MACHINE LEARNING OPERATIONS

Basic_mlflow

UNLIMITED

Presented by Asst. Prof. Dr. Tuchsanai Ploysuwan

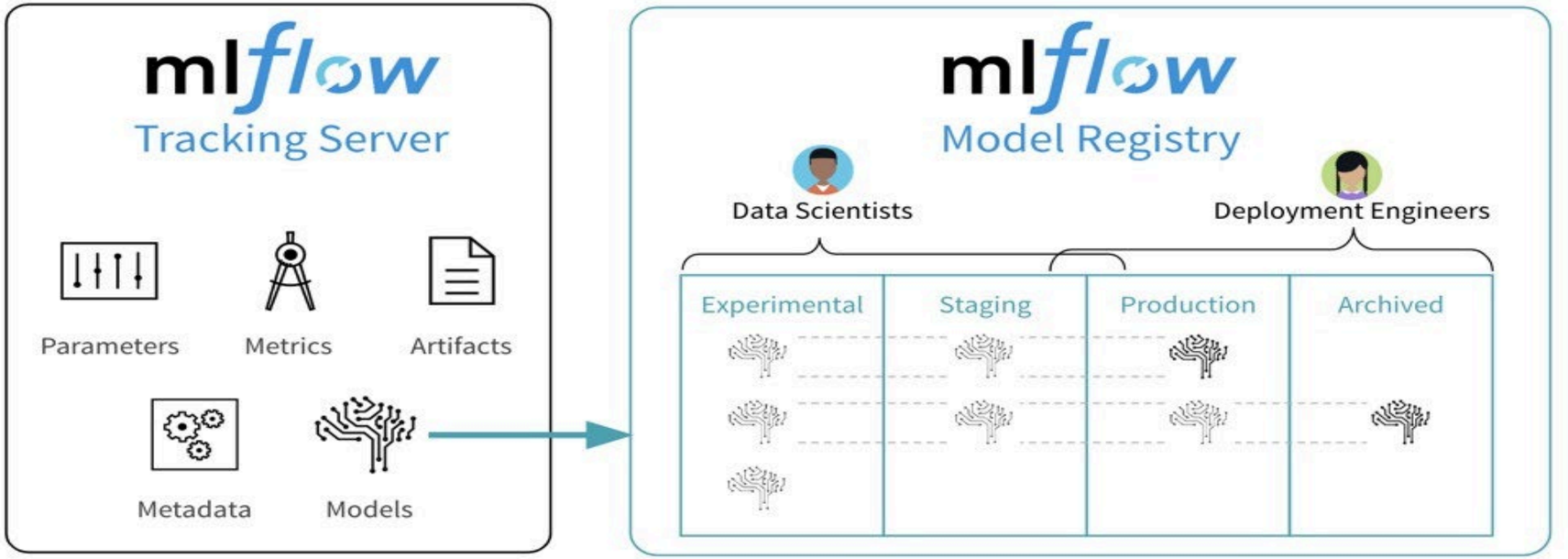
🔍 MLflow คืออะไร?

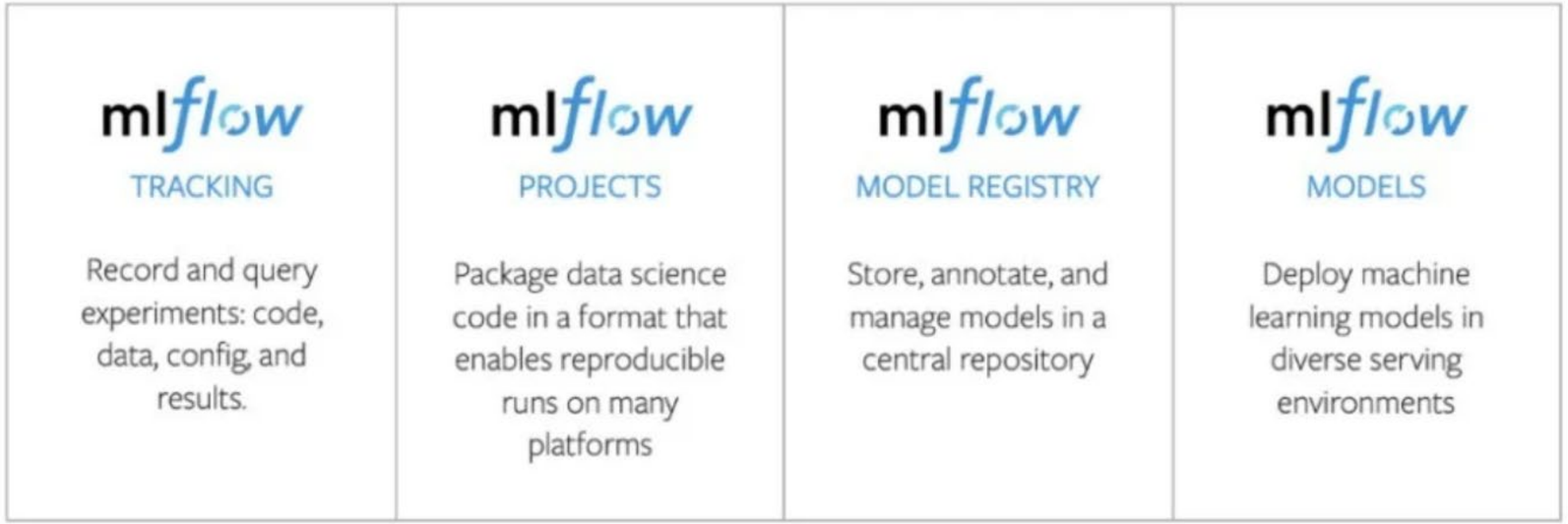
MLflow เป็น Open Source Platform สำหรับจัดการ Machine Learning Lifecycle ทั้งหมด พัฒนาโดย Databricks และเปิดตัวครั้งแรกในปี 2018 ปัจจุบันเป็นหนึ่งในเครื่องมือ MLOps ที่ได้รับความนิยมมากที่สุด โดยมีจุดเด่นคือความยืดหยุ่นสูง รองรับหลายภาษาโปรแกรม (Python, R, Java, REST API) และสามารถทำงานร่วมกับ ML Framework ยอดนิยมได้ทุกตัว

ปัญหาที่ MLflow แก้ไข

ในการพัฒนา Machine Learning แบบดั้งเดิม Data Scientists มักประสบปัญหาหลายประการ:

1. **Reproducibility Crisis**: ไม่สามารถทำซ้ำผลการทดลองได้เพราะลืมว่าใช้ hyperparameters อะไร ใช้ data version ไหน หรือใช้ code version ใด
2. **Experiment Chaos**: มี notebook หลายสิบไฟล์ที่ชื่อคล้ายกัน เช่น `model_v1.ipynb`, `model_v2_final.ipynb`, `model_v2_final_real.ipynb` จนไม่รู้ว่าอันไหนดีที่สุด
3. **Collaboration Difficulty**: ทีมงานไม่สามารถแชร์และเปรียบเทียบผลการทดลองได้อย่างมีประสิทธิภาพ
4. **Deployment Gap**: การนำ Model ไป Deploy ต้องเขียน code ใหม่เกือบทั้งหมด





องค์ประกอบหลัก 4 ส่วนของ MLflow Platform

MLflow ประกอบด้วย 4 Components ที่ทำงานร่วมกันครอบคลุม ML Lifecycle ทั้งหมด:

📋 **ตารางสรุป MLflow Components**

Component	Icon	Tagline	หน้าที่หลัก	ใช้เมื่อไหร่?
Tracking	📊	"บันทึกทุกการทดลอง"	บันทึก Parameters, Metrics, Artifacts	ตอน Train/Experiment
Projects	📦	"รันที่ไหนก็ได้ผลเหมือนกัน"	Package code ให้ reproduce ได้	ตอนแชร์ code กับทีม
Models	🤖	"Train ครั้งเดียว Deploy ได้ทุกที่"	Package model ให้ deploy ได้หลายที่	ตอน Deploy model
Registry	📋	"จัดการ Model เหมือน Git"	Version control สำหรับ models	ตอนจัดการ Production

สถานการณ์	Component ที่ใช้	เหตุผล
กำลังทดลอง hyperparameters หลายค่า	📊 Tracking	บันทึกและเปรียบเทียบผลได้
ต้องการให้เพื่อนร่วมทีมรัน code เดียวกัน	📦 Projects	มี dependencies ครบ รันได้ผลเหมือนกัน
จะนำ model ไป deploy บน server	🤖 Models	Package model ให้พร้อม serve
มี model หลาย versions ต้องจัดการ	📋 Registry	ติดตาม version, จัดการ Production
ทำทุกอย่างครบวงจร	ใช้ทั้ง 4	ครอบคลุม ML Lifecycle

💡 ตัวอย่าง Use Case จริง

ขั้นตอน	สิ่งที่ทำ	Component
1	Data Scientist ทดลอง model หลายแบบ บันทึก accuracy แต่ละแบบ	📊 Tracking
2	เลือก model ที่ดีที่สุด แพ็คโค้ดให้ทีม ML Engineer	📦 Projects
3	ML Engineer นำ model ไป deploy เป็น REST API	🤖 Models
4	เมื่อมี model ใหม่ที่ดีกว่า เปลี่ยน Production version	📋 Registry

MLflow Tracking

📊 รายละเอียด MLflow Tracking

📊 1. MLflow Tracking (เน้นใน Lab นี้)

> "บันทึกทุกการทดลอง เปรียบเทียบได้ทุกผลลัพธ์"

MLflow Tracking คือระบบบันทึกและติดตามการทดลอง Machine Learning ช่วยให้คุณไม่ต้องจำว่าใช้ hyperparameters อะไร ได้ผลเท่าไหร่

สิ่งที่บันทึกได้:

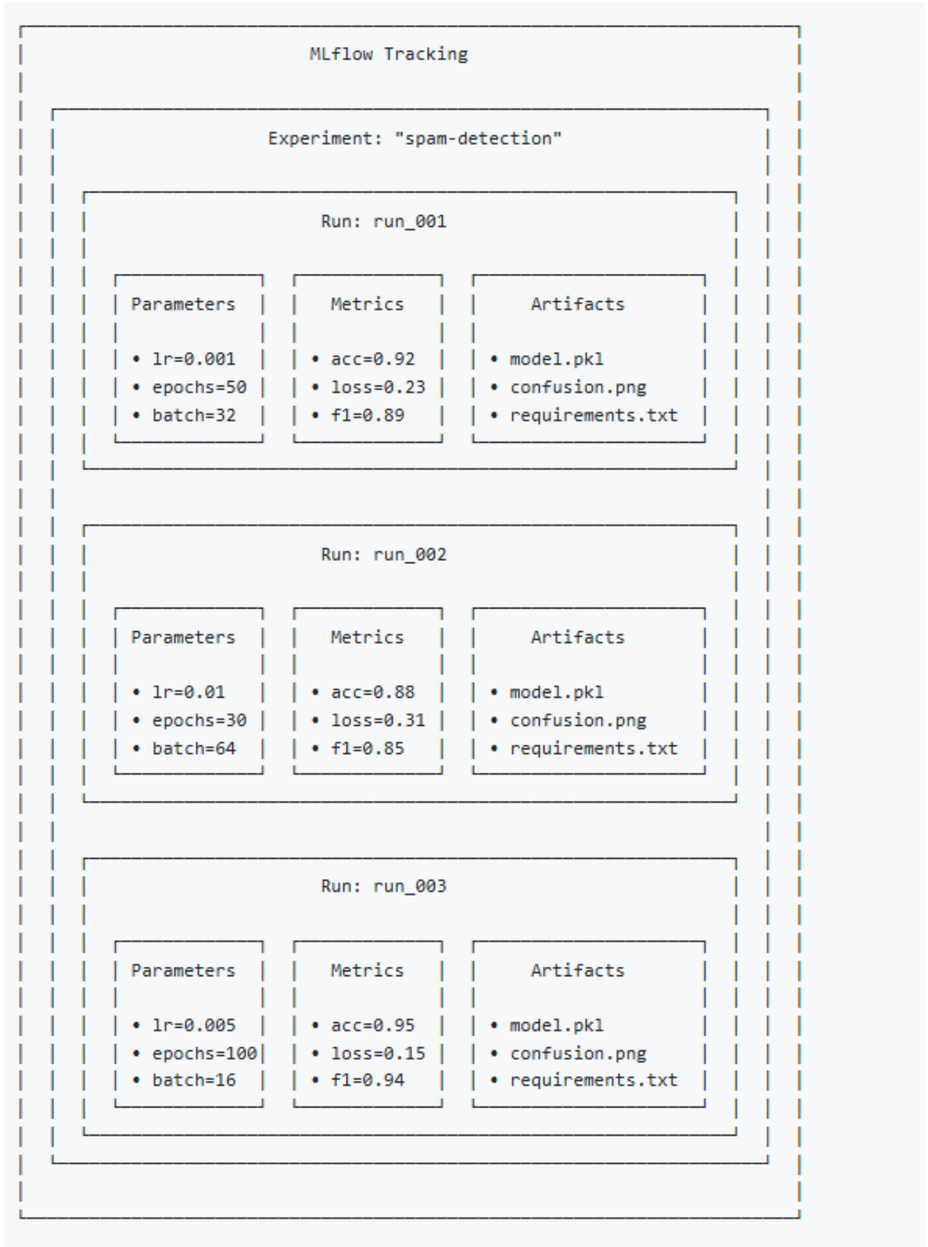
ประเภท	คำอธิบาย	ตัวอย่าง
Parameters	ค่าที่ตั้งก่อนทดลอง (Input)	`learning_rate=0.001`, `epochs=100`
Metrics	ค่าที่วัดได้จากการทดลอง (Output)	`accuracy=0.95`, `loss=0.05`
Artifacts	ไฟล์ที่สร้างจากการทดลอง	`model.pkl`, `confusion_matrix.png`
Tags	ข้อมูลเพิ่มเติมสำหรับจัดกลุ่ม	`developer=john`, `env=production`

ตัวอย่างการใช้งาน:

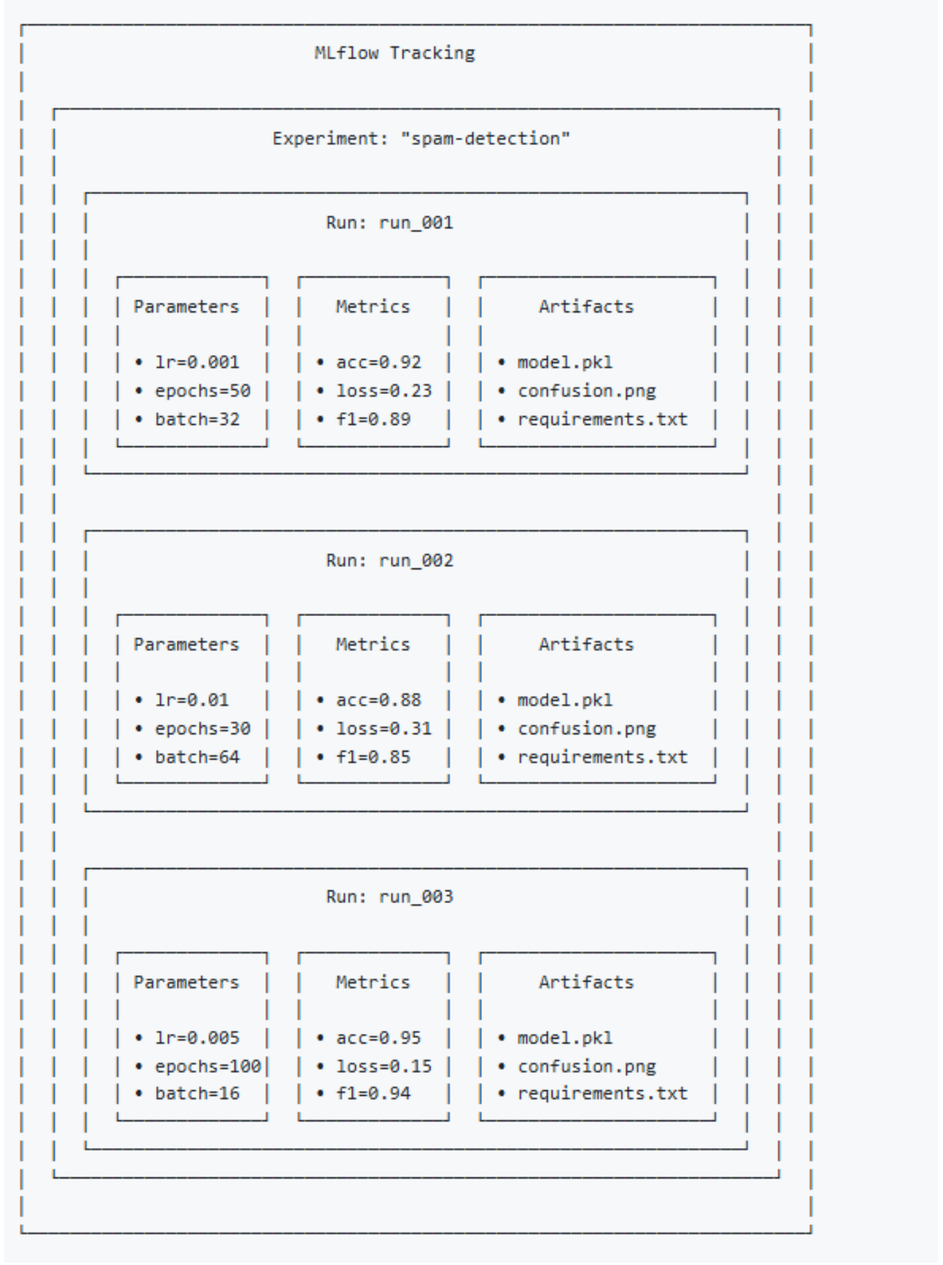
```
import mlflow

mlflow.set_experiment("my-experiment")

with mlflow.start_run():
    mlflow.log_param("learning_rate", 0.001)    # บันทึก Parameter
    mlflow.log_metric("accuracy", 0.95)         # บันทึก Metric
    mlflow.log_artifact("model.pkl")            # บันทึก Artifact
```



MLflow Tracking



Experiment: "spam-detection" - Summary Table

PARAMETERS

Run	Learning Rate	Epochs	Batch Size	Status
run_001	0.001	50	32	Baseline
run_002	0.01	30	64	Underfitting
run_003	0.005	100	16	⭐ Best Model

METRICS

Run	Accuracy	Loss	F1-Score	Improvement
run_001	0.92	0.23	0.89	-
run_002	0.88	0.31	0.85	-4.35%
run_003	0.95	0.15	0.94	+3.26%

ARTIFACTS

Run	Files Logged
run_001	model.pkl, confusion.png, requirements.txt
run_002	model.pkl, confusion.png, requirements.txt
run_003	model.pkl, confusion.png, requirements.txt

KEY INSIGHTS

📊 Best Accuracy: run_003 (0.95) ← +3.26% improvement over baseline
📉 Lowest Loss: run_003 (0.15) ← 34.8% reduction from baseline
🎯 Best F1-Score: run_003 (0.94) ← +5.6% improvement over baseline

💡 Insight: Smaller batch size (16) + more epochs (100) + moderate lr (0.005) yields the best performance for spam detection task

⚠️ Warning: run_002 shows underfitting due to insufficient epochs (30) and high learning rate (0.01)

Mlflow Project

📦 รายละเอียด MLflow Projects

📦 2. MLflow Projects

> "เขียนครั้งเดียว รันที่ไหนก็ได้ผลเหมือนกัน"

MLflow Projects คือ standard format สำหรับ packaging ML code ให้สามารถ reproduce ได้

องค์ประกอบหลัก:

```
my_project/
├── MLproject             # ไฟล์กำหนดค่า entry points
├── conda.yaml            # Dependencies (conda)
├── requirements.txt      # Dependencies (pip)
└── train.py              # โค้ดหลัก
```

ตัวอย่าง MLproject file:

```yaml
name: my_ml_project

conda_env: conda.yaml

entry_points:
  main:
    parameters:
      learning_rate: {type: float, default: 0.01}
      epochs: {type: int, default: 100}
    command: "python train.py --lr {learning_rate} --epochs {epochs}"
```

การรัน Project:

```
# รันจาก local
mlflow run ./my_project -P learning_rate=0.001

# รันจาก GitHub
mlflow run git@github.com:user/repo.git -P epochs=50
```

Mlflow Models

🤖 3. MLflow Models

> "Train ครั้งเดียว Deploy ได้ทุกที่"

MLflow Models คือ standard format สำหรับ packaging trained model ให้ deploy ได้หลาย platforms

Model Flavors ที่รองรับ:

Category	Flavors
Traditional ML	sklearn, xgboost, lightgbm, catboost
Deep Learning	pytorch, tensorflow, keras
Big Data	spark, h2o
Other	onnx, prophet, statsmodels, custom

Deployment Targets:

Target	คำอธิบาย
Local	รัน prediction บนเครื่อง
REST API	Deploy เป็น REST endpoint
Docker	Package เป็น Docker container
AWS SageMaker	Deploy บน Amazon SageMaker
Azure ML	Deploy บน Azure Machine Learning
Spark	รันบน Apache Spark cluster

ตัวอย่างการบันทึกและ Deploy Model:

```
# บันทึก Model
import mlflow.sklearn
mlflow.sklearn.log_model(model, "my_model")

# Deploy เป็น REST API
# mlflow models serve -m runs:/<run_id>/my_model -p 5000
```

Mlflow Registry

📋 รายละเอียด MLflow Model Registry

> "จัดการ Model เหมือนจัดการ Code ด้วย Git"

Model Registry คือ centralized store สำหรับจัดการ model versioning และ lifecycle เปรียบเทียบได้กับ Git ที่ใช้จัดการ source code

---

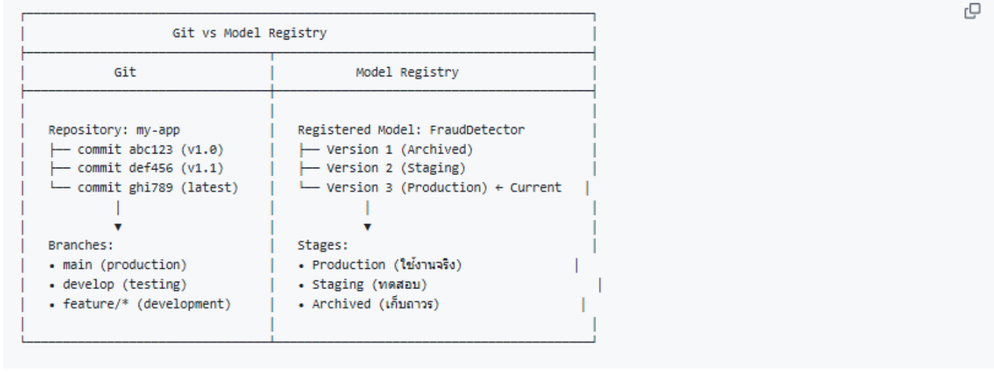
🤔 ทำไมต้องมี Model Registry?

