AgentOps Studio User Documentation

The Visual IDE for Building Production-Ready AI Agent Systems

Version 1.0 | June 17, 2025

Table of Contents

1. Getting Started

What is AgentOps Studio?

AgentOps Studio is a visual development environment that lets you build, test, and deploy AI agent systems without writing glue code. Think of it as GitHub Actions meets Postman, but specifically designed for LLM orchestration.

Why Use AgentOps Studio?

As a software developer building internal agent systems, you're likely tired of:

- Writing boilerplate code to chain LLM calls
- Debugging prompt flows through console logs
- Guessing at token costs until the invoice arrives
- Manually switching between different AI providers

AgentOps Studio solves these problems by providing a visual canvas where you can drag, drop, and connect AI capabilities while seeing real-time costs and performance metrics.

System Requirements

- **Browser**: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- **Screen Resolution**: Minimum 1280x720, recommended 1920x1080
- **Internet**: Stable connection (5 Mbps+ recommended)

- **Accounts Needed**:
  - AgentOps Studio account (free tier available)
  - API keys for your chosen AI providers (Claude, OpenAI, Groq, etc.)

Quick Start Checklist

1. **Create Your Account**

   - Visit agentops.studio (https://agentops.studio)
   - Sign up with your work email
   - Verify your email address

2. **Configure Your Workspace**

   - Navigate to Settings → API Keys
   - Add your provider keys (start with just one)
   - Test the connection using the "Ping" button

3. **Explore the Interface**

   - **Canvas**: Central area for building pipelines
   - **Node Palette**: Left sidebar with available components
   - **Inspector**: Right panel for node configuration
   - **Metrics Console**: Bottom dock showing costs/latency

---

2. Core Concepts

Understanding Agent Pipelines

An agent pipeline is a visual representation of your AI workflow. Each pipeline consists of:

- **Nodes**: Individual AI capabilities (LLM calls, tools, logic)
- **Edges**: Data flow connections between nodes
- **Context**: Shared state that flows through the pipeline
- **Triggers**: What starts your pipeline (API call, schedule, event)

Node Types Explained

AgentOps Studio provides several node categories:

LLM Nodes

- **Claude Node**: Anthropic's Claude for complex reasoning
- **GPT Node**: OpenAI models for general tasks
- **Groq Node**: Ultra-fast inference for time-sensitive operations
- **Gemini Node**: Google's multimodal capabilities

Processing Nodes

- **Transform**: Modify data structure or format
- **Filter**: Conditional logic and routing

- **Aggregate**: Combine multiple inputs
- **Split**: Parallelize operations

Integration Nodes

- **HTTP Request**: Call external APIs
- **Database Query**: Read/write to your data stores
- **File Operations**: Process documents and media
- **Knowledge Graph**: Query your organization's knowledge base

Data Flow Philosophy

Data in AgentOps Studio flows like water through pipes:

1. Each node receives input from connected predecessors
2. Processes the data according to its configuration
3. Outputs results to connected successors
4. All data is typed and validated at each step

This ensures predictable behavior and makes debugging straightforward.

---

3. Building Your First Agent Pipeline

Let's build a practical example: an agent that processes support tickets, categorizes them, and drafts responses.

Step 1: Create a New Pipeline

1. Click the **"+ New Pipeline"** button in your workspace
2. Name it "Support Ticket Handler"
3. Add a description: "Categorizes and responds to customer support tickets"

Step 2: Add Your First Node

1. From the Node Palette, drag a **"Trigger"** node onto the canvas
2. Double-click to configure:
    - Type: `Webhook`
    - Path: `/support/ticket`
    - Method: `POST`
3. This node will receive incoming ticket data

Step 3: Process the Ticket

1. Add a **Claude Node** for understanding the ticket:
    - Drag from palette and position to the right of your trigger
    - Connect the trigger's output to Claude's input
    - Configure the prompt:

```
Analyze this support ticket and extract:
1. Category (billing, technical, feature-request, other)
2. Urgency (low, medium, high, critical)
3. Key issues mentioned
4. Customer sentiment

Ticket: {{input.ticket_text}}
```

Step 4: Route Based on Category

1. Add a **Filter Node** after Claude:
   - Connect Claude's output to the filter input
   - Add routing rules:
     - If `category == "billing"` → Route to billing handler
     - If `category == "technical"` → Route to tech handler
     - Else → Route to general handler

