Sessions Feature Documentation

Last Updated: October 20, 2025
Feature Status: ✅ Fully Implemented
API Version: v1

Table of Contents

PROF

Overview

Sessions are the core feature of the AI Agent API, providing **isolated, stateful execution environments** for interactions with Claude Code through the official Anthropic SDK. Each session maintains its own:

- **Working directory** for file operations
- **Conversation history** (messages and context)
- **Tool execution state** and permissions
- **Cost tracking** and metrics
- **SDK configuration** and settings

Sessions enable:

- ✅ Interactive conversations with Claude Code
- ✅ Persistent file operations across interactions
- ✅ Tool execution with permission management
- ✅ Session forking for branching workflows
- ✅ Cost and usage tracking per session
- ✅ MCP (Model Context Protocol) server integration

What is a Session?

A **Session** represents a complete, isolated interaction environment between a user and Claude Code. Think of it as:

- **Container**: Encapsulates all resources needed for Claude Code execution
- **Workspace**: Provides isolated working directory for file operations
- **Conversation**: Maintains full message history and context
- **State Machine**: Tracks status through well-defined lifecycle states
- **Cost Center**: Accumulates usage metrics and API costs

Key Characteristics

Characteristic	Description
Isolation	Each session has its own working directory, separate from other sessions
Persistence	Messages, tool calls, and files persist throughout session lifecycle
Statefulness	Sessions maintain conversation context and execution state
Traceability	Full audit trail of messages, tool calls, and permission decisions
Resumability	Paused or completed sessions can be resumed or forked

---

PROF

Architecture

Layered Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    API Layer (FastAPI)                        │
│                 app/api/v1/sessions.py                        │
│                    17 REST Endpoints                          │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│                      Service Layer                            │
│  ┌─────────────────────┐    ┌──────────────────────────────┐ │
│  │ SessionService      │→   │ SDKIntegratedSessionService  │ │
│  │ (Business Logic)    │    │ (Claude SDK Integration)     │ │
│  └─────────────────────┘    └──────────────────────────────┘ │
└─────────────────────────────────────────────────────────────┘
```

```
                         ↓
┌──────────────────────────────────────────────────────────────┐
│                   Domain Layer                                │
│  ┌────────────┐  ┌──────────────┐  ┌─────────────────────┐   │
│  │  Session   │  │  Message     │  │   ToolCall          │   │
│  │  (Entity)  │  │ (ValueObj)   │  │ (ValueObj)          │   │
│  └────────────┘  └──────────────┘  └─────────────────────┘   │
└──────────────────────────────────────────────────────────────┘
                         ↓
┌──────────────────────────────────────────────────────────────┐
│                Repository Layer                               │
│  ┌──────────────────┐  ┌──────────────────────────────────┐  │
│  │ SessionRepo      │  │ MessageRepo / ToolCallRepo       │  │
│  │ (Data Access)    │  │ (Data Access)                    │  │
│  └──────────────────┘  └──────────────────────────────────┘  │
└──────────────────────────────────────────────────────────────┘
                         ↓
┌──────────────────────────────────────────────────────────────┐
│                  Persistence Layer                            │
│         PostgreSQL Database (SQLAlchemy ORM)                  │
└──────────────────────────────────────────────────────────────┘
```

Integration Components

```
┌──────────────────────────────────────────────────────────────┐
│              Claude SDK Integration                           │
│  ┌────────────────────┐  ┌─────────────────────────────────┐ │
│  │ ClaudeSDKClient    │  │ PermissionService               │ │
│  │ Manager            │  │ (Tool Access Control)           │ │
│  └────────────────────┘  └─────────────────────────────────┘ │
│  ┌────────────────────┐  ┌─────────────────────────────────┐ │
│  │ MessageProcessor   │  │ EventBroadcaster                │ │
│  │ (Stream Handling)  │  │ (WebSocket Events)              │ │
│  └────────────────────┘  └─────────────────────────────────┘ │
└──────────────────────────────────────────────────────────────┘
```

Supporting Services

- **StorageManager**: Working directory creation and archival
- **AuditService**: Audit logging for session operations
- **MCPConfigBuilder**: Dynamic MCP server configuration
- **MetricsCollector**: Cost and usage tracking

Session Lifecycle

Complete Lifecycle Flow

