HPE GreenLake for Compute Ops Management bare metal provisioning with Ansible

Living Lab Experience

Contents

Automatic bare metal provisioning with HPE Compute Ops Management and Ansible

A new project, an open-source initiative hosted on GitHub has been released recently that aims to enhance the integration between HPE GreenLake for Compute Ops Management and Ansible. This endeavor is focused on making it easier to configure, manage and provision bare metal servers at scale.

Automatic bare metal provisioning refers to the process of automatically deploying and configuring physical servers or bare metal machines using automated tools such as Ansible in this project. The goal is to enable quick and easy provisioning of servers managed by HPE GreenLake for Compute Ops Management and enable the long list of benefits of automatic bare metal provisioning.

The initial aim of this project was to focus on server provisioning for the ESXi, RHEL and Windows Server platforms. However, it also aims to provide an overview of the various capabilities of the Compute Ops Management API. The project effectively demonstrates a wide range of COM API interactions, covering everything from initial installation (Day0 operations) through the early stages of active use (Day1) to ongoing maintenance (Day2) with automated firmware updates.

Operations include the capture of server information, the creation of server settings (bios, storage, OS provisioning) the creation of server groups with server settings, the addition of servers to a server group, the launch of firmware updates and the monitoring of task execution and the management of task errors.

In this project, automating the provisioning of operating systems on bare metal servers is made simple and accessible to anyone with basic knowledge of Ansible, HPE Compute Ops Management, and kickstart techniques. While it is generally a complex process that requires a wide range of skills, this project simplifies it with the use of auto-customized kickstarts, auto-generated ISO files and by exploiting the very interesting functions of HPE Compute Ops Management server groups.

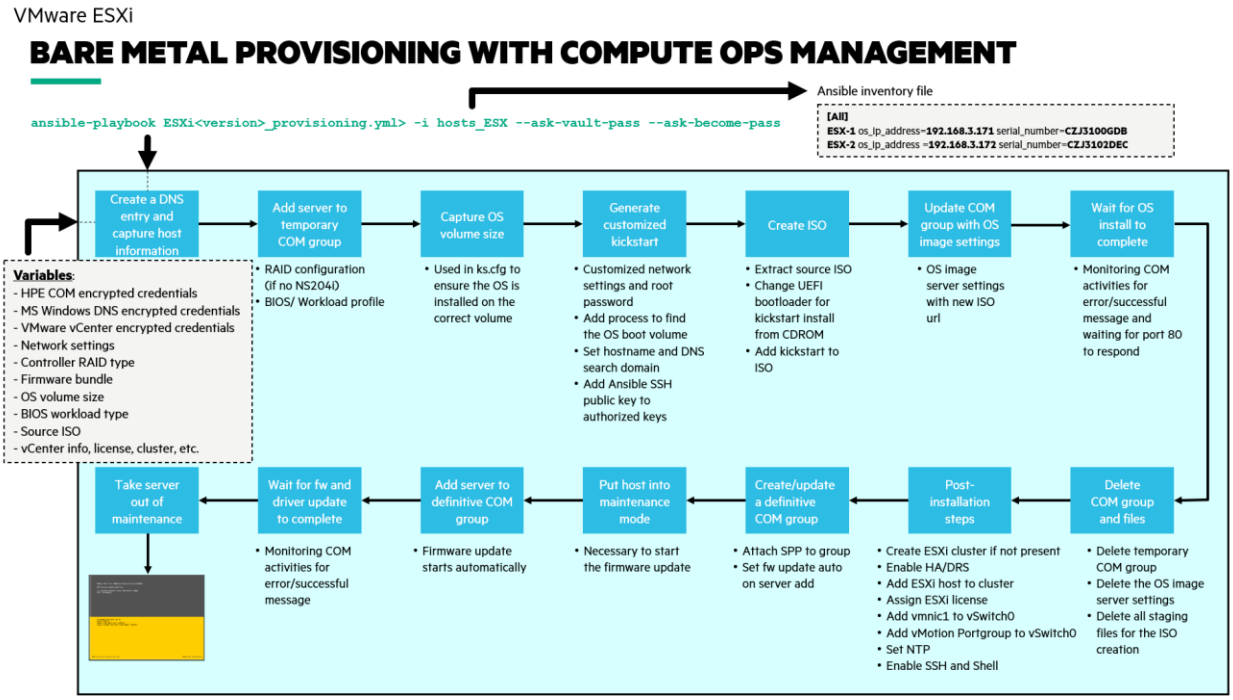
Objectives

The purpose of this lab is to show you the different steps that are necessary to use this project and ultimately provision a server with VMware ESXi, the main steps are as follows:

1. Configure VS Code and clone the GitHub project repository.
2. Prepare a Rocky Linux VM freshly installed to be the Ansible control node.
3. Set the different variables that will be used by Ansible to configure your server.
4. Run an Ansible playbook to provision one ESXi server in a cluster.
5. Run an Ansible playbook to decommission the ESXi server.

Process flow

The following diagram describes the various process steps and sequences integrated into the ESXi playbook that you will be using in this lab:



Lab Setup Overview

Virtual Machines Configuration:

- Primary Workstation with Windows 11:

Your main point of entry in this lab will be through a virtual machine running Windows 11. Consider this as your jump station, where you'll operate Visual Studio Code.

- Ansible Control Node on Rocky Linux 9.3:

The second virtual machine you'll use is set up with Rocky Linux 9.3. This machine serves as the Ansible Control Node, from which you'll execute Ansible commands and manage various playbooks.

The physical infrastructure hosting these virtual machines and the HPE ProLiant DL160/DL365 servers you'll be using for the bare metal provisioning are in our Houston lab in Texas. These servers are already onboarded to a Compute Ops management instance on the HPE GreenLake platform.

Task 1: Accessing the Lab and setting up your development environment

You'll start by connecting to your designated Windows VM using Remote Desktop Protocol (RDP). You will then use VS Code and clone the GitHub project repository into your development workspace, then link it to the Linux VM by creating an SSH connection which will facilitate direct editing and configuration on the Linux system. Then you'll install and configure all the required components for the project on the Linux VM.

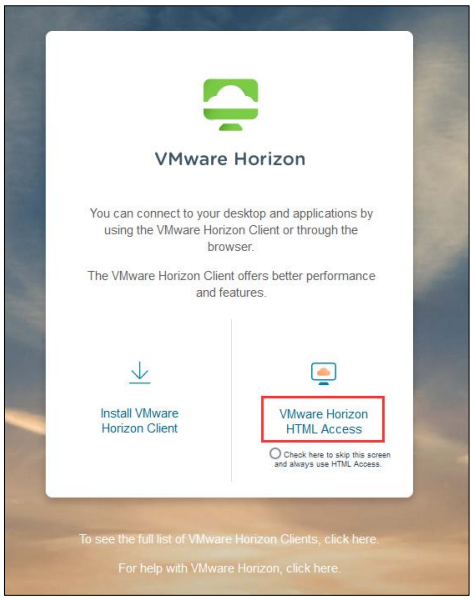
Once this is done, you'll customize the project variables and complete the process by provisioning one of the servers available within our server pool on the HPE GreenLake platform.

Commands that you should copy and paste, or type in your terminal during this lab look like this:

```
Whoami
```

Step 1: Connecting to the Lab

- Open a Chrome or similar HTML5 capable browser and navigate to: https://xx.xx.xx.xx/portal/
- Choose VMware Horizon HTML Access

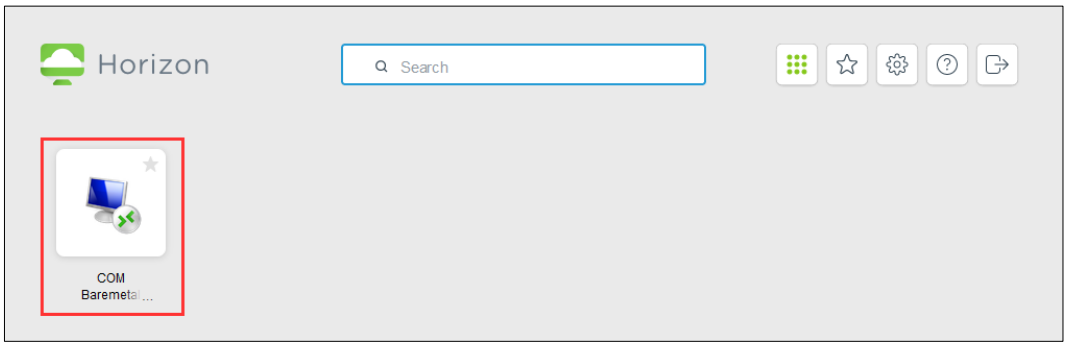


- Login with your provided credentials

  Important note: Your credentials and team assignments can be found on the sheet of paper issued to you by your lab supervisor. Please return the sheet to your supervisor when you have completed this lab.

  Since Horizon is presenting a remote desktop session inside of your desktop's browser, it may be helpful to hit **F11** at this time to put the browser in full screen mode.

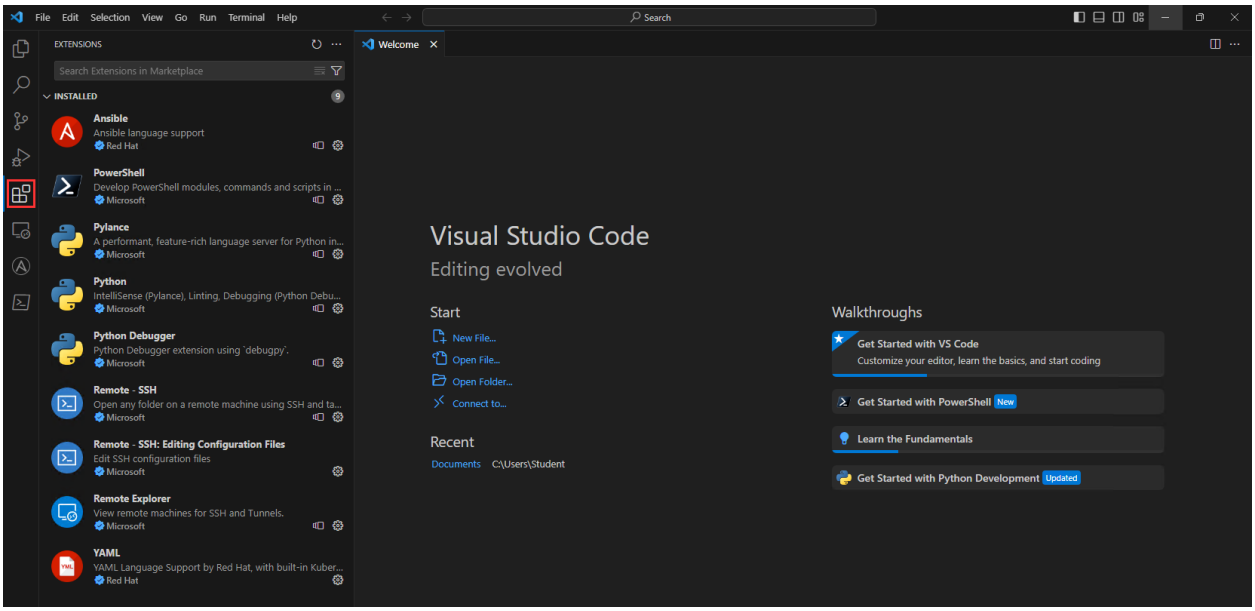
- Click on the RDP icon to launch a remote desktop to your Windows virtual machine:



Step 2: VS Code configuration

- Open **Visual Studio Code** (VS Code) using the shortcut on the VM desktop.

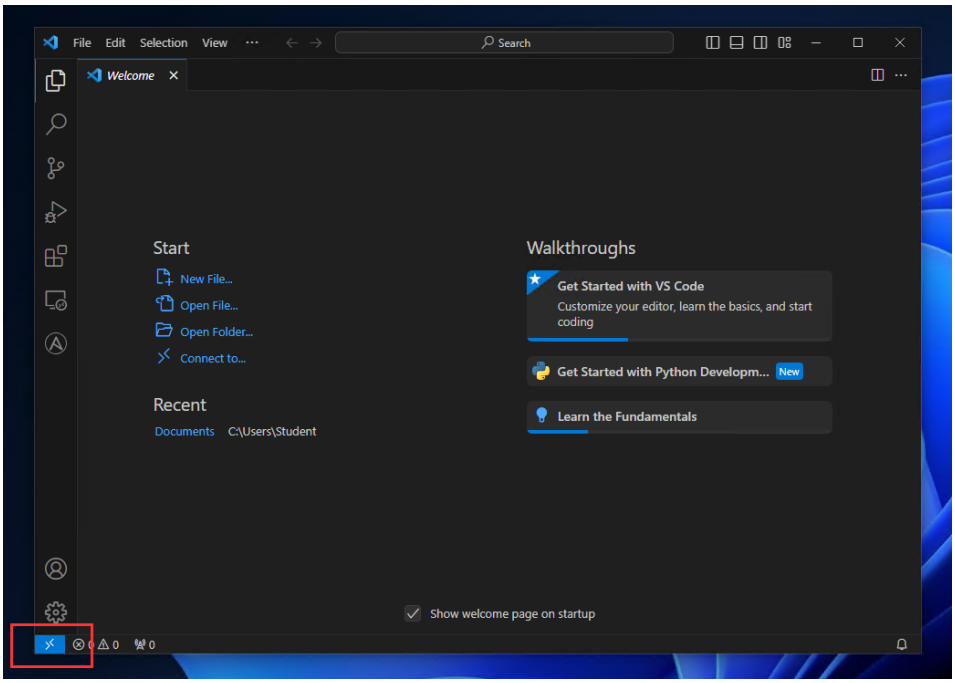
- Notice that we already installed a few VS Code extensions, you can find them by clicking on the in **Extensions** icon in the Activity Bar on the side of VS Code:



  To start working with our GitHub project, you need at least the following extensions:
  - **Ansible**: adds language support for Ansible.
  - **Remote – SSH**: lets you use any remote machine with a SSH server as your development environment. This can greatly simplify development and troubleshooting in a wide variety of situations.

- You can now close the extensions page by clicking on the **Extensions** icon again.

- Next, click on the remote SSH icon in the bottom left-hand corner of VS Code:

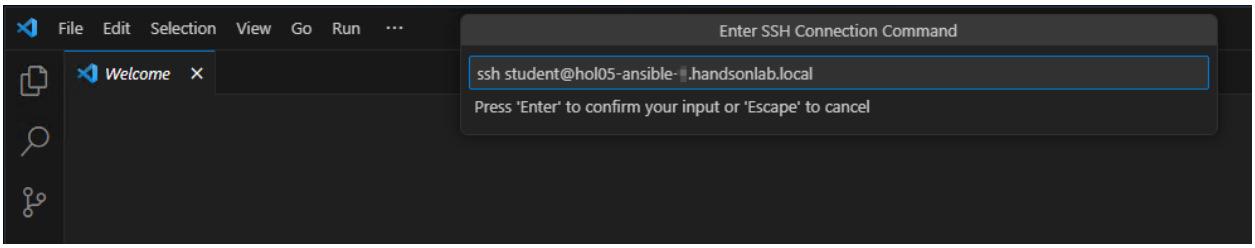


Remote SSH is a feature provided by VS Code that allows users to connect to and interact with a remote machine or server using the Secure Shell (SSH) protocol. This functionality is made available through the extension "Remote - SSH" which has been installed from the VS Code Extensions Marketplace.

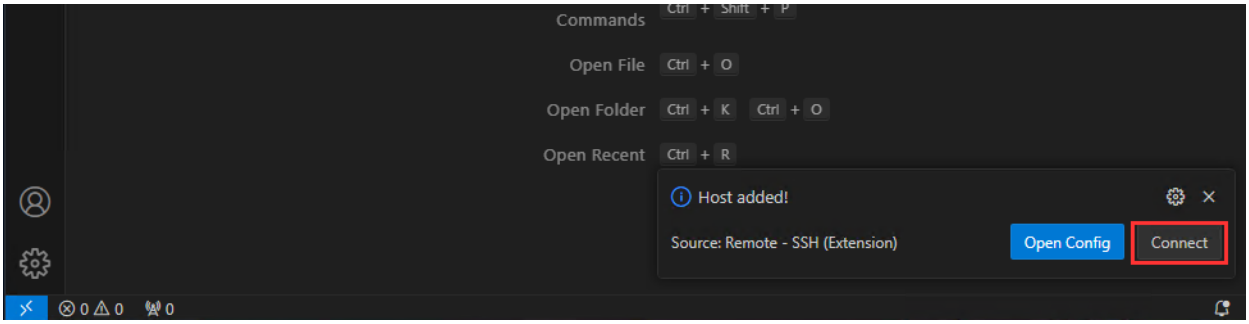
- Select **Connect to Host...**
- Then click on **+Add New SSH Host...**
- Then enter the following SSH command to connect to your designated Linux virtual machine that you'll use as your Ansible Control Node:

  `ssh student@hol05-ansible-x.handsonlab.local`

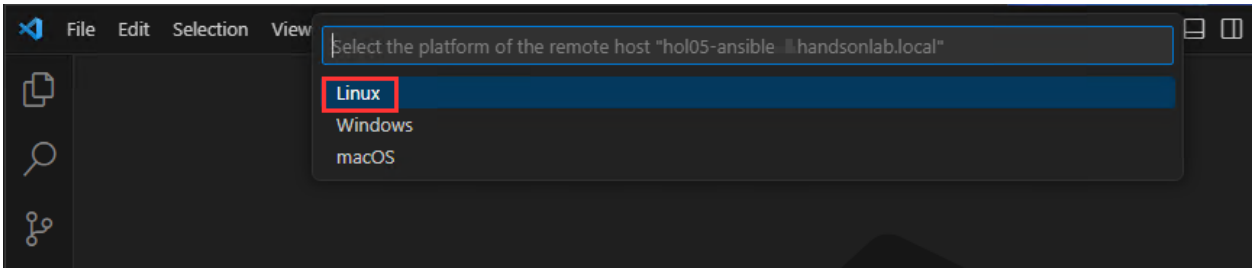
  with x your team number.



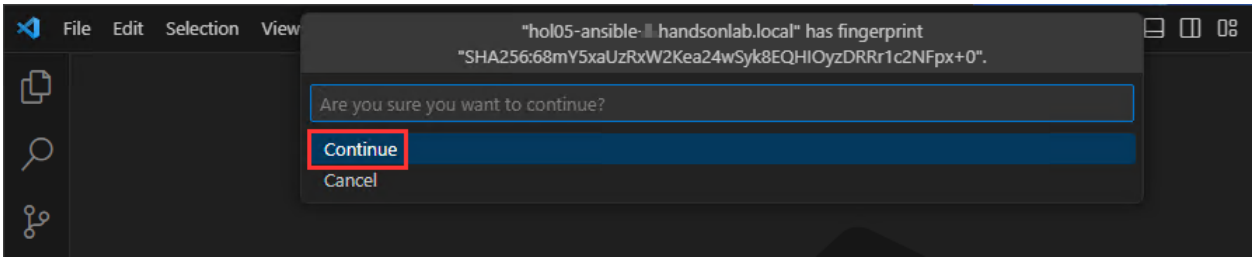
- Then select **C:\Users\Student\.ssh\config**
- Then click on **Connect** from the pop-up window:



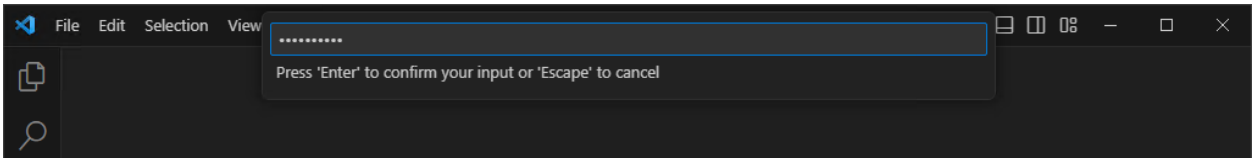
- A new VS Code window opens, select **Linux**:



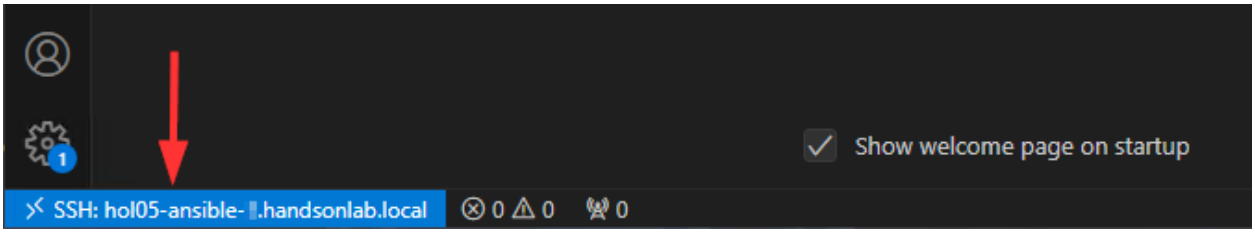
- Click on **Continue** at the fingerprint message:



- Next enter the password: **xxxxxxxxx** then press **Enter:**

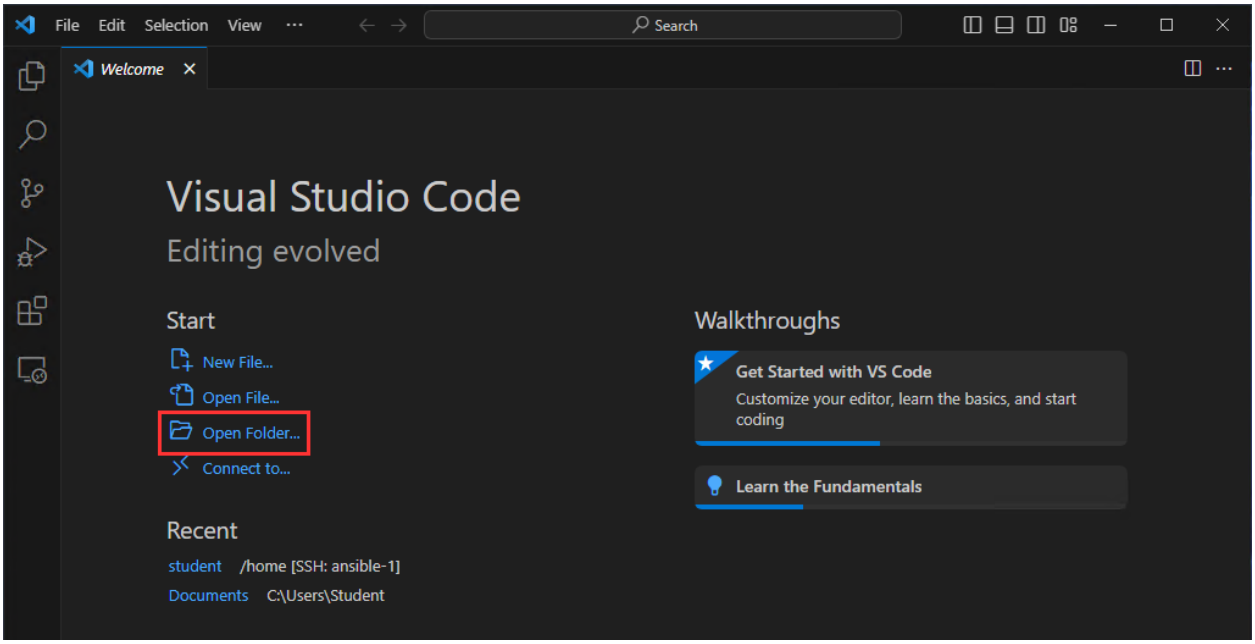


The SSH connection has been established and is displayed on the status bar:

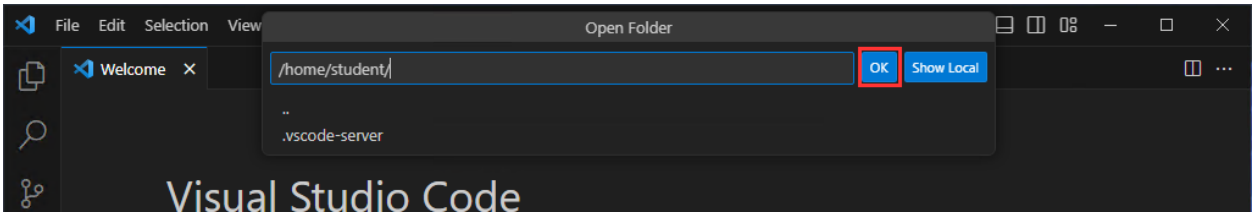


You currently have two instances of VS Code running. Retain the window with the new SSH connection and close the previous one.

