Integration Guide

This guide covers integrating the Proxmox MCP Server with various external systems, AI models, workflow automation, and monitoring solutions.

🤖 AI Integration with Ollama

Setup and Configuration

Installing Ollama

```
# Install Ollama
curl -fsSL https://ollama.ai/install.sh | sh

# Start Ollama service
systemctl start ollama
systemctl enable ollama

# Pull recommended models
ollama pull llama2
ollama pull codellama
ollama pull mistral
```

Model Selection

Model	Use Case	Memory Requirements	Performance
`llama2`	General analysis and recommendations	4GB	Good
`llama2:13b`	Advanced analysis with better context	8GB	Better
`codellama`	Code analysis and troubleshooting	4GB	Specialized
`mistral`	Fast responses, good for alerts	2GB	Fast
`dolphin-mixtral`	Complex reasoning tasks	16GB	Excellent

Configuration

```
# config/.env
OLLAMA_HOST=http://localhost:11434
OLLAMA_MODEL=llama2
OLLAMA_TIMEOUT=60
ENABLE_AI_FEATURES=true

# Advanced configuration
OLLAMA_TEMPERATURE=0.7
OLLAMA_TOP_P=0.9
OLLAMA_MAX_TOKENS=1000
```

AI-Powered Features

Performance Analysis

```
from src.ai.advisor import ai_advisor

# Analyze VM performance
vm_data = {
    'vmid': 100,
    'name': 'web-server-01',
    'node': 'pve1',
    'cpu': 0.85,
    'maxcpu': 4,
    'mem': 3221225472,  # 3GB
    'maxmem': 4294967296,  # 4GB
    'uptime': 86400,  # 1 day
    'status': 'running'
}

analysis = await ai_advisor.analyze_vm_performance(vm_data)
print(f"Performance Status: {analysis['performance_status']}")
print(f"AI Recommendations: {analysis['ai_recommendations']}")
```

Resource Sizing

```
# Get sizing recommendations
sizing = await ai_advisor.suggest_resource_sizing(
    workload_type='web_server',
    requirements={
        'expected_concurrent_users': 500,
        'peak_requests_per_hour': 10000,
        'database_size': '50GB',
        'framework': 'django',
        'caching': 'redis',
        'ssl_termination': 'nginx'
    }
)

print(f"Recommended CPU: {sizing['base_recommendations']['cpu_cores']} cores")
print(f"Recommended Memory: {sizing['base_recommendations']['memory_gb']} GB")
```

Troubleshooting Assistant

```
# Get troubleshooting help
issue = "VM is experiencing high CPU usage and slow response times"
system_data = {
    'vm_id': 100,
    'current_metrics': {
        'cpu_usage': 95,
        'memory_usage': 78,
        'disk_io': 'high',
        'network_io': 'normal'
    },
    'recent_changes': [
        'Increased traffic by 200%',
        'Added new application module'
    ],
    'error_logs': [
        'Database connection timeout',
        'Memory allocation failed'
    ]
}

troubleshooting = await ai_advisor.troubleshoot_issue(issue, system_data)
print(f"Likely Causes: {troubleshooting['ai_troubleshooting']}")
```

Custom AI Prompts

Creating Custom Analysis Templates

```
# src/ai/custom_prompts.py
CUSTOM_PROMPTS = {
    'security_analysis': """
    As a Proxmox security expert, analyze the following system configuration:

    Configuration: {config}
    Security Events: {security_events}

    Provide:
    1. Security risk assessment
    2. Vulnerability identification
    3. Hardening recommendations
    4. Compliance considerations
    """,

    'capacity_planning': """
    As a Proxmox capacity planning expert, analyze the following usage trends:

    Historical Data: {historical_data}
    Growth Projections: {growth_projections}

    Provide:
    1. Resource utilization forecast
    2. Scaling recommendations
    3. Cost optimization suggestions
    4. Timeline for capacity expansion
    """
}

# Usage
async def analyze_security(config_data, security_events):
    prompt = CUSTOM_PROMPTS['security_analysis'].format(
        config=config_data,
        security_events=security_events
    )
    return await ai_advisor._query_ollama(prompt)
```

