IBM Watsonx Orchestrate - Setup Guide for MeetingFlowAI

Overview

This guide covers setting up IBM Watsonx for the MeetingFlowAI project, which automates post-meeting workflows using AI agents.

---

1. What You Need

Prerequisites

- ✅ IBM Cloud account (you have this)
- ✅ Watsonx.ai API Key (you have this)
- ✅ Node.js 14+ or Python 3.9+ installed
- ✅ Project ID or Space ID from IBM Cloud

Get Your Credentials

1. Go to IBM Cloud Console
2. Navigate to **watsonx.ai** service
3. Copy these values:
   - **API Key** (you already have this)
   - **Project ID** (found in your watsonx.ai project)
   - **Service URL** (usually `https://us-south.ml.cloud.ibm.com`)

---

2. Architecture for MeetingFlowAI

```
┌─────────────────┐
│   Frontend      │
│   (React)       │
└────────┬────────┘
         │
         │ HTTP REST
         │
┌────────▼─────────────────────────────────┐
│   Backend (Node.js/Python)               │
│                                          │
│   ┌──────────────────────────────────┐   │
│   │ Watsonx.ai SDK                   │   │
│   │ - Granite model for NLP          │   │
│   │ - Extract meeting data           │   │
│   └─────────────┬────────────────────┘   │
│                 │                        │
│   ┌─────────────▼────────────────────┐   │
│   │ Orchestrate Agent                │   │
│   │ - Process workflow               │   │
│   │ - Handle multi-step automation│   │
│   └─────────────┬────────────────────┘   │
│                 │                        │
│   ┌─────────────▼────────────────────┐   │
│   │ Integration Layer                │   │
│   │ - Salesforce CRM API             │   │
│   │ - Email (optional)               │   │
│   └──────────────────────────────────┘   │
└──────────────────────────────────────────┘
```

---

3. Integration Approach

Option A: Watsonx.ai SDK (Recommended for MVP)

Use watsonx.ai directly for AI extraction without full Orchestrate platform.

Pros:

- ✅ Faster setup (just API calls)
- ✅ Full control over workflow
- ✅ Perfect for 48-hour hackathon
- ✅ Uses IBM Granite models

Cons:

- ❌ You build orchestration logic yourself

Option B: Full Watsonx Orchestrate

Use the Orchestrate platform to build agents with UI.

Pros:

- ✅ Visual agent builder
- ✅ Built-in CRM connectors
- ✅ Multi-channel deployment (Slack, Teams, etc.)

Cons:

- ❌ Steeper learning curve
- ❌ Requires Orchestrate subscription
- ❌ More configuration overhead

Recommendation for Hackathon: Use Option A (Watsonx.ai SDK)

You'll get faster development and full control. Build orchestration in your backend.

---

4. Key Watsonx.ai Features for Your Project

A. Text Generation/Inference

Extract structured data from meeting transcripts using Granite models.

Model to use: `ibm/granite-13b-chat-v2` or `ibm/granite-3-8b-instruct`

Input: Meeting transcript **Output:** Structured JSON with extracted fields

B. Prompt Engineering

Design prompts to extract:

- Customer name, company, role
- Pain points
- Budget/timeline
- Decision makers
- Next steps

C. LangChain Integration (Optional)

Use LangChain with watsonx.ai for:

- RAG (Retrieval Augmented Generation)
- Multi-step chains
- Structured output parsing

---

5. Authentication Setup

Environment Variables (.env)

Create a `.env` file in your backend root:

```
# IBM Watsonx.ai Credentials
WATSONX_AI_APIKEY=your_api_key_here
WATSONX_AI_PROJECT_ID=your_project_id_here
WATSONX_AI_URL=https://us-south.ml.cloud.ibm.com
WATSONX_AI_AUTH_TYPE=iam
WATSONX_AI_VERSION=2024-05-31

# Salesforce (for CRM integration)
SALESFORCE_CLIENT_ID=your_sf_client_id
SALESFORCE_CLIENT_SECRET=your_sf_secret
SALESFORCE_USERNAME=your_sf_username
SALESFORCE_PASSWORD=your_sf_password
SALESFORCE_SECURITY_TOKEN=your_sf_token

# Application
PORT=3000
NODE_ENV=development
```

⚠️ **IMPORTANT:** Add `.env` to `.gitignore`!

---

6. Installation

For Node.js Backend:

```
npm install @ibm-cloud/watsonx-ai
npm install dotenv express
npm install jsforce  # For Salesforce integration
```

For Python Backend:

```
pip install ibm-watsonx-ai
pip install python-dotenv fastapi uvicorn
pip install simple-salesforce  # For Salesforce integration
```