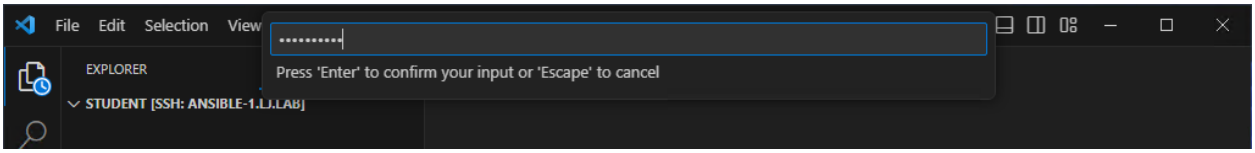
- Next, you can open a folder on the Linux VM, click on **Open Folder...**



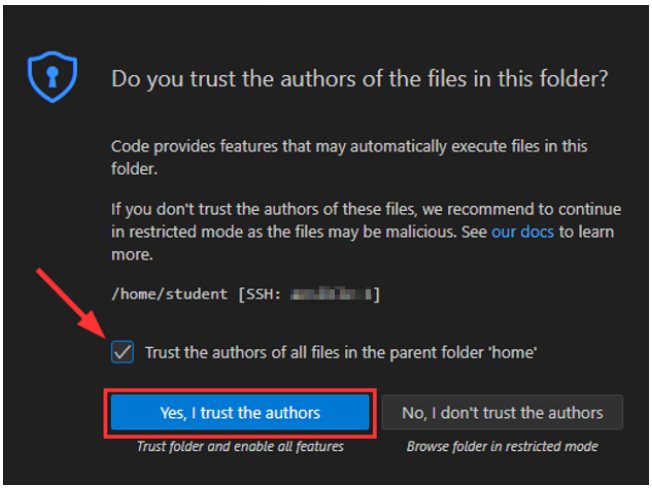
- The default **/home/student** folder should be proposed, simply click on **OK**:



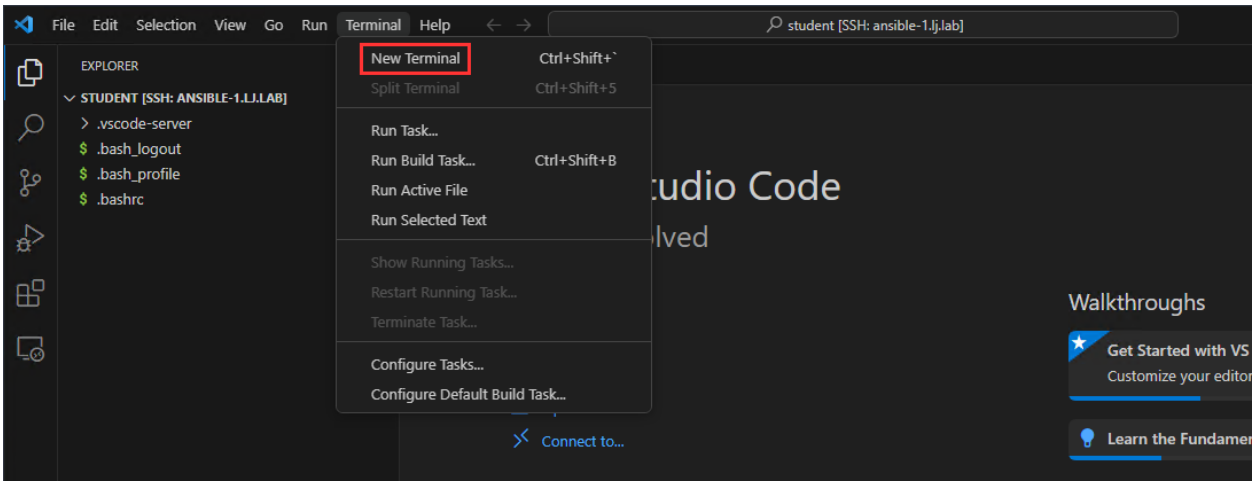
- A new dialogue pops up, you need to re-select **Linux** and re-enter the **xxxxxxx** password then press **Enter**:



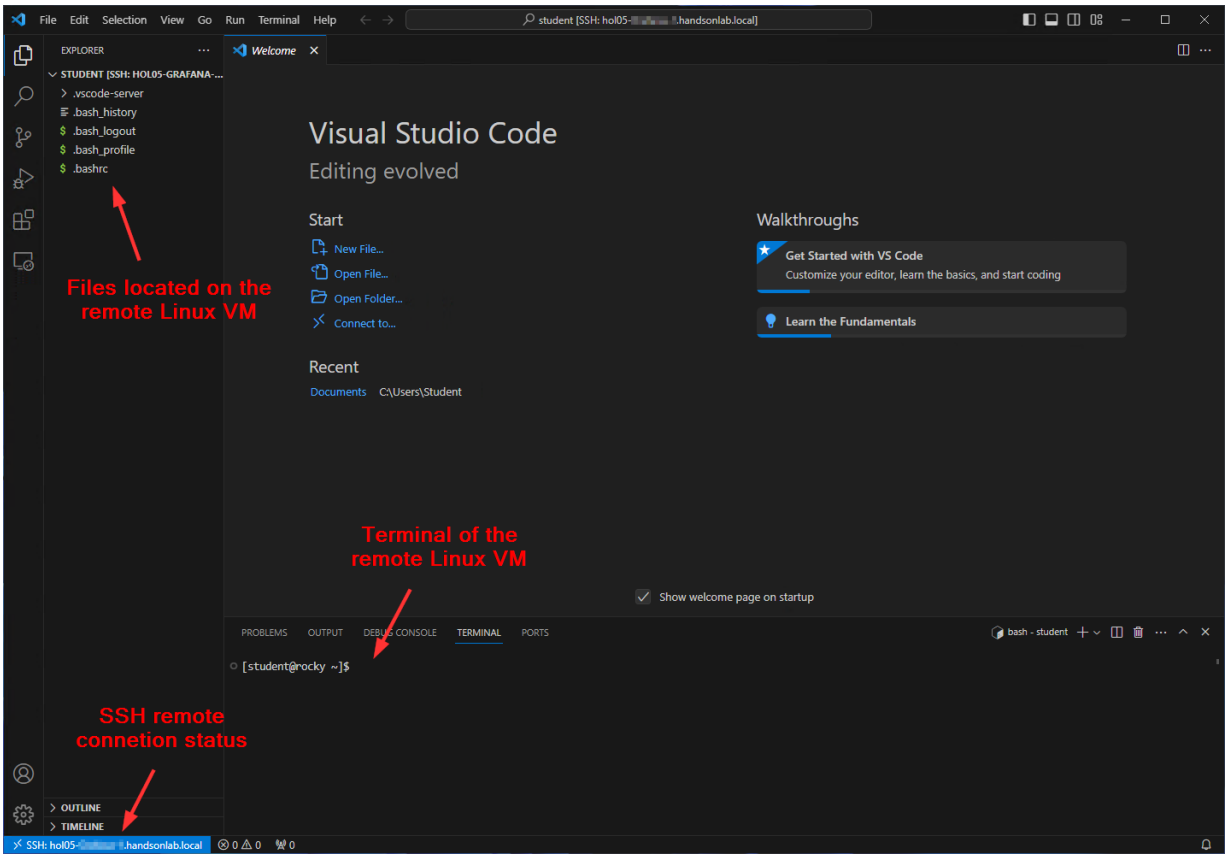
- Once connected, VS Code returns a `Do you trust the authors of the files in this folder?`. Check the trust the authors option and click **Yes, I trust the authors**:



- Next, make the Linux terminal visible by clicking on **Terminal** / **New Terminal**:



- You should end up with the following workspace:



Step 3: Cloning the HPE-COM-baremetal GitHub repository

To get started with the HPE-COM-baremetal GitHub project on your remote Linux Virtual Machine, you'll need to clone the repository. However, before you can do that, it's essential to ensure that **git** is installed on your VM.

- From the terminal, execute the following command:

  `sudo dnf -y install git`

- After successfully installing **git**, it is necessary to restart VS Code to enable the integration of git within the editor. Upon reopening VS Code, you'll be prompted to re-enter your password for confirmation.

- To clone the HPE-COM-baremetal GitHub repository, ensure that you're in your home directory (`/home/student` using **pwd**) before proceeding with the cloning process. Once you're in the correct directory, use:

  `git clone https://github.com/jullienl/HPE-COM-baremetal`

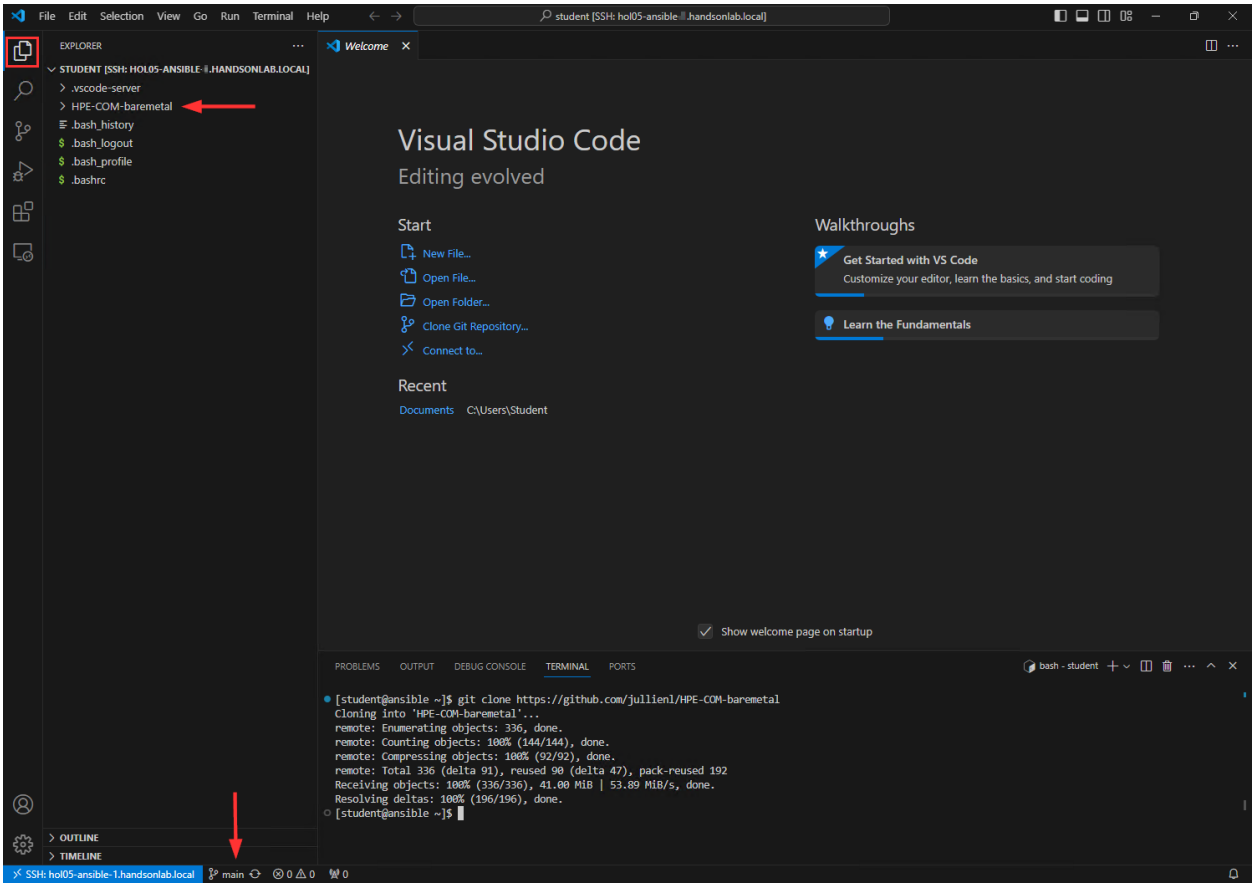
  When you perform a clone, you are essentially creating a full-fledged copy of the repository's data, which includes the history of all the changes that have been made to the files and the ability to track future changes.

  Two important new elements must now be present in your VS Code workspace:

  – A new folder **HPE-COM-baremetal** created in your student home folder- Git clone duplicated all the data the repository contains—the code, branches, tags, and commits.

  Note: It may be necessary to open the Explorer view by clicking on the **Explorer** icon in the activity bar.

  – The source control branch **main** displayed at the bottom on the **Status** bar- establishes a connection between the remote repository and the newly created local clone. This enables you to sync future updates by pulling in changes or pushing out your own.



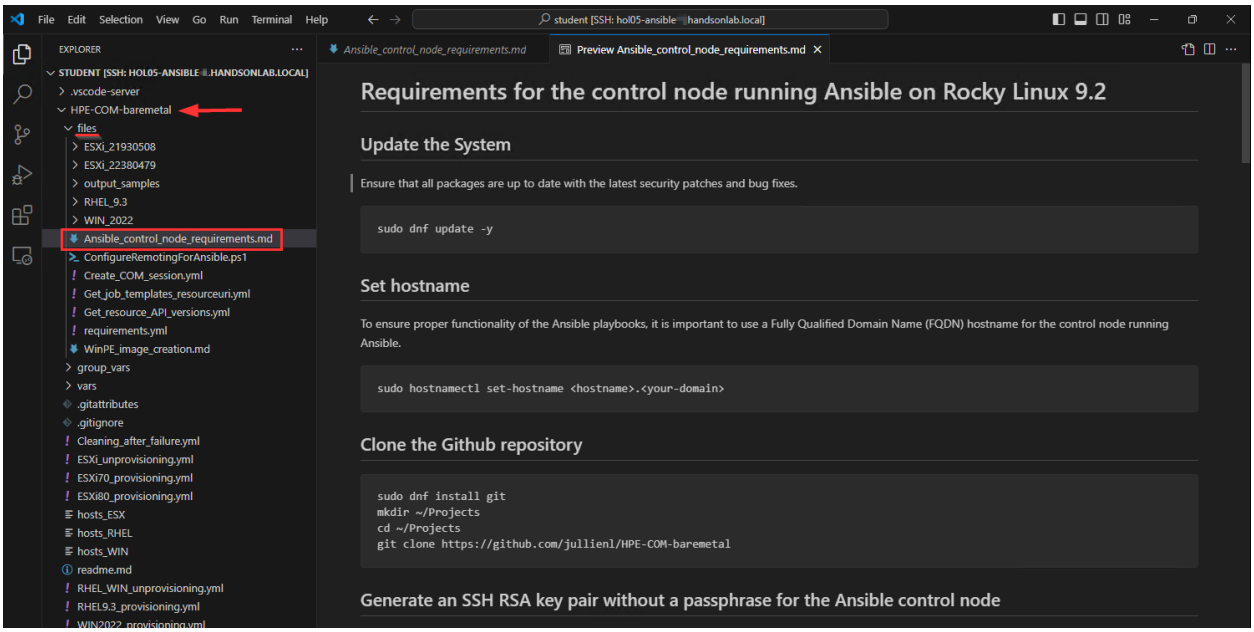
Task 2: Preparation of the Linux Virtual Machine

The preparation of the VM main objective is to turn this out-of-the box Rocky Linux VM, freshly installed as your Ansible control node of this project.

The Ansible control node is the machine where Ansible is installed and from which all tasks and playbooks are executed. It plays several critical roles in an Ansible-managed environment. This machine will also serve as a web server to host the custom ISO images that will be generated, and from which provisioned servers will boot from.

Note: The HPE-COM-baremetal GitHub repository contains a page that outlines every necessary step for setting up the Ansible control node on Rocky Linux. That same page can also be found in your home directory in **HPE-COM-baremetal/files/Ansible_control_node_requirements.md**

Note: To get a preview of the **Ansible_control_node_requirements.md** markdown file in Visual Studio Code, you can use the keyboard shortcut: **CTRL** + **SHIFT** + **V**



The steps you'll carry out in the following section roughly correspond to the various steps described in this file.

Step 1: Set hostname

- To ensure proper functionality of the Ansible playbooks, it is important to use a Fully Qualified Domain Name (FQDN) hostname for the control node running Ansible. Enter:

  `sudo hostnamectl set-hostname hol05-ansible-x.handsonlab.local`

  with x your team number.

- At the sudo prompt, enter **********

- You can check the modification using:

  `hostname`

Step 2: Generate an SSH RSA key pair without a passphrase

SSH public key authentication is mandatory for Ansible to control hosts as it allows Ansible to authenticate with the managed nodes without manually entering passwords, which is essential for automation. So, the next step is to generate an SSH key pair using openssh.

- Openssh is installed by default on Rocky Linux so it is not necessary to install it. Just enter:

  ```
  ssh-keygen -t rsa -f ~/.ssh/id_rsa -N ""
  ```

  - **N ""** indicates that the passphrase is an empty string i.e., no passphrase. This is to prevent Ansible from asking for the passphrase when running a playbook.

- You can check that your id_rsa private key really doesn't have a passphrase by trying to display the key details with the following command:

  ```
  ssh-keygen -y -f ~/.ssh/id_rsa
  ```

  If there is no passphrase, your public key will be printed out.

  Note: When you use an SSH private key that is protected by a passphrase, you need to provide a way for Ansible to use that passphrase when it connects to managed nodes. A common method to handle this situation is by using ssh-agent, see https://stackoverflow.com/questions/50277495/how-to-run-an-ansible-playbook-with-a-passphrase-protected-ssh-private-key

  Caution: Always be cautious with the handling of SSH private keys. Without a passphrase, ensure that they are kept in very secure storage and that permissions are set correctly to prevent unauthorized access (chmod 600 ~/.ssh/id_rsa).
  - Read and write for the owner
  - No permissions for group
  - No permissions for others

  Note: A new **.ssh** folder has been created in your home directory, containing both public and private SSH keys.

Step 3: Install ISO creation tools

- Next come the various tools required for ISO creation, which need to be installed. Enter:

  ```
  sudo dnf -y install epel-release
  ```

  The epel-release package contains the Extra Packages for Enterprise Linux (EPEL) repository configuration. The main goal of this extra repository is to provide easy access to packages that are not included in the official repositories of Enterprise Linux distributions.

  ```
  sudo dnf -y install mkisofs
  ```

  mkisofs is a command-line utility that is also used in this project to create ISO 9660 file system images. It is very similar to *genisoimage*, and in fact, *genisoimage* is a fork of *mkisofs* with additional features.

Step 3: Install Ansible and requirements

- Next comes the installation of Ansible and its various requirements. Enter:

  ```
  sudo dnf -y install python3-pip
  ```

  ```
  pip3 install setuptools-rust wheel
  ```

  Instructs pip—the Python package installer—to install two distinct Python packages: *setuptools-rust* and *wheel*. Each of these packages serves a specific purpose in the Python ecosystem, especially when dealing with the installation of packages that include Rust extensions.

- Then to install Ansible, run:

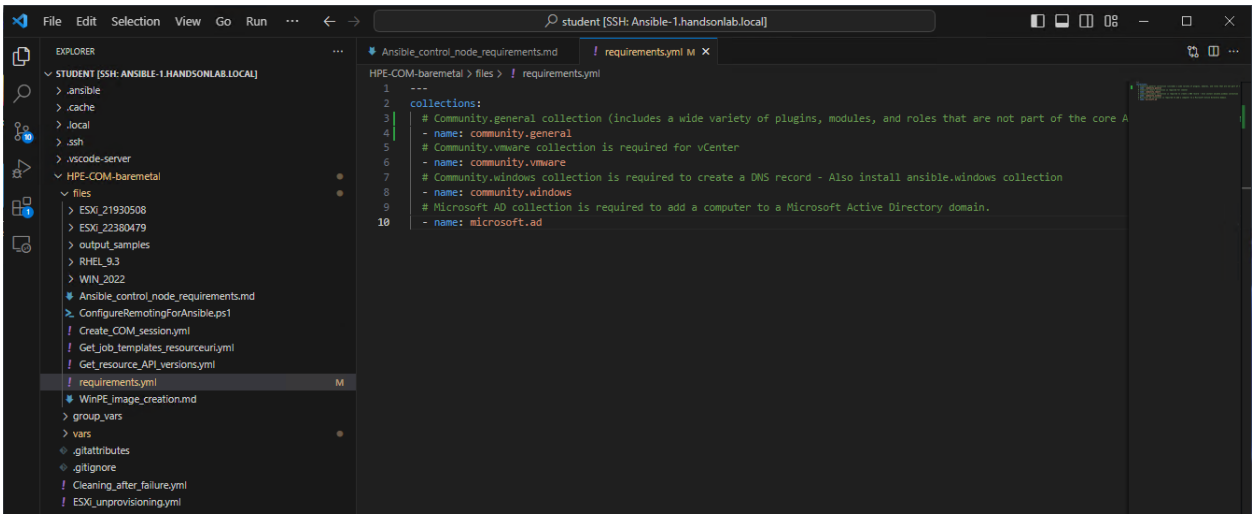
  ```
  pip3 install ansible-core
  ```

Step 4: Install Ansible Collections

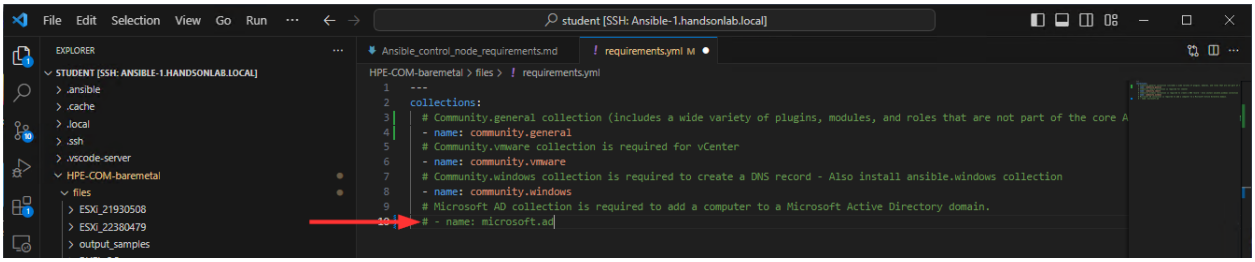
- Next comes the installation of the different Ansible collections that are used in this project, enter:

  `cd HPE-COM-baremetal`

- Next open the file **requirements.yml** located in the **files/** directory:



- This file contains a list of collections to be installed. Optionally, as you are going to provision VMware ESXi only, you can comment out the last line of this file so that the Microsoft.ad collection is not installed (just add a **#** at the beginning of the line or press **CTRL** + **/**):



- Once modified, press **CTRL** + **S** to save the change.

- Then run on the terminal:

  `ansible-galaxy collection install -r files/requirements.yml`

  This command uses a yaml file to provide a list of the various collections to be installed.

Step 5: Install VMware collection requirements

- For the VMware collection, there are a few other steps to run, enter:

  `pip3 install --upgrade pip setuptools`

  `pip3 install --upgrade git+https://github.com/vmware/vsphere-automation-sdk-python.git`

  `pip3 install -r ~/.ansible/collections/ansible_collections/community/vmware/requirements.txt`

Step 6: Install Windows collection requirements

An important task to ensure the smooth operation of this project is the pre-creation of DNS records for all hosts that will be provisioned. For

this reason, each playbook includes a task to create a DNS record on a Windows DNS server defined in the \vars folder. Ansible uses the Windows Remote Management (WinRM) listener to execute commands remotely on Windows, and for this, the *pywinrm* library must be installed.

- To install the *pywinrm* library, enter:

  ```
  pip3 install pywinrm
  ```

  pywinrm is the Python library that allows you to interact with the WinRM service running on the Windows DNS server to perform the DNS record operations.

Step 7: Install json_query filter

The *json_query* filter enables the filtration and transformation of JSON data within Ansible playbooks. This particular filter isn't bundled with the core Ansible package; rather, it comes with the *community.general* collection that has been added through the *requirements.yml* file earlier. However, to function correctly, *json_query* relies on the *jmespath* Python library—an additional dependency that must be installed separately.

- To install the *jmespath* Python library, enter:

  ```
  pip3 install jmespath
  ```

Step 8: Install nginx web service

You need to set up a web server that will host the customized operating system ISO images from which the server that you'll provision will boot from. To achieve this, you'll be using the Linux VM and Nginx will serve as the web server software.

- To install Nginx, enter:

  ```
  sudo dnf -y install nginx
  sudo systemctl enable nginx
  sudo systemctl start nginx
  sudo firewall-cmd --permanent --add-service=http
  sudo firewall-cmd --reload
  ```

- To enable ngnix directory browsing, you must enter:

  ```
  sudo sed -i '0,/server {/s//&\n        autoindex on;/' /etc/nginx/nginx.conf
  ```

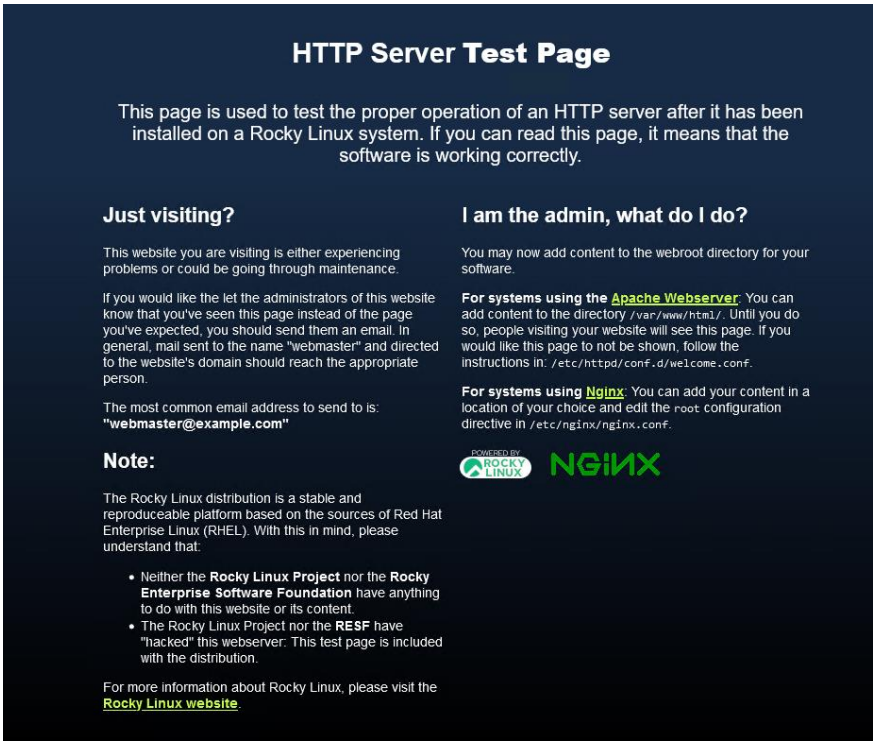
  This command is used to modify the nginx configuration file: /etc/nginx/nginx.conf. It adds a new line containing **autoindex on;** immediately after the first occurrence of the pattern **server {** . This is the only parameter required to activate directory browsing.

