Claude Code Integration Points Analysis

Overview

Analysis of all endpoints that integrate with Claude Code CLI to determine which require refactoring for async execution.

Integration Points

1. Sessions API: `/api/v1/sessions/{id}/query`

File: `app/api/v1/sessions.py:365-459`

Usage Pattern: Interactive Chat

```
@router.post("/{session_id}/query")
async def send_message(...):
    # Sends message and waits for Claude response
    async for message in service.send_message(
        session_id=session_id,
        user_id=current_user.id,
        message_text=request.message,
    ):
        last_message = message

    return SessionQueryResponse(...)  # Returns after completion
```

Behavior: **SYNCHRONOUS** (blocks until response)

User Expectation: ✅ **Correct** - Users expect to wait

PROF

- Similar to ChatGPT interface
- Real-time conversation
- Immediate feedback needed
- WebSocket streaming available for progress

Recommendation: **NO CHANGES NEEDED**

- Keep synchronous behavior
- This is the expected UX for interactive sessions
- Users are actively waiting for responses

---

2. Tasks API: `/api/v1/tasks/{id}/execute`

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

Usage Pattern: Automation & Background Jobs

```
@router.post("/{task_id}/execute")
async def execute_task(...):
    # Creates session, sends message, waits for completion
    execution = await service.execute_task(
        task_id=task_id,
        trigger_type="manual",
        variables=request.variables,
    )

    return TaskExecutionResponse(...)  # Returns after completion
```

Behavior: **SYNCHRONOUS** (blocks until completion)

User Expectation: × **INCORRECT** - Users expect fire-and-forget

- Automated workflows
- Long-running operations (K8s health checks, reports)
- Scheduled tasks
- No need for immediate response

Recommendation: **REFACTOR TO ASYNC** (via Celery)

- Queue task execution to Celery
- Return immediately with status="queued"
- Background worker processes task
- Poll for status or use webhooks

---

3. WebSocket API: `/ws/sessions/{id}`

File: `app/api/v1/websocket.py`

PROF

Usage Pattern: Real-time Streaming

```
@router.websocket("/ws/sessions/{session_id}")
async def session_websocket(...):
    # Streams messages in real-time
    async for message in service.send_message(...):
        await websocket.send_json(message)
```

Behavior: **STREAMING** (progressive updates)

User Expectation: ✅ **Correct** - Real-time updates

- Live streaming of Claude responses
- Progressive display of tool executions

- Interactive monitoring

Recommendation: **NO CHANGES NEEDED**

- Streaming is ideal for this use case
- Already provides real-time feedback

Shared Infrastructure

SDKIntegratedSessionService.send_message()

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

This method is used by **ALL** integration points:

```
async def send_message(
    self,
    session_id: UUID,
    user_id: UUID,
    message_text: str,
) -> AsyncIterator[Message]:
    """Send message to Claude and stream responses."""
    # 1. Setup SDK client
    # 2. Send to Claude
    # 3. Process response stream
    # 4. Persist to database
    # 5. Broadcast to WebSocket subscribers
```

Design Decision: **KEEP AS-IS**

Rationale:

PROF

- ✅ Correct behavior for interactive sessions
- ✅ Already streams responses (good for WebSocket)
- ✅ Can be wrapped in Celery for tasks (no changes needed)
- ✅ Single source of truth for Claude integration

Usage by endpoint:

- Sessions: Calls directly (synchronous wait is expected)
- Tasks: Will wrap in Celery task (async via queue)
- WebSocket: Streams responses (progressive updates)

Refactoring Strategy

What TO Refactor

Tasks Execution Only:

```
OLD:
POST /tasks/{id}/execute
    ↓ BLOCKS
TaskService.execute_task()
    ↓
SessionService.send_message()
    ↓ BLOCKS 60-120s
Claude Code execution
    ↓
HTTP 202 response

NEW:
POST /tasks/{id}/execute
    ↓
Queue to Celery
    ↓ IMMEDIATE
HTTP 202 response (status=queued)

[Background Worker]
    ↓
SessionService.send_message()
    ↓ BLOCKS (but in worker, not HTTP thread)
Claude Code execution
    ↓
Update status=completed
```

