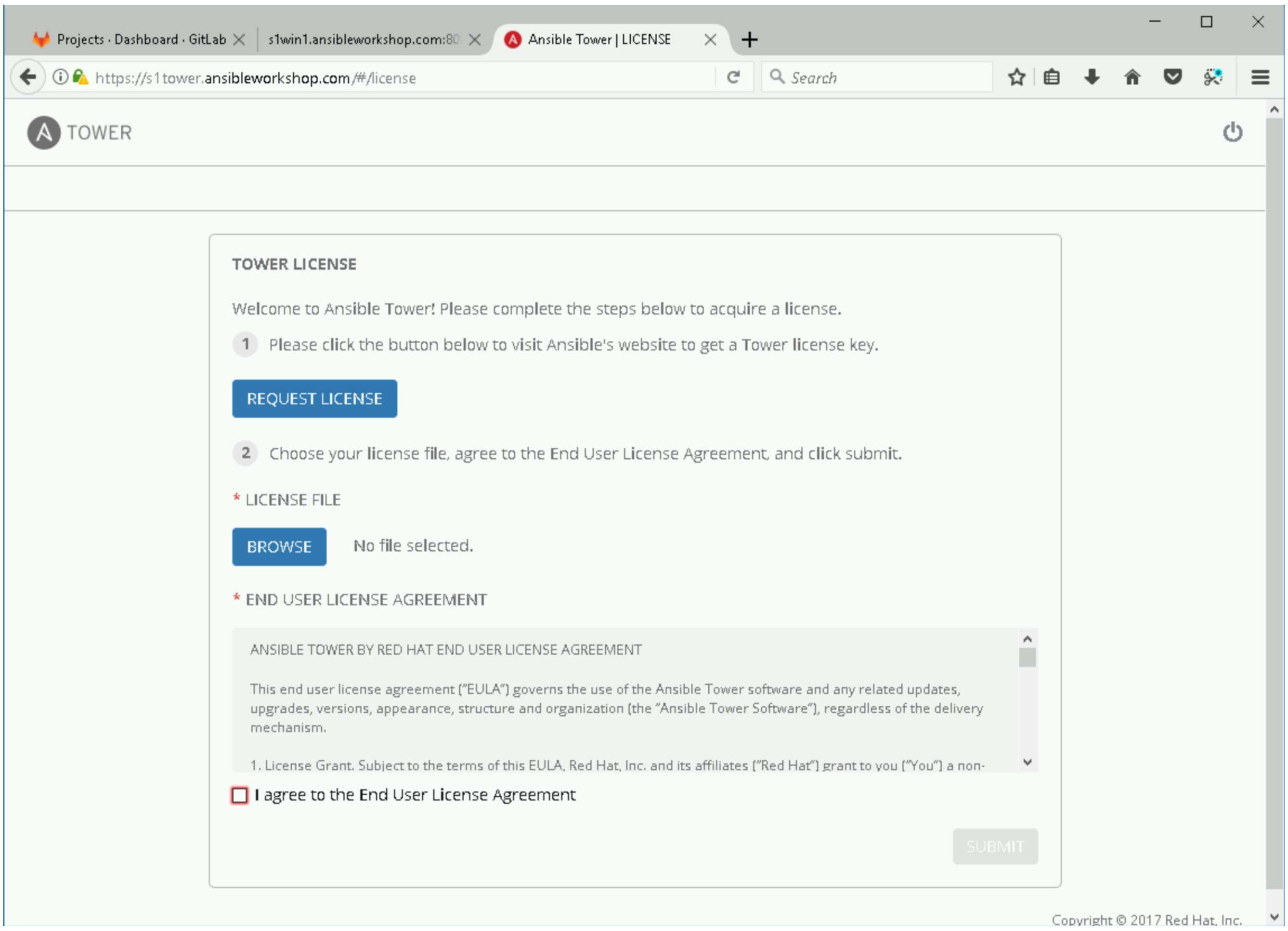
- Credentials
- Projects
- Inventory
- Job Template

Logging into Tower and Installing the License Key

Step 1:

To log in, use the username `admin` and the password `ansibleWS`. Note that typically AD/LDAP authentication would be setup. However, that is beyond the scope of this workshop.

