Getting Started

Tugdual Grall

Technical Evangelist

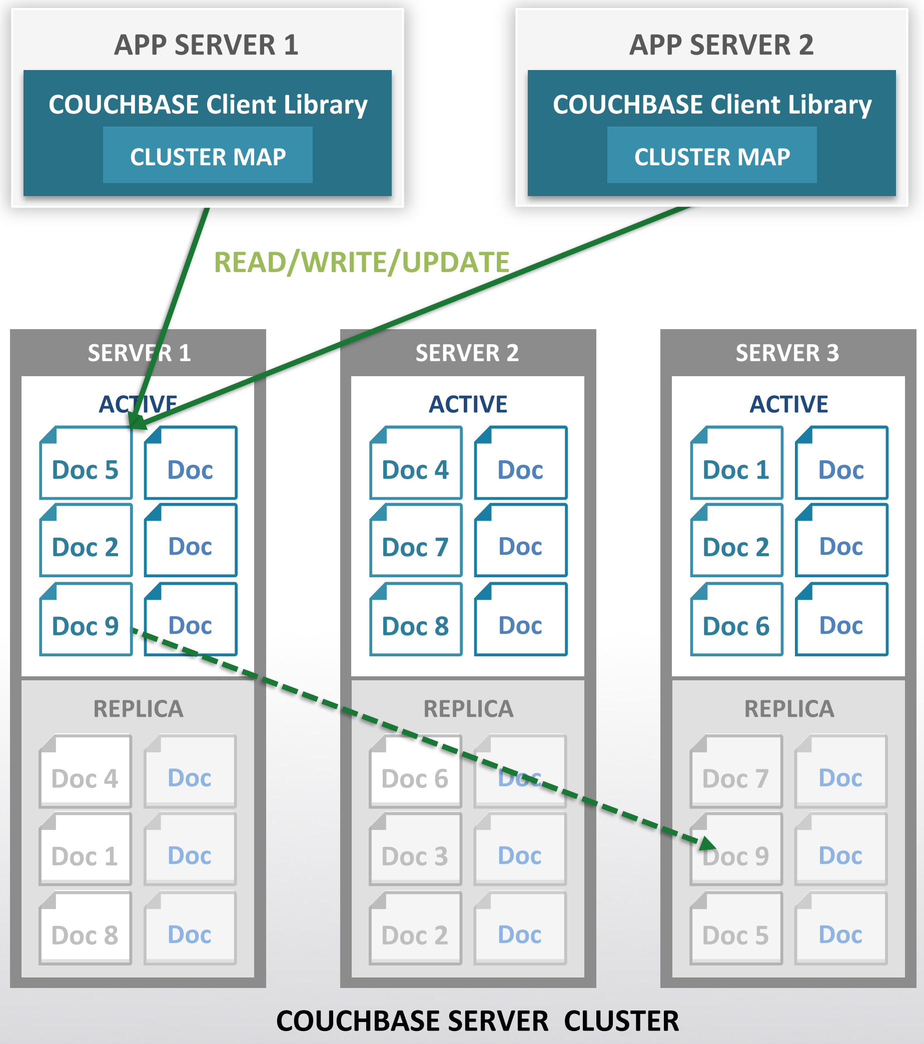
Couchbase

Dev Track Agenda

10:30 - 11:10 am	Getting Started : Installation and Core operations
11:20 - 12:00pm	Getting Started : Advanced Operations and Patterns
01:00 - 01:40 pm	N1QL: An Early Peek at Couchbase's document database query language
01:50 - 02:30 pm	Document Your World
02:40 - 03:20 pm	Indexing and Querying
03:40 - 04:20 pm	Power Techniques With Indexing
04:30 - 05:10 pm	Exploring Common Models and Integrations

