API Documentation

The Enhanced Two-Tiered Multi-Agent Orchestration System provides a comprehensive REST API for managing workflows, executing commands, and monitoring system health.

Base URL

```
https://mcp.xplaincrypto.ai
```

Authentication

All API endpoints require authentication using an API key passed in the `X-API-Key` header.

```
curl -H "X-API-Key: your_api_key_here" https://mcp.xplaincrypto.ai/health
```

Rate Limiting

- **Default Limit**: 100 requests per minute per IP address
- **Headers**: Rate limit information is returned in response headers
- `X-RateLimit-Limit` : Maximum requests per window
- `X-RateLimit-Remaining` : Remaining requests in current window
- `X-RateLimit-Reset` : Time when the rate limit resets

Response Format

All API responses follow a consistent JSON format:

```
{
  "success": true,
  "data": {},
  "error": null,
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

Endpoints

Health Check

Check the health status of the orchestration system.

Endpoint: `GET /health`

Response:

```
{
  "status": "healthy",
  "timestamp": "2024-01-15T10:30:00.000Z",
  "version": "2.0.0",
  "uptime_seconds": 3600.5
}
```

Example:

```
curl -H "X-API-Key: your_api_key" https://mcp.xplaincrypto.ai/health
```

Workflow Management

Start Workflow

Start a new deployment workflow (auto-detects AI module vs task prompt).

Endpoint: `POST /workflow`

Request Body:

```
{
  "repository_url": "https://github.com/user/repo.git",
  "task_prompt": "Optional task description for traditional workflows",
  "target_host": "optional_target_host",
  "environment_variables": {
    "ENV_VAR": "value"
  }
}
```

Response:

```
{
  "success": true,
  "workflow_id": "abc123def456",
  "workflow_type": "ai_module",
  "status": "running",
  "endpoint": "http://mcp.xplaincrypto.ai:3000",
  "health_check": "http://mcp.xplaincrypto.ai:3000/health"
}
```

Examples:

AI Module Workflow:

```
curl -X POST https://mcp.xplaincrypto.ai/workflow \
  -H "Content-Type: application/json" \
  -H "X-API-Key: your_api_key" \
  -d '{
    "repository_url": "https://github.com/user/crypto-dashboard.git"
  }'
```

Task Prompt Workflow:

```
curl -X POST https://mcp.xplaincrypto.ai/workflow \
  -H "Content-Type: application/json" \
  -H "X-API-Key: your_api_key" \
  -d '{
    "repository_url": "https://github.com/user/simple-api.git",
    "task_prompt": "Deploy a Python Flask API with authentication and rate limiting"
  }'
```

List Workflows

Get a list of all active workflows.

Endpoint: `GET /workflows`

Response:

```json
{
  "workflows": [
    {
      "workflow_id": "abc123def456",
      "workflow_type": "ai_module",
      "status": "running",
      "repository_url": "https://github.com/user/repo.git",
      "created_at": "2024-01-15T10:00:00.000Z"
    }
  ]
}
```

Example:

```
curl -H "X-API-Key: your_api_key" https://mcp.xplaincrypto.ai/workflows
```

Get Workflow Status

Get detailed status information for a specific workflow.

Endpoint: `GET /workflows/{workflow_id}`

Response:

```
{
  "workflow_id": "abc123def456",
  "workflow_type": "ai_module",
  "status": "running",
  "repository_url": "https://github.com/user/repo.git",
  "target_host": "mcp.xplaincrypto.ai",
  "created_at": "2024-01-15T10:00:00.000Z",
  "updated_at": "2024-01-15T10:05:00.000Z",
  "deployment_logs": [
    "Task: install_dependencies - SUCCESS",
    "Task: build_application - SUCCESS",
    "Task: start_application - SUCCESS"
  ],
  "config": {
    "name": "crypto-dashboard",
    "version": "1.0.0",
    "port": 3000
  }
}
```

Example:

```
curl -H "X-API-Key: your_api_key" https://mcp.xplaincrypto.ai/workflows/abc123def456
```

Stop Workflow

Stop a running workflow and clean up resources.

Endpoint: `DELETE /workflows/{workflow_id}`

Response:

```
{
  "success": true,
  "workflow_id": "abc123def456",
  "status": "stopped"
}
```

Example:

```
curl -X DELETE -H "X-API-Key: your_api_key" https://mcp.xplaincrypto.ai/workflows/ab-
c123def456
```

Command Execution

Execute SSH Command

Execute a command on the target server via SSH.

Endpoint: `POST /execute`

Request Body:

```
{
  "command": "docker ps",
  "timeout": 300,
  "working_directory": "/opt/projects",
  "environment_variables": {
    "PATH": "/usr/local/bin:/usr/bin:/bin"
  },
  "security_level": "medium"
}
```

Security Levels:
- `low`: Basic validation
- `medium`: Standard security checks (default)
- `high`: Strict validation with command whitelist
- `critical`: Maximum security with minimal allowed commands

Response:

```
{
  "success": true,
  "stdout": "CONTAINER ID   IMAGE     COMMAND    CREATED    STATUS    PORTS    NAMES\n",
  "stderr": "",
  "exit_code": 0,
  "execution_time": 0.234,
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

Example:

```
curl -X POST https://mcp.xplaincrypto.ai/execute \
  -H "Content-Type: application/json" \
  -H "X-API-Key: your_api_key" \
  -d '{
    "command": "ls -la /opt",
    "security_level": "medium"
  }'
```

Security and Monitoring

Get Security Report

Generate a comprehensive security report for the specified time period.

Endpoint: `GET /security/report?hours=24`

Query Parameters:
- `hours`: Number of hours to include in the report (default: 24)

Response:

```json
{
  "report_period": {
    "start_time": "2024-01-14T10:30:00.000Z",
    "end_time": "2024-01-15T10:30:00.000Z",
    "duration_hours": 24
  },
  "summary": {
    "total_events": 1250,
    "total_risk_score": 150,
    "average_risk_score": 0.12,
    "critical_threats": 0,
    "high_threats": 2
  },
  "event_breakdown": {
    "authentication": 45,
    "command_execution": 890,
    "network_access": 315
  },
  "security_level_breakdown": {
    "low": 1100,
    "medium": 140,
    "high": 8,
    "critical": 2
  },
  "top_users": {
    "system": 800,
    "admin": 250,
    "api_user": 200
  },
  "top_source_ips": {
    "127.0.0.1": 600,
    "192.168.1.100": 400,
    "10.0.0.50": 250
  },
  "critical_threats": [],
  "high_threats": [
    {
      "indicator_type": "suspicious_ip",
      "value": "192.168.1.200",
      "severity": "high",
      "description": "Multiple failed login attempts: 8",
      "first_seen": "2024-01-15T09:00:00.000Z",
      "last_seen": "2024-01-15T09:30:00.000Z",
      "count": 8
    }
  ],
  "recommendations": [
    "Consider blocking 1 suspicious IP addresses with multiple failed login attempts",
    "Review and restrict dangerous command execution permissions"
  ]
}
```

Example:

```
curl -H "X-API-Key: your_api_key" "https://mcp.xplaincrypto.ai/security/report?
hours=48"
```

AI Module Templates

Get AI Module Template

Get a template `ai-module.yaml` file for a specific module type.

Endpoint: `GET /ai-module/template/{module_type}`

Module Types:
- `web_app` : Web applications (React, Vue, Angular, etc.)
- `api` : REST APIs and microservices
- `ml_model` : Machine learning models and services
- `data_pipeline` : Data processing pipelines
- `microservice` : General microservices
- `automation` : Automation scripts and tools
- `infrastructure` : Infrastructure components

Response:

```
{
  "module_type": "web_app",
  "template": "name: \"my-web-app\"\nversion: \"1.0.0\"\ndescription: \"A modern web application\"\nmodule_type: \"web_app\"\nbuild_command: \"npm run build\"\ntest_command: \"npm test\"\nstart_command: \"npm start\"\ndependencies:\n  - \"react\"\n  - \"express\"\ndev_dependencies:\n  - \"jest\"\n  - \"eslint\"\nport: 3000\nsecurity:\n  cors_enabled: true\n  rate_limiting: true\n"
}
```

Example:

```
curl -H "X-API-Key: your_api_key" https://mcp.xplaincrypto.ai/ai-module/template/api
```

