Prereqs — Toolkit & Accounts (do these once)

Software (PC)

- IntelliJ IDEA (install **Python** plugin)
- **Google Cloud SDK** (`gcloud`)
- **Docker Desktop** (includes docker compose)
- **Git**
- **Python 3.11+**
- **Poetry** (dependency & venv manager)
- (Optional) **Postman**; otherwise use `curl`

Accounts

- **GCP** (billing enabled; 1 project)
- **GitHub** (private repo)
- **OpenAI API key** (you asked to use OpenAI)

Baseline Decisions (sane defaults)

- **Local-first:** Redis, Postgres, Qdrant in Docker to avoid cost. You can switch to **GCP Memorystore** and **Cloud SQL** later without code changes.
- **Secrets:** GCP Secret Manager (never commit keys).

- **Audit:** OpenTelemetry → GCP Cloud Logging (write-only from agents).
- **Embeddings:** Start with **OpenAI** (simple), later you can swap to local models.
- **Package mgmt: Poetry**.
- **Language:** Python (services via **FastAPI**).

Ultimate Phase-1 Plan (14 days)

Day 1 — Project & Repo Setup, Secure Foundation

Goal: New repo + Poetry project + GCP project + first secret in Secret Manager.

1. **Repo**

```
git clone <your-private-repo-url> agent-factory && cd agent-factory
```

2. **Poetry project**

```
poetry init -n
poetry env use 3.11
poetry add fastapi uvicorn pydantic httpx python-dotenv
poetry add --group dev pytest coverage
```

3. **Scaffold**

```
/agent-factory
  /src/agent_factory
    __init__.py
    /core            # config, logging, clients
    /services
      /audit         # Compliance & Audit Kernel
```

```
    /memory        # Cognitive Engine adapters
    /protocol      # Protocol Fabric endpoints (MVP)
  /tools           # Certified tool implementations
/tests
docker-compose.yml
pyproject.toml
.env.example
README.md
```

4. **GCP project (replace IDs)**

```
gcloud init
gcloud projects create agent-factory-123456
gcloud config set project agent-factory-123456
```

5. **Enable APIs**

```
gcloud services enable \
iam.googleapis.com \
secretmanager.googleapis.com \
artifactregistry.googleapis.com \
logging.googleapis.com \
cloudbuild.googleapis.com
```

(Add redis/sql later only if you choose managed DBs.)

6. **Secrets — OpenAI key**

```
gcloud secrets create openai-api-key --replication-policy=automatic
printf "sk-your-openai-key" | gcloud secrets versions add
openai-api-key --data-file=-
gcloud auth application-default login   # sets ADC for local dev
```

Success: Repo exists, Poetry works, GCP project set, OpenAI key stored in Secret Manager.

Day 2 — IAM & Compliance Kernel Identity

Goal: Least-privilege service account that can **write logs** but **not read/alter**.

1. **Service account**

```
gcloud iam service-accounts create agent-worker \
  --display-name="Agent Worker"
```

2. **Grant minimal logging role**

```
gcloud projects add-iam-policy-binding agent-factory-123456 \

--member="serviceAccount:agent-worker@agent-factory-123456.iam.gservic
eaccount.com" \
  --role="roles/logging.logWriter"
```

3. **Local dev uses ADC** (already set Day 1). No key files.

Success: SA created; it can write logs.

Day 3 — Local Databases with Docker (default) or GCP Managed (optional)

Goal: Stand up **Redis**, **Postgres**, and **Qdrant** locally. (Managed options below.)

docker-compose.yml

```
version: "3.9"
services:
  redis:
    image: redis:7
    ports: ["6379:6379"]

  postgres:
```

```
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: agentdb
    ports: ["5432:5432"]

  qdrant:
    image: qdrant/qdrant:latest
    ports: ["6333:6333"]
```

```
docker compose up -d
```

Managed options (optional now / later)

- Memorystore (Redis):

```
gcloud services enable redis.googleapis.com
gcloud redis instances create short-term-memory --size=1
--region=us-central1 --tier=BASIC
```

- Cloud SQL (Postgres):

```
gcloud services enable sqladmin.googleapis.com
gcloud sql instances create procedural-memory \
  --database-version=POSTGRES_14 --region=us-central1 --cpu=1
--memory=4GB
```

Success: `docker ps` shows redis/postgres/qdrant up; you can connect on localhost.

