Couchbase

Developing with the Couchbase 2.x Java SDK

Workshop Day 2

Data Modeling Basics

JSON

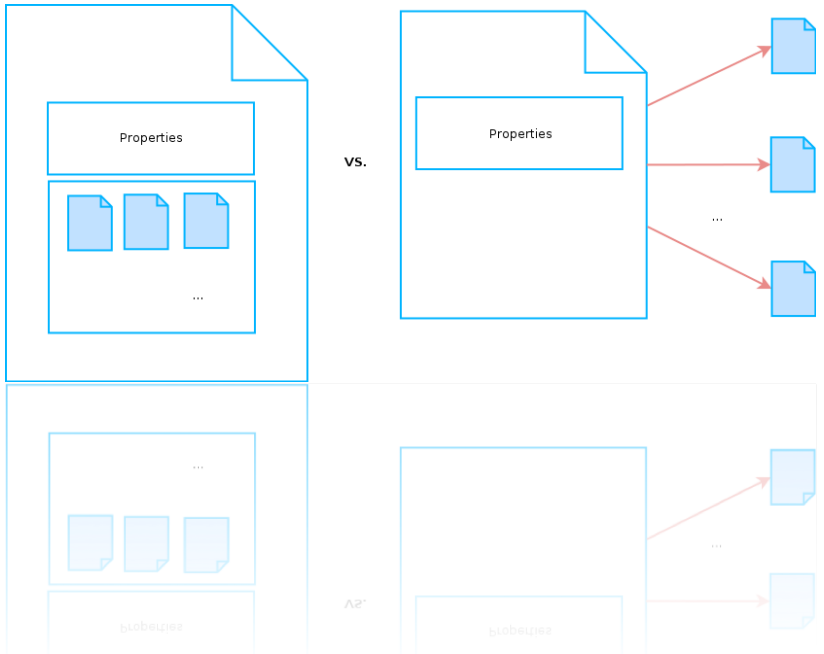
- Java Script Object Notation
  - Meta data
  - Document Value

```
"meta" :
{
  "id" : "person::david",
  "rev" : "1-0002bce0000000000",
  "flags" : 0,
  "expiration":0,
  "type":"json"
}

"doc" :
{
  "type" : "person",
  "uid":"david",
  "firstname":"David",
  "lastname":"Maier",
  "birthday": 330004800000,
  "email":"david.maier@couchbase.com"
}
```

Normalization vs. De-Normalization

- Normalized
  - Uses key references for 1-many relationships
  - Reduces data duplicates
  - Smaller document size
- De-Normalized
  - Uses nested documents
  - Aggregated view of data
  - Allows atomic access
  - No client side joins



Normalization vs. De-Normalization

DE-NORMALIZED

```
{
  "type" : "organization",
  "oid" : "CB",
  "name" : "Couchbase",
  "street" : "2440 West El Camino Real Suite 101",
  "city" : "Mountain View",
  "state" : "California"
  "employees" :
    [
      {
        "uid":"david",
        "firstname":"David",
        "lastname":"Maier",
        "birthday": 1402920000000,
        "email":"david.maier@couchbase.com"
      },
      ...
    ]
}
```

NORMALIZED

```
{
  "type" : "organization",
  "name" : "Couchbase",
  "street" : "2440 West El Camino Real Suite 101",
  "city" : "Mountain View",
  "state" : "California"
  "employees" : ["person::david", "person::perry", "person::dipti", ... ]
}
```

Atomic Counters

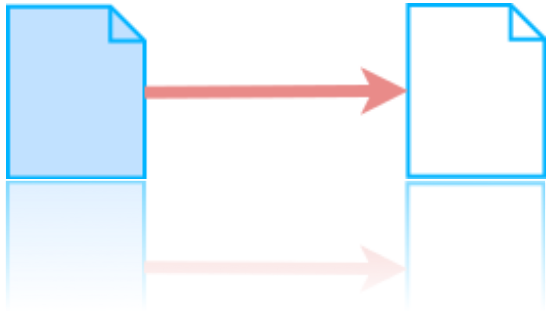
- Similar to sequences / auto-incrementing columns from the relational world
- Initialize and increment a counter value
- Use the counter as part of the key


```
id = client.incr("count::person");
```

```
client.add("person::" + id, doc);
```

Reference Documents for Lookups

- Second document which references the primary one
- Needs to be maintained by the application



