

0.1 PREREQUISITES & ACCOUNTS

Welcome to Day 0 of the watsonx Workshop Series 🖐️

This is our “pre-flight check” session. The goal is simple: by the end of this module, you’ll know exactly what you need (laptop, software, cloud accounts, repos) so that Day 1 can be 100% hands-on instead of 100% debugging.

AUDIENCE & WORKSHOP OVERVIEW

This workshop is designed for:

- **Data scientists & ML engineers** who want to go from “LLM playground” to RAG and agents in production.
- **Developers & architects** who need to connect LLMs to real systems (APIs, data stores, governance).
- **Technical leaders** evaluating how watsonx.ai, local LLMs (Ollama), and a RAG accelerator fit into their stack.

You don't need to be a deep learning researcher, but you should be comfortable with:

- Basic Python (functions, virtualenvs, `pip`, Jupyter).
- Running commands in a terminal.
- Very basic Docker concepts (build image, run container).

WORKSHOP STRUCTURE

We'll work across **3 core days** plus an optional Day 0 and optional Capstone:

- **Day 0 (Monday, 2h)** – Environment setup
 - Install tools, clone repos, test notebooks.
- **Day 1 (Tuesday)** – LLMs & Prompting (Ollama vs watsonx.ai)
- **Day 2 (Wednesday)** – RAG (Retrieval-Augmented Generation)
- **Day 3 (Thursday)** – Orchestration, Agents & Recap

TECHNICAL PREREQUISITES

Before you can follow the labs, make sure your machine meets these requirements.

OPERATING SYSTEM

You should be able to use any of:

- **Windows 10+**
- **macOS 12+**
- **Linux** (Ubuntu 20.04+, Debian, Fedora, etc.)

If you're on a locked-down corporate laptop, you may need help from IT to install Docker or run containers.

MINIMUM HARDWARE

These are not hard limits, but they're good guidelines:

- **CPU:** 4+ cores
- **RAM:** 16 GB recommended (8 GB possible with smaller models)
- **Disk:** 20–30 GB free (Docker images + models + notebooks)

For local LLMs via Ollama:

- Tiny models (0.5B–1B parameters) are fine on 8 GB RAM.
- 7B models are happier with ~16 GB RAM

ACCOUNTS & ACCESS

To use watsonx.ai you need an IBM Cloud account and access to the watsonx services.

IBM CLOUD

1. Create or use an existing IBM Cloud account.
2. Ensure you have access to:
 - **watsonx.ai**
 - (Optional, but recommended) **watsonx.governance**
 - (Optional) **watsonx.orchestrate**

Your instructor / organizer should tell you:

- Which **region** to use (e.g., `us-south`).
- Whether you'll use a shared project or create your own.
- If there is a pre-configured resource group.

WATSONX PROJECT INFORMATION

You will need:

- IBM Cloud API key
- **watsonx endpoint URL**

e.g. `https://us-south.ml.cloud.ibm.com`

TOOLS TO INSTALL BEFORE DAY 0 (OPTIONAL BUT STRONGLY RECOMMENDED)

If you have time *before* the workshop, install these locally so Day 0 is just validation.

GIT

- **Windows:** Download Git for Windows from the official site and follow the installer.
- **macOS:** Git usually comes via Xcode Command Line Tools:

```
xcode-select --install
```

- **Linux (Ubuntu example):**

```
sudo apt-get update  
sudo apt-get install -y git
```

PYTHON 3.11

- **Windows:** Install from python.org and check “Add to PATH”.
- **macOS (Homebrew):**

REFERENCE REPOSITORIES & ASSETS







During the workshop you will clone and/or have access to:

REPOSITORIES

- **simple-ollama-environment** Minimal Python 3.11 + Jupyter + Ollama setup, with:
 - Docker support.
 - `notebooks/ollama_quickstart.ipynb`.
- **simple-watsonx-environment** Minimal Python 3.11 + Jupyter + watsonx.ai integration:
 - `.env.sample` for credentials.
 - `notebooks/watsonx_quickstart.ipynb`.
 - Dockerfile + Makefile for easy setup.
- **watsonx-workshop** The repository that hosts:
 - This documentation.
 - The **accelerator/** folder:
 - `rag/` – retrieval + pipeline code.
 - `service/` – FastAPI API.
 - `tools/` – ingestion & evaluation scripts.

