Couchbase Developer Training: Java

Raghavan N. Srinivas
Developer Advocate

Couchbase

Objective

- A lecture and Hands-on based program
  - To get started on programming with Couchbase using Java
  - To learn about the basic operations, asynchronous and advanced operations
  - To learn about secondary indexing with views and queries (a Couchbase Server 2.0 feature)
  - To enhance applications with views and queries
  - To learn some troubleshooting tips and tricks

Speaker Introduction

- Architect and Evangelist working with developers
- Speaker at JavaOne, RSA conferences, Sun Tech Days, JUGs and other developer conferences
- Taught undergrad and grad courses
- Technology Evangelist at Sun Microsystems for 10+ years
- Still trying to understand and work more effectively on Java and distributed systems
- Couchbase Developer Advocate working with Java and Ruby developers
- Philosophy: "*Better to have an unanswered question than a unquestioned answer*"

Schedule Day One

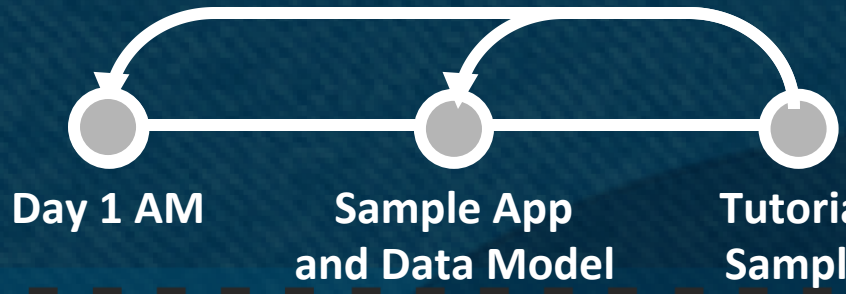
- Morning
  - Introduction and High Level Architecture
  - Getting Started
  - Asynchronous and Advanced operations
  - Using JSON in Couchbase
  - Document Design and Beer Data
- Afternoon
  - Sample App. – Intro. and Data Model
  - Tutorial Sample Exercise

SCHEDULE

(DAY 1 MORNING)

SCHEDULE

(DAY 1 AFTERNOON)

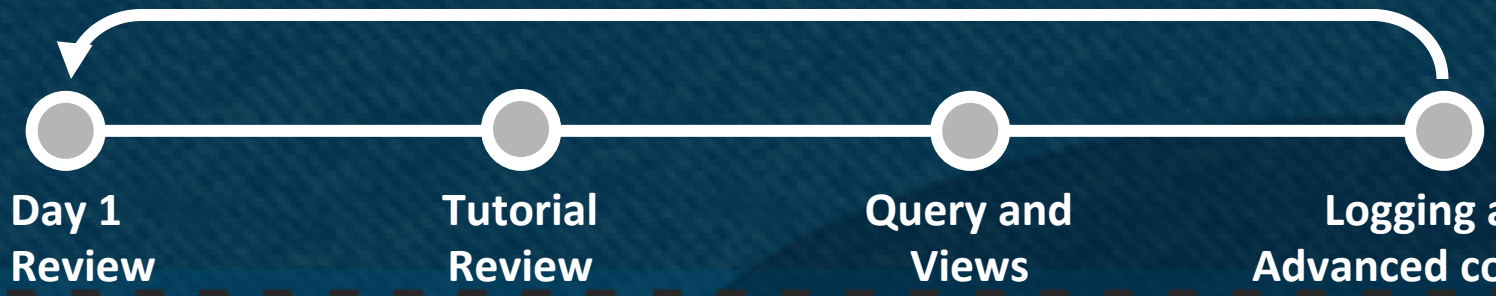


Schedule Day Two

- Morning
  - Quick Review of Day One
  - Tutorial Sample exercises review
  - Introduction to Query and Views
  - Logging and other Advanced concepts
- Afternoon
  - Exercises on Views
  - Enhancement of Tutorial for Views
  - Sample app. walkthrough and enhancement
  - Final Review, Roadmap and conclusions

SCHEDULE

(DAY 2 MORNING)



SCHEDULE

(DAY 2 AFTERNOON)

Views Exercises

Tutorial for Views

Sample App. enhancement

Final Review and Conclusions

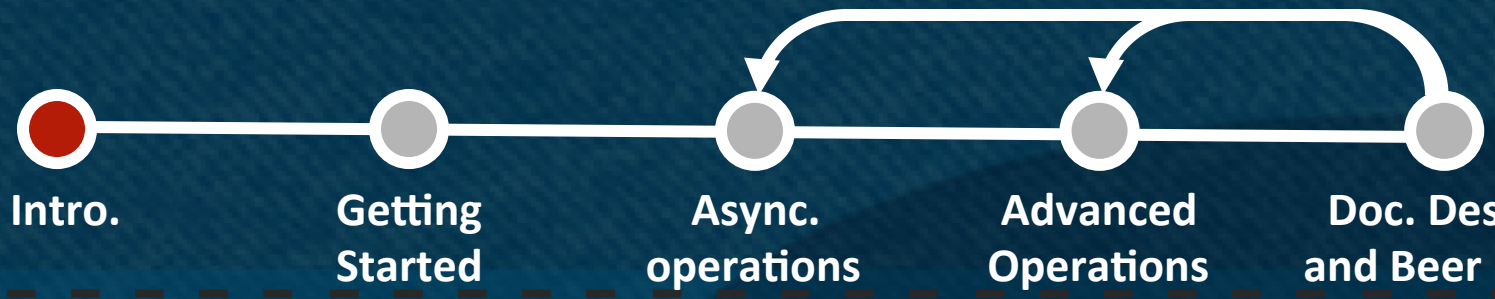
Software Requirements

- You will need to install the following software on your laptops:
  - Web Browser
  - Java 6
  - SSH Client (if planning to use Amazon EC2 or other cloud provider)
  - Tomcat 7 (for the web based exercise)
  - Eclipse (Java EE version with JSP support) or your favorite IDE and the M2E Plugin from **within Eclipse** **http://download.eclipse.org/technology/m2e/milestones/1.1**
  - Couchbase Java SDK (http://www.couchbase.com/develop/java/next)
  - PDF Viewer

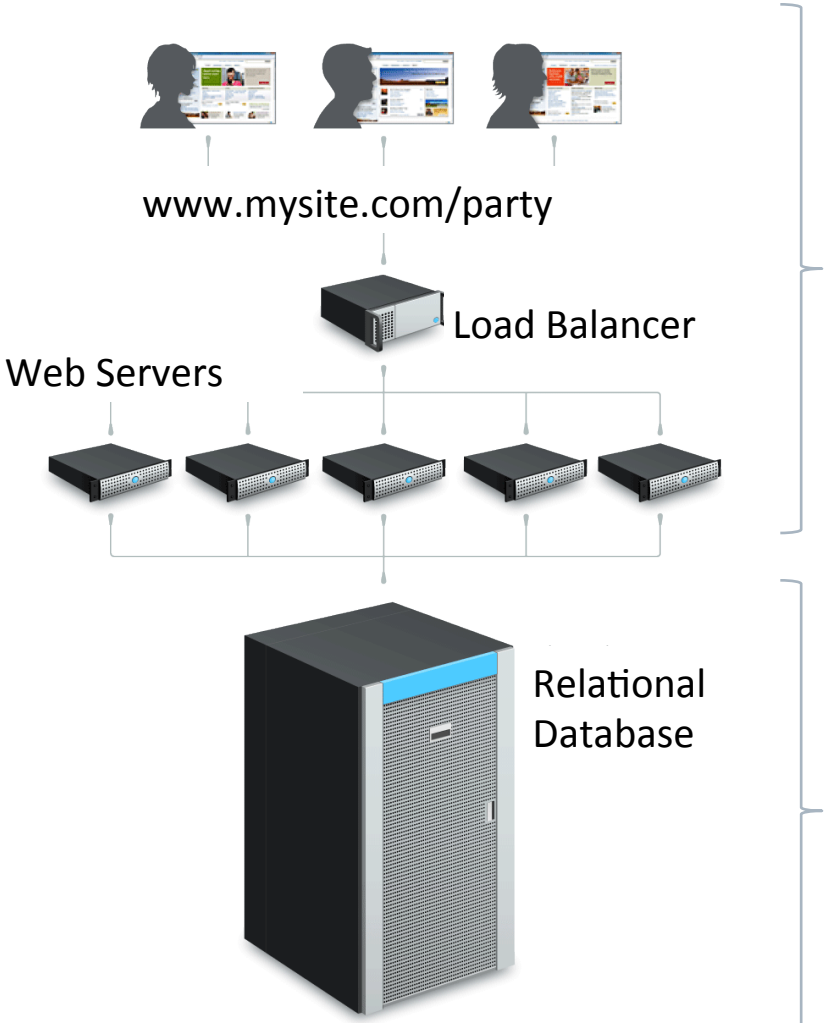
Overview of Training Material/Exercises

- After installing the prerequisite software
  - Unzip CouchbaseJavaTraining_v1.zip (appropriate version will be provided) into HOL_ROOT directory
  - Couchbase client JARs and dependencies are provided in the JARs subdirectory (for convenience)
  - Outline is in CouchbaseJavaOutline.pdf
  - Exercise statements are in CouchbaseJavaPresentation.pdf
  - Lab manual (with sketchy instructions) are in CouchbaseJavaManual.pdf
  - Solutions to exercises (sometimes partial) are in the Exercises sub directory
  - Exercises are sequential in nature (i.e. exercises depend on successful execution of earlier exercises)

HIGH LEVEL ARCHITECTURE



Distributed Application Design

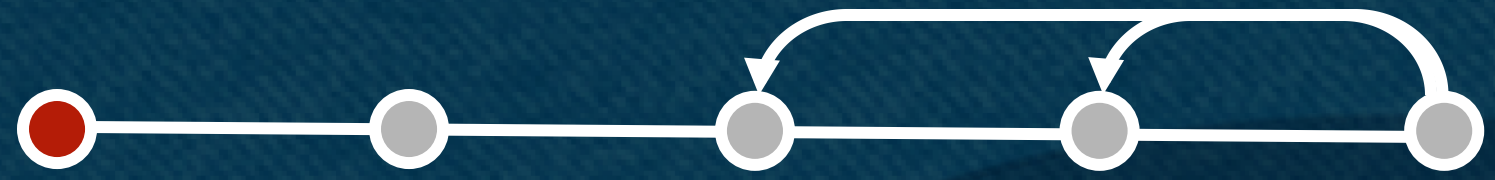


- Expensive and disruptive sharding
- Doesn't perform at Web Scale

Eight Fallacies of Distributed Computing

- The Network is reliable
- Latency is zero
- Bandwidth is infinite
- The Network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The Network is homogenous

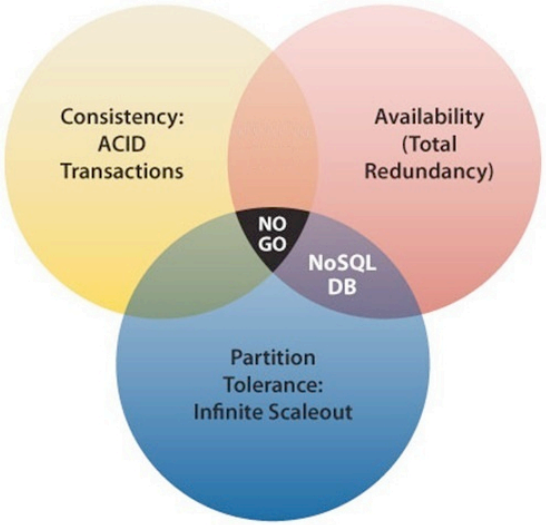
A BIT OF NOSQL



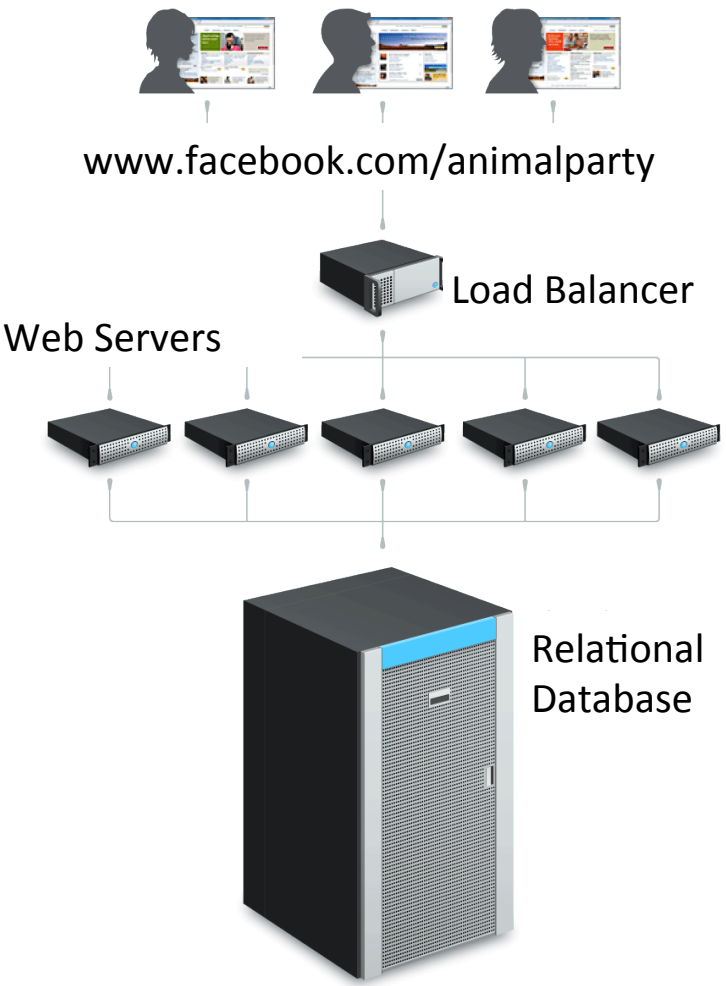
CAP Theorem

The CAP theorem states that *it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:*

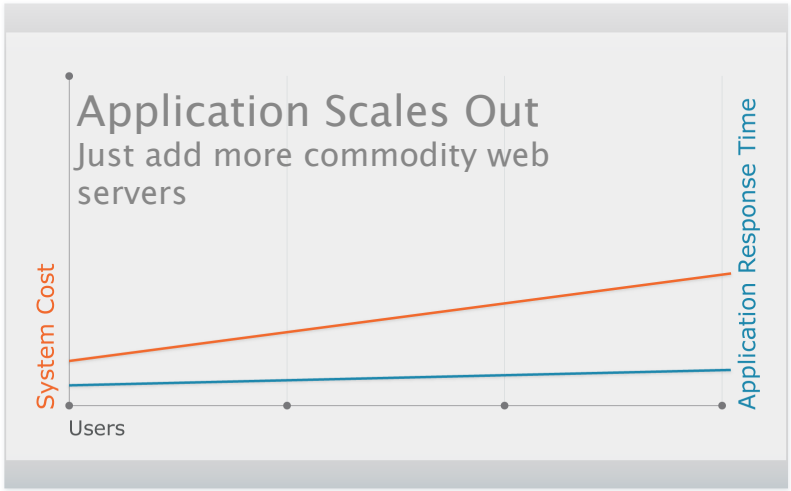
- Consistency means that each node/client always has the same view of the data,
- Availability means that all clients can always read and write,
- Partition tolerance means that the system works well across physical network partitions.

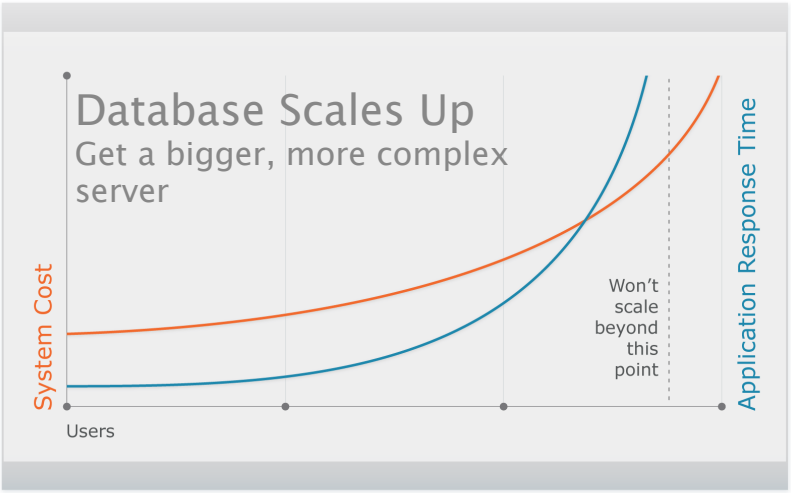


Web Application Architecture and Performance



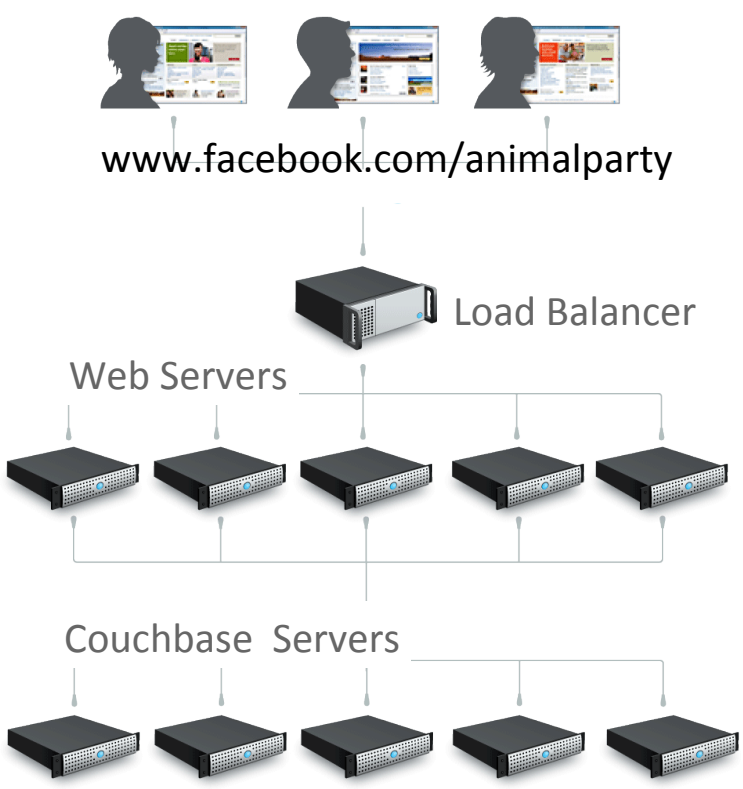
- Expensive and disruptive sharding
- Doesn't perform at Web Scale



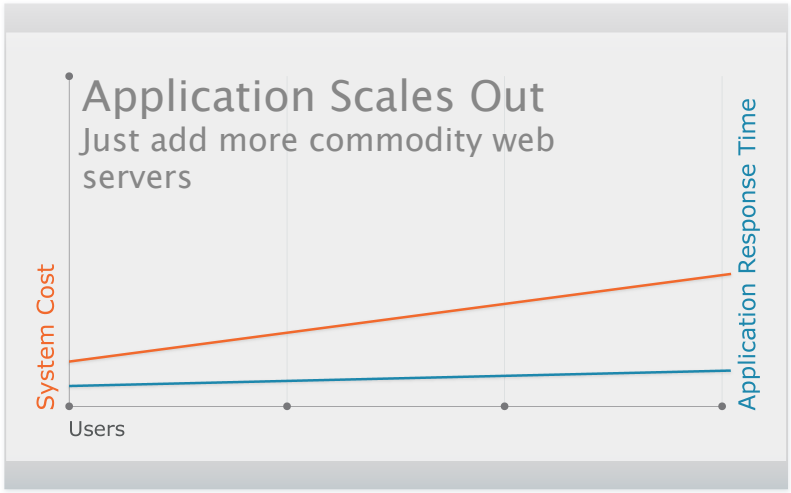


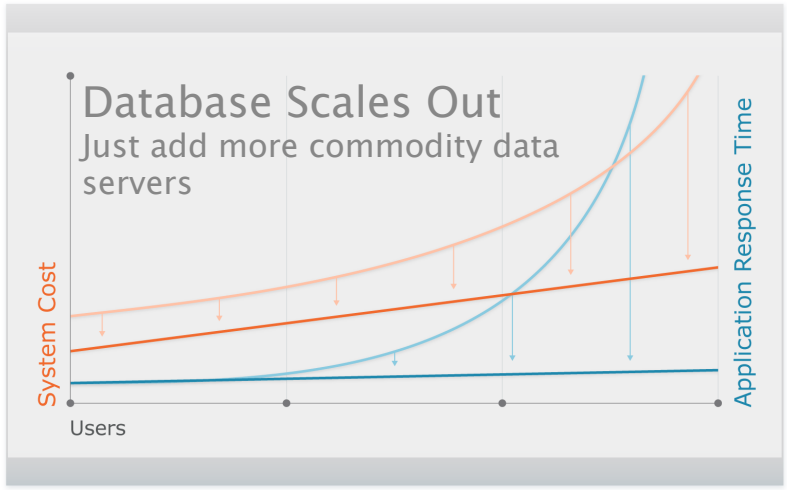
Couchbase data layer scales like application logic tier

Data layer now scales with linear cost and constant performance.



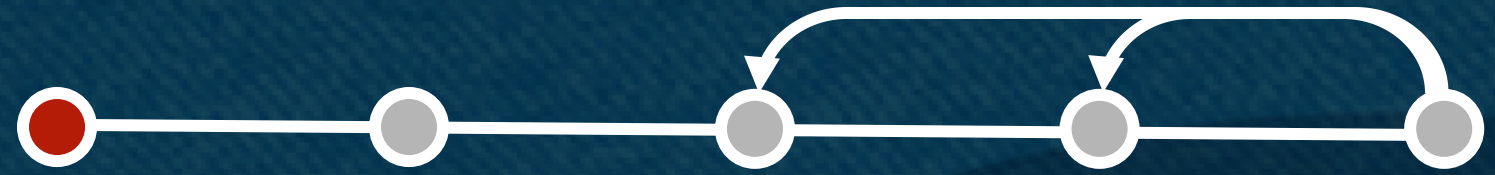
Horizontally scalable, schema-less, auto-sharding, high-performance at Web Scale





Scaling out flattens the cost and performance curves.

WHAT IS COUCHBASE



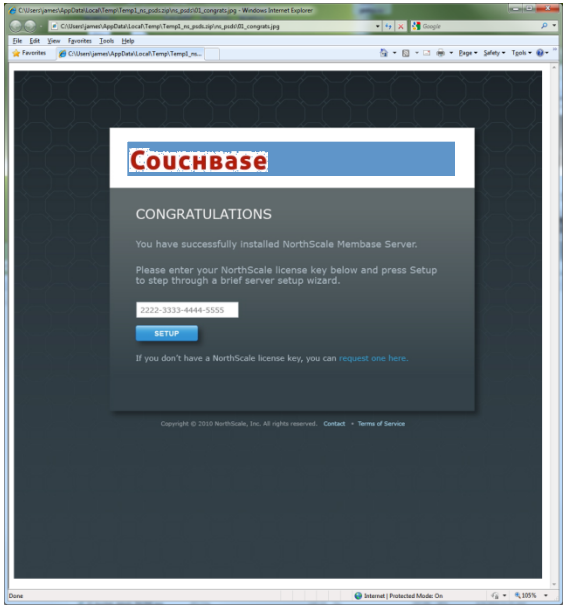
What is Couchbase?