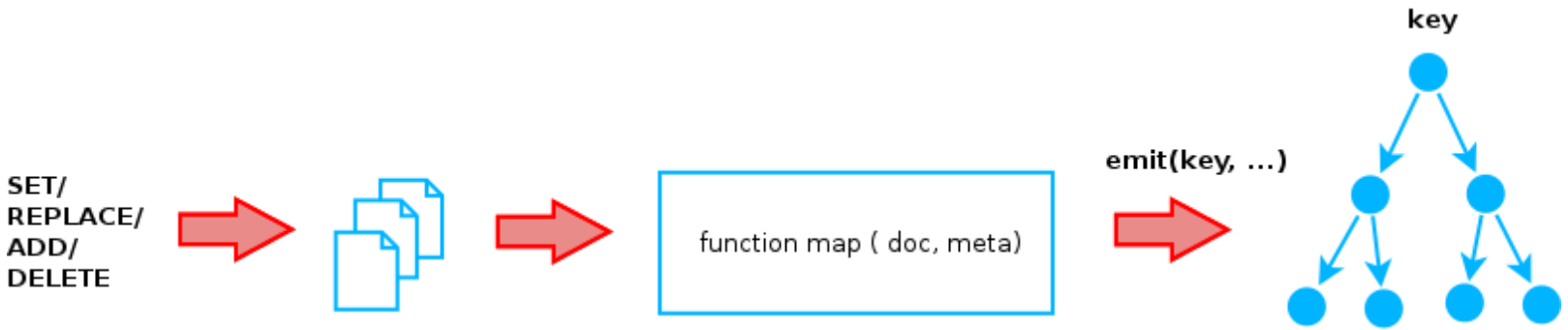
```
"email::david.maier@couchbase.com" : { "ref" : "person::david" }
```

Querying via Views

Views

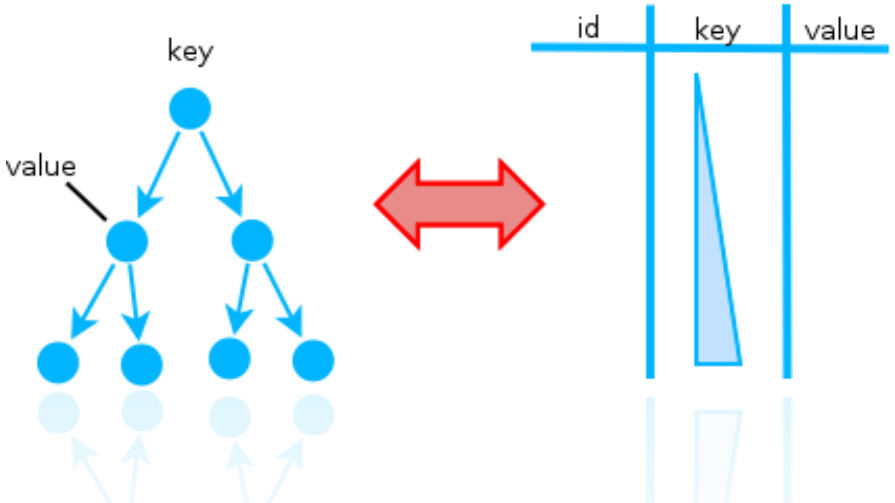
- Organized in Design Documents
- Incremental Map-Reduce
- Spread indexing load across nodes

Map	Reduce
Process, filter, map and emit a row	Aggregate mapped data Built in: _count, _sum, _stats



Views

- Multiple roles
  - A Primary Index to access all document id-s
  - A Secondary Index as an alternative access path
  - A View provides you an alternative view on your data



The Observer Pattern and RxJava

The Patterns

- In the Observer pattern, Observers are used to observe a subject (Observable). The Observers are realizing (signaled by the Observable) if the state of the Observable is changing by being able to react on such an event. It's easy to see that you can use it also for asynchronous data access.
- The Iterator pattern allows to iterate through a collection of objects without the need to know the internal structure of the collection of objects. To combine the Observer pattern with the Iterator pattern means that you can receive a data stream (instead the whole collection of data) by then reacting on the arrival of new data items.
- The functional aspect is that you may pass functions (E.G. as anonymous classes or lambda expressions) as the arguments to an Observable.

