Milestone 1: Submission Checklist

MLOps Course - Module 2

Pre-Submission Verification

Use this checklist to verify your submission before the deadline. Complete each section to ensure you meet all requirements.

Deliverables Checklist

FastAPI Service (Local)

- ☐ `main.py` exists with FastAPI app exposing `/predict` endpoint
- ☐ Pydantic request model defined for input validation
- ☐ Pydantic response model defined for output schema
- ☐ Model artifact (`model.pkl` or similar) included in repository
- ☐ Model loads deterministically at startup (not per-request)
- ☐ `requirements.txt` OR `pyproject.toml` with exact version pinning
- ☐ README describes lifecycle position (input → model → API → consumer)

Cloud Run Deployment

- ☐ Cloud Run service URL is publicly accessible with HTTPS
- ☐ GCP Artifact Registry image reference documented
- ☐ Evidence of successful inference (screenshot or curl output)
- ☐ Cold start behavior analysis included in documentation

Serverless Function (GCP Cloud Functions)

- ☐ Cloud Function code implementing prediction logic
- ☐ Deployment configuration documented
- ☐ Deployment logs captured (screenshot or text)
- ☐ Function invocation tested and working

Comparative Report

- ☐ FastAPI container vs Cloud Function comparison included
- ☐ Lifecycle differences (stateful vs stateless) explained
- ☐ Artifact loading strategies compared
- ☐ Latency characteristics documented (cold starts, warm instances)
- ☐ Reproducibility considerations discussed

Documentation

- ☐ README includes setup and deployment instructions
- ☐ API usage examples with sample requests/responses
- ☐ Lifecycle stage explanations present
- ☐ Model-API interaction clearly described
- ☐ Deployment URLs included and accessible

Rubric Evidence Map

Use this table to verify you have evidence for each graded criterion:

Criterion	Points	Evidence Location	Verified
Correct API implementation with Pydantic schemas	2	`main.py` - endpoint and model definitions	[ ]
Deterministic artifact loading	2	`main.py` - startup loading logic	[ ]
Reproducible environment	2	`requirements.txt` or `pyproject.toml`	[ ]
Successful HTTPS deployment on Cloud Run	2	Cloud Run URL + screenshot/curl output	[ ]
Proper registry workflow	2	Artifact Registry image reference	[ ]
Working Cloud Function inference	2	Function URL + invocation evidence	[ ]
Clear deployment stage explanation	2	README - Lifecycle section	[ ]
Artifact management documentation	2	README - Model-API interaction section	[ ]
Latency/cold start comparison	1	Comparative report section	[ ]
Statelessness/reproducibility comparison	1	Comparative report section	[ ]

Criterion	Points	Evidence Location	Verified
Clear instructions and organized code	1	Overall README and repository structure	[ ]

Common Pitfalls

** Warning:** Hardcoding credentials or API keys

Never commit secrets to your repository. Use environment variables and GCP IAM:

```
# Bad
API_KEY = "sk-abc123..."

# Good
API_KEY = os.environ.get("API_KEY")
```

** Warning:** Model artifact not versioned or reproducibly loadable

Your model must load identically every time. Verify with:

```
# Ensure deterministic loading
model = joblib.load("model.pkl")
assert model.predict([[1, 2, 3]]) == model.predict([[1, 2, 3]])
```

** Warning:** Only testing warm instance latency

Cold starts behave differently. Test both:

```
# Wait 15+ minutes, then test cold start
curl -w "\nTime: %{time_total}s\n" $CLOUD_RUN_URL/predict
# Immediately test warm instance
curl -w "\nTime: %{time_total}s\n" $CLOUD_RUN_URL/predict
```

** Warning:** Loading model on every request instead of at startup

Incorrect (slow):

```
@app.post("/predict")
def predict(data: InputModel):
    model = joblib.load("model.pkl")  # Loaded every request!
    return model.predict(...)
```

Correct (fast):

```
model = joblib.load("model.pkl")  # Loaded once at startup

@app.post("/predict")
```

```
def predict(data: InputModel):
    return model.predict(...)
```

** Warning:** Cloud Run service not publicly accessible

Ensure you've configured public access:

```
gcloud run services add-iam-policy-binding SERVICE_NAME \
    --member="allUsers" \
    --role="roles/run.invoker"
```

** Warning:** Missing Pydantic validation schemas

Both request and response must use Pydantic models:

```
from pydantic import BaseModel

class PredictRequest(BaseModel):
    features: list[float]

class PredictResponse(BaseModel):
    prediction: float
    model_version: str
```

Automated Sanity Checks

Run these commands locally before submitting:

File Existence Checks

```
# Check required files exist
echo "=== Checking required files ==="
test -f main.py && echo "✓ main.py exists" || echo "✗ main.py missing"
test -f README.md && echo "✓ README.md exists" || echo "✗ README.md missing"
(test -f requirements.txt || test -f pyproject.toml) && echo "✓ Dependency
   file exists" || echo "✗ No requirements.txt or pyproject.toml"
ls *.pkl 2>/dev/null && echo "✓ Model artifact found" || echo "✗ No .pkl
   model artifact found"
```

Dependency Pinning Validation

```
# Check for unpinned dependencies
echo "=== Checking for unpinned dependencies ==="
if [ -f requirements.txt ]; then
    grep -E '^[a-zA-Z]' requirements.txt | grep -v '==' | grep -v '^#' &&
        echo " Found unpinned dependencies above" || echo "✓ All
        dependencies appear pinned"
fi
```

FastAPI Local Test

```bash
# Test FastAPI locally
echo "=== Testing FastAPI locally ==="
pip install -r requirements.txt
uvicorn main:app --host 0.0.0.0 --port 8000 &
sleep 3
curl -X POST "http://localhost:8000/predict" \
    -H "Content-Type: application/json" \
    -d '{"features": [1.0, 2.0, 3.0]}'
pkill -f uvicorn
```