เปรียบเทียบ: ไม่มี vs มี Model Registry

คำถามที่พบบ่อย	😰 ไม่มี Model Registry	😊 มี Model Registry
"Model ที่รันบน Production คือไฟล์ไหน?"	ไม่รู้... น่าจะเป็น `model_final_v2_new.pkl` ?	`FraudDetector` version 3, stage: Production ✅
"Model นี้ train ด้วย hyperparameters อะไร?"	จำไม่ได้... ลองหาใน notebook ดู	คลิกดู lineage → Run `abc123`, lr=0.001, epochs=100 ✅
"Model เมื่อวานดีกว่า rollback ได้ไหม?"	ไม่ได้... ลบไฟล์เก่าไปแล้ว 😭	ได้เลย! เปลี่ยน version 2 เป็น Production ✅
"ใครเป็นคน deploy model นี้? เมื่อไหร่?"	ไม่รู้... ไม่มี log	`john@company.com`, 2024-01-15 14:30:00 ✅
"Model version ไหนดีที่สุด?"	ต้องเปิด notebook เทียบเอง	ดูใน Registry UI เปรียบเทียบ metrics ได้เลย ✅
"ทีม QA จะทดสอบ model ใหม่ได้ยังไง?"	ส่งไฟล์ให้ทาง Slack/Email	เปลี่ยน stage เป็น Staging แล้วแจ้งทีม ✅

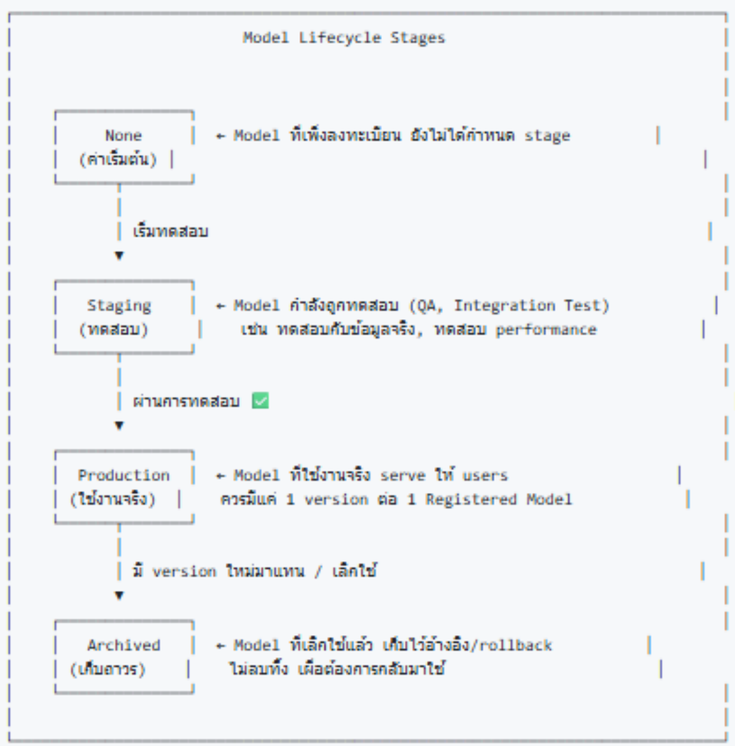
🔄 เปรียบเทียบ Model Registry กับ Git

แนวคิด	Git (Code)	Model Registry (Model)
Repository	Git Repository	Registered Model
Version	Commit	Model Version
Branch/Tag	main, develop, v1.0	Production, Staging, Archived
History	Commit History	Version History
Rollback	`git checkout <commit>`	Transition to previous version
Collaboration	Pull Request	Stage Transition + Approval
Metadata	Commit Message	Description, Tags, Lineage



🏷️ Model Stages อธิบายละเอียด

Model Registry มี 4 Stages สำหรับจัดการ lifecycle ของ model:



สรุป Stage แต่ละประเภท:

Stage	สัญลักษณ์	คำอธิบาย	ใครใช้?
None	🆕	เพิ่งลงทะเบียน ยังไม่พร้อม	Data Scientist (กำลังพัฒนา)
Staging	🧪	กำลังทดสอบ อาจมีหลาย versions	QA Team, ML Engineer
Production	🚀	ใช้งานจริง ควรมี 1 version	End Users, Applications
Archived	📦	เลิกใช้แล้ว เก็บไว้อ้างอิง	ไม่มีใครใช้ (backup)

📊 ตัวอย่าง Workflow จริง

สมมติทีมพัฒนา Fraud Detection Model:

Workflow อธิบาย:

```
วันที่ 1: John สร้าง Version 1 (Baseline)
    → ลงทะเบียนเป็น FraudDetector Version 1
    → ทดสอบผ่าน → เปลี่ยนเป็น Production

วันที่ 15: Jane สร้าง Version 2 (ดีกว่า)
    → ลงทะเบียนเป็น FraudDetector Version 2
    → ทดสอบใน Staging
    → ผ่านการทดสอบ → เปลี่ยนเป็น Production
    → Version 1 ถูกเปลี่ยนเป็น Archived

วันที่ 30: Jane สร้าง Version 3 (ดีที่สุด)
    → ลงทะเบียน → ทดสอบ → Production
    → Version 2 → Archived

วันที่ 45: John กำลังทดสอบ Version 4
    → อยู่ใน Staging รอทดสอบ
    → ถ้าผ่าน จะขึ้น Production แทน Version 3
```