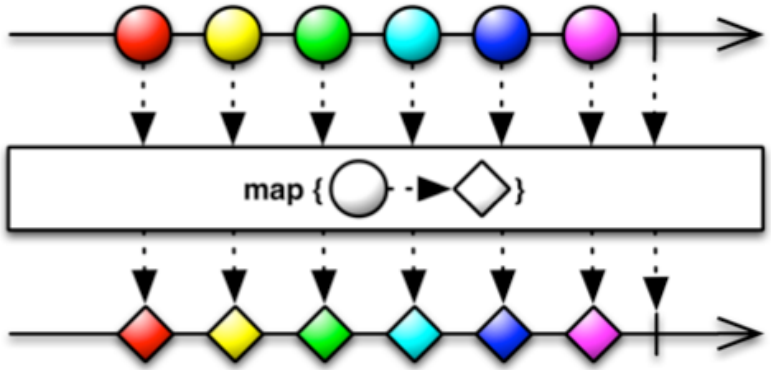
Observable

- Observable<T> and Observer<T>
- Observable = a Stream of Data
- Rx operators to manipulate the stream

	Single	Multiple
Sync ("pull")	T	Iterable<T>
Async ("push)	Future<T>	Observable<T>

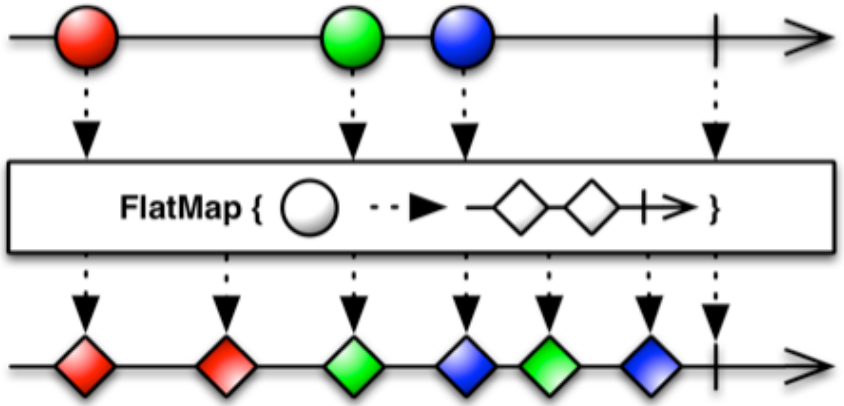
Transformation

- Observable<T> source
- Observable<R> transformed = source,map(function)
- map(function) transforms each T into R



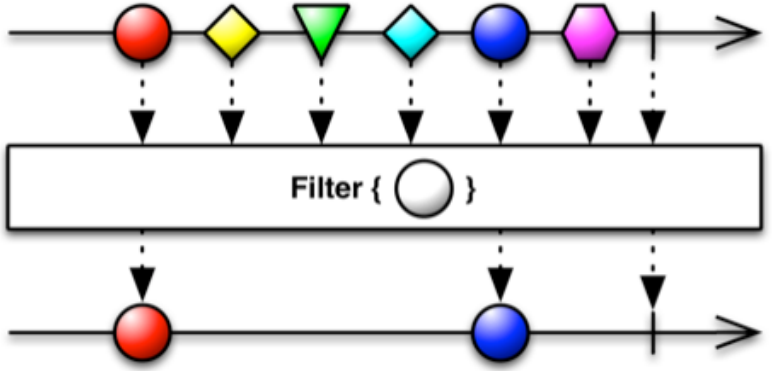
Transformation

- flatMap is like map but emits again an Observable which is then flattened into one Observable (output stream)
- When the transformation must be also async.



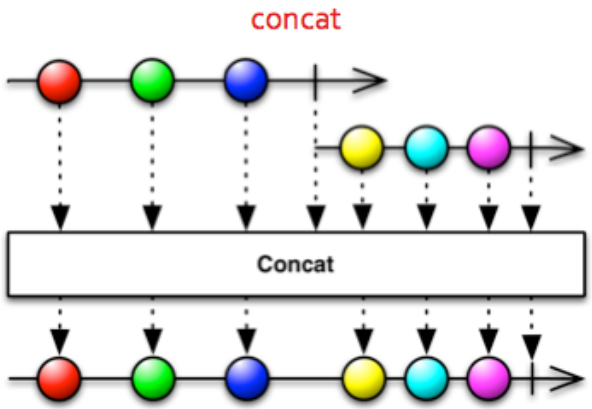
Filter

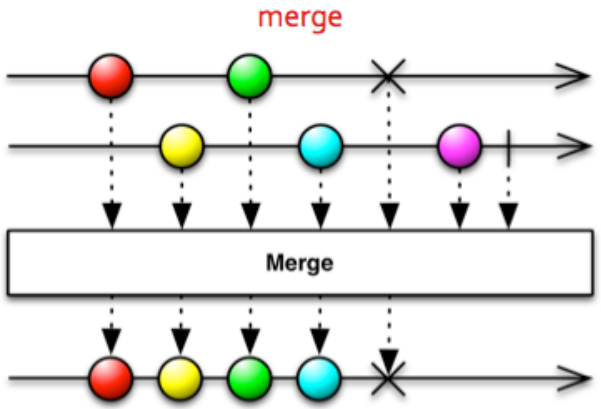
- Observable<T> filter(predicate)
- E.G.
  - **first()**
  - **take(n)**
  - **skip(n)**
  - **…**