🔄 n8n Workflow Automation

Setup and Configuration

n8n Installation

```
# Using Docker
docker run -it --rm \
  --name n8n \
  -p 5678:5678 \
  -e N8N_BASIC_AUTH_ACTIVE=true \
  -e N8N_BASIC_AUTH_USER=admin \
  -e N8N_BASIC_AUTH_PASSWORD=admin123 \
  -v n8n_data:/home/node/.n8n \
  n8nio/n8n

# Using npm
npm install n8n -g
n8n start
```

Webhook Configuration

```
# config/.env
N8N_HOST=http://localhost:5678
N8N_WEBHOOK_URL=http://localhost:5678/webhook
ENABLE_N8N_INTEGRATION=true

# n8n environment
N8N_BASIC_AUTH_ACTIVE=true
N8N_BASIC_AUTH_USER=admin
N8N_BASIC_AUTH_PASSWORD=secure-password
WEBHOOK_URL=http://n8n:5678/
```

Workflow Examples

VM Lifecycle Automation

```
{
  "name": "VM Provisioning Workflow",
  "nodes": [
    {
      "parameters": {
        "httpMethod": "POST",
        "path": "vm-provision",
        "responseMode": "responseNode"
      },
      "name": "VM Provision Webhook",
      "type": "n8n-nodes-base.webhook"
    },
    {
      "parameters": {
        "url": "http://proxmox-mcp-server:8080/api/v1/vms",
        "authentication": "genericCredentialType",
        "genericAuthType": "httpHeaderAuth",
        "sendBody": true,
        "bodyParameters": {
          "parameters": [
            {
              "name": "node",
              "value": "={{$json.node}}"
            },
            {
              "name": "vmid",
              "value": "={{$json.vmid}}"
            },
            {
              "name": "config",
              "value": "={{$json.config}}"
            }
          ]
        }
      },
      "name": "Create VM",
      "type": "n8n-nodes-base.httpRequest"
    },
    {
      "parameters": {
        "amount": 30,
        "unit": "seconds"
      },
      "name": "Wait for VM Creation",
      "type": "n8n-nodes-base.wait"
    },
    {
      "parameters": {
        "url": "http://proxmox-mcp-server:8080/api/v1/vms/{{$json.vmid}}/start",
        "authentication": "genericCredentialType",
        "genericAuthType": "httpHeaderAuth",
        "sendBody": true
      },
      "name": "Start VM",
      "type": "n8n-nodes-base.httpRequest"
    }
  ]
}
```

Performance Monitoring Workflow

```
{
  "name": "Performance Alert Workflow",
  "nodes": [
    {
      "parameters": {
        "rule": {
          "interval": [
            {
              "field": "cronExpression",
              "expression": "*/5 * * * *"
            }
          ]
        }
      },
      "name": "Every 5 Minutes",
      "type": "n8n-nodes-base.cron"
    },
    {
      "parameters": {
        "url": "http://proxmox-mcp-server:8080/api/v1/cluster/resources",
        "authentication": "genericCredentialType"
      },
      "name": "Get Cluster Resources",
      "type": "n8n-nodes-base.httpRequest"
    },
    {
      "parameters": {
        "functionCode": "// Check for high resource usage\nconst resources = $json;
\nconst alerts = [];\n\nfor (const vm of resources.vms || []) {\n  const cpuUsage =
(vm.cpu / vm.maxcpu) * 100;\n  const memUsage = (vm.mem / vm.maxmem) * 100;\n  \n  if
(cpuUsage > 90 || memUsage > 90) {\n    alerts.push({\n      type: 'high_usage',\n
vmid: vm.vmid,\n      name: vm.name,\n      cpu_usage: cpuUsage,\n      mem_usage:
memUsage,\n      severity: cpuUsage > 95 || memUsage > 95 ? 'critical' : 'warn-
ing'\n    });\n  }\n}\n\nreturn alerts.map(alert => ({ json: alert }));"
      },
      "name": "Analyze Performance",
      "type": "n8n-nodes-base.function"
    },
    {
      "parameters": {
        "conditions": {
          "string": [
            {
              "value1": "={{$json.severity}}",
              "operation": "equal",
              "value2": "critical"
            }
          ]
        }
      },
      "name": "Critical Alert?",
      "type": "n8n-nodes-base.if"
    },
    {
      "parameters": {
        "message": "🚨 CRITICAL: VM {{$json.name}} ({{$json.vmid}}) has high resource
usage!\n\nCPU: {{$json.cpu_usage}}%\nMemory: {{$json.mem_usage}}%\n\nImmediate action
required!",
        "chatId": "@proxmox_alerts"
      },
      "name": "Send Critical Alert",
      "type": "n8n-nodes-base.telegram"
```

```
    }
  ]
}
```

