Case Study 2: MLOps Report

Team: 10 (Solo)
Student: Ujjwal Pandit
Date: October 7, 2025

1. Virtual Machine Setup (20 points)

1.a. SSH Access Setup (10 points)

Objective: Establish secure SSH access to the virtual machine and replace the shared class key with a unique personal key.

Steps Taken:

- **Initial Access with Shared Key:** We set the correct permissions and connected using the `student-admin_key` via the command `chmod 400 student-admin_key` and `ssh -i student-admin_key -p 22010 student-admin@paffenroth-23.dyn.wpi.edu`.
- **Generated a New Personal Keypair:** We generated a new, personal ED25519 keypair on the local machine using `ssh-keygen -f my_key -t ed25519`.
- **Secured the VM with Our Personal Key:** We added the `my_key.pub` to the `~/.ssh/authorized_keys` file on the VM and removed any `student-admin` shared key entries to enforce exclusive access.
- **Verified Exclusive Access:** We successfully verified that access was restricted to our new key by running `ssh -i my_key -p 22010 student-admin@paffenroth-23.dyn.wpi.edu`.

Automation: The process is idempotently automated in `deploy.sh`. The script detects if our personal key already works; if not, it uses the shared key once to add `my_key.pub` and remove the shared key before switching to our personal key for all subsequent actions.

1.b. Environment Configuration (10 points)

Objective: Configure the VM to run our products from the GitHub repository.

Steps Taken:

- **Cloned the Repository:** We cloned the repository with `git clone https://github.com/Madargaach-Holdings/hugging_face_mood_app.git`.
- **Created Python Virtual Environment:** We created a Python virtual environment, upgraded `pip`, and installed all dependencies using the commands `python3 -m venv`

`venv && source venv/bin/activate` and `pip install --upgrade pip && pip install -r requirements.txt`.
- **Token Configuration:** The Hugging Face token is written to a secure file on the VM at `~/.config/hugging_face_mood_app.env` as `HF_TOKEN=...` with permissions set to `mode 600`.

Automation: All of these steps are automated in `deploy.sh`. The script handles `apt` installation, a fresh clone, venv creation, requirements installation, token file creation, and service setup.

2. Deployment of Products on Virtual Machines (30 points)

Both products (API-based and locally executed) run from the same codebase and entrypoint (`app.py`), satisfying the assignment requirement.

2.a. Deployment of API-based Product (15 points)

The API-based product path uses Hugging Face's Inference API with the `HF_TOKEN` sourced from the secure environment file on the VM. The Gradio app starts as a `systemd` user service and connects to the Inference API for text generation. No secrets are hard-coded; the token is securely injected via the environment file created by `deploy.sh`.

2.b. Deployment of Locally Executed Product (15 points)

The local path uses a lightweight DistilBERT model loaded on the VM, which is cached after the first run for faster subsequent runs. The same `app.py` supports both modes via configuration, making the deployment uniform. The service runs under `systemd` with `Restart=always`, ensuring it starts on reboot and automatically restarts on crashes.

Supporting Operations and Resilience

- **`scripts/smoke_test.sh`:** A simple HTTP 200 health check for the live URL.
- **`scripts/recover.sh`:** A guarded recovery script that first attempts to restart the user service. If the service is still unhealthy, it escalates to running `deploy.sh` with a file lock and 30-minute cooldown to prevent infinite loops.
- **Optional Cron Watchdog:** An optional cron job running every 10 minutes invokes `scripts/recover.sh` to maintain continuous availability without causing "redeploy storms."

3. Automated Deployment & Auto-Recovery (20+ points)

The final part of the case study was to implement and test a recovery mechanism that could automatically redeploy our application in case of a VM failure.

3.a. Script for Automated Deployment

A comprehensive Bash script, `deploy.sh`, was implemented to automate all manual deployment steps and serve as our single-command recovery entrypoint. The script intelligently checks if the VM already trusts our personal SSH key; if not, it temporarily uses the class `student-admin_key` to add our `my_key.pub`, removes the shared key entry, and switches to our personal key for all subsequent actions. It then proceeds to install required system packages on the VM (Git and Python venv), clones the application repository from GitHub (`Madargaach-Holdings/hugging_face_mood_app`), creates a Python virtual environment, upgrades `pip`, and installs dependencies from `requirements.txt`. Finally, it writes a secure environment file to the VM containing the `HF_TOKEN` for API access, configures a systemd user service with `Restart=always`, enables lingering, and starts the service. The script concludes by verifying and reporting the service status. This provides an idempotent, one-command deployment that can be safely re-run after outages or VM rebuilds.