- Multitenant Document (JSON-based) Store
- Distributed
- Fault Tolerant
  - Persistent
  - Replicated
- Schema-Free

Qualities

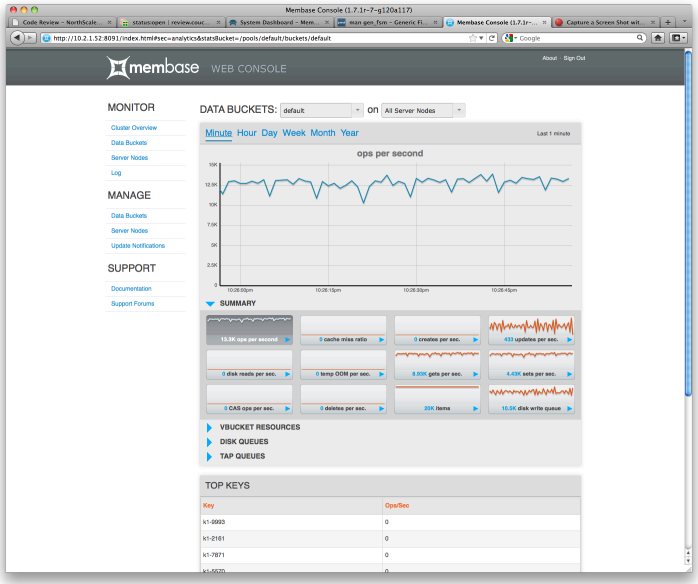
- Simple
  - memcached API
- Uniformly FAST
  - Sub-millisecond response times
  - Predictable scalability
- Elastic
  - Easy scalable

Couchbase Server is Simple, Fast, Elastic



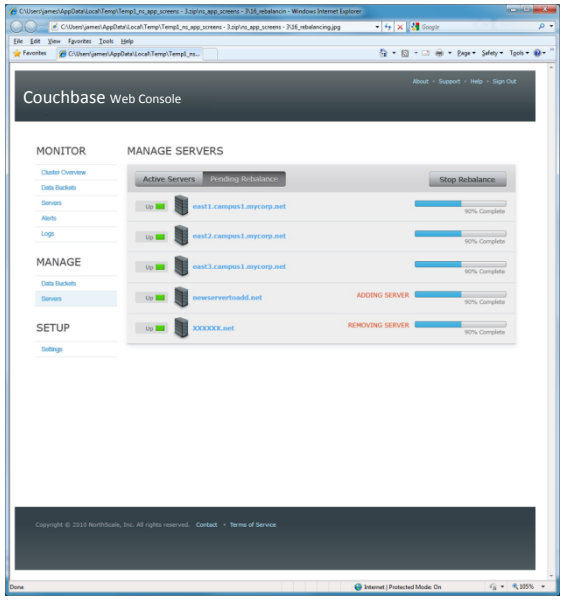
- Five minutes or less to a working cluster
  - Downloads for Windows, Linux and OSX
  - Start with a single node
  - One button press joins nodes to a cluster
- Easy to develop against
  - Just SET and GET – no schema required
  - Drop it in. 10,000+ existing applications already "speak Couchbase" (via memcached)
  - Practically every language and application framework is supported, out of the box
- Easy to manage
  - One-click failover and cluster rebalancing
  - Graphical and programmatic interfaces
  - Configurable alerting

Couchbase Server is Simple, Fast, Elastic



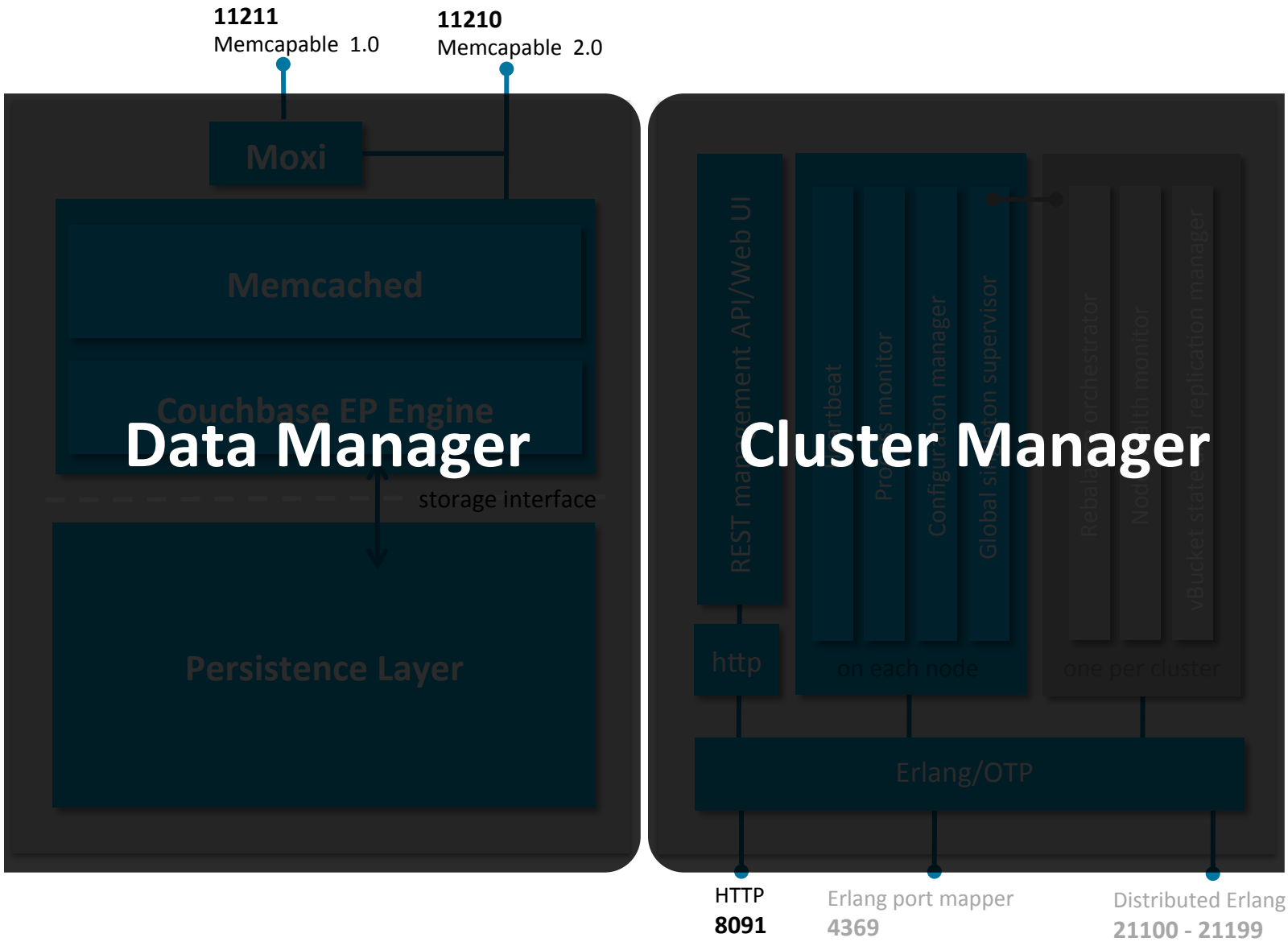
- Predictable
  - "Never keep an application waiting"
  - Quasi-deterministic latency and throughput
- Low latency
  - Built-in Memcached technology
  - Auto-migration of hot data to lowest latency storage technology (RAM, SSD, Disk)
  - Selectable write behavior – asynchronous, synchronous (on replication, persistence)
- High throughput
  - Multi-threaded
  - Low lock contention
  - Asynchronous wherever possible
  - Automatic write de-duplication

Couchbase Server is Simple, Fast, Elastic

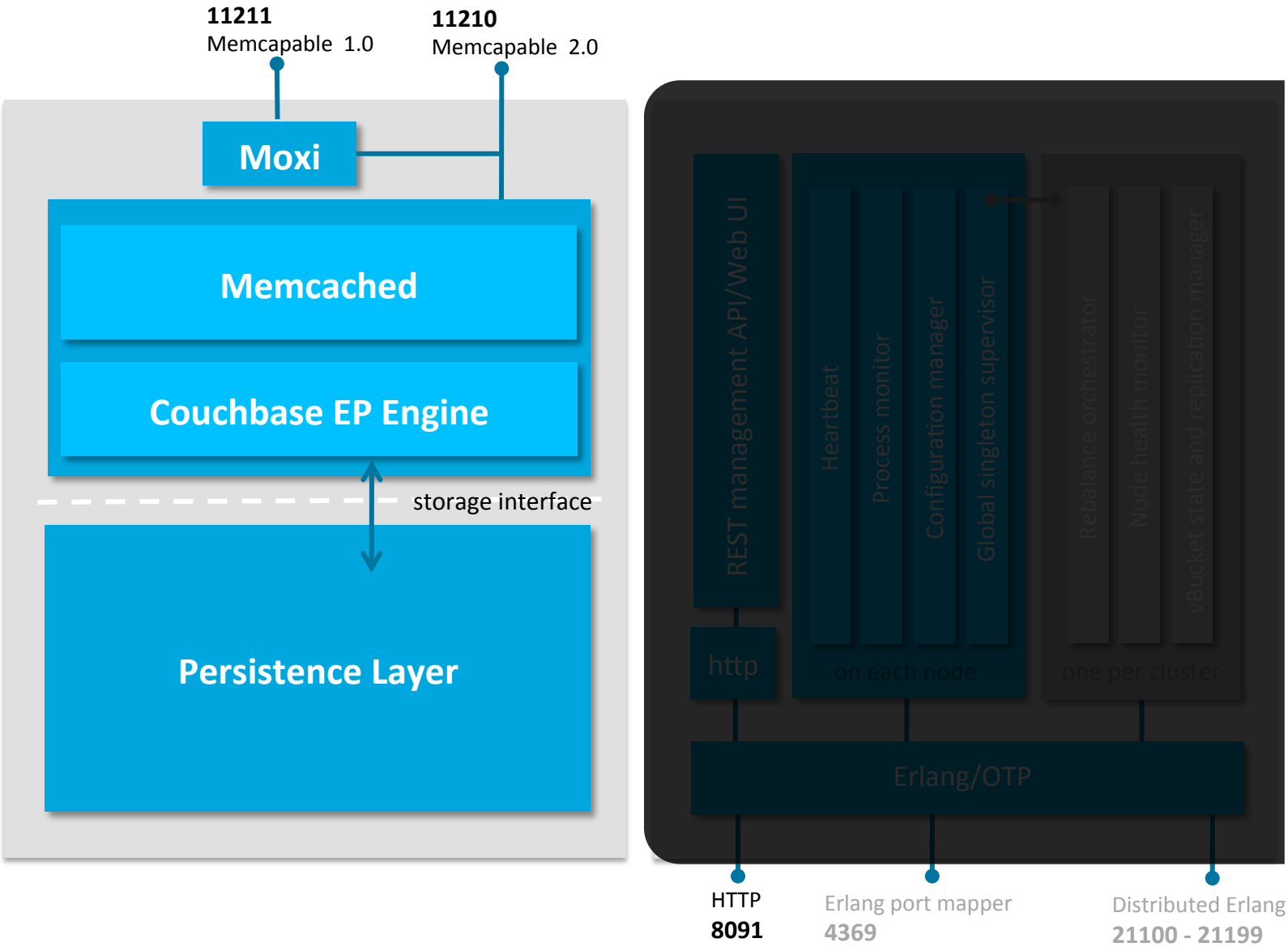


- Zero-downtime elasticity
  - Spread I/O and data across commodity servers (or VMs)
  - Consistent performance with linear cost
  - Dynamic rebalancing of a live cluster
- All nodes are created equal
  - No special case nodes
  - Clone to grow
- Extensible
  - Change feeds
  - Real-time map-reduce
  - RESTful interface for management

