



Cassandra Day
2022

Stargate – An OSS API Layer for C*

Sponsored by DataStax

Agenda

01



Stargate Data gateway
What, Why and How

02



Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



Tooling
SDK, K8ssandra

06



What's NEXT ?
v2 & revised arch

Agenda

01



Stargate Data gateway
What, Why and How

02



Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



Tooling
SDK, K8ssandra

06



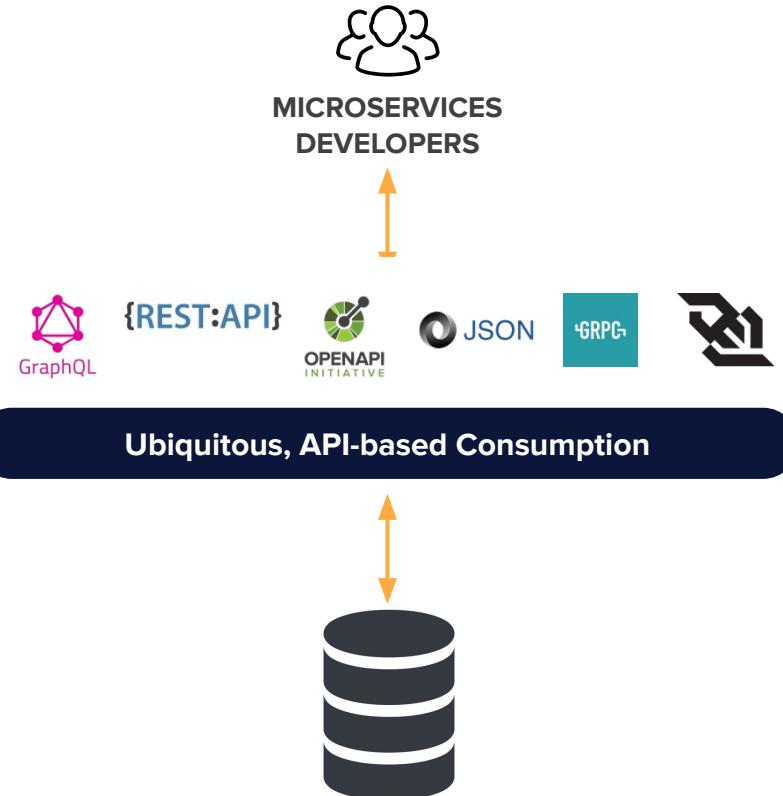
What's NEXT ?
v2 & revised arch

Data Gateway



Developers want the option to use modern APIs and development gateways.

Cassandra is a database designed for the new standard and the associated APIs that facilitate the first choice of developers.



Stargate Overview

An open source API framework for data

Stargate makes it easy to use a database for any application workload by adding plugin support for new APIs, data types, and access methods



The image shows a screenshot of the Stargate GitHub repository and its official landing page.

GitHub Repository (Top Right):

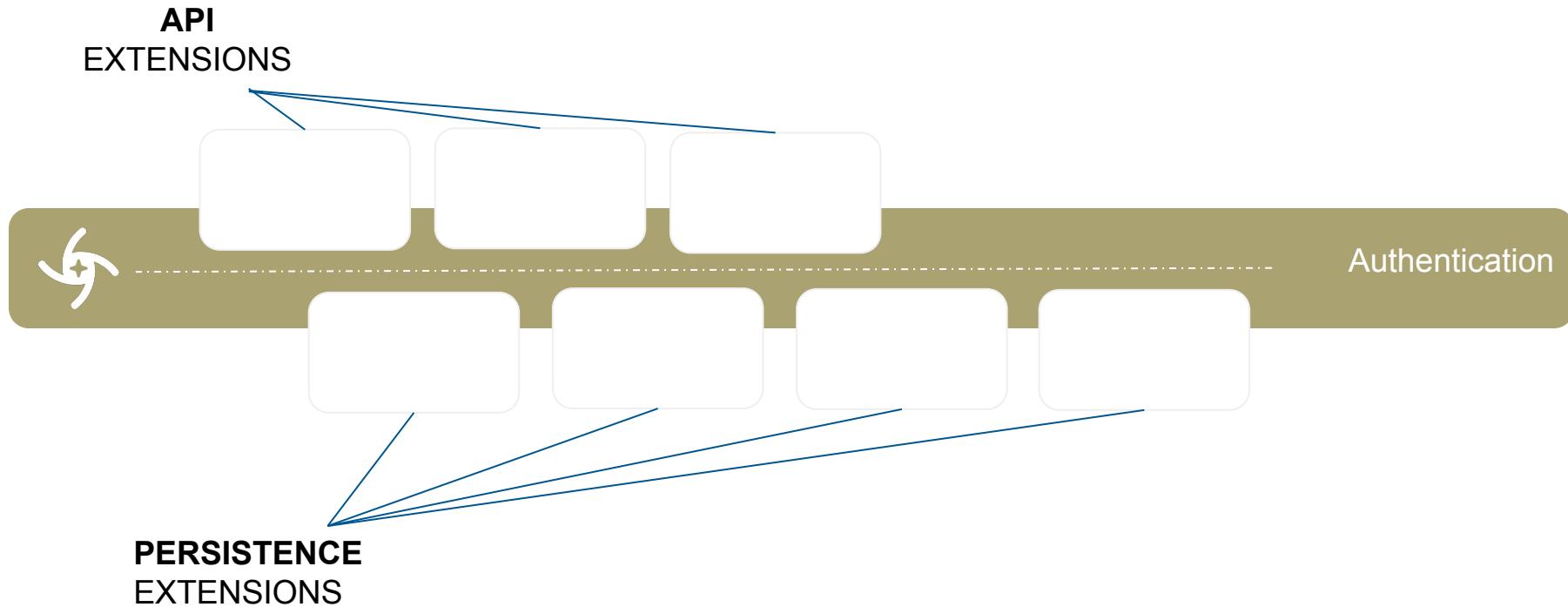
- Repository name: stargate / stargate
- Public status
- Watch: 21, Fork: 91, Star: 659 (circled in red)
- Code tab selected
- Branches: master, 83 branches
- Tags: 111 tags
- Actions, Projects, Security, Insights buttons
- Code dropdown menu
- About section: An open source data gateway
- Project links: stargate.io, Java, GraphQL, REST, Cassandra, CQL
- Readme, Apache-2.0 license, Code of conduct, Stars: 659, Watchers: 21, Forks: 91
- Releases: 111, Release v1.0.64 (Latest, 4 days ago), 110 releases
- Packages: 13, io.stargate.stargate, io.stargate.db.persistence-api, io.stargate.db.persistence-common, > 10 packages
- Contributors: 35 (circled in red)

Stargate Landing Page (Bottom Left):

