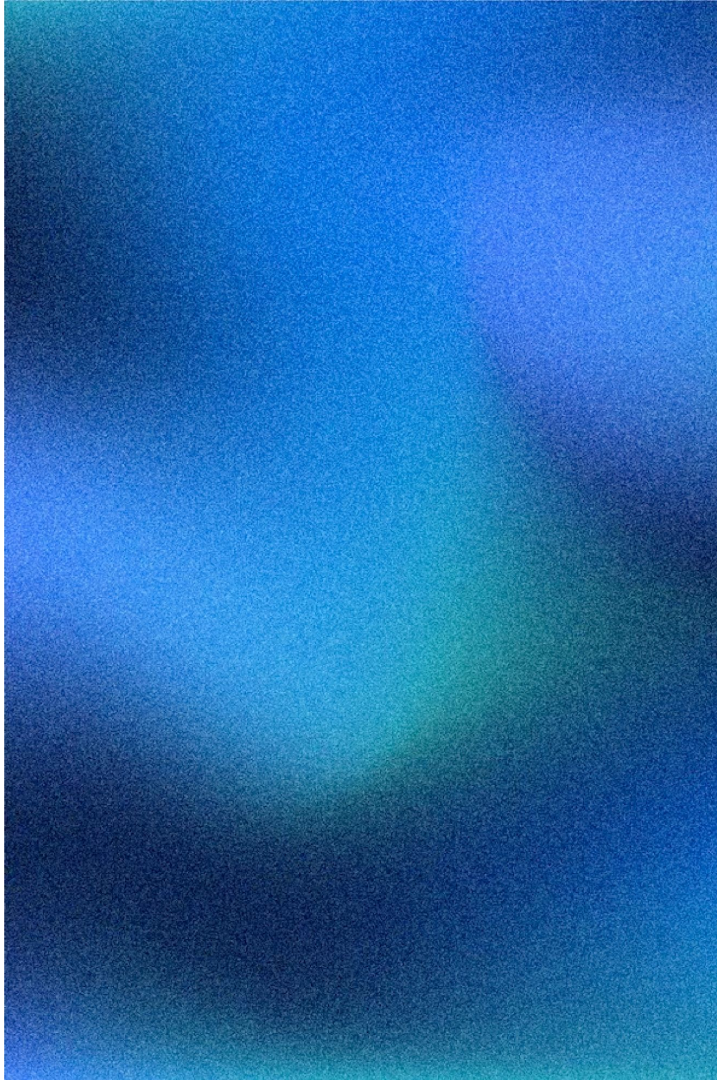
[nagels]

Graph Meetup Mumbai

Roll your own Graph-driven MCP Server



About Us: Ghlen Nagels

CEO of Nagels IT Consulting

Digital Nomad, loves to run, enjoys craft beers

Email: ghlen@nagels.tech

Website: https://nagels.tech

About Us: Nabeel Parkar

Open Source Engineer

South African somehow ended up in India

Email: nabeel@ex3.dev

Website: https://ex3.dev

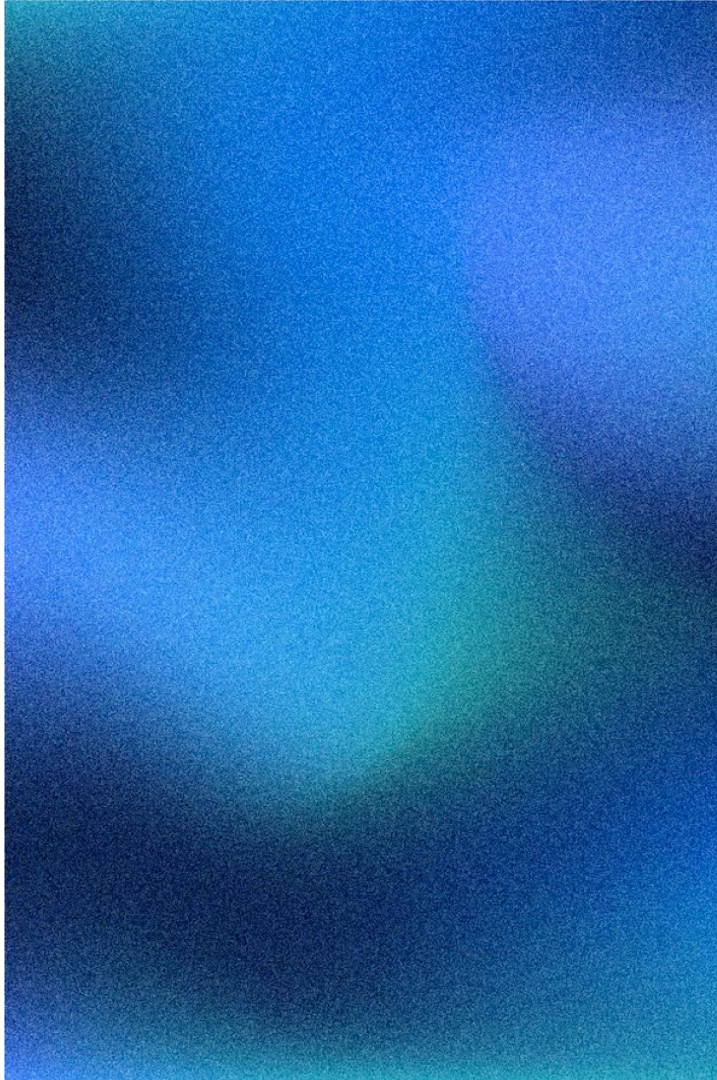
About Nagels IT Consulting

- Human First Developer Teams
- Focus on International SMEs
- Specializes in Cloud Applications and Graph Technologies
- Consulting, Training & Development
- [**We’re Hiring**]

Let’s Build Your Tech Together

The Game Plan

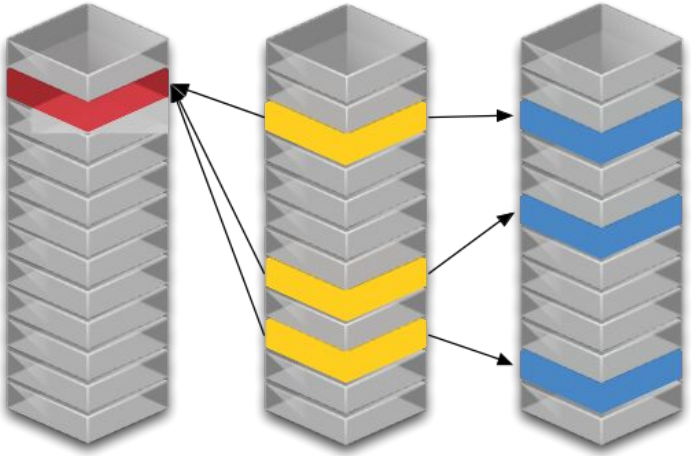
- **Neo4J:** birds eye view
- **MCP:** The high level
- **Developing:** Driver concepts
- **Showoff time:** Combining it all!



