IBM watsonx Code Assistant for Red Hat Ansible Lightspeed - Technical Sales Level 3

Description	IBM watsonx Code Assistant for Red Hat Ansible Lightspeed - Technical Sales Level 3
Author(s)	Christopher Bienko (cdbienko@us.ibm.com)
Copyright	

> Christopher Bienko *(Principal, IBM Global Sales Enablement)* introduces IBM watsonx Code Assistant's generative AI capabilities and lays the groundwork for the hands-on training that will follow. *[6 min]*

i. Automation is indispensable to modern IT strategy

Despite the innovations and advancements made in the domain of automation, IBM sellers and partners know first-hand from discussions with clients that many businesses are still struggling to keep up with their IT operations.

The rapid pace of technological innovation— in particular, areas such as AI and machine learning —are obviously challenging for any organization to strategize and plan around. But smaller, more practical challenges also stand in the way of these businesses. The fact remains that IT operations, and wrangling those operations in an efficient and streamlined manner, remains a difficult problem to solve.

Three primary pain points that IBM consistently hears from the marketplace include: an ever-increasing skills gap in IT management; that Day 2 operations continue to be labor-intensive, mostly manual endeavors; and that the complexity of the systems needing to be managed are out-pacing many organization’s ability to adapt.

All of these pain points are potential automation challenges to be solved. Each of them impedes a company's ability to move quickly and adapt for the future. And as such, for many IBM clients, solving these automation challenges have become an indispensable element in their strategy to modernize IT.

UNPRECEDENTED RATE OF GENERATIVE AI ADOPTION

Even though generative AI is relatively new, the widespread popularity of ChatGPT has created significant interest in the notion of large language models (LLMs) and foundation models (FMs) — and what they can do for business. It took quite some time for enterprises to start moving toward traditional AI.

In contrast, generative AI has experienced massive early adoption: 80% of enterprises are already working with, or planning to leverage FMs, and plan to adopt generative AI in their use cases and workflow. Moreover, the following data points to an ever-growing adoption trend for generative AI:

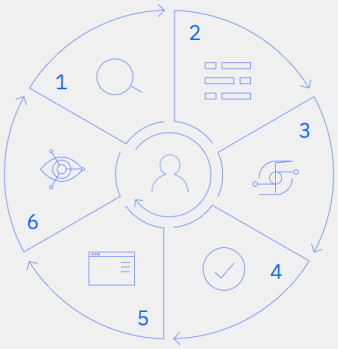
- `Scale Zeitgeist 2023 AI Readiness Report` notes that with the companies they reviewed, 21% have generative AI models in production; 29% are experimenting with generative AI and another 31% are planning to work with generative AI models; a total of 81% are either working with or planning to work with generative AI models
- `Goldman Sachs` has estimated that generative AI will have a very deep economic impact – raising global Gross Domestic Product (GDP) by 7% within 10 years, reflecting the technology's huge potential.
- `Boston Consulting Group (BCG)` noted that generative AI is expected to represent 30% of the overall market by 2025

ii. Generative AI-assisted code lifecycle management achieves what LLMs alone cannot

Following the debut of OpenAI's ChatGPT, the marketplace has been awash with competing large language model (LLM) and generative AI-based assistants. It's one thing to train and deploy an LLM; it's another thing entirely to make it applicable and tangibly beneficial for business.

What separates IBM watsonx Code Assistant (WCA) offerings from competing vendors in the marketplace? The design and implementation of WCA is purposely built to assist, using generative AI, software and code lifecycle management.

In short, generative AI-assisted code lifecycle management helps to achieve what large language models cannot achieve on their own. It is what distinguished WCA from other code assistants in the marketplace today.



1. **Understand** application and runtime environments
2. **Plan** based on analysis of application code
3. **Transform** code with optimized design and architecture
4. **Validate** outcomes with auto-generated unit tests
5. **Deploy** with automated processes
6. **Maintain** with runtime insights

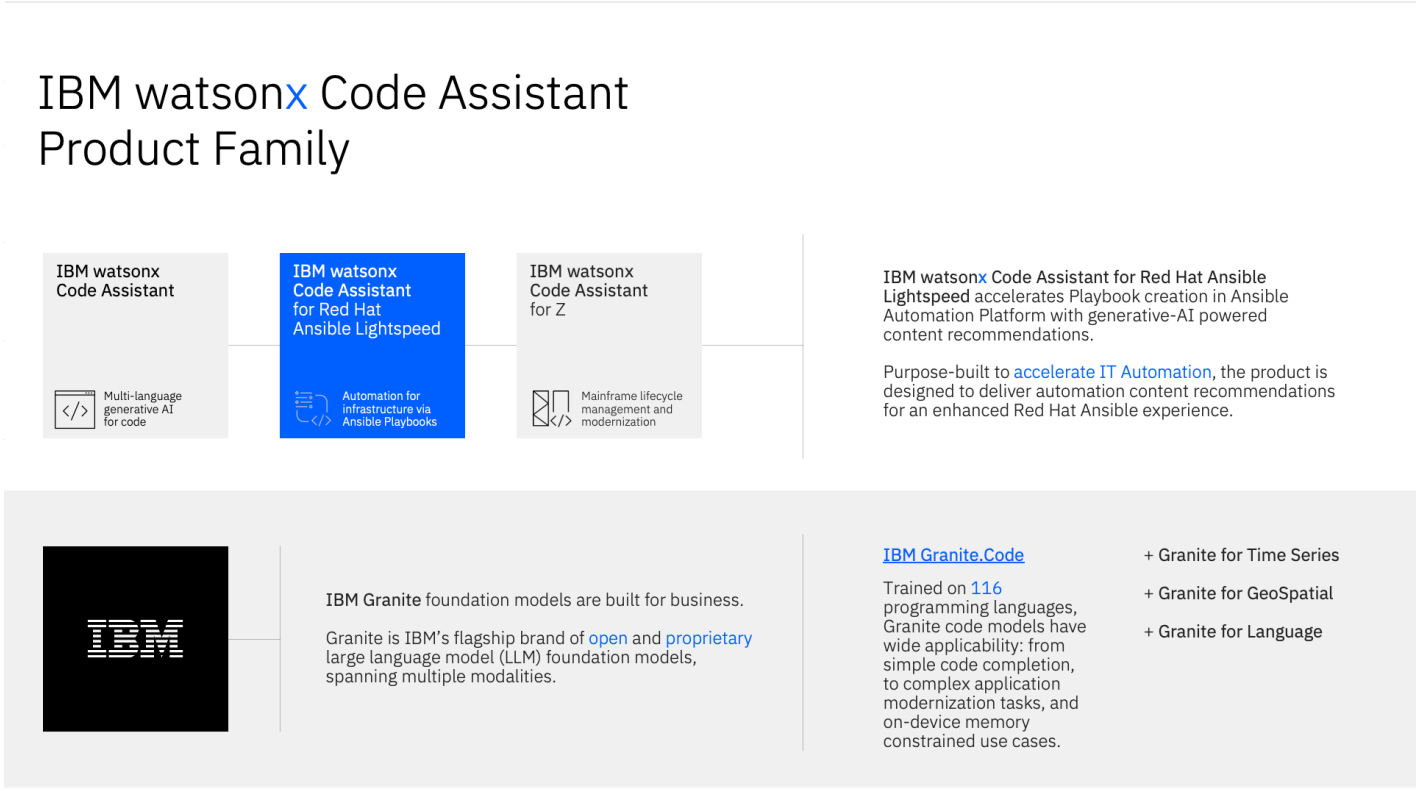
Generative AI-assisted
code lifecycle management
achieves what LLMs alone cannot →

1. Code lifecycle management begins with **understanding** client code, through training across a myriad of programming languages and specializations in paradigms such as Ansible Automation Platform, and applies that understanding across a client's application and runtime environments.
2. Users are able to **plan** next steps based on generative AI analysis of their existing application code.
3. Operations teams can rapidly **transform** their codebases with optimized design and architecture that is recommended according to IBM Granite's best-practice models.
4. Administrators can **validate** the outcomes with automatically generated unit tests.
5. Afterwards, they can **deploy** those services and applications using automated processes like Ansible's automation engine.
6. Over the course of that application or code's lifecycle, generative AI can **maintain** healthy operations with runtime insights.

iii. Introducing the IBM watsonx Code Assistant product family

IBM watsonx Code Assistant is the flagship offering in a suite of generative AI code assistant products, which also include offerings for Ansible Automation Platform (IBM watsonx Code Assistant for Red Hat Ansible Lightspeed) and IBM Systems modernization (IBM watsonx Code Assistant for Z).

These solutions accelerate software development tasks with AI-powered capabilities including context-aware code generation, explanation, documentation, translation, and unit test generation. It does so while maintaining the principles of trust, security, and compliance with regards to IBM client's data and intellectual property. Developers and IT Operators can utilize WCA to speed up application modernization efforts and generate Ansible-based automation jobs to rapidly scale out (or scale up) IT environments.

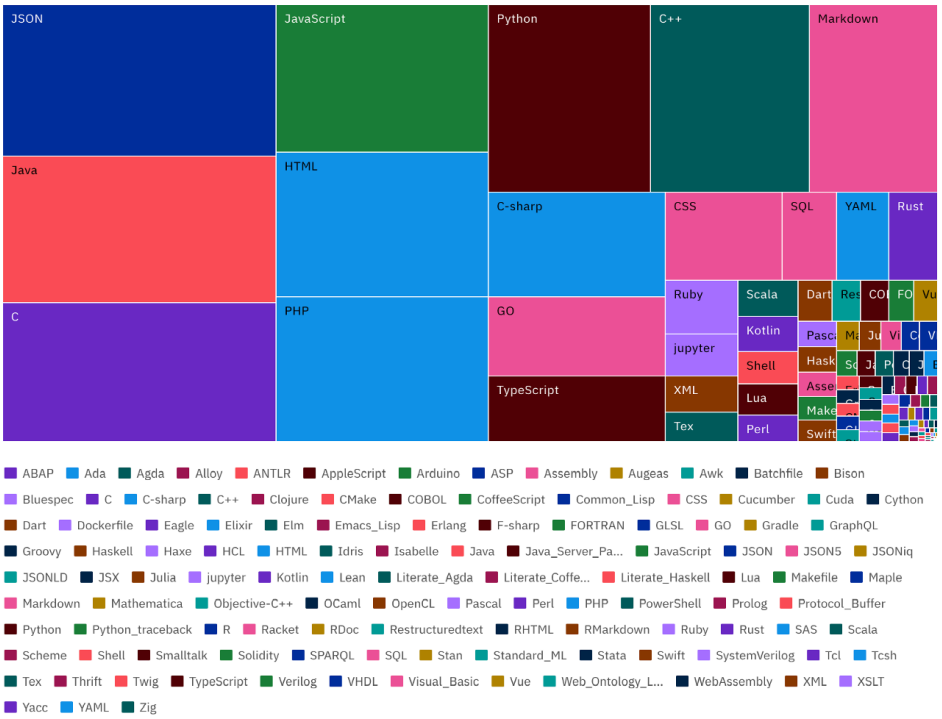


IBM watsonx Code Assistant products are powered by **IBM Granite** foundation models that include state-of-the-art large language models designed for code. For offerings such as WCA for Ansible Lightspeed and WCA for Z, bespoke code models — tailored to working with Ansible Automation Platform and COBOL-to-Z use cases, respectively —are invoked. Universally true for all of the watsonx Code Assistant offerings is that they are geared towards helping IT teams create high-quality code using AI-generated recommendations, based on natural language requests or existing source code. These AI models, and the recommendations they generate, are seamlessly integrated via extensions with the world's most popular developer integrated development environments (IDE), including Visual Studio Code and Eclipse.

Granite is IBM's flagship brand of open and proprietary LLMs, spanning multiple modalities. Granite models exist for code, languages, time series, and GeoSpatial — with additional modalities expected in future.

IBM Granite Code Model

Trained on +3 trillion tokens of code, spanning 116 programming languages.



IBM Granite code models are a series of decoder-only models for code generative tasks, trained with code written in 116 different programming languages. The Granite code models family consists of models ranging in size from 3 to 34 billion parameters, in both a base model and instruction-following model variants. These models have a range of uses, from complex application modernization tasks to on-device memory-constrained use cases.

The larger the block size for a particular language on this chart, the larger percentage of training corpus data of that language was used to train the Granite code model. Languages and formats such as Java, C, JSON, JavaScript, HTML, and PHP are subjects in which the model “Majors” and excels. Other languages such as Ruby, SQL, and Swift could be considered “Minors” where the generalized code model can work with the language, but has less training data to base those recommendations on. These percentages and training data volumes will continue to evolve as the Granite code models mature.

WATSONX CODE ASSISTANT vs. WCA FOR ANSIBLE LIGHTSPEED?

For those familiar with other IBM watsonx Code Assistant offerings— such as WCA for Red Hat Ansible Lightspeed and WCA for Z —the generalized code model approach, as seen here, differs from the specialized code model approach of those two aforementioned offerings.

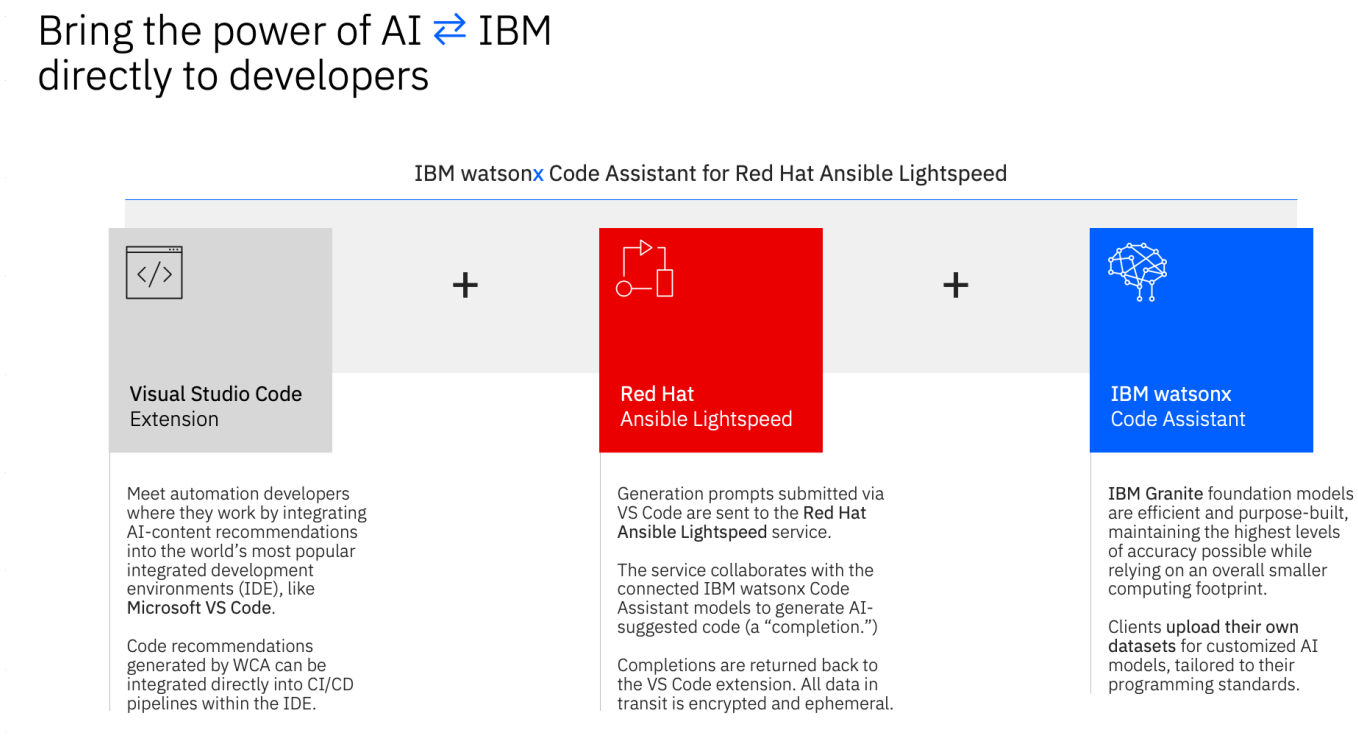
- The **WCA for Ansible Lightspeed** flavor of IBM Granite code models specializes (Majors) only in Ansible Playbooks and YAML
- Similarly, the IBM Granite code model used by **WCA for Z** specializes in transforming COBOL mainframe code into modernized Java code

Ansible Playbooks (YAML) and mainframe (COBOL) code are both supported (Minor) languages for the generalized IBM Granite code models— and therefore are supported by IBM watsonx Code Assistant —but if a client wishes to specialize in those particular languages and frameworks, they would be well advised to utilize the bespoke *WCA for Ansible Lightspeed* and *WCA for Z* offerings, respectively, to do so.

iv. Solution architecture of IBM watsonx Code Assistant for Red Hat Ansible Lightspeed

IBM watsonx Code Assistant for Red Hat Ansible Lightspeed meets developers where they are: with a rich plugin via VS Code extensions, where developers input their prompts directly in the code editor. Prompts are sent to the Ansible Lightspeed service, and the service sends a suggestion back (a *completion*) that's powered by IBM Granite LLMs for code.

It is important to note that all data in transit is encrypted and ephemeral so users can be confident and have trust in the security of the service during this exchange. In terms of data security, client Ansible playbooks and customized models that they may potentially have are stored in client-owned Cloud Object Storage and are not shared with IBM, Red Hat, or any other clients.



> In order to utilize IBM watsonx Code Assistant for Red Hat Ansible Lightspeed, a client must have an existing license for Red Hat Ansible Automation Platform (the "red tile" component in the center of the diagram), as well as a license for IBM watsonx Code Assistant for Red Hat Ansible Lightspeed (the "blue tile" on the right of the diagram).

Generative AI has recently demonstrated proficiency in creating syntactically correct and contextually relevant application code in a variety of programming languages. For example, if trained on a large dataset of Ansible Playbooks, generative AI models can be fine-tuned to understand the nuances of Playbook syntax and structure. An enterprise organization with dozens or hundreds of Playbooks within their IT estate today would have a rich corpus of training data on-hand that could be used to fine-tune AI models that are tailored to the automation needs and programming style or standards of that particular company.

WHAT ARE PLAYBOOKS?

Ansible Playbooks instruct Ansible's automation engine on how to execute tasks in a step-by-step manner. Playbooks defines roles, tasks, handlers, and other configurations; in turn, these attributes allow developers and users to codify complex orchestration scenarios. Conceptually, think of a Playbook as a recipe book for system administration: each recipe (or Playbook) spells out the steps required to achieve a particular system state or to complete a given operation.

One of the standout features of Ansible Playbooks is that they are *idempotent*: executing Playbooks multiple times on the same system won't create additional "side effects" (unintended operations or creation of unwanted artifacts) after the first successful run. This ensures consistency and reliability across deployments of the Red Hat Ansible Automation Platform (**AAP**).

As you will see throughout the hands-on training material, generative AI models provide a natural language prompt to users which in turn is understood and translated by the AI models into the necessary Ansible Task code. For example, a user might describe a desired system state in plain language (*"I want a Playbook to install and start an Apache web server"*) and the model will generate the appropriate Ansible Tasks for a Playbook.

All of this is achieved without physically writing code or requiring much programming expertise. Not only does this speed up the automation process by cutting the time needed to author Playbooks, but it also democratizes access to automation in general. Even those within a company with limited Ansible or programming expertise will be able to produce effective Playbooks. There are plenty of caveats of course, and thorough validation and testing of AI-generated code will be needed before being put into production. However, the productivity gains and broadening of skillsets within an organization can be tremendous. And as a whole, generative AI brings the original goals of Red Hat Ansible Automation Platform (the democratization of automation for everything) that much closer to a reality.

---

v. Lab objectives

The material covered for this hands-on training is intended to prepare IBM sellers and business partners with the skills necessary to create Ansible automation tasks using the generative AI capabilities of WCA. The curriculum will leverage WCA's generative AI code recommendations for automating cloud-based and infrastructure-based automation tasks. In-depth explanations accompanying Ansible Playbook templates will also explain:

- How WCA uses **natural language prompts**, as well as Ansible Playbook contents, to generate contextually-aware Task code recommendations
- **Post-processing** capabilities that refine the generative AI suggestions into syntactically correct code (adherent to best practices)
- How WCA provides **content source matching** attribution and "explainability" for all AI-generated content
- Leveraging WCA's **model tuning** capabilities to tailor content and code recommendations to an organization's standards, best practices, and programming styles

---

vi. Next steps

The module ahead will outline the evaluation criteria for IBM sellers and business partners. Afterwards, you will setup your local environment with the necessary pre-requisites for getting started with the hands-on material.

i. Evaluation Criteria for IBM Technical Sellers and Business Partners

To receive the Level 3 badge (*Generative AI for Code with watsonx Code Assistant Technical Sales Intermediate*), IBMers and Business Partners must demonstrate mastery of the skills learned throughout the various modules of these hands-on labs and coursework. Level 3 skills requirements— and the way participants will be evaluated —differs depending on job role.

IBM TECHNICAL SELLERS

IBM Sales and Tech Sales must develop and record a **Stand & Deliver** presentation, which will be uploaded to the IBM Stand & Deliver platform for evaluation. This video is intended to simulate your delivery of a "live" demo in front of a client — on camera.

IBMers will have flexibility in defining a hypothetical client, the pain points that customer has, and the goals they aspire to. The recording will then cover the seller's hands-on demonstration and pitch to the client of the value of the IBM solution using the environments and techniques of this lab.

BUSINESS PARTNERS

Business partners must pass a **skills evaluation quiz** after completing the hands-on portion of the course. The quiz consists of multiple choice questions, with four possible responses (and only one correct answer) for each question. Participants must pass the quiz with a grade of 80% or higher.

The quiz questions will ask you about on-screen text or descriptions that come up as you work through the lab guide. Answers to quiz questions can only be determined by carefully following the instructions of the hands-on lab.

---

ii. *IBMer* Stand & Deliver Assessment

IBMers — SUBMIT RECORDINGS HERE

Submit your Stand & Deliver recording online using IBM YourLearning.

Hands-on demonstrations of IBM watsonx Code Assistant for Red Hat Ansible Lightspeed are available on Seismic, as well as embedded within the various modules of this documentation. The authors highly recommend tailoring the Stand & Deliver to your own presentation style and expertise. Captivating and delighting clients with your unique delivery is paramount!