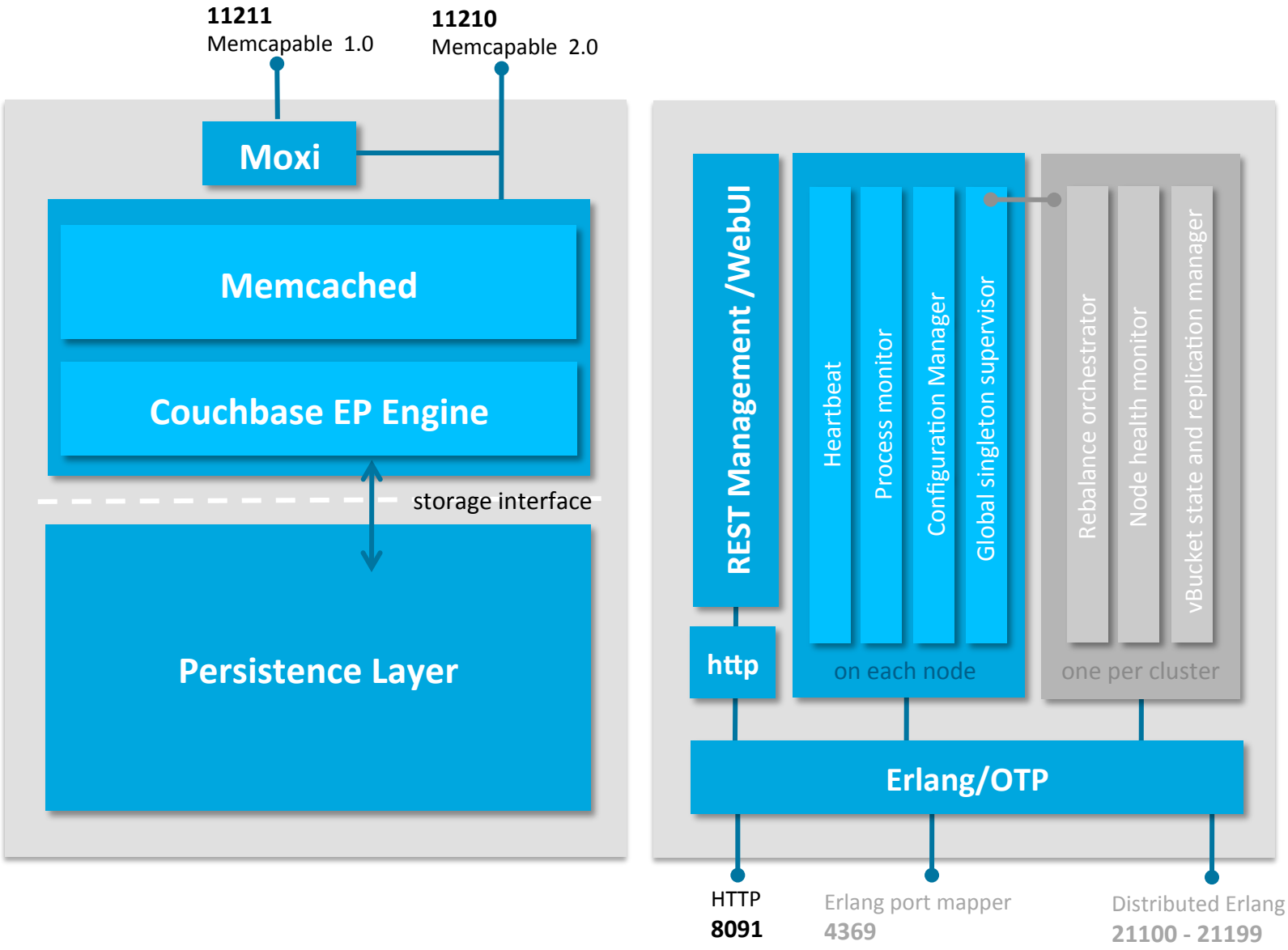
Couchbase Server 1.8 Architecture



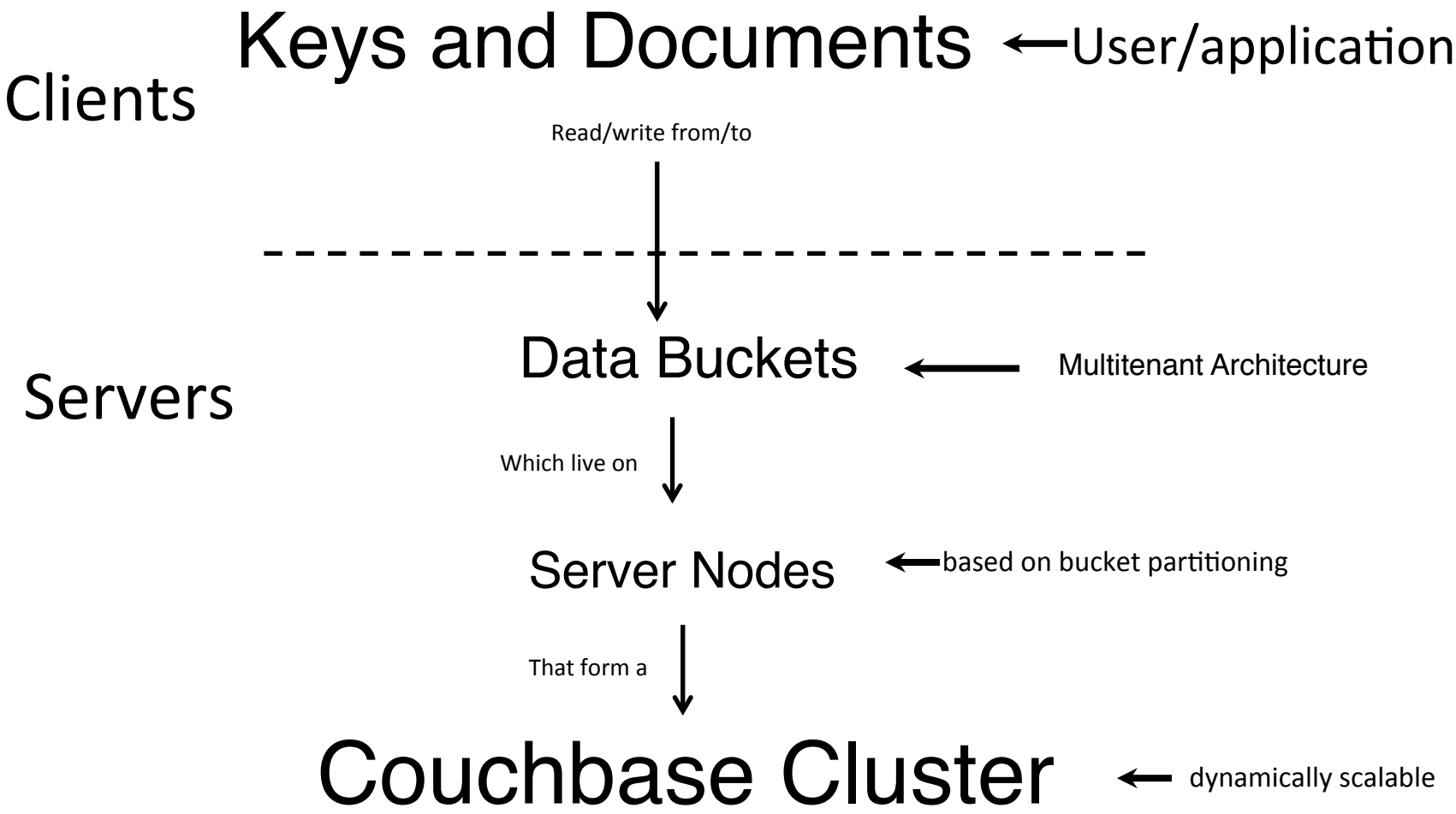
Couchbase Server 1.8 Architecture



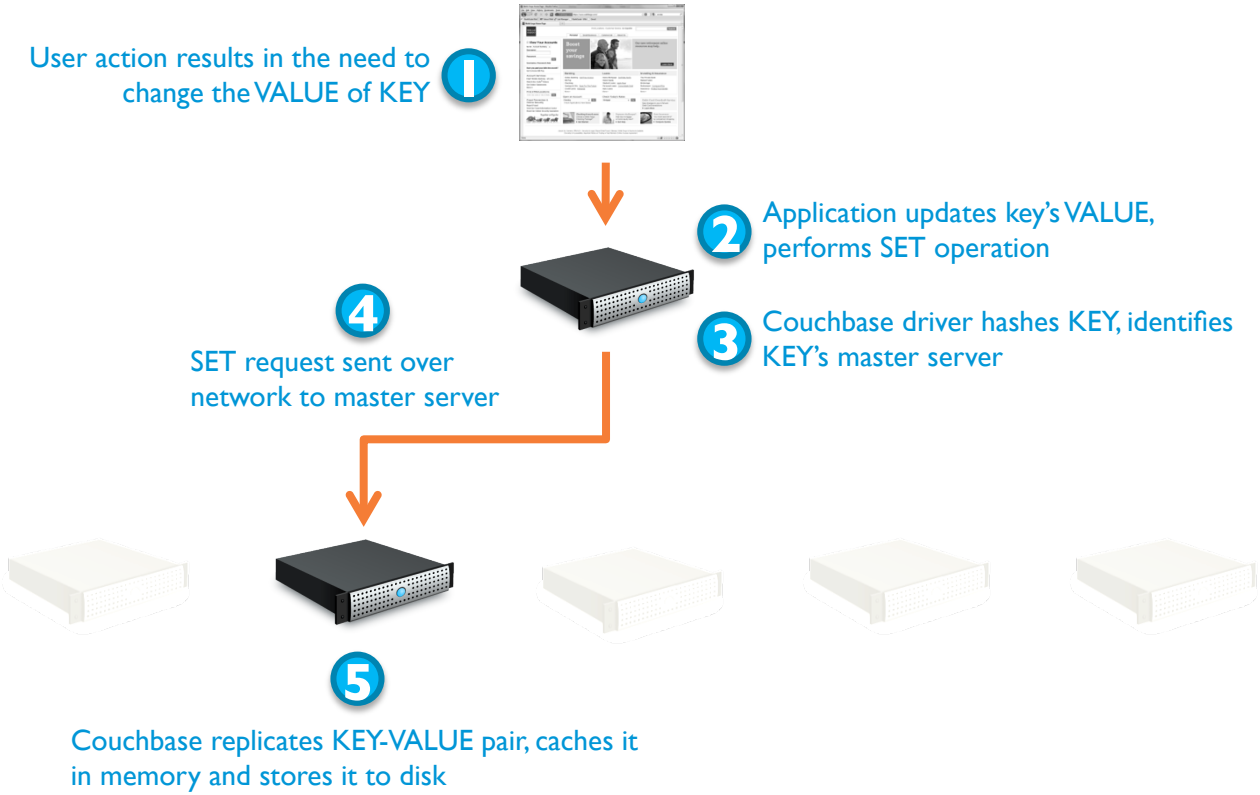
Component Architecture



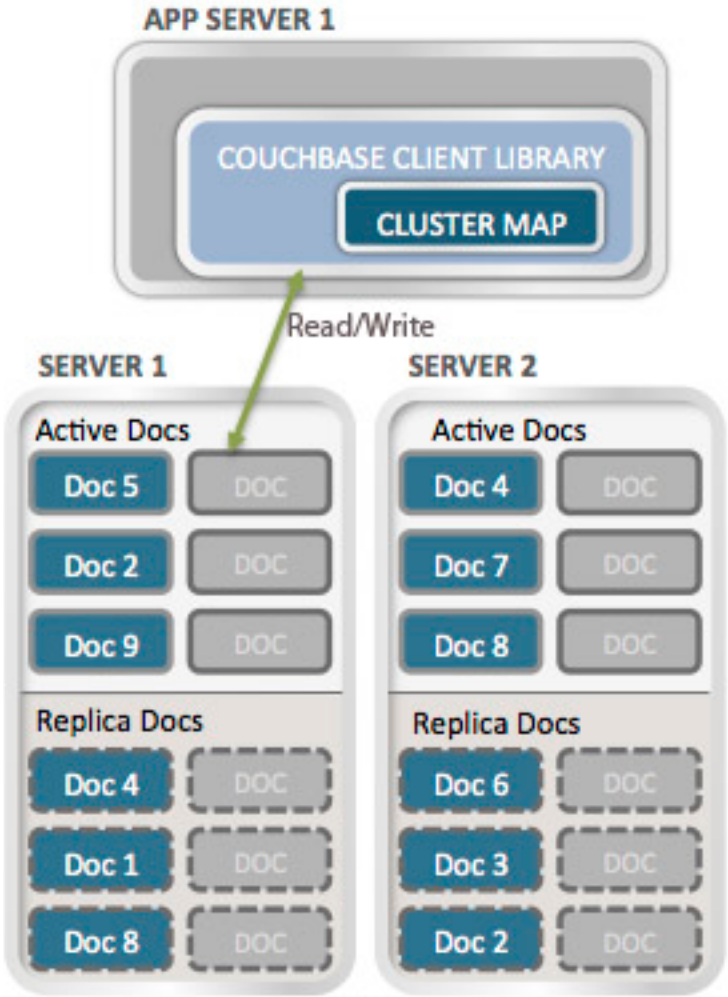
Key Concepts



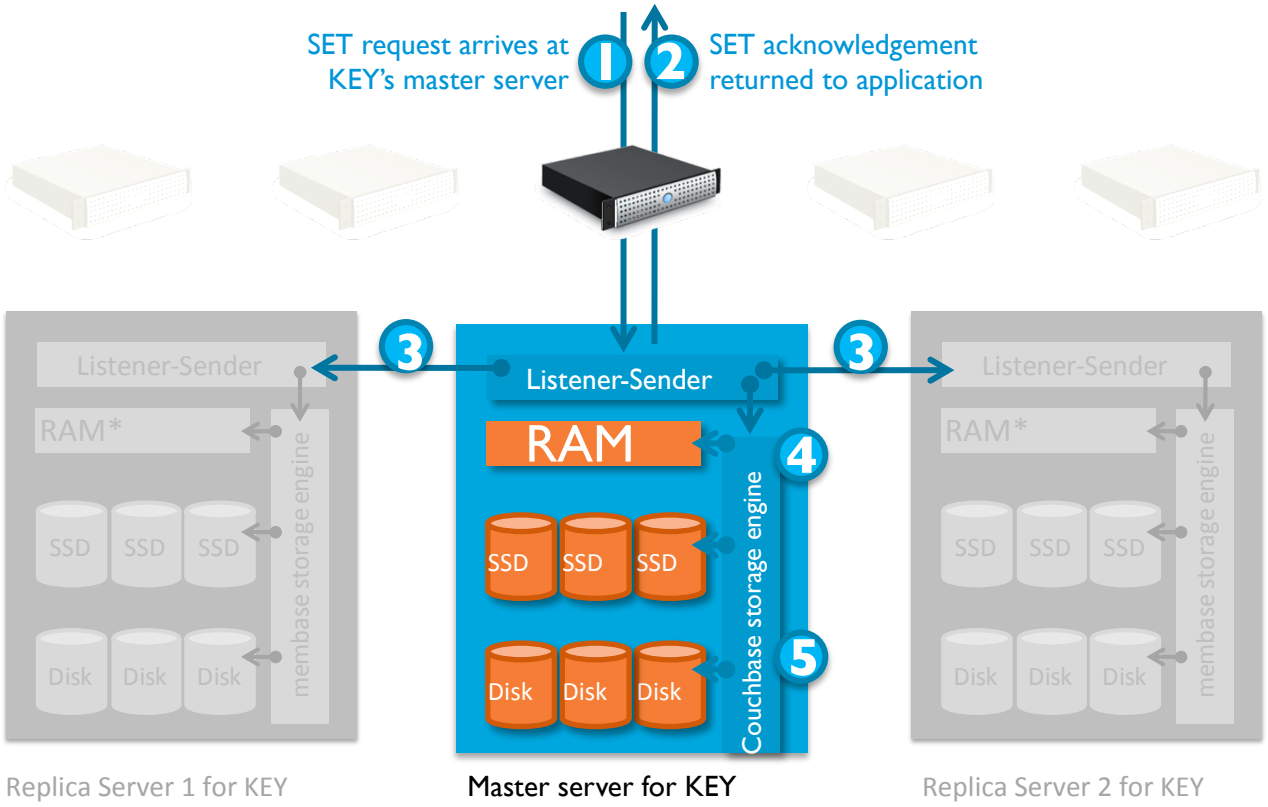
Architecture



Couchbase basic Architecture

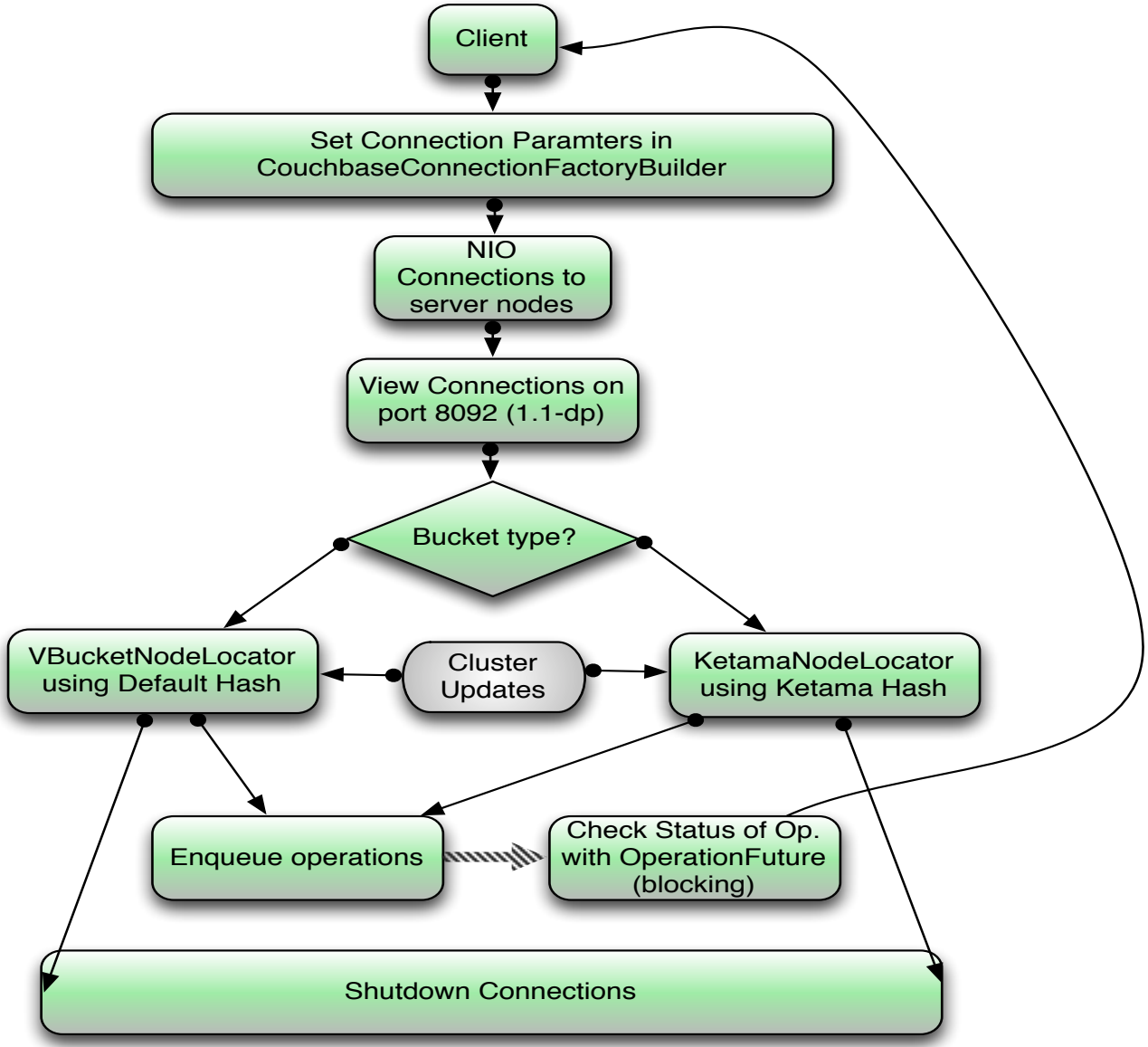


Operations workflow

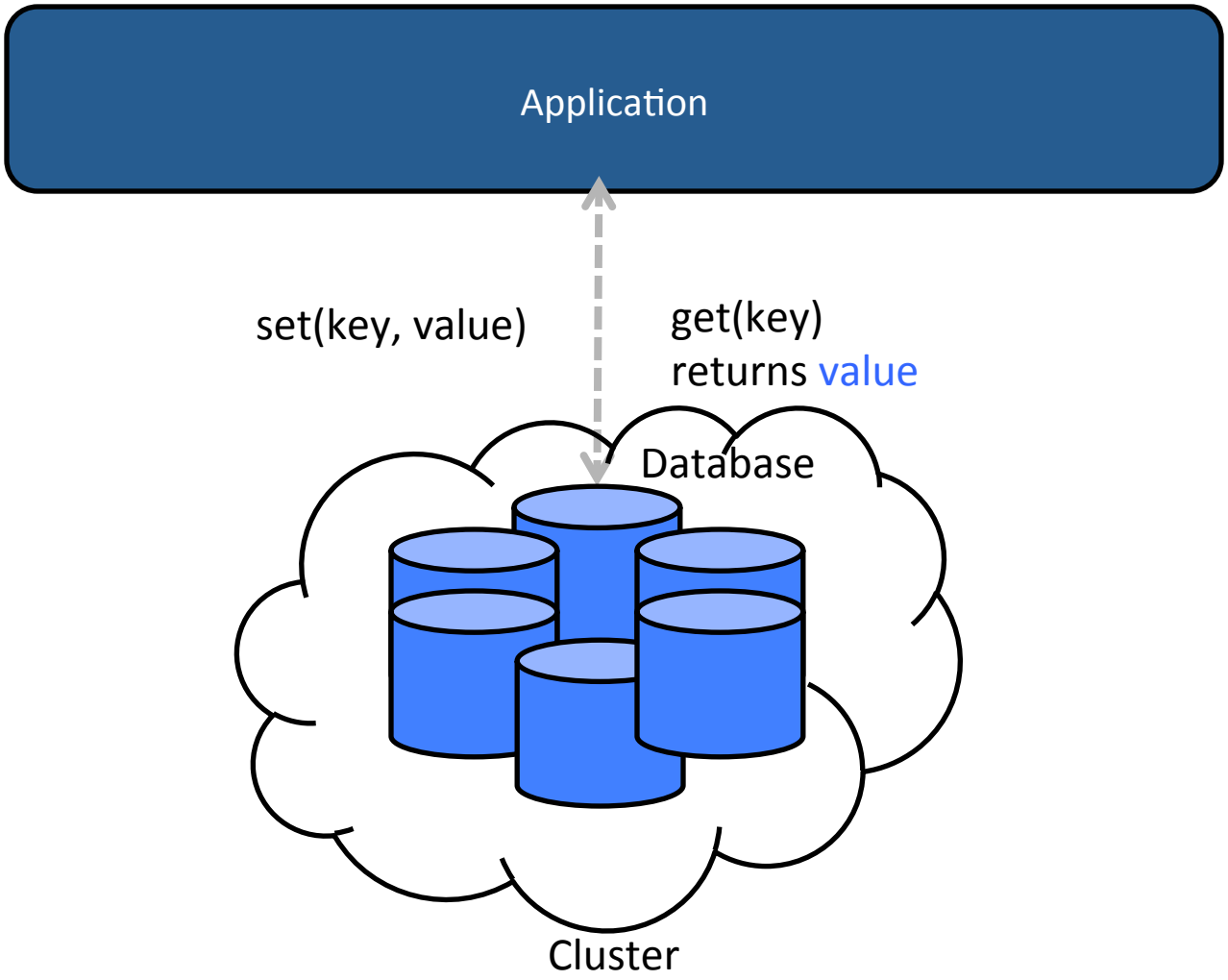


Durability/HA comes first from off-node replication, fallback to disk persistence

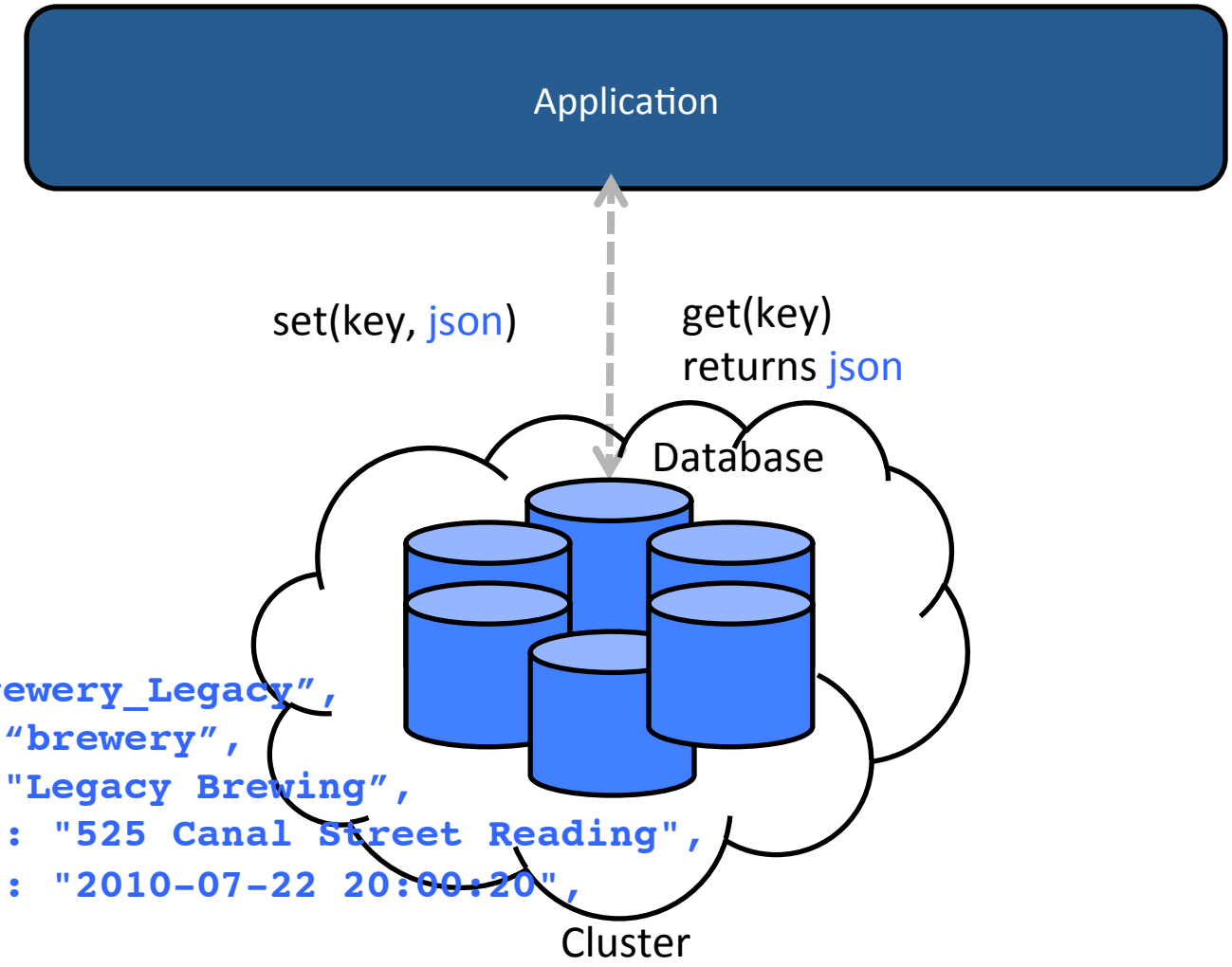
High Level Architecture of Java client library



A Distributed Hash Table



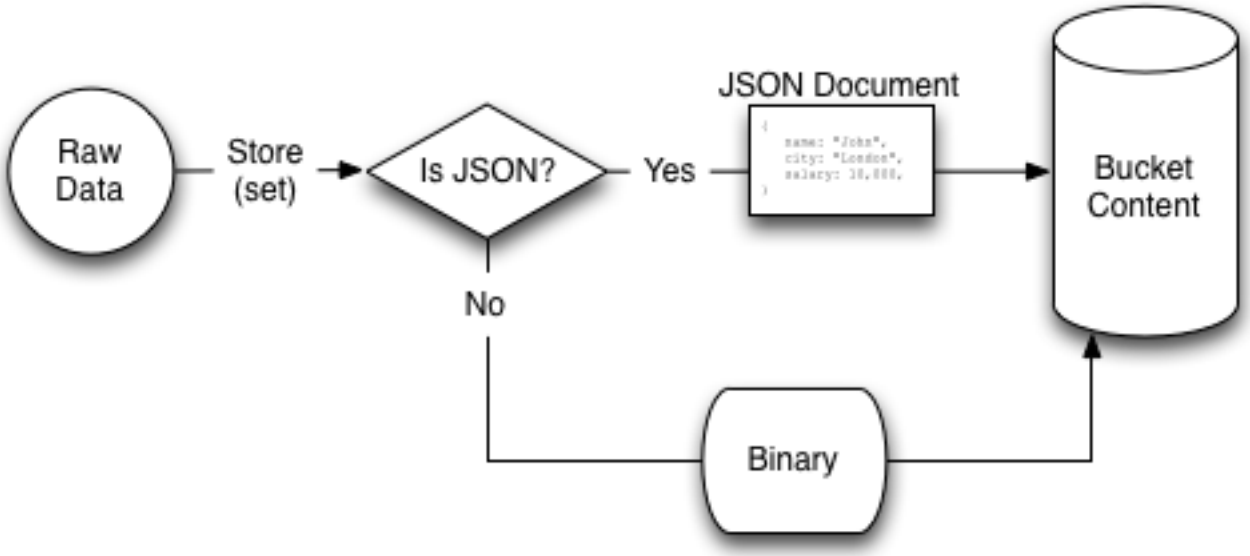
A Distributed ~~Hash Table~~ Document Store



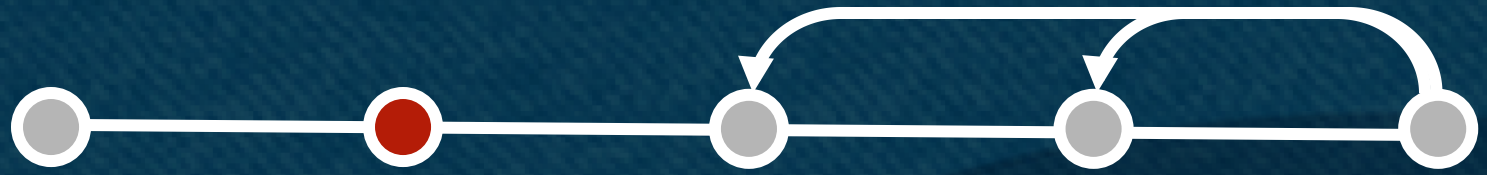

```
{
“id": ”brewery_Legacy”,
“type” : “brewery”,
"name" : "Legacy Brewing”,
“address": "525 Canal Street Reading",
"updated": "2010-07-22 20:00:20",
}
```

A JSON-based document store

- JSON is processed specially on the server



GETTING STARTED



Installation

- RedHat – RPM
- Ubuntu – dpkg
- Windows
- Mac OS
- GUI Installer
- Unattended installation
- Quickstart once installed
- http://<hostname>:8091

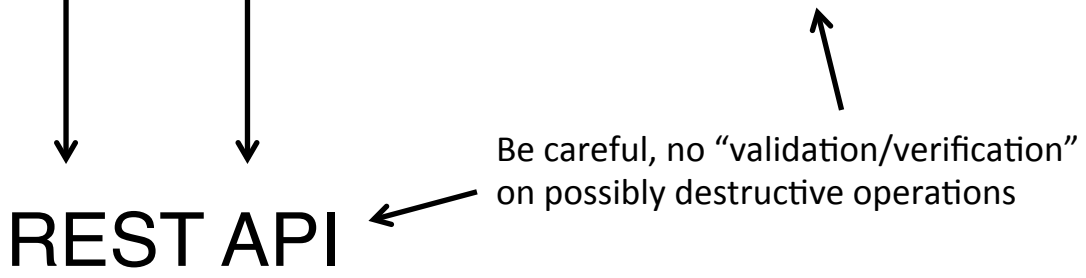
Management Tools

Web Console

- http://<IP>:8091
- JQUERY-based
- Simple/easy management/monitoring
- “pretty graphs”

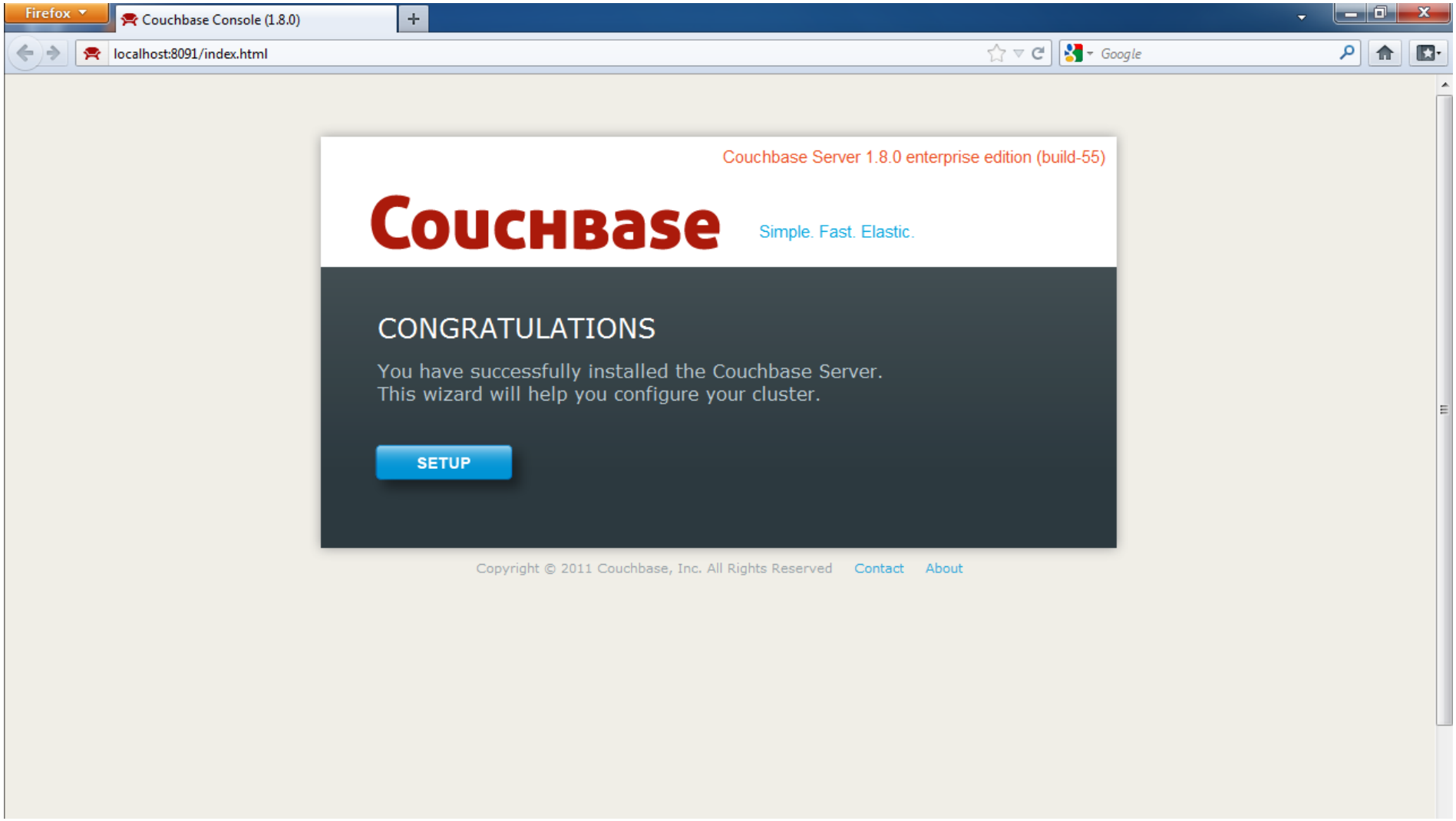
CLI

- Python script: /opt/couchbase/bin/couchbase-cli
- Useful for scripting, automated management



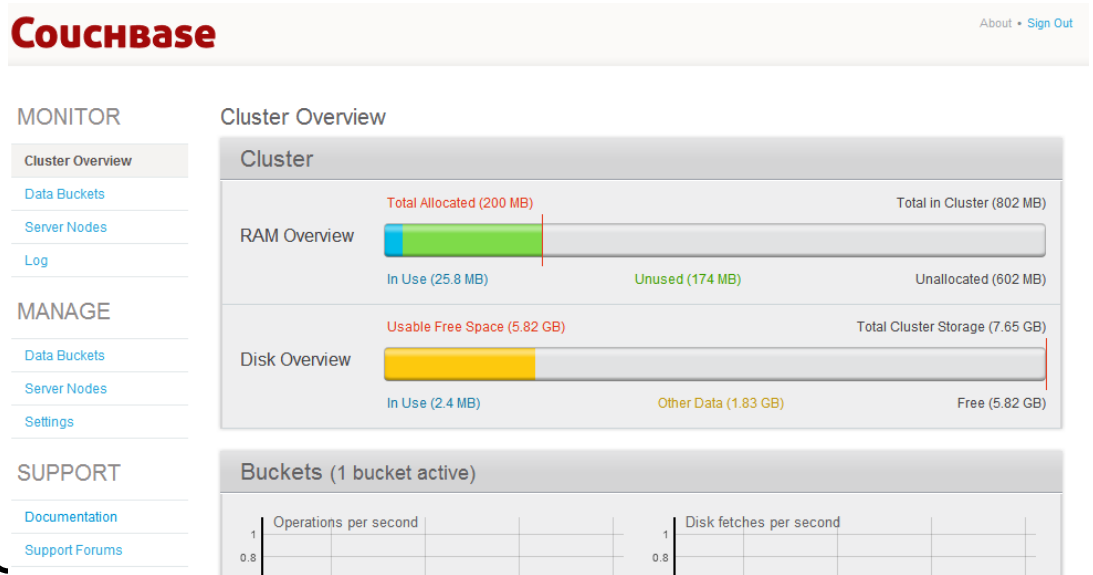
- RESTful API (JSON)
- “Authoritative” API
- Programmatic access for management/monitoring
- Management channel for “smart” clients
- No “data” manipulation
- Documented at: http://couchbase.org/wiki/display/membase/Membase+Management+REST+API

Web Setup


Firefox
Couchbase Console (1.8.0)
localhost:8091/index.html
Google
Couchbase Server 1.8.0 enterprise edition (build-55)
Couchbase
Simple. Fast. Elastic.
CONGRATULATIONS
You have successfully installed the Couchbase Server.
This wizard will help you configure your cluster.
SETUP
Copyright © 2011 Couchbase, Inc. All Rights Reserved
Contact
About

Web Console

navigation



usage

traffic

server states

Exercise 1a – Node Up and Running

- Objective
  - to setup the Couchbase server on localhost
- Steps
  - ssh (if required) into machine
  - Install Couchbase (example):
    - rpm -i couchbase-server-enterprise_<version>.rpm
  - Launch the Web Console at
    - http://<host>:8091
  - Run the Setup Wizard and accept the defaults for each step (except the per node RAM quota for the **default** bucket in step 2, which should be lower since you will create another bucket in the next step)

Exercise 1b – Create another bucket

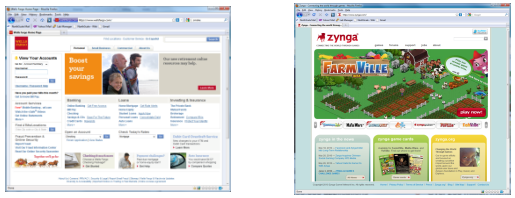
- Objective
  - to setup another bucket on the couchbase server
- Steps
  - Same steps as earlier and create another bucket with the rest of the RAM

COUCHBASE CLIENT INTERFACE

Buckets and vBuckets

- Buckets
  - Compartments for storing data
  - Unit of resource control
  - Individual replication, port, authentication configuration
  - Support Multi-tenancy (multiple applications or multiple threads of the same application)
- vBuckets
  - Internal structure
  - Distribute bucket information across cluster
  - Support scalability, replicas and failover
- Clients communicate with buckets, but use vBuckets

Data buckets are secure Couchbase "slices"



Application user

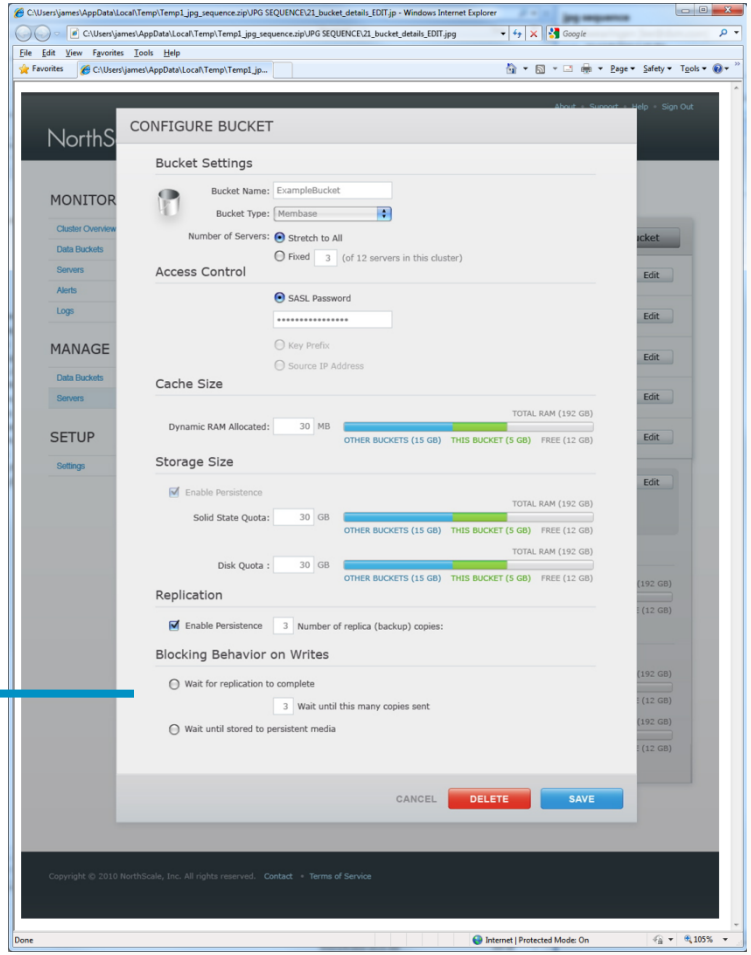
Web application server

Bucket 1

Bucket 2

Aggregate Cluster Memory and Disk Capacity

Couchbase data servers



In the data center

On the administrator console

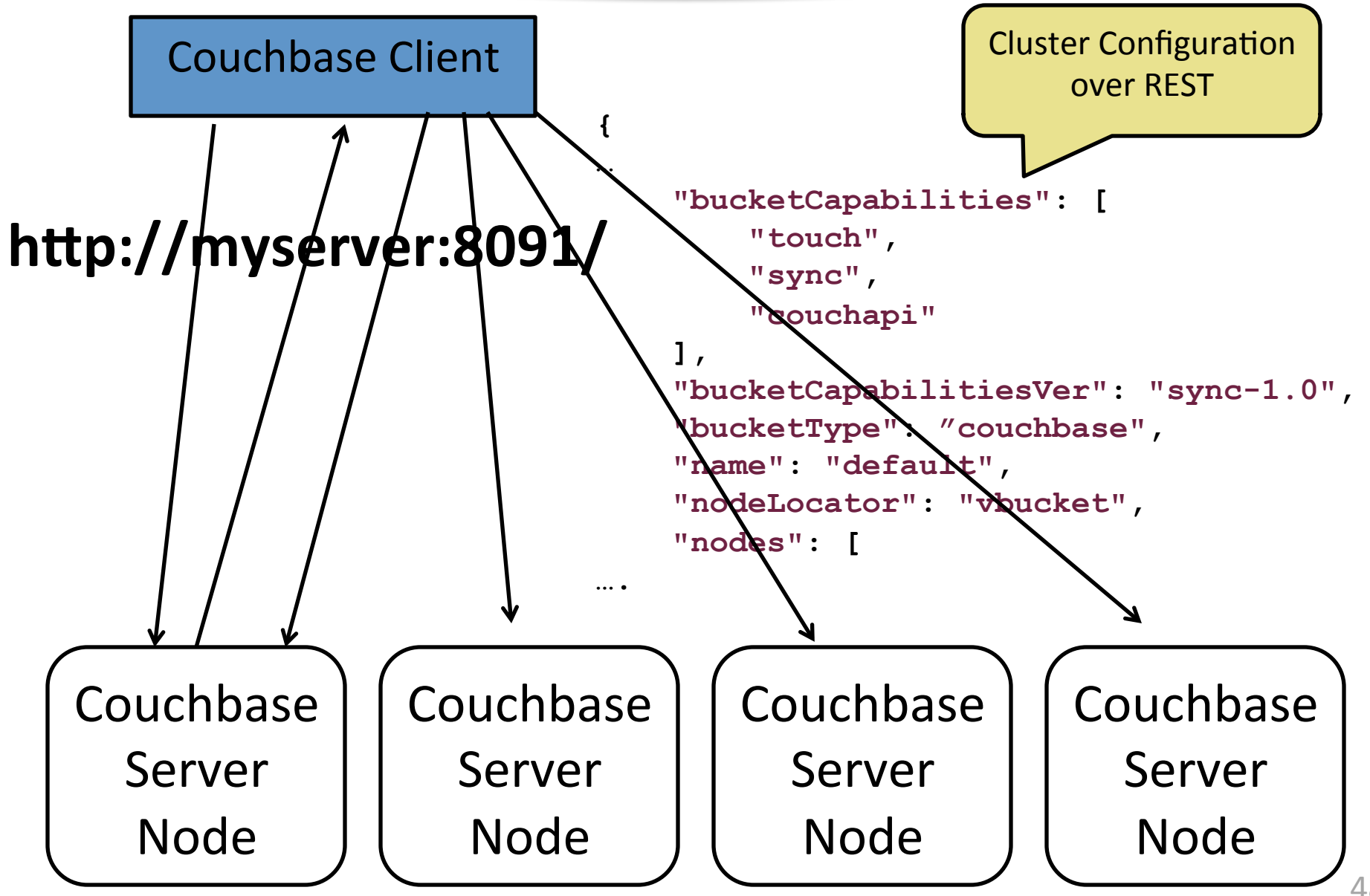
Client Deployments

- Smart Clients
  - Operate in conjunction with cluster and vBucket configuration
  - Automatically reads/writes to the correct node in cluster
  - Reacts automatically to topology changes
- Client-Side Moxi
  - Allows legacy (memcached) clients to use Couchbase
  - Moxi operates in conjunction with cluster
  - Clients communicate with Moxi on local port

