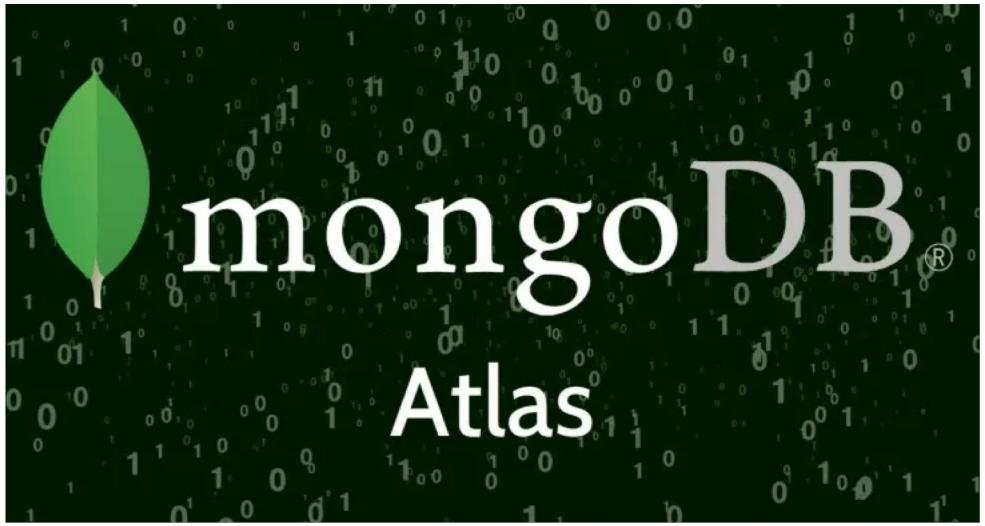







# Toronto MongoDB Atlas Workshop

Eugene Kang, Sr. Solutions Architect, MongoDB
Jeegar Ghodasara, Sr. Solutions Architect, MongoDB
Will Chow, Sr. Solutions Architect, MongoDB
Amy Rosenberg, Manager, Field Marketing, MongoDB



# Agenda

Introductions and Kickoff

MongoDB Technical Overview

Atlas Overview and Demo

CRUD Operations and Indexes/Explain Plans

Aggregation Framework

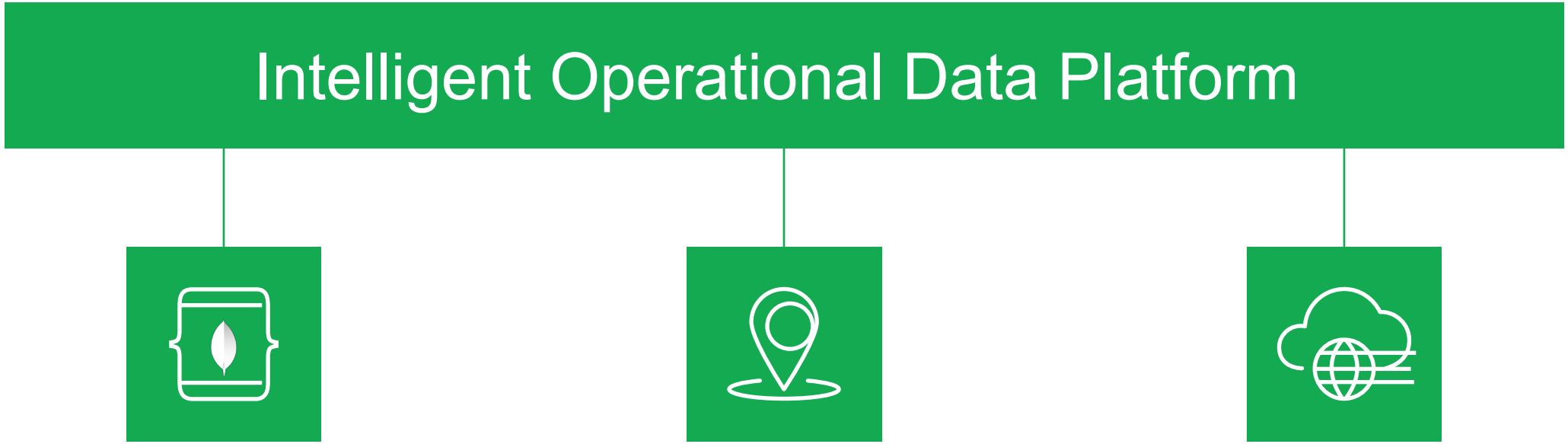
MongoDB Stitch

MongoDB Charts

Closing Remarks

# MongoDB Technical Overview

# Why MongoDB?


Intelligent Operational Data Platform
Best way to work with data
Intelligently put data where you need it
Freedom to run anywhere

# Cases for Change

MongoDB is a modern, operational database that supports a polyglot data strategy – on-premise and in the cloud. This allows us to drive several business critical topics with our customers.

**Cloud Data Strategy**
Leveraging the right data platforms as part of your overall cloud strategy helps to avoid vendor lock-in.

**Legacy Modernization**
Current legacy technology stacks can't cope with the range of new business requirements – we can help you modernise in a highly efficient and effective way

**Mainframe Offloading**
Reduce cost and MIPS on legacy mainframes and enable data to be leveraged for new use cases.

**Single View**
Provide a holistic view of data entities (e.g. customer) across multiple underlying, disconnected source systems.

**Operational Intelligence**
Solving the problem of deriving value from existing EDW or Hadoop-based data lake solutions in real-time

**Internet of Things (IoT)**
MongoDB can help you overcome Scalability & Performance issues that are not being met by many current IoT solutions.

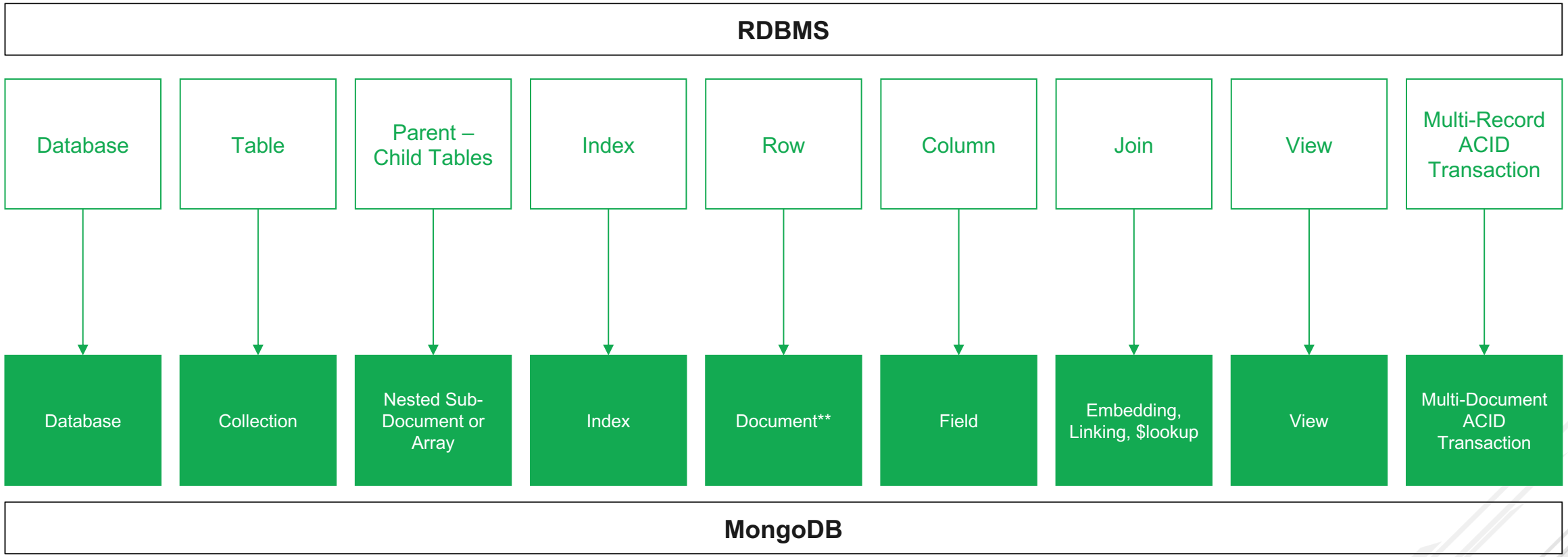
**Distributed Ledger/Blockchain**
MongoDB is the ideal database/persistence layer for enterprise-grade Distributed Ledger applications

# Leading modern general purpose database

| | | | | | |
|---|---|---|---|---|---|
| Adobe<br>Creative Cloud | ADP<br>HR Mobile Application | amadeus<br>Real-Time Travel Search | AstraZeneca<br>Mobile Drug Applications | BARCLAYS<br>Mobile Banking | Bell<br>Multi-Screen TV |
| BOSCH<br>Internet of Things Platform | CapitalOne<br>Predictive Messaging | Checkr<br>Background Checks as a Service | cisco<br>Order Capture | coinbase<br>Cryptocurrency Platform | DHA<br>Logistics Modernization |
| DELL<br>E-Commerce System | U.S. Immigration and Customs Enforcement<br>Entry Decision System | DWP Digital<br>Social Security Benefits Program | VA U.S. Department of Veterans Affairs<br>Mobile App for Patient Data | ebay<br>Product Catalog | EPIC GAMES<br>Gaming Platform |
| experian health<br>Single View of Patient | Genentech<br>Genetic Analysis | Goldman Sachs<br>Online Lending | HM Revenue & Customs<br>Online Tax Returns | Morgan Stanley<br>Swap Equities Management | new balance<br>Real-Time Analytics |
| OTTO<br>E-Commerce Personalization | pitney bowes<br>E-Commerce Platform | salesforce<br>Marketing Cloud | sky<br>Video Streaming | splunk><br>Log Metadata Store | sprinklr<br>Social Media Management |
| SQUARESPACE<br>Website Platform | STAPLES<br>Product Catalog | Symantec<br>Identity Theft Protection | ThermoFisher SCIENTIFIC<br>Mass Spectrometer Instrumentation | T-Mobile<br>Subscriber QoS | URBN URBAN OUTFITTERS INC.<br>Shopping Cart |

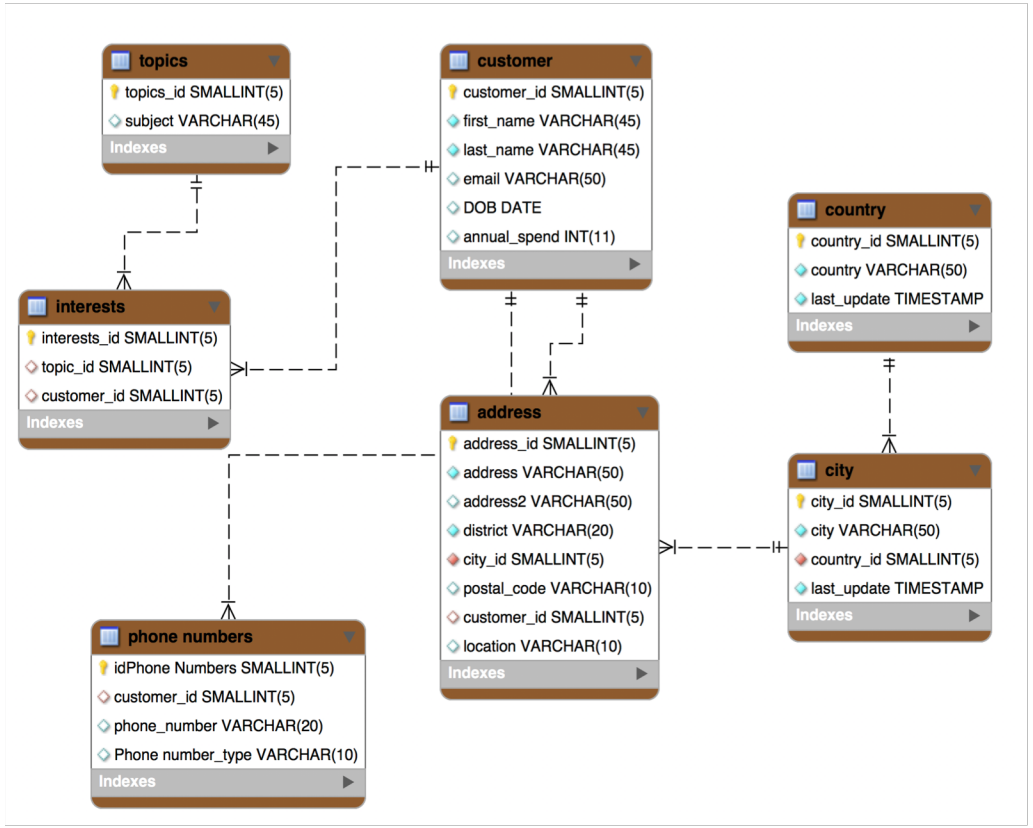
# Some Terminology

## A comparison




** Proper document schema design yields more entity data per document than found in a relational database row

# **Easy:** Contrasting data models




**Tabular (Relational) Data Model**

Related data split across multiple records and tables

```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first" : "John",
    "last" : "Doe" },
  "address" : [
    { "location" : "work",
      "address" : {
        "street" : "16 Hatfields",
        "city" : "London",
        "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point", "coord" : [
          51.5065752,-0.109081]}},
+   {...}
  ],
  "phone" : [
    { "location" : "work",
      "number" : "+44-1234567890"},
+   {...}
  ],
  "dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75")
}
```

**Document Data Model**

Related data contained in a single, rich document

# **Flexible:** Adapt to change

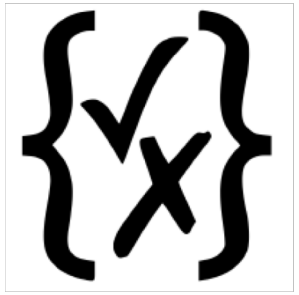
```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first" : "John",
    "last" : "Doe" },
  "address" : [
    { "location" : "work",
      "address" : {
        "street" : "16 Hatfields",
        "city" : "London",
        "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point", "coord" : [
        51.5065752,-0.109081]}},
+   {...}
  ],
"dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75")

}
```

```
{
  "_id" : ObjectId("5ad88534e3632e1a35a58d00"),
  "name" : {
    "first" : "John",
    "last" : "Doe" },
  "address" : [
    { "location" : "work",
      "address" : {
        "street" : "16 Hatfields",
        "city" : "London",
        "postal_code" : "SE1 8DJ"},
      "geo" : { "type" : "Point", "coord" : [
        51.5065752,-0.109081]}},
+   {...}
  ],
  "phone" : [
    { "location" : "work",
      "number" : "+44-1234567890"},
+   {...}
  ],
  "dob" : ISODate("1977-04-01T05:00:00Z"),
  "retirement_fund" : NumberDecimal("1292815.75")
}
```

**Add new fields dynamically at runtime**

# Flexible: Govern



## JSON Schema

Enforces strict schema structure over a complete collection for data governance & quality

- Builds on document validation introduced by restricting new content that can be added to a document
- Enforces presence, type, and values for document content, including nested array
- Simplifies application logic

**Tunable:** enforce document structure, log warnings, or allow complete schema flexibility

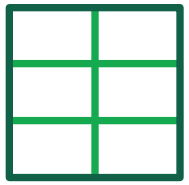
**Queryable:** identify all existing documents that do not comply

