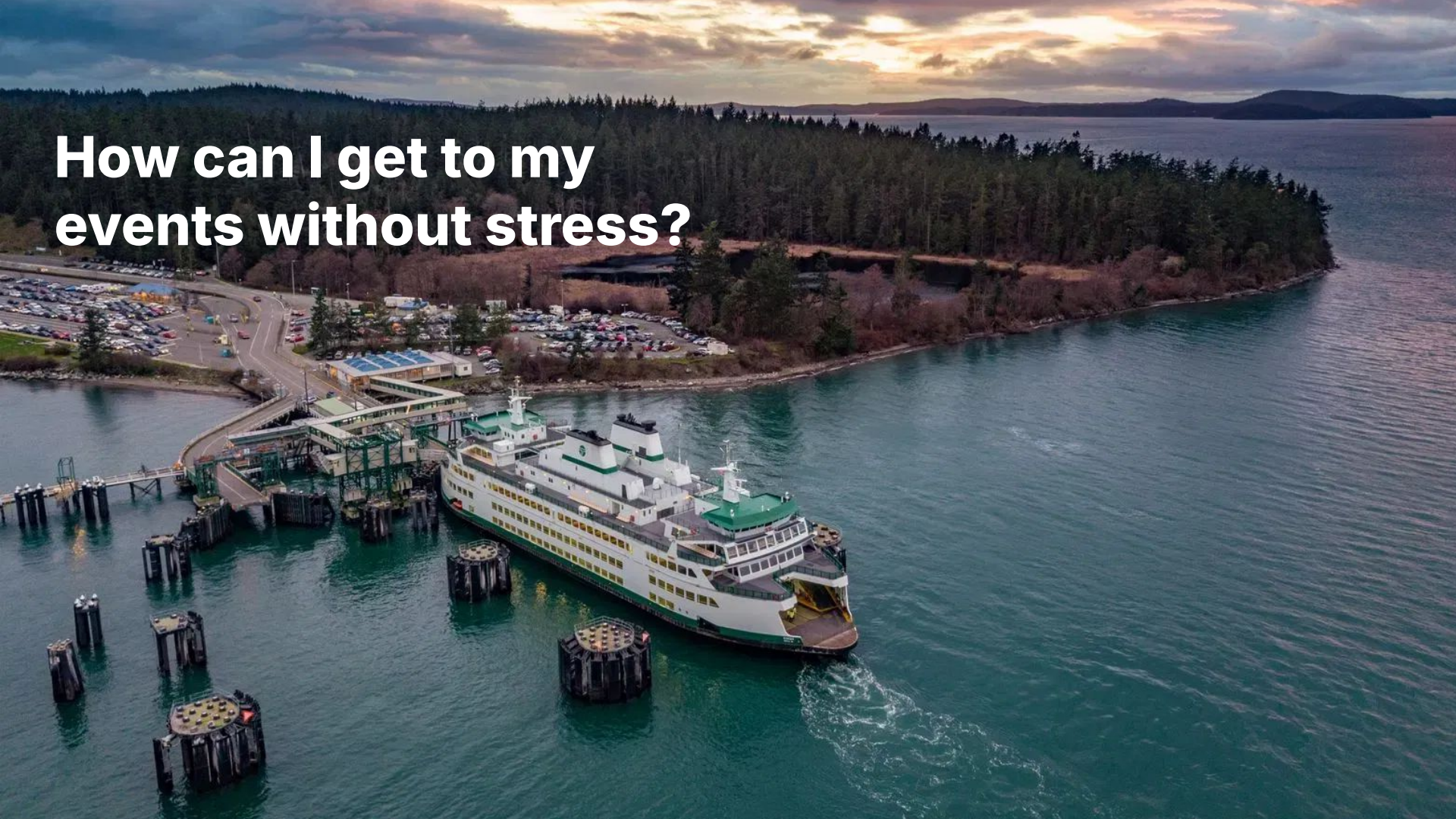
Building Search-Aligned Autonomous Systems with MCP, Elastic, and Tron?

