DataStax

Developers

Connecting to Astra DB in IOS app

Building efficient IOS applications in Swift that connect to Astra

01

Intro – What are we doing?

02

Prerequisites

03

Core Workshop – Connecting to Astra in Swift

04

Completed App – Demo

05

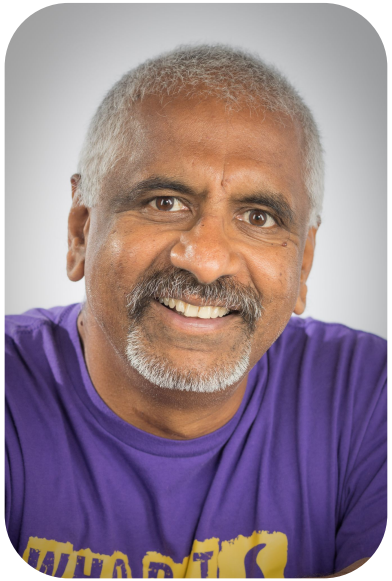
Takeaways

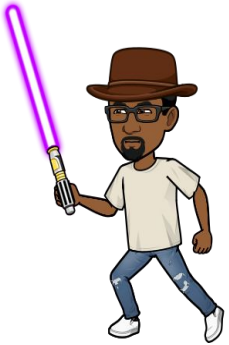
06

SWAG

Agenda

Developer Advocate





@rags

@ragsns

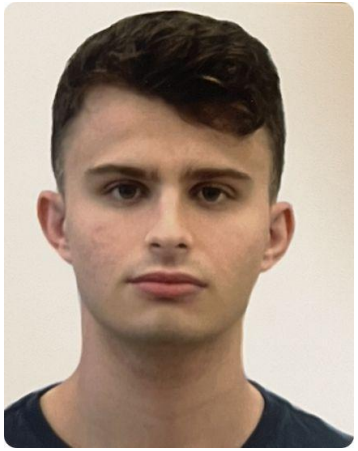
@ragss

- Developer/Architect
- Mechanical Engineer (so many moons ago)
- Distributed systems
- Love to teach and communicate
- Inner loop == developer productivity

Raghavan “Rags” Srinivas

@victor-micha

@vmic2002

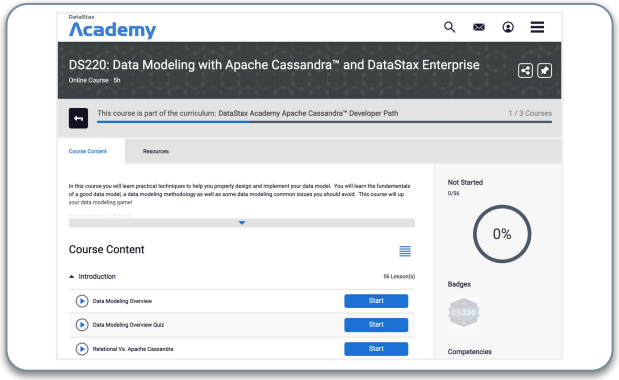


- Developer Relations * Product Marketing Intern
- Entering 3rd year Major in Software Engineering at McGill University

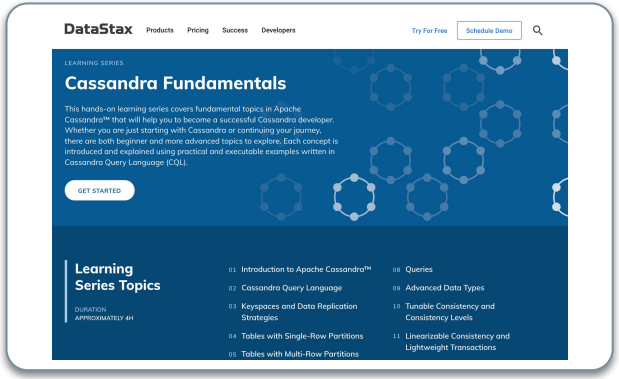
Victor Micha

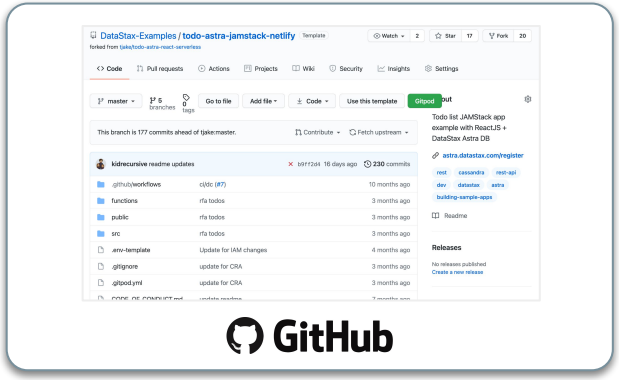
DataStax Developers Crew

Academy academy.datastax.com



Datastax /dev datastax.com/dev





01

Intro - What are we doing?