3.b. Resilience Testing and Active Monitoring

We implemented and validated a layered resilience strategy combining systemd auto-restart, active health monitoring, and guarded redeployment:

- **Health Monitoring (`scripts/smoke_test.sh`)**: A lightweight HTTP smoke test checks the live URL for an HTTP 200 response. This test is fast to run and safe to execute frequently.
- **Guarded Recovery (`scripts/recover.sh`)**: This script runs the smoke test. If the application is unhealthy, it first attempts a lightweight fix by restarting the user service via SSH. If the service restart fails or the URL remains unhealthy, it escalates to a full redeploy by invoking `deploy.sh`. The script includes safety measures such as a lock file to prevent multiple recoveries from running at once and a cooldown window to avoid rapid-fire redeploys. It also uses structured logging to `recover.log` for observability and auditability. The Hugging Face token is read from a local, user-only config file to prevent hardcoding secrets.
- **Watchdog (`cron`)**: A conservative cron job runs the recovery script periodically (e.g., every 10 minutes). The combination of the lock and cooldown ensures that the system only takes action when truly needed.
- **Resilience Test: VM Wipe and Rebuild**: We simulated a catastrophic failure by having the VM wiped/rebuilt. On reconnect, we encountered “Host key verification failed,” confirming the rebuild. We resolved it by removing the old host key from `~/.ssh/known_hosts` and re-accepting the new one. Using the bootstrap `student-admin_key` for initial access, we executed `deploy.sh`. The script successfully re-secured the VM with our personal SSH key, installed all dependencies, cloned the repository, created the virtual environment, and installed and started the systemd service. The application came back online and remained available at the

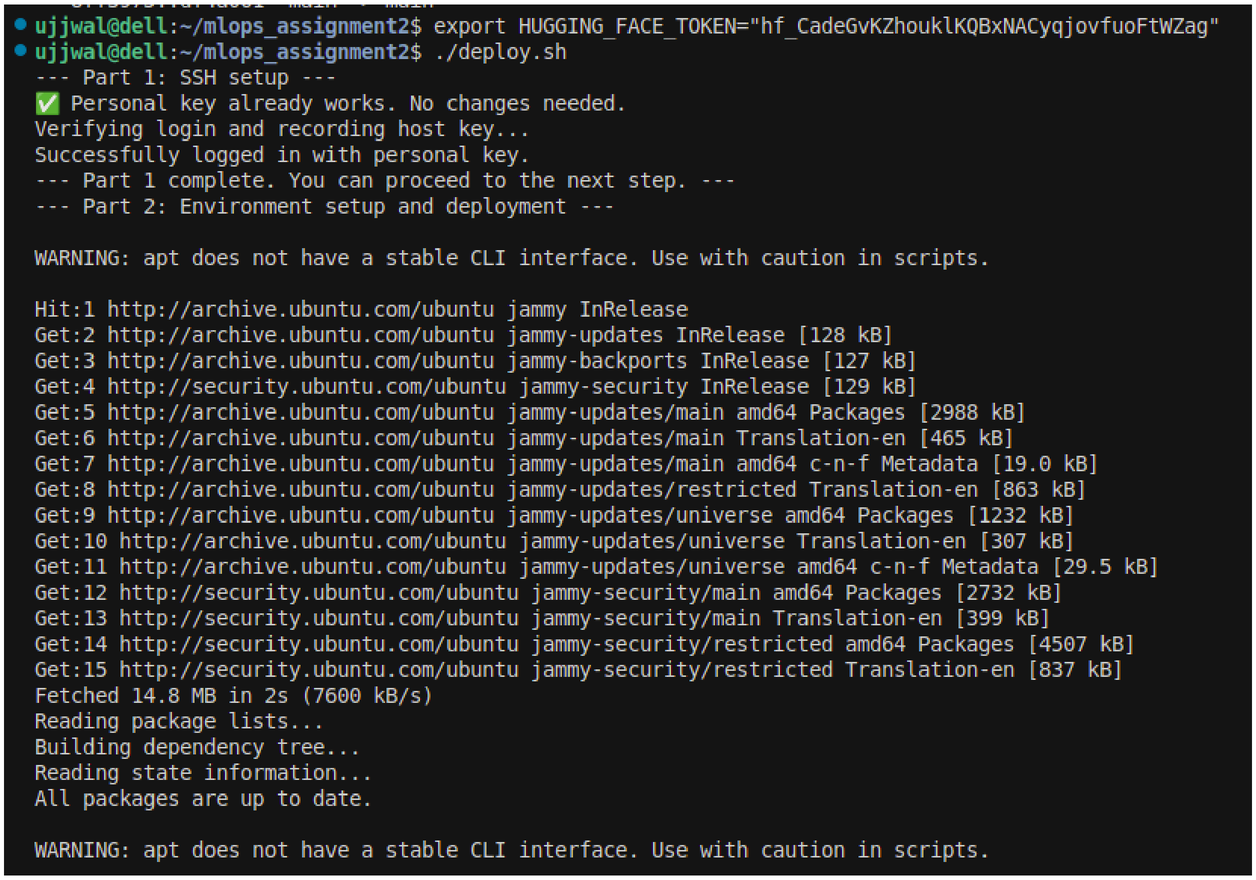
professor-provided URL, demonstrating that our automated deployment and recovery process is effective for catastrophic failures.

