Integration Guide

This guide provides detailed instructions for integrating the VMware MCP Server with Ollama (local LLM) and n8n (workflow automation).

Table of Contents

Ollama Integration

Prerequisites

1. **Install Ollama**
   ```bash
   # Linux/macOS
   curl -fsSL https://ollama.ai/install.sh | sh

# Or download from https://ollama.ai/download
```

1. **Pull Required Models**
   ```bash
   # Pull recommended model
   ollama pull llama3.2

# Alternative models
ollama pull codellama
ollama pull mistral
ollama pull gemma2
```

1. **Start Ollama Service**
   ```bash
   # Start Ollama daemon
   ollama serve

# Verify it's running
curl http://localhost:11434/api/tags
```

Configuration

1. **Environment Variables**
   ```bash
   # Enable Ollama integration
   export ENABLE_OLLAMA=true
   ```

```
# Configure Ollama endpoint
export OLLAMA_HOST=http://localhost:11434

# Set default model
export OLLAMA_MODEL=llama3.2

# Set timeout
export OLLAMA_TIMEOUT=30
```

1. **Docker Compose Setup**
   ```yaml
   # Already included in docker-compose.yml
   ollama:
   image: ollama/ollama:latest
   container_name: ollama
   ports:
   - "11434:11434"
     volumes:
   - ollama_data:/root/.ollama
     restart: unless-stopped
     ```

Usage Examples

1. **VM Performance Analysis**
   ```python
   # Using MCP tool
   result = await mcp_server.call_tool("analyze_vm_performance", {
   "vm_name": "web-server-01",
   "user_role": "operator"
   })

print(result)
# Output: AI analysis of VM performance with recommendations
```

1. **VM Sizing Recommendations**
   ```
   python
   result = await mcp_server.call_tool("suggest_vm_sizing", {
       "workload_description": "High-traffic web application with database",
       "requirements": {
           "expected_users": 10000,
           "peak_traffic": "5000 concurrent users",
           "database_size": "500GB",
           "availability": "99.9%"
       }
   })
   ```

2. **Troubleshooting Assistance**
   ```
   python
   result = await mcp_server.call_tool("troubleshoot_issue", {
       "issue_description": "VM is running slowly and consuming high CPU",
   ```

```
        "vm_name": "app-server-02"
    })
```

Custom AI Workflows

1. **Create Custom Analysis**
   ```python
   from src.ollama_integration import ollama_integration

async def custom_capacity_analysis(cluster_data):
prompt = f"""
Analyze this VMware cluster capacity data and predict:
1. When resources will be exhausted
2. Recommended scaling actions
3. Cost optimization opportunities

```
    Data: {cluster_data}
    """

    return await ollama_integration.generate_response(prompt, {
        "type": "capacity_planning",
        "data": cluster_data
    })
```

```

1. **Streaming Responses**
   ```
   python
   async def stream_troubleshooting_steps(issue):
       async for chunk in ollama_integration.stream_response(
           f"Provide step-by-step troubleshooting for: {issue}"
       ):
           print(chunk, end="", flush=True)
   ```

n8n Integration

Prerequisites

1. **Install n8n**
   ```bash
   # Using npm
   npm install n8n -g

# Using Docker
docker run -it –rm –name n8n -p 5678:5678 n8nio/n8n

# Using Docker Compose (included in project)
docker-compose up n8n
```

```

1. **Access n8n Interface**
   - Open http://localhost:5678
   - Create admin account
   - Configure basic settings
```

Configuration

1. **Environment Variables**
   ```bash
   # Enable n8n integration
   export ENABLE_N8N=true

# Configure webhook URL
export N8N_WEBHOOK_URL=http://localhost:5678/webhook

# Set API key (if using authentication)
export N8N_API_KEY=your-api-key-here
```

1. **Webhook Setup in n8n**
   ```json
   {
     "nodes": [
       {
         "parameters": {
           "httpMethod": "POST",
           "path": "vmware-events",
           "options": {}
         },
         "name": "Webhook",
         "type": "n8n-nodes-base.webhook",
         "typeVersion": 1,
         "position": [250, 300]
       }
     ]
   }
   ```

Workflow Examples

1. **VM Lifecycle Management**
   ```json
   {
     "name": "VM Lifecycle Automation",
     "nodes": [
       {
         "parameters": {
           "path": "vm-lifecycle"
         },
         "name": "VM Event Webhook",
         "type": "n8n-nodes-base.webhook"
       },
       {
         "parameters": {
           "conditions": {
             "string": [
               {
                 "value1": "={{$json[\"event_type\"]}}",
                 "value2": "vm_created"
   ```