02

Prerequisites

03

Core Workshop - Connecting to Astra in Swift

04

Completed App - Demo

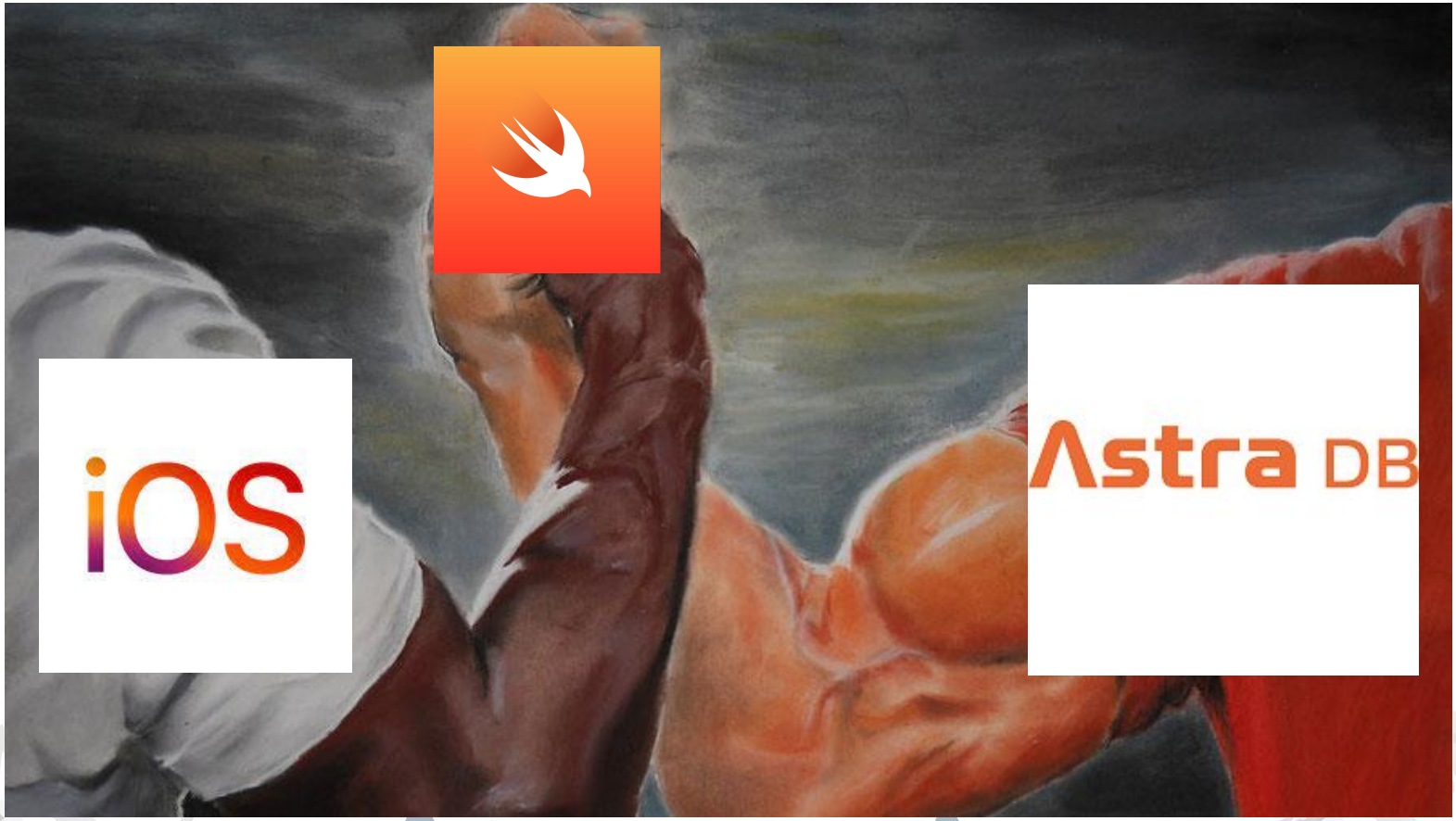
05

Takeaways

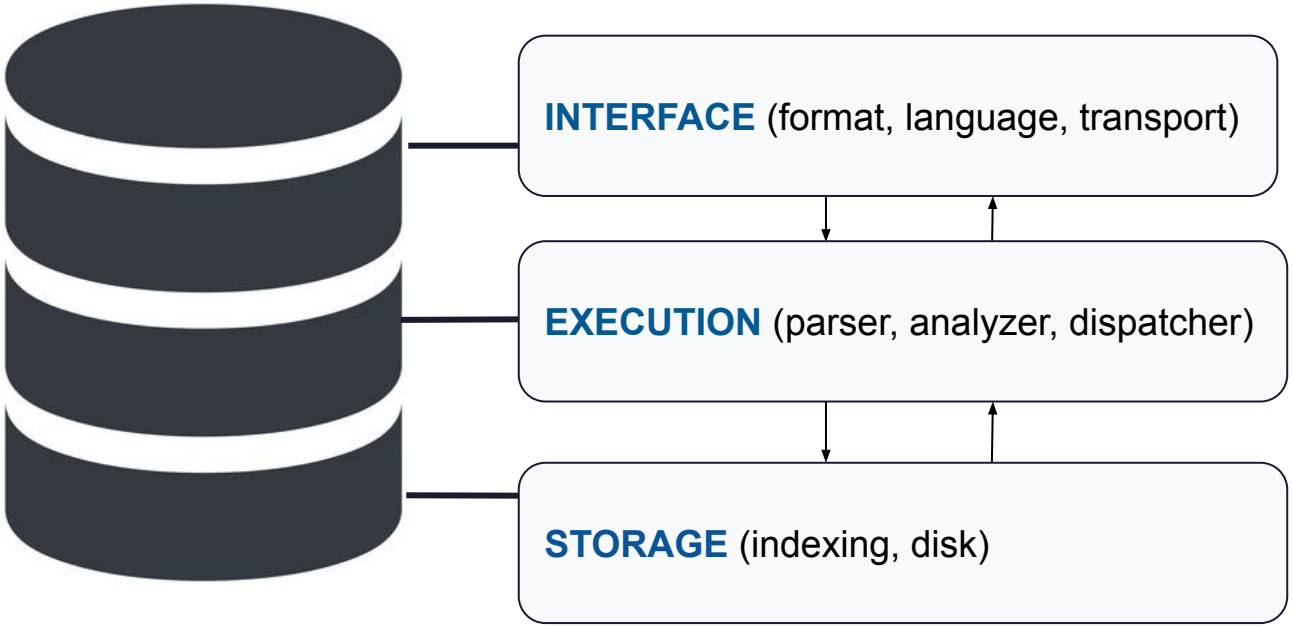
06

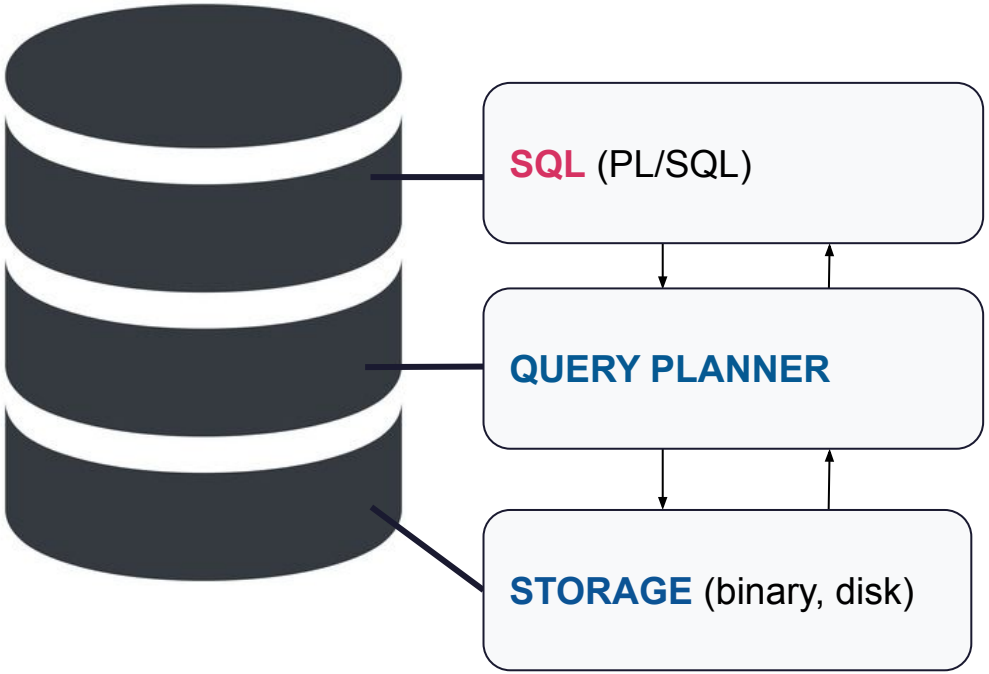
SWAG

Agenda



Objective


INTERFACE (format, language, transport)
EXECUTION (parser, analyzer, dispatcher)
STORAGE (indexing, disk)

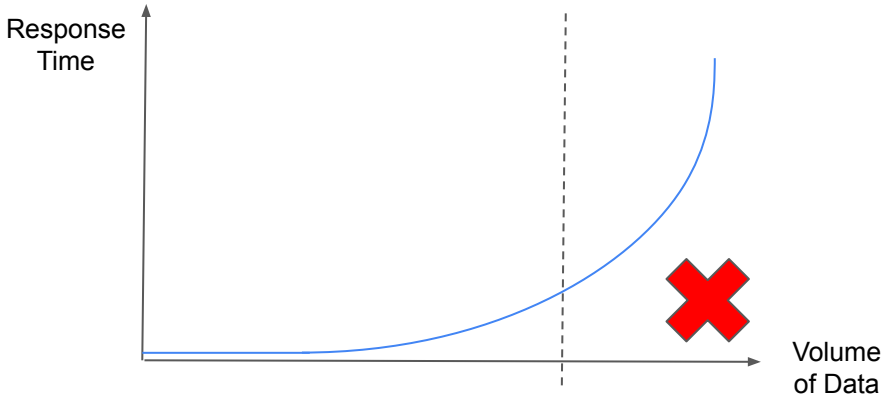

SQL (PL/SQL)
QUERY PLANNER
STORAGE (binary, disk)