- Then to activate the new configuration, restart nginx using:

  ```
  sudo systemctl restart nginx
  ```

  Note: By default, nginx web site is in */usr/share/nginx/html*.

- To test nginx, open a browser and navigate to http://hol05-ansible-X.handsonlab.local with X your team number. You should get:



Step 9: Install rsync

rsync is a utility for efficiently transferring and synchronizing files across computer systems, by using differential data transfer to minimize network usage. It is used in this project to copy ISO image files.

- To install rsync, enter:

  ```
  sudo dnf -y install rsync
  ```

Step 10: Ensure your Ansible Control node time is correct

The last step only involves a critical verification task, ensuring that your Ansible machine's clock is accurately synchronized. This synchronization is essential for time-sensitive playbook operations, such as task monitoring operations, which use time-based activity filtering. This check must be performed before running any playbook associated with this project. If the time isn't right, these playbooks might not work as expected.

Note: It is highly recommended to use NTP (Network Time protocol) for synchronizing the Ansible Control node's time. Although NTP has been configured for this laboratory environment, it is prudent to perform a verification check to ensure accurate synchronization.

- To check the time, let's first change the time zone for Atlanta:

  ```
  export TZ=America/New_York
  ```

- Then to get the time, run:

  ```
  date
  ```

- Make sure the time is correct. If not, ask the instructor for assistance.

This completes the configuration of the Ansible Control Node for VMware ESXi hosts deployment, and we can now move on to the next section.

Task3: Setting up the different variables of the project

The next step is the configuration of the different variables that will be used by this project.

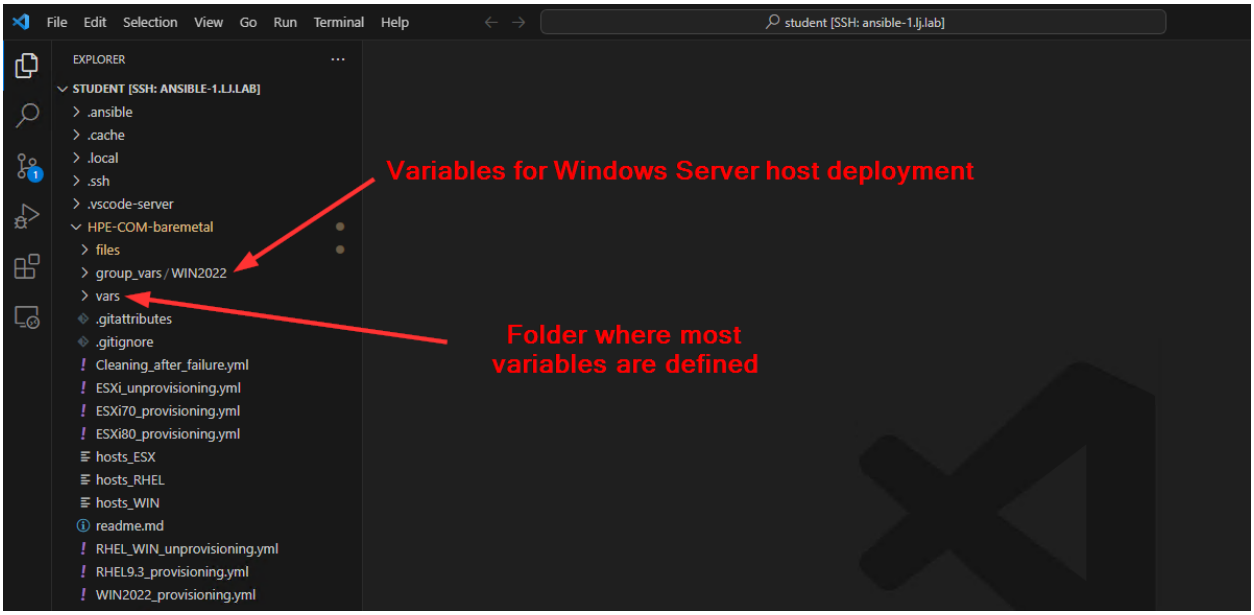
Variables in Ansible are an essential aspect of creating dynamic and reusable playbooks. They allow you to manage configurations and deployments across different environments, hosts, and groups by changing variable values rather than hardcoding information.

There are different types of variables in this project. Here is a list of different types of variables you will encounter in this project:

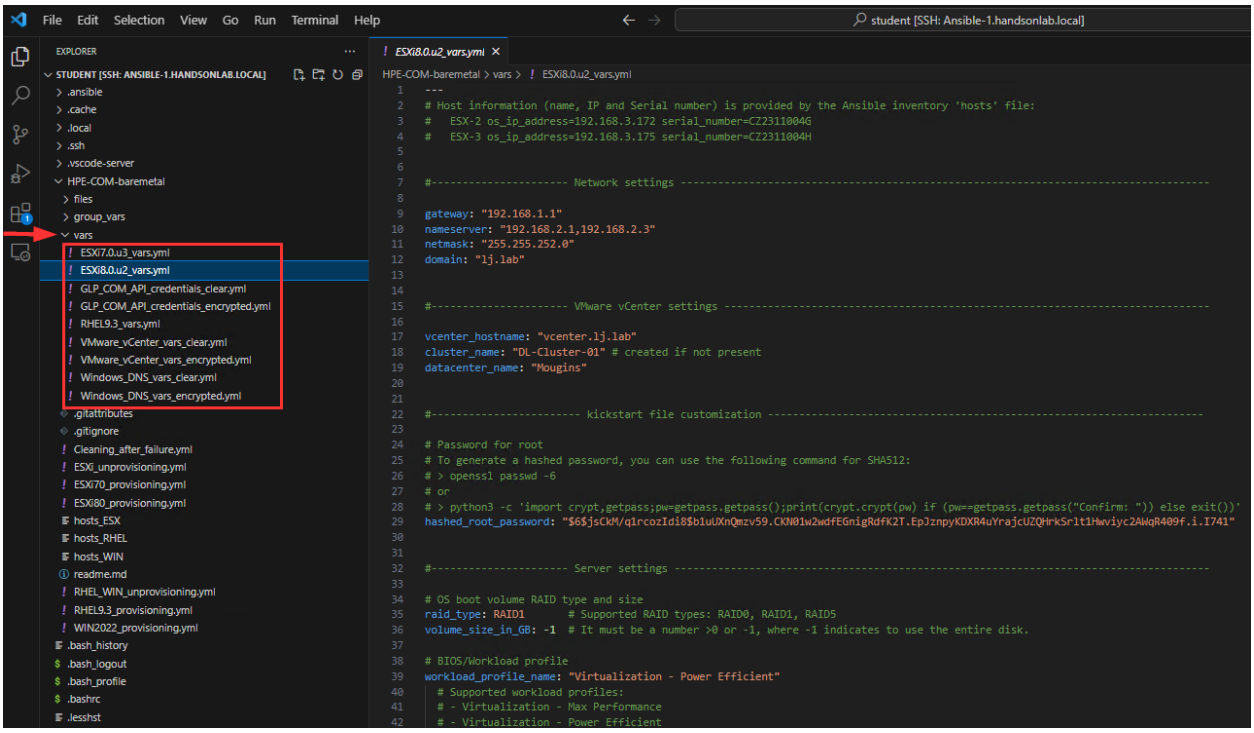
- **Inventory Variables**: Defined within the inventory file for specific hosts or groups.
- **Playbook Variables**: Specified directly within playbooks or included files and tasks.
- **Host Variables**: Pertaining to individual hosts, often defined within the inventory or host_vars directory.
- **Group Variables**: Applicable to groups of hosts, usually placed within the inventory or group_vars directory.
- **Facts** ("ansible_facts"): Gathered by Ansible from the target machines or from Compute Ops Management, containing any sort of information like network interfaces, IP addresses, disk space, etc.
- **Magic Variables**: Special built-in variables such as `hostvars`, `group_names`, `groups`, `inventory_hostname`, `play_hosts`.

Understanding how and when these variables are used will help you manage this Ansible project effectively, ensuring that they can adapt over time to meet new requirements without significant rework.

Variables are mostly all defined in the **/vars** folder of this project.



You'll find in **vars/** different files where the variables are defined for the different operation systems:



If we focus on VMware ESXi provisioning, we have 4 main variable files, some are in clear text, others are encrypted as they contain sensitive data (i.e. passwords, licenses, etc.):

- **ESXi7.0.u2_vars.yml** or **ESXi8.0.u2_vars.yml**: store the various variables necessary to customize the ESXi kickstart, to provision ESXi in a cluster on a vCenter server. Each file corresponds to a ESXi version and includes:
  - Network settings, VMware vCenter settings, Storage settings (OS RAID and size), BIOS/Workload profile
  - Source ISO image to use for the provisioning.
  - Firmware bundle to use to update the server once the provisioning is complete, and update policies to define the downgrade and software and drivers installation policies.
  - Kickstart file settings.

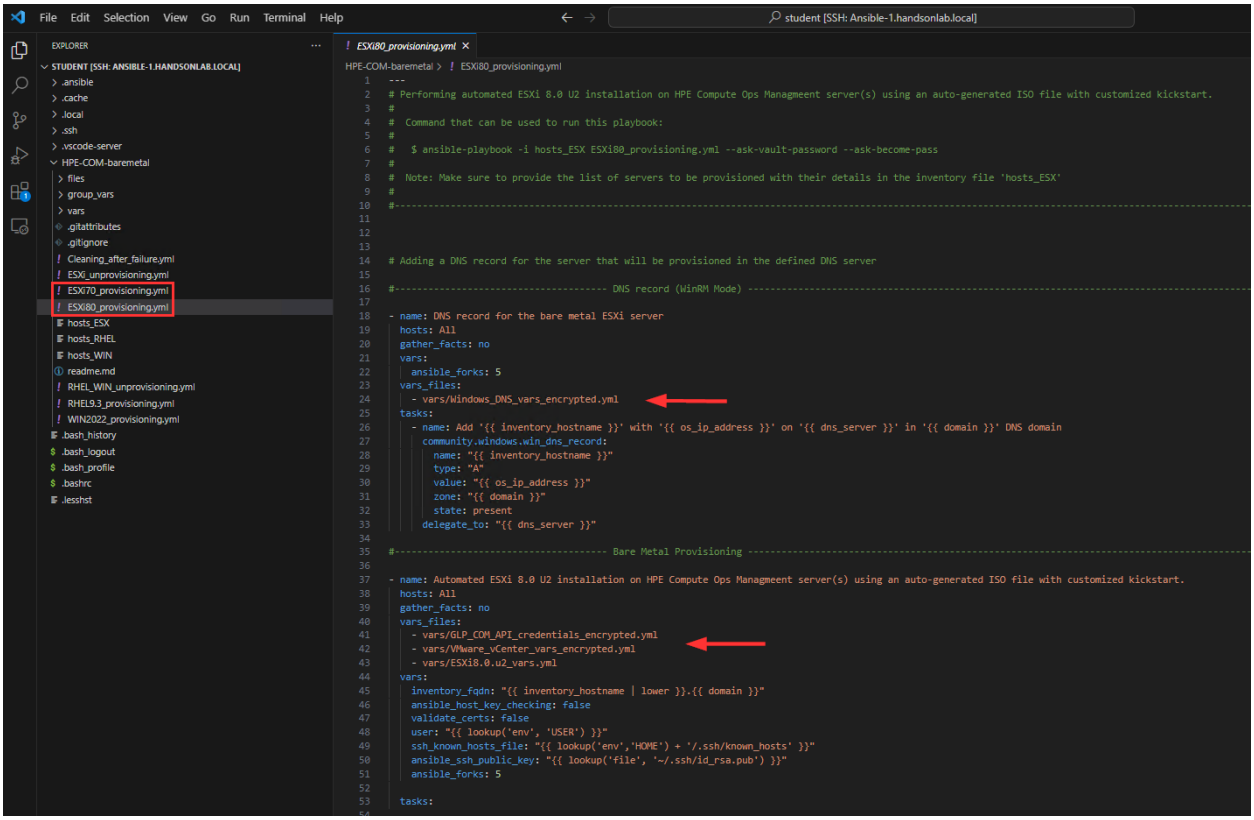
- **GLP_COM_API_credentials_encrypted.yml**: stores the API client credentials that are used to create a session with the HPE GreenLake for Compute Ops Management API. Note that the encrypted file has been generated using the Ansible vault command to secure the credentials. The *GLP_COM_API_credentials_clear.yml* file illustrates the contents of the encrypted file to be supplied.

  Note: You will see later in the lab how to encrypt your data with Ansible but if you need to learn more about Ansible vault, see **Protecting sensitive data with Ansible vault**

- **VMware_vCenter_vars_encrypted.yml**: stores the VMware vCenter encrypted credentials, the ESXi license to be assigned to the hosts and the password of the ESXi root user. The clear content is illustrated in *VMware_vCenter_vars_clear.yml*

- **Windows_DNS_vars_encrypted.yml**: stores the Windows DNS domain, DNS server name and WinRM information. WinRM is used by Ansible to create DNS records of new provisioned hosts on a Windows DNS server. The clear content is illustrated in Windows_*DNS*_vars_clear.yml.

These variable files are all defined in the main Ansible playbooks **ESXi70_provisioning.yml** and **ESXi80_provisioning.yml** for provisioning VMware ESXi 7 and 8 both located at the root of this project.



For this lab, you need to configure the variables to suit the lab environment and your team number.

Step 1: Set Windows_DNS_vars_encrypted.yml

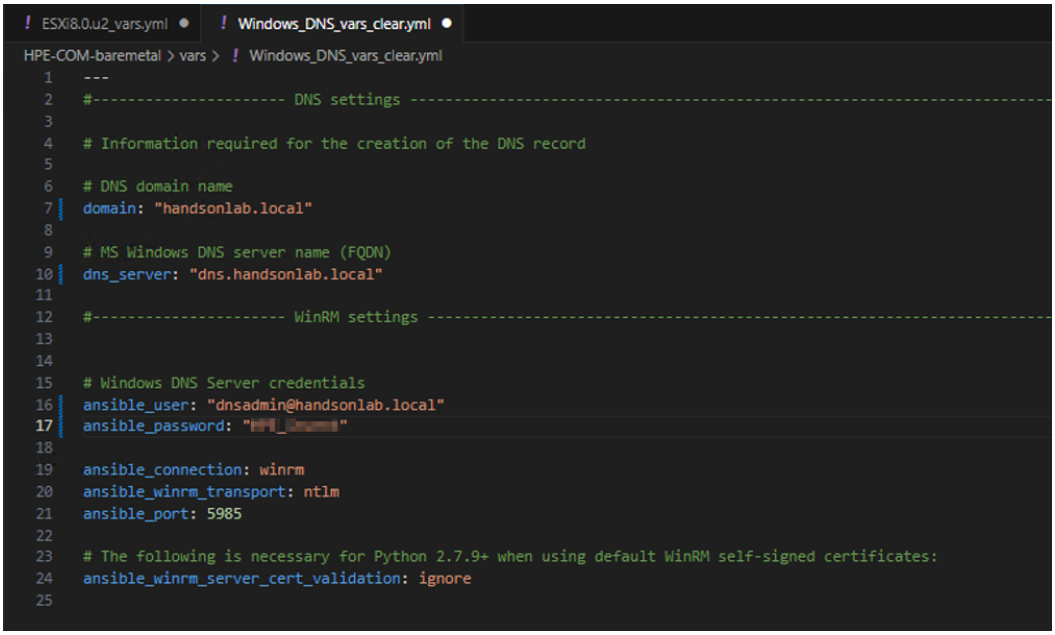
- For the first variable file **vars/Windows_DNS_vars_encrypted.yml** defined in this playbook, you need to generate your own encrypted data as it contains sensitive information. You can use the clear file **vars/Windows_DNS_vars_clear.yml** as a template to create first your clear data. Use the following values:

Variable name	Value
domain	handsonlab.local
dns_server	dns.handsonlab.local
ansible_user	dnsadmin@handsonlab.local
ansible_password	xxxxxxxxxx

  Important note: Ensure that your value is consistently enclosed within quotation marks, ensuring there are no spaces between the value and the adjacent opening and closing quotation marks.

  Note: **dnsadmin@handsonlab.local** is a member of the **Remote Management Users** security group (allows connection to remote Windows DNS server via WinRM) and member of the **DNSAdmins** security group (allows DNS records to be updated).

- You can keep the other default values.



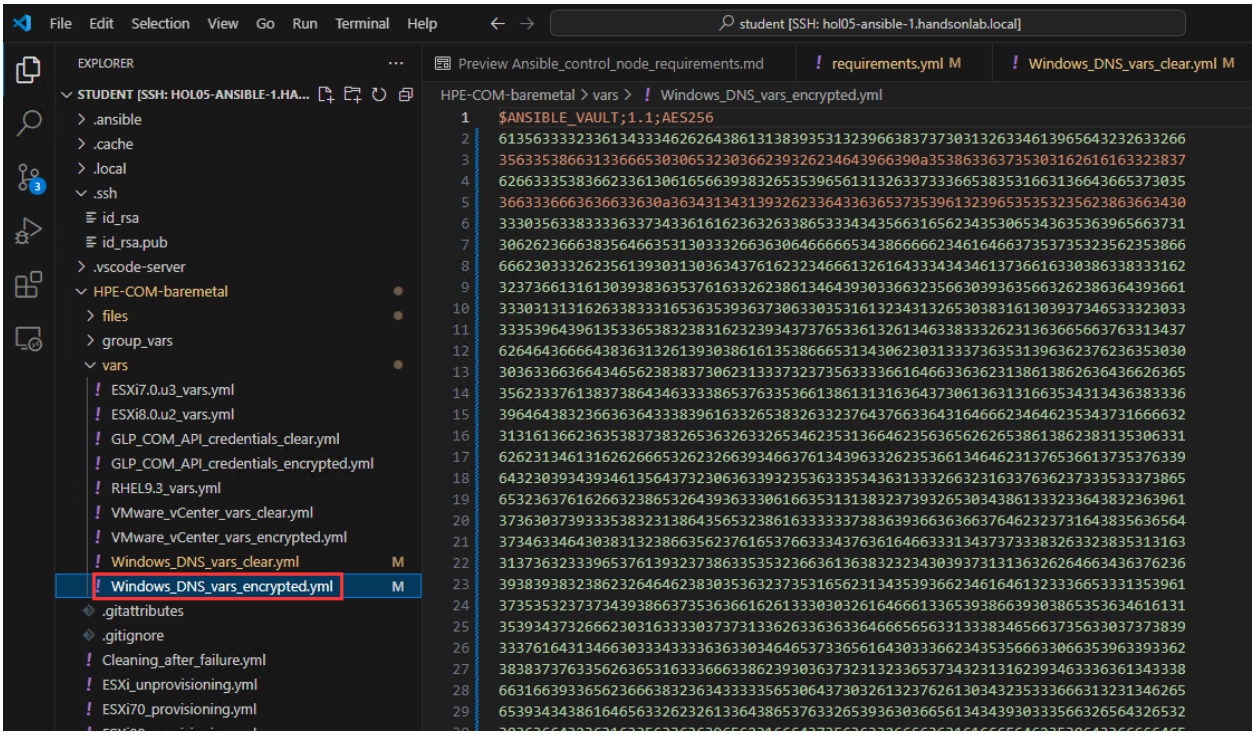
- Make sure your changes have been saved by pressing **CTRL** + **S**.

- Then to generate an encrypted vault file with Ansible, you must use the **ansible-vault** command-line tool that comes bundled with Ansible. From the terminal, run:

  ```
  ansible-vault encrypt vars/Windows_DNS_vars_clear.yml --output vars/Windows_DNS_vars_encrypted.yml
  ```

- After running this command, you will be prompted to enter a vault password. Enter a password of your choice. Once you confirm the password, the command will generate a new encrypted file **Windows_DNS_vars_encrypted.yml** with your sensitive data.

- You can click on the new **Windows_DNS_vars_encrypted.yml** file to check that the data is unreadable in VS Code:



- To check the operation, you can open the encrypted file using:

  `ansible-vault view vars/Windows_DNS_vars_encrypted.yml`

- At the vault password prompt, enter the password you set previously and check that the contents are correct:

```
[student@ansible HPE-COM-baremetal]$ ansible-vault encrypt vars/Windows_DNS_vars_clear.yml --output vars/Windows_DNS_vars_encrypted.yml
New Vault password:
Confirm New Vault password:
Encryption successful
[student@ansible HPE-COM-baremetal]$ ansible-vault view vars/Windows_DNS_vars_encrypted.yml
Vault password:
---
#---------------------- DNS settings --------------------------------------------------------------------------------

# Information required for the creation of the DNS record

# DNS domain name
domain: "handsonlab.local"

# MS Windows DNS server name (FQDN)
dns_server: "dns.handsonlab.local"

#---------------------- WinRM settings ------------------------------------------------------------------------------


# Windows DNS Server credentials
ansible_user: "dnsadmin@handsonlab.local"
ansible_password: "[redacted]"
:...skipping...
```

Step 2: Set GLP_COM_API_credentials_encrypted.yml

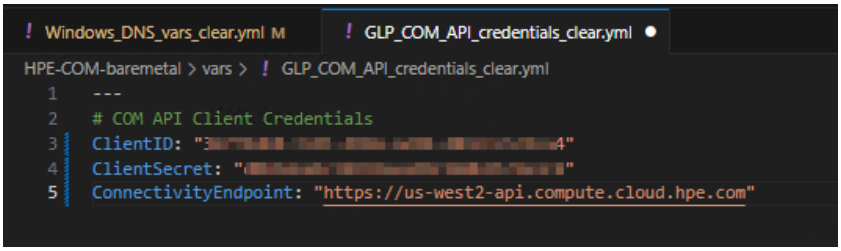
The second variable file defined in this playbook contains the client credentials for the Compute Ops Management API, which are also highly sensitive data and therefore need to be vaulted.

- Open the file **vars/ GLP_COM_API_credentials_clear.yml** and use the following values according to your team number:

Variable name	Value
ClientID and ClientSecret	For Team 1 to Team 5: ClientID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ClientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxx For Team 6 to Team 10 ClientID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ClientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxx For Team 11 to Team 15 ClientID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ClientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxx For Team 16 to Team 20 ClientID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ClientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxx For Team 21 to Team 25 ClientID: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ClientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxx
ConnectivityEndpoint	https://us-west2-api.compute.cloud.hpe.com

Note: These API client credentials have been issued by the instructor through the HPE GreenLake platform which are indispensable for enabling this GitHub project's interaction with the COM API.

- Make sure your changes have been saved by pressing **CTRL** + **S**.



- Then to generate the encrypted vault file, run:

```
ansible-vault encrypt vars/GLP_COM_API_credentials_clear.yml --output
vars/GLP_COM_API_credentials_encrypted.yml
```

Be sure to use the same vault password as before.

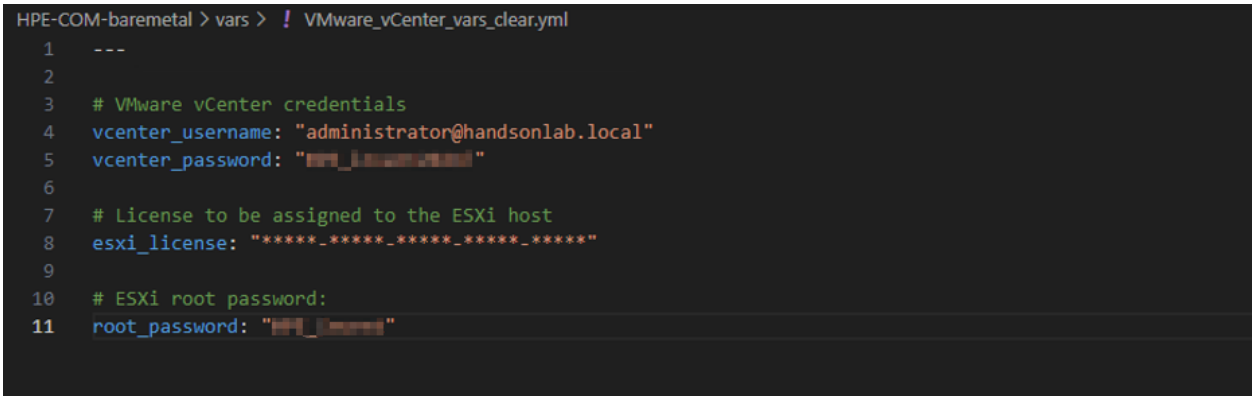
Step 3: Set VMware_vCenter_vars_encrypted.yml

The third variable file contains the VMware vCenter sensitive data and must also be vaulted.

