Week 6: Agent SDK - Running Agents Programmatically

01. Session Goals

- Understand the Claude Agent SDK architecture
- Use the `query()` function to run agents in TypeScript and Python
- Build headless agents for automation
- Handle sessions, permissions, and error recovery
- Deploy agents in sandboxed environments with Daytona

02. Block 1: Theory - The Agent SDK (30 min)

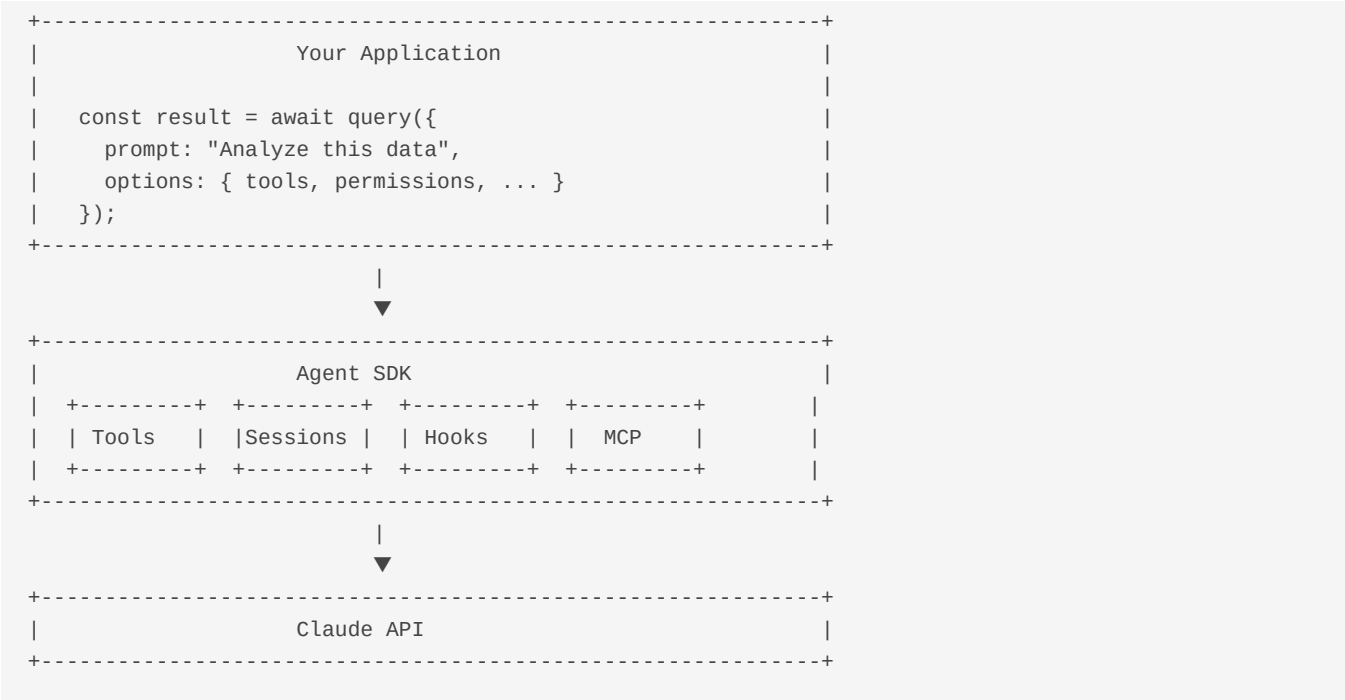
Why Run Agents Programmatically?
Claude Code is interactive. Great for developers. But for production:

- Need to run without human in the loop
- Need to process batches of tasks
- Need to integrate into existing systems
- Need to schedule and automate

The Agent SDK lets you embed Claude's agent capabilities in your code.

SDK Architecture

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.