- Introduction *[6 min]*
- Generating Code *[10 min]*
- Content Source Matching and Post-Processing *[10 min]*
- Task Description Tuning and Model Customization *[15 min]*
- Stand & Deliver with Paul Zikopoulos *[8 min]*

The evaluation criteria described below **only applies to IBMers**, who must record a Stand & Deliver to receive credit for this Level 3 course. By default, IBMers will be evaluated by their First Line Manager (FLM) — although they may request another manager to evaluate their Stand & Deliver, if appropriate. Instructions on how to submit a Stand & Deliver are included within the activity on YourLearning.

IBM Technical Sellers need to include **all** six of the following elements in their Stand & Deliver recording to receive a Level 3 badge:

1. Seller articulated their client's pain point(s) and the value proposition of using *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed*.
2. Seller highlighted use cases for *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed*.
3. Seller demonstrated and discussed several of the key differentiated capabilities of *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed* that deliver on the value proposition on point one.
4. Seller highlighted benefits to their client (this is the why the client can't live without these benefits section).
5. Seller highlighted benefits to their client's customers (what will the client be able to deliver to their customers that they could not without this product).
6. Seller closed the demonstration with a call to action for their client that could include: a workshop, a deeper dive into the product meeting, or a Proof of Experience (PoX).

---

iii. *Business Partner* Quiz Assessment

> ⚠ **PARTNERS — COMPLETE ASSESSMENT HERE**
>
> Complete your assessment online using IBM Training:
>
> https://learn.ibm.com/mod/subcourse/view.php?id=284786

The skills evaluation quiz **only applies to business partners** – IBMers must record a Stand & Deliver. The quiz consists of multiple choice questions, with four possible responses (and only one correct answer) for each question. The quiz questions will ask you about on-screen text or descriptions that come up as you work through the lab guide. Participants must pass the quiz with a grade of 80% or higher.

Answers to the quiz can only be determined by carefully following the instructions of the hands-on lab.

---

iv. Next steps

In the following section, you will prepare your hands-on lab environment with the necessary services and configurations.

> If you require assistance or run into issues with the hands-on lab, help is available.
>
> - **Environment issues:** The lab environment is managed by IBM Technology Zone. Opening a support case ticket is recommended for issues related to the hands-on environment (provisioning, running, and so on.)
> - **Documentation issues:** If there is an error in the lab documentation, or if you require additional support in completing the material, open a thread on the #wca-ansible-techzone-support Slack channel.
> - **Product questions:** For questions related to IBM watsonx Code Assistant capabilities, sales opportunities, roadmap, and other such matters, open a thread on the #watsonx-code-assistant Slack channel.
>
> Frequently asked questions and troubleshooting steps are documented below.

i. Reserving the lab environments

Before getting started with *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed* (**WCA**), an environment must be reserved and deployed via the *IBM Technology Zone* (**ITZ**).

You will require access to the ITZ in order to reserve your environment and complete the lab. If you do not yet have access or an account with the ITZ, you will need to register for one.

There are **TWO** environments that you must reserve from ITZ:

- `Request a Red Hat Account`: responsible for generating unique access credentials for *IBM watsonx Code Assistant* and *Red Hat Ansible Lightspeed* authorizations
- `WCA for Ansible Essentials Plan - Visual Studio Desktop 1.4`: virtualized machine prepared with Visual Studio Code and lab demonstration scripts pre-installed; you will authenticate within this environment using the Red Hat account requested from ITZ

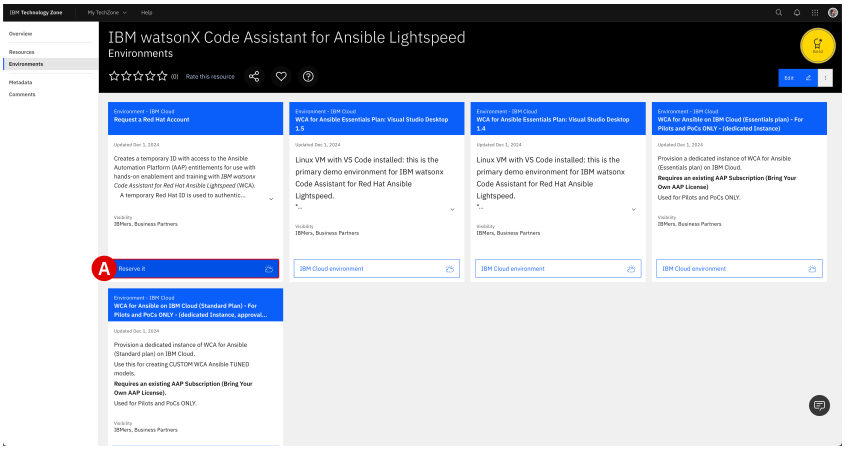
Follow along with the instructions below to request and configure these environments.

---

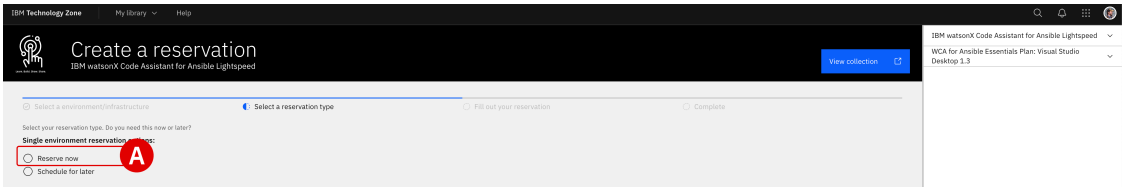
1. Click the IBM Technology Zone link below. Locate the **Request a Red Hat Account** tile, hover over the **IBM Cloud environment** button with your cursor, and then click **Reserve it**[A]:

   > **URL:** https://techzone.ibm.com/collection/ibm-watson-x-code-assistant-for-ansible-lightspeed/environments

   > **FULLSCREEN IMAGES**
   >
   > Click on any of the screenshots within this documentation to enlarge the image.



2. From the *Single environment reservation options*, select **Reserve now**[A].



3. Supply additional details about your ITZ reservation request:

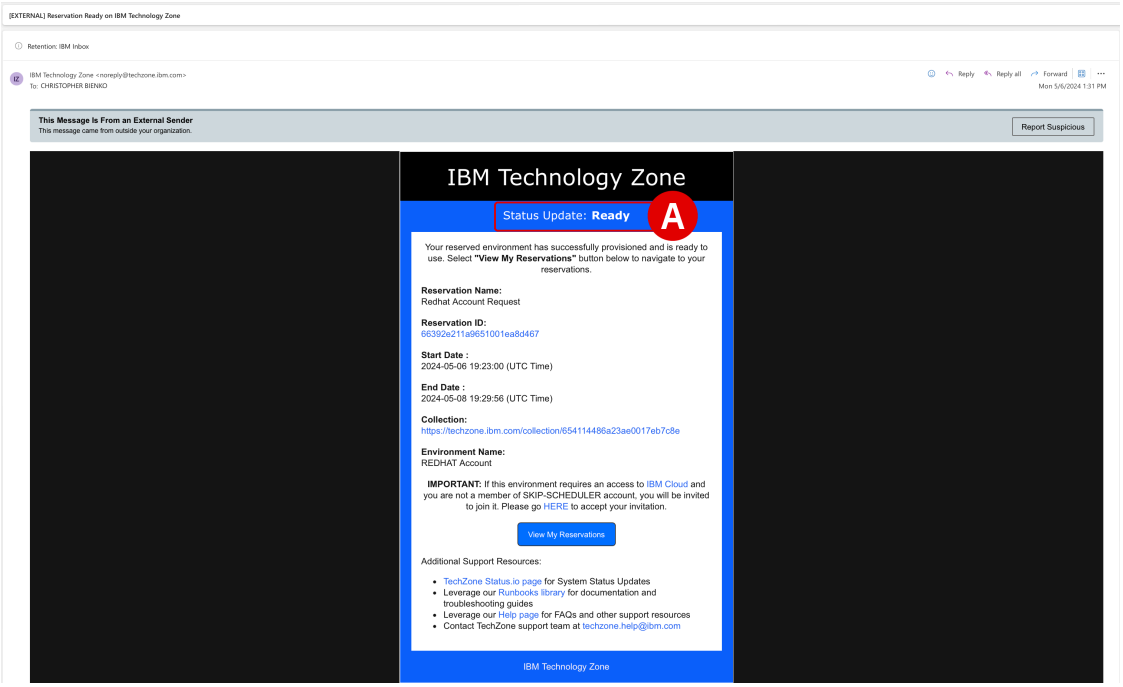
Field	Value
`Name`	Give your reservation a unique name.
`Purpose`	If reserving for L3 training, select *Education*. If delivering a PoC, select *Pilot* and provide a Sales Opportunity number.
`Describe`	If reserving for L3, enter *WCA for Ansible Lightspeed training*. If delivering a PoC, enter the PoC and client details.
`Preferred Geography`	Select the region and data center geographically closest to your location.
`End Date and Time`	Select a time and date for when the reservation will expire.

When satisfied, verify that you agree to the *Terms and Conditions* for the environment and finalize your reservation request by clicking **Submit**.

> **PROVISIONING TIMES**
>
> Red Hat account creation take approximately **5-10 minutes** to complete from the time that you click submit. If you navigate to the **My Reservations** tab of the ITZ, you can monitor the progress of your reservation. Wait for the ITZ reservation to be marked as *"Ready"* before attempting to start the lab.

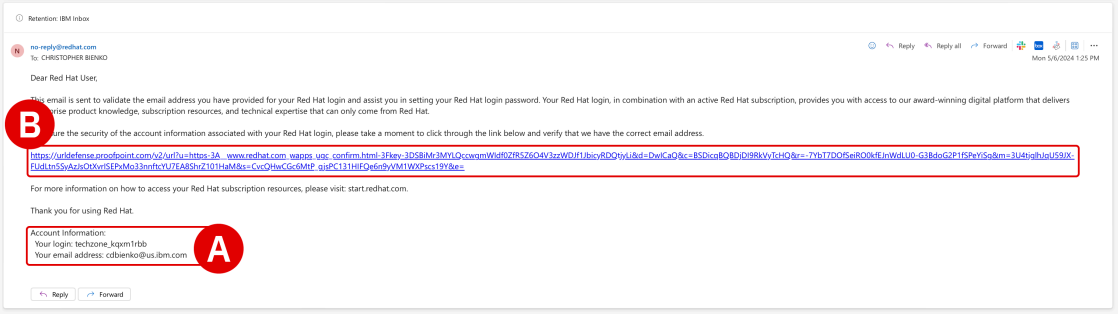
4. When the Red Hat account request has been processed by IBM Technology Zone, you will receive a **pair** of emails: one from ITZ and a second from Red Hat.

   - `Reservation Ready on IBM Technology Zone` : You can ignore the contents of this email, as the relevant account and licensing information are contained in the Red Hat email. Confirm that the ITZ email states that **Status Update: Ready**[A].
   - `Red Hat Login Email Verification` : This email, addressed from a `no-reply@redhat.com` account[B], contains the resources necessary for accessing your uniquely-generated Red Hat credentials. The lab guide steps that follow will instruct you on how to set those up and how to use them for accessing your *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed* entitlements inside VS Code.



5. With the `Red Hat Login Email Verification` email open, take note of two essential pieces of information:

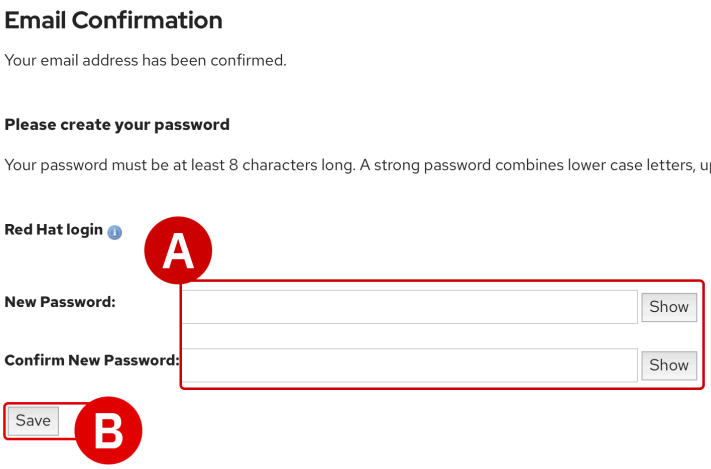
- Locate the **Account Information**[A] details at the bottom of the email. `Your login:` is the unique Red Hat account assigned for you by ITZ. `Your email address:` will be the address associated with your ITZ account. **Record** both to a notepad for reference later.
- Click the **URL**[B] located within the body of the invitation email to finalize your account registration with Red Hat.



---

6. An *Email Confirmation* page will load within your web browser.

- Note that the value of `Red Hat login` is the same as the one recorded in *Step 5*
- **Create** a new `Password` [A] and record this to a notepad for reference later
- When ready, click **Save**[B] to finalize registration



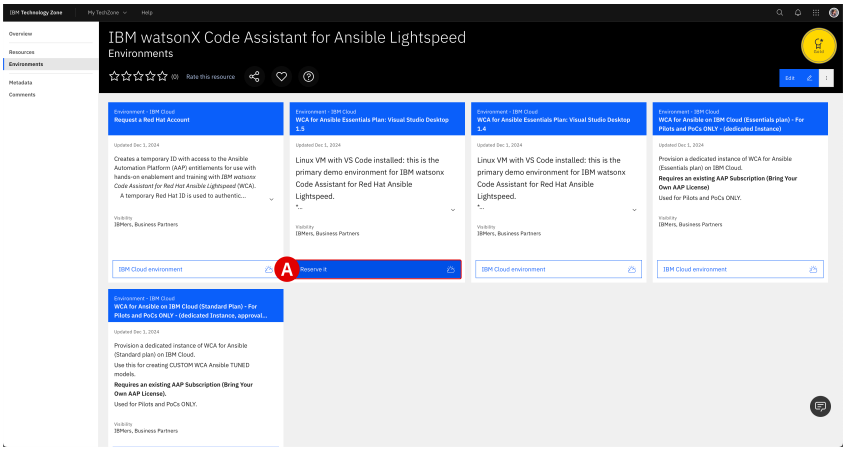
REGISTRATION IS REQUIRED

- If you already have a personal account with Red Hat, **you must still register for a new account** using the invitation URL provided
- **Do not** attempt to use a personal Red Hat account in the later steps of the *Setup & Troubleshooting* guide, as that account will not have access to the WCA services needed to perform the training
- Red Hat accounts created for this training will automatically be de-authorized and deleted by IBM Technology Zone after the reservation period has ended

7. Now you must request your second ITZ environment, this time for the virtualized machine (VM) environment. **Open** the IBM Technology Zone reservation link below:

> **URL:** https://techzone.ibm.com/collection/ibm-watson-x-code-assistant-for-ansible-lightspeed/environments

- Locate the **WCA for Ansible Essentials Plan: Visual Studio Desktop 1.5** tile
- Hover over the **IBM Cloud environment** button with your cursor
- Click **Reserve it**[A] to continue



8. From the *Single environment reservation options*, select **Reserve now**.

9. Supply additional details about your ITZ reservation request:

Field	Value
`Name`	Give your reservation a unique name.
`Purpose`	If reserving for L3 training, select *Education*. If delivering a PoC, select *Pilot* and provide a Sales Opportunity number.
`Describe`	If reserving for L3, enter *WCA for Ansible Lightspeed training*. If delivering a PoC, enter the PoC and client details.
`Preferred Geography`	Select the region and data center geographically closest to your location.
`Customer Data`	Select *No, I will not be using customer data* if using for education purposes.

Field	Value
`End Date and Time`	Select a time and date for when the reservation will expire.
`VPN Access`	Set to *Disabled*.

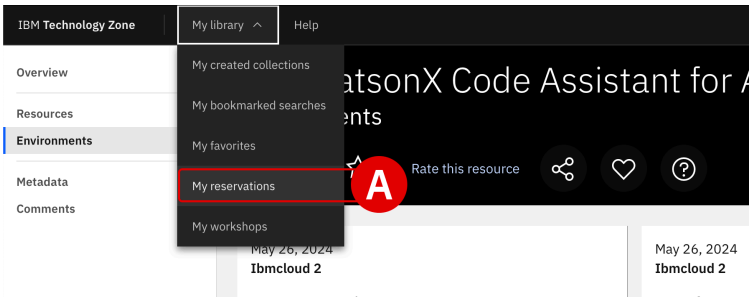
When satisfied, verify that you agree to the *Terms and Conditions* for the environment and finalize your reservation request by clicking **Submit**.

> **PROVISIONING TIMES**
>
> Red Hat account creation take approximately **15-30 minutes** to complete from the time that you click submit. If you navigate to the **My Reservations** tab of the ITZ, you can monitor the progress of your reservation. Wait for the ITZ reservation to be marked as *"Ready"* before attempting to start the lab.

---

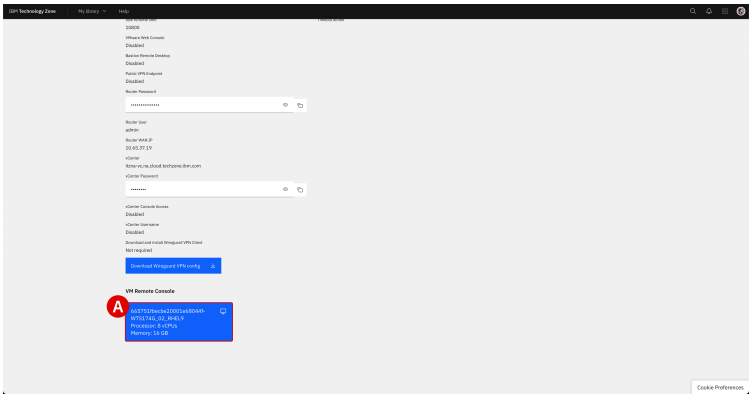
ii. Accessing the VM

10. Once the ITZ reservation has been marked as *"Ready"*, access connection details for the environment by either clicking the shortcut in the ITZ email or by drilling down into the **My Reservations** tab[A] on the ITZ web portal.



---

11. Scroll down to the bottom of the reservation page and click the blue **VM Remote Console** button[A] to launch the VM interface.

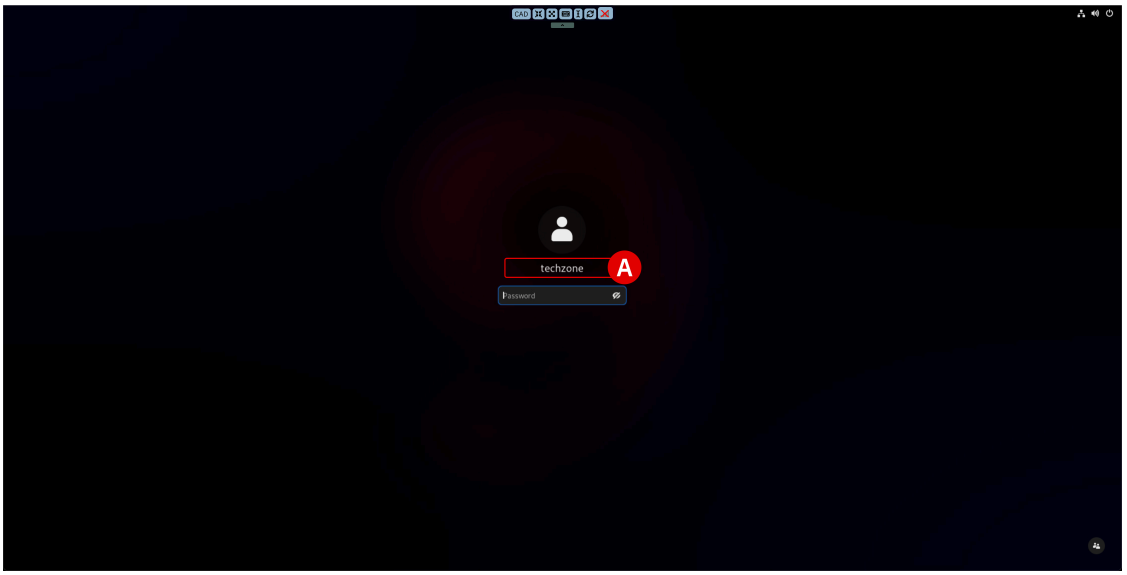


12. Select the `techzone` [A] login when prompted.

    - Supply the following for the password:

      ```
      IBMDem0s!
      ```

    - Hit `Enter ⏎` to complete the VM login



13. Once you have successfully authenticated, click the **Activities**[A] button in the top-left corner of the interface to pull open the list of available applications from the bottom of the screen.

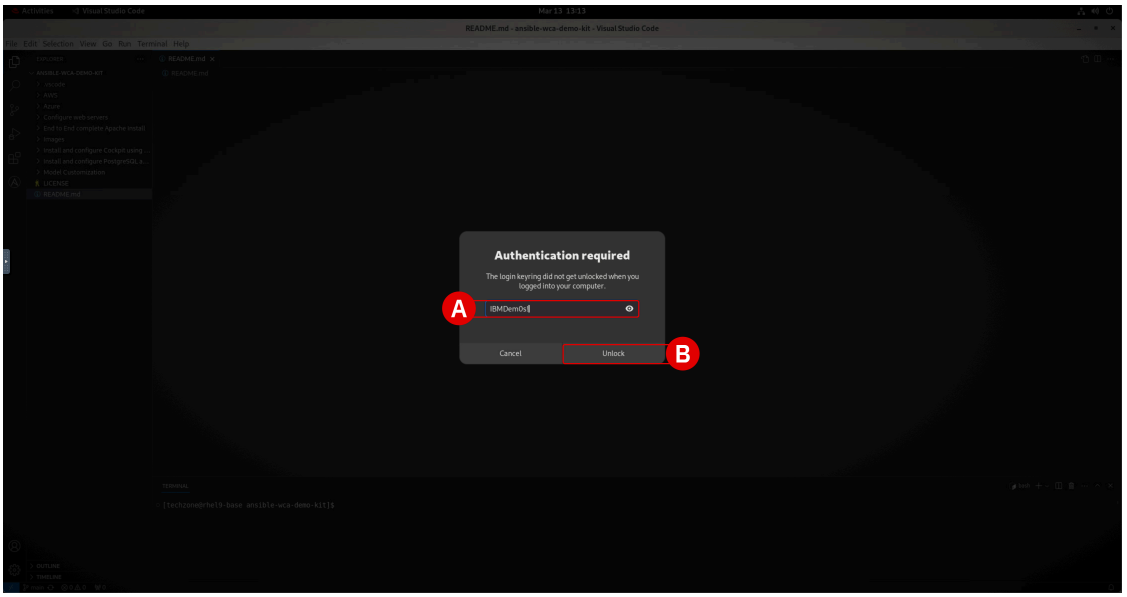
    - Click the **Visual Studio Code**[B] application shortcut at the bottom of the desktop to start up the service.
    - If you wish to copy and paste instructions directly from the lab documentation into the virtual machine (VM), it is recommended that you open the GitHub instructions **inside** the VM's web browser (Firefox). This will allow you to copy instructions to the VM's clipboard and paste instructions inside the VS Code editor.
    - You may access the web browser at any time by clicking the **Activities**[A] button and then launching **Firefox**[C], as shown.