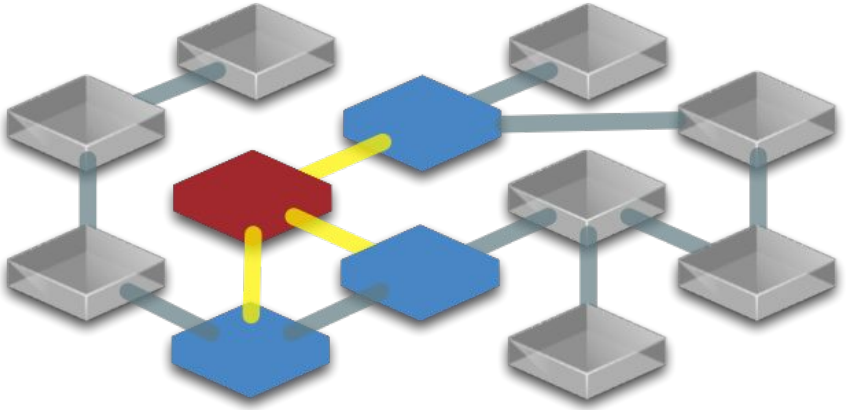
Neo4J: Birds Eye View

[nagels]

Relational Database



Graph Database



Cypher Query Language example

Cypher

```
(A:Human)-[:LOVES]->(B:Human)
```

Cypher Basic Query Structure

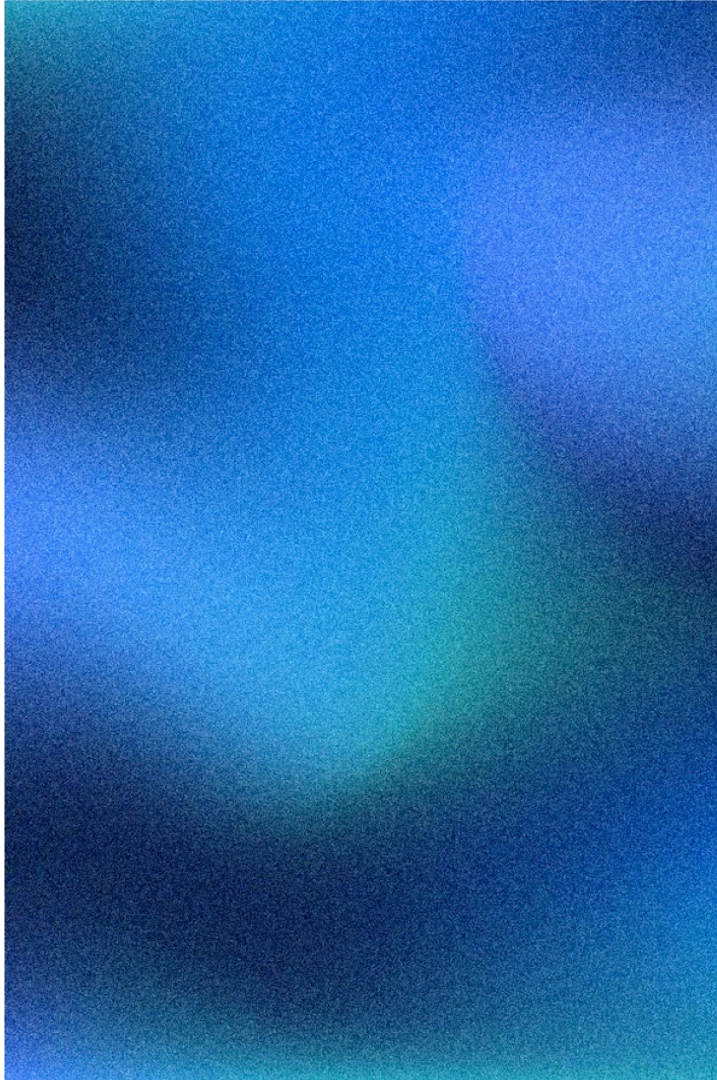
Basic query structure

```
(OPTIONAL) MATCH graph pattern

WHERE predicates

RETURN expression
```

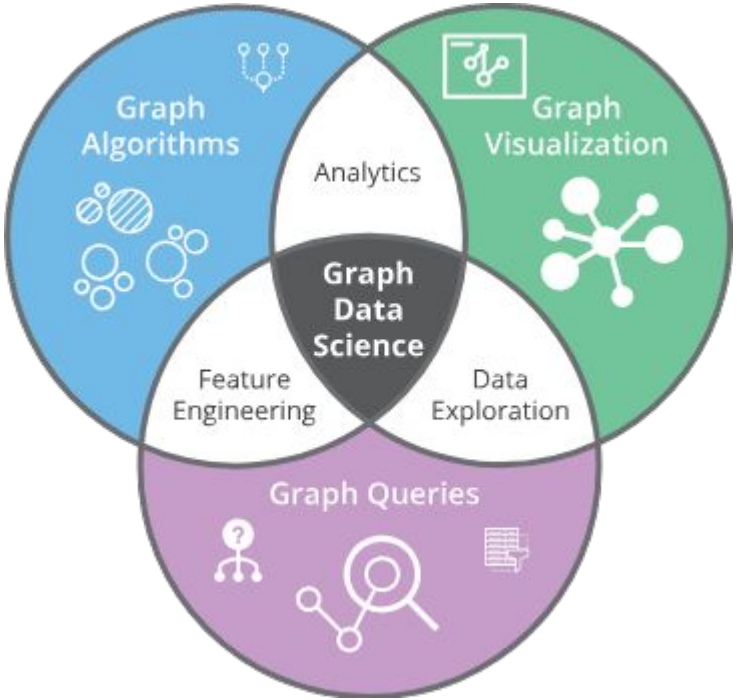
Let's Build Your Tech Together

What to know about Graphs

- Highly **efficient** with connected data
- Extremely friendly to **data science**
- **Different paradigm** from relational DB
- **Cypher** is the common query language
- **Easy to understand**, hard to master
- One of the **fastest growing** technologies