Outcome

With systemd's `Restart=always`, our health check, guarded recovery script, and cron-based watchdog, the service self-heals from common process crashes and escalates appropriately for more severe failures (including VM resets), while avoiding "redeploy storms."

Evidence

Live App URL: http://paffenroth-23.dyn.wpi.edu:8010/



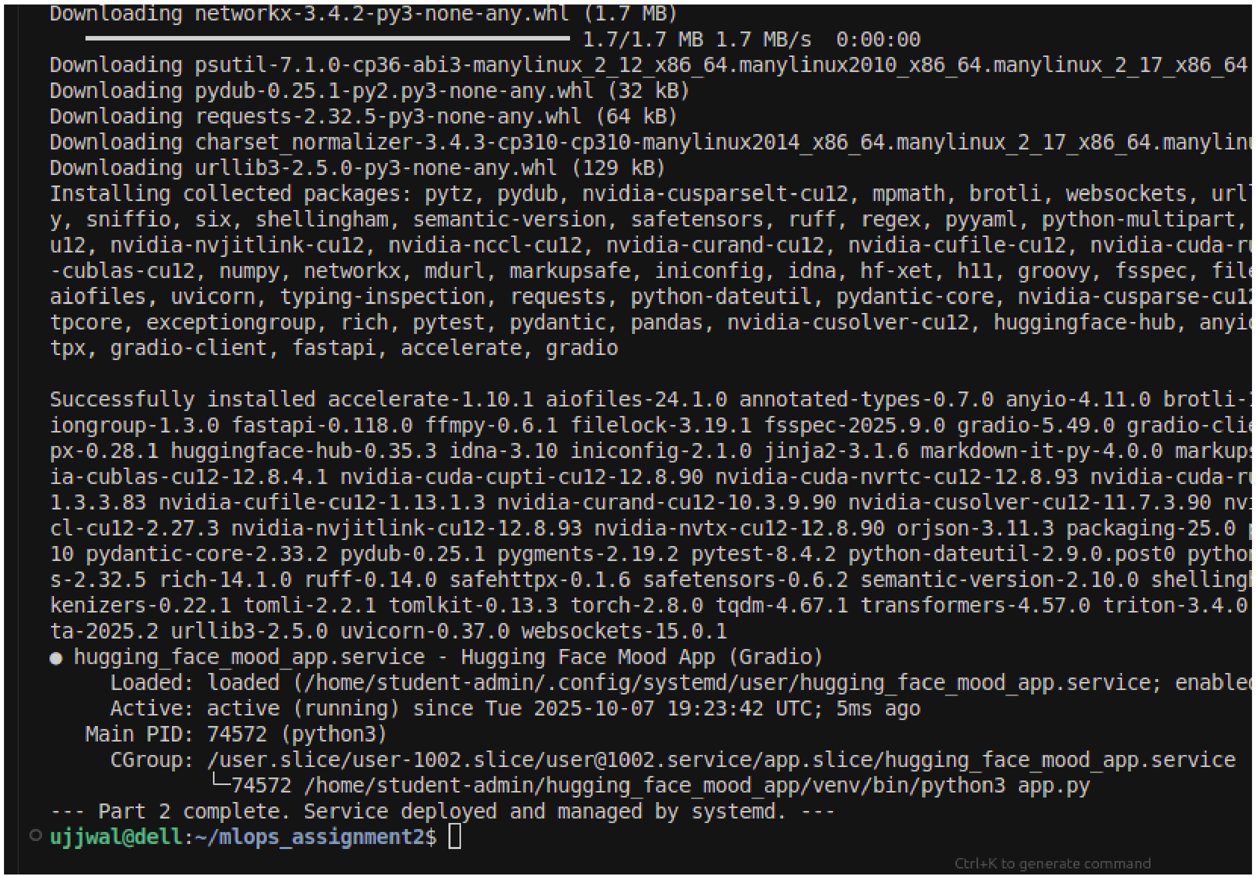
```
● ujjwal@dell:~/mlops_assignment2$ export HUGGING_FACE_TOKEN="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"
● ujjwal@dell:~/mlops_assignment2$ ./deploy.sh
  --- Part 1: SSH setup ---
  ✅ Personal key already works. No changes needed.
  Verifying login and recording host key...
  Successfully logged in with personal key.
  --- Part 1 complete. You can proceed to the next step. ---
  --- Part 2: Environment setup and deployment ---

  WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

  Hit:1 http://archive.ubuntu.com/ubuntu jammy InRelease
  Get:2 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
  Get:3 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
  Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
  Get:5 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2988 kB]
  Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [465 kB]
  Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [19.0 kB]
  Get:8 http://archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [863 kB]
  Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1232 kB]
  Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [307 kB]
  Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [29.5 kB]
  Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [2732 kB]
  Get:13 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [399 kB]
  Get:14 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [4507 kB]
  Get:15 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [837 kB]
  Fetched 14.8 MB in 2s (7600 kB/s)
  Reading package lists...
  Building dependency tree...
  Reading state information...
  All packages are up to date.

  WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

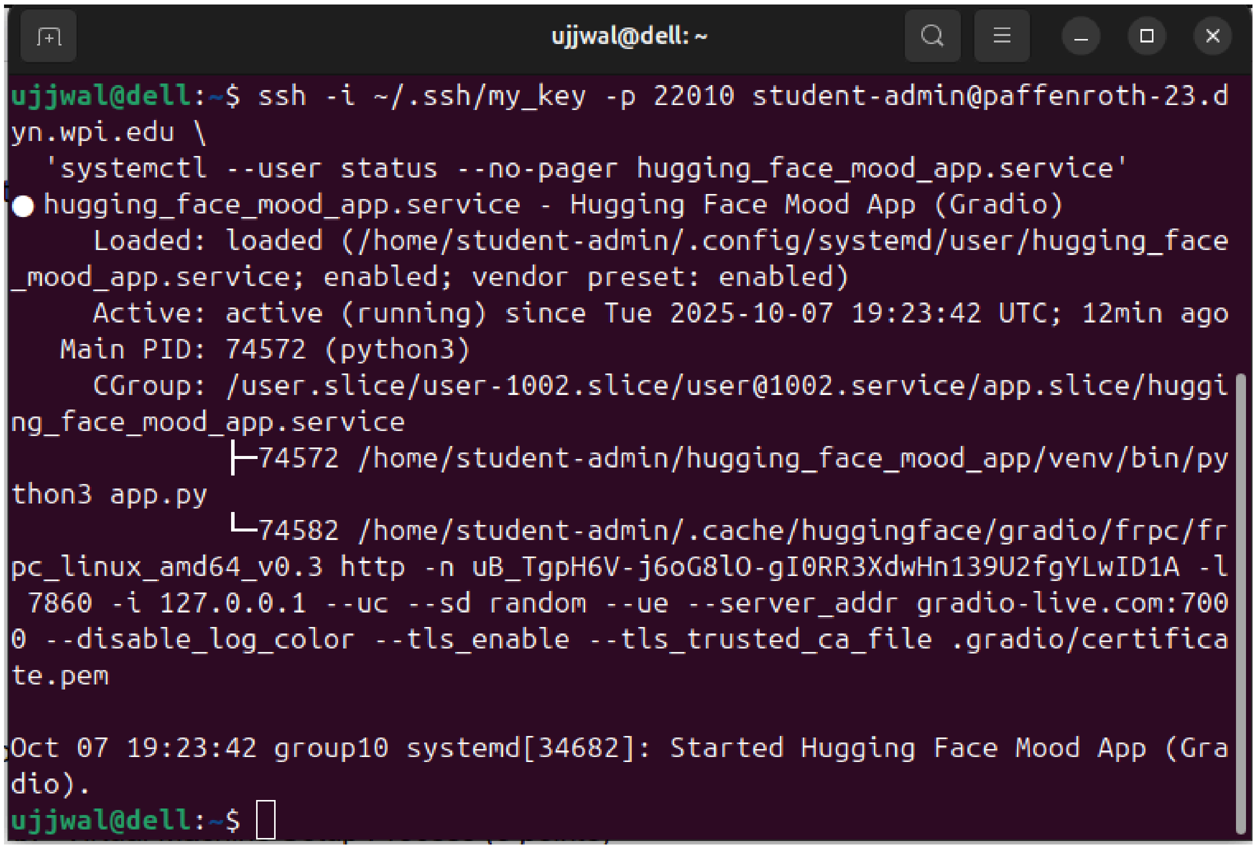
The above snippet was captured when part 1 and part 2 were complete. It logs in the VM with our personal key and then deploys the Case Study 1 there following the instructions given in deploy.sh file.