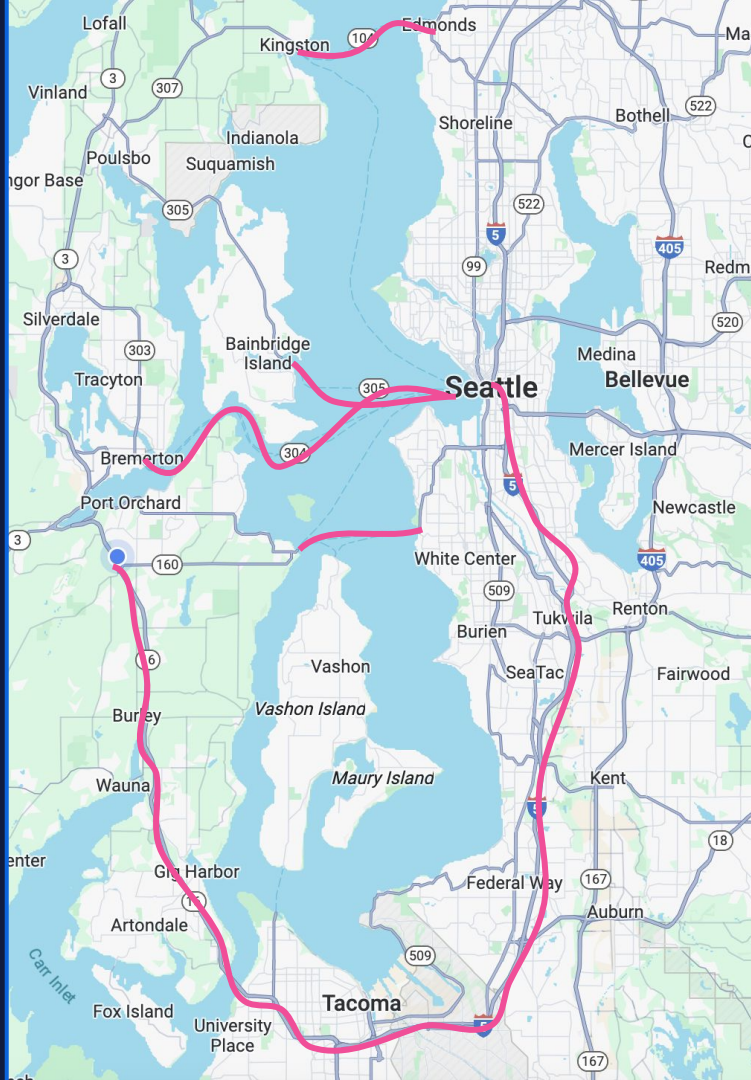
By Justin Castilla

justin.castilla@elasti.co

elastic | The Search AI Company


How can I get to my
events without stress?

Every day I have to find the best route to get to my meetups, concerts, dinners, etc.




Google Maps helps, but doesn't consider schedules

How I built this solution

Python + FastMCP + Elasticsearch + LLM = Easier Commute?

Step 1

Gather all of the information needed to make my commuting decisions

My Calendar

- Events
  - time
  - location
  - description
  - presenting?

WSDOT API

- Terminal locations
- Sailing Schedule
- Service Delays

Google Maps API

- Drive time estimation
- Traffic monitoring

How I built this solution

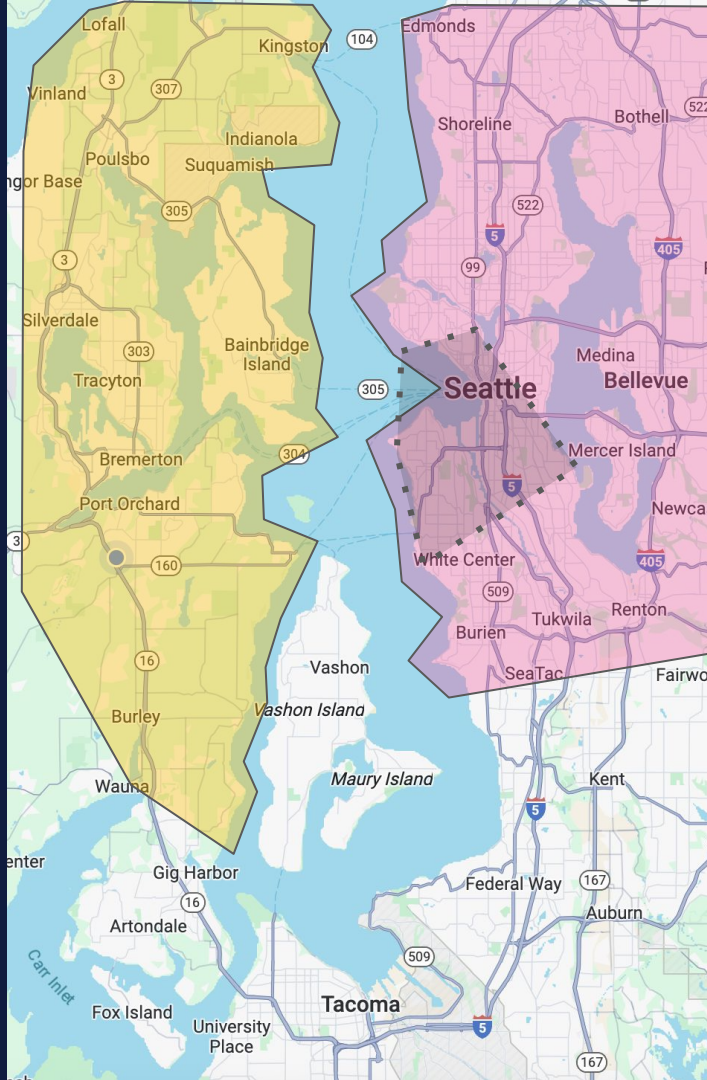
Python + FastMCP + Elasticsearch + LLM = Easier Commute?