Learn Graphs at the Graph Academy

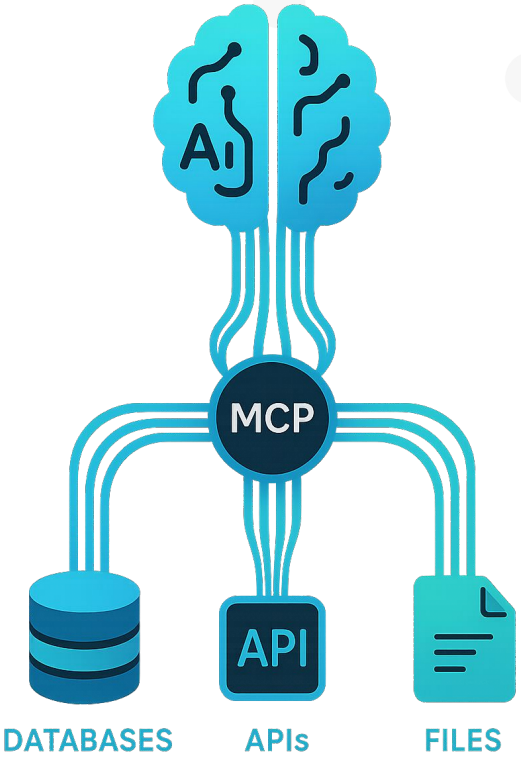


https://graphacademy.neo4j.com/

MCP: The High Level

Model Context Protocol (MCP)

Connecting AI to Real-World Data

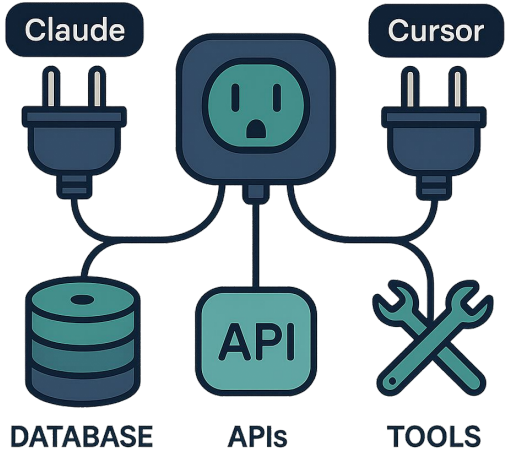


14

[nagels]

What is MCP:

The Model Context Protocol (MCP) is an open standard that enables AI assistants to securely connect to external data sources and tools in real-time.



Think of it as...

A **universal adapter** that lets AI assistants "plug into" various data sources and tools without requiring custom integrations for each AI platform.

Core Benefits of MCP

Standardization

Write once, use with many AI platforms (Claude, Cursor, VS Code, etc.)

Security

User-controlled access with permission prompts

Real-time Data

Access to live, current information (no stale training data)

Extensibility

Easy to add new tools and data sources

Industry Adoption

Major AI Players

- Google
- Microsoft
- AWS
- OpenAI

Agent Frameworks

- LangChain
- Pydantic-AI
- CrewAI
- LlamaIndex

Developer Tools

- Claude Desktop
- Cursor AI IDE
- VsCode

Growing Ecosystem

Thousands of community-created MCP servers

The Problem MCP Solves

Data Silos

AI models can’t access your live business data

Custom Integrations

Each AI platform requires separate tool development

Security Concerns

No standardized way to control AI access to sensitive data

Stale Information

AI responses based on outdated training data

Real-World Use Cases

Customer Support

- Query Neo4j knowledge graphs for troubleshooting
- Real-time customer history access
- Dynamic problem resolution

Development & DevOps

- Database schema queries and diagnostics
- Automated infrastructure management
- Architecture-based code generation

Business Intelligence

- Natural language queries against live graph analytics
- Path analysis between business entities
- Customer churn prediction

Enterprise Knowledge Management

- Live documentation and policy access
- Real-time compliance checking
- Dynamic content generation

MCP Components: Tools & Resources

Tools - Functions that AI can execute

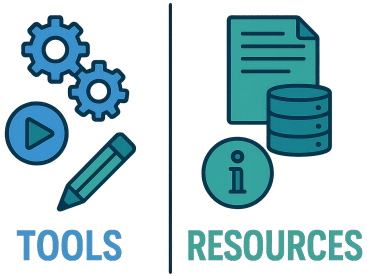
- Query Neo4j knowledge graphs for troubleshooting
- Real-time customer history access
- Dynamic problem resolution

Resources - Static or semi-static data for context

- Natural language queries against live graph analytics
- Path analysis between business entities
- Customer churn prediction

Simple Distinction

Tools: "What can I do?" (Actions)
Resources: "What do I need to know?" (Context)



Tools Example: Neo4j Cypher Query

tool.json

```
{
  "name": "run_cypher_query",
  "description": "Execute a Cypher query against Neo4j",
  "inputSchema": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "The Cypher query to execute"
      },
      "parameters": {
        "type": "object",
        "description": "Query parameters"
      }
    },
    "required": ["query"]
  }
}
```

Resources Example: Database Schema

resource.json

```
{
  "uri": "neo4j://schema",
  "name": "Database Schema",
  "description": "Neo4j database schema including node labels, relationship types, and property keys",
  "mimeType": "application/json"
}
```

How MCP Works: JSON-RPC 2.0

Foundation Protocol

- UTF-8 encoded messages
- Request/response pattern with unique IDs
- Support for notifications (no response expected)
- Batch operations for multiple requests

list_tools_request.json

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {}
}
```

list_tools_response.json

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [...]
  }
}
```

Transport Mechanisms

STDIO (Standard Input/Output)

- Local development & desktop apps
- Client launches server as subprocess
- Messages via stdin/stdout