# Schema Validation Example

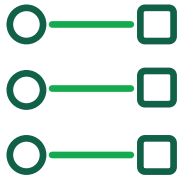
```
db.runCommand({
  collMod: "orders",
  validator: {
    $jsonSchema: {
      bsonType: "object",
      additionalProperties: false,
      required: ["item", "price"],
      properties: {
       _id: {},
       item: {
            bsonType: "string",
            description: "'item' must be a string and is required"
        },
        price: {
          bsonType: "decimal",
          description: "'price' must be a decimal and is required"
        },
        quantity: {
          bsonType: ["int", "long"],
          minimum: 1,
          maximum: 100,
          exclusiveMaximum: true,
          description:
            "'quantity' must be short or long integer between 1 and 99"
        }
      }
    }
  }
});
```

# **Versatile:** Multiple data models, rich query functionality


*JSON Documents*



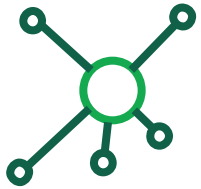
*Tabular*



*Key-Value*


*Text*

*Geospatial*



*Graph*

## Rich Queries

Point | Range | Geospatial | Faceted Search | Aggregations | JOINs | Graph Traversals

# Fully Indexable

## Fully featured secondary indexes

### Index Types

- Primary Index
  - Every Collection has a primary key index
- Compound Index
  - Index against multiple keys in the document
- MultiKey Index
  - Index into arrays
- Text Indexes
  - Support for text searches
- GeoSpatial Indexes
  - 2d & 2dSphere indexes for spatial geometries
- Hashed Indexes
  - Hashed based values for sharding

### Index Features

- TTL Indexes
  - Single Field indexes, when expired delete the document
- Unique Indexes
  - Ensures value is not duplicated
- Partial Indexes
  - Expression based indexes, allowing indexes on subsets of data
- Case Insensitive Indexes
- supports text search using case insensitive search
- Sparse Indexes
  - Only index documents which have the given field

# **Easy:** MongoDB Multi-Document ACID Transactions


## Just like relational transactions

- Multi-statement, familiar relational syntax
- Easy to add to any application
- Multiple documents in 1 or many collections and databases

## ACID guarantees

- Snapshot isolation, all or nothing execution
- No performance impact for non-transactional operations

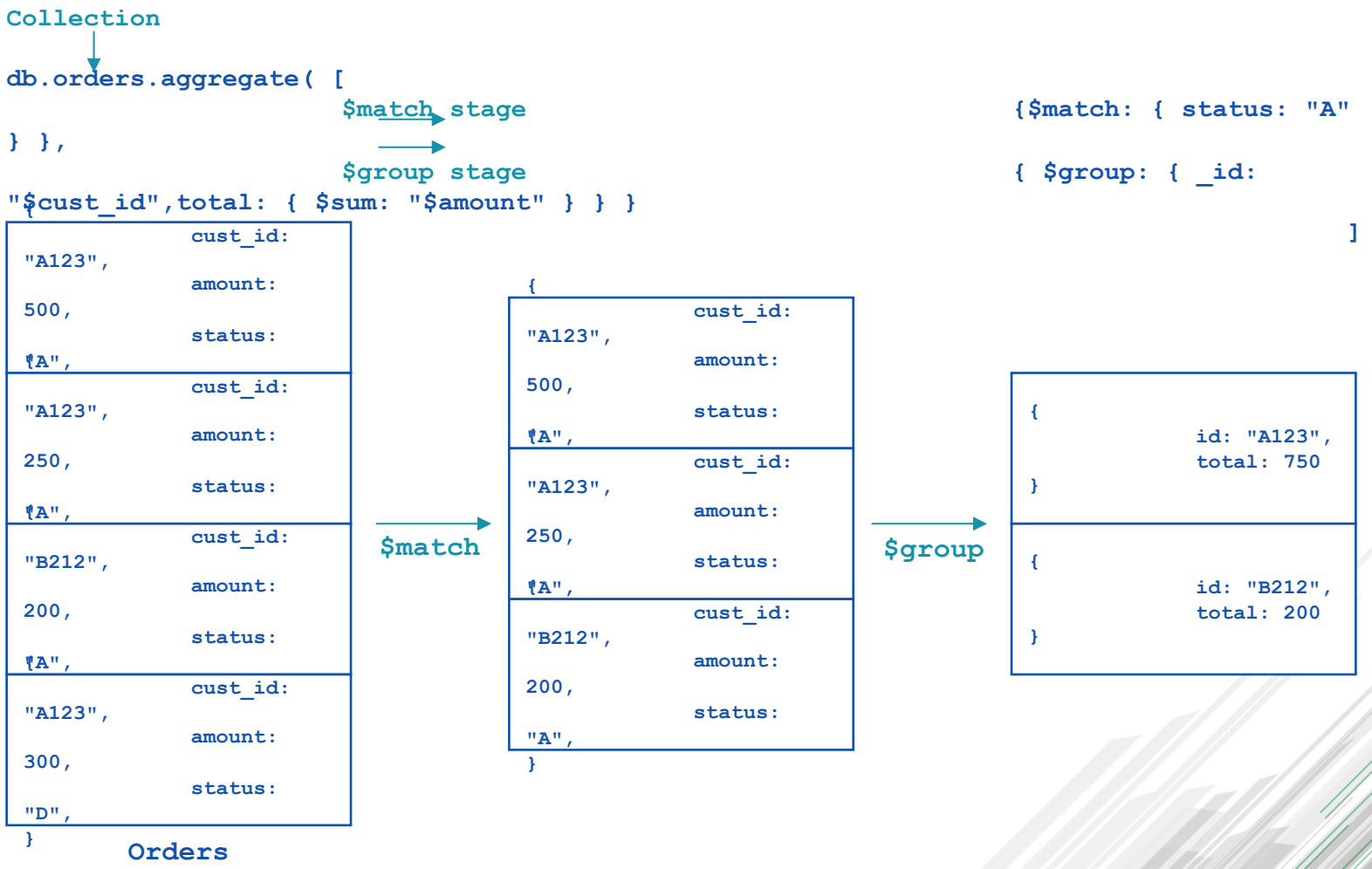
## Schedule

- MongoDB 4.0: replica set
- MongoDB 4.2: extended to sharded clusters

# Aggregations

Advanced data processing pipeline for transformations and analytics

- Multiple stages
- Similar to a unix pipe
  - Build complex pipeline by chaining commands together
- Rich Expressions



# Aggregation Features

## A feature rich analytical framework

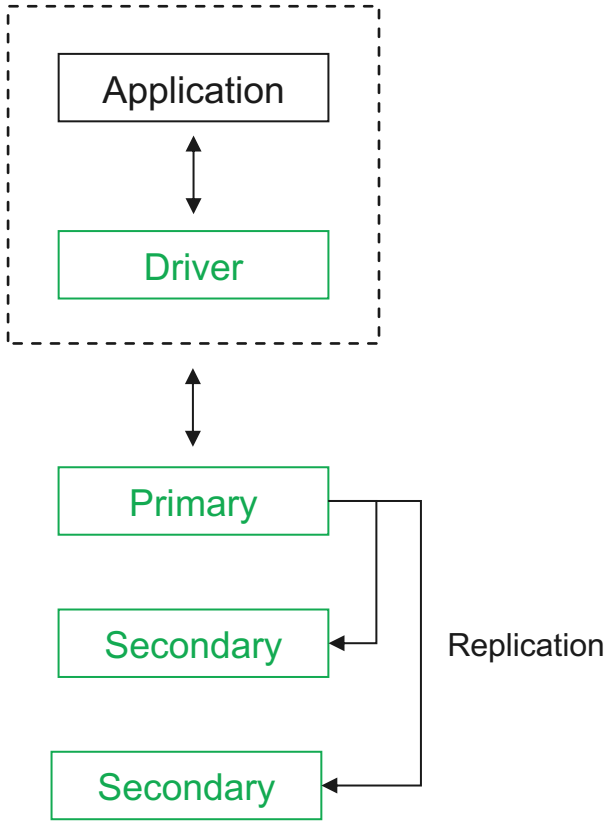
### Pipeline Stages

- $match
- $group
- $facet
- $geoNear
- $graphLookup
- $lookup
- $project
- $sort
- $unwind

### Operators

- Mathematical
  - $add, $abs, $substract, $multiply, $divide, $log, $log10, $stdDevPop, $stdDevSam, $avg, $sqrt, $pow, $sum, $zip, $convert, etc.
- Array
  - $push, $reduce, $reverseArray, $addToSet, $arrayElemAt, $slice, etc.
- Conditionals
  - $and, $or, $eq, $lt, $lte, $gt, $gte, $cmp, $cond, $switch, $in, etc
- Date
  - $dateFromParts, $dateToParts, $dateFromString, $dateToString, $dayOfMonth, $isoWeek, $minute, $month, $year, etc.
- String
  - $toUpper, $toLower, $substr, $strcasecmp, $concat, $split, etc.
- Laterals
  - $exp, $let, $literal, $map, $type, etc

# MongoDB Replica Sets




## Replica Set – 2 to 50 copies

## Self-healing

## Data Center Aware

## Addresses availability considerations:

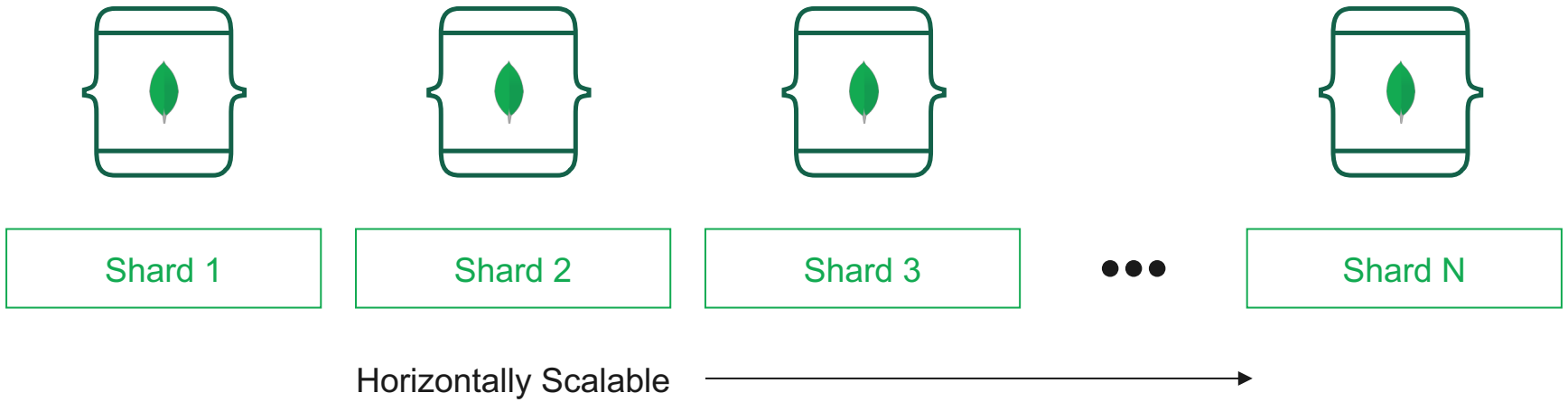
- High Availability
- Disaster Recovery
- Maintenance

## Workload Isolation: operational & analytics

# Serving global audiences with relational databases

# MongoDB’s native replica sets puts your entire database right next to users

# **Scaling MongoDB:** Automatic Sharding

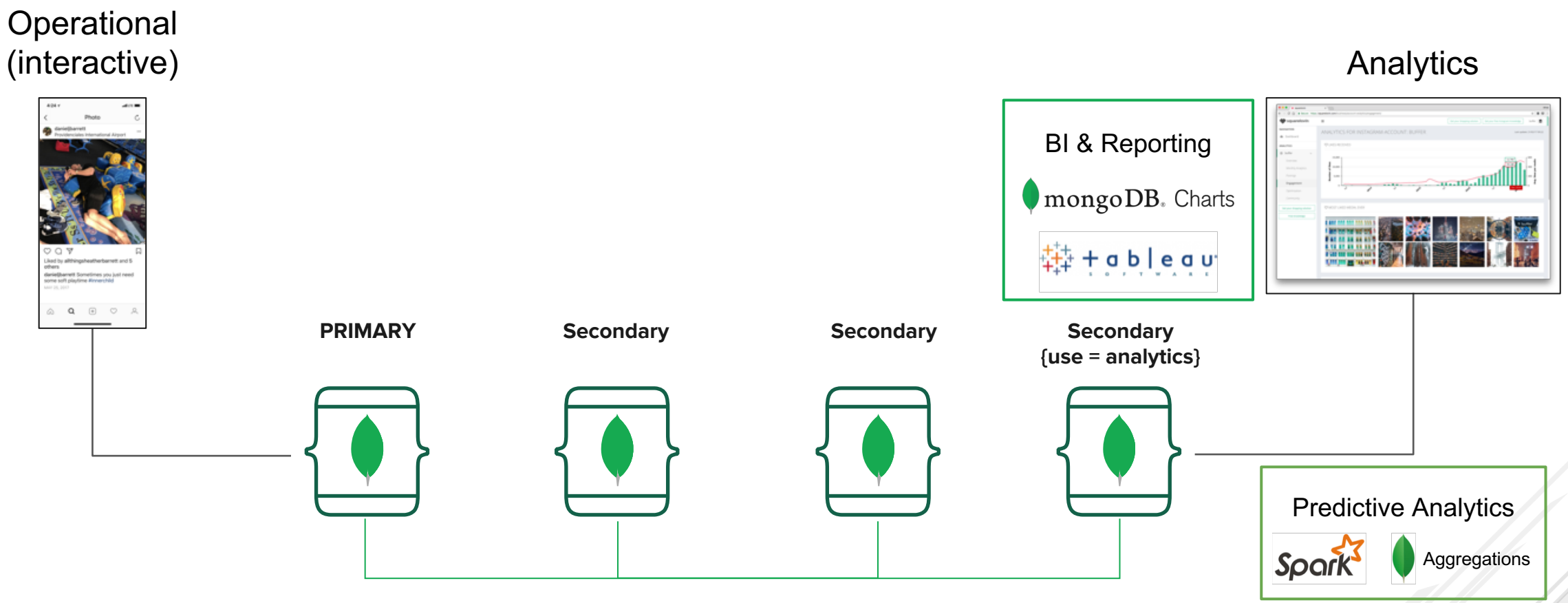



Multiple sharding policies: hashed, ranged, zoned

Increase or decrease capacity as you go

Automatic balancing for elasticity

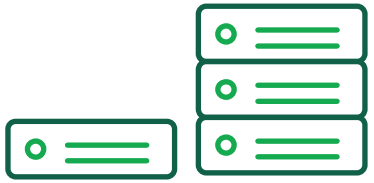
# Co-locating operational and analytical workloads



# Freedom to run anywhere

**Local**

**On-premises**
Server & Mainframe



**Private cloud**

**Hybrid cloud**

**Public cloud**

**Fully managed cloud service**

- Database that runs the same everywhere
- Leverage the benefits of a multi-cloud strategy
- Global coverage
- Avoid lock-in

