# 1. CoordinationAgent (Central Orchestrator)

from uagents import Agent, Context, Model

coordination\_agent = Agent(name="coordination\_agent", seed="coordination\_seed")

class HTTPRequest(Model):

    type: str

    data: dict

class ComplianceMessage(Model):

    source: str

    payload: dict

@coordination\_agent.on\_event("startup")

async def on\_startup(ctx: Context):

    ctx.logger.info("🚀 CoordinationAgent is up")

@coordination\_agent.protocol("http", path="/compliance", method="POST")

async def handle\_http(ctx: Context, req: HTTPRequest):

    t = req.type.lower()

    if t == "transaction":

        await ctx.send("AMLAlertAgent", req.data)

    elif t == "kyc":

        await ctx.send("KYCValidatorAgent", req.data)

    elif t == "reg\_fetch":

        await ctx.send("RegWatcherAgent", {})

    else:

        ctx.logger.warn(f"Unknown type: {req.type}")

    return {"status":"started","routed\_to":req.type}

@coordination\_agent.on\_message(model=ComplianceMessage)

async def handle\_compliance(ctx: Context, sender: str, msg: ComplianceMessage):

    ctx.logger.info(f"📨 Received from {msg.source}: {msg.payload}")

2. RegWatcherAgent (Regulation Monitor)from uagents import Agent, Context, Model

import requests

from bs4 import BeautifulSoup

class ComplianceMessage(Model): source: str; payload: dict

reg\_watcher\_agent = Agent(name="RegWatcherAgent", seed="reg\_fetch\_seed")

@reg\_watcher\_agent.on\_event("startup")

async def fetch\_reg\_updates(ctx: Context):

    sources = {"CBUAE": "https://www.centralbank.ae/...rss",

               "DFSA": "https://www.difc.ae/news/rss",

               "ADGM": "https://www.adgm.com/.../rss.xml"}

    updates = []

    for name,url in sources.items():

        try:

            resp=requests.get(url,timeout=5)

            feed=BeautifulSoup(resp.text,"xml")

            for item in feed.find\_all("item")[:3]:

                updates.append({"regulator":name,

                                "title":item.title.text,

                                "link":item.link.text})

        except Exception as e:

            ctx.logger.error(f"Fetch failed {name}: {e}")

    await ctx.send("coordination\_agent",ComplianceMessage(source="RegWatcherAgent",payload={"updates":updates}))

3. AMLAlertAgent (Transaction Monitor)from uagents import Agent, Context, Model

class Transaction(Model): tx\_id: str; details: dict

class ComplianceMessage(Model): source: str; payload: dict

aml\_agent = Agent(name="AMLAlertAgent", seed="aml\_seed")

@aml\_agent.on\_message(model=Transaction)

async def detect\_aml(ctx: Context, sender: str, msg: Transaction):

    if msg.details.get("amount",0)>100000:

        alert={"tx\_id":msg.tx\_id,"issues":["High-value"]}

        await ctx.send("coordination\_agent",ComplianceMessage(source="AMLAlertAgent",payload={"alert":alert}))

        ctx.logger.info(f"🚩 AML alert {msg.tx\_id}")

    else:

        ctx.logger.debug(f"✔ Tx {msg.tx\_id} clean")

4. KYCValidatorAgent (Identity Screening)

from uagents import Agent, Context, Model

class KYCRequest(Model):

    user\_id: str

    doc\_data: dict

class ComplianceMessage(Model):

    source: str

    payload: dict

kyc\_agent = Agent(name="KYCValidatorAgent", seed="kyc\_seed")

SANCTION\_LIST = {"EvilCorp"}

PEP\_LIST = {"PoliticianA"}

@kyc\_agent.on\_message(model=KYCRequest)

async def validate(ctx: Context, sender: str, msg: KYCRequest):

    issues = []

    if msg.user\_id in SANCTION\_LIST:

        issues.append("Sanction hit")

    if msg.user\_id in PEP\_LIST:

        issues.append("PEP hit")

    result = {"user\_id": msg.user\_id, "passed": not issues, "issues": issues}

    await ctx.send(

      "coordination\_agent",

      ComplianceMessage(source="KYCValidatorAgent", payload={"kyc\_result": result})

    )

    ctx.logger.info(f"{'✅' if not issues else '⚠️'} KYC {msg.user\_id}")

# 5. RiskScorerAgent (Risk Aggregator)

from uagents import Agent, Context, Model

class ComplianceMessage(Model):

    source: str

    payload: dict

risk\_agent = Agent(name="RiskScorerAgent", seed="risk\_scorer\_seed")

@risk\_agent.on\_message(model=ComplianceMessage)

async def score\_risk(ctx: Context, sender: str, msg: ComplianceMessage):

    score = 0.1

    details = {"source": msg.source}

    # AML alerts

    alert = msg.payload.get("alert")

    if alert:

        score += 0.5

        details["aml\_issues"] = alert.get("issues")

    # KYC failures

    kyc = msg.payload.get("kyc\_result")

    if kyc and not kyc.get("passed", True):

        score += 0.3

        details["kyc\_issues"] = kyc.get("issues")

    score = min(score, 1.0)

    # Forward scored message

    await ctx.send(

        "coordination\_agent",

        ComplianceMessage(source="RiskScorerAgent", payload={"risk\_score": score, "details": details})

    )

    ctx.logger.info(f"🔢 Risk score: {score}")

# 6. ReportGenAgent (Report Builder)

import uuid

from datetime import datetime

from uagents import Agent, Context, Model

class ComplianceMessage(Model):

    source: str

    payload: dict

class Report(Model):

    report\_id: str

    timestamp: str

    entries: list

report\_agent = Agent(name="ReportGenAgent", seed="report\_gen\_seed")

# In-memory store

ENTRIES = []

@report\_agent.on\_message(model=ComplianceMessage)

async def compile\_report(ctx: Context, sender: str, msg: ComplianceMessage):

    ENTRIES.append({

        "source": msg.source,

        "payload": msg.payload,

        "received\_at": datetime.utcnow().isoformat()

    })

    report = Report(

        report\_id=str(uuid.uuid4()),

        timestamp=datetime.utcnow().isoformat(),

        entries=ENTRIES.copy()

    )

    await ctx.send("NotifierAgent", report)

    ctx.logger.info(f"📑 Generated report {report.report\_id} ({len(ENTRIES)} entries)")

# 7. NotifierAgent (Dispatch) import requests

import requests

from uagents import Agent, Context, Model

class Report(Model):

    report\_id: str

    timestamp: str

    entries: list

notifier\_agent = Agent(name="NotifierAgent", seed="notifier\_seed")

@notifier\_agent.on\_message(model=Report)

async def dispatch\_report(ctx: Context, sender: str, report: Report):

    ctx.logger.info(f"✅ Notifier received report {report.report\_id} with {len(report.entries)} entries")

    # Optional: forward to external webhook

    # requests.post("https://dashboard/api/reports", json=report.dict(), timeout=5)