HTTP+SSE (Legacy - 2024)

- Deprecated but widely supported
- SSE for server-to-client messages
- HTTP POST for client-to-server

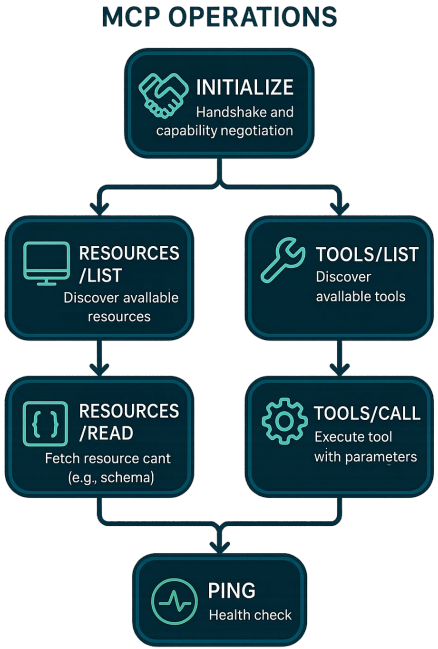
Streamable HTTP (New - 2025)

- Single GET and POST HTTP endpoint
- Optional SSE streaming
- Session management with headers

Message Types in MCP

Core Operations

1. **Initialize** - Handshake and capability negotiation
2. **Resources/List** - Discover available resources
3. **Resources/Read** - Fetch resource content (e.g., schema)
4. **Tools/List** - Discover available tools
5. **Tools/Call** - Execute tool with parameters
6. **Ping** - Health check



MCP Authorization & Security

OAuth 2.1 Integration

- **Standard Protocol:** Based on OAuth 2.1 with PKCE for security
- **Transport Level:** Authorization at HTTP transport layer
- **Optional Implementation:** Not required for all MCP servers

Security Features

- **HTTPS Required:** All authorization endpoints must use HTTPS
  Token Validation: Access tokens validated on every request
- **PKCE Required:** Proof Key for Code Exchange for all clients
- **Secure Storage:** OAuth 2.1 best practices for token handling

Grant Types Supported

- **Authorization Code:** User-based authentication (human end users)
- **Client Credentials:** Application-to-application authentication
- **Dynamic Client Registration:** Automatic client registration (RFC7591)

Discovery & Fallbacks

- **Metadata Discovery:** OAuth 2.0 Authorization Server Metadata (RFC8414)
- **Default Endpoints:** `/authorize`, `/token`, `/register` fallbacks
- **Third-Party Auth:** Support for delegated authorization servers

MCP Roadmap & Future

Validation & Registry

- **Reference Client Implementations:** High-quality AI applications demonstrating protocol features
- **Compliance Test Suites:** Automated verification for clients, servers, and SDKs
- **MCP Registry:** Centralized discovery and API layer for third-party marketplaces
- **Developer Ecosystem:** Tools for confident MCP implementation

Multimodality & Streaming

- **Additional Media Types:** Video and other modalities beyond text
- **Streaming Support:** Multipart, chunked messages for interactive experiences
- **Bidirectional Communication:** Real-time, interactive data flows

Agent Enhancements

- **Agent Graphs:** Complex agent topologies through namespacing
- **Interactive Workflows:** Human-in-the-loop experiences with granular permissions
- **Direct Communication:** Standardized patterns for end-user interaction

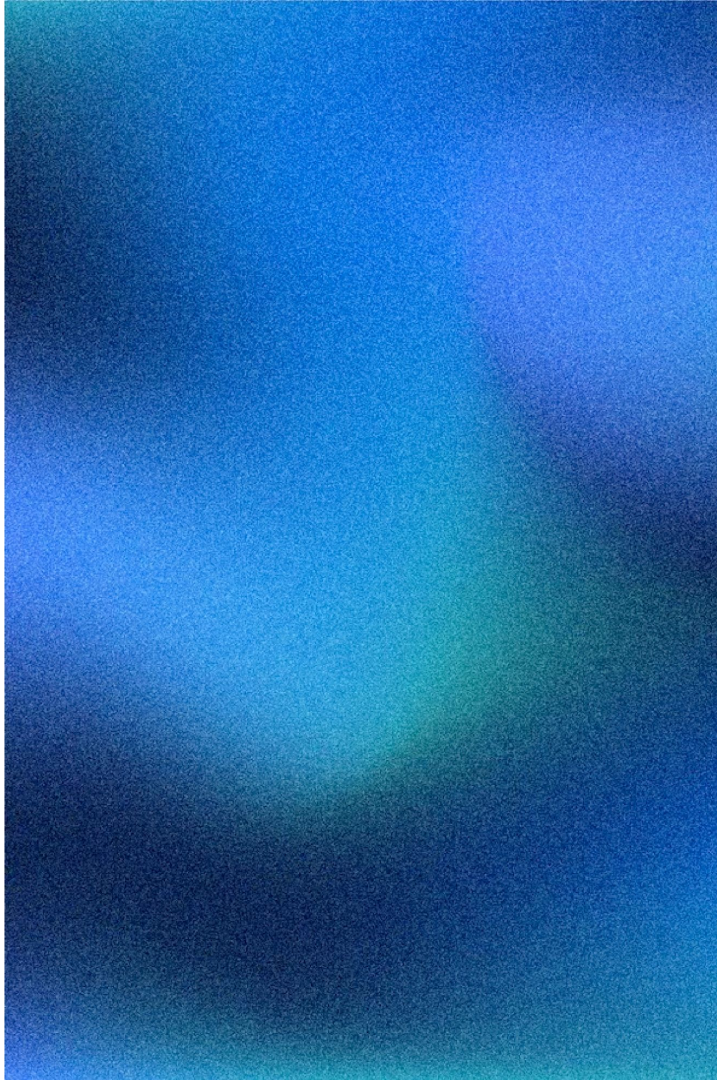
Governance & Community

- **Community-Led Development:** Collaborative ecosystem for diverse use cases
- **Transparent Standardization:** Clear contribution processes and formal industry standardization