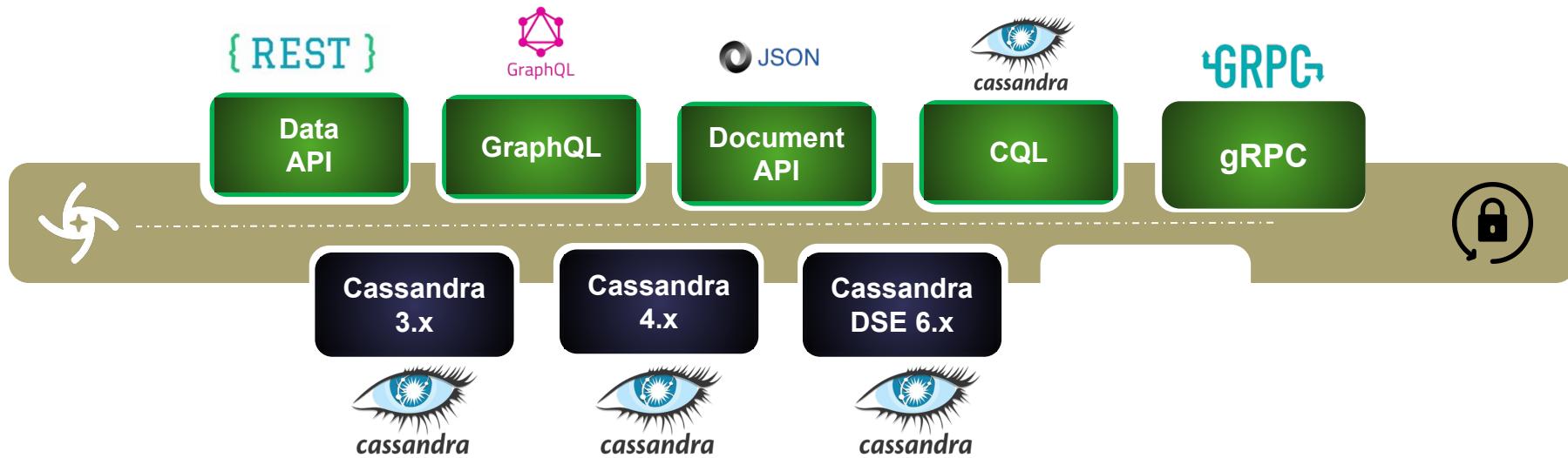
- Stargate logo
- Astronaut illustration holding a briefcase and a wrench
- Section title: OPEN SOURCE DATA GATEWAY
- Description: Stargate is an open source data gateway that sits between your app and your databases. Stargate brings together an API platform and data request coordination code into one OSS project.
- Get Started button
- Github button
- Text: Stargate is the official data API for DataStax Astra and DataStax Enterprise! Try It Now!
- Recent commits:

 - health-checker: Bumping version to v1.0.65-SNAPSHOT (#2059) 4 days ago
 - metrics-jersey: Bumping version to v1.0.65-SNAPSHOT (#2059) 4 days ago