```
  Downloading networkx-3.4.2-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 1.7 MB/s  0:00:00
  Downloading psutil-7.1.0-cp36-abi3-manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64
  Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
  Downloading requests-2.32.5-py3-none-any.whl (64 kB)
  Downloading charset_normalizer-3.4.3-cp310-cp310-manylinux2014_x86_64.manylinux_2_17_x86_64.manylin
  Downloading urllib3-2.5.0-py3-none-any.whl (129 kB)
  Installing collected packages: pytz, pydub, nvidia-cusparselt-cu12, mpmath, brotli, websockets, url
  y, sniffio, six, shellingham, semantic-version, safetensors, ruff, regex, pyyaml, python-multipart,
  u12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufile-cu12, nvidia-cuda-r
  -cublas-cu12, numpy, networkx, mdurl, markupsafe, iniconfig, idna, hf-xet, h11, groovy, fsspec, file
  aiofiles, uvicorn, typing-inspection, requests, python-dateutil, pydantic-core, nvidia-cusparse-cu12
  tpcore, exceptiongroup, rich, pytest, pydantic, pandas, nvidia-cusolver-cu12, huggingface-hub, anyic
  tpx, gradio-client, fastapi, accelerate, gradio

  Successfully installed accelerate-1.10.1 aiofiles-24.1.0 annotated-types-0.7.0 anyio-4.11.0 brotli-
  iongroup-1.3.0 fastapi-0.118.0 ffmpy-0.6.1 filelock-3.19.1 fsspec-2025.9.0 gradio-5.49.0 gradio-clie
  px-0.28.1 huggingface-hub-0.35.3 idna-3.10 iniconfig-2.1.0 jinja2-3.1.6 markdown-it-py-4.0.0 markups