Custom Workflow Integration

Creating Custom Webhooks

```python
# src/n8n_integration.py
from .n8n_integration import n8n_integration

async def trigger_custom_workflow(workflow_name: str, data: dict):
    """Trigger a custom n8n workflow."""
    return await n8n_integration.trigger_webhook(workflow_name, {
        'custom_data': data,
        'timestamp': datetime.now().isoformat(),
        'source': 'proxmox-mcp-custom'
    })

# Usage examples
await trigger_custom_workflow('vm-backup-failed', {
    'vmid': 100,
    'error': 'Storage full',
    'node': 'pve1'
})

await trigger_custom_workflow('maintenance-window', {
    'start_time': '2024-01-15T02:00:00Z',
    'duration': '4 hours',
    'affected_nodes': ['pve1', 'pve2']
})
```

Workflow Templates

```
# Backup automation workflow
BACKUP_WORKFLOW = {
    "name": "Automated Backup",
    "trigger": "schedule",
    "schedule": "0 2 * * *",  # Daily at 2 AM
    "steps": [
        {
            "type": "get_vms",
            "filter": {"status": "running", "template": False}
        },
        {
            "type": "create_backup",
            "config": {
                "storage": "backup-storage",
                "mode": "snapshot",
                "compress": "zstd"
            }
        },
        {
            "type": "cleanup_old_backups",
            "retention": {"days": 30, "count": 7}
        },
        {
            "type": "notify",
            "channels": ["telegram", "email"]
        }
    ]
}

# Maintenance workflow
MAINTENANCE_WORKFLOW = {
    "name": "Node Maintenance",
    "trigger": "webhook",
    "steps": [
        {
            "type": "migrate_vms",
            "source_node": "{{webhook.node}}",
            "strategy": "load_balance"
        },
        {
            "type": "wait",
            "duration": "5m"
        },
        {
            "type": "update_node",
            "node": "{{webhook.node}}"
        },
        {
            "type": "reboot_node",
            "node": "{{webhook.node}}"
        },
        {
            "type": "wait_for_node",
            "timeout": "10m"
        },
        {
            "type": "migrate_vms_back",
            "target_node": "{{webhook.node}}"
        }
    ]
}
```

📊 Monitoring Integration

Prometheus Integration

Metrics Configuration

```python
# src/monitoring/metrics.py
from prometheus_client import Counter, Histogram, Gauge, CollectorRegistry

# Custom metrics registry
registry = CollectorRegistry()

# API metrics
api_requests_total = Counter(
    'proxmox_mcp_api_requests_total',
    'Total API requests',
    ['method', 'endpoint', 'status'],
    registry=registry
)

api_request_duration = Histogram(
    'proxmox_mcp_api_request_duration_seconds',
    'API request duration',
    ['method', 'endpoint'],
    registry=registry
)

# Proxmox metrics
proxmox_vms_total = Gauge(
    'proxmox_vms_total',
    'Total number of VMs',
    ['node', 'status'],
    registry=registry
)

proxmox_vm_cpu_usage = Gauge(
    'proxmox_vm_cpu_usage_percent',
    'VM CPU usage percentage',
    ['vmid', 'name', 'node'],
    registry=registry
)

proxmox_vm_memory_usage = Gauge(
    'proxmox_vm_memory_usage_bytes',
    'VM memory usage in bytes',
    ['vmid', 'name', 'node'],
    registry=registry
)

# AI metrics
ai_analysis_duration = Histogram(
    'proxmox_mcp_ai_analysis_duration_seconds',
    'AI analysis duration',
    ['analysis_type'],
    registry=registry
)

ai_recommendations_total = Counter(
    'proxmox_mcp_ai_recommendations_total',
    'Total AI recommendations generated',
    ['recommendation_type'],
    registry=registry
)
```