The `query()` Function

The core of the SDK. One function to run an agent. Available in both TypeScript and Python:

TypeScript:

```
import { query } from "@anthropic-ai/claude-agent-sdk";

const result = await query({
  prompt: "Research Acme Corp and summarize findings",
  options: {
    maxTurns: 10,
    systemPrompt: "You are a research analyst...",
  }
});

console.log(result.text);
```

Python:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions

async def main():
    async for message in query(
        prompt="Research Acme Corp and summarize findings",
        options=ClaudeAgentOptions(
            max_turns=10,
            system_prompt="You are a research analyst..."
        )
    ):
        if hasattr(message, "result"):
            print(message.result)

asyncio.run(main())
```

Key difference: Python uses async iterators and streams by default, while TypeScript can await the final result directly.

Key Parameters

Parameter	Type	Description
`prompt`	string	The task for the agent
`options.maxTurns`	number	Maximum agentic loops
`options.systemPrompt`	string	Custom system instructions
`options.model`	string	Model to use (see below)
`options.tools`	Tool[]	Available tools
`options.mcpServers`	McpServer[]	MCP connections
`options.permissions`	Permission[]	Auto-granted permissions
`options.abortController`	AbortController	Cancellation signal

Model Selection

Choose the right model for your use case:

Model	ID	Best For	Cost
Sonnet 4.5	`claude-sonnet-4-5-20250514`	Most agent work (default)	$$
Opus 4.5	`claude-opus-4-5-20250514`	Complex reasoning	$$$$
Haiku	`claude-haiku-3-5-20241022`	Fast, simple tasks	$

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
const result = await query({
  prompt: "Complex analysis task",
  options: {
    model: "claude-opus-4-5-20250514",  // Use Opus for hard problems
    maxTurns: 10,
  }
});
```

Guidelines:

- Use Sonnet 4.5 for most production workloads (best balance of capability and cost)
- Use Opus 4.5 when you need stronger reasoning or the task is failing with Sonnet
- Use Haiku for sub-agents, simple lookups, or high-volume batch processing

Reference: Claude Models Overview ■

Built-in Tools
The SDK includes Claude Code's tools:

Tool	Purpose
`Read`	Read files
`Write`	Write files
`Edit`	Edit files
`Bash`	Execute commands
`Glob`	Find files by pattern
`Grep`	Search file contents
`WebSearch`	Search the web
`WebFetch`	Fetch web pages

Custom Tools
Beyond built-in tools, you can define your own. Custom tools in the Agent SDK are implemented as in-process MCP servers. This is how you connect Claude to any API, database, or service.

Creating a custom tool:

Custom tools require the `createSdkMcpServer` and `tool` helper functions. Use Zod for type-safe schemas:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import { query, tool, createSdkMcpServer } from "@anthropic-ai/claude-agent-sdk";
import { z } from "zod";

// Create an MCP server with custom tools
const crmServer = createSdkMcpServer({
  name: "crm-tools",
  version: "1.0.0",
  tools: [
    tool(
      "search_crm",
      "Search the CRM for contacts or companies by name",
      {
        searchQuery: z.string().describe("Search query (name, company, or email)"),
        recordType: z.enum(["contact", "company"]).describe("Type of record")
      },
      async (args) => {
        const response = await fetch(
          `https://api.crm.com/search?q=${args.searchQuery}&type=${args.recordType}`
        );
        const data = await response.json();

        return {
          content: [{
            type: "text",
            text: JSON.stringify(data, null, 2)
          }]
        };
      }
    )
  ]
});

// Use custom tools with streaming input (required for MCP)
async function* generateMessages() {
  yield {
    type: "user" as const,
    message: {
      role: "user" as const,
      content: "Find information about Acme Corp in our CRM"
    }
  };
}

for await (const message of query({
  prompt: generateMessages(),
  options: {
    mcpServers: {
      "crm-tools": crmServer
    },
    allowedTools: ["mcp__crm-tools__search_crm"]
  }
})) {
  if (message.type === "result" && message.subtype === "success") {
    console.log(message.result);
  }
}
```

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

Important notes:

- Custom MCP tools require streaming input mode (async generator for prompt)
- Tool names follow the pattern: `mcp__{server_name}__{tool_name}`
- Use Zod schemas for type-safe input validation

When to create custom tools:

Scenario	Example
Internal APIs	Company CRM, inventory system, billing
Third-party services	Stripe, Twilio, HubSpot
Database queries	Custom SQL against your warehouse
External data	Tavily, Firecrawl, Bright Data
Specialized logic	Lead scoring algorithm, pricing calculator

Custom tools vs external MCP servers:

External MCP servers (GitHub, Notion, PostgreSQL) are great for standardized, reusable integrations. In-process custom tools via `createSdkMcpServer` are better when you need:

- Tight integration with your application logic
- Custom authentication flows
- Business-specific operations
- Tools that only make sense for your use case

Demo
Live demo: Run a simple agent programmatically.

TypeScript:

```
import { query } from "@anthropic-ai/claude-agent-sdk";

async function main() {
  const { text } = await query({
    prompt: "What files are in the current directory?",
    options: {
      maxTurns: 3,
    }
  });

  console.log(text);
}

main();
```

Python:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions

async def main():
    async for message in query(
        prompt="What files are in the current directory?",
        options=ClaudeAgentOptions(
            max_turns=3,
            allowed_tools=["Bash", "Glob"]
        )
    ):
        if hasattr(message, "result"):
            print(message.result)

asyncio.run(main())
```

---

03. Block 2: Lab 1 - Your First SDK Agent (30 min)

Task: Build a Database Analyzer Agent
Create an agent that analyzes SQLite databases and produces reports. Choose TypeScript or Python based on your preference.

Step 1: Set up the project:

TypeScript:

```
mkdir agents/file-analyzer
cd agents/file-analyzer
npm init -y
npm install @anthropic-ai/claude-agent-sdk typescript ts-node zod
npx tsc --init
```

Python:

```
mkdir agents/file-analyzer
cd agents/file-analyzer
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install claude-agent-sdk pandas
```

Step 2: Create the agent code:

TypeScript (`src/index.ts`):

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import { query } from "@anthropic-ai/claude-agent-sdk";

interface AnalysisResult {
  text: string;
  toolCalls: number;
}

async function analyzeFile(filePath: string): Promise<AnalysisResult> {
  let toolCalls = 0;

  const result = await query({
    prompt: `Analyze the SQLite database at ${filePath}. Provide:
    1. List of tables and row counts
    2. Schema for each table (columns and types)
    3. Summary statistics for numeric columns
    4. Key relationships between tables`,
    options: {
      maxTurns: 5,
      onToolCall: (tool) => {
        toolCalls++;
        console.log(`Tool called: ${tool.name}`);
      }
    }
  });

  return {
    text: result.text,
    toolCalls
  };
}

// Run
analyzeFile('../data/startup-funding.db')
  .then(result => {
    console.log('\n=== Analysis Result ===\n');
    console.log(result.text);
    console.log(`\nTotal tool calls: ${result.toolCalls}`);
  });
```

Python (`analyzer.py`):

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions

async def analyze_file(file_path: str) -> dict:
    """Analyze a SQLite database and return structured results."""
    tool_calls = 0
    result_text = ""

    prompt = f"""Analyze the SQLite database at {file_path}. Provide:
    1. List of tables and row counts
    2. Schema for each table (columns and types)
    3. Summary statistics for numeric columns
    4. Key relationships between tables"""

    async for message in query(
        prompt=prompt,
        options=ClaudeAgentOptions(
            max_turns=5,
            allowed_tools=["Read", "Bash", "Glob"]
        )
    ):
        if message.type == "tool_use":
            tool_calls += 1
            print(f"Tool called: {message.name}")
        if hasattr(message, "result"):
            result_text = message.result

    return {
        "text": result_text,
        "tool_calls": tool_calls
    }

async def main():
    result = await analyze_file('../data/startup-funding.db')
    print('\n=== Analysis Result ===\n')
    print(result["text"])
    print(f'\nTotal tool calls: {result["tool_calls"]}')

if __name__ == "__main__":
    asyncio.run(main())
```

Step 3: Run the agent:

TypeScript:

```
npx ts-node src/index.ts
```

Python:

```
python analyzer.py
```

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

Success Criteria

- [ ] Agent runs without errors
- [ ] File is analyzed correctly
- [ ] Tool calls are logged
- [ ] Output is structured