```
              }
            ]
          }
        },
        "name": "Check Event Type",
        "type": "n8n-nodes-base.if"
      },
      {
        "parameters": {
          "to": "admin@company.com",
          "subject": "New VM Created: {{$json[\"vm_data\"][\"name\"]}}",
          "text": "A new VM has been created with the following details:\n\nName:
{{$json[\"vm_data\"][\"name\"]}}\nCPU: {{$json[\"vm_data\"][\"cpu_count\"]}}\nMemory:
{{$json[\"vm_data\"][\"memory_mb\"]}}MB"
        },
        "name": "Send Email Notification",
        "type": "n8n-nodes-base.emailSend"
      }
    ]
  }
```

2. **Performance Monitoring**

```
json
  {
    "name": "Performance Alert Workflow",
    "nodes": [
      {
        "parameters": {
          "path": "performance-alert"
        },
        "name": "Performance Webhook",
        "type": "n8n-nodes-base.webhook"
      },
      {
        "parameters": {
          "conditions": {
            "number": [
              {
                "value1": "={{$json[\"metrics\"][\"cpu_utilization_percent\"]}}",
                "operation": "larger",
                "value2": 80
              }
            ]
          }
        },
        "name": "Check CPU Threshold",
        "type": "n8n-nodes-base.if"
      },
      {
        "parameters": {
```

```
          "channel": "#alerts",
          "text": "🚨 High CPU Alert: {{$json[\"resource_name\"]}}\nCPU Usage: {{$json[\"metrics\"][\"cpu_utilization_percent\"]}}%"
        },
        "name": "Send Slack Alert",
        "type": "n8n-nodes-base.slack"
      }
    ]
  }
```

3. **Maintenance Automation**

```
json
  {
    "name": "Maintenance Window Automation",
    "nodes": [
      {
        "parameters": {
          "path": "maintenance-window"
        },
        "name": "Maintenance Webhook",
        "type": "n8n-nodes-base.webhook"
      },
      {
        "parameters": {
          "functionCode": "// Create maintenance tasks\nconst tasks = [\n  { action: 'migrate_vms', priority: 1 },\n  { action: 'enter_maintenance', priority: 2 },\n  { action: 'apply_patches', priority: 3 },\n  { action: 'exit_maintenance', priority: 4 }\n];\n\nreturn tasks.map(task => ({ json: task }));"
        },
        "name": "Generate Tasks",
        "type": "n8n-nodes-base.function"
      },
      {
        "parameters": {
          "url": "http://vmware-mcp-server:8080/api/tools/{{$json[\"action\"]}}",
          "options": {
            "headers": {
              "Authorization": "Bearer {{$env[\"MCP_API_TOKEN\"]}}"
            }
          }
        },
        "name": "Execute VMware Action",
        "type": "n8n-nodes-base.httpRequest"
      }
    ]
  }
```

Integration Usage

1. **Send Events from VMware MCP Server**
   ```python
   from src.n8n_integration import n8n_integration

# Send VM event
await n8n_integration.send_vm_event("vm_created", {
"name": "new-web-server",
"cpu_count": 4,
"memory_mb": 8192,
"created_by": "admin"
})

# Send performance alert
await n8n_integration.send_alert(
"high_cpu_usage",
"warning",
"CPU usage exceeded 80%",
{"vm_name": "web-server-01", "cpu_percent": 85}
)
```

1. **Custom Webhook Endpoints**
   ```
   python
   # Send to custom webhook
   await n8n_integration.send_custom_webhook("backup-status", {
       "backup_type": "vm_snapshot",
       "status": "completed",
       "vm_list": ["web-01", "db-01", "app-01"],
       "duration": "45 minutes"
   })
   ```

Combined Workflows

AI-Powered Automation

1. **Intelligent Performance Optimization**
   ```python
   async def intelligent_performance_workflow(vm_name):
   # Get VM performance data
   vm_data = await vm_ops.get_vm_resource_usage(vm_name)

   # Get AI analysis
   ai_analysis = await ollama_integration.analyze_vm_performance(vm_data)

   # Send to n8n for automated actions
   await n8n_integration.trigger_workflow("performance_optimization", {
   "vm_name": vm_name,
   "performance_data": vm_data,
   "ai_recommendations": ai_analysis,
   "automation_level": "supervised" # or "automatic"
   ```

```
})
```

2. **Predictive Maintenance**

```python
async def predictive_maintenance_workflow():
# Get cluster resources
cluster_data = await resource_ops.get_cluster_resources()

# AI analysis for capacity planning
capacity_analysis = await ollama_integration.generate_response(
“Analyze cluster capacity and predict maintenance needs”,
{“cluster_data”: cluster_data}
)