As soon as you login, you will prompted to request a license or browse for an existing license file



TOWER LICENSE

Welcome to Ansible Tower! Please complete the steps below to acquire a license.

1 Please click the button below to visit Ansible's website to get a Tower license key.

REQUEST LICENSE

2 Choose your license file, agree to the End User License Agreement, and click submit.

* LICENSE FILE

BROWSE No file selected.

* END USER LICENSE AGREEMENT

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

This end user license agreement ("EULA") governs the use of the Ansible Tower software and any related updates, upgrades, versions, appearance, structure and organization (the "Ansible Tower Software"), regardless of the delivery mechanism.

1. License Grant. Subject to the terms of this EULA, Red Hat, Inc. and its affiliates ("Red Hat") grant to you ("You") a non-

I agree to the End User License Agreement

SUBMIT

Copyright © 2017 Red Hat, Inc.

Projects · Dashboard · GitLab
s1win1.ansibleworkshop.com:80
Ansible Tower | LICENSE
https://s1tower.ansibleworkshop.com/#/license
Search
TOWER

Uploading a License

Step 2:

At the commandline in your Tower instance download the encrypted license file via the curl command.

```
curl -O https://s3.amazonaws.com/ansible-tower-workshop-license/license
```

Then Decrypt the license file via Ansible Vault. **The instructor should provide the password**

```
ansible-vault decrypt license

...

Vault password:
```

Now use curl to POST the license to the Tower API endpoint. Please edit this command to the use the `workshop_prefix` for your workshop. This command exports your workshop prefix so that the next command can use that data in the curl command.

```
export WORKSHOP_PREFIX=" "
```

Now send the `license` to Tower via the API. Make certain to use your student number in the URL.

```
curl -k https://s#tower.ansibleworkshop.com/api/v1/config/ \
     -H 'Content-Type: application/json' \
     -X POST \
     --data @license \
     --user admin:ansibleWS
```

Creating a Machine Credential

Credentials are utilized by Tower for authentication when launching jobs against machines, synchronizing with inventory sources, and importing project content from a version control system.

There are many types of credentials including machine, network, and various cloud providers. In this workshop, we are using a **machine** credential.

Step 1:

Select the gear icon ⚙

Step 2:

Select CREDENTIALS

Step 3:

Click on ADD + ADD

Step 4:

Complete the form using the following entries:

NAME	Git Credential
DESCRIPTION	SCM credential for playbook sync
ORGANIZATION	Default
TYPE	Source Control
USERNAME	student#
PASSWORD	<your AD account password - instructor provided>

Notice here we've made a change from our previous examples. Previously we were using basic authentication with a local `Adminstrator` account. Now we are switching to an AD user and Kerberos authentication. We will also update our inventory variables to reflect Kerberos.

[2.1 tower add machine credential] | *2.1-tower-add-machine-credential.png*

Add Machine Credential

Step 5:

Select SAVE SAVE

Create an SCM Credential

Our first credential was to access our Windows machines. We need another to access our source code repository. Repeat the process as above, but with the following details:

NAME	Ansible Workshop Credential
DESCRIPTION	Machine credential for run job templates during workshop
ORGANIZATION	Default
TYPE	Machine
USERNAME	student#
PASSWORD	<your AD account password - instructor provided>

Make sure you select SAVE!

[2.1 tower add scm credential] | *2.1-tower-add-scm-credential.png*

Add SCM Credential

Creating a Project

A Project is a logical collection of Ansible playbooks, represented in Tower. You can manage playbooks and playbook directories by either placing them manually under the Project Base Path on your Tower server, or by placing your playbooks into a source code management (SCM) system supported by Tower, including Git, Subversion, and Mercurial.