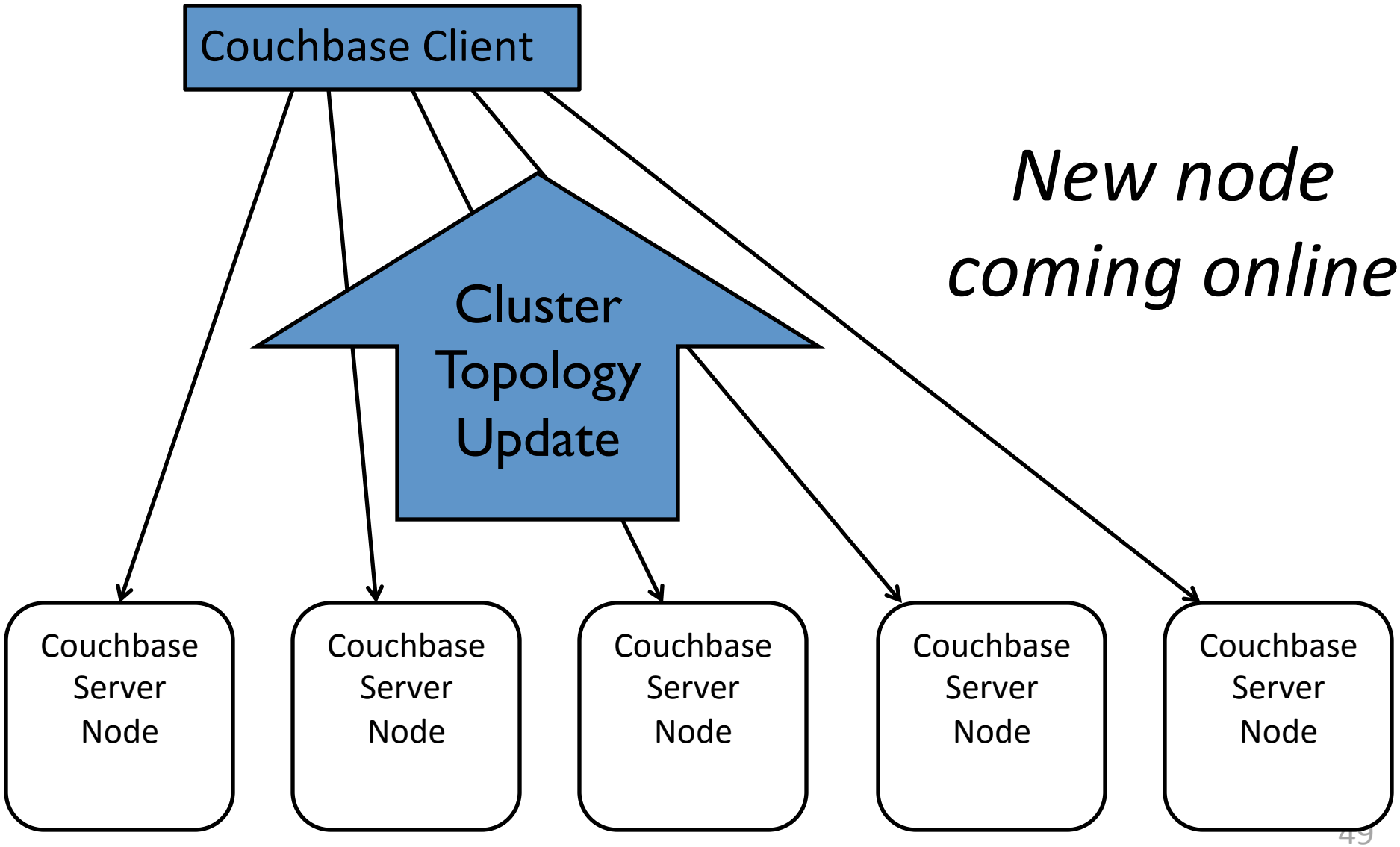
Smart Clients

- Communicate directly with cluster
- Understand cluster topology
- Automatically distribute requests to cluster nodes
- Automatically direct requests on topology changes
- Automatically direct requests during failover

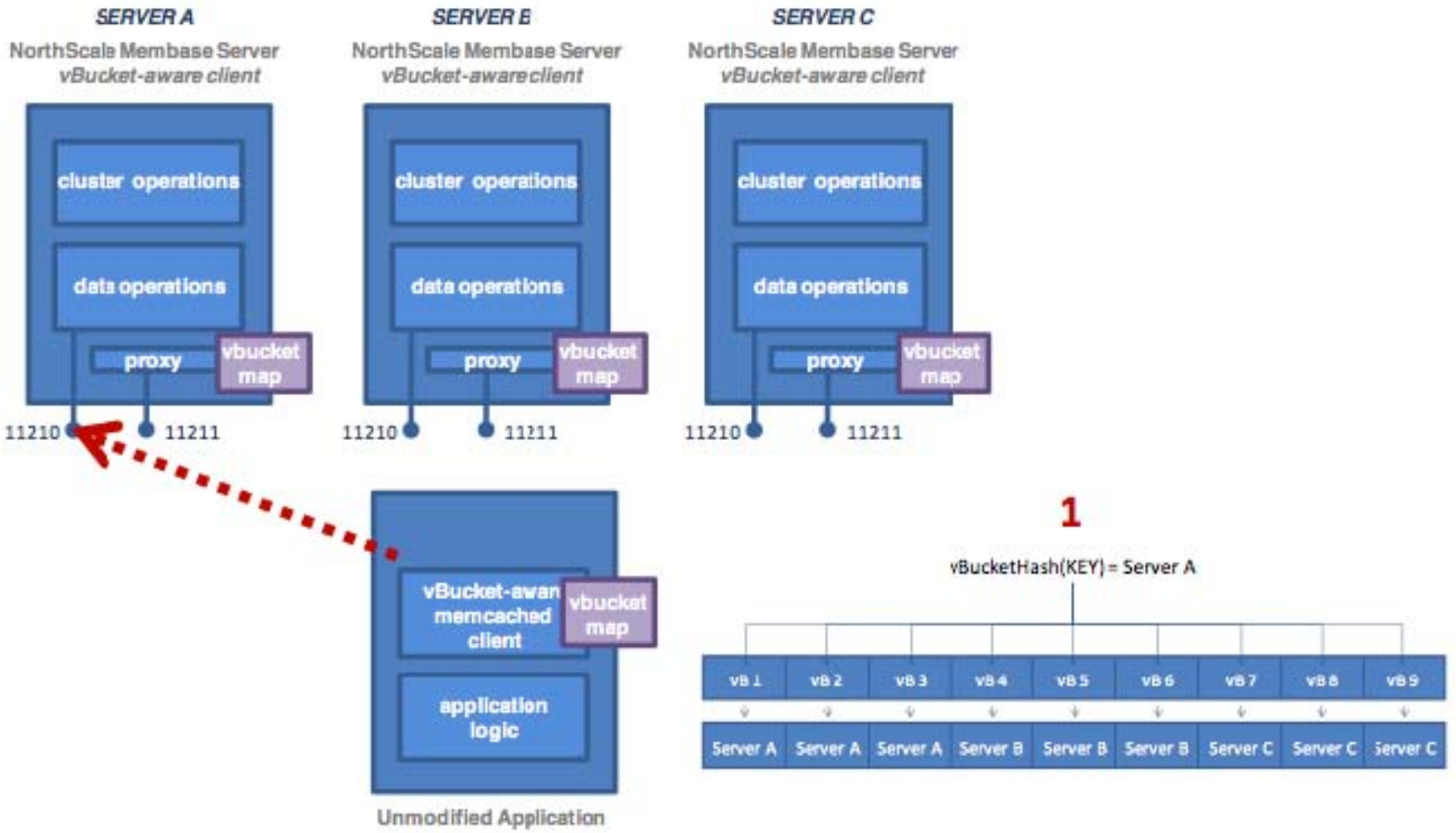
Client Setup: Getting Cluster Configuration



Client at Runtime: Adding a node



Smart Client Deployment



Clients During Rebalance/Failover

- Client can connect to any node in the cluster
- If the primary node fails, client library will continue to communicate with other nodes in the cluster and will mark the primary node as down
- The default timeout for a node is 2.5 seconds
- The server comes back online after marking the node as down

Clients During Rebalance/Failover

- Client writes to node in cluster
- During a Rebalance
  - vBucket Configuration is Updated
  - Client switches to new topology once rebalance complete
- During a failover
  - Client writes to replica node once failover is complete
- Clients should cope with transient read/write failures

Couchbase Clients

- Couchbase ('smart') Clients
  - libcouchbase (C)
  - .NET
  - Java
  - PHP
  - Ruby
  - Python
  - Perl (3rd party)
- Any Memcached Library
  - Use client-side proxy

Opening a Connection (Java)

- Connect to the Cluster URI of any cluster node

```
List<URI> uris = new LinkedList<URI>();

uris.add(URI.create("http://127.0.0.1:8091/pools"));
try {
  client = new CouchbaseClient(uris, "default", "");
} catch (Exception e) {
  System.err.println("Error connecting to Couchbase: " +
e.getMessage());
  System.exit(0);
}
```

Opening a Connection (.NET)

- Create XML Configuration

```
<couchbase>
   <servers bucket="private" bucketPassword="private">
     <add uri="http://10.0.0.33:8091/pools/default"/>
   </servers>
 </couchbase>
```

- Connect

```
var client = new CouchbaseClient())
```

Opening a Connection (P*, Ruby)

- PHP

```
$cb_obj = new Couchbase("localhost", "user", "pass",
"default");
```

- Python

```
from couchbase.couchbaseclient import
VBucketAwareCouchbaseClient as couchbaseClient
server = {"ip":"127.0.0.1", "port" : "8091" };
v = couchbaseClient(server, 'default')
```

- Ruby

```
require 'couchbase'


client = Couchbase.new "http://127.0.0.1:8091/pools/
default"
```

Authentication

- Buckets authenticated using SASL
- Create a ConnectionFactory, then client:

```
List<URI> baseURIs = new ArrayList<URI>();
baseURIs.add(base);
CouchbaseConnectionFactory cf = new
          CouchbaseConnectionFactory(baseURIs,
             "userbucket", "password");

client = new CouchbaseClient(cf);
```

Setting connection parameters

Parameter	Description	Default
`MaxReconnectDelay`	Maximum number of milliseconds to wait between reconnect attempts.	30 secs.
`ShouldOptimize`	If true, low-level optimization is in effect.	true
`OpQueueMaxBlockTime`	Get the maximum amount of time (in milliseconds) a client is willing to wait to add a new item to a queue.	10 millisecs
`OpTimeout`	Operation timeout used by this connection.	2.5 secs
`TimeoutExceptionThreshold`	Maximum number of timeout exception for shutdown connection.	998

Setting connection parameters (optional)

```
CouchbaseConnectionFactoryBuilder cfb = new
    CouchbaseConnectionFactoryBuilder();

// wait up to 10 seconds for an operation to succeed
cfb.setOpTimeout(10000);

// wait up to 5 seconds when trying to enqueue an operation
cfb.setOpQueueMaxBlockTime(5000);

// Don't optimize – more network traffic but lower latency
cfb.setShouldOptimize(false);

// Lower the Timeout exception threshold
cfb.setTimeoutExceptionThreshold(100);

CouchbaseClient myclient = new
              CouchbaseClient(cfb.buildCouchbaseConnection(baseURIs,
                  "default", "default", ""));

// Use the client with the new parameters
```

Java Client Library Versions

- Couchbase-client and spy
  - Couchbase-client : Smart client that adapts to the network topology changes
  - Spy : Low level libraries that implement the memcached, TAP and other interfaces (both binary and ascii).
- Server 1.8 Compatible
  - 1.0.1/2.8.0 (introduction of Couchbase client)
  - 1.0.2/2.8.0 (fixes Memcache buckets imbalance in a cluster)
- Server 2.0 Compatible
  - 1.1-dp/2.8.1 (Views)

Java Client Library Dependencies

- common-codecs
  - Base 64 encoding. Encoders and decoders
- jettison
  - JSON Processing
- netty
  - HTTP comet streaming
- httpcore-nio
  - High performance NIO sockets (for views)
- httpcore
  - Needed by httpcore-nio

Exercise 2 – Set up Eclipse Project and JARs

- Objective
  - to setup the Couchbase SDK and its dependencies
- Steps
  - Import HelloCouchbase into Eclipse as a Maven project
  - Modify the pom.xml (if required) and run the project
- If you do not want to use Maven and Eclipse OR want to use your own IDE, copy the JAR files and include them in your classpath as described in

  http://www.couchbase.com/docs/couchbase-sdk-java-1.1/hello-world.html

Exercise 3 – Set up connection parameters

- Objective
  - to learn about the parameters that can be set for the connection via the **`CouchbaseConnectionFactoryBuilder`** `class`
- Steps
  - Setup a Couchbase connection
  - Use the **`CouchbaseConnectionFactoryBuilder`** `class to set the following parameters.`
    - **ShouldOptimize**
    - **OpQueueMaxBlockTime**
    - **OpTimeout**

PROTOCOL OVERVIEW

Data is stored as key/document pairs

- Key is the identifier
  - A string without spaces
  - May use separators/identifiers; for example person_93847
  - Must be unique within a bucket
- Document is pure bytestring (JSON adds more meaning)
  - No implied value or understanding on server-side (generally)
  - Integers are valid for certain operations (increment, decrement)
  - Can store strings, images, or serialized objects
  - Server will try to discern JSON from non-JSON

Metadata

- Server stores metadata with each key/value pair
  - Expiry (TTL)
  - CAS (checksum) value
  - Flags
- Expiry is used by server to delete values after the specified time
- Flags can be used to identify the data type of stored information
  - In Java, this information is used to serialize the data type of the object stored

Time To Live (TTL)

- TTL
  - Property to set expiration on the document
  - the actual value sent may either be
  - Unix time (number of seconds since January 1, 1970, as a 32-bit value
  - OR number of seconds starting from current time. number of seconds may not exceed 60*60*24*30 (number of seconds in 30 days)
  - if the number sent by a client is larger than
  - that, the server will consider it to be real Unix time value rather than an offset from current time.

Common client Features

- All operations are atomic
- All data operations require key
- No implicit locking (no row lock, etc.)
- Different clients implement core protocol
- Some clients offer language/environment unique functionality

Store Operations

Operation	Description
add()	Adds new value if key does not exist or returns error.
replace()	Replaces existing value if specified key already exists.
set()	Sets new value for specified key.

All sets allow for ‘expiry’ time specified in seconds:

Expiry Value	Description
Values < 30*24*60*60	Time in seconds to expiry
Values > 30*24*60*60	Absolute time from epoch for expiry

Retrieve Operations

Operation	Description
get()	Get a value.
getAndTouch()	Get a value and update the expiry time.
getBulk()	Get multiple values simultaneously, more efficient.
gets()	Get a value and the CAS value.

Update Operations

Operation	Description
append()	Appends data to an existing key.
prepend()	Prepends data to an existing key.
incr()	Increments an integer value by specified amount, default 1.
decr()	Decrements an integer value by specified amount, default 1.
replace()	Replaces an existing key/value pair.
touch()	Updates the expiry time for a given item.

Exercise 4: Connect, Store, Retrieve

- Objective
  - to learn about some of the operations supported on the Couchbase client
- Steps
  - Set up a Couchbase connection
  - Modify the program provided to use the following operations
    - set(), get(), add(), replace(), touch(), append(), prepend(), etc.
  - Try different parameters available

ASYNCHRONOUS OPERATIONS

Couchbase Java Client Asynchronous Functions

- Synchronous commands
  - Force wait on application until return from server
- Asynchronous commands
  - Allow background operation until response available
  - Operations return **Future** object
  - Useful when data on server not in-memory, or during sets/updates

Asynchronous functions

- A Typical Asynchronous call
  - An asynchronous call that returns a Future object (Non blocking)
  - Process something (in the meantime) that does not depend on the result and the result is still being computed on the server
  - Either
    - Call the **get()** method on the Future object (blocking call) OR
    - Call the **cancel()** method to cancel the operation

It's Asynchronous

- Set Operation is asynchronous

```
// Do an asynchronous set
OperationFuture<Boolean> setOp =
    client.set(KEY, EXP_TIME, VALUE);

// Do something

// Check to see if our set succeeded
// We block when we call the get()
if (setOp.get().booleanValue()) {
      System.out.println("Set Succeeded");
} else {
      System.err.println("Set failed: " +
          setOp.getStatus().getMessage());
}
```

It's Asynchronous

- Even a get can be async

```
GetFuture getOp = client.asyncGet(KEY);

// do some work

if ((getObject = getOp.get()) != null) {
    System.out.println("Asynchronous Get Succeeded: " + getObject);
} else {
    System.err.println("Asynchronous Get failed: " +
        getOp.getStatus().getMessage());
}
```

A more realistic Asynchronous Call Example

- Process something while waiting for the results from the server (a mashup example)

```
// Non-blocking call
BulkFuture<Map<String, Object>> bf  = c.asyncGetBulk(input);

// Render the available data
$("#beer").autocomplete("getBeer.jsp");

// block until result becomes available
mp = bf.get();
$("#beerimage").autocomplete("getBeerImages.jsp");
```

Exercise 5: Asynchronous Operations

- Objective
  - to learn about some of the asynchronous operations supported on the Couchbase client
- Steps
  - Set up a Couchbase connection
  - Modify the program provided to use the following operations
    - asyncGet(), set(), delete(), etc.
    - Use the return Future value
    - Try different parameters available

OBJECT SERIALIZATION

Transcoding

- Transcoders encode/decode native objects to bytestring
- Uses SerializingEncoder by default
  - Serializes Java objects to Byte Streams
- You can override this (but more than likely you don't)

A Transcoder that reverses a String

- A Transcoder that uses the SerializingTranscoder

```
class StringTranscoder implements Transcoder<String> {

    final SerializingTranscoder delegate = new SerializingTranscoder();

    public boolean asyncDecode(CachedData d) {
        return delegate.asyncDecode(d);
    }

    public String decode(CachedData d) {
        StringBuffer s;
        return (String) delegate.decode(d);
    }

    public CachedData encode(String o) {
        return delegate.encode(new StringBuffer(o).reverse().toString());
    }

    public int getMaxSize() {
        return delegate.getMaxSize();
    }

}
```

Using the Transcoder

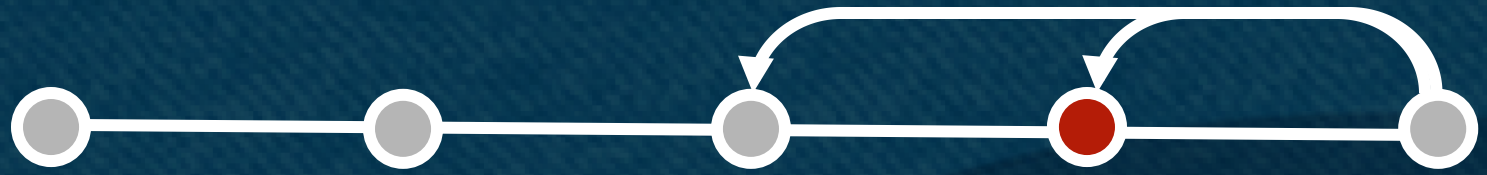
- Using the Transcoder

```
Transcoder<String> transcoder = new StringTranscoder();

URI local = new URI("http://localhost:8091/pools");
List<URI> baseURIs = new ArrayList<URI>();
baseURIs.add(local);

CouchbaseClient c = new
    CouchbaseClient(baseURIs, "default", "");
c.set("key", 2, "Madam", transcoder).get();
System.out.println(c.get("key")); // madaM
```

Exercise 6: Transcoder

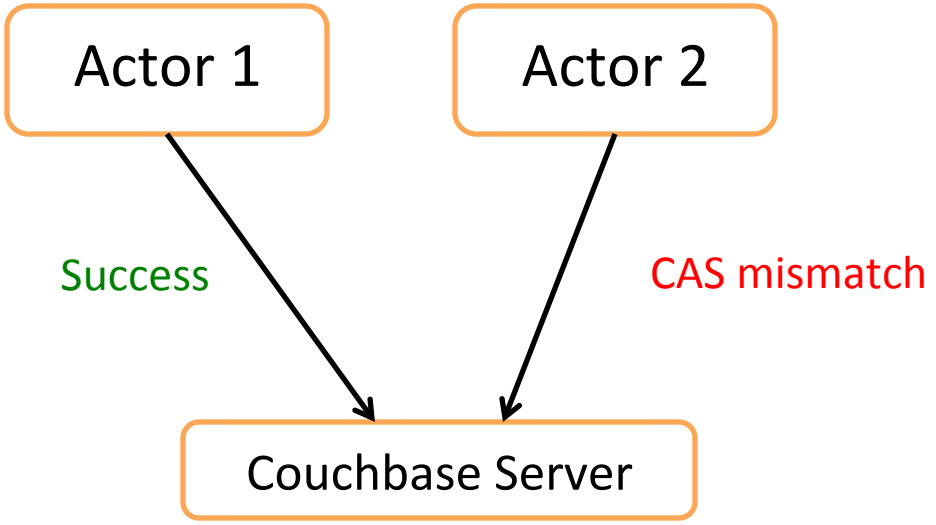
- Objective
  - to learn defining and using a Transcoder
- Steps
  - Set up a Couchbase connection
  - Implement a Transcoder for a String that
    - Replaces the first occurrence of a space with an underscore

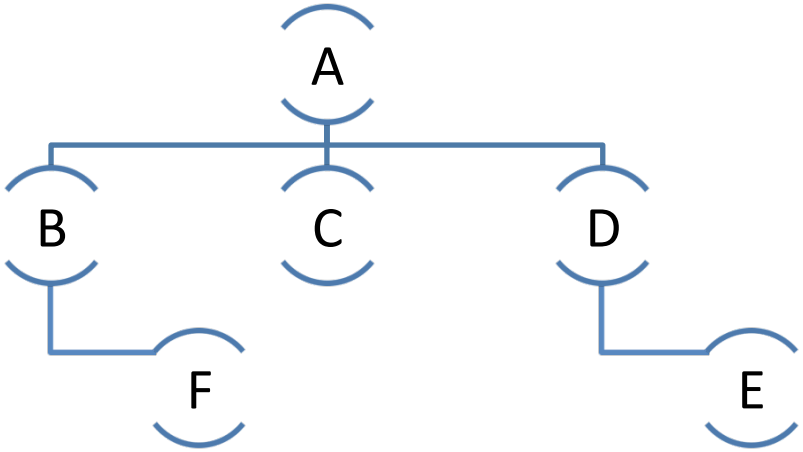
CONCURRENCY



Distributed System Design: Concurrency Controls

- Compare and Swap Operations
  - Often referred to as “CAS”
  - Optimistic concurrency control
  - Available with many mutation operations, depending on client.
- Get with Lock
  - Often referred to as “GETL”
  - Pessimistic concurrency control
  - Locks have a short TTL
  - Locks released with CAS operations
  - Useful when working with object graphs





Check and Set/Compare and Swap (CAS)

- Uses checksum to validate a change to a value:
  - Client gets key and checksum (cas_token)
  - Client updates using key and checksum
  - If checksum doesn't match, update fails
- Client can only update if the key + checksum match
- Used when multiple clients access the same data
- First client updates with checksum
- Subsequent client updates fail without the right checksum
- Use `CASMutation` and `CASMutator` for higher level of abstraction

CAS Operation

- CAS Example

```
CASValue<Object> casv = client.gets(KEY);
Thread.sleep(random.nextInt(1000)); // a random workload

// Wake up and do a set based on the previous CAS value
Future<CASResponse> setOp =
        client.asyncCAS(KEY, casv.getCas(), random.nextLong());

// Wait for the CAS Response
try {
  if (setOp.get().equals(CASResponse.OK)) {
    System.out.println("Set Succeeded");
  } else {
    System.err.println("Set failed: ");
  }
}
…
```

CAS Mutator and Mutation

- Using CAS Mutator and Mutation

```
CASMutation<String> mutation = new CASMutation<String>() {
    public String getNewValue(String current) {
        return current.replaceAll(getUserNameToken(), "");
    }
};

Transcoder<String> transcoder = new StringTranscoder();
CASMutator<String> mutator = new
    CASMutator<String>(client, transcoder);
// Key “CurrentUsers” will be mutated atomically
mutator.cas("CurrentUsers", null, 0, mutation);
```

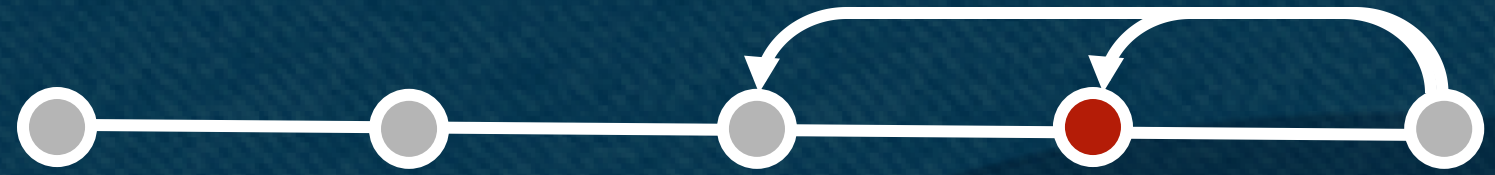
Exercise 7a: Concurrency Operations

- Objective
  - to learn about Concurrency operations supported on the Couchbase client
- Steps
  - Set up a Couchbase connection
  - Use the following operations
    - gets() and aynchCAS() to demonstrate contention between two separate clients

Exercise 7b: Concurrency Operations

- Objective
  - to learn about Concurrency operations supported on the Couchbase client
- Steps
  - Set up a Couchbase connection
  - Modify the previous program to use CASMutator() and CASMutation()

BULK GET


Intro.
Getting Started
Async. operations
Advanced Operations
Doc. Design and Beer Data

getBulk : Getting Multiple Key Values

- Getting values in Bulk

```
// Get as strings
Map<String, Object> mp=new HashMap<String, Object>();
mp = c.getBulk("red", "sox");
System.out.println(mp);

Collection<String> input = new ArrayList<String>();
input.add("red");
input.add("sox");

// Get as Collection
mp = c.getBulk(input);
System.out.println(input);
System.out.println(mp);
```

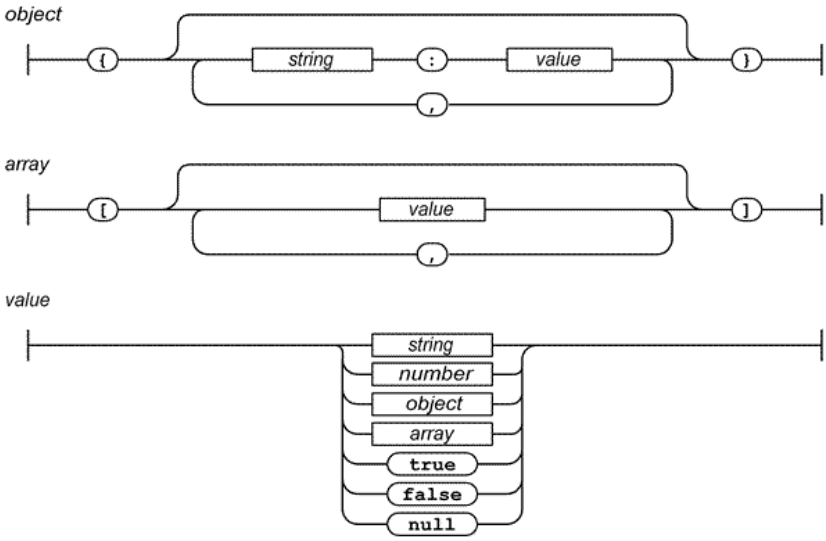
Exercise 8: Getting Multiple Key Values

- Objective
  - to learn about getting multiple Key Values
- Steps
  - Set up a Couchbase connection
  - Use getBulk() in it's different forms to get Multiple key values
  - Modify the above program to use asyncGetBulk()

USING JSON IN COUCHBASE

Why use JSON?

