Task Execution Flow: Complete Technical Deep Dive

Table of Contents

1. Overview
2. Complete Execution Flow
3. Claude Code CLI Integration
4. Session State Transitions
5. File Locations and Logs
6. Monitoring and Debugging

Overview

This document provides a complete technical deep dive into how task execution works in the AI-Agent-API, from HTTP request to Claude Code CLI subprocess and back.

What Happens When You Execute a Task

```
HTTP POST /api/v1/tasks/{id}/execute
    ↓
Task Service creates execution record
    ↓
Session Service creates dedicated session
    ↓
Claude SDK Client Manager spawns Claude Code CLI subprocess
    ↓
Message sent to Claude via subprocess stdin/stdout
    ↓
Claude Code CLI executes in isolated working directory
    ↓
Responses streamed back and persisted
    ↓
Optional report generated
    ↓
HTTP 202 Accepted returned to client
```

Complete Execution Flow

Phase 1: API Request Handling

File: `app/api/v1/tasks.py:258-312`

```python
@router.post("/{task_id}/execute")
async def execute_task(
```

```
    task_id: UUID,
    request: TaskExecuteRequest,  # Contains variables dict
    current_user: User,
    db: AsyncSession,
):
```

Steps:

1. **Authentication** via JWT token (line 262)
2. **Get Task** from database (line 281)
3. **Ownership Check** (line 289-294)
4. **Call TaskService.execute_task()** (line 297-301)

HTTP Request:

```
POST /api/v1/tasks/75318f39-9feb-4ff6-a381-f377d38fd1be/execute
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...
Content-Type: application/json

{
  "variables": {
    "cluster_name": "production"
  }
}
```

Phase 2: Task Service Execution

File: app/services/task_service.py:138-334

Step 2.1: Validate Task (lines 176-186)

PROF

```
task = await self.task_repo.get_by_id(task_id)
if not task:
    raise TaskNotFoundError()
if not task.is_active:
    raise ValidationError("Task is not active")
```

Database Query:

```
SELECT * FROM tasks
WHERE id = '75318f39-9feb-4ff6-a381-f377d38fd1be'
  AND is_deleted = false;
```

Step 2.2: Create Task Execution Record (lines 189-211)

```
execution = TaskExecution(
    id=uuid4(),
    task_id=task.id,
    user_id=task.user_id,
    trigger_type="manual",
    variables={"cluster_name": "production"},
    status=TaskExecutionStatus.PENDING,
)
```

Database Insert:

```
INSERT INTO task_executions (
    id, task_id, user_id, trigger_type,
    variables, status, started_at
) VALUES (
    '660e8400-e29b-41d4-a716-446655440001',
    '75318f39-9feb-4ff6-a381-f377d38fd1be',
    '94d9f5a2-1257-43ac-9de2-6d86421455a6',
    'manual',
    '{"cluster_name": "production"}',
    'pending',
    NOW()
);
```

Step 2.3: Create Dedicated Session (lines 215-244)

PROF

```
session_service = SDKIntegratedSessionService(...)

session = await session_service.create_session(
    user_id=task.user_id,
    name=f"Task: Kubernetes Cluster Health Check (manual)",
    description="Automated execution of task 'Kubernetes Cluster Health
Check'",
    sdk_options=task.sdk_options,  # Contains MCP config, allowed_tools,
model, etc.
)
```

What This Creates:

- New session record in `sessions` table
- Isolated working directory: `/workspace/me/repositories/ai-agent/ai-agent-api/data/agent-workdirs/active/{session_id}/`
- Session status: `CREATED`

Database Insert:

```
INSERT INTO sessions (
    id, user_id, name, description, mode, status,
    working_directory_path, sdk_options, created_at
) VALUES (
    '770e8400-e29b-41d4-a716-446655440002',
    '94d9f5a2-1257-43ac-9de2-6d86421455a6',
    'Task: Kubernetes Cluster Health Check (manual)',
    'Automated execution...',
    'interactive',
    'created',
    '/workspace/.../data/agent-workdirs/active/770e8400...',
    '{...}',
    NOW()
);
```