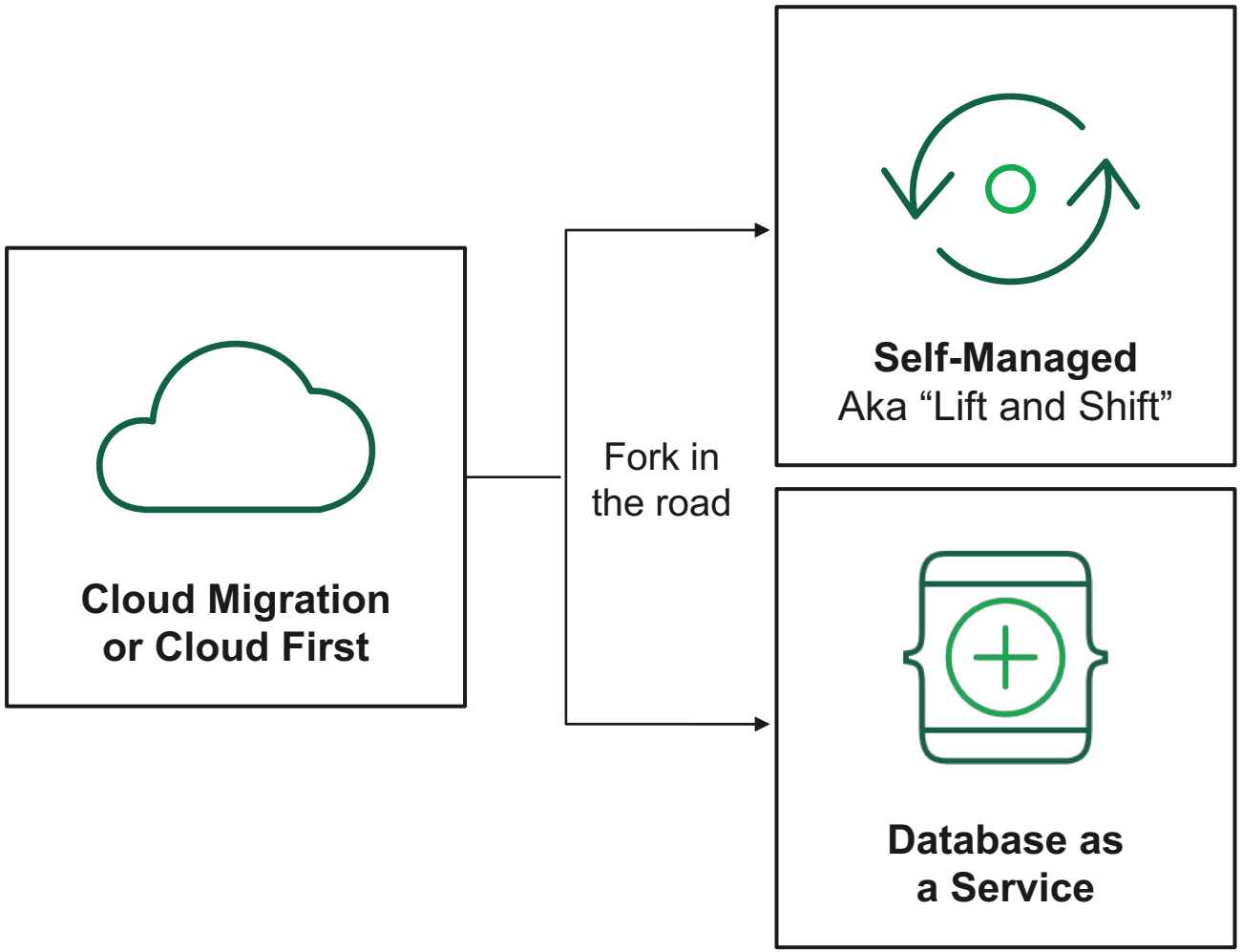
**Convenience:** same codebase, same APIs, same tools, wherever you run

# **Easy:** Drivers and Frameworks

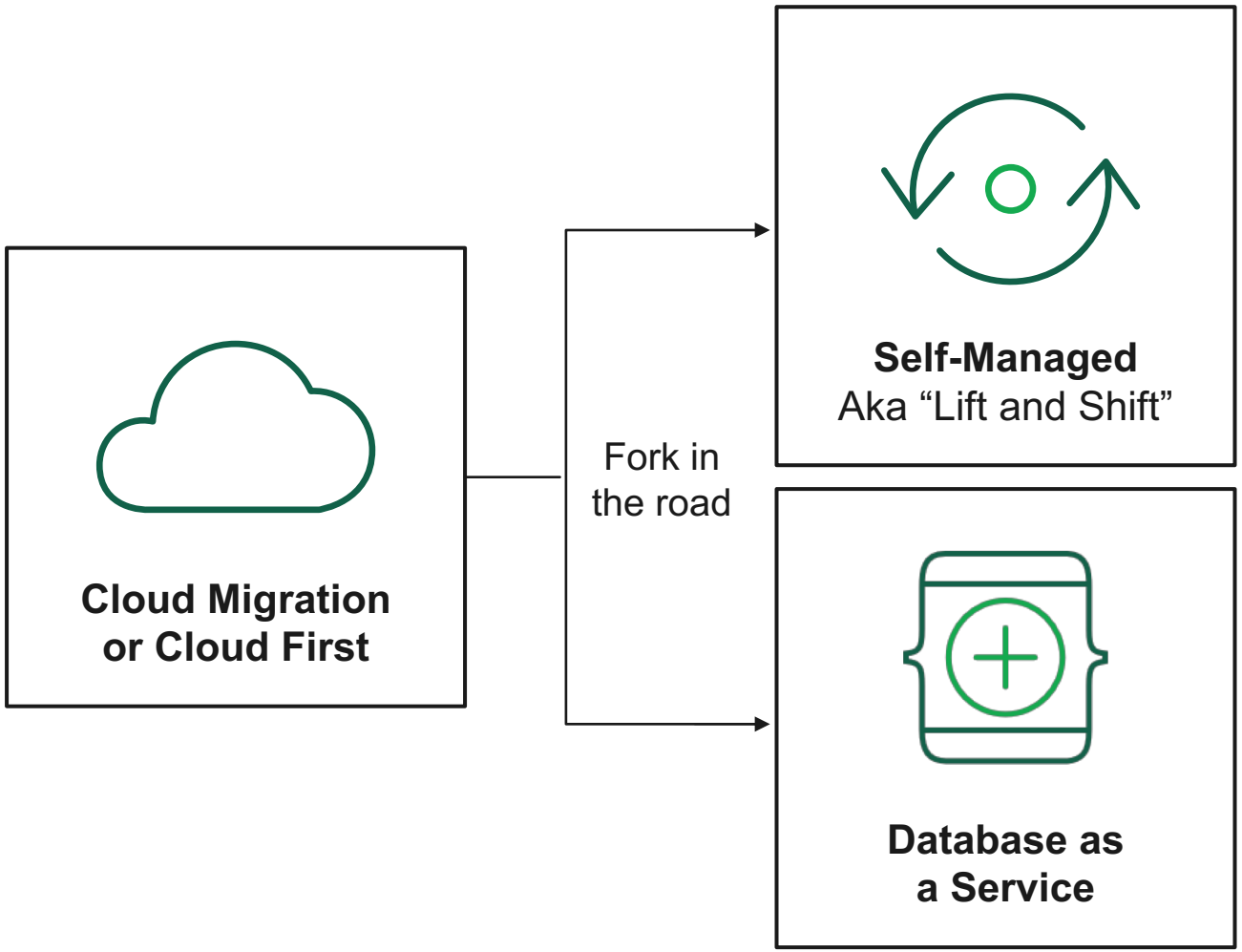
## Drivers

## Frameworks

# **2 choices as you move to the cloud:** Self-Managed or DBaaS

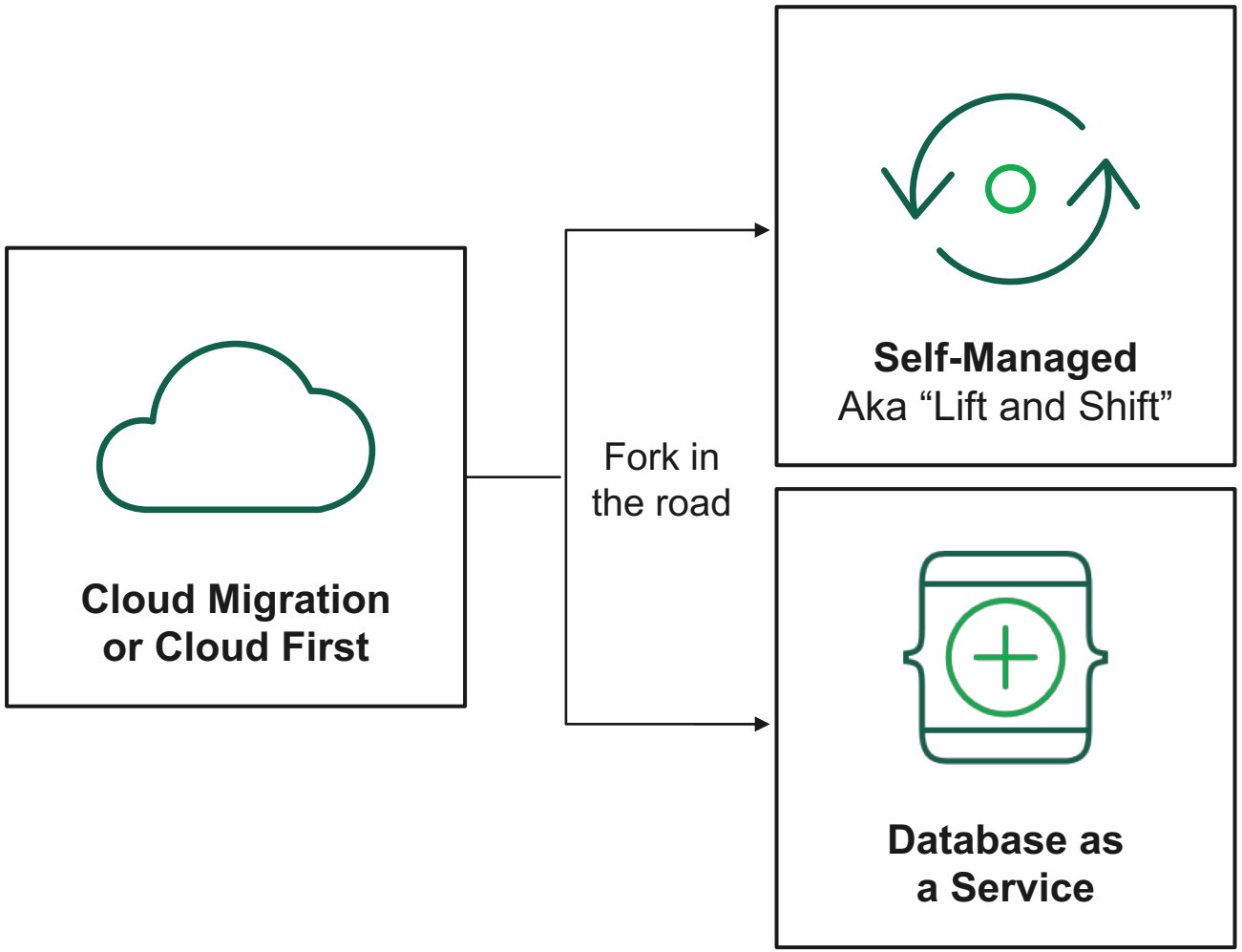


# **2 choices as you move to the cloud:** Self-Managed or DBaaS




1. Provision instances and storage
2. Configure HA
3. Configure security
4. Configure backup/restore
5. Monitoring & alerting
6. Ongoing upgrades & maintenance

# **2 choices as you move to the cloud:** Self-Managed or DBaaS




1. Provision instances and storage
2. Configure HA
3. Configure security
4. Configure backup/restore
5. Monitoring & alerting
6. Ongoing upgrades & maintenance

**Choose instance, hit deploy, available in a couple of minutes**

# Atlas

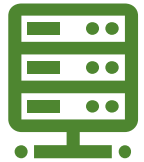
unlocks **agility** and **reduces cost**

Self-service and elastic

Global and highly available

Secure by default

Comprehensive monitoring



Managed backup

Cloud agnostic

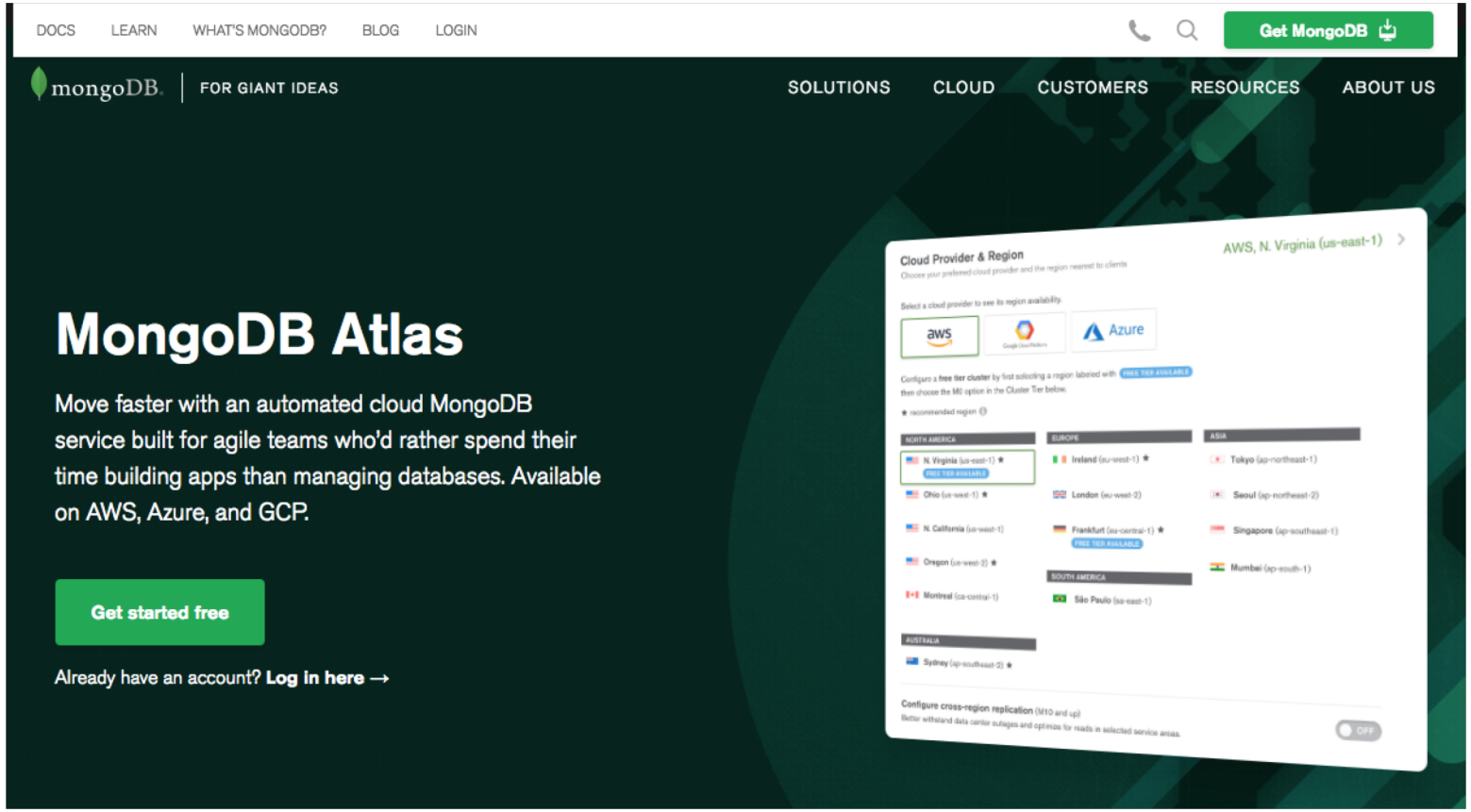
mongoDB

# Atlas Demo

# Atlas Workshop

# Sign Up for Free

https://www.mongodb.com/cloud/atlas




# EXERCISE 1: MONGODB ATLAS & USERS

# AFTER EXERCISE 1, YOU SHOULD BE ABLE TO...

- Deploy a MongoDB Atlas Cluster (free-tier M0 or paid M10+) (Go to https://www.mongodb.com/cloud/atlas )
- Secure the cluster, e.g. user and IP whitelisting
- Understand the basic features of Atlas