Getting Started

Cluster-wide Basic Operation

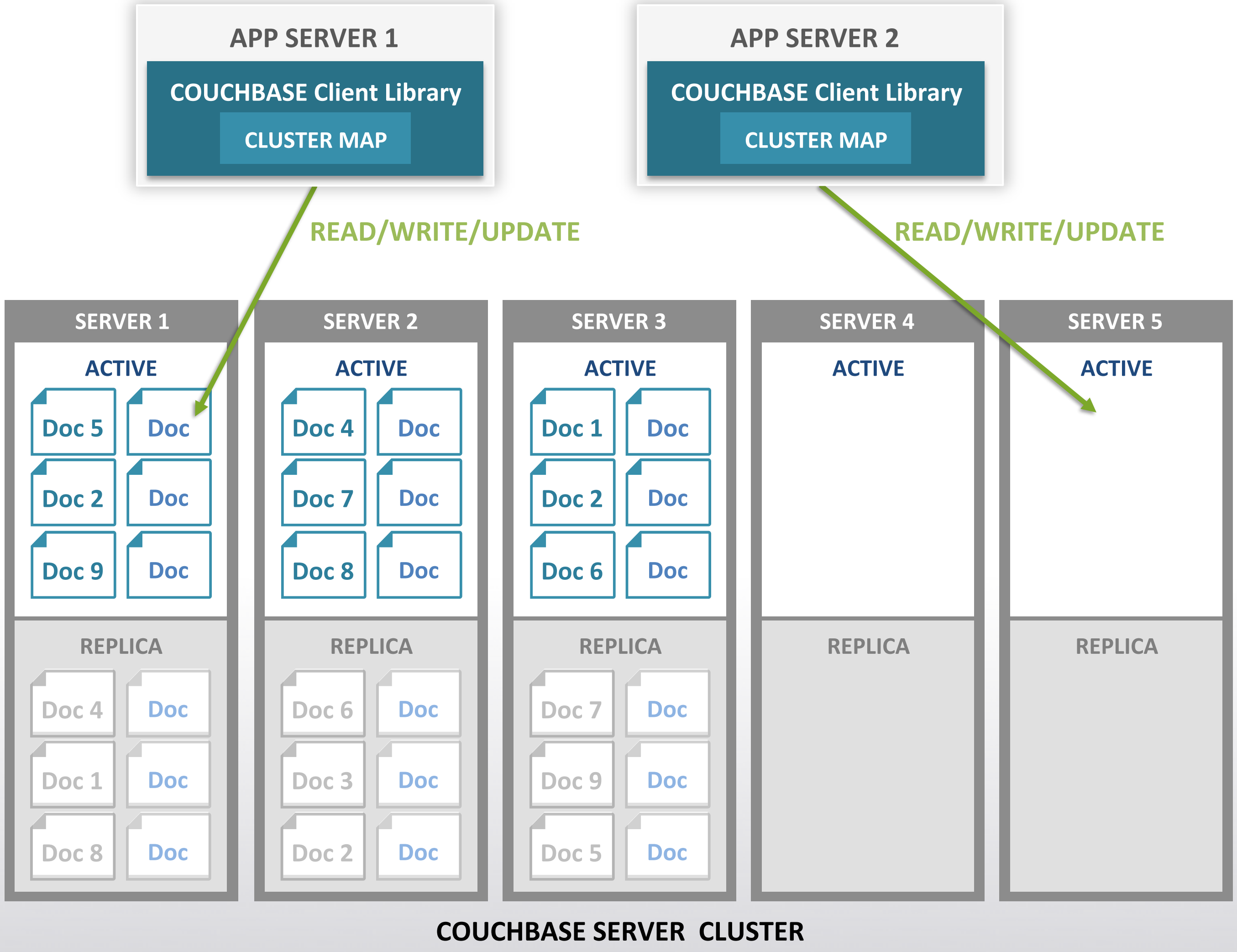


- **Docs distributed evenly across servers**
- **Each server stores both active and replica docs**
  Only one server active at a time
- **Client library provides app with simple interface to database**
- **Cluster map provides map to which server doc is on**
  App never needs to know
- **App reads, writes, updates docs**
- **Multiple app servers can access same document at same time**

