Understanding DevOps Project

✅ PART 1: Viva Questions

1. **What is the objective of our project?**

   The main objective of our project was to automate the setup and training of a machine learning model using DevOps tools like Ansible and Docker. We aimed to make the training process reproducible, scalable, and portable across machines. For example, training a fake news classifier on one system and reproducing the same results on another without manual setup.

   → *"To automate the setup and training of a machine learning model using Ansible and Docker, and make it portable using a Docker image pushed to DockerHub."*

   Example: Fake News Detection in a News Agency

   Suppose a news agency wants to detect fake news. Their data scientists develop a model, but deployment is inconsistent across systems.

   Why is Deployment Inconsistent?

   In a collaborative ML project, multiple people (data scientists, engineers) may try to **run, train, or update** the model — but on **different machines** (Windows, Linux, macOS) with different versions of:

   - Python (e.g., 3.7 vs. 3.10)
   - Libraries (e.g., scikit-learn==0.24.2 vs 1.2.1)
   - Missing system tools (e.g., pip, gcc, nltk, etc.)
   - Varying folder structures (some users run from /home, some from C:/Users/)

   Even worse, if someone edits train.py, the model may behave differently across systems — and that **breaks reproducibility**.

Imagine this:

- A news agency's ML team has built a fake news classifier.
- One person runs it and gets 94% accuracy.
- Another person runs it on their laptop and gets 80%.

- Yet another person can’t even install dependencies properly.

Why?

- One person is using Python 3.10, another has 3.8.
- One has nltk installed, another forgot to download stopwords.
- Someone else doesn’t have the correct CSV file path.
- Someone’s machine doesn't support Docker permissions or has a firewall blocking SSH.

This leads to:

- Debugging nightmares
- Hours wasted trying to “make it work on my machine”
- Frustration and failed deployment to production

My project solves this exact problem by packaging the **entire environment** — OS, dependencies, Python packages, and training script — into a Docker container.
Then, using **Ansible**, I automate the setup of this container across machines.
And using **DockerHub**, I ensure everyone runs **the exact same code**, with **the exact same result**, using just one command.

Summarizing:

- Automating the environment setup
- Dockerizing the model training
- Sharing a working container (via DockerHub)

So now, any teammate, anywhere, can reproduce the model with a single command.

🔄 **Summary Comparison**

Problem Without Automation	**Solved by Your Project**
Different Python versions	Docker uses fixed base image (python:3.9)
Missing libraries	Dockerfile installs everything via pip
Inconsistent training behavior	Same code, same environment = same result
Manual installation on each system	Ansible automates setup
No way to share config or image	DockerHub stores and shares the image globally

2. **What problem does our project solve?**

In machine learning projects, different systems may have different dependencies, versions, and OS configurations, leading to inconsistent results. Our project solves this by automating environment setup and packaging everything inside a Docker image, so it runs identically anywhere. For instance, a teammate using Windows can get the same output as someone on Ubuntu just by running the Docker container.

→ *"Manual setup issues and environment inconsistencies. We automate everything to make ML training reproducible, sharable, and consistent across systems."*

3. **What is DevOps or MLOps in this context?**
   → *"Using DevOps tools like Docker and Ansible in ML pipelines to automate setup, reduce human error, and ensure consistent deployments."*

4. **Why did you use Docker?**

   Docker is a containerization platform that packages software and its dependencies into containers. We used Docker to create a container that includes Python, scikit-learn, and our training script. For example, we used a Dockerfile to build an image that runs `train.py`, which trains the model and saves `model.pkl` and `vectorizer.pkl`.

   → *"To containerize the ML code and its dependencies so it can run consistently on any machine."*

5. **Why did you use Ansible?**

   Ansible is an open-source automation tool that uses YAML playbooks to automate tasks. In our project, we used Ansible to automate the installation of Python, Docker, copying project files to the correct location, building the Docker image, and running the container. This allowed us to train the fake news classifier without any manual steps.