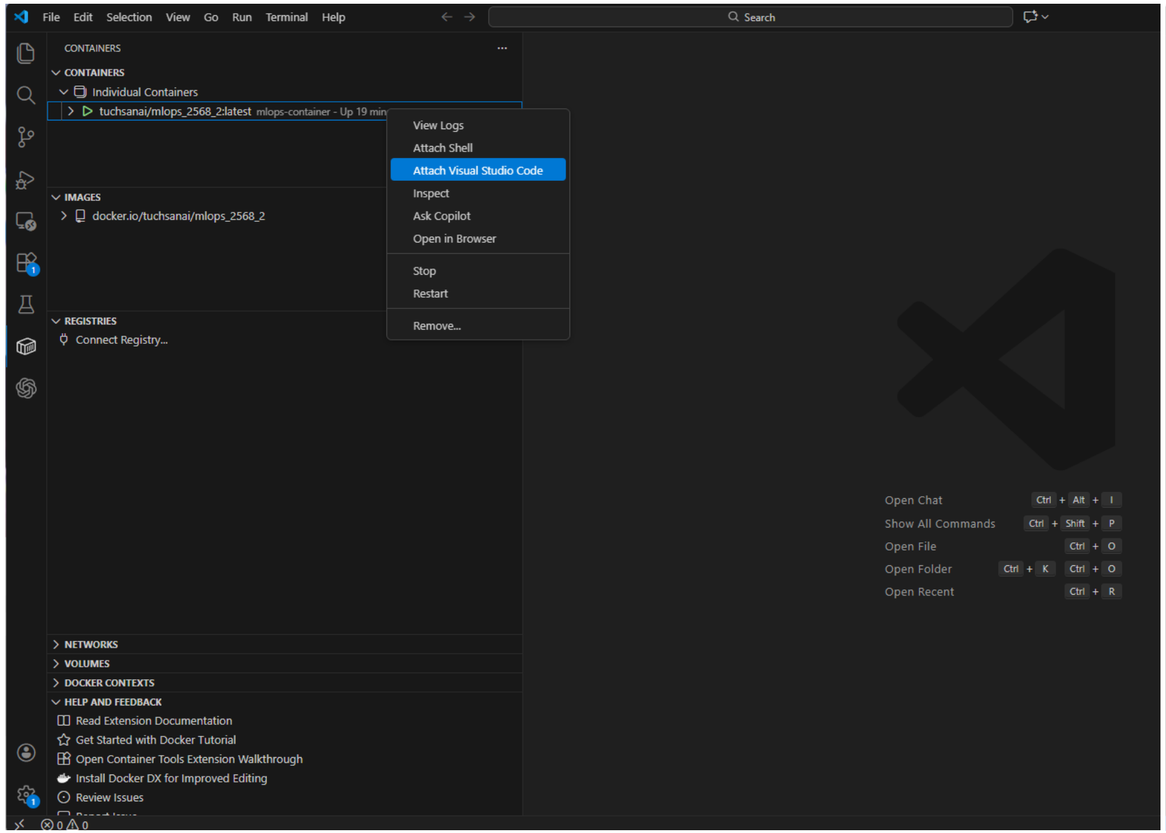
How to install and set up MLflow

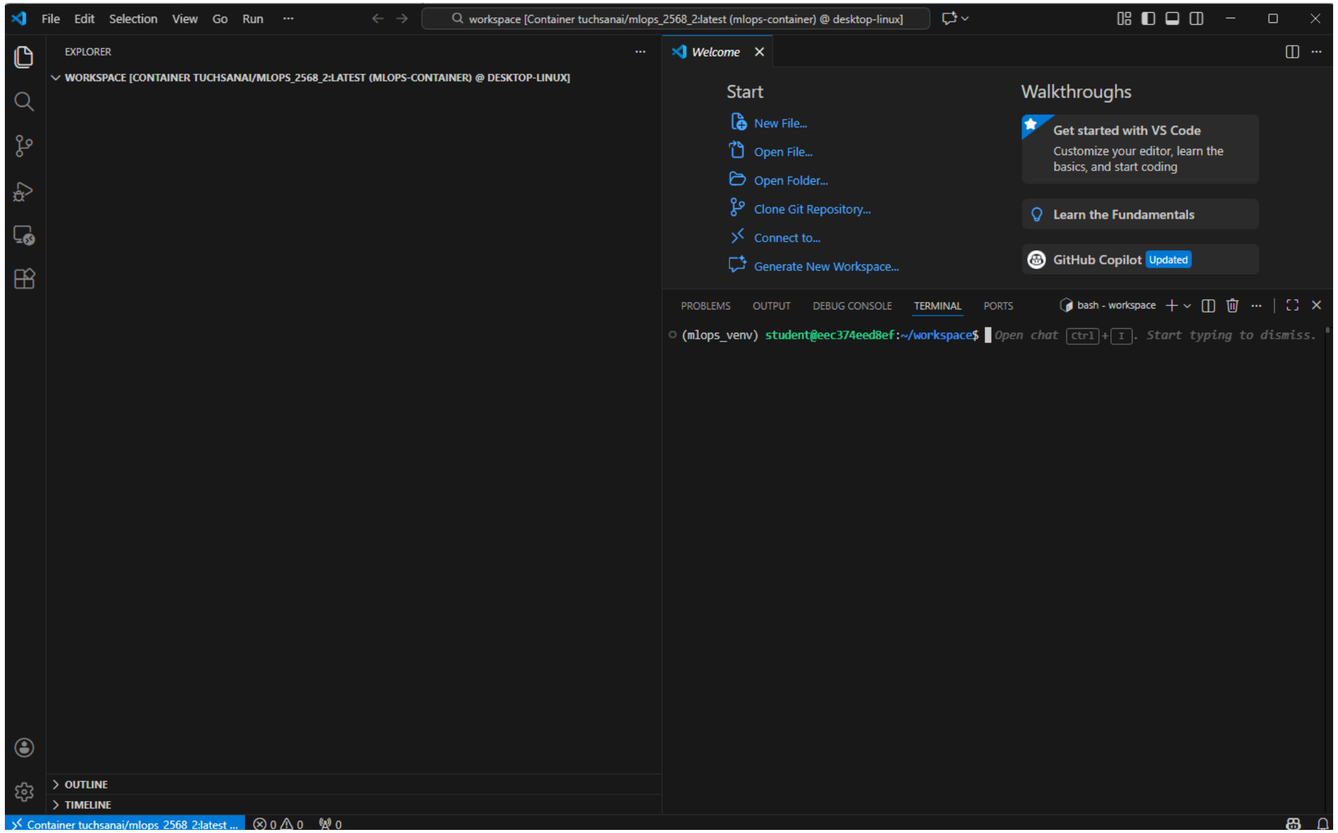
All LAB Run in Dev Container

การใช้งาน LAB ผ่าน Docker

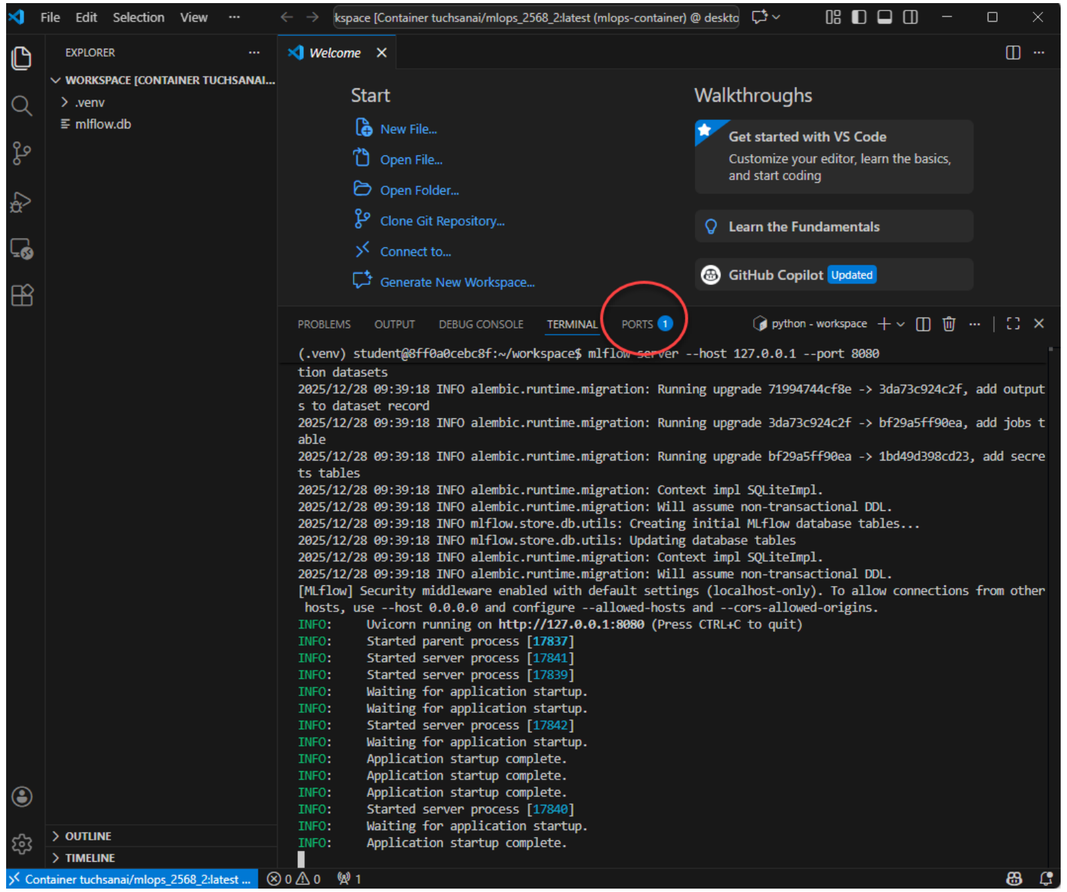
สำหรับผู้ที่ต้องการใช้งานผ่าน Docker Container ที่ติดตั้ง Library พื้นฐานไว้ครบถ้วนแล้ว สามารถรันคำสั่งดังนี้:

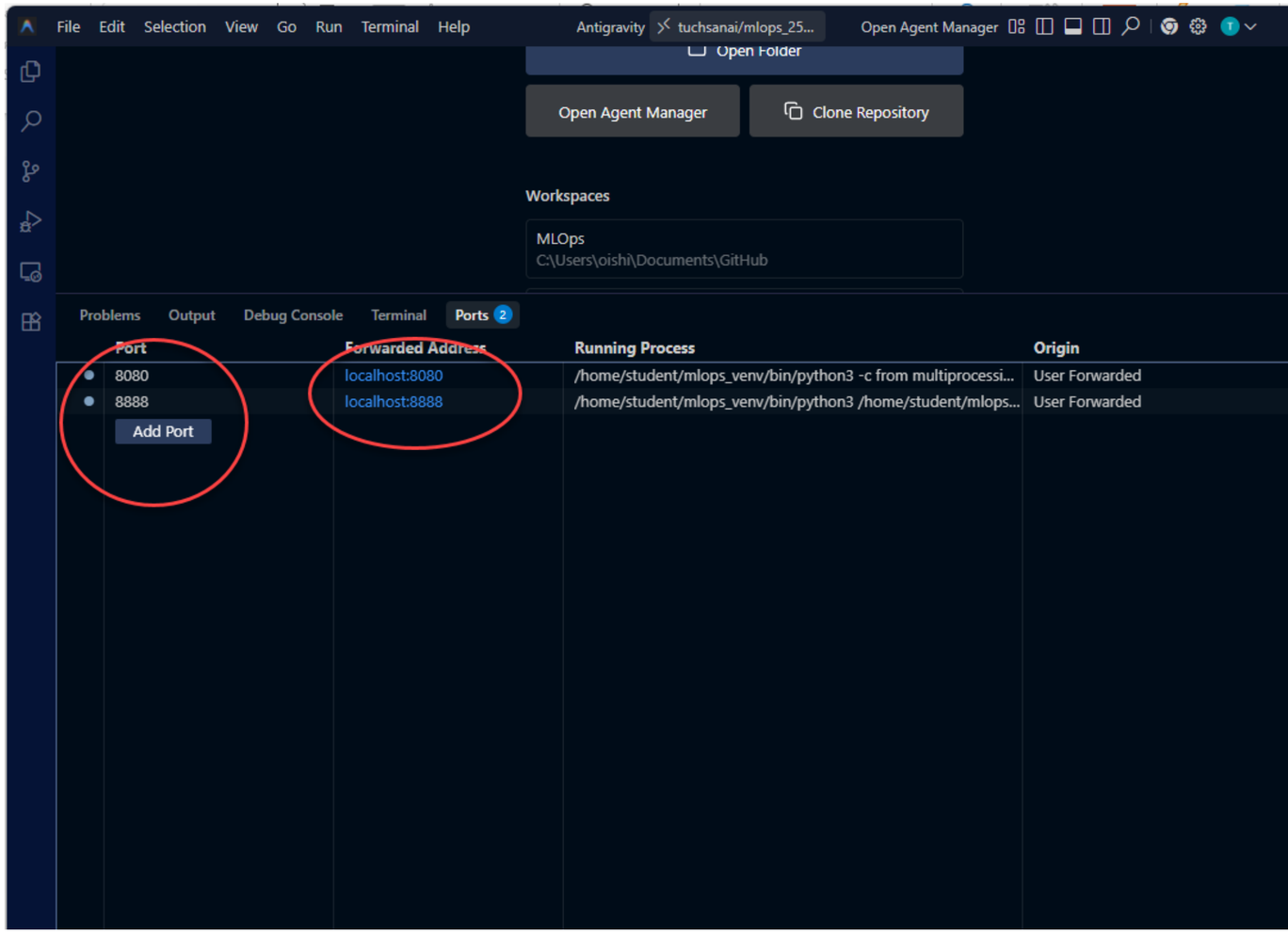
```
docker run -d -p 8080:8080 -p 8888:8888  --name mlops-container tuchsanai/mlops_2568_2:latest
```