- JSON (JavaScript Object Notation)
  - Lightweight Data-interchange format
  - Easy for humans to read and manipulate
  - Easy for machines to parse and generate (many parsers are available)
  - JSON is language independent (although it uses similar constructs)



JSON Basics

- JSON (JavaScript Object Notation) data type support
  - Number (either integer or floating point)
  - String – supporting Unicode characters and backslash escape
    - “A String”
  - Boolean – true or false
    - { “value”: true}
  - Array – a list of values enclosed in square brackets
    - [“one”, “two”, “three”]
  - Object – a set of key/value pairs i.e. an associative array or hash. Key must be string, value can be any supported JSON values

```
{
   “servings” : 4, “difficulty” : “Easy”, “cooktime” : 60, “title” : “Rasam”
}
```

Using JSON in Couchbase

- Presidents.json

```
[
      {"presidency":"1",
      "name":"George Washington",
      "wikipedia_entry":"http://en.wikipedia.org/wiki/George_Washington",
      "took_office":1789,
      "left_office":1797,
      "party":"Independent",
      "portrait":"GeorgeWashington.jpg",
      "thumbnail":"thmb_GeorgeWashington.jpg",
      "home_state":"Virginia",
      "type":"president"},
      {"presidency":"2",
      "name":"John Adams",
      "wikipedia_entry":"http://en.wikipedia.org/wiki/John_Adams",
      "took_office":1797,
      "left_office":1801,
      "party":"Federalist ",
      "portrait":"JohnAdams.jpg",
      "thumbnail":"thmb_JohnAdams.jpg",
      "home_state":"Massachusetts",
      "type":"president"},
     …
]
```

Using JSON in Couchbase

- President Class

```
class President
{
  String presidency;
  String name;
  String wikipedia_entry;
  String took_office;
  String left_office;
  String party;
  String portrait;
  String thumbnail;
  String notable;
  String home_state;
  String type;
}
```

Using JSON in Couchbase

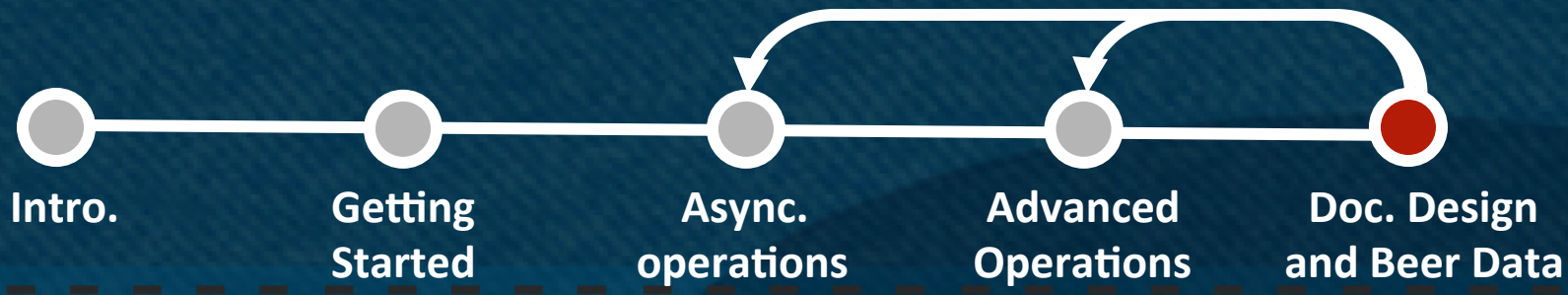
- You could use any Java JSON libraries (this example uses Gson)
- Use this class and the JSON file

```
Gson gson = new Gson();
President[] Presidents = gson.fromJson(new
        FileReader("Presidents.json"),
        President[].class);

for (President entry : Presidents) {
    String JSONentry = gson.toJson(entry);
    c.set(entry.presidency, 1200, JSONentry);
}
```

Exercise 9: Using JSON in Couchbase

- Objective
  - to learn about using JSON in Couchbase
- Steps
  - Set up a Couchbase connection
  - Use Gson libraries to load the Presidents (presidents.json will be provided) into Couchbase
  - Modify the data to include another field (like a Notable)
  - View the Key and Value using the admin console on the server
  - Modify the schema to add a field(s) and view the Key and Value on the server

COUCHBASE 2.0: DOCUMENTS



What is Couchbase Server 2.0

- Identical feature set to Couchbase Server 1.8
  - Cache-layer
  - High-performance
  - Cluster
  - Elastic
  - Core interface
- Adds
  - Document based storage (JSON)
  - Views with materialized indexes
  - Querying

Document Driven

- Use JSON to store documents
  - Replace serialized objects
  - Custom structures
- Documents define a "record" of data
- Store/Update/Retrieve using same protocol
- JSON parsed by the server View system

A JSON Document

```
{
      “id": "beer_Hoptimus_Prime",
      “type”: “beer”,
      "abv": 10.0,
      "brewery": "                ng Co.",
      "category":              can Ale",
      "name": "Hoptimus Prime",
      "style": "Imperial or Double India Pale Ale",
}
```

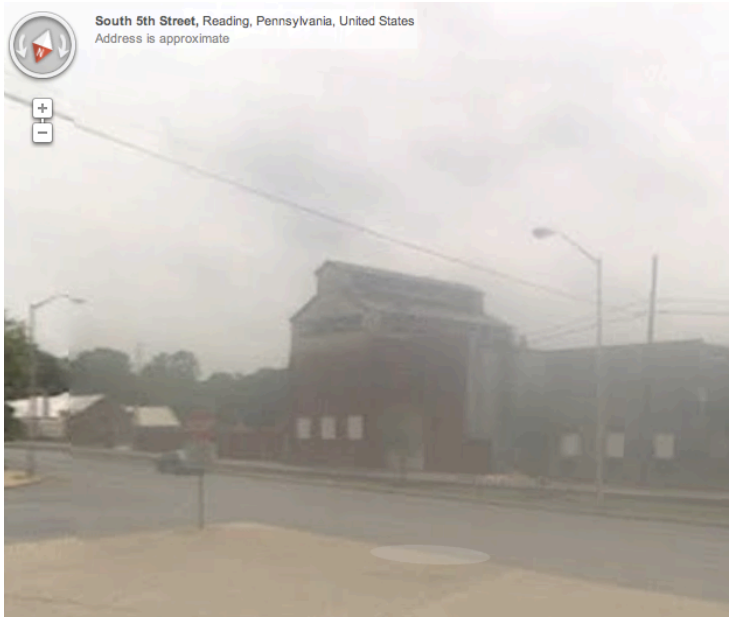
The primary key

The type information

A float

Other Documents and Document Relationships

```
{
      “id": "beer_Hoptimus_Prime",
      “type” : “beer”,
      "abv": 10.0,
      "brewery": ”brewery_Legacy_Brewing_Co",
      "category": "North American Ale",
      "name": "Hoptimus Prime",
      "style": “Double India Pale Ale”
}
```




```
{
      “id": ”brewery_Legacy_Brewing_Co”,
      “type” : “brewery”,
      "name" : "Legacy Brewing Co.",
      "address": "525 Canal Street
Reading, Pennsylvania, 19601 United
States",
       "updated": "2010-
     "latitude": -75.928
     "longitude": 40.325725
}
```

Afterthought

Simplicity of Document Oriented Datastore

- Schema is optional (schema evolves with the app.)
  - Technically, each document has an implicit schema
  - Extend the schema at any time!
    - Need a new field? Add it. Define a default for similar objects which may not have this field yet.
    - These changes to the schema (as the application evolves) can be tracked by a version number or other fields (as needed)
- Data is self-contained
  - Documents more naturally support the world around you, the data structures around you
- Model data for your App/Code instead for the Database

Adding a Document: Observations and Considerations

- Observations
  - Conversion to document was very simple, many JSON options
  - Flexible schema: Did not need to add the latitude and longitude to every record
  - Flexible schema: Can add the brewery detail later
- Considerations
  - Use a "type" field for high level filtering on object types
  - Why use a particular key/_id : **"beer_My_Brew"**
  - Should I have a TTL?

Common Questions when Adopting Couchbase

Q: What if I need to fetch referenced documents?

A: Simply get them one after another or use another View.

Q: How can I update just a small portion of a document?

A: The best approach is to keep the document model live in your application, then use CAS operations to store modified documents. The Ruby sample application has a good example.

Q: I currently use serialized objects with memcached or Membase, can I do this still with Couchbase Server?

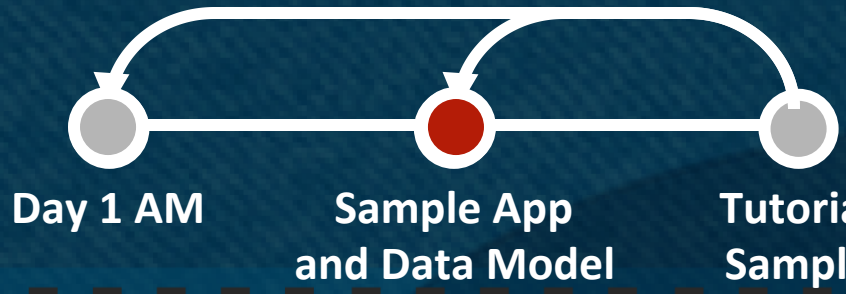
A: Absolutely! Everything previously supported and used is still there. JSON offers advantages with heterogenenous platform support and preparing for Couchbase 2.0 views.

Exercise 10: Loading Beer and Brewery Data

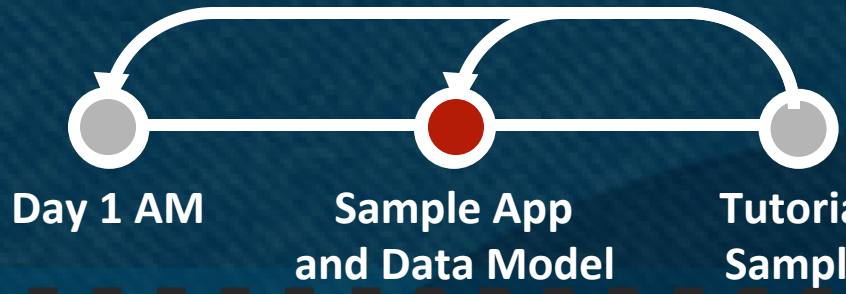
- Objective
  - Loading the Beer Data into Couchbase so that it can be used in the sample App.
- Steps
  - Set up a Couchbase connection
  - Use Gson libraries to load the Beers (beers.json will be provided) and Breweries (breweries.json) into Couchbase
  - View the Key and Value using the admin console on the server
  - Modify the schema to add a field(s) and view the Key and Value on the server

SCHEDULE

(DAY 1 AFTERNOON)



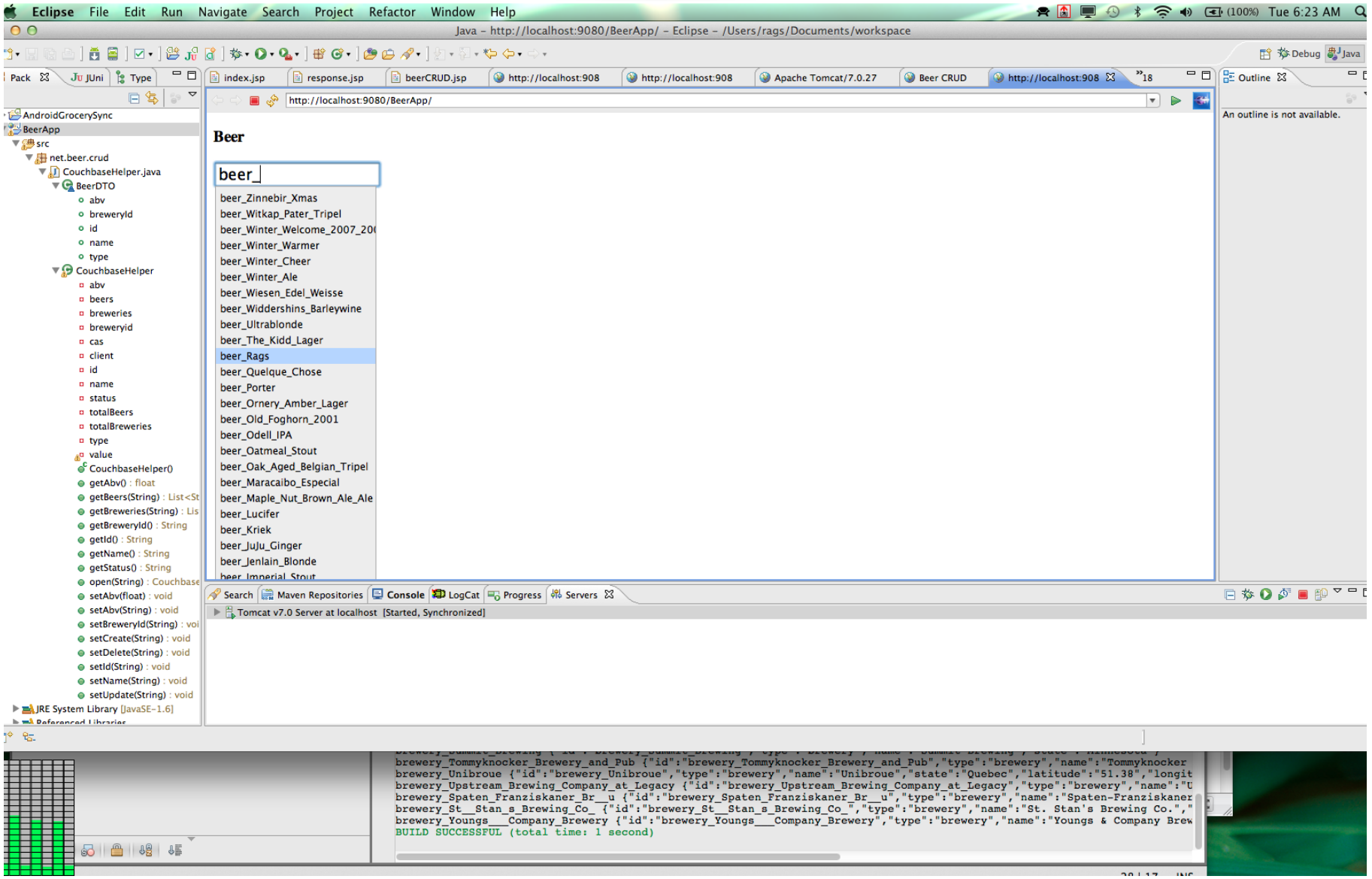
SAMPLE BEER APP



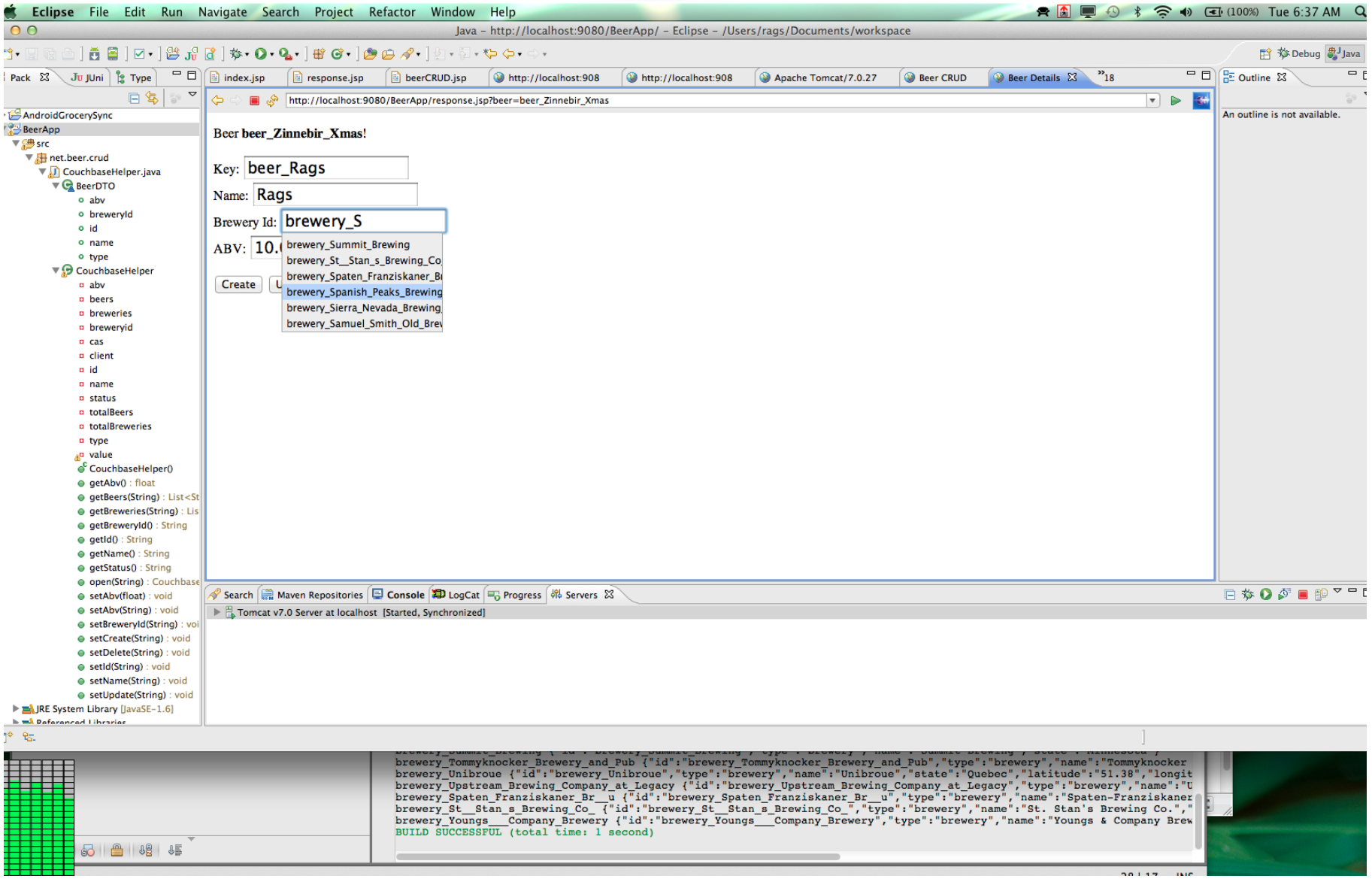
Sample Application

- A CRUD application for manipulating the Beer and Brewery Data
  - JSP-based application
  - Couchbase Helper class for getting/updating and deleting values

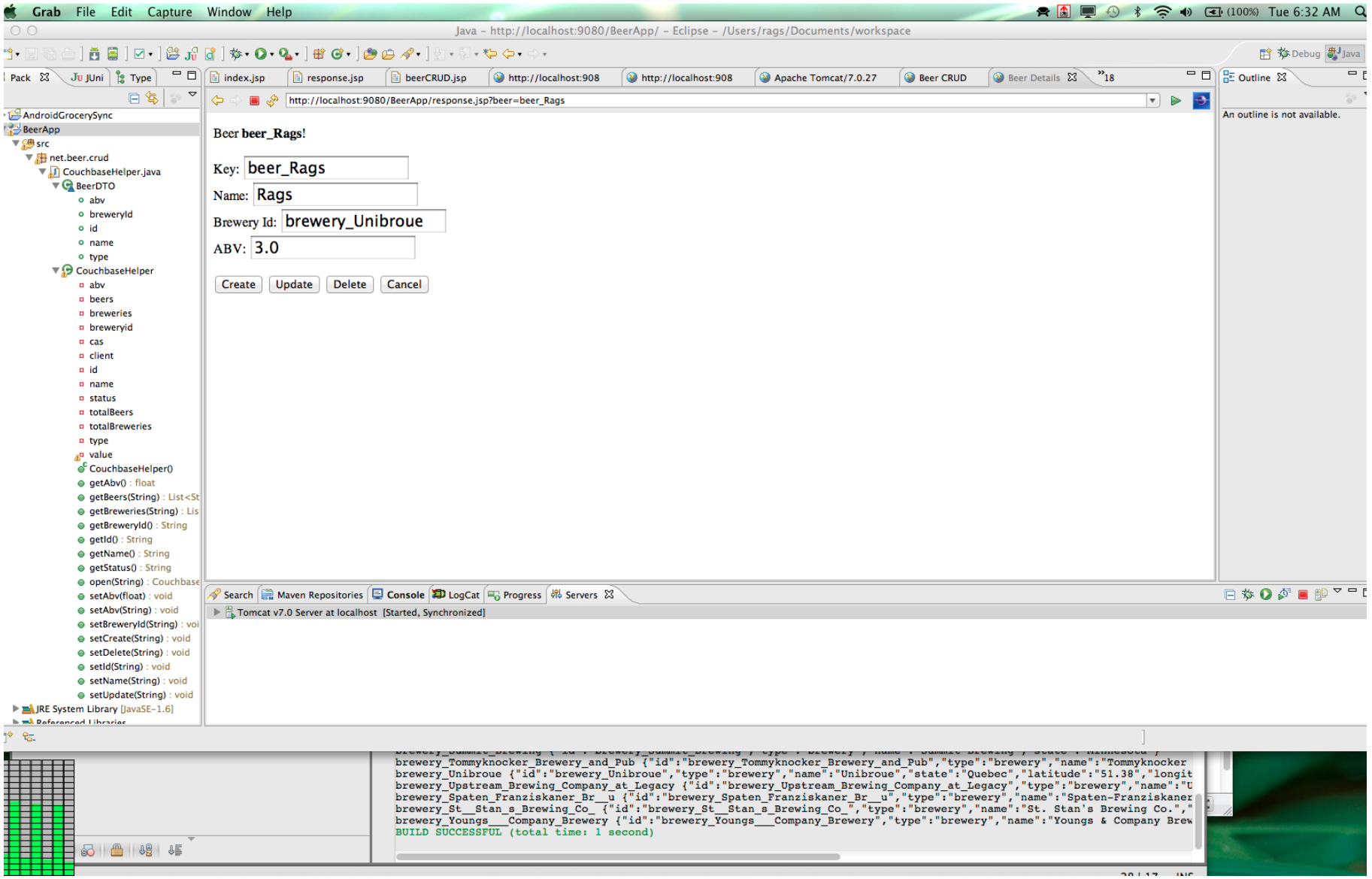
Sample Application (Screen shots)



