MCP Server Integration Guide

🌐 Overview

ContextFlow AI includes a comprehensive MCP (Model Context Protocol) server that enables seamless integration with modern IDEs and development environments. This headless interface allows developers to access all orchestration functionality directly from their preferred development tools.

🚀 Quick Start

Starting the MCP Server

```
# Start MCP server (default port 8002)
python -m mcp_server.main

# Or with custom configuration
MCP_SERVER_PORT=8003 MCP_API_TOKEN=your-token python -m mcp_server.main
```

Basic Client Usage

```
from mcp_server.client import MCPClient, MCPConfig

# Configure client
config = MCPConfig(
    host="localhost",
    port=8002,
    token="your-api-token"
)

# Use client
async with MCPClient(config) as client:
    # Deploy a repository
    workflow = await client.create_workflow(
        repository_url="https://github.com/user/app.git",
        task_prompt="Deploy secure web application"
    )

    print(f"Workflow started: {workflow['workflow_id']}")
```

🔧 IDE Integrations

Cursor Integration

Setup

1. Install the ContextFlow AI extension from Cursor marketplace
2. Configure MCP server endpoint in settings
3. Authenticate with your API token

Usage Examples

```python
# cursor_integration.py
import asyncio
from mcp_server.client import MCPClient, MCPConfig

async def cursor_deploy():
    """Deploy current project from Cursor"""
    config = MCPConfig(
        host="localhost",
        port=8002,
        token="your-token"
    )

    async with MCPClient(config) as client:
        # Get current project repository
        repo_url = get_current_repo_url()  # Cursor API

        # Deploy with AI assistance
        workflow = await client.create_workflow(
            repository_url=repo_url,
            task_prompt="Deploy with SSL, monitoring, and auto-scaling"
        )

        # Stream progress in Cursor terminal
        async for progress in client.stream_workflow_progress(workflow["workflow_id"]):
            print_to_cursor_terminal(progress)

            if progress.get("status") == "completed":
                show_cursor_notification("Deployment completed successfully!")
                break

# Run from Cursor command palette
if __name__ == "__main__":
    asyncio.run(cursor_deploy())
```

Cursor Commands

- `ContextFlow: Deploy Current Project` - Deploy current repository
- `ContextFlow: Monitor Deployments` - View active deployments
- `ContextFlow: View Logs` - Stream deployment logs
- `ContextFlow: SSH Execute` - Run commands on deployment target

DeepAgent Integration

Setup

```javascript
// deepagent-config.js
const { MCPClient } = require('@contextflow/mcp-client');

const client = new MCPClient({
    host: 'localhost',
    port: 8002,
    token: process.env.CONTEXTFLOW_TOKEN
});

module.exports = { client };
```

Usage Examples

```
// deepagent-plugin.js
const { client } = require('./deepagent-config');

class ContextFlowPlugin {
    async deployProject(projectPath, options = {}) {
        try {
            const repoUrl = await this.getRepositoryUrl(projectPath);

            const workflow = await client.createWorkflow({
                repositoryUrl: repoUrl,
                taskPrompt: options.prompt || "Deploy application with best practices",
                securityLevel: options.securityLevel || "medium"
            });

            // Show progress in DeepAgent UI
            this.showProgressPanel(workflow.workflowId);

            return workflow;
        } catch (error) {
            this.showError(`Deployment failed: ${error.message}`);
        }
    }

    async monitorDeployment(workflowId) {
        const progressStream = client.streamProgress(workflowId);

        for await (const progress of progressStream) {
            this.updateProgressPanel(progress);

            if (progress.status === 'completed') {
                this.showSuccess('Deployment completed successfully!');
                break;
            }
        }
    }
}

module.exports = ContextFlowPlugin;
```