  ia-cublas-cu12-12.8.4.1 nvidia-cuda-cupti-cu12-12.8.90 nvidia-cuda-nvrtc-cu12-12.8.93 nvidia-cuda-ru
  1.3.3.83 nvidia-cufile-cu12-1.13.1.3 nvidia-curand-cu12-10.3.9.90 nvidia-cusolver-cu12-11.7.3.90 nvi
  cl-cu12-2.27.3 nvidia-nvjitlink-cu12-12.8.93 nvidia-nvtx-cu12-12.8.90 orjson-3.11.3 packaging-25.0 p
  10 pydantic-core-2.33.2 pydub-0.25.1 pygments-2.19.2 pytest-8.4.2 python-dateutil-2.9.0.post0 pythor
  s-2.32.5 rich-14.1.0 ruff-0.14.0 safehttpx-0.1.6 safetensors-0.6.2 semantic-version-2.10.0 shellingl
  kenizers-0.22.1 tomli-2.2.1 tomlkit-0.13.3 torch-2.8.0 tqdm-4.67.1 transformers-4.57.0 triton-3.4.0
  ta-2025.2 urllib3-2.5.0 uvicorn-0.37.0 websockets-15.0.1
● hugging_face_mood_app.service - Hugging Face Mood App (Gradio)
     Loaded: loaded (/home/student-admin/.config/systemd/user/hugging_face_mood_app.service; enabled
     Active: active (running) since Tue 2025-10-07 19:23:42 UTC; 5ms ago
   Main PID: 74572 (python3)
     CGroup: /user.slice/user-1002.slice/user@1002.service/app.slice/hugging_face_mood_app.service
             └─74572 /home/student-admin/hugging_face_mood_app/venv/bin/python3 app.py
--- Part 2 complete. Service deployed and managed by systemd. ---
○ ujjwal@dell:~/mlops_assignment2$ 
```

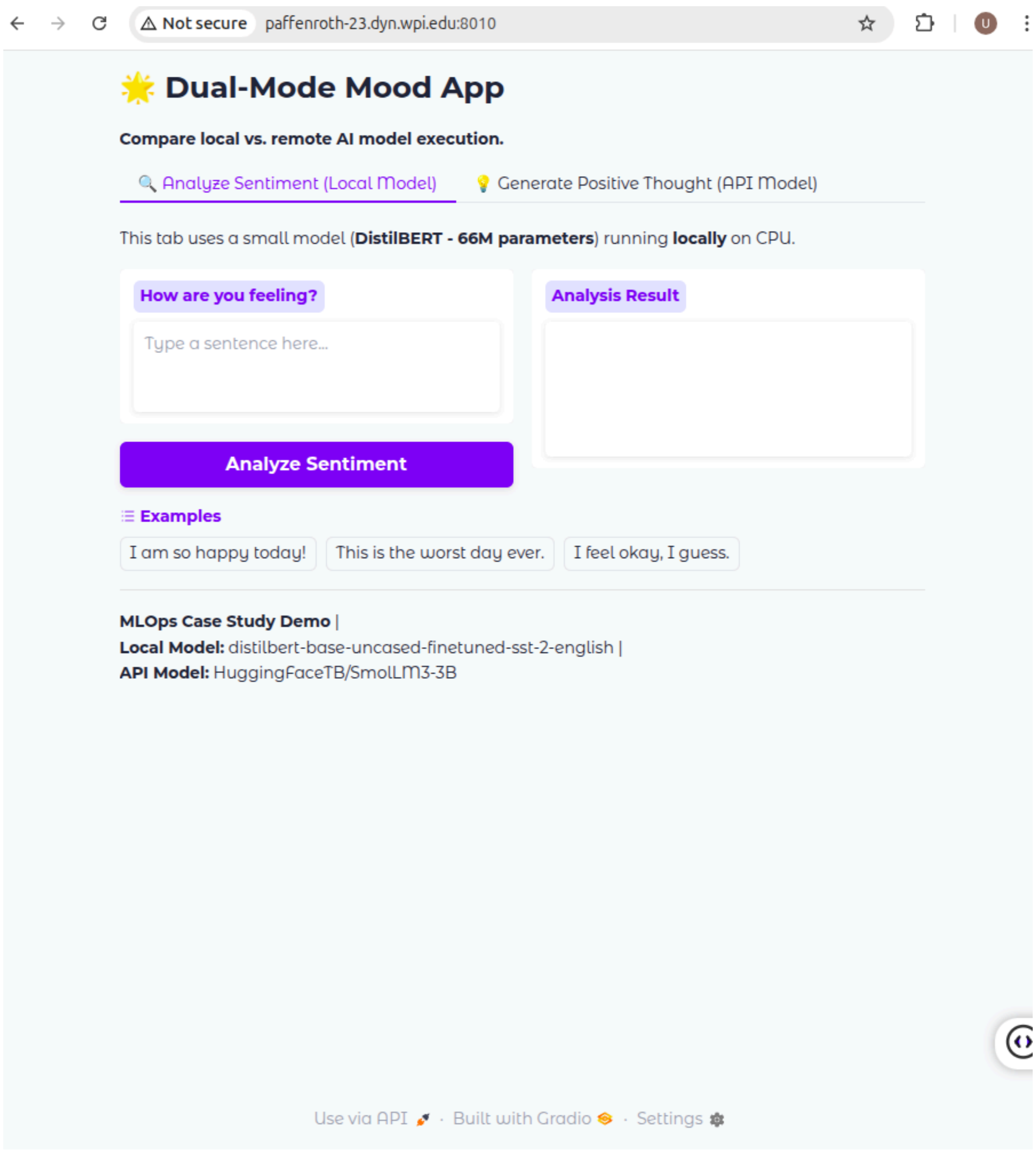
This snippet was taken when the Case Study 1 was deployed for the first time in the VM.