# EXERCISE 1: LET'S DO IT!

- Deploy M0 instance on your choice of cloud provider
- Give the cluster a name
- Create a user via Atlas (db.createuser())
- testuser / testuser


# EXERCISE 2: MONGODB COMPASS & CRUD

## AT THE END OF THIS EXERCISE 2, YOU SHOULD BE ABLE TO...

- Launch *Compass* (tool to explore, visualize, optimize, and modify MongoDB data) and connect to that cluster
- Use Compass to create a database, create a collection, and import a data set into that collection
- Use Compass to create, read, update, and delete (CRUD) documents in a collection

# EXERCISE 2: LET'S DO IT!

- Download and Install Compass: https://bit.ly/2S2jozr
- Connect to Atlas with Compass
- Create a database and collection
  - Database: movies
  - Collection: movies
- Download data: https://bit.ly/2MpsvoJ
- Import data using Compass

# CONNECTION STRINGS

| Key | Value |
| --- | --- |
| Server | <<insert your server details here>> |
| SRV Record | Yes on Switch |
| Authentication | Username / Password |
| Username | <<insert your username that you created>> |
| Password | <<insert your password that you created>> |
| Auth DB | admin |

# CRUD: LET'S CREATE...

```
db.movies.insertOne(
{
   "title": "Ghostbusters",
   "Year": 1984,
   "Rated": "PG",
   "Runtime": 105,
   "Type": "movie",
   "Genres": ["comedy", "action"] ,
   "Director": "Ivan Reitman",
   "Writers": ["dan aykroyd", "Harold ramis"]
}
)
```

# SCHEMA AND DATA TYPES

# (COMPASS)

# CRUD: LET'S READ...

```
db.movies.findOne()
db.movies.find().pretty()
```

# CRUD: LET'S UPDATE...

```
db.movies.updateOne(
   {
        title: "Ghostbusters"
   },
   {
        $set: {
        imdb: { id: "tt0087332", rating: 7.8, votes: 312798 }
        }
   }
)
```

# CRUD: LET'S DELETE...

```
db.movies.deleteOne(
{
        title: "Ghostbusters"
}
)
```

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | |
| “Comedy” as the only genre | |
| “Comedy” or “Drama” | |
| “Comedy” and “Drama” | |
| IMDB Rating>8.0 and PG Rating | |
| title starting with  “Dr. Strangelove” | |

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | db.movies.find({genres: "Comedy"}) |
| “Comedy as only genre | |
| “Comedy” or “Drama” | |
| “Comedy” and “Drama” | |
| IMDB Rating>8.0 and PG Rating | |
| title starting with “Dr. Strangelove” | |

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | db.movies.find({genres: "Comedy"}) |
| “Comedy as only genre | db.movies.find({genres:["Comedy"]}) |
| “Comedy” or “Drama” | |
| “Comedy” and “Drama” | |
| IMDB Rating>8.0 and PG Rating | |
| title starting with  “Dr. Strangelove” | |

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | db.movies.find({genres: "Comedy"}) |
| “Comedy as only genre | db.movies.find({genres:["Comedy"]}) |
| “Comedy” or “Drama” | db.movies.find({genres:{$in:["Comedy", "Drama"]}}) |
| “Comedy” and “Drama” | |
| IMDB Rating>8.0 and PG Rating | |
| title starting with  “Dr. Strangelove” | |

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | db.movies.find({genres: "Comedy"}) |
| “Comedy as only genre | db.movies.find({genres:["Comedy"]}) |
| “Comedy” or “Drama” | db.movies.find({genres:{$in:["Comedy", "Drama"]}}) |
| “Comedy” and “Drama” | db.movies.find({ genres: { $all: ["Comedy", "Drama"] } }) |
| IMDB Rating>8.0 and PG Rating | |
| title starting with  “Dr. Strangelove” | |

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | db.movies.find({genres: "Comedy"}) |
| “Comedy as only genre | db.movies.find({genres:["Comedy"]}) |
| “Comedy” or “Drama” | db.movies.find({genres:{$in:["Comedy", "Drama"]}}) |
| “Comedy” and “Drama” | db.movies.find({ genres: { $all: ["Comedy", "Drama"] } }) |
| IMDB Rating>8.0 and PG Rating | db.movies.find({"imdb.rating" : {$gt: 8.0}, rated:"PG"}) |
| title starting with  “Dr. Strangelove” | |

# CRUD: FIND ME MOVIES...

| QUESTION | ANSWER |
| --- | --- |
| From 1987 | db.movies.find({year:1987}) |
| “Comedy” as one of their genres | db.movies.find({genres: "Comedy"}) |
| “Comedy as only genre | db.movies.find({genres:["Comedy"]}) |
| “Comedy” or “Drama” | db.movies.find({genres:{$in:["Comedy", "Drama"]}}) |
| “Comedy” and “Drama” | db.movies.find({ genres: { $all: ["Comedy", "Drama"] } }) |
| IMDB Rating>8.0 and PG Rating | db.movies.find({"imdb.rating" : {$gt: 8.0}, rated:"PG"}) |
| Title starting with “Dr. Strangelove” | db.movies.find({title: {$regex: '^Dr. Strangelove'}}) |


# EXERCISE 3: INDEXES AND EXPLAIN PLAN

## AT THE END OF THIS EXERCISE, YOU SHOULD BE ABLE TO...

- Identify how a query was run using the Explain Plan
- Create an index on a collection
- See how an index affects query plans

# INDEXES AND EXPLAIN PLAN

# COMPASS

# ADDING INDEX

- Rule of thumb (ESR):
  - Equality
  - Sort
  - Range


# EXERCISE 4: AGGREGATION FRAMEWORK

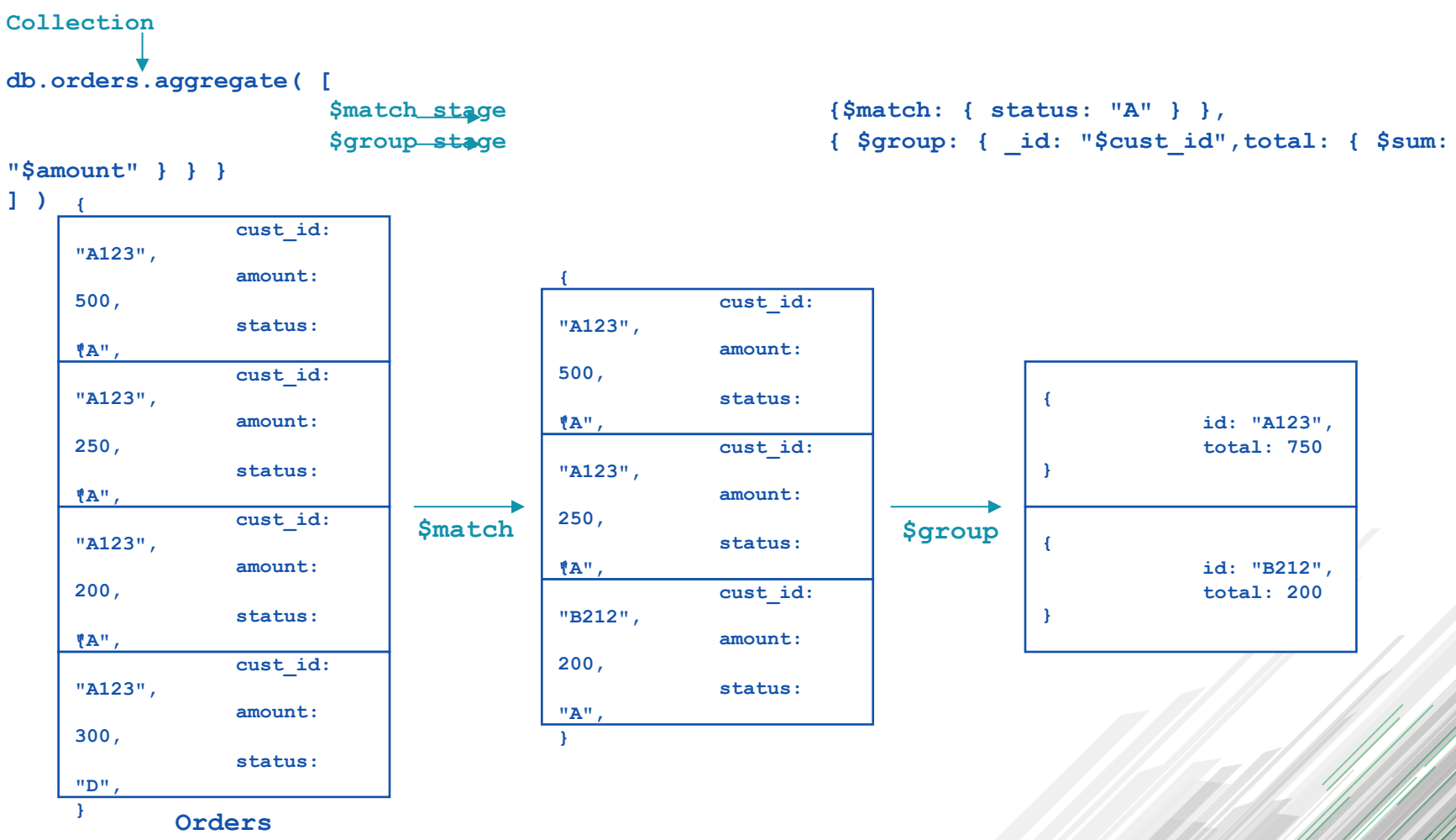
# AT THE END OF THIS EXERCISE 4 YOU SHOULD BE ABLE TO...

- Have a basic understanding of what the aggregation framework is
- Write some basic queries using the aggregation framework

# Aggregations

Advanced data processing pipeline for transformations and analytics

- Multiple stages
- Similar to a unix pipe
  - Build complex pipeline by chaining commands together
- Rich Expressions



# Aggregation Features

A feature rich analytical framework

## Pipeline Stages

- $match
- $group
- $facet
- $geoNear
- $graphLookup
- $lookup
- $project
- $sort
- $unwind

## Operators

- Mathematical
  - $add, $abs, $substract, $multiply, $divide, $log, $log10, $stdDevPop, $stdDevSam, $avg, $sqrt, $pow, $sum, $zip, etc.
- Array
  - $push, $reduce, $reverseArray, $addToSet, $arrayElemAt, $slice, etc.
- Conditionals
  - $and, $or, $eq, $lt, $lte, $gt, $gte, $cmp, $cond, $switch, $in, etc
- Date
  - $dateFromParts, $dateToParts, $dateFromString, $dateToString, $dayOfMonth, $isoWeek, $minute, $month, $year, etc.
- String
  - $toUpper, $toLower, $substr, $strcasecmp, $concat, $split, etc.
- Laterals
  - $exp, $let, $literal, $map, $type, etc

# LET’S PLAY WITH AGGREGATION PIPELINE BUILDER IN COMPASS

# AGGREGATION WORKSHOP

| QUESTION | ANSWER |
| --- | --- |
| How can you use $match to find all comedies? | |
| | |
| | |

# AGGREGATION WORKSHOP

| QUESTION | ANSWER |
| --- | --- |
| How can you use $match to find all comedies? | $match {genres: “Comedy”} |
| | |
| | |

# AGGREGATION WORKSHOP

| QUESTION | ANSWER |
| --- | --- |
| How can you use $match to find all comedies? | $match {genres: “Comedy”} |
| How can you use $unwind to create an individual document for each country? | |
| | |

# AGGREGATION WORKSHOP

| QUESTION | ANSWER |
| --- | --- |
| How can you use $match to find all comedies? | $match {genres: “Comedy”} |
| How can you use $unwind to create an individual document for each country? | $unwind {path: “$countries”} |
| | |

# AGGREGATION WORKSHOP

| QUESTION | ANSWER |
| --- | --- |
| How can you use $match to find all comedies? | $match {genres: “Comedy”} |
| How can you use $unwind to create an individual document for each country? | $unwind {path: “$countries”} |
| How can you use $group to count all the comedies grouped by country? | |

# AGGREGATION WORKSHOP

| QUESTION | ANSWER |
| --- | --- |
| How can you use $match to find all comedies? | $match {genres: “Comedy”} |
| How can you use $unwind to create an individual document for each country? | $unwind {path: “$countries”} |
| How can you use $group to count all the comedies grouped by country? | $group {<br>  _id: “$countries”,<br>  count: {$sum:1}<br>} |

# Lunch Break

# MongoDB Stitch

# A bit of background on why...

## More businesses driven by Web/Mobile Apps

- **Mobile internet usage has surpassed desktop**
  - **56%** of usage from Mobile/Tablet
- **Apps are expected to work across web, mobile, etc.**
  - **58%** of apps are developed for multiple platforms
- **Apps must be highly available, globally distributed**
  - **48%** of apps are deployed to the cloud

## Creating/Integrating services drives development

- **More is expected of developers**
  - **64%** of developers identified as full-stack
  - Leading to more time spent on integration/security
- **Microservice architecture gaining traction**
  - **69%** of teams considering MSA for new projects
- **Too much time spent on maintenance/integration**
  - **41%** of time spent maintaining
  - Only **39%** on new projects

# It’s easier to develop with MongoDB & Atlas, but building applications is still tough because…

- Applications still need somewhere for the app logic to live

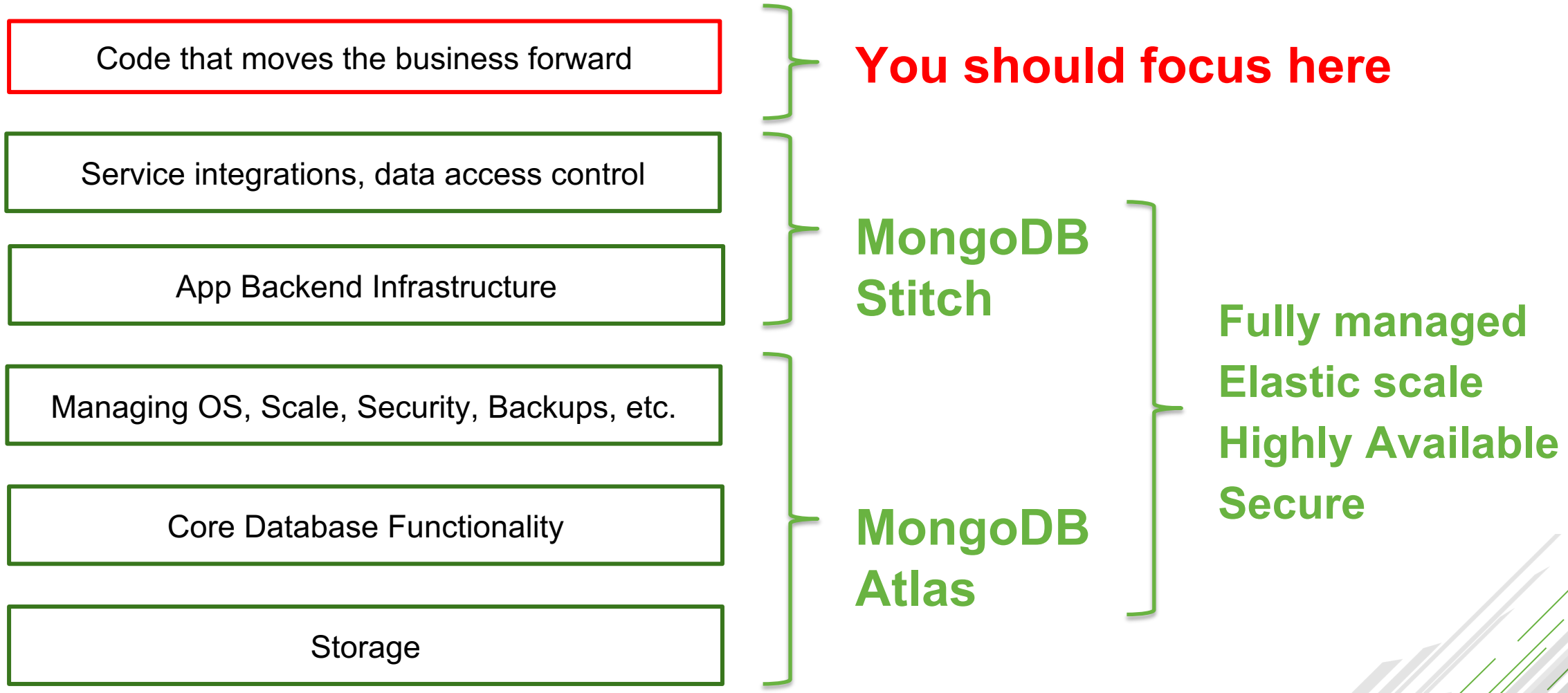
  **This comes will all the same headaches of maintaining databases the traditional way**