OLTP: Online Transaction Processing

Fast Processing
Transactional
High number of transactions
Hot / Live Data
“Row oriented”
Normalized Data (3NF)

OLAP: Online Analytical Processing

Slow Queries
Historical
High volume of data
Cold Data
“Column oriented”
Denormalized Data



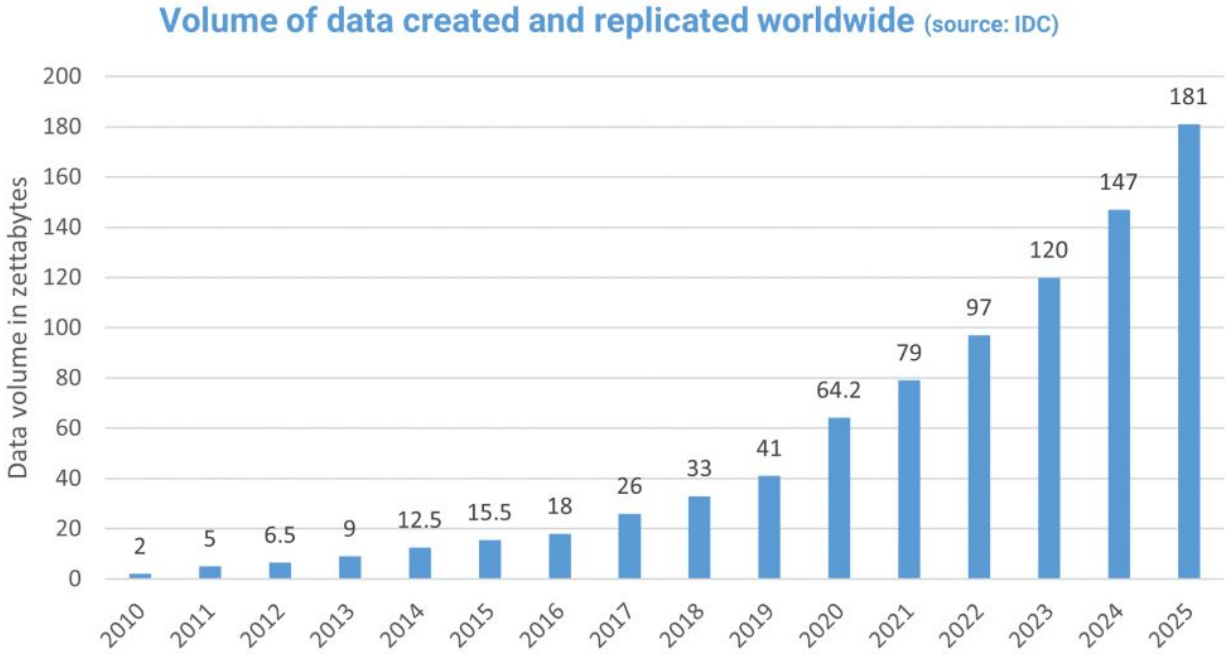
Relational Databases are versatile

- They have been designed to run on a single machine

- New requirements
  - V: VOLUME
  - V: VELOCITY
  - V: VARIETY

WHAT'S A ZETTABYTE?	
1 kilobyte	1,000
1 megabyte	1,000,000
1 gigabyte	1,000,000,000
1 terabyte	1,000,000,000,000
1 petabyte	1,000,000,000,000,000
1 exabyte	1,000,000,000,000,000,000
1 zettabyte	1,000,000,000,000,000,000,000

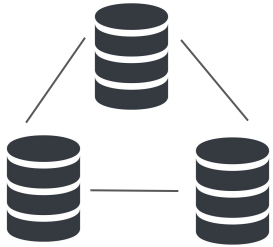
SOURCES: CISCO



Relation Databases have limited scalability

- Meetup names on June 11, 2009 in San Francisco

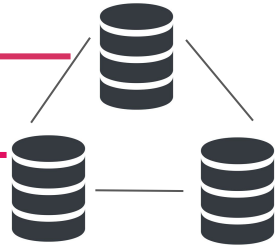
- Web Giants to needed more scalable systems
  - Distributed by design (horizontal scalability)
  - Data is replicated
  - Cope with the variety of data



Not Only SQL

AP vs CP (CA = not distributed)

Server vs **VM** vs **Container** vs **Cloud**



SQL or **JSON** or **CQL** or **N1QL** or **Cypher...**

Query Parser and execution engine

Text, Binary, Graph

Ledger Databases *awsqldb, ksql_db*	Time-Series Databases *Influx, OpenTSDB, Prometheus*	Tabular Databases ***Cassandra**, Hbase, Bigtable*	Object Databases *S3, Minio, Ceph*
Document Databases *MongoDB, Elastic, DocDb*	Graph Databases *Neo4j, Datastax Graph, Titan*	Key/Value Databases *Redis, Dynamo, Memcache*	

DataStax

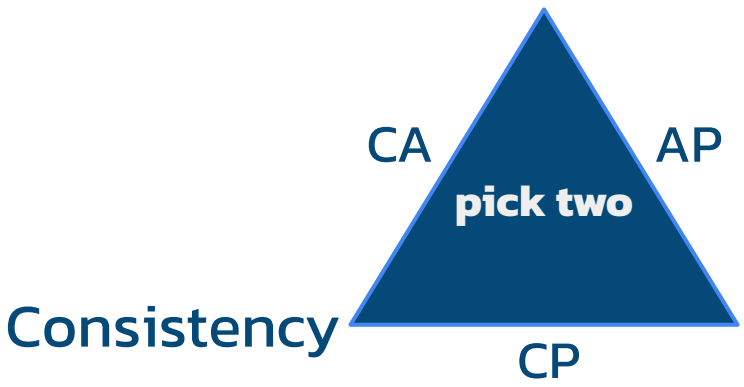
sky

NoSQL Databases

Origin of the term “NoSQL”

- Meetup name on June 11, 2009 in San Francisco
  - Catchy hashtag intended to refer to databases like BigTable and DynamoDB
  - Meetup presentations: Cassandra, MongoDB, CouchDB, HBase, Voldemort, Dynomite, and Hypertable
- Sometimes referred to “Not only SQL”

The CAP Theorem



Astra DB

cassandra

DynamoDB

riak

redis

BigTable

APACHE HBASE

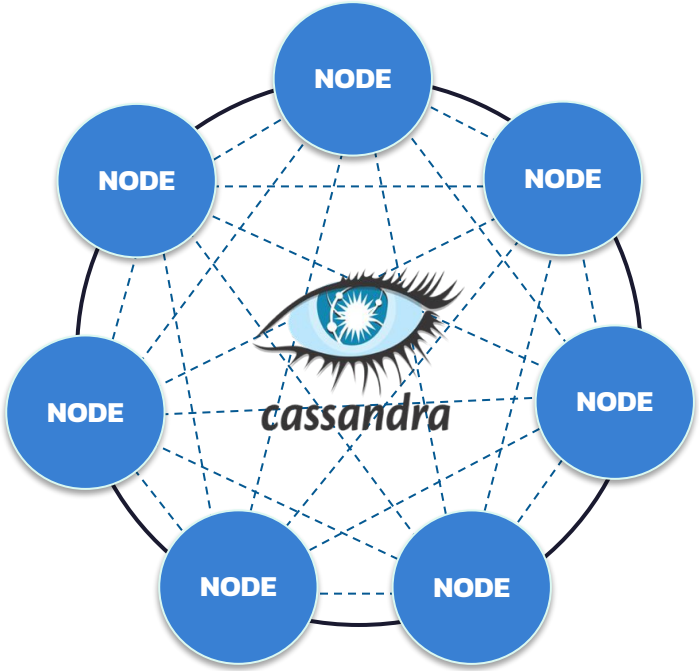
mongoDB.