Step 5: Generate Responses

1. Add specialized response nodes for each category
2. For the technical route, add a **Groq Node** (for speed):

```
Generate a helpful technical support response for:
Issue: {{input.key_issues}}
Sentiment: {{input.sentiment}}

Be concise, professional, and include next steps.
```

Step 6: Test Your Pipeline

1. Click the **"Test"** button in the toolbar
2. Paste a sample ticket in the test panel
3. Watch the execution flow through your pipeline
4. Check the Metrics Console for cost and latency

Step 7: Save and Deploy

1. Click **"Save Pipeline"**
2. Toggle the "Active" switch to enable the webhook
3. Copy your webhook URL from the deployment panel

---

4. Working with Nodes

Node Configuration Deep Dive

Every node has three configuration sections:

Basic Settings

- **Name**: Descriptive identifier for the node
- **Description**: What this node does in your pipeline
- **Retry Policy**: How to handle failures

Provider Settings

- **Model Selection**: Choose specific models/versions
- **Temperature**: Control creativity vs consistency
- **Max Tokens**: Limit response length and cost
- **Timeout**: Maximum wait time

Advanced Settings

- **System Prompt**: Set consistent behavior
- **Response Format**: JSON, Markdown, Plain text
- **Caching**: Enable for repeated queries
- **Fallback Provider**: Backup if primary fails

Prompt Engineering in AgentOps

Effective prompts are crucial for reliable agent systems. Here's how to write them:

Use Variables

```
Analyze the {{input.document_type}} for:
- Main topics discussed
- Action items mentioned
- Deadlines specified

Document content:
{{input.content}}
```

Structure for Clarity

```
Role: You are a senior financial analyst.
Context: Our company uses GAAP accounting standards.
Task: Review this expense report for policy compliance.
Format: Return a JSON object with fields: compliant (boolean), issues (array), suggestions (array)
```

Include Examples

```
Categorize this message as one of: [urgent, normal, low]

Examples:
"Server is down!" → urgent
"Monthly report attached" → normal
"FYI article link" → low

Message: {{input.message}}
```

Connecting Nodes Effectively

Data Transformation When connecting nodes with different data expectations:

1. Use Transform nodes to reshape data
2. Access nested properties with dot notation: `{{input.result.category}}`
3. Use fallback values: `{{input.urgency || "medium"}}`

Parallel Processing Split complex tasks for efficiency:

1. Use a Split node to parallelize independent operations
2. Process in parallel (e.g., translate to 5 languages simultaneously)
3. Use an Aggregate node to combine results

Error Handling Build resilient pipelines:

1. Add error outputs to critical nodes
2. Route errors to notification or logging nodes
3. Implement graceful degradation paths

---

5. Managing Costs and Performance

Understanding the Metrics Console

The Metrics Console provides real-time insights:

Cost Tracking

- **Token Usage**: Input + output tokens per node
- **Cost Breakdown**: Price per operation
- **Running Total**: Cumulative pipeline cost
- **Projected Monthly**: Based on current usage

Performance Metrics

- **Node Latency**: Time spent in each operation
- **Pipeline Duration**: Total end-to-end time
- **Queue Time**: Waiting for resources
- **Success Rate**: Percentage of successful runs

Optimization Strategies

Reduce Costs

1. **Use Appropriate Models**: Don't use GPT-4 for simple classifications
2. **Implement Caching**: Cache repeated queries (e.g., standard responses)
3. **Optimize Prompts**: Shorter prompts = fewer input tokens
4. **Set Token Limits**: Prevent runaway responses

Improve Performance

1. **Parallelize When Possible**: Split independent operations
2. **Use Groq for Speed**: When latency matters more than cost
3. **Implement Timeouts**: Fail fast on slow operations
4. **Regional Deployment**: Deploy close to your users

Budget Controls

Set spending limits at multiple levels:

Pipeline Level

- Daily/monthly cost caps
- Automatic pause when exceeded
- Email alerts at 80% threshold

Workspace Level

- Organization-wide monthly budget
- Per-user allocations
- Department cost centers

---

6. Knowledge Graph Integration

What is a Knowledge Graph?