```
┌─────────────┐
│   CREATED   │  ← Session created, working directory initialized
└─────────────┘
       │ send_message()
       ↓
┌─────────────┐
│ CONNECTING  │  ← SDK client being initialized
└─────────────┘
       │ client ready
       ↓
┌─────────────┐
│   ACTIVE    │  ← Ready for interaction
└─────────────┘
       │ send_message()
       ↓
┌─────────────┐
│ PROCESSING  │  ← Handling message/tool execution
└─────────────┘
       │ complete
       ├────→ ACTIVE (continue)
       │
       ├────→ PAUSED (manual pause)
       │
       ├────→ COMPLETED (natural end)
       │
       ├────→ FAILED (error occurred)
       │
       └────→ TERMINATED (manual termination)
                    ↓
             ┌─────────────┐
             │  ARCHIVED   │  ← Long-term storage
             └─────────────┘
```

PROF

State Transition Rules

From Status	To Status	Trigger	Description
CREATED	CONNECTING	First message sent	Initialize SDK client
CONNECTING	ACTIVE	SDK client ready	Session operational
CONNECTING	FAILED	SDK init error	Session initialization failed
ACTIVE	PROCESSING	Message sent	Processing user message
ACTIVE	PAUSED	Manual pause	Suspend activity
ACTIVE	COMPLETED	Natural completion	Conversation completed
ACTIVE	FAILED	Error	Execution error
ACTIVE	TERMINATED	Manual termination	User stops session

From Status	To Status	Trigger	Description
PROCESSING	ACTIVE	Processing done	Ready for next message
PROCESSING	COMPLETED	Final message	Conversation finished
PROCESSING	FAILED	Error during processing	Processing failed
PAUSED	ACTIVE	Resume	Reactivate session
COMPLETED	ARCHIVED	Archive operation	Move to long-term storage
FAILED	ARCHIVED	Archive operation	Archive failed session
TERMINATED	ARCHIVED	Archive operation	Archive terminated session

Session Status States

CREATED

- **Meaning**: Session entity created, working directory initialized
- **Operations**: Can send first message, can terminate
- **Metadata**: Working directory created but SDK client not yet initialized

CONNECTING

- **Meaning**: Initializing Claude SDK client and setting up MCP servers
- **Operations**: Transitioning to ACTIVE or FAILED
- **Metadata**: SDK client being created with permissions and hooks

ACTIVE

- **Meaning**: Session ready for interaction
- **Operations**: Send messages, pause, terminate
- **Characteristics**: Conversation can continue, all tools available

PROF

PROCESSING

- **Meaning**: Currently executing user message or tool calls
- **Operations**: Waiting for completion
- **Characteristics**: Cannot send new messages until complete

PAUSED

- **Meaning**: Temporarily suspended
- **Operations**: Can resume or terminate
- **Use Case**: User wants to pause workflow without ending session

WAITING

- **Meaning**: Waiting for external input or approval

- **Operations**: Continue or terminate
- **Use Case**: Permission callback requires user decision

COMPLETED

- **Meaning**: Session finished successfully
- **Operations**: Can fork to continue, can archive
- **Characteristics**: No more interaction unless resumed/forked

FAILED

- **Meaning**: Session encountered unrecoverable error
- **Operations**: Can fork to retry, can archive
- **Metadata**: `error_message` field contains failure details

TERMINATED

- **Meaning**: Manually stopped by user
- **Operations**: Can archive, can fork
- **Characteristics**: Deliberate end by user action

ARCHIVED

- **Meaning**: Moved to long-term storage
- **Operations**: Read-only access
- **Characteristics**: Working directory compressed and stored

---

Session Modes

INTERACTIVE

- **Description**: Standard conversational mode
- **Use Case**: Back-and-forth Q&A, code development, debugging
- **Characteristics**:
  - User sends messages and receives responses
  - Full tool access with permission management
  - Session persists conversation context

NON_INTERACTIVE

- **Description**: Single-shot execution mode
- **Use Case**: Batch processing, automated tasks
- **Characteristics**:
  - Execute task and complete
  - Minimal interaction
  - Auto-complete on task finish

FORKED

- **Description**: Branched from parent session
- **Use Case**: Experiment with different approaches, rollback points
- **Characteristics**:
    - Inherits parent configuration
    - Optional working directory copy
    - Independent execution path
    - Tracks `parent_session_id`

Core Components

Session Entity (Domain Model)

Location: `app/domain/entities/session.py`