14. VS Code will load and then present you with an *Authentication Required* splash screen.

- Enter the same password[A] used to log into the VM:

```
IBMDem0s!
```

- Click **Unlock**[B]

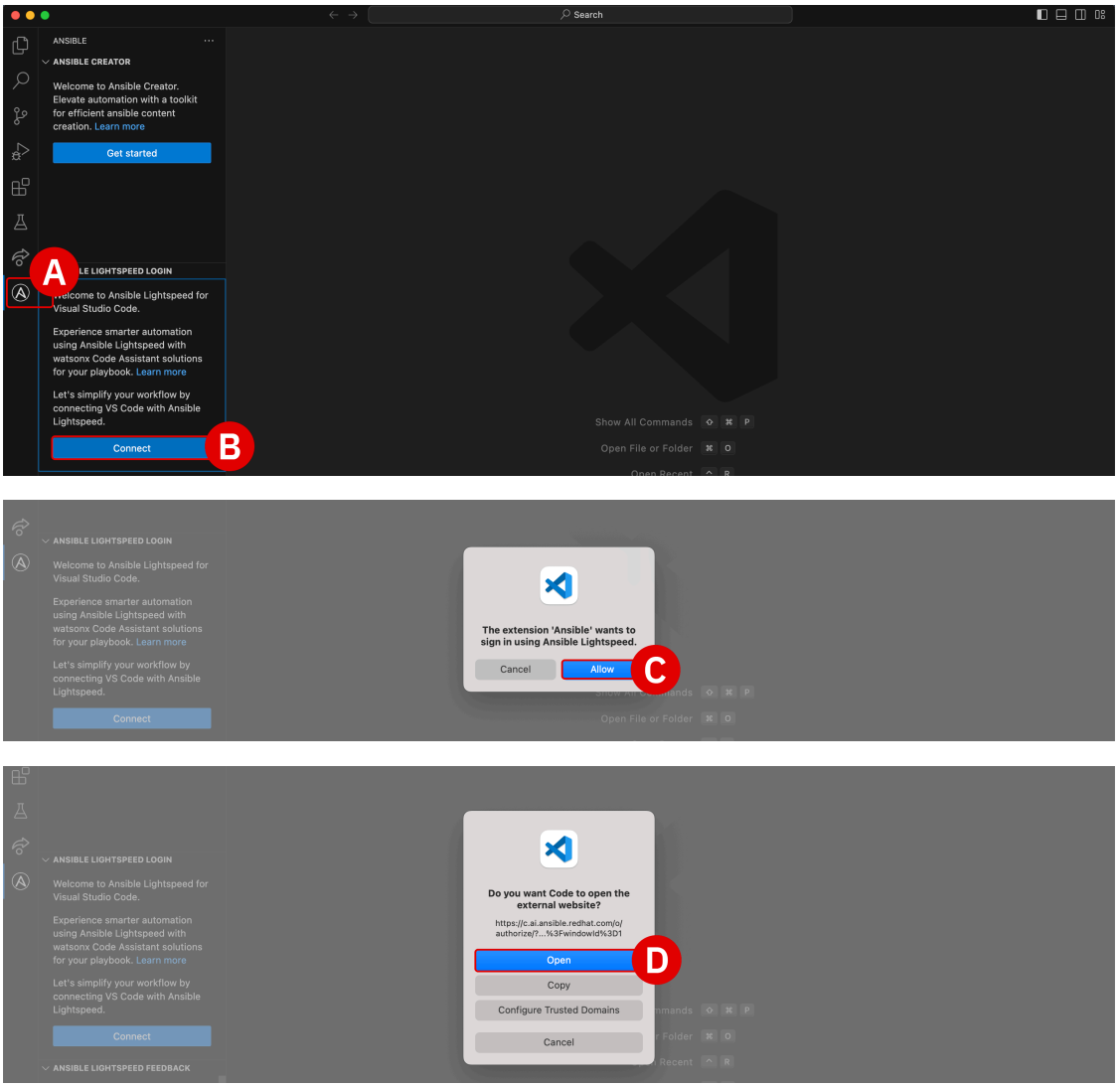


iii. Authorizing VS Code with WCA

Now you must authorize the VS Code environment for use with WCA, using the Red Hat account details that were generated for you in *Steps 5-6*.

15. Within your VS Code environment click the **Ansible** plugin[A] (denoted by the `A` logo) on the left-hand side of the interface.

    - Two panels will open along the left side of the interface
    - Within the *Ansible Lightspeed Login* panel, click the blue **Connect** button[B]
    - `The extension Ansible wants to sign in using Ansible Lightspeed`: click **Allow**[C]
    - `Do you want Code to open the external website?`: click **Open**[D]



---

16. A web browser will load with the header `Log in to Ansible Lightspeed with IBM watsonx Code Assistant` — this is where you will supply your registration details recorded in *Step 6* in order to authenticate the VS Code plugin with WCA.

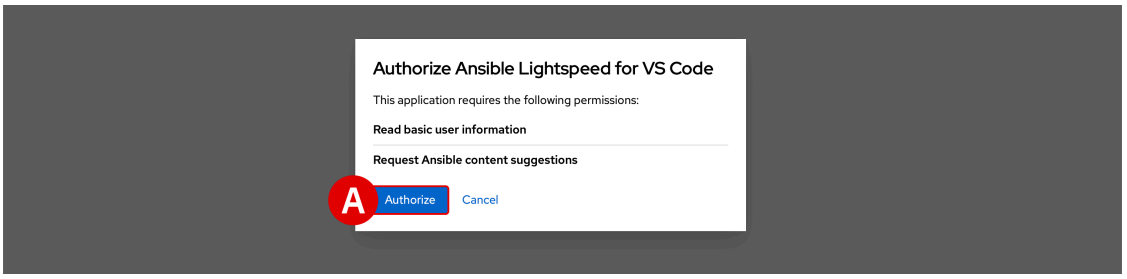
    - Click the **Log in with Red Hat** button
    - Provide the `Username` and `Password` recorded in *Step 6* of this module

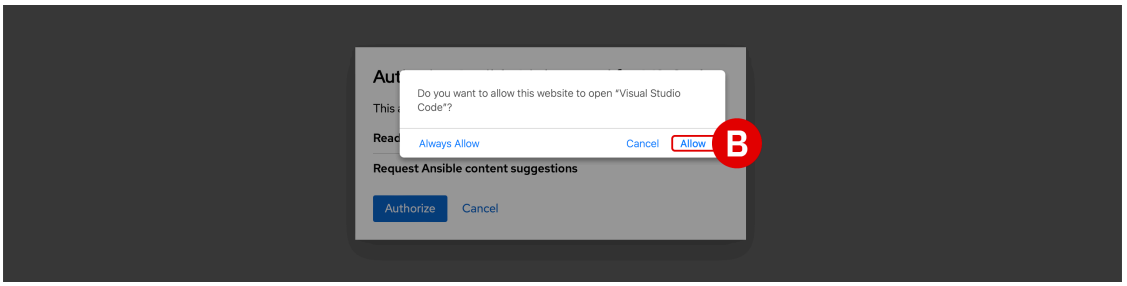
- Click **Submit** to continue
- When prompted for **We need a little more information**, set **Job Role** to `Student`
- Click the **Submit** button to finalize your account activation

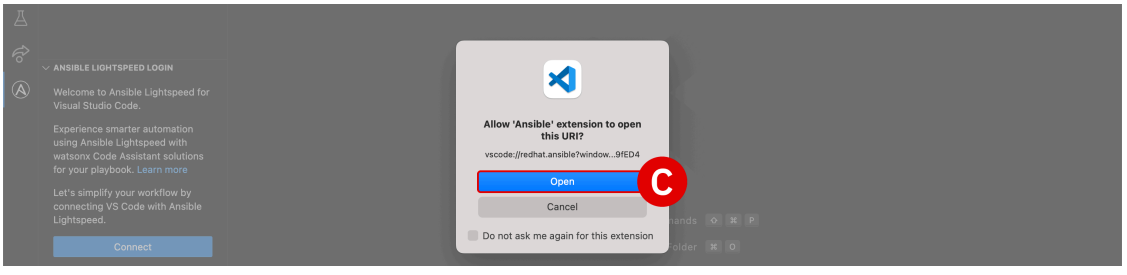
> **COPY AND PASTE INSTRUCTIONS INTO A VIRTUAL MACHINE**
>
> As you are running the lab environment inside a virtual machine (VM), it is not possible to "paste" lab instructions or information from your local machine's clipboard directly into the VM.
>
> If you wish to copy and paste instructions directly from the lab documentation, it is recommended that you open the GitHub instructions **inside** the VM's web browser (Firefox). This will allow you to copy instructions to the VM's clipboard and paste instructions inside the VS Code editor.

---

17. After logging in with Red Hat, the web browser will display the prompt to `Authorize Ansible Lightspeed for VS Code`.
    - Click **Authorize**[A]
    - `Allow this site to open the vscode link with Visual Studio Code - URL Handler?`: click **Open URL**



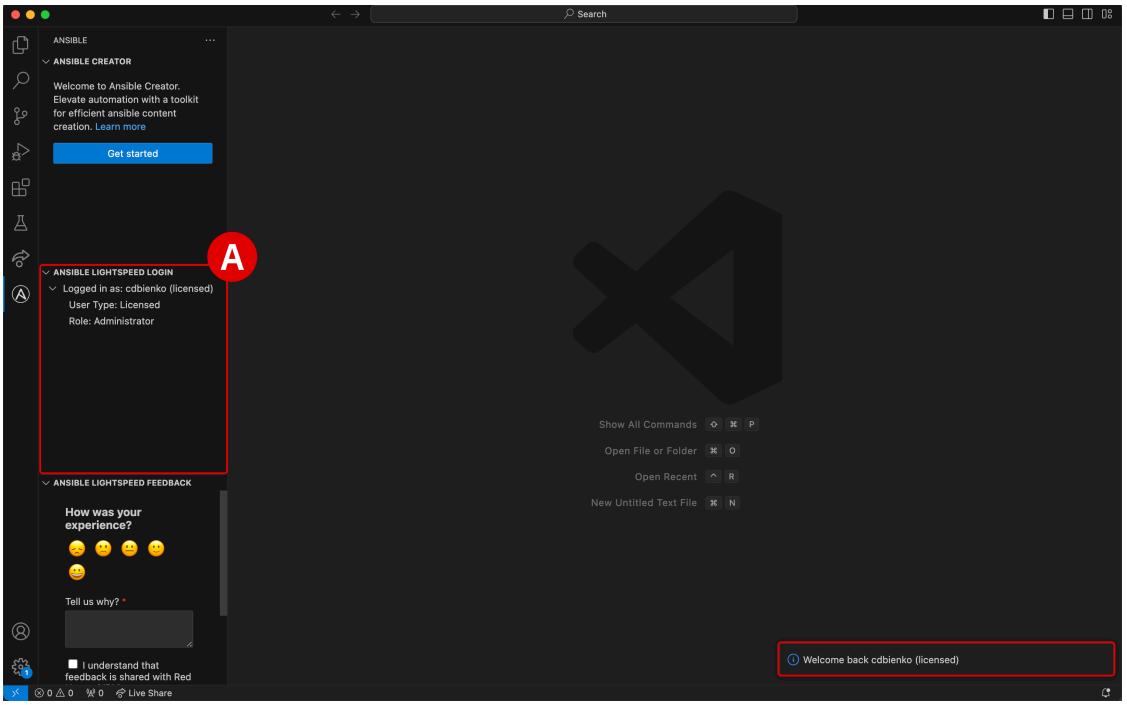




---

18. At this stage, the Ansible extension for VS Code is now authenticated and connected to *IBM watsonx Code Assistant*.
    - Verify[A] that the environment is logged in as your unique `Username` and that the `User Type: Licensed`

- A notification pop-up will also appear in the bottom-right corner of the VS Code interface confirming the successful login



CLICK TO EXPAND — FAILURE TO LOGIN OR TIMED OUT

- If the authentication procedure in *Steps 15-17* takes too long, activation of the plugin will be "timed out" and the VS Code environment will display an error message[B] in the bottom-right corner
- Click the **Connect**[C] button as shown and repeat *Steps 15-17* as before; the login process should run smoother (and faster) on the second attempt

iv. [OPTIONAL] Download demo assets to local machine

19. If you wish to complete the hands-on lab using a local installation of VS Code (instead of using the provided Virtual Machine), you may do so — but you'll need to clone (download) the accompanying demo assets first. To do so, first install the GitHub CLI library.

    - Execute the following command within a Terminal console to clone (via `git`) the supporting demo assets repository (`ansible-wca-demo-kit.git`) to your local machine:

      ```
      git clone https://github.com/chetan-hireholi/ansible-wca-demo-kit
      ```

    - Open the local `ansible-wca-demo-kit` folder within VS Code to access the demo assets for the remaining sections of the lab

---

v. Troubleshooting and support

> If you require assistance or run into issues with the hands-on lab, help is available.
>
> - **Environment issues:** The lab environment is managed by IBM Technology Zone. Opening a support case ticket is recommended for issues related to the hands-on environment (provisioning, running, and so on.)
> - **Documentation issues:** If there is an error in the lab documentation, or if you require additional support in completing the material, open a thread on the #wca-ansible-techzone-support Slack channel.
> - **Product questions:** For questions related to IBM watsonx Code Assistant capabilities, sales opportunities, roadmap, and other such matters, open a thread on the #watsonx-code-assistant Slack channel.

As you settle in to the environment and begin your training, you may encounter unexpected warnings or errors. Many of these can be safely ignored or can be easily rectified. This section will serve as a running list of frequently asked questions and troubleshooting techniques. Click on any of the following topics for additional details.

FAILED TO CONNECT TO THE SERVER / `"YOU DON'T HAVE ACCESS TO IBM WATSONX..."`

This warning will occur when the Ansible plugin for VS Code needs to be re-authenticated with WCA. It can occur after an extended period of inactivity or a system restart. For example, if your lab environment is running inside a VM, pausing or restarting the VM may produce this error.

To re-authenticate:

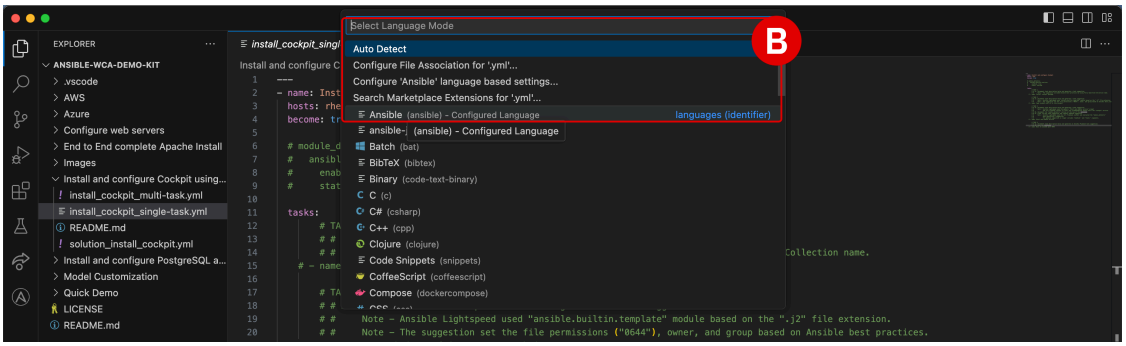
- **Sign out** from the *VS Code* application by clicking the User icon[A] in the bottom-left corner of the interface, hover over your username, and then click **Sign Out**[B]
- **If you are running this environment inside a virtual machine (VM)**, closing and restarting the VM *will not* resolve the issue — you must sign out from the *VS Code* application, not the VM
- Once logged out, follow from *Step 7* of the *Setup & Troubleshooting* to re-authenticate with WCA

CODE RECOMMENDATIONS ARE NOT GENERATING

Ansible Lightspeed and WCA will only generate code recommendations for *Ansible Playbooks* and *YAML* files. VS Code will typically auto-detect the programming language of the document you're working with, but on occassion you may need to manually specify the language. Even if working with a YAML file, you'll still need to specify the language mode as `Ansible` for the Lightspeed plugin to engage.

To set the language mode correctly:

- In the bottom-right corner of the VS Code interface, hover over the **Select Language Mode** toggle[A]
- A console will appear at the top of VS Code with a drop-down list of options[B]
- Click `Ansible` from the suggested languages, or enter the text yourself and hit `Enter ⏎`
- Confirm that the Select Language Mode toggle in the bottom-right corner displays `Ansible`

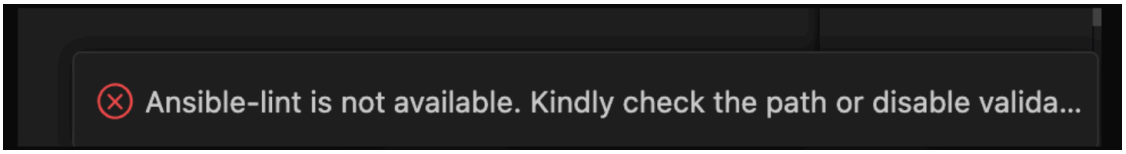


"ANSIBLE-LINT IS NOT AVAILABLE."

`ansible-lint` checks Playbooks for practices and behavior that could potentially be improved and can fix some of the most common ones for you. It will constantly check your Ansible syntax as you type and provide recommendations for how to improve it.

- You can safely **ignore** this error if it occurs during the lab exercises
- If you wish to install `ansible-lint` on your local machine, execute the following instruction within a Terminal console:

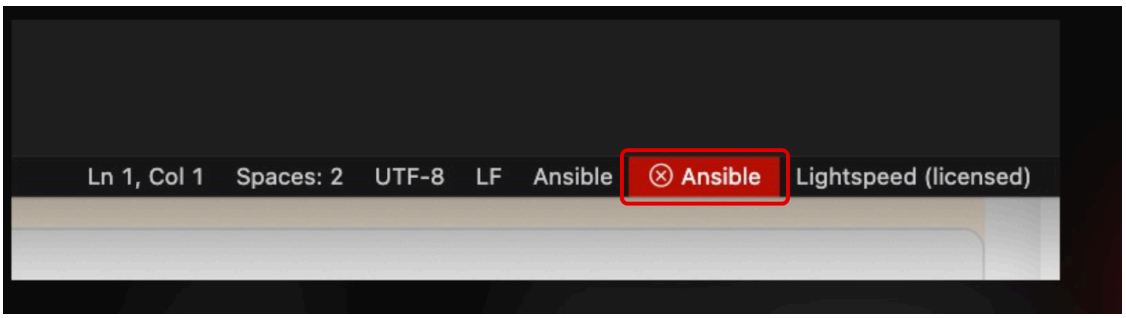
```
python3 -m pip install --upgrade --user ansible-lint
```



RED *ANSIBLE* ICON ALONG BOTTOM-RIGHT INTERFACE

The *Ansible* extension for VS Code will check your local machine to determine if Red Hat Ansible has been installed locally. If you have not set up Ansible (the standalone version) on your local machine previously, this tile will display as red.

You can safely **ignore** this error if it occurs during the lab exercises.

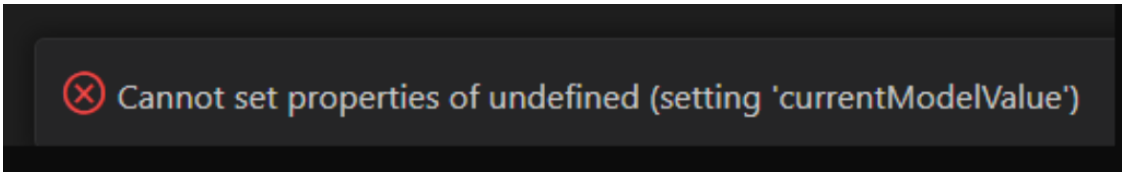


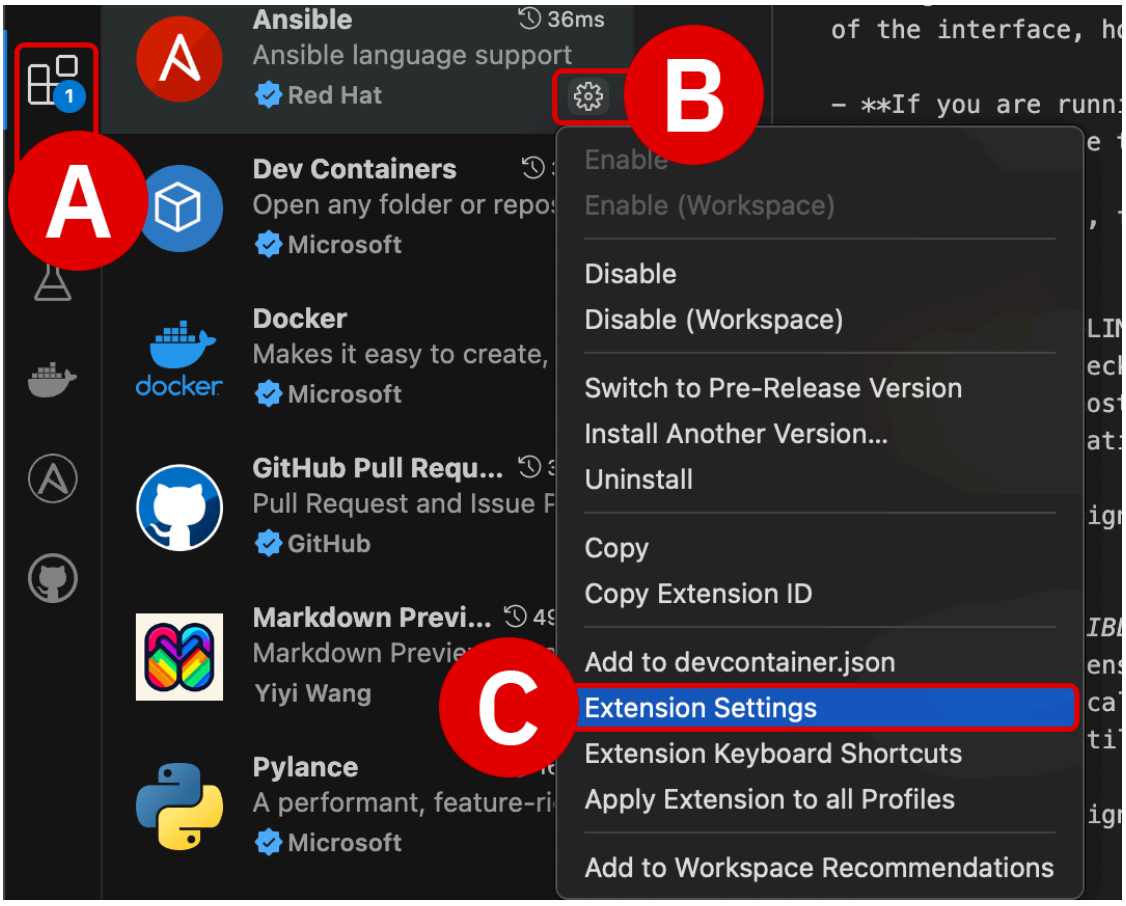
CANNOT SET PROPERTIES OF UNDEFINED (SETTING 'currentModelValue')

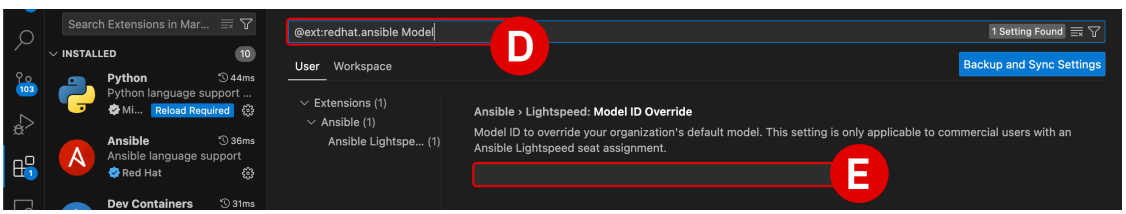
Make sure that the **Model ID Override** field is set to `empty` in your Ansible for VS Code extension settings.