Day 4 — Compliance & Audit Kernel (API + OTEL to GCP Logging)

Goal: Minimal FastAPI service to accept audit events and send to GCP Logging; all events carry trace IDs.

```
poetry add opentelemetry-sdk opentelemetry-exporter-otlp \
opentelemetry-api opentelemetry-instrumentation-fastapi
google-cloud-logging
```

src/agent_factory/services/audit/main.py (sketch)

```
from fastapi import FastAPI, Request
from google.cloud import logging_v2
import uuid, time

app = FastAPI(title="Compliance & Audit Kernel")

client = logging_v2.Client()
logger = client.logger("agent-audit")

@app.post("/log")
async def log_event(req: Request):
    body = await req.json()
    trace_id = body.get("trace_id") or uuid.uuid4().hex
    event = {
        "trace_id": trace_id,
        "ts": time.time(),
        "event_type": body.get("event_type", "unknown"),
        "payload": body.get("payload", {}),
    }
    logger.log_struct(event, severity="INFO")
    return {"status": "ok", "trace_id": trace_id}
```

Run & test

```
poetry run uvicorn agent_factory.services.audit.main:app --reload
curl -X POST http://127.0.0.1:8000/log -H "Content-Type:
application/json" \
  -d '{"event_type":"boot","payload":{"service":"audit"}}'
```

Success: You see the log in **GCP → Logging** under `agent-audit`.

Day 5 — Cognitive Engine Adapters (Redis, Postgres, Qdrant)

Goal: Simple clients & smoke tests.

```
poetry add redis sqlalchemy psycopg2-binary qdrant-client
python-dotenv
```

`src/agent_factory/core/clients.py` (sketch)

```
import os, redis
from sqlalchemy import create_engine, text
from qdrant_client import QdrantClient

REDIS_URL = os.getenv("REDIS_URL","redis://localhost:6379/0")
POSTGRES_URL =
os.getenv("POSTGRES_URL","postgresql+psycopg2://postgres:postgres@loca
lhost:5432/agentdb")
QDRANT_URL = os.getenv("QDRANT_URL","http://localhost:6333")

def get_redis(): return redis.Redis.from_url(REDIS_URL,
decode_responses=True)
def get_engine(): return create_engine(POSTGRES_URL, future=True)
def get_qdrant(): return QdrantClient(url=QDRANT_URL)
```

`.env.example`

```
GCP_PROJECT=agent-factory-123456
ENV=dev
REDIS_URL=redis://localhost:6379/0
POSTGRES_URL=postgresql+psycopg2://postgres:postgres@localhost:5432/ag
entdb
QDRANT_URL=http://localhost:6333
```

Success: quick script sets/gets Redis; creates a table; pings Qdrant.

Day 6 — Protocol Fabric (MVP A2A/MCP-lite)

Goal: Lightweight discovery + invoke endpoints.

```
poetry add pydantic[email]
```

src/agent_factory/services/protocol/main.py (sketch)

```
from fastapi import FastAPI
from pydantic import BaseModel
import uuid

class AgentCard(BaseModel):
    name: str
    version: str
    capabilities: list[str]
    endpoints: dict

class InvokeRequest(BaseModel):
    tool: str
    args: dict

app = FastAPI(title="Protocol Fabric MVP")

@app.get("/agent-card")
def card():
    return AgentCard(
        name="Protocol Fabric",
        version="0.1.0",
        capabilities=["invoke","discover"],
        endpoints={"invoke":"/invoke","health":"/health"}
    )

@app.post("/invoke")
def invoke(req: InvokeRequest):
    trace_id = uuid.uuid4().hex
    # wire to tools later
    return {"trace_id": trace_id, "tool": req.tool, "status":"stub"}
```

```
@app.get("/health")
def health(): return {"ok": True}
```

Success: `curl` returns an agent card JSON; `/invoke` replies a stub.

Day 7 — “Certified Tool” Template + Tests + Audit Hooks

Goal: One canonical tool interface (schema, audit before/after, errors).

```
poetry add requests
```

`src/agent_factory/tools/base.py` (pattern)

```
from pydantic import BaseModel, Field
import requests, os

AUDIT_URL = os.getenv("AUDIT_URL","http://127.0.0.1:8000/log")

class ToolError(Exception): ...

class ToolInput(BaseModel):
    pass

def audit(event_type:str, payload:dict):
    try: requests.post(AUDIT_URL,
json={"event_type":event_type,"payload":payload}, timeout=3)
    except: pass

def run_tool(name:str, fn, args:dict):
    audit("tool.call", {"tool":name, "args":args})
    try:
        out = fn(**args)
        audit("tool.return", {"tool":name, "result":out})
        return out
    except Exception as e:
        audit("tool.error", {"tool":name, "error":str(e)})
```

```
        raise
```

Example tool `file_writer`

```
from pydantic import BaseModel
from .base import run_tool
import os

class FileWriterInput(BaseModel):
    file_path: str
    content: str

def file_writer_impl(file_path:str, content:str):
    os.makedirs(os.path.dirname(file_path), exist_ok=True)
    with open(file_path,"w",encoding="utf-8") as f: f.write(content)
    return {"bytes": len(content)}

def file_writer(args:dict):
    data = FileWriterInput(**args)
    return run_tool("file_writer", file_writer_impl, data.dict())
```

Test `tests/test_file_writer.py`

```
from src.agent_factory.tools.file_writer import file_writer
def test_file_writer(tmp_path):
    p = tmp_path/"a/b.txt"
    out = file_writer({"file_path":str(p),"content":"hello"})
    assert out["bytes"]==5
    assert p.read_text()=="hello"

poetry run pytest -q
```

Success: Tests pass; audit logs land in GCP.