Custom Collectors

```python
# src/monitoring/collectors.py
from prometheus_client import CollectorRegistry
from .proxmox_client import proxmox_client

class ProxmoxCollector:
    """Custom Prometheus collector for Proxmox metrics."""

    def collect(self):
        """Collect metrics from Proxmox."""
        try:
            # Collect VM metrics
            vms = await proxmox_client.get_vms()
            for vm in vms:
                yield GaugeMetricFamily(
                    'proxmox_vm_cpu_usage_percent',
                    'VM CPU usage percentage',
                    labels=['vmid', 'name', 'node'],
                    value=vm.get('cpu', 0) / vm.get('maxcpu', 1) * 100
                )

                yield GaugeMetricFamily(
                    'proxmox_vm_memory_usage_bytes',
                    'VM memory usage in bytes',
                    labels=['vmid', 'name', 'node'],
                    value=vm.get('mem', 0)
                )

            # Collect cluster metrics
            cluster_status = await proxmox_client.get_cluster_status()
            yield GaugeMetricFamily(
                'proxmox_cluster_quorate',
                'Cluster quorum status',
                value=1 if cluster_status.get('quorate') else 0
            )

        except Exception as e:
            logger.error(f"Error collecting Proxmox metrics: {e}")

# Register collector
registry.register(ProxmoxCollector())
```

Grafana Dashboards

VM Performance Dashboard

```
{
  "dashboard": {
    "title": "Proxmox VM Performance",
    "panels": [
      {
        "title": "VM CPU Usage",
        "type": "graph",
        "targets": [
          {
            "expr": "proxmox_vm_cpu_usage_percent",
            "legendFormat": "{{name}} ({{vmid}})"
          }
        ],
        "yAxes": [
          {
            "min": 0,
            "max": 100,
            "unit": "percent"
          }
        ]
      },
      {
        "title": "VM Memory Usage",
        "type": "graph",
        "targets": [
          {
            "expr": "proxmox_vm_memory_usage_bytes / 1024 / 1024 / 1024",
            "legendFormat": "{{name}} ({{vmid}})"
          }
        ],
        "yAxes": [
          {
            "min": 0,
            "unit": "bytes"
          }
        ]
      },
      {
        "title": "Top CPU Consumers",
        "type": "table",
        "targets": [
          {
            "expr": "topk(10, proxmox_vm_cpu_usage_percent)",
            "format": "table"
          }
        ]
      }
    ]
  }
}
```

Cluster Overview Dashboard

```
{
  "dashboard": {
    "title": "Proxmox Cluster Overview",
    "panels": [
      {
        "title": "Cluster Status",
        "type": "singlestat",
        "targets": [
          {
            "expr": "proxmox_cluster_quorate",
            "legendFormat": "Quorate"
          }
        ],
        "valueMaps": [
          {"value": "1", "text": "HEALTHY"},
          {"value": "0", "text": "UNHEALTHY"}
        ]
      },
      {
        "title": "Node Status",
        "type": "table",
        "targets": [
          {
            "expr": "proxmox_node_online",
            "format": "table"
          }
        ]
      },
      {
        "title": "Resource Utilization",
        "type": "graph",
        "targets": [
          {
            "expr": "avg(proxmox_node_cpu_usage_percent) by (node)",
            "legendFormat": "CPU - {{node}}"
          },
          {
            "expr": "avg(proxmox_node_memory_usage_percent) by (node)",
            "legendFormat": "Memory - {{node}}"
          }
        ]
      }
    ]
  }
}
```

Alerting Rules

Prometheus Alerting

```yaml
# alerts/proxmox_alerts.yml
groups:
  - name: proxmox.rules
    rules:
      - alert: ProxmoxVMHighCPU
        expr: proxmox_vm_cpu_usage_percent > 90
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High CPU usage on VM {{ $labels.name }}"
          description: "VM {{ $labels.name }} ({{ $labels.vmid }}) on node {{ $la-
bels.node }} has CPU usage above 90% for more than 5 minutes."

      - alert: ProxmoxVMHighMemory
        expr: proxmox_vm_memory_usage_percent > 95
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "High memory usage on VM {{ $labels.name }}"
          description: "VM {{ $labels.name }} ({{ $labels.vmid }}) on node {{ $la-
bels.node }} has memory usage above 95%."

      - alert: ProxmoxNodeDown
        expr: proxmox_node_online == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Proxmox node {{ $labels.node }} is down"
          description: "Node {{ $labels.node }} has been down for more than 1 minute."

      - alert: ProxmoxClusterNotQuorate
        expr: proxmox_cluster_quorate == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Proxmox cluster has lost quorum"
          description: "The Proxmox cluster is not quorate and may not be able to per-
form operations."

      - alert: ProxmoxStorageFull
        expr: proxmox_storage_usage_percent > 90
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Proxmox storage {{ $labels.storage }} is nearly full"
          description: "Storage {{ $labels.storage }} on node {{ $labels.node }} is {{
$value }}% full."
```