Couchbase

Apache Cassandra

NoSQL Distributed Decentralised Database Management System

- Big Data Ready
- Read / Write Performance
- Linear Scalability
- Highest Availability
- Self-Healing and Automation
- Geographical Distribution
- Platform Agnostic
- Vendor Independent



1. **NO** Single Point of Failure
2. Scales for writes and reads
3. Application can contact any node
   (in case of failure - just contact next one)

Master-less (Peer-to-Peer) Architecture

The CAP Theorem

Astra DB

cassandra

DynamoDB

riak

redis

BigTable

APACHE HBASE

mongoDB.

Couchbase

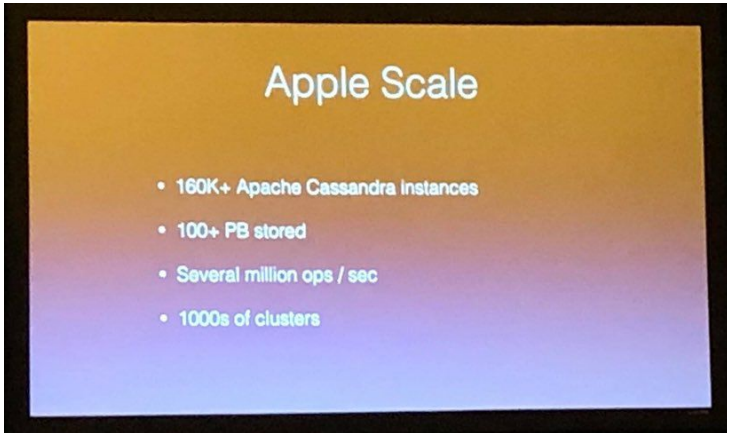
Apache Cassandra @ Netflix

- 98% of streaming data is stored in Apache Cassandra
- Data ranges from customer details to viewing history to billing and payments
- Foundational datastore for serving millions of operations per second

- 30 million ops/sec on most active single cluster
- 500 TB most dense single cluster
- 9216 CPUs in biggest cluster

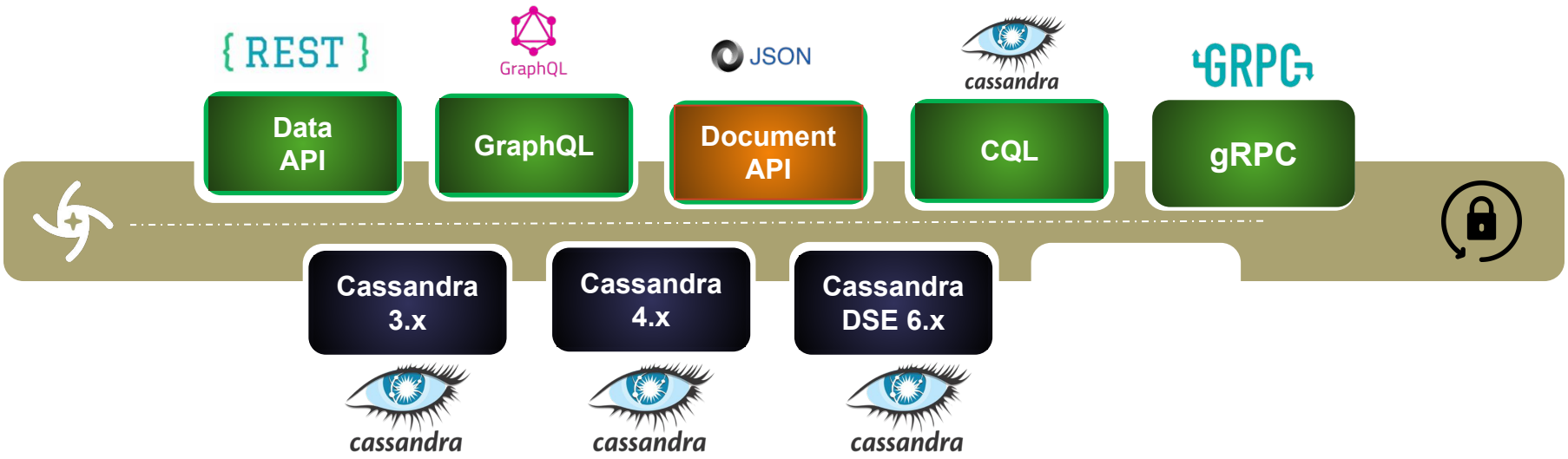
O(100)	Clusters
O(10000)	Instances
O(10,000,000)	Replications per second
O(100,000,000)	Operations per second
O(1,000,000,000,000,000)	Petabytes of data

dtsx.io/cassandra-at-netflix

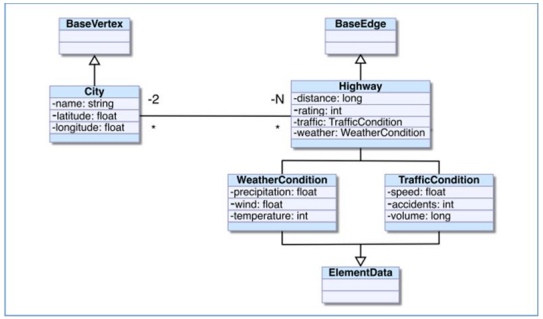


And many others...

Cassandra Biggest Users (and Developers)



About Stargate



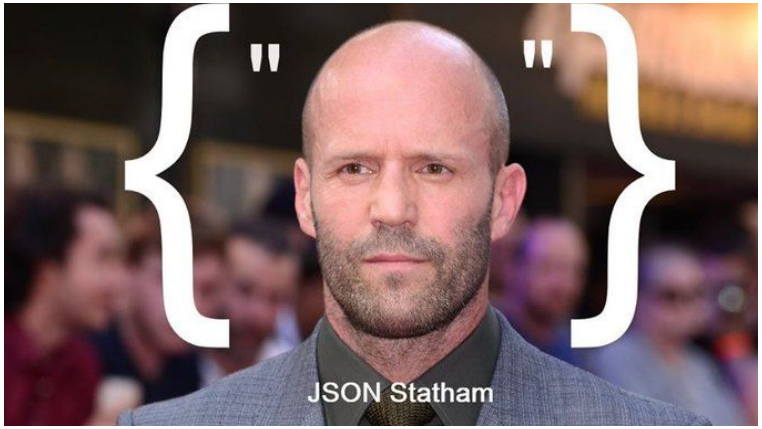
- Cassandra already handles both **JSON** and **nested objects**
  - INSERT JSON, SELECT JSON
  - Set<>, List<>,Map<>, and User Defined type (UDT) even nested

- But...
  - Strongly coupled with a SCHEMA VALIDATION
  - Dangerous with tombstones on updates

```
select json title,url,tags from videos;
```

```
INSERT INTO videos JSON '{
   "videoid":"e466f561-4ea4-4eb7-8dcc-126e0fbfd578",
     "email":"clunven@sample.com",
     "title":"A JSON videos",
     "upload":"2020-02-26 15:09:22 +00:00",
     "url": "http://google.fr",
     "frames": [1,2,3,4],
     "tags":   [ "cassandra","accelerate", "2020"]
}';
```