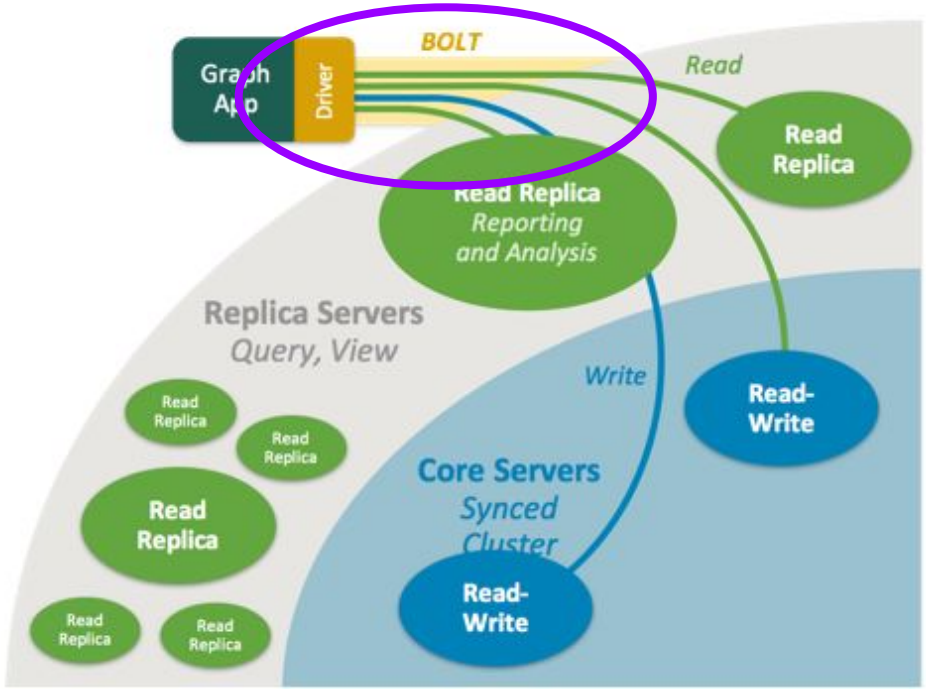
Developing: Driver Concepts:

Required Knowledge

- Basic Networking
- OOP
- Streams
- Basic High-Availability Concepts
- Common DB Paradigms



Understanding this layer = [Good Performance]



[nagels]

Canonical * Driver Model

[nagels]

Uniform & Idiomatic

[nagels]

All Neo4j Drivers

Thanks to the Neo4j contributor community, there are additionally drivers for almost every popular programming language, most of which mimic existing database driver idioms and approaches. Get started with your stack now, see the dedicated page for more detail.

Java	Spring	Neo4j-OGM	.NET	JavaScript
Python	Go	Ruby	PHP	Erlang/Elixir
Perl	C/C++	Clojure	Haskell	R

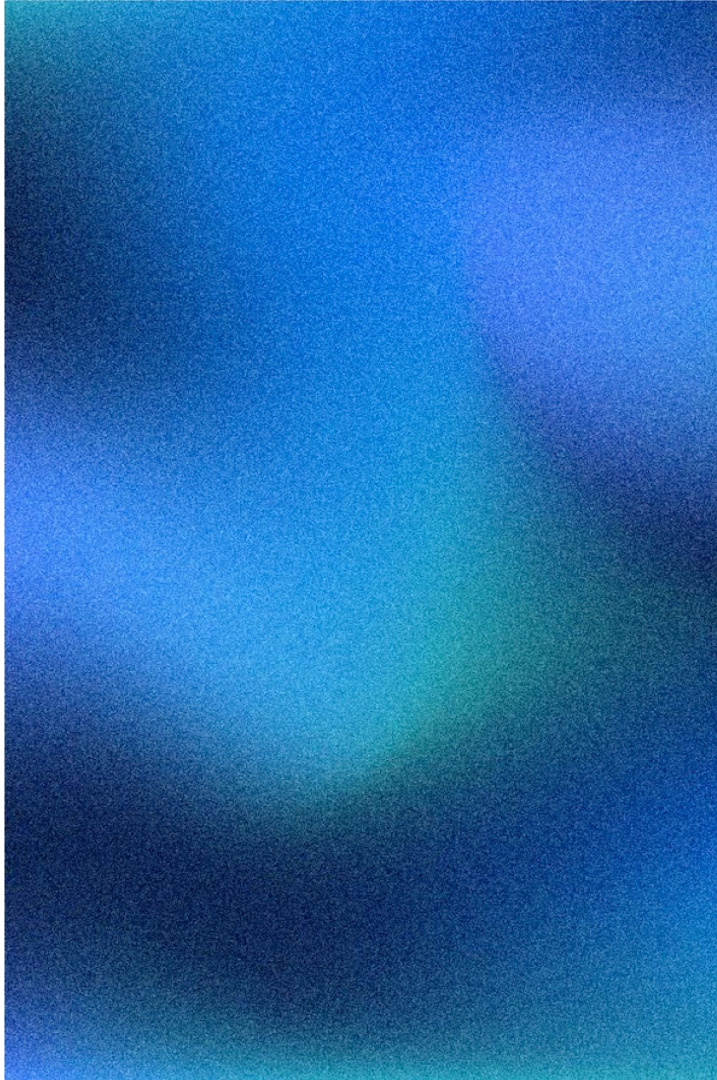
If you are new to development, we recommend the Jetbrains IDE ↗ for a good developer experience, which also comes with a Neo4j Database plugin ↗ .

https://neo4j.com/developer/language-guides/#neo4j-drivers

[nagels]

Driver Concepts

- Driver
- Session
- Transaction
- Statement
- Result



Driver

The **driver** is the thread-safe backbone providing access to Neo4j. It owns a **connection pool** through which all database connectivity occurs and can spawn **sessions** for carrying out work.