Couchbase

User Configured Replica Count = 1

Add Nodes to Cluster

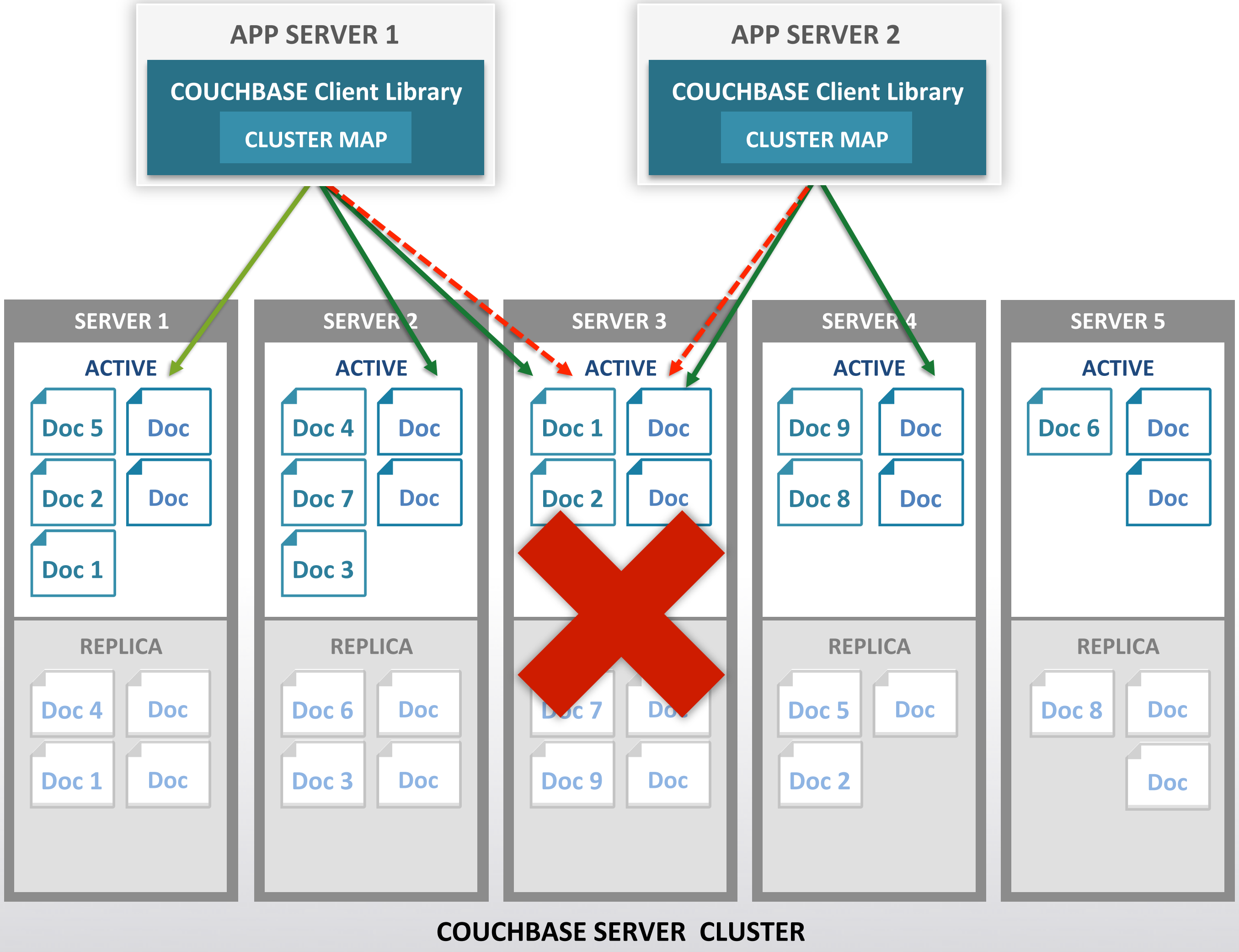


- **Two servers added One-click operation**
- **Docs automatically rebalanced across cluster**
  Even distribution of docs
  Minimum doc movement
- **Cluster map updated**
- **App database calls now distributed over larger number of servers**

User Configured Replica Count = 1

Couchbase

Fail Over Node



- **App servers accessing docs**
- **Requests to Server 3 fail**
- **Cluster detects server failed**
  Promotes replicas of docs to active
  Updates cluster map
- **Requests for docs now go to appropriate server**
- **Typically rebalance would follow**