WHAT YOU WILL HAVE AFTER DAY 0

By the end of Day 0, you should have:

-  **Cloned:**
 - `simple-ollama-environment`
 - `simple-watsonx-environment`
 - `watsonx-workshop` (with `accelerator/` and `labs-src/`)
-  **Working Jupyter** in both env repos.
-  A basic **Ollama chat** running from `ollama_quickstart.ipynb`.
-  A basic **Granite / watsonx.ai call** running from `watsonx_quickstart.ipynb`.
-  The `accelerator/` folder available locally.
-  All reference notebooks (`labs-src/` and `accelerator` notebooks) opening in Jupyter.

When those boxes are ticked, you're ready to hit the ground running on **Day 1**.

0.2 SETUP `simple-ollama-environment`

In this section we'll get your **local LLM sandbox** running: Python 3.11, Jupyter, and [Ollama](#) packaged together in a reproducible way.

You can choose either:

- A **Docker-first** setup (recommended for consistency), or
- A **local virtual environment** using your host's Python 3.11 and an existing Ollama install.

GOAL

By the end of this lab you will:

- Have the **simple-ollama-environment** repo cloned.
- Be able to launch a Jupyter Notebook.
- Run `notebooks/ollama_quickstart.ipynb` and chat with a local LLM (e.g., `qwen2.5:0.5b-instruct` or `llama3.2:1b`).

REPOSITORY OVERVIEW

Once cloned, you'll see something like:

```
simple-ollama-environment/  
├── Dockerfile  
├── Makefile  
├── pyproject.toml  
├── README.md  
├── assets/  
│   └── screenshot.png (example)  
└── notebooks/  
    └── ollama_quickstart.ipynb
```

Key pieces:

- **Dockerfile** Builds a container image that bundles:
 - Python 3.11
 - Jupyter
 - Ollama (server + CLI)
 - A small pre-pulled model (configurable)
- **Makefile** Cross-platform shortcuts for:
 - `make install` – local venv + kernel.
 - `make build-container` – Docker image.
 - `make run-container` – run image with ports & volumes.

STEP 1 – CLONE THE REPOSITORY

Pick or create a parent folder for all workshop repos:

```
mkdir -p ~/projects/watsonx-workshop  
cd ~/projects/watsonx-workshop
```

Clone:

```
git clone https://github.com/ruslanmv/simple-ollama-environment.git  
cd simple-ollama-environment
```

You should now be inside the repo root.

STEP 2 – CHOOSE SETUP PATH

You have two main options.

OPTION A – DOCKER (RECOMMENDED)

Best if:

- You want minimal local setup.
- You're happy to let Docker handle Python + Ollama.

A.1 BUILD THE CONTAINER IMAGE

From the repo root:

```
make build-container
```

Under the hood this runs `docker build` and creates an image (for example `simple-ollama-environment:latest`) that includes:

- Python 3.11 + dependencies.
- Jupyter.
- Ollama server + client.
- A tiny pre-pulled model (configurable via `PREPULL` build arg).

A.2 RUN THE CONTAINER

STEP 3 – INSTALL & CONFIGURE OLLAMA MODELS

If you're using the Docker image with `OLLAMA_PREPULL`, some models may already be present. Otherwise, you can pull them yourself.

PULL A SMALL MODEL

Examples:

```
# From host or inside container:  
ollama pull qwen2.5:0.5b-instruct  
ollama pull llama3.2:1b
```

These are small enough to work well on most laptops.