To verify this:

- Click the **Extensions** tab[A] along the left-hand interface
- Click the **Manage** icon[B] on the right side of the *Ansible* extension tile, then drill down into **Extension Settings**[C]
- Add the text `Model` to the search filter[D] at the top of the Extension Settings panel
- Clear the input field[E] of any model IDs and leave it blank
- Close the Extension Settings panel by clicking `X` and return to the Ansible Playbook







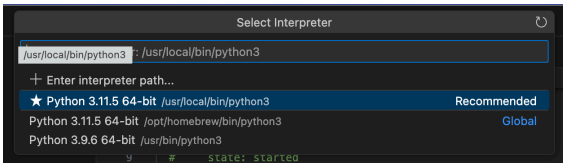
SPAWN C:\Windows\system32\cmd.exe ENOENT

This warning is not related to Ansible or WCA. You can safely **ignore** this error if it occurs during the lab exercises.

PYTHON DRIVERS ARE MISSING

The WCA extension for VS Code requires that Python drivers are included within the workspace. These are usually configured within VS Code by default, but can be easily set if necessary. Look for a `Python` tile adjacent to the `Ansible` tile along the bottom-right corner of the VS Code interface. If it is not set, **click** the tile and select the `Python 3.11.5 64-bit` drivers.

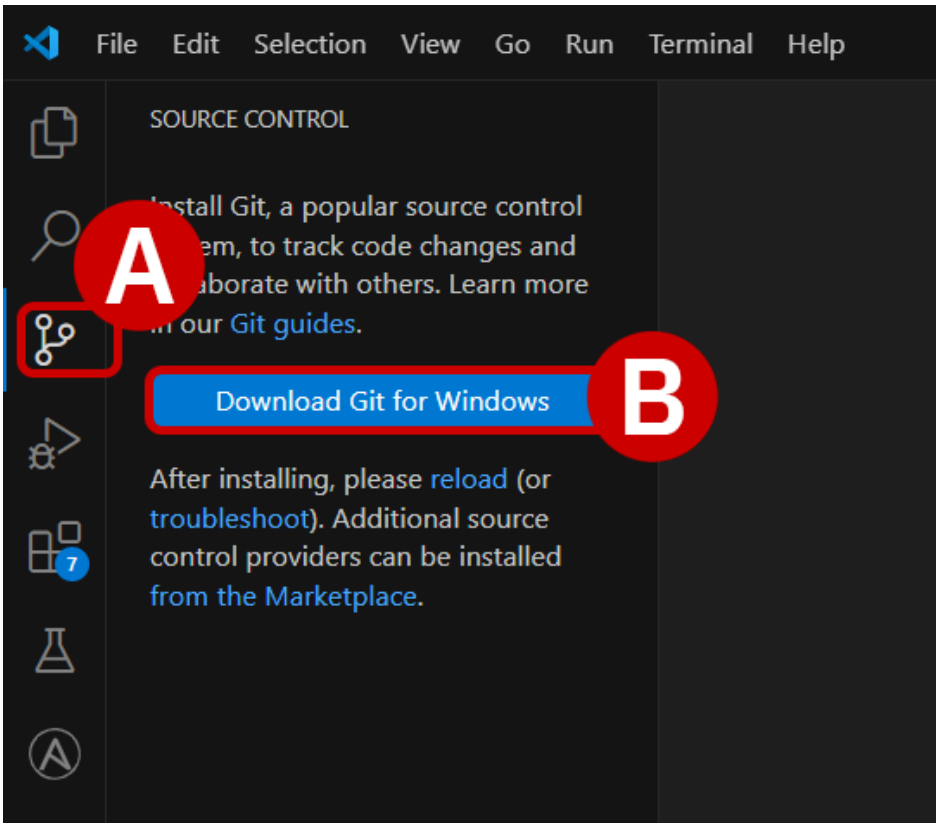
- Click the gold-colored `Select python environment` button at the bottom-right of the interface
- From the console at the top of the VS Code environment, select the recommended `Python 3.11.5 64-bit` option and hit `Enter ⏎` to confirm

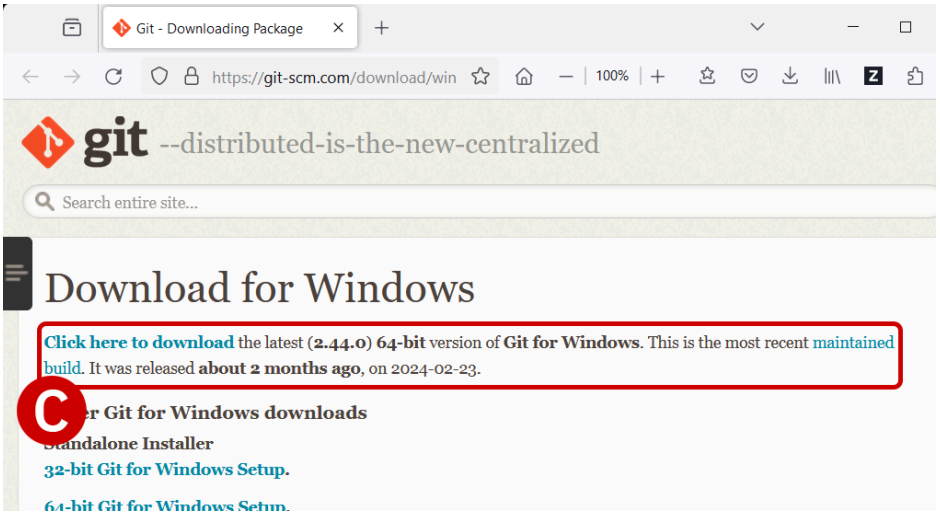


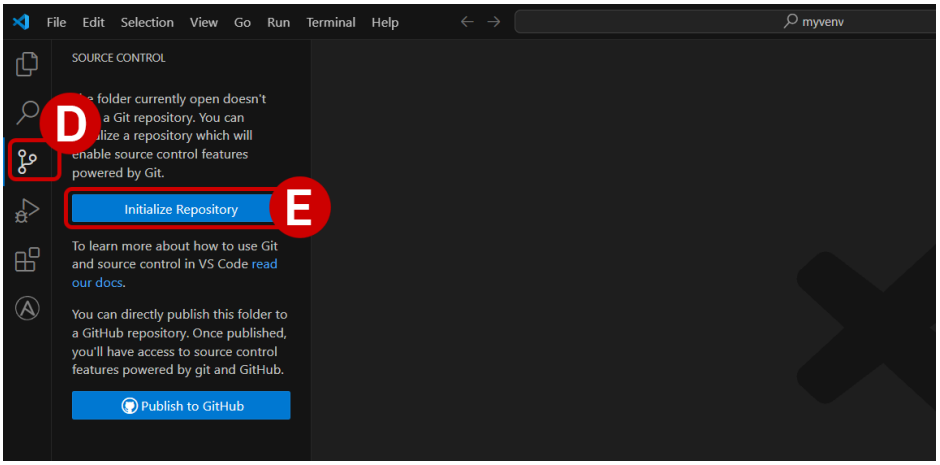
GIT NOT INSTALLED

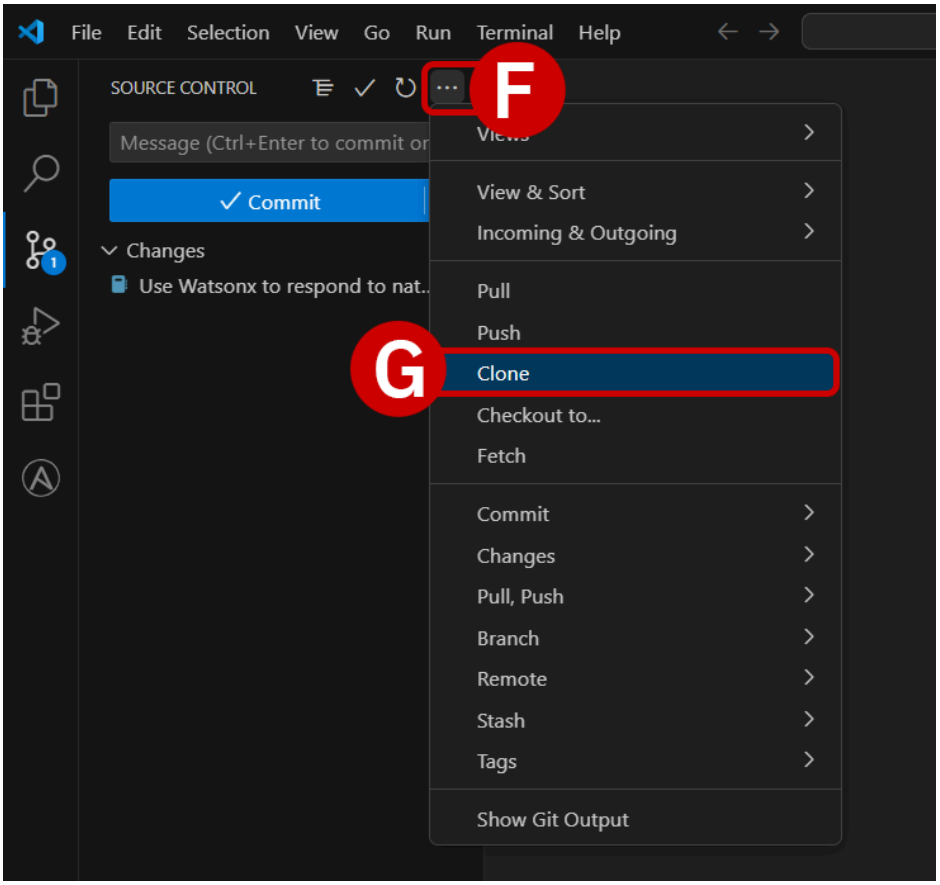
If your machine has not used **Git** previously, you may be prompted by VS Code to install it before attempting a `clone` request. This is more commonplace on Windows operating systems, but some MacOS users may need to install Git as well. The following instructions will guide you through the process:

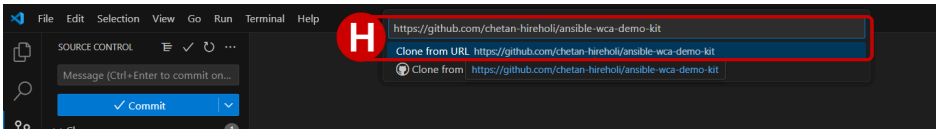
- With the VS Code application open, click the **Source Control**[A] tab from the left-hand interface and then click **Download Git**[B].
- A web browser will open to the git-scm.com Downloads page.
  - **Download**[C] the version recommended for your machine's particular operating system.
  - The recommended version will be displayed first at the top of the list.
  - Execute the installer on your machine and follow along with the prompts to finish installing Git. Accept the license agreement and accept the default values on each page.
- Return to VS Code and once again click the **Source Control**[D] tab, then click the blue **Initialize Repository**[E] button.
- From the top of the **Source Control** tab, click the three dots `...`[F] icon to expand a drop-down menu of options. Click on the **Clone**[G] option.
- Follow the instructions for the remainder of this section to clone the `ansible-wca-demo-kit` repository to your local machine[H].




Git - Downloading Package
https://git-scm.com/download/win
100%
git --distributed-is-the-new-centralized
Search entire site...
Download for Windows
Click here to download the latest (2.44.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released about 2 months ago, on 2024-02-23.
C
Standalone Installer
32-bit Git for Windows Setup.
64-bit Git for Windows Setup.


File Edit Selection View Go Run Terminal Help
myvenv
SOURCE CONTROL
D
Initialize Repository
E
To learn more about how to use Git and source control in VS Code read our docs.
You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by git and GitHub.
Publish to GitHub


File Edit Selection View Go Run Terminal Help
SOURCE CONTROL
F
Message (Ctrl+Enter to commit or
Commit
Changes
Use Watsonx to respond to nat...
View & Sort
Incoming & Outgoing
Pull
Push
G
Clone
Checkout to...
Fetch
Commit
Changes
Pull, Push
Branch
Remote
Stash
Tags
Show Git Output



COPY AND PASTE INSTRUCTIONS INTO A VIRTUAL MACHINE

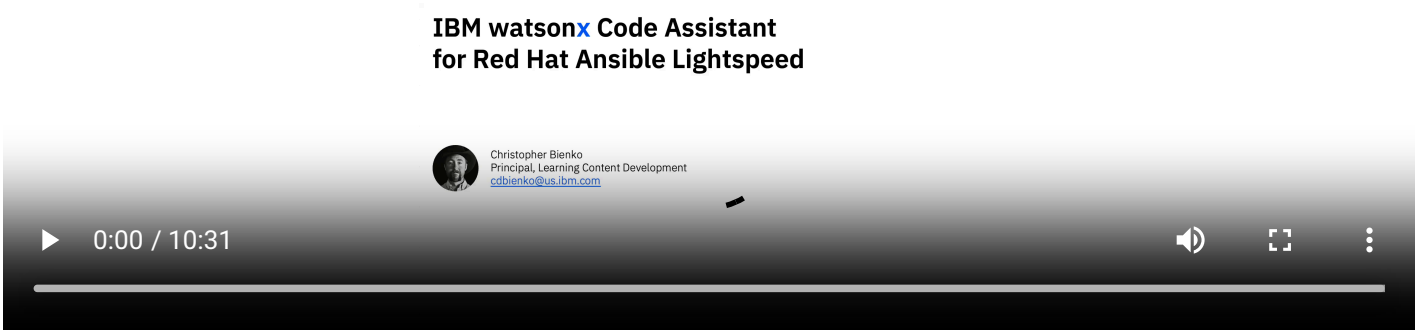
As you are running the lab environment inside a virtual machine (VM), it is not be possible to "paste" lab instructions from your local machine's clipboard directly into the VM.

If you wish to copy and paste instructions directly from the lab documentation, it is recommended that you open the GitHub instructions **inside** the VM's web browser (Firefox). This will allow you to copy instructions to the VM's clipboard and paste instructions inside the VS Code editor.

---

vi. Next steps

The following section will cover the fundamentals of AI-recommended code generation for Ansible Tasks.



> Christopher Bienko *(Principal, IBM Global Sales Enablement)* demonstrates key elements and hands-on components of the `Generating Code` module. *[10 min]*

i. Generating Code with IBM watsonx Code Assistant for Red Hat Ansible Lightspeed

An **Ansible Task** is a statement in Ansible's automation script (the YAML-based Playbooks you will be working with) that declares a single action to be executed. This might be installing a package, copying a file, or shutting down a service on a remote machine. Each *Task* represents an idempotent operation (an action that can be repeated multiple times and deliver the same result every time) that aligns the remote managed node to the specified state. Idempotent operations also ensure consistency across multiple executions, guaranteeing the same steps are taken on each execution of the task.

After you have learned the fundamentals of generating Ansible Task code blocks using *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed* (**WCA**), you'll be ready to shape and tailor the AI-generated code recommendations using WCA's model tuning capabilities.

---

deploy_monitoring.yml

local_dev > lightspeed_tp > deploy_monitoring.yml

```yaml
---
- name: Deploy monitoring
  hosts: monitoring
  become: true

  # module_defaults:
  #   ansible.posix.firewalld:
  #     permanent: true

  tasks:
    # - name: Include redhat.rhel_system_roles.cockpit

    # - name: Copy files/cockpit.conf to /etc/cockpit/

    # - name: Restart cockpit service

    # - name: Allow cockpit through firewall
```

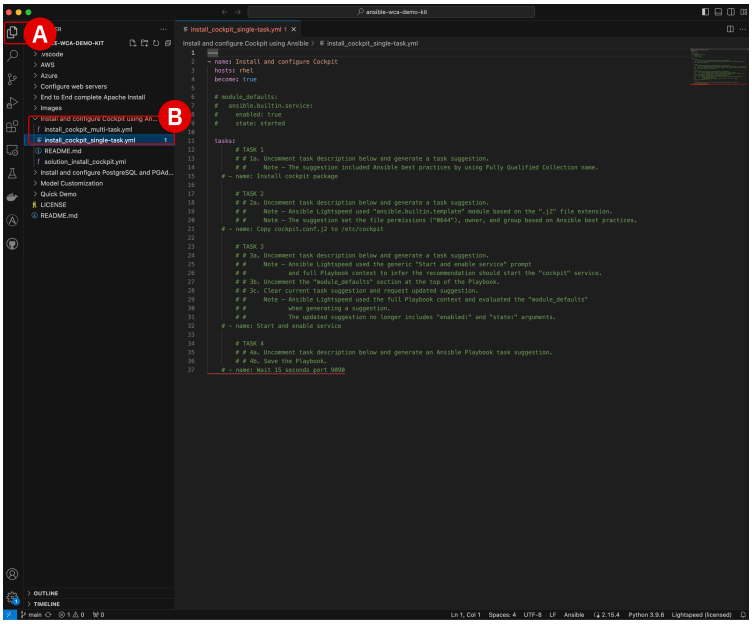
ii. Single task Ansible operations

The process of creating AI-generated **code recommendations** is as simple as modifying the natural language (plain English) Task descriptions of an action that is to be executed, which always start with `- name:` and are followed by some description of the task to be performed. Ansible Tasks are often preceded with the prefix `#`, indicating developer comments or documentation. After the natural language description of the automation Task has been set by the user, WCA handles the rest.

WCA is also capable of generating multiple Ansible Tasks from more complex natural language descriptions— what is referred to as **multi-task** code generation —which you will experiment with later in this module. However, to get started, let's begin with the basics of generating code for **single task** use cases.

1. Begin by opening the `install_cockpit_single-task.yml` Playbook from the list of assets in the Explorer browser.
   - Click the **Explorer** tab from the left-hand interface[A]
   - Drill down into the `Install and configure Cockpit using Ansible` subdirectory[B]
   - Double-click the `install_cockpit_single-task.yml` Playbook
   - A replica of the Playbook code is also included below in the documentation

> The red highlighting within the editor reminds users that the `tasks:` section contains no valid `-name:` task definitions. This is part of WCA's code validation process which runs automatically and alerts users to syntax errors in their code. You can safely ignore these warnings for now, as you will be un-commenting and generating valid `-name:` task definitions in the following steps.



~/Documents/ansible-wca-demo-kit/install and configure Cockpit using Ansible/install_cockpit_single-task.yml

```
---
- name: Install and configure Cockpit
  hosts: rhel
  become: true

  # module_defaults:
  #   ansible.builtin.service:
  #     enabled: true
  #     state: started

  tasks:
        # TASK 1
        # # 1a. Uncomment task description below and generate a task suggestion.
        # #     Note - The suggestion included Ansible best practices by using Fully Qualified
Collection name.
    # - name: Install cockpit package

        # TASK 2
        # # 2a. Uncomment task description below and generate a task suggestion.
        # #     Note - Ansible Lightspeed used "ansible.builtin.template" module based on the ".j2"
file extension.
        # #     Note - The suggestion set the file permissions ("0644"), owner, and group based on
Ansible best practices.
    # - name: Copy cockpit.conf.j2 to /etc/cockpit

        # TASK 3
        # # 3a. Uncomment task description below and generate a task suggestion.
        # #     Note - Ansible Lightspeed used the generic "Start and enable service" prompt
        # #            and full Playbook context to infer the recommendation should start the
"cockpit" service.
        # # 3b. Uncomment the "module_defaults" section at the top of the Playbook.
        # # 3c. Clear current task suggestion and request updated suggestion.
        # #     Note - Ansible Lightspeed used the full Playbook context and evaluated the
"module_defaults"
        # #            when generating a suggestion.
        # #            The updated suggestion no longer includes "enabled:" and "state:" arguments.
    # - name: Start and enable service

        # TASK 4
        # # 4a. Uncomment task description below and generate an Ansible Playbook task suggestion.
        # # 4b. Save the Playbook.
    # - name: Wait 15 seconds port 9090
```

The `install_cockpit_single-task.yml` Playbook code above warrants some explanation before we move on with making AI-generated modifications to it:

- **Line 2** essentially marks the beginning of the Playbook instructions, the purpose of which is to automate the process of installing and configuring **Cockpit** for Red Hat Ansible.
- **Lines 3-4** define variables that will remain static throughout the remainder of the Playbook. These variables will be referenced by the AI-generated code suggestions at a later stage. This is a key capability of the offering and one which you will explore in much finer details later on in this module.
- **Lines 6-9** are variables which have been *commented out* and therefore are invisible to the execution of the Ansible script **and** not examined by WCA for context when generating code recommendations. You will experiment with how removing the `#` comment blocks impacts the recommendations of task block code. "Uncommenting" these lines of code will make them viable for execution and these lines will afterwards be considered as valid Playbook "context" for AI code generation.