Step 2.4: Update Execution Status to RUNNING (lines 237-244)

```
execution.session_id = session.id
execution.status = TaskExecutionStatus.RUNNING
await self.task_execution_repo.update(
    str(execution.id),
    session_id=session.id,
    status="running",
)
```

Database Update:

```
UPDATE task_executions
SET session_id = '770e8400-e29b-41d4-a716-446655440002',
    status = 'running',
    updated_at = NOW()
WHERE id = '660e8400-e29b-41d4-a716-446655440001';
```

Step 2.5: Render Prompt Template (lines 247-250)

```
rendered_prompt = self._render_prompt_template(
    task.prompt_template,
    variables={"cluster_name": "production"},
)
```

Template Rendering:

```
Input Template:
"IMPORTANT: This is a READ-ONLY health check...
Perform a comprehensive Kubernetes cluster health check
using ~/.kube/config for the {cluster_name} cluster:..."

Variables:
{"cluster_name": "production"}

Rendered Output:
"IMPORTANT: This is a READ-ONLY health check...
Perform a comprehensive Kubernetes cluster health check
using ~/.kube/config for the production cluster:..."
```

Step 2.6: Send Message Through Session (lines 254-257)

```
message = await session_service.send_message(
    session_id=str(session.id),
    message_content=rendered_prompt,
)
```

This triggers the most complex part of the system...

Phase 3: SDK Integrated Session Service

File: `app/services/sdk_session_service.py:83-255`

Step 3.1: Validate Session State (lines 128-140)

PROF

```
session = await self.get_session(session_id, user_id)

if session.status not in [SessionStatus.CREATED, SessionStatus.ACTIVE,
SessionStatus.CONNECTING]:
    raise SessionNotActiveError()
```

Step 3.2: Session State Transitions (lines 143-170)

Transition 1: CREATED → CONNECTING (lines 149-152)

```
session.transition_to(SessionStatus.CONNECTING)
await self.session_repo.update(session_id, status="connecting")
await self.db.commit()
```

Database Update:

```
UPDATE sessions
SET status = 'connecting', updated_at = NOW()
WHERE id = '770e8400-e29b-41d4-a716-446655440002';
```

Setup SDK Client (lines 155-160)

```
if not self.sdk_client_manager.has_client(session_id):
    await self._setup_sdk_client(session, user_id)
```

Transition 2: CONNECTING → ACTIVE (lines 162-165)

```
session.transition_to(SessionStatus.ACTIVE)
await self.session_repo.update(session_id, status="active")
await self.db.commit()
```

Transition 3: ACTIVE → PROCESSING (lines 172-175)

```
session.transition_to(SessionStatus.PROCESSING)
await self.session_repo.update(session_id, status="processing")
await self.db.commit()
```

Step 3.3: Setup SDK Client (lines 256-373)

File: `app/services/sdk_session_service.py:256-373`

PROF

3.3.1: Build MCP Configuration (lines 279-303)

```
from app.mcp import MCPConfigBuilder

mcp_config_builder = MCPConfigBuilder(mcp_server_repo)
mcp_config = await mcp_config_builder.build_session_mcp_config(
    user_id=user_id,
    include_sdk_tools=True,  # Includes kubernetes_readonly, database,
monitoring
)
```

What This Does:

- Loads SDK MCP servers from `app/mcp/mcp_servers.py`

    - `kubernetes_readonly` - 3 tools (get_pods, get_deployments, get_namespaces)
    - `database` - 2 tools (query_database, list_tables)
    - `monitoring` - 2 tools (check_health, get_metrics)
- Loads user's personal MCP servers from `mcp_servers` table
- Loads global MCP servers from `mcp_servers` table where `is_global=true`
- Merges all into single MCP config dict