Combine

- E.G.
  - **merge(observableOfT)**
  - **concat(observableOfT)**
  - **…**





Subscribe

- By passing an Observer or the following Actions
  - **onError**
  - **onNext**
  - **onComplete**

Error Handling

- In the subscribe
  - **onError(function)**
- Return default value
  - **onErrorReturn(T defaultValue)**
- Defer to a backup stream on error
  - **onErrorResumeNext(Observable<T> backup)**
- Retry on error
  - **retry(predicate)**

The 2.x Java API

Cluster and Bucket

- Cluster
  - **Bootstrap the Connection**
  - **Just needs one or multiple IP-s / host names**
- Bucket
  - **Perform data operations**
  - **Thread-Safe**
  - **Reuse it (Singleton)**

Document

- Properties
  - **Content**
  - **Metadata (id, CAS value, expiry)**
- Types
  - **JsonDocument**
  - **BinaryDocument**
  - **SerializableDocument**
  - **RawJsonDocument (to use own JSON Serializer)**

Operations

- Get
- GetFromReplica
- GetAndLock
- Touch
- Upsert
- Insert
- Replace
- Append
- Prepend
- Unlock
- …

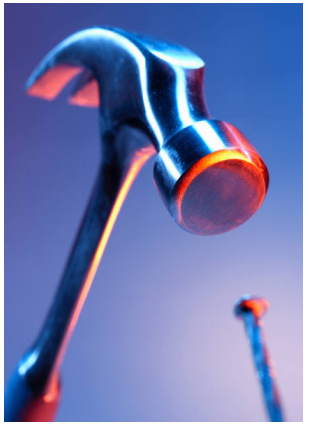
Management

- ClusterManager
  - **info**
  - **hasBucket**
  - **getBucket**
  - **insert/remove/updateBucket**
- BucketManager
  - **info**
  - **flush**
  - **get/insert/upsert/remove/publishDesignDocument**

Hands-on

https://github.com/dmaier-couchbase/cb-workshop

Project Setup



Prepare the project by performing the following steps:

- Check out the example project via Git
- Change to directory '1' and open the pom.xml file
- Add a dependency to the latest Couchbase library

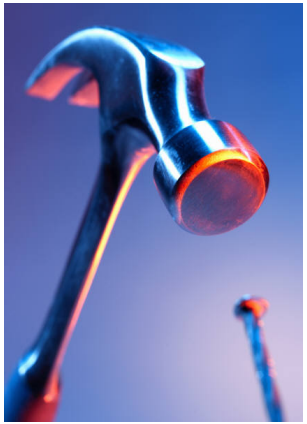
```
<dependencies>
    <dependency>
      <groupId>com.couchbase.client</groupId>
      <artifactId>java-client</artifactId>
      <version>$v</version>
    </dependency>
</dependencies>
```

Connecting to Couchbase

Implement a Factory which returns a single CouchbaseCluster instance based on the configuration in 'cb.properties'

Implement a Factory which returns a single Bucket instance based on the configuration in 'cb.properties'

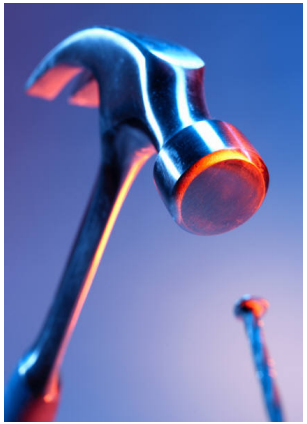
Test the Connection to Couchbase!



CRUD Operations

Create a company document which has the following properties:

- Id
- Type (“company”)
- Name
- Address



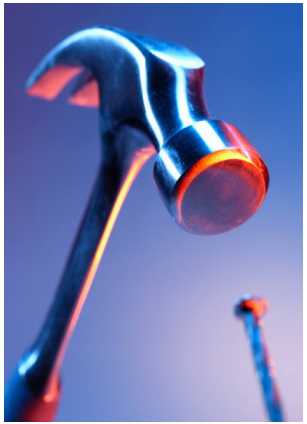
Create 7 user documents those have the following properties:

- Uid
- Type (“user”)
- First name
- Last name
- Email address
- Birthday

CRUD Operations

Assign every user to a company!