Step 1:

Click on PROJECTS at the upper left

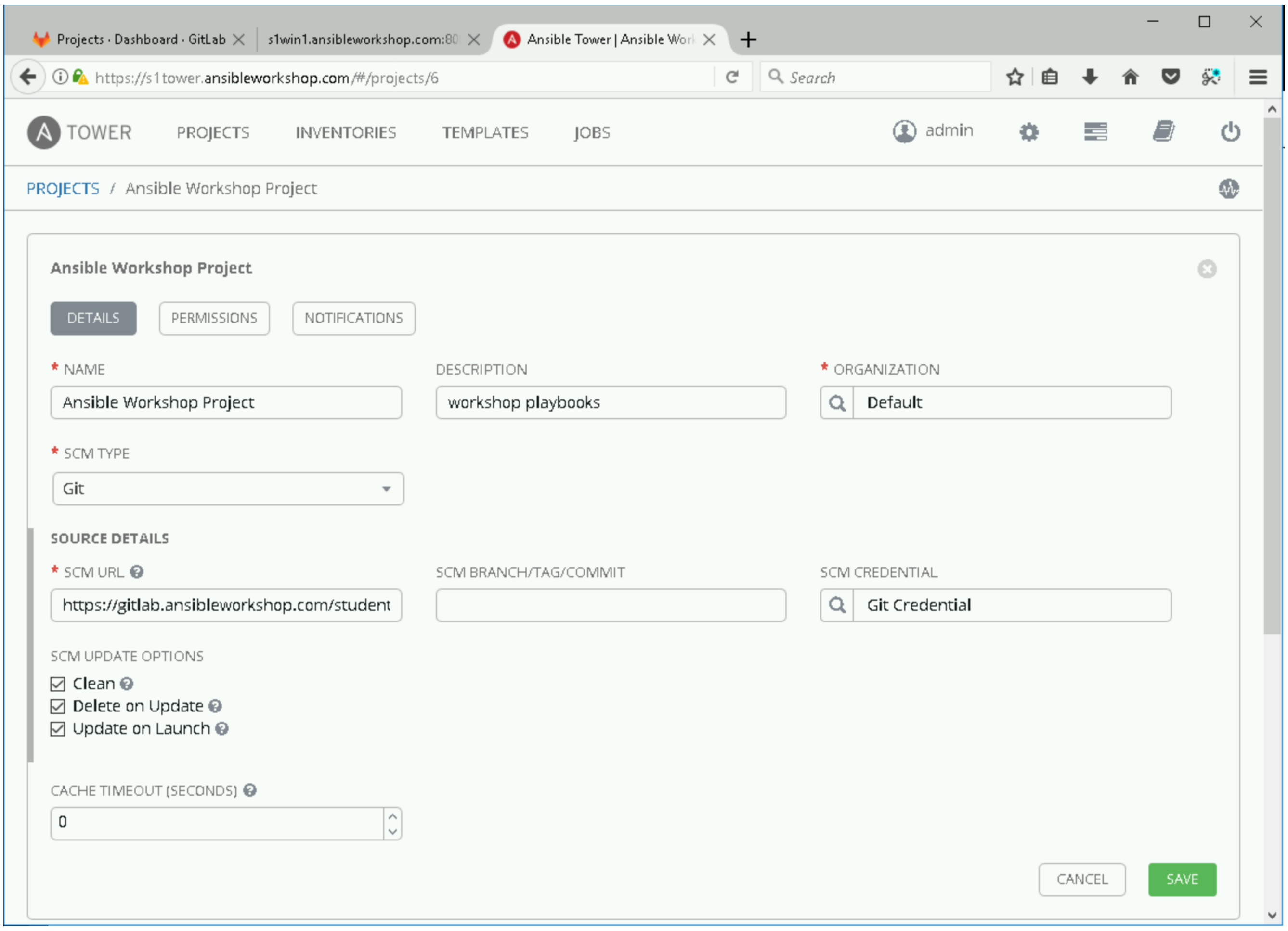
Step 2:

Select ADD + ADD

Step 3:

Complete the form using the following entries (using your student number

NAME	Ansible Workshop Project
DESCRIPTION	workshop playbooks
ORGANIZATION	Default
SCM TYPE	Git
SCM URL	https://gitlab.ansibleworkshop.com/student#/student#-playbooks.git
SCM BRANCH	
SCM CREDENTIAL	Git Credential
SCM UPDATE OPTIONS	- **[] Clean - []** Delete on Update - [*] Update on Launch


Projects · Dashboard · GitLab
s1win1.ansibleworkshop.com:80
Ansible Tower | Ansible Wor
https://s1tower.ansibleworkshop.com/#/projects/6
Search
TOWER
PROJECTS
INVENTORIES
TEMPLATES
JOBS
admin
PROJECTS / Ansible Workshop Project
DETAILS
PERMISSIONS
NOTIFICATIONS
CANCEL
SAVE

Ansible Workshop Project

* NAME	DESCRIPTION	* ORGANIZATION
Ansible Workshop Project	workshop playbooks	Default
* SCM TYPE		
Git		

SOURCE DETAILS

* SCM URL	SCM BRANCH/TAG/COMMIT	SCM CREDENTIAL
https://gitlab.ansibleworkshop.com/student		Git Credential

SCM UPDATE OPTIONS
- ☑ Clean
- ☑ Delete on Update
- ☑ Update on Launch

CACHE TIMEOUT (SECONDS)
0

Defining a Project

Step 4:

Select SAVE SAVE

Creating a Inventory

An inventory is a collection of hosts against which jobs may be launched. Inventories are divided into groups and these groups contain the actual hosts. Groups may be sourced manually, by entering host names into Tower, or from one of Ansible Tower’s supported cloud providers.

An Inventory can also be imported into Tower using the `tower-manage` command and this is how we are going to add an inventory for this workshop.

Step 1:

Click on INVENTORIES

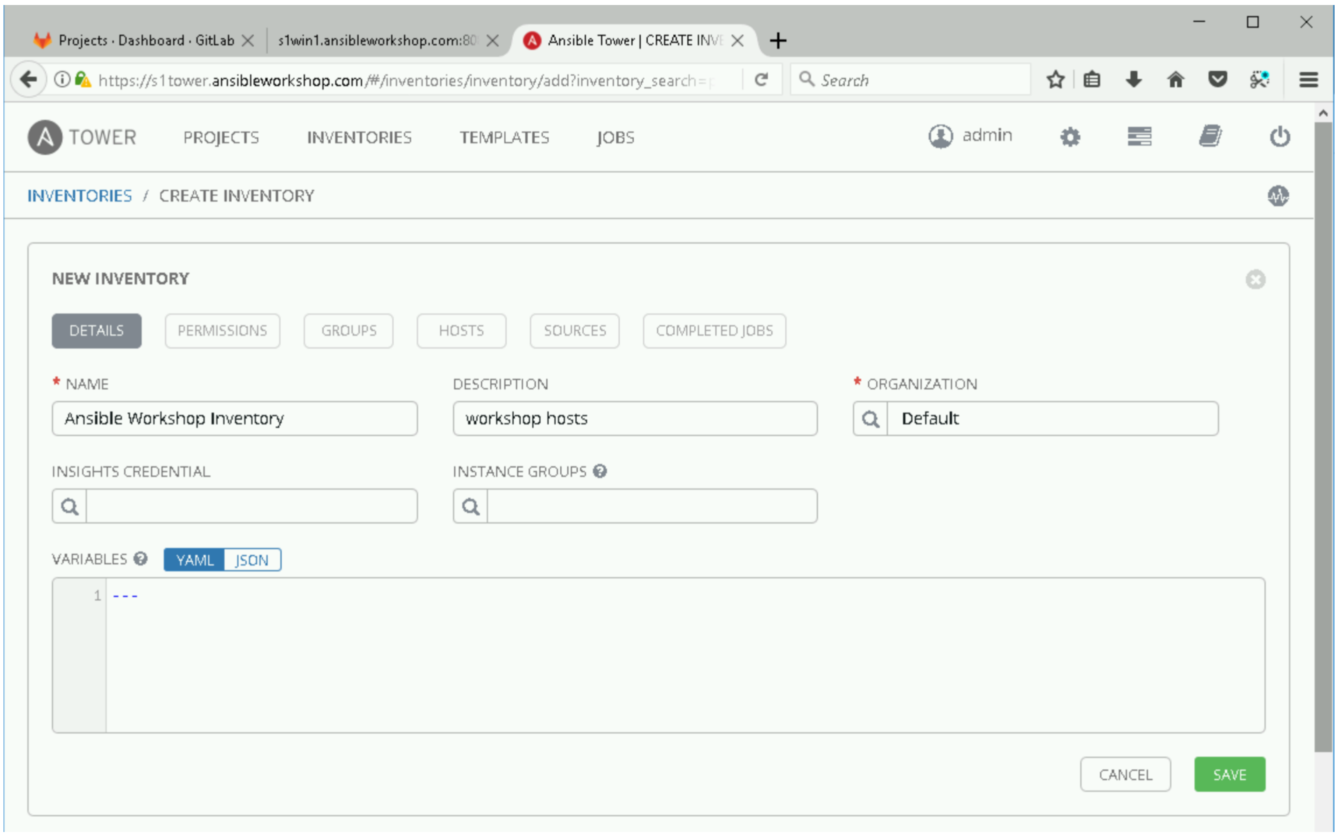
Step 2:

Select ADD and select Inventory + ADD

Step 3:

Complete the form using the following entries

NAME	Ansible Workshop Inventory
DESCRIPTION	workshop hosts
ORGANIZATION	Default



Projects · Dashboard · GitLab
s1win1.ansibleworkshop.com:80
Ansible Tower | CREATE INV
https://s1tower.ansibleworkshop.com/#/inventories/inventory/add?inventory_search=
Search
TOWER
PROJECTS
INVENTORIES
TEMPLATES
JOBS
admin
INVENTORIES / CREATE INVENTORY
NEW INVENTORY
DETAILS
PERMISSIONS
GROUPS
HOSTS
SOURCES
COMPLETED JOBS
INSIGHTS CREDENTIAL
INSTANCE GROUPS
VARIABLES
YAML
JSON
CANCEL
SAVE

* NAME	DESCRIPTION	* ORGANIZATION
Ansible Workshop Inventory	workshop hosts	Default

```
---
```

Create an Inventory

Step 4:

Select SAVE SAVE

Step 5:

Using putty, login into your tower node if you closed the window previously

```
s#tower.ansibleworkshop.com
```

Step 6:

Use the `tower-manage` command to import an existing inventory. (*Be sure to replace <username> with your actual username*)

```
sudo tower-manage inventory_import --source=/home/ec2-user/hosts --inventory
-name="Ansible Workshop Inventory"
```

You should see output similar to the following:

[at tm stdout] | *at_tm_stdout.png*

Importing an inventory with tower-manage

Feel free to browse your inventory in Tower. You should now notice that the inventory has been populated with Groups and that each of those groups contain hosts.

[at inv group] | *at_inv_group.png*

Inventory with Groups

Step 7:

TODO - Update the inventory in tower to use ansible_winrm_transport: kerberos TODO - Test with credssp as well? Do I need to enable that with the powershell script with special options?

End Result

At this point, we are doing with our basic configuration of Ansible Tower. In exercise 2.2, we will be solely focused on creating and running a job template so you can see Tower in action. :icons: font :imagesdir: images

Exercise 2.2 - Ad-hoc commands in Tower

TODO - Walk thru ad-hoc commands in Tower similar to the adhoc commands in the original exercise. Call out that the job execution is logged.

==

TODO

End Result

TODO

Exercise 2.3 - Creating and Running a Job Template

TODO - Update below for IIS/Windows …

A job template is a definition and set of parameters for running an Ansible job. Job templates are useful to execute the same job many times.