   → *"To automate system configuration — installing Python, Docker, and running the container, both locally and remotely."*

6. **What is DockerHub and how did you use it?**

   DockerHub is a public registry for Docker images. We pushed our trained Docker image (`alamsaim/fakenews-model`) to DockerHub so others can pull and run it without needing the source code. For example, one can simply run `docker run alamsaim/fakenews-model` to retrain the model.

7. **What is the role of the Dockerfile?**

A Dockerfile is a text document with instructions to build a Docker image. Our Dockerfile uses the `python:3.9` base image, installs required Python packages using `requirements.txt`, copies our training script and CSVs, and runs `train.py`. When this image is built and run, it trains the fake news model automatically.

→ *"It defines the steps to build the Docker image — installs dependencies, copies code, and runs the training script."*

8. **Why are two playbooks used? What is the difference?**
   → *"playbook-local.yml runs on the local machine, and playbook-remote.yml is for remote setup via SSH using an inventory."*

9. **What ML algorithm did you use and why?**
   → *"PassiveAggressiveClassifier — it's fast and effective for binary classification tasks like fake news detection."*

10. **What are the input and output files?**
    → *"Inputs: train.csv, test.csv. Outputs: model.pkl, vectorizer.pkl stored in /opt/ml/output."*

11. **How is remote execution done?**
    → *"Via SSH. I set up OpenSSH on a remote system and used Ansible with an inventory file to provision and run the pipeline remotely."*

12. **How can someone else run your project?**
    → *"By cloning the GitHub repo and either using Ansible or Docker to run the training without setup hassles."*

13. **How did you test the correctness of your implementation?**
    → *"By running Docker locally and remotely, verifying model accuracy and file outputs after training."*

14. **What challenges did you face during implementation?**
    → *"Setting up SSH and Docker permissions, understanding remote Ansible, ensuring volume mounts worked."*

15. **How is this project useful in real-world ML development?**
    → *"It reflects how companies automate ML deployment — ensuring reproducibility, scalability, and collaboration."*

16. **What is SSH and why did you need it?**
    SSH (Secure Shell) allows secure remote access to another machine over a network. I used it to run my Ansible playbook remotely — configuring another machine's environment as if I were physically there.

17. **What is OpenSSH?**
    OpenSSH is an open-source toolset for SSH. It provides the sshd service, which allows remote login, and the ssh command to connect.

18. **What is Ansible and what role does it play in your project?**
    Ansible automates system tasks. I used it to install Python, pip, Docker, and run my Docker container, either on my local machine or a remote system.

19. **What is Docker and why use it here?**
    Docker packages code and dependencies in a container so it runs the same everywhere — perfect for reproducible ML workflows.

20. **What is DockerHub and why did you use it?**
    DockerHub is a registry for Docker images. I pushed my trained ML container there (alamsaim/fakenews-model) so anyone can pull and run it without setup.

21. **What are .pkl files and why are they important in ML?**
    .pkl files are Python's way of serializing (saving) objects. In ML, we use them to save trained models or preprocessors for later use.

22. **What is model.pkl?**
    It contains the trained PassiveAggressiveClassifier that detects fake news. It can be reused for prediction without retraining.

23. **What is vectorizer.pkl?**
    It stores the TF-IDF vectorizer used to convert raw text into numerical features — essential for making predictions consistent with training.

24. **What is the role of test.csv and train.csv in the project?**

    These files contain labeled news text data.

    - train.csv is used by train.py to fit the TF-IDF vectorizer and train the model.
    - test.csv is used to evaluate accuracy and print the confusion matrix.
      ➡️ Example:
      If train.csv contains fake and real news headlines, the model learns to detect patterns. test.csv validates how well it learned.

25. **What are the roles of each file in your project folder?**

    - train.py: trains the ML model
    - Dockerfile: builds container environment
    - requirements.txt: lists Python dependencies
    - playbook-local.yml: automates setup on the same system
    - playbook-remote.yml: automates setup on another system
    - inventory.ini: defines remote machines

- README.md: documentation
- model.pkl, vectorizer.pkl: training outputs

26. Why host this on GitHub if remote setup is possible?

GitHub allows version control, sharing, and integration. Even with remote setup, GitHub acts as the ***source of truth*** *— and enables teammates to clone, edit, or automate via CI/CD later.*

27. How does Docker ensure reproducibility?

Docker images contain:

- OS
- Python version
- All libraries
  So the same image runs identically on any system.