QUICK HEALTH CHECK

With the container running, you can test:

```
curl http://localhost:11434/api/tags
```

You should see JSON listing available models.

STEP 4 – RUN `ollama_quickstart.ipynb`

Now let's test end-to-end.

1. Open **Jupyter** (either inside the container or local).
2. Navigate to `notebooks/`.
3. Open `ollama_quickstart.ipynb`.
4. Run the cells top to bottom.

You should see something along the lines of:

```
import ollama

response = ollama.chat(
    model="qwen2.5:0.5b-instruct",
    messages=[{"role": "user", "content": "Tell me a joke about AI and coffee."}],
)
print(response["message"]["content"])
```

If everything is wired correctly, you'll get a text response from the model.

HOW THIS RELATES TO RAG & THE ACCELERATOR

Right now, you're just sending plain prompts to a local model, but the same patterns will be used later when you:

- Implement a **local RAG notebook** (`rag_local_ollama.ipynb`).
- Compare local RAG vs watsonx.ai RAG.
- Treat local LLMs and watsonx.ai as interchangeable “generation backends” in the **accelerator**.

What you're learning here:

- How to:
 - Talk to Ollama's HTTP API / Python client.
 - Run notebooks in a controlled environment.
- Will directly transfer to:
 - Calling watsonx.ai in the other repo.
 - Plugging a watsonx.ai LLM into the `accelerator/rag/pipeline.py`.

TROUBLESHOOTING

OLLAMA NOT REACHABLE

- Make sure the container is running (`docker ps`) or the desktop app/service is started.
- Check that `curl http://localhost:11434/api/tags` returns JSON.
- In Docker: ensure you mapped `-p 11434:11434`.

JUPYTER TOKEN ISSUES

- If the browser asks for a token:

```
docker logs simple-ollama-env | grep "http://127.0.0.1"
```

Copy the URL with the token.





MODEL TOO BIG / OUT OF MEMORY

- If 7B or 13B models crash:
 - Use smaller models (0.5B–1B).
 - Close other applications.
 - Reduce concurrency.

PORTS ALREADY IN USE

CHECKLIST

Before moving on:

-  Repo cloned (`simple-ollama-environment`)
-  Dependencies installed (Docker image or venv)
-  Jupyter starts successfully
-  `ollama_quickstart.ipynb` runs a model and prints a response

If all green: you're ready to set up `simple-watsonx-environment` next.

0.3 SETUP `simple-watsonx-environment`

Now we'll set up your **watsonx.ai sandbox**: a clean Python 3.11 + Jupyter environment that knows how to talk to Granite / Llama models hosted on IBM watsonx.ai.

You can run it **locally** (virtualenv) or via **Docker** with minimal fuss.

GOAL

By the end of this lab you will:

- Have the **simple-watsonx-enviroment** repo cloned.
- Provide **IBM Cloud credentials** via a `.env` file.
- Run `notebooks/watsonx_quickstart.ipynb` and generate text with a Granite model.
- Understand how this environment relates to the **RAG accelerator** you'll use on Day 2–3.

REPOSITORY OVERVIEW

The repo layout looks like:

```
simple-watsonx-enviroment/  
├── Dockerfile  
├── Makefile  
├── pyproject.toml  
├── .env.sample  
├── notebooks/  
│   └── watsonx_quickstart.ipynb  
└── scripts/  
    ├── mac/  
    ├── ubuntu/  
    └── windows/
```

Key components:

- **Dockerfile** Builds a container with:
 - Python 3.11.
 - Jupyter.
 - `ibm-watsonx-ai` SDK.
 - `langchain-ibm` for LLM integration.
- **Makefile** Offers shortcuts like:
 - `make install` – local venv + Jupyter kernel.
 - `make build-container` – build Docker image.

STEP 1 – CLONE THE REPOSITORY

From your main workshop folder:

```
cd ~/projects/watsonx-workshop # or your path  
git clone https://github.com/ruslanmv/simple-watsonx-enviroment.git  
cd simple-watsonx-enviroment
```

You now have both env repos side by side.