Step 1

Gather all of the information needed to make my commuting decisions

Step 2

Establish a timetable of ferry routes and group them by regions and county



How I built this solution

Python + FastMCP + Elasticsearch + LLM = Easier Commute?

Step 1

Gather all of the information needed to make my commuting decisions

Step 2

Establish a timetable of ferry routes and group them by regions and county

Step 3

Write every single function I would need to determine drive times, ferry sailings, event creation and lookup

```
fetch_ferry_schedules()
fetch_terminals()
find_nearby_terminals()
drive_time()
get_ferry_times()
get_ferry_times_by_dir()
geocode_county()
search_events()
create_event()
```

How I built this solution

Python + FastMCP + Elasticsearch + LLM = Easier Commute?

Step 1

Gather all of the information needed to make my commuting decisions

Step 2

Establish a timetable of ferry routes and group them by regions and county

Step 3

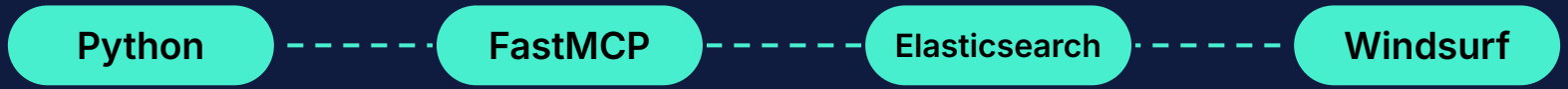
Write every single function I would need to determine drive times, ferry sailings, event creation and lookup

Step 4

Give an LLM access to these functions and provide **VERY** clear instructions

How I built this solution

Python + FastMCP + Elasticsearch + LLM = Easier Commute?



Easier language to use for fast prototyping of MCP servers, interacting with Elastic, and general hackery.

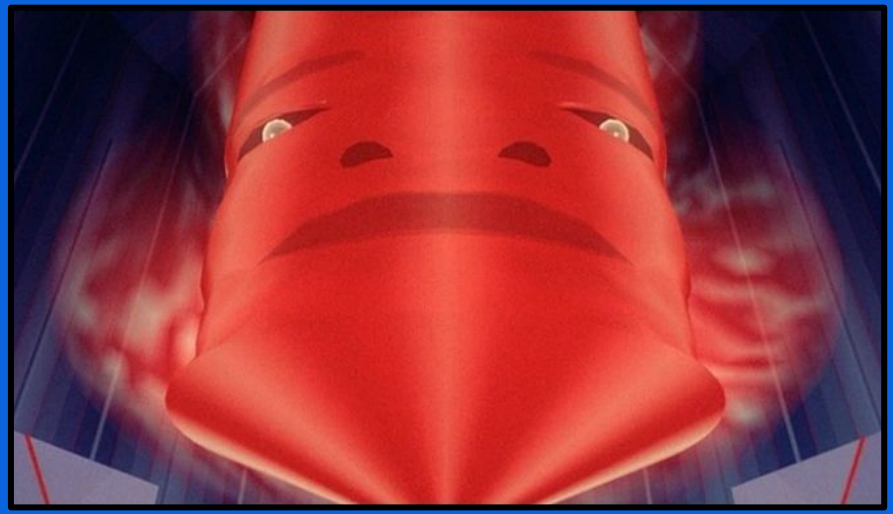
Creates a server of resources, tools, and prompts for the LLM to use

Hold my calendar events in a document store, with semantic and hybrid search at the ready

Yell at an LLM until all of my project goals are met, and then serve as an LLM guinea pig

MCP, like the MCP from Tron?

