Application development

Need a layer between your app and the database

Standard answer: what you need is a **driver**.

Your App

Your API

DRIVER

Your Serverless
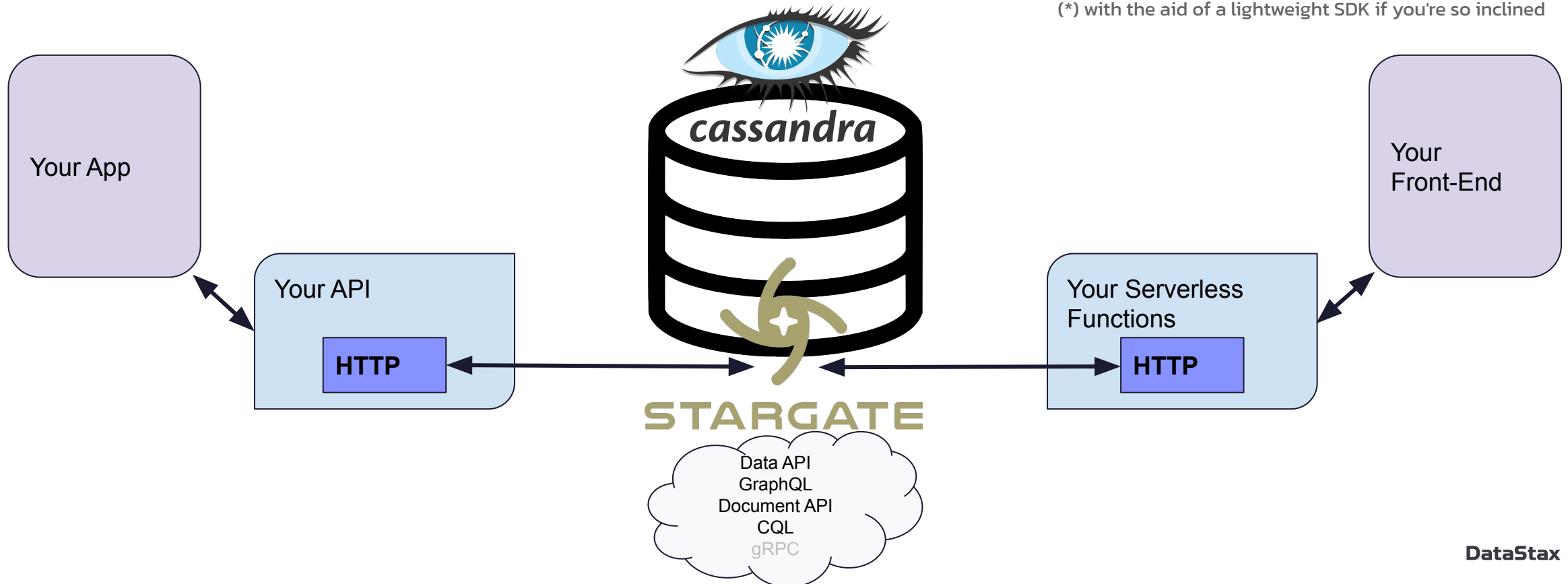Functions

DRIVER
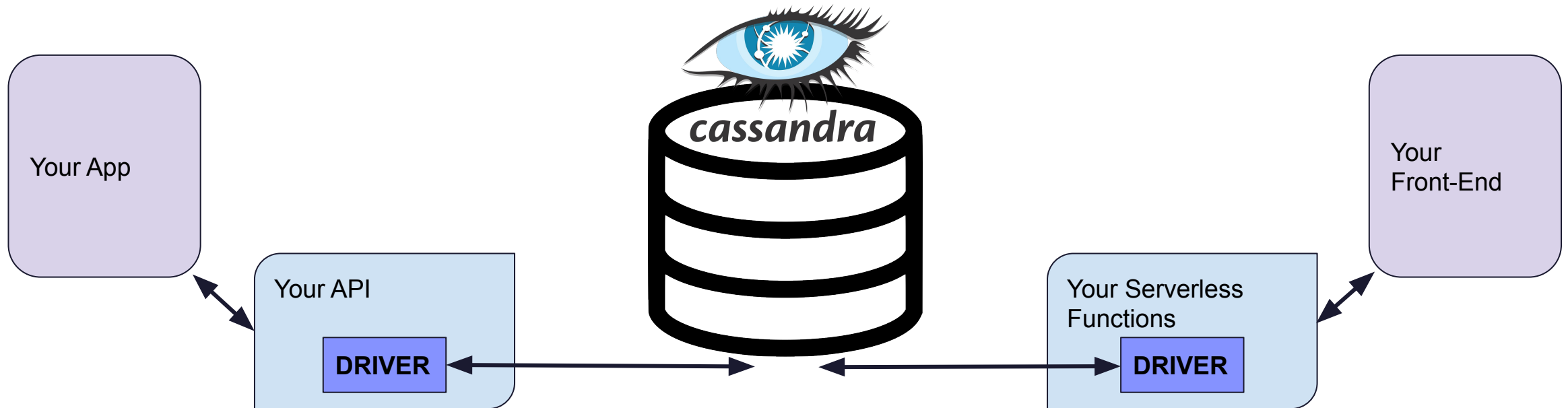
Your
Front-End

Application development

Need a layer between your app and the database

Alternative answer: can place an *API Data Layer* and do simple HTTP requests*

(*) with the aid of a lightweight SDK if you're so inclined

Your App

cassandra

Your
Front-End

Your API

HTTP

STARGATE

Your Serverless
Functions

HTTP

Data API
GraphQL
Document API
CQL
gRPC

DataStax

Drivers

Cassandra drivers exist for most popular languages

Connectivity	Query	Parsing Results
★ Token & Datacenter Aware	★ CQL Support	★ Lazy Load
★ Load Balancing Policies	★ Schema Management	★ Object Mapper
★ Retry Policies	★ Sync/Async/Reactive API	★ Spring Support
★ Reconnection Policies	★ Query Builder	★ Paging
★ Connection Pooling	★ Compression	
★ Health Checks	★ Paging	
★ Authentication	Authorization		
★ SSL		