Discussion Questions

1. How many turns did the agent need?
2. Which tools did it use?
3. What would happen with a malformed CSV?

---

04. BREAK (10 min)

---

05. Block 3: Theory - Sessions and Streaming (30 min)

Sessions for Context Preservation
Sessions maintain state across multiple queries:

```
import { query, Session } from "@anthropic-ai/claude-agent-sdk";

// Create a session
const session = new Session({
  systemPrompt: "You are a data analyst helping with lead research.",
});

// First query - sets context
await query({
  prompt: "I'm working on analyzing leads for a B2B SaaS company.",
  session,
});

// Second query - has context from first
const result = await query({
  prompt: "What columns would be most important in a leads dataset?",
  session,
});

// Session remembers the conversation
```

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

When to Use Sessions

Scenario	Use Session?	Why
One-off task	No	No context needed
Multi-step workflow	Yes	Steps build on each other
Interactive conversation	Yes	Need conversation history
Batch processing (independent)	No	Each item is isolated
Batch processing (related)	Maybe	Depends on relationships

Streaming Responses

For long-running tasks, stream results:

```
import { query } from "@anthropic-ai/claude-agent-sdk";

async function streamingQuery() {
  for await (const message of query({
    prompt: "Research the top 5 CRM platforms and compare them",
    options: {
      maxTurns: 10,
    }
  })) {
    if (message.type === 'text') {
      process.stdout.write(message.content);
    } else if (message.type === 'tool_use') {
      console.log(`\n[Using tool: ${message.name}]\n`);
    } else if (message.type === 'tool_result') {
      console.log(`[Tool result received]\n`);
    }
  }
}
```

Message Types in Stream

Type	Description	Use
`text`	Text being generated	Show to user
`tool_use`	Tool being called	Log/display status
`tool_result`	Tool response	Debug/log
`error`	Error occurred	Handle gracefully
`done`	Query complete	Finalize

Permission Handling

Control what the agent can do:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
const result = await query({
  prompt: "Update the config file with new settings",
  options: {
    permissions: {
      // Auto-approve these
      allow: [
        { tool: 'Read', pattern: '*' },
        { tool: 'Bash', pattern: 'npm:*' },
      ],
      // Auto-deny these
      deny: [
        { tool: 'Bash', pattern: 'rm:*' },
        { tool: 'Write', pattern: '*.env' },
      ],
      // Ask for these (default)
      // Everything else prompts the user
    },
    // Or run fully automated
    dangerouslyAllowAllTools: true, // Use with caution!
  }
});
```

Error Handling

Agents can fail. Handle gracefully:

```
import { query, AgentError, ToolError } from "@anthropic-ai/claude-agent-sdk";

try {
  const result = await query({
    prompt: "Risky operation...",
    options: { maxTurns: 5 }
  });
} catch (error) {
  if (error instanceof ToolError) {
    console.error(`Tool ${error.toolName} failed: ${error.message}`);
    // Maybe retry with different approach
  } else if (error instanceof AgentError) {
    console.error(`Agent error: ${error.message}`);
    // Log and alert
  } else {
    throw error; // Unknown error
  }
}
```

---

06. Block 4: Lab 2 - Build a Headless GTM Agent (45 min)

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

Task: Create a Lead Enrichment Service
Build a service that enriches leads automatically, running headlessly.

Step 1: Create project structure:

```
mkdir -p agents/lead-enricher/src
cd agents/lead-enricher
npm init -y
npm install @anthropic-ai/claude-agent-sdk typescript ts-node zod
```

Step 2: Create `src/types.ts`:

```
export interface Lead {
  name: string;
  company: string;
  email: string;
  title?: string;
}

export interface EnrichedLead extends Lead {
  companySize?: string;
  industry?: string;
  recentNews?: string[];
  linkedinUrl?: string;
  enrichedAt: Date;
  score?: number;
}