28. What is the use of CMD ["python", "train.py"] in Dockerfile?

This tells Docker to automatically run train.py when the container starts.

➡️ Without it, the container would start and do nothing unless a command is manually given.

29. What does vectorizer.pkl do?

It saves the TF-IDF transformation logic so that incoming text can be converted in the **same way** during inference.
➡️ Without it, the model can't understand new input.

30. What is SSH and why is it needed in this project?

SSH allows secure remote access to another machine.
➡️ It's used by Ansible to run playbook-remote.yml on your friend's laptop over network.

31. What's the importance of pushing to DockerHub?

So others don't have to rebuild the image — they just pull and run.
➡️ Example:

docker pull alamsaim/fakenews-model

32. What's the difference between build and run in Docker?

- build: Creates an image (a blueprint)
- run: Starts a container (an instance of that image)

33. What is the benefit of using --rm in docker run?

It removes the container after execution, keeping your system clean.

➡️ Useful for stateless training like this project.

34. What does the inventory.ini **file do?**

This file defines the remote machine(s) Ansible should connect to over SSH.

➡️ Example:

```
[remote]
192.168.1.55 ansible_user=keshu17 ansible_ssh_private_key_file=~/.ssh/id_rsa
```

This tells Ansible to connect to `192.168.1.55` as user `keshu17` and run tasks remotely.

35. How does Ansible know where to run tasks remotely?

Through inventory.ini, which lists the IP address and login credentials.

36. How is your project relevant to MLOps?

It introduces automation, reproducibility, and collaboration into the ML workflow — key pillars of MLOps.

37. How do you know the container is working correctly?

- Terminal output shows accuracy and confusion matrix
- Files model.pkl, vectorizer.pkl appear in output/

38. What is the difference between building a Docker image manually and using Ansible to build it?

- Manual: You run docker build and docker run commands yourself.
- Ansible: Automates these commands + installs Docker if needed.

  ➡️ Example:

  Instead of running: ***docker build -t fakenews-model .***

  You just run: ***ansible-playbook playbook-local.yml***

39. How would you extend this project in future?

Add a prediction API (Flask), automate training on GitHub Actions, or deploy it to a Kubernetes cluster.

40. What happens if Docker isn't installed on the system?

If using Ansible, it will install Docker.
If running manually, Docker commands will fail.

41. Why is reproducibility important in ML?

So that others can verify your work, and the same model can be retrained later with exact behavior.

42. What command pushes a Docker image to DockerHub?

docker push alamsaim/fakenews-model

43. What is /opt/ml/ and why was it used?

It's a structured directory used in ML pipelines (e.g., AWS SageMaker).
➡️ We mimic that by storing source and output files there.

44. Can you deploy this on cloud VMs?

Yes. Just configure inventory.ini with the cloud VM IP and credentials.

45. What does Become: yes mean in Ansible?

It allows Ansible to run tasks as root using sudo.

46. Why separate local and remote playbooks?

To keep logic modular.
Remote playbooks need SSH configuration; local ones don’t.

47. What is the purpose of `requirements.txt`?

`requirements.txt` lists the Python libraries needed for the project: `pandas`, `numpy`, `scikit-learn`, `nltk`. Docker uses this file to install these dependencies inside the image during build. This ensures the container has the correct environment to run `train.py`.

48. What would be a next step to extend this project?

Add a prediction API (Flask), automate training on GitHub Actions, or deploy it to a Kubernetes cluster.Serve it with Flask/FastAPI as an API.