VS Code Integration

Extension Setup

```
// package.json (VS Code extension)
{
    "name": "contextflow-ai",
    "displayName": "ContextFlow AI",
    "description": "AI-powered DevOps orchestration",
    "version": "1.0.0",
    "engines": {
        "vscode": "^1.74.0"
    },
    "categories": ["Other"],
    "activationEvents": [
        "onCommand:contextflow.deploy",
        "onCommand:contextflow.monitor"
    ],
    "main": "./out/extension.js",
    "contributes": {
        "commands": [
            {
                "command": "contextflow.deploy",
                "title": "Deploy with ContextFlow AI"
            },
            {
                "command": "contextflow.monitor",
                "title": "Monitor Deployments"
            }
        ],
        "configuration": {
            "title": "ContextFlow AI",
            "properties": {
                "contextflow.serverUrl": {
                    "type": "string",
                    "default": "http://localhost:8002",
                    "description": "MCP server URL"
                },
                "contextflow.apiToken": {
                    "type": "string",
                    "description": "API token for authentication"
                }
            }
        }
    }
}
```

```
// extension.ts
import * as vscode from 'vscode';
import { MCPClient, MCPConfig } from '@contextflow/mcp-client';

export function activate(context: vscode.ExtensionContext) {
    const deployCommand = vscode.commands.registerCommand('contextflow.deploy', async (
) => {
        const config = vscode.workspace.getConfiguration('contextflow');
        const client = new MCPClient({
            host: config.get('serverUrl', 'http://localhost:8002'),
            token: config.get('apiToken', '')
        });

        try {
            const workspaceFolder = vscode.workspace.workspaceFolders?.[0];
            if (!workspaceFolder) {
                vscode.window.showErrorMessage('No workspace folder found');
                return;
            }

            const repoUrl = await getRepositoryUrl(workspaceFolder.uri.fsPath);
            const prompt = await vscode.window.showInputBox({
                prompt: 'Describe your deployment requirements',
                value: 'Deploy secure web application with monitoring'
            });

            if (!prompt) return;

            const workflow = await client.createWorkflow({
                repositoryUrl: repoUrl,
                taskPrompt: prompt
            });

            vscode.window.showInformationMessage(
                `Deployment started: ${workflow.workflowId}`
            );

            // Show progress in output channel
            const outputChannel = vscode.window.createOutputChannel('ContextFlow AI');
            outputChannel.show();

            for await (const progress of client.streamProgress(workflow.workflowId)) {
                outputChannel.appendLine(JSON.stringify(progress, null, 2));

                if (progress.status === 'completed') {
                    vscode.window.showInformationMessage('Deployment completed!');
                    break;
                }
            }

        } catch (error) {
            vscode.window.showErrorMessage(`Deployment failed: ${error.message}`);
        }
    });

    context.subscriptions.push(deployCommand);
}
```

📡 MCP Server API Reference

Authentication

All requests require Bearer token authentication:

```
Authorization: Bearer your-api-token
```

Core Endpoints

Health Check

```
GET /health
```

Response:

```
{
    "status": "healthy",
    "timestamp": "2025-07-15T10:30:00Z",
    "version": "2.0.0",
    "components": {
        "orchestrator": "active",
        "document_manager": "active",
        "context_manager": "active",
        "ssh_manager": "active"
    }
}
```

Create Workflow

```
POST /workflows
Content-Type: application/json

{
    "repository_url": "https://github.com/user/app.git",
    "task_prompt": "Deploy secure web application",
    "security_level": "medium",
    "ssh_config": {
        "host": "deploy.example.com",
        "user": "deploy",
        "key_path": "/path/to/key"
    }
}
```

Response:

```
{
    "workflow_id": "wf_abc123",
    "status": "started",
    "message": "Workflow created and started successfully",
    "created_at": "2025-07-15T10:30:00Z"
}
```

Stream Progress

```
GET /progress/{workflow_id}/stream
Accept: text/event-stream
```