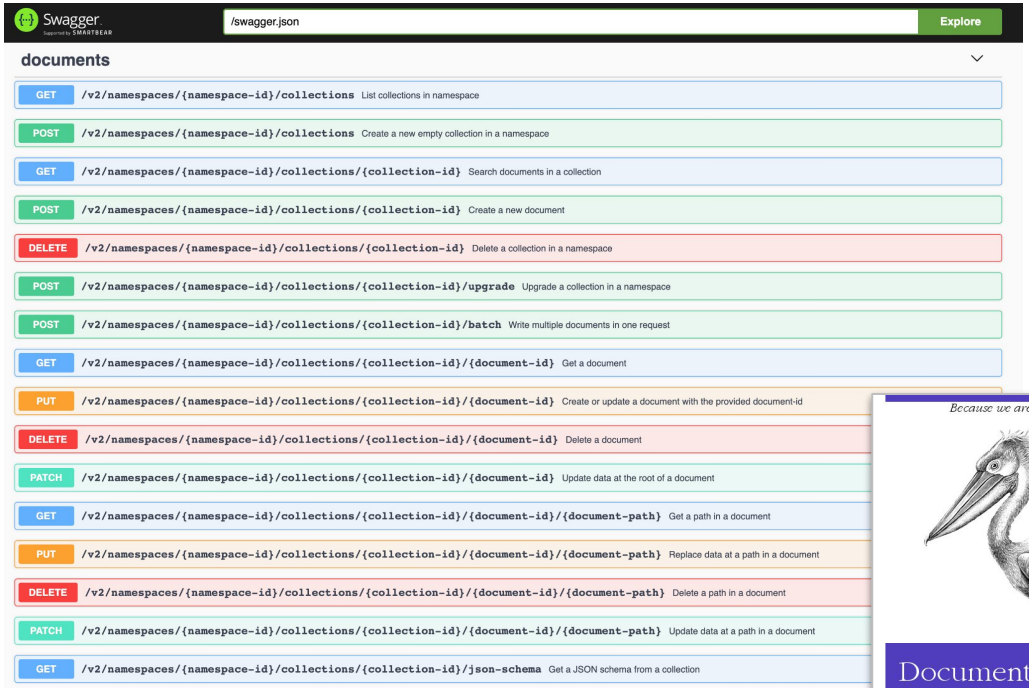
JSON && Cassandra ? Yes but....



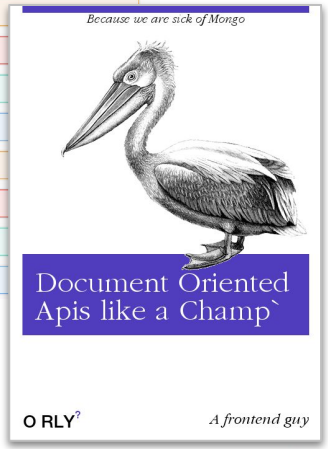
“What rules ?”

- 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

👊 Schemaless 👊



- Schemaless! (validationless)
- Add any JSON document to a collection
- Uses “Document shredding” approach