MCP Config Structure:

```
{
    "kubernetes_readonly": {
        "command": "npx",
        "args": ["-y", "@modelcontextprotocol/server-kubernetes"],
        "env": {"KUBECONFIG": "/home/msalah/.kube/config"}
    },
    "database": {
        "command": "python",
        "args": ["/path/to/mcp_database_server.py"],
        "env": {}
    },
    # ... more servers
}
```

3.3.2: Merge MCP Config into Session (lines 305-326)

```
sdk_options_dict = session.sdk_options.to_dict()
sdk_options_dict["mcp_servers"] = mcp_config

session.sdk_options = SDKOptions.from_dict(sdk_options_dict)

await self.session_repo.update(
    str(session.id),
    sdk_options=sdk_options_dict
)
await self.db.commit()
```

PROF

Database Update:

```
UPDATE sessions
SET sdk_options = '{
    "model": "claude-3-5-sonnet-20241022",
    "max_tokens": 16384,
    "allowed_tools": ["Read", "Write", "Bash"],
    "mcp_servers": {
        "kubernetes_readonly": {...},
        "database": {...}
```

```
    }
}',
updated_at = NOW()
WHERE id = '770e8400-e29b-41d4-a716-446655440002';
```

3.3.3: Create Permission Callback (lines 328-332)

```
permission_callback =
self.permission_service.create_permission_callback(
    session_id=session.id,
    user_id=user_id,
)
```

What This Creates:
A Python callable that gets called BEFORE each tool use:

```
async def permission_callback(tool_name: str, args: dict) -> bool:
    # Check if tool is in allowed_tools list
    # Log permission decision to audit_logs table
    # Return True/False
```

3.3.4: Create Hooks (lines 334-361)

```
hooks = {
    "PreToolUse": [
        HookMatcher(hooks=[
            create_audit_hook(session.id, audit_service),
            create_tool_tracking_hook(session.id, db, tool_call_repo),
        ]),
    ],
    "PostToolUse": [
        HookMatcher(hooks=[
            create_audit_hook(session.id, audit_service),
            create_tool_tracking_hook(session.id, db, tool_call_repo),
            create_cost_tracking_hook(session.id, db, session_repo),
        ]),
    ],
}
```

Hook Functions:

- **Audit Hook**: Logs to `audit_logs` table
- **Tool Tracking Hook**: Creates records in `tool_calls` table with status, args, results
- **Cost Tracking Hook**: Updates session cost metrics (input_tokens, output_tokens, total_cost)

3.3.5: Create SDK Client (lines 363-372)

```
await self.sdk_client_manager.create_client(
    session=session,
    permission_callback=permission_callback,
    hooks=hooks,
)
```

This calls into `ClaudeSDKClientManager`...

Phase 4: Claude SDK Client Manager

File: `app/claude_sdk/client_manager.py:56-139`

Step 4.1: Build ClaudeAgentOptions (lines 102, 209-260)

```
options = self._build_options(session, permission_callback, hooks)
```

Converts to Official SDK Format:

```
ClaudeAgentOptions(
    allowed_tools=["Read", "Write", "Bash"],
    disallowed_tools=[],
    permission_mode="default",
    model="claude-3-5-sonnet-20241022",
    max_turns=None,
    cwd="/workspace/.../data/agent-workdirs/active/770e8400.../",
    mcp_servers={
        "kubernetes_readonly": {...},
        "database": {...},
    },
    system_prompt=None,
    env={},
    add_dirs=[],
    settings=None,
    can_use_tool=<permission_callback function>,
    permission_prompt_tool_name="stdio",
    hooks={...},
)
```

Step 4.2: Create Official SDK Client (line 105)

```
client = ClaudeSDKClient(options=options)
```

What This Is:

- Official `claude-agent-sdk` package client
- NOT our custom code
- Installed via `pip install claude-agent-sdk`

Step 4.3: Connect to Claude Code CLI (lines 108-126)

```
await client.connect(prompt=None)
```

What This Does (Inside Official SDK):

