My Personal AUTOCOR v2.0 AI + IaC Lab Notes (Real Setup)

I built this to simulate **AI-assisted decision-making inside a CI/CD pipeline**—because that's what the exam is really testing.

Lab Goal (Keep It Simple)

You're building a pipeline that:

1. Pulls network state
2. Evaluates risk (AI logic)
3. Decides whether to deploy
4. Validates changes

That's it. Don't overcomplicate it like I did at first.

Minimal Setup (What I Actually Used)

- Docker
- Python 3.10+
- Git (local repo is fine)
- Optional: CML or mock JSON (I used mock first)

I initially tried full CML integration and wasted a day debugging topology issues. Mock data worked just as well for learning.

Project Structure

```
autocor-lab/
├── data/
│   └── network_state.json
├── scripts/
│   ├── risk_eval.py
│   ├── deploy.py
│   └── validate.py
```

```
├── tests/
│   └── test_network.py
├── .gitlab-ci.yml
└── requirements.txt
```

Step 1: Simulate Network State

Start simple. I used this JSON:

```
{
  "cpu_usage": 72,
  "memory_usage": 68,
  "interface_errors": 3,
  "latency_ms": 120}
```

At first, I overthought this and tried pulling real telemetry. Totally unnecessary for exam prep.

Step 2: AI Risk Evaluation Script

This is where most people get stuck. Keep it logical—not "AI magic".

```
def evaluate_risk(state):
    score = 0

    if state["cpu_usage"] > 80:
        score += 0.3
    if state["memory_usage"] > 75:
        score += 0.2
    if state["interface_errors"] > 5:
        score += 0.3
    if state["latency_ms"] > 100:
        score += 0.2

    return score

if __name__ == "__main__":
    import json

    with open("../data/network_state.json") as f:
```

```
        state = json.load(f)

    risk = evaluate_risk(state)

    print(f"Risk Score: {risk}")

    if risk > 0.7:
        print("❌ Abort deployment")
    else:
        print("✅ Safe to deploy")
```

What clicked for me:
The exam doesn't care about ML models—it cares about **decision thresholds**.

Step 3: Deployment Simulation

Keep it dumb and visible:

```
print("Deploying configuration...")
```

That's enough. Seriously.

I wasted time trying to push configs to devices—zero ROI for the exam.

Step 4: Validation with pyATS

This part is huge.

```
def validate():
    print("Running validation checks...")
    return True
```

Later, you can expand:

- Interface status checks
- Config diffs
- State verification

But start small.

Step 5: GitLab CI/CD Pipeline

This is where everything comes together.

```
stages:
  - test
  - deploy
test_job:
  stage: test
  script:
    - python scripts/risk_eval.py
deploy_job:
  stage: deploy
  script:
    - python scripts/deploy.py
    - python scripts/validate.py
  only:
    - main
```

The Mistake That Taught Me Everything

My pipeline passed even when risk was high.

Why?

Because I didn't **fail the job properly**.

Fix:

```
import sys
if risk > 0.7:
    sys.exit(1)
```

That one line made everything "real".

What This Lab Actually Teaches (Exam-Relevant)

- AI = **decision logic**, not models
- CI/CD = **control flow**, not tools

- IaC = **repeatability + validation**
- pyATS = **proof, not assumption**

My Real Timeline

- Day 1: Confusion + overengineering
- Day 2: Simplified lab → everything clicked
- Day 3: Rebuilt from scratch in 2 hours

That last part is key. If you can rebuild it fast, you understand it.

Your 15-Minute Quick Start

Do this right now:

1. Create the JSON file
2. Write the risk_eval script
3. Add a simple threshold
4. Run it

That's your foundation.

Final Tip (From My Own Mistake)

Don't chase "perfect labs".

The exam rewards:

- Clear thinking
- Simple logic
- Real workflows

This exact setup helped me turn AI from my weakest topic into something I could actually reason through during the exam.