Couchbase

User Configured Replica Count = 1

Couchbase SDK

Official SDKs

Java Microsoft® .NET Ruby C PROGRAMMING Python php

Community SDKs

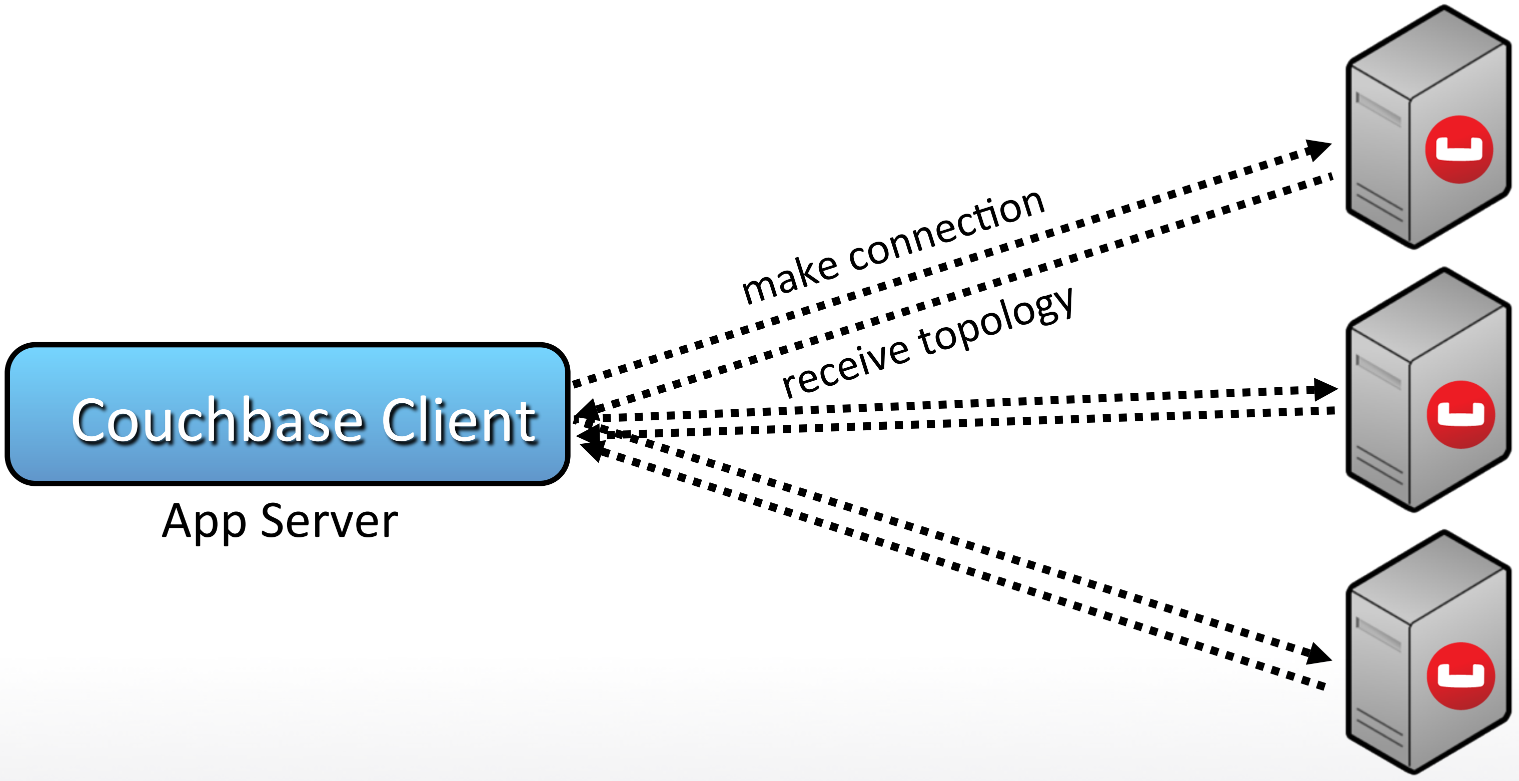
node js JRuby Go Clojure

www.couchbase.com/develop

Client Architecture Overview

- Based on the information given, the Client tries to establish an initial connection.
- Once that’s done, it connects to a streaming API (HTTP chunked).
- Cluster updates are fetched from that connection.
- Failover/Add/Remove scenarios all update the clients in near real-time – no application restarts required!
- Key/Value access is done directly against the nodes.
- For View access one of the nodes is picked out which aggregates the results.