Creating a Job Template

Step 1:

Select TEMPLATES

Step 2:

Click on ADD + ADD, and select JOB TEMPLATE

Step 3:

Complete the form using the following values

NAME	IIS Basic Job Template
DESCRIPTION	Template for the iis-basic-playbook
JOB TYPE	Run
INVENTORY	Ansible Workshop Inventory
PROJECT	Ansible Workshop Project
PLAYBOOK	iis-basic-playbook/site.yml
MACHINE CREDENTIAL	Ansible Workshop Credential
LIMIT	web
OPTIONS	- [*] Enable Privilege Escalation

[at jt detail] | *at_jt_detail.png*

Job Template Form

Step 4:

Click SAVE SAVE and then select ADD SURVEY [Add]

Step 5:

Complete the survey form with following values

PROMPT	Please enter a test message for your new website
DESCRIPTION	Website test message prompt
ANSWER VARIABLE NAME	iis_test_message
ANSWER TYPE	Text

MINIMUM/MAXIMUM LENGTH	Use the defaults
DEFAULT ANSWER	Be creative, keep it clean, we're all professionals here

[at survey detail] | *at_survey_detail.png*

Survey Form

Step 6:

Select ADD + ADD

Step 7:

Select SAVE SAVE

Step 8:

Back on the main Job Template page, select SAVE SAVE again.

Running a Job Template

Now that you've sucessfully creating your Job Template, you are ready to launch it. Once you do, you will be redirected to a job screen which is refreshing in realtime showing you the status of the job.

Step 1:

Select TEMPLATES

Alternatively, if you haven't navigated away from the job templates creation page, you can scroll down to see all existing job templates

Step 2:

Click on the rocketship icon for the **IIS Basic Job Template**

Step 3:

When prompted, enter your desired test message

[at survey prompt] | *at_survey_prompt.png*

Survey Prompt

Step 4:

Select LAUNCH LAUNCH

Step 5:

Sit back, watch the magic happen

One of the first things you will notice is the summary section. This gives you details about your job such as who launched it, what playbook it's running, what the status is, i.e. pending, running, or complete.

[at job status] | *at_job_status.png*

Job Summary

Scrolling down, you will be able to see details on the play and each task in the playbook.

[at job tasklist] | *at_job_tasklist.png*

Play and Task Details

To the right, you can view standard output; the same way you could if you were running Ansible Core from the command line.

[at job stdout] | *at_job_stdout.png*

Job Standard Output

Step 6:

Once your job is sucessful, navigate to your new website

```
http://s#win1.ansibleworkshop.com
```

If all went well, you should see something like this, but with your own custom message of course.

[at web tm] | *at_web_tm.png*

New Website with Personalized Test Message

End Result

At this point in the workshop, you’ve experienced the core functionality of Ansible Tower. But wait… there’s more! You’ve just begun to explore the possibilities of Ansible Core and Tower. Take a look at the resources page in this guide to explore some more features.

Exercise 3.0 - Using Ansible for Windows Patching

Walk through a patching process including things such as:

- WSUS selecting / approving patches - maybe have a WSUS server for the lab
- Downloading packages before patching window
- Scheduling patching - How to group nodes
- Notification on failure of patching
- Review/Confirmation of patches deployed

Header 1

TODO

Step 1:

TODO

Step 2:

TODO

Header 2

TODO

Step 1:

TODO

End Result

TODO

Exercise 3.1 - Using Ansible for Desired State Config

Review use cases around using new desired state config capabilities

Header 1

TODO

Step 1:

TODO

Step 2:

TODO

Header 2

TODO

Step 1:

TODO

End Result

TODO

Exercise 3.2 - Using Ansible for Self Service non-admin users

Examples where user can restart services, servers, etc without being administrators…

Header 1

TODO

Step 1:

TODO

Step 2:

TODO

Header 2

TODO

Step 1:

TODO

End Result

TODO

Wrap Up

That wraps up what we have planned for today. We hope you've learned something valuable about Ansible and Ansible Tower that you can apply in your daily role.

Q&A

What do you think? How can we help you understand Ansible Tower better?

Thank you for your time and participation!