1. Spawns subprocess: `claude-code` CLI
2. Sets working directory to `cwd` option
3. Passes MCP config via env variables or config file
4. Opens stdin/stdout pipes for communication
5. Waits for "ready" signal from CLI

Actual Command Executed:

```
cd /workspace/.../data/agent-workdirs/active/770e8400.../

claude-code \
  --model claude-3-5-sonnet-20241022 \
  --mcp-servers '{"kubernetes_readonly": {...}}' \
  --allowed-tools "Read,Write,Bash" \
  --permission-mode default
```

PROF

Environment Variables Set:

```
ANTHROPIC_API_KEY=sk-ant-...
KUBECONFIG=/home/msalah/.kube/config
```

Step 4.4: Store Client in Pool (line 129)

```
self._clients[session_id] = client
```

Client Pool State:

```
```

```
{
    UUID('770e8400-e29b-41d4-a716-446655440002'): <ClaudeSDKClient
object>
}
```

Phase 5: Send Message to Claude

File: `app/services/sdk_session_service.py:180-226`

Step 5.1: Get Client from Pool (line 180)

```
client = await self.sdk_client_manager.get_client(session_id)
```

Step 5.2: Send Query to Claude (line 190)

```
await client.query(message_text)
```

What This Does (Inside Official SDK):

1. Writes message to Claude Code CLI stdin:

```
{
  "type": "user_message",
  "content": "IMPORTANT: This is a READ-ONLY health check...\nPerform a
comprehensive Kubernetes cluster health check..."
}
```

PROF

2. Claude Code CLI receives message
3. Sends to Anthropic API
4. Starts streaming response

Step 5.3: Process Response Stream (lines 194-226)

```
message_processor = MessageProcessor(
    db=self.db,
    message_repo=self.message_repo,
    tool_call_repo=self.tool_call_repo,
    session_repo=self.session_repo,
    event_broadcaster=self.event_broadcaster,
)
```

```
async for message in message_processor.process_message_stream(
    session=session,
    sdk_messages=client.receive_response(),  # Stream from Claude Code
CLI stdout
):
    yield message
```

Stream Format from Claude Code CLI:

```
{"type": "assistant_message_start", "id": "msg_123"}
{"type": "content_block_delta", "delta": {"text": "I'll check the
Kubernetes cluster health..."}}
{"type": "tool_use", "name": "Bash", "input": {"command": "kubectl get
nodes"}}
{"type": "tool_result", "tool_use_id": "tool_123", "output": "NAME
STATUS  AGE\nnode1   Ready   10d"}
{"type": "content_block_delta", "delta": {"text": "The cluster has 1
node in Ready state..."}}
{"type": "message_done", "usage": {"input_tokens": 1500,
"output_tokens": 800}}
```

What MessageProcessor Does:

1. Converts SDK messages to domain `Message` entities
2. Persists each message to `messages` table
3. Persists tool calls to `tool_calls` table
4. Updates session metrics (message_count, total_tokens)
5. Broadcasts to WebSocket subscribers (if any)

Database Inserts:

PROF

```
-- User message
INSERT INTO messages (
    id, session_id, message_type, role, content, sequence_number,
created_at
) VALUES (
    uuid_generate_v4(),
    '770e8400-e29b-41d4-a716-446655440002',
    'user',
    'user',
    'IMPORTANT: This is a READ-ONLY health check...',
    1,
    NOW()
);

-- Assistant message
INSERT INTO messages (
    id, session_id, message_type, role, content, sequence_number,
```

```
created_at
) VALUES (
    uuid_generate_v4(),
    '770e8400-e29b-41d4-a716-446655440002',
    'assistant',
    'assistant',
    'I''ll check the Kubernetes cluster health...',
    2,
    NOW()
);

-- Tool call
INSERT INTO tool_calls (
    id, session_id, message_id, tool_name, tool_input, tool_output,
status, created_at
) VALUES (
    uuid_generate_v4(),
    '770e8400-e29b-41d4-a716-446655440002',
    <message_id>,
    'Bash',
    '{"command": "kubectl get nodes"}',
    'NAME    STATUS  AGE\nnode1   Ready   10d',
    'completed',
    NOW()
);
```