```
class Session:
    # Identity
    id: UUID
    user_id: UUID
    name: Optional[str]

    # Configuration
    mode: SessionMode
    sdk_options: SDKOptions
    working_directory_path: str

    # State
    status: SessionStatus
    parent_session_id: Optional[UUID]
    is_fork: bool

    # Metrics
    total_messages: int
    total_tool_calls: int
    total_cost_usd: float
    api_input_tokens: int
    api_output_tokens: int

    # Timestamps
    created_at: datetime
    updated_at: datetime
    started_at: Optional[datetime]
    completed_at: Optional[datetime]
```

PROF

Session Service

Location: `app/services/session_service.py`

Responsibilities:

- Business logic for session management
- Quota validation
- Working directory management
- Authorization checks
- Session state transitions

Key Methods:

- `create_session()`: Create new session with validation
- `get_session()`: Retrieve with authorization check
- `pause_session()` / `resume_session()`: Control session state
- `terminate_session()`: End session gracefully
- `fork_session_advanced()`: Create forked session with working directory copy
- `archive_session_to_storage()`: Archive to S3 or filesystem

SDK Integrated Session Service

Location: `app/services/sdk_session_service.py`

Extends: SessionService

Additional Responsibilities:

- Claude SDK client lifecycle management
- Message sending through official SDK
- MCP server configuration
- Permission callback setup
- Hook installation (audit, tracking, cost)

Key Methods:

- `send_message()`: Send message through SDK and stream responses
- `_setup_sdk_client()`: Initialize SDK with permissions, hooks, MCP config
- `cleanup_session_client()`: Disconnect SDK client

Session Repository

Location: `app/repositories/session_repository.py`

Responsibilities:

- Data access layer for session CRUD operations
- Query filtering (by user, status, mode)
- Soft delete support
- Active session counting for quota enforcement

API Endpoints Overview

Core Session Operations (5 endpoints)

Method	Endpoint	Purpose
POST	`/sessions`	Create new session
GET	`/sessions/{session_id}`	Get session details
GET	`/sessions`	List user's sessions
POST	`/sessions/{session_id}/query`	Send message to session
DELETE	`/sessions/{session_id}`	Terminate and clean up session

Session Control (2 endpoints)

Method	Endpoint	Purpose
POST	`/sessions/{session_id}/resume`	Resume paused/completed session
POST	`/sessions/{session_id}/pause`	Pause active session

Data Access (3 endpoints)

Method	Endpoint	Purpose
GET	`/sessions/{session_id}/messages`	List session messages
GET	`/sessions/{session_id}/tool-calls`	List session tool calls
GET	`/sessions/{session_id}/workdir/download`	Download working directory as tar.gz

Advanced Operations (7 endpoints)

Method	Endpoint	Purpose
POST	`/sessions/{session_id}/fork`	Fork existing session
POST	`/sessions/{session_id}/archive`	Archive working directory
GET	`/sessions/{session_id}/archive`	Get archive metadata
GET	`/sessions/{session_id}/hooks`	Get hook execution history
GET	`/sessions/{session_id}/permissions`	Get permission decision history
GET	`/sessions/{session_id}/metrics/snapshots`	Get historical metrics
GET	`/sessions/{session_id}/metrics/current`	Get current session metrics

PROF

Session Management

Creating a Session

Endpoint: `POST /api/v1/sessions`

Inputs:

- `template_id` (optional): Use session template as base
- `name` (optional): Human-readable session name
- `working_directory` (optional): Custom working directory path
- `allowed_tools` (optional): Tool access patterns (glob)
- `system_prompt` (optional): Custom system prompt for Claude
- `sdk_options` (optional): Claude SDK configuration overrides
- `metadata` (optional): Custom key-value data

Background Processing:

1. **Quota Validation**: Check user hasn't exceeded `max_concurrent_sessions`
2. **Template Loading** (if `template_id` provided):
   - Fetch template from database
   - Increment template usage counter
   - Merge template config with request overrides
3. **SDK Options Building**:
   - Parse allowed_tools, system_prompt
   - Create `SDKOptions` value object
   - Set defaults (model: claude-3-5-sonnet-20241022, max_turns: 20)
4. **Session Entity Creation**:
   - Generate UUID
   - Set mode to INTERACTIVE
   - Set status to CREATED
5. **Working Directory Creation**:
   - Create isolated directory at `data/agent-workdirs/active/{session_id}`
   - Set permissions
   - Store path in session
6. **Database Persistence**:
   - Convert domain entity to SQLAlchemy model
   - Insert into `sessions` table
   - Flush and commit transaction
7. **Audit Logging**:
   - Log session creation event
   - Record user_id, mode, sdk_options

PROF

Outputs:

- `SessionResponse` with:
  - Session ID (UUID)
  - Initial status: "created"
  - Working directory path
  - SDK configuration
  - HATEOAS links (self, query, messages, stream)
  - Metrics (all zeros initially)

Example:

```
{
  "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "user_id": "94d9f5a2-1257-43ac-9de2-6d86421455a6",
  "name": "Debug API Issue",
  "status": "created",
  "working_directory": "/data/agent-workdirs/active/f47ac10b-58cc-4372-
a567-0e02b2c3d479",
  "allowed_tools": ["*"],
  "message_count": 0,
  "tool_call_count": 0,
  "total_cost_usd": 0.0,
  "_links": {
    "self": "/api/v1/sessions/f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "query": "/api/v1/sessions/f47ac10b-58cc-4372-a567-
0e02b2c3d479/query",
    "stream": "/api/v1/sessions/f47ac10b-58cc-4372-a567-
0e02b2c3d479/stream"
  }
}
```

Getting Session Details

Endpoint: `GET /api/v1/sessions/{session_id}`

Background Processing:

1. **Database Query**: Fetch session by ID (exclude soft-deleted)
2. **Authorization Check**:
   - If session.user_id matches current user → Allow
   - If current user is admin → Allow
   - Otherwise → 403 Forbidden
3. **Entity to Response Conversion**:
   - Use `session_to_response()` mapper
   - Convert Enum statuses to strings
   - Parse sdk_options for allowed_tools
   - Handle numeric type conversions

PROF

Outputs: Full `SessionResponse` with current metrics

Listing Sessions

Endpoint: `GET /api/v1/sessions?status={status}&is_fork={bool}&page={n}&page_size={n}`

Background Processing:

1. **Database Query**:
   - Filter by current user_id
   - Apply pagination (offset/limit)

    - Order by created_at DESC
2. **Post-Query Filtering** (in-memory):
    - Filter by status if provided
    - Filter by is_fork if provided
3. **Count Total**: For pagination metadata
4. **Response Building**:
    - Convert each session to SessionResponse
    - Add HATEOAS links
    - Wrap in PaginatedResponse

Outputs:

```
{
  "items": [...sessions...],
  "total": 25,
  "page": 1,
  "page_size": 10,
  "pages": 3
}
```

Deleting a Session

Endpoint: `DELETE /api/v1/sessions/{session_id}`

Background Processing:

1. **Authorization Check**: Verify ownership or admin role
2. **SDK Client Cleanup**: Disconnect Claude SDK client if active
3. **Working Directory Archival**:
    - If working directory exists → Archive to storage
    - Compress as tar.gz
    - Move to archives directory
4. **Soft Delete**:
    - Set `deleted_at` timestamp
    - Keep record in database
    - Exclude from future queries
5. **Audit Log**: Record deletion event

PROF

Outputs: 204 No Content (successful deletion)

Message Handling

Sending a Message

Endpoint: `POST /api/v1/sessions/{session_id}/query`

Inputs:

- `message`: User message text (1-50,000 chars)
- `fork`: Boolean - fork session before sending message

Background Processing Flow:

1. Session Validation

- Fetch session from database
- Check authorization (ownership or admin)
- Validate status (must be CREATED, ACTIVE, or CONNECTING)

2. Status Transition: CREATED → CONNECTING (First message only)

- Update database: status = "connecting"
- Commit transaction

3. SDK Client Setup (First message only)

MCP Configuration Building:

```
┌──────────────────────────────────────────┐
│  MCPConfigBuilder.build_session_mcp_config()│
└──────────────────────────────────────────┘
            │
            │
            ├──→ SDK MCP Servers (3 servers, 7 tools):
            │    - kubernetes_readonly (3 tools)
            │    - database (2 tools)
            │    - monitoring (2 tools)
            │
            ├──→ User Personal MCP Servers:
            │    - Query user's MCP servers from DB
            │    - Filter by user_id, enabled=true
            │
            └──→ Global MCP Servers:
                 - Query organization-wide servers
                 - Filter by is_global=true, enabled=true
```

PROF

Permission Callback Creation:

```
permission_callback = permission_service.create_permission_callback(
    session_id=session.id,
    user_id=user_id
)
# Callback evaluates tool access against:
# - allowed_tools patterns
# - user permissions
# - custom policies
```

Hook Installation:

```
hooks = {
    "PreToolUse": [
        audit_hook,            # Log tool execution
        tool_tracking_hook     # Record tool call in DB
    ],
    "PostToolUse": [
        audit_hook,
        tool_tracking_hook,
        cost_tracking_hook     # Update session costs
    ]
}
```

SDK Client Creation:

- Create ClaudeSDKClient with session config
- Set permission callback
- Install hooks
- Store in ClientManager cache

4. Status Transition: CONNECTING → ACTIVE

- Update database: status = "active", started_at = now()
- Commit transaction

5. Status Transition: ACTIVE → PROCESSING

- Update database: status = "processing"
- Commit transaction

PROF

6. Message Sending Through SDK

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

7. Response Stream Processing

MessageProcessor Flow:

```
SDK Response Stream
      ↓
┌─────────────────┐
│ MessageProcessor│
└────────┬────────┘
         │
```

```
            ├──→ Parse message type (user/assistant/tool_use/tool_result)
            ├──→ Create Message value object
            ├──→ Persist to messages table
            ├──→ Update session metrics (total_messages++, token counts)
            ├──→ Broadcast to WebSocket subscribers
            ├──→ Yield to API caller (async generator)
            └──→ Continue until stream ends
```

8. Status Transition: PROCESSING → ACTIVE

- Update database: status = "active"
- Commit transaction

9. Error Handling

If any error occurs:

- Catch exception
- Transition to FAILED status
- Set error_message field
- Commit transaction
- Re-raise exception

Outputs:

```
{
  "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "status": "active",
  "parent_session_id": null,
  "is_fork": false,
  "message_id": "b8e5c3a1-9d7f-4e6b-8c3a-1f2e3d4c5b6a",
  "_links": {
    "self": "/api/v1/sessions/f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "message": "/api/v1/sessions/f47ac10b-58cc-4372-a567-
0e02b2c3d479/messages/b8e5c3a1-9d7f-4e6b-8c3a-1f2e3d4c5b6a",
    "stream": "/api/v1/sessions/f47ac10b-58cc-4372-a567-
0e02b2c3d479/stream"
  }
}
```

PROF

Session Control

Pausing a Session

Endpoint: `POST /api/v1/sessions/{session_id}/pause`

Use Cases:

- User needs to step away
- Want to suspend without losing state
- Planning to resume later

Background Processing:

1. Validate session is ACTIVE
2. Transition to PAUSED status
3. SDK client remains connected but inactive

Outputs: SessionResponse with status="paused"

Resuming a Session

Endpoint: `POST /api/v1/sessions/{session_id}/resume`

Inputs:

- `fork`: Boolean - fork before resuming instead of resuming in-place

Background Processing:

If fork=false:

1. Validate session is PAUSED or COMPLETED
2. Check not in terminal state (FAILED, TERMINATED, ARCHIVED)
3. Transition to ACTIVE status
4. Session can accept new messages

If fork=true:

1. Call `fork_session_advanced()` instead
2. Create new session with same config
3. Copy working directory contents
4. Return new forked session

PROF

Outputs: SessionResponse (original or new forked session)

Data Access

Listing Messages

Endpoint: `GET /api/v1/sessions/{session_id}/messages?limit={n}`

Background Processing:

1. Authorization check
2. Query messages table: `session_id = ? ORDER BY created_at DESC LIMIT ?`
3. Convert Message models to MessageResponse
4. Return array (newest first)

Message Types:

- `user`: User-sent messages
- `assistant`: Claude's responses
- `system`: System messages
- `result`: Final execution results

Listing Tool Calls

Endpoint: `GET /api/v1/sessions/{session_id}/tool-calls?limit={n}`

Background Processing:

1. Authorization check
2. Query tool_calls table: `session_id = ? ORDER BY created_at DESC LIMIT ?`
3. Convert ToolCall models to ToolCallResponse
4. Include permission decision and execution metrics

Tool Call Fields:

- `tool_name`: Which tool was called
- `tool_input`: Input parameters
- `tool_output`: Execution result
- `status`: pending/success/error
- `permission_decision`: allow/deny
- `duration_ms`: Execution time

Downloading Working Directory

Endpoint: `GET /api/v1/sessions/{session_id}/workdir/download`

Background Processing:

1. Authorization check
2. Verify working directory exists
3. Create temporary tar.gz archive:
   - Compress entire working directory
   - Write to temp file
4. Stream file to client:
   - Set Content-Type: application/gzip
   - Set Content-Disposition: attachment
   - Stream file chunks
5. Schedule cleanup: Delete temp file after 10 seconds

PROF

Outputs: Binary tar.gz file stream

Advanced Features

Session Forking

Endpoint: `POST /api/v1/sessions/{session_id}/fork`

Use Cases:

- Try different approaches without losing original
- Create checkpoint/rollback points
- Experiment with parameters

Inputs:

- `name`: Name for forked session
- `fork_at_message`: Fork from specific message index (optional)
- `include_working_directory`: Copy files (default: true)

Background Processing:

1. Get parent session
2. Create new session with mode=FORKED
3. Copy SDK options from parent
4. Set parent_session_id and is_fork=true
5. If include_working_directory:
    - Recursively copy all files from parent workdir
    - Preserve directory structure
    - Copy to new session's workdir
6. Log fork action to audit

Outputs: SessionResponse for new forked session

Session Archival

Endpoint: `POST /api/v1/sessions/{session_id}/archive`

Inputs:

PROF

- `upload_to_s3`: Upload to S3 or local filesystem (default: true)
- `compression`: Compression algorithm (default: "gzip")

Background Processing:

1. Get session and validate working directory exists
2. Create StorageArchiver:
    - Provider: s3 or filesystem (based on settings)
    - Configure bucket/region for S3
3. Archive working directory:
    - Create tar.gz of all files
    - Generate manifest (file list with sizes)
    - Upload to storage
    - Calculate total size
4. Store archive metadata in database:
    - archive_path (S3 URL or file path)

    - size_bytes
    - manifest (JSON)
    - status ("completed")
    - archived_at timestamp
5. Audit log archival

Outputs:

```
{
  "id": "...",
  "session_id": "...",
  "archive_path": "s3://bucket/archives/session-{id}.tar.gz",
  "size_bytes": 1048576,
  "compression": "gzip",
  "manifest": {
    "files": [
      {"path": "main.py", "size": 1024},
      {"path": "data.json", "size": 512}
    ],
    "total_files": 2
  },
  "status": "completed",
  "archived_at": "2025-10-20T10:30:00Z"
}
```

Hook Execution History

Endpoint: `GET /api/v1/sessions/{session_id}/hooks?limit={n}`

Background Processing:

1. Authorization check
2. Query hook_executions table for session
3. Return list of hook executions with:
    - hook_type (PreToolUse/PostToolUse)
    - tool_use_id
    - input/output data
    - continue_execution flag
    - duration

Use Case: Debug hook behavior, audit tool execution

Permission Decision History

Endpoint: `GET /api/v1/sessions/{session_id}/permissions?limit={n}`

Background Processing:

1. Authorization check

PROF

2. Query permission_decisions table for session
3. Return list of decisions with:
   - tool_name
   - input_data
   - decision (allow/deny)
   - reason
   - interrupted flag

Use Case: Understand why tools were allowed/denied

Metrics Snapshots

Endpoint: `GET /api/v1/sessions/{session_id}/metrics/snapshots?limit={n}`

Background Processing:

1. Authorization check
2. Query metrics_snapshots table for time-series data
3. Return historical snapshots showing metrics over time

Use Case: Track cost/usage trends during session

Current Metrics

Endpoint: `GET /api/v1/sessions/{session_id}/metrics/current`

Background Processing:

1. Authorization check
2. Fetch real-time metrics from MetricsCollector
3. Return current metrics:
   - total_messages, total_tool_calls
   - total_cost_usd
   - token counts (input/output/cache)
   - error/retry counts

PROF

SDK Integration

Claude SDK Client Lifecycle