**Low to No Value**

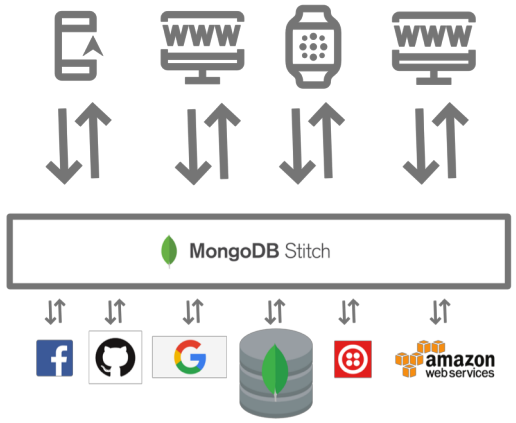
- Applications are expected to have features like
  - Facebook sign in
  - Push notifications
  - Email confirmations
  - Robust offline features

**Non Differentiating**

# Focus Your Energy Where You Can Make a Difference



# Save weeks of development and thousands of lines of code




| Without Stitch | With Stitch |
|---|---|
| Provision backend server | Not needed |
| Install backend runtime environment | Not needed |
| Code user authentication | {Simple JSON config} |
| Code data access controls | {Simple JSON config} |
| Code against each external service API | Code against a single SDK/API |
| Poll the database for changes | Not needed |
| Code REST API for frontend to use backend | Not needed |
| Add code to make backend HA | Not needed |
| Add code to scale backend | Not needed |
| Code backend application logic | Provide code for Stitch Functions |
| Code application frontend | Code application frontend using single SDK |
| Monitor & manage backend infrastructure | Not needed |

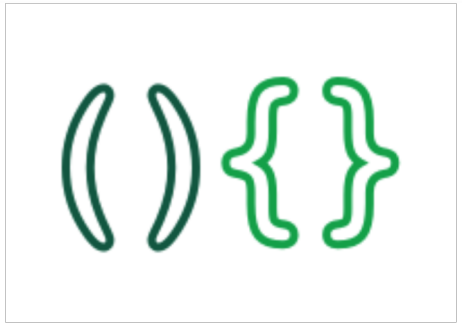
mongoDB

# MongoDB Stitch Serverless Platform – Services

**Stitch QueryAnywhere**

Brings MongoDB's rich query language safely to the edge

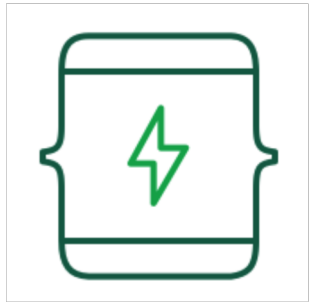
iOS, Android, Web, IoT



**Stitch Functions**

Integrate microservices + server-side logic + cloud services

Build full apps, or Data as a Service through custom APIs



**Stitch Triggers**

Real-time notifications let your application functions react in response to database changes, as they happen

**Stitch Mobile Sync**

Automatically synchronizes data between documents held locally in MongoDB Mobile and your backend database

**(coming soon)**

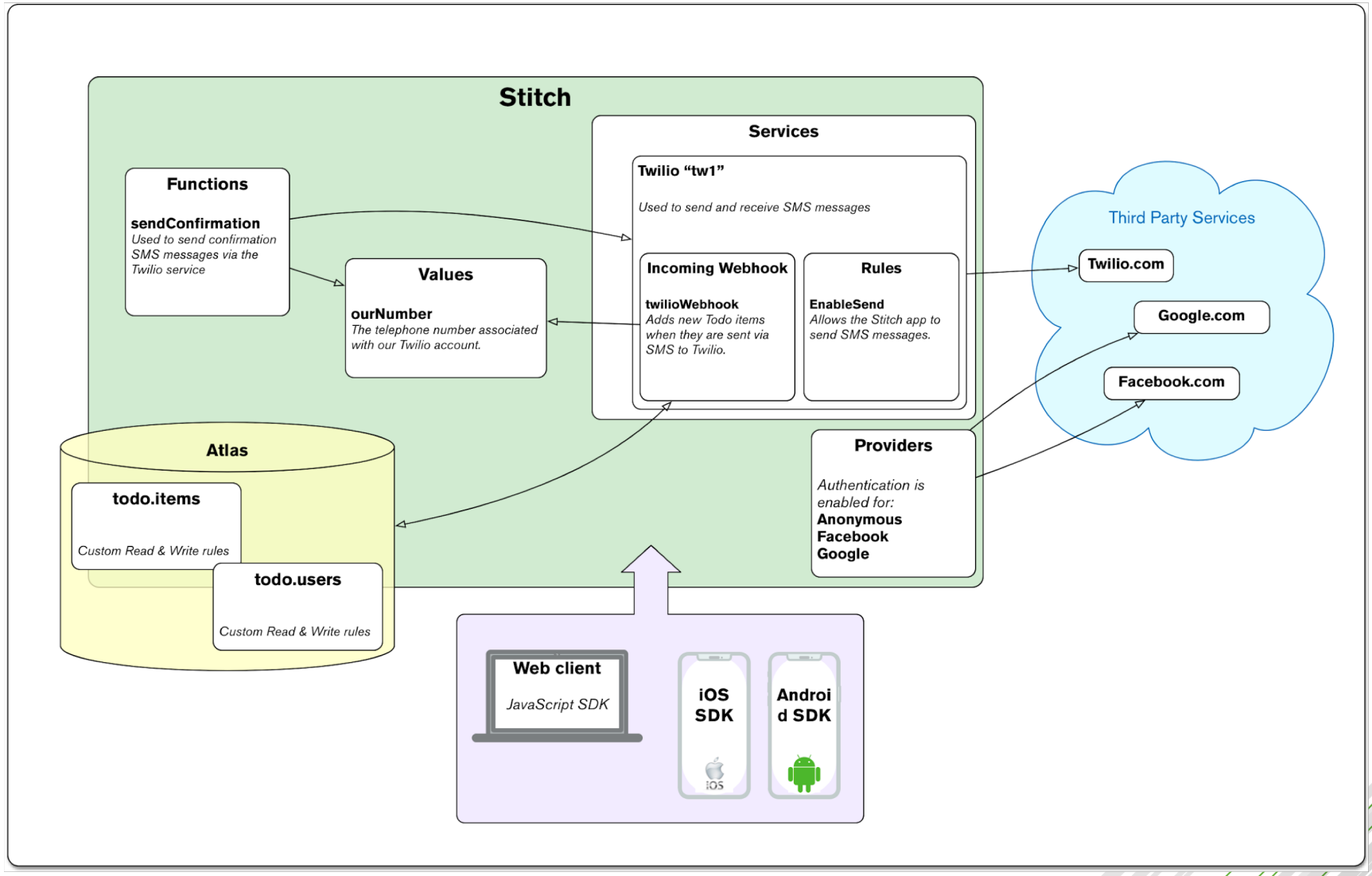
Streamlines app development with simple, secure access to data and services from the client with thousands of lines less code to write and no infrastructure to manage – getting your apps to market faster while reducing operational costs.

mongoDB

# ToDo App Demo

- Google Authentication
- Twilio, for SMS integration

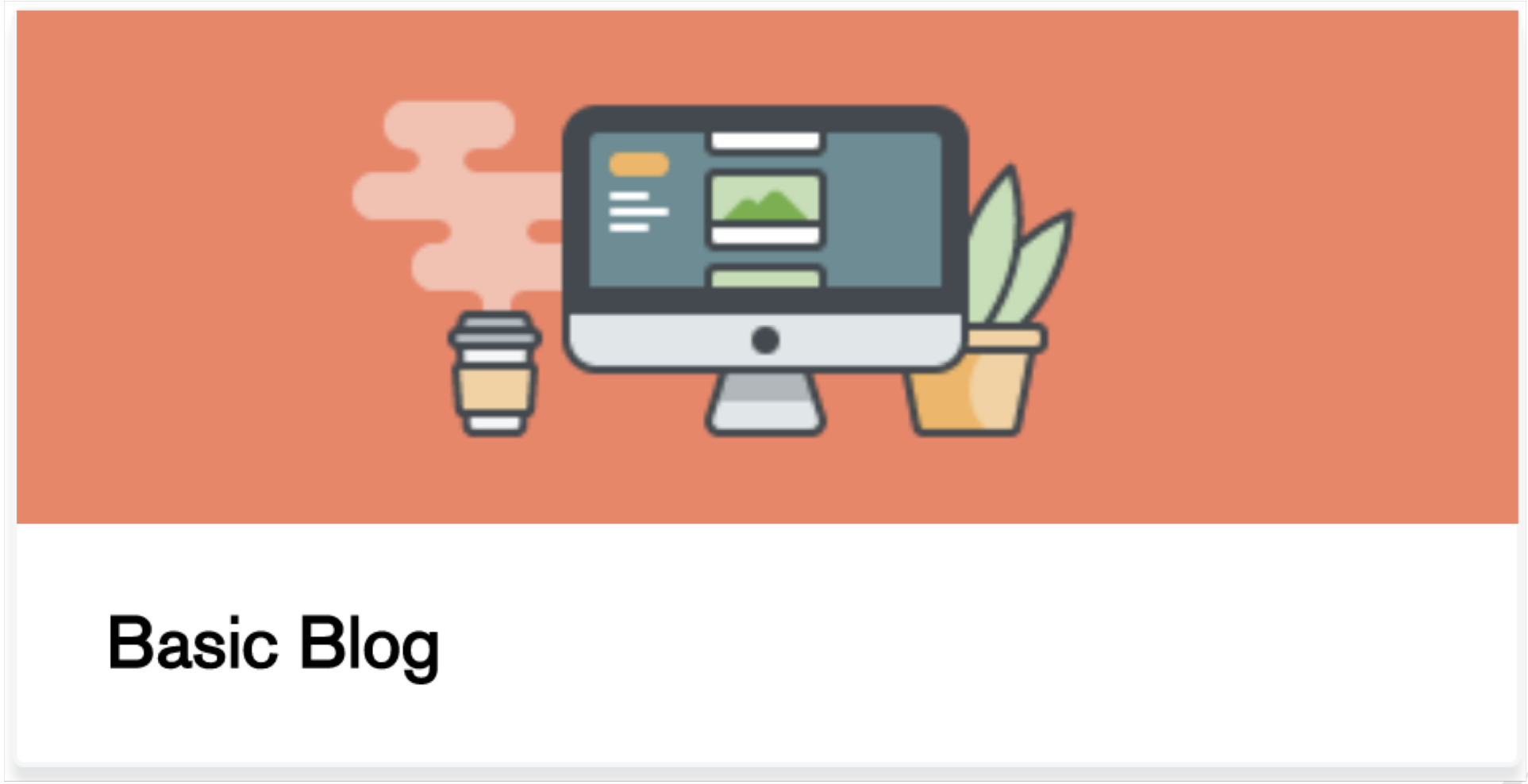
- Send text messages to add items to my ToDo list.




mongoDB

# MongoDB Stitch Workshop

# Workshop



# Workshop - Basic Blog

- What will we build?
  - a blog with a comments section backed by Stitch

- Prerequisites
  - MongoDB Atlas Account
  - MongoDB cluster hosted in Atlas

- Estimated Time to Complete: ~15 minutes

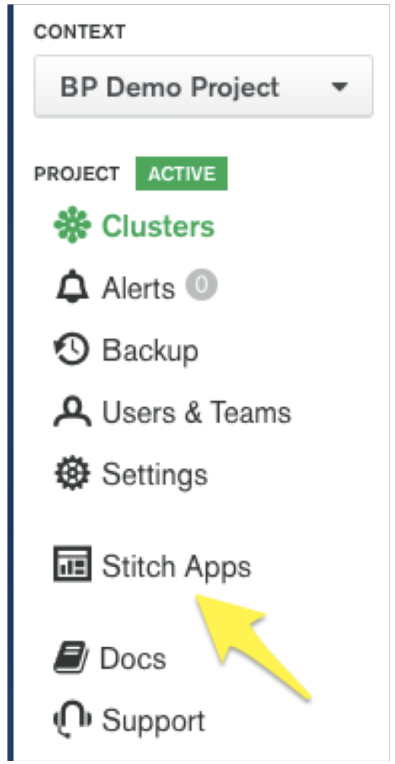
# Workshop - Basic Blog - Step 1

- Open a text editor
  - create `blog.html`

```
<html>
  <head>
  </head>
  <body>
    <h3>This is a great blog post</h3>
    <div id="content">
      I like to write about technology because I want to get on the
      front page of hacker news.
    </div>
    <hr>
    <div id="comments"></div>
  </body>
</html>
```

# Workshop - Basic Blog - Step 2

- Create a Stitch Application




1. Click **Stitch Apps** in the left-hand corner navigation of the Atlas console
2. Click **Create New Application**
3. Give the application a name (e.g. **Blog Tutorial**)
4. Click **Create**
5. Wait for your application to initialize

# Workshop - Basic Blog - Step 3

- Turn on Anonymous Authentication
  - From the **Getting Started** page
    - Enable **Anonymous Authentication** under the **Turn On Authentication** heading