Step 5.4: Update Session Status (lines 228-231)

```
session.transition_to(SessionStatus.ACTIVE)
await self.session_repo.update(session_id, status="active")
await self.db.commit()
```

PROF

Phase 6: Mark Execution Complete

File: app/services/task_service.py:259-270

```
execution.status = TaskExecutionStatus.COMPLETED
execution.completed_at = datetime.utcnow()
execution.result_message_id = message.id

await self.task_execution_repo.update(
    str(execution.id),
    status="completed",
    completed_at=execution.completed_at,
    result_message_id=message.id,
)
```

Database Update:

```
UPDATE task_executions
SET status = 'completed',
    completed_at = NOW(),
    result_message_id = <final_message_id>,
    duration_seconds = EXTRACT(EPOCH FROM (NOW() - started_at)),
    updated_at = NOW()
WHERE id = '660e8400-e29b-41d4-a716-446655440001';
```

Phase 7: Generate Report (Optional)

File: `app/services/task_service.py:272-295`

```
if task.generate_report:
    report_service = ReportService(...)

    report = await report_service.generate_from_session(
        session_id=session.id,
        user_id=task.user_id,
        title=f"Task Execution Report: {task.name}",
        format=task.report_format,  # "markdown"
        auto_generated=True,
    )

    execution.report_id = report.id
```

What This Does:

1. Fetches all messages from session
2. Fetches all tool calls from session
3. Aggregates session metrics
4. Formats content based on `report_format` (markdown/html/json/pdf)
5. Saves report file to `/workspace/.../data/reports/{report_id}.md`
6. Creates record in `reports` table

PROF

Database Inserts:

```
INSERT INTO reports (
    id, session_id, user_id, title, format,
    file_path, auto_generated, created_at
) VALUES (
    '880e8400-e29b-41d4-a716-446655440003',
    '770e8400-e29b-41d4-a716-446655440002',
    '94d9f5a2-1257-43ac-9de2-6d86421455a6',
    'Task Execution Report: Kubernetes Cluster Health Check',
```

```
    'markdown',
    '/workspace/.../data/reports/880e8400.../report.md',
    true,
    NOW()
);

UPDATE task_executions
SET report_id = '880e8400-e29b-41d4-a716-446655440003'
WHERE id = '660e8400-e29b-41d4-a716-446655440001';
```

Phase 8: Audit Logging

File: app/services/task_service.py:299-306

```
await self.audit_service.log_task_executed(
    task_id=task.id,
    execution_id=execution.id,
    user_id=task.user_id,
    trigger_type="manual",
    status="completed",
)
```

Database Insert:

```
INSERT INTO audit_logs (
    id, user_id, action_type, resource_type, resource_id,
    action_details, ip_address, user_agent, created_at
) VALUES (
    uuid_generate_v4(),
    '94d9f5a2-1257-43ac-9de2-6d86421455a6',
    'task.executed',
    'task_execution',
    '660e8400-e29b-41d4-a716-446655440001',
    '{
        "task_id": "75318f39-9feb-4ff6-a381-f377d38fd1be",
        "trigger_type": "manual",
        "status": "completed",
        "session_id": "770e8400-e29b-41d4-a716-446655440002"
    }',
    '192.168.1.100',
    'curl/7.68.0',
    NOW()
);
```

PROF

Phase 9: HTTP Response

File: app/api/v1/tasks.py:303-312

```
response = TaskExecutionResponse.model_validate(execution)
response._links = Links(
    self=f"/api/v1/task-executions/{execution.id}",
    task=f"/api/v1/tasks/{task.id}",
    session=f"/api/v1/sessions/{execution.session_id}",
    report=f"/api/v1/reports/{execution.report_id}",
)

return response  # HTTP 202 Accepted
```