export interface EnrichmentResult {
  lead: EnrichedLead;
  success: boolean;
  error?: string;
  duration: number;
}
```

Step 3: Create `src/enricher.ts`:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import { query } from "@anthropic-ai/claude-agent-sdk";
import { Lead, EnrichedLead, EnrichmentResult } from './types';

const ENRICHMENT_PROMPT = `
You are a lead research specialist. Given a lead, research their company and provide:

1. Company size (employees)
2. Industry/vertical
3. Recent news (last 3 months)
4. LinkedIn profile URL if findable

Be concise. If you can't find information, say "Not found" for that field.

Output as JSON:
{
  "companySize": "...",
  "industry": "...",
  "recentNews": ["...", "..."],
  "linkedinUrl": "..."
}
`;

export async function enrichLead(lead: Lead): Promise<EnrichmentResult> {
  const startTime = Date.now();

  try {
    const result = await query({
      prompt: `Research this lead:
Name: ${lead.name}
Company: ${lead.company}
Email: ${lead.email}
Title: ${lead.title || 'Unknown'}

${ENRICHMENT_PROMPT}`,
      options: {
        maxTurns: 8,
        tools: ['WebSearch', 'WebFetch'],
        dangerouslyAllowAllTools: true,
      }
    });

    // Parse the JSON from response
    const jsonMatch = result.text.match(/\{[\s\S]*\}/);
    if (!jsonMatch) {
      throw new Error('No JSON found in response');
    }

    const enrichment = JSON.parse(jsonMatch[0]);

    const enrichedLead: EnrichedLead = {
      ...lead,
      ...enrichment,
      enrichedAt: new Date(),
    };

    return {
      lead: enrichedLead,
      success: true,
      duration: Date.now() - startTime,
    };
```

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

Step 4: Create `src/index.ts`:

```
import { enrichBatch } from './enricher';
import { Lead } from './types';
import * as fs from 'fs';

// Sample leads (in production, load from CSV or API)
const leads: Lead[] = [
  {
    name: "Sarah Chen",
    company: "TechCorp",
    email: "sarah@techcorp.com",
    title: "VP Engineering"
  },
  {
    name: "Mike Johnson",
    company: "DataFlow Inc",
    email: "mike@dataflow.io",
    title: "CTO"
  },
  // Add more leads...
];

async function main() {
  console.log(`Starting enrichment of ${leads.length} leads...\n`);

  const results = await enrichBatch(leads, 2);

  // Summary
  const successful = results.filter(r => r.success).length;
  const failed = results.filter(r => !r.success).length;
  const totalDuration = results.reduce((sum, r) => sum + r.duration, 0);

  console.log('\n=== Enrichment Complete ===');
  console.log(`Successful: ${successful}`);
  console.log(`Failed: ${failed}`);
  console.log(`Total time: ${(totalDuration / 1000).toFixed(1)}s`);
  console.log(`Avg per lead: ${(totalDuration / results.length / 1000).toFixed(1)}s`);

  // Save results
  fs.writeFileSync(
    'output/enriched-leads.json',
    JSON.stringify(results, null, 2)
  );
  console.log('\nResults saved to output/enriched-leads.json');
}

main().catch(console.error);
```

Step 5: Run the enricher:

```
mkdir -p output
npx ts-node src/index.ts
```

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

Bonus: Add Scoring
Extend the enricher to score leads after enrichment:

```
// In enricher.ts, add:

export async function scoreLead(lead: EnrichedLead): Promise<number> {
  const result = await query({
    prompt: `Score this lead from 0-100 based on:
- Company size (larger = higher score for B2B)
- Title seniority (VP/C-level = higher)
- Industry fit (tech = higher)

Lead:
${JSON.stringify(lead, null, 2)}

Return only a number 0-100.`,
    options: {
      maxTurns: 1,
    }
  });

  return parseInt(result.text.trim()) || 50;
}
```

Deliverable

- Working lead enrichment service
- Batch processing with concurrency control
- JSON output file with enriched leads
- Summary statistics

---

07. Block 5: Sandboxed Execution with Daytona (Bonus)

Why Sandboxing Matters
Running agents in production means running code that Claude generates. This creates risks:

- Agents might execute destructive commands
- File system access could affect other processes
- Runaway loops could consume resources
- Untrusted data could lead to code injection

Sandboxing solves this by isolating agent execution in controlled environments.