- A framework made by Anthropic with agreed upon communication protocols and schemas
- Creates ***resources***, ***tools***, ***prompts***, and other services available to LLMs to access and execute
- Allows LLMs to reach out and gather more information, take actions on behalf of users, and generally do more than just sit on their dated knowledge
- Kind of the hot thing right now.



MCP (Master Control Program), from Disney's Tron, 1982

The three major components of MCP

Resources

Resources represent data or files that an MCP client can read.

This allows LLMs to access files, database content, configuration, or dynamically generated information relevant to the conversation.

```
@mcp.resource("transit://ferries/terminals")
```

Resource templates extend this concept by allowing clients to request dynamically generated resources based on parameters passed in the URI.

```
@mcp.resource("transit://ferries/terminals/{terminal_id}/schedule/{date}")
```

The three major components of MCP

Resources

```
@mcp.resource(
    uri="transit://ferries/terminals",
    name="Ferry Terminals",
    description="List of ferry terminals and their associated code numbers")
def fetch_terminals():
    url = 'https://wsdot.wa.gov/Ferries/API/Terminals/rest/terminalbasics'
    params = {'apiaccesscode': wdot_api_key}
    resp = requests.get(url, params=params)
    resp.raise_for_status()
    data = resp.json()
    return data
```

The three major components of MCP

Tools

Tools in MCPs transform regular Python functions into capabilities that LLMs can invoke during conversations.

When an LLM decides to use a tool:

1. It sends a request with parameters based on the tool's schema.
2. The MCP validates these parameters against your function's signature. +Pydantic for the win!
3. The function executes with the validated inputs.
4. The result is returned to the LLM, which can use it in its response.

```
@mcp.tool("look_up_event")
```

The three major components of MCP

Tools

```
@mcp.tool(
    name="create_event",
    description="Create a new event in Elasticsearch."
)
def create_event(eventDoc) -> dict:
    resp = es.index(index="events", document=event_doc)
    return {"event_id": resp["id"], "event": resp["_source"]}
```

The three major components of MCP

Prompts

Prompts are reusable message templates that help LLMs generate structured, purposeful responses.

When a client requests a prompt:

1. MCP finds the corresponding prompt definition.
2. If it has parameters, they are validated against your function signature.
3. Your function executes with the validated inputs.
4. The generated message is returned to the LLM to guide its response.
5. This allows you to define consistent, reusable templates that LLMs can use across different clients and contexts.

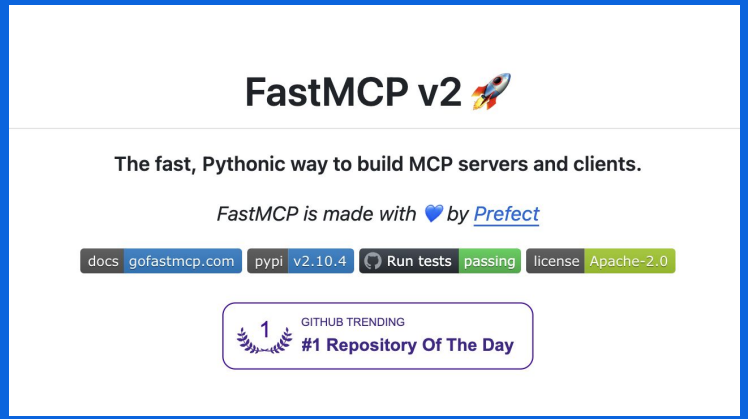
```
@mcp.prompt("get_there_with_a_buffer")
```

The three major components of MCP

Prompts

```
@mcp.prompt(
   name='get_there_with_a_buffer',
   description="Instructions for finding a route with buffer before event time")
def get_there_by(origin, destination, event_time, buffer_minutes=10):
   return f"""
 1. The starting location is {origin}
 2. Find the two nearest ferries from {origin} and find drive times to them.
 3. Determine the arrival time at the destination terminal
 4. Determine the drive time from the destination terminal to {destination}
 5. Find times that get the user to {destination} by {buffer_minutes} before
    {event_time}
   """
```

FastMCP, to make things fast and easy!



github.com/jlowin/fastmcp

https://gofastmcp.com/

- MCP is powerful but implementing it involves a lot of boilerplate
  a. server setup
  b. protocol handlers
  c. content types
  d. error management.
- FastMCP is high-level and Pythonic
- In most cases, decorating a function is all you need.
- Part of the MCP framework
- But also it's own project
- VERY active

Check out the code:

https://github.com/justincastilla/kitsap_commute_MCP

Elastic's MCP Server

https://github.com/elastic/mcp-server-elasticsearch

Thank you!