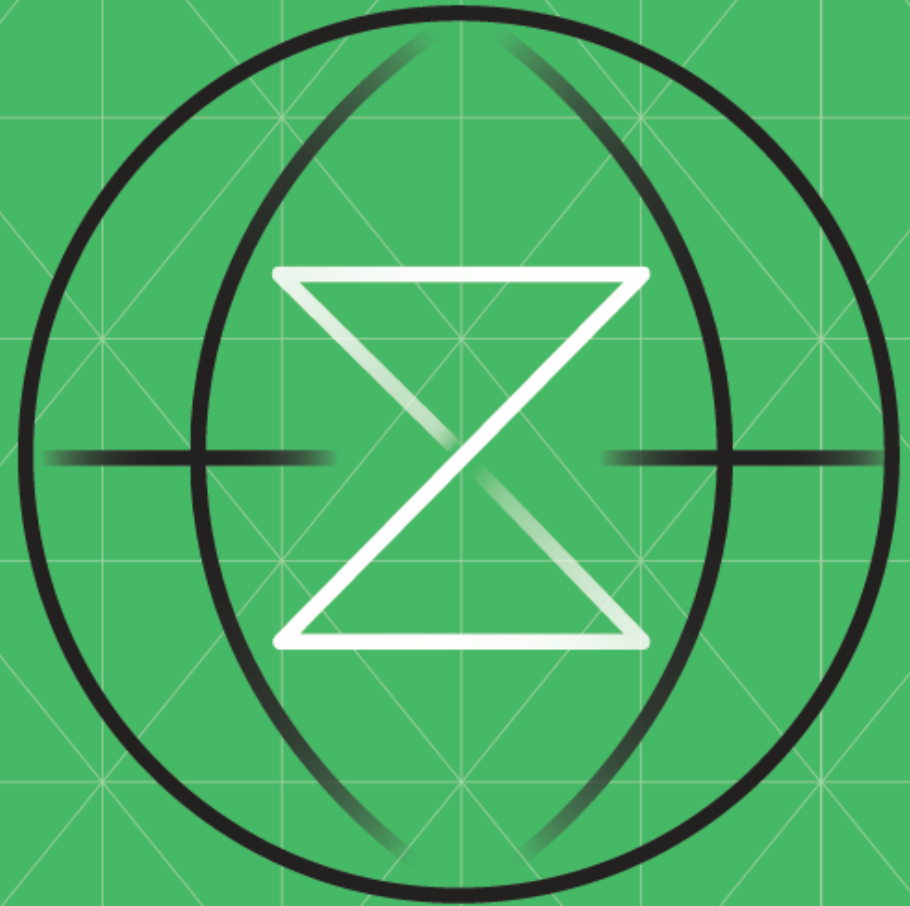


ANSB2

Hitting The Playbooks

- [HITTING THE PLAYBOOKS](#)
- [1 REVISIT YOUR WORK](#)
- [2 POWERING UP YOUR VSCODE](#)
- [3 TELL ME MORE](#)
- [4 TRY OUT THESE EXAMPLES](#)
- [5 SYNTAX HINTS](#)
- [6 ENCODING MATTERS](#)
- [7 CONVERT AND TRANSFORM](#)
- [8 FOLLOW THE STEPS](#)
- [9 DON'T STOP NOW!](#)



HITTING THE PLAYBOOKS

The Challenge

With your environment set up, you are now able to take charge of these Ansible playbooks and really make things happen.

In this challenge, you will learn about EBCDIC and ASCII character encodings, as well as the commands used to convert between them, then link together a few commands in a slightly more complex playbook to handle the completion of this chapter in what has hopefully been an exciting series of challenges.