HTTP Response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
  "id": "660e8400-e29b-41d4-a716-446655440001",
  "task_id": "75318f39-9feb-4ff6-a381-f377d38fd1be",
  "session_id": "770e8400-e29b-41d4-a716-446655440002",
  "status": "completed",
  "trigger_type": "manual",
  "variables": {"cluster_name": "production"},
  "created_at": "2025-10-24T02:00:00Z",
  "started_at": "2025-10-24T02:00:01Z",
  "completed_at": "2025-10-24T02:00:45Z",
  "duration_seconds": 44,
  "report_id": "880e8400-e29b-41d4-a716-446655440003",
  "_links": {
    "self": "/api/v1/task-executions/660e8400-e29b-41d4-a716-
446655440001",
    "task": "/api/v1/tasks/75318f39-9feb-4ff6-a381-f377d38fd1be",
    "session": "/api/v1/sessions/770e8400-e29b-41d4-a716-446655440002",
    "report": "/api/v1/reports/880e8400-e29b-41d4-a716-446655440003"
  }
}
```

Claude Code CLI Integration

Official SDK Architecture

```
Our API Service
    ↓ uses
claude-agent-sdk (Python package)
    ↓ spawns
```

```
claude-code (CLI subprocess)
    ↓ calls
Anthropic API
```

How Messages Flow

```
1. API → SDK Client
   client.query("Check cluster health")

2. SDK Client → Claude Code CLI stdin
   {"type": "user_message", "content": "Check cluster health"}

3. Claude Code CLI → Anthropic API
   POST https://api.anthropic.com/v1/messages
   Headers: x-api-key: sk-ant-...
   Body: {model: "claude-3-5-sonnet-20241022", messages: [...]}

4. Anthropic API → Claude Code CLI (streaming)
   data: {"type": "content_block_start"}
   data: {"type": "content_block_delta", "delta": {...}}
   data: {"type": "message_delta", "delta": {...}}

5. Claude Code CLI → SDK Client stdout (streaming)
   {"type": "assistant_message_start"}
   {"type": "content_block_delta", "delta": {...}}
   {"type": "tool_use", "name": "Bash", "input": {...}}

6. SDK Client → MessageProcessor (our code)
   Converts JSON to domain Message entities
   Persists to database
   Broadcasts to WebSockets
```

PROF

Claude Code CLI Command

What Gets Executed:

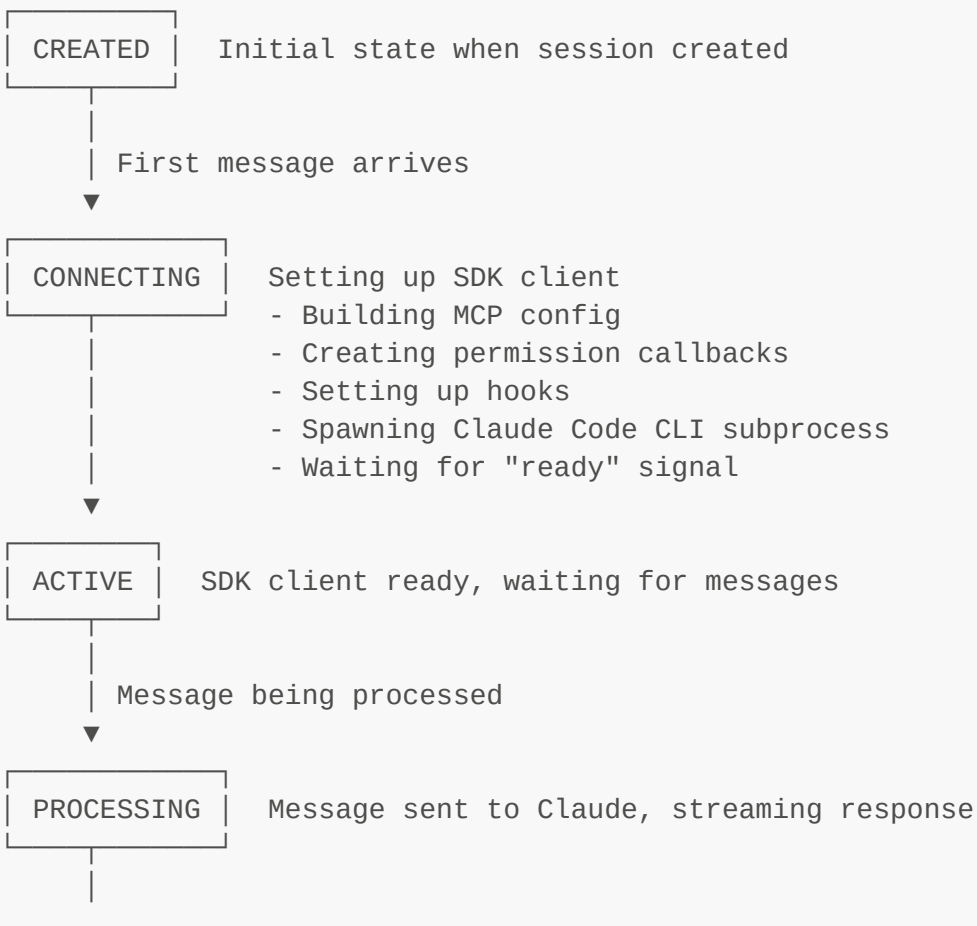
```
# Working directory
cd /workspace/me/repositories/ai-agent/ai-agent-api/data/agent-
workdirs/active/770e8400-e29b-41d4-a716-446655440002/