# Workshop - Basic Blog - Step 4

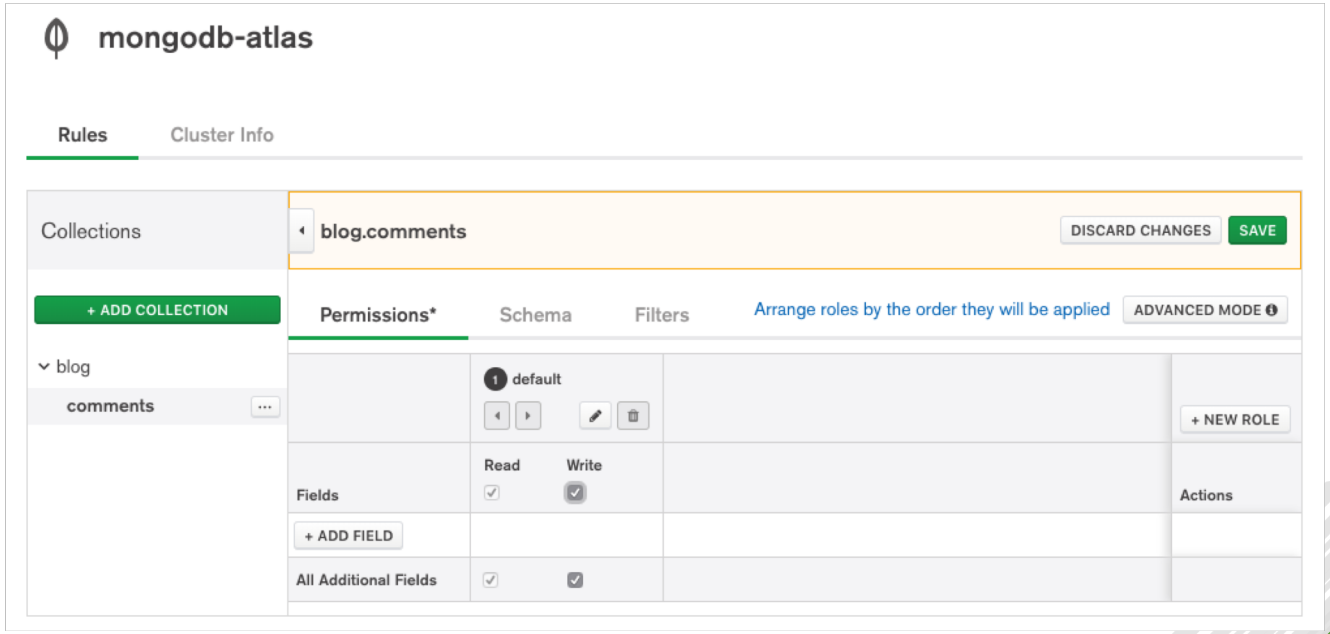
- Configure `blog.comments` MongoDB Collection

1. Click **Rules** under MongoDB Atlas in the left-hand navigation
2. Click **Add Collection**
3. Enter `blog` for the **Database Name**
4. Enter `comments` for the **Collection Name**
5. Select **No Template**
6. Click **Add Collection**

# Workshop - Basic Blog - Step 5

- Select the **Permissions** tab of the rules for the collection
  - enable reading and writing to the `comments` collection

1. Click the **Read** and **Write** checkboxes for the **default** role
2. Click **Save**




mongoDB

# Workshop - Basic Blog - Step 6

- Add a Commenting System to the Blog Post
  - add the following `<script>` tag to the `<head>` section of `blog.html`

```
<script src="https://s3.amazonaws.com/stitch-sdks/js/bundles/4.0.0/stitch.js"></script>
```

# Workshop - Basic Blog - Step 7

- Initialize Stitch Clients
  - Paste the following `<script>` tag in `blog.html` beneath the tag that imports Stitch SDK

```
<script>
  // Initialize the App Client
  const client = stitch.Stitch.initializeDefaultAppClient("<your-app-id>");
  // Get a MongoDB Service Client
  const mongodb = client.getServiceClient(
    stitch.RemoteMongoClient.factory,
    "mongodb-atlas"
  );
  // Get a reference to the blog database
  const db = mongodb.db("blog");
</script>
```

Replace `<your-app-id>` with your **Stitch App ID**

* find your **App ID** on the **Clients** page of the Stitch UI

mongoDB

# Workshop - Basic Blog - Step 8

- Query and Display Comments on Page Load
  - Add the following function to the `<script>` tag

```
function displayComments() {
  db.collection("comments")
    .find({}, {limit: 1000})
    .asArray()
    .then(docs => {
      const html = docs.map(doc => `<div>${doc.comment}</div>`);
      document.getElementById("comments").innerHTML = html;
    });
}
```

# Workshop - Basic Blog - Step 9

- Log In and Display Comments On Load
  - Setup anonymous authentication in `blog.html`
    - Add the following function to the `<script>` tag:

```
function displayCommentsOnLoad() {
  client.auth
    .loginWithCredential(new stitch.AnonymousCredential())
    .then(displayComments)
    .catch(console.error);
}
```

- Update the `<body>` tag in `blog.html` with

```
<body onload="displayCommentsOnLoad()">
```

# Workshop - Basic Blog - Step 10

- Add a Form for Submitting Comments
  - Add the `addComment()` function to the `<script>` tag

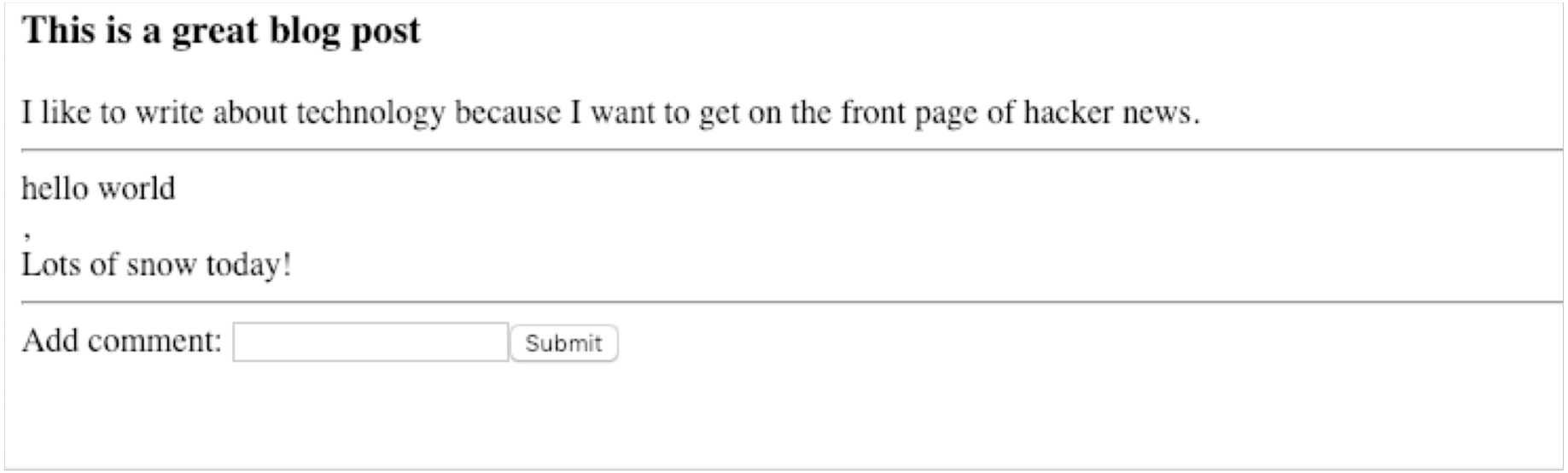
```
function addComment() {
  const newComment = document.getElementById("new_comment");
  console.log("add comment", client.auth.user.id)
  db.collection("comments")
    .insertOne({ owner_id : client.auth.user.id, comment: newComment.value })
    .then(displayComments);
  newComment.value = "";
}
```

- Add a comment form in the `<body>` of `body.html`

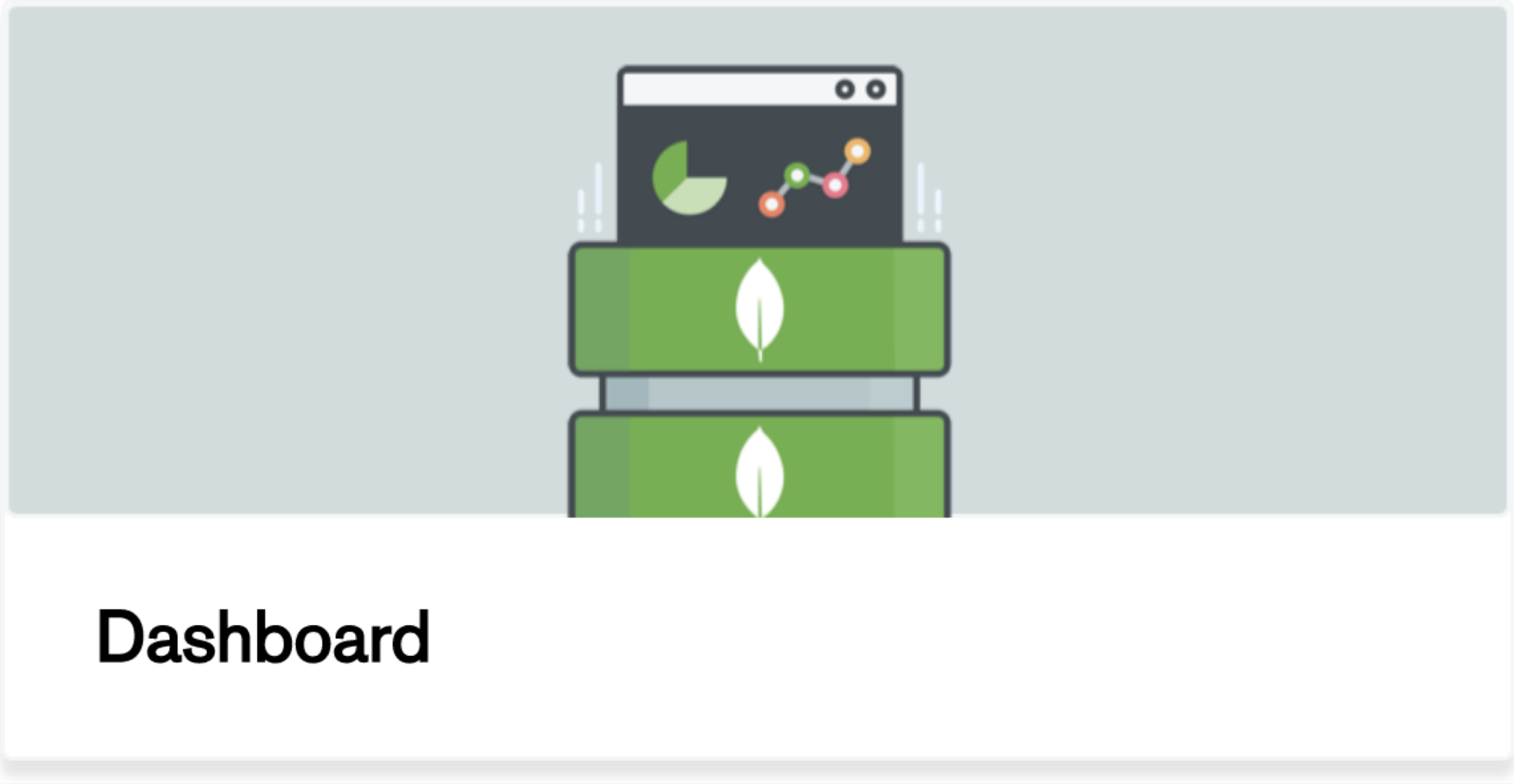
```
<hr>
Add comment:
<input id="new_comment"><input type="submit" onClick="addComment()">
```

# Workshop - Basic Blog - Summary

At this point, you should be able to refresh the page and submit a comment on the blog post.



# Workshop

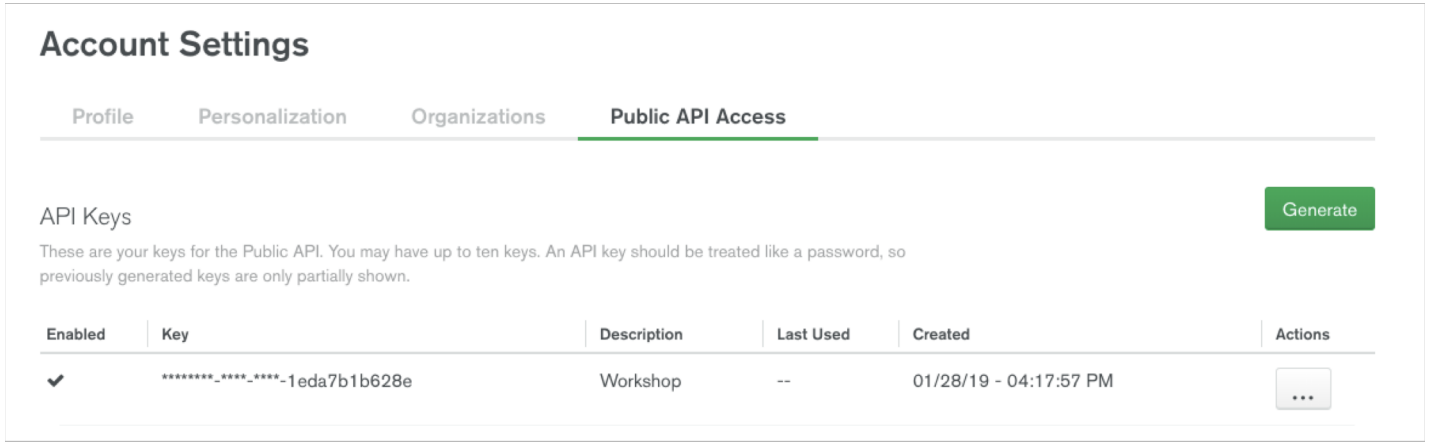


# Workshop - Dashboard

- What will we build?
  - a real-time dashboard that displays customer purchase data from the point-of-sale system of a hypothetical pizza restaurant
  - The dashboard will have an automatically updating chart of recent customer purchases and a table of the most popular toppings from the last 100 pizza orders
- Prerequisites
  - ✓ **MongoDB Atlas** Account
  - ✓ MongoDB cluster hosted in Atlas
  - A MongoDB Atlas User **API Key** for your cluster
  - **Node.js** version 6.0.0 or newer
  - A properly installed copy of stitch-cli that has been added to your system PATH
  - A local copy or clone of the stitch-examples GitHub repository
- Estimated Time to Complete: 25 minutes

# Workshop - Generate API Key

1. On the upper-right hand corner, click on your user name and select **Account**. Click on **Public API Access**.
2. In the **API Keys** section, click **Generate**.
3. Type a description and click **Generate**.
4. If prompted for two-factor authentication, enter the code and click **Verify**. Then click **Generate** again.
5. **Copy and Record the key immediately**. Atlas displays the full key one time only. You will not be able to view the full key again.
6. Record the key in a secure place. After you record the key, click **Close**.




mongoDB

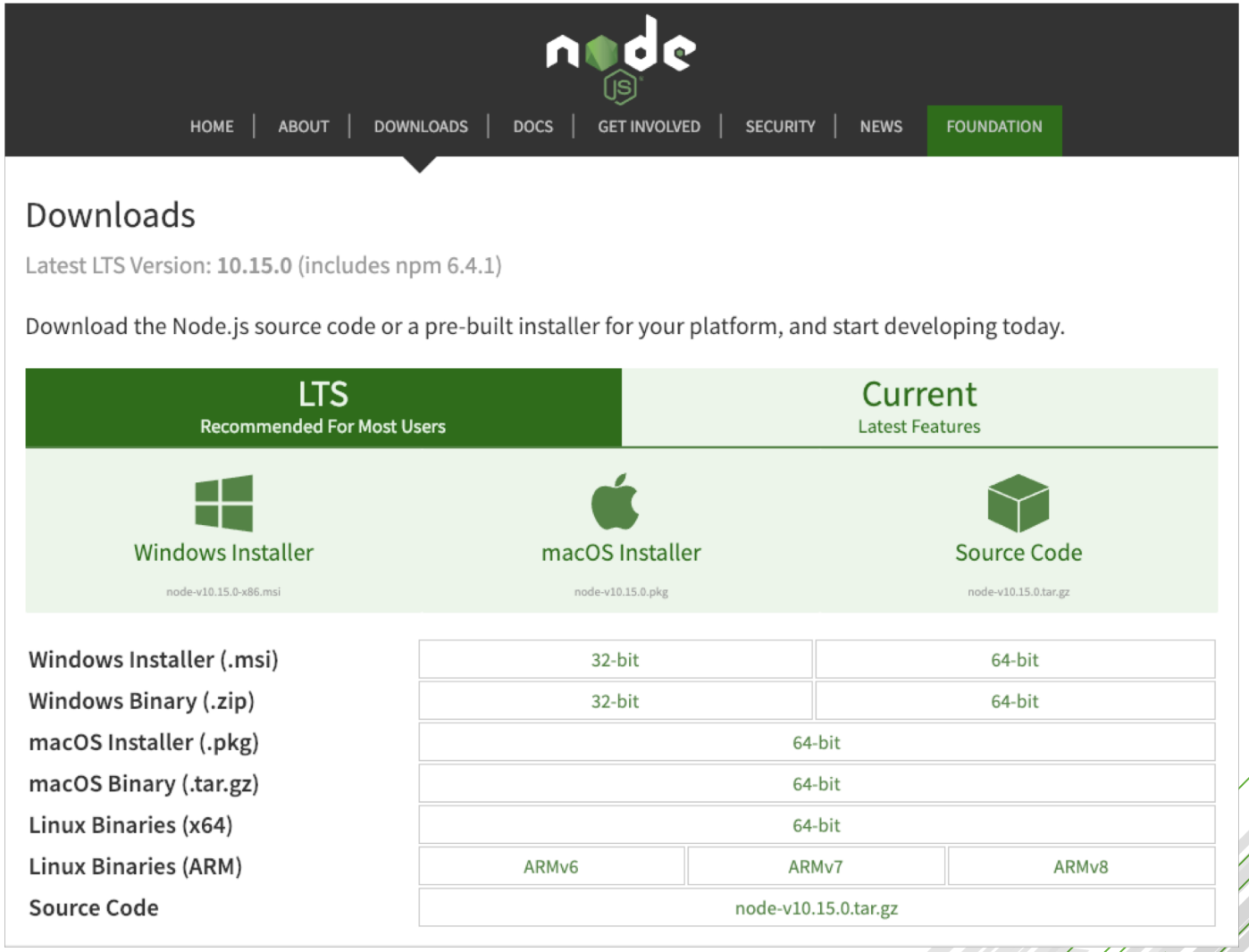
# Workshop - Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.