STEP 2 – CONFIGURE `.env` (CREDENTIALS)

This is the most important step: teaching the environment how to authenticate to watsonx.ai.

1. Copy the sample file:

```
cp .env.sample .env
```

2. Edit `.env` with your IBM Cloud details:

```
# Preferred variables
IBM_CLOUD_API_KEY=your_api_key_here
IBM_CLOUD_URL=https://us-south.ml.cloud.ibm.com
IBM_CLOUD_PROJECT_ID=your_project_id_here

# Compatibility aliases (optional)
WATSONX_APIKEY=${IBM_CLOUD_API_KEY}
WATSONX_URL=${IBM_CLOUD_URL}
PROJECT_ID=${IBM_CLOUD_PROJECT_ID}
```

Where to find these values:

- **IBM_CLOUD_API_KEY**
 - IBM Cloud console → Manage → Access (IAM) → API keys.
- **IBM_CLOUD_URL**

STEP 3 – CHOOSE SETUP PATH

OPTION A – LOCAL (VIRTUALENV)

From the repo root:

```
make install
```

This will:

- Create a virtual environment.
- Install Python dependencies from `pyproject.toml`.
- Register a Jupyter kernel, e.g. “**Python 3.11 (watsonx-env)**”.

Start Jupyter:

```
jupyter notebook
```

Then choose the **watsonx-env** kernel when opening notebooks.

OPTION B – DOCKER (RECOMMENDED FOR TEAM CONSISTENCY)

From the repo root:

```
make build-container  
make run-container
```

STEP 4 – RUN `watsonx_quickstart.ipynb`

Time to confirm that credentials + environment are correct.

1. Open Jupyter (local or container).
2. Navigate to `notebooks/`.
3. Open `watsonx_quickstart.ipynb`.
4. Run the cells in order.

A typical pattern inside the notebook looks like:

```
import os
from dotenv import load_dotenv
from ibm_watsonx_ai import APIClient, Credentials
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams

load_dotenv()

api_key = os.getenv("IBM_CLOUD_API_KEY") or os.getenv("WATSONX_APIKEY")
url = os.getenv("IBM_CLOUD_URL") or os.getenv("WATSONX_URL")
project_id = os.getenv("IBM_CLOUD_PROJECT_ID") or os.getenv("PROJECT_ID")

credentials = Credentials(url=url, api_key=api_key)
client = APIClient(credentials=credentials, project_id=project_id)
```

```
model_id = "ibm/granite-13b-instruct-v2"
prompt = "Write a short story about a robot who wants to be a painter."
```

```
params = {
    GenParams.DECODING_METHOD: "greedy",
    GenParams.MAX_NEW_TOKENS: 200
```

If everything is configured correctly, you'll see model output printed in the notebook.

OPTIONAL: LANGCHAIN INTEGRATION

If you prefer LangChain style:

```
from langchain_ibm import WatsonxLLM
from dotenv import load_dotenv
import os

load_dotenv()
api_key = os.getenv("IBM_CLOUD_API_KEY") or os.getenv("WATSONX_APIKEY")
url = os.getenv("IBM_CLOUD_URL") or os.getenv("WATSONX_URL")
project_id = os.getenv("IBM_CLOUD_PROJECT_ID") or os.getenv("PROJECT_ID")

llm = WatsonxLLM(
    model_id="ibm/granite-13b-instruct-v2",
    url=url,
    apikey=api_key,
    project_id=project_id,
    params={"decoding_method": "greedy", "max_new_tokens": 128},
)

print(llm.invoke("Give me 3 study tips for Python."))
```

We'll build on this pattern in later labs.