- Open **vars/VMware_vCenter_vars_clear.yml** and use:

Variable name	Value
vcenter_username	administrator@handsonlab.local
vcenter_password	xxxxxxxxxxxxxxxxxxxxxxxxxxxx
esxi_license	Keep it as is! A temporary evaluation license will be assigned automatically.
root_password	xxxxxxxxxxxxxxxxxxxxxxxxxxxx

- Make sure your changes have been saved by pressing **CTRL** + **S**.



- Then run:

```
ansible-vault encrypt vars/VMware_vCenter_vars_clear.yml --output vars/VMware_vCenter_vars_encrypted.yml
```

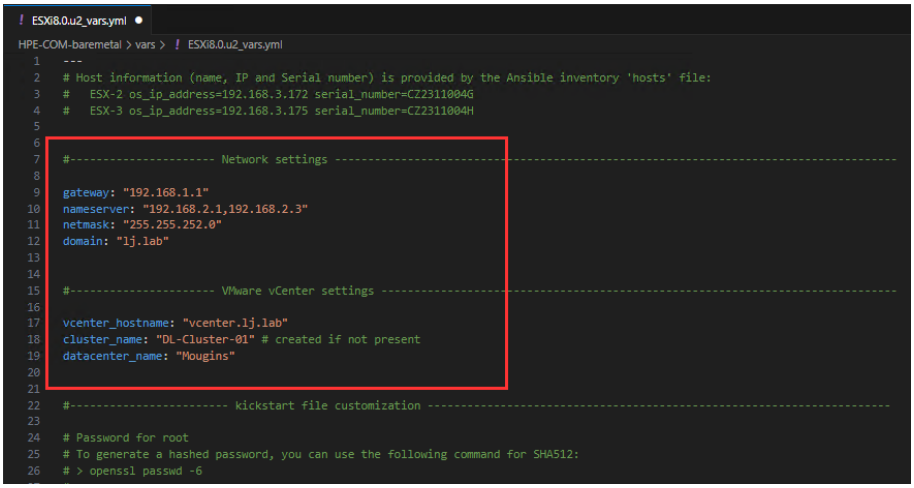
Again, be sure to use the same vault password as earlier.

Step 4: Set ESXi8.0.u2_vars.yml

- Open the last variable file, **ESXi8.0.u2_vars.yml**.

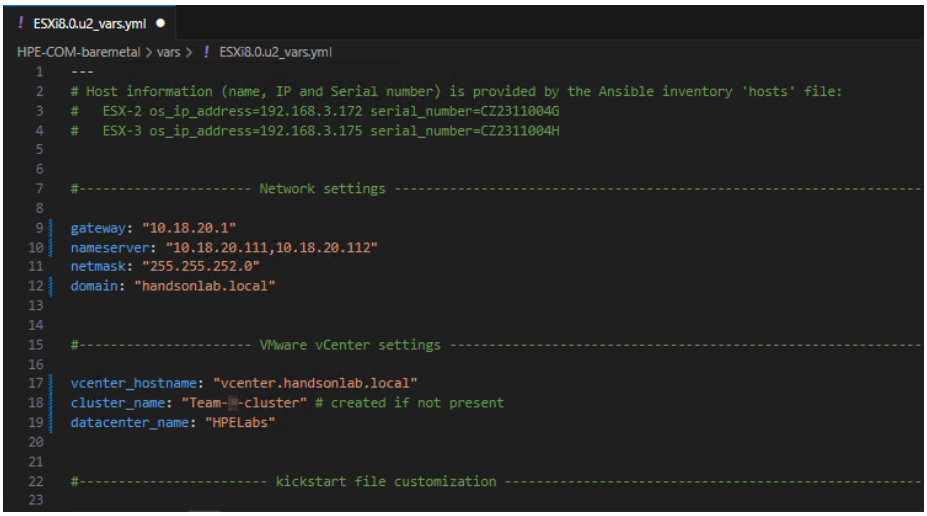
This file contains all settings related to ESXi, HPE GreenLake for Compute Ops Management, the ISO image to use, etc.

- The first part includes the network and VMware vCenter settings:



- Replace those with the following values:

Variable name	Value
gateway	10.18.20.1
nameserver	10.18.20.111,10.18.20.112
netmask	255.255.252.0
domain	handsonlab.local
vcenter_hostname	vcenter.handsonlab.local
cluster_name	Team-X-cluster (with X your team number)
datacenter_name	HPELabs



- Next, is the ESXi root password that needs to be defined. Hashing the root password here is crucial for security reasons. A plain text root password would pose a significant security risk. If someone were to gain access to your variable file or to the kickstart file where this variable is used, they could easily read and use that plain text password to compromise your system. Hashing obscures the actual password, so even if someone accesses the file, they won't be able to turn that hash back into its original data and eventually connect to the ESXi servers.

  Note: In Unix-type systems, passwords are stored and managed using their hashed value, which explains why hashing is used in kickstarts.

  To hash your password, enter from your terminal:

  `openssl passwd -6`

  The **-6** option specifies that you want to generate a password using the SHA-512 hashing algorithm, which is currently considered a secure method of creating hashed passwords.

- Then type twice **xxxxxxxxx**, the ESXi root password that was defined earlier in **VMware_vCenter_vars_encrypted.yml**.

  This command returns a long string of characters, this is the hash of your password that will be used by the OS to create the root user.

- Copy the hash string.

  Warning: Make sure you select the entire string to avoid any problems connecting to ESXi once the OS has been installed!

```
[student@ansible-1 HPE-COM-baremetal]$ openssl passwd -6
Password:
Verifying - Password:
$6$jP7LoKWbGmhMdBJ7$3kNrm5WuAydyvNfHQCL.AzNzcm4DwsVwz5ZXLYT05eDOKGg6fKL3OQ/O1BgnLen13BE.TQ6hEzEIUFDXYvhfW.
[student@ansible-1 HPE-COM-baremetal]$
```

  Note: To copy a text under a Linux terminal, you generally select the text using your mouse and then right-click.

- And paste it to the **hashed_root_password** variable value in **ESXi8.0.u2_vars.yml**.

```
#------------------------ kickstart file customization ---------------------------------------------------------------

# Password for root
# To generate a hashed password, you can use the following command for SHA512:
# > openssl passwd -6
# or
# > python3 -c 'import crypt,getpass;pw=getpass.getpass();print(crypt.crypt(pw) if (pw==getpass.getpass("Confirm: ")) else exit())'
hashed_root_password: "$6$jP7LoKWbGmhMdBJ7$3kNrm5WuAydyvNfHQCL.AzNzcm4DwsVwz5ZXLYT05eDOKGg6fKL3OQ/O1BgnLen13BE.TQ6hEzEIUFDXYvhfW."
```

- Next, you will find sections for the HPE GreenLake for Compute Ops Management parameters, such as the type of RAID you wish to define for the OS volume (RAID 0, 1 or 5), but also the size of the OS volume, the bios workload profile to be defined, the firmware bundles to be used, the name of the server group that will be defined at the end, the firmware downgrade policy and whether you wish to update at the end only the firmware or also the HPE drivers and software:

```
#---------------------- Server settings ----------------------------------------------------------------------------

# OS boot volume RAID type and size
raid_type: RAID1       # Supported RAID types: RAID0, RAID1, RAID5
volume_size_in_GB: -1  # It must be a number >0 or -1, where -1 indicates to use the entire disk.

# BIOS/Workload profile
workload_profile_name: "Virtualization - Power Efficient"
  # Supported workload profiles:
  # - Virtualization - Max Performance
  # - Virtualization - Power Efficient
  # - General Peak Frequency Compute
  # - General Power Efficient Compute
  # - General Throughput Compute
  # - Low Latency


#---------------------- COM server group settings ------------------------------------------------------------------

# Name of the definitive server group where to place the host at the end of provisioning (created if not already present)
server_group_name: "ESXi_group"
# Name of the firmware server setting to be defined in definitive server group (created if not already present)
firmware_server_setting_name: "ESXi firmware baseline"

# Firmware baselines to be defined in the firmware server setting of the definitive server group
#  Note: it is mandatory to define at least one firmware baseline for Gen10/Gen10+ servers and one for Gen11 servers.
Gen10_10p_firmware_bundle_name: "2023.09.00.00"  # "2023.03.00.00"
Gen10_10p_patch_name_associated_with_firmware_bundle: "2023.09.00.01" # "2023.03.00.03"
Gen11_firmware_bundle_name: "2023.10.00.00" # "2023.04.00.00"
Gen11_patch_name_associated_with_firmware_bundle: "" #"2023.04.00.04"

# Downgrade components policy to match baseline (boolean)
# If "true", any component version higher than the baseline will be downgraded to match the baseline.
firmware_downgrade_policy: true

# Install HPE drivers and software policy in the definitive server group
# If "true", firmware updates will include HPE drivers and software.
install_SWDrivers_policy: false

#---------------------- ISO settings -------------------------------------------------------------------------------
```

- You can keep the same existing values, except for:

Variable name	Value
server_group_name	Team-X-ESXi_group (with X your team number)
firmware_server_setting_name	Team-X-ESXi-firmware-baseline (with X your team number)
Gen10_10p_firmware_bundle_name	2023.03.00.00
Gen10_10p_patch_name_associated_with_firmware_bundle	None, so just put two quotation marks: ""
Gen11_firmware_bundle_name	2023.04.00.00
Gen11_patch_name_associated_with_firmware_bundle	2023.04.00.05
firmware_downgrade_policy	false
install_SWDrivers_policy	False (should already be defined)

Note: When dealing with Boolean values, such as **false** or **true**, do not enclose them in quotation marks within Ansible configurations.

```
#---------------------- COM server group settings ------------------------------------------------------------------------

# Name of the definitive server group where to place the host at the end of provisioning (created if not already present)
server_group_name: "Team-1-ESXi_group"
# Name of the firmware server setting to be defined in definitive server group (created if not already present)
firmware_server_setting_name: "Team-1-ESXi-firmware-baseline"

# Firmware baselines to be defined in the firmware server setting of the definitive server group
#  Note: it is mandatory to define at least one firmware baseline for Gen10/Gen10+ servers and one for Gen11 servers.
Gen10_10p_firmware_bundle_name: "2023.09.00.00"  # "2023.03.00.00"
Gen10_10p_patch_name_associated_with_firmware_bundle: "" # "2023.03.00.03"
Gen11_firmware_bundle_name: "2023.04.00.00"
Gen11_patch_name_associated_with_firmware_bundle: "2023.04.00.05"

# Downgrade components policy to match baseline (boolean)
# If "true", any component version higher than the baseline will be downgraded to match the baseline.
firmware_downgrade_policy: false

# Install HPE drivers and software policy in the definitive server group
# If "true", firmware updates will include HPE drivers and software.
install_SWDrivers_policy: false
```

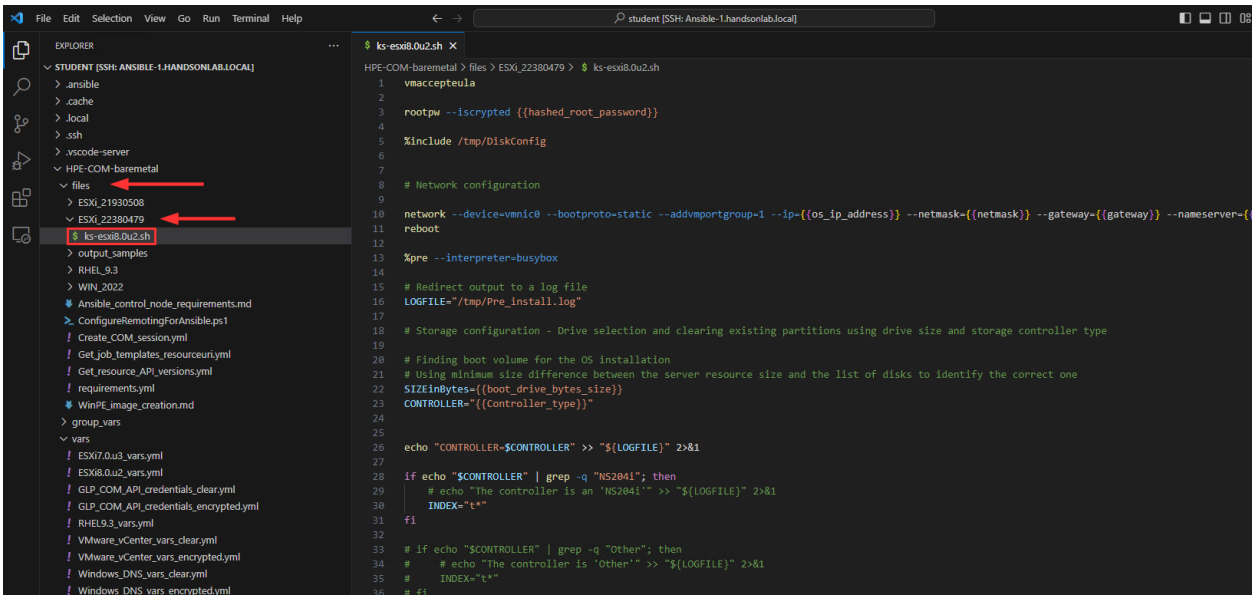
- Next is the ISO settings, this section defined the URL where the ESXi 8 ISO image can be found and its name, enter:

Variable name	Value
src_iso_url	http://webserver.handsonlab.local/deployment
src_iso_file	It must be the same file as the one already defined, which you can check by browsing the URL above.
src_iso_directory	Keep the same
staging_directory	Keep the same

```
#----------------------- ISO settings ---------------------------------------------------------------------------------

# URL where the ISO image can be found
src_iso_url: "http://webserver.handsonlab.local/deployment"
# ISO file name
src_iso_file: "VMware-ESXi-8.0.2-22380479-HPE-802.0.0.11.4.0.14-Sep2023.iso"

# Directory on the Ansible control node where the source ISO will be copied (will be created if not already present).
# Note: You must define a directory in your user home directory using {{ lookup('env','HOME') }}.
src_iso_directory: "{{ lookup('env','HOME') }}/ISOs/esxiisosrc"

# Directory on the Ansible control node to stage all files to build and generate the new ISO image with the custom kickstart (will be
# Note: You must define a directory in your user home directory using {{ lookup('env','HOME') }}.
staging_directory: "{{ lookup('env','HOME') }}/staging"


#----------------------- Kickstart file -------------------------------------------------------------------------------

# Folder located in /files to store the kickstart file
# Recommended to use the template name: ESXi_<build>. (<build> can be found in boot.cfg at the root of the iso)
esxi_build: "ESXi_22380479"

# Name of the kickstart file
kickstart: "ks-esxi8.0u2.sh"
```

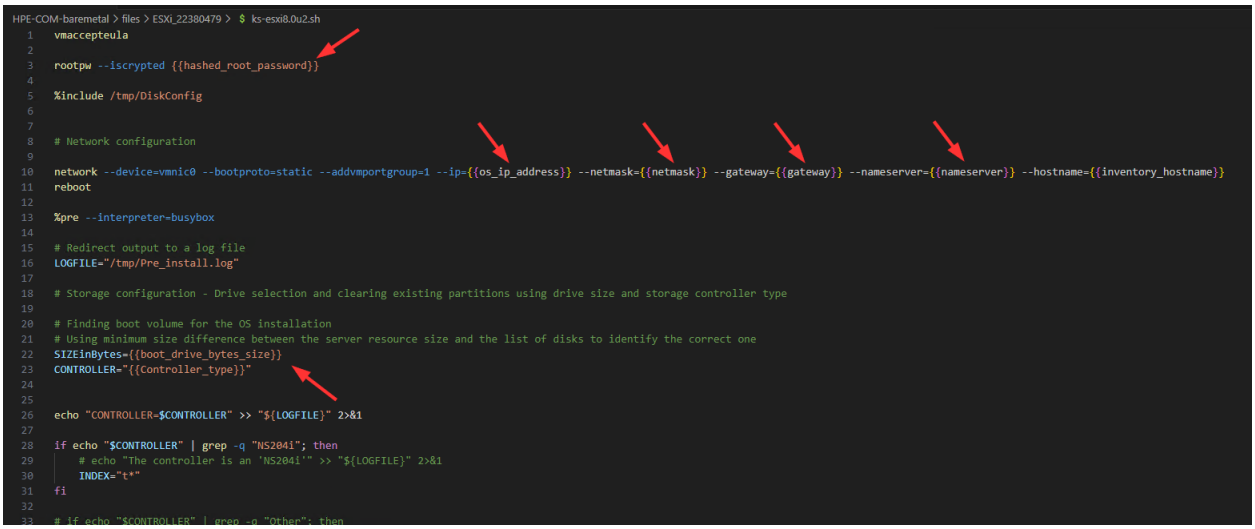
- And finally, the kickstart file information, this section defined the location of the ESXi kickstart file. This is the file that is used to automate the installation process of ESXi, and can include directives for installation or upgrade, root password configuration, network settings, disk partitioning, and post-installation scripts...

The kickstart file for ESXi 8.0u2 is in **files/ESXi_22380479**:



The kickstart includes some logic to ensure that the target disk for OS installation is correctly identified to avoid data loss or other issues when several volumes are presented to a host. It also includes directives for post-installation scripts, which involve renaming the host and incorporating the Ansible SSH public key into the host's authorized keys file, thereby enabling passwordless SSH authentication by Ansible to ensure a secure and seamless automation of operations.

There is nothing to modify in this kickstart file as it uses Ansible Jinja2 template variables defined with the **{{ … }}** syntax for variable substitution. Basically, wherever you see {{ }} in this file, it's a place where dynamic content will be inserted (using the variables you have previously set like for *{{hashed_root_password}}*) or some computation based on variables will occur at runtime (like for *{{boot_drive_bytes_size}}* and *{{Controller_type}}* collected from Compute Ops Management during the playbook execution):



- This completes the configuration of the different variables of this project.

- Make sure that all your files are saved, and before moving on to the next section, diligently verify all settings AGAIN. Ignoring this step may lead to provisioning failures that can result in extensive time spent on troubleshooting later.

  Note: The most common error is adding spaces between your value and the incoming and closing quotation marks! So ensure this is not the case.

- As a final check, make sure that all your encrypted files contain the correct settings by using:

  ```
  ansible-vault view vars/GLP_COM_API_credentials_encrypted.yml
  ansible-vault view vars/VMware_vCenter_vars_encrypted.yml
  ansible-vault view vars/Windows_DNS_vars_encrypted.yml
  ```

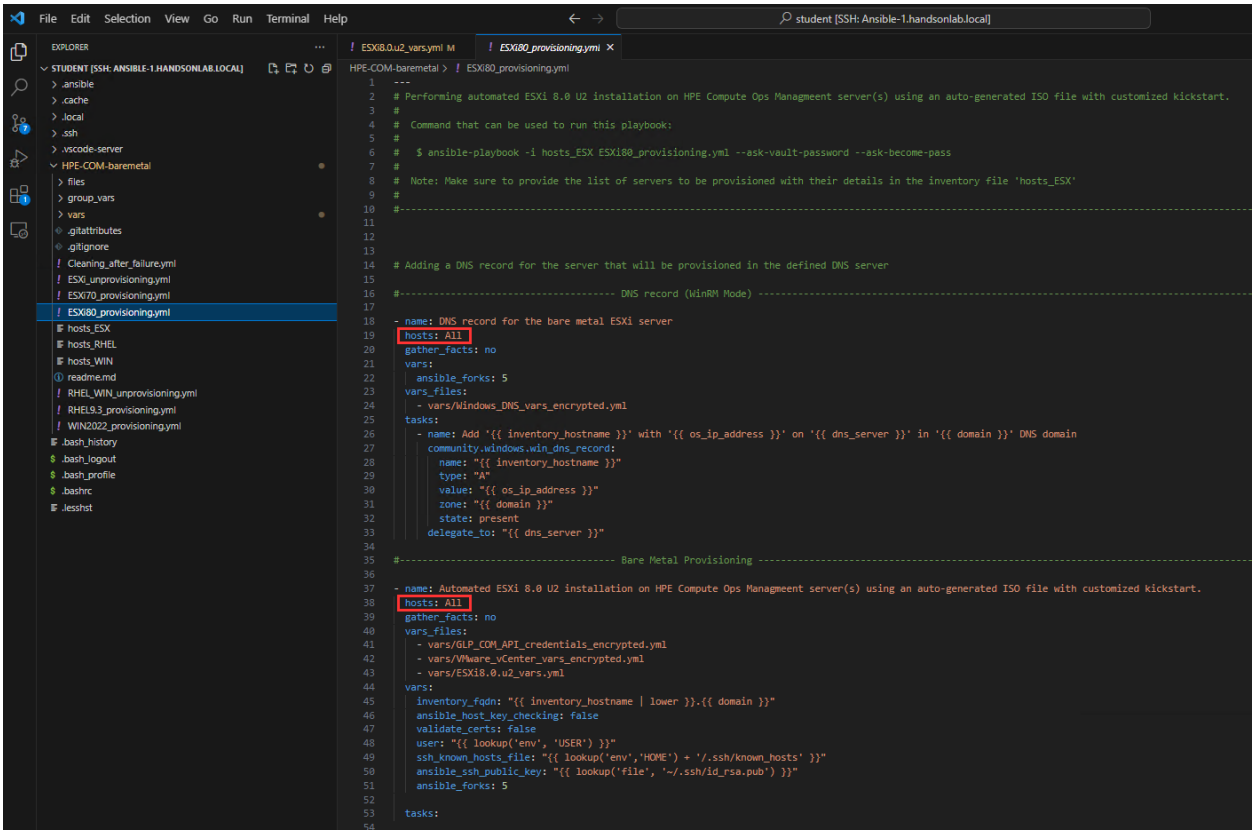
Once everything is checked and correct, you can move on to the next section.

Task 4: Configuration of the inventory file

The playbooks to provision VMware ESXi, Windows Server and RHEL are located at the project's root directory. Two are available for VMware ESXi, one for ESXi 7.0 and one for 8.0.

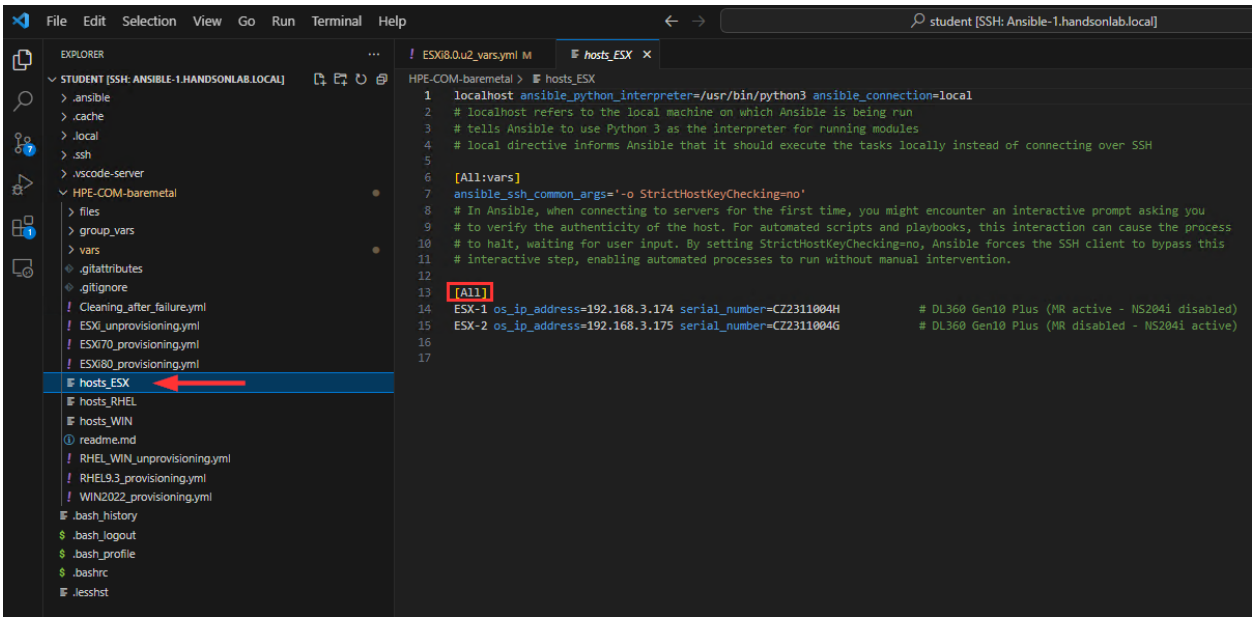
- Open **ESXi80_provisioning.yml,** the one built for ESXi 8.0.

