

Quick Start: Real-Time Feedback Pipeline

Get up and running with the real-time feedback pipeline in 5 minutes.

Prerequisites

- Python 3.8+
- Docker and docker-compose (for Kafka)
- Backend dependencies installed

Step 1: Install Dependencies

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
pip install -r requirements.txt
```

This installs `kafka-python==2.0.2` along with other dependencies.

Step 2: Choose Your Setup

Option A: Without Kafka (Logging Only)

Perfect for development and testing. No external dependencies required.

Configuration (.env):

```
ENABLE_KAFKA=false
ENABLE_OUTCOME_LOGGING=true
```

[Skip to Step 4](#)

Option B: With Kafka (Full Real-Time)

Recommended for production. Enables real-time event streaming.

[Continue to Step 3](#)

Step 3: Setup Kafka (Option B Only)

Quick Setup with Docker

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
./scripts/setup_kafka.sh
```

This script will:

- Start Zookeeper and Kafka containers
- Create required topics (agent-outcomes, agent-metrics, agent-alerts)
- Start Kafka UI on <http://localhost:8080>

Configuration (.env):

```
ENABLE_KAFKA=true
KAFKA_BOOTSTRAP_SERVERS=localhost:9092
KAFKA_OUTCOME_TOPIC=agent-outcomes
```

Verify Kafka is Running

```
# Check containers
docker ps | grep kafka

# Check topics
docker exec -it kafka kafka-topics --list --bootstrap-server localhost:9092
```

Step 4: Test the Pipeline

Run Unit Tests

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
pytest tests/test_feedback_pipeline.py -v
```

Expected output:

```
tests/test_feedback_pipeline.py::test_outcome_event_creation PASSED
tests/test_feedback_pipeline.py::test_action_dispatcher_success PASSED
tests/test_feedback_pipeline.py::test_orchestration_dispatcher_workflow PASSED
...
```

Run Example Script

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
python examples/dispatcher_example.py
```

This demonstrates:

- Single agent dispatch
- Multi-agent workflow
- Event querying
- Custom hooks

Step 5: Integrate with Your Code

Basic Usage

```

import asyncio
from core.action_dispatcher import create_dispatcher
from agents.react import ReactAgent

async def main():
    # Create dispatcher
    dispatcher = create_dispatcher(
        kafka_servers="localhost:9092", # Or None for logging only
        enable_metrics=True
    )

    # Create agent
    agent = ReactAgent()

    # Dispatch action
    result = await dispatcher.dispatch(
        agent=agent,
        action_type="reasoning",
        input_data={"query": "Analyze customer sentiment"},
        context={},
        tags=["customer", "sentiment"]
    )

    print(f"Status: {result['status']}")
    print(f"Duration: {result['duration_ms']:.2f}ms")
    print(f"Event ID: {result['event_id']}")

asyncio.run(main())

```

Workflow Orchestration

```

from core.action_dispatcher import OrchestrationDispatcher
from agents.react import ReactAgent
from agents.evaluator import EvaluatorAgent

async def main():
    orchestrator = OrchestrationDispatcher()

    result = await orchestrator.execute_workflow(
        agents=[ReactAgent(), EvaluatorAgent()],
        task="Process compliance report",
        tags=["compliance", "reporting"]
    )

    for output in result['outputs']:
        print(f"{output['agent']}: {output['status']}")

asyncio.run(main())

```

Step 6: Monitor Events

View in Kafka UI (Option B)

Open <http://localhost:8080> in your browser:

- Navigate to Topics → agent-outcomes
- View real-time messages
- Monitor partition distribution

Query Programmatically

```
from core.feedback_pipeline import get_feedback_pipeline, OutcomeStatus

pipeline = get_feedback_pipeline()

# Get recent events
recent = pipeline.get_recent_events(count=10)

# Get failures
failures = pipeline.get_recent_events(
    count=20,
    status=OutcomeStatus.FAILURE
)

# Get events for specific agent
agent_events = pipeline.get_recent_events(
    count=10,
    agent_name="react_agent"
)
```

Check Logs

```
# Backend logs
tail -f logs/app.log

# Kafka logs
docker-compose -f docker-compose.kafka.yml logs -f kafka
```

Common Tasks

Start Kafka

```
docker-compose -f docker-compose.kafka.yml up -d
```

Stop Kafka

```
docker-compose -f docker-compose.kafka.yml down
```

View Kafka Logs

```
docker-compose -f docker-compose.kafka.yml logs -f
```

Reset Kafka (Delete All Data)

```
docker-compose -f docker-compose.kafka.yml down -v
./scripts/setup_kafka.sh
```

Test Kafka Connection

```
from kafka import KafkaProducer

try:
    producer = KafkaProducer(bootstrap_servers='localhost:9092')
    producer.close()
    print("\u2713 Kafka connection successful")
except Exception as e:
    print(f"\u2718 Kafka connection failed: {e}")
```

Troubleshooting

Issue: Kafka not connecting

Solution:

```
# Check if containers are running
docker ps | grep kafka

# Restart containers
docker-compose -f docker-compose.kafka.yml restart

# Check logs for errors
docker-compose -f docker-compose.kafka.yml logs kafka
```

Issue: Topics not created

Solution:

```
# Manually create topics
docker exec -it kafka kafka-topics --create \
    --bootstrap-server localhost:9092 \
    --replication-factor 1 \
    --partitions 3 \
    --topic agent-outcomes
```

Issue: Events not being published

Solution:

1. Check `.env` has `ENABLE_KAFKA=true`
2. Verify Kafka is running: `docker ps | grep kafka`
3. Check agent logs for Kafka errors
4. Test Kafka connection (see above)

Issue: Import errors

Solution:

```
# Reinstall dependencies
pip install -r requirements.txt

# Verify kafka-python is installed
pip show kafka-python
```

Performance Tuning

For High Volume

```
# Increase batch size
KAFKA_BATCH_SIZE=500
KAFKA_LINGER_MS=200

# Increase topic partitions
docker exec -it kafka kafka-topics --alter \
    --bootstrap-server localhost:9092 \
    --topic agent-outcomes \
    --partitions 10
```

For Low Latency

```
# Reduce batch size
KAFKA_BATCH_SIZE=10
KAFKA_LINGER_MS=10

# Disable compression
KAFKA_COMPRESSION=none
```

Next Steps

1. **Build Dashboards:** Create real-time dashboards using Kafka Streams or KSQL
2. **Set Up Alerts:** Configure alerts based on failure rates or latency
3. **Analytics:** Export events to data warehouse for historical analysis
4. **Monitoring:** Integrate with Prometheus/Grafana for system monitoring

Resources

- [Full Documentation](#) (README_FEEDBACK_PIPELINE.md)
- [Kafka Documentation](#) (<https://kafka.apache.org/documentation/>)
- [kafka-python Documentation](#) (<https://kafka-python.readthedocs.io/>)
- [Example Code](#) (examples/dispatcher_example.py)

Support

For issues or questions:

1. Check the [Full Documentation](#) (README_FEEDBACK_PIPELINE.md)
2. Review [test cases](#) (tests/test_feedback_pipeline.py) for examples
3. Check Kafka logs for connection issues

Ready to go! Start dispatching actions and see real-time outcome events flowing through your pipeline.