```
ujjwal@dell:~$ ssh -i ~/.ssh/my_key -p 22010 student-admin@paffenroth-23.d
yn.wpi.edu \
  'systemctl --user status --no-pager hugging_face_mood_app.service'
● hugging_face_mood_app.service - Hugging Face Mood App (Gradio)
     Loaded: loaded (/home/student-admin/.config/systemd/user/hugging_face
_mood_app.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2025-10-07 19:23:42 UTC; 12min ago
   Main PID: 74572 (python3)
     CGroup: /user.slice/user-1002.slice/user@1002.service/app.slice/huggi
ng_face_mood_app.service
             ├─74572 /home/student-admin/hugging_face_mood_app/venv/bin/py
thon3 app.py
             └─74582 /home/student-admin/.cache/huggingface/gradio/frpc/fr
pc_linux_amd64_v0.3 http -n uB_TgpH6V-j6oG8lO-gI0RR3XdwHn139U2fgYLwID1A -l
 7860 -i 127.0.0.1 --uc --sd random --ue --server_addr gradio-live.com:700
0 --disable_log_color --tls_enable --tls_trusted_ca_file .gradio/certifica
te.pem

Oct 07 19:23:42 group10 systemd[34682]: Started Hugging Face Mood App (Gra
dio).
ujjwal@dell:~$
```

This snippet explains the quick check of our service if it is up and running through our local terminal. The status is active.



⚠ Not secure paffenroth-23.dyn.wpi.edu:8010

🌟 Dual-Mode Mood App

Compare local vs. remote AI model execution.

🔍 Analyze Sentiment (Local Model)
💡 Generate Positive Thought (API Model)

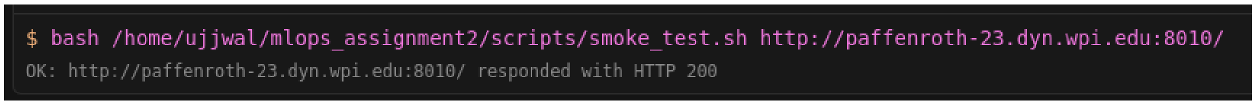
This tab uses a small model (**DistilBERT - 66M parameters**) running **locally** on CPU.

How are you feeling?
Type a sentence here...
Analysis Result
Analyze Sentiment
Examples
I am so happy today!
This is the worst day ever.
I feel okay, I guess.

MLOps Case Study Demo |
Local Model: distilbert-base-uncased-finetuned-sst-2-english |
API Model: HuggingFaceTB/SmolLM3-3B

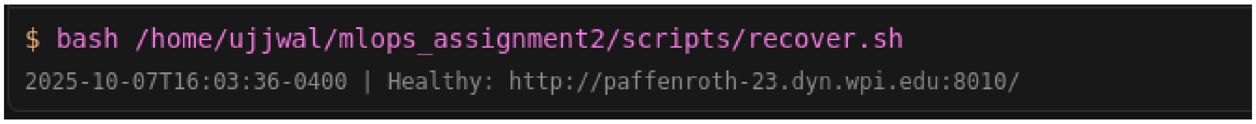
Use via API · Built with Gradio · Settings

This snippet explains that our service is running in the port provided by the professor.



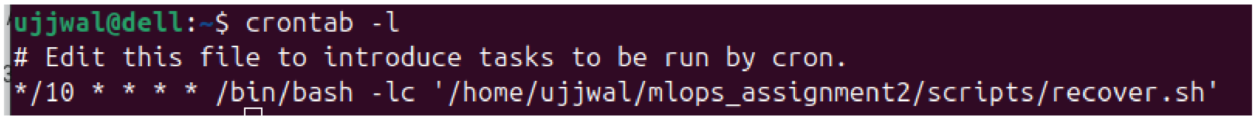
```
$ bash /home/ujjwal/mlops_assignment2/scripts/smoke_test.sh http://paffenroth-23.dyn.wpi.edu:8010/
OK: http://paffenroth-23.dyn.wpi.edu:8010/ responded with HTTP 200
```

This script runs the smoke test against the live URL and we can see it responded with HTTP 200 stating that app is up and running.