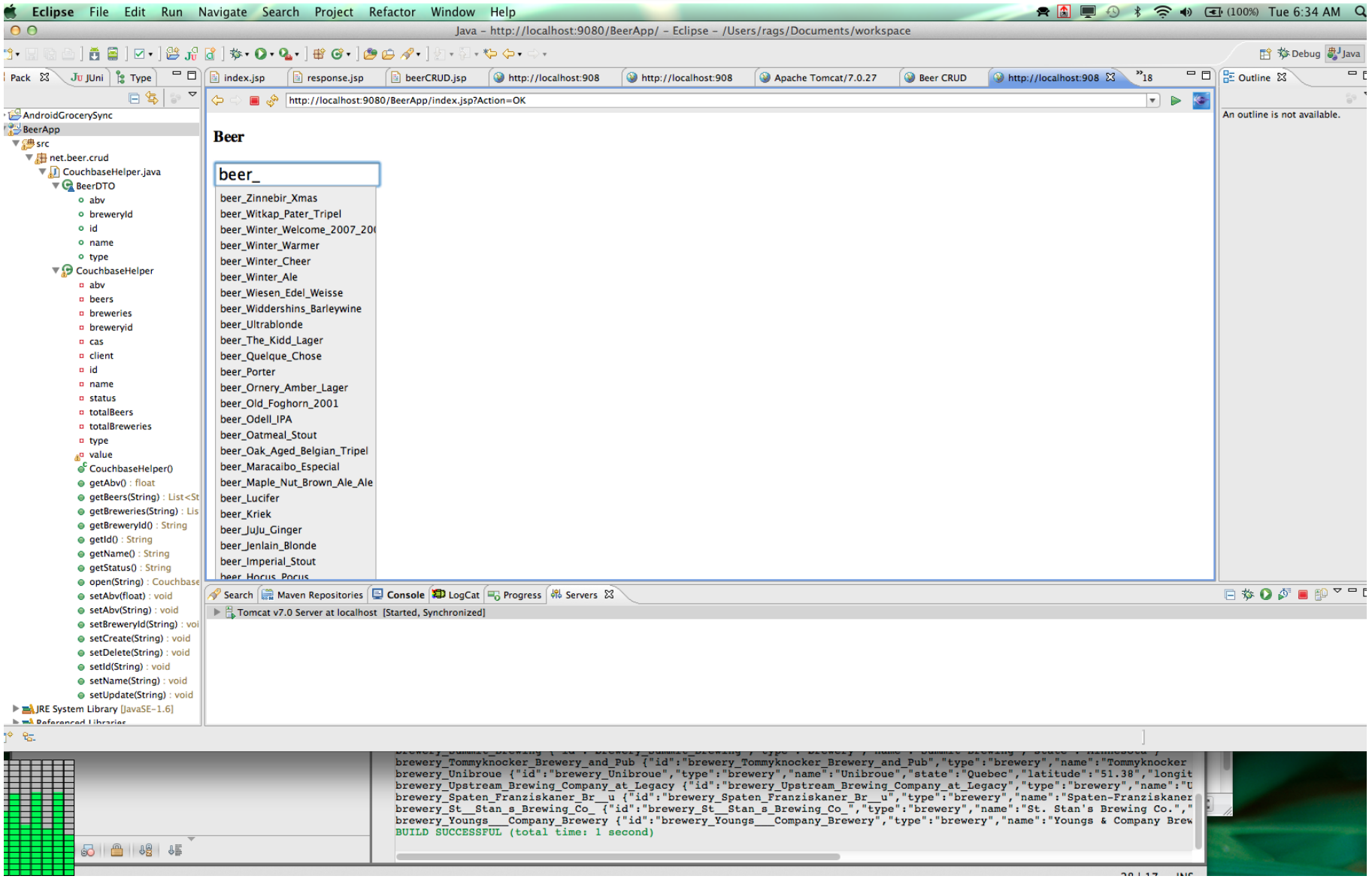
Sample Application (Screen shots)



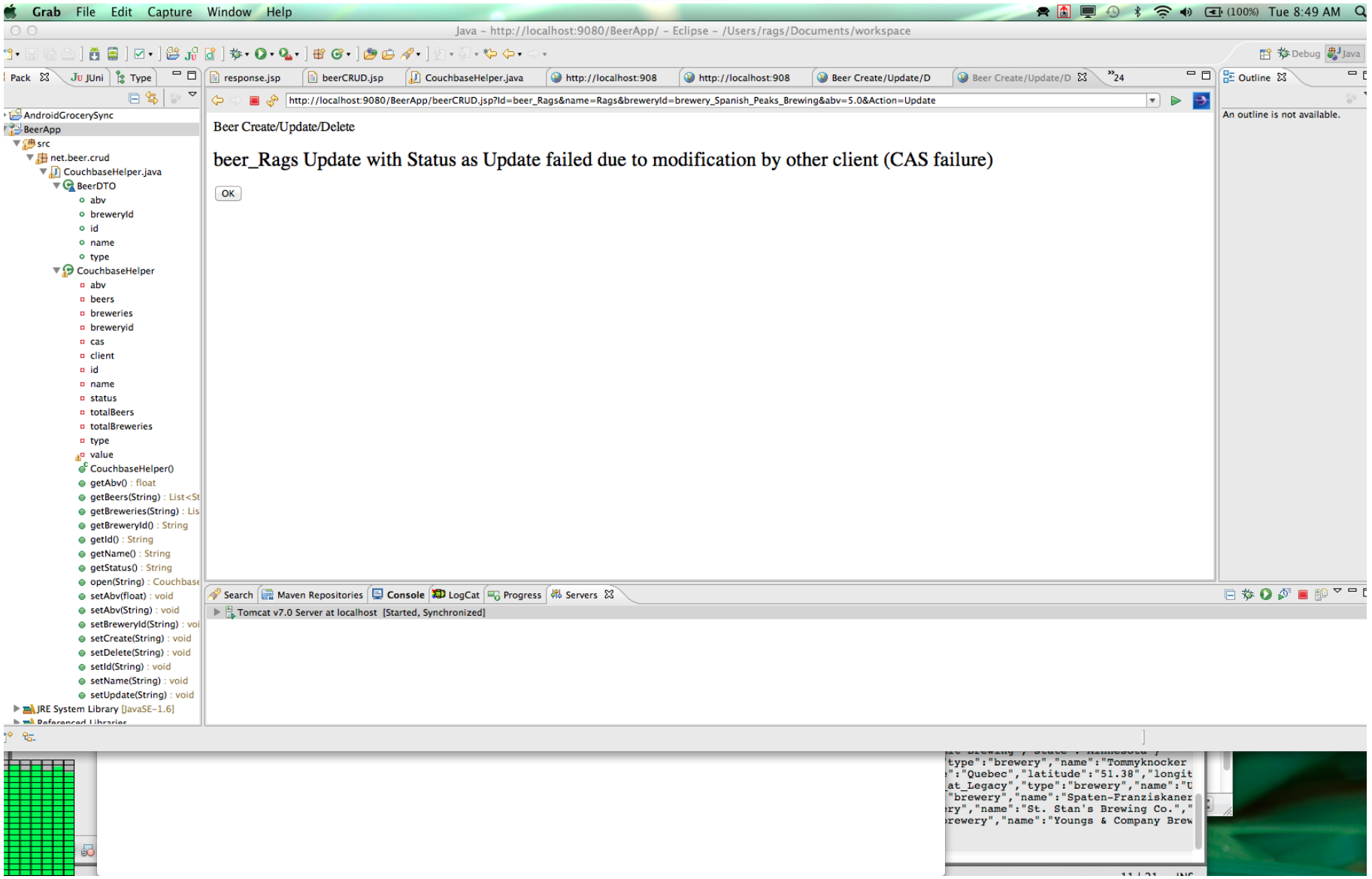
Sample Application (Screen shots)



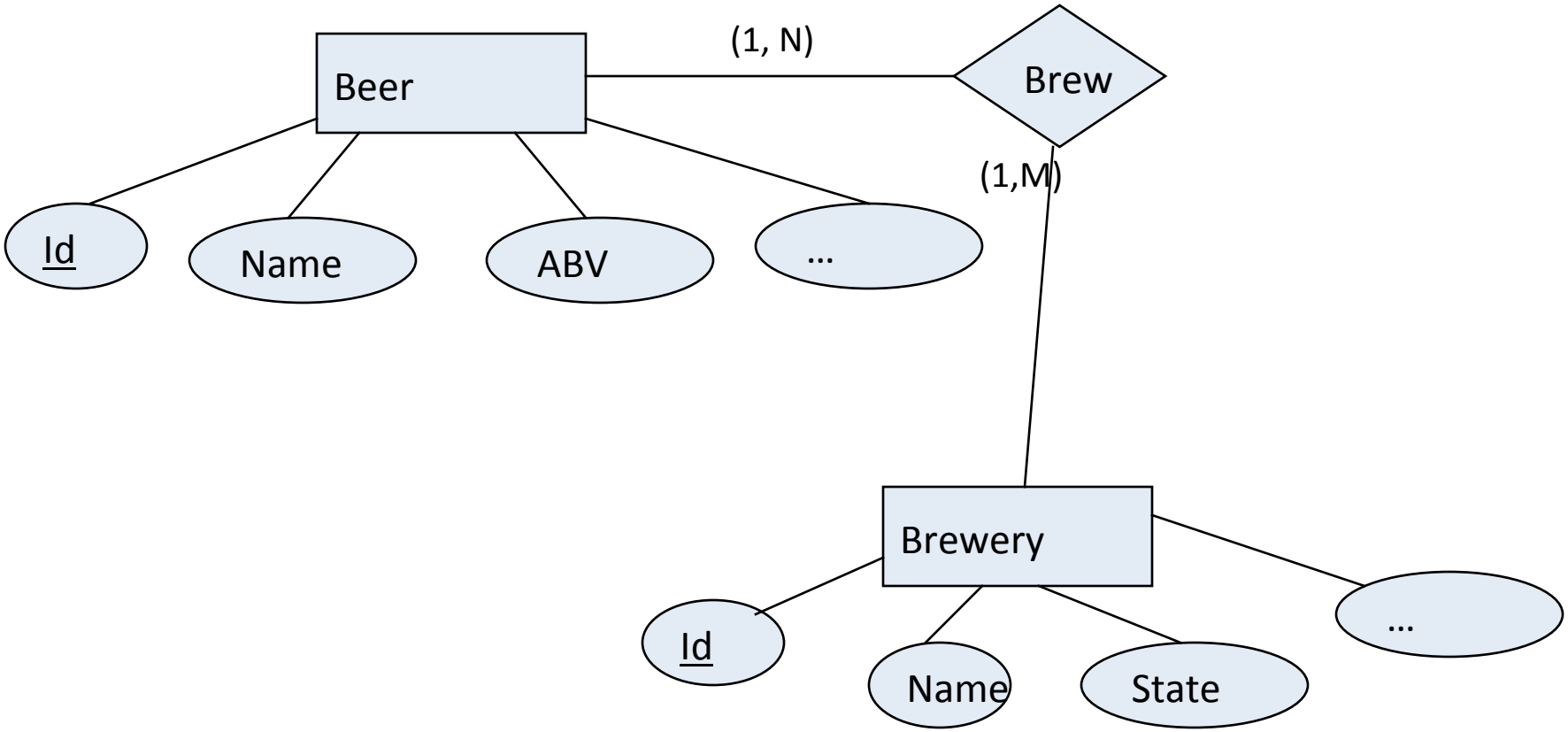
Sample Application (Screen shots)



Sample Application (Screen shots)



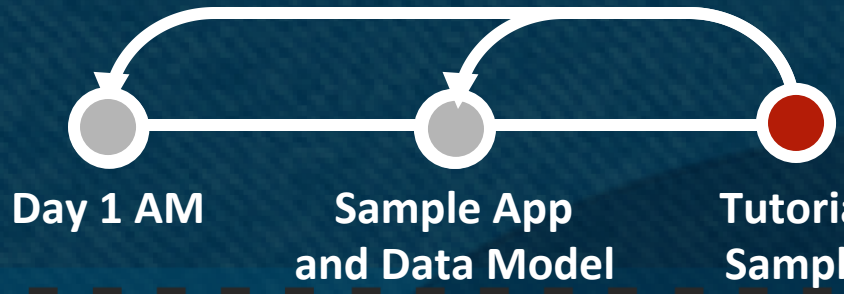
E-R Diagram for Sample Beer application



Exercise 11: Using Couchbase in a web-based app.

- Objective
  - to learn about using Couchbase in a simple web-based application
- A simple servlet/JSP-based form
  - That does simple CRUD
    - Create a Couchbase Connection
    - Use simple operations such as set, get, add, replace and cas
    - Demonstrate how you can use CAS
- Steps
  - Set up a Couchbase connection
  - Set up some Helper classes (some classes which uses Views will be provided)
  - Use Couchbase via the Helper classes

TUTORIAL EXAMPLE



Exercise 12a: Using Couchbase in a CLI-based app.

- Objective
  - to learn about using Couchbase in a Command-line based application
- Requirements
  - A multi-user Chat application
- Steps
  - Outlined later

Description

- A multi-threaded CLI-based chat application that
  - Provides the recent chat history when user joins chat for the first time
  - Lists how many users and user names (like User-1, User-2, etc.) are participating in the chat currently
  - A thread that wakes up periodically and lists the recent chats

Approach

- A multi-threaded CLI-based chat application that
  - Creates a Couchbase connection
  - Registers the user
  - Provides a count of users (past and current)
  - Shows list of current users
  - Shows list of recent chat messages
  - Persists Messages as Message-1, … Message-n for a certain amount of time (60 mins.)
  - A background thread that scans all recent chats and lists them as <User> <Message>
  - Unregisters the user
  - Shuts down the Couchbase connection

Exercise 12b

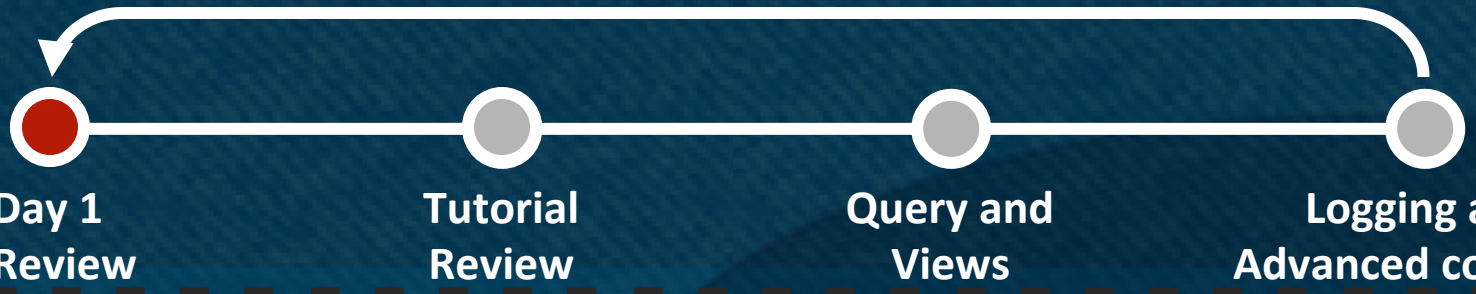
- Enhance the application
  - to print the list of users whenever a new user joins or leaves the chat room
  - to print the list of messages from a given user given an input like

    ```
    /user 4
    ```

  - Other miscellaneous features to the application

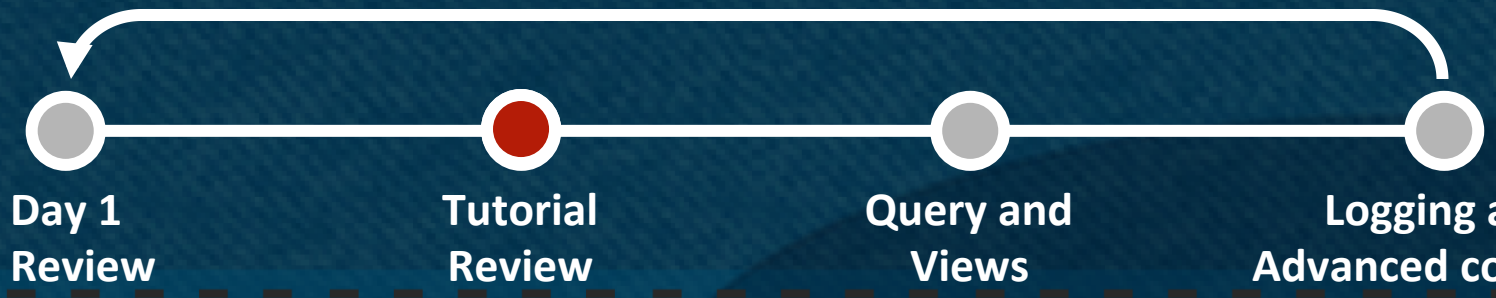
SCHEDULE

(DAY 2 MORNING)



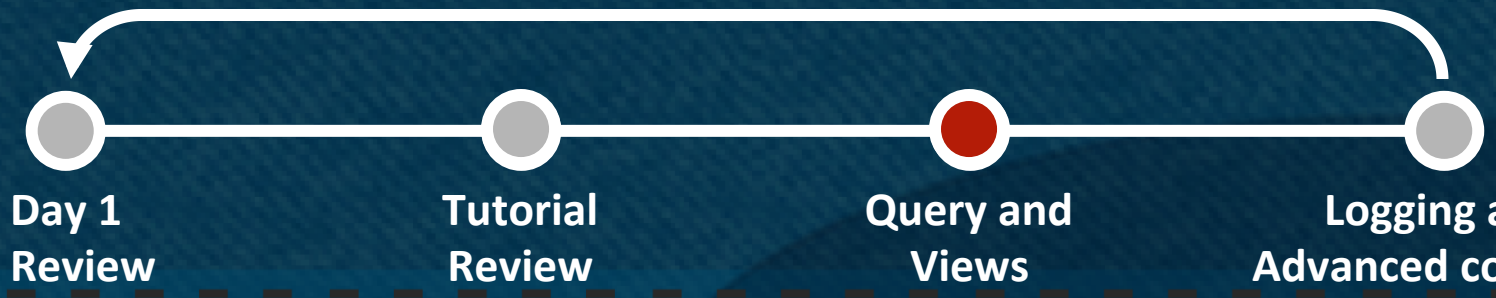
SCHEDULE

(DAY 2 MORNING)

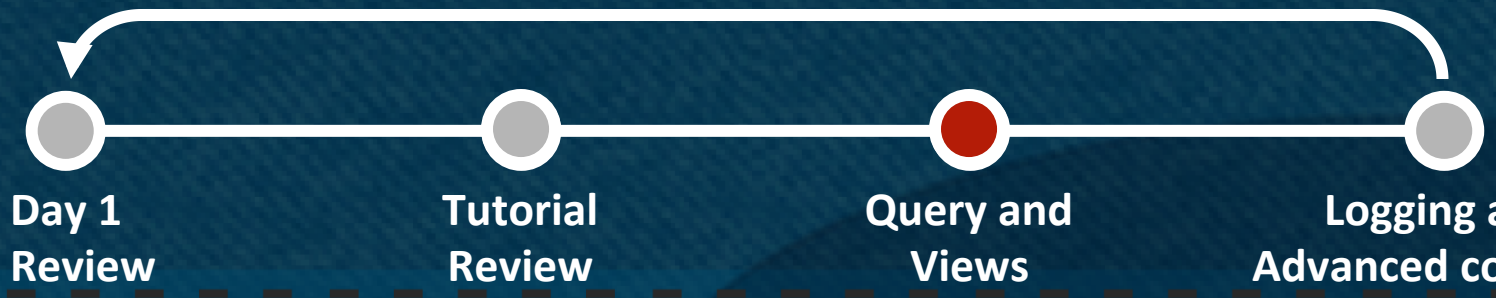


SCHEDULE

(DAY 2 MORNING)



THE VIEW AND QUERY API



What Are Views

- Views create perspectives on a collection of documents
- Views can cover a few different use cases
  - Simple secondary indexes (the most common)
  - Aggregation functions
    - Example: count the number of North American Ales
  - Organizing related data
- Use Map/Reduce
  - Map defines the relationship between fields in documents and output table
  - Reduce provides method for collating/summarizing
- Views materialized indexes
  - Views are not create until accessed
  - Data writes are fast (no index)
  - Index updates all changes since last update

Secondary Indexing and Views

UserId	Message	Type	date
<User-1>	Hello London	Message	Mon Mar-19 09:02:02
<User-2>	Hello Berlin	Message	Mon Mar-19 10:03:05
<User-3>	Hello Rome	Message	Mon Mar-19 10:05:06
<User-3>	Hello Zurich	Message	Mon Mar-19 10:06:08
<User-3>	Hello Paris	Message	Mon Mar-19 10:06:09
<User-4>	Hello Amsterdam	Message	Mon Mar-19 10:07:03
<User-5>	Hello Madrid	Message	Mon Mar-19 10:11:01

All JSON documents of Type Message since Mon Mar-19 10:00:00

Map Functions

- Map outputs one or more rows of data
- Map outputs:
  - Document ID
  - View Key (user configurable)
  - View Value
- View Key Controls
  - Sorting
  - Querying
- Key + Value stored in the Index
- Maps provide query base

Map Examples

- Basic Function

```
map(doc)
{
    emit(KEY,VALUE);
}
```

- Map by name

```
map(doc)
{
    emit(doc.name, null);
}
```

- Querying works with emitted key (must JSON value)
  - Explicit (key=XXX)
  - Key range (startkey and endkey)

Secondary Index

- Output key as array:

```
map(doc)
{
    emit([doc.name, doc.salary], null);
}
```

- Query by specifying array:
  - ?key=['brown','20000']
- Range:
  - ?startkey=['brown',0]&endkey=['thomas','30000']

Reduce

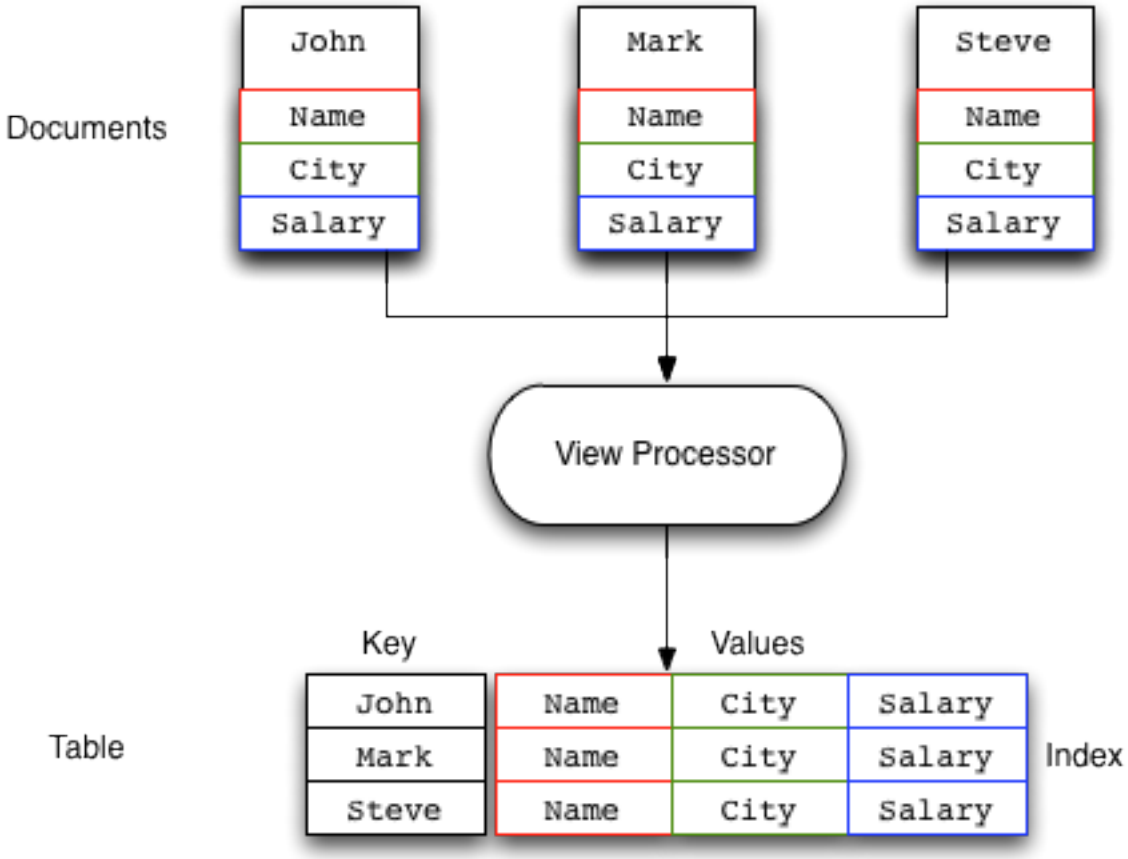
- Reduce works on output values generated by emit():

```
map(doc)
{
    emit(doc.name, doc.salary);
}
```

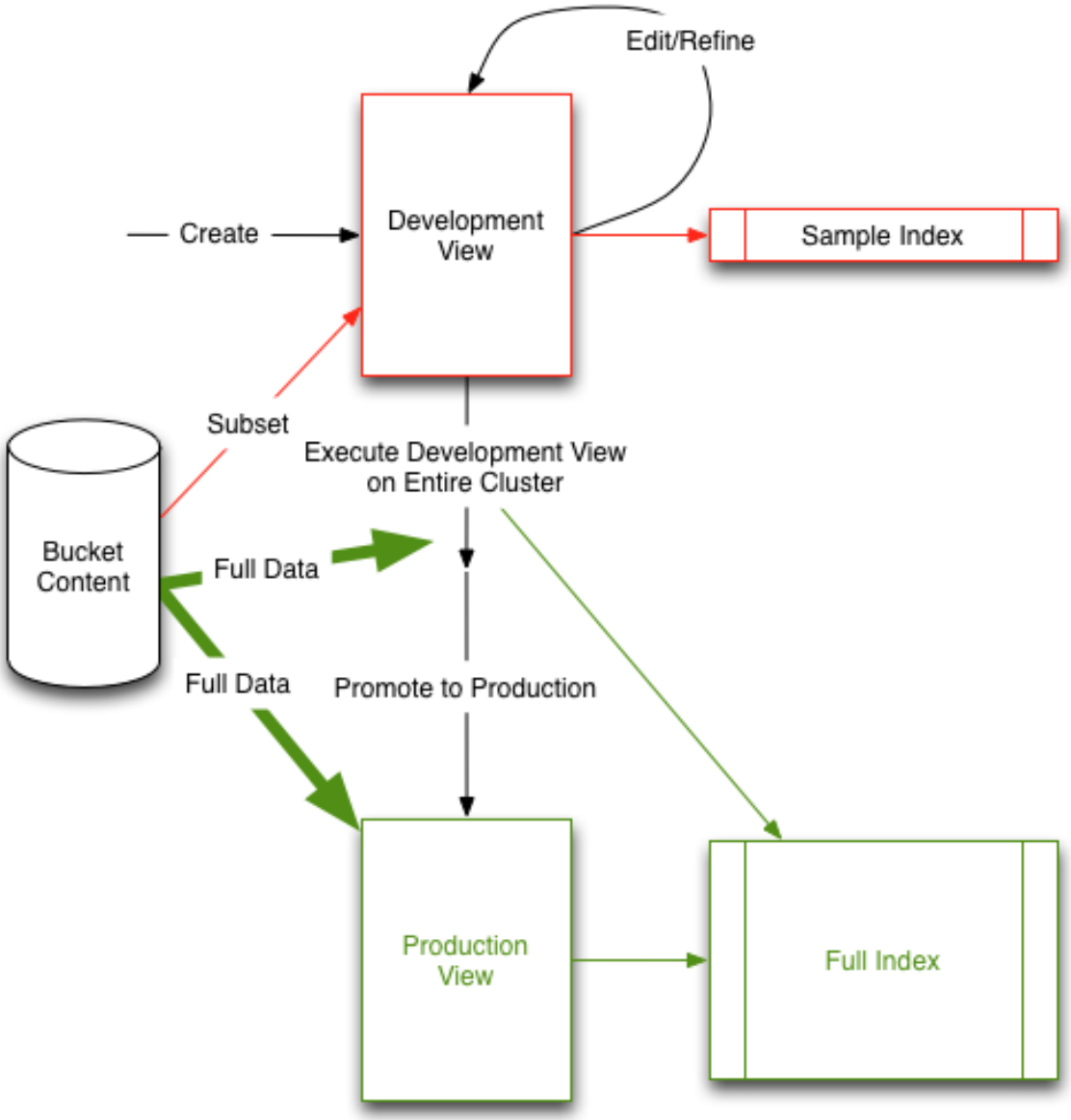
- Built-in reduce
  - _count
  - _sum
  - _stats

View Processing

- Extract fields from JSON documents and produce an index of the selected information

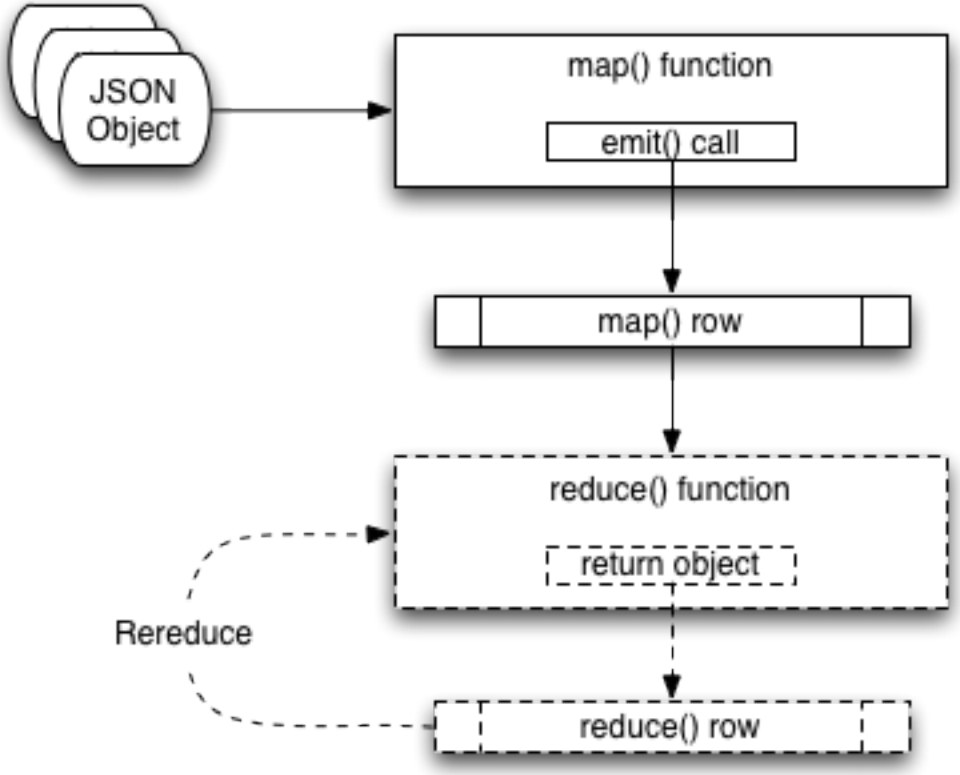


Defining Views



View Creation using incremental Map/Reduce

- map function creates a mapping between input data and output
- reduce function provides a summary (such as count, sum, etc.)