Before You Begin

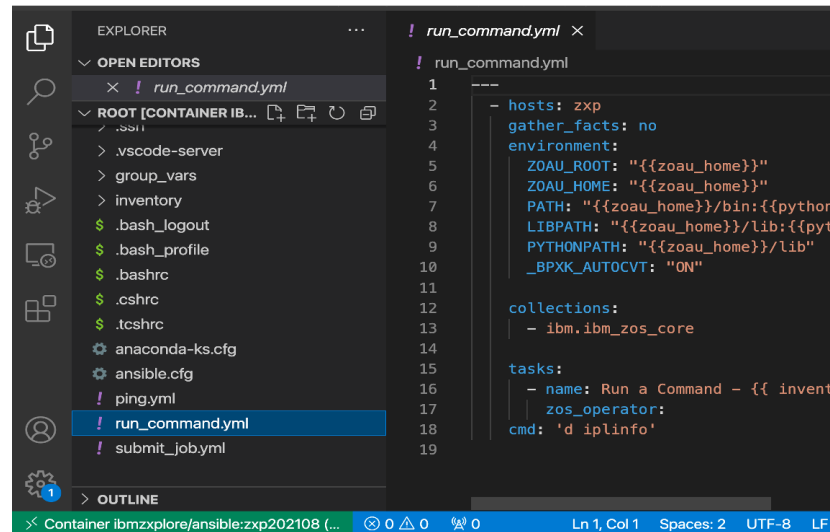
Hopefully you don't skip right to the end, because there's a LOT of information about the mainframe you need to see ... pretty much everything in here.

If you have arrived here by doing all of the previous challenges, then you're in the right place. Welcome.

Investment

Steps	Duration
9	120 minutes

1 REVISIT YOUR WORK



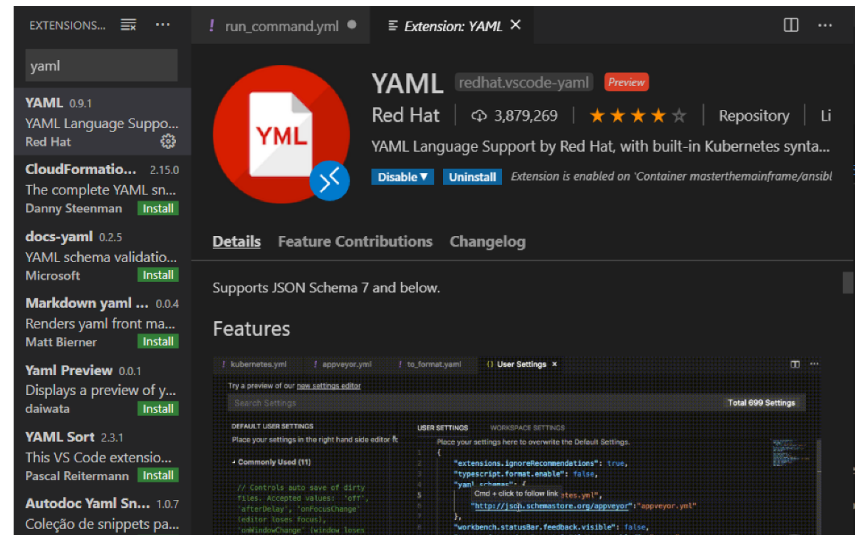
```
1 ---
2 - hosts: zxp
3   gather_facts: no
4   environment:
5     ZOAU_ROOT: "{{zoau_home}}"
6     ZOAU_HOME: "{{zoau_home}}"
7     PATH: "{{zoau_home}}/bin:{{python_path}}"
8     LIBPATH: "{{zoau_home}}/lib:{{python_libpath}}"
9     PYTHONPATH: "{{zoau_home}}/lib"
10    _BPXK_AUTOCVT: "ON"
11
12   collections:
13     - ibm.ibm_zos_core
14
15   tasks:
16     - name: Run a Command - {{ inventory_hostname }}
17       zos_operator:
18         cmd: 'd iplinfo'
```

In addition to Ansible knowing it should look in the **zos_core** collection, you also need to spell out where **ZOAU** is installed.

Leaving these out will cause the command to fail, but if you made it this far, you probably figured that out.

ANS21230301-1116

2 POWERING UP YOUR VSCODE



You may have run into previous problems getting your YAML file formatted correctly, and learned that

****indentation matters**!**

There is a YAML extension by Red Hat for VSCode that can detect errors before you submit playbooks, give greater visibility of document structure, and provide autocompletion where supported.

It is not required, but may be helpful.