Day 8 — 3 Core Tools (Ingest, Embed, Vector Upsert)

Goal: Build 3 tools and expose through Protocol Fabric `/invoke`.

```
poetry add sentence-transformers unstructured pymupdf qdrant-client
```

- **ingest_file** → returns list of text chunks + metadata
- **embed_texts** → OpenAI embeddings **or** local `sentence-transformers`
- **upsert_vectors** → Qdrant add with metadata

Wire `/invoke` to route `tool` name to the tool function from a registry.

Success: You can ingest a PDF/MD, embed, upsert, and see vectors in Qdrant.

Day 9 — +2 Tools (Search Index, Vector Search) + CLI

Goal: Reach 5 certified tools; basic CLI to run tools via Protocol Fabric.

```
poetry add typer rich
```

- **index_metadata** → store doc metadata in Postgres
- **vector_search** → search Qdrant top-k and return doc refs

CLI `src/agent_factory/services/tools/cli.py`

- `list-tools`, `run <tool> --args '{...}'`

Success: CLI lists and runs tools; audit logs recorded.

Day 10 — Knowledge Curator Agent + Retrieval Benchmark (≥90%)

Goal: An “Archivist” service that chains tools to build your RAG store + test retrieval.

```
poetry add langchain langchain-openai langchain-community
```

- Curator runs: ingest → embed → upsert → index
- Create `data/benchmarks/qa.jsonl` (20 Qs, expected doc ids)
- Test: For each Q, perform `vector_search` (k=5) and check if expected doc appears → **≥90% hit-rate**

Success: Benchmark ≥90% on your 20 Qs.

Day 11 — Compliance Integration Everywhere

Goal: Every tool/agent call logs **before**, **after**, **error** with trace IDs.

- Ensure all tools use `run_tool(...)` wrapper.
- Add HTTP middleware in services to add/propagate a `trace_id`.
- Verify logs are structured (tool, args, outcome) and visible in GCP Logging.

Success: End-to-end flow visible in Cloud Logging with consistent `trace_id`.

Day 12 — Personas + Prompt Hygiene

Goal: Encode the **Master Tool Craftsman** and **Librarian Archivist** personas.

- In agent prompts, add persona instructions (robustness, error handling, metadata discipline).
- Add input validation notes in tools (Pydantic models already in place).

Success: Agents include persona behaviors (visible in outputs / code they generate).

Day 13 — CI Pipeline (GitHub Actions) + Security Review L1

Goal: CI green; minimal threat model doc.

`.github/workflows/ci.yml` (sketch)

```
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: {python-version: '3.11'}
      - name: Install Poetry
        run: pipx install poetry
      - name: Install deps
        run: poetry install
      - name: Tests
        run: poetry run pytest -q
```

Security L1 doc `/docs/security/level1-review.md`

- Secrets in Secret Manager
- Least privilege IAM
- Validate & sanitize inputs
- Rate limiting (add simple in Protocol Fabric)
- Don’t log sensitive values (redaction)

Success: CI green; security doc committed.

Day 14 — Acceptance Review & Tag

Checklist

- Compliance Kernel receives events; logs in GCP with trace IDs
- Cognitive Engine: Redis/Postgres/Qdrant operational
- Protocol Fabric: `/agent-card`, `/invoke`, `/health`
- **5 certified tools**, unit-tested, audited
- Knowledge Curator builds RAG; **≥90%** retrieval hit-rate
- CI green; Level-1 Security review doc
- Basic runbook in `/docs/runbook.md`

```
git add .
git commit -m "Phase 1 complete"
git tag phase-1-complete
git push --tags
```

Success: All boxes checked; you're Phase-2 ready.

Optional: Switch to GCP-Managed Memory Later

- **Redis → Memorystore:** update `REDIS_URL` to the instance endpoint.
- **Postgres → Cloud SQL:** use a Cloud SQL connector or public IP + SSL.
- **Qdrant → Managed:** you can move to a hosted vector DB (e.g., Qdrant Cloud, Pinecone) by swapping the client and URL. Keep your tool interfaces unchanged.

Environment & Run Tips (IntelliJ)

- Add `.env` and set **Run Configurations**:
    - `AUDIT_URL=http://127.0.0.1:8000/log`
    - URLs for Redis/Postgres/Qdrant (local or managed)
    - Export OpenAI key to env at runtime:
      `export OPENAI_API_KEY=$(gcloud secrets versions access latest --secret="openai-api-key")`