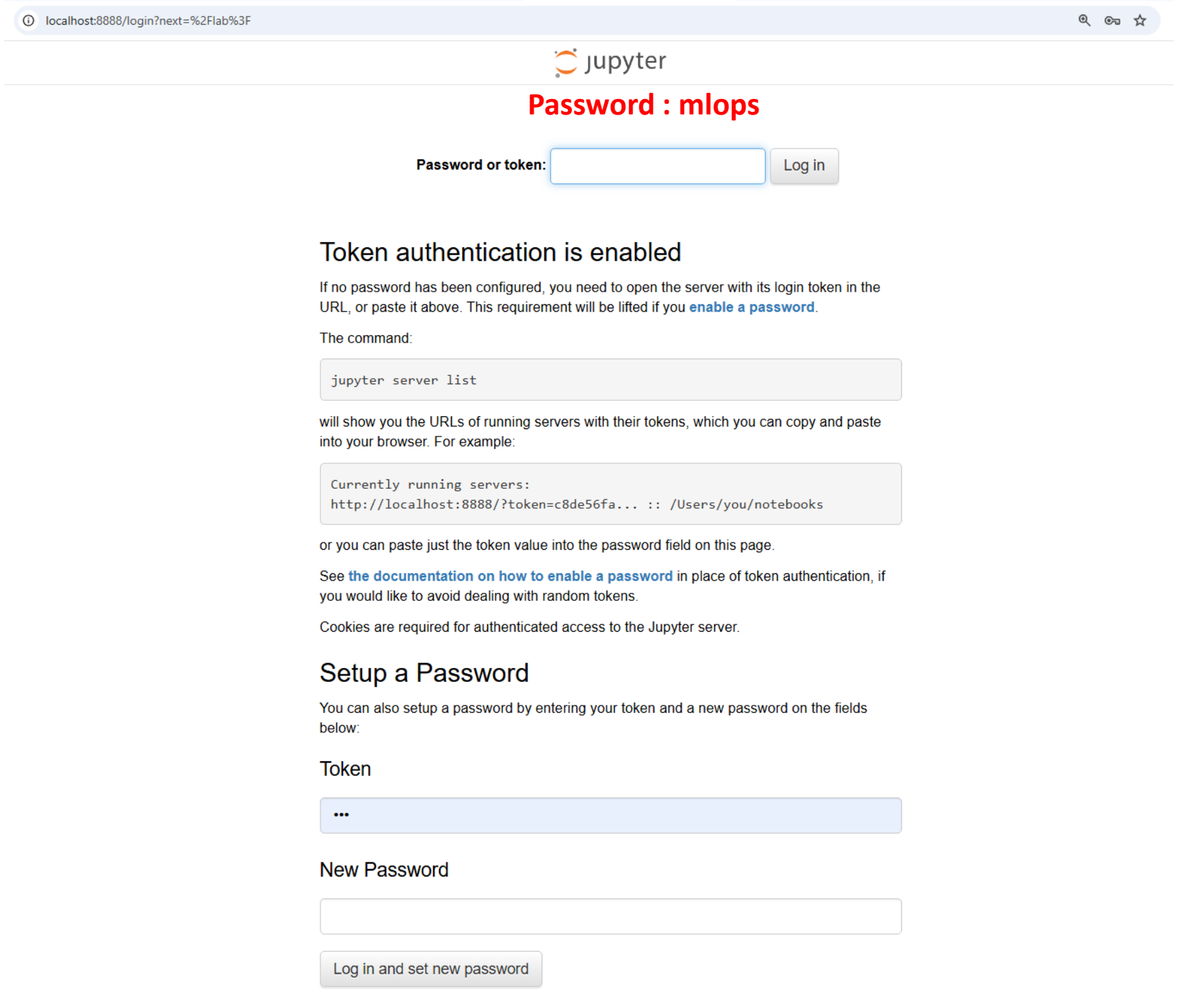


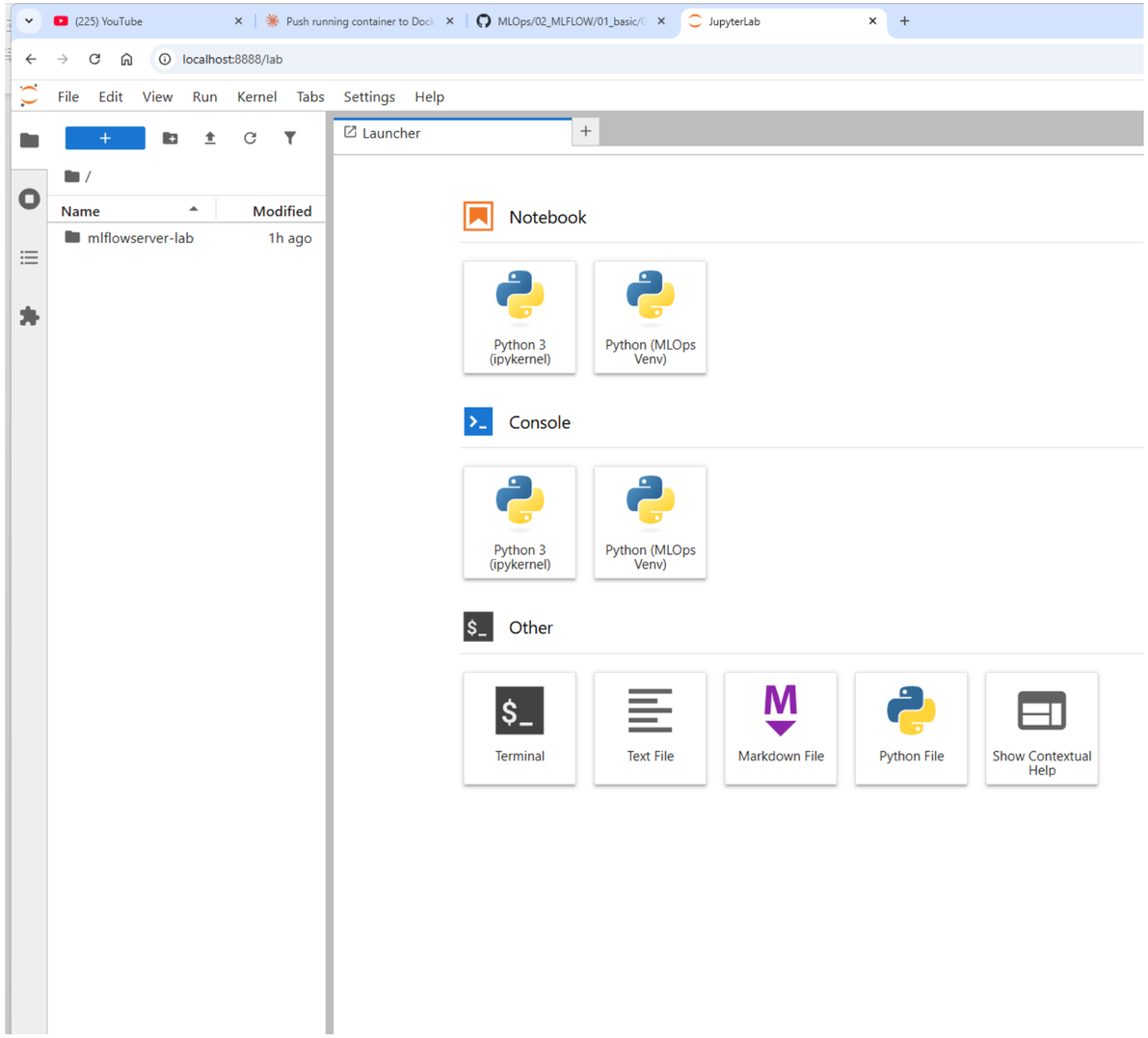


Add port 8080 สำหรับ mlflow
Add port 8888 สำหรับ jupyter






localhost:8888/login?next=%2Flab%3F
jupyter
Password : mlops
Password or token:
Log in
Token authentication is enabled
If no password has been configured, you need to open the server with its login token in the URL, or paste it above. This requirement will be lifted if you enable a password.
The command:
jupyter server list
will show you the URLs of running servers with their tokens, which you can copy and paste into your browser. For example:
Currently running servers:
http://localhost:8888/?token=c8de56fa... :: /Users/you/notebooks
or you can paste just the token value into the password field on this page.
See the documentation on how to enable a password in place of token authentication, if you would like to avoid dealing with random tokens.
Cookies are required for authenticated access to the Jupyter server.
Setup a Password
You can also setup a password by entering your token and a new password on the fields below:
Token
New Password
Log in and set new password


(225) YouTube
Push running container to Doc
MLOps/02_MLFLOW/01_basic/
JupyterLab
localhost:8888/lab
File Edit View Run Kernel Tabs Settings Help
/
Name
Modified
mlflowserver-lab
1h ago
Launcher
Notebook
Python 3 (ipykernel)
Python (MLOps Venv)
Console
Python 3 (ipykernel)
Python (MLOps Venv)
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help

🚀 ขั้นตอนการทำ Lab

ขั้นตอนที่ 1: สร้างโฟลเดอร์สำหรับ Lab

```
# สร้างโฟลเดอร์สำหรับเก็บไฟล์ Lab
mkdir -p mlflowserver-lab

# เข้าไปในโฟลเดอร์
cd mlflowserver-lab
```

ขั้นตอนที่ 1: สร้าง Virtual Environment

> 💡 **ทำไมต้องใช้ Virtual Environment?**
>
> Virtual Environment ช่วยแยก Package ของแต่ละโปรเจกต์ออกจากกัน ป้องกันปัญหา Package Version ขัดแย้งกัน

```
# 1.1 สร้าง Virtual Environment ชื่อ .venv
python -m venv .venv

# 1.2 เปิดใช้งาน Virtual Environment
source .venv/bin/activate

# 1.3 อัพเดท pip ให้เป็นเวอร์ชันล่าสุด
python -m pip install --upgrade pip
```

✅ **ตรวจสอบ:** เมื่อเปิดใช้งานสำเร็จ จะเห็น `(.venv)` นำหน้า prompt

ขั้นตอนที่ 2: ติดตั้ง Package ที่จำเป็น

```
pip install mlflow[extras] scikit-learn pandas numpy
```

รายละเอียด Package:

Package	หน้าที่
`mlflow[extras]`	MLflow เวอร์ชันสมบูรณ์ รวม Model Serving, SQL Storage และ Dependencies ทั้งหมด
`scikit-learn`	Library สำหรับ Machine Learning
`pandas`	จัดการข้อมูลแบบ DataFrame
`numpy`	คำนวณทางคณิตศาสตร์

📌 แบบที่ 1: In-Memory (สำหรับทดลองใช้งานเร็ว)

```
nohup mlflow server --host 0.0.0.0 --port 8080 > mlflow.log 2>&1 &
```

ข้อดี: เริ่มต้นได้เร็ว ไม่ต้องตั้งค่าอะไรเพิ่ม
ข้อเสีย: ข้อมูลหายเมื่อปิด Server

ผลลัพธ์ที่คาดหวัง:

```
INFO:     Started server process [28550]
INFO:     Application startup complete.
Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
```

📌 แบบที่ 2: Persistent Storage (แนะนำสำหรับใช้งานจริง) ⭐

```
# 3.1 สร้างโฟลเดอร์เก็บข้อมูล
mkdir -p mlruns_db mlartifacts

# 3.2 เปิด Server ให้เข้าถึงได้จากทุก IP
nohup mlflow server \
  --host 0.0.0.0 --port 8080 \
  --backend-store-uri sqlite:///mlruns_db/mlflow.db \
  --artifacts-destination ./mlartifacts \
  --serve-artifacts > mlflow.log 2>&1 &
```

🗂️ อธิบายโครงสร้างไฟล์:

```
โปรเจกต์ของคุณ/
├── mlruns_db/
│   └── mlflow.db          # ฐานข้อมูล SQLite เก็บ Experiment และ Run Metadata
├── mlartifacts/           # โฟลเดอร์เก็บ Model Files และ Artifacts
└── .venv/                 # Virtual Environment
```

ส่วนประกอบ	หน้าที่
Backend Store	เก็บข้อมูล Experiment, Run, Parameters, Metrics
Artifact Store	เก็บไฟล์โมเดล, กราฟ, ไฟล์ผลลัพธ์อื่นๆ

+ New

- Home
- Experiments
- Models
- Prompts

Welcome to MLflow

Information about UI telemetry
MLflow collects usage data to improve the product. To confirm your preferences, please visit the settings page in the navigation sidebar. To learn more about what data is collected, please visit the documentation.

Get started

Log traces
Trace LLM applications for debugging and monitoring.

Run evaluation
Iterate on quality with offline evaluations and comparisons.

Train models
Track experiments, parameters, and metrics throughout training.

Register prompts
Manage prompt updates and collaborate across teams.