Client Setup: Getting Cluster Configuration



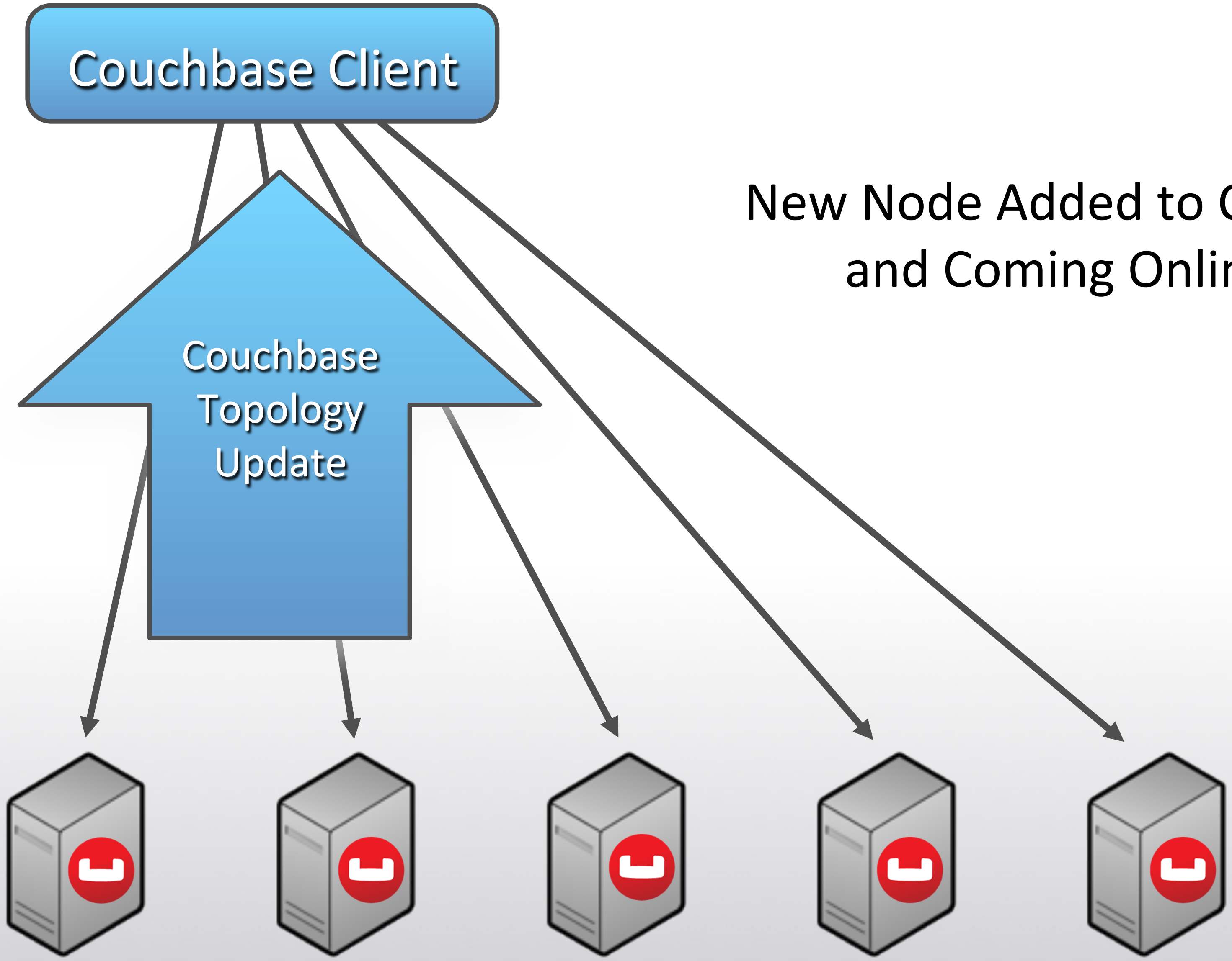
Couchbase

Bootstrap

1. `GET /pools`
2. Look for the "default" pools
3. `GET /pools/default`
4. Look for the "buckets" hash which contains the bucket list
5. `GET /pools/default/buckets`
6. Parse the list of buckets and extract the one provided by the application
7. `GET /pools/default/buckets/`

Couchbase

Client Setup: Getting Cluster Configuration



New Node Added to Cluster and Coming Online

SDK & libcouchbase Dependency

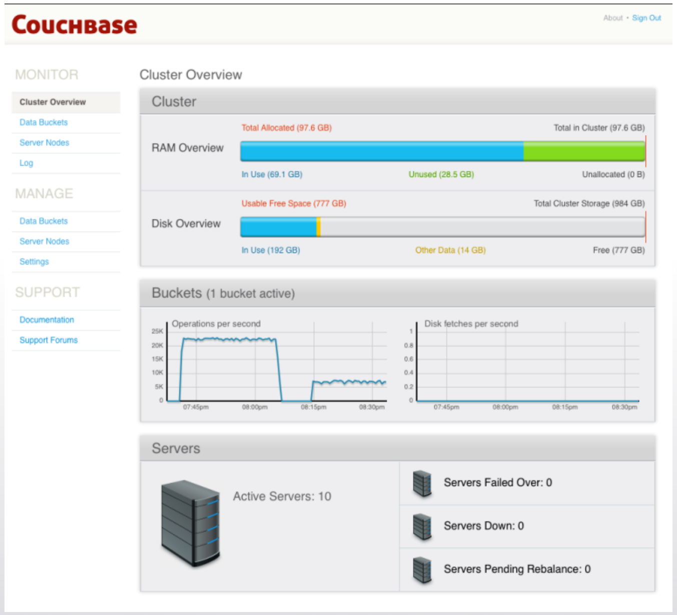
- **Java, .Net, C, Go** are native
  - The SDK does not have dependency on other language
- **Ruby, PHP, Python, Node** have dependency to **libcouchbase**
  - They are "wrapper" on the top of the C library
- **Scala, Clojure, JRuby** are using Couchbase Java SDK