Response (Server-Sent Events):

```
data: {"status": "analyzing", "progress": 10, "message": "Analyzing repository struc-
ture"}

data: {"status": "building", "progress": 30, "message": "Building application"}

data: {"status": "deploying", "progress": 70, "message": "Deploying to production"}

data: {"status": "completed", "progress": 100, "message": "Deployment completed suc-
cessfully"}
```

Document Management

Upload Document

```
POST /documents
Content-Type: application/json

{
    "filename": "deployment-guide.md",
    "content": "# Deployment Guide\n...",
    "content_type": "text/markdown"
}
```

Search Context

```
POST /context/search
Content-Type: application/json

{
    "query": "kubernetes deployment best practices",
    "limit": 10,
    "filters": {
        "type": "documentation",
        "tags": ["kubernetes", "deployment"]
    }
}
```

SSH Command Execution

Execute Command

```
POST /ssh/execute
Content-Type: application/json

{
    "command": "docker ps",
    "security_level": "medium",
    "timeout": 30
}
```

Response:

```
{
    "success": true,
    "stdout": "CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS
NAMES\n...",
    "stderr": "",
    "exit_code": 0,
    "execution_time": 1.23
}
```

🔧 Client Libraries

Python Client

```python
from mcp_server.client import MCPClient, MCPConfig

# Basic usage
config = MCPConfig(host="localhost", port=8002, token="your-token")

async with MCPClient(config) as client:
    # Deploy
    workflow = await client.create_workflow(
        repository_url="https://github.com/user/app.git",
        task_prompt="Deploy with SSL and monitoring"
    )

    # Monitor progress
    async for progress in client.stream_workflow_progress(workflow["workflow_id"]):
        print(f"Progress: {progress['progress']}% - {progress['message']}")
        if progress["status"] in ["completed", "failed"]:
            break

    # Execute SSH command
    result = await client.execute_ssh_command("systemctl status nginx")
    print(f"Command output: {result['stdout']}")
```

JavaScript/TypeScript Client

```javascript
import { MCPClient, MCPConfig } from '@contextflow/mcp-client';

const client = new MCPClient({
    host: 'localhost',
    port: 8002,
    token: 'your-token'
});

// Deploy application
const workflow = await client.createWorkflow({
    repositoryUrl: 'https://github.com/user/app.git',
    taskPrompt: 'Deploy microservice with database'
});

// Stream progress
client.streamProgress(workflow.workflowId, (progress) => {
    console.log(`${progress.progress}%: ${progress.message}`);

    if (progress.status === 'completed') {
        console.log('Deployment completed successfully!');
    }
});
```

Go Client

```go
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/contextflow-ai/go-client"
)

func main() {
    client := contextflow.NewClient(&contextflow.Config{
        Host:  "localhost",
        Port:  8002,
        Token: "your-token",
    })

    // Deploy application
    workflow, err := client.CreateWorkflow(context.Background(), &context-
flow.WorkflowRequest{
        RepositoryURL: "https://github.com/user/app.git",
        TaskPrompt:    "Deploy Go microservice",
    })
    if err != nil {
        log.Fatal(err)
    }

    fmt.Printf("Workflow started: %s\n", workflow.WorkflowID)

    // Monitor progress
    progressChan, err := client.StreamProgress(context.Background(), work-
flow.WorkflowID)
    if err != nil {
        log.Fatal(err)
    }

    for progress := range progressChan {
        fmt.Printf("Progress: %d%% - %s\n", progress.Progress, progress.Message)

        if progress.Status == "completed" {
            fmt.Println("Deployment completed!")
            break
        }
    }
}
```

🔒 Security Configuration

Authentication Setup

```
# Generate secure API token
openssl rand -hex 32

# Set environment variables
export MCP_API_TOKEN="your-secure-token"
export MCP_SERVER_HOST="0.0.0.0"
export MCP_SERVER_PORT="8002"
```