Experiments

View all

Name	Time created	Last modified	Description	Tags
Default	12/28/2025, 04:39:18 PM	12/28/2025, 04:39:18 PM	-	

Discover new features

View all

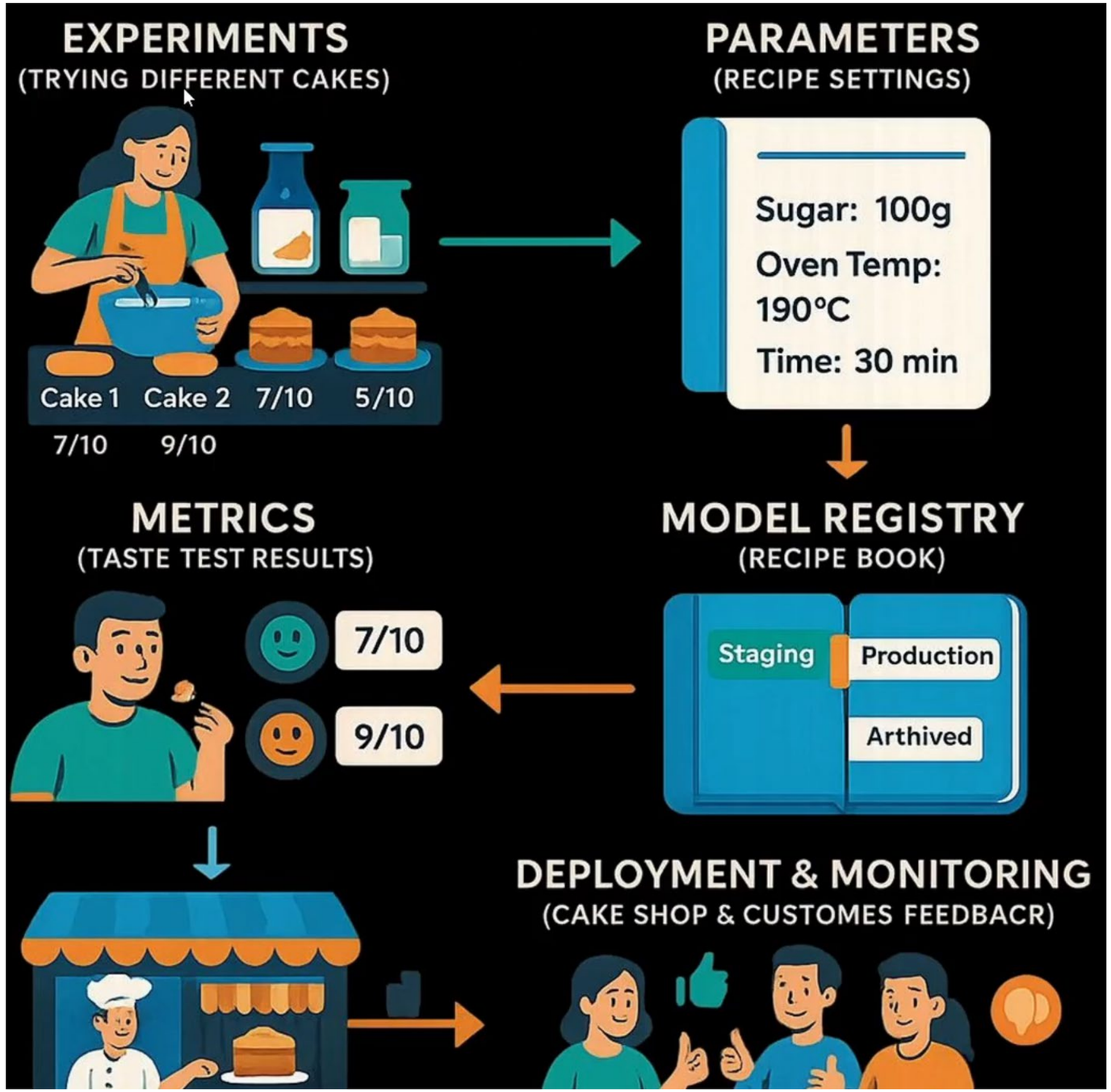
MLflow MCP server
Connect your coding assistants and AI applications to MLflow and automatically analyze your experiments and traces.

Optimize prompts
Access the state-of-the-art prompt optimization algorithms such as MIPROv2, GEPA, through MLflow Prompt Registry.

Agents-as-a-judge
Leverage agents as a judge to perform deep trace analysis and improve your evaluation accuracy.

Dataset tracking
Track dataset lineage and versions and effectively drive the quality improvement loop.

MLflow Tracking



องค์ประกอบหลัก 4 ส่วนของ MLflow Platform

MLflow ประกอบด้วย 4 Components ที่ทำงานร่วมกันครอบคลุม ML Lifecycle ทั้งหมด:

📋 ตารางสรุป MLflow Components

Component	Icon	Tagline	หน้าที่หลัก	ใช้เมื่อไหร่?
Tracking	📊	"บันทึกทุกการทดลอง"	บันทึก Parameters, Metrics, Artifacts	ตอน Train/Experiment
Projects	📦	"รันที่ไหนก็ได้ผลเหมือนกัน"	Package code ให้ reproduce ได้	ตอนแชร์ code กับทีม
Models	🤖	"Train ครั้งเดียว Deploy ได้ทุกที่"	Package model ให้ deploy ได้หลายที่	ตอน Deploy model
Registry	📋	"จัดการ Model เหมือน Git"	Version control สำหรับ models	ตอนจัดการ Production

MLflow Tracking คือระบบบันทึกและติดตามการทดลอง Machine Learning ช่วยให้คุณไม่ต้องจำว่าใช้ hyperparameters อะไร ได้ผลเท่าไหร่

สิ่งที่บันทึกได้:

ประเภท	คำอธิบาย	ตัวอย่าง
Parameters	ค่าที่ตั้งก่อนทดลอง (Input)	`learning_rate=0.001` , `epochs=100`
Metrics	ค่าที่วัดได้จากการทดลอง (Output)	`accuracy=0.95` , `loss=0.05`
Artifacts	ไฟล์ที่สร้างจากการทดลอง	`model.pkl` , `confusion_matrix.png`
Tags	ข้อมูลเพิ่มเติมสำหรับจัดกลุ่ม	`developer=john` , `env=production`

ตัวอย่างการใช้งาน:

```
import mlflow

mlflow.set_experiment("my-experiment")

with mlflow.start_run():
    mlflow.log_param("learning_rate", 0.001)    # บันทึก Parameter
    mlflow.log_metric("accuracy", 0.95)         # บันทึก Metric
    mlflow.log_artifact("model.pkl")            # บันทึก Artifact
```

🏗️ องค์ประกอบหลักของ MLflow Tracking

MLflow Tracking มีองค์ประกอบสำคัญ 5 ส่วนที่ทำงานร่วมกันเป็นระบบ:

1️⃣ Experiment (การทดลอง)

Experiment คือกลุ่มของ Runs ที่เกี่ยวข้องกัน เปรียบเสมือน "โปรเจกต์" หรือ "หัวข้อการทดลอง" ทุก Run ต้องอยู่ภายใต้ Experiment ใดเสมอ

หลักการตั้งชื่อ Experiment:

- ใช้ชื่อที่สื่อความหมายชัดเจน เช่น `fraud-detection-lstm`, `customer-churn-prediction`
- หลีกเลี่ยงชื่อทั่วไป เช่น `test`, `experiment1`
- ใช้ kebab-case หรือ snake_case เพื่อความสม่ำเสมอ

```
Experiment: "Image Classification"
├── Run 1: ทดลองด้วย ResNet50
├── Run 2: ทดลองด้วย VGG16
├── Run 3: ทดลองด้วย EfficientNet
└── Run 4: ทดลองด้วย MobileNet
```

ตัวอย่างการใช้งาน:

```python
import mlflow

# สร้างหรือเลือก Experiment
# ถ้า Experiment นี้มีอยู่แล้ว จะเลือกใช้อันเดิม
# ถ้ายังไม่มี จะสร้างใหม่อัตโนมัติ
mlflow.set_experiment("image-classification")

# ดู Experiment ID ที่กำลังใช้อยู่
experiment = mlflow.get_experiment_by_name("image-classification")
print(f"Experiment ID: {experiment.experiment_id}")
print(f"Artifact Location: {experiment.artifact_location}")
```

Experiment Properties:

Property	คำอธิบาย
`experiment_id`	Unique identifier (auto-generated)
`name`	ชื่อ Experiment ที่ตั้ง
`artifact_location`	ที่เก็บ Artifacts
`lifecycle_stage`	`active` หรือ `deleted`
`tags`	Metadata เพิ่มเติม

2️⃣ Run (การรันทดลอง)

Run คือการทดลองแต่ละครั้ง เป็นหน่วยพื้นฐานที่สุดของ MLflow Tracking ทุกครั้งที่ train model ด้วย hyperparameters ชุดใหม่ ควรสร้าง Run ใหม่

Run ประกอบด้วย:

- **Parameters**: ค่าที่ตั้งก่อนเริ่ม (input)
- **Metrics**: ค่าที่วัดได้ (output)
- **Artifacts**: ไฟล์ที่สร้างขึ้น
- **Metadata**: เวลาเริ่ม, เวลาจบ, สถานะ, Git commit hash ฯลฯ
- **Tags**: ข้อมูลเพิ่มเติมสำหรับการค้นหาและจัดกลุ่ม

รูปแบบการสร้าง Run:

```python
# แบบที่ 1: ใช้ Context Manager (แนะนำ)
# Run จะปิดอัตโนมัติเมื่อออกจาก with block
with mlflow.start_run(run_name="resnet50-experiment"):
    # โค้ดการทดลองอยู่ตรงนี้
    mlflow.log_param("learning_rate", 0.001)
    mlflow.log_metric("accuracy", 0.95)


# แบบที่ 2: Nested Runs (สำหรับ Hyperparameter Tuning)
with mlflow.start_run(run_name="parent-run"):
    for lr in [0.001, 0.01, 0.1]:
        with mlflow.start_run(run_name=f"child-lr-{lr}", nested=True):
            mlflow.log_param("learning_rate", lr)
```

3️⃣ Parameters (พารามิเตอร์)

Parameters คือค่าที่เราตั้งก่อนเริ่มการทดลอง (Input Configuration) **เป็นค่าคงที่ตลอดการทดลอง ไม่เปลี่ยนแปลง**

ประเภทของ Parameters:

ประเภท	ตัวอย่าง	คำอธิบาย
Model Hyperparameters	learning_rate, batch_size, epochs, dropout	ค่าที่ควบคุมพฤติกรรมการเรียนรู้ของ Model
Data Parameters	train_split, validation_split, image_size, augmentation	ค่าที่เกี่ยวกับการเตรียมข้อมูล
Architecture	model_type, num_layers, hidden_units, activation	โครงสร้างของ Model
Training Config	optimizer, loss_function, early_stopping_patience	การตั้งค่าการ Train
Environment	random_seed, gpu_id, num_workers	การตั้งค่าสภาพแวดล้อม

วิธีการบันทึก Parameters:

```
# วิธีที่ 1: บันทึกทีละค่า
mlflow.log_param("learning_rate", 0.001)
mlflow.log_param("batch_size", 32)
mlflow.log_param("epochs", 100)

# วิธีที่ 2: บันทึกหลายค่าพร้อมกัน (แนะนำ)
mlflow.log_params({
    "optimizer": "adam",
    "dropout": 0.5,
    "hidden_units": 256,
    "activation": "relu"
})

# วิธีที่ 3: บันทึกจาก argparse หรือ config dict
import argparse
args = argparse.Namespace(lr=0.001, batch=32)
mlflow.log_params(vars(args))
```

ข้อควรระวัง:

- Parameter value ต้องเป็น string หรือ number เท่านั้น
- ไม่สามารถแก้ไข Parameter หลังจากบันทึกแล้ว
- ชื่อ Parameter ควรสื่อความหมายและใช้รูปแบบเดียวกันตลอด

4️⃣ Metrics (เมตริก)

Metrics คือค่าที่วัดได้จากการทดลอง (Output/Results) **สามารถบันทึกได้หลายครั้งพร้อม** step number **เพื่อติดตามการเปลี่ยนแปลงตามเวลา**