Download and Install Node.js:
https://nodejs.org/en/download/



mongoDB



# Workshop - Install stitch-cli

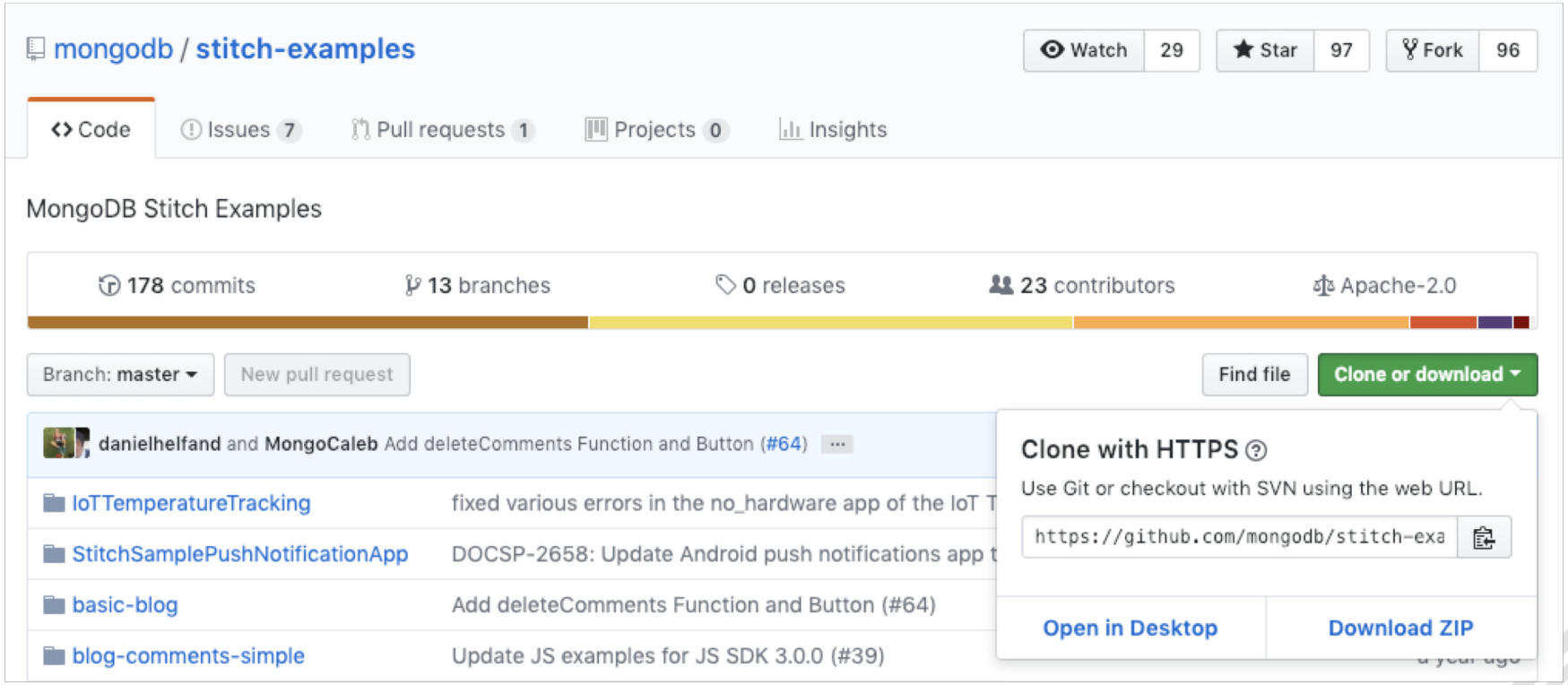
The easiest way to install **stitch-cli** is with the npm package manager. Ensure that you have Node.js installed, then run the following command in your shell:

```
npm install -g mongodb-stitch-cli
```

# Workshop - Get a Local Copy of the Stitch Examples

https://github.com/mongodb/stitch-examples/

```
git clone https://github.com/mongodb/stitch-examples.git
```




mongoDB

# Workshop - Dashboard

`dashboard/appdir/dashboard-stitch` contains configurations for the following Stitch entities:

- An **email/password authentication provider**.
- A MongoDB service that will link with your Atlas cluster. The service has custom read and write **Rules** defined for the SalesReporting. Receipts collection. The rules remove sensitive fields, such as a customer's credit card number, from read results and require specific fields to be included when inserting new documents.
- A **Stitch Function** named **salesTimeline**.
  - The function returns data from MongoDB for orders made between the provided start and end timestamps.
- A **Stitch Function** named **getPopularToppings**.
  - The function returns a sorted count of the most popular toppings from the last 100 pizza orders stored in MongoDB.
- An **HTTP service** named PizzaOrderAPI.
  - The service exposes a webhook named addOrder that adds customer order data to MongoDB when it is sent in the body of a POST request to the webhook. The webhook requires a secret query parameter to validate requests.

# Workshop - Dashboard - Step 1

## Import a New Stitch Application

1. Navigate to the Pre-Configured Application Directory

   ```
   cd dashboard/appdir/dashboard-stitch
   ```

1. Authenticate a MongoDB Atlas User

   ```
   stitch-cli login --username=<MongoDB Cloud username> --api-key=<MongoDB
   Atlas API Key>
   ```

1. Optional: Update MongoDB Atlas Cluster Name.

   Edit `dashboard-stitch/services/mongodb-atlas/config.json`

1. Import the Application Directory. Run the following command from within the dashboard-stitch directory:

   ```
   stitch-cli import
   ```

# Workshop - Dashboard - Step 2

## Create an Email/Password User

1. Open your application in the Stitch UI.
2. Select **Users** from the left-side navigation
3. Click **Add New User**.
4. Enter the following user information then click **Create**:

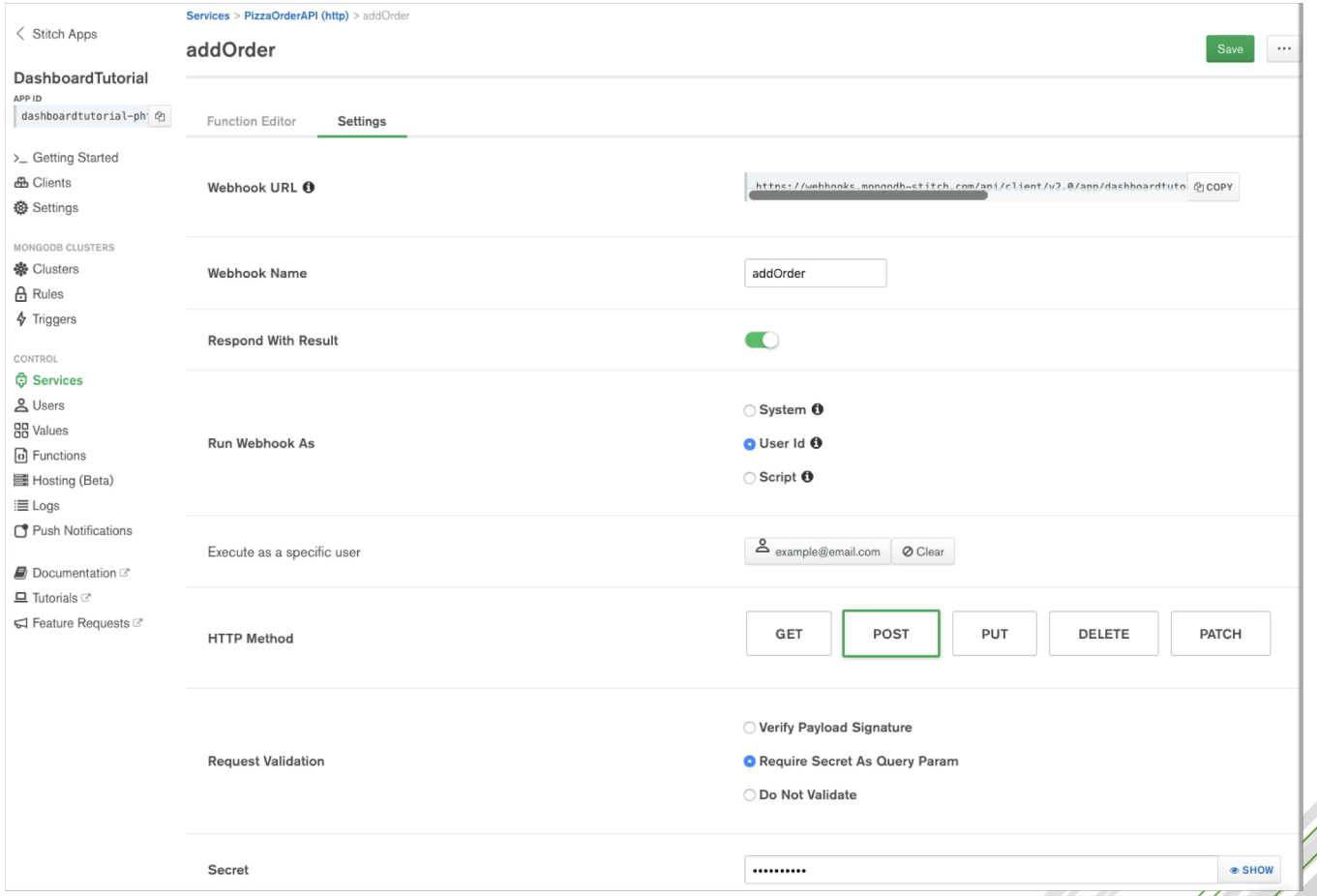
| | |
|---|---|
| Username | `example@email.com` |
| Password | `mypassword` |

# Workshop - Dashboard - Step 3

**Configure the addOrder Webhook to Run as the New User**

1. Select **Services** from the left-side navigation
2. Select the **PizzaOrderAPI** service then click the **addOrder** webhook.
3. From the **Settings** tab, set **Run Webhook As** to **User Id** then click the **Select User** button that appears.
4. Choose the email/password user that you just created from the list of users then click **Select User**.
5. Click **Save** to save your changes to the webhook configuration.

Note the values of the **Webhook URL** and **Secret**.



# Workshop - Dashboard - Client

The dashboard client code is located in the `stitch-examples/dashboard` directory. There are two files that communicate with Stitch:

## dashboard.js

This file contains the dashboard client JavaScript code. It queries data from MongoDB using the **salesTimeline** and **getPopularToppings** Functions that we imported in the previous section.

## data_generator.js

This script simulates customer orders being entered into a point-of-sale system at a pizza shop. The order data is uploaded to MongoDB Atlas by including it as the body of a POST request sent to the **addOrder** webhook.

# Workshop - Dashboard - Step 4

## Configure the Data Generation Script

1. In `stitch-examples/dashboard/data_generator.js`, find and replace `<YOUR WEBHOOK>` with the Webhook URL you got from the addOrder configuration page. Make sure to append the secret as in the following prototype:

   ```
   <webhook url>?secret=yummypizza
   ```

1. Navigate to the `stitch-examples/dashboard` directory and run the following command to install all Node.js dependencies:

   ```
   npm install
   ```

# Workshop - Dashboard - Step 5

## Connect the Dashboard Client to your Stitch Applications

1. In `stitch-examples/dashboard/dashboard.js`, find and replace `<YOUR APP ID>` with your Stitch application's App ID.

## Run the Data Generation Script `data_generator.js`

1. Run the following command in the dashboard directory to begin uploading simulated order data to your MongoDB Atlas cluster:

```
node data_generator.js
```

# Workshop - Dashboard - Step 6

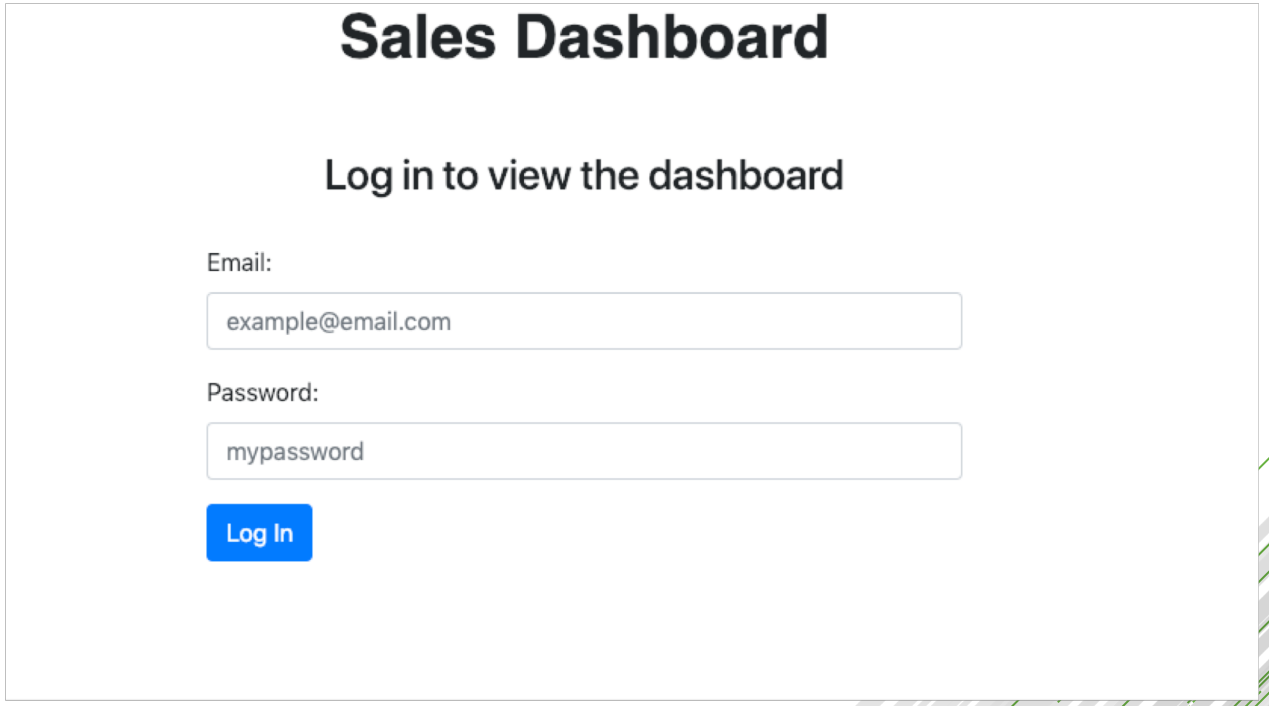
## Launch the Dashboard

From a new shell window, run the following command in the dashboard directory to begin serving the dashboard locally:

```
npm start
```