What is Daytona?
Daytona provides secure sandboxes where AI agents can safely execute code. Key features:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

Feature	Benefit
Isolated filesystems	Agents can't access host files
Resource limits	CPU, memory, and time constraints
Code execution	Run Python, TypeScript, bash safely
File upload/download	Move data in and out of sandbox
Preview URLs	Expose ports for web apps running in sandbox
API access	Programmatic control from your app

Setting Up Daytona

Step 1: daytona.io ■

Step 2: Install the SDK:

Python:

```
pip install daytona-sdk
```

TypeScript:

```
npm install @daytonaio/sdk
```

Core Daytona Concepts

Creating a Sandbox:

Python:

```
from daytona_sdk import Daytona

daytona = Daytona()  # Uses DAYTONA_API_KEY env var
sandbox = daytona.create()  # Creates a Python sandbox by default
```

TypeScript:

```
import { Daytona } from "@daytonaio/sdk";

const daytona = new Daytona();  // Uses DAYTONA_API_KEY env var
const sandbox = await daytona.create({ language: "python" });
```

Executing Code Directly

The `code_run()` method executes code directly in the sandbox:

Python:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
# Execute Python code directly
response = sandbox.process.code_run("""
import pandas as pd
df = pd.read_csv('/workspace/leads.csv')
print(f"Rows: {len(df)}")
print(df.describe())
""")
print(response.result)
```

TypeScript:

```
// Execute Python code directly
const response = await sandbox.process.codeRun(`
import pandas as pd
df = pd.read_csv('/workspace/leads.csv')
print(f"Rows: {len(df)}")
print(df.describe())
`);
console.log(response.result);
```

File Operations

Upload and download files:

Python:

```
# Upload a file to the sandbox
sandbox.fs.upload_file("/local/path/leads.csv", "/workspace/leads.csv")

# Download results from sandbox
sandbox.fs.download_file("/workspace/output/report.json", "/local/output/report.json")
```

TypeScript:

```
// Upload a file to the sandbox
await sandbox.fs.uploadFile("/local/path/leads.csv", "/workspace/leads.csv");

// Download results from sandbox
await sandbox.fs.downloadFile("/workspace/output/report.json", "/local/output/report.json");
```

Preview URLs for Web Applications

If your agent builds a dashboard or web app in the sandbox:

Python:

```
# Get a public URL for a port in the sandbox
preview_url = sandbox.get_preview_link(3000)
print(f"View dashboard at: {preview_url}")
```

TypeScript:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
// Get a public URL for a port in the sandbox
const previewUrl = sandbox.getPreviewLink(3000);
console.log(`View dashboard at: ${previewUrl}`);
```

Session-Based Execution

For long-running processes or stateful operations:

Python:

```
# Create a persistent session
session = sandbox.process.create_session("data-analysis")

# Execute commands in the session
sandbox.process.execute_session_command(
    session_id="data-analysis",
    command="cd /workspace && python setup.py"
)

# Run more commands with preserved state
sandbox.process.execute_session_command(
    session_id="data-analysis",
    command="python analyze.py --verbose"
)
```

Integrating Claude with Daytona

The power of Daytona comes from having Claude generate code, then executing it safely:

Python example:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import anthropic
from daytona_sdk import Daytona

# Initialize both SDKs
client = anthropic.Anthropic()
daytona = Daytona()
sandbox = daytona.create()

# Upload data to sandbox
sandbox.fs.upload_file("startup-funding.db", "/workspace/funding.db")

# Have Claude generate analysis code
message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[{
        "role": "user",
        "content": """Write Python code to:
        1. Connect to SQLite database at /workspace/funding.db
        2. Query funding rounds by stage and calculate totals
        3. Print a summary table

        Use sqlite3 and pandas. Only output the Python code, no explanations."""
    }]
)

# Extract the generated code
code = message.content[0].text

# Execute Claude's code in the sandbox
response = sandbox.process.code_run(code)
print("Analysis Results:")
print(response.result)

# Cleanup
sandbox.delete()
```

TypeScript example:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import Anthropic from "@anthropic-ai/sdk";
import { Daytona } from "@daytonaio/sdk";

async function analyzeWithClaude() {
  const client = new Anthropic();
  const daytona = new Daytona();
  const sandbox = await daytona.create({ language: "python" });

  // Upload data
  await sandbox.fs.uploadFile("startup-funding.db", "/workspace/funding.db");

  // Have Claude generate analysis code
  const message = await client.messages.create({
    model: "claude-sonnet-4-20250514",
    max_tokens: 1024,
    messages: [{
      role: "user",
      content: `Write Python code to:
        1. Connect to SQLite database at /workspace/funding.db
        2. Query funding rounds by stage and calculate totals
        3. Print a summary table

        Use sqlite3 and pandas. Only output the Python code, no explanations.`
    }]
  });

  // Execute Claude's code in the sandbox
  const code = message.content[0].type === "text" ? message.content[0].text : "";
  const response = await sandbox.process.codeRun(code);
  console.log("Analysis Results:");
  console.log(response.result);

  // Cleanup
  await sandbox.delete();
}
```