ประเภทของ Metrics:

ประเภท	ตัวอย่าง	คำอธิบาย
Performance Metrics	accuracy, loss, f1_score, precision, recall, auc_roc	วัดประสิทธิภาพ Model
Training Metrics	train_loss, val_loss, train_acc, val_acc	วัดระหว่างการ Train
Time Metrics	training_time, inference_time, epoch_time	วัดเวลาที่ใช้
Resource Metrics	gpu_memory, cpu_usage	วัดการใช้ทรัพยากร
Custom Metrics	business_metric, custom_score	Metrics ที่กำหนดเอง

วิธีการบันทึก Metrics:

```
# วิธีที่ 1: บันทึก Metric ค่าเดียว (Final metric)
mlflow.log_metric("accuracy", 0.95)
mlflow.log_metric("f1_score", 0.92)

# วิธีที่ 2: บันทึก Metric ที่เปลี่ยนแปลงตามเวลา
# step parameter ใช้ระบุลำดับ (เช่น epoch number)
for epoch in range(100):
    train_loss, val_loss = train_one_epoch()
    mlflow.log_metric("train_loss", train_loss, step=epoch)
    mlflow.log_metric("val_loss", val_loss, step=epoch)

# วิธีที่ 3: บันทึกหลาย Metrics พร้อมกัน
mlflow.log_metrics({
    "precision": 0.89,
    "recall": 0.94,
    "f1": 0.91
}, step=epoch)

# วิธีที่ 4: บันทึก Metric พร้อม timestamp
mlflow.log_metric("accuracy", 0.95, timestamp=1234567890)
```

ความแตกต่างระหว่าง Parameters และ Metrics:

คุณสมบัติ	Parameters	Metrics
เวลาบันทึก	ก่อนเริ่มทดลอง	ระหว่าง/หลังทดลอง
จำนวนค่า	ค่าเดียวต่อชื่อ	หลายค่าได้ (ต่าง step)
การเปลี่ยนแปลง	คงที่	เปลี่ยนได้ตามเวลา
วัตถุประสงค์	บอกว่าทำอะไร (Input)	บอกว่าได้ผลอย่างไร (Output)
ตัวอย่าง	learning_rate=0.001	accuracy=0.95

5️⃣ Artifacts (สิ่งประดิษฐ์)

Artifacts คือไฟล์ที่สร้างจากการทดลอง สามารถเป็นไฟล์ประเภทใดก็ได้ MLflow จะเก็บไว้ใน Artifact Store

ประเภทของ Artifacts:

ประเภท	ตัวอย่าง	คำอธิบาย
Models	model.pkl, model.h5, model.pt, model.onnx	ไฟล์ Model ที่ Train เสร็จ
Visualizations	confusion_matrix.png, loss_curve.png, roc_curve.png	กราฟและภาพ
Data	predictions.csv, feature_importance.json, test_results.parquet	ข้อมูลผลลัพธ์
Code	train.py, config.yaml, requirements.txt	Source code และ config
Reports	report.html, summary.pdf	รายงานสรุป

วิธีการบันทึก Artifacts:

```python
import mlflow
import matplotlib.pyplot as plt
import os

# สร้างโฟลเดอร์สำหรับเก็บไฟล์ชั่วคราว
os.makedirs("outputs/models", exist_ok=True)
os.makedirs("outputs/plots", exist_ok=True)
os.makedirs("outputs/data", exist_ok=True)

with mlflow.start_run():

    # ----- วิธีที่ 1: บันทึกไฟล์เดียว -----
    # ไฟล์จะถูกเก็บที่ root ของ artifacts/
    mlflow.log_artifact("model.pkl")                    # → artifacts/model.pkl
    mlflow.log_artifact("confusion_matrix.png")         # → artifacts/confusion_matrix.png

    # ----- วิธีที่ 2: บันทึกไฟล์ไปยัง subdirectory -----
    # ใช้ artifact_path กำหนดโฟลเดอร์ปลายทาง
    mlflow.log_artifact("model.pkl", artifact_path="models")          # → artifacts/models/model.pkl
    mlflow.log_artifact("predictions.csv", artifact_path="data")      # → artifacts/data/predictions.csv

    # ----- วิธีที่ 3: บันทึกทั้งโฟลเดอร์ -----
    # ไฟล์ทั้งหมดในโฟลเดอร์จะถูกบันทึก
    mlflow.log_artifacts("./outputs/plots")                           # → artifacts/[ไฟล์ทั้งหมดใน plots]
    mlflow.log_artifacts("./outputs/plots", artifact_path="visualizations")  # → artifacts/visualizations/[ไฟล์]

    # ----- วิธีที่ 4: บันทึก dict เป็น JSON (MLflow 2.0+) -----
    config = {"learning_rate": 0.001, "batch_size": 32}
    mlflow.log_dict(config, artifact_file="config/hyperparameters.json")  # → artifacts/config/hyperparameters.json

    # ----- วิธีที่ 5: บันทึก text (MLflow 2.0+) -----
    mlflow.log_text("RandomForest v1.0", artifact_file="models/model_metadata.txt")  # → artifacts/models/model_metadata.txt

    # ----- วิธีที่ 6: บันทึก figure จาก matplotlib (MLflow 1.13+) -----
    fig, ax = plt.subplots()
    ax.plot([1, 2, 3], [1, 4, 9])
    ax.set_title("Loss Curve")
    mlflow.log_figure(fig, artifact_file="plots/loss_curve.png")      # → artifacts/plots/loss_curve.png
    plt.close(fig)  # ปิด figure หลังบันทึกเพื่อป้องกัน memory leak
```

Artifact Organization (ผลลัพธ์ที่ได้จาก Code ด้านบน):

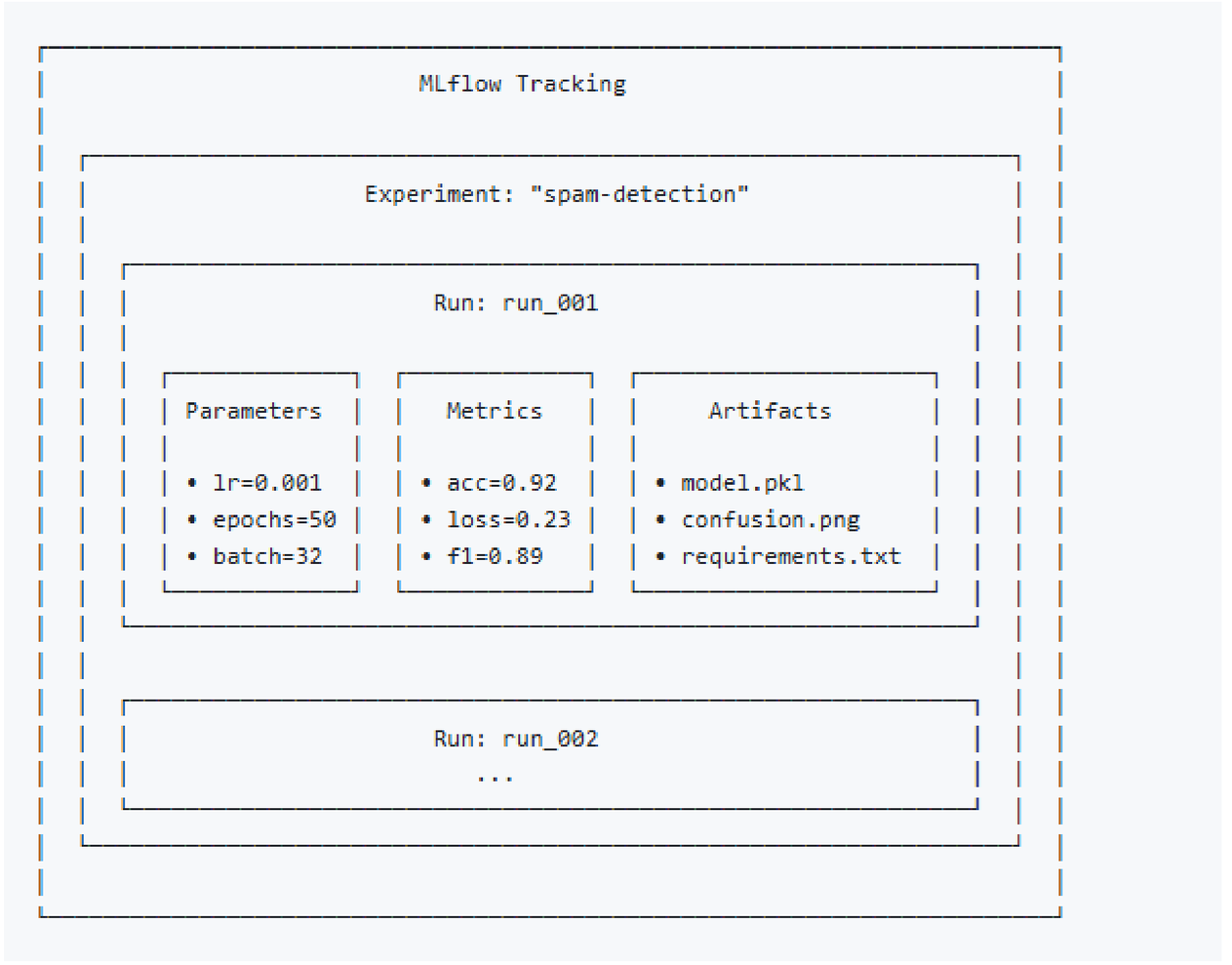
```
artifacts/
│
├── model.pkl                      ← วิธีที่ 1: log_artifact("model.pkl")
├── confusion_matrix.png           ← วิธีที่ 1: log_artifact("confusion_matrix.png")
│
├── models/
│   ├── model.pkl                  ← วิธีที่ 2: log_artifact("model.pkl", artifact_path="models")
│   └── model_metadata.txt         ← วิธีที่ 5: log_text(..., artifact_file="models/model_metadata.txt")
│
├── data/
│   └── predictions.csv            ← วิธีที่ 2: log_artifact("predictions.csv", artifact_path="data")
│
├── plots/
│   └── loss_curve.png             ← วิธีที่ 6: log_figure(fig, artifact_file="plots/loss_curve.png")
│
├── visualizations/
│   └── [ไฟล์จาก ./outputs/plots]  ← วิธีที่ 3: log_artifacts("./outputs/plots", artifact_path="visualizations")
│
└── config/
    └── hyperparameters.json       ← วิธีที่ 4: log_dict(config, artifact_file="config/hyperparameters.json")
```

สรุปความแตกต่างของแต่ละวิธี:

วิธี	Function	Input	Output Location
1	`log_artifact(local_path)`	ไฟล์ในเครื่อง	`artifacts/` (root)
2	`log_artifact(local_path, artifact_path)`	ไฟล์ในเครื่อง	`artifacts/{artifact_path}/`
3	`log_artifacts(local_dir)`	โฟลเดอร์ในเครื่อง	`artifacts/` หรือ `artifacts/{artifact_path}/`
4	`log_dict(dict, artifact_file)`	Python dict	`artifacts/{artifact_file}`
5	`log_text(text, artifact_file)`	Python string	`artifacts/{artifact_file}`
6	`log_figure(fig, artifact_file)`	Matplotlib figure	`artifacts/{artifact_file}`