For every user create an email lookup document! (Optional)



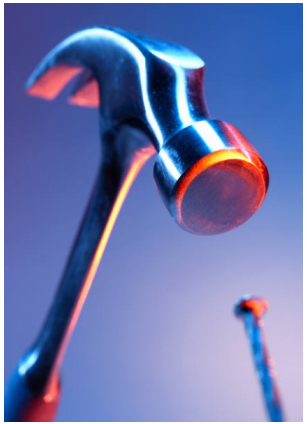
Querying via Views

Create a new Design Document with the name 'persons'!

Create a View with the name 'by_birthday'

Implement a View which:

- Filters users
- Emits the birthday as a date array and so as a compound key
- Emits the first name and the last name as the value



Query the View by using the web browser:

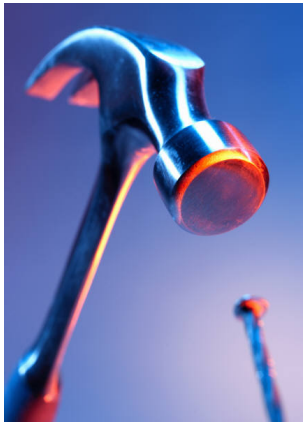
- Use the built-in reduce function _count and group the result by the year of birth

Querying via Views

Query the View by using the web browser:

- Use the built-in reduce function _count and group the result by the year of birth

Query the View by using the Java client!



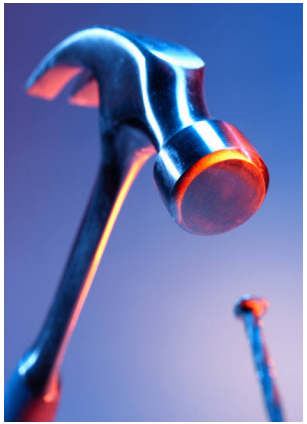
Querying via N1QL

Make sure that Primary Index is created!

Also create a Secondary Index on the type attribute!

Create a Secondary Index on the last name!

- Create the indexes programmatically by using the Index API



Querying via N1QL

Query for users by their last name!

- Use the builder API for building the N1QL statement

Query users of a specific company and with a specific last name by using a JOIN!

- Use a query string for this purpose

Thank you

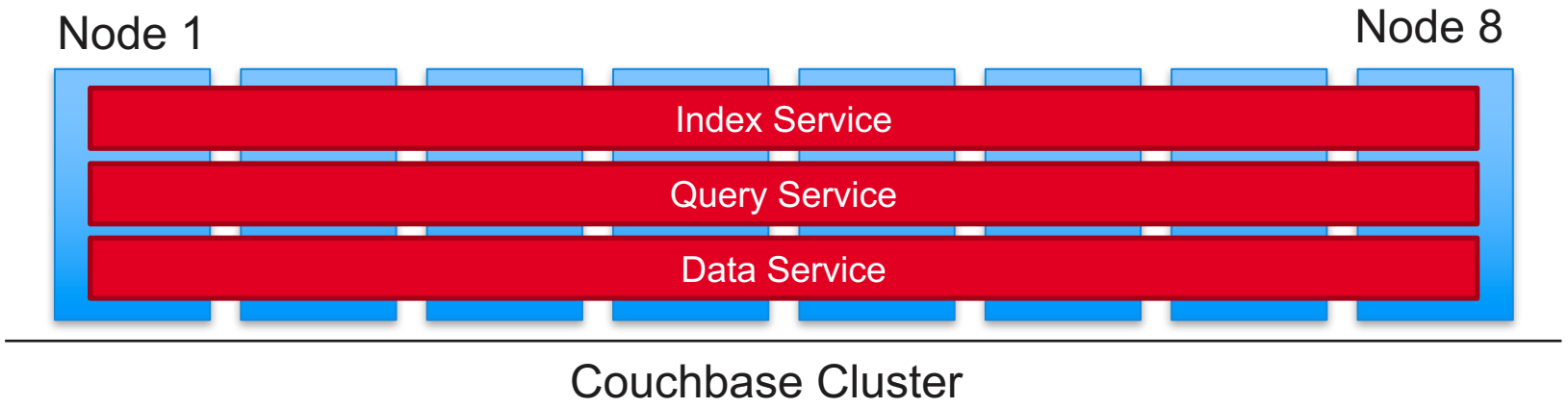
Q&A

What's new in 4.0?