[nagels]

Driver: Typical example

create_driver.php

```php
use Laudis\Neo4j\Basic\Driver;
use Laudis\Neo4j\Databags\DriverConfiguration;

$driver = Driver::create(
  uri: 'neo4j://user:mypassword@localhost:7687',
  configuration: DriverConfiguration::create()->withUserAgent('MyApp/1.0.0')
);
```

Driver: Connectivity

```
<scheme>://<user>:<password>@<host>:<port>
```

- Scheme determines the **behaviour** of the driver
- User and password is **optional**
- The host is the IP or Hostname to connect to **initially**
- The port is self explanatory

Driver: Schema Connectivity

Certificate Type	Neo4j Causal Cluster	Neo4j Standalone Server	Direct Connection to Cluster Member
Unencrypted	`neo4j`	`neo4j`	`bolt`
Encrypted with Full Certificate	`neo4j+s`	`neo4j+s`	`bolt+s`
Encrypted with Self-Signed Certificate	`neo4j+ssc`	`neo4j+ssc`	`bolt+ssc`
Neo4j AuraDB ↗	`neo4j+s`	N/A	N/A

Driver: Neo4j Scheme: Client side routing

1. Driver fetches the **routing table** from the original connection URI

1. The routing table is a **list of addresses** and their role

1. When creating a session the driver **lends a connection** based on the **session and server role**

Driver: Configuration

Name	Type	Description
uri	String	Uri for initial connection
auth	Dictionary	Authentication information
user_agent	String	The user agent information
encrypted	Boolean	Enable encryption

Driver: Authentication

Name	Data	Description
Basic	username + password	Default and basic, also works using URI
OIDC	token	Use an OpenID connect token
Kerberos	token	Use a Kerberos token
None	-	Acces without a password, also works using URI

Driver: Key Takeaways

- Excellent candidate **Singleton Pattern**
- **UX is the same** regardless of underlying protocol
- Neo4j schema is **default**
- Bolt schema trades **flexibility** for **performance**
- Connection URI is only for the **initial connection**
- A connection is against a Neo4j Instance, **not a database**
- The **connection pool** can be configured at this level

Session

Sessions are lightweight containers for causally chained sequences of transactions. They borrow **connections** from the connection pool as required and chain transactions using **bookmarks**.

43

[nagels]

Session: Typical Code Example

create_session.php

```php
use Laudis\Neo4j\Databags\SessionConfiguration;
use Laudis\Neo4j\Enum\AccessMode;

$session = $driver->createSession(SessionConfiguration::create()
  ->withDatabase('my-database')
  ->withAccessMode(AccessMode::READ())
);
```

Session: Configuration

Name	Type	Description
default_access_mode	String	READ or WRITE
database	Dictionary	The database the session works on
bookmarks	List<String>	Bookmarking
imp_user	String	Which user to impersonate
fetch_size	Integer	How many rows are fetched at once

Session: Bookmarks: The Problem It Solves

bookmarks_problem.php

```php
$session->run('CREATE (x:Test {name: "test"})');
// This may throw an exception
// as it may not find a node with name "test"
$session->run('MATCH (x:Test {name: "test"}) RETURN x')->first();
```

Session: Bookmarks

- Neo4j provides a bookmark **identifier**
  marking the **future state** when the work is done

- The session **stores** this bookmark and
  passes it onto the next request

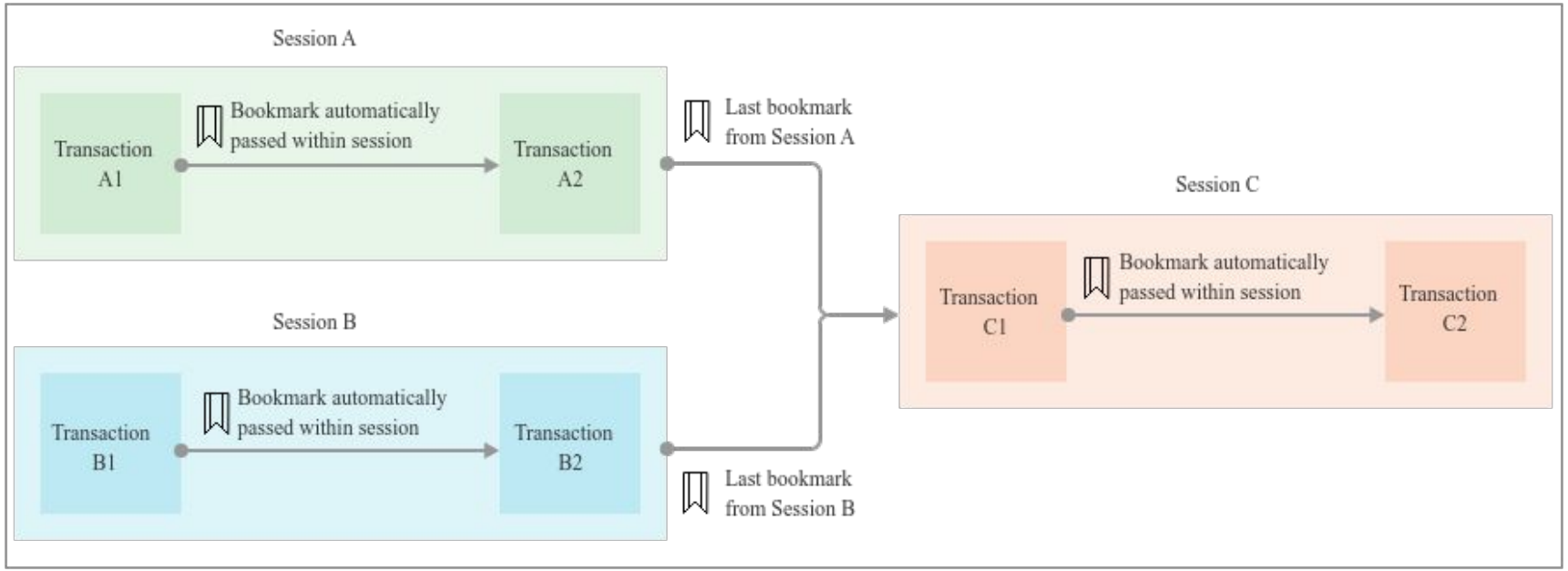
- Neo4j will only handle the new request
  once the **state of the database matches** the identifier