DataStax

Drivers

Cassandra drivers exist for most popular languages

Documentation:
docs.datastax.com/en/developer/nodejs-driver/4.6/

Examples today:
github.com/hemidactylus/cassandra-nodejs-drivers-practice

Installing

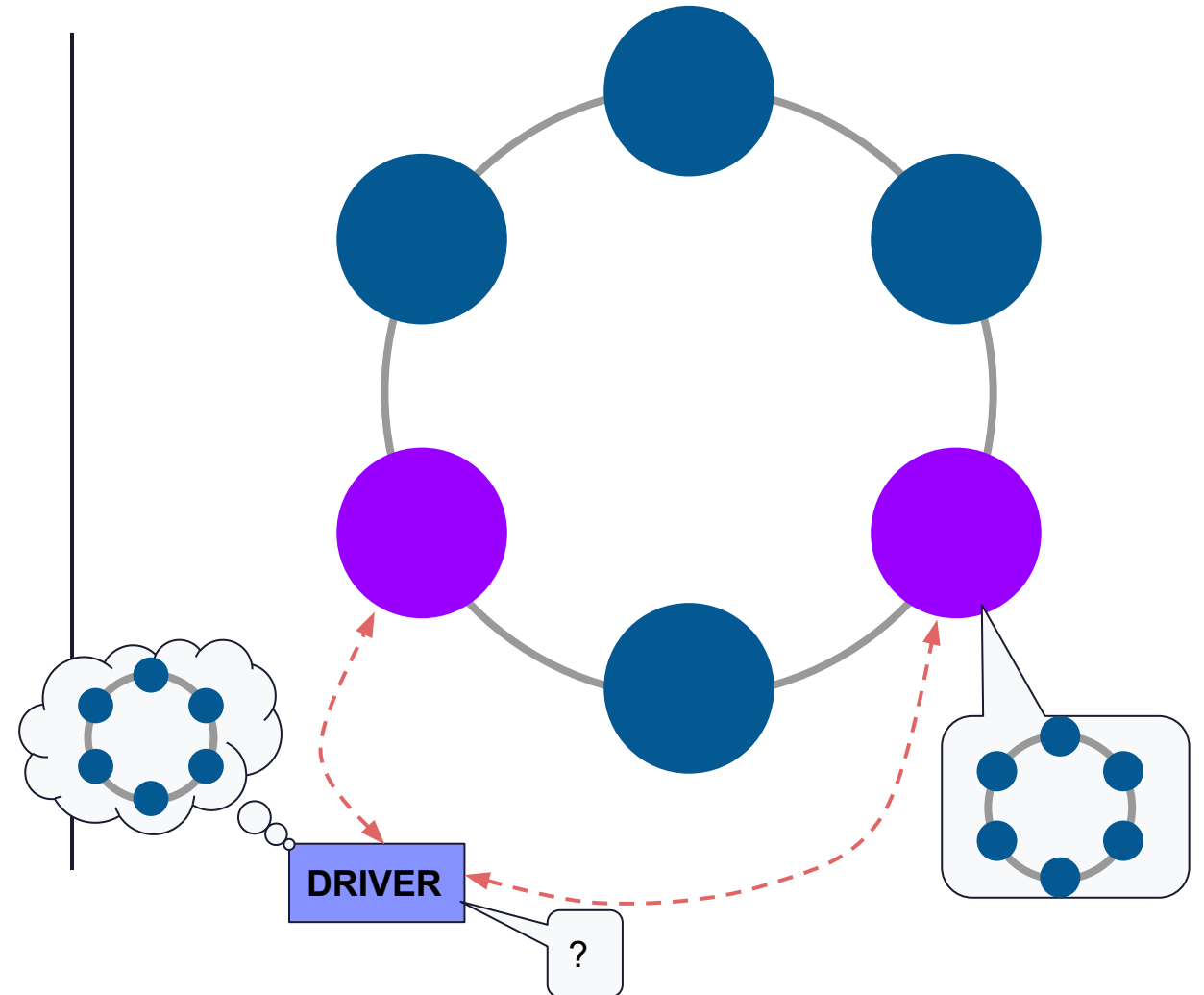Same drivers for: Cassandra, DSE, Astra DB

```
npm install cassandra-driver
```

```
{
  "dependencies": {
   "cassandra-driver": "^4.6.3"
  }
}
```

Connecting

Contact Points

- Only one necessary

- Unless that node is down

- Better ~3 nodes for resilience

- From there, drivers discover whole cluster

DRIVER

?

DataStax

Connecting

Create a `Client` with connection parameters, then call `.connect()`

The `client` should be a singleton; shut it down to release resources

```javascript
const { Client } = require("cassandra-driver");

const config = {
    contactPoints: ['12.34.56.78', '78.56.34.12'],
    localDataCenter: 'datacenter1',
    keyspace: 'chemistry'
};
const client = new Client(config);
await client.connect();
// ...
await client.shutdown();
```

Connecting

If you have an Astra DB instance, just change `config`:

```javascript
const { Client } = require('cassandra-driver');
const config = {
  cloud: {
    secureConnectBundle: process.env.SECURE_CONNECT_BUNDLE
  },
  credentials: {
    username: process.env.ASTRA_DB_CLIENT_ID,
    password: process.env.ASTRA_DB_CLIENT_SECRET
  },
  keyspace: 'chemistry'
};
const client = new Client(config);
await client.connect();
// ...
await client.shutdown();
```

DataStax

Mini-example

Connect, read rows and print them

```javascript
const { Client } = require("cassandra-driver");
const config = {
    contactPoints: ['12.34.56.78', '78.56.34.12'],
    localDataCenter: 'datacenter1',
    keyspace: 'chemistry'
};
const client = new Client(config);

async function main() {
  await client.connect();
  const query = 'SELECT symbol, name FROM elements; ';
  client.execute(query
    ).then(result => {
    const rows = result.rows;
      rows.forEach( row => {
        console.log('Symbol for %s is %s', row['name'], row['symbol']);
    });
    client.shutdown().then( () => console.log('** Client closed'));
    });
}

main();
```