🔗 External System Integration

LDAP/Active Directory Integration

```python
# src/auth/ldap_auth.py
import ldap
from ldap import LDAPError

class LDAPAuthenticator:
    def __init__(self, server_uri, bind_dn, bind_password, user_base_dn):
        self.server_uri = server_uri
        self.bind_dn = bind_dn
        self.bind_password = bind_password
        self.user_base_dn = user_base_dn

    async def authenticate(self, username, password):
        """Authenticate user against LDAP."""
        try:
            conn = ldap.initialize(self.server_uri)
            conn.simple_bind_s(self.bind_dn, self.bind_password)

            # Search for user
            search_filter = f"(sAMAccountName={username})"
            result = conn.search_s(
                self.user_base_dn,
                ldap.SCOPE_SUBTREE,
                search_filter,
                ['cn', 'memberOf']
            )

            if not result:
                return None

            user_dn = result[0][0]
            user_attrs = result[0][1]

            # Authenticate user
            user_conn = ldap.initialize(self.server_uri)
            user_conn.simple_bind_s(user_dn, password)

            # Get user groups
            groups = user_attrs.get('memberOf', [])
            role = self._map_groups_to_role(groups)

            return {
                'username': username,
                'role': role,
                'groups': groups
            }

        except LDAPError as e:
            logger.error(f"LDAP authentication failed: {e}")
            return None

    def _map_groups_to_role(self, groups):
        """Map LDAP groups to application roles."""
        group_role_mapping = {
            'CN=Proxmox-Admins,OU=Groups,DC=company,DC=com': 'admin',
            'CN=Proxmox-Operators,OU=Groups,DC=company,DC=com': 'operator',
            'CN=Proxmox-Viewers,OU=Groups,DC=company,DC=com': 'viewer'
        }

        for group in groups:
            if isinstance(group, bytes):
                group = group.decode('utf-8')
            if group in group_role_mapping:
```

```
                return group_role_mapping[group]

        return 'viewer'  # Default role
```

Slack Integration

```python
# src/integrations/slack.py
import httpx
from typing import Dict, Any

class SlackIntegration:
    def __init__(self, webhook_url: str, channel: str = None):
        self.webhook_url = webhook_url
        self.channel = channel

    async def send_alert(self, alert_type: str, message: str, data: Dict[str, Any] = No
ne):
        """Send alert to Slack."""
        color_map = {
            'critical': '#FF0000',
            'warning': '#FFA500',
            'info': '#00FF00'
        }

        attachment = {
            'color': color_map.get(alert_type, '#808080'),
            'title': f'Proxmox MCP Alert - {alert_type.upper()}',
            'text': message,
            'fields': [],
            'timestamp': int(datetime.now().timestamp())
        }

        if data:
            for key, value in data.items():
                attachment['fields'].append({
                    'title': key.replace('_', ' ').title(),
                    'value': str(value),
                    'short': True
                })

        payload = {
            'attachments': [attachment]
        }

        if self.channel:
            payload['channel'] = self.channel

        async with httpx.AsyncClient() as client:
            response = await client.post(self.webhook_url, json=payload)
            return response.status_code == 200

# Usage
slack = SlackIntegration(
    webhook_url='https://hooks.slack.com/services/YOUR/WEBHOOK/URL',
    channel='#proxmox-alerts'
)

await slack.send_alert('critical', 'VM 100 has high CPU usage', {
    'vm_id': 100,
    'cpu_usage': '95%',
    'node': 'pve1'
})
```