API Extensions and Persistence Extensions



API Extensions and Persistence Extensions



9042

8090

8082

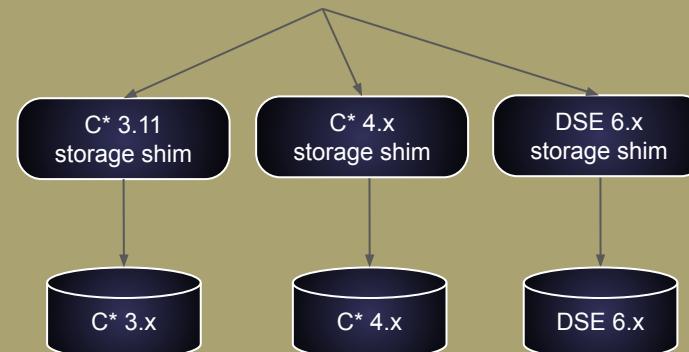
8080

8084



Data + Document APIs

GraphQL APIs



9042

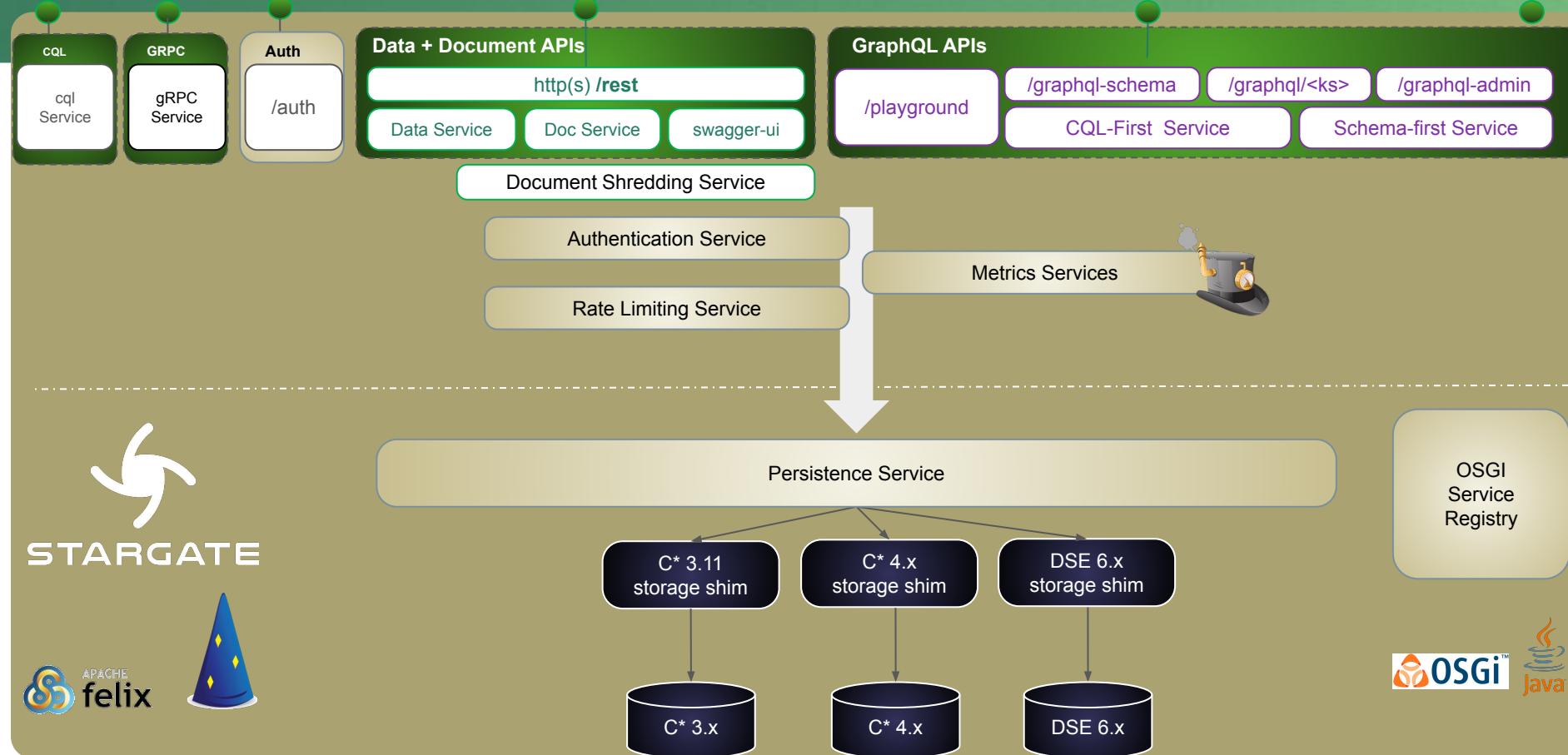
8090

8081

8082

8080

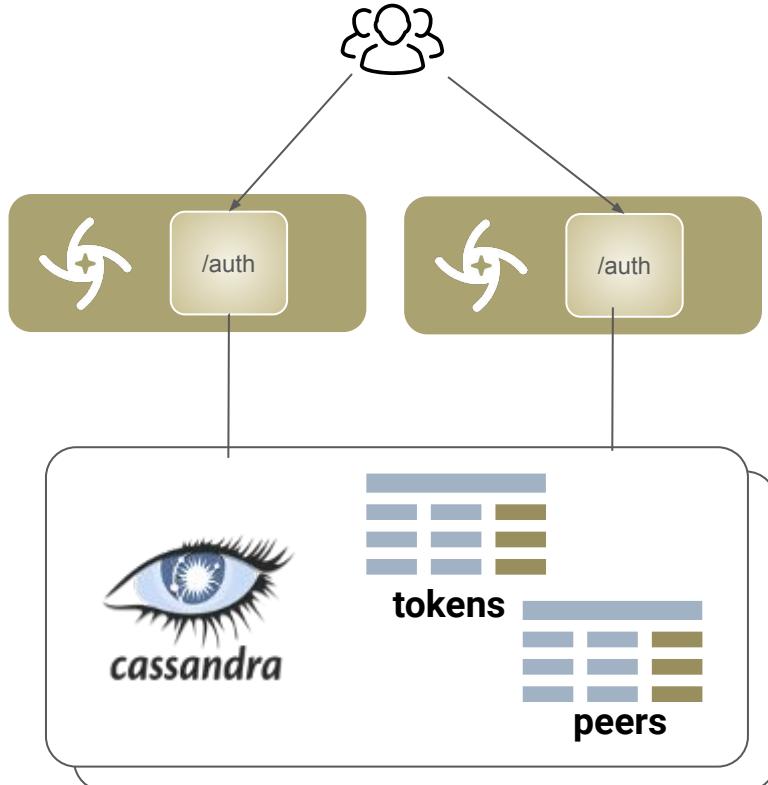
8084



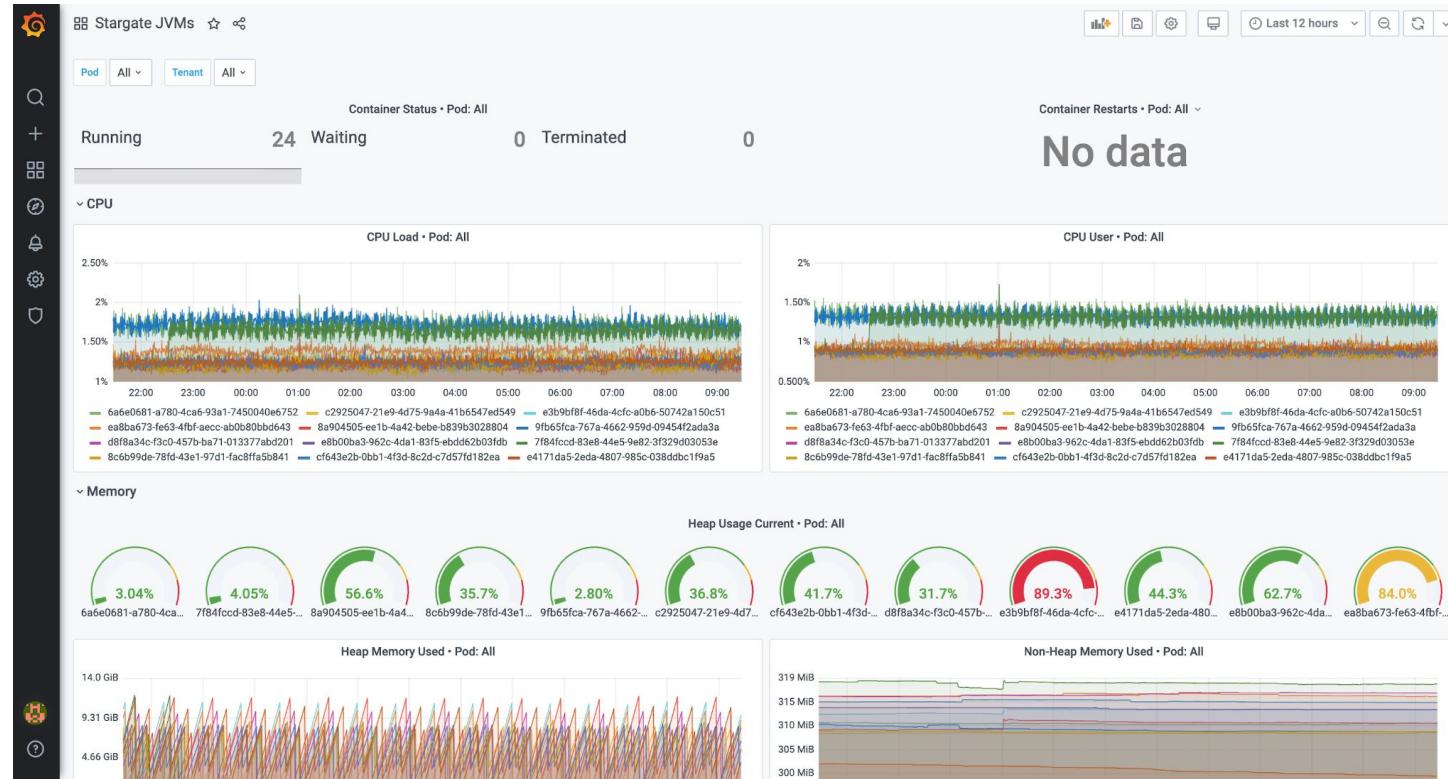
Authentication / Authorization

- Use Auth and get tokens (user+pwd)
- Tokens shared across instances
- Leverage on Cassandra roles (RBAC)
- -Dstargate.auth_tokenttl =X

```
curl -L -X POST 'http://localhost:8081/v1/auth' \
-H 'Content-Type: application/json' \
--data-raw '{
    "username": "cassandra",
    "password": "cassandra"
}'
```



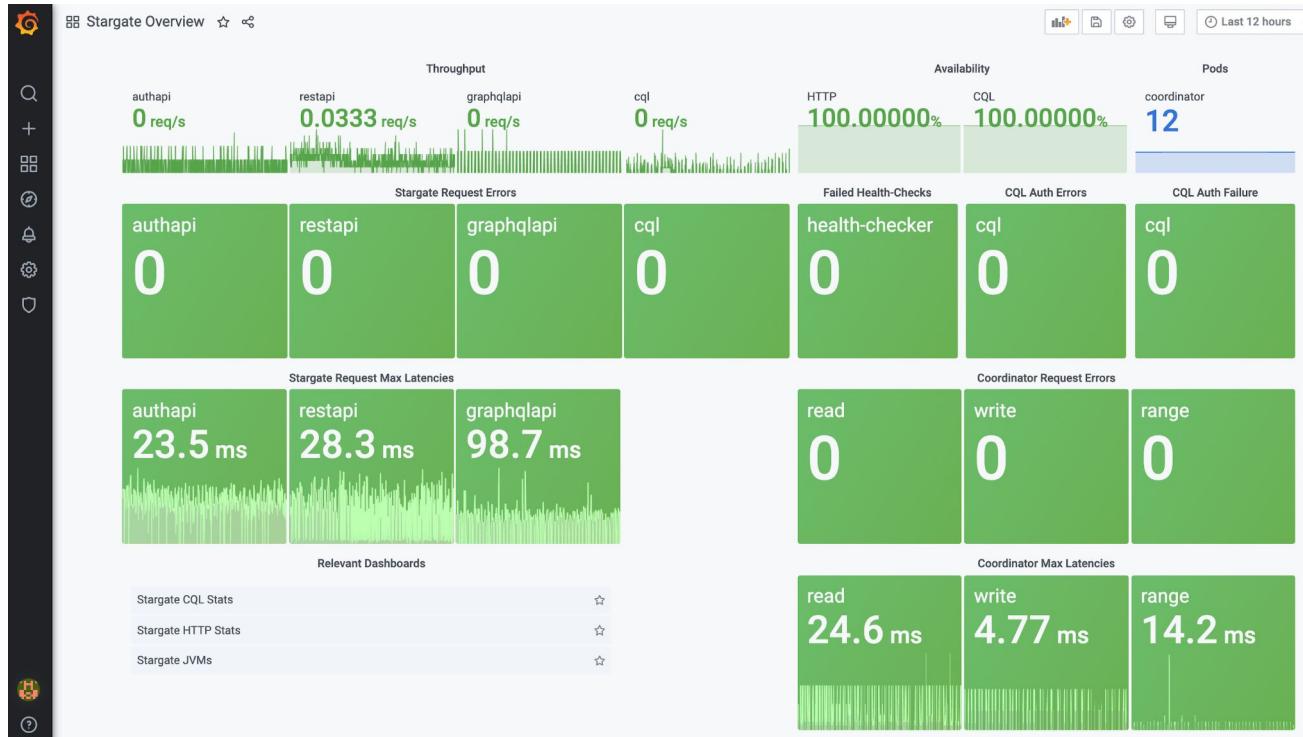
Monitoring (JVM)



