Analytics Vidhya

Flow and States management in CrewAI

Instructor

Alessandro Romano

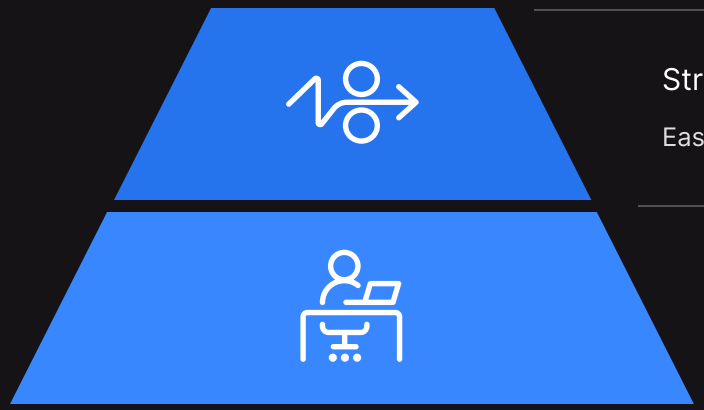
Senior Data Scientist - Kuehne Nagel

Flows

Streamlined Workflow Creation

Easily link Crews and tasks to build intricate AI workflows.

Flows



Streamlined Workflow Creation

Easily link Crews and tasks to build intricate AI workflows.

State Management

Smoothly handle and share state across tasks within workflows.

Flows

Streamlined Workflow Creation

Easily link Crews and tasks to build intricate AI workflows.

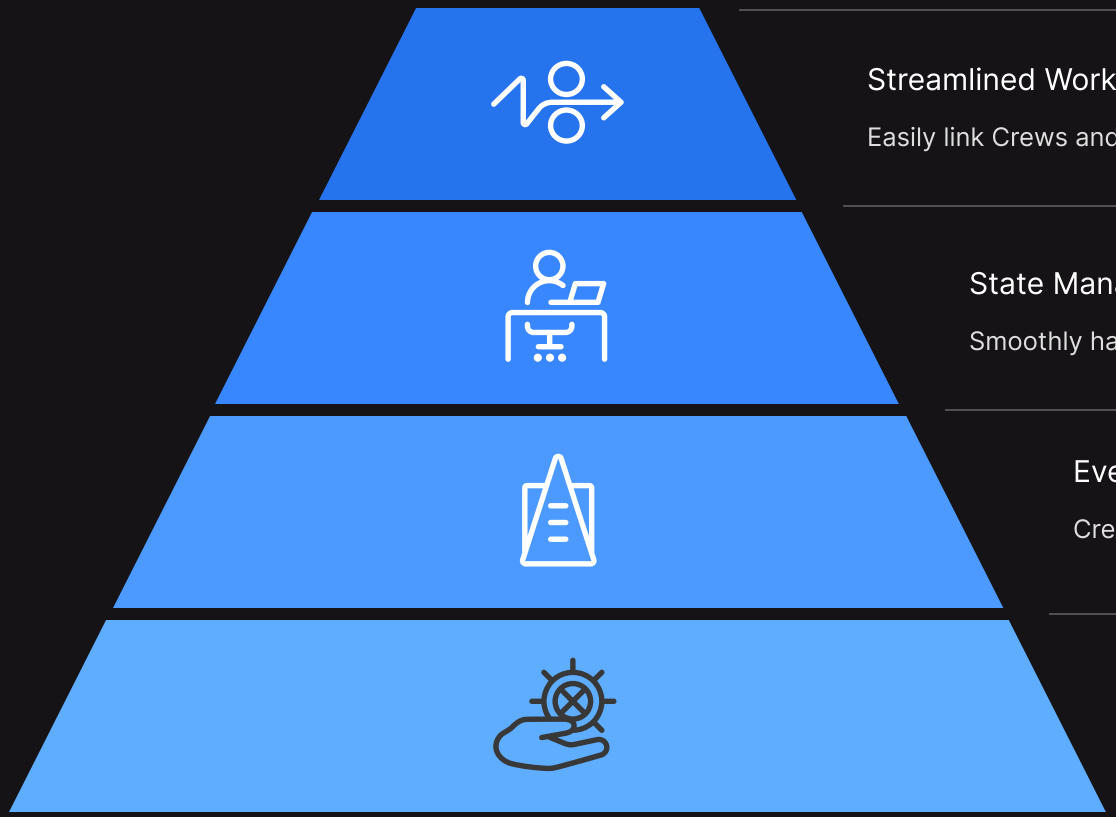
State Management

Smoothly handle and share state across tasks within workflows.