Running Agent SDK with Daytona

Combine the Agent SDK with Daytona for full agentic workflows in a sandbox:

Python:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import asyncio
from daytona_sdk import Daytona
from claude_agent_sdk import query, ClaudeAgentOptions

async def run_sandboxed_agent(task: str, data_file: str):
    """Run an agent in a Daytona sandbox with file access."""

    daytona = Daytona()
    sandbox = daytona.create()

    try:
        # Upload data file to sandbox
        sandbox.fs.upload_file(data_file, f"/workspace/{data_file}")

        # Run the agent with sandbox execution
        async for message in query(
            prompt=f"""
            You are running in a sandboxed environment.
            Working directory: /workspace
            Data file: /workspace/{data_file}

            Task: {task}

            Execute any analysis code directly. The environment has Python
            with pandas, numpy, and matplotlib installed.
            """,
            options=ClaudeAgentOptions(
                allowed_tools=["Read", "Write", "Bash"],
                permission_mode="bypassPermissions"  # Safe because sandboxed
            )
        ):
            if hasattr(message, "result"):
                print(message.result)

        # Download results
        sandbox.fs.download_file("/workspace/output/results.json", "./results.json")

    finally:
        sandbox.delete()

asyncio.run(run_sandboxed_agent(
    task="Analyze the funding data and generate summary statistics by stage",
    data_file="startup-funding.db"
))
```

TypeScript:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
import { Daytona } from "@daytonaio/sdk";
import { query } from "@anthropic-ai/claude-agent-sdk";

async function runSandboxedAgent(task: string, dataFile: string) {
  const daytona = new Daytona();
  const sandbox = await daytona.create({ language: "python" });

  try {
    // Upload data file
    await sandbox.fs.uploadFile(dataFile, `/workspace/${dataFile}`);

    // Run the agent
    for await (const message of query({
      prompt: `
        You are running in a sandboxed environment.
        Working directory: /workspace
        Data file: /workspace/${dataFile}

        Task: ${task}

        Execute any analysis code directly.
      `,
      options: {
        allowedTools: ["Read", "Write", "Bash"],
        permissionMode: "bypassPermissions"
      }
    })) {
      if ("result" in message) {
        console.log(message.result);
      }
    }

    // Download results
    await sandbox.fs.downloadFile("/workspace/output/results.json", "./results.json");

  } finally {
    await sandbox.delete();
  }
}
```

Sandboxing Patterns

Pattern 1: Batch Processing

```
For each data file:
1. Create sandbox
2. Upload file
3. Run agent analysis
4. Download results
5. Delete sandbox
```

Pattern 2: Long-Running Agent

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
1. Create persistent sandbox
2. Upload all data files
3. Run multiple agent tasks
4. Stream results back
5. Delete sandbox when done
```

Pattern 3: User-Triggered Analysis

```
1. User uploads file to your app
2. Create sandbox for this user
3. Run agent on their data
4. Return results
5. Cleanup after timeout
```

When to Use Sandboxing

Scenario	Sandbox?	Why
Development/testing	No	Direct execution is faster
Internal automation	Maybe	Depends on trust level
User-provided data	Yes	Can't trust user input
Production pipelines	Yes	Defense in depth
Code generation tasks	Yes	Always sandbox generated code

Exercise: Run Your Analyzer in a Sandbox

1. Sign up for a Daytona free tier account
2. Modify your file analyzer to run in a sandbox
3. Upload startup-funding.db to the sandbox
4. Run the analysis agent
5. Download the results

---

08. Wrap-Up (15 min)

Key Takeaways

1. `query()` is the core - One function to run agents programmatically (TypeScript or Python)
2. Sessions preserve context - Use for multi-step workflows
3. Stream for UX - Show progress on long tasks
4. Permissions matter - Control what agents can do in production
5. Handle errors - Agents fail; build resilient systems
6. Sandbox production agents - Use Daytona for safe code execution in containerized environments

Homework

Part 1: Build an Automated Report Generator

1. Create an agent that:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

- Connects to the startup-funding.db
- Analyzes funding trends
- Generates a markdown report
- Saves to output folder

2. Add features:
   - Streaming progress output
   - Error handling with retries
   - Session-based follow-up questions
3. Run analysis queries like:
   - Funding trends by quarter
   - Top investors by deal count
   - Industry breakdown
4. Document:
   - Code structure
   - How you handled errors
   - Performance observations

Part 2: Daily Metrics Monitoring Agent (Bonus)

Build an agent that runs automated metrics checks against the funding database:

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.