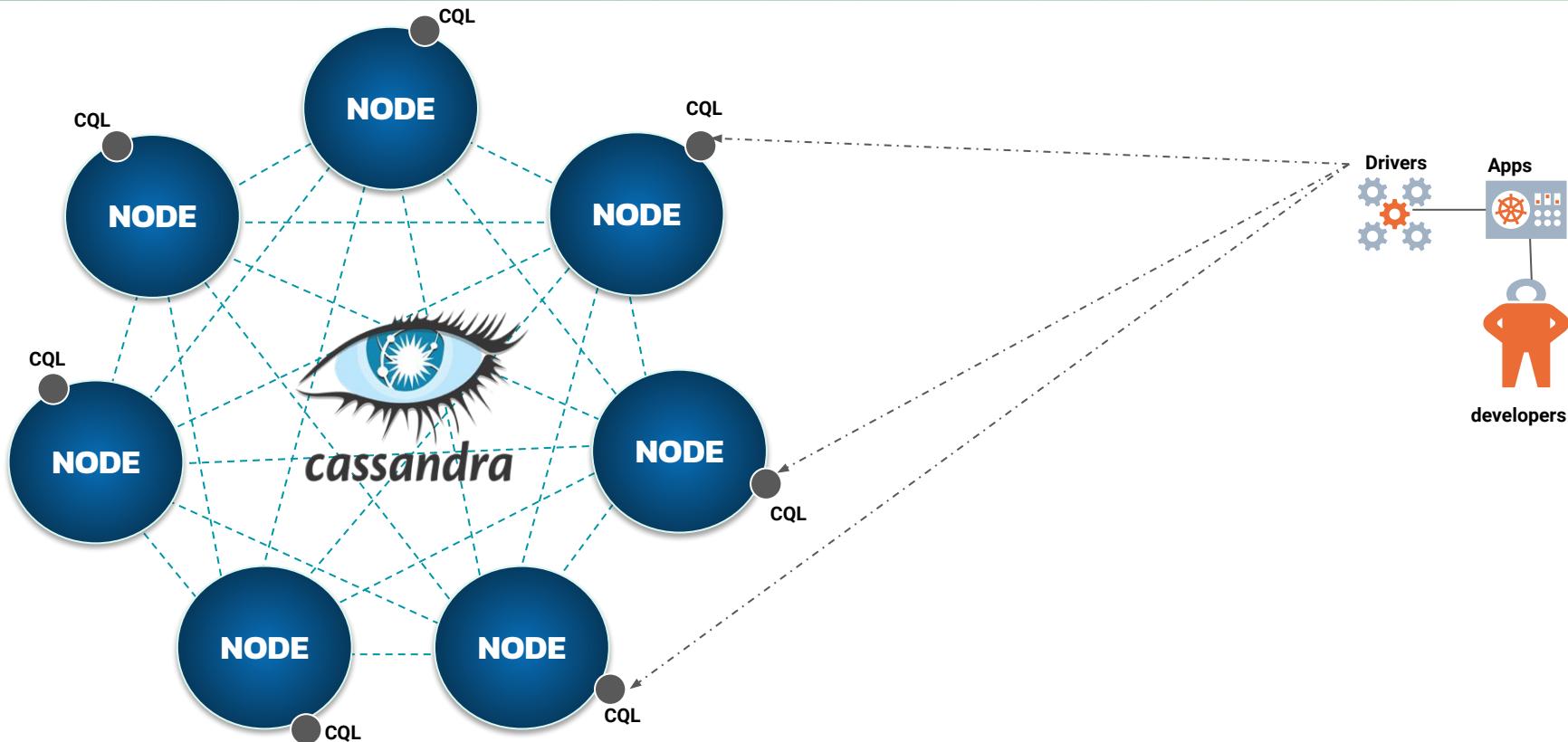
Monitoring (HTTP Traffic)



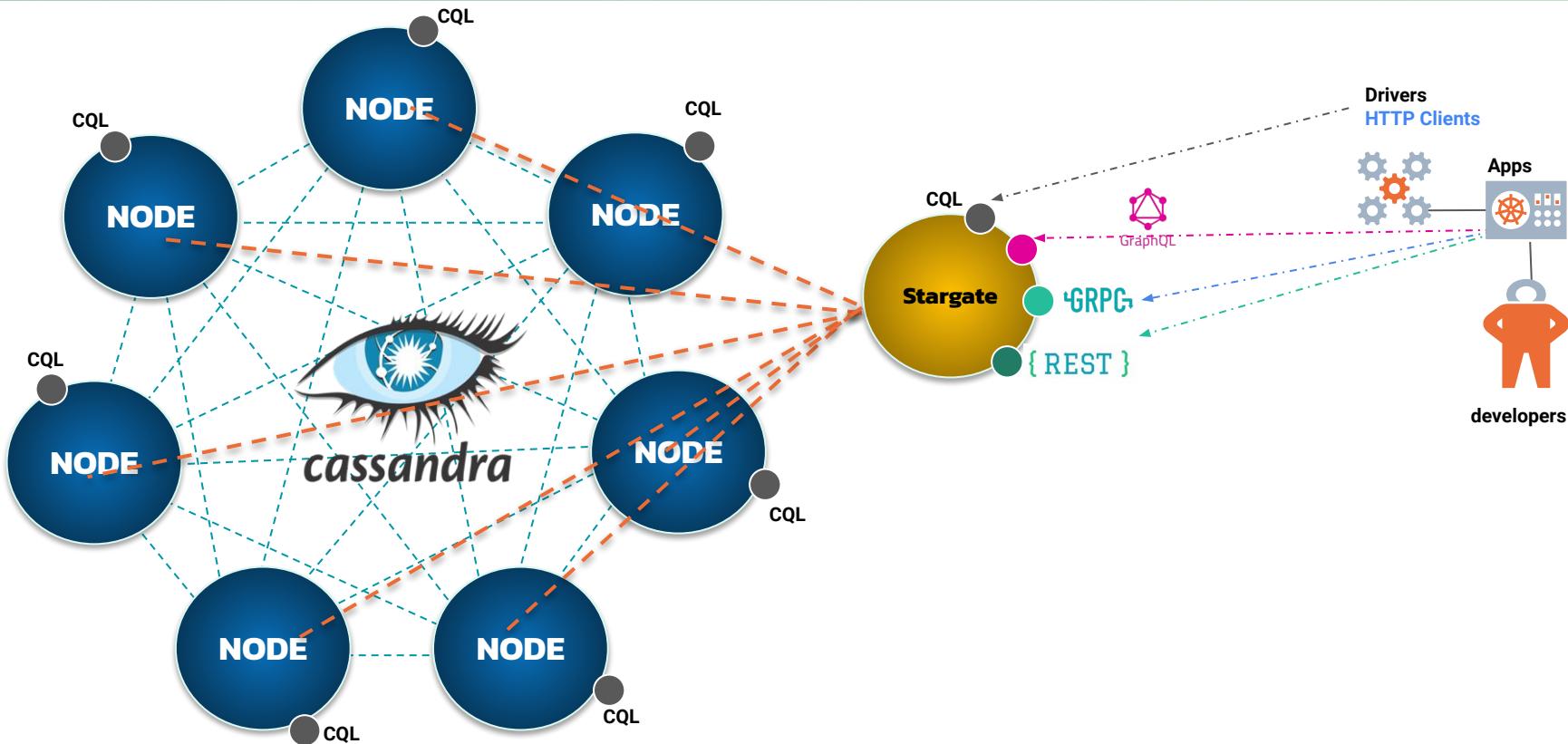
Monitoring (Services)