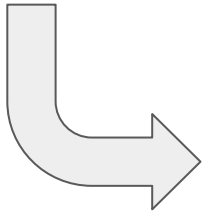


Stargate Document API

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

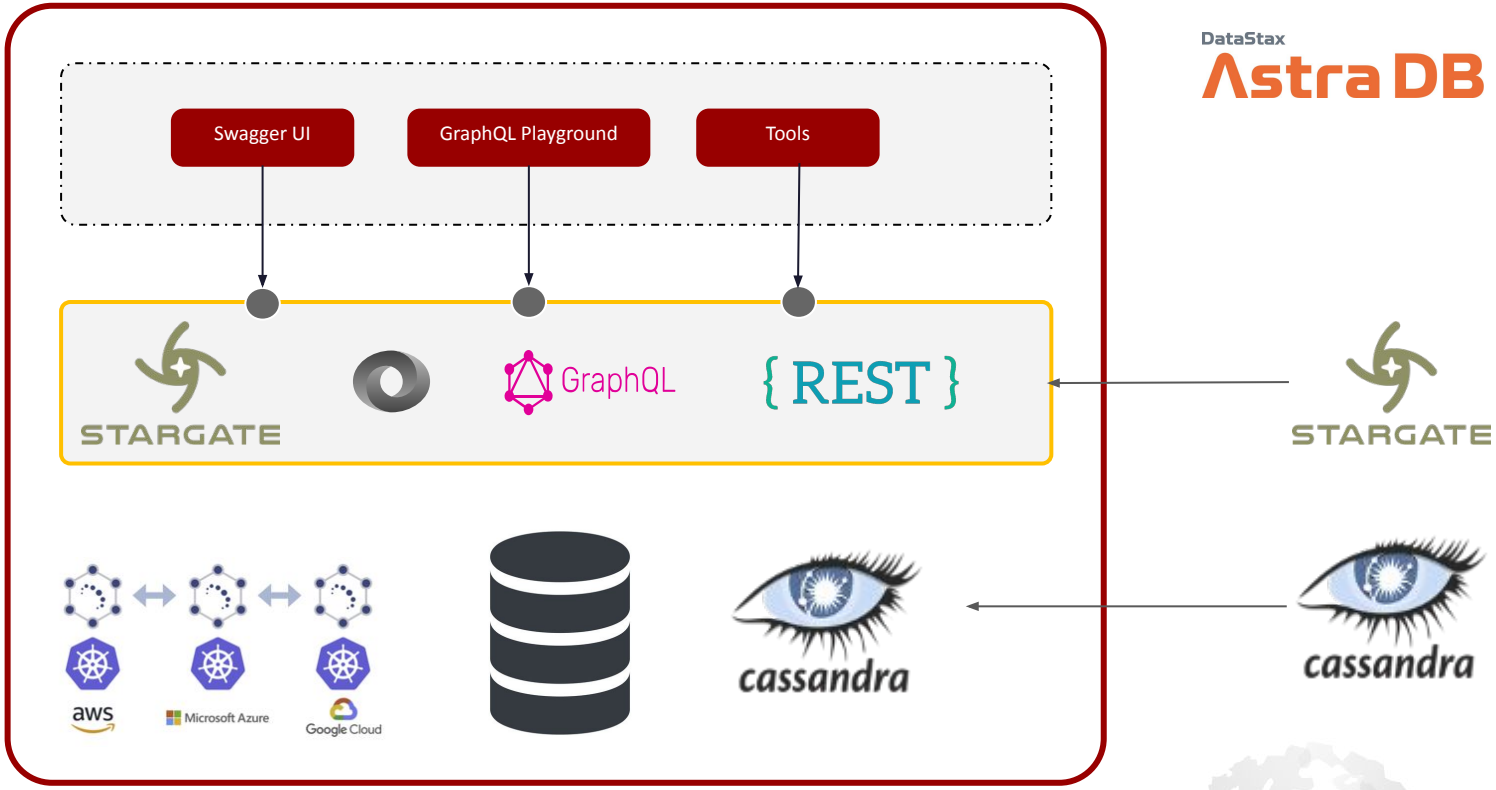
```
{"a": { "b": 1 }, "c": 2}
```

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

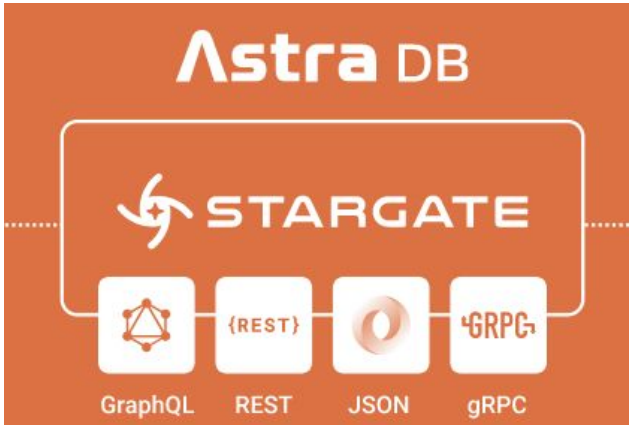


key	p0	p1	dbl_value
x	a	b	1
x	c	null	2

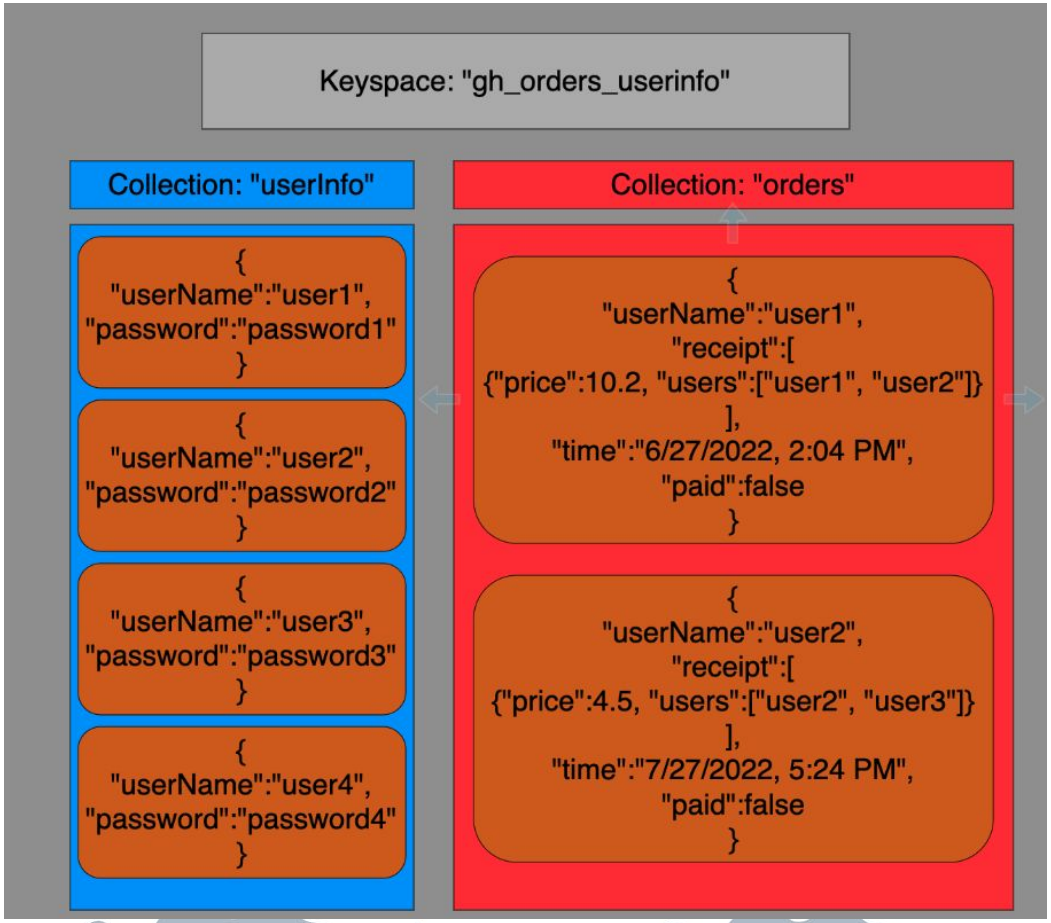
Document Shredding



Astra = Cassandra + Stargate in SaaS



About the app


Keyspace: "gh_orders_userinfo"
Collection: "userInfo"
Collection: "orders"
{
"userName":"user1",
"password":"password1"
}
{
"userName":"user2",
"password":"password2"
}
{
"userName":"user3",
"password":"password3"
}
{
"userName":"user4",
"password":"password4"
}
{
"userName":"user1",
"receipt":[
{"price":10.2, "users":["user1", "user2"]}
],
"time":"6/27/2022, 2:04 PM",
"paid":false
}
{
"userName":"user2",
"receipt":[
{"price":4.5, "users":["user2", "user3"]}
],
"time":"7/27/2022, 5:24 PM",
"paid":false
}

01

Intro – What are we doing?

02

Prerequisites

03

Core Workshop – Connecting to Astra in Swift

04

Completed App – Demo

05

Takeaways

06

SWAG

Agenda

Download Xcode and create Astra DB

For more info see -> “Prerequisites” in repo:
https://github.com/datastaxdevs/workshop-IOS-Swift-Astra#prerequisites

Create collections “userInfo” and “orders”

For more info see -> “Now you should create your own collection using swagger ui” in repo:
https://github.com/datastaxdevs/workshop-IOS-Swift-Astra#now-you-should-create-your-own-collection-using-swagger-ui

Intro to HTTP requests

For more info see -> “For beginners to databases” in repo:
https://github.com/datastaxdevs/workshop-IOS-Swift-Astra#for-beginners-to-databases

HTTP requests – cURL – Swift

01

Intro – What are we doing?

02

Prerequisites

03

Core Workshop – Connecting to Astra in Swift

04

Completed App – Demo

05

Takeaways

06

SWAG

Agenda

Using Swift in Xcode

```
struct UserInfo : Codable {
    let userName : String
    let password : String
}
let ASTRA_DB_ID = "REPLACE_ME"
let ASTRA_DB_TOKEN = "REPLACE_ME"
let ASTRA_DB_REGION = "REPLACE_ME"
let ASTRA_DB_KEYSPACENAME = "REPLACE_ME"
```

Add Model and Variables

```
func getURLRequest(httpMethod:String, endUrl:String)->URLRequest{
    //add code here
}
```

Create getURLRequest() function

To POST an Order/UserInfo:

1. Encode an instance of an Order/UserInfo struct into an instance of the Data struct using an instance of the JSONEncoder class
2. Create instance of URLRequest struct and set the HTTP request info with the appropriate information (URL, HTTP headers, HTTP method)
3. Make the HTTP request to the server using the URLSession class, taking two parameters: the instance of URLRequest struct and the Data to POST
4. Ensure that the response code indicates a successful POST

POST order/userInfo

To PATCH an Order/UserInfo:

- Same process as POST, except change HTTP method from POST to PATCH
- Encode a different struct which only encompasses the password, not the username

PATCH userInfo (change password)

To GET an Order/UserInfo:

1. Create instance of URLRequest struct and set the HTTP request info with the appropriate information (URL, HTTP headers, HTTP method)
2. Make the HTTP request to the server using the URLSession class, taking the instance of URLRequest struct as a parameter

-> A JSON string is returned: {"data":{"4890b95d-a27e-4259-9654-f524826d09ba":{"password":"pass","userName":"vmic"}}}

-> We need to process this string to make it look like this:

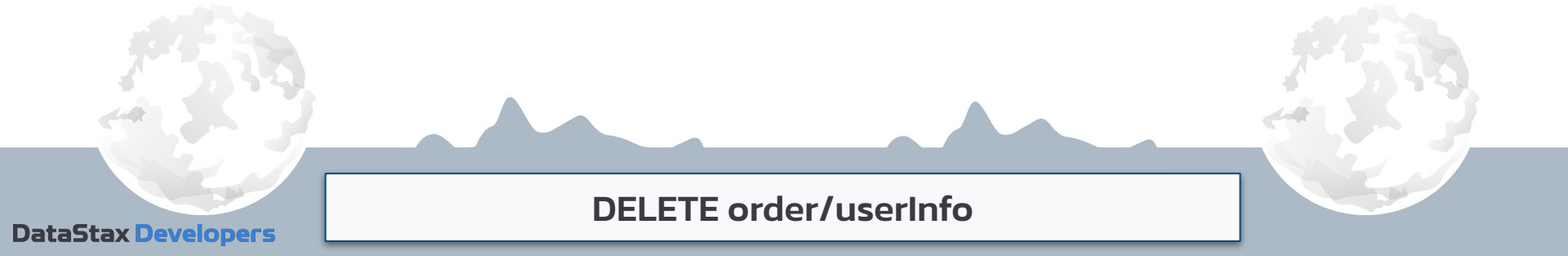
{"4890b95d-a27e-4259-9654-f524826d09ba":{"password":"pass","userName":"vmic"}}

-> Then convert it into a Dictionary or map where the keys are docIDs and the values are UserInfo/Order

"4890b95d-a27e-4259-9654-f524826d09ba" -> UserInfo(userName:”vmic”, password:”pass”)

To DELETE a doc using the Document API, we need the docID

1. Create instance of URLRequest struct and set the HTTP request info with the appropriate information (URL, HTTP headers, HTTP method)
2. Make the HTTP request to the server using the URLSession class, taking the instance of URLRequest struct as a parameter
3. Ensure that the response code indicates a successful DELETE



DELETE order/userInfo

To GET all docs:

Perform GET request

If there is no page-state{

    We are done

} else {

While there is a page-state {

    Perform GET request with page-state from previous GET request

    Update page-state

}

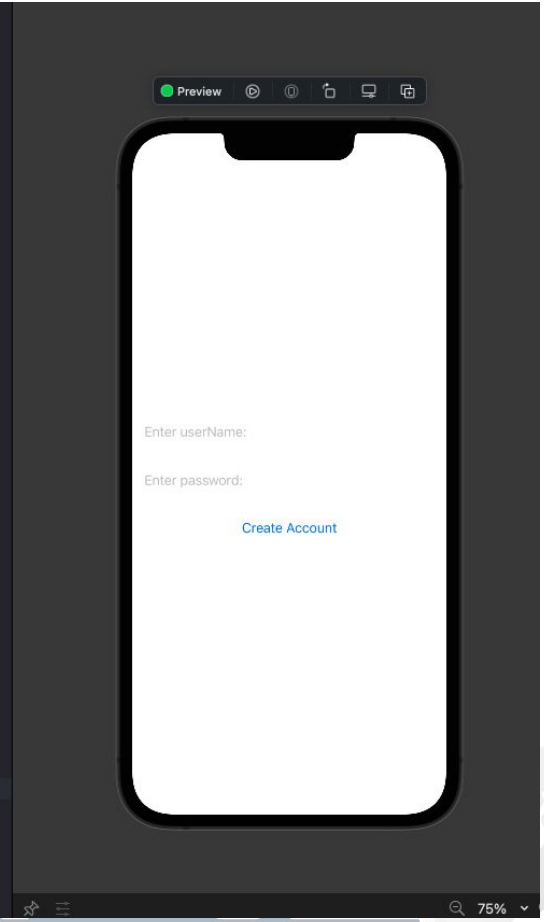
GET ALL orders

```swift
//
//  ContentView.swift
//  GroupAccounting
//
//  Created by Victor Micha on 8/5/22.
//

import SwiftUI

struct ContentView: View {
    @State private var userName=""
    @State private var password=""
    var body: some View {
        VStack{
            TextField("Enter userName:", text: $userName)
                .padding(.all, 15)
            TextField("Enter password:", text: $password)
                .padding(.all, 15)
            Button("Create Account"){
                if (userName.isEmpty||password.isEmpty){
                    print("Username and passowrd cannot be empty!")
                    return
                }
                Task{
                    let dict = try await getUserInfo(userName: userName)
                    if (dict.count==1){
                        print("Account already exists. Cannot create new one")
                        return
                    }
                    try await postUserInfo(userInfo: UserInfo(userName:userName,
                        password: password))
                }
            }.padding(.all, 15)
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```



Simple view to create account

01

Intro – What are we doing?

02

Prerequisites

03

Core Workshop – Connecting to Astra in Swift

04

Completed App – Demo

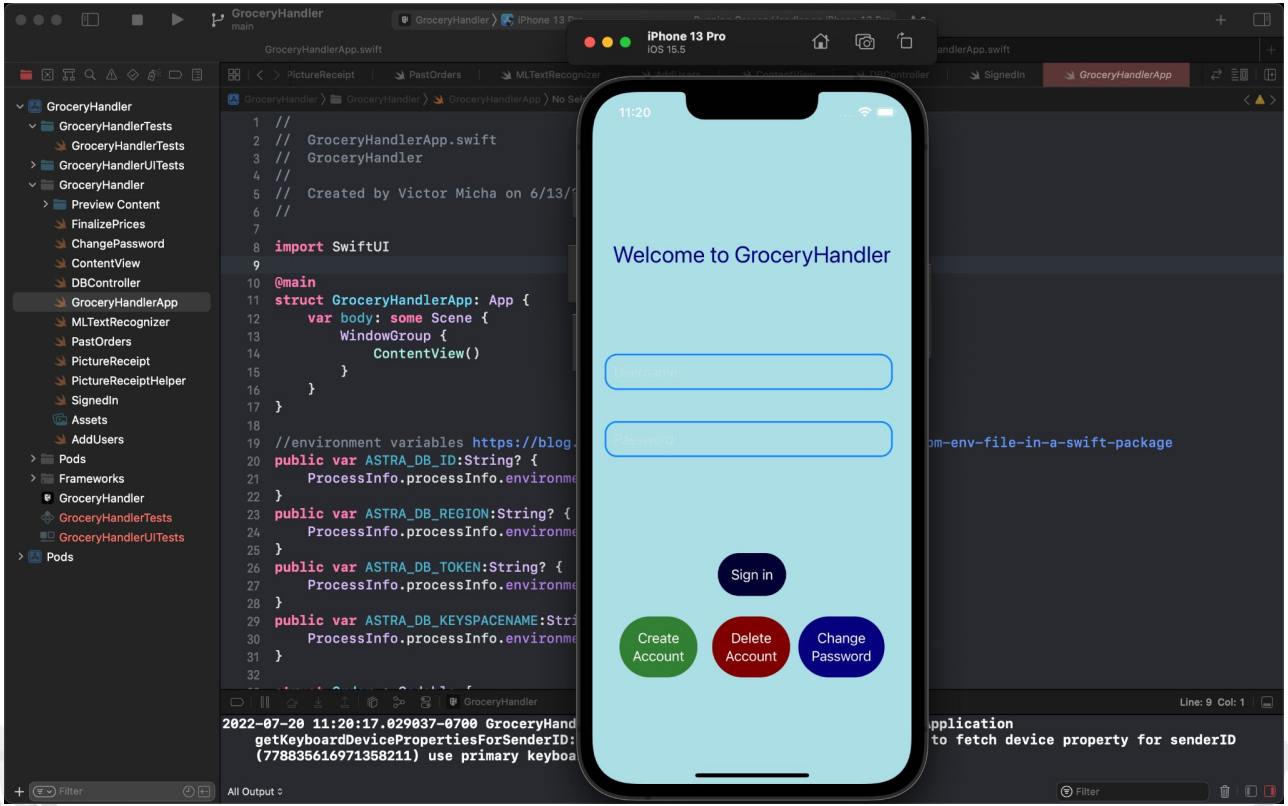
05

Takeaways

06

SWAG

Agenda



Demo of completed app

01

Intro - What are we doing?

02

Prerequisites

03

Core Workshop - Connecting to Astra in Swift

04

Completed App - Demo

05

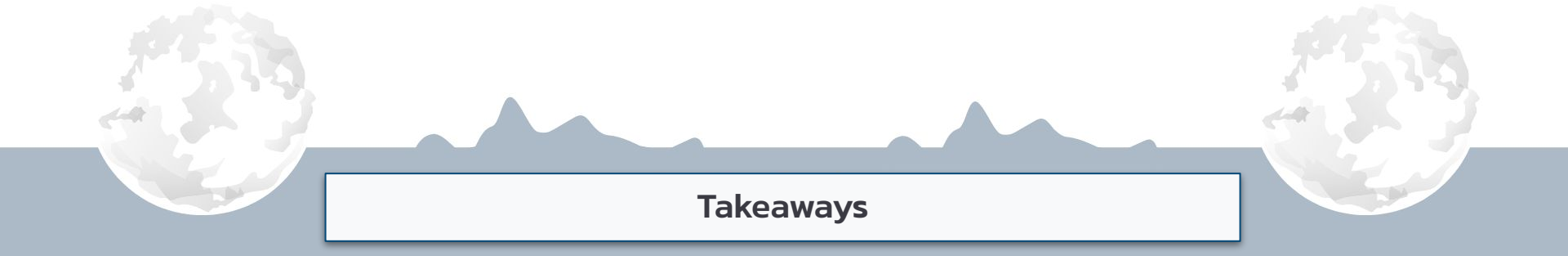
Takeaways

06

SWAG

Agenda

1. Many big companies such as Netflix, Apple, Instagram etc use Cassandra because of its reliability, efficiency, and ease of use
2. Performing HTTP requests in an IOS app to connect to Datastax Astra is Swift ;) and painless
3. Using Document API to connect to Astra DB is easy and allows a lot of the management of the database to be abstracted from the developer so he/she can focus on working on the app itself



Takeaways

01

Intro - What are we doing?

02

Prerequisites

03

Core Workshop - Connecting to Astra in Swift

04

Completed App - Demo

05

Takeaways

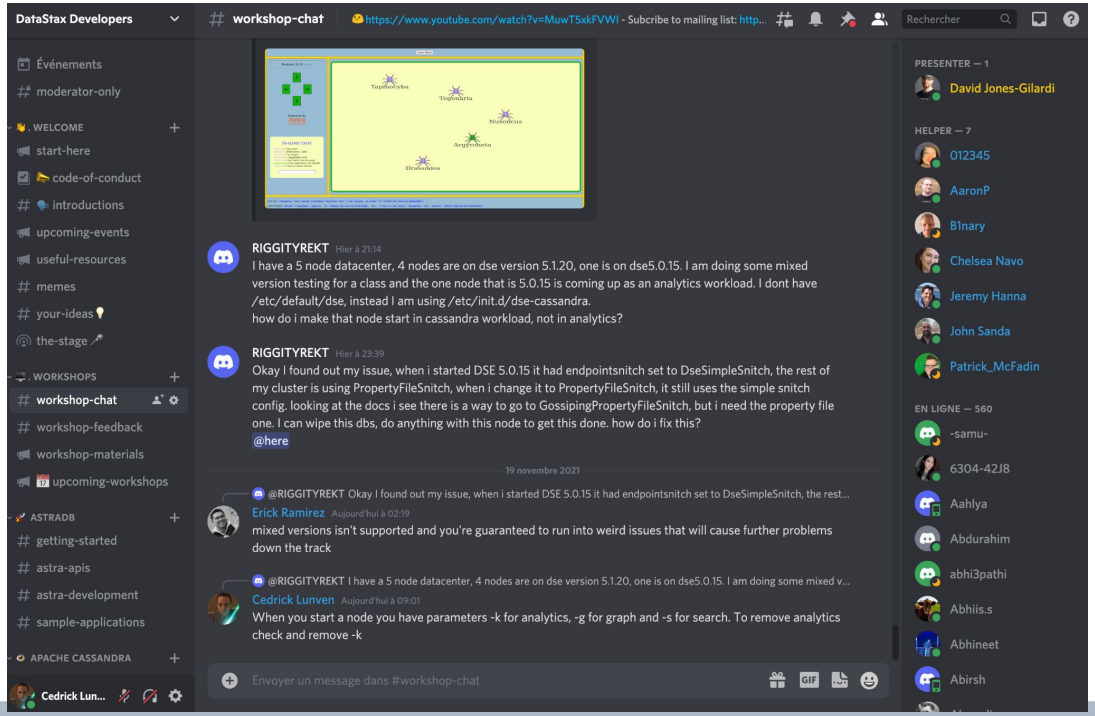
06

SWAG

Agenda

Join our 19k Discord Community

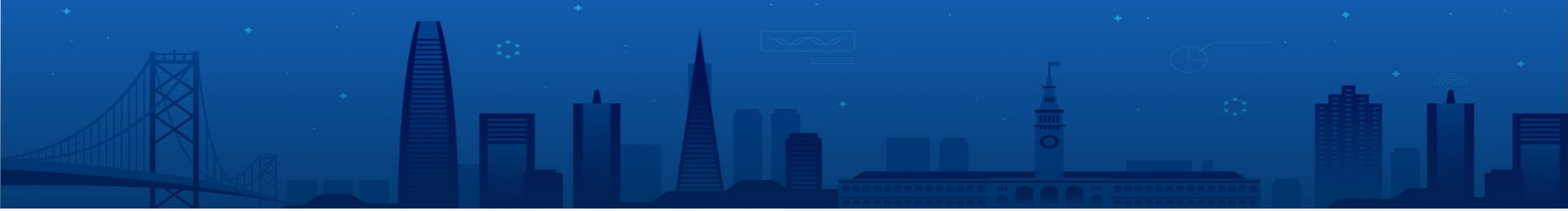
DataStax Developers



!discord

dtsx.io/discord

Discord Community



Astra Day San Francisco
August 24th 2022
3PM – 7PM

Register: bit.ly/SFAstraDB

Who should attend:
Sr Managers, IT Decision Makers, Architects, & App/Dev Managers

Astra DB

AGENDA

- 2:30 - 3:00 Registration
- 3:00 - 3:30 Welcome + Why Real-Time Data Matters
- 3:30 - 4:30 How Astra DB Propels Digital Transformation + Product Demo
- 4:30 - 5:00 Customer Success Fireside Chat with Samar Abbas, CTO Temporal
- 5:15 - 5:30 Closing + Open Q&A
- 5:30 - 7:00 Reception (Open bar + hors d'oeuvre)

Who wants some SWAG?

It’s time for some questions...

Thank You!