  An Ansible playbook is a YAML file that defines automation tasks in a sequence and specifies the hosts (inventory) on which to run those tasks. Note that in this playbook, **hosts: All** is defined for the first two groups of tasks:



  This *hosts* attribute specifies the group of hosts or servers on which the playbook tasks should be performed.

- Now, open the **hosts_ESX** inventory file, also found at the root of the project. This file is given as an example for use with the two ESXi provisioning playbooks. You'll notice that the same **All** group is defined:



  Ansible utilizes inventory files to list and organize the servers that Ansible will manage. These files provide Ansible with information regarding accessible machines, as well as any relevant variables specific to groups or individual hosts.

  Within the scope of this project, we leverage these inventory files to specify the target servers for deployment.

- So, the next step of the lab is to modify this inventory file example with information about the target server you're going to provision. Only three parameters are required:
  - **Hostname** –the OS hostname that will be configured (cautious: it is not the FQDN!).
  - **IP address** –the IP address that will be assigned to the operating system.
  - **Serial number** –the serial number of the server to be provisioned.

- For the hostname, the first parameter of the line, enter: **ESX-TeamX** (with X your team number)

- For **os_ip_address**, use the following table to carefully select the IP address assigned to your team:

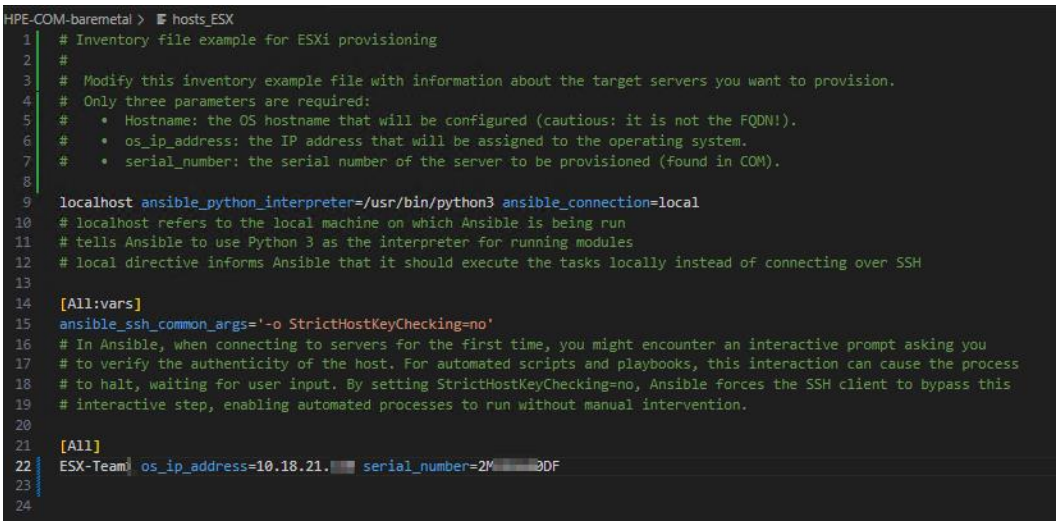
Team	IP address
Team1	10.18.21.140
Team2	10.18.21.141
Team3	10.18.21.142
Team4	10.18.21.143
Team5	10.18.21.144
Team6	10.18.21.145
Team7	10.18.21.146
Team8	10.18.21.147
Team9	10.18.21.148
Team10	10.18.21.149
Team11	10.18.21.150
Team12	10.18.21.151
Team13	10.18.21.152
Team14	10.18.21.153
Team15	10.18.21.154
Team16	10.18.21.155
Team17	10.18.21.156
Team18	10.18.21.157
Team19	10.18.21.158
Team20	10.18.21.159
Team21	10.18.21.160
Team22	10.18.21.161
Team23	10.18.21.162
Team24	10.18.21.163
Team25	10.18.21.164

- For **serial_number**, use the following table to carefully select the serial number of the server assigned to your team:

Team	Serial number
Team1	2M294600DF
Team2	2M294600BG
Team3	2M294600DH
Team4	2M2946009Z
Team5	2M294600B0
Team6	2M294600B1
Team7	2M294600DJ
Team8	2M294600DG
Team9	2M294600CP
Team10	2M294600BH
Team11	2M294600D6
Team12	2M294600DC
Team13	2M294600BJ
Team14	2M294600C4
Team15	2M294600HJ
Team16	2M293800KC
Team17	MXQ3500KS4
Team18	MXQ3500KS3
Team19	MXQ3500KRY
Team20	MXQ3500KS0
Team21	MXQ3500KS9
Team22	MXQ3500KS2
Team23	MXQ3500KS1
Team24	MXQ3500KRZ
Team25	MXQ3500KS7

- Copy the serial number and paste it to the inventory file.

- Then delete the second target server available in this inventory example, as we want you to deploy just one server. You should end up with the following content:



- If all is as above, save your file using CTRL + S

Congratulations! You have successfully completed the configuration phase. You may now proceed to the connection to the HPE GreenLake platform.

Task 5: Playbook execution

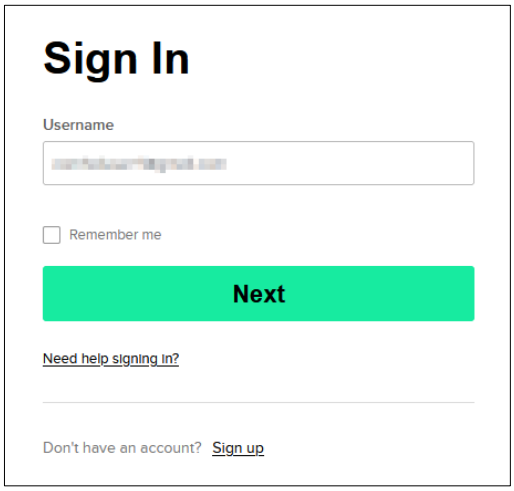
Everything is now prepared for the initiation of the provisioning process. I would characterize the upcoming step as "un jeu d'enfant," a French idiom, or in English, "a piece of cake." This suggests that the more challenging portion is behind you, and what's ahead should be straightforward and effortless. However, this smooth provisioning hinges on the accuracy of your parameters—if there's an oversight or error in the setup, you'll likely notice it promptly.

Step 1: Connection to the HPE GreenLake platform

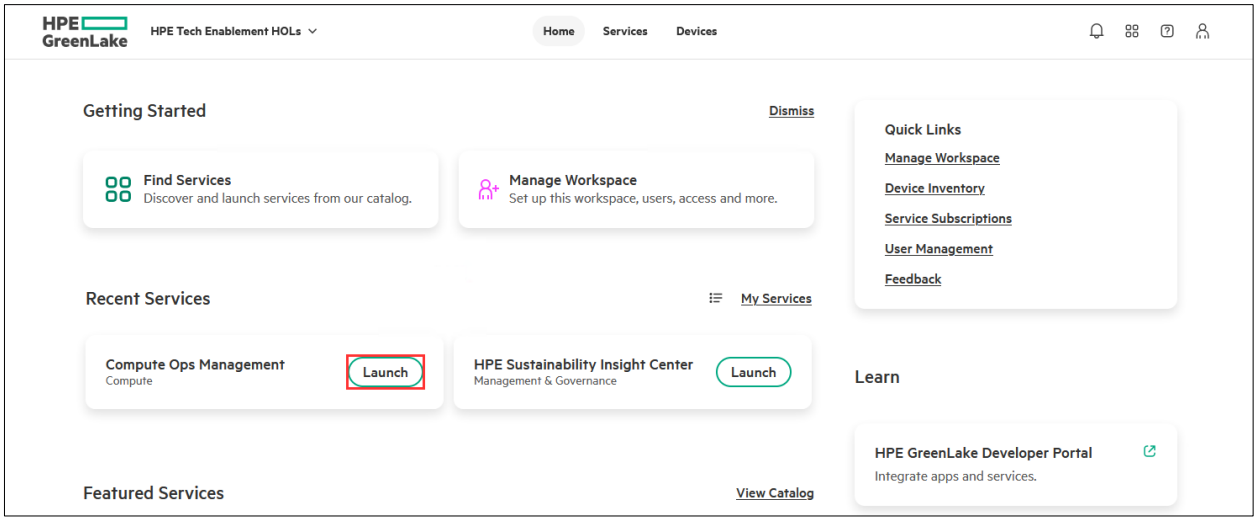
Before we start the provisioning, let's get ready to track the progress of the different tasks on the HPE GreenLake platform.

- Open a browser and go to https://common.cloud.hpe.com
- Login with your provided credentials

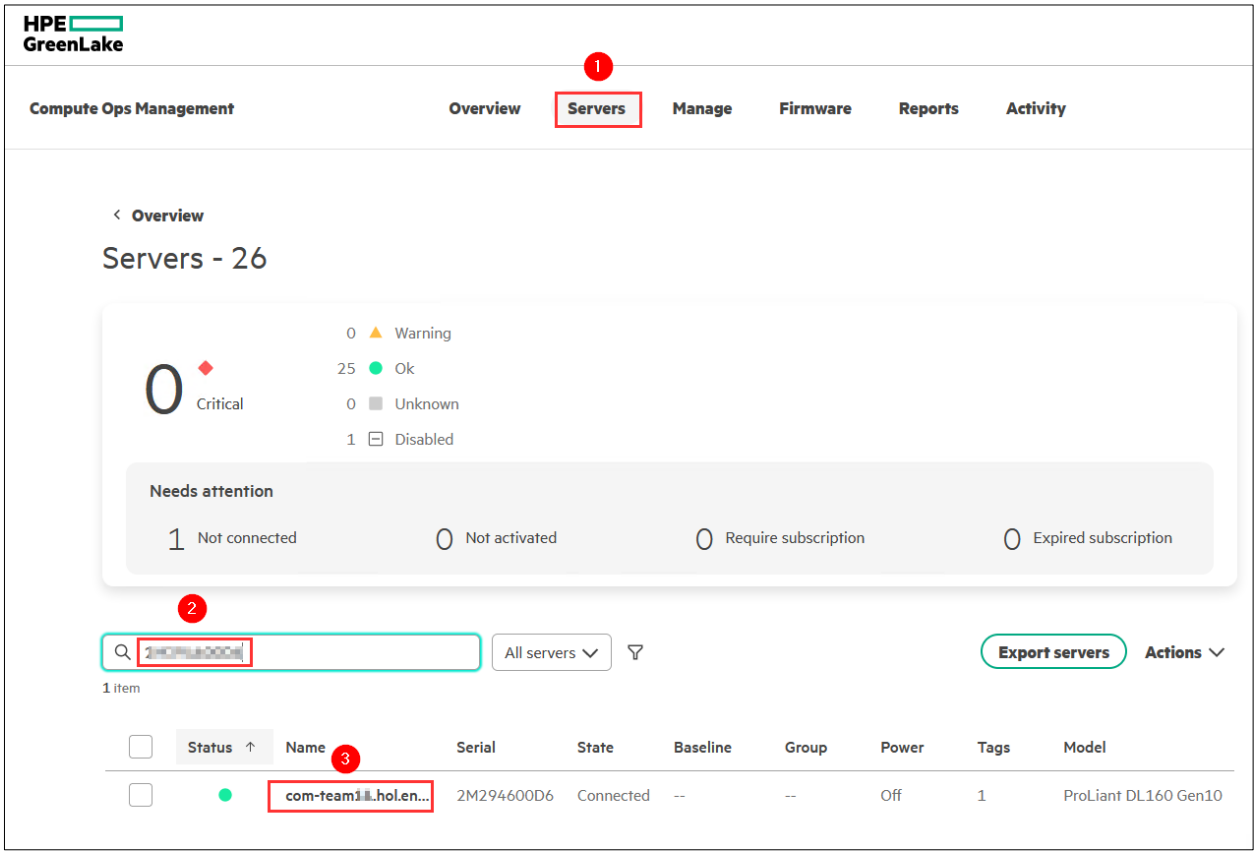
  Important note: Your credentials and team assignments can be found on the sheet of paper issued to you by your lab supervisor. Please return the sheet to your supervisor when you have completed this lab.



- Then click on **Compute Ops Management**:



- Next, click on **Servers**, then in the search tool, input the serial number of the server allocated to your team:



- Then click on the server's name.

  Note: All servers to be provisioned must be connected to the HPE GreenLake platform (i.e. the status must be "connected"). Their power status can be "on" or "off" and attached to a group or not.

Step 2: Executing the Ansible playbook

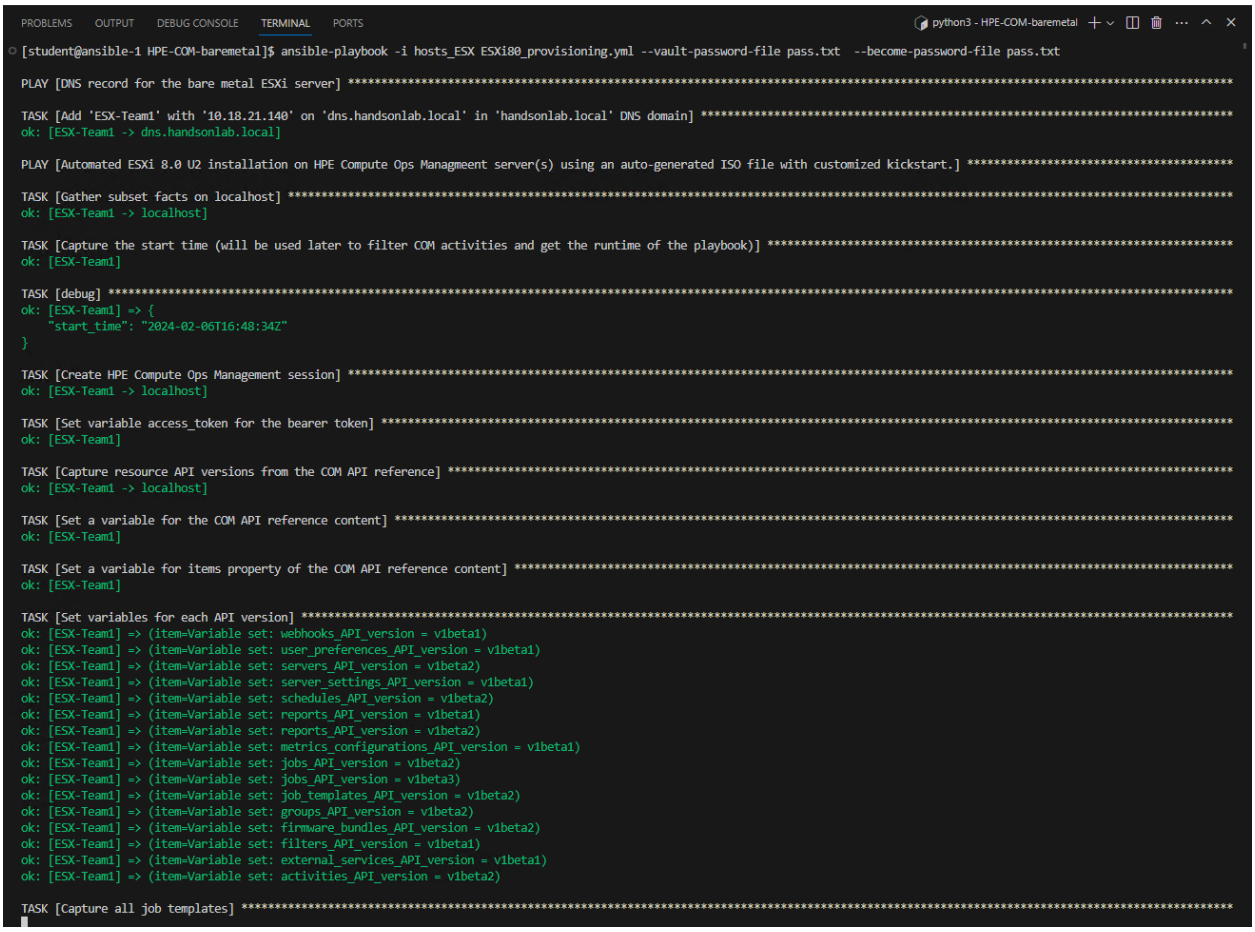
You are now ready to proceed with the provisioning process.

- Return to the VS Code terminal.
- To provision every host listed within the **[All]** group of your inventory—even though you currently have only one, but the concept remains the same—you need to execute just a single command:

  ```
  ansible-playbook ESXi80_provisioning.yml -i hosts_ESX --ask-vault-pass --ask-become-pass
  ```

  This command initiates the Ansible playbook named ***ESXi80_provisioning.yml***. This playbook contains a series of tasks that will be executed on the hosts specified in the ***hosts_ESX*** inventory file.

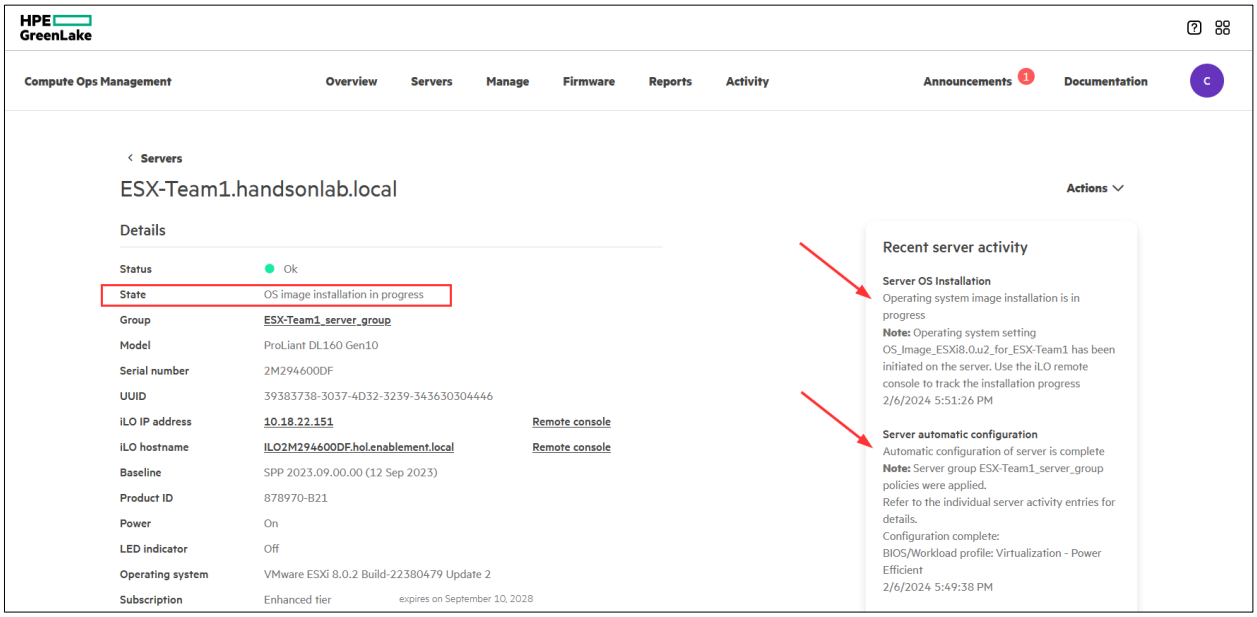
  The *--ask-vault-pass* flag prompts the user to enter the password for decrypting your Ansible Vault-protected files. The --ask-become-pass flag requests the user to input the privilege escalation (become) password which is required if the playbook needs elevated permissions to run certain tasks on the target hosts.
- At the prompt for the vault password, enter your vault password defined earlier. For the BECOME (sudo) password, enter: **xxxxxxxx**
- Once the playbook starts up, the various tasks are executed and displayed in the terminal window. And since this can sometimes happen very quickly, you can always scroll up to see the tasks you've missed.



Step 3: Monitoring task progress

To efficiently monitor the progress of your playbook tasks, you can switch back and forth between VS Code and the Compute Ops Management interface.

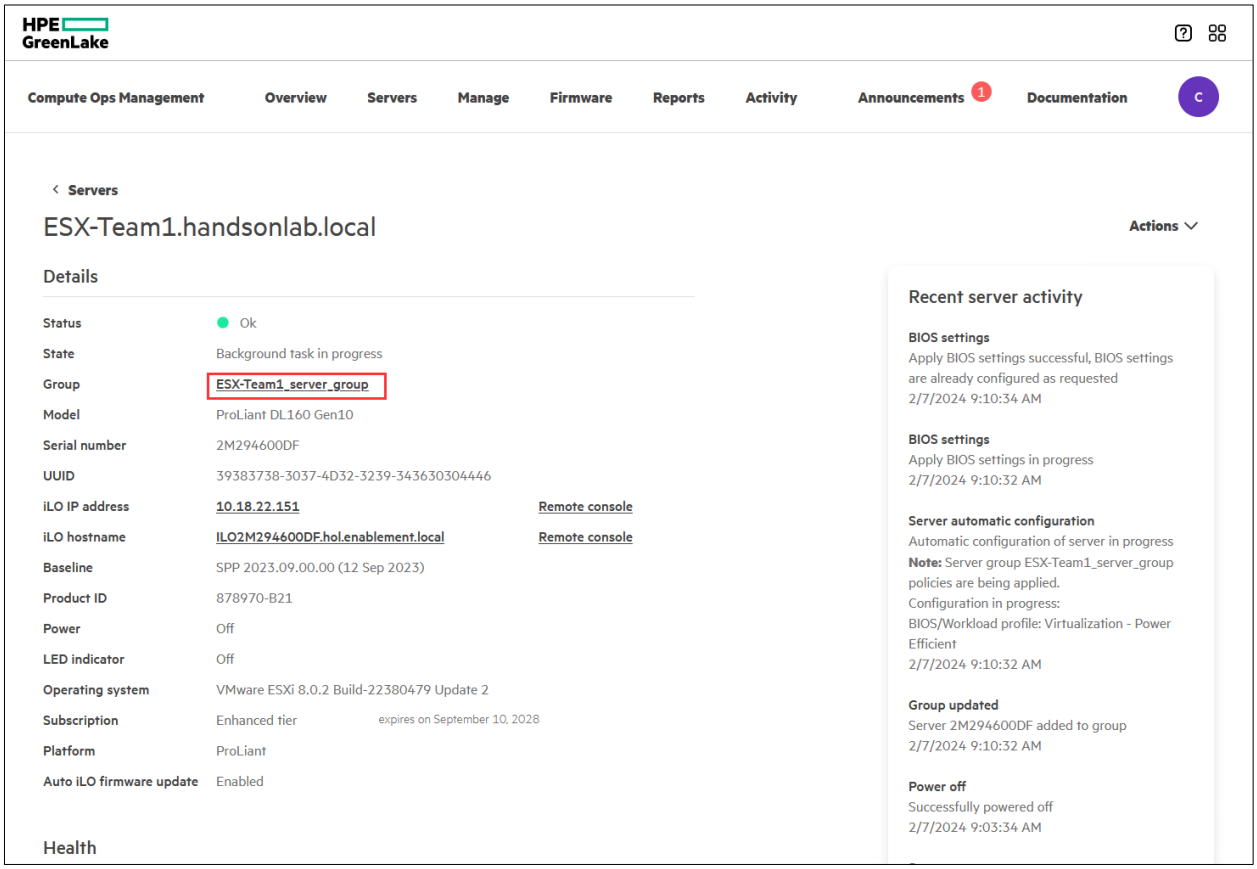
- In Compute Ops Management, you can mainly focus on the **State** attribute value on the server page to track job progress, and you can also check the activity panel on the right for more granular details:



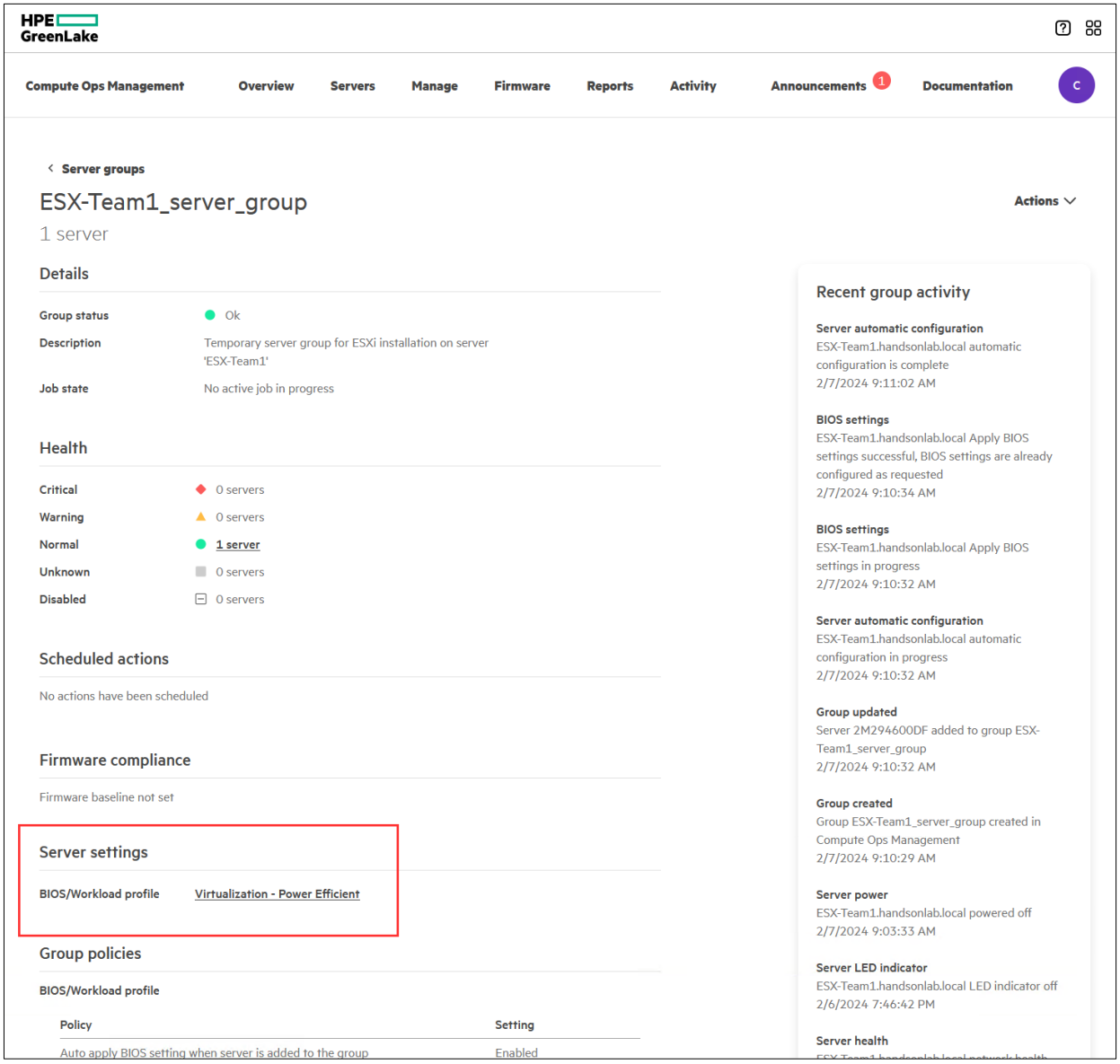
- One of the first steps is to detect the server's storage controller and disk size. This is a very important step in the installation process in the kickstart file to determine the target disk for the operating system installation:

```
TASK [Set controller type if disk found and when no NS204i and no SR/MR controller with disk available found] ***********************************
ok: [ESX-Team1] => (item=never)

TASK [Exit when no NS204i and no SR/MR controller and no other controller with disks are found] ***********************************************
skipping: [ESX-Team1]

TASK [Set a variable for the name of the storage controller found] ***************************************************************************
ok: [ESX-Team1]

TASK [debug] ********************************************************************************************************************************
ok: [ESX-Team1] => {
    "msg": "Storage controller found: 'SATA Storage System' with '2' drives"
}

TASK [Set controller type when no NS204i and no SR/MR controller with disk available found] *************************************************
ok: [ESX-Team1]

TASK [Set a controller type variable for kickstart creation when an NS204i is present] ******************************************************
skipping: [ESX-Team1]

TASK [Set a controller type variable for kickstart creation when there is no NS204i but an SR/MR controller is present.] ***************************
skipping: [ESX-Team1]

TASK [debug] ********************************************************************************************************************************
ok: [ESX-Team1] => {
    "msg": "Controller type found that will be used by kickstart: Local"
}

TASK [Check if server settings 'RAID1' already exist] ***************************************************************************************
skipping: [ESX-Team1]

TASK [Create internal storage configuration using 'RAID1' with a volume size of '-1' GB for the OS boot volume when it does not exist] ***********
skipping: [ESX-Team1]

TASK [Result of the 'RAID1' creation task] **************************************************************************************************
skipping: [ESX-Team1]
```

- A temporary server group is then created in Compute Ops Management to configure server storage and bios:



- If you go to the server group information, you'll only see the bios settings (and no storage settings) because the DL160/DL365 server you're using is not equipped with RAID storage controllers:

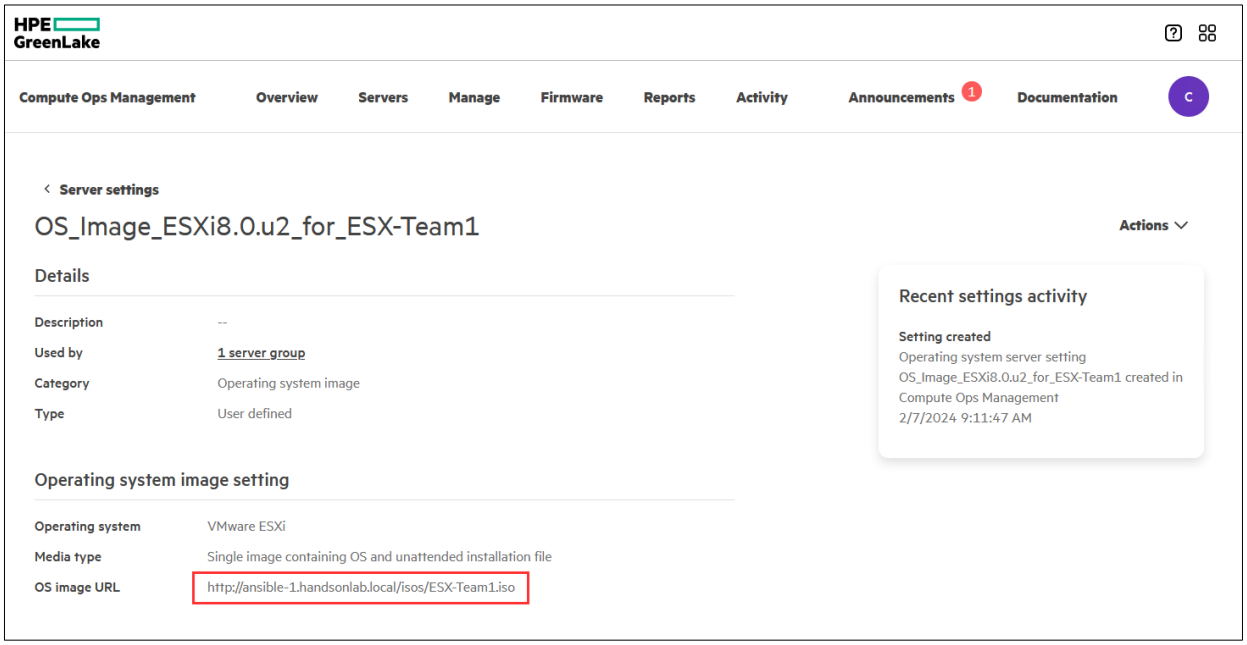


- Once the server bios configuration is complete, the next lengthy task to appear in the VS Code terminal is the installation of the operating system image:

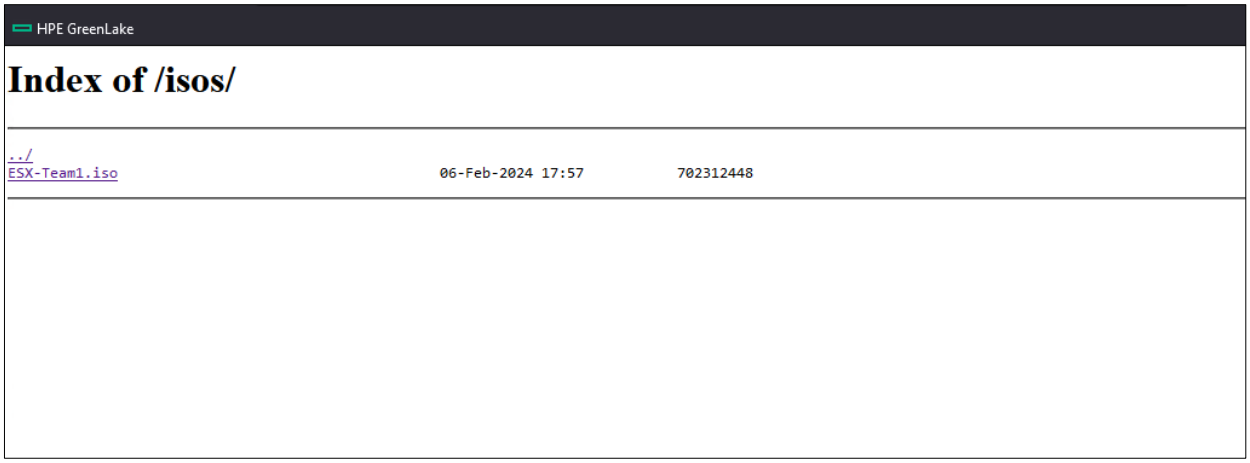
```
TASK [Set variables for the id and the resourceUri of the OS Image configuration 'OS_Image_ESXi8.0.u2_for_ESX-Team1' just updated] ****************
skipping: [ESX-Team1]

TASK [debug] ********************************************************************************************************************************
ok: [ESX-Team1] => {
    "OS_image_server_setting_resourceuri": "/compute-ops/v1beta1/server-settings/600e06ab-d472-42f0-9386-e24c394ca61c"
}

TASK [debug] ********************************************************************************************************************************
ok: [ESX-Team1] => {
    "OS_image_server_setting_id": "600e06ab-d472-42f0-9386-e24c394ca61c"
}

TASK [Update temporary group 'ESX-Team1_server_group' to add OS image installation settings with NS204i] ************************************
ok: [ESX-Team1 -> localhost]

TASK [Display response of the group 'ESX-Team1_server_group' update request with NS204i] ************************************************
ok: [ESX-Team1] => {
    "server_group_result.msg": "OK (1625 bytes)"
}

TASK [Update temporary group 'ESX-Team1_server_group' to add OS image installation settings with storage controller (without NS204i)] ********
skipping: [ESX-Team1]

TASK [Display response of the group 'ESX-Team1_server_group' update request with storage controller] ************************************
skipping: [ESX-Team1]

TASK [Start OS image installation in group 'ESX-Team1_server_group'] ************************************************************************
ok: [ESX-Team1 -> localhost]

TASK [Display response of the OS image installation request in group 'ESX-Team1_server_group'] ****************************************
ok: [ESX-Team1] => {
    "group_os_installation_result.msg": "OK (958 bytes)"
}

TASK [Wait for the OS image installation of server '2M294600DF' to complete] ****************************************************************
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the OS image installation of server '2M294600DF' to complete (60 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the OS image installation of server '2M294600DF' to complete (59 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the OS image installation of server '2M294600DF' to complete (58 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the OS image installation of server '2M294600DF' to complete (57 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the OS image installation of server '2M294600DF' to complete (56 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the OS image installation of server '2M294600DF' to complete (55 retries left).
```

- If you look in Compute Ops Management, the server group has been updated with a new server parameter, an operating system image:

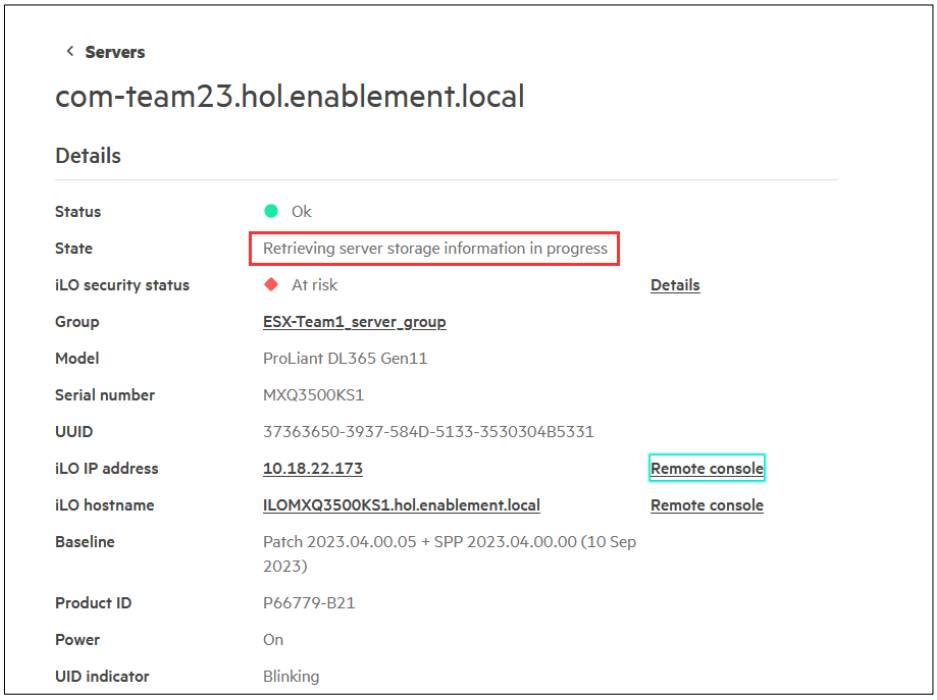
- You can click on this operating system image to see the settings that have been applied:



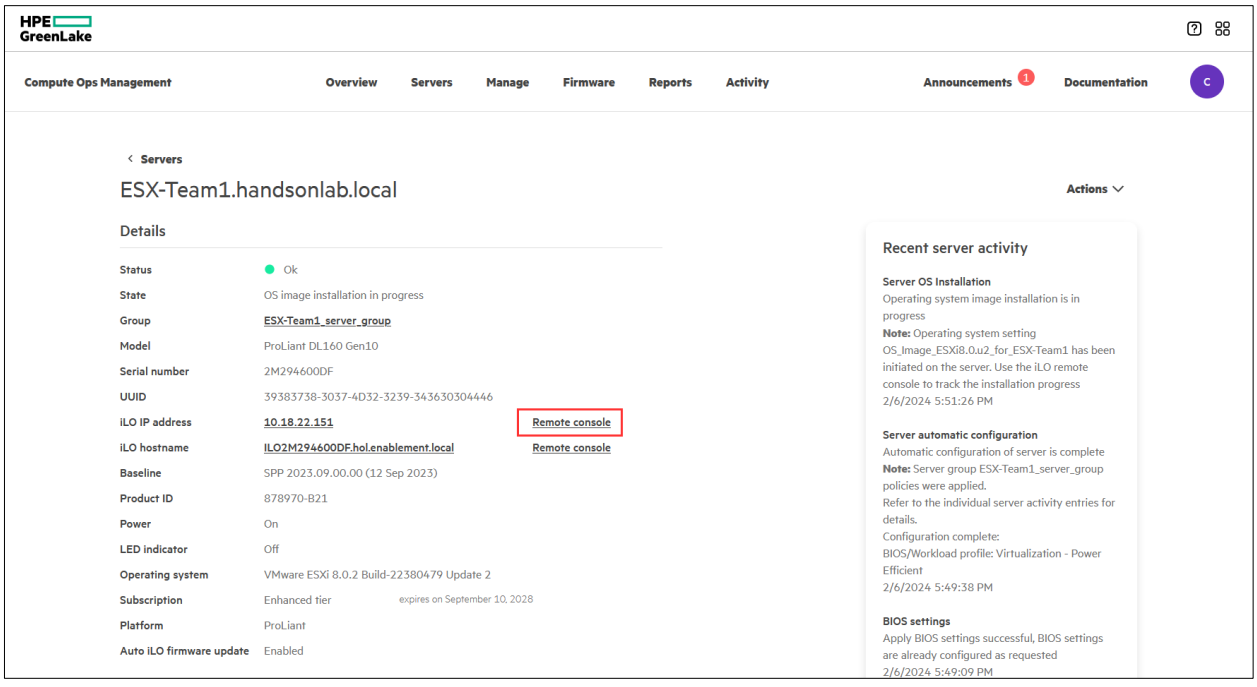
- This image is located on your Ansible Control Node nginx web server. You can open a browser and visit the URL http://hol05-ansible-X.handsonlab.local/isos with X your team number and check that the file is present:



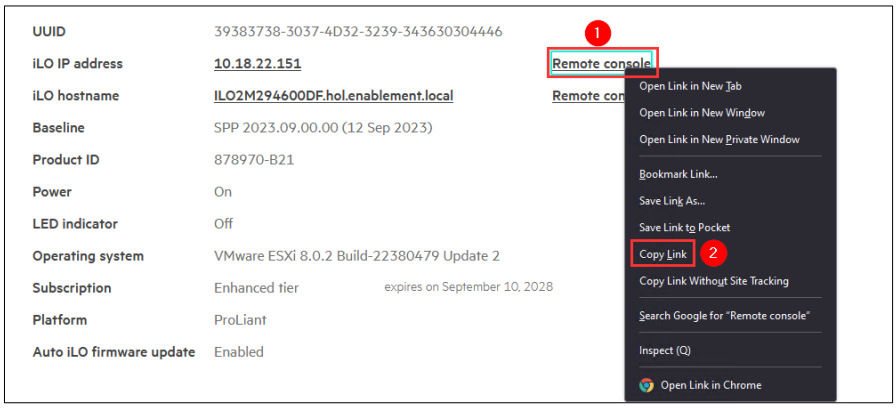
- Depending on the server model you are using, the server storage information task can take several minutes. Please be patient!



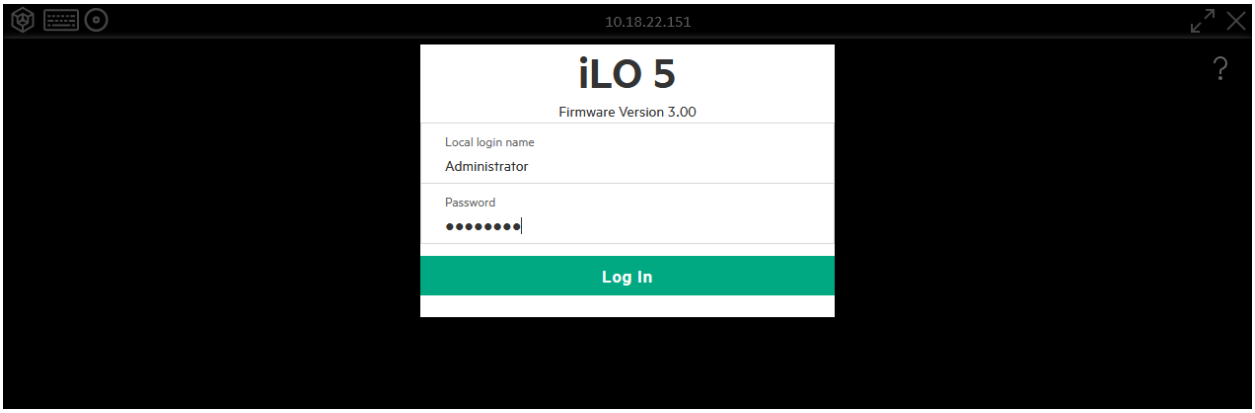
- To visually confirm that the server is booting from the ISO image and initiating the installation of ESXi, you can use the iLO remote console. Go back to your server's page and click on the **Remote console** link:



Note: If you are unable to open the remote console page, right-click on the Remote console link on the Compute Ops Management page and click **Copy Link**. Then open a new browser tab and paste the iLO remote console link, which should allow you to access the iLO console.

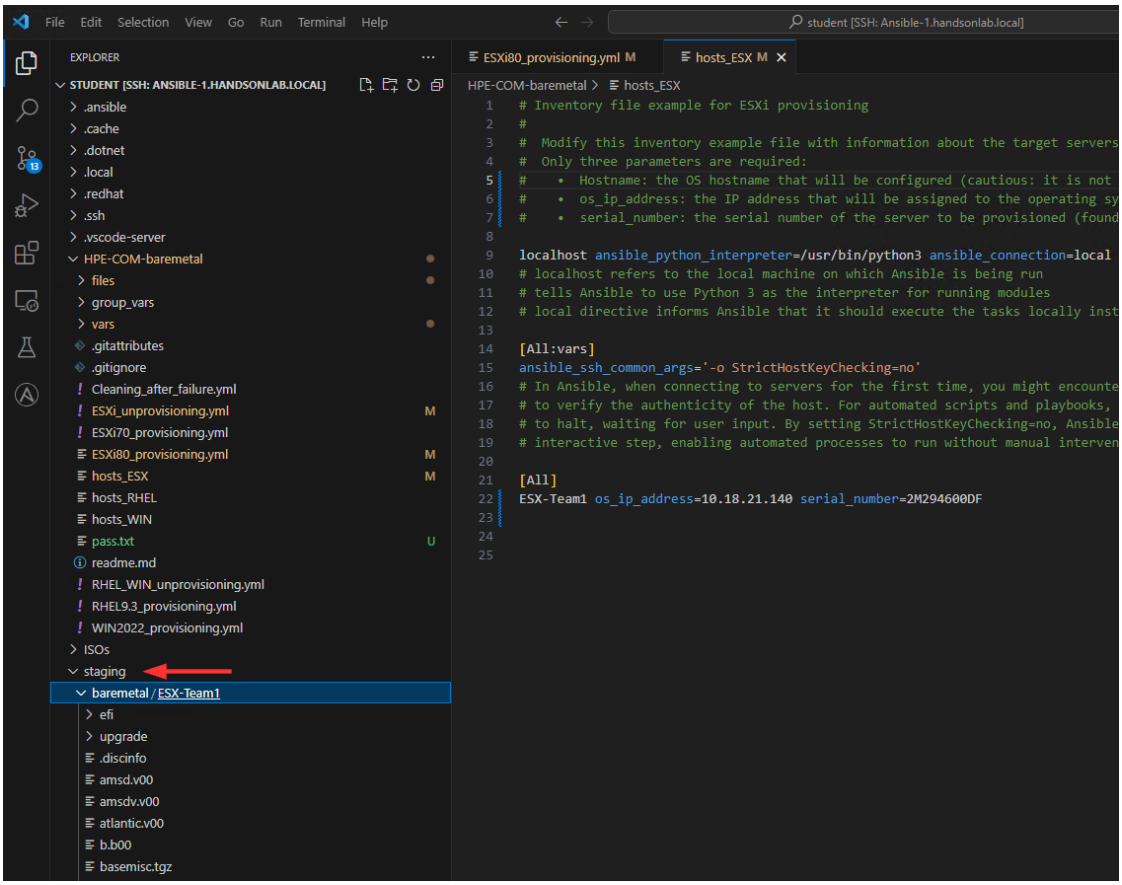


- For the login name and password, use: **Administrator** / **xxxxxxxxxx**



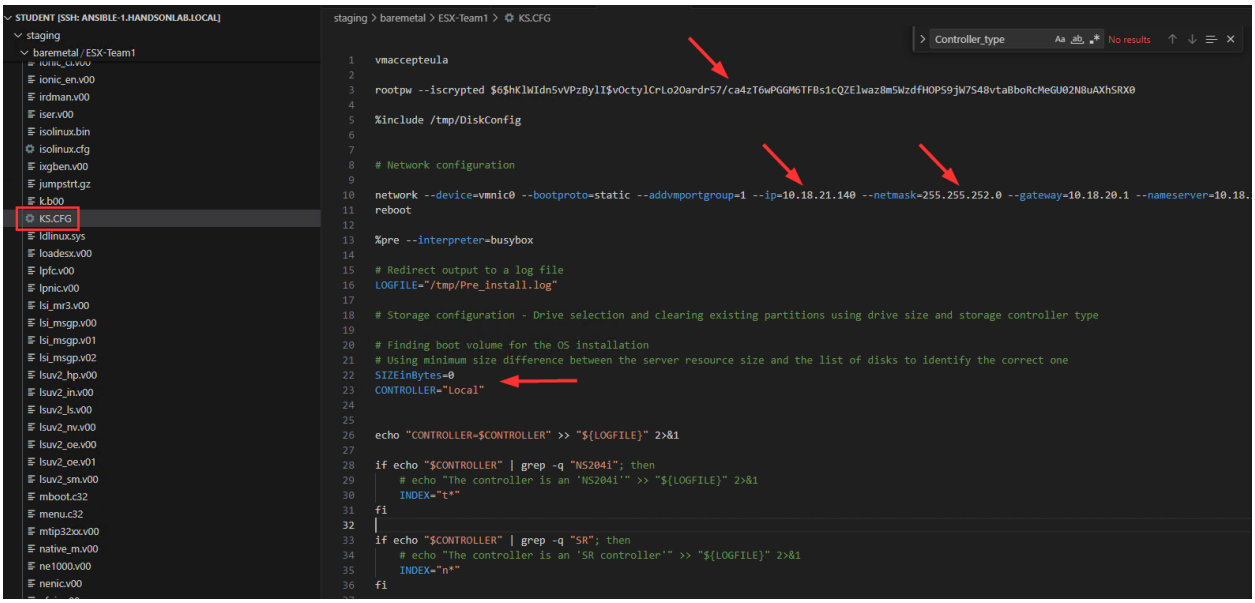
- When the operating system installation will start, the iLO virtual media will be configured by Compute Ops Management to boot once from the ISO image that Ansible generated with the custom kickstart:

- The content used to produce this image has been temporarily saved in the **staging/baremetal/ESX-Team**X folder:



- The ESXi installation script (i.e. kickstart file) is read during installation:

- You can check its contents by opening the KS.CFG file in this folder:

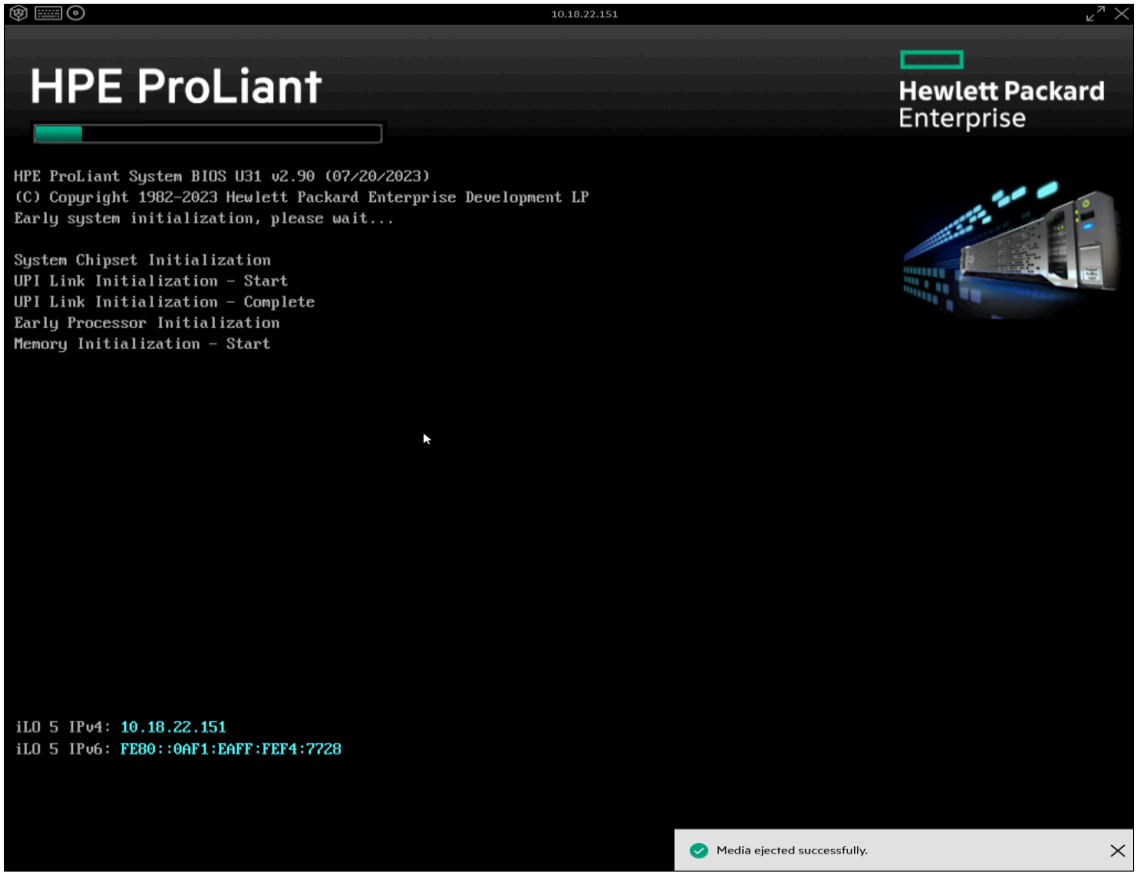


Note that variable substitutions have taken place during the process.