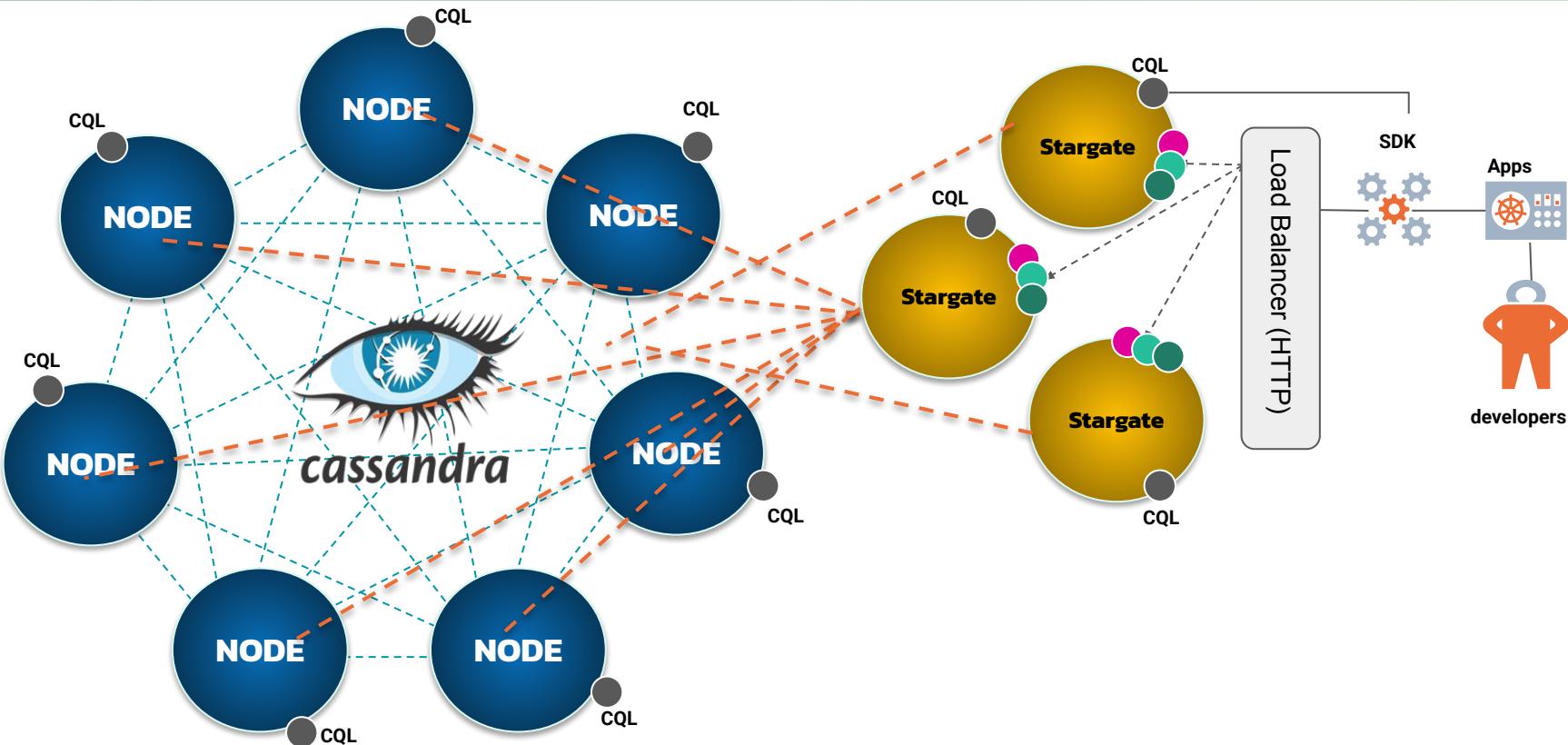
Connecting to your cluster (Before)



Connecting to your cluster (with Stargate)



Connecting to your cluster (with Stargates)



Agenda

01



Stargate Data gateway
What, Why and How

02

{REST}

Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



Tooling
SDK, K8ssandra

06



What's NEXT ?
v2 & revised arch

Stargate "REST" Api(s)

- Expose existing CQL schema as a RESTful endpoint, or create new CQL schema
- Most familiar API style for most teams
- Swagger integration

schemas	
GET	/v1/keyspaces/{keyspaceName}/tables/{tableName}/columns Retrieve all columns
POST	/v1/keyspaces/{keyspaceName}/tables/{tableName}/columns Add a column
GET	/v1/keyspaces/{keyspaceName}/tables/{tableName}/columns/{columnName} Retrieve a column
PUT	/v1/keyspaces/{keyspaceName}/tables/{tableName}/columns/{columnName} Update a column
DELETE	/v1/keyspaces/{keyspaceName}/tables/{tableName}/columns/{columnName} Delete a column
GET	/v1/keyspaces Return all keyspaces
GET	/v1/keyspaces/{keyspaceName}/tables Return all tables
POST	/v1/keyspaces/{keyspaceName}/tables Add a table
GET	/v1/keyspaces/{keyspaceName}/tables/{tableName} Return a table
DELETE	/v1/keyspaces/{keyspaceName}/tables/{tableName} Delete a table
GET	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/columns Get all columns

Quick start: https://stargate.io/docs/stargate/1.0/quickstart/quick_start-rest.html

REST Api(s) expose DDL and best practices DML



some CQL capabilities left out

GET	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/columns	Get all columns
POST	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/columns	Create a column
GET	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/columns/{columnName}	Get a column
PUT	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/columns/{columnName}	Update a column
DELETE	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/columns/{columnName}	Delete a column
DELETE	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/indexes/{indexName}	Drop an index from keyspace
GET	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/indexes	Get all indexes for a given table
POST	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}/indexes	Add an index to a table's column
GET	/v2/schemas/keyspaces/{keyspaceName}	Get a keyspace
DELETE	/v2/schemas/keyspaces/{keyspaceName}	Delete a keyspace
GET	/v2/schemas/keyspaces	Get all keyspaces
POST	/v2/schemas/keyspaces	Create a keyspace
GET	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}	Get a table
PUT	/v2/schemas/keyspaces/{keyspaceName}/tables/{tableName}	Replace a table definition

DDL (Schema)

GET	/v2/keyspaces/{keyspaceName}/{tableName}/rows	Retrieve all rows
GET	/v2/keyspaces/{keyspaceName}/{tableName}	Search a table
POST	/v2/keyspaces/{keyspaceName}/{tableName}	Add row
GET	/v2/keyspaces/{keyspaceName}/{tableName}/{primaryKey}	Get row(s)
PUT	/v2/keyspaces/{keyspaceName}/{tableName}/{primaryKey}	Replace row(s)
DELETE	/v2/keyspaces/{keyspaceName}/{tableName}/{primaryKey}	Delete row(s)
PATCH	/v2/keyspaces/{keyspaceName}/{tableName}/{primaryKey}	Update part of a row(s)

Agenda

01



Stargate Data gateway
What, Why and How

02



Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



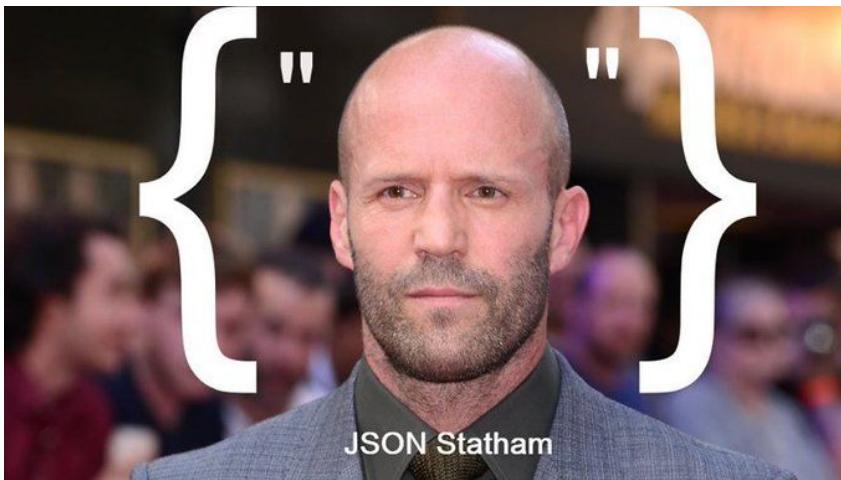
Tooling
SDK, K8ssandra

06



What's NEXT ?
v2 & revised arch

"Schemaless"



"What rules ?"