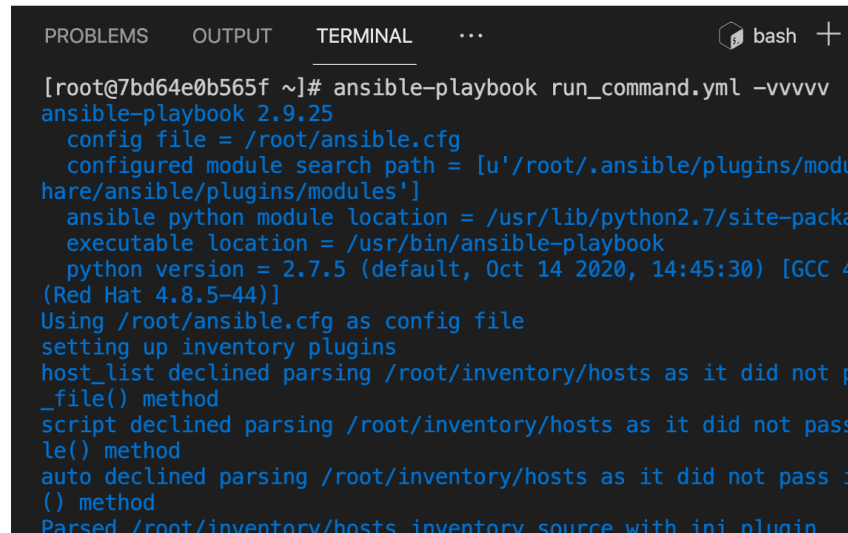
In fact, there are extensions for Ansible, and even Ansible specifically on Z, that you might want to check out as you go forward.

They can drastically cut down on development time as you gain experience (*as long as they don't conflict with one another* - something you usually find out by trial-and-error)

3 TELL ME MORE

Do you ever wish you could get just a little more information about what's happening while an Ansible playbook is progressing?

I mean, it's nice when things just work and don't bother you, but sometimes you just want some more clues to help solve a problem.

A terminal window with tabs for PROBLEMS, OUTPUT, and TERMINAL. The terminal shows the command 'ansible-playbook run_command.yml -vvvvv' being executed. The output displays various configuration details such as the config file path, module search path, python module location, and executable location. It also shows that several parsing methods (host_list, _file(), script, le(), auto, ()) have declined parsing the inventory file, and finally, the inventory source was parsed using the ini plugin.

```
PROBLEMS OUTPUT TERMINAL ... bash +
[root@7bd64e0b565f ~]# ansible-playbook run_command.yml -vvvvv
ansible-playbook 2.9.25
  config file = /root/.ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.5 (default, Oct 14 2020, 14:45:30) [GCC 4.8.5]
  (Red Hat 4.8.5-44)]
Using /root/.ansible.cfg as config file
setting up inventory plugins
host_list declined parsing /root/inventory/hosts as it did not pass its
_file() method
script declined parsing /root/inventory/hosts as it did not pass its
le() method
auto declined parsing /root/inventory/hosts as it did not pass its
() method
Parsed /root/inventory/hosts inventory source with ini plugin
```

Add a `-v` after your command to make it a little more verbose.

Throw in a few more v's to make it even more verbose. Use `-vvvvv` to get the most information available.

ANSIBLE|230301-1116

4 TRY OUT THESE EXAMPLES

Remember how we said examples are good documentation? That *definitely* applies here!

In the z/OS core collection are examples where you're most likely going to use them.

Examples

```
- name: Job output with ddname
  zos_job_output:
    job_id: "STC02560"
    ddname: "JESMSGLG"

- name: JES Job output without ddname
  zos_job_output:
    job_id: "STC02560"

- name: JES Job output with all ddnames
  zos_job_output:
    job_id: "STC*"
    job_name: "*"
    owner: "IBMUSER"
    ddname: "?"
```

In most cases, you can just copy/paste these from the page into your code, and then adjust to meet your specific needs.

Remember that you can find the examples at: https://ibm.github.io/z_ansible_collections_doc/ibm_zos_core/docs/ansible_content.html

5 SYNTAX HINTS

state

The final state desired for specified data set.

If `state=absent` and the data set does not exist on the managed node, no action taken, module completes successfully with `changed=False`.

If `state=absent` and the data set does exist on the managed node, remove the data set, module completes successfully with `changed=True`.

If `state=absent` and `volumes` is provided, and the data set is not found in the catalog, the module attempts to perform catalog using supplied `name` and `volumes`. If the attempt to catalog the data set catalog is successful, then the data set is removed. Module completes successfully with `changed=True`.

If `state=absent` and `volumes` is provided, and the data set is not found in the catalog, the module attempts to perform catalog using supplied `name` and `volumes`. If the attempt to catalog the data set catalog fails, then no action is taken. Module completes successfully with `changed=False`.