LIVE DEMO

DataStax

Connectivity, options

Load balancing, retry policy, reconnection policies (you can also write your own)

```javascript
const { Client, policies } = require('cassandra-driver');
const config = {
    contactPoints: ['12.34.56.78', '78.56.34.12'],
    localDataCenter: 'datacenter1',
    keyspace: 'chemistry'
};

const client = new Client({
  ...config,
  ...{
    'policies': {
      'loadBalancing': new policies.loadBalancing.TokenAwarePolicy(
        new policies.loadBalancing.DCAwareRoundRobinPolicy('datacenter1')
      ),
      'reconnection': new policies.reconnection. ExponentialReconnectionPolicy(
        2000,             // base value
        10 * 60 * 1000,   // max value
        false,            // no, don't "start with no delay"
      ),
      'retry': new policies.retry.FallthroughRetryPolicy()
    }
  }
});
```

DataStax

Querying

Running a query

CQL statements are passed to the client's `execute()` method

```
const creationCommand = `CREATE TABLE IF NOT EXISTS metals
  (kind TEXT,
  name TEXT,
  density FLOAT,
  PRIMARY KEY ((kind), name));`;

await client.execute(creationCommand);
```

LIVE DEMO

Notable features:

- *concurrent execution primitives*
- *speculative query execution*

- **Promise-based API**
- **Callback-based API**

DataStax

Write queries – 1

A single CQL command string with everything in it

```
await client.execute(`INSERT INTO metals (kind, name, density)
    VALUES ('regular', 'copper', 0.01);`);
```

We can do better (trouble with: serialization, escaping, injections)

LIVE DEMO

Write queries – 2

Command / parameters, bound to one another

```
await client.execute(
  "INSERT INTO metals (kind, name, density) VALUES (?, ?, ?);",
  ['regular', 'palladium', 12.02],
  {
    hints: [
      'text',
      'text',
      'float'
    ]
  }
);
```

LIVE DEMO

Must provide hints for types to be properly serialized (expect errors otherwise)

DataStax

Write queries – 3

Enter **prepared statements**: schema info comes from server

```javascript
const metalInsertionStatement = ` INSERT INTO metals (kind, name, density)
    VALUES (?, ?, ?); `;

await client.execute(
  metalInsertionStatement,
  ['regular', 'zinc', 7.14],
  {prepare: true}
);

await client.execute(
  metalInsertionStatement,
  {
    kind: 'regular',
    name: 'mercury',
    density: 13.534
  },
  {prepare: true}
);
```

LIVE DEMO

mapping to Cassandra types

statements are on nodes, waiting to be re-used

DataStax

Write queries – 4

Promise-based insertion

```javascript
const metalInsertionStatement = `INSERT INTO metals
    (kind, name, density) VALUES (?, ?, ?);`;

client.execute(
  metalInsertionStatement,
  ['band!', 'AC/DC', 9999.99],
  {prepare: true}
).then(
  r => console.log('AC/DC inserted!')
).catch(
  err => console.error('Did something go wrong?', err)
);
```

LIVE DEMO

Write queries – 5

Callback-based insertion

```javascript
const metalInsertionStatement = `INSERT INTO metals
    (kind, name, density) VALUES (?, ?, ?);`;

client.execute(
  metalInsertionStatement,
  ['band!', 'Iron Maiden', 9999.99],
  {prepare: true},
  (err, res) => {
    if (! err){
      console.log('Iron Maiden inserted!');
    }else{
      console.error('ERROR inserting Iron Maiden:', err);
    }
  }
)
```

LIVE DEMO

DataStax

Reading queries – 1

Promise-based (parameters passed as for insertions)

```
client.execute(
   'SELECT name, density FROM metals WHERE kind = ?;',
   ['band!'],
   {prepare: true}
).then(result => {
   const rows = result.rows;
   rows.forEach( row => {
      console.log('Band %s has density %s', row['name'], row['density']);
   });
});
```

LIVE DEMO

DataStax

Reading queries – 2

Callback-based (just pass a fourth argument to **execute()**)

```
client.execute(
  'SELECT density FROM metals WHERE kind = ? AND name = ?;',
  ['regular', 'copper'],
  {prepare: true},
  (err, res) => {
    const row = res.first();
    console.log('Copper density = %s', row['density']);
  }
);
```