- You want to insert and retrieve any JSON documents efficiently
- Allow "schemaless" (Validation-less)
- Write to a single document is a single batch of statements
- Read from a single document is a single SELECT statement.
- Limit Tombstones with range deletes

Different JSON shapes



```
{  
  "comment": "found the Golden Fleece",  
  "group": "The Argonauts",  
  "properties": {  
    "class": "myth",  
    "nationality": "Greece"  
  }  
}
```

```
{  
  "movies": [  
    "The Expendables",  
    "Fast and Furious"  
  ],  
  "properties": {  
    "birthdate": 1967,  
    "class": "person",  
    "nationality": "English"  
  },  
  "surname": "Statham"  
}
```



Stargate Document API

- Schemaless! (validation-less)
- Add any JSON document to a collection
- Uses "Document shredding" approach

The screenshot shows the Stargate Document API's Swagger UI interface. At the top, there's a navigation bar with the Stargate logo, the URL '/swagger.json', and a 'Explore' button. Below the navigation, the title 'documents' is centered. A list of API endpoints is displayed, each with a color-coded button indicating the HTTP method (e.g., GET, POST, PUT, DELETE, PATCH) and the corresponding URL path. The descriptions for each endpoint provide a brief summary of the operation.

Method	Path	Description
GET	/v2/namespaces/{namespace-id}/collections	List collections in namespace
POST	/v2/namespaces/{namespace-id}/collections	Create a new empty collection in a namespace
GET	/v2/namespaces/{namespace-id}/collections/{collection-id}	Search documents in a collection
POST	/v2/namespaces/{namespace-id}/collections/{collection-id}	Create a new document
DELETE	/v2/namespaces/{namespace-id}/collections/{collection-id}	Delete a collection in a namespace
POST	/v2/namespaces/{namespace-id}/collections/{collection-id}/upgrade	Upgrade a collection in a namespace
POST	/v2/namespaces/{namespace-id}/collections/{collection-id}/batch	Write multiple documents in one request
GET	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}	Get a document
PUT	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}	Create or update a document with the provided document-id
DELETE	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}	Delete a document
PATCH	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}	Update data at the root of a document
GET	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}/{document-path}	Get a path in a document
PUT	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}/{document-path}	Replace data at a path in a document
DELETE	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}/{document-path}	Delete a path in a document
PATCH	/v2/namespaces/{namespace-id}/collections/{collection-id}/{document-id}/{document-path}	Update data at a path in a document
GET	/v2/namespaces/{namespace-id}/collections/{collection-id}/json-schema	Get a JSON schema from a collection

Quick start: https://stargate.io/docs/stargate/1.0/quickstart/quick_start-document.html

Document Shredding (1 / 3)

```
create table <name> (
    key text,
    p0 text,
    ... p[N] text,
    bool_value boolean,
    txt_value text, d
    bl_value double, leaf text
)
```

Document Shredding (2 / 3)

```
{"a": { "b": 1 }, "c": 2}
```

The document would be “shredded” into rows looking like this:

key	p0	p1	dbl_value
x	a	b	1
x	c	null	2

Document Shredding (3 / 3)

For data with an array, such as:

```
{"a": { "b": 1 }, "c": [{"d": 2}]}
```

there would be two rows, like so:

key	p0	p1	p2	dbl_value
x	a	b	null	1
x	c	[0]	d	2

Agenda

01



Stargate Data gateway
What, Why and How

02



Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



Tooling
SDK, K8ssandra

06



What's NEXT ?
v2 & revised arch



History

GraphQL

- 2012 Created by Facebook
Used internally for mobile apps
- 2015 Facebook give talk at ReactJs Conf
- 2015 Facebook open sourced GraphQL
- 2016 Github announced move to GQL

Definition

GraphQL is an application programming interface (API) query language and server-side runtime that prioritises giving customers precisely the data they request.

landscape.graphql.org/?category=graph-ql-adopter&grouping=category&style=borderless

Why?



GraphQL

Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```

Ask for what you want

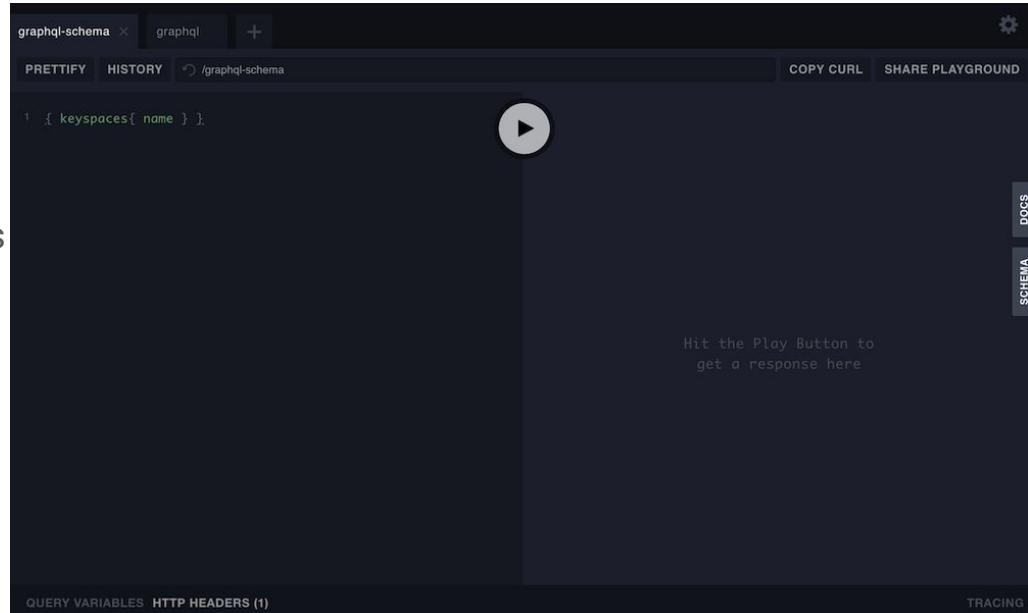
```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```

Stargate GraphQL API

- Expose existing CQL schema as a GraphQL endpoint
- Allows more fine-grained control of what fields returned
- Compatible with (Apollo) federation
- Easy experimentation in GraphQL playground



Quick start: stargate.io/docs/latest/quickstart/qs-graphql-cql-first.html

GraphQL "CQL-first"

The screenshot shows the GraphQL playground interface with a dark theme. The left pane contains a GraphQL mutation query:

```
1 mutation {
2   books: createTable(
3     keyspaceName:"keyspace3",
4     tableName:"books",
5     partitionKeys: [ # The keys required to access your data
6       { name: "title", type: {basic: TEXT} }
7     ]
8     values: [ # The values associated with the keys
9       { name: "author", type: {basic: TEXT} }
10    ]
11  )
12  authors: createTable(
13    keyspaceName:"keyspace3",
14    tableName:"authors",
15    partitionKeys: [
16      { name: "name", type: {basic: TEXT} }
17    ]
18    clusteringKeys: [ # Secondary key used to access values with
19      { name: "title", type: {basic: TEXT}, order: "ASC" }
20    ]
21  )
22 }
```

The right pane shows the response from the GraphQL server:

```
{ "data": { "books": true, "authors": true } }
```

Below the code editor, there are sections for "QUERY VARIABLES" and "HTTP HEADERS (1)".

QUERY VARIABLES

```
1 {
2   "x-cassandra-token": "7c37bda5-7360-4d39-96bc-9765db5773bc"
3 }
```

HTTP HEADERS (1)

```
1 {
2   "x-cassandra-token": "7c37bda5-7360-4d39-96bc-9765db5773bc"
3 }
```

At the bottom right, there are buttons for "COPY CURL", "SHARE PLAYGROUND", "DOCS", and "SCHEMA".

GraphQL "CQL-first"

The screenshot shows a GraphQL playground interface with the following details:

- GraphQL Schema:** graphql-schema
- Query:** graphql
- Path:** /graphql/keyspace3
- Buttons:** PRETTIFY, HISTORY, COPY CURL, SHARE PLAYGROUND
- Side Navigation:** DOCS, SCHEMA
- Main Area (Left):** A code editor containing a GraphQL mutation query:

```
1 # Write your query or mutation here
2 mutation {
3   moby: insertBooks(value: {title:"Moby Dick", author:"Herman Melville"})
4     value {
5       title
6     }
7   }
8   catch22: insertBooks(value: {title:"Catch-22", author:"Joseph Heller"})
9     value {
10       title
11     }
12 }
```
- Main Area (Right):** A JSON response tree showing the mutation results:

```
{
  "data": [
    {
      "moby": {
        "value": {
          "title": "Moby Dick"
        }
      },
      "catch22": {
        "value": {
          "title": "Catch-22"
        }
      }
    }
  ]
}
```
- Bottom Left:** QUERY VARIABLES, HTTP HEADERS (1)
1
2 "x-cassandra-token": "7c37bda5-7360-4d39-96bc-9765db5773bc"
3
- Bottom Right:** TRACING

GraphQL "Schema-first"

```
type Book @key @cql_entity(name: "book") @cql_input {
    title: String! @cql_column(partitionKey: true, name: "book_title")
    isbn: String @cql_column(clusteringOrder: ASC)
    author: [String] @cql_index(name: "author_idx", target: VALUES)
}

type Address @cql_entity(target: UDT) @cql_input {
    street: String
    city: String
    state: String
    zipCode: String @cql_column(name: "zip_code")
}

#...
|
type Query {
    bookByTitleAndIsbn(title:String!, isbn:String): [Book]
    readerByNameAndUserid(name:String!, user_id:Uuid): [Reader]
}

type Mutation {
    insertBook(book:BookInput!): Book
    updateBook(book:BookInput!): Boolean @cql_update
    deleteBook(book:BookInput!): Boolean
    insertReader(reader:ReaderInput!): Reader
    deleteReader(reader:ReaderInput!): Boolean
    insertLibCollection(libColl: LibCollectionInput!): LibCollection
    deleteLibCollection(libColl: LibCollectionInput!): Boolean
}
```

GraphQL "Schema-first"

```
mutation {
  deploySchema(
    keyspace: "library"
    expectedVersion: "1da4f190-b7fd-11eb-8258-1ff1380eaff5"
    schema: """
      # Stargate does not require definition of fields in @key,
      # it uses the primary key
      type Book @key @cql_entity(name: "book") @cql_input {
        title: String! @cql_column(partitionKey: true, name: "book_title")
        isbn: String @cql_column(clusteringOrder: ASC)
        author: [String] @cql_index(name: "author_idx", target: VALUES)
      }
      ...
    }
}
```

GraphQL "Schema-first"

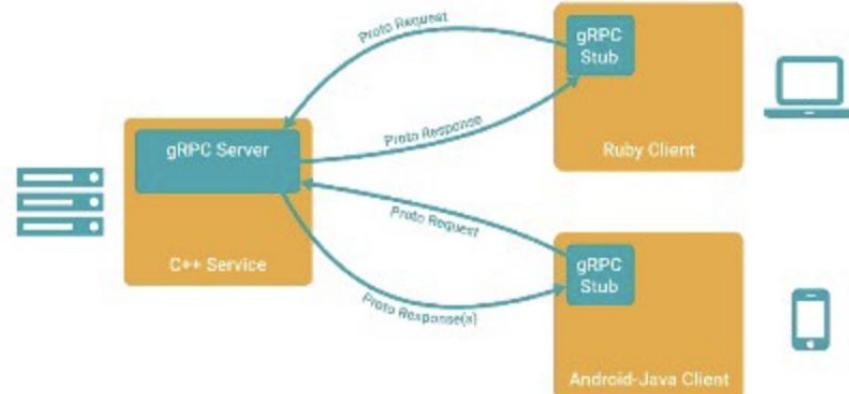
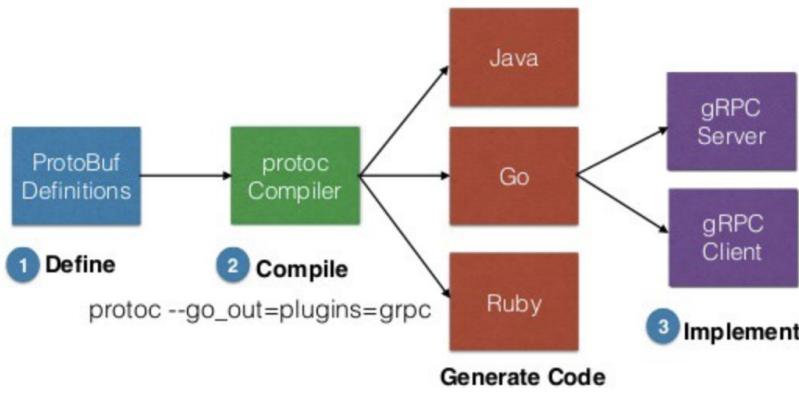
```
query fetchBook {
  book(title: "Native Son") {
    title
    author
  }
}

type Query {
bookGT(
  title: String
  isbn: String @cql_where(field: "isbn", predicate: GT)
): [Book]
}

type Query {
readerCONTAINS(
  reviews: ReviewInput! @cql_where(field: "reviews", predicate: CONTAINS)
): [Reader]
}