```
$ bash /home/ujjwal/mlops_assignment2/scripts/recover.sh
2025-10-07T16:03:36-0400 | Healthy: http://paffenroth-23.dyn.wpi.edu:8010/
```

This script signifies that the recovery script detected the website as healthy and exited cleanly.



```
ujjwal@dell:~$ crontab -l
# Edit this file to introduce tasks to be run by cron.
*/10 * * * * /bin/bash -lc '/home/ujjwal/mlops_assignment2/scripts/recover.sh'
```

Crontab is configured and will append new healthy lines every 10 minutes.

4. Additional Insights, Challenges, and Future Improvements (4 points)

Challenges and Lessons Learned:

- **Host key rotation after VM rebuilds** caused “Host key verification failed.” We resolved it with `ssh-keygen -R` and automated trust-on-first-use. **Lesson:** Always plan for infrastructure identity changes.
- **Key management matters.** Ensuring correct file permissions and removing the shared class key immediately prevented accidental shared access.
- **Systemd auto-restart** covers many failures, but not all (e.g., corrupt environment, code changes). A separate watchdog with conservative cadence, lock, and cooldown prevented “recovery storms.”
- **Secrets handling:** Keeping `HF_TOKEN` out of code and scripts while making it available to both the VM (via an environment file) and local recovery (via a config file) was critical for safety and operability.

What Worked Well:

- **Idempotent `deploy.sh`** made both first-time setup and post-wipe recovery deterministic.
- **`recover.sh`’s layered approach** (restart before redeploy) minimized downtime and avoided unnecessary heavy actions.
- **`smoke_test.sh`** gave a fast SLO check, enabling simple monitoring and a clean pass/fail signal.

Limitations Observed:

- The health check is **HTTP-only** and doesn’t verify model correctness or API quota; it can return `200` even if the service is degraded.
- Recovery depends on **GitHub availability** and the token being valid; if either fails, a redeploy cannot complete.
- The **single-VM architecture** limits scalability and redundancy; resilience is “restart and redeploy,” not high availability.

Potential Future Improvements:

- **Observability:** Add metrics and structured logs (latency, error rates) and alerts (Slack/Telegram) from `recover.sh` on restart/redeploy, and from the app for errors.
- **Health checks:** Add a dedicated `/healthz` endpoint that exercises both tabs (local and API) with a cheap inference to validate end-to-end behavior.
- **Configuration:** Externalize VM/user/ports/token via environment variables or a `.env` file and support multiple environments; add retries with backoff for network operations.
- **Security:** Use a secrets manager (e.g., GitHub Actions secrets for CI, or a local password store) and rotate SSH keys regularly.

- **Delivery:** Pin Python dependencies; add CI to lint shell scripts and run smoke tests; consider Dockerizing for reproducibility; optionally add Nginx/TLS termination for production hygiene.
- **Infrastructure-as-Code:** Consider Ansible/Terraform to rebuild VMs and services from scratch reliably; add periodic backups of critical configs.
- **Deployment strategies:** Introduce versioned releases and a simple blue/green or canary on the VM (two service unit files switching ports) to reduce risk during upgrades.