give me end to end python example to deploy a ML model to vertex AI and test it

Here's a modern (2025–2026 era) **end-to-end Python example** showing how to:

- Train a small model (scikit-learn or XGBoost)
- Save it in a way Vertex AI likes
- Create a custom container (fastest & cleanest way in 2025+)
- Deploy to Vertex AI Prediction
- Send online predictions

Python

```python
# end-to-end-vertex-ai-deploy-2025.py
"""
End-to-end example: train → save → build container → deploy → predict
Vertex AI custom container approach (recommended in 2025+)

Requirements:
    pip install google-cloud-aiplatform google-cloud-storage scikit-learn xgb
    gcloud auth application-default login
"""

import os
import json
from pathlib import Path
import pickle
from datetime import datetime

import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import accuracy_score

from google.cloud import aiplatform
from google.cloud.aiplatform.gapic.schema import predict

# ─── 1. Config ──────────────────────────────────────────────────────────

PROJECT_ID = "your-project-id"           # ← change
LOCATION = "us-central1"                 # or europe-west4, etc.
BUCKET_NAME = f"{PROJECT_ID}-vertex-models"
TIMESTAMP = datetime.now().strftime("%Y%m%d_%H%M%S")
```

```
MODEL_DISPLAY_NAME = f"fraud-xgb-{TIMESTAMP}"
ENDPOINT_DISPLAY_NAME = f"fraud-endpoint-{TIMESTAMP}"

# Artifact paths
ARTIFACT_URI = f"gs://{BUCKET_NAME}/{MODEL_DISPLAY_NAME}"
CONTAINER_IMAGE = "europe-west4-docker.pkg.dev/your-project-id/vertex-predict

# ── 2. Train & save a simple model ──────────────────────────────────────

X, y = make_classification(
    n_samples=50_000, n_features=20, n_informative=15,
    random_state=42, class_sep=1.2
)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

model = HistGradientBoostingClassifier(max_iter=200, random_state=42)
model.fit(X_train, y_train)

print("Train accuracy:", accuracy_score(y_train, model.predict(X_train)))
print("Test  accuracy:", accuracy_score(y_test,  model.predict(X_test)))

# Save model + preprocessor info
model_dir = Path("artifacts/model")
model_dir.mkdir(parents=True, exist_ok=True)

with open(model_dir / "model.pkl", "wb") as f:
    pickle.dump(model, f)

# Minimal metadata (Vertex AI likes to see this sometimes)
metadata = {"framework": "scikit-learn", "version": "1.5", "trained": TIMESTA
with open(model_dir / "metadata.json", "w") as f:
    json.dump(metadata, f, indent=2)

# ── 3. Upload model artifacts to GCS ────────────────────────────────────

!gsutil -m cp -r artifacts/model/* gs://{BUCKET_NAME}/{MODEL_DISPLAY_NAME}/

# ── 4. Create FastAPI serving application (prediction server) ──────────

fastapi_code = """\
# app/main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import pickle
import numpy as np
from typing import List

app = FastAPI(title="Vertex AI Custom Prediction")

with open("/model/model.pkl", "rb") as f:
    model = pickle.load(f)
```

```
class PredictRequest(BaseModel):
    instances: List[List[float]]

class PredictResponse(BaseModel):
    predictions: List[int]
    probabilities: List[List[float]]

@app.post("/predict", response_model=PredictResponse)
async def predict(body: PredictRequest):
    try:
        X = np.array(body.instances, dtype=np.float32)
        if X.ndim != 2:
            raise ValueError("Expected 2D array")

        proba = model.predict_proba(X).tolist()
        preds = model.predict(X).tolist()

        return {"predictions": preds, "probabilities": proba}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

@app.get("/health")
async def health():
    return {"status": "healthy"}
"""

with open("app/main.py", "w") as f:
    f.write(fastapi_code)

# Minimal requirements
with open("app/requirements.txt", "w") as f:
    f.write("""\
fastapi==0.115.0
uvicorn==0.30.6
numpy==1.26.4
scikit-learn==1.5.2
""")

# Very simple Dockerfile
dockerfile = f"""\
FROM python:3.11-slim

WORKDIR /app

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

COPY main.py .

COPY --from=local /artifacts/model /model
```

```
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
"""

with open("Dockerfile", "w") as f:
    f.write(dockerfile)

# ── 5. Build & push container (run these commands manually or via cloudbuil
"""
# Option A - local build (needs docker installed)
docker build -t {CONTAINER_IMAGE} .

# Option B - Cloud Build (recommended for production)
gcloud builds submit --tag {CONTAINER_IMAGE} \
  --project {PROJECT_ID} \
  --timeout=20m
"""

print("\nBuild & push the container with one of the commands above ↑\n")

# ── 6. Upload model to Vertex AI Model Registry ──────────────────────────

aiplatform.init(project=PROJECT_ID, location=LOCATION, staging_bucket=ARTIFAC

model = aiplatform.Model.upload(
    display_name=MODEL_DISPLAY_NAME,
    artifact_uri=ARTIFACT_URI,
    serving_container_image_uri=CONTAINER_IMAGE,
    serving_container_environment_variables={
        "GUNICORN_CMD_ARGS": "--timeout 120"
    },
    serving_container_ports=[8080],
    description="Fraud detection model - HistGradientBoosting"
)

print(f"Model uploaded: {model.resource_name}")

# ── 7. Deploy to an endpoint ─────────────────────────────────────────────

endpoint = model.deploy(
    deployed_model_display_name=f"deployed-{TIMESTAMP}",
    machine_type="n1-standard-4",
    min_replica_count=1,
    max_replica_count=2,
    traffic_percentage=100,
)

print(f"Endpoint created: {endpoint.resource_name}")

# ── 8. Test prediction ──────────────────────────────────────────────────

test_instances = [
    X_test[0].tolist(),
```

```
    X_test[1].tolist(),
    X_test[42].tolist(),
]

prediction = endpoint.predict(instances=test_instances)

print("\nPrediction result:")
print(json.dumps(prediction.predictions, indent=2))

# Or lower-level way (good for debugging)
instances_proto = [
    {"content": json.dumps(inst)} for inst in test_instances
]

response = endpoint.raw_predict(
    http_body=predict.HttpBody(
        data=json.dumps({"instances": test_instances}).encode("utf-8"),
        content_type="application/json"
    )
)

print("\nRaw response:")
print(response.data.decode("utf-8"))
```

Quick modern alternatives / notes (2025–2026)

Approach	Build time	Cold start	Flexibility	Recommendation 2025+
scikit-learn + sklearnserver	~3 min	slow	low	legacy
XGBoost + custom handler	~5 min	medium	medium	still ok
FastAPI / TorchServe / vLLM	8–15 min	fast	high	**preferred**
Model Garden / Publisher model	—	very fast	low	when you can

Most recommended path today → **custom FastAPI container** (like the example above)

Good luck with the deployment!
Let me know if you want the torch / lightgbm / private endpoint / batch prediction / explainable AI version too.