Workload and Scaling

Horizontal scaling

- Partitions a dataset onto one or more homogenous nodes
- Each node runs the same mixed workloads
- Re-partition dataset with additional hardware capacity

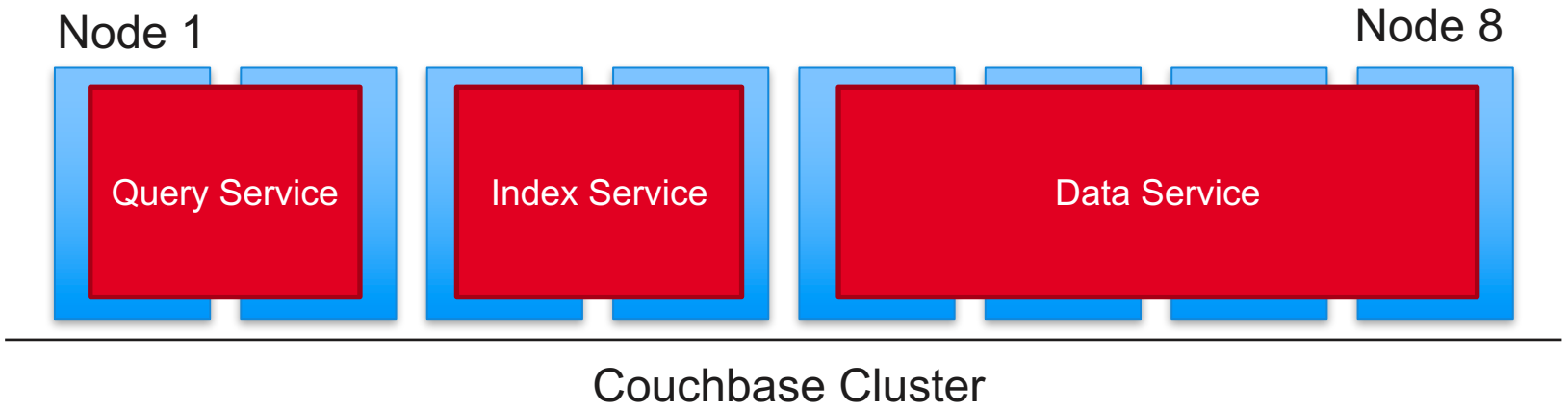


Multi-Dimensional Scaling

Isolated Service for minimized interference

- Independent "zones" for query, index, and data services

Minimize indexing and query overhead on core key-value operations.