CONNECTION TO THE `accelerator`/ `PROJECT`

The `accelerator` inside `watsonx-workshop/accelerator/` is where you'll build a **production-like RAG service**:

- **Core RAG logic:**

- `rag/retriever.py`
- `rag/pipeline.py`
- `rag/prompt.py`

- **API:**

- `service/api.py` – FastAPI app exposing `POST /ask`.
- `service/deps.py` – holds configuration (URL, API key, project, index names).

- **Tools:**

- `tools/chunk.py`, `tools/extract.py`, `tools/embed_index.py`, `tools/eval_small.py`

- **UI:**

- `ui/app.py` – Streamlit front-end.

The patterns you used in `watsonx_quickstart.ipynb`:

REFERENCE NOTEBOOKS IN `labs-src/` AND `accelerator/assets/notebook/`

Once your environment is stable, it's worth quickly skimming some reference notebooks:

RAG & VECTOR DB EXAMPLES (`labs-src/`)

- **Elasticsearch + LangChain** `use-watsonx-elasticsearch-and-langchain-to-answer-questions-rag.ipynb`
- **Elasticsearch Python SDK** `use-watsonx-and-elasticsearch-python-sdk-to-answer-questions-rag.ipynb`
- **Chroma + LangChain** `use-watsonx-chroma-and-langchain-to-answer-questions-rag.ipynb`

These will inspire your implementation of:

- RAG pipelines in Day 2 labs.
- `retriever.py` & `pipeline.py` in the accelerator.

ACCELERATOR NOTEBOOKS (`accelerator/assets/notebook/`)

- Ingestion & indexing:
 - `Process_and_Ingest_Data_into_Vector_DB.ipynb`

TROUBLESHOOTING

401 / 403 – AUTHENTICATION ERRORS

- Verify:
 - `IBM_CLOUD_API_KEY` is correct.
 - You pasted the whole key (no trailing spaces).
 - You're using the correct `IBM_CLOUD_URL` for your region.
 - The project ID is valid and you have access.

“PROJECT NOT FOUND” / 404

- Double-check the **Project ID** in the watsonx.ai UI.
- Ensure you're using the right region and project/space type.

.env NOT LOADING






- Make sure `.env` is in the repo root (same folder as `Makefile`, `Dockerfile`).
- Ensure the notebook calls `load_dotenv()` at the top.
- If running via Docker, confirm `--env-file .env` is passed.

JUPYTER KERNEL MISSING

- Re-run:

CHECKLIST

Before moving to the final Day 0 step:

-  `simple-watsonx-environment` cloned.
-  `.env` configured with:
 - API key
 - URL
 - Project/space ID
-  Dependencies installed (local venv or Docker image).
-  `watsonx_quickstart.ipynb` runs and returns a Granite response.
-  You know where the `accelerator/` project is and can open its notebooks.

Next up: we'll run a **combined verification** of both environments.

0.4 VERIFY BOTH ENVIRONMENTS

At this point you've set up:

- `simple-ollama-environment` – local LLM sandbox.
- `simple-watsonx-environment` – `watsonx.ai` sandbox.

This final Day 0 module is a **sanity check** to make sure everything works *together*, and that you're ready for Day 1.

GOAL

- Confirm you can:
 - Run a local model via Ollama **inside a notebook**.
 - Run a Granite model via watsonx.ai **inside a notebook**.
- Confirm that:
 - The `accelerator/` folder is present and notebooks open.
 - The `labs-src/` reference notebooks open.
- End with a clear **ready / not ready** checklist.

QUICK VERIFICATION SCRIPT / NOTEBOOK

You can create a tiny notebook (e.g. `verify_envs.ipynb`) in your main folder that does:

```
# verify_envs.ipynb

import os
from dotenv import load_dotenv

print("🔍 Verifying environments...")

# 1) Test Ollama client
try:
    import ollama
    print("✅ ollama Python package is importable")

    res = ollama.chat(
        model="qwen2.5:0.5b-instruct", # or any model you've pulled
        messages=[{"role": "user", "content": "Say hello from Ollama."}],
    )
    print("Ollama says:", res["message"]["content"])
except Exception as e:
    print("❌ Ollama check failed:", e)
```

You don't have to create this combined notebook, but it's a nice, quick sanity check.