A Knowledge Graph (KG) in AgentOps represents your organization's information as interconnected entities. This allows your agents to:

- Access company-specific knowledge
- Understand relationships between concepts
- Provide grounded, factual responses
- Reduce hallucinations

Setting Up Your Knowledge Graph

Step 1: Connect Your Data Sources

1. Navigate to Knowledge → Data Sources
2. Add connections:
    - Documentation repositories (GitHub, GitLab)

- Databases (PostgreSQL, MySQL)
- File storage (Google Drive, Dropbox)
- APIs (CRM, ERP systems)

Step 2: Configure Ingestion

1. Set update frequency (real-time, hourly, daily)
2. Define extraction rules:
    - What constitutes an entity
    - How to identify relationships
    - Which fields to index

Step 3: Review and Refine

1. Explore the graph visualization
2. Verify entity extraction accuracy
3. Add manual connections if needed
4. Set access permissions

Using Knowledge Graph Nodes

Basic Query Node

```
Type: Knowledge Graph Query
Query: "Find all policies related to {{input.topic}}"
Return: "Top 5 most relevant documents"
Include Context: true
```

Relationship Traversal

```
Type: Graph Traversal
Start: "{{input.employee_name}}"
Relationship: "reports_to"
Depth: 3
Return: "Management chain"
```

Fact Checking

```
Type: Fact Verification
Statement: "{{input.claim}}"
Confidence Threshold: 0.8
Return Sources: true
```

GraphRAG Pattern

Combine graph queries with LLM reasoning:

1. **Query KG** for relevant entities
2. **Extract context** from connected nodes

3. **Pass to LLM** with grounded facts
4. **Generate response** with citations

This pattern dramatically reduces hallucinations while maintaining conversational quality.

---

7. Context Copilot (Coming June 19)

Overview

Context Copilot is AgentOps Studio's revolutionary debugging assistant that uses:

- Eye tracking or cursor position
- Voice commands
- Screen context
- Your codebase knowledge graph

How It Will Work

Activation

1. Install the Context Copilot browser extension
2. Grant necessary permissions (camera for eye tracking, microphone)
3. Calibrate eye tracking (30-second process)

Usage Flow

1. Look at problematic code
2. Say "Hey Copilot" or press hotkey
3. Ask your question naturally
4. See relevant code graph highlight
5. Receive contextual suggestions

Preparing for Context Copilot

Index Your Codebase

1. Connect your repositories to Knowledge Graph
2. Enable AST (Abstract Syntax Tree) analysis
3. Let the system build code relationships

Configure Preferences

- Preferred explanation style (concise vs detailed)
- Language-specific settings
- Privacy boundaries (which code to exclude)

---

8. Deployment Options

Development Environment

Local Testing

- Use the built-in test runner
- Mock external services
- Validate with sample data

Staging Deployment

- One-click staging environment
- Separate API keys and quotas
- A/B testing capabilities

Production Deployment

Option 1: Managed Hosting

- AgentOps Cloud handles everything
- Automatic scaling
- Built-in monitoring
- SLA guarantees

Option 2: Self-Hosted

Export as Docker Compose

1. Click Deploy → Export → Docker Compose
2. Receive `docker-compose.yml` with:
    - All node configurations
    - Environment variables template
    - Health check endpoints
    - Scaling recommendations

Export as Kubernetes

1. Click Deploy → Export → Kubernetes
2. Receive Helm chart with:
    - Deployment manifests
    - Service definitions
    - ConfigMaps for settings
    - HPA for autoscaling

Option 3: Serverless Functions

1. Export individual nodes as Lambda/Cloud Functions
2. Use for event-driven architectures
3. Pay only for execution time

Integration Methods

Webhook Integration

```
import requests

response = requests.post(
    "https://api.agentops.studio/pipelines/YOUR_PIPELINE_ID/trigger",
    headers={"Authorization": "Bearer YOUR_API_KEY"},
    json={"ticket_text": "Customer needs help with login"}
)
```

SDK Integration

```
from agentops import Pipeline

pipeline = Pipeline("YOUR_PIPELINE_ID")
result = await pipeline.run({
    "ticket_text": "Customer needs help with login"
})
```

Event Streaming

```
from agentops import StreamingPipeline

async with StreamingPipeline("YOUR_PIPELINE_ID") as pipeline:
    async for event in pipeline.stream(input_data):
        print(f"Node {event.node_name}: {event.result}")
```

---

9. Best Practices

Pipeline Design Principles

Single Responsibility Each pipeline should do one thing well:

- □ "Handle all customer communications"
- □ "Process support tickets"
- □ "Generate weekly reports"

Idempotency Pipelines should produce the same output for the same input:

- Use deterministic prompts (low temperature)
- Avoid time-based logic without clear boundaries
- Implement proper error handling

Observability First Build with debugging in mind:

- Use descriptive node names
- Add logging nodes at critical points
- Include context in error messages
- Tag pipelines with business metadata

Security Best Practices

API Key Management

- Never hardcode keys
- Use environment-specific keys
- Rotate keys quarterly
- Monitor key usage

Data Privacy

- Implement PII detection nodes
- Use redaction before sending to LLMs
- Store sensitive data separately
- Comply with regional regulations

Access Control

- Use role-based permissions
- Separate dev/prod access
- Audit pipeline modifications
- Implement approval workflows

Performance Patterns

Caching Strategy

```
Request → Cache Check → [Cache Hit] → Return
                ↓
           [Cache Miss] → LLM → Cache Write → Return
```

Batch Processing Instead of processing items individually:

1. Collect items in a queue
2. Process in batches of 10-50
3. Parallelize batch processing
4. Aggregate results

Circuit Breaker Pattern Protect against provider failures:

1. Track failure rate
2. Open circuit after threshold
3. Route to fallback provider
4. Periodically test recovery

---

10. Troubleshooting

Common Issues and Solutions

Pipeline Won't Start

- Check: Are all required node configurations complete?
- Check: Do you have valid API keys?
- Check: Is your trigger properly configured?
- Solution: Use the "Validate" button to find issues

Unexpected Costs

- Check: Token limits on each node
- Check: Retry policies (retries multiply costs)
- Check: Cache configuration
- Solution: Enable cost alerts and caps

Slow Performance

- Check: Sequential operations that could be parallel
- Check: Provider selection (some are faster)
- Check: Response token limits
- Solution: Use the Performance Analyzer tool

Inconsistent Results

- Check: Temperature settings (lower = more consistent)
- Check: Prompt ambiguity
- Check: Input data quality
- Solution: Add validation nodes

Debug Mode

Enable debug mode for detailed insights:

1. Click the bug icon in the toolbar
2. See:
   - Raw input/output for each node
   - Timing breakdowns
   - Token usage details
   - Error stack traces

Getting Help

Documentation

- In-app help: Click ? on any component
- Video tutorials: agentops.studio/tutorials (https://agentops.studio/tutorials)
- API reference: docs.agentops.studio (https://docs.agentops.studio)

Community Support

- Discord: discord.gg/agentops (https://discord.gg/agentops)
- Forum: community.agentops.studio (https://community.agentops.studio)
- Stack Overflow: Tag `agentops-studio`

Enterprise Support

- Email: support@agentops.studio
- SLA: 4-hour response time
- Dedicated Slack channel
- Monthly architecture reviews

Appendix A: Keyboard Shortcuts

Action	Windows/Linux	Mac
Save Pipeline	Ctrl+S	Cmd+S
Run Pipeline	Ctrl+Enter	Cmd+Enter
Duplicate Node	Ctrl+D	Cmd+D
Delete Node	Delete	Delete
Undo	Ctrl+Z	Cmd+Z
Redo	Ctrl+Y	Cmd+Shift+Z
Search Nodes	Ctrl+F	Cmd+F
Toggle Metrics	Ctrl+M	Cmd+M

Appendix B: Node Reference

LLM Nodes

Claude Node

- Models: Claude 3 Opus, Sonnet, Haiku
- Strengths: Complex reasoning, large context
- Best for: Analysis, writing, coding

GPT Node

- Models: GPT-4, GPT-3.5 Turbo
- Strengths: General purpose, wide knowledge
- Best for: Varied tasks, creative content

Groq Node

- Models: Llama 3, Mixtral
- Strengths: Ultra-fast inference
- Best for: Real-time responses, high volume

Gemini Node

- Models: Gemini Pro, Pro Vision
- Strengths: Multimodal capabilities
- Best for: Image analysis, long context

Utility Nodes

Transform Node

- JSONPath expressions

- JavaScript functions
- Template rendering

Validate Node

- Schema validation
- Business rule checking
- Data quality assertions

Cache Node

- TTL-based expiration
- Key generation strategies
- Cache warming options

End of User Documentation - Version 1.0