```
Session Created (CREATED)
        ↓
First Message Sent
        ↓
_setup_sdk_client() called
        ↓
┌───────────────────────────────┐
│  1. Build MCP Configuration   │
│     - Merge SDK + User MCP    │
│      - Update session config  │
└───────────────────────────────┘
            │
            ↓
┌───────────────────────────────┐
│  2. Create Permission Callback│
│      - Link to PermissionService│
└───────────────────────────────┘
            │
            ↓
┌───────────────────────────────┐
│  3. Install Hooks             │
│      - PreToolUse: audit, tracking│
│      - PostToolUse: audit, tracking, cost│
└───────────────────────────────┘
            │
            ↓
┌───────────────────────────────┐
│  4. Create SDK Client         │
│      - Store in ClientManager │
│      - Cache by session_id    │
└───────────────────────────────┘
            │
            ↓
Status: CONNECTING → ACTIVE
        ↓
Client Ready for Messages
        ↓
...interaction...
        ↓
Session Terminated/Deleted
        ↓
cleanup_session_client() called
        ↓
SDK Client Disconnected
```

MCP Configuration

Sessions automatically integrate with **MCP (Model Context Protocol) servers**:

Built-in SDK Tools (7 tools across 3 servers):

1. `kubernetes_readonly`: cluster info, pod status, logs
2. `database`: query, schema inspection
3. `monitoring`: metrics, health checks

User Personal MCP Servers:

- Configured per-user in database
- Added to session's MCP config automatically

Global MCP Servers:

- Organization-wide servers
- Available to all users

Dynamic Configuration:

```
# MCP config built at session start
mcp_config = {
    "kubernetes_readonly": {...},
    "database": {...},
    "monitoring": {...},
    "user_mcp_server_1": {...},
    "global_mcp_server_1": {...}
}
```

Permission System

Permission Callback Flow

```
Tool Execution Requested
        ↓
┌─────────────────────────────────┐
│  Permission Callback Triggered  │
└────────────┬────────────────────┘
             ↓
┌─────────────────────────────────┐
│  PermissionService.evaluate()   │
│  - Check allowed_tools patterns │
│  - Check custom policies        │
│  - Check user permissions       │
└────────────┬────────────────────┘
             ↓
        Decision Made
        ↙         ↘
    ALLOW          DENY
      ↓              ↓
Execute Tool    Block Tool
      ↓              ↓
Record Decision  Record Decision
      ↓              ↓
Continue      Return Error
```

PROF

Permission Modes

- **default**: Standard permission checking
- **strict**: Deny by default, explicit allows only
- **permissive**: Allow most tools, deny dangerous ones
- **custom**: Use custom policy rules

Cost Tracking

Cost Accumulation

Per Message:

- Input tokens × input price
- Output tokens × output price
- Cache creation tokens × cache price
- Cache read tokens × cache read price

Token Pricing (model-specific):

```
# claude-3-5-sonnet-20241022
input_price_per_1k = 0.003
output_price_per_1k = 0.015
cache_creation_per_1k = 0.00375
cache_read_per_1k = 0.0003
```

Hooks Integration:

```
# PostToolUse hook updates costs
await cost_tracking_hook(event)
# → Parse token usage from SDK response
# → Calculate cost
# → Update session.total_cost_usd
# → Update session.api_*_tokens counters
```

Accessing Costs:

PROF

- Session response includes `total_cost_usd`
- Real-time: GET `/sessions/{id}/metrics/current`
- Historical: GET `/sessions/{id}/metrics/snapshots`
- User-level: GET `/monitoring/costs/user/{user_id}`

Working Directories

Structure

```
data/agent-workdirs/
├── active/
│   ├── {session-id-1}/
│   │   ├── main.py
│   │   ├── data.json
│   │   └── output/
│   │       └── results.txt
│   └── {session-id-2}/
│       └── ...
└── archives/
    ├── {session-id-1}.tar.gz
    └── {session-id-2}.tar.gz
```

Lifecycle

1. **Creation**: Session created → Working directory created at `data/agent-workdirs/active/{session_id}`
2. **Usage**: Claude Code writes/reads files during execution
3. **Persistence**: Files persist throughout session lifecycle
4. **Archival**: Session archived → Directory compressed to tar.gz → Moved to archives/
5. **Cleanup**: Session deleted → Directory archived → Can be removed after retention period

File Operations

Claude Code can:

- **Read**: Access all files in working directory
- **Write**: Create new files, modify existing files
- **Execute**: Run scripts/commands in working directory
- **Navigate**: cd into subdirectories

All file operations are **isolated** per session - no cross-session file access.

Error Handling

Common Errors

PROF

Error	Status Code	Cause	Resolution
Session Not Found	404	Invalid session_id or deleted session	Verify session ID
Not Authorized	403	User doesn't own session and not admin	Check ownership
Session Not Active	409	Trying to send message to inactive session	Resume or fork session
Quota Exceeded	429	Too many concurrent sessions	Close unused sessions
SDK Error	500	Claude SDK client error	Check logs, retry

Error Response Format