Alternatively, you can simply:

```
# 2) Test watsonx.ai client
try:
    • Run ollama_quickstart.ipynb in simple-ollama-environment.
    load_dotenv() # pick up .env from simple-watsonx-environment if you run this there
    • Run watsonx_quickstart.ipynb in simple-watsonx-environment.
    from ibm_watsonx_ai import Credentials
    from ibm_watsonx_ai.foundation_models import ModelInference
    from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams

    api_key = os.getenv("IBM_CLOUD_API_KEY") or os.getenv("WATSONX_APIKEY")
    url = os.getenv("IBM_CLOUD_URL") or os.getenv("WATSONX_URL")
    project_id = os.getenv("IBM_CLOUD_PROJECT_ID") or os.getenv("PROJECT_ID")
```


PAIR CHECK EXERCISE

If you're in a classroom setting, do a quick **pair verification**:

1. Pair up with someone next to you.
2. Each person shows:
 - Jupyter running in **simple-ollama-environment**.
 - `ollama_quickstart.ipynb` successfully returns a model response.
3. Then each person shows:
 - Jupyter running in **simple-watsonx-environment**.
 - `watsonx_quickstart.ipynb` successfully returns a Granite response.

This often surfaces:

- Small typos in `.env`.
- Misconfigured paths.
- Port conflicts.

And you get to practice explaining what you did – which already reinforces Day 1 concepts.

CONFIRM ACCELERATOR & NOTEBOOK PACKS

Next, verify your project scaffolding is complete.

CHECK THE `accelerator/` DIRECTORY

From the `watsonx-workshop` repo root:

```
ls accelerator
```

You should see something like:

```
assets/  assettypes/  config.yaml  rag/  service/  tools/  ui/  ...
```

Try opening one of the accelerator notebooks (read-only is fine for now):

- `accelerator/assets/notebook/notebook:Create_and_Deploy_QnA_AI_Service.ipynb`

Make sure:

- Jupyter loads the notebook.
- You can scroll through the cells.

CHECK THE `labs-src/` FOLDER

From the same repo:

```
ls labs-src
```

COMMON FAILURE MODES

Here are some frequent issues and what to do about them.

OLLAMA ISSUES

- “Connection refused” / timeout
 - Ensure Ollama server is running:
 - In Docker: container up with port 11434 exposed.
 - Local: Ollama app/service started.
- “Model not found”
 - Pull the model:

```
ollama pull qwen2.5:0.5b-instruct
```
- Out-of-memory
 - Use smaller models (e.g., 0.5B–1B variants).

WATSONX.AI ISSUES

WHAT TO DO IF SOMETHING FAILS

If you hit issues:

1. Capture the error

- Copy the error message and the command you ran.

2. Ask for help

- Instructor / Slack / Teams channel.





3. Fallback paths

- If local Docker or Ollama is blocked:
 - You can still follow many labs in the watsonx environment.
 - Or use a pre-provisioned VM / cloud notebook if your team provides one.

The key is: by the time Day 1 starts, you should at least have **one working LLM path** (preferably both).

END-OF-DAY 0 CHECKLIST

Tick off each of these:

-  `simple-ollama-environment`:
 - Repo cloned.
 - Jupyter working.
 - `ollama_quickstart.ipynb` returns a model response.
-  `simple-watsonx-environment`:
 - Repo cloned.
 - `.env` configured with valid IBM Cloud API key, URL, project ID.
 - Jupyter working.
 - `watsonx_quickstart.ipynb` returns a Granite response.
-  `accelerator/`:
 - Folder present.
 - Notebooks under `accelerator/assets/notebook/` open.
-  `labs-src/`:
 - Notebooks open and are readable.