---

7. Quick Start Code Examples

Node.js Example (Backend API Endpoint)

```
// server.js
require('dotenv').config();
const { WatsonXAI } = require('@ibm-cloud/watsonx-ai');

// Initialize Watsonx client
const watsonxAI = WatsonXAI.newInstance({
    version: process.env.WATSONX_AI_VERSION,
    serviceUrl: process.env.WATSONX_AI_URL,
});

async function extractMeetingData(meetingText) {
    const prompt = `
You are an AI assistant that extracts structured information from sales
meeting transcripts.

Extract the following information from the meeting transcript and return
ONLY a valid JSON object:
{
  "customer_name": "full name",
  "company": "company name",
  "role": "job title",
  "pain_points": ["pain point 1", "pain point 2"],
  "budget": "mentioned budget or 'Not mentioned'",
  "timeline": "urgency/timeline or 'Not mentioned'",
  "decision_makers": ["decision maker 1"],
  "next_steps": ["action 1", "action 2"]
}

Meeting Transcript:
${meetingText}

JSON Output:`;

    const params = {
        input: prompt,
        modelId: 'ibm/granite-13b-chat-v2',
        projectId: process.env.WATSONX_AI_PROJECT_ID,
        parameters: {
            max_new_tokens: 500,
            temperature: 0.3,  // Lower for more deterministic output
        },
    };

    try {
        const response = await watsonxAI.generateText(params);
        const generatedText = response.result.results[0].generated_text;

        // Parse JSON from response
        const jsonMatch = generatedText.match(/\{[\s\S]*\}/);
        if (jsonMatch) {
            return JSON.parse(jsonMatch[0]);
        }
        throw new Error('No JSON found in response');
    } catch (error) {
        console.error('Watsonx AI Error:', error);
        throw error;
    }
}

module.exports = { extractMeetingData };
```

Python Example (FastAPI Backend)

```
# main.py
import os
from dotenv import load_dotenv
from ibm_watsonx_ai.foundation_models import Model
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
```

```python
import json
import re

load_dotenv()

# Initialize Watsonx credentials
credentials = {
    "url": os.getenv("WATSONX_AI_URL"),
    "apikey": os.getenv("WATSONX_AI_APIKEY")
}

project_id = os.getenv("WATSONX_AI_PROJECT_ID")

def extract_meeting_data(meeting_text: str) -> dict:
    prompt = f"""
You are an AI assistant that extracts structured information from sales
meeting transcripts.

Extract the following information from the meeting transcript and return
ONLY a valid JSON object:
{{
  "customer_name": "full name",
  "company": "company name",
  "role": "job title",
  "pain_points": ["pain point 1", "pain point 2"],
  "budget": "mentioned budget or 'Not mentioned'",
  "timeline": "urgency/timeline or 'Not mentioned'",
  "decision_makers": ["decision maker 1"],
  "next_steps": ["action 1", "action 2"]
}}

Meeting Transcript:
{meeting_text}

JSON Output:"""

    # Initialize model
    model = Model(
        model_id="ibm/granite-13b-chat-v2",
        params={
            GenParams.MAX_NEW_TOKENS: 500,
            GenParams.TEMPERATURE: 0.3,
        },
        credentials=credentials,
        project_id=project_id
    )

    try:
        # Generate response
        response = model.generate_text(prompt=prompt)

        # Extract JSON from response
        json_match = re.search(r'\{[\s\S]*\}', response)
        if json_match:
            return json.loads(json_match.group(0))
        raise ValueError("No JSON found in response")
    except Exception as e:
        print(f"Watsonx AI Error: {e}")
        raise
```

---

8. API Endpoint Design

POST /api/meeting/process

Request:

```
{
  "meeting_text": "Meeting notes here..."
}
```

Response:

```
{
  "success": true,
  "meeting_id": "uuid",
  "extracted_data": {
    "customer_name": "John Smith",
    "company": "Acme Corp",
    "role": "VP of Sales",
    "pain_points": ["Manual data entry", "Slow reporting"],
    "budget": "$50,000",
    "timeline": "Need solution by Q1",
    "decision_makers": ["John Smith", "Sarah Johnson (CFO)"],
    "next_steps": ["Send proposal", "Schedule demo"]
  },
  "crm_updated": true,
  "crm_record_id": "salesforce_id_123",
  "time_saved": 15
}
```

---

9. Integrating LangChain (Optional Enhancement)

If you want to use LangChain with watsonx:

Node.js:

```
npm install langchain @langchain/community
```

```
const { WatsonxAI } = require('@langchain/community/llms/watsonx_ai');

const llm = new WatsonxAI({
    apiKey: process.env.WATSONX_AI_APIKEY,
    projectId: process.env.WATSONX_AI_PROJECT_ID,
    serviceUrl: process.env.WATSONX_AI_URL,
    modelId: 'ibm/granite-13b-chat-v2',
    maxNewTokens: 500,
});

const response = await llm.call(prompt);
```

Python:

```
pip install langchain langchain-ibm
```

```
from langchain_ibm import WatsonxLLM

llm = WatsonxLLM(
    model_id="ibm/granite-13b-chat-v2",
    url=os.getenv("WATSONX_AI_URL"),
    apikey=os.getenv("WATSONX_AI_APIKEY"),
    project_id=os.getenv("WATSONX_AI_PROJECT_ID"),
    params={
        "max_new_tokens": 500,
        "temperature": 0.3,
    }
)

response = llm.invoke(prompt)
```