Hands On

Quick Start : Couchbase Server

Download from
http://www.couchbase.com/download

Install via .rpm, .deb, .exe, or .app



Cluster Overview

Cluster

RAM Overview: Total Allocated (97.6 GB); Total in Cluster (97.6 GB); In Use (69.1 GB); Unused (28.5 GB); Unallocated (0 B)

Disk Overview: Usable Free Space (777 GB); Total Cluster Storage (984 GB); In Use (192 GB); Other Data (14 GB); Free (777 GB)

Buckets (1 bucket active)

Servers

Active Servers: 10

Servers Failed Over: 0

Servers Down: 0

Servers Pending Rebalance: 0

EASY

!

Couchbase

Quick Start : Client

1. Go to the developer page http://www.couchbase.com/develop

2. Select your language

3. Follow the "Getting Started" page

- Java
- .Net
- Ruby
- PHP
- C
- Python
- Node
- …

Couchbase

Demonstration

Installing Couchbase Server & Client

Basics: Retrieve

- **get (key)**
  - Retrieve a document
- **gets(key)**
  - Retrieve a document and the CAS value associated with the object (more on this in a bit)

Couchbase

Basics: Create, Update, Delete

- **set (key, value)**
  - Store a document, overwrites if exists
- **add (key, value)**
  - Store a document, error/exception if it already exists
- **replace (key, value)**
  - Store a document, error/exception if doesn't exist
- **delete(key)**
  - Delete the document from the system

Couchbase

Fundamentals

- Couchbase is structured as a Key-Value store: every Document has a Key and a Value.
- Key can be up to 250 characters long.
- Keys are unique, within a database (bucket), there can only be one document with a associated key.
- Keys are completely in the control of the application developer, there is no internal mechanism for key generation.
- Values can be JSON, strings, numbers, binary blobs, or a special positive atomic counter (unsigned integer).
- Values can be up to 20MB in size.

Couchbase

Dealing with keys

- You are responsible of the key
- Keys can be build from:
  - UUID
  - Atomic Counter
  - Date/TimeStamp
  - Contains Separator:
    - User:001
    - Product_XYZ

Couchbase

What about the value ?

- Couchbase stores the data the way you send them
- For example:
  - `cb.set(“mykey”, “This is my Value”);`
  - `cb.set(“mykey”, 100.2);`
  - `cb.set(“mykey”, javaObjectSerialized);`
- What about JSON?
  - `cb.set(“mykey”, "{\"msg\" : \"This is the value\"}");`
  - `cb.set(“mykey”, json.toJson( myJavaObject) );`
  - Your application deals with the JSON Document

Q&A

Next Session : Advanced Operations and Patterns

Getting Started – Part 2

Basics: Retrieve

- **get (key)**
  - Retrieve a document
- **gets(key)**
  - Retrieve a document and the CAS value associated with the object (more on this in a bit)

Basics: Create, Update, Delete

- **set (key, value)**
  - Store a document, overwrites if exists
- **add (key, value)**
  - Store a document, error/exception if it already exists
- **replace (key, value)**
  - Store a document, error/exception if doesn't exist
- **delete(key)**
  - Delete the document from the system

Couchbase

Other Options

- **Durability Requirements**
  - `PersistTo.ONE`
  - `ReplicateTo.ONE`
  - Simply add this to your method call when needed
    - `ops = cb.set(“mykey”, myObject, PersistTo.ONE);`
- **Time to Live (TTL)**
  - Used to delete the value after the specified time

    -
      ```
      cb.set(“mykey”, 30 x 24 x 60 x 60, myObject);
      // keep the value in Couchbase for 30 days.
      ```

Couchbase

Atomic Integers

Atomic Counters are a special structure in Couchbase, they are executed in order and are Positive Integer Values

- **set (key, value)**
  - Use set to initialize the counter
    - cb.set(“my_counter”, 1)
- **incr (key)**
  - Increase an atomic counter value, default by 1
    - cb.incr(“my_counter”) # now it’s 2
- **decr (key)**
  - Decrease an atomic counter value, default by 1
    - cb.decr(“my_counter”) # now it’s 1