## Log In to the Dashboard

Navigate to **localhost:8080** in your browser




mongoDB

# Workshop - Dashboard - Summary

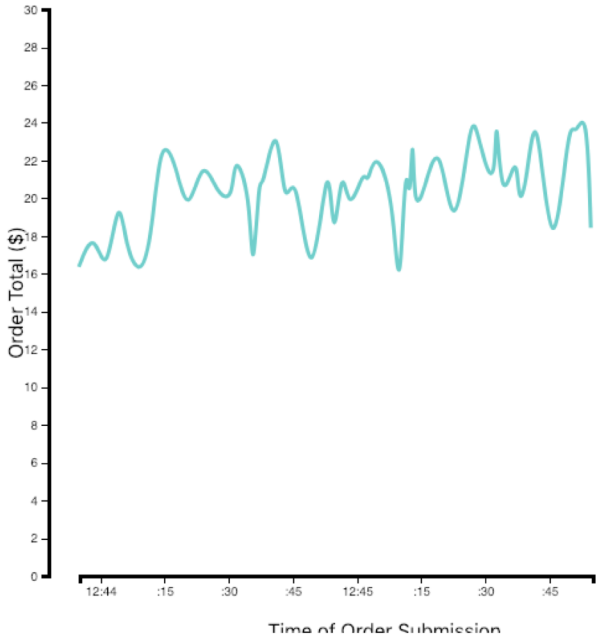
## Sales Dashboard

Logged in as: example@email.com

### Popular Toppings

| Topping | # Recent Orders |
|---|---|
| Pineapple | 14 |
| Sausage | 12 |
| Bacon | 12 |
| Spinach | 11 |
| Mushrooms | 11 |
| Pepperoni | 10 |
| Extra cheese | 9 |
| Green peppers | 9 |
| Onions | 7 |
| Black olives | 5 |

### Recent Pizza Sales




mongoDB

# Today:

Link to tutorial

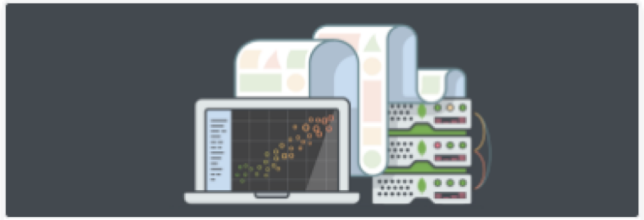
Stitch > Tutorials

## MongoDB Stitch Tutorials

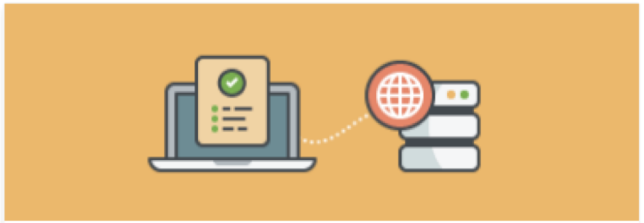
The tutorials in this section walk through creating real applications with MongoDB Stitch. Follow along to get practical experience working with Stitch, or use one of the completed applications as a starting point for your own idea.

Basic Blog

Dashboard



Temperature Tracker (IoT Integration)



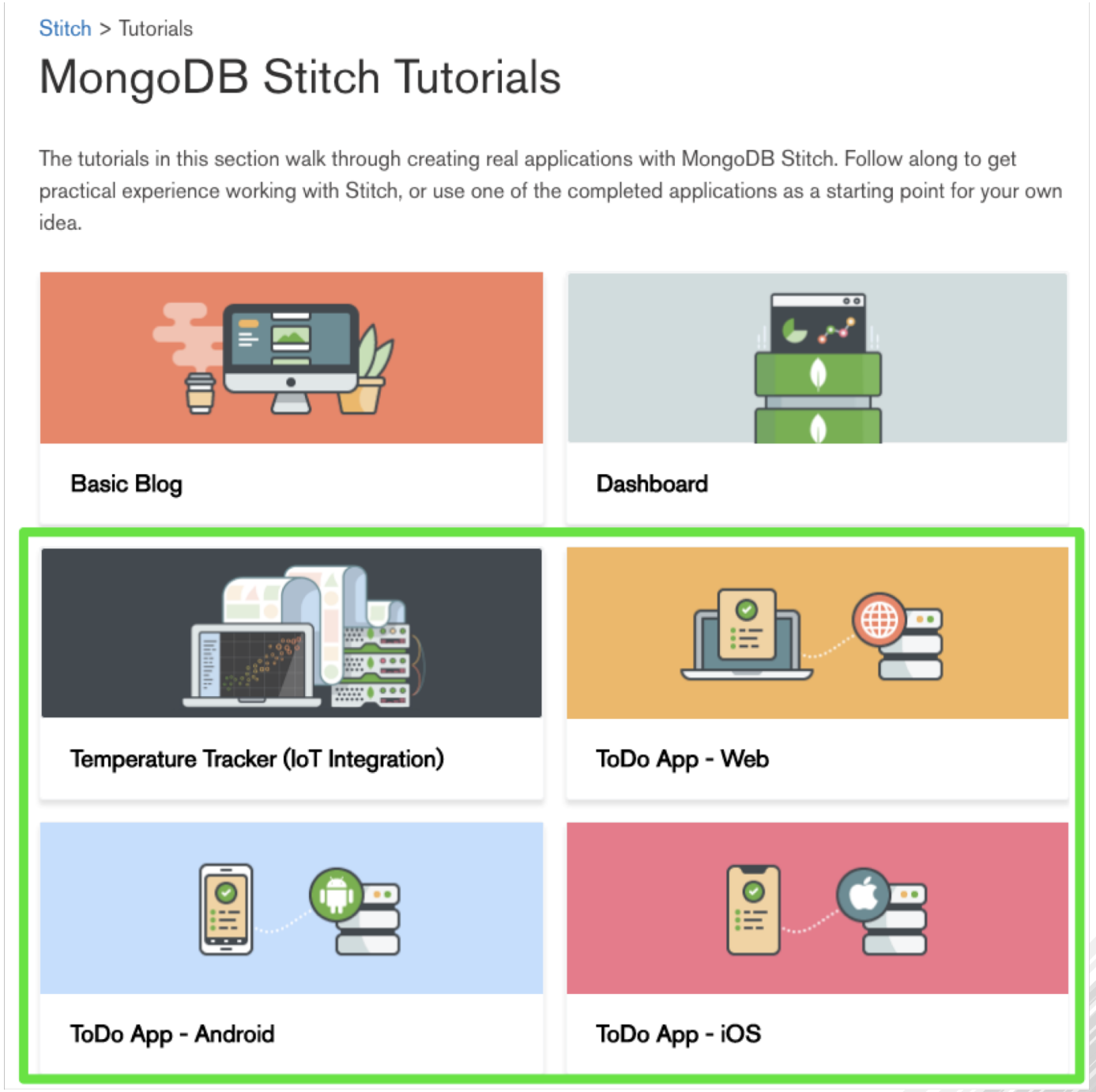
ToDo App - Web

ToDo App - Android

ToDo App - iOS

mongoDB

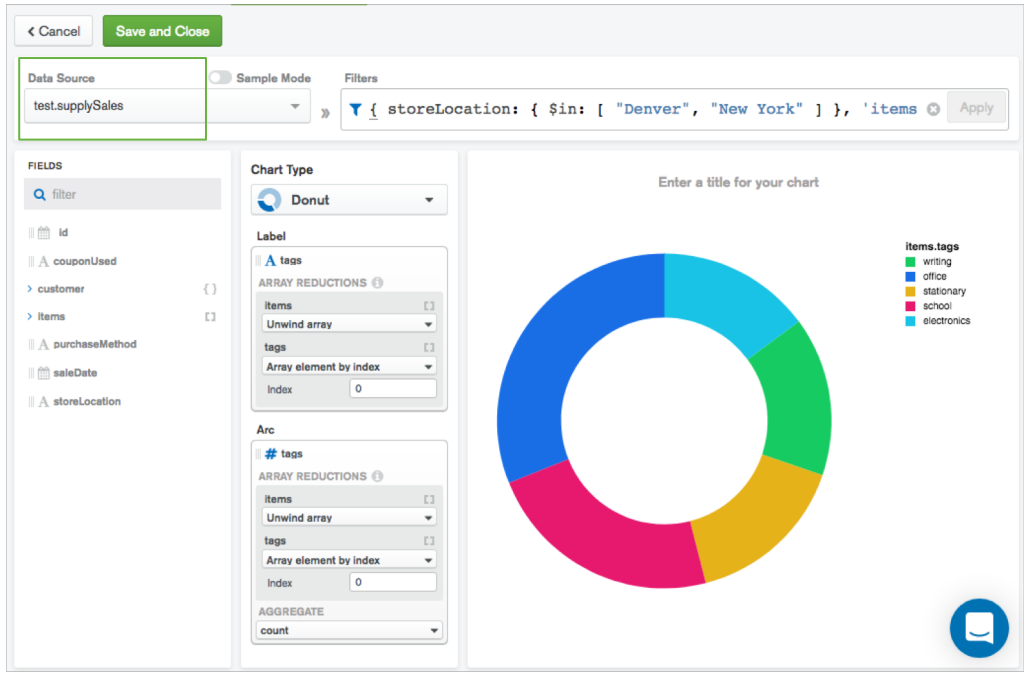
# Tomorrow:




mongoDB

# MongoDB Charts

# MongoDB Charts: Create, Visualize, Share



**Work with complex data**

**Connect to data sources securely.**
**Filter. Sample. Visualize.**

**Share dashboards and collaborate**

mongoDB

# Charts Workshop - Visualizing Movie Details

- What will we build?
  - MongoDB Charts dashboard  with a data source containing information gleaned from IMDb and Rotten Tomatoes.
- Prerequisites
  - MongoDB Atlas Account
  - MongoDB cluster hosted in Atlas
- Estimated Time to Complete: ~10 minutes

# Charts Workshop - Step 1

## Add the Data Collection as a Data Source

1. In MongoDB Atlas, click on **Charts** on the left side
2. Click the **Data Sources** tab
3. Click **New Data Source**
4. Select your Atlas Deployment in your project
5. Click **Connect**
6. Select the `movies.movies` collection
7. Click Set **Permissions**, leave the permissions as the default
8. Click **Publish Data Source**

# Charts Workshop - Step 2

## Create a New Dashboard

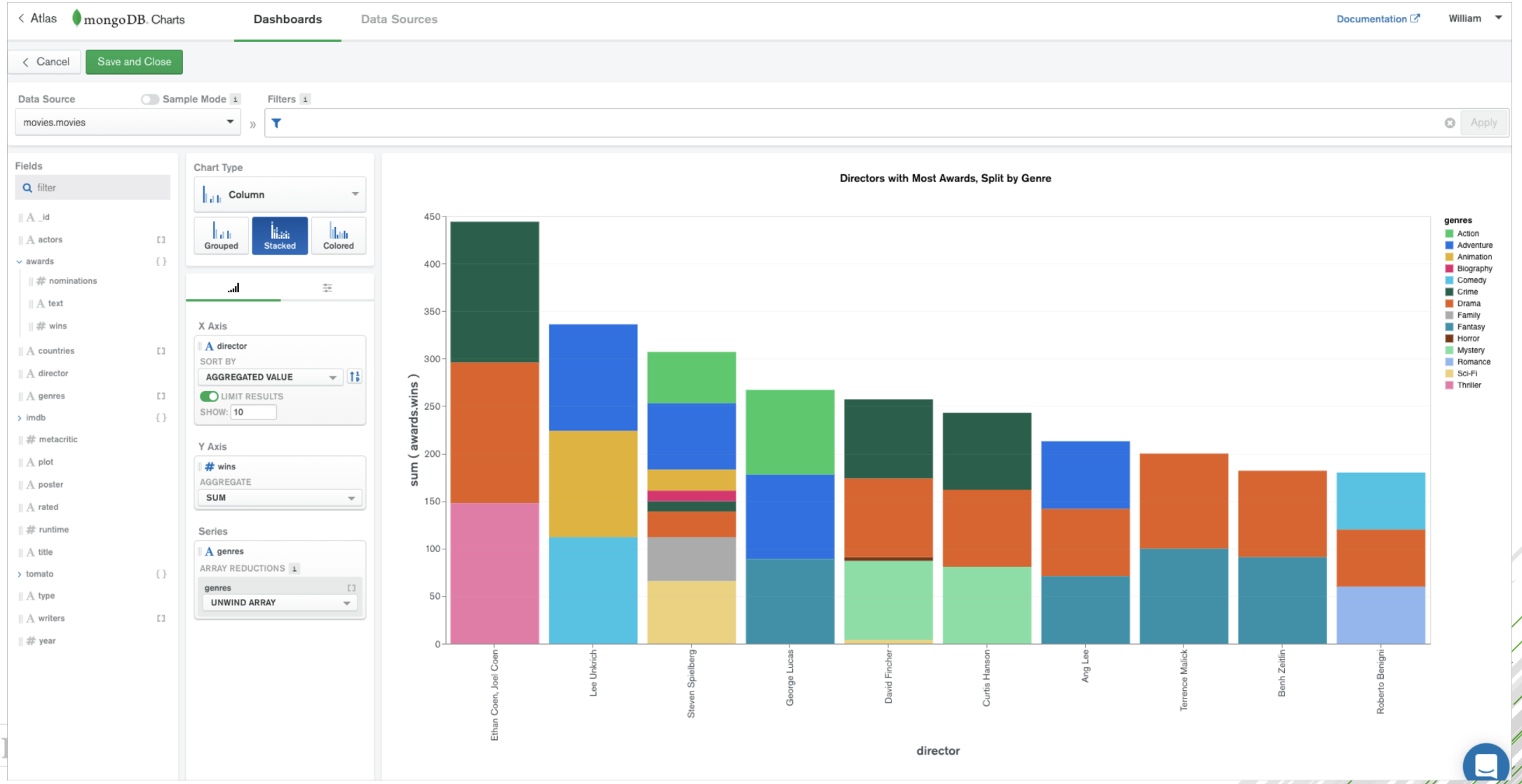
1. Click the **Dashboards** tab
2. Click the **New Dashboard** button
3. Enter the Title: `Movie Details`
4. Click **Create**

# Charts Workshop - Step 3

## Create Chart Showing Directors with the Most Awards

1. Click **Add Chart**
2. In the Data Source dropdown, select `movies.movies`
3. Select Chart Type: **Column** / **Stacked**
4. X Axis: `director`
   - Sort By: Aggregated Value, Descending
   - Limit: 10
5. Y Axis: `awards.wins`
   - Aggregate: sum
6. Series: `genres`
   - Array Reduction: Unwind Array
7. Title: `Directors with Most Awards, Split by Genre`
8. Click **Save and Close**

# Charts Workshop - Summary



# Thank You

# APPENDIX