2. Locate `TASK 1` on **Line 15** of the YAML file, which handles installation of **Cockpit** for Ansible. Cockpit is an interactive server administration interface that provides a graphical overview of statistics and configurations for a system or systems within a network.

```
    # - name: Install cockpit package
```

Pay attention to the indentation and characters used on **Line 15**, which in sequence from left to right are as follows:

- begins with `Tab ⇥` (or `Space` whitespaces) for indentation
- a `#` character to "comment out" the line's contents
- a whitespace `Space` character
- `- name:` which signifies the start of a Task definition
- and finally the natural language description of the Task

> **INDENTATION LEVELS AND WHITESPACE**
>
> Similar to Python, Ansible and YAML-based Playbooks are very sensitive to whitespacing and indentation. Indentations (such as the `Tab ⇥` in this example) denote different hierarchies and code nesting levels within the YAML structure.
>
> You may use `Space` instead of `Tab ⇥` if you prefer, but be sure to **use indentations consistently**: choose to use either `Tab ⇥` or `Space` for indenting lines of code, and do not interchange between the two.

3. To generate code for `TASK 1`, first **uncomment** the line of code (remove the `#` character from the start of a line).
   - Highlight the line(s) of code you wish to uncomment and then press `⌘ Cmd` + `?` for macOS or `^ Ctrl` + `?` for Windows
   - You can repeat those keystrokes with the line(s) selected to toggle between commenting or uncommenting lines of code
   - *Tip:* commented out lines of code in VS Code will appear as green text

   Afterwards, **Line 15** should look like the following — beginning with a single `Tab ⇥`

```
    - name: Install cockpit package
```

4. Now you are ready to begin generating code. Place your cursor on **Line 15** and hit `Enter ⏎`
   - Wait for WCA to engage and generate the suggested (in *grey, italicized text*) code block for executing the task
   - This temporary code suggestion is entirely generated by AI
   - As a user, you have the option to either:
     a. **Accept** the code recommendation as-given by pressing `Tab ⇥`

b. **Modify** the recommended code by highlighting and replacing the italicized text

> **FAILED TO CONNECT TO THE SERVER /** `"YOU DON'T HAVE ACCESS TO IBM WATSONX..."`

This warning will occur when the Ansible plugin for VS Code needs to be re-authenticated with WCA. It can occur after an extended period of inactivity or a system restart. For example, if your lab environment is running inside a VM, pausing or restarting the VM may produce this error.

To re-authenticate:

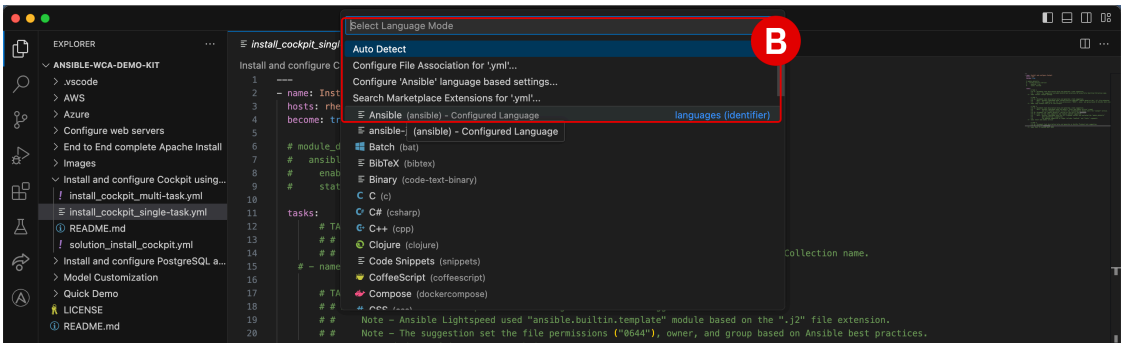
- **Sign out** from the *VS Code* application by clicking the User icon[A] in the bottom-left corner of the interface, hover over your username, and then click **Sign Out**[B]
- **If you are running this environment inside a virtual machine (VM)**, closing and restarting the VM *will not* resolve the issue — you must sign out from the *VS Code* application, not the VM
- Once logged out, follow from *Step 7* of the *Setup & Troubleshooting* to re-authenticate with WCA

CODE RECOMMENDATIONS ARE NOT GENERATING

Ansible Lightspeed and WCA will only generate code recommendations for *Ansible Playbooks* and *YAML* files. VS Code will typically auto-detect the programming language of the document you're working with, but on occassion you may need to manually specify the language. Even if working with a YAML file, you'll still need to specify the language mode as `Ansible` for the Lightspeed plugin to engage.

To set the language mode correctly:

- In the bottom-right corner of the VS Code interface, hover over the **Select Language Mode** toggle[A]
- A console will appear at the top of VS Code with a drop-down list of options[B]
- Click `Ansible` from the suggested languages, or enter the text yourself and hit `Enter ⏎`
- Confirm that the Select Language Mode toggle in the bottom-right corner displays `Ansible`

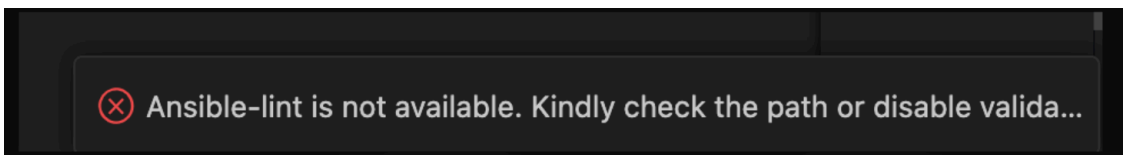


"ANSIBLE-LINT IS NOT AVAILABLE."

`ansible-lint` checks Playbooks for practices and behavior that could potentially be improved and can fix some of the most common ones for you. It will constantly check your Ansible syntax as you type and provide recommendations for how to improve it.

- You can safely **ignore** this error if it occurs during the lab exercises
- If you wish to install `ansible-lint` on your local machine, execute the following instruction within a Terminal console:

```
python3 -m pip install --upgrade --user ansible-lint
```



5. Hit `Tab ⇥` to accept the suggested code and then compare with the `SOLUTION` tab below.

TEMPLATE

```
# TASK 1
- name: Install cockpit package
```

SOLUTION

```
# TASK 1
- name: Install cockpit package
ansible.builtin.package:
    name: cockpit
    state: present
```

As part of the plain-text description of the Task, WCA was asked to include the `cockpit` Role, part of the Red Hat Enterprise Linux System Roles Certified Content Collection. The AI-generated code suggestion invoked a Fully Qualified Collection Name (**FQCN**) - `ansible.builtin.package`

Making use of FQCNs where possible is a recommended best practice and is a prime example of the many ways in which the offering infuses post-processing capabilities within the AI-generated code produced by WCA.

---

6. Additional examples of infusing best-practices into AI-generated code recommendations can be found in `TASK 2` (**Line 21** of the unmodified template or **Line 25** after *Step 5*):

- Uncomment `- name: Copy cockpit.conf.j2 to /etc/cockpit`
- Hit `Enter ⏎` to generate the task code recommendation and accept the AI-suggested code (without modifications) by pressing `Tab ⇥`
- Compare your results with the `SOLUTION` tab below

TEMPLATE

```
# TASK 2
- name: Copy cockpit.conf.j2 to /etc/cockpit
```

SOLUTION

```
# TASK 2
- name: Copy cockpit.conf.j2 to /etc/cockpit
ansible.builtin.template:
    src: cockpit.conf.j2
    dest: /etc/cockpit/cockpit.conf
    owner: root
    group: root
    mode: '0644'
```

The AI-generated code recommendation will copy `cockpit.conf` to the target host.

Take note of the fact that the recommendation included the `mode:` argument and set the Linux file permissions to `0644`, neither of which were things explicitly requested in the Task `-name` description, but are both additions which adhere to best practices around defining Ansible automaton tasks. Setting a file permission to `0644` specifies read and write permissions for User and Group levels within the Linux OS, and provides only read permissions to all others.

iii. Multi-task Ansible operations

Up to this point, we've kept a narrow aperture on AI-generated recommendations for single tasks — examining and experimenting with generating Ansible code task by task, one at a time. However, a powerful WCA feature is the ability to combine multiple task descriptions into a single natural language prompt; in turn, WCA is able to parse that instruction, decompose the instruction into discrete Ansible Task parts, and return a complete code recommendation for achieving the author's intended goal.

Syntactically, multiple tasks are combined into a single natural language expression through the use of **ampersand** (`&`) characters. Simply write out all the automation task descriptions on a single line, separating each description with a `&` character. The line must also *begin* with a `#` character for reasons that will be explained shortly.

---

7. To illustrate, let's look at a multi-task Ansible Playbook: `install_cockpit_multi-task.yml`

   - The contents of this Playbook should look familiar to you already: it is essentially the same Playbook examined in *Steps 1-6* (`install_cockpit_single-task.yml`), re-written in an equivalent multi-task expression
   - Each of the Task descriptions from the previous Playbook have been consolidated into a single description on **Line 12**, separated by `&` characters

   ~/Documents/ansible-wca-demo-kit/install and configure Cockpit using Ansible/install_cockpit_multi-task.yml

```
1  ---
2  - name: Install and configure Cockpit
3  hosts: rhel
4  become: true
5
6  module_defaults:
7      ansible.builtin.service:
8      enabled: true
9      state: started
10
11 tasks:
12     # Install cockpit package & Copy cockpit.conf.j2 to /etc/cockpit & Start and enable service
   & Wait 15 seconds port 9090
```

---

8. There are two crucial distinctions between *single task* and *multi-task* code generation: formatting and execution.

   - *Formatting*: Notice that **Line 12** does not begin with `-name:` , as was the case with single task descriptions
   - *Execution*: In order to generate AI code recommendations for multi-task descriptions, **Line 12** must *stay commented out* (the `#` *must remain* at the start of the line)

   What is the rationale behind this? When WCA's generative AI capabilities parse **Line 12**, its output will include multiple `-name:` tasks, each containing potentially multiple lines of instructions, based on how many `&` -delineated task descriptions are included on the line.

Therefore, the way in which the code generation step is executed on **Line 12** is a consequence of the formatting decision. Execution of a code generation step on a commented-out (`#`) line containing `&` delineators is recognized by WCA as a unique case that will be acted upon as a multi-task statement.

---

9. Place your cursor at the end of **Line 12**, and *without* removing the `#` character, press `Enter ⏎` to execute the code generation step. Be aware that generating code for multi-task descriptions will take longer compared to a single task.

   Compare the `MULTI-TASK` solution tab with the `SINGLE TASK` solution (copied over from *Step 6*). How did the multi-task code generation fare compared to the single task approach?

MULTI-TASK

```yaml
---
- name: Install and configure Cockpit
hosts: rhel
become: true

module_defaults:
    ansible.builtin.service:
    enabled: true
    state: started

tasks:
    - name: Install cockpit package
    ansible.builtin.package:
        name: cockpit
        state: present

    - name: Copy cockpit.conf.j2 to /etc/cockpit
    ansible.builtin.template:
        src: cockpit.conf.j2
        dest: /etc/cockpit/cockpit.conf
        owner: root
        group: root
        mode: '0644'

    - name: Start and enable service
    ansible.builtin.service:
        name: cockpit.socket

    - name: Wait 15 seconds port 9090
    ansible.builtin.wait_for:
        port: 9090
        delay: 15
```

SINGLE TASK

```yaml
---
- name: Install and configure Cockpit
hosts: rhel
become: true

module_defaults:
    ansible.builtin.service:
    enabled: true
    state: started

tasks:
    - name: Install cockpit package
    ansible.builtin.package:
        name: cockpit
        state: present

    - name: Copy cockpit.conf.j2 to /etc/cockpit
    ansible.builtin.template:
        src: cockpit.conf.j2
        dest: /etc/cockpit/cockpit.conf
        owner: root
        group: root
        mode: '0644'

    - name: Start and enable service
    ansible.builtin.service:
        name: cockpit
        state: started
        enabled: true

    - name: Wait 15 seconds port 9090
    ansible.builtin.wait_for:
        port: 9090
        delay: 15
```

10. Comparing the two results, the only notable difference between the two approaches are **Lines 27-29** from the `SINGLE TASK` generative AI approach.

    Both the `SINGLE TASK` and `MULTI-TASK` suggestions for that particular task *satisfy* the request made by the user. However, whether the single task or multi-task approach resulted in a better code suggestion is up to the judgement of the programmer. Nearly 90% of the remaining code was identical between the two approaches and was achieved in far fewer lines of code (and less typing) using the multi-task approach.

    > The variability of generative AI suggestions is a fascinating topic and one that we will dive more deeply into with the module ahead.

11. Before moving on to other product features, experiment by creating a new Ansible Playbook in your workspace using the code template below. Suggestions will be given on how to perform the same automation task using single and multi-task generation approaches.

    - **Save** the YAML file as `create_ec2_single_multi.yml` (if you forget to save the file, WCA will not generate recommendations)

- Copy the following code block to your clipboard using the `+` icon in the top-right corner of the panel and paste into the newly created YAML file

> **HOW TO CREATE NEW `YAML` PLAYBOOKS**

Note: You need to copy and paste the contents of the Playbook into a **New File...** within the *same* Lightspeed project directory that was used for the previous lab modules in order for the VS Code extension to engage.

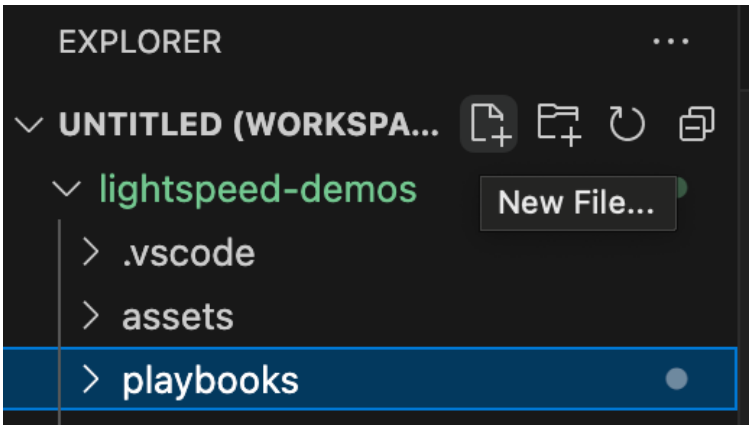
To create a new YAML Playbook within a VS Code environment:

a. Copy the contents of the Playbook to clipboard using the button in the top-right corner of the lab guide code block.

ıd launch the service

```
ginx webserver and launch the service
akcshetty@in.ibm.com)
```

b. Return to your VS Code environment. In the top-left corner of the interface, with your Ansible Lightspeed folder selected, click the **New File...** button.

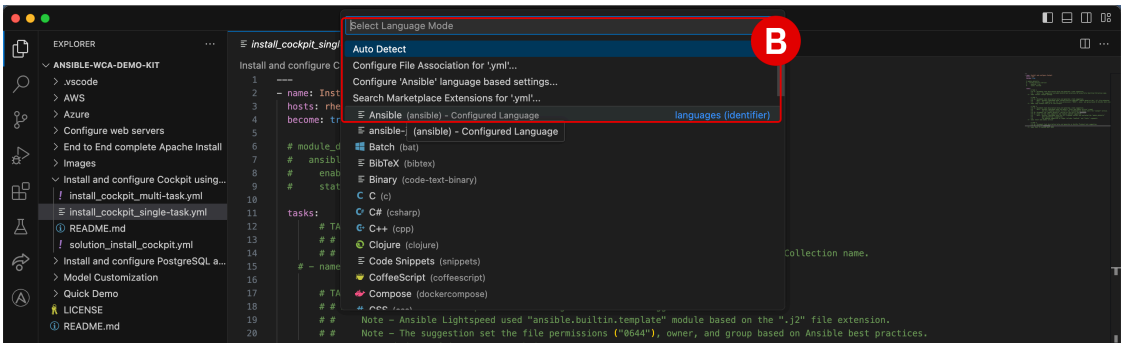


c. Name the file to a description of your choosing, ending with `.yml` as the filetype. Set it to `CustomPlaybook.yml`, for example. Save it to one of the directories in the `ansible-wca-demo-kit` folder.

d. Paste the clipboard contents into the YAML file and follow along with the suggestions below.

> **COPY AND PASTE CODE WITHIN THE VM**

Information "copied" to your local machine's clipboard cannot be "pasted" directly into the virtual machine (VM) environment or VS Code. If you wish to copy and paste instructions directly from the lab documentation, it is recommended that you open the GitHub instructions **inside** the VM's web browser (Firefox). This will allow you to copy instructions to the VM's clipboard and paste instructions inside the VS Code editor.

> **ANSIBLE LIGHTSPEED IS MISSING OR CODE RECOMMENDATIONS ARE NOT GENERATING**
>
> Ansible Lightspeed and WCA will only generate code recommendations for *Ansible Playbooks* and *YAML* files. VS Code will typically auto-detect the programming language of the document you're working with, but on occassion you may need to manually specify the language. Even if working with a YAML file, you'll still need to specify the language mode as `Ansible` for the Lightspeed plugin to engage.
>
> To set the language mode correctly:
>
> - In the bottom-right corner of the VS Code interface, hover over the **Select Language Mode** toggle[A]
> - A console will appear at the top of VS Code with a drop-down list of options[B]
> - Click `Ansible` from the suggested languages, or enter the text yourself and hit `Enter ⏎`
> - Confirm that the Select Language Mode toggle in the bottom-right corner displays `Ansible`
>
> 
>
>
> 
>

~/Documents/ansible-wca-demo-kit/create_ec2_single_multi.yml

```
---
- name: Provision an EC2 instance
  hosts: all
  tasks:
```

12. First, configure the Playbook for single task generation by adding the following code snippet into **Line 5** (remembering to properly indent with `Tab ⇥` or `Space` characters):

```
    - name: create vpc named demo
```

13. Your workspace Playbook should look identical to the `TEMPLATE` tab below. Place your cursor at the end of the newly-created **Line 5** and hit `Enter ⏎` to execute single task code generation. Compare your results to the `SOLUTION` tab. Record your results to a notepad so that you can compare the results later.

    TEMPLATE

```yaml
---
- name: Provision an EC2 instance
  hosts: all
  tasks:
        - name: create vpc named demo
```

    SOLUTION

```yaml
---
- name: Provision an EC2 instance
  hosts: all
  tasks:
        - name: create vpc named demo
          amazon.aws.ec2_vpc_net:
            name: demo
            cidr_block: 10.0.0.80/16
            tags:
              Name: demo
            tenancy: default
          register: vpc
```

---

14. Now it's time to re-write the Playbook for multi-task code generation. The beauty of natural language statements is that your approach can be as terse or verbose as you want. The more verbose and descriptive, the more *prescriptive* you can be in terms of influencing the AI-generated code recommendations from WCA.

    The topic of *"prompt tuning"* will be explored in much greater detail in the Task Description Tuning and Model Customization module. But for now:

    - **Replace** all of the code inside `create_ec2_single_multi.yml` with the `TEMPLATE` code block (below)
    - Place your cursor at the end of **Line 5** (remember not to remove the `#` character) and hit `Enter ⏎`, then accept the recommendation with `Tab ⇥`
    - Compare your results to the `SOLUTION` tab and to the result from *Step 13*

TEMPLATE

```
---
- name: Provision an EC2 instance
  hosts: all
  tasks:
        # create vpc named demo & create security group for demo vpc and allow ssh and http & create internet gateway named demogateway for demo vpc & create subnet named demosubnet in demo vpc & create route table for subnet demosubnet and gateway demogateway & create t2.small ec2 instance in subnet demosubnet
```

SOLUTION

```yaml
---
- name: Provision an EC2 instance
  hosts: all
  tasks:
        # create vpc named demo & create security group for demo vpc and allow ssh and http &
create internet gateway named demogateway for demo vpc & create subnet named demosubnet in demo
vpc & create route table for subnet demosubnet and gateway demogateway & create t2.small ec2
instance in subnet demosubnet
        - name: Create vpc named demo
          amazon.aws.ec2_vpc_net:
            name: demo
            cidr_block: 10.0.0.80/16
            tags:
              Name: demo
            tenancy: default
          register: vpc

        - name: Create security group for demo vpc and allow ssh and http
          amazon.aws.ec2_security_group:
            name: demo
            description: demo security group
            vpc_id: "{ vpc.vpc.id }"
            rules:
              - proto: tcp
                from_port: 22
                to_port: 22
                cidr_ip: 0.0.0.0/0
              - proto: tcp
                from_port: 80
                to_port: 80
                cidr_ip: 0.0.0.0/0
          register: sg

        - name: Create internet gateway named demogateway for demo vpc
          amazon.aws.ec2_vpc_igw:
            vpc_id: "{ vpc.vpc.id }"
            tags:
              Name: demogateway
          register: igw

        - name: Create subnet named demosubnet in demo vpc
          amazon.aws.ec2_vpc_subnet:
            cidr: 10.0.0.80/24
            vpc_id: "{ vpc.vpc.id }"
            tags:
              Name: demosubnet
          register: subnet

        - name: Create route table for subnet demosubnet and gateway demogateway
          amazon.aws.ec2_vpc_route_table:
            vpc_id: "{ vpc.vpc.id }"
            tags:
              Name: demosubnet
            subnets:
              - "{ subnet.subnet.id }"
            routes:
              - dest: 0.0.0.0/0
                gateway_id: "{ igw.gateway_id }"
          register: route_table

        - name: Create t2.small ec2 instance in subnet demosubnet
          amazon.aws.ec2_instance:
            key_name: "{ _key_name_ }"
```

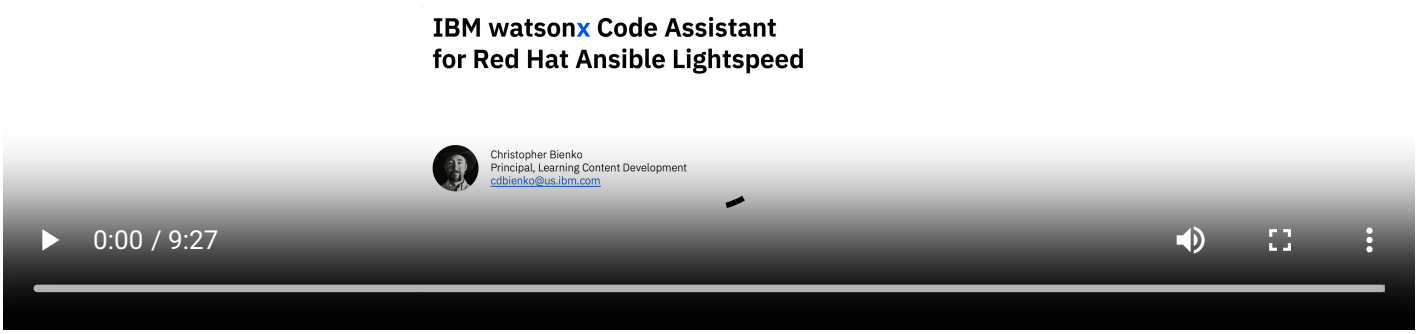
```
            instance_type: t2.small
            image: "{ _image_ }"
            wait: true
            vpc_subnet_id: "{ subnet.subnet.id }"
            security_group: demo
          register: ec2
```

> ⚠ **`TIMEOUT` WARNING**
>
> It may take several moments for WCA to process and return code recommendations for a multi-task description as complex as this one. If you receive a time-out warning, try executing the code generation step by pressing `Enter ⏎` a second time.