49. What is the difference between image and container?

- **Image** = blueprint
- **Container** = running instance of the blueprint

50. What happens when you run docker run fakenews-model?

It runs train.py inside the container, which:

- Loads CSVs
- Trains model
- Saves .pkl files
- Prints accuracy

51. What's your conclusion on this project?

The project successfully automates ML training using Ansible and Docker. It solves reproducibility issues, supports remote collaboration, and matches real-world MLOps principles. Deliverables (Docker image, model, automation scripts) were tested and validated on local and remote setups.

52. What is the difference between playbook-local.yml and playbook-remote.yml?

`playbook-local.yml` is used to run Ansible tasks on the same machine, while `playbook-remote.yml` targets another machine using SSH and an inventory file. Both perform the same tasks — setting up the ML environment and running Docker — but on different hosts.

53. What are `.pkl` files and what do `model.pkl` and `vectorizer.pkl` contain?

`.pkl` files are serialized objects in Python. `model.pkl` contains the trained classifier, and `vectorizer.pkl` contains the TF-IDF transformer. These files allow us to reuse the model for prediction later without retraining.

PART 2: 🔗 Code-Flow + File Usage (End-to-End Breakdown)

📁 **Let's walk through the exact role and linkage of each file:**

🔹 **1. train.py**

💡 **ML Logic Script**

- Loads train.csv, test.csv
- Trains model using TF-IDF and PassiveAggressiveClassifier

- Outputs accuracy & confusion matrix
- Saves:
    - model.pkl (the trained ML model)
    - vectorizer.pkl (the TF-IDF text vectorizer)

This is the **heart of the ML process**, executed **inside the Docker container**.

🔹 **2. train.csv & test.csv**

💡 **Training and testing datasets**

- Used by train.py
- Must be present in app/ when copied by Ansible or Docker

🔹 **3. Dockerfile**

💡 **Defines the ML Training Container**

- Base image: python:3.9
- Copies your code (train.py, CSVs, requirements.txt)
- Installs dependencies using:

RUN pip install --no-cache-dir -r requirements.txt

- Runs train.py automatically:

CMD ["python", "train.py"]

💡 This file is **used by both Ansible and manual Docker builds** to produce the image fakenews-model.

🔹 **4. requirements.txt**

💡 **Lists Python dependencies**

-pandas

-scikit-learn

-numpy

-nltk

Used during Docker image build via pip install.

🔹 **5. playbook-local.yml**

💡 **Local Automation with Ansible**

- Runs on your own machine
- Does the following:
    - Installs Python + Docker (if needed)
    - Copies app/ to /opt/ml/
    - Builds Docker image: docker build -t fakenews-model /opt/ml
    - Runs container:

docker run --rm -v /opt/ml/output:/opt/ml/output fakenews-model

✅ Best for demo on your own laptop.

🔹 **6. playbook-remote.yml**

💡 **Remote Automation with Ansible**

- Same as playbook-local.yml, but targets another machine over SSH
- Requires:
    - SSH server on remote machine
    - IP address + username defined in inventory

✅ Proves that automation works **remotely**, which is key to MLOps workflows.

🔹 **7. inventory.ini**

💡 **Defines Remote Machine for Ansible**

Example:

[remote]

192.168.1.50 ansible_user=keshu17 ansible_ssh_private_key_file=~/.ssh/id_rsa

Used with:

ansible-playbook -i inventory.ini playbook-remote.yml

🔹 **8. model.pkl & vectorizer.pkl**

💡 **Training Outputs**

- Saved by train.py into /opt/ml/output
- Proves that the model trained successfully
- Can be used for prediction in future deployments

✅ These are **proof of successful model training** and key deliverables.

👩‍🏫 Part 3: Presentation Script :

✅ **Introduction**

"Good morning. I'm Keshu Shukla, and I built a project that automates machine learning setup and training using Ansible and Docker. Let me explain the project by walking you through a real-world example."

✅ **Objective (Use-Case Driven)**

"Imagine an ML engineer wants to train a fake news detection model using a dataset. Manually setting up dependencies like Python, scikit-learn, and handling file structures can be time-consuming and error-prone — especially when collaborating remotely.
My objective was to solve this by:

1. Using Ansible to automate setup — no manual steps
2. Using Docker to package the training script into a reusable image
3. Sharing the trained image via DockerHub
   This makes ML training repeatable, shareable, and setup-free."