LIVE DEMO

DataStax

Reading queries – 3

Using the `eachRow()` method

```
client.eachRow(
  'SELECT name, density FROM metals WHERE kind = ?;',
  ['regular'],
  {prepare: true},
  (n, row) => {
    console.log('Metal %s is %s with density %s', n, row.name, row.density)
  },
  err => {
    // if err is null, it just means 'no more rows left'
    if (err) {
      console.error('Error reading regular metals:', err)
    } else {
      console.log('Finished reading.');
      // cleanup such as client.shutdown() ...
    }
  }
);
```

LIVE DEMO

DataStax

Batches

`client.batch()` – can be promise- or callback-based as well

```javascript
const densityUpdateStatement = 'UPDATE metals SET density = ? WHERE kind = ? AND name = ?;';
const batchForMetals = [
  {
    query: metalInsertionStatement,
    params: {
      kind: 'regular',
      name: 'silver',
      density: 10.49
    }
  },
  {
    query: densityUpdateStatement,
    params: [
      8.95,
      'regular',
      'copper'
    ]
  }
];
client.batch(batchForMetals, { prepare: true }
  ).then(function() {
    console.log('The whole batch succeeded');
  }).catch(function(err) {
    console.error('Batch has failed: ', err);
  });
```

LIVE DEMO

DataStax

Execution profiles – 1

Define several *execution profiles* when instantiating the client ...

```javascript
const { ExecutionProfile, types, policies } = require('cassandra-driver');

const defaultProfile = new ExecutionProfile('default', {
  readTimeout: 4000,
  consistency: types.consistencies.localQuorum
})
const pedanticProfile = new ExecutionProfile('pedantic', {
  consistency: types.consistencies.all,
  retry: new policies.retry. IdempotenceAwareRetryPolicy()
})
const localizedProfile = new ExecutionProfile('europe', {
  // let's pretend there's a datacenter we want to direct *some* queries to...
  loadBalancing: new policies.loadBalancing.DCAwareRoundRobinPolicy('overseas_dc')
})

const client = new Client({
  ...config,
  ...{
    profiles: [
      defaultProfile,
      pedanticProfile,
      localizedProfile
    ]
  }
});
```

DataStax

Execution profiles – 2

… and use one or the other in your mixed-workload queries

```javascript
const qry = 'SELECT density FROM metals WHERE kind = ? AND name = ?;';
const arg = ['regular', 'copper']

const row1 = (await client.execute(qry, arg,
  {
    executionProfile: 'pedantic'
  }
)).first();
console.log('Copper density = %s', row1['density']);

const row2 = (await client.execute(qry, arg,
  {
    executionProfile: 'europe'
  }
)).first();
console.log('Copper density = %s', row2['density']);

const row3 = (await client.execute(qry, arg)).first();     // will use the default profile
console.log('Copper density = %s', row3['density']);
```

Other query options

Consistency level, timeouts, page size, ...

```javascript
const { types, policies } = require('cassandra-driver');
client.eachRow(
  'SELECT name, density FROM metals WHERE kind = ?',
  ['regular'],
  {
    prepare: true,
    autoPage: true,    // this option relevant for eachRow() only
    consistency: types.consistencies.localQuorum,
    fetchSize: 2,
    readTimeout: 1500, // milliseconds
    policies: {
      retry: new policies.retry.FallthroughRetryPolicy(),
    }
  },
  (n, row) => {
    // 'n' will be an in-page index!
    console.log('Metal %s is %s with density %s', n, row.name, row.density)
  },
  err => {
    if(err){
      console.error('Error reading regular metals:', err)
    }
  }
);
```