Views – adapting to changing schemas

- Original JSON document

```
{ "email" : "rags@acm.org", "name" : "Rags Srinivas" }
```

- View

```
function(doc) {
    emit([doc.name, doc.email], null);
}
```

- New JSON document

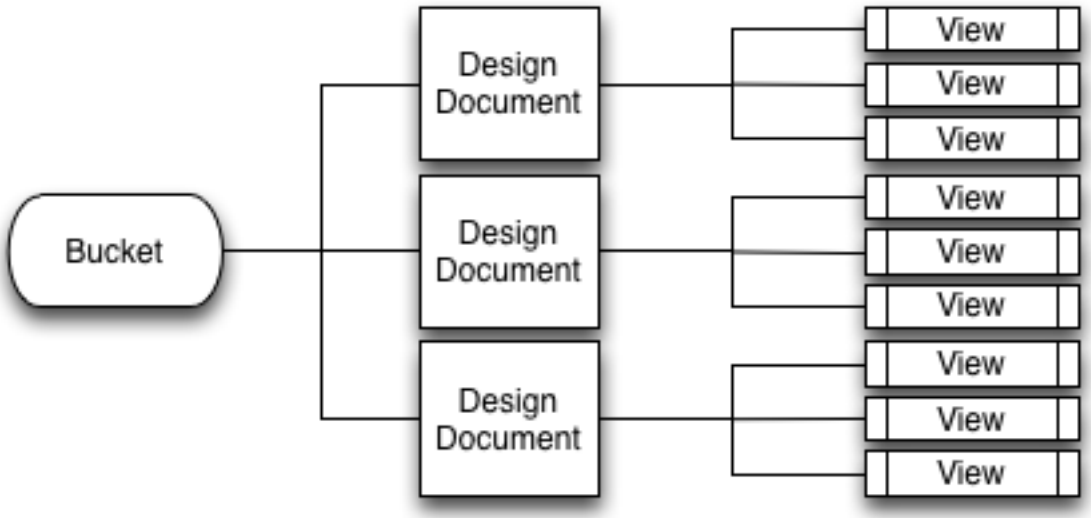
```
{ "email" : "rags@acm.org", "first" : "Rags", "last" :
"Srinivas" }
```

- New View

```
function(doc) {
if (doc.name != null) {
     emit([doc.name, doc.email], null);
} else {
    emit([doc.last + "," + doc.first, doc.email], null);
}
```

Views

- Buckets contain Design Documents that contain Views



Views – Details

- Views
  - All the views within a design document is incrementally updated when the view is accessed (lazily by default)
  - The entire view is recreated if the view definition has changed
  - Views can be conditionally updated by specifying the stale argument to the view query
  - The index information stored on disk consists of the combination of both the key and value information defined within your view.
  - During a view query, the index information for the given view query on *all* the vBuckets within the cluster is collated and returned to the client.

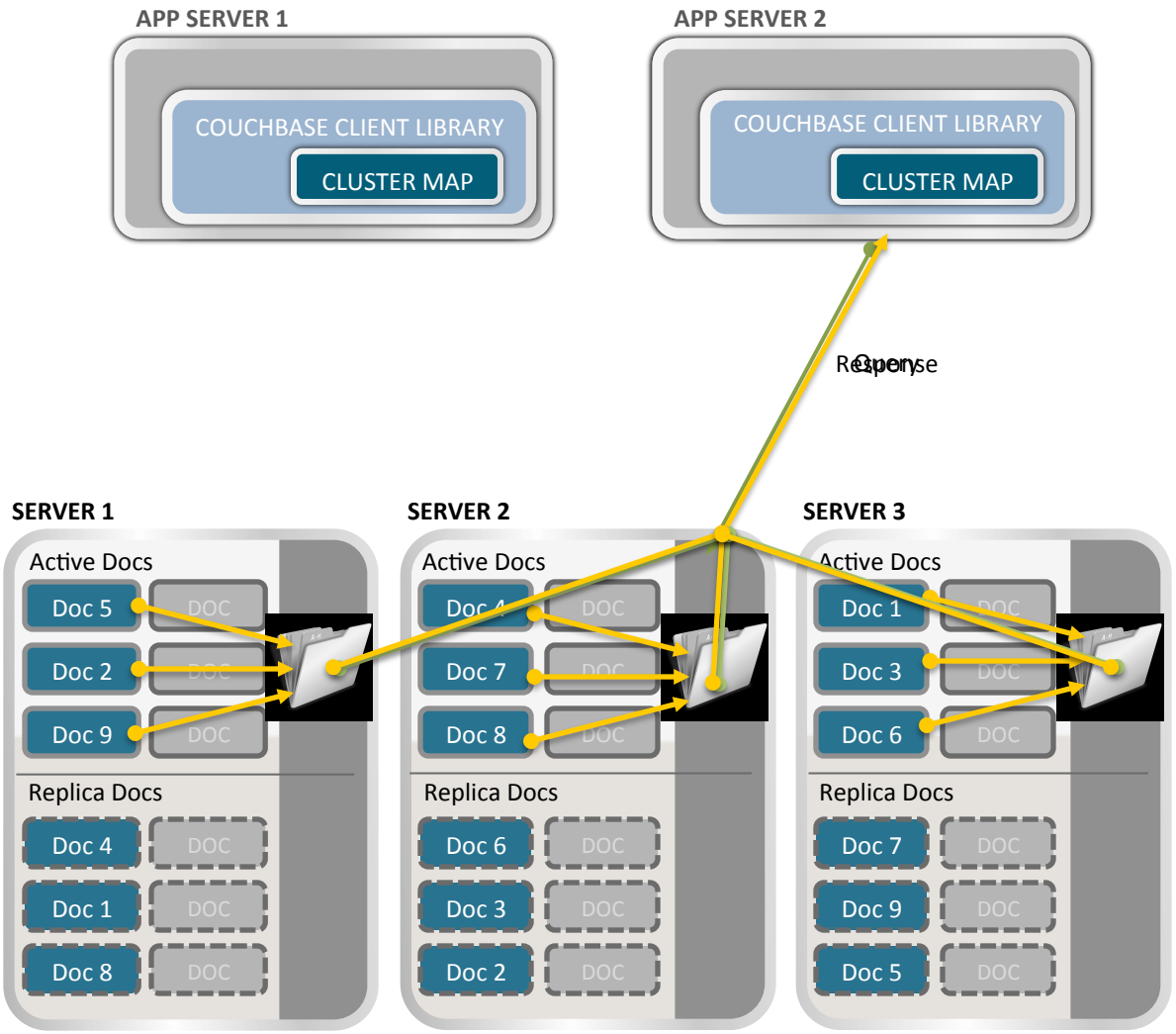
Stale Views

- Views are updated when accessed, not during data insert
- Default is to update the view after it has been accessed (update_after)
- Control through the stale query argument
  - stale=ok accesses non-updated view
  - stale=update_after updates the view after access
  - stale=false updates view before data is returned

Working with Queries and Views

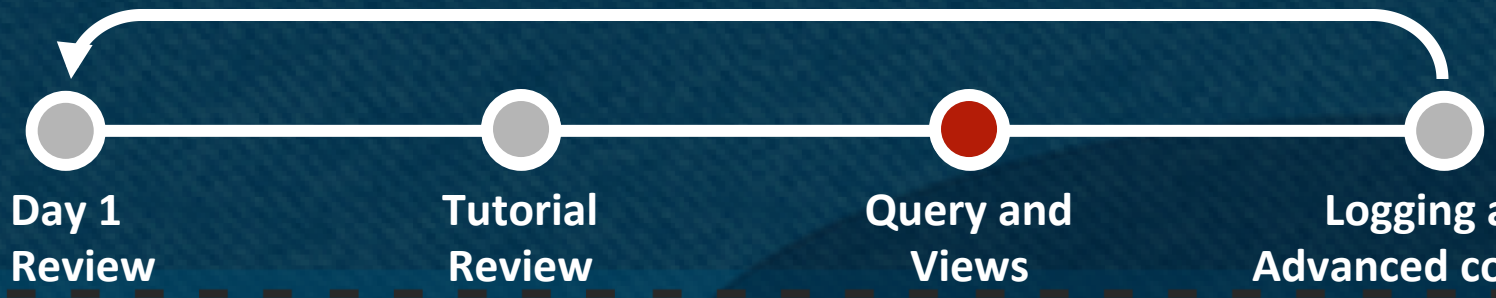
```
Date today = new Date(System.currentTimeMillis());
System.out.println("Date: " + today);
today.setHours(10);
today.setMinutes(0);

Query query = new Query();


query.setRange(String.valueOf(today.getTime()),
                String.valueOf(currentTime));
query.setReduce(false);
query.setIncludeDocs(true);
query.setStale(Stale.FALSE);

View view = client.getView("chatrags", "messages");
```

Indexing and Querying



- Indexing work is distributed amongst nodes
  - Large data set possible
  - Parallelize the effort
- Each node has index for data stored on it
- Queries combine the results from required nodes

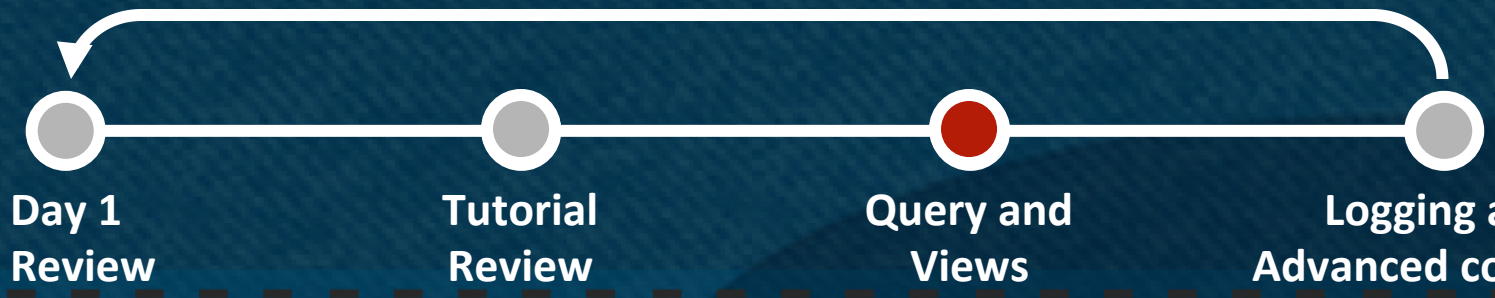
THE QUERY API



Query APIs

```
// map function
function (doc) {
  if (doc.type == "beer") {
     emit(doc._id, null);
  }
}

// Java code
Query query = new Query();

query.setReduce(false);
query.setIncludeDocs(true);
query.setStale(Stale.FALSE);
```

THE VIEW API


Day 1
Review
Tutorial
Review
Query and
Views
Logging and
Advanced concepts

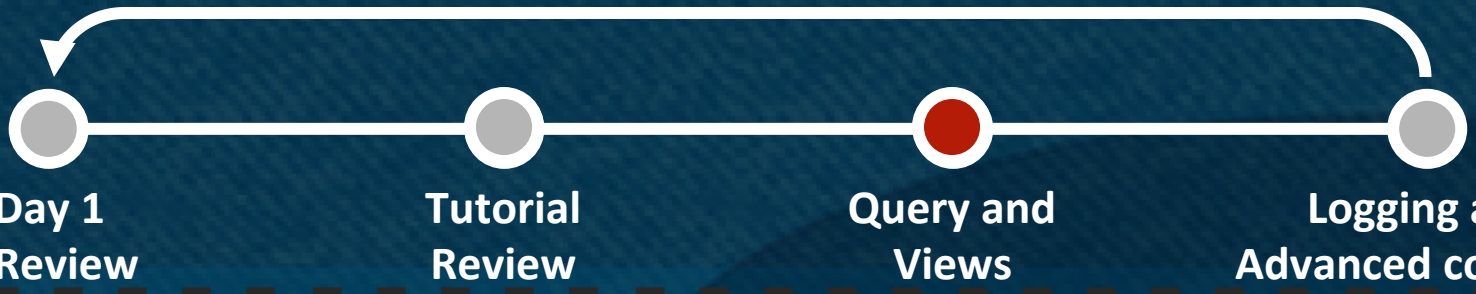
View APIs with Java

```
// map function
function (doc) {
  if (doc.type == "beer") {
     emit(doc._id, null);
  }
}

// Java code
View view = client.getView("beers", "beers");

ViewResponse result = client.query(view, query);

Iterator<ViewRow> itr = result.iterator();

while (itr.hasNext()) {
    row = itr.next();
    doc = (String) row.getDocument();
    // do something
}
```

Querying in Development/Production modes

- Querying modes
  - Development/Production modes
  - Production environment involves processes
- How to set it
  - In Java, with the properties file, command line or `System.setProperty()`

USING THE APIS



Querying with Java – Custom Limit

```
// map function
function (doc) {
  if (doc.type == "beer") {
     emit(doc._id, null);
  }
}

// Java code
View view = client.getView("beers", "beers");

Query query = new Query();

query.setReduce(false);
query.setIncludeDocs(true);
query.setStale(Stale.FALSE);
Query.setLimit(5);
```

Querying with Java – Descending Order

```
// map function
function (doc) {
  if (doc.type == "beer") {
     emit(doc.abv, null);
  }
}

// Java code
View view = client.getView("beers", "beers_by_abv");

Query query = new Query();

query.setReduce(false);
query.setIncludeDocs(true);
query.setStale(Stale.FALSE);
Query.setDescending(true);
```

Implementing a Inner Left Join

- An example
  - “Join” the beers and breweries data based on brewery Id.
- Steps
  - Define a view for one of the data documents (or table)
  - Access the view on the first document
  - Iterate over the rows
  - For each row
    - Access another view using the “foreign key” i.e. brewery ID on the second document
    - Iterate over the rows

Implementing a Join

```
View view2 = client.getView("breweries", "breweries");

Query query2 = new Query();
query2.setIncludeDocs(true);
query2.setStale(Stale.FALSE);

// Set the brewery Id and get the brewery details
query2.setKey(beer.breweryId);

ViewResponse result2 = client.query(view2, query2);

Iterator<ViewRow> itr2 = result2.iterator();

while (itr2.hasNext()) {
    row2 = itr2.next();
    doc = (String) row2.getDocument();
    // do something
}
```

Querying with Java – Custom View Key Range

```
View view2 = client.getView("breweries", "breweries");

Query query2 = new Query();
query2.setIncludeDocs(true);
query2.setStale(Stale.FALSE);

// Set the brewery Id and get the brewery details
query2.setRange(beer.breweryId, beer.breweryId);

ViewResponse result2 = client.query(view2, query2);

Iterator<ViewRow> itr2 = result2.iterator();

while (itr2.hasNext()) {
    row2 = itr2.next();
    doc = (String) row2.getDocument();
    // do something
}
```

USING THE QUERY API FOR CALCULATION

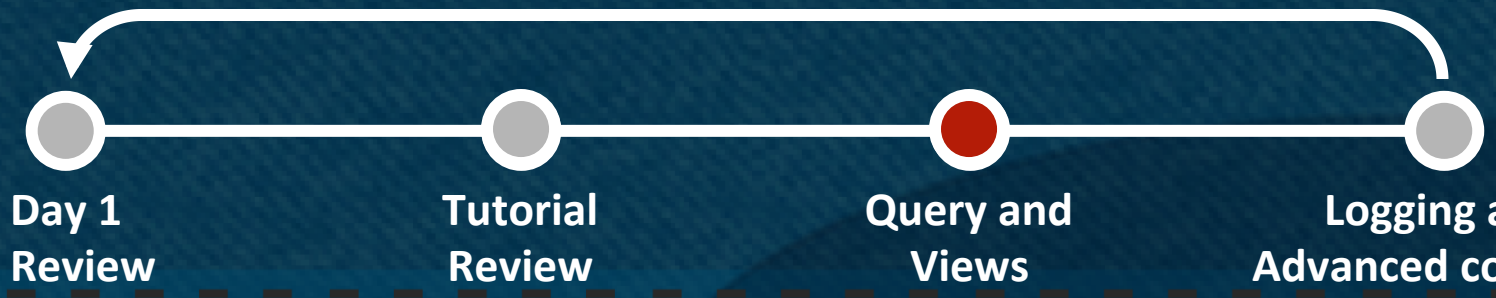
Querying with Java – Custom Reduce

```
// map function
function(doc) {
  if (doc.type == "beer") {
    if(doc.abv) {
      emit(doc.abv, 1);
    }}}

//reduce function
_count

// Java code
View view = client.getView("beers", "beers_count_abv");
query.setGroup(true);

while (itr.hasNext()) {
    row = itr.next();
    System.out.println(String.format("%s: %s”,
        row.getKey(), row.getValue()));
}
```

View Calculation Result

```
10: 2
9.6: 5
9.1: 1
9: 3
8.7: 2
8.5: 1
8: 2
7.5: 4
7: 4
6.7: 1
6.6: 1
6.2: 1
6: 2
5.9: 1
5.6: 1
5.2: 1
5: 1
4.8: 2
4.5: 1
4: 2
```

PAGING OVER VIEWS



Pagination Support

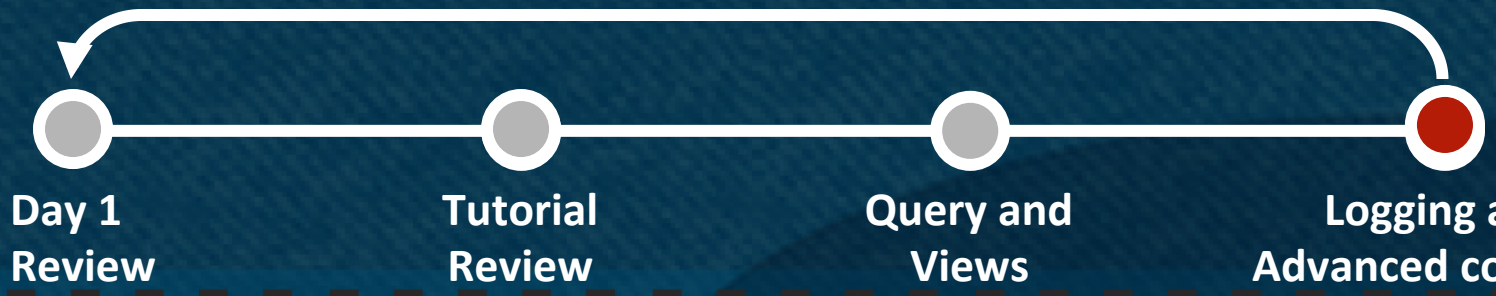
- Pagination
  - Developer needs to write a lot of boiler plate code to support pagination
  - Often error prone (repeating rows or off-by-one) which has a detrimental effect on usability and accuracy
  - Couchbase views support pagination

Querying with Java– Custom View Pagination

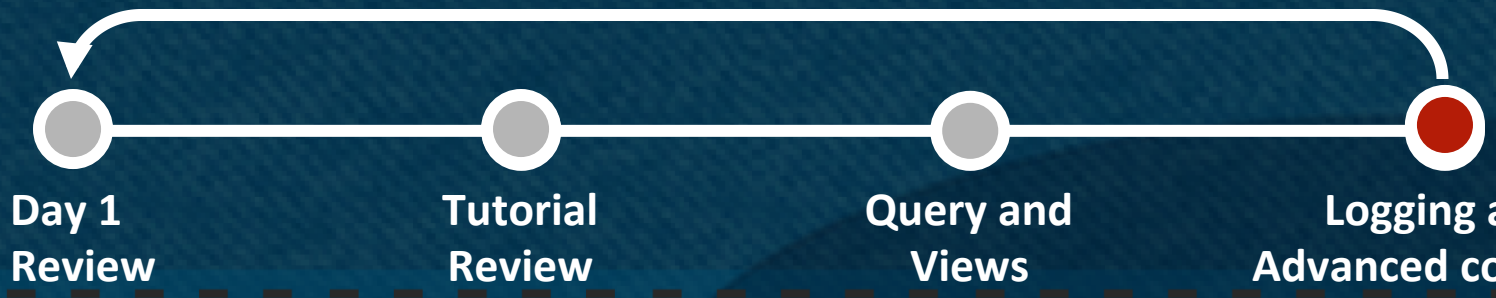
```
Paginator op = client.paginatedQuery(view, query, 10);

ViewRow row;
int count=0;

while (op.hasNext()) {
    row = op.next();

    System.out.println(String.format("Key/value is %s: %s”,
        row.getKey(), row.getValue()));
    String doc = (String) client.get(row.getId());

    BeerDTO beer = new Gson().fromJson(doc, BeerDTO.class);

    System.out.println(String.format("%s: %s %s %f", beer.id,
                beer.name, beer.breweryId, beer.abv));
}
```

SCHEDULE

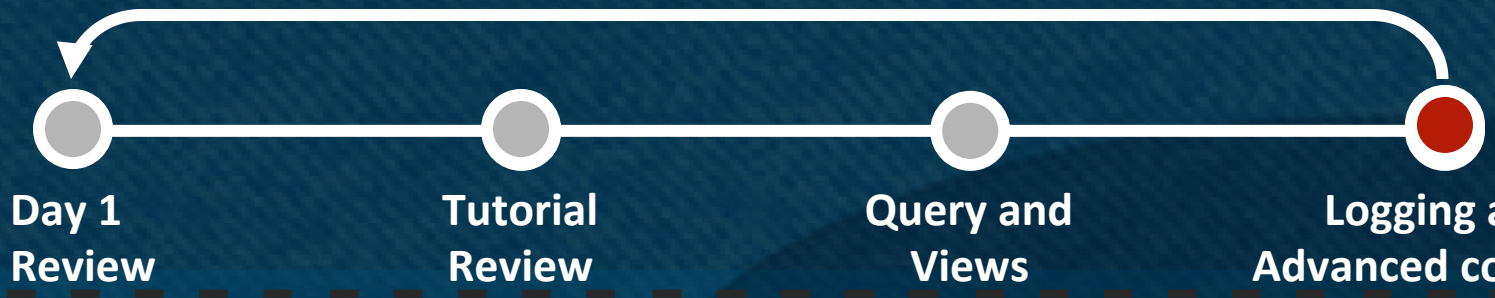
(DAY 2 MORNING)



ADVANCED CONCEPTS



LOGGING


Day 1
Review
Tutorial
Review
Query and
Views
Logging and
Advanced concepts

Logging – Configuration

- Configuring Logging
  - There are two logging systems if the customer is using couchbase-client and spymemcached
    - Couchbase client uses JDK logging mostly, except where it extends spymemcached
    - spymemcached uses it's own internal logging by default, which can be passed over to JDK logging or log4j
- In the future, we plan to consolidate around something (likely, slf4j)

Logging – Configuration

- Logging type
  - JDK logging

    -Dnet.spy.log.LoggerImpl=net.spy.memcached.compat.log.SunLogger
  - log4j logging

    -Dnet.spy.log.LoggerImpl=net.spy.memcached.compat.log.Log4JLogger

Logging Levels

- JDK logging
  - SEVERE
  - WARNING
  - INFO
  - CONFIG
  - FINE
  - FINER
  - FINEST
- Log4j logging
  - ALL
  - DEBUG
  - ERROR
  - FATAL
  - INFO
  - OFF
  - TRACE
  - WARN

Logging

- Programmatically telling Spy to use JDK logging (for example)

```
// Tell spy to use the JDK logging
Properties systemProperties = System.getProperties();
systemProperties.put("net.spy.log.LoggerImpl",
        "net.spy.memcached.compat.log.SunLogger");
System.setProperties(systemProperties);

Logger topLogger = java.util.logging.Logger.getLogger("");
```

Logging

- Setting consolehandler

```
// Handler for console (reuse it if it already exists)
Handler consoleHandler = null;
//see if there is already a console handler
for (Handler handler : topLogger.getHandlers()) {
  if (handler instanceof ConsoleHandler) {
    //found the console handler
    consoleHandler = handler;
    break;
  }
}
```

Logging (cont'd)

```
if (consoleHandler == null) {
  //there was no console handler found, create a new one
  consoleHandler = new ConsoleHandler();
  topLogger.addHandler(consoleHandler);
}

//set the console handler to finest:
consoleHandler.setLevel(java.util.logging.Level.FINEST);

Logger.getLogger("net.spy.memcached").setLevel(Level.FINEST);
Logger.getLogger("com.couchbase.client").setLevel(Level.FINEST);
```

Logging (with command line parameters)

- Command Line Parameter

```
-Djava.util.logging.config.file=logging.properties
```

- logging.properties file

```
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter
com.couchbase.client.vbucket.level = FINEST
com.couchbase.client.vbucket.config.level = FINEST
com.couchbase.client.level = FINEST
```

Logging – Interpreting logs

- Interpreting logs, high level
  - Multiple threads will log, one from the caller (i.e., the webapp container, etc.) and some things from the client library IO threads.
  - What does a server failure look like
    - Operation would timeout
    - MemcachedNode would start complaining and reconnecting
    - If the cluster node that we were listening to, the thread listening to cluster sees failure, start whining about reconnecting
  - What are all of these "wanted 2 got 4" messages in the log? Basic verbose log checking.