---

iv. Next steps

In the next section, you will examine in detail WCA's post-processing and content source attribution capabilities.



> Christopher Bienko *(Principal, IBM Global Sales Enablement)* demonstrates key elements of the `Content Source Matching and Post-Processing` module. *[10 min]*

i. Why code attribution and post-processing matters for generative AI

At this point, you should now be well acquainted with *Ansible Tasks* and *Playbooks*. Ansible Playbooks are comprised of multiple Tasks — which you have been using *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed's* (**WCA**) generative AI capabilities to define and expand upon.

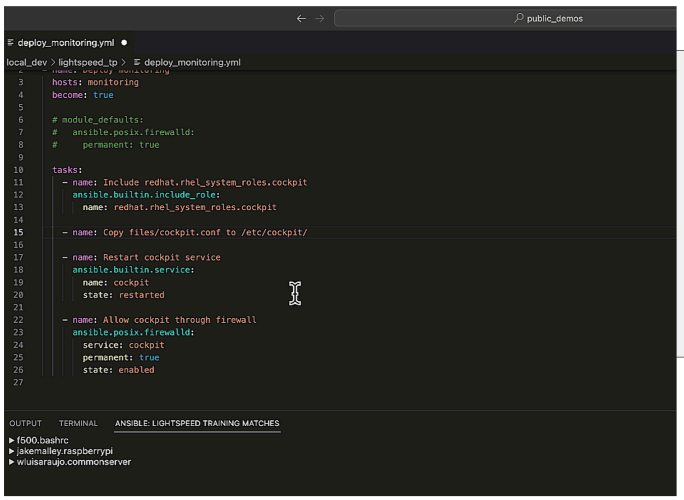
If a Playbook contains multiple "tasks" that together achieve some singular goal or purpose, you might want to group these tasks into a single reusable unit. For example, a string of "tasks" might provision infrastructure for separate development, test, and production clusters from a major cloud vendor. Patterns for automating endpoints from these vendors can be codified into **Ansible Modules**, which are supported by Red Hat and the Ansible community writ large.

And finally, collections of Playbooks and Modules can be organized into "blueprints" for automation tasks that are frequently used together to achieve some goal; a goal which you might want to make repeatable or shareable with other teams within your business. These blueprints are referred to as **Ansible Roles**. They provide a structured way to organize tasks, templates, files, and variables, and are available to drop directly into Ansible Playbooks. Roles make it possible to easily manage and set up complex automation tasks, essentially providing a rubric to streamline automation projects.

> Collectively, these Playbooks, Modules, and Roles form a **comprehensive ecosystem of business and community-driven support** for patterning automation to work with the broadest range of vendors, technologies, and clouds.

Ansible Galaxy is a Red Hat-curated, community-driven repository for *Ansible Roles*. Via communities such as Galaxy, thousands of Roles are available for Ansible users to leverage within their own Playbooks. Ansible Galaxy is also key to how generative AI code recommendations from WCA are attributed back to original content sources and authors.

IBM watsonx Code Assistant for Red Hat Ansible Lightspeed



KEY CAPABILITY: Content Source Matching

↗ Ansible Lightspeed ensures **transparency** and **trust** in Generative AI by consistently providing clear references to the original sources of content.

↗ Each generated suggestion will include a potential training data source, attribution if available to the author of the data source, and its usage license (if applicable).

Maintain high levels of accuracy and transparency of AI models through data source attribution.

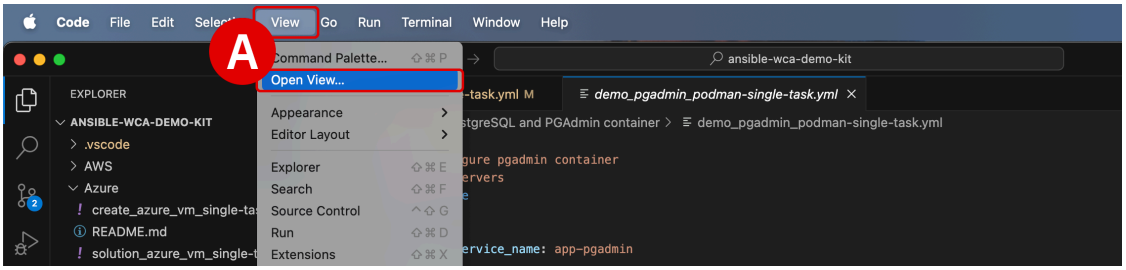
Tune **IBM watsonx** Foundation Models with your own data set to apply organizational best practices.

ii. Content source matching

A powerful capability within WCA is **Content Source Matching** (often referred to as "code explainability"), which attempts to match AI-generated code suggestions to the training data and sources that were utilized in generating the suggested Task code.

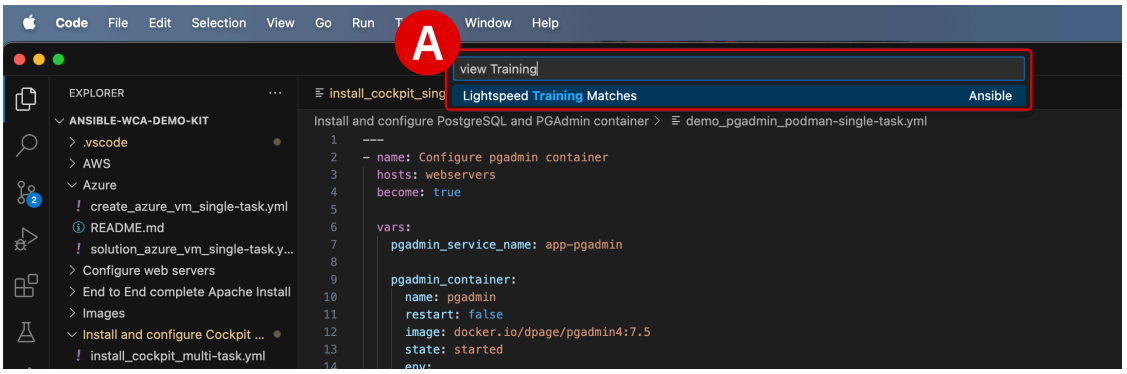
These code attribution suggestions are created using a **k-NN** (K-Nearest Neighbors) algorithm that examines **Ansible Galaxy** and training data repositories in search of the nearest related content to the AI-generated code suggestions.

1. To enable **Content Source Matching** capabilities within WCA, navigate to the main menu bar for VS Code and drill down into **View** > **Open View...** [A]



2. The console[A] along the top of the VS Code interface is now activated and awaiting a prompt. Type `Lightspeed Training Matches` and hit `Enter ⏎` to confirm the selection.

- After making this configuration change to VS Code, all requests for WCA to generate code recommendations will now open a panel at the bottom of the VS Code interface
- It is under the `Ansible` tab that **content source matching** results will be displayed
- Other tabs will also displayed for variety of options: `Output`, `Debug Console`, `Terminal`, `Ports`, and `Comments`



3. Open the `install_pgsql-single-task.yml` Ansible Playbook, which is included within the `ansible-wca-demo-kit` directory. The full location of the Playbook, as well as the contents, are available in the code block below.

~/Documents/ansible-wca-demo-kit/install and configure PostgreSQL and PGAdmin container/install_pgsql-single-task.y >

```
---
- name: Configure Database servers
  hosts: databases
  become: true

  tasks:
      # TASK 1
      # - name: Install postgresql-server

      # TASK 2
      # Ansible Lightspeed used an easy-to-understand natural language prompt and suggested the
correct, more complex PostgreSQL CLI command to initiate the database.
      # Ansible Lightspeed used best practices and kept the task idempotent by including
creates: /var/lib/pgsql/data/postgresql.conf in the suggestion.
      # - name: Run postgresql setup command

      # TASK 3
      # Ansible Lightspeed used natural language prompt and added state: started and enabled:
true module arguments based on Start and enable... in the Ansible task description.
      # - name: Start and enable postgresql service
```

4. Generate a code recommendation for the task on **Line 8** by placing your cursor at the end of the line and hitting `Enter ⏎`.

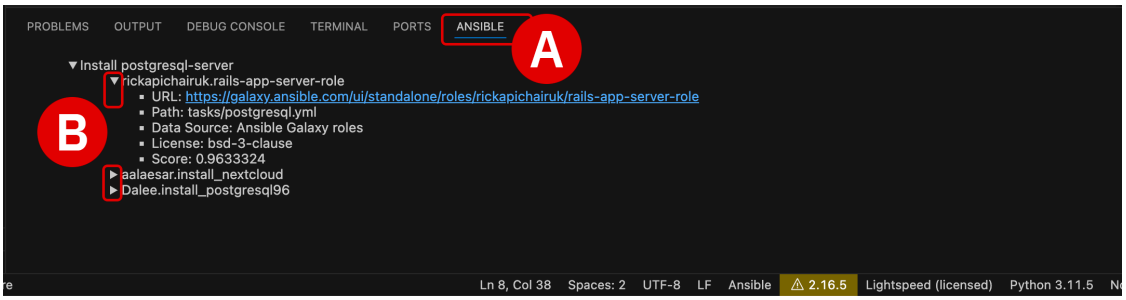
- Pay attention to the code attribution details associated with this recommendation, which will be appearing under the `Ansible` tab at the bottom of the VS Code interface once the code recommendation is accepted with `Tab ⇥`

- You need to **accept** the AI-generated code suggestions (using the `Tab ⇥` key) *before* the `Ansible` content source matching tab will provide details about the code's origins

```
---
- name: Install postgresql-server
  ansible.builtin.package:
    name: postgresql-server
    state: present
```

---

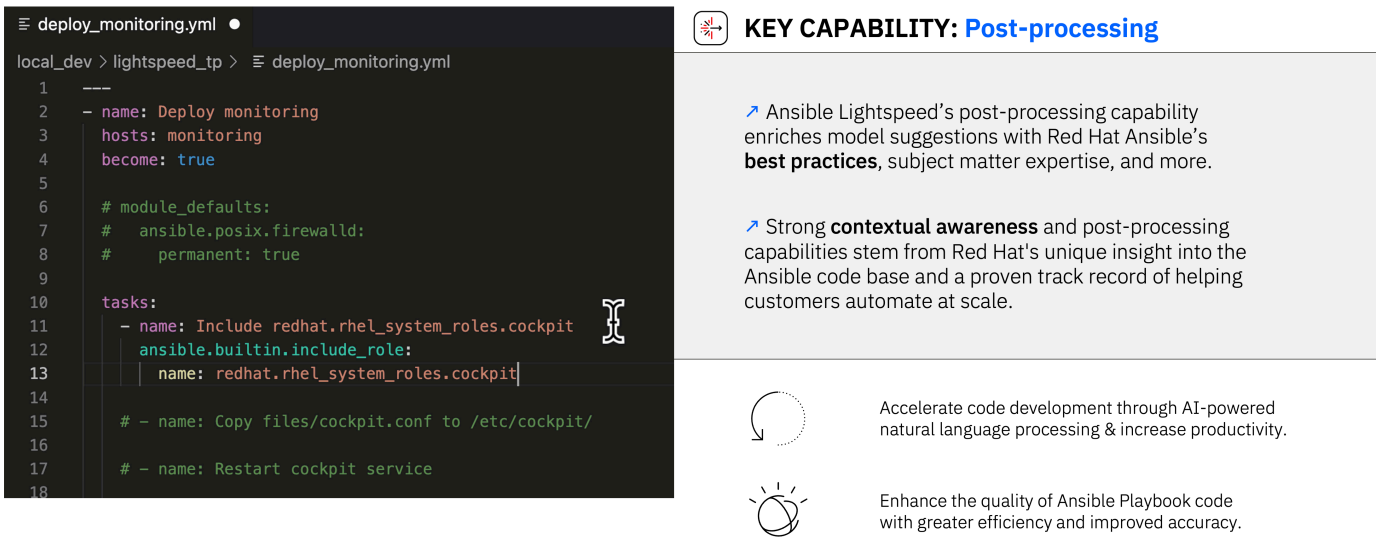
5. The three most likely content sources used in training the WCA model— which produced the AI-generated code recommendations —are listed within the `Ansible` **tab**[A]. Recall from earlier that these code attribution suggestions are created using a **k-NN** algorithm that searches **Ansible Galaxy** repositories for the nearest related content to the AI-generated code suggestions.



Clicking the arrow icon to the left of each **attribution**[B] will expand further details about the source. Information about the `URL`, `Path`, `Data Source`, `License`, and `Score` are displayed (where available) under each listing. Red Hat-certified and maintained collections, as well as contributors to open source projects on Ansible Galaxy, are the primary sources for Ansible Lightspeed model training and are the content sources you are most likely to see matched to AI-generated code recommendations.

---

6. Clicking on the `URL` field will redirect your web browser back to the precise collections and sources on **Ansible Galaxy** from which the code recommendations were derived. Here you can learn much richer details about the status of the project, any associated open source repositories involved (such as GitHub), contributions and activities ongoing with the code base, the author(s) involved, and many more intricacies.



iii. Post-processing capabilities

Another element of code generation that WCA excels at is understanding **context** within the Playbook it is executing against. If a variable or attribute is defined earlier within that Playbook, it will be recalled and referenced— where it makes sense to do so —in the generation of subsequent lines of code. You may have already noticed these **post-processing capabilities** in your experimentations with WCA-generated code suggestions. Post-processing of Task descriptions and YAML file contents helps generate contextually aware, accurate Ansible content suggestions.

However, one way to make this feature quite obvious is to take a previously-generated block of Ansible Task code, update the value assigned to a named variable *earlier* in the Playbook, and then regenerate the Task code block. In theory, the newly-generated Task block will use the updated variable name (and differ from how the code block was originally generated).

7. Using the Explorer tab, create a **New File...** YAML Playbook named `demo_provision_ec2_instance.yml` anywhere within the `/ansible-wca-demo-kit/` directory.
    - You may save the Playbook in the root of the folder or any of the subdirectories
    - Be sure to reference the expandable tooltips below for instructions on how to perform these operations and how to configure the Playbook language for use with WCA
    - **Copy and paste** the following code block into the `demo_provision_ec2_instance.yml` Playbook and then save the file

HOW TO CREATE NEW `YAML` PLAYBOOKS

Note: You need to copy and paste the contents of the Playbook into a **New File...** within the *same* Lightspeed project directory that was used for the previous lab modules in order for the VS Code extension to engage.

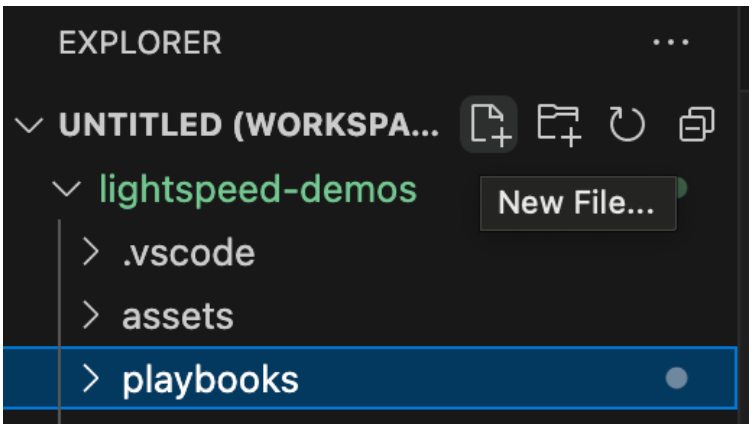
To create a new YAML Playbook within a VS Code environment:

a. Copy the contents of the Playbook to clipboard using the button in the top-right corner of the lab guide code block.

ıd launch the service

```
ginx webserver and launch the service
akcshetty@in.ibm.com)
```

b. Return to your VS Code environment. In the top-left corner of the interface, with your Ansible Lightspeed folder selected, click the **New File...** button.



c. Name the file to a description of your choosing, ending with `.yml` as the filetype. Set it to `CustomPlaybook.yml`, for example. Save it to one of the directories in the `ansible-wca-demo-kit` folder.

d. Paste the clipboard contents into the YAML file and follow along with the suggestions below.

COPY AND PASTE CODE WITHIN THE VM

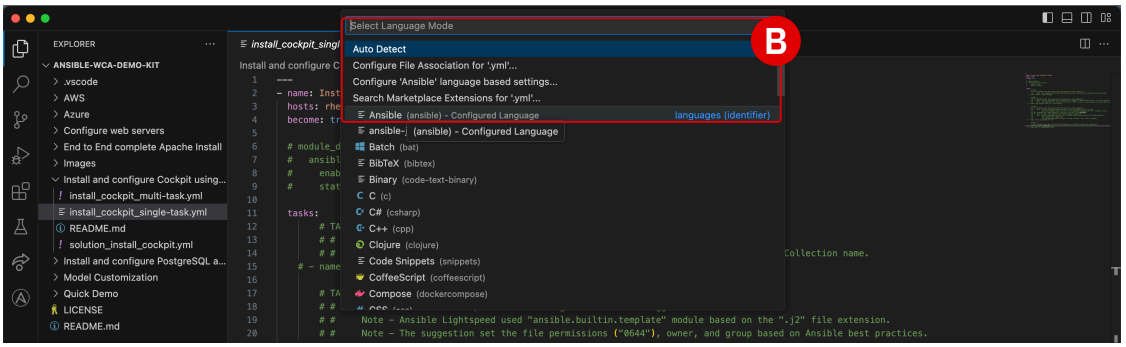
Information "copied" to your local machine's clipboard cannot be "pasted" directly into the virtual machine (VM) environment or VS Code. If you wish to copy and paste instructions directly from the lab documentation, it is recommended that you open the GitHub instructions **inside** the VM's web browser (Firefox). This will allow you to copy instructions to the VM's clipboard and paste instructions inside the VS Code editor.

ANSIBLE LIGHTSPEED IS MISSING OR CODE RECOMMENDATIONS ARE NOT GENERATING

Ansible Lightspeed and WCA will only generate code recommendations for *Ansible Playbooks* and *YAML* files. VS Code will typically auto-detect the programming language of the document you're working with, but on occassion you may need to manually specify the language. Even if working with a YAML file, you'll still need to specify the language mode as `Ansible` for the Lightspeed plugin to engage.

To set the language mode correctly:

- In the bottom-right corner of the VS Code interface, hover over the **Select Language Mode** toggle[A]
- A console will appear at the top of VS Code with a drop-down list of options[B]
- Click `Ansible` from the suggested languages, or enter the text yourself and hit `Enter ⏎`
- Confirm that the Select Language Mode toggle in the bottom-right corner displays `Ansible`



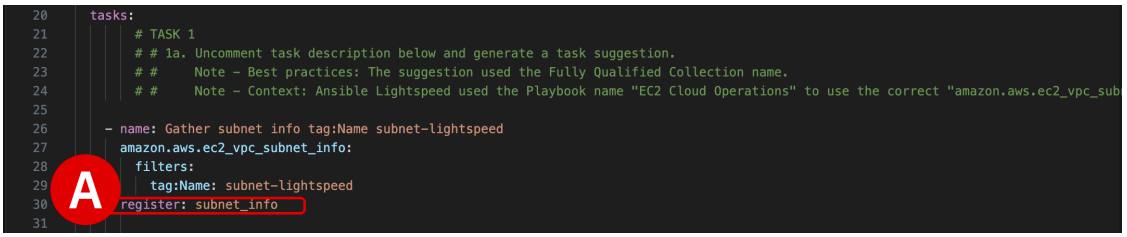
demo_provision_ec2_instance.yml

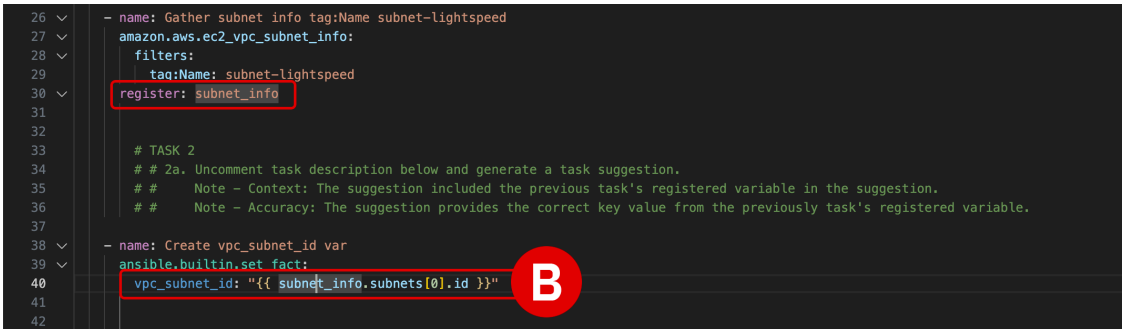
```yaml
---
- name: EC2 Cloud Operations
  hosts: localhost
  connection: local
  gather_facts: false

  module_defaults:
    group/aws:
    region: us-east-1

#  vars:
#    ec2_instance:
#    name: lightspeed-instance-01
#    key_name: lightspeed-keypair
#    image_id: ami-016eb5d644c333ccb # RHEL 9 us-east-1
#    tags:
#      function: lightspeed-demo
#    security_group: secgroup-lightspeed

  tasks:
        # TASK 1
        # # 1a. Uncomment task description below and generate a task suggestion.
        # #     Note - Best practices: The suggestion used the Fully Qualified Collection name.
        # #     Note - Context: Ansible Lightspeed used the Playbook name "EC2 Cloud Operations"
to use the correct "amazon.aws.ec2_vpc_subnet_info" module.

    - name: Gather subnet info tag:Name subnet-lightspeed

        # TASK 2
        # # 2a. Uncomment task description below and generate a task suggestion.
        # #     Note - Context: The suggestion included the previous task's registered variable
in the suggestion.
        # #     Note - Accuracy: The suggestion provides the correct key value from the
previously task's registered variable.

    - name: Create vpc_subnet_id var

        # TASK 3
        # # 3a. Uncomment task description "Provision t3.micro instance" below and generate a
task suggestion.
        # #     Note - Efficiency: The suggestion provides practical variable examples to
improve efficiency.

    - name: Provision t3.micro instance

        # # 3b. Remove the above task and suggestion.
        # #     Uncomment 2nd task description "Provision t3.micro instance using ec2_instance
var".
        # #     Generate an updated suggestion.
        # #     Note - Context: The updated suggestion includes the "ec2_instance variable
fields in the suggestion"

    - name: Provision t3.micro instance using ec2_instance var
```

8. `TASK 1` (**Line 26**) and `TASK 2` (**Line 33**) are responsible for gathering information about a network subnet that is to be provisioned and then creating a virtual private cloud (VPC) definition based on those details.