If `state=present` and the data set does not exist on the managed node, create and catalog the data set, module completes successfully with `changed=True`.

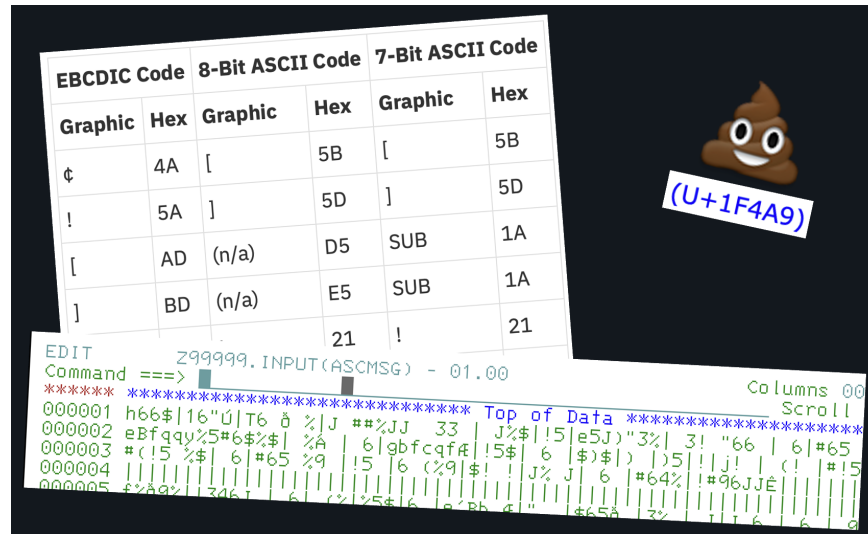
The design philosophy for Ansible tasks is to describe *what needs to be done* as succinctly as possible, without it being confusing. This is in comparison to describing *how things need to be done*.

If you look at the **zos_data_set** module, for example, the key difference between a “create” and a “delete” is the `state` parameter, which is set to “present” for create and “absent” for delete.

You basically write what you might describe as the “ideal outcome” of that step.

You will find similar phrasing and approaches in the non-z/OS modules as well.

6 ENCODING MATTERS



A long time ago, back in the 1950s, IBM came up with a way of representing characters in computer memory called “extended binary-coded decimal interchange code” [EBCDIC \(IBM-1047\)](#).

Another standard to represent characters came around later (1961 onwards) called “American Standard Code for Information Interchange,” [ASCII \(also referred to as ISO8859-1\)](#).

They are both equally capable, although if you’re looking at text on a mainframe, there’s a very good chance it is stored and processed as EBCDIC, whereas text you see from the internet, or in an mobile app is likely ASCII, or [UNICODE](#).

The world of character encoding is a wild one, and if you’re in need of some bedtime reading material, there are plenty of videos and articles written on the topic.

Just know this: Data sets on IBM zOS are EBCDIC by default, and we need to convert or translate ASCII if we want to read that in VSCode, for example.

TELL ME MORE ABOUT THIS WHOLE CHARACTER ENCODING THING

I'm so glad you asked!

1. Picture a single bit of memory, which can either be a 1 or a 0, on or off. With that one bit, you can represent two numbers, Zero or One.
2. Put another bit next to it, and you've doubled the amount of numbers you can represent, now 4.
3. Add another, you're up to 8, then 16, and so on.

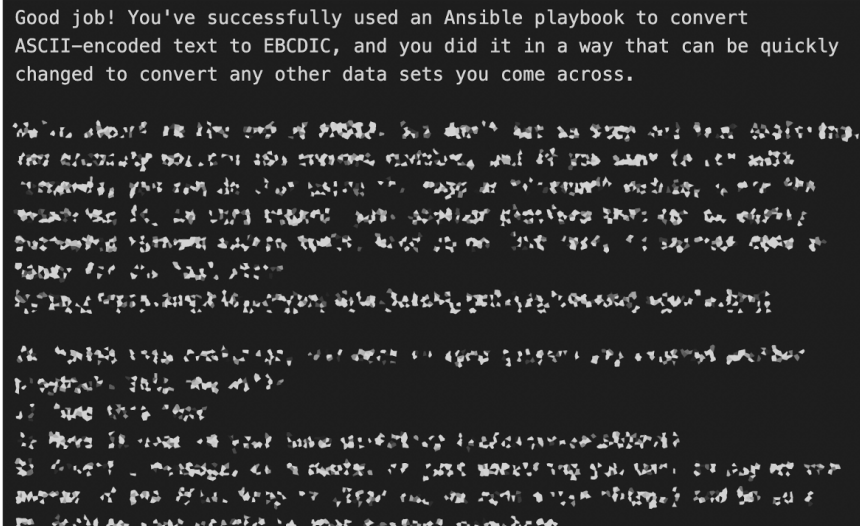
Character encoding is simply an attempt at representing a set of characters electronically using bits.