TLS Configuration

```
# mcp_server/tls_config.py
import ssl
import uvicorn

def create_ssl_context():
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain('/path/to/cert.pem', '/path/to/key.pem')
    return context

# Start server with TLS
if __name__ == "__main__":
    uvicorn.run(
        "main:app",
        host="0.0.0.0",
        port=8002,
        ssl_keyfile="/path/to/key.pem",
        ssl_certfile="/path/to/cert.pem"
    )
```

Rate Limiting

```
# Built-in rate limiting
from fastapi import Request
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter
app.add_exception_handler(429, _rate_limit_exceeded_handler)

@app.post("/workflows")
@limiter.limit("10/minute")
async def create_workflow(request: Request, ...):
    # Endpoint implementation
    pass
```

📊 Monitoring & Observability

Metrics Collection

```
from prometheus_client import Counter, Histogram, generate_latest

# Metrics
workflow_counter = Counter('contextflow_workflows_total', 'Total workflows created')
deployment_duration = Histogram('contextflow_deployment_duration_seconds', 'Deployment
duration')

@app.get("/metrics")
async def metrics():
    return Response(generate_latest(), media_type="text/plain")
```

Logging Configuration

```
import structlog

logger = structlog.get_logger()

@app.middleware("http")
async def logging_middleware(request: Request, call_next):
    start_time = time.time()

    response = await call_next(request)

    logger.info(
        "request_completed",
        method=request.method,
        url=str(request.url),
        status_code=response.status_code,
        duration=time.time() - start_time
    )

    return response
```

🚀 Deployment

Docker Deployment

```
# Dockerfile.mcp-server
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY mcp_server/ ./mcp_server/
COPY *.py ./

EXPOSE 8002

CMD ["python", "-m", "mcp_server.main"]
```

```yaml
# docker-compose.yml
version: '3.8'

services:
  mcp-server:
    build:
      context: .
      dockerfile: Dockerfile.mcp-server
    ports:
      - "8002:8002"
    environment:
      - MCP_API_TOKEN=${MCP_API_TOKEN}
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    volumes:
      - ./keys:/app/keys:ro
    depends_on:
      - redis
      - postgres
```

Kubernetes Deployment

```yaml
# k8s/mcp-server.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: contextflow-mcp-server
spec:
  replicas: 3
  selector:
    matchLabels:
      app: contextflow-mcp-server
  template:
    metadata:
      labels:
        app: contextflow-mcp-server
    spec:
      containers:
      - name: mcp-server
        image: contextflow/mcp-server:2.0.0
        ports:
        - containerPort: 8002
        env:
        - name: MCP_API_TOKEN
          valueFrom:
            secretKeyRef:
              name: contextflow-secrets
              key: mcp-api-token
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
---
apiVersion: v1
kind: Service
metadata:
  name: contextflow-mcp-server
spec:
  selector:
    app: contextflow-mcp-server
  ports:
  - port: 8002
    targetPort: 8002
  type: LoadBalancer
```

🔧 Troubleshooting

Common Issues

Connection Refused

```
# Check if server is running
curl http://localhost:8002/health

# Check logs
docker logs contextflow-mcp-server

# Verify port binding
netstat -tlnp | grep 8002
```

Authentication Errors

```
# Verify token
import os
print(f"Token: {os.getenv('MCP_API_TOKEN')}")

# Test authentication
curl -H "Authorization: Bearer your-token" http://localhost:8002/health
```

Performance Issues

```
# Enable debug logging
import logging
logging.basicConfig(level=logging.DEBUG)

# Monitor resource usage
import psutil
print(f"CPU: {psutil.cpu_percent()}%")
print(f"Memory: {psutil.virtual_memory().percent}%")
```

Debug Mode

```
# Start server in debug mode
DEBUG=true python -m mcp_server.main

# Enable verbose logging
LOG_LEVEL=DEBUG python -m mcp_server.main
```

The MCP server integration provides a powerful headless interface for modern development workflows, enabling seamless integration with IDEs and development tools while maintaining enterprise-grade security and performance.