✅ **Folder Structure + File Roles**

"Here's the folder structure:

- train.py: loads train.csv, trains a PassiveAggressiveClassifier using TF-IDF, saves model.pkl and vectorizer.pkl
- Dockerfile: creates a container with all necessary ML dependencies
- playbook-local.yml: installs Docker, runs the container
- playbook-remote.yml: does the same, but on a remote system using SSH
- requirements.txt: defines Python dependencies"

(Show this live from terminal or file explorer)

✅ **Live Implementation Flow (Narrated)**

"Let me walk you through the pipeline:

1. The user runs ansible-playbook playbook-local.yml
2. This installs Docker (if missing), copies app/, and builds a Docker image
3. Then runs the container, which executes train.py
4. Training results — including model accuracy and confusion matrix — print in terminal
5. Artifacts are saved to /opt/ml/output → shown as model.pkl and vectorizer.pkl"

✅ **Remote Execution (Simplified Explanation)**

"I also tested the same setup on a remote Windows laptop. I configured OpenSSH, created an inventory file, and used:

ansible-playbook -i inventory.ini playbook-remote.yml

This shows the same result — proving the automation works across machines."

✅ **DockerHub Integration**

"To make it easier to use by others, I pushed the Docker image to DockerHub:

docker pull alamsaim/fakenews-model

docker run --rm -v $(pwd)/output:/opt/ml/output alamsaim/fakenews-model

This runs training in an isolated container without any setup."

✅ **Deliverables (and briefly show them):**

“The main deliverables were:

1. playbook-local.yml — for local automation,
2. playbook-remote.yml — for remote system configuration,
3. Dockerfile — to build the container, and
4. A working Docker image available on DockerHub (alamsaim/fakenews-model).”

✅ **Conclusion**

"This project fully automates the training of an ML model using Ansible and Docker, ensuring reproducibility and remote deployment. All deliverables — playbooks, Dockerfile, trained model are completed and tested. It automates the setup and training pipeline using modern DevOps tools, making ML projects more consistent, collaborative, and production-ready. This aligns with MLOps best practices and is ready to scale or extend for real-world applications."

“The project is implemented as an automated ML pipeline. Using Ansible, I provision the environment and trigger Docker build and run processes. The Docker container encapsulates the ML logic. The training script runs inside the container, and the model + vectorizer are saved to a shared folder. The image can also be pulled from DockerHub, making it scalable and reproducible.”

✅ PART 4: Complete Implementation Flow

Step-by-Step (With Internal Code-Flow)

✅ **Step 1: Run Ansible Playbook (Local or Remote)**

ansible-playbook ansible/playbook-local.yml

Or, for remote setup:

ansible-playbook -i ansible/inventory.ini ansible/playbook-remote.yml

🔧 *What this step does:*

- Starts Ansible on the controller system (your machine)
- Uses the instructions in the YAML file to:
    - Update packages (apt update)
    - Install Python & pip
    - Install Docker
    - Ensure Docker service is running
    - Create a target directory (/opt/ml/)
    - Copy the app/ folder contents to /opt/ml/
    - Build a Docker image from /opt/ml/Dockerfile
    - Run a Docker container from that image, with output volume mounted

---

✅ **Step 2: Ansible Copies Project Files to the Target System**

📂 Files from app/:

- train.py
- requirements.txt
- Dockerfile
- train.csv, test.csv

Are copied to:

/opt/ml/ # on the local or remote system

✅ **Step 3: Docker Image is Built via Ansible**

Ansible runs this on the target system:

docker build -t fakenews-model /opt/ml

🧱 *How this works:*

- The Dockerfile is read and executed line by line:
- FROM python:3.9-slim
- WORKDIR /app
- COPY . .