Couchbase

Couchbase Patterns

- Atomic Counter for keys
  - Find your ID by numbers, loop on the values, …
- Lookup
  - Lookup a document/values using multiple keys
- Lists
  - Lookup pattern, using list of values
- Indices and Queries

Use a Counter

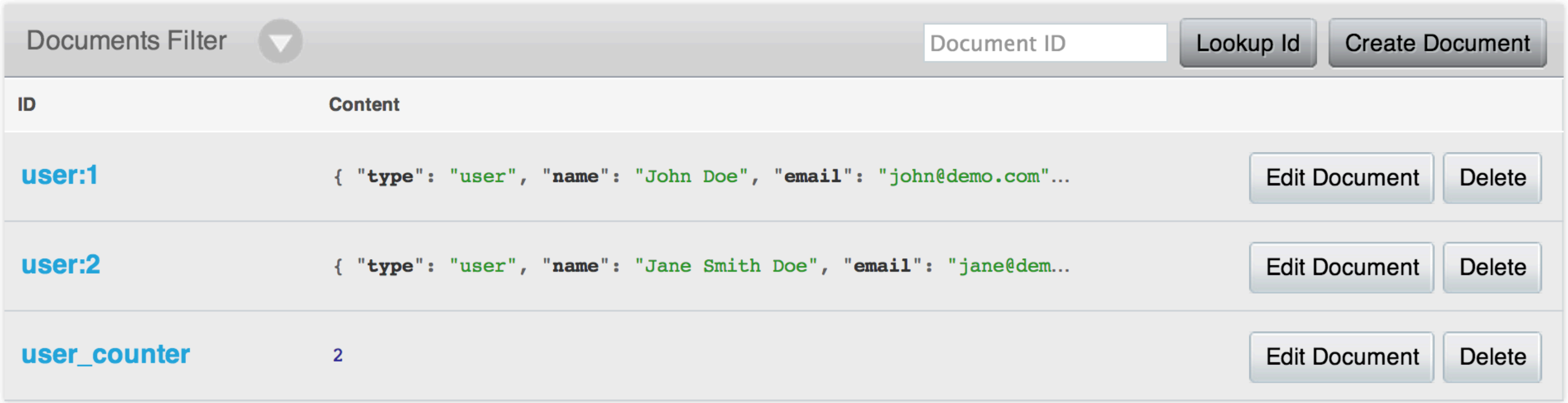
```
cb = new CouchbaseClient(uris, "default", "");
Gson json = new Gson()

// create a new User
User user = new User("John Doe", "john@demo.com", "7621387216");

long userCounter =  cb.incr("user_counter", 1, 1);
cb.set( "user:"+ userCounter, json.toJson(user) );

// create a new user;
user = new User("Jane Smith Doe", "jane@demo.com", "355662216");
userCounter =  cb.incr("user_counter", 1);
cb.set( "user:"+ userCounter, json.toJson(user) );
```

Use a Counter



ID	Content		
user:1	{ "type": "user", "name": "John Doe", "email": "john@demo.com"...	Edit Document	Delete
user:2	{ "type": "user", "name": "Jane Smith Doe", "email": "jane@dem...	Edit Document	Delete
user_counter	2	Edit Document	Delete

Couchbase

Lookup

```
cb = new CouchbaseClient(uris, "default", "");
Gson json = new Gson()

// create a new User
User user = new User("John Doe", "john@demo.com", "7621387216");

long userCounter =  cb.incr("user_counter", 1);
String key = "user:"+ userCounter;
cb.set(key, json.toJson(user) );
// create another key for lookup
cb.set(“email:john@demo.com”, key);

// Find by email
String keyToUser = cb.get(“email:john@demo.com”);
Object user = cb.get(keyToUser);
```

Couchbase

Agile Model Development

Simple Example in Ruby

```
# example.rb
require “./user.rb”

u1 = User.new({
   :email => robin@couchbase.com,
   :name => “Robin Johnson”,
   :title => “Developer Advocate”,
   :twitter => “@rbin”
})

u1.save
```

```
# user.rb
require “rubygems”
require “couchbase”

class User
  attr_accessor  :name, :email, :title, :twitter

  def initialize(attr = {})
     attr.each do |name, value|
        setter = "#{name}=”
        next unless respond_to?(setter)
        send(setter, value)
      end
  end

  def save
     client = Couchbase.bucket
     client.set(@email.downcase, self.to_json)
  end
end
```