A mini-API

A mini-API

```javascript
const express = require('express');
const api = express();
api.use(express.json());
api.listen(5000, () => {
  console.log('API ready on port 5000');
});

const { Client } = require('cassandra-driver');
const config = ... // FILL ME!
const client = new Client(config);

api.get('/metal', (req, res) => {
  // get all metals
  client.execute(
    'SELECT name, density FROM metals WHERE kind = ?',
    ['regular'],
    {prepare: true}
  ).then( qres => {
    const rows = qres.rows;
    res.send(rows);
  })
});
```

```javascript
api.get('/metal/:name', (req, res) => {
  // get a specific metal
  client.execute(
    'SELECT name, density FROM metals WHERE kind = ? AND name = ?',
    ['regular', req.params.name],
    {prepare: true}
  ).then( qres => {
    if (qres.rowLength > 0){
      const row = qres.first();
      res.send(row);
    } else {
      res.status(404).send('Not found');
    }
  });
});

api.post('/metal', (req, res) => {
  // upsert a metal
  const newMetal = {
    ...req.body,
    ...{kind: 'regular'}
  };
  console.log(newMetal);
  client.execute(
    'INSERT INTO metals (kind, name, density) VALUES (?, ?, ?)',
    newMetal,
    {prepare: true}
  ).then( () => {
    res.send({inserted: true});
  })
});
```

LIVE DEMO

```
curl -s localhost:5000/metal | jq

curl -s localhost:5000/metal/silver | jq

curl -s -XPOST localhost:5000/metal \
  --data '{"density": 101.11, "name": "armonium"}' \
  -H "Content-Type: application/json" | jq

curl -s localhost:5000/metal | jq
```

DataStax

Pagination

Pagination

Read queries return the first "page" of results (default `fetchSize = 5000`)

Automatic paging available (with *async iterator*):

```
const qry = 'SELECT name, density FROM metals WHERE kind = ?;';
const arg = ['regular'];
const fetchSize = 2;
const qOpts =  {
  prepare: true,
  fetchSize: fetchSize
};

const result = await client.execute(qry, arg, qOpts);
for await (const row of result) {
  console.log('Metal %s has density %s', row.name, row.density)
}
```

LIVE DEMO

But sometimes manual paging preferrable ...

Pagination – a second way

Passing `autoPage:  true` to `eachRow()`

```javascript
const qry = 'SELECT name, density FROM metals WHERE kind = ?';
const arg = ['regular'];
const fetchSize = 2;
const qOpts =    {
  prepare: true,
  fetchSize: fetchSize
};
client.eachRow(
  qry,
  arg,
  {
    ...{autoPage: true},
    ...qOpts
  },
  (n, row) => {
    // 'n' will be an **in-page** index
    console.log('Metal %s is %s with density %s', n, row.name, row.density)
  },
  err => {
    if(err){
      console.error('Error reading regular metals:', err)
    }
  }
);
```

LIVE DEMO

DataStax

Manual pagination – 1

Passing an explicit `pageState` (string) to next call

```javascript
const qry = 'SELECT name, density FROM metals WHERE kind = ?;  ';
const arg = ['regular'];
const fetchSize = 2;
const qOpts = {
  prepare: true,
  fetchSize: fetchSize
};


let resSet = await client.execute(qry, arg, qOpts);
console.log('first page:');
for(var row of resSet.rows){
  console.log('Metal %s has density %s', row.name, row.density)
}
while(resSet.pageState){
  console.log('another page:');
  resSet = await client.execute(qry, arg, {...qOpts, ...{pageState: resSet.pageState}});
  for(var row of resSet.rows){
    console.log('Metal %s has density %s', row.name, row.density)
  }
}
```

LIVE DEMO

Manual pagination – 2

Invoking `nextPage()` within `eachRow()`

```
const qry = 'SELECT name, density FROM metals WHERE kind = ?;';
const arg = ['regular'];
const fetchSize = 2;
const qOpts =          {
  prepare: true,
  fetchSize: fetchSize
};
client.eachRow(
  qry,
  arg,
  qOpts,
  function (n, row) {
    // Invoked per each row in all the pages
    console.log('Metal %s has density %s', row.name, row.density)
  },
  function (err, result) {
    if (typeof result !== undefined) {
      pageState = result.pageState;
      console.log("   [found pageState: ", pageState, "]");
      if(pageState !== null) {
        console.log("    [Requesting next page]");
        result.nextPage();
      } else {
        console.log("    [End of results]");
      }
    } else {
      // some error to handle here
    }
  });
```