type Query {
booksIn(
  title: [String!] @cql_where(field: "title", predicate: IN)
): [Book]
}
```

gRPC



@hostirosti @grpcio

Comparing Stargate APIs

Drivers



Cassandra Query Language

SQL like Table Model

Structured Data

Key-Value Data

Strong Types

Minimal query overhead



gRPC

Structured Data (CQL)

Lighter weight

Native driver alternative

Low query overhead



GraphQL

Hierarchy of types and fields

Structured Data

Key-Value Data

Low query overhead

Open API



REST

Row based

Structured Data

Key-Value Data

Weaker Types

High query overhead



Document

JSON Documents

Semi-Structured Data

Weaker Types

High query overhead



More Performant

More Flexible

Agenda

01



Stargate Data gateway
What, Why and How

02

{REST}

Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



Tooling
SDK, K8ssandra

06



What's NEXT ?
v2 & revised arch

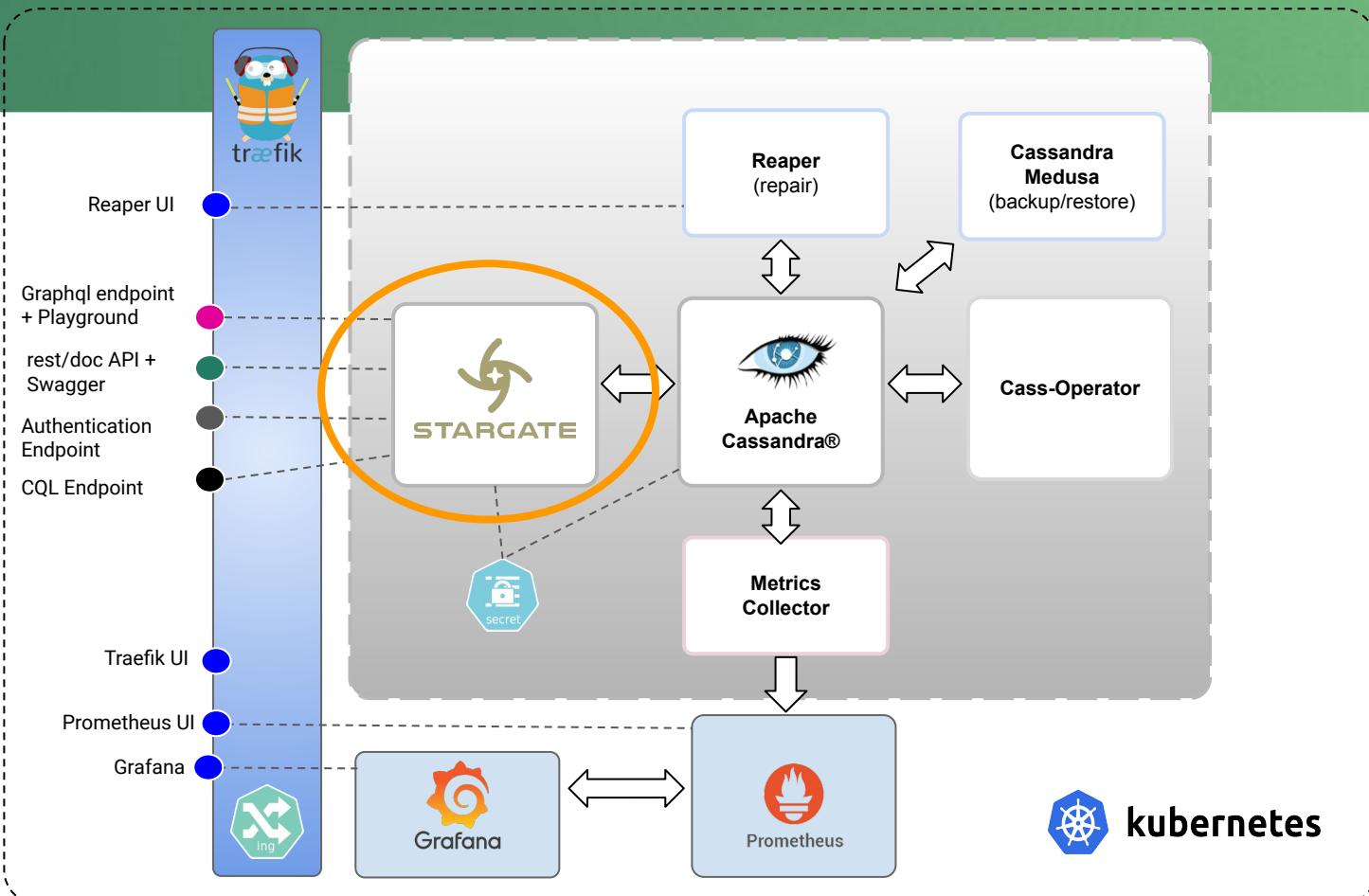
Docker

The screenshot shows the Docker Hub user profile for the organization **stargateio**. The profile page includes the user's logo (a yellow gear with a green lightning bolt), name, community type, authors, and joining date. Below the profile, a section titled "Repositories" displays 9 repositories, each with a thumbnail, name, author, update time, download count, and star count.

Repository	Author	Last Updated	Downloads	Stars
stargateio/stargate-3_11	stargateio	3 days ago	100K+	4
stargateio/stargate-4_0	stargateio	3 days ago	50K+	4
stargateio/stargate-dse-68	stargateio	3 days ago	8.8K	6
stargateio/restapi	stargateio	5 days ago	5.0K	0
stargateio/stargate-3_11	stargateio	3 days ago	100K+	4
stargateio/stargate-4_0	stargateio	3 days ago	50K+	4
stargateio/stargate-dse-68	stargateio	3 days ago	8.8K	6
stargateio/restapi	stargateio	5 days ago	5.0K	0
stargateio/stargate-3_11	stargateio	3 days ago	100K+	4



K8SANDRA



SDK (Java, Javascript, Python)

The image shows the Stargate API documentation interface. At the top left is the Stargate logo. To its right are two buttons: "Swagger" and "GraphQL Playground". Below these are five green cards representing different API endpoints:

- REST DATA APIs**:
 - V1: Tables, Rows, Prim.Key
 - V2: Tables, Rows, Prim.Key, Keyspaces, Indexes, UDT
- DOCUMENT API**:
 - Namespaces
 - Sub Doc..
 - Json Schema
 - Collections
 - Documents
- GraphQL**:
 - CQL FIRST: Tables, UDT, Values, Keyspaces, Indexes
 - Schema FIRST: Schemas, Entity, Query, Mutations, Federation
- CQL**: CqlSession
- GRPC**: Queries, Batches



Postman

The screenshot shows the Postman application interface. The left sidebar contains navigation links: Product, Pricing, Enterprise, Resources and Support, Explore, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace is titled "DataStax Astra DB Stargate". A "Create 2 tables, book and reader" POST request is selected. The "Body" tab is active, showing a GraphQL mutation. The "QUERY" section contains the following code:

```
1 mutation createTables {
2   book: createTable(
3     keyspaceName:"{{gkeyspace}}",
4     tableName:"{{gtable1}}",
5     partitionKeys: [ # The keys required to access your data
6       { name: "title", type: {basic: TEXT} }
7     ]
8     clusteringKeys: [
9       { name: "author", type: {basic: TEXT} }
10    ]
11  )
12  reader: createTable(
13    keyspaceName:"{{gkeyspace}}",
14    tableName:"{{gtable2}}",
15    partitionKeys: [
16      { name: "name", type: {basic: TEXT} }
17    ]
18    clusteringKeys: [ # Secondary key used to access values within the row
19      { name: "user_id", type: {basic: UUID}, order: "ASC" }
20    ]
21    columnDefinitions: [
22      { name: "birthdate", type: {basic: DATE} },
23      { name: "email", type: {basic: SET, info:{ subTypes: [ { basic: TEXT } ] } } },
24      { name: "reviews", type: {basic: TUPLE, info: { subTypes: [ { basic: TEXT } ] } } },
25      { name: "addresses", type: {basic: LIST, info: { subTypes: [ { basic: TEXT } ] } } }
26    ]
27  )
28}
```

<https://www.postman.com/datastax/workspace/datastax-astra-db-stargate>

Agenda

01



Stargate Data gateway
What, Why and How

02

{REST}

Exploring Apis
Rest DML and DDL

03



Exploring Apis
Document oriented

04



Exploring Apis
GraphQL and gRPC

05



Tooling
SDK, K8ssandra

06



What's NEXT ?
v2 & revised arch

Next: upcoming Stargate V2

The first v2 beta just came out!

stargate.io/2022/09/11/stargate-v2-beta.html

Each service in its pod

can scale them independently

Use gRPC internally

a new "bridge" introduced

From Spring Boot to Quarkus

native compilation, faster binary

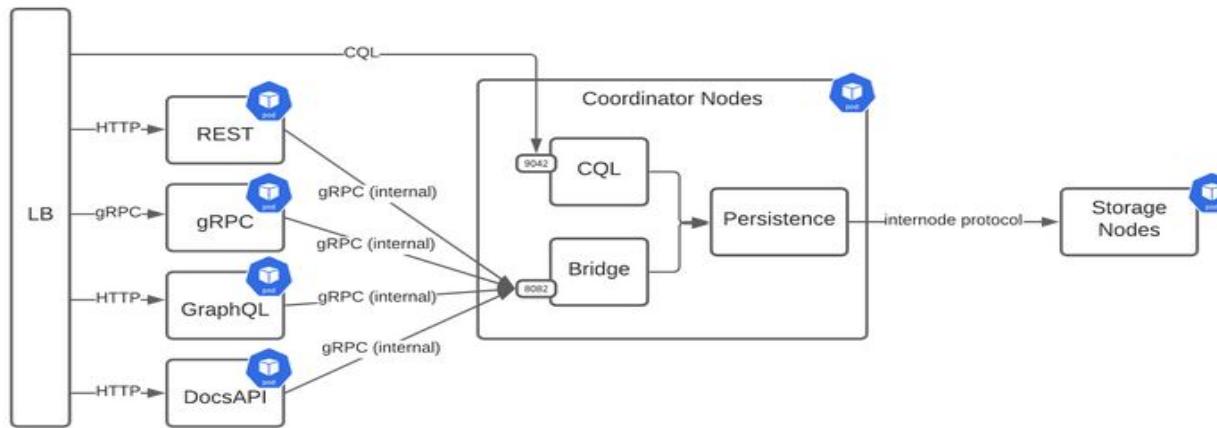
Document API revision

also long-term plans for new Doc API features

V2, redesigned architecture

Meet the winning design

After weighing the pros and cons of the other potential designs, we finally settled on what we believe is the best option for Stargate v2. Take a look at the diagram below.



Source: stargate.io/2021/11/02/introducing-the-design-for-stargate-v2.html

Find out more & get involved



Stargate Discord

discord.com/invite/5gY8GDB

Website

stargate.io

Github

github.com/stargate/stargate

Twitter

[@stargateio](https://twitter.com/stargateio)

Linkedin

[linkedin.com/groups/9091327](https://www.linkedin.com/groups/9091327)



Thank you!



Sponsored by DataStax