Bước 1 (ADK) — Shop Agent on-prem

Tài liệu này "zoom" sâu vào **Bước 1** của dự án học ADK → MCP → A2A → AP2. Mục tiêu là bạn có ngay một agent chạy tại chỗ (on-prem), biết gọi **tools** đơn giản (tra cứu sản phẩm, tính phí ship…) và sẵn sàng để gắn vào các bước sau.

---

1) Mục tiêu của Bước 1

- Nắm chắc **ADK**: tạo 1 agent, gắn **tools** (hàm Python), chạy local qua **CLI/Dev UI**, và (tuỳ chọn) mở **API server** để test REST.
- Chuẩn bị nền cho **Bước 2 (MCP)**, **Bước 3 (A2A)** và **Bước 4 (AP2)**.

---

2) Các chức năng nên có (nhỏ nhưng "đủ bài")

1. **Tool tra cứu sản phẩm cục bộ**
2. Input: `sku | name`
3. Output: `{status, product{sku,name,price,currency}}`
4. Dữ liệu: dict hoặc `data/products.json` (100% on-prem).
5. **Tool tính phí ship đơn giản**
6. Input: `destination, weight_kg`
7. Output: `{status, shipping_fee, currency}` (công thức đơn giản).
8. **(Tuỳ chọn) Tool xem thời gian theo thành phố**
9. Dùng để demo tool-calling, kiểm tra khả năng xử lý tham số.
10. **(Tuỳ chọn) Memory tối giản**
11. In-memory (chưa cần DB) để ghi vài tình trạng/nhật ký ngắn.

---

3) Cấu trúc dự án tối thiểu

```
adk-step1/
└─ shop_agent/
   ├─ __init__.py
   ├─ agent.py
```

```
    ├─ data/
    │  └─ products.json         # nếu không hard-code
    └─ .env                     # cấu hình model (nếu dùng cloud) hoặc endpoint
on-prem
```

---

4) Cài đặt & chạy

Tuỳ chọn A — Nhanh nhất (dùng Gemini qua API key)

```
python -m venv .venv && source .venv/bin/activate
pip install google-adk
```

Tạo `shop_agent/.env`:

```
GOOGLE_GENAI_USE_VERTEXAI=FALSE
GOOGLE_API_KEY=YOUR_API_KEY
```

Chạy:

```
adk run shop_agent      # CLI
adk web                 # Dev UI tại http://127.0.0.1:8000
adk api_server          # mở FastAPI local để test REST
```

Tuỳ chọn B — On-prem tuyệt đối (Ollama/vLLM qua LiteLLM)

- Dùng **LiteLLM** để trỏ tới **Ollama** (localhost:11434) hoặc **vLLM/OpenAI-compatible** endpoint tự host.
- Phù hợp yêu cầu on-prem của doanh nghiệp.

---

5) Mẫu code tham chiếu

`shop_agent/agent.py`

```
import datetime
from zoneinfo import ZoneInfo
from google.adk.agents import Agent, LlmAgent
from google.adk.models.lite_llm import LiteLlm

# ---------------- Tools ----------------
```

```
CATALOG = {
    "SKU001": {"sku": "SKU001", "name": "Anime Figure A", "price": 29.9,
"currency": "USD"},
    "SKU002": {"sku": "SKU002", "name": "Manga Volume 1", "price": 12.5,
"currency": "USD"},
}

def find_product(query: str) -> dict:
    """
    Find product by SKU or substring of name.
    Returns: {status: 'success'|'error', product?: {...}, error_message?: str}
    """
    q = query.strip().lower()
    # match by SKU
    for k, v in CATALOG.items():
        if k.lower() == q:
            return {"status": "success", "product": v}
    # match by name substring
    for p in CATALOG.values():
        if q in p["name"].lower():
            return {"status": "success", "product": p}
    return {"status": "error", "error_message": f"No product matches '{query}'"}


def calc_shipping(destination: str, weight_kg: float) -> dict:
    """Naive shipping fee: base 2.0 + zone + 0.8*weight_kg"""
    zone_fee = 0.0 if destination.lower() in ["hanoi", "hochiminh", "saigon"]
else 1.5
    fee = round(2.0 + zone_fee + 0.8 * float(weight_kg), 2)
    return {"status": "success", "shipping_fee": fee, "currency": "USD"}


def get_current_time(city: str) -> dict:
    tzmap = {"new york": "America/New_York", "hanoi": "Asia/Bangkok"}
    if city.lower() not in tzmap:
        return {"status": "error", "error_message": f"Unknown timezone for
'{city}'"}
    now = datetime.datetime.now(ZoneInfo(tzmap[city.lower()]))
    return {"status": "success", "report": now.strftime("%Y-%m-%d %H:%M:%S
%Z%z")}


# ---------------- Agent (Gemini/API) ----------------
# Cách 1: dùng model cloud (nhanh để bootstrap)
root_agent = Agent(
    name="shop_agent",
    model="gemini-2.0-flash",  # thay model phù hợp
    description="Shopping helper with local catalog & shipping calculator.",
```

```
    instruction=(
        "You are a helpful assistant for shopping. "
        "Always call tools when user asks about catalog, price, or shipping
cost."
    ),
    tools=[find_product, calc_shipping, get_current_time],
)


# ---------------- Agent (On-prem) ----------------
# Cách 2: dùng LiteLLM → vLLM/Ollama/OpenAI-compatible endpoint
# Bỏ comment để dùng on-prem tuyệt đối
"""
VLLM_BASE = "http://localhost:8000/v1"   # endpoint tự host
MODEL_ID  = "google/gemma-2-9b-it"       # model bạn đang phục vụ

root_agent = LlmAgent(
    model=LiteLlm(
        model=MODEL_ID,
        api_base=VLLM_BASE,
        # api_key="YOUR_API_KEY_IF_ANY",
    ),
    name="shop_agent_vllm",
    instruction="You are a helpful shopping assistant.",
    tools=[find_product, calc_shipping, get_current_time],
)
"""
```