LIVE DEMO

DataStax

Manual pagination – 3

Using `client.stream()` to avoid clogging the local buffer

```
const qry = 'SELECT name, density FROM metals WHERE kind = ?';
const arg = ['regular'];
const fetchSize = 2;
const qOpts =        {
  prepare: true,
  fetchSize: fetchSize
};


client.stream(qry, arg, qOpts
  ).on('readable', function () {
    console.log("    [Readable rows]");
    // there are emitted rows to be read
    var row;
    while (row = this.read()) {
      console.log('Metal %s has density %s', row.name, row.density)
    }
}).on('end', function () {
    // no more rows to be consumed in the stream
    console.log("    [End of results]");
});
```

LIVE DEMO

Object mapper

Auto-discovery of schema from nodes; can handle denormalization (e.g. as multiple inserts)

```javascript
const { mapping } = require('cassandra-driver');

const mapper = new mapping.Mapper(client, {
  models: {
    'Element': { tables: ['elements'] },
  }
});
const elementMapper = mapper.forModel('Element');

const newElement = {name: 'Philosophium', symbol: 'Ph',
  atomic_mass: 12.3, atomic_number: 120};
await elementMapper.insert(newElement);
console.log('New element inserted.');

const myElement = await elementMapper.get({symbol: 'Ph'});
console.log('Element name = %s', myElement.name);

const symbol = 'Ph';
await elementMapper.remove({ symbol });
console.log('New element deleted.');
```
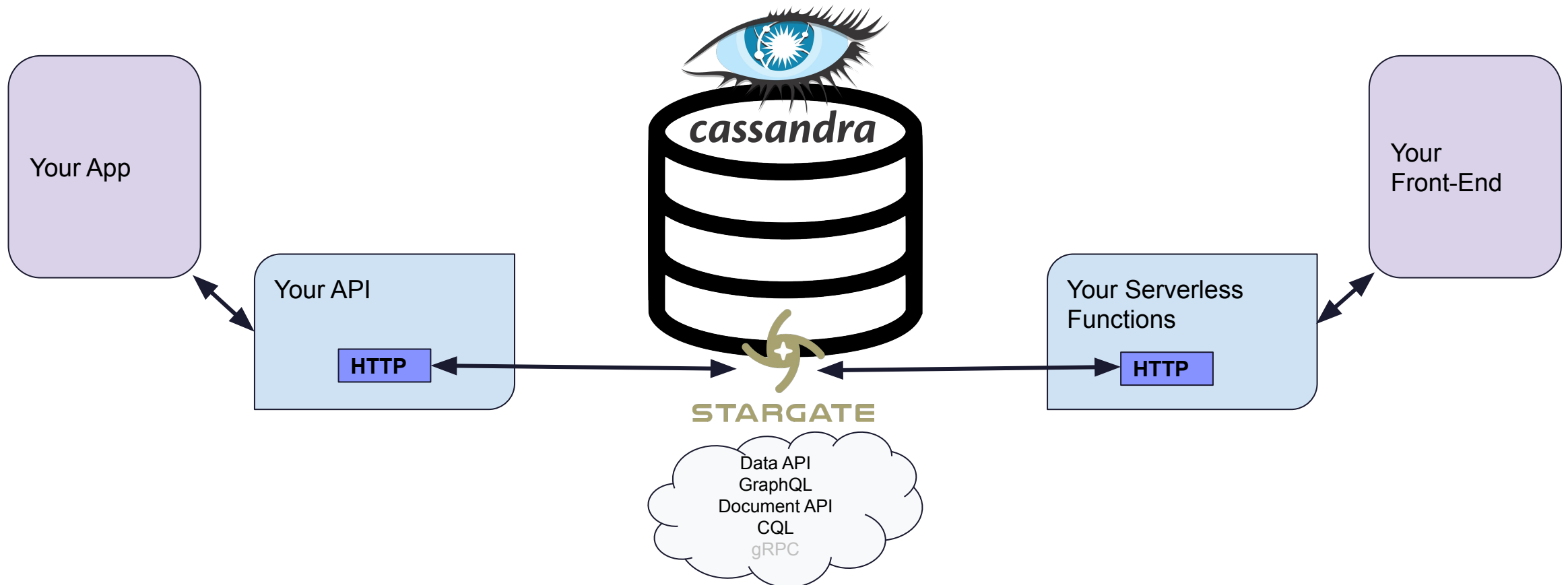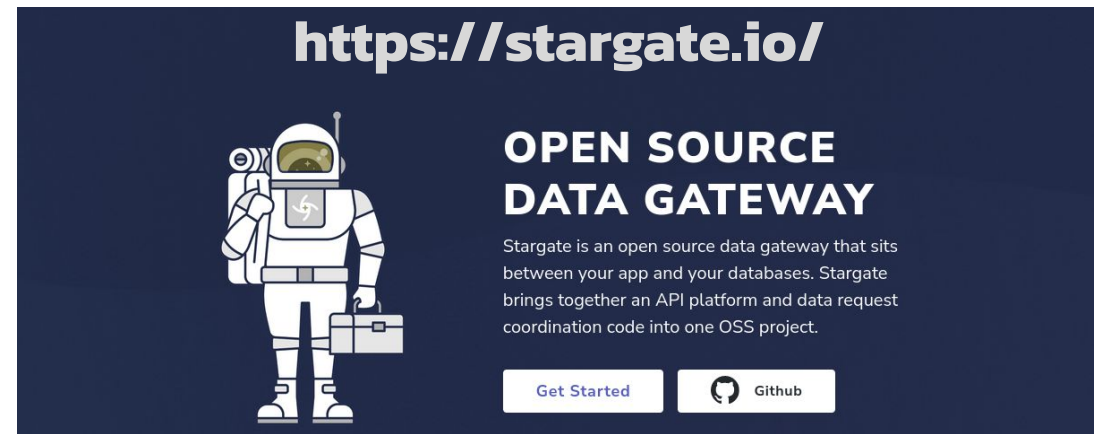
LIVE DEMO