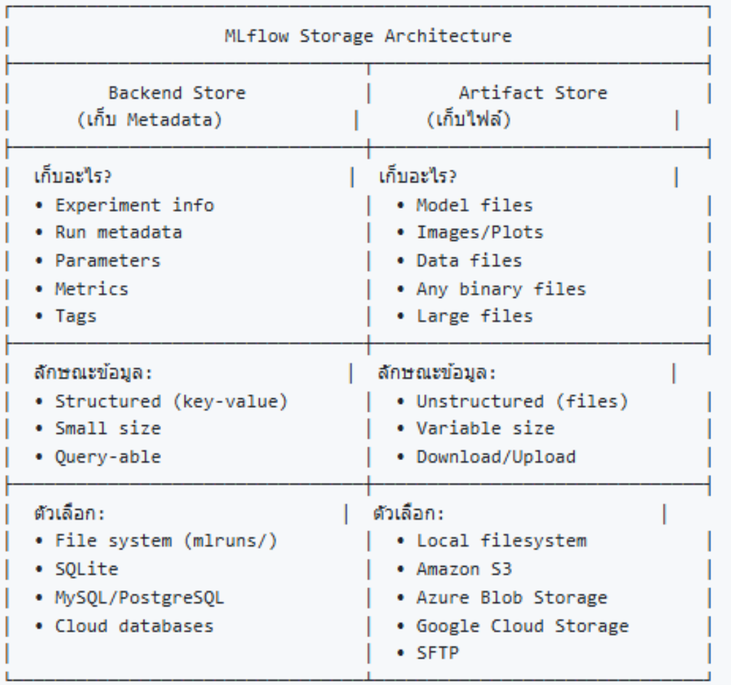
> 💡 **หมายเหตุ**: วิธีที่ 1-3 ใช้ `artifact_path` (ระบุโฟลเดอร์) ส่วนวิธีที่ 4-6 ใช้ `artifact_file` (ระบุ path เต็มรวมชื่อไฟล์)

📊 ภาพรวมความสัมพันธ์ขององค์ประกอบ



🗄️ Backend Store vs Artifact Store

MLflow แยกการเก็บข้อมูลออกเป็น 2 ส่วนที่ชัดเจน เพื่อประสิทธิภาพและความยืดหยุ่นในการ Scale:



ทำไมต้องแยก?

1. **Performance**: Metadata ถูก query บ่อย ต้องเร็ว ส่วนไฟล์ใหญ่เข้าถึงไม่บ่อย
2. **Scalability**: สามารถ scale แต่ละส่วนแยกกันตามความต้องการ
3. **Cost Optimization**: ใช้ storage ที่เหมาะสมกับประเภทข้อมูล
4. **Flexibility**: เลือก backend ได้ตามโครงสร้างพื้นฐานที่มี

ใน Lab นี้เราใช้:

- Backend Store: **SQLite** (ไฟล์ `mlflow.db`) - เหมาะสำหรับการเรียนรู้และทีมขนาดเล็ก
- Artifact Store: **Local filesystem** (โฟลเดอร์ `mlartifacts/`) - เก็บไฟล์ในเครื่อง

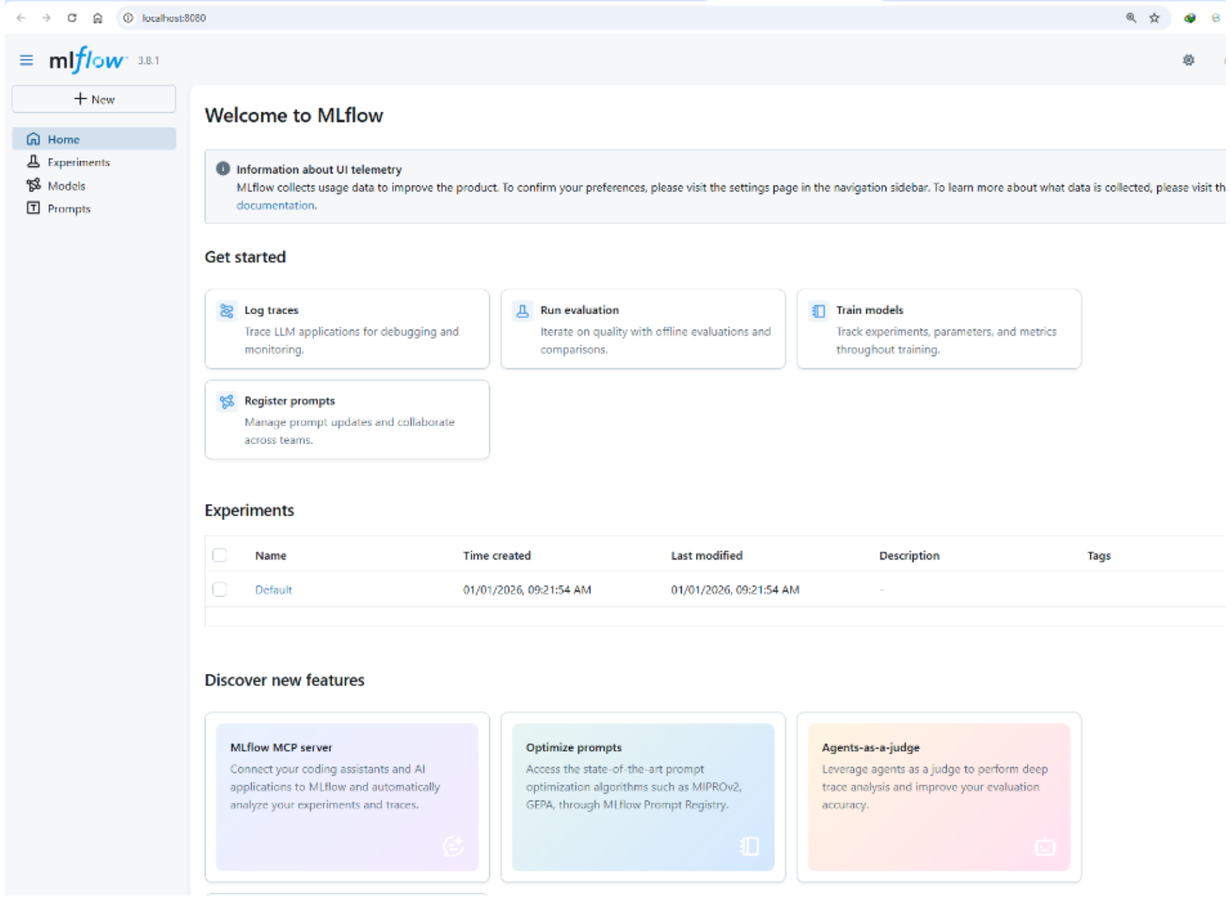
Pre-requisite: เตรียมความพร้อมก่อนเริ่ม Lab

สิ่งที่ต้องมี

ก่อนเริ่ม Lab นี้ ต้องมี MLflow Server รันอยู่และเข้าถึงได้ที่ `http://127.0.0.1:8080` ก่อนเริ่ม Lab นี้ ต้องมี jupyter notebook รันอยู่และเข้าถึงได้ที่ `http://127.0.0.1:8888`

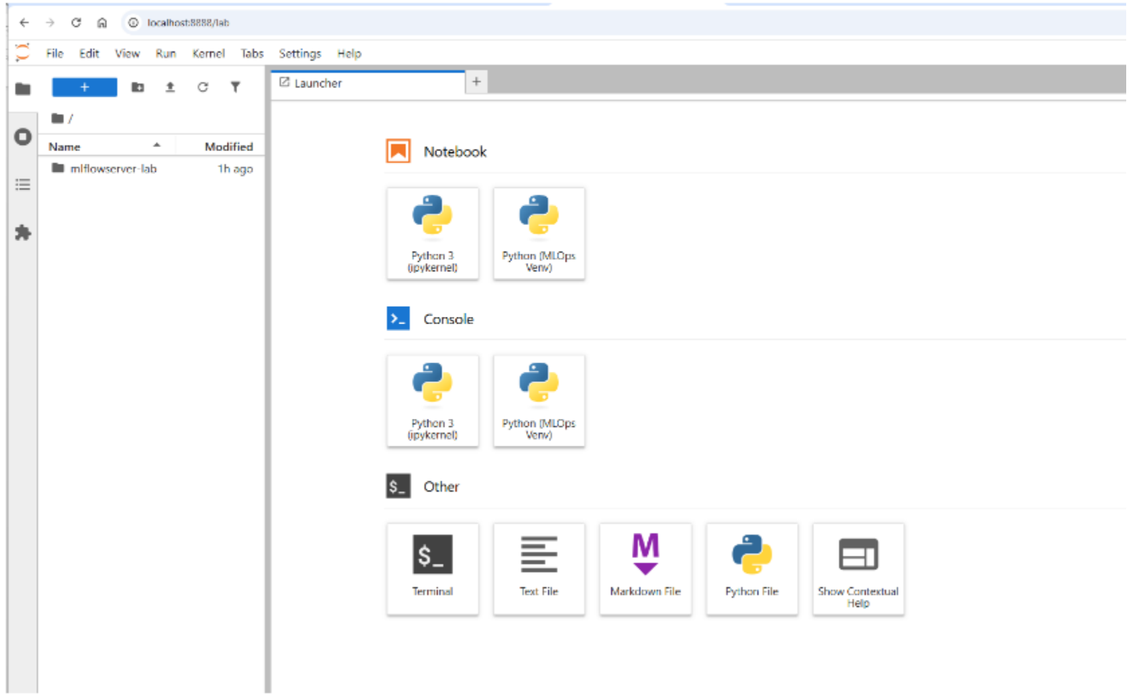
ตรวจสอบ MLflow Server

เปิด Browser แล้วไปที่: http://127.0.0.1:8080



ตรวจสอบ Jupyter Server

เปิด Browser แล้วไปที่: http://127.0.0.1:8888



📚 ส่วนที่ 1: การเชื่อมต่อ MLflow Server

แนวคิด

ก่อนใช้งาน MLflow Tracking ต้องกำหนด **Tracking URI** เพื่อบอกว่าจะเก็บข้อมูลการทดลองไว้ที่ไหน

ฟังก์ชันสำคัญ

ฟังก์ชัน	คำอธิบาย
`mlflow.set_tracking_uri(uri)`	กำหนด URL ของ MLflow Server
`mlflow.get_tracking_uri()`	ตรวจสอบ URL ที่กำหนดไว้

ทางเลือกอื่นสำหรับ Tracking URI

ประเภท	ตัวอย่าง URI	คำอธิบาย
Local File	`file:///path/to/mlruns`	เก็บในโฟลเดอร์ local
Remote Server	`http://server:5000`	เก็บบน MLflow Server
Databricks	`databricks`	เก็บบน Databricks
SQLite	`sqlite:///mlflow.db`	เก็บใน SQLite database

[1]:
```
import mlflow
import os

# กำหนด Tracking URI ไปยัง MLflow Server
mlflow.set_tracking_uri("http://127.0.0.1:8080")

# ตรวจสอบว่าเชื่อมต่อถูกต้อง
print(f"✅ MLflow Tracking URI: {mlflow.get_tracking_uri()}")
```

```
✅ MLflow Tracking URI: http://127.0.0.1:8080
```