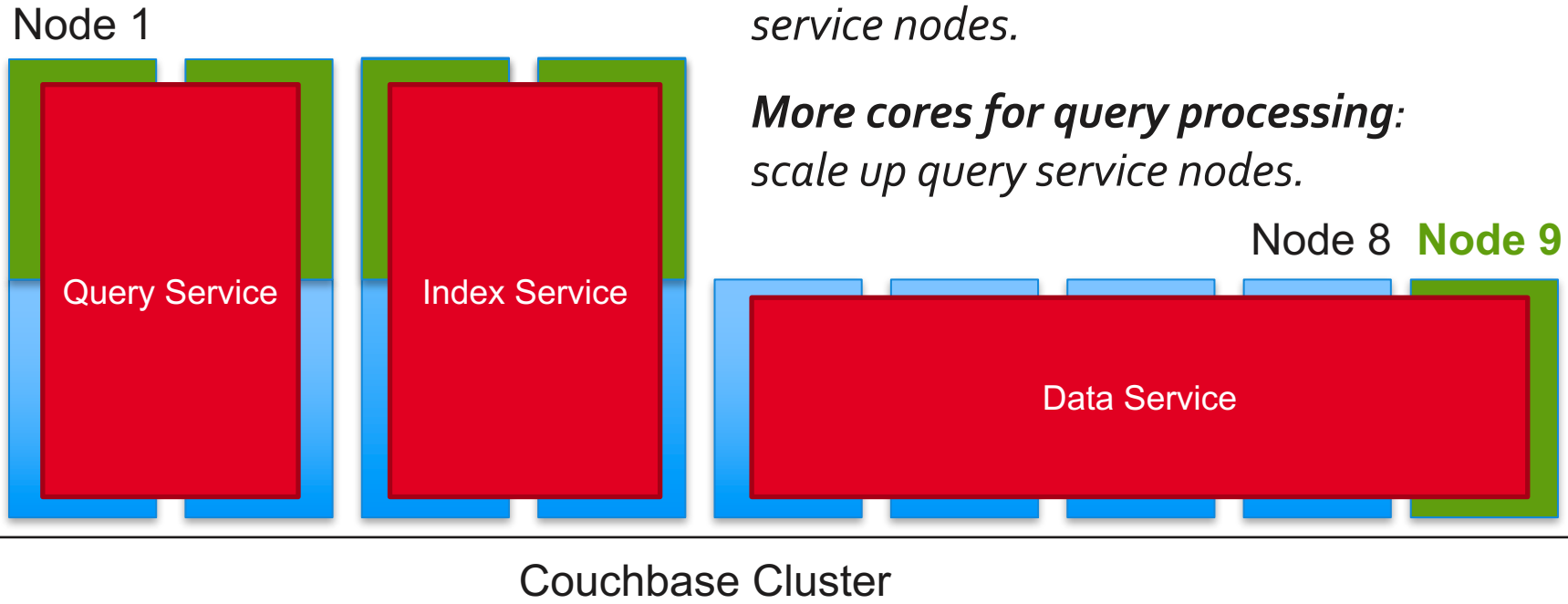


Multi-Dimensional Scaling

Independent scalability for the best computational capacity per service

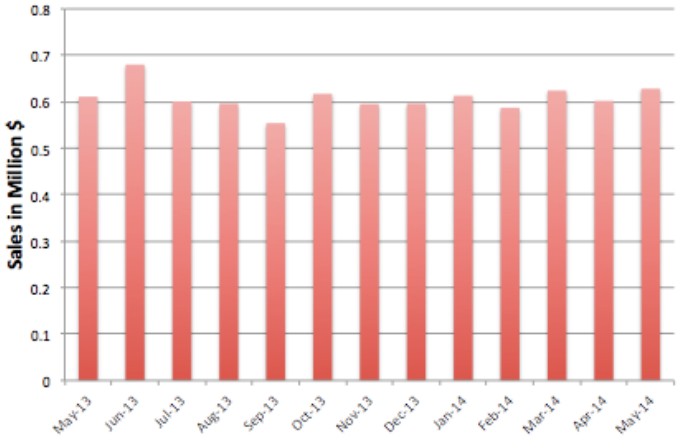
***More storage for indexing**: scale up index service nodes.*

***More cores for query processing**: scale up query service nodes.*



N1QL is Expressive: E-Commerce Examples


```
SELECT product.name, SUM(items.count) AS unitsSold
FROM purchases UNNEST purchases.lineItems AS items
JOIN product ON KEYS items.product
GROUP BY product
ORDER BY unitsSold DESC LIMIT 10
```




```
SELECT SUBSTR(purchases.purchasedAt, 0, 7) AS month,
    ROUND(SUM(product.unitPrice * items.count)/1000000, 3)
revenueMillion
FROM purchases UNNEST purchases.lineItems AS items
JOIN product ON KEYS items.product
GROUP BY SUBSTR(purchases.purchasedAt, 0, 7)
ORDER BY month
```

N1QL is Expressive: Image Processing Example

```
SELECT
    m.image AS gallery_image,
    s.image AS source_image, s.x, s.y,
    m.r AS gallery_r, m.g AS gallery_g, m.b AS gallery_b,
    s.r AS source_r, s.g AS source_g, s.b AS source_b,
FROM SourcePixels s
JOIN GalleryMeans m ON KEY s.rgb
```