# Trigger maintenance planning workflow
await n8n_integration.send_maintenance_notification(
“predictive_maintenance”,
[“cluster-01”],
{
“predicted_date”: “2024-02-15”,
“ai_analysis”: capacity_analysis,
“confidence”: “high”
}
)
```

Example n8n Workflow with AI Integration

```
{
  "name": "AI-Powered VM Optimization",
  "nodes": [
    {
      "parameters": {
        "path": "ai-optimization"
      },
      "name": "Optimization Webhook",
      "type": "n8n-nodes-base.webhook"
    },
    {
      "parameters": {
        "url": "http://ollama:11434/api/generate",
        "options": {
          "headers": {
            "Content-Type": "application/json"
          },
          "body": {
            "model": "llama3.2",
            "prompt": "Analyze VM performance: {{$json[\"vm_data\"]}} and suggest op-
timizations",
            "stream": false
          }
        }
      },
      "name": "Get AI Analysis",
      "type": "n8n-nodes-base.httpRequest"
    },
    {
      "parameters": {
        "functionCode": "// Parse AI response and create action plan\nconst aiResponse
= JSON.parse($input.first().json.body).response;\nconst actions = [];\n\nif (aiRe-
sponse.includes('increase memory')) {\n  actions.push({type: 'modify_resources',
memory_increase: true});\n}\n\nif (aiResponse.includes('reduce CPU')) {\n  ac-
tions.push({type: 'modify_resources', cpu_reduce: true});\n}\n\nreturn
actions.map(action => ({json: action}));"
      },
      "name": "Parse AI Recommendations",
      "type": "n8n-nodes-base.function"
    },
    {
      "parameters": {
        "url": "http://vmware-mcp-server:8080/api/tools/modify_vm_resources",
        "options": {
          "headers": {
            "Authorization": "Bearer {{$env[\"MCP_TOKEN\"]}}"
          },
          "body": {
            "vm_name": "{{$json[\"vm_name\"]}}",
            "cpu_count": "{{$json[\"new_cpu_count\"]}}",
            "memory_mb": "{{$json[\"new_memory_mb\"]}}"
          }
        }
      },
      "name": "Apply Optimizations",
      "type": "n8n-nodes-base.httpRequest"
    }
  ]
}
```

Troubleshooting

Common Issues

1. **Ollama Connection Issues**
   ```bash
   # Check if Ollama is running
   curl http://localhost:11434/api/tags

# Check logs
docker logs ollama

# Restart service
docker-compose restart ollama
```

1. **n8n Webhook Issues**
   ```bash
   # Test webhook endpoint
   curl -X POST http://localhost:5678/webhook/test \
   -H "Content-Type: application/json" \
   -d '{"test": "data"}'

# Check n8n logs
docker logs n8n
```

1. **Integration Health Checks**
   ```python
   # Check integration status
   ollama_status = await ollama_integration.health_check()
   n8n_status = await n8n_integration.health_check()

print(f"Ollama: {ollama_status}")
print(f"n8n: {n8n_status}")
```

Performance Optimization

1. **Ollama Performance**
   ```bash
   # Use GPU acceleration (if available)
   docker run --gpus all ollama/ollama

# Optimize model loading
export OLLAMA_NUM_PARALLEL=4
export OLLAMA_MAX_LOADED_MODELS=2
```

1. **n8n Performance**
   ```bash
   # Increase worker processes
   export N8N_WORKERS=4

# Configure database for better performance
export DB_TYPE=postgresdb
   ```

export DB_POSTGRESDB_HOST=postgres
```

### Monitoring and Logging

1. **Enable Debug Logging**
   ```bash
   export LOG_LEVEL=DEBUG
   export ENABLE_AUDIT_LOG=true
   ```

2. **Monitor Integration Metrics**
   ```python
   # Custom metrics collection
   from prometheus_client import Counter, Histogram

ollama_requests = Counter('ollama_requests_total', 'Total Ollama requests')
n8n_webhooks = Counter('n8n_webhooks_total', 'Total n8n webhooks')
```

Best Practices

1. **Security**
   - Use API keys for n8n authentication
   - Implement rate limiting for AI requests
   - Validate all webhook payloads

2. **Performance**
   - Cache AI responses for similar queries
   - Use async operations for all integrations
   - Implement circuit breakers for external services

3. **Reliability**
   - Implement retry logic with exponential backoff
   - Use health checks to monitor service availability
   - Set appropriate timeouts for all operations

4. **Monitoring**
   - Track integration usage metrics
   - Monitor response times and error rates
   - Set up alerts for service failures

For additional support, refer to the main documentation or contact the development team.
```