---

10. Salesforce Integration

Using jsforce (Node.js):

```
const jsforce = require('jsforce');

async function updateSalesforce(extractedData) {
    const conn = new jsforce.Connection({
        loginUrl: 'https://login.salesforce.com'
    });

    await conn.login(
        process.env.SALESFORCE_USERNAME,
        process.env.SALESFORCE_PASSWORD +
process.env.SALESFORCE_SECURITY_TOKEN
    );

    // Create contact
    const contact = await conn.sobject('Contact').create({
        FirstName: extractedData.customer_name.split(' ')[0],
        LastName: extractedData.customer_name.split(' ')[1],
        Title: extractedData.role,
        Company: extractedData.company,
    });

    return contact.id;
}
```

Using simple-salesforce (Python):

```
from simple_salesforce import Salesforce

def update_salesforce(extracted_data):
    sf = Salesforce(
        username=os.getenv('SALESFORCE_USERNAME'),
        password=os.getenv('SALESFORCE_PASSWORD'),
        security_token=os.getenv('SALESFORCE_SECURITY_TOKEN')
    )

    # Create contact
    contact = sf.Contact.create({
        'FirstName': extracted_data['customer_name'].split()[0],
        'LastName': extracted_data['customer_name'].split()[1],
        'Title': extracted_data['role'],
        'Company': extracted_data['company'],
    })

    return contact['id']
```

---

11. Testing Your Setup

Test Script (Node.js):

```
// test-watsonx.js
require('dotenv').config();
const { extractMeetingData } = require('./server');

const sampleMeeting = `
Meeting with John Smith from Acme Corp on Nov 21, 2025.
John is VP of Sales and mentioned their team struggles with manual CRM data
entry taking 2 hours per day.
They have a budget of $50,000 and need a solution by Q1 2026.
Decision makers include John and Sarah Johnson (CFO).
Next steps: Send proposal by Friday and schedule demo for next week.
`;

extractMeetingData(sampleMeeting)
    .then(result => {
        console.log('✅ Extraction successful!');
        console.log(JSON.stringify(result, null, 2));
    })
    .catch(error => {
        console.error('❌ Error:', error);
    });
```

Run: `node test-watsonx.js`

---

12. Troubleshooting

Common Issues:

1. Authentication Error

```
Error: Unauthorized (401)
```

Fix: Check your API key in `.env`

2. Project ID Error

```
Error: Project not found
```

Fix: Verify `WATSONX_AI_PROJECT_ID` is correct in IBM Cloud

3. Model Not Available

```
Error: Model not found
```

Fix: Use `ibm/granite-13b-chat-v2` or check available models in your project

4. SSL Certificate Errors

```
Error: self-signed certificate
```

Fix: Add to `.env`:

```
WATSONX_AI_DISABLE_SSL=true
WATSONX_AI_AUTH_DISABLE_SSL=true
```

---

13. Best Practices

Prompt Engineering Tips:

1. **Be specific:** Tell the model EXACTLY what format you want
2. **Use examples:** Provide sample outputs in your prompt
3. **Lower temperature:** Use 0.1-0.3 for deterministic extraction
4. **Validate output:** Always parse and validate JSON responses

Error Handling:

```
try {
    const data = await extractMeetingData(text);
```

```
        // Validate required fields
        if (!data.customer_name || !data.company) {
            throw new Error('Missing required fields');
        }
    } catch (error) {
        // Return graceful error to frontend
        return { success: false, error: error.message };
    }
```

---

14. Next Steps for Your Team

Abdullah (Backend):

1. ✅ Set up `.env` with credentials
2. ✅ Choose Node.js or Python
3. ✅ Implement `/api/meeting/process` endpoint
4. ✅ Test extraction with sample meeting
5. ✅ Integrate Salesforce API

Claire (AI Engineer):

1. ✅ Refine extraction prompts
2. ✅ Test different Granite models
3. ✅ Optimize parameters (temperature, tokens)
4. ✅ Handle edge cases
5. ✅ Coordinate with backend on data format

goodgame#069 (Frontend):

1. ✅ Build meeting input form
2. ✅ Show loading state during processing
3. ✅ Display extracted data beautifully
4. ✅ Dashboard for meeting history

---

15. Resources

Official Documentation:

- Watsonx.ai Python SDK
- Watsonx.ai Node.js SDK
- Watsonx Orchestrate Docs
- LangChain + Watsonx Tutorial

Example Projects:

- Watsonx Node.js Examples
- Watsonx Python Examples

Support:

- IBM Cloud Support Portal
- Watsonx Discord/Slack community
- Stack Overflow (tag: `ibm-cloud`)

---

16. Demo Checklist

Before submission, ensure:

- ✅ End-to-end flow works (input → extract → CRM → display)
- ✅ UI is polished and professional
- ✅ Error handling for bad inputs
- ✅ Time saved metric calculates correctly
- ✅ Code is on GitHub with good README
- ✅ .env.example provided (without secrets)
- ✅ Demo video clearly shows IBM Watsonx usage
- ✅ Mention IBM Granite model in demo

Good luck with your hackathon! 🚀

Questions? Add them to your team Slack/Discord channel.