Gợi ý: luôn trả về dict có `status` rõ ràng để LLM dễ xử lý nhánh lỗi/thành công.

---

6) Chạy & kiểm thử nhanh

- **CLI**

```
adk run shop_agent
```

Ví dụ hỏi: “tìm SKU001”, “ship về Hanoi nặng 1.2 kg bao nhiêu?”, “giờ ở New York?”.

- **Dev UI**

```
adk web
# mở http://127.0.0.1:8000 → chọn "shop_agent" trong dropdown
```

Dùng **Events/Trace** để quan sát tool-call, input/output.

- **API server** (sẵn sàng nối Django/Ingress)

```
adk api_server
```

Mẫu cURL (tham khảo)

```
# tuỳ endpoint ADK in ra khi start api_server
curl -X POST http://127.0.0.1:8000/v1/chat
  -H 'Content-Type: application/json'
  -d '{"agent": "shop_agent", "message": "Tính ship về Hanoi nặng 1.5kg"}'
```

---

7) Unit test tối thiểu (không phụ thuộc LLM)

`tests/test_tools.py`

```
from shop_agent.agent import find_product, calc_shipping

def test_find_product_by_sku():
    out = find_product("SKU001")
    assert out["status"] == "success"
    assert out["product"]["sku"] == "SKU001"

def test_find_product_not_found():
    out = find_product("nope")
    assert out["status"] == "error"


def test_calc_shipping_hanoi():
    out = calc_shipping("Hanoi", 1.5)
    assert out["status"] == "success"
    assert out["shipping_fee"] > 0
```

Chạy:

```
pip install pytest
pytest -q
```

---

8) Definition of Done (điều kiện “PASS” Bước 1)

- [ ] **Chạy được**: `adk run shop_agent` trả lời trơn tru, câu hỏi cần tool phải **gọi tool**.
- [ ] **Quan sát được tool-call** trong Dev UI → Events/Trace, input/output đúng schema.
- [ ] **Xử lý lỗi gọn**: SKU không tồn tại → phản hồi `{status:"error", ...}` và hội thoại lịch sự.
- [ ] **Không “ảo tưởng”**: câu hỏi lý thuyết → trả lời mạch lạc, không cố gọi tool.
- [ ] **On-prem OK**: đổi sang **LiteLLM + vLLM/Ollama** vẫn chạy, tool-calling giữ nguyên.
- [ ] **Unit test pass**: `pytest` cho `find_product` / `calc_shipping`.
- [ ] **API sẵn sàng**: `adk api_server` hoạt động để gắn vào Django/Ingress sau này.

---

9) Mẹo & lỗi thường gặp

- `.env` thiếu biến → model không chạy (nếu dùng cloud). Kiểm tra thông số xác thực.
- Windows chạy `adk web` có thể cần `--no-reload`.
- Dùng vLLM/OpenAI-compatible: đảm bảo endpoint hỗ trợ **function/tool calling**.
- Tool schema: luôn có `status`, mô tả lỗi rõ ràng (`error_message`) để dễ debug.

---

10) Bước tiếp theo (nhá hàng Bước 2)

- Tạo **MCP server** cục bộ (REST) cung cấp tool “Tra cứu giá/ tồn kho/ đơn hàng”.
- Tích hợp agent ADK với **MCP client** để gọi tool qua **MCP** (thay vì gọi trực tiếp).
- Giữ nguyên các bài test logic để đảm bảo thay đổi kiến trúc không phá chức năng.