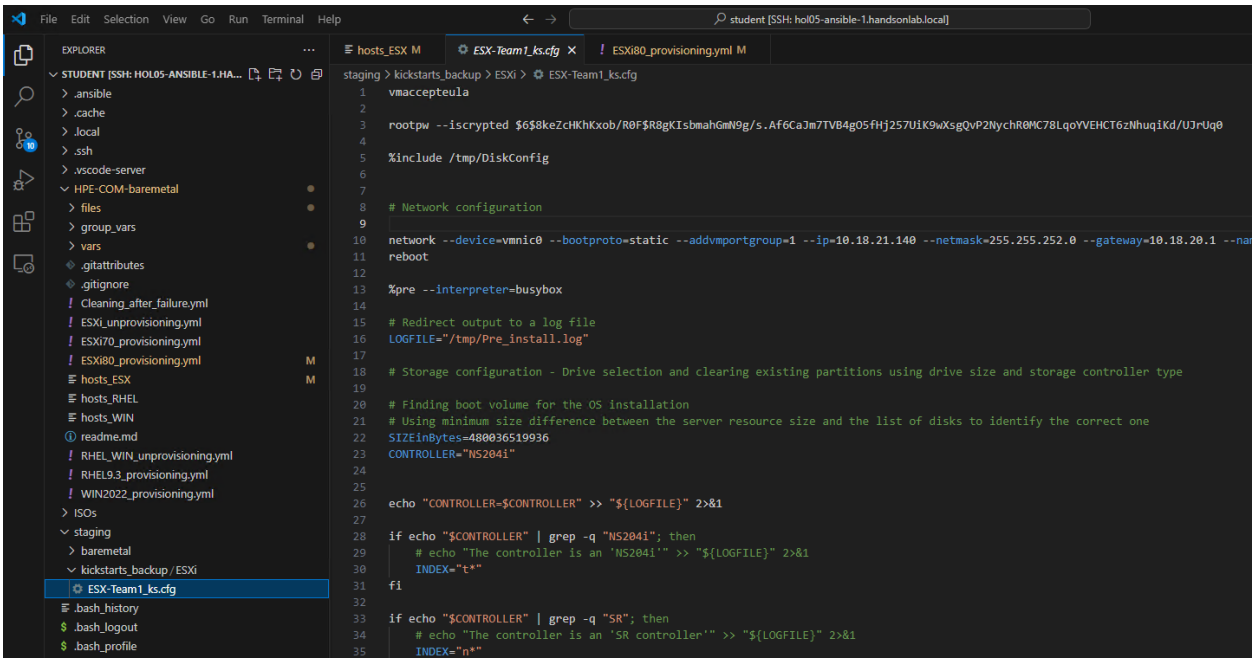
- Later during the ESXi installation, the following warning message regarding processor support can be displayed on the ESXi console. You can ignore this message, as it will not stop the installation:

- Once ESXi has been installed successfully, the server reboots:

- And the iLO virtual media is ejected automatically:



- Next the playbook performs several post-installation steps. Firstly, a bit of cleaning will be done. All your files in the staging folder will be deleted. However, the kickstart file is preserved for record-keeping purposes. You'll find the archived kickstart file in **staging/kickstarts_backup/ESXi**.

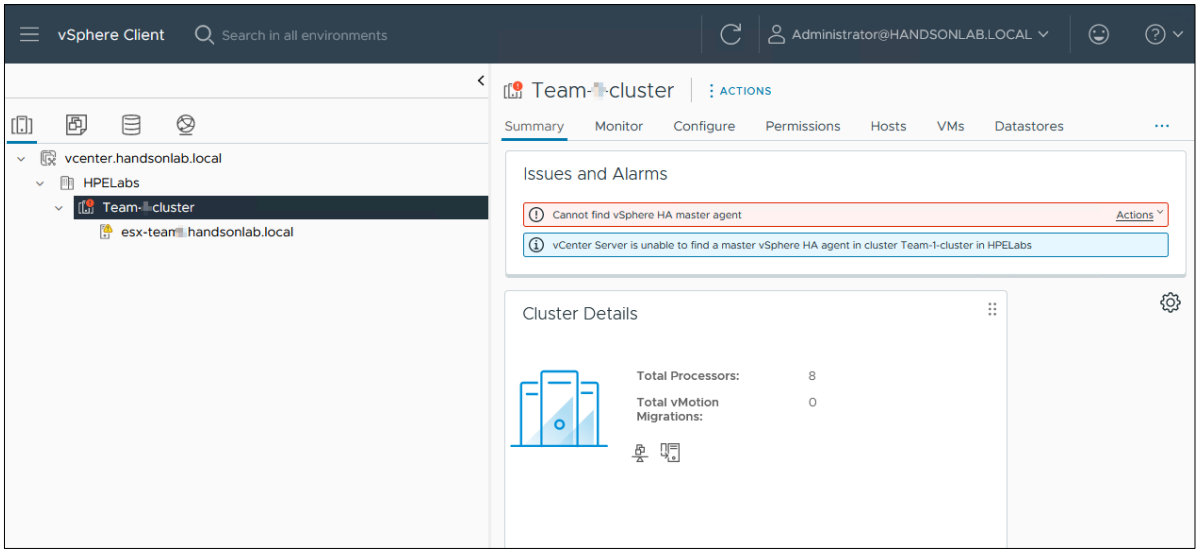


- Secondly, as we are using ESXi, there are certain post-installation tasks involving VMware vCenter such as creating and configuring the ESXi cluster, connecting the ESXi host to the cluster, configuring the ESXi host (i.e. power management, NTP, SSH, Shell and vSwitch are configured) and enabling maintenance mode to prepare the host for firmware updates.

- You can monitor the progress of these tasks in VMware vCenter using:

  - URL: https://vcenter.handsonlab.local/ui
  - User: administrator@handsonlab.local
  - Password: xxxxxxxxxxxxxxxxxxxxxxxxx

- Make sure you open the cluster created by your playbook named **Team-X-cluster** (with X your team number):



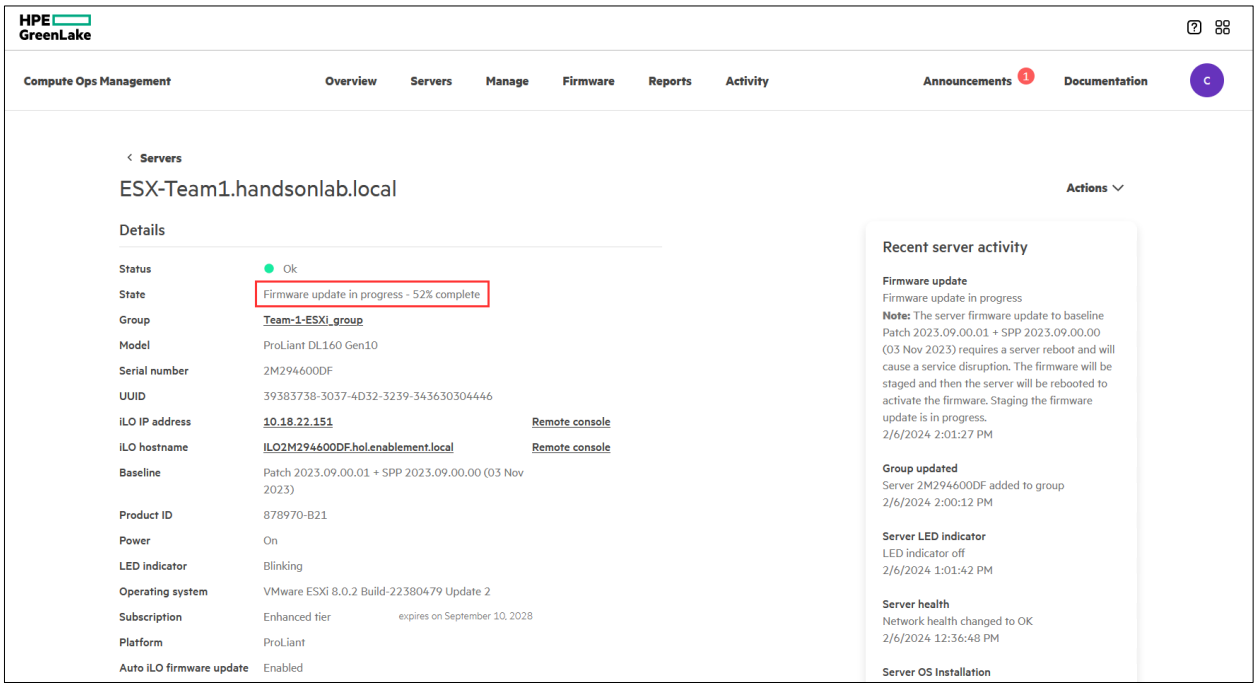
- Thirdly, a firmware update task is executed. This step is particularly important, as we want the server's firmware to be up to date before we put it into production.

  This is the second long-running task to appear in the VS Code terminal. The firmware update task is initiated by a Compute Ops Management group's firmware update job:

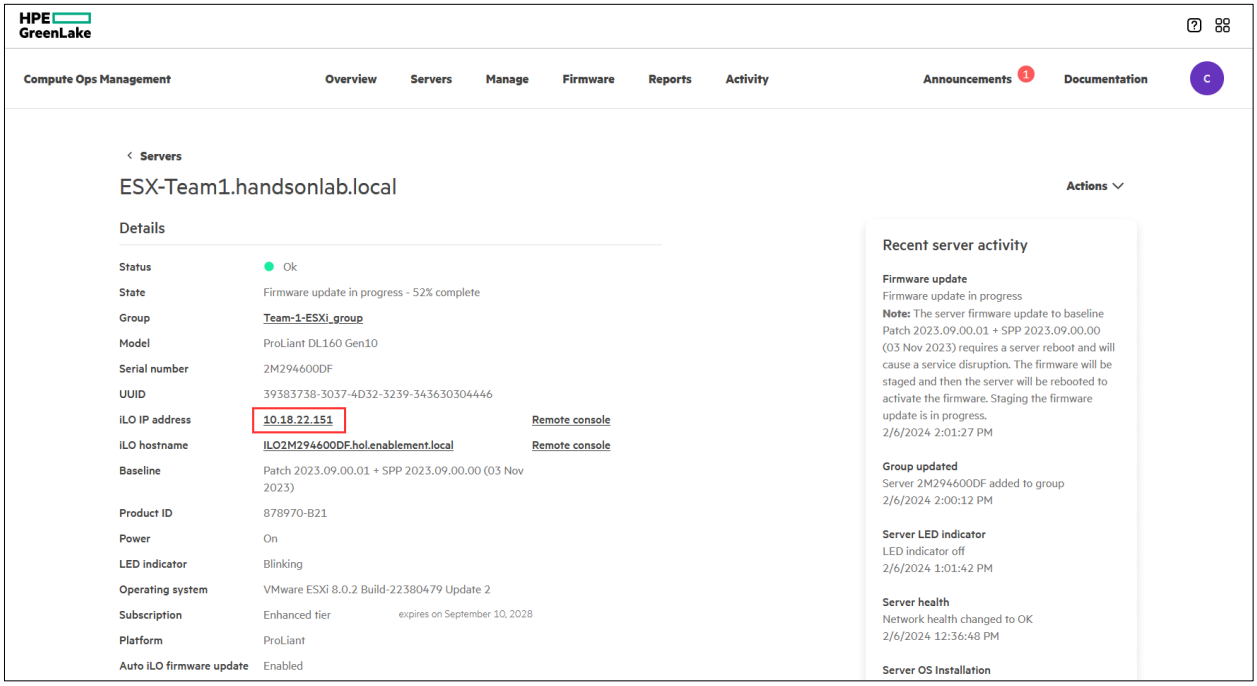
```
TASK [Set fact for the list of server IDs member of the definitive server group (first host only)] ************************************
ok: [ESX-Team1]

TASK [Display the server IDs found (first host only)] ***************************************************************************************
ok: [ESX-Team1] => {
    "server_ids": [
        "878970-B21+2M294600DF"
    ]
}

TASK [Create HPE Compute Ops Management session] ********************************************************************************************
ok: [ESX-Team1 -> localhost]

TASK [Set variable access_token for the bearer token] ***************************************************************************************
ok: [ESX-Team1]

TASK [Start a group firmware update job using the defined firmware baseline (first host only)] **********************************************
ok: [ESX-Team1 -> localhost]

TASK [Display response of the group firmware update job (first host only)] ******************************************************************
ok: [ESX-Team1] => {
    "group_firmware_update_job.msg": "OK (1078 bytes)"
}

TASK [Capture the resourceUri of the group firmware update job (first host only)] ***********************************************************
ok: [ESX-Team1]

TASK [Display the group firmware update job resource uri (first host only)] *****************************************************************
ok: [ESX-Team1] => {
    "group_firmware_update_job_resourceUri": "/compute-ops/v1beta3/jobs/cdfa6a8b-0412-42a6-8cef-52f67198d984"
}

TASK [Wait for the group firmware update job to complete (first host only)] *****************************************************************
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the group firmware update job to complete (first host only) (600 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the group firmware update job to complete (first host only) (599 retries left).
FAILED - RETRYING: [ESX-Team1 -> localhost]: Wait for the group firmware update job to complete (first host only) (598 retries left).
```

  Note: The firmware update job can be carried out quickly if your server is already up to date.

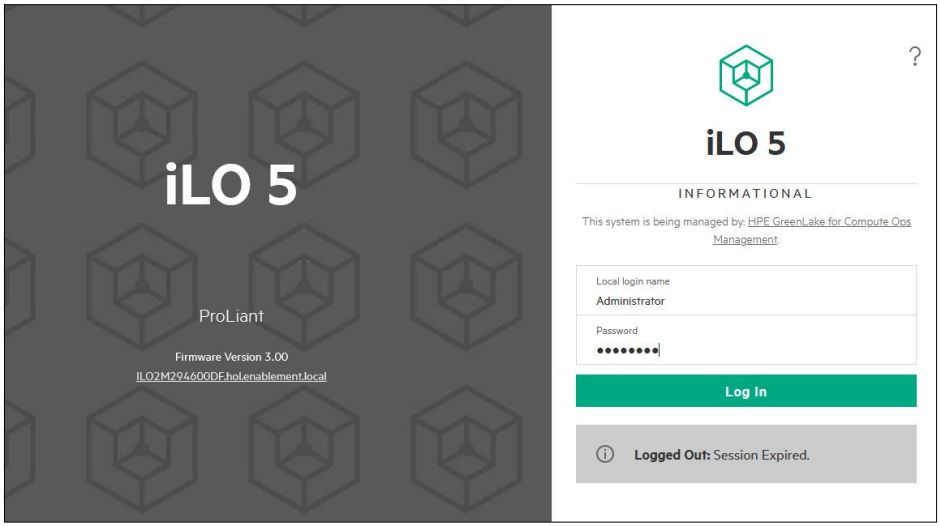
- In Compute Ops Management, the progress of the firmware update is displayed:



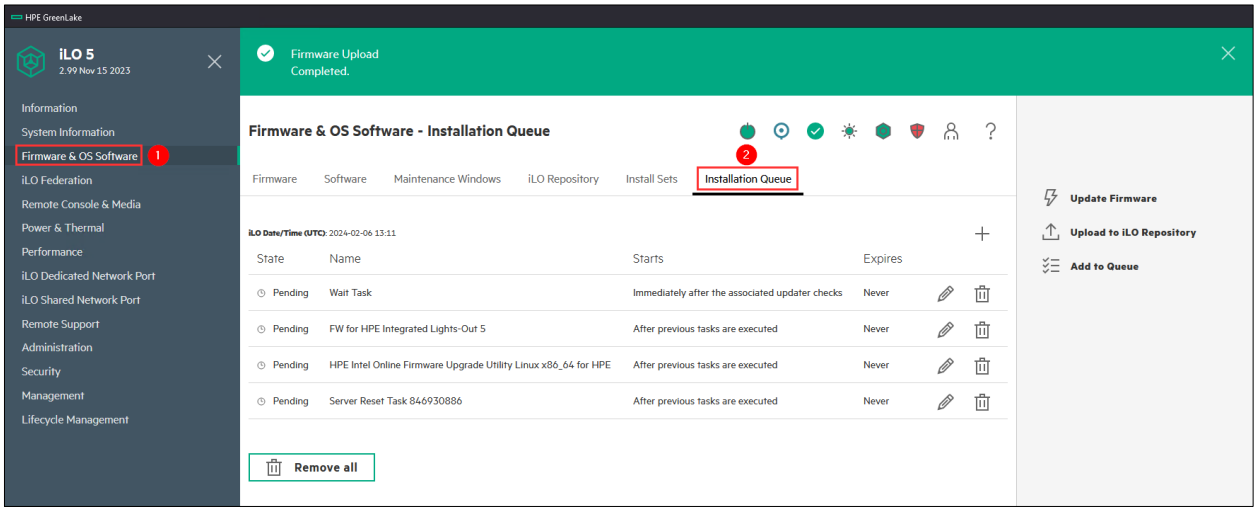
- But you can also visit the iLO web interface to get more information about the packages that will be installed (if any). Click on the iLO IP address on the server page:



- For the login name and password, use again: **Administrator** / **xxxxxxxxxx**

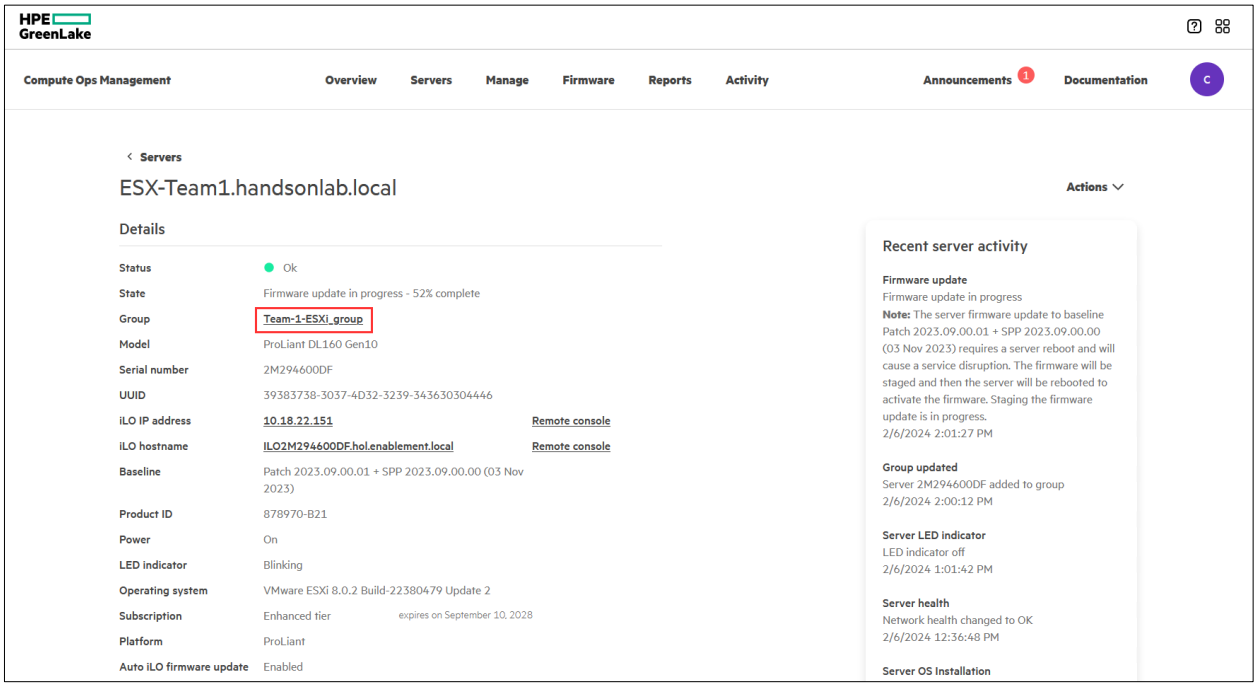


- Then go to the Installation Queue menu in Firmware & OS Software:



The various packages that are about to be installed are listed on this page. Note that server reset tasks may also be listed for firmware activation.