Email Integration

```python
# src/integrations/email.py
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders

class EmailIntegration:
    def __init__(self, smtp_server: str, smtp_port: int, username: str, password: str):
        self.smtp_server = smtp_server
        self.smtp_port = smtp_port
        self.username = username
        self.password = password

    async def send_alert_email(self, to_addresses: list, subject: str, body: str,
                               alert_type: str = 'info', attachments: list = None):
        """Send alert email."""
        msg = MIMEMultipart()
        msg['From'] = self.username
        msg['To'] = ', '.join(to_addresses)
        msg['Subject'] = f'[Proxmox MCP - {alert_type.upper()}] {subject}'

        # HTML body with styling
        html_body = f"""
        <html>
        <body>
            <div style="font-family: Arial, sans-serif;">
                <div style="background-color: {'#ffebee' if alert_type == 'critical' el
se '#fff3e0' if alert_type == 'warning' else '#e8f5e8'};
                            padding: 20px; border-radius: 5px; margin-bottom: 20px;">
                    <h2 style="color: {'#c62828' if alert_type == 'critical' else '#ef6
c00' if alert_type == 'warning' else '#2e7d32'};">
                        Proxmox MCP Alert - {alert_type.upper()}
                    </h2>
                </div>
                <div style="padding: 20px;">
                    {body.replace('\n', '<br>')}
                </div>
                <div style="margin-top: 20px; padding: 10px; background-color: #f5f5f5;
border-radius: 3px;">
                    <small>Generated by Proxmox MCP Server at
{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</small>
                </div>
            </div>
        </body>
        </html>
        """

        msg.attach(MIMEText(html_body, 'html'))

        # Add attachments
        if attachments:
            for file_path in attachments:
                with open(file_path, 'rb') as attachment:
                    part = MIMEBase('application', 'octet-stream')
                    part.set_payload(attachment.read())
                    encoders.encode_base64(part)
                    part.add_header(
                        'Content-Disposition',
                        f'attachment; filename= {file_path.split("/")[-1]}'
                    )
                    msg.attach(part)
```

```
        # Send email
        try:
            server = smtplib.SMTP(self.smtp_server, self.smtp_port)
            server.starttls()
            server.login(self.username, self.password)
            server.send_message(msg)
            server.quit()
            return True
        except Exception as e:
            logger.error(f"Failed to send email: {e}")
            return False
```

Webhook Integration

```python
# src/integrations/webhooks.py
import httpx
import hmac
import hashlib
from typing import Dict, Any, Optional

class WebhookIntegration:
    def __init__(self, webhook_url: str, secret: Optional[str] = None):
        self.webhook_url = webhook_url
        self.secret = secret

    async def send_webhook(self, event_type: str, data: Dict[str, Any],
                           headers: Dict[str, str] = None) -> bool:
        """Send webhook with optional signature verification."""
        payload = {
            'event_type': event_type,
            'timestamp': datetime.now().isoformat(),
            'data': data
        }

        request_headers = {
            'Content-Type': 'application/json',
            'User-Agent': 'Proxmox-MCP-Server/1.0'
        }

        if headers:
            request_headers.update(headers)

        # Add signature if secret is provided
        if self.secret:
            payload_json = json.dumps(payload, sort_keys=True)
            signature = hmac.new(
                self.secret.encode(),
                payload_json.encode(),
                hashlib.sha256
            ).hexdigest()
            request_headers['X-Signature-SHA256'] = f'sha256={signature}'

        try:
            async with httpx.AsyncClient(timeout=30.0) as client:
                response = await client.post(
                    self.webhook_url,
                    json=payload,
                    headers=request_headers
                )
                return response.status_code < 400
        except Exception as e:
            logger.error(f"Webhook delivery failed: {e}")
            return False

# Usage
webhook = WebhookIntegration(
    webhook_url='https://your-system.com/webhooks/proxmox',
    secret='your-webhook-secret'
)

await webhook.send_webhook('vm_created', {
    'vmid': 100,
    'name': 'web-server-01',
    'node': 'pve1'
})
```

📋 Integration Checklist

AI Integration

- [ ] Ollama installed and configured
- [ ] Models downloaded and tested
- [ ] AI features enabled in configuration
- [ ] Custom prompts configured
- [ ] Performance monitoring for AI requests

Workflow Automation

- [ ] n8n installed and accessible
- [ ] Webhook endpoints configured
- [ ] Basic workflows imported and tested
- [ ] Custom workflows created
- [ ] Workflow monitoring enabled

Monitoring Integration

- [ ] Prometheus configured and scraping metrics
- [ ] Grafana dashboards imported
- [ ] Alerting rules configured
- [ ] Alert channels tested
- [ ] Custom metrics implemented

External Systems

- [ ] Authentication systems integrated (LDAP/AD)
- [ ] Notification channels configured (Slack/Email)
- [ ] Webhook endpoints tested
- [ ] API integrations documented
- [ ] Error handling implemented

This integration guide provides comprehensive coverage of connecting the Proxmox MCP Server with various external systems. Each integration includes practical examples and configuration details for immediate implementation.