Couchbase

Add Lookup Class Method

```
# example.rb
require “./user.rb”

u1 = User.new({
   :email => robin@couchbase.com,
   :name => “Robin Johnson”,
   :title => “Developer Advocate”,
   :twitter => “@rbin”
})

u1.save

u1 = User.find_by_email(“robin@couchbase.com”)

if u1
  puts “User Found!”
  puts u1.inspect
else
  puts “User Not Registered!”
end
```

```
# user.rb
require “rubygems”
require “couchbase”

class User
attr_accessor  :name, :email, :title, :twitter

def initialize(attr = {}) ... end

def save
   c = Couchbase.bucket
   c.set(@email.downcase, self.to_json)
end

def self.find_by_email(email)
   c = Couchbase.bucket
   doc = c.get(email.downcase)
   return doc ? User.new(doc) : nil
end

end
```

Agile Model Development

```
# example.rb
require “./user.rb”

u1 = User.find_by_email(“robin@couchbase.com”)

if u1
   u1.fb_id = “682829292”
   u1.fb_token = “f02jdjd020d8373730djd02”
   u1.save
else
   # create user
end
```

```
# user.rb
require “rubygems”
require “couchbase”

class User
attr_accessor  :name, :email, :title, :twitter
attr_accessor :fb_id, :fb_token

def initialize(attr = {}) ... end

def save ... end

def self.find_by_email(email)
   c = Couchbase.bucket
   doc = c.get(email.downcase)
    return doc ? User.new(doc) : nil
end

end
```

Concurrency & more

Compare and Swap

Optimistic Concurrency in a Distributed System

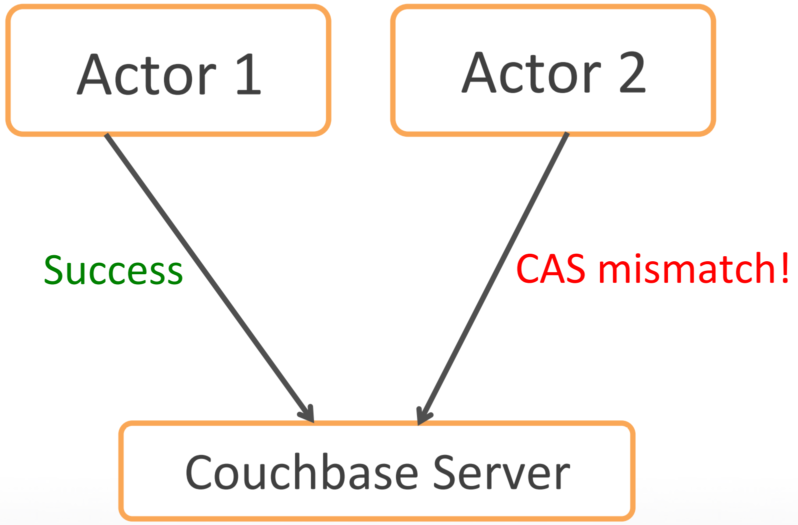
```
# actors.rb

c = Couchbase.bucket
c.set("mydoc", { :myvalue => nil }

doc1, flags, cas = c.get("mydoc",
    :extended => true)

#
c.set ("mydoc", { "myvalue": true }, :cas => cas)

# will fail because cas has changed
c.set ("mydoc", { "myvalue": true }, :cas => cas)
```



Couchbase

Check Status before Saving

```
OperationStatus ops = null;
int numberOfAttemptsLeft = 5;

do {
   CASValue listOfTagWithCas =  cb.gets(KEY);
   if (listOfTagWithCas == null) {
      cb.set(KEY, tag).getStatus();
      return;
   } else {
      ops = cb.append(listOfTagWithCas.getCas(), KEY, ","+ tag).getStatus();
      if (ops.isSuccess()) {
        return;
      }
   }

   numberOfAttemptsLeft--;
} while (numberOfAttemptsLeft > 0);


if (ops != null && !ops.isSuccess()) {
   throw new Exception("Failed to update item '"+ KEY+"' too many times, giving up!");
}
```

Couchbase

Replica Read

- Read the replica when you cannot reach the node responsible of the data

```
try  {
  value = (String)cb.get(key);
  System.out.println("Master node read");

} catch (Exception e ) {
  value = (String)cb.getFromReplica(key);
  System.out.println("Doing a Replica Read");

}
```

Couchbase

Conclusion

- Couchbase Server is installed
- Couchbase Client SDK is installed
- Core operations to save and get the data

- What's next?
  - How do you design your data? Document Design
  - How do you find your data? Indexing and Querying
  - Do even more… Common Models and Integration

Couchbase

Q&A

Next Session : Advanced Operation and Sample Applications

Couchbase