```
// daily-metrics-agent.ts
import { query } from "@anthropic-ai/claude-agent-sdk";

interface MetricCheck {
  name: string;
  query: string;
  baseline: string;
  threshold: number; // Alert if deviation exceeds this %
}

const METRICS: MetricCheck[] = [
  {
    name: 'Weekly Funding Volume',
    query: `SELECT SUM(amount_usd) as total FROM funding_rounds
            WHERE funding_date >= date('now', '-7 days')`,
    baseline: `SELECT AVG(weekly_total) FROM (
      SELECT SUM(amount_usd) as weekly_total FROM funding_rounds
      WHERE funding_date >= date('now', '-90 days')
      GROUP BY strftime('%Y-%W', funding_date))`,
    threshold: 0.2
  },
  {
    name: 'Series A Deal Count',
    query: `SELECT COUNT(*) as count FROM funding_rounds
            WHERE stage = 'Series A'
            AND funding_date >= date('now', '-30 days')`,
    baseline: `SELECT AVG(monthly_count) FROM (
      SELECT COUNT(*) as monthly_count FROM funding_rounds
      WHERE stage = 'Series A'
      GROUP BY strftime('%Y-%m', funding_date))`,
    threshold: 0.3
  }
];

async function runDailyMetricsCheck() {
  const result = await query({
    prompt: `You are a metrics monitoring agent. Run these checks and report anomalies:

${METRICS.map(m => `
**${m.name}**
Current: ${m.query}
Baseline: ${m.baseline}
Alert threshold: ${m.threshold * 100}% deviation
`).join('\n')}

For each metric:
1. Run both queries against startup-funding.db
2. Calculate % deviation from baseline
3. If deviation > threshold, flag as anomaly
4. For anomalies, suggest what might have caused the change

Return a structured report.`,
    options: {
      maxTurns: 10,
    }
  });
```

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.


```
  console.log('Daily Metrics Report');
  console.log('='.repeat(40));
  console.log(result.text);
```

What this teaches:

- Running scheduled agent tasks
- Combining SQL queries with LLM analysis
- Anomaly detection patterns
- Building monitoring systems with agents

Stretch goal: Extend the agent to post alerts to Slack or email when anomalies are detected.

Next Week Preview
Week 7: Evals - Building dashboards to track agent performance and iterate until alignment

---

09. Facilitator Notes

Common Issues
1. API key not set: Ensure `ANTHROPIC_API_KEY` environment variable
2. Rate limiting: Add delays between batch items
3. Tool permission errors: Check permission configuration
4. JSON parsing fails: Agent output isn't always clean JSON

Environment Setup
Participants need:

- Node.js 18+ (for TypeScript) OR Python 3.9+ (for Python)
- npm/yarn or pip
- Anthropic API key
- TypeScript or Python knowledge (basic)
- Optional: Daytona API key for sandboxing lab

Timing Adjustments
- Lab 1 is simpler, can extend discussion if concepts unclear
- Lab 2 is substantial, can simplify to single-lead enrichment
- Bonus scoring can be homework if time short

Code Quality Tips
- Emphasize type safety with TypeScript
- Show proper async/await patterns
- Discuss error handling strategies
- Mention rate limiting for production

Copyright © 2026 AGI Ventures Canada Inc. All rights reserved. The information in this document is proprietary and provided for informational purposes only and shall not create a warranty of any kind.