Learn more about N1QL @ query.couchbase.com/tutorial

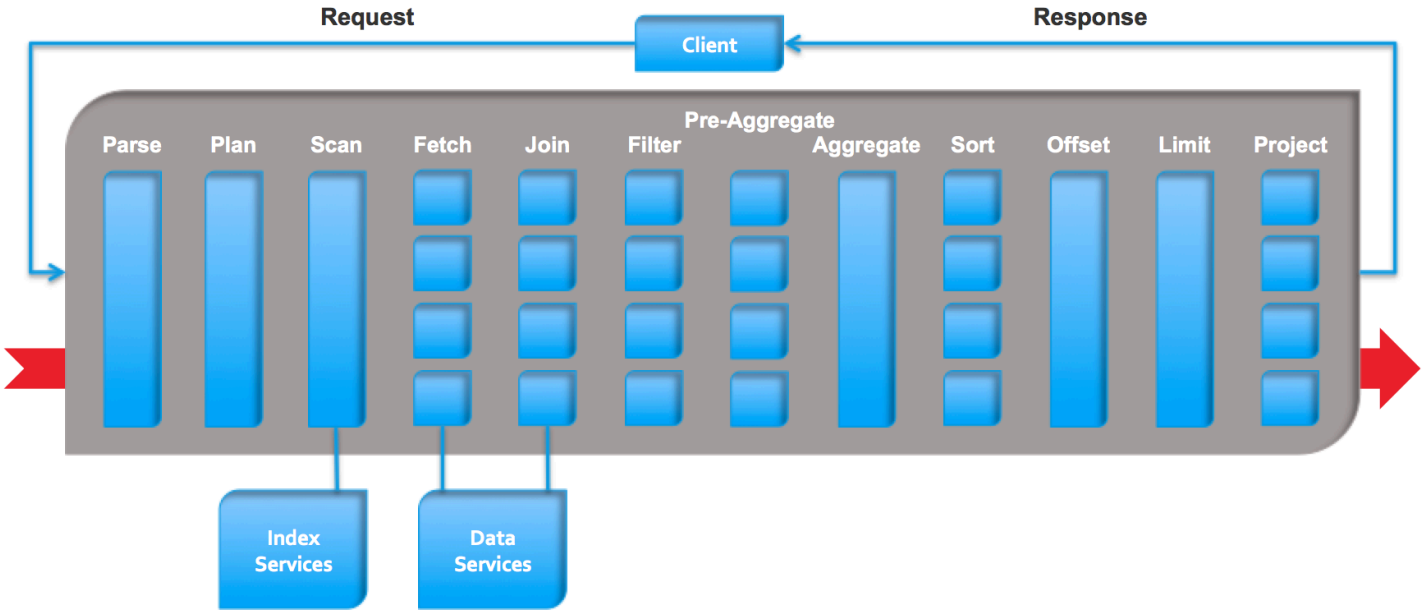
Source: http://www.metanautix.com/tr01

Metanautix

N1QL is Declarative: What vs. How

You specify **WHAT**

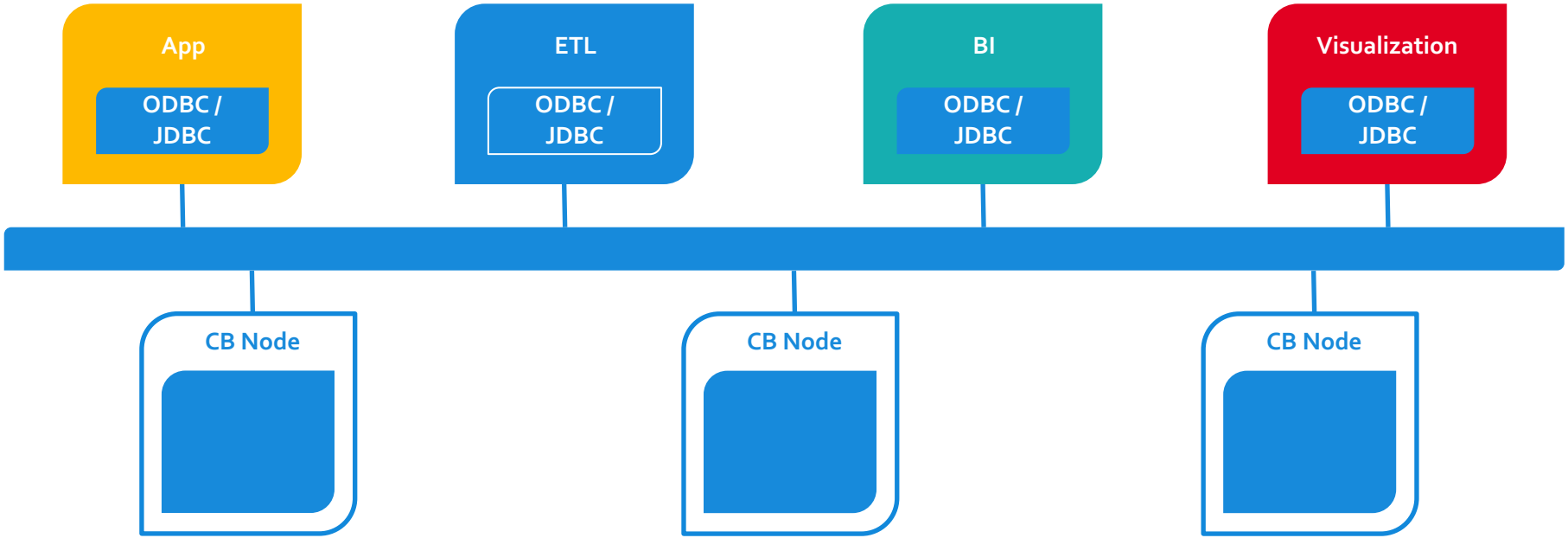
Couchbase Server figures out **HOW**



N1QL Connects To Your Enterprise Ecosystem