Stargate: an alternative to drivers

cassandra

Your App

Your API

HTTP

STARGATE

Your Serverless
Functions

HTTP

Your
Front-End

Data API
GraphQL
Document API
CQL
gRPC

DataStax

Stargate: "just HTTP requests"

Document API

GraphQL

REST

…

```javascript
const { createClient } = require("@astrajs/collections");
// 
const
// curl --request PATCH \
// --u                                          wicklow.com/blog" \
const    -H
// c      -H       const fetch = require('node-fetch')
const    -H
// t      -d       exports.handler = async function (event) {
const               const body = JSON.parse(event.body)
       na              const genre = body.genre
       gi              const pageState = body.pageState
});       "r             const url = process.env.ASTRA_GRAPHQL_ENDPOINT
// 9                    const query = `
con                    query {
// c      }              movies_by_genre (
con      }'               value: { genre: ${JSON.stringify(genre)}},
       na                  orderBy: [year_DESC],
       group: "u            options: { pageSize: 6, pageState: ${JSON.stringify(pageState)} }
});                        ) {
// now user              values {
// create a                year,
const userS                title,
  title: "t                duration,
  url: "htt                synopsis,
  metrics:                 thumbnail
    posts:                 }
    rating:              pageState
  }                    }
}                    }
});                    `
// now user
// get a su           const response = await fetch(url, {
const userE            method: 'POST',
// now user            headers: {
// search a              "Content-Type": "application/json",
const users             "x-cassandra-token": process.env.ASTRA_DB_APPLICATION_TOKEN
                       },
                       body: JSON.stringify({ query })
                     })
```

DataStax

Conclusion

Resources for today's practice

CODE SAMPLE　　github.com/hemidactylus/cassandra-nodejs-drivers-practice

ASTRA DB　　astra.datastax.com

STARGATE　　stargate.io

DataStax

Stargate-related references

WORKSHOPS

www.datastax.com/workshops
Past workshops at: www.youtube.com/datastaxdevs

SELF-SERVICE

github.com/datastaxdevs
Self-contained material from workshops

DataStax

Resources for developers

LEARN

academy.datastax.com
Free online courses – Cassandra certifications

ASK/SHARE

community.datastax.com
Ask/answer community user questions – share your expertise

CONNECT

Follow us @DataStaxDevs
We are on YouTube – Twitter – Twitch!

CONTACT

dtsx.io/discord
Discord Community Server

DataStax

Now go and create your wonderful Cassandra-backed application!

DataStax

Thank You!

DataStax