Event-Driven Architecture

Create dynamic, responsive workflows with event-based actions.

Flows



Streamlined Workflow Creation

Easily link Crews and tasks to build intricate AI workflows.

State Management

Smoothly handle and share state across tasks within workflows.

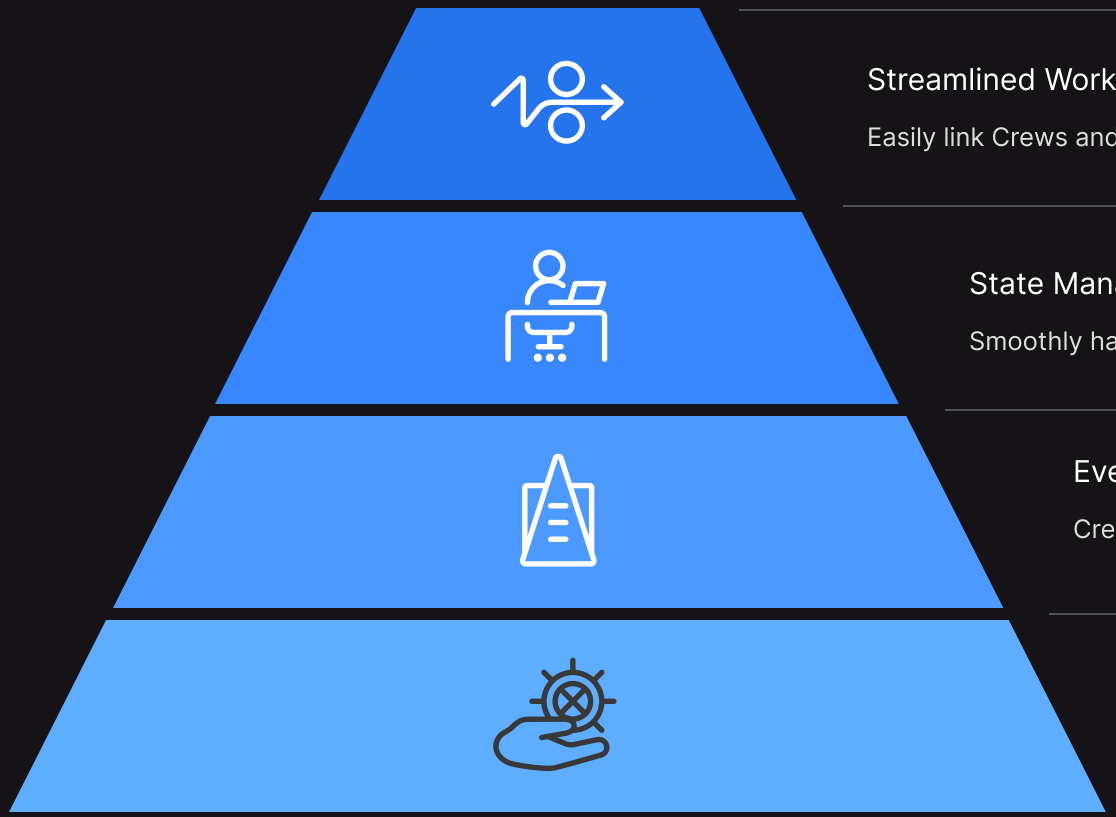
Event-Driven Architecture

Create dynamic, responsive workflows with event-based actions.

Flexible Control Flow

Supports conditional logic, loops, and branching for customized processes.

Flows



Streamlined Workflow Creation

Easily link Crews and tasks to build intricate AI workflows.

State Management

Smoothly handle and share state across tasks within workflows.

Event-Driven Architecture

Create dynamic, responsive workflows with event-based actions.

Flexible Control Flow

Supports conditional logic, loops, and branching for customized processes.