Logging Samples - INFO

- Connecting to a 2-node cluster

```
Apr 19, 2012 12:01:14 PM net.spy.memcached.MemcachedConnection
createConnections
INFO: Added {QA sa=/192.168.3.114:11210, #Rops=0, #Wops=0, #iq=0,
topRop=null, topWop=null, toWrite=0, interested=0} to connect queue
Apr 19, 2012 12:01:14 PM net.spy.memcached.MemcachedConnection
createConnections
INFO: Added {QA sa=/192.168.3.113:11210, #Rops=0, #Wops=0, #iq=0,
topRop=null, topWop=null, toWrite=0, interested=0} to connect queue
```

Logging Samples – FINE

- Logs showing an operation

Logging Samples – FINE

```
FINE: Added Cmd: 1 Opaque: 1 Key: 0000000001RAG Cas: 0 Exp: 3 Flags: 0
Data Length: 24 to {QA sa=ubuntu/192.168.3.113:11210, #Rops=0,
#Wops=0, #iq=1, topRop=null, topWop=null, toWrite=0, interested=0}
Apr 19, 2012 12:01:15 PM net.spy.memcached.MemcachedConnection
handleIO
FINE: No selectors ready, interrupted: false
Apr 19, 2012 12:01:15 PM net.spy.memcached.MemcachedConnection
handleInputQueue
FINE: Handling queue
Apr 19, 2012 12:01:15 PM net.spy.memcached.MemcachedConnection
addOperation
FINE: Added Cmd: 0 Opaque: 2 Key: 0000000001RAG to {QA sa=ubuntu/
192.168.3.113:11210, #Rops=0, #Wops=0, #iq=0, topRop=null, topWop=Cmd:
1 Opaque: 1 Key: 0000000001RAG Cas: 0 Exp: 3 Flags: 0 Data Length: 24,
toWrite=0, interested=0}
Apr 19, 2012 12:01:15 PM
net.spy.memcached.protocol.TCPMemcachedNodeImpl fixupOps
FINE: Setting interested opts to 4
Apr 19, 2012 12:01:15 PM net.spy.memcached.MemcachedConnection
handleIO
FINE: Done dealing with queue.
```

Logging Samples – WARNING

- Showing a node down

Logging Samples – WARNING

```
INFO: Reconnecting due to exception on {QA sa=centos/
192.168.3.114:11210, #Rops=2, #Wops=0, #iq=0, topRop=Cmd: 1 Opaque: 4
Key: 0000000002RAG Cas: 0 Exp: 3 Flags: 0 Data Length: 24,
topWop=null, toWrite=0, interested=1}
java.io.IOException: Disconnected unexpected, will reconnect.
net.spy.memcached.MemcachedConnection.handleReads(MemcachedConnection.
java:452)
net.spy.memcached.MemcachedConnection.handleIO(MemcachedConnection.jav
a:380)
net.spy.memcached.MemcachedConnection.handleIO(MemcachedConnection.jav
a:242)
com.couchbase.client.CouchbaseConnection.run(CouchbaseConnection.java:
283)
Apr 19, 2012 12:31:11 PM net.spy.memcached.MemcachedConnection
queueReconnect
WARNING: Closing, and reopening {QA sa=centos/192.168.3.114:11210,
#Rops=2, #Wops=0, #iq=0, topRop=Cmd: 1 Opaque: 4 Key: 0000000002RAG
Cas: 0 Exp: 3 Flags: 0 Data Length: 24, topWop=null, toWrite=0,
interested=1}, attempt 0.
```

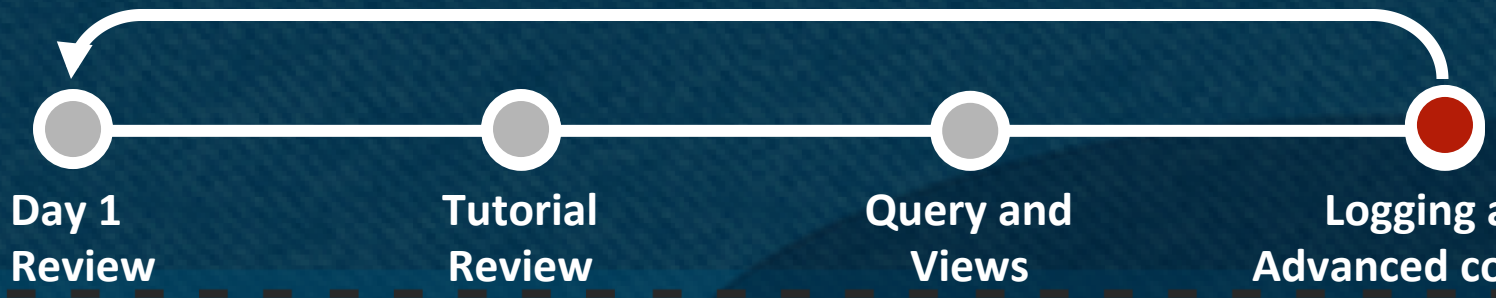
Logging Samples – WARNING (Cont'd)

```
Apr 19, 2012 12:31:11 PM
net.spy.memcached.protocol.TCPMemcachedNodeImpl setupResend
WARNING: Discarding partially completed op: Cmd: 1 Opaque: 4 Key:
0000000002RAG Cas: 0 Exp: 3 Flags: 0 Data Length: 24
Apr 19, 2012 12:31:11 PM net.spy.memcached.protocol.BaseOperationImpl
wasCancelled
FINE: was cancelled.
Apr 19, 2012 12:31:11 PM
net.spy.memcached.protocol.TCPMemcachedNodeImpl setupResend
WARNING: Discarding partially completed op: Cmd: 0 Opaque: 5 Key:
0000000002RAG
Apr 19, 2012 12:31:11 PM net.spy.memcached.protocol.BaseOperationImpl
wasCancelled
FINE: was cancelled.
```

Exercise 13

- Include JDK Logging in any of your earlier programs and try different levels of logging

A PEEK INTO TUNING



Setting connection parameters

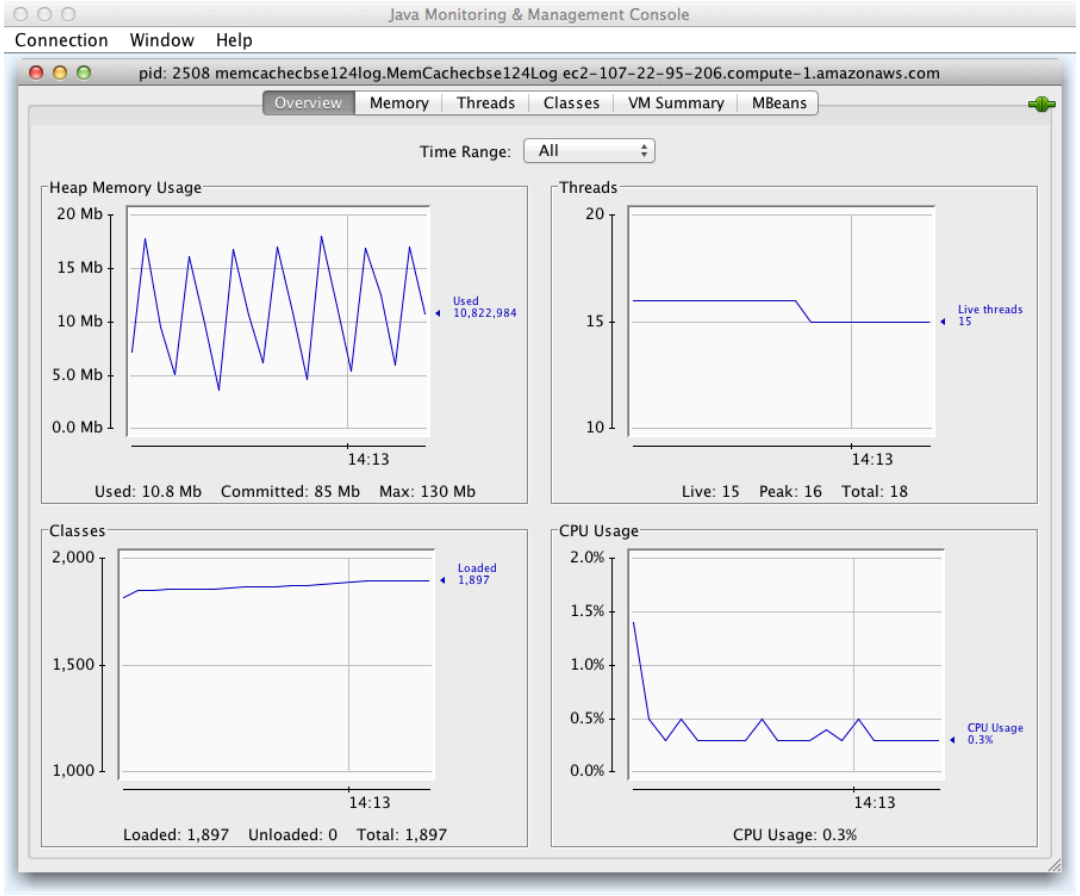
Parameter	Description	Default
`MaxReconnectDelay`	Maximum number of milliseconds to wait between reconnect attempts.	30 secs.
`ShouldOptimize`	If true, low-level optimization is in effect.	true
`OpQueueMaxBlockTime`	Get the maximum amount of time (in milliseconds) a client is willing to wait to add a new item to a queue.	10 millisecs
`OpTimeout`	Operation timeout used by this connection.	2.5 secs
`TimeoutExceptionThreshold`	Maximum number of timeout exception for shutdown connection.	998

Setting connection parameters

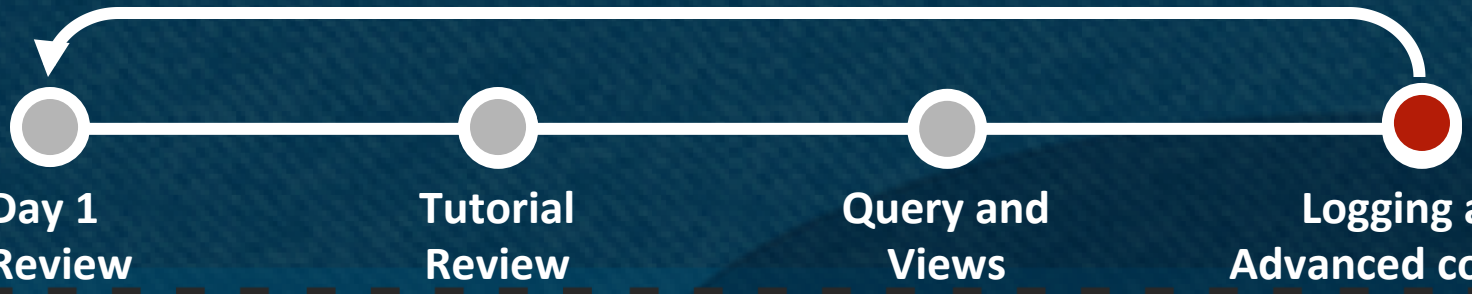
```
CouchbaseConnectionFactoryBuilder cfb = new
CouchbaseConnectionFactoryBuilder();

// wait up to 10 seconds for an operation to succeed
cfb.setOpTimeout(10000);

// wait up to 5 seconds when trying to enqueue an operation
cfb.setOpQueueMaxBlockTime(5000);

// Don't optimize – more network traffic but lower latency
cfb.setShouldOptimize(false);

// Lower the Timeout exception threshold
cfb.setTimeoutExceptionThreshold(100);
```

JVM Tuning

- Garbage Collection
  - **`-XX:+UseConcMarkSweepGC`**
  - **`-XX:MaxGCPauseMillis=850`**
- Heap Size
- Jconsole



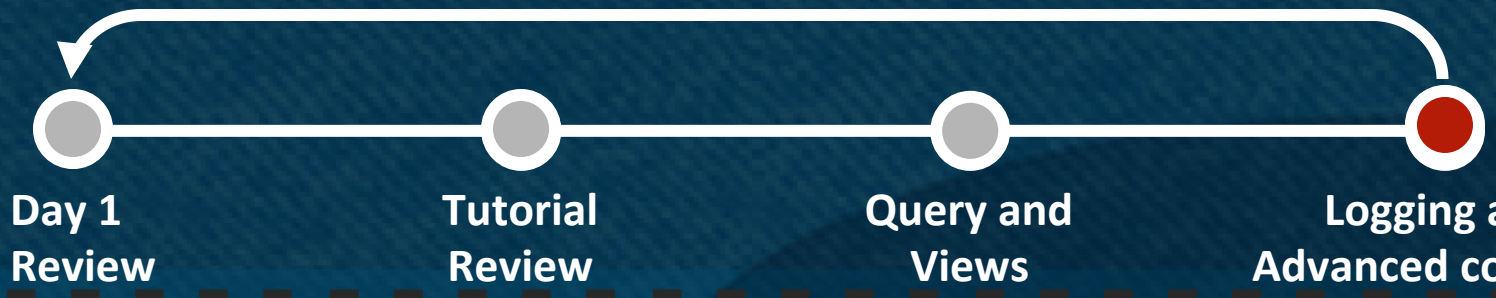
THREAD SAFETY



Thread safety

- Java Client Library is thread-safe
  - Couchbase client objects can be shared among multiple threads (reentrant)
- Multiple Couchbase client objects in multiple threads (connection pooling) are not normally required

TOPOLOGY CHANGES



Topology Changes

- A host (re)joins the connection
  - A reconfiguration happens and the node becomes available.
- The host shuts down the connection (TCP RST)
  - A reconfiguration happens and the node is marked as down. A reconnect will be attempted.
- The host just disappears, but the connection had already been built and was previously active
- The host isn't there (no TCP SYN-ACK)
  - After 1000 consecutive timeouts, we drop the connection and try to reconnect

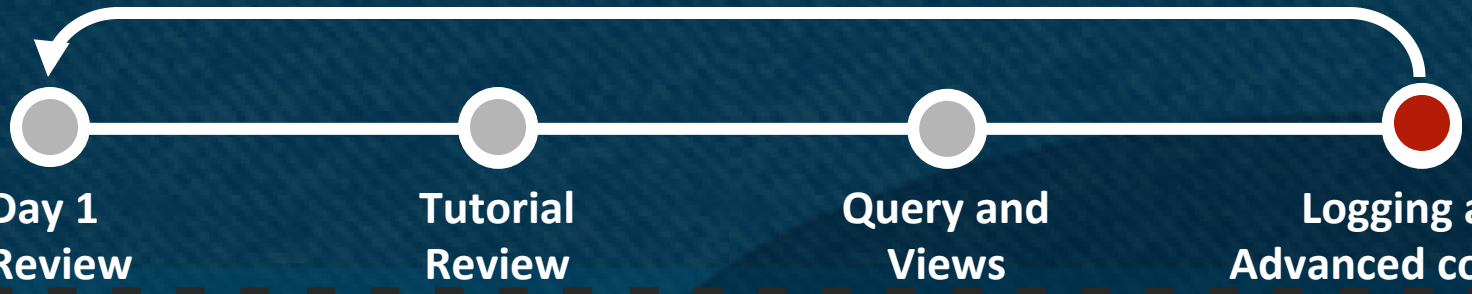
Logging Samples – WARNING

```
INFO: Reconnecting due to exception on {QA sa=centos/
192.168.3.114:11210, #Rops=2, #Wops=0, #iq=0, topRop=Cmd: 1 Opaque: 4
Key: 0000000002RAG Cas: 0 Exp: 3 Flags: 0 Data Length: 24,
topWop=null, toWrite=0, interested=1}
java.io.IOException: Disconnected unexpected, will reconnect.
net.spy.memcached.MemcachedConnection.handleReads(MemcachedConnection.
java:452)
net.spy.memcached.MemcachedConnection.handleIO(MemcachedConnection.jav
a:380)
net.spy.memcached.MemcachedConnection.handleIO(MemcachedConnection.jav
a:242)
com.couchbase.client.CouchbaseConnection.run(CouchbaseConnection.java:
283)
Apr 19, 2012 12:31:11 PM net.spy.memcached.MemcachedConnection
queueReconnect
WARNING: Closing, and reopening {QA sa=centos/192.168.3.114:11210,
#Rops=2, #Wops=0, #iq=0, topRop=Cmd: 1 Opaque: 4 Key: 0000000002RAG
Cas: 0 Exp: 3 Flags: 0 Data Length: 24, topWop=null, toWrite=0,
interested=1}, attempt 0.
```

Logging Samples – WARNING (Cont'd)

```
Apr 19, 2012 12:31:11 PM
net.spy.memcached.protocol.TCPMemcachedNodeImpl setupResend
WARNING: Discarding partially completed op: Cmd: 1 Opaque: 4 Key:
0000000002RAG Cas: 0 Exp: 3 Flags: 0 Data Length: 24
Apr 19, 2012 12:31:11 PM net.spy.memcached.protocol.BaseOperationImpl
wasCancelled
FINE: was cancelled.
Apr 19, 2012 12:31:11 PM
net.spy.memcached.protocol.TCPMemcachedNodeImpl setupResend
WARNING: Discarding partially completed op: Cmd: 0 Opaque: 5 Key:
0000000002RAG
Apr 19, 2012 12:31:11 PM net.spy.memcached.protocol.BaseOperationImpl
wasCancelled
FINE: was cancelled.
```

TECHNIQUES (EXPONENTIAL BACKOFF)



Exponential backoff

```
 public OperationFuture<Boolean> contSet(String key, int exp,
        Object value, int tries) {
…
      do {
        if (backoffexp > tries) {
          throw new RuntimeException("Could not perform a set after "
                  + tries + " tries.");
        }
        result = cbc.set(key, exp, value);
        if (result.getStatus().isSuccess()) {
          break;
        }
        if (backoffexp > 0) {
          double backoffMillis = Math.pow(2, backoffexp);
          backoffMillis = Math.min(1000, backoffMillis); // 1 sec max
          Thread.sleep((int) backoffMillis);
        }
        backoffexp++;

      } while (status.getMessage().equals("Temporary failure"));
```

SCHEDULE

(DAY 2 AFTERNOON)

Views Exercises

Tutorial for Views

Sample App. enhancement

Final Review and Conclusions

Exercise 14a: Listing Beer and Brewery Data

- Objective
  - Listing the Beer and Brewery Data using Views
- Steps
  - Setup Views on the server (in development) for listing Beer and Brewery Data
  - Write a program to
    - Set up a Couchbase connection
    - Use the views to display the data and try different options
    - Shutdown the connection

Exercise 14b: Implementing a Join

- Objective
  - Implementing a left inner join
- Steps
  - Use the views setup earlier
  - Write a program to
    - Set up a Couchbase connection
    - Use the views to implement an inner left join using the brewery Id
    - Shutdown the connection

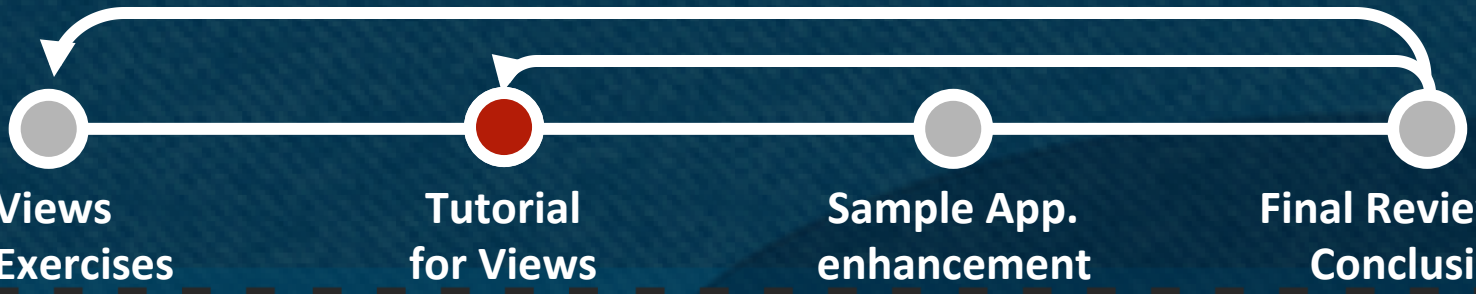
Exercise 14c: Listing Beer by ABV

- Objective
  - Using views in Java
- Steps
  - Setup a view to list Beers by ABV
  - Write a program to
    - Set up a Couchbase connection
    - Use the view to list Beers by ABV
    - Shutdown the connection

Exercise 14d: Counting Beers by ABV

- Objective
  - Using Map and Reduce functions
- Steps
  - Write a view that includes both the Map and Reduce function that counts Beers by ABV
  - Write a program to
    - Set up a Couchbase connection
    - Use the view to list the information
    - Shutdown the connection

EXTENDING TUTORIAL FOR VIEWS



Exercise 15: Enhance Chat Application for Views

- Objective
  - to learn about using Views
- Prerequisite
  - A multi-user Chat application
- Steps
  - Outlined later

Description

- Define Views on the Server
- Define a Message class to encapsulate the JSON object (and a User class as required)
- Enhance the multi-threaded CLI-based chat application that
  - Provides the recent chat history when user joins chat for the first time based on Views
  - Lists how many users and user names (like User-1, User-2, etc.) are participating in the chat currently using Views
  - A thread that wakes up periodically and lists the recent chats using Views

Approach

- A multi-threaded CLI-based chat application that
  - Creates a Couchbase connection
  - Registers the user
  - Provides a count of users (past and current)
  - Shows list of current users
  - Shows list of recent chat messages
  - Persists Messages as Message-1, … Message-n for a certain amount of time (60 mins.) as JSON document (Message)
  - A background thread that uses Views and outputs them <User> <Message>
  - Unregisters the user
  - Shuts down the Couchbase connection

Extending Tutorial for Views (Step 1)

```
// Create a Message Class
public class Message {
    public String userId;
    public String message;
    public String type = "Message";
    public long date;
    public Message(String userId, String message) {
        super();
        this.userId = userId;
        this.message = message;
        date = System.currentTimeMillis();
    }
}
```

Extending Tutorial for Views (Step 2)

```
// Store a JSON message to the chat
long messageId = client.incr("Messages", 1, 1);
// Write out in JSON
String json = new Gson().toJson(new
    Message(getUserNameToken(), input));
client.set("Message:" + messageId, 3600, json);
```

Extending Tutorial for Views (Step 3)

```
// Write the View on the server

function (doc) {
  if (doc.type == "Message")
    if (doc.date != null)
      emit(doc.date.toString());
}

// Reduce

_count
```

Extending Tutorial for Views (Step 4)

```
// Use the View
query.setRange(String.valueOf(lastMessageSeen),
                String.valueOf(currentTime));

ViewResponse result = client.query(view, query);

Iterator<ViewRow> itr = result.iterator();
ViewRow row;

while (itr.hasNext()) {
    row = itr.next();
    String doc = (String) row.getDocument();
    Message message = new Gson().fromJson(doc, Message.class);

    System.out.println(String.format("%s: %s [%tc]", message.userId,
        message.message, new Date(message.date)));
    // Remember the last message seen date.
    lastMessageSeen = message.date + 1;
}
```

SCHEDULE

(DAY 2 AFTERNOON)

Views Exercises

Tutorial for Views

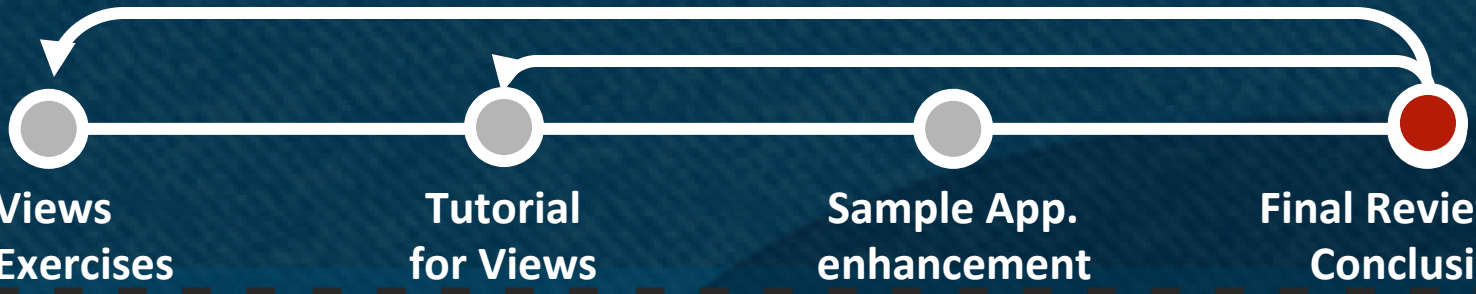
Sample App. enhancement

Final Review and Conclusions

Exercise 16: Enhance the sample Beer web-based App.

- Objective
  - to learn about using Views
- Prerequisite
  - The sample web-based beer app.
- Steps
  - Enhance the app. to answer queries such as
    - Breweries in a particular state
    - States that have breweries that have a Beer with ABV > 9.0 (for example)
    - Other queries

RESOURCES AND SUMMARY



Java Client Library Roadmap

- Server 1.8 Compatible
  - 1.0.1/2.8.0
  - 1.0.2/2.8.0 (Memcache Node imbalance)
  - 1.0.x/2.8.0 (Replica Read)
- Server 2.0 Compatible
  - 1.1-dp/2.8.1 (Views)
  - 1.1-dp2/2.8.x (Observe)
  - 1.1/2.8.x (Final)
- Other Features
  - Spring Integration
  - Your feedback?

Resources, Summary and Call For Action

- Couchbase Server Downloads
  - http://www.couchbase.com/downloads-all
  - http://www.couchbase.com/couchbase-server/overview
- Views
  - http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-views.html
- Developing with Client libraries
  - http://www.couchbase.com/develop/java/current
- Couchbase Java Client Library wiki – tips and tricks
  - http://www.couchbase.com/wiki/display/couchbase/Couchbase+Java+Client+Library

THANKS - Q&A