9. Generate suggested code for `TASK 1` (**Line 26**) by placing the cursor at the end of the line, hitting `Enter ⏎`, and then confirming with `Tab ⇥`. Afterwards, perform the same steps to generate code for `TASK 2` as well.

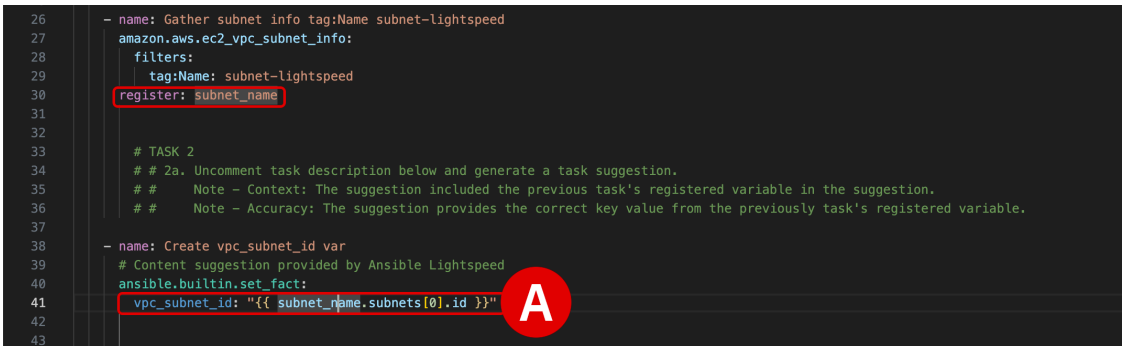
   - The first round of WCA-generated code produces a code block[A] with a `register: subnet_info` (**Line 30**), the result of which is to assign this VPC definition to a variable `subnet_info`. Nothing terribly complicated or surprising about that.
   - The WCA-generated code that follows for the task on **Line 38** (previously *Line 33* in the raw template) recommends a code block[B] with `vpc_subnet_id: "{ { subnet_info.subnets[0].subnet_id } }"` as the value associated with the VPC's subnet ID. Critically, the variable `subnet_info` that was generated in the previous Task[A] is also referenced in the second Task. This demonstrates the contextual awareness of WCA in action.





10. Change the name of variable `subnet_info` to `subnet_name` on **Line 30**. Then delete the code block recommendations under **Line 38** and regenerate the task.

    - Notice that the new block[A] of `TASK 2` now references the variable `subnet_name` that was modified just a moment ago in `TASK 1`
    - WCA has generated code for `TASK 2` that takes into account the **modified context and variables** from `TASK 1` of the Playbook

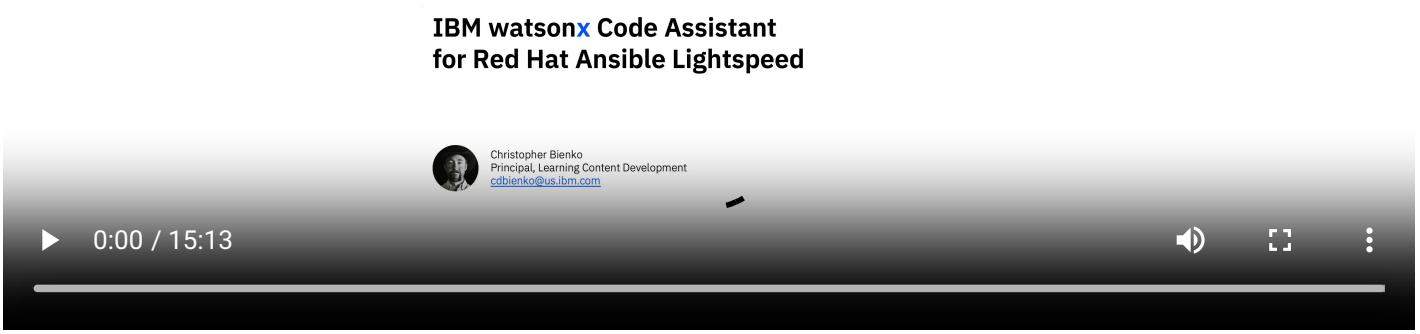


Continue experimenting with WCA's contextual awareness and post-processing capabilities. Try adjusting other variables within the Playbook and study how these modifications impact the generation of later blocks of Task code within the Playbook.

---

iv. Next steps

In the next section, you will begin experimenting with customized Ansible Playbooks and testing how changes to Ansible Task natural language descriptions impacts the recommended code produced by WCA.



> Christopher Bienko *(Principal, IBM Global Sales Enablement)* demonstrates key elements of the `Task Description Tuning and Model Customization` module. *[15 min]*

i. Fine-tuning task prompts

In this section, you will experiment with customized Ansible Playbooks and test how changes made to an Ansible Task's natural language descriptions can impact the recommended code produced by *IBM watsonx Code Assistant for Red Hat Ansible Lightspeed* (**WCA**).

UNPREDICTABLE RESULTS FROM GENERATIVE AI & LLMs

A consequence of using generative AI and Large Language Models (LLMs) is that the recommended code output of these systems will never be 100% consistent for each and every execution. For this reason, LLMs are referred to as "non-deterministic" systems — as opposed to "deterministic" systems. This is both a strength and weakness of LLMs. As you will observe in this module, even a slight modification to the Task description or the smallest change to how a Playbook is structured— the descriptions used, the variables set, and so on —will influence AI-generated code recommendations.

The iterative process that you will go through in this module can be viewed in three different ways:

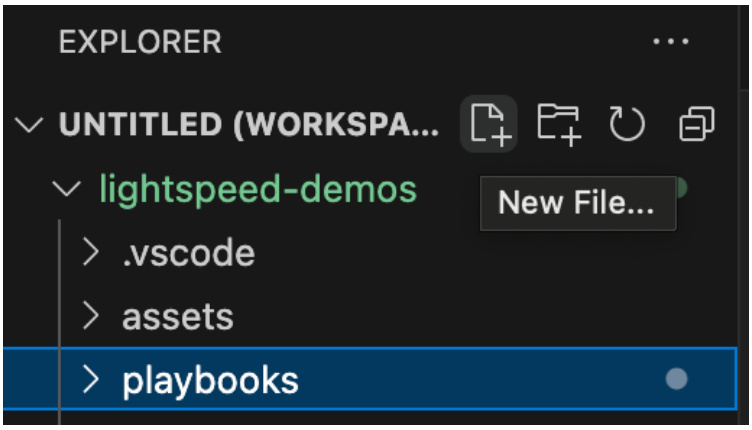
1. On the one hand, it shows the sensitivity of generative AI models to even the most nuanced change in **natural language prompts** — for good or bad. Generative AI can produce tremendous work and that output is further guided along by best practices built-in from Red Hat and IBM. But in the end, the AI can only infer user intent from the natural language descriptions supplied to it. The more clearly a user defines their Task descriptions and intent, the more likely that WCA will correctly generate code which mirrors that intent; conversely, the less precise those descriptions are, the more likely WCA will misinterpret and miss the mark. Precision *is* key for the disambiguation of natural language prompts.
2. Human feedback and **humans-in-the-loop** are essential to these formative stages of generative AI. As offerings like WCA mature, the natural language processing capabilities of the service will continue to be refined and improved. Additional packages, functions, and training data from Ansible Galaxy (as well as other sources) are continuously being added to IBM Granite's LLMs for code, which will in turn continually improve the AI-generated code recommendations made to users.
3. By using the **Model Tuning** capabilities built into WCA, organizations and users are able to customize the recommendations produced by the service, tuning a domain-specific LLM with the organization's *own* Ansible Playbooks. The content and code recommendations that WCA suggests can be tailored to an organization's standards, best practices, and programming styles. These capabilities will be explored later in this module.

The take-away here is that **your results may vary**: they may differ from the `SOLUTION` code presented in the steps below. Keep this in mind as you work through the examples in this section and understand that it is not a bug, but rather a consequence of working with generative AI in general.

The precision with which a Playbook author describes Ansible Tasks in natural language will determine the accuracy and effectiveness of WCA's generated code recommendations.

---

1. A template `CUSTOM PLAYBOOK #1` YAML file has been prepared for you below.
   - Copy and paste the code from the `TEMPLATE` tab into a **New File...** within VS Code
   - Name and save the file as you see fit: for example, `customplaybook1.yml`

> **HOW TO CREATE NEW `YAML` PLAYBOOKS**
>
> **Note:** You need to copy and paste the contents of the Playbook into a **New File...** within the *same* Lightspeed project directory that was used for the previous lab modules in order for the VS Code extension to engage.
>
> To create a new YAML Playbook within a VS Code environment:
>
> **a.** Copy the contents of the Playbook to clipboard using the button in the top-right corner of the lab guide code block.
>
> ```
> ıd launch the service
> ```
>
> ```
> ginx webserver and launch the service
> akcshetty@in.ibm.com)
> ```
>
> **b.** Return to your VS Code environment. In the top-left corner of the interface, with your Ansible Lightspeed folder selected, click the **New File...** button.
>
> 
>
>
> **c.** Name the file to a description of your choosing, ending with `.yml` as the filetype. Set it to `CustomPlaybook.yml`, for example. Save it to one of the directories in the `ansible-wca-demo-kit` folder.
>
> **d.** Paste the clipboard contents into the YAML file and follow along with the suggestions below.

COPY AND PASTE CODE WITHIN THE VM

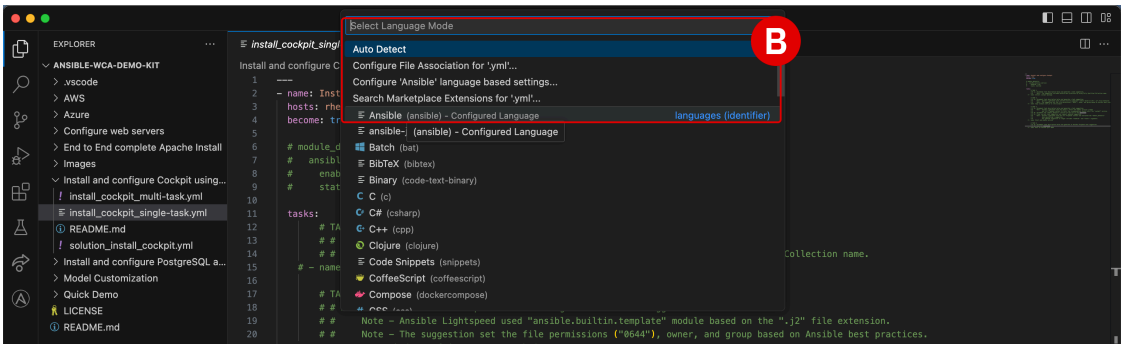
Information "copied" to your local machine's clipboard cannot be "pasted" directly into the virtual machine (VM) environment or VS Code. If you wish to copy and paste instructions directly from the lab documentation, it is recommended that you open the GitHub instructions **inside** the VM's web browser (Firefox). This will allow you to copy instructions to the VM's clipboard and paste instructions inside the VS Code editor.

ANSIBLE LIGHTSPEED IS MISSING OR CODE RECOMMENDATIONS ARE NOT GENERATING

Ansible Lightspeed and WCA will only generate code recommendations for *Ansible Playbooks* and *YAML* files. VS Code will typically auto-detect the programming language of the document you're working with, but on occassion you may need to manually specify the language. Even if working with a YAML file, you'll still need to specify the language mode as `Ansible` for the Lightspeed plugin to engage.

To set the language mode correctly:

- In the bottom-right corner of the VS Code interface, hover over the **Select Language Mode** toggle[A]
- A console will appear at the top of VS Code with a drop-down list of options[B]
- Click `Ansible` from the suggested languages, or enter the text yourself and hit `Enter ⏎`
- Confirm that the Select Language Mode toggle in the bottom-right corner displays `Ansible`



CUSTOM PLAYBOOK 1

```
1  # CUSTOM PLAYBOOK #1 — Invoke 2 modules to automatically update 2 types of servers.
2
3  ---
4  # TASK 1
5  - name: Update web servers
6    hosts: webservers
7    become: true
8
9    tasks:
10     - name: Ensure apache is at the latest version
11
12     - name: Write the apache config file
13
14 # TASK 2
15 - name: Update db servers
16   hosts: databases
17   become: true
18
19   tasks:
20     - name: Ensure postgresql is at the latest version
21
22     - name: Ensure that postgresql is started
```

2. The template `CUSTOM PLAYBOOK #1` contains two sets of tasks:
   - `TASK 1` (**Lines 5-12**) checks if the web server software is up to date and runs an update if necessary
   - `TASK 2` (**Lines 15-22**) checks if the database server software is up to date and runs an update if necessary

3. Examine the instruction on **Line 12**, which prompts Ansible to create ("write") a configuration file for an Apache webserver:

```
- name: Write the apache config file
```

4. Place your cursor at the end of **Line 12** and press `Enter ⏎` to generate WCA-recommended code for the task. Accept the recommendation by pressing `Tab ⇥`.

   Two tabs are presented below:

   - `AI-GENERATED CODE` shows the output from running WCA's generative AI capabilities on **Line 12** of the unmodified *CUSTOM PLAYBOOK #1* YAML template.
   - `SOLUTION CODE` shows the expected (correct) code for performing the task that was written by a human programmer. In theory, the AI-generated code *should* be as good— or even superior to —the manually-written solution code.

AI-GENERATED CODE

```
- name: Write the apache config file
ansible.builtin.template:
    src: templates/apache.conf.j2
    dest: /etc/apache2/sites-available/000-default.conf
    owner: root
    group: root
    mode: '0644'
```

SOLUTION CODE

```
- name: Write the apache config file
ansible.builtin.template:
    src: /srv/httpd.j2
    dest: /etc/httpd.config
    mode: "0644"
```

---

5. As you can see, the `AI-GENERATED CODE` recommendations satisfy some of the requests, but also miss the mark in a few areas. When compared to the `SOLUTION CODE` tab:

- Both tabs appropriately utilize the `ansible.builtin.template` Module, which for this request is absolutely correct. **Interesting fact:** earlier (circa 2023) versions of WCA produced code recommendations for the *same task* as this one, which *did not* align with the expected solution. Instead, the recommendation was for the `ansible.builtin.copy:` Module to invoke `content: "{{ _content_ }}"`, which is not correct. This speaks to the continuously improving capabilities of the WCA and IBM Granite models.
- `AI-GENERATED CODE` recommended including statements that explicitly set `owner: root` and `group: root`, which are both appropriate when permissions are set to `mode: '0644'`. While not strictly necessary, these additional statements arguably are an *improvement* over the `SOLUTION CODE` results. It's also worth noting that the request to write the apache config file did not explicitly request `mode: '0644'` in the natural language description, but WCA nevertheless recommended it (post-processing) as this is a best practice for deploying Apache webservers.
- The `src` and `dest` variables are not in agreement across the `AI-GENERATED CODE` and `SOLUTION CODE` tabs. This is an area for improvement. The natural language description wasn't precise about these details; therefore, this is an opportunity where more detailed and verbose instructions could better steer the recommendations WCA returns back with.

> In general, **the more ambiguous the Task description, the greater the likelihood that WCA will misinterpret the author's intent** and suggest unwanted Ansible automation jobs. To help disambiguate our intention, Playbook authors should use more precise natural language terms and descriptions.

---

6. Delete the WCA-suggested lines of code from *Step 4* from the Playbook.

- Rewrite the description on **Line 12** to the following code block, then press `Enter ⏎` and `Tab ⇥` to accept the new WCA code recommendations
- Take note of the much more precise language used to describe the `src` and `dest` variables

```
- name: Write the apache config file where src equals httpd.j2 and dest equals httpd.config
```

7. The resulting WCA-recommended code should be similar to the `AI-GENERATED CODE` tab below. When compared to the `SOLUTION CODE` tab:

   - The `src: httpd.j2` recommendation in the `AI-GENERATED CODE` tab is an exact match to the natural language description set in *Step 6*, and only slightly different (in terms of the directory path used) to the `src: /srv/httpd.j2` variable in the `SOLUTION CODE` tab.
   - The `dest: /etc/httpd/conf/httpd.conf` recommendation in the `AI-GENERATED CODE` tab, unfortunately, still deviates from the `httpd.config` (not the same as `httpd.conf`) destination that was requested in the natural language description. Close, but still far from exact and not matching our specifications. However, it's a vast improvement over the `dest: /etc/apache2/sites-available/000-default.conf` destination that was recommended as a result of *Step 4*'s more ambiguous task description.
   - Perplexingly, the `mode: '0644'` recommendation that was made previously in *Step 4* has been left out of the suggested task code. We can speculate as to why— perhaps the more precise natural language description made in *Step 6* prompted WCA to only generate code for exactly what was specified —but the "black box" nature of generative AI means that we cannot know for certain. Perhaps with another, even more precise iteration the `AI-GENERATED CODE` will match the `SOLUTION CODE`?

   AI-GENERATED CODE

   ```
   1  - name: Write the apache config file where src equals httpd.j2 and dest equals httpd.config
   2  ansible.builtin.template:
   3      src: httpd.j2
   4      dest: /etc/httpd/conf/httpd.conf
   ```

   SOLUTION CODE

   ```
   1  - name: Write the apache config file
   2  ansible.builtin.template:
   3      src: /srv/httpd.j2
   4      dest: /etc/httpd.config
   5      mode: "0644"
   ```

8. Iterate on the lessons learned in *Step 7* and modify the task description to include more details on the permissions applied to the Apache webserver.

   - Delete the WCA-suggested lines of code from *Step 6* from the Playbook.
   - Re-write the description on **Line 12** to the following code block, then press `Enter ⏎` and `Tab ⇥` to accept the new WCA code recommendations. Take note of the much more precise language used to describe the `mode` variable.

   ```
   - name: Write the apache config file where src equals httpd.j2 and dest equals httpd.config and
   mode equals 0644
   ```

9. Once again, the resulting WCA-recommended code should be similar to the `AI-GENERATED CODE` tab below. When compared to the `SOLUTION CODE` tab:

   - WCA correctly picked up on the `mode: '0644'` request made in the modified task description. There is now alignment between the two tabs.
   - With only a few iterations and by disambiguating the natural language description of the Ansible Task to be performed, the code recommendations produced by WCA have been markedly improved.

   AI-GENERATED CODE

   ```
   - name: Write the apache config file where src equals httpd.j2 and dest equals httpd.config
   and mode equals 0644
   ansible.builtin.template:
       src: httpd.j2
       dest: /etc/httpd/conf/httpd.conf
       mode: '0644'
   ```

   SOLUTION CODE

   ```
   - name: Write the apache config file
   ansible.builtin.template:
       src: /srv/httpd.j2
       dest: /etc/httpd.config
       mode: "0644"
   ```

---

ii. Model customization

Since every organization is different, WCA allows users to customize the AI model output to your organization's unique Ansible Playbooks. This allows for personalized code recommendations that are a better fit to your business' unique needs and more reflective of the programming standards set within your organization.

In this scenario, your organization has its own set of Ansible Playbooks that leverage your preferred cloud provider, that uses specific Ansible Modules to manage OpenShift clusters.

---

10. Open the `create-openshift-cluster-ibm-cloud.yml` Playbook located within the `Model Customization` subdirectory of the hands-on lab templates. The full directory address, as well as the Playbook code, are encapsulated in the following code block.

    ~/ansible-wca-demo-kit/Model Customization/create-openshift-cluster-ibm-cloud.yml

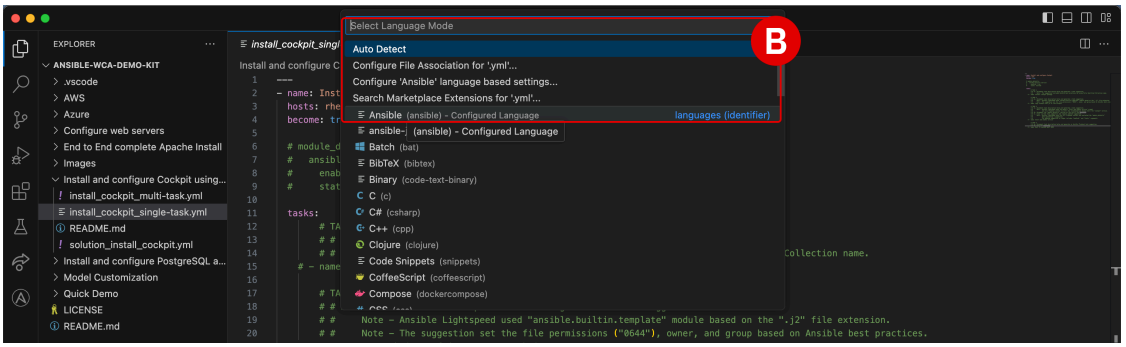
    ```
    ---
    - name: Deploy infrastructure
      hosts: all

      tasks:
        - name: Create an OpenShift cluster
    ```

ANSIBLE LIGHTSPEED IS MISSING OR CODE RECOMMENDATIONS ARE NOT GENERATING

Ansible Lightspeed and WCA will only generate code recommendations for *Ansible Playbooks* and *YAML* files. VS Code will typically auto-detect the programming language of the document you're working with, but on occassion you may need to manually specify the language. Even if working with a YAML file, you'll still need to specify the language mode as `Ansible` for the Lightspeed plugin to engage.

To set the language mode correctly:

- In the bottom-right corner of the VS Code interface, hover over the **Select Language Mode** toggle[A]
- A console will appear at the top of VS Code with a drop-down list of options[B]
- Click `Ansible` from the suggested languages, or enter the text yourself and hit `Enter ⏎`
- Confirm that the Select Language Mode toggle in the bottom-right corner displays `Ansible`



11. Before performing any model tuning tasks, try generating code recommendations for the OpenShift cluster creation task by placing your cursor at the end of **Line 6**, pressing `Enter ⏎`, and then accepting the default recommendations by pressing `Tab ⇥`. The resulting code recommendations— displayed below —represent the standard, unmodified output from WCA's *IBM Granite* base models.