# Environment
export ANTHROPIC_API_KEY="sk-ant-..."
export KUBECONFIG="/home/msalah/.kube/config"

# Command (simplified - actual has more args)
claude-code \
  --model claude-3-5-sonnet-20241022 \
  --mcp-servers-config /tmp/mcp-config-770e8400.json \
```

```
  --allowed-tools "Read,Write,Bash" \
  --permission-mode default \
  --working-directory .
```

MCP Config File (`/tmp/mcp-config-770e8400.json`):

```
{
  "kubernetes_readonly": {
    "command": "npx",
    "args": ["-y", "@modelcontextprotocol/server-kubernetes"],
    "env": {
      "KUBECONFIG": "/home/msalah/.kube/config"
    }
  },
  "database": {
    "command": "python",
    "args": ["/path/to/mcp_database_server.py"],
    "env": {}
  }
}
```

Session State Transitions



PROF

```
     | Response complete
     ▼
┌──────────┐
│ ACTIVE │  Ready for next message
└──────────┘

(On error: → FAILED)
(On terminate: → TERMINATED)
```

File Locations and Logs

Working Directories

Active Sessions:

```
/workspace/me/repositories/ai-agent/ai-agent-api/data/agent-
workdirs/active/{session_id}/
```

Archived Sessions:

```
/workspace/me/repositories/ai-agent/ai-agent-api/data/agent-
workdirs/archives/{session_id}/
```

Claude Code CLI Session Logs

Location Pattern:

```
/home/msalah/.claude/projects/{project_hash}/{session_id}.jsonl
```

PROF

Project Hash Calculation:

```
working_directory = "/workspace/me/repositories/ai-agent/ai-agent-
api/data/agent-workdirs/active/770e8400-e29b-41d4-a716-446655440002"
project_hash = base64url_encode(working_directory)
             = "-workspace-me-repositories-ai-agent-ai-agent-api-data-
agent-workdirs-active-770e8400-e29b-41d4-a716-446655440002"
```

Full Path Example:

```
/home/msalah/.claude/projects/-workspace-me-repositories-ai-agent-ai-
agent-api-data-agent-workdirs-active-770e8400-e29b-41d4-a716-
```

```
446655440002/770e8400-e29b-41d4-a716-446655440002.jsonl
```

Log Format (JSONL):