- RUN pip install --no-cache-dir -r requirements.txt
- CMD ["python", "train.py"]

➡️ It creates a Docker image named fakenews-model that contains:

- The Python runtime
- All required packages
- Your training script
- Input CSVs

✅ Step 4: Docker Container is Run from That Image

Ansible executes:

docker run --rm -v /opt/ml/output:/opt/ml/output fakenews-model

🧠 *What happens now:*

- Docker container starts
- train.py is auto-run (due to CMD ["python", "train.py"])
- Inside train.py, the following happens:

🔍 Internal Flow of train.py:

```
# Load data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

# Vectorize text
vectorizer = TfidfVectorizer(...)
X_train = vectorizer.fit_transform(train['text'])
X_test = vectorizer.transform(test['text'])

# Train model
model = PassiveAggressiveClassifier()
model.fit(X_train, train['label'])
```

```
# Predict + Evaluate
y_pred = model.predict(X_test)
print(accuracy_score(...))
print(confusion_matrix(...))

# Save results
joblib.dump(model, '/opt/ml/output/model.pkl')
joblib.dump(vectorizer, '/opt/ml/output/vectorizer.pkl')
```

✅ Step 5: Output is Stored to Host Machine

Because of:

-v /opt/ml/output:/opt/ml/output

➡️ These files are saved **outside the container** and are accessible from the host:

- /opt/ml/output/model.pkl
- /opt/ml/output/vectorizer.pkl

These are your **key project artifacts**.

✅ (Optional) Step 6: Docker Image is Pushed to DockerHub

If done manually using:

docker tag fakenews-model alamsaim/fakenews-model

docker push alamsaim/fakenews-model

➡️ This makes the image globally available.

Now, anyone (including you or teammates) can skip the Ansible steps and just run:

docker pull alamsaim/fakenews-model

docker run --rm -v $(pwd)/output:/opt/ml/output alamsaim/fakenews-model

---

✅ **Project Summary in Terms of Flow**

Phase	Tool	What Happens
Setup	Ansible	Installs Docker, copies files, builds image
Build	Docker	Creates containerized ML environment
Train	Docker (via train.py)	Runs ML model, prints accuracy, saves files
Output	Host machine	Artifacts saved outside container via volume mount
Share	DockerHub (optional)	Others can use same image without setup

---

PART 5: ✅ How DockerHub Is Essential in This Project Demonstration

---

Automate the ML training pipeline and make it reproducible and portable across systems using DevOps tools (Ansible + Docker).

🧠 **Without DockerHub:**

If you don’t use DockerHub, then:

- Every user must **build the Docker image from scratch** using the Dockerfile
- This requires:
    - Having your code folder
    - Installing Docker
    - Running:
    - docker build -t fakenews-model .

✅ It works…
⛔ But it violates the **“portable and sharable”** aspect of the objective.

✅ **With DockerHub:**

Once you push your image to DockerHub:

Others can just do:

docker pull alamsaim/fakenews-model

docker run --rm -v $(pwd)/output:/opt/ml/output alamsaim/fakenews-model

🚀 They don't need:

- Your source code
- Your Dockerfile
- To build anything manually

🔗 **Why Is That Important?**

Without DockerHub	**With DockerHub**
Everyone must build the image manually	Image is ready to run anywhere
Source code must be shared separately	Image contains code + environment pre-bundled
Errors during Docker build possible	Eliminates build-time dependency issues
Not scalable or collaboration-friendly	Works like a plug-and-play ML training unit

---

✅ **In Your Project’s Context:**

After using Ansible to build and test the image locally (which satisfies automation), you push the final image to DockerHub.

➡️ This becomes a **“published version”** of your ML training pipeline.

It’s essential for:

- Demonstrating reusability
- Allowing remote teammates to run without setup

- Proving you've packaged your solution end-to-end

📜 **In Your Presentation/Viva, Say:**

"DockerHub was essential for fulfilling the 'reproducibility and portability' part of the project objective. Once the Docker image was tested, I pushed it to DockerHub (alamsaim/fakenews-model). Now, any user can reproduce the training process with just two Docker commands — without needing my source code or any setup."