    - When using the standard *IBM Granite* model recommendations, it recommends making use of the `ansible.builtin.command` Ansible Module — which is a standard, best-practice way to perform these types of deployments.
    - However, *your* particular organization (and the public cloud provider they use for deploying OpenShift clusters) might require the use of different Ansible Modules. The following steps will explore how to tailor WCA-generated code blocks to the unique needs of a business.

STANDARD *IBM GRANITE* MODEL RECOMMENDATIONS

```yaml
---
- name: Deploy infrastructure
  hosts: all

  tasks:
    - name: Create an OpenShift cluster
      ansible.builtin.command: oadm create-cluster --wait=false --name=openshift-cluster
      register: oadm_create_cluster
      changed_when: "'created' in oadm_create_cluster.stdout"
      failed_when: "'already exists' not in oadm_create_cluster.stdout"
```

DATA PREPARATION & MODEL TRAINING

Given the scope and time available for this hands-on training, **participants will not be manually creating the training data or training models themselves.** Instead, you will utilize a prepared customized model (`Model ID`) and experiment from *Step 12* and onwards with how the tuned `Model ID` impacts WCA's code recommendations.

- The process of building Ansible training data and tuning customized models for generative AI is potentially a complex and time-consuming one.
- For example, the model tuning employed in the steps ahead for the creation of an OpenShift cluster according to an enterprise organization's specific standards requires approximately **4 hours** to train and generate a customized AI model.

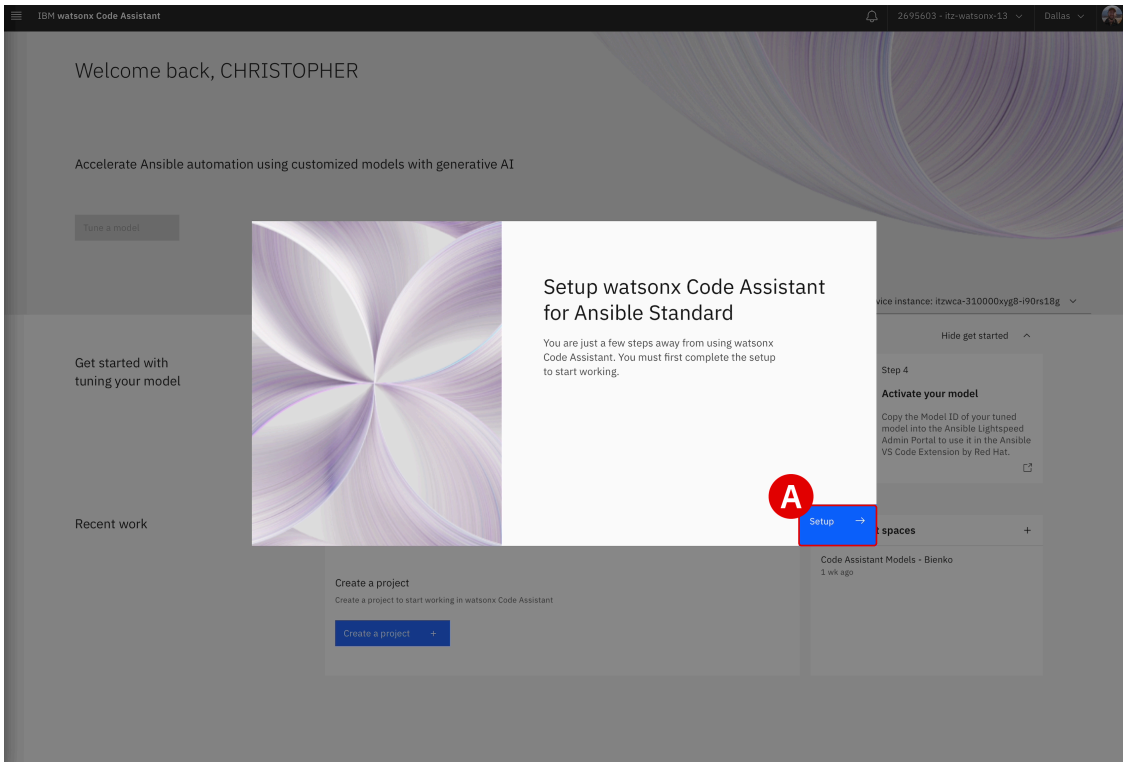
In preparation for model tuning, an organization or user can transform their existing Playbooks into training data for WCA using the open-source **Ansible Parser Tool**. The tool analyzes Ansible Playbooks and generates a single **JSONL** (`ftdata.jsonl` formatted) file that can be uploaded to WCA's model tuning studio for developing customized AI models.

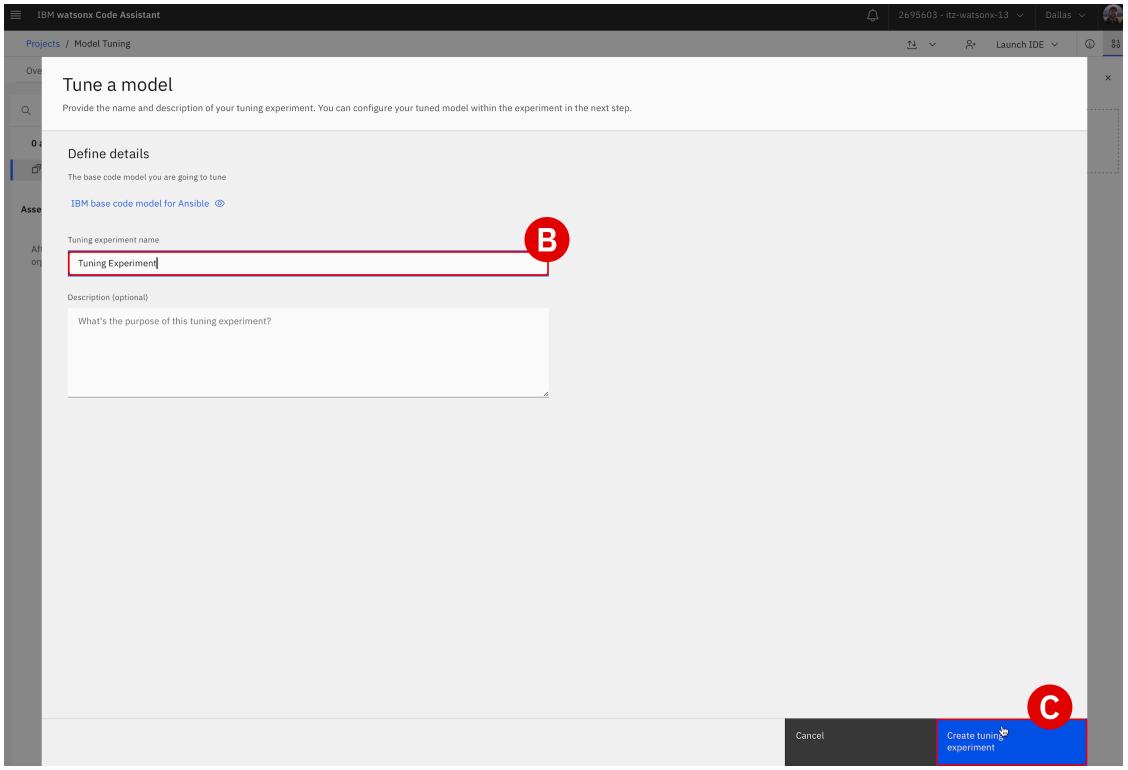
- The process of calibrating and running the Ansible Parser Tool is time-consuming and potentially complex, depending on the scope of Playbooks that an organization wishes to analyze and prepare for model tuning. This falls outside the scope of the hands-on material for this training.
- For the purposes of demonstration, a preconfigured `openshift-tune-micro.jsonl` was used for the model tuning example.

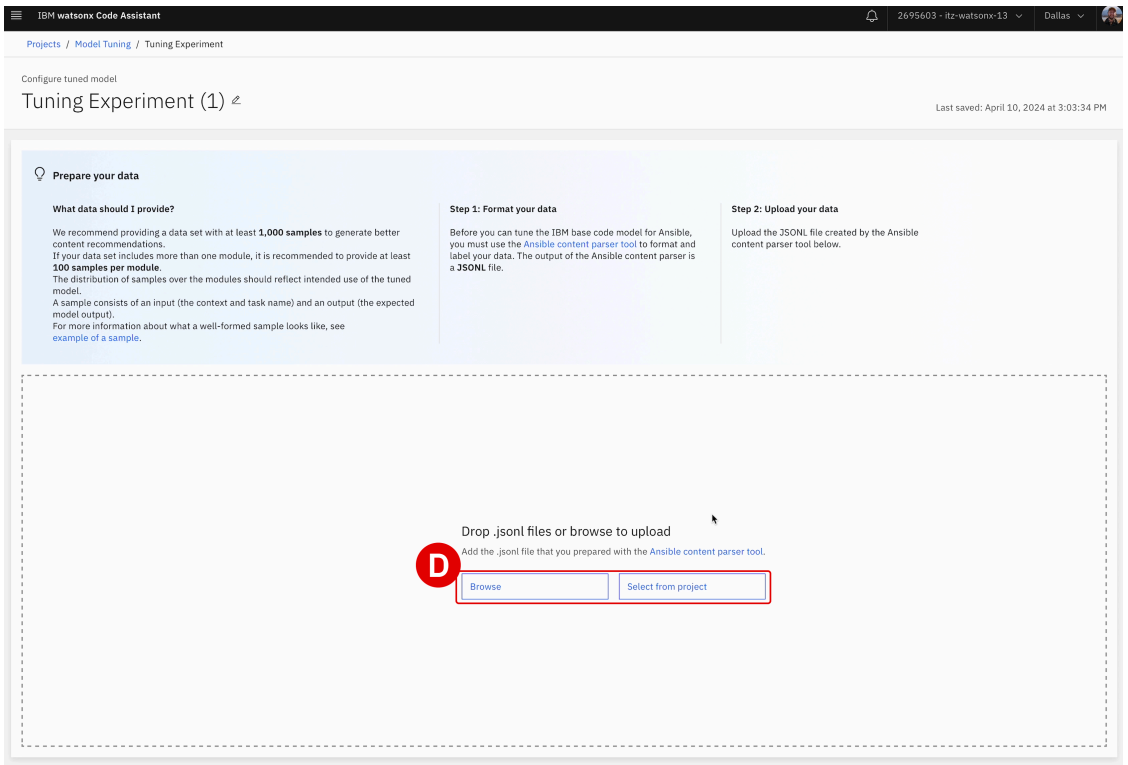
With the Ansible training data prepared, the *IBM watsonx Code Assistant* on IBM Cloud service was configured to execute a new model tuning experiment.

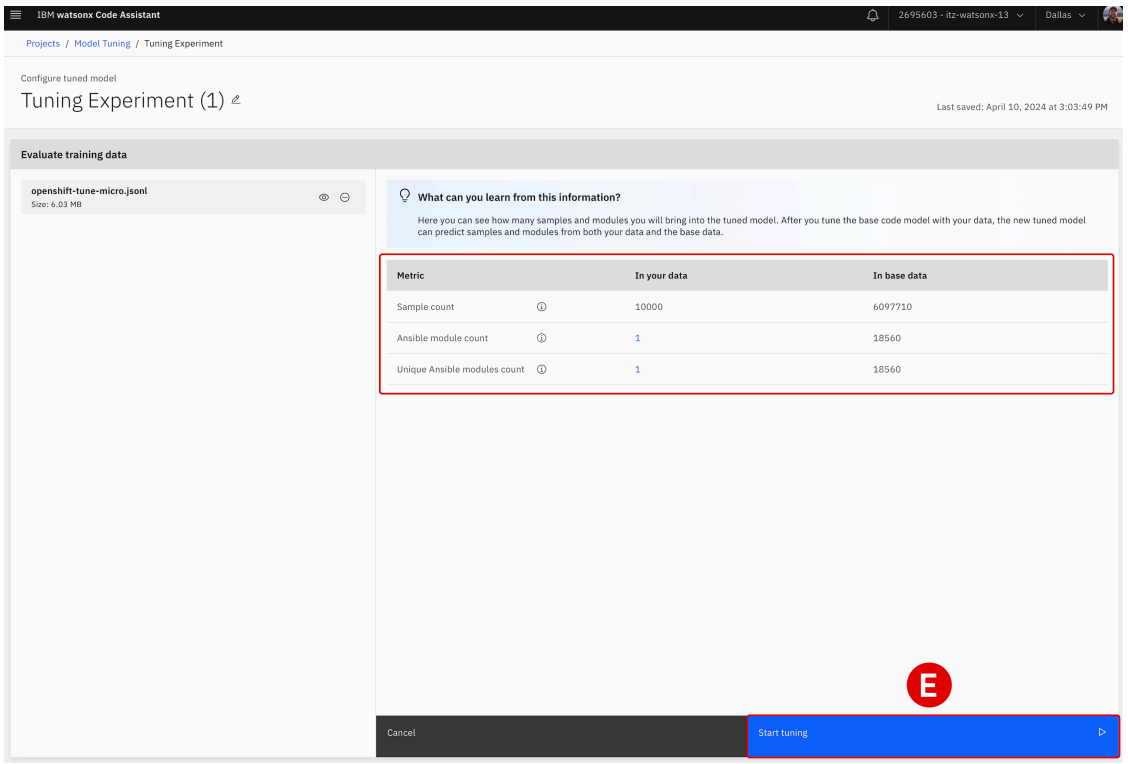
- WCA provides a graphical user experience[A] to streamline the model tuning and customization process
- **Tuning experiment name**[B] set to `Tuning Experiment` and confirmed with **Create tuning experiment**[C]
- The `openshift-tune-micro.jsonl` training data from Ansible Parser Tool is uploaded into WCA[D]
- A summary of metrics— including parameters for sampling and Ansible modules —are displayed within the model tuning wizard before **Start tuning**[E] is selected to kick off the model tuning operating
- After the tuning operation has ended, a training loss graph reveals how accurate the model's predictions are over the training data set, across multiple tuning cycles
- As the number of tuning cycles increase, the training loss rating will tend to decrease
- Model tuning for these particular metrics and parameters takes approximately **4 hours** to complete

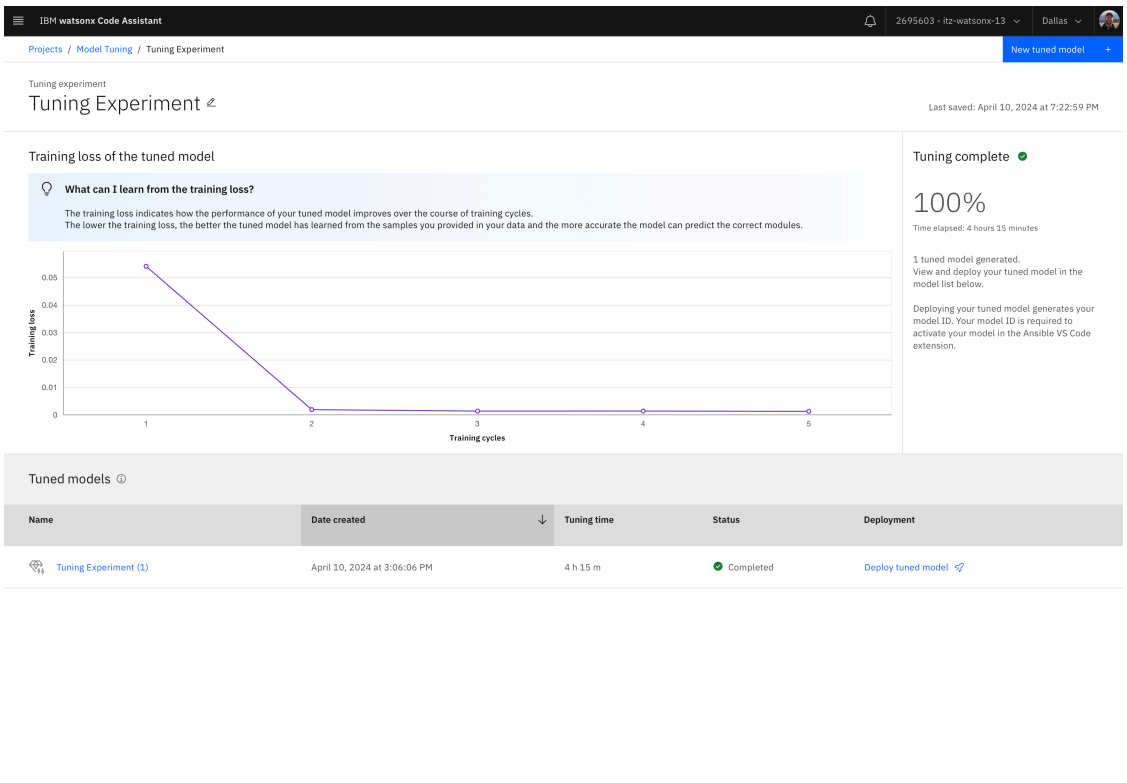
CLICK TO EXPAND — ACCOMPANYING SCREENSHOTS






IBM watsonx Code Assistant
2695603 - itz-watsonx-13
Dallas
Projects / Model Tuning / Tuning Experiment
Configure tuned model
Tuning Experiment (1)
Last saved: April 10, 2024 at 3:03:34 PM
Prepare your data
What data should I provide?
We recommend providing a data set with at least 1,000 samples to generate better content recommendations.
If your data set includes more than one module, it is recommended to provide at least 100 samples per module.
The distribution of samples over the modules should reflect intended use of the tuned model.
A sample consists of an input (the context and task name) and an output (the expected model output).
For more information about what a well-formed sample looks like, see example of a sample.
Step 1: Format your data
Before you can tune the IBM base code model for Ansible, you must use the Ansible content parser tool to format and label your data. The output of the Ansible content parser is a JSONL file.
Step 2: Upload your data
Upload the JSONL file created by the Ansible content parser tool below.
Drop .jsonl files or browse to upload
Add the .jsonl file that you prepared with the Ansible content parser tool.
D
Browse
Select from project

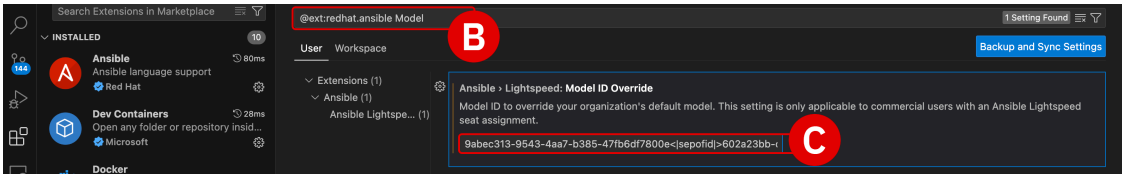

IBM watsonx Code Assistant
2695603 - itz-watsonx-13
Dallas
Projects / Model Tuning / Tuning Experiment
Configure tuned model
Tuning Experiment (1)
Last saved: April 10, 2024 at 3:03:49 PM
Evaluate training data
openshift-tune-micro.jsonl
Size: 6.03 MB
What can you learn from this information?
Here you can see how many samples and modules you will bring into the tuned model. After you tune the base code model with your data, the new tuned model can predict samples and modules from both your data and the base data.
Metric
In your data
In base data
Sample count
10000
6097710
Ansible module count
1
18560
Unique Ansible modules count
1
18560
E
Cancel
Start tuning



12. To **override** the *IBM Granite* base model recommendations, you must provide a pointer to a customized *Model ID* (prepared and hosted earlier on WCA) using the VS Code extension.

> **Custom Model ID**:
>
> ```
> dca38e8f-d98c-44ae-a015-aee0fa4d4360<|sepofid|>069d3e51-c813-407a-94e7-c2c9da414b85
> ```

- Look for Ansible under the *Installed* services in the **Extensions** panel.
- Click the **Manage** ("cogwheel") icon located just to the right of the Ansible tile
- From the drop-down options, click **Settings**[A]
- Within the search bar at the top of the panel, add the text `Model` to filter the results[B]
- Copy and paste the **Model ID** value (above) into the empty `Ansible > Lightspeed: Model ID Override` field[C], then click the `X` in the top-left corner of the panel to save and exit *Settings*



13. With the tuned model, WCA will recommend code using modules, functions, and other details specific to your organization's private IT environment.

    - Return to the `tuning-example.yml` Playbook from *Step 10* and delete the code recommendations generated in *Step 11*
    - Regenerate the code recommendations for the task by placing your cursor at the end of **Line 6**, hitting `Enter ⏎`, and then `Tab ⇥`
    - Observe how the tuned *Model ID* has modified WCA's code recommendations by toggling between the `TUNED MODEL` and `STANDARD MODEL` tabs below

    TUNED MODEL

```yaml
---
- name: Deploy infrastructure
hosts: all

tasks:
    - name: Create an OpenShift cluster
    register: cluster_create_output
    when: cluster_output.rc != 0
    ibm.cloudcollection.ibm_container_cluster:
        name: test_cluster
        datacenter: "{{ datacenter }}"
        machine_type: "{{ machine_type }}"
        hardware: "{{ hardware }}"
        kube_version: 4.14_openshift
        public_vlan_id: "{{ public_vlan_id }}"
        private_vlan_id: "{{ private_vlan_id }}"
        default_pool_size: "{{ default_worker_pool_size }}"
        entitlement: "{{ entitlement }}"
```

    STANDARD MODEL

```yaml
---
- name: Deploy infrastructure
hosts: all

tasks:
    - name: Create an OpenShift cluster
    ansible.builtin.command: oadm create-cluster --wait=false --name=openshift-cluster
    register: oadm_create_cluster
    changed_when: "'created' in oadm_create_cluster.stdout"
    failed_when: "'already exists' not in oadm_create_cluster.stdout"
```

Using the tuned *Model ID*, WCA has returned a recommendation to use the `ibm.cloudcollection.ibm_container_cluster` Module for the IBM Cloud, instead of the standard `ansible.builtin.command` Module from *Step 11*. This more accurately conforms to the organization's Ansible automation standards and requirements.

> ⚠ **REMOVE THE CUSTOMIZED *MODEL ID* WHEN FINISHED**
>
> Remember to **clear** the `Ansible > Lightspeed: Model ID Override` in the *Settings* after you have completed *Step 13*, otherwise all subsequent code generation requests made to WCA will be produced using the customized AI model.

---

iii. Conclusion

This concludes the hands-on components of the Level 3 course, but your learning and experimentation doesn't need to end here. Participants are encouraged to follow the Level 3 accreditation steps (depending on your role).