Error Handling

Error Response Format

```
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid request parameters",
    "details": {
      "field": "repository_url",
      "reason": "URL format is invalid"
    }
  },
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

HTTP Status Codes

- `200 OK` : Request successful
- `201 Created` : Resource created successfully
- `400 Bad Request` : Invalid request parameters

- `401 Unauthorized` : Invalid or missing API key
- `403 Forbidden` : Insufficient permissions
- `404 Not Found` : Resource not found
- `429 Too Many Requests` : Rate limit exceeded
- `500 Internal Server Error` : Server error
- `503 Service Unavailable` : Service temporarily unavailable

Common Error Codes

- `INVALID_API_KEY` : API key is invalid or missing
- `RATE_LIMIT_EXCEEDED` : Too many requests in time window
- `VALIDATION_ERROR` : Request validation failed
- `WORKFLOW_NOT_FOUND` : Specified workflow ID not found
- `COMMAND_BLOCKED` : Command blocked by security policy
- `SSH_CONNECTION_FAILED` : Unable to establish SSH connection
- `DEPLOYMENT_FAILED` : Deployment process failed
- `TIMEOUT_ERROR` : Operation timed out

SDK and Client Libraries

Python SDK Example

```python
import requests

class OrchestratorClient:
    def __init__(self, base_url, api_key):
        self.base_url = base_url
        self.headers = {"X-API-Key": api_key}

    def start_workflow(self, repository_url, task_prompt=None):
        data = {"repository_url": repository_url}
        if task_prompt:
            data["task_prompt"] = task_prompt

        response = requests.post(
            f"{self.base_url}/workflow",
            json=data,
            headers=self.headers
        )
        return response.json()

    def get_workflow_status(self, workflow_id):
        response = requests.get(
            f"{self.base_url}/workflows/{workflow_id}",
            headers=self.headers
        )
        return response.json()

# Usage
client = OrchestratorClient("https://mcp.xplaincrypto.ai", "your_api_key")
result = client.start_workflow("https://github.com/user/repo.git")
print(result)
```

JavaScript SDK Example

```javascript
class OrchestratorClient {
    constructor(baseUrl, apiKey) {
        this.baseUrl = baseUrl;
        this.headers = {
            'X-API-Key': apiKey,
            'Content-Type': 'application/json'
        };
    }

    async startWorkflow(repositoryUrl, taskPrompt = null) {
        const data = { repository_url: repositoryUrl };
        if (taskPrompt) {
            data.task_prompt = taskPrompt;
        }

        const response = await fetch(`${this.baseUrl}/workflow`, {
            method: 'POST',
            headers: this.headers,
            body: JSON.stringify(data)
        });

        return await response.json();
    }

    async getWorkflowStatus(workflowId) {
        const response = await fetch(`${this.baseUrl}/workflows/${workflowId}`, {
            headers: this.headers
        });

        return await response.json();
    }
}

// Usage
const client = new OrchestratorClient('https://mcp.xplaincrypto.ai', 'your_api_key');
client.startWorkflow('https://github.com/user/repo.git')
    .then(result => console.log(result));
```

Webhooks (Future Feature)

Webhook Events

The system will support webhooks for real-time notifications:

- `workflow.started`: Workflow has been initiated
- `workflow.completed`: Workflow completed successfully
- `workflow.failed`: Workflow failed
- `security.threat_detected`: Security threat detected
- `system.health_check_failed`: Health check failure

Webhook Payload Example

```json
{
  "event": "workflow.completed",
  "timestamp": "2024-01-15T10:30:00.000Z",
  "data": {
    "workflow_id": "abc123def456",
    "workflow_type": "ai_module",
    "repository_url": "https://github.com/user/repo.git",
    "endpoint": "http://mcp.xplaincrypto.ai:3000"
  }
}
```

Best Practices

API Usage

1. **Always include error handling** in your client code
2. **Respect rate limits** and implement exponential backoff
3. **Use HTTPS** for all API calls in production
4. **Store API keys securely** and rotate them regularly
5. **Monitor API usage** and set up alerts for failures

Security

1. **Use strong API keys** (minimum 32 characters)
2. **Implement IP whitelisting** when possible
3. **Log all API calls** for audit purposes
4. **Use appropriate security levels** for command execution
5. **Regularly review security reports**

Performance

1. **Cache responses** when appropriate
2. **Use pagination** for large result sets
3. **Implement connection pooling** for high-volume usage
4. **Monitor response times** and optimize accordingly
5. **Use async/await** patterns for better performance