What NOT to Refactor

Sessions API:

- ✅ Keep synchronous behavior
- ✅ Users expect to wait
- ✅ WebSocket provides streaming alternative
- ✅ No changes needed

PROF

Core Integration:

- ✅ `SDKIntegratedSessionService.send_message()` stays synchronous
- ✅ `ClaudeSDKClientManager` stays as-is
- ✅ Message processing stays as-is
- ✅ All repositories stay as-is

Impact Analysis

Minimal Changes Required

Only these files need modification:

1. **NEW**: `app/celery_app.py` - Celery application
2. **NEW**: `app/celery_tasks/task_execution.py` - Background task
3. **MODIFY**: `app/services/task_service.py` - Add async execution method
4. **MODIFY**: `app/api/v1/tasks.py` - Queue tasks instead of executing
5. **MODIFY**: `app/domain/entities/task_execution.py` - Add QUEUED status

No Changes Needed

- ✅ `app/services/sdk_session_service.py` - Reused as-is
- ✅ `app/api/v1/sessions.py` - No changes
- ✅ `app/api/v1/websocket.py` - No changes
- ✅ `app/claude_sdk/` - All SDK integration code stays same
- ✅ `app/repositories/` - All repositories stay same
- ✅ Database schema - Only add celery_task_id field

Code Reuse Pattern

The Celery task will **reuse** existing code:

```python
# app/celery_tasks/task_execution.py

@celery_app.task
def execute_task_async(execution_id, task_id, user_id, variables):
    """Background task execution."""

    # 1. Initialize services (same as API endpoint)
    session_service = SDKIntegratedSessionService(...)
    task_service = TaskService(...)

    # 2. Call existing execution logic
    # REUSES the same code that sessions API uses!
    result = await session_service.send_message(
        session_id=session_id,
        user_id=user_id,
        message_text=rendered_prompt,
    )

    # 3. Update execution status
    await task_execution_repo.update(
        execution_id,
        status="completed",
    )
```

PROF

Benefits:

- ✅ No duplication of business logic
- ✅ No changes to core integration
- ✅ Sessions and tasks use same code paths

- ✅ Easy to test
- ✅ Maintainable

Testing Impact

No Regression Risk for Sessions

Since sessions API is **unchanged**:

- ✅ Existing session tests still pass
- ✅ No risk to interactive functionality
- ✅ WebSocket streaming unaffected
- ✅ Claude Code integration unchanged

New Tests Only for Tasks

Only need new tests for:

- Task queuing to Celery
- Background execution
- Status transitions (QUEUED → RUNNING → COMPLETED)
- Error handling in workers

Migration Path

Phase 1: Add Async Option (Week 1)

Add `execution_mode` parameter to tasks:

```
POST /api/v1/tasks/{id}/execute
{
  "variables": {...},
  "execution_mode": "async"  // NEW: "async" or "sync"
}
```

PROF

- Default: `async` (queue to Celery)
- Fallback: `sync` (original behavior)
- **Sessions API**: Unaffected

Phase 2: Monitor and Tune (Week 2)

- Monitor Celery workers
- Tune concurrency settings
- Verify no impact on sessions
- **Sessions API**: Unaffected

Phase 3: Deprecate Sync Mode (Week 3+)

- Make `async` the only option
- Remove `sync` fallback
- **Sessions API**: Still unaffected

Conclusion

Summary

- **Sessions API**: Perfect as-is (synchronous for interactive use)
- **Tasks API**: Needs async refactoring (background execution)
- **Core Integration**: No changes needed (reused by both)

Key Insight

The synchronous behavior is **NOT a bug** for sessions - it's the **correct design** for interactive chat.

The issue only affects tasks where users expect fire-and-forget behavior.

Refactoring Scope

Very Limited:

- Only task execution needs changes
- Core Claude Code integration stays same
- Sessions API completely untouched
- Minimal risk of regression

Document Created: 2025-10-24
Related: `celery-background-tasks-integration.md`
Status: Analysis Complete