Implement Flows

@start()

- Marks a method as the starting point of a Flow.
- Flexibility
- Parallel Execution

```python
from crewai.flow.flow import Flow, listen, start

class ActorFlow(Flow):

    @start()
    def generate_actor(self):
        # Return a random actor name
        random_actor = "Generate a random actor"
        return random_actor

    @listen(generate_actor)
    def generate_famous_movie(self, actor):
        # Find a famous movie with the given actor
        famous_movie = f"Find a famous movie starring {actor}"
        return famous_movie

# Initialize the flow and start the process
flow = ActorFlow()
print(flow.kickoff())
```

Implement Flows

@listen()

- Marks a method to listen for another task's output in the Flow
- Runs when a specified task emits an output
- Process output as an argument

```python
from crewai.flow.flow import Flow, listen, start

class ActorFlow(Flow):

    @start()
    def generate_actor(self):
        # Return a random actor name
        random_actor = "Generate a random actor"
        return random_actor

    @listen(generate_actor)
    def generate_famous_movie(self, actor):
        # Find a famous movie with the given actor
        famous_movie = f"Find a famous movie starring {actor}"
        return famous_movie

# Initialize the flow and start the process
flow = ActorFlow()
print(flow.kickoff())
```

Flow

or_ Function

Triggers listener when any specified method emits output.

```python
from crewai.flow.flow import Flow, listen, or_, start

class OrExampleFlow(Flow):

    @start()
    def start_method(self):
        return "Hello from the start method"

    @listen(start_method)
    def second_method(self):
        return "Hello from the second method"

    @listen(or_(start_method, second_method))
    def logger(self, result):
        print(f"Logger: {result}")

flow = OrExampleFlow()
flow.kickoff()
```

Flow

or_ Function

Triggers listener when any specified method emits output.

and_ Function

Triggers listener only when all specified methods emit output.

```python
from crewai.flow.flow import Flow, listen, or_, start

class OrExampleFlow(Flow):

    @start()
    def start_method(self):
        return "Hello from the start method"

    @listen(start_method)
    def second_method(self):
        return "Hello from the second method"

    @listen(or_(start_method, second_method))
    def logger(self, result):
        print(f"Logger: {result}")

flow = OrExampleFlow()
flow.kickoff()
```

Flow

or_ Function

Triggers listener when any specified method emits output.

and_ Function

Triggers listener only when all specified methods emit output.

@router() Decoder

Enables conditional routing based on method output.

```python
from crewai.flow.flow import Flow, listen, or_, start

class OrExampleFlow(Flow):

    @start()
    def start_method(self):
        return "Hello from the start method"

    @listen(start_method)
    def second_method(self):
        return "Hello from the second method"

    @listen(or_(start_method, second_method))
    def logger(self, result):
        print(f"Logger: {result}")

flow = OrExampleFlow()
flow.kickoff()
```

Flow State Management

- **Structured State Management**

  Ensures data consistency and type safety

```python
from crewai.flow.flow import Flow, listen, start

class UnstructuredExampleFlow(Flow):

    @start()
    def first_method(self):
        self.state.message = "Hello from structured flow"
        self.state.counter = 0

    @listen(first_method)
    def second_method(self):
        self.state.counter += 1
        self.state.message += " - updated"

    @listen(second_method)
    def third_method(self):
        self.state.counter += 1
        self.state.message += " - updated again"

        print(f"State after third_method: {self.state}")

flow = UnstructuredExampleFlow()
flow.kickoff()
```

Flow State Management

- **Structured State Management**

  Ensures data consistency and type safety

- **Schema Models**

  Leverages Pydantic's BaseModel

```python
from crewai.flow.flow import Flow, listen, start

class UnstructuredExampleFlow(Flow):

    @start()
    def first_method(self):
        self.state.message = "Hello from structured flow"
        self.state.counter = 0

    @listen(first_method)
    def second_method(self):
        self.state.counter += 1
        self.state.message += " - updated"

    @listen(second_method)
    def third_method(self):
        self.state.counter += 1
        self.state.message += " - updated again"

        print(f"State after third_method: {self.state}")

flow = UnstructuredExampleFlow()
flow.kickoff()
```

Flow State Management

- **Structured State Management**

  Ensures data consistency and type safety

- **Schema Models**

  Leverages Pydantic's BaseModel

- **Developer Support**

  Enables validation and auto-completion

```python
from crewai.flow.flow import Flow, listen, start

class UnstructuredExampleFlow(Flow):

    @start()
    def first_method(self):
        self.state.message = "Hello from structured flow"
        self.state.counter = 0

    @listen(first_method)
    def second_method(self):
        self.state.counter += 1
        self.state.message += " - updated"

    @listen(second_method)
    def third_method(self):
        self.state.counter += 1
        self.state.message += " - updated again"

        print(f"State after third_method: {self.state}")

flow = UnstructuredExampleFlow()
flow.kickoff()
```

Thank You