With letters and characters, things get a lot more confusing. A coding system needs to allow for a large set of characters, while also being efficient. Consider that certain characters only appear in certain spoken and written languages, and other characters are rarely seen outside of a computer.

Some encoding favor efficiency, others flexibility. It pays to not take the encoding of the text you're using for granted.

8 FOLLOW THE STEPS

The instructions to finish this challenge are in the transformed text, which you can see clearly, but only if you succeeded in converting it.



```
Good job! You've successfully used an Ansible playbook to convert
ASCII-encoded text to EBCDIC, and you did it in a way that can be quickly
changed to convert any other data sets you come across.

We're about at the end of this. You don't get as much as you could get,
but enough to get you started. And if you want to get more
information, you can do that by going to the page at https://www.ibm.com/docs/en/ansible-playbook/2.9.0?topic=ansible-playbook-2.9.0.
We're about at the end of this. You don't get as much as you could get,
but enough to get you started. And if you want to get more
information, you can do that by going to the page at https://www.ibm.com/docs/en/ansible-playbook/2.9.0?topic=ansible-playbook-2.9.0.
We're about at the end of this. You don't get as much as you could get,
but enough to get you started. And if you want to get more
information, you can do that by going to the page at https://www.ibm.com/docs/en/ansible-playbook/2.9.0?topic=ansible-playbook-2.9.0.
```

Read those instructions carefully, and begin crafting another Ansible playbook.

This one will be a little bit more complicated, and use a little bit of logic, but by now, we know that you are up to the challenge.

Something to remember - the *ping.yml* playbook lists "zxp" as a specific host - if a playbook does not specify hosts, then *ansible-playbook* will try and use all hosts in the inventory; if any hosts in the inventory are not accessible, you may see errors such as

```
fatal: [192.86.32.153]: UNREACHABLE! => {"changed": false, "msg": "Data could not be sent to
remote host \"192.86.32.153\". Make sure this host can be reached over ssh: ssh: connect to host
192.86.32.153 port 22: Connection refused\r\n", "unreachable": true}
```

To avoid this, change the inventory/hosts, or the hosts in the playbook.

To validate this step for completion, submit **CHKAANS2** from **ZXP.PUBLIC.JCL** and make it count.

9 DON'T STOP NOW!



Your mind should be whirring right now, and that's a good thing!

The last thing you need is for us to tell you what to do, because you've probably got a list of things you *want* to try.

Maybe you can automate some of that nice ASCII art we got from REXX a while back? Perhaps there's a better way of running one of your python programs automatically.

ANSB21230301-1116

WHY ALL THE PREP WORK?

Getting a program to work once is good. Getting it to work hundreds, if not thousands of times after that, the exact same way, takes careful orchestration.

In the past, that meant that a person, or a team of people, had to manually configure a whole group of systems, and carefully set them up so that a program could run on them. They would manage every step of the process, and doubling the amount of programs running meant everyone had to work twice as hard.

That's the old way!

Docker containers, and orchestration that happens in solutions like Ansible and Kubernetes, means that scaling up in capability doesn't mean adding complexity at every step. It's an important revolution in Enterprise IT that not only produces more capable solutions, but frees up staff to work on other tasks - like coming up with the next greatest thing, instead of setting up yet another system.

Nice job - let's recap	Next up ...
<p>As you can see, integrating z/OS into Ansible isn't just about all of the new things you can do, but how much you can simplify or automate existing tasks that would typically require a great deal of manual steps or deep institutional knowledge.</p> <p>Now that you know both aspects of automation, there's really no limit to what you can accomplish.</p>	<p>If you've saved these Ansible challenges for last, then you're done with all of the spelled-out challenges. Congratulations! All that is left to do (aside from a quick victory dance) is to read up on the current Student Contest, and start writing your First Prize acceptance speech.</p> <p>(always stay optimistic!)</p>