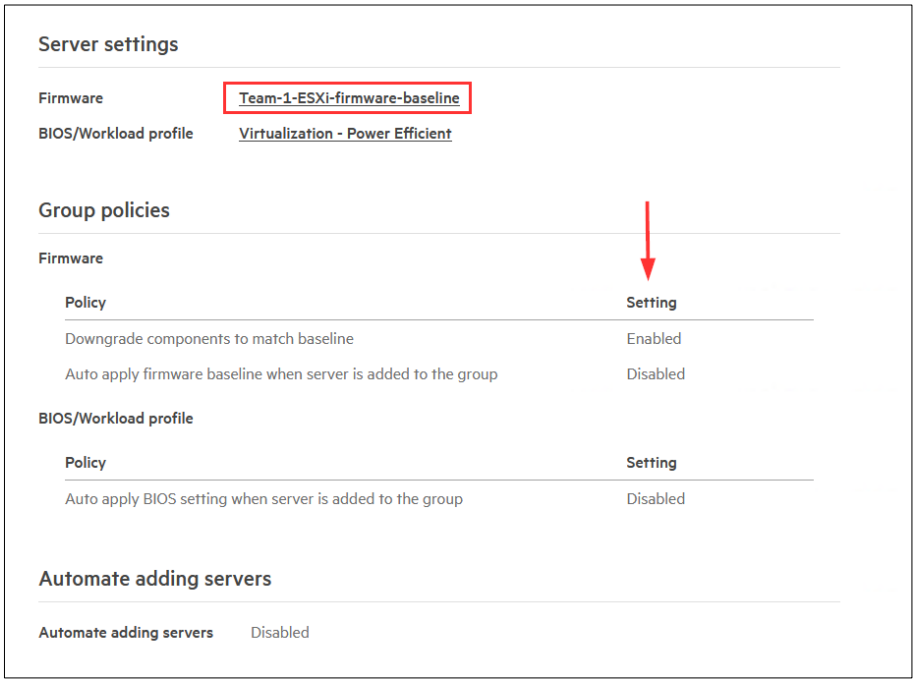
- These packages are defined by the bundle assigned to the new server group. Go back to the Compute Ops Management interface, and open the **Team-X-ESXi_group** in the server page:



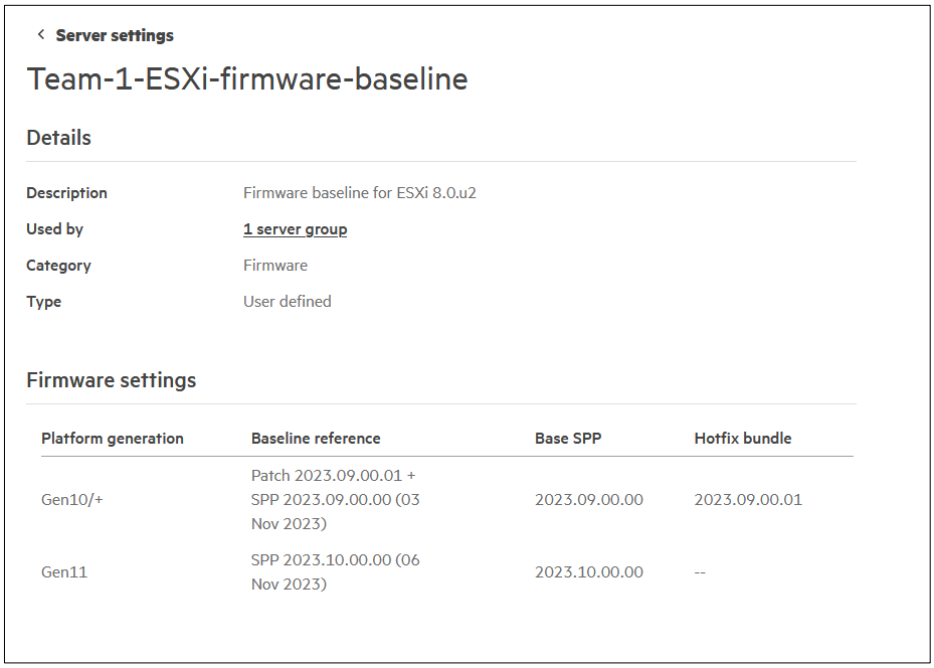
This newly established group is the ultimate configuration resource for your ESXi production servers. The previous group, which contained the OS image settings with a customized kickstart, has been removed following the completion of the OS installation process.

The creation of this new group ensures uniformity across all ESX host members by standardizing the firmware baseline and enforcing consistent BIOS/workload profile settings. This consolidation guarantees that all hosts adhere to the same operational standards, streamlining management and minimizing variability in your server environment.

- If you scroll down to the **Server** settings section, you will notice the firmware baseline with the group policies that have been applied:



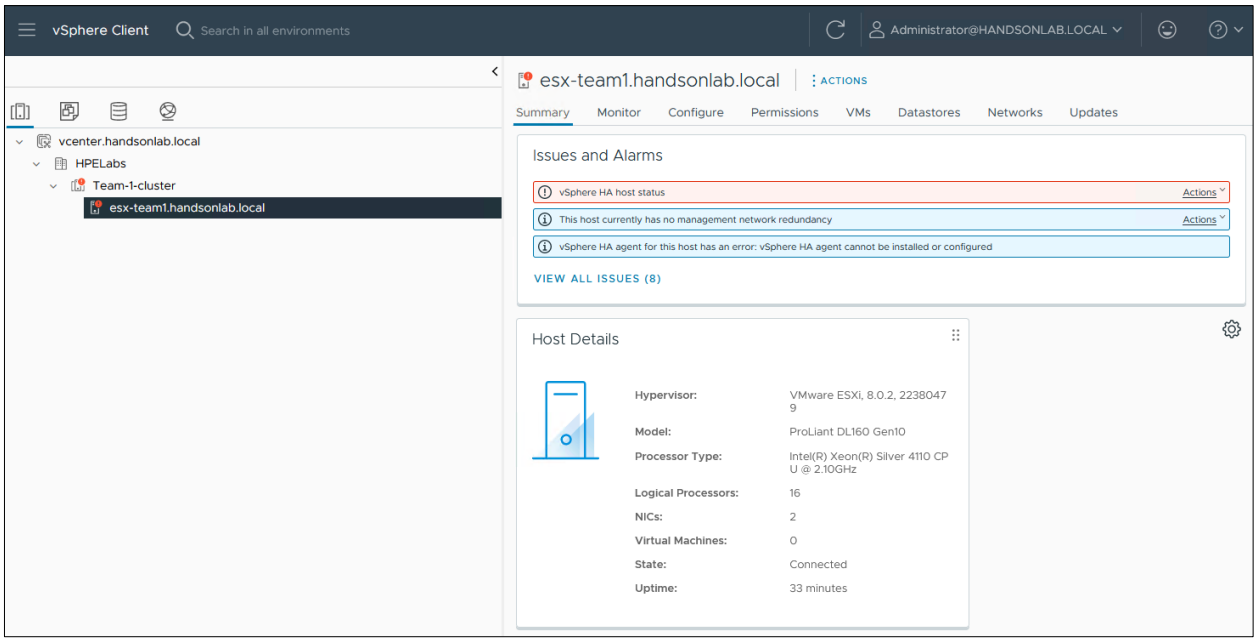
- You can open the **Team-X-ESXi-firmware-baseline** settings to access details of the firmware package:



- Once the firmware update task has been completed, a job results report is provided in the terminal, together with a report on firmware update activity:

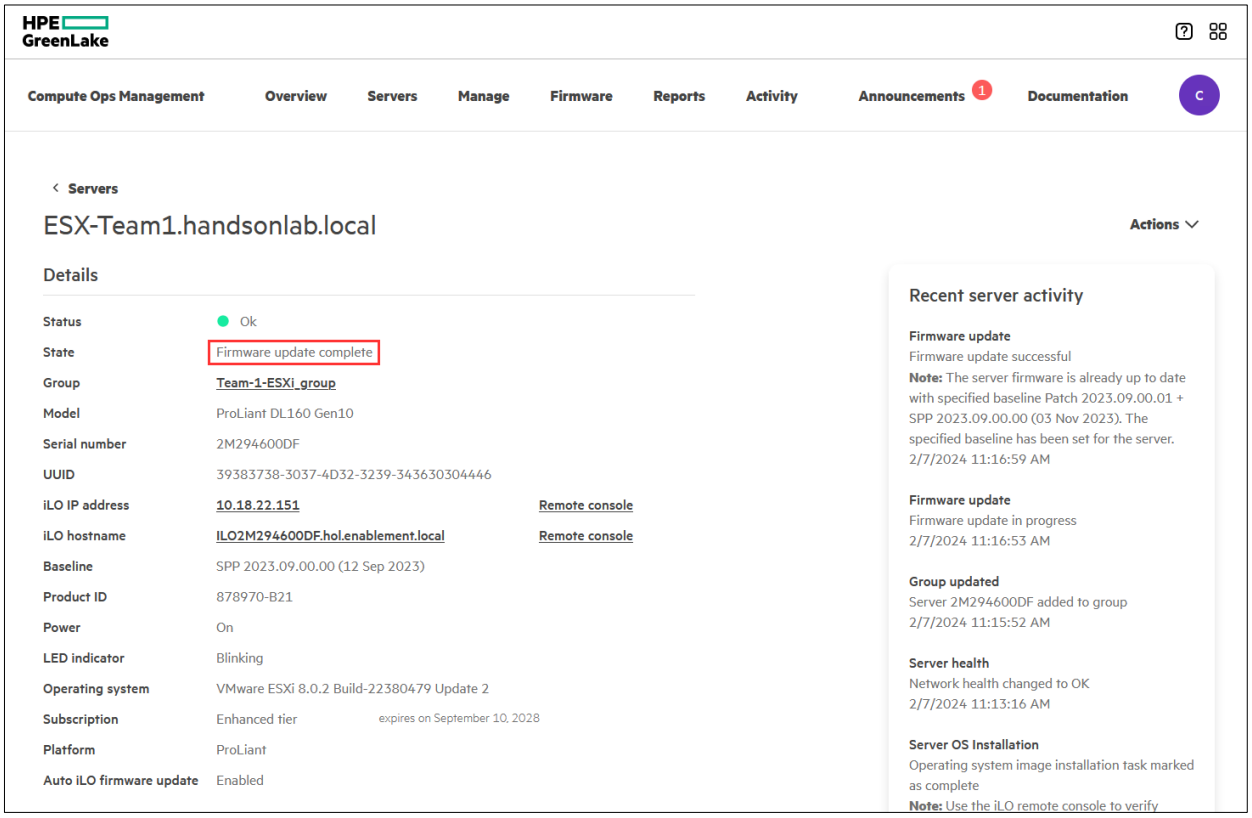
```
TASK [Display the group firmware update job result (first host only)] ************************************************************************************************************
ok: [ESX-Team1] => {
    "msg": [
        "State: COMPLETE",
        "ResultCode: SUCCESS",
        "Status: Complete"
    ]
}

TASK [Get the latest activity of server '2M294600DF'] ****************************************************************************************************************************
ok: [ESX-Team1 -> localhost]

TASK [Extract items containing 'Firmware update'] ********************************************************************************************************************************
ok: [ESX-Team1]

TASK [Display the firmware update activity status when successful] ***************************************************************************************************************
ok: [ESX-Team1] => {
    "msg": [
        "Message: Firmware update successful",
        "",
        "**Note:** The server firmware is already up to date with specified baseline Patch 2023.09.00.01 + SPP 2023.09.00.00 (03 Nov 2023). The specified baseline has been set for th
e server.",
        ""
    ]
}
```

- In vCenter, the host is taken out of maintenance:



Note: Several errors should be displayed concerning HA because you only have one host in your cluster and concerning management network redundancy because only one NIC is connected to the server.

- In Compute Ops Management, the task is shown as complete:



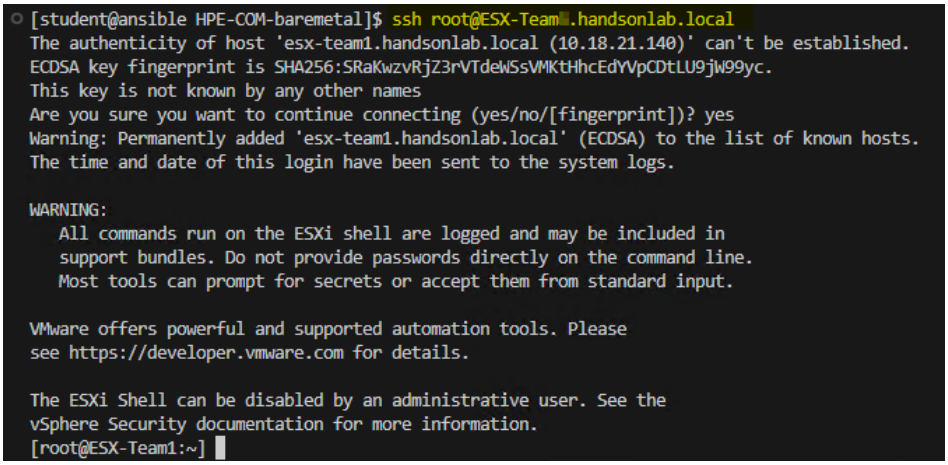
- A final status of the entire installation is then displayed by the playbook:

```
TASK [Get runtime] ************************************************************************************************************************************
ok: [ESX-Team1]

TASK [Status of the OS installation with the firmware update with HPE drivers and software] ***********************************************************
skipping: [ESX-Team1]

TASK [Status of the OS installation with the firmware update without HPE drivers and software] ********************************************************
ok: [ESX-Team1] => {
    "msg": [
        "ESX-Team1.handsonlab.local installation and firmware update completed successfully!",
        "OS is configured and running and added to the vCenter cluster 'Team-1-cluster'",
        "To SSH to the new host from Ansible control node, use: ssh root@ESX-Team1.handsonlab.local",
        "The provisioning task took 0:32:00."
    ]
}

TASK [Status of the OS installation with firmware update failure] *************************************************************************************
skipping: [ESX-Team1]

PLAY RECAP ********************************************************************************************************************************************
ESX-Team1                  : ok=147  changed=25   unreachable=0    failed=0    skipped=75   rescued=0    ignored=1
```

- Once the server provisioning is complete, you'll be able to access the ESXi CLI (enabled during the playbook execution) from the Ansible Control node using its SSH public key for password-less entry by utilizing:

  `ssh root@ESX-TeamX.handsonlab.local`

  with X your team number.



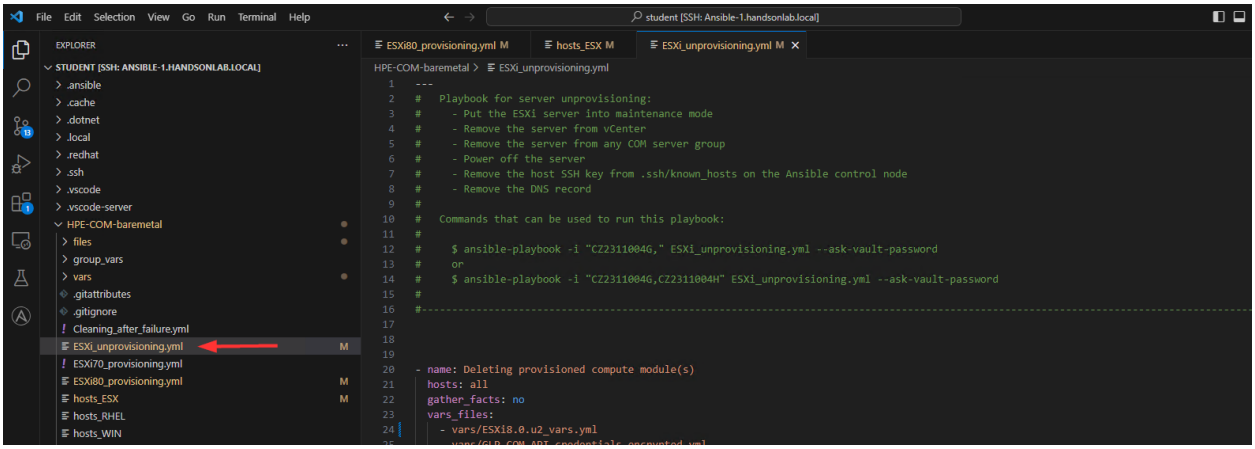
- Type **exit** to leave the ESXi CLI and return to the Rocky Linux terminal.

Congratulations on achieving this milestone! If you're reading this, it signifies that the installation has concluded. Your ESXi is now fully operational for production use, seamlessly integrated into a cluster. It is currently running the most up-to-date firmware versions as specified in your ESXi production group within Compute Ops Management, ensuring optimal performance and reliability.

Step 4: Decommissioning the ESX Host

For those looking to reverse the process, decommissioning your ESX host can be a critical part of managing your server lifecycle. Whether you're aiming to change the server role or update the server to a new OS version, decommissioning can be efficient and secure without disrupting the existing infrastructure.

- To do this, a different playbook is available at the root of the project:



Note: Given the inherent risks associated with decommissioning servers, it is crucial to implement stringent precautions.

- To initiate the decommission process using this playbook, you must supply a list of server serial numbers targeted for shutdown. For your specific ESX host decommissioning, you can execute the command below:

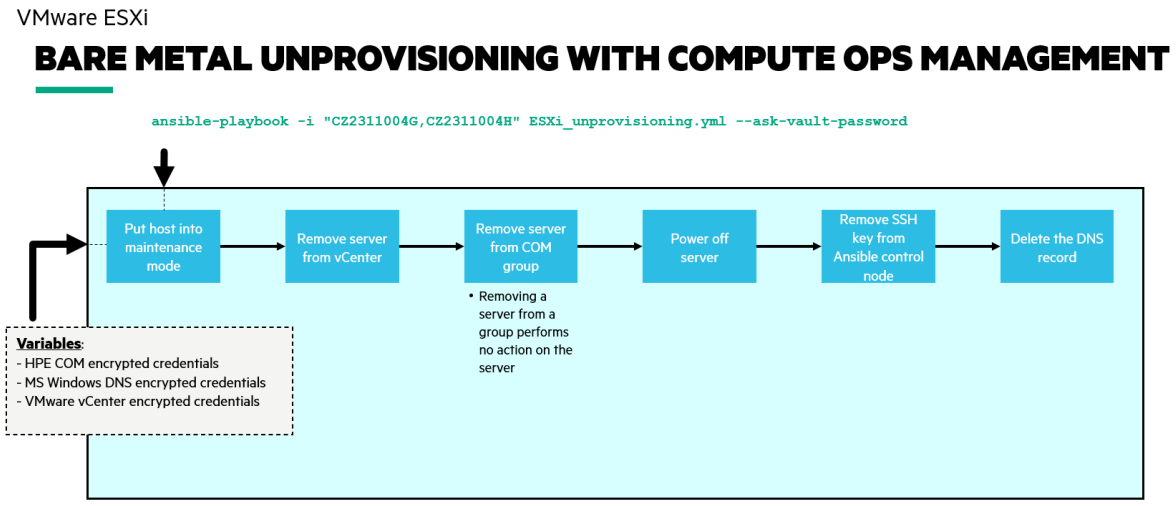
  `ansible-playbook -i "<Your server serial number>," ESXi_unprovisioning.yml --ask-vault-password`

  Important: Be sure to enter the server serial number assigned to your team!

  Remember: It's crucial to include a comma following the serial number; this is mandatory even if you're entering just one serial number. Additionally, there's no need to use sudo or the BECOME privilege escalation for this un-provisioning operation.

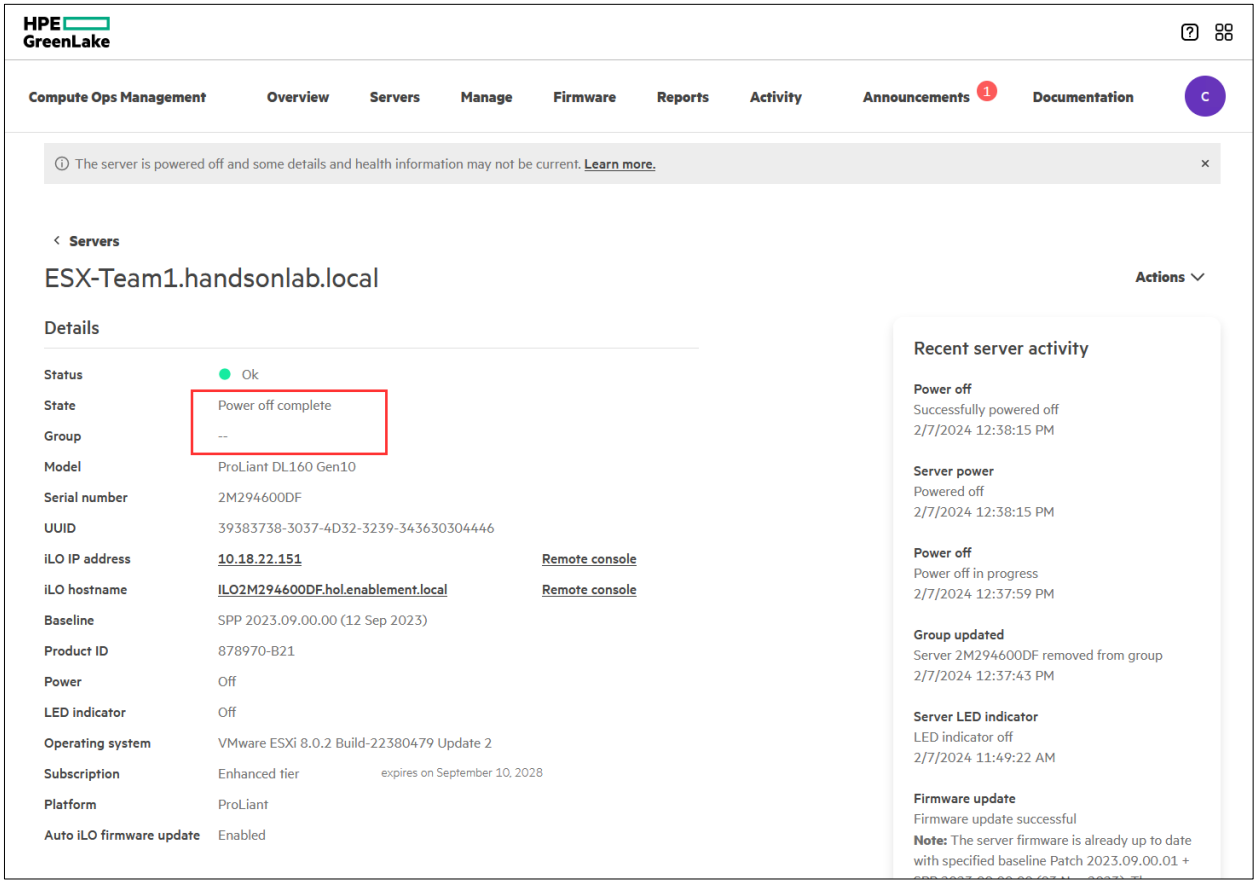
- At the prompt for the vault password, type the vault password you defined earlier to encrypt your sensitive variables.

- The un-provisioning process flow is much lighter than for provisioning:



- You can track the progress of tasks on the terminal, vCenter and Compute Ops Management.

- The final state of your server in Compute Ops Management is 'Power off complete', and the server should no longer have an assigned group:



- Your server is now ready for the next provisioning.

This concludes the lab.

Summary

During this lab, you've gained firsthand understanding of automatic bare metal provisioning by utilizing Ansible alongside HPE GreenLake for Compute Ops Management and kickstart methods. The practical experience should have illustrated the significant benefits of integrating Compute Ops management with tools like Ansible—minimizing manual tasks, accelerating configuration, and promoting uniformity, productivity, and scalability within your data center operations through automation and swift resource reassignment.

If you encounter any questions, opinions, or problems during your own testing, it's highly recommended that you report them on the project's issues page on GitHub. This can help improve the project and potentially benefit other users who may be facing similar issues.

In this lab you have provision only one server, if you want to watch a video where parallel provisioning is demonstrated, see: https://www.youtube.com/watch?v=_ySgROdd_Bw

This project also includes playbooks for provisioning Red Hat Enterprise Linux and Windows Server with the same seamless experience and speed. But that's outside the scope of this lab and if you want to find out more, you can watch the following videos:

- Efficient Bare Metal Provisioning for RHEL 9.3 with HPE Compute Ops Management and Ansible: https://www.youtube.com/watch?v=6_o8yB4cvag
- Efficient Bare Metal Provisioning for Windows Server with HPE Compute Ops Management and Ansible: https://www.youtube.com/watch?v=A6RD6nIAFmw

Want more?

- Various sample scripts for the Compute Ops Management API, including Ansible playbooks, PowerShell and Python scripts, are available here.
- An API collection for Postman is accessible online and offers developers the opportunity to interact with a variety of HPE APIs. These include APIs for HPE GreenLake for Compute Ops Management, HPE OneView, HPE iLO, HPE OneView Global Dashboard, among others.
- Back home, you can head to the HPE Demonstration Portal and request a time slot (https://hpedemoportal.ext.hpe.com/)
- Request a 90-day Compute Ops Management evaluation (https://www.hpe.com/us/en/compute/management-software.html?dmodal=modal-edbaf#)
- Learn more about the HPE GreenLake Developer portal at https://developer.greenlake.hpe.com/
- Discover the HPE Developer community at https://developer.hpe.com/

LEARN MORE AT

hpe.com/us/en/greenlake.html

Appendix 1

Ansible Control Node configuration

- Command that you must modify then copy and paste into your terminal window and run:

  `sudo hostnamectl set-hostname hol05-ansible-X.handsonlab.local`
  (make sure you change X with your team number)

- Commands that you can copy and paste into your terminal window and execute at once:

```
sudo dnf -y install git
git clone https://github.com/jullienl/HPE-COM-baremetal
cd HPE-COM-baremetal/
ssh-keygen -t rsa -f ~/.ssh/id_rsa -N ""
sudo dnf -y install epel-release
sudo dnf -y install mkisofs
sudo dnf -y install python3-pip
pip3 install setuptools-rust wheel
pip3 install ansible-core
ansible-galaxy collection install -r files/requirements.yml
pip3 install --upgrade pip setuptools
pip3 install --upgrade git+https://github.com/vmware/vsphere-automation-sdk-python.git
pip3 install -r ~/.ansible/collections/ansible_collections/community/vmware/requirements.txt
pip3 install --upgrade git+https://github.com/vmware/vsphere-automation-sdk-python.git
pip3 install pywinrm
pip3 install jmespath
sudo dnf -y install nginx
sudo systemctl enable nginx
sudo systemctl start nginx
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload
sudo sed -i '0,/server {/s//&\n        autoindex on;/' /etc/nginx/nginx.conf
sudo systemctl restart nginx
sudo dnf -y install rsync
```