```
{"type":"user_message","content":"Check cluster
health","timestamp":"2025-10-24T02:00:01.123Z"}
{"type":"assistant_message_start","id":"msg_123","timestamp":"2025-10-
24T02:00:02.456Z"}
{"type":"content_block_delta","delta":{"text":"I'll
check..."},"timestamp":"2025-10-24T02:00:02.789Z"}
{"type":"tool_use","name":"Bash","input":{"command":"kubectl get
nodes"},"timestamp":"2025-10-24T02:00:03.012Z"}
{"type":"tool_result","tool_use_id":"tool_123","output":"NAME...\nnode1.
..","timestamp":"2025-10-24T02:00:05.345Z"}
```

Generated Reports

Location:

```
/workspace/me/repositories/ai-agent/ai-agent-
api/data/reports/{report_id}/report.{format}
```

Example:

```
/workspace/me/repositories/ai-agent/ai-agent-api/data/reports/880e8400-
e29b-41d4-a716-446655440003/report.md
```

PROF

API Server Logs

Location:

```
/workspace/me/repositories/ai-agent/ai-agent-api/logs/api.log
```

Format:

```
2025-10-24 02:00:00,123 - app.services.task_service - INFO - Starting
task execution {"task_id": "75318f39...", "trigger_type": "manual"}
2025-10-24 02:00:01,456 - app.services.sdk_session_service - INFO -
Setting up SDK client {"session_id": "770e8400..."}
2025-10-24 02:00:02,789 - app.claude_sdk.client_manager - INFO - Claude
SDK client connected successfully {"session_id": "770e8400..."}
```

Monitoring and Debugging

Key Database Tables

1. Task Execution Status:

```
SELECT id, task_id, session_id, status,
       started_at, completed_at, error_message
FROM task_executions
WHERE id = '660e8400-e29b-41d4-a716-446655440001';
```

2. Session Details:

```
SELECT id, status, working_directory_path,
       message_count, total_tokens, sdk_options
FROM sessions
WHERE id = '770e8400-e29b-41d4-a716-446655440002';
```

3. Messages in Session:

```
SELECT id, message_type, role, content, sequence_number, created_at
FROM messages
WHERE session_id = '770e8400-e29b-41d4-a716-446655440002'
ORDER BY sequence_number;
```

4. Tool Calls:

PROF

```
SELECT id, tool_name, tool_input, tool_output, status,
       started_at, completed_at
FROM tool_calls
WHERE session_id = '770e8400-e29b-41d4-a716-446655440002'
ORDER BY created_at;
```

5. Audit Trail:

```
SELECT id, action_type, resource_type, action_details, created_at
FROM audit_logs
WHERE user_id = '94d9f5a2-1257-43ac-9de2-6d86421455a6'
  AND resource_type IN ('task', 'task_execution')
ORDER BY created_at DESC
LIMIT 50;
```

Finding Claude Code CLI Logs

Step 1: Get session working directory from database

```
SELECT working_directory_path
FROM sessions
WHERE id = '770e8400-e29b-41d4-a716-446655440002';
-- Result: /workspace/.../data/agent-workdirs/active/770e8400.../
```

Step 2: Convert to project hash

```
# Replace / with - and remove leading /
# /workspace/... becomes -workspace-...
PROJECT_HASH="-workspace-me-repositories-ai-agent-ai-agent-api-data-agent-workdirs-active-770e8400-e29b-41d4-a716-446655440002"
```

Step 3: Find log file

```
CLAUDE_LOG="/home/msalah/.claude/projects/$PROJECT_HASH/770e8400-e29b-41d4-a716-446655440002.jsonl"

# View log
cat "$CLAUDE_LOG" | jq .

# Watch log in real-time
tail -f "$CLAUDE_LOG" | jq .
```

PROF

Related Documentation

- Tasks API Documentation - Complete API reference
- Session Management - Session lifecycle details
- Claude SDK Integration - SDK architecture
- MCP Configuration - MCP server setup

Document Version: 1.0
Last Updated: 2025-10-24
Author: AI-Agent-API Documentation Team