Pydantic Schema Verification

```bash
# Check for Pydantic models in main.py
echo "=== Checking for Pydantic schemas ==="
grep -E "class.*BaseModel" main.py && echo "✓ Pydantic models found" ||
   echo "✗ No Pydantic BaseModel classes found"
grep -E "@app\.(post|get).*response_model" main.py && echo "✓ Response
   model specified" || echo " Consider adding response_model to endpoint"
```

Cloud Run Deployment Verification

```bash
# Test Cloud Run endpoint (replace with your URL)
echo "=== Testing Cloud Run deployment ==="
CLOUD_RUN_URL="https://your-service-abc123.run.app"
curl -s -o /dev/null -w "%{http_code}" "$CLOUD_RUN_URL/predict" \
    -X POST \
    -H "Content-Type: application/json" \
    -d '{"features": [1.0, 2.0, 3.0]}' | \
    grep -q "200" && echo "✓ Cloud Run returns 200" || echo "✗ Cloud Run not
        responding correctly"
```

Cloud Function Verification

```bash
# Test Cloud Function endpoint (replace with your URL)
echo "=== Testing Cloud Function ==="
FUNCTION_URL="https://us-central1-your-project.cloudfunctions.net/predict"
curl -s -o /dev/null -w "%{http_code}" "$FUNCTION_URL" \
    -X POST \
    -H "Content-Type: application/json" \
    -d '{"features": [1.0, 2.0, 3.0]}' | \
    grep -q "200" && echo "✓ Cloud Function returns 200" || echo "✗ Cloud
        Function not responding correctly"
```

Latency Comparison Test

```bash
# Compare cold start vs warm latency
echo "=== Latency Comparison ==="
echo "Cloud Run cold start (after 15min idle):"
curl -w "Total time: %{time_total}s\n" -s -o /dev/null \
    -X POST "$CLOUD_RUN_URL/predict" \
    -H "Content-Type: application/json" \
    -d '{"features": [1.0, 2.0, 3.0]}'

echo "Cloud Run warm request (immediate follow-up):"
curl -w "Total time: %{time_total}s\n" -s -o /dev/null \
    -X POST "$CLOUD_RUN_URL/predict" \
    -H "Content-Type: application/json" \
    -d '{"features": [1.0, 2.0, 3.0]}'
```

Self-Assessment Questions

Answer these questions honestly before submitting:

Reproducibility

- ☐ **"Can someone clone my repo and run the FastAPI service in under 5 minutes?"**
    - Test: Clone to a different directory and follow your own setup instructions
- ☐ **"Are ALL my dependencies pinned to exact versions?"**
    - Check: no >=, <=, ~=, or ^ in your dependency file
- ☐ **"Does my model load identically every time?"**
    - Test: Load model twice and verify predictions match

Deployment

- ☐ **"Is my Cloud Run service publicly accessible right now?"**
    - Test: Open the URL in an incognito browser window
- ☐ **"Does my Cloud Function respond correctly?"**
    - Test: Invoke via curl from a different machine/network
- ☐ **"Have I captured evidence of both deployments working?"**
    - Screenshots or curl output saved and included

API Design

- ☐ **"Do my Pydantic schemas validate input correctly?"**
    - Test: Send malformed JSON and verify you get a 422 error
- ☐ **"Does my API return structured responses?"**
    - All responses should match the Pydantic response model

Lifecycle Understanding

- ☐ **“Can I explain where my deployment fits in the ML lifecycle?”**
  - Should connect: data → training → artifact → API → consumer
- ☐ **“Have I documented monitoring touchpoints?”**
  - Where would you add logging, metrics, alerts?

Comparative Analysis

- ☐ **“Have I tested and documented cold start behavior for both patterns?”**
  - Cloud Run and Cloud Functions behave differently
- ☐ **“Can I explain the trade-offs between stateful and stateless deployment?”**
  - When would you choose each pattern?
- ☐ **“Have I compared reproducibility across deployment patterns?”**
  - Container vs function: which is more reproducible and why?

Final Verification

- ☐ **“Have I committed and pushed all my changes?”**

```
git status  # Should show "nothing to commit, working tree clean"
```

- ☐ **“Are all my deployment URLs included in the README?”**
  - Cloud Run URL and Cloud Function URL both documented
- ☐ **“Is my repository accessible to the instructor?”**
  - Public repo: Anyone can view
  - Private repo: Check Settings → Collaborators

Quick Reference: Expected Repository Structure

```
your-repo/
├── main.py                     # FastAPI application
├── model.pkl                   # Trained model artifact
├── requirements.txt            # OR pyproject.toml
├── Dockerfile                  # For Cloud Run deployment
├── cloud_function/             # Cloud Function code
│   ├── main.py                 # Function entry point
│   └── requirements.txt        # Function dependencies
├── README.md                   # Documentation with:
│   ├── Setup instructions
│   ├── API usage examples
│   ├── Deployment URLs
│   ├── Lifecycle explanation
│   └── Comparative analysis
├── screenshots/                # Optional: deployment evidence
│   ├── cloud_run_response.png
│   └── cloud_function_logs.png
└── tests/                      # Optional: local tests
    └── test_api.py
```

Submission Confirmation

Once all checks pass:

1. Verify your Cloud Run URL responds (format: `your-service.run.app/predict`)
2. Verify your Cloud Function URL responds (format: `region-project.cloudfunctions.net/pred`
3. Confirm comparative analysis is included in README
4. Copy your repository URL (format: `github.com/YOUR_USERNAME/YOUR_REPO`)
5. Submit the URL via the course submission system