```
{
  "detail": "Session f47ac10b-58cc-4372-a567-0e02b2c3d479 not found"
}
```

Failed Session Handling

When session fails:

1. Status → FAILED
2. `error_message` field populated
3. SDK client disconnected
4. Can still access messages/tool calls (read-only)
5. Can fork to retry

Security and Authorization

Ownership Model

- **Regular Users**: Can only access their own sessions
- **Admin Users**: Can access all sessions

Authorization Checks

Every endpoint performs:

```
if session.user_id != current_user.id and current_user.role != "admin":
    raise HTTPException(status_code=403, detail="Not authorized")
```

Soft Delete

PROF

Deleted sessions:

- Set `deleted_at` timestamp
- Excluded from queries (`deleted_at IS NULL`)
- Data retained for audit/recovery
- Can be hard-deleted later by admin

Common Use Cases

1. Interactive Development Session

```
1. POST /sessions → Create session
2. POST /sessions/{id}/query {"message": "Create a Flask app"}
3. POST /sessions/{id}/query {"message": "Add authentication"}
```

```
4. GET /sessions/{id}/workdir/download → Download code
5. DELETE /sessions/{id} → Clean up
```

2. Experiment with Forking

```
1. POST /sessions → Create session
2. POST /sessions/{id}/query {"message": "Implement feature X"}
3. POST /sessions/{id}/fork → Create fork
4. POST /sessions/{fork_id}/query {"message": "Try approach Y"}
   (Compare results between original and fork)
```

3. Long-Running Session with Pausing

```
1. POST /sessions → Create session
2. POST /sessions/{id}/query {"message": "Start complex task"}
3. POST /sessions/{id}/pause → Pause for review
   (Review results, decide next steps)
4. POST /sessions/{id}/resume → Continue
5. POST /sessions/{id}/query {"message": "Continue with..."}
```

4. Session Archival for Compliance

```
1. POST /sessions → Create session
2. ...complete work...
3. POST /sessions/{id}/archive → Archive to S3
4. GET /sessions/{id}/archive → Get archive metadata
   (Archive stored for compliance/audit)
```

PROF

Troubleshooting

Session Stuck in CONNECTING

Cause: SDK client initialization failed
Solution:

1. Check logs for SDK errors
2. Verify MCP server configurations
3. Terminate and recreate session

Permission Denied on Tool Execution

Cause: Tool not in allowed_tools or custom policy denies
Solution:

1. GET /sessions/{id}/permissions → Check decision history
2. Update session's allowed_tools pattern
3. Or update custom policies

High Costs

Cause: Large context or many tool calls
Solution:

1. GET /sessions/{id}/metrics/current → Check token usage
2. GET /sessions/{id}/tool-calls → Review tool call frequency
3. Optimize prompts to reduce tokens
4. Use caching for repeated context

Working Directory Lost

Cause: Session deleted without archival
Solution:

- Always archive before deleting if files needed
- Download working directory first

Related Files

API Layer

- `app/api/v1/sessions.py` - All 17 endpoints (1034 lines)

Service Layer

PROF

- `app/services/session_service.py` - Business logic (509 lines)
- `app/services/sdk_session_service.py` - SDK integration (389 lines)

Domain Layer

- `app/domain/entities/session.py` - Session entity (209 lines)
- `app/domain/value_objects/message.py` - Message value object
- `app/domain/value_objects/tool_call.py` - ToolCall value object

Repository Layer

- `app/repositories/session_repository.py` - Data access (246 lines)
- `app/repositories/message_repository.py` - Message persistence
- `app/repositories/tool_call_repository.py` - Tool call persistence

Schemas

- `app/schemas/session.py` - Request/response models (228 lines)
- `app/schemas/mappers.py` - Entity/model conversions (78 lines)

Infrastructure

- `app/claude_sdk/` - SDK client management, permissions, hooks
- `app/services/storage_manager.py` - Working directory management
- `app/services/audit_service.py` - Audit logging
- `app/mcp/config_builder.py` - MCP configuration

Document Version: 1.0
Last Reviewed: October 20, 2025
Next Review: After Phase 2 testing completion