Session: Bookmarks: The Problem It Creates

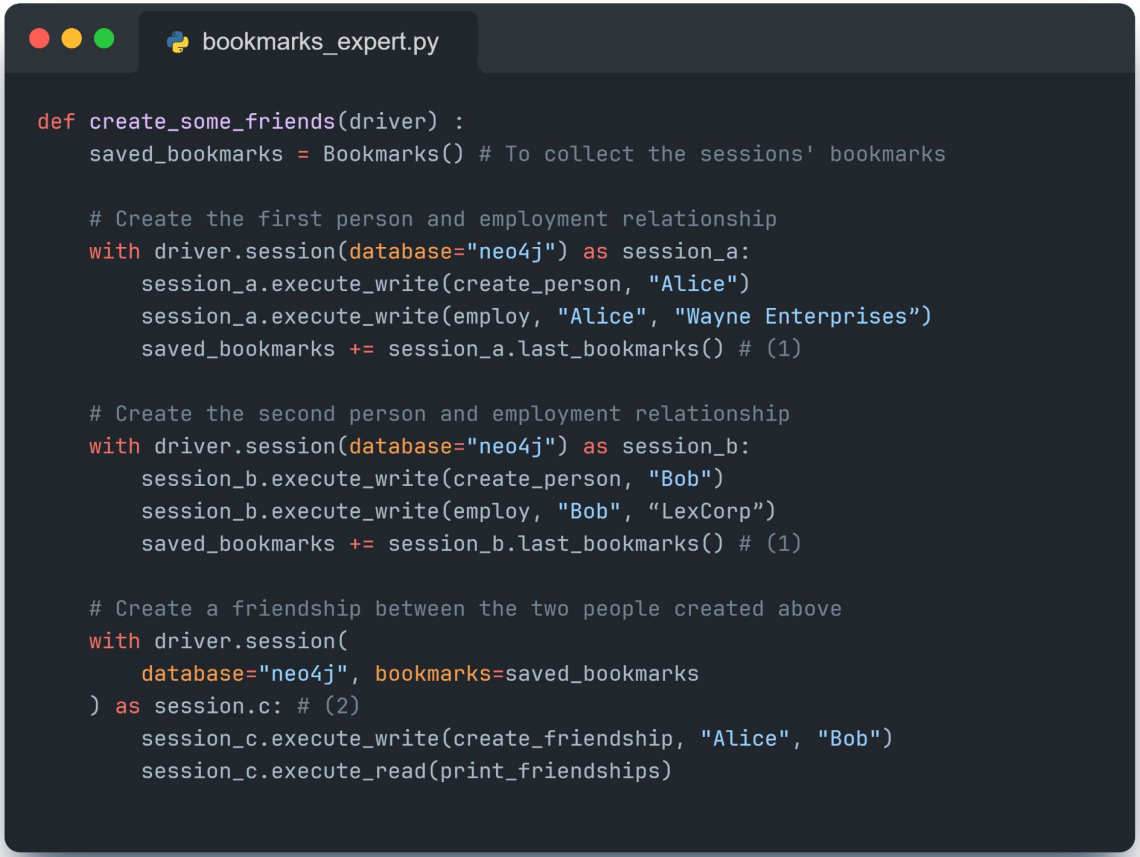
bookmarks_problem.php

```php
$session->run(<<<'CYPHER'
UNWIND range(1, 100) AS x
CREATE (n:Node {id: x})
CYPHER)

// Processing this request may take an inordinate
// amount of time
$session->run('MATCH (x:OtherNode {slug: ''}')
```

Session: Bookmarks: Expert Edition



Session: Bookmarks: Expert Edition



Session: Key Takeaways

- When in doubt, **create a new session**
- Many **open sessions** means many **open connections**
- Only reuse a session if the **work is related**
- **Don’t worry** about bookmarks until you are optimising
- The session determines **the database** you work on
- Finetune result fetching and memory usage with **fetch_size** config

Transaction

Transactions are atomic units of work that may contain one or more query. Each transaction is bound to a single **connection** and is represented in the causal chain by a **bookmark**.

52

[nagels]

Transaction: Configuration

Name	Type	Description
metadata	Dictionary	Metadata to pass to Neo4j
timeout	Integer	The maximum duration of the transaction

Transaction: Types

- **Auto-committed** queries
- **Transaction** functions
- **Unmanaged** transactions

Transaction: Auto-committed

- Simple
- Intuitive
- May fail in complex scenarios

auto_commited_transaction.php

```php
$session->run(
  'MERGE (user {email: $email})',
  ['email' => 'abc@hotmail.com'],
);

$statement = new Statement(
  'MERGE (user {email: $email})',
  ['email' => 'abc@hotmail.com']
);
$session->runStatement($statement, 'default');

$results = $session->runStatements([
  Statement::create('MATCH (x) RETURN x LIMIT 100'),
  Statement::create(
    'MERGE (x:Person {email: $email})',
    ['email' => 'abc@hotmail.com']
  )
]);
```

Transaction: Transient Errors

- **Database** related (memory, leader election change, ...)
- **Temporary**
- Requires a **retry**

Transaction: Transaction Functions

- Manages Transient Errors for you
- Simple
- A bit more boilerplate
- Works in every scenario
- Need to be idempotent

transactions.php

```php
use Laudis\Neo4j\Contracts\TransactionInterface;

// Do a simple merge and return the result
$result = $client->writeTransaction(static function (TransactionInterface $tsx) {
  $result = $tsx->run('MERGE (x {y: "Z"}:X) return x');
  return $result->first()->get('x')['y'];
});

// Will result in an error
$client->readTransaction(static function (TransactionInterface $tsx) {
  $tsx->run('MERGE (x {y: "z"}:X) return x');
});

// This is a poorly designed transaction function
$client ->writeTransaction(static function (TransactionInterface $tsx) use ($externalCounter) {
  $externalCounter->incrementNodesCreated();
  $tsx->run('MERGE (x {y: $id}:X) return x', ['id' => Uuid::v4()]);
});

// This achieves the same effect but is safe in case it should be retried.
// The function is now idempotent.
$id = Uuid::v4();
$client->writeTransaction(static function (TransactionInterface $tsx) use ($id) {
  $tsx->run('MERGE (x {y: $id}:X) return x', ['id' => $id]);
});
$externalCounter->incrementNodesCreated();
```

Transaction: Unmanaged

- Simple concept
- Complete control
- Punishes forgetfulness

unmanaged_transactions.php

```php
use Laudis\Neo4j\Databags\Statement;

$tsx = $session->beginTransaction();

$result = $tsx->run('MATCH (x) RETURN x LIMIT 100');
$result = $tsx->runStatement(Statement::create('MATCH (x) RETURN x LIMIT 100'));
$results = $tsx->runStatements([Statement::create('MATCH (x) RETURN x LIMIT 100')]);

// $tsx->rollback();
$tsx->commit();
```

Transaction: Key Takeaways

- **Functions** are the preferred way
- **Complicated** scenarios require **unmanaged** transactions
- **One connection** can have many **transactions** open
- Transactions **cannot be nested**
- Keep it **simple** by using **auto-committed** transactions

Statement

A statement is a single instruction consisting of a cypher string and a parameter dictionary.

61

[nagels]

Statement: Code Example

statement.php

```php
use Laudis\Neo4j\Databags\Statement ;

$session->run(new Statement(
  <<<'CYPHER'
  MATCH (p:Person {name: $name}) - [a:Authored] -> (b:Book)
  WHERE a.createdAt > $startDate
  RETURN b.title as title
  CYPHER,
  [
    'name' => 'Mark Twain',
    'startDate' => new DateTime('1876-01-01')
  ]
))
```

Statement Sidenote: Domain Specific Objects

Starred PHP rows can be used as a parameter

Cypher	Php
null	* `null`
string	* `string`
integer	* `int`
float	* `float`
boolean	* `bool`
Map	* `\Laudis\Neo4j\Types\CypherMap`
List	* `\Laudis\Neo4j\Types\CypherList`
Point	* `\Laudis\Neo4j\Contracts\PointInterface` **
Date	* `\Laudis\Neo4j\Types\Date`
Time	* `\Laudis\Neo4j\Types\Time`
LocalTime	* `\Laudis\Neo4j\Types\LocalTime`
DateTime	* `\Laudis\Neo4j\Types\DateTime`
DateTimeZoneId	* `\Laudis\Neo4j\Types\DateTimeZoneId`
LocalDateTime	* `\Laudis\Neo4j\Types\LocalDateTime`
Duration	* `\Laudis\Neo4j\Types\Duration`
Node	`\Laudis\Neo4j\Types\Node`
Relationship	`\Laudis\Neo4j\Types\Relationship`
Path	`\Laudis\Neo4j\Types\Path`

Statement: Key Takeaways

- Use **parameters everywhere**
- Parameters help with **query caching**
- Parameters stop **Cypher injections**
- Certain **objects** can be parameters
- Parameters are a **dictionary** whose **keys** directly map to their Cypher **parameter name**.

Result

Results are the values returned by a query in the form of **rows**. Each result is bound to a single **connection** and can be **consumed** by iterating over it.

65

[nagels]

Results: Cursors By Default

results.php

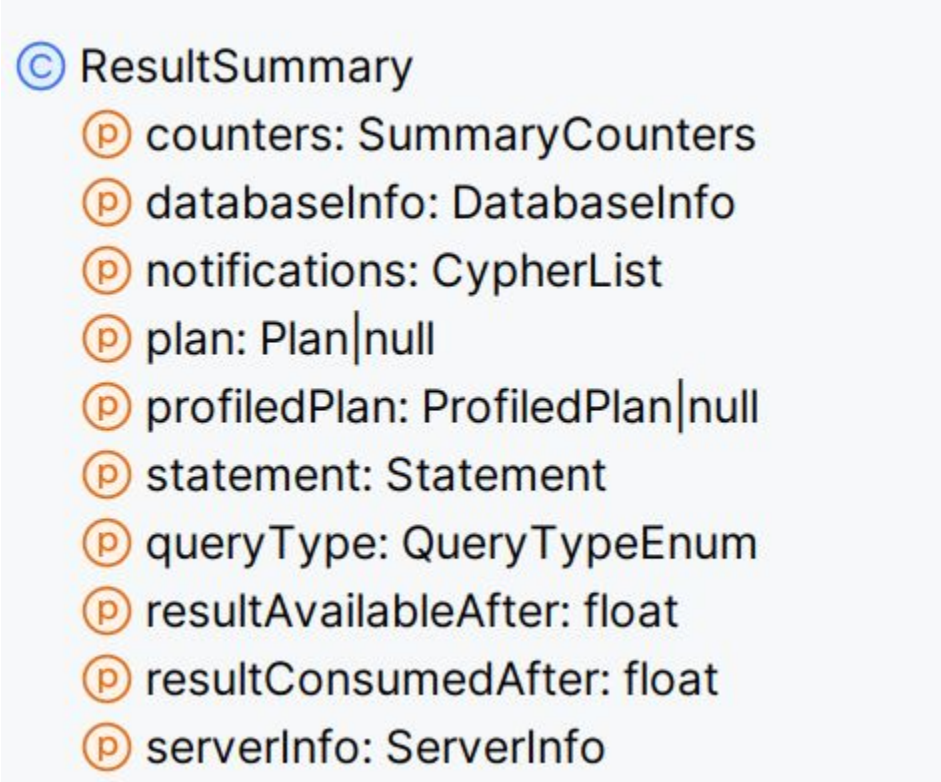
```php
$results = $session->run('UNWIND range(1, 100) AS x RETURN x');

foreach ($results as $row) {
  echo $row->get('x')->getLabels()->first();
}

// Undefined behaviour across drivers.
// - results may be cached and can be reiterated
// - results may throw an exception
// - results may just iterate over an empty set
foreach ($results as $row) {
  echo $row->get('x')->getLabels()->first();
}
```

Results: Metadata

result_metadata.php

```php
$results = $session->run(<<<'CYPHER'
UNWIND range(1, 100) AS x
CREATE (n:Node {id: x})
RETURN n
CYPHER);

echo $results->getSummary()->getCounters()->getNodesCreated(); // 100
```

Results: Metadata Overview



Results: Key Takeaways

- Results are **cursors**, read them once
- Results keep a lot of **useful metadata** for debugging or logging
- Results keep a **connection open** until they are consumed
- Results return neo4j **specific objects** too
- Querying some of the results metadata requires **full consumption** of the results

Final Takeaways

- Understand the building blocks
- Optimise by using lower level configurations carefully
- Debug and Profile with result summaries
- Graph Databases support real time applications
- Graph Databases have great ecosystems for data science
- Have fun!

[nagels]

Rolling Your own MCP Server

DEMO TIME

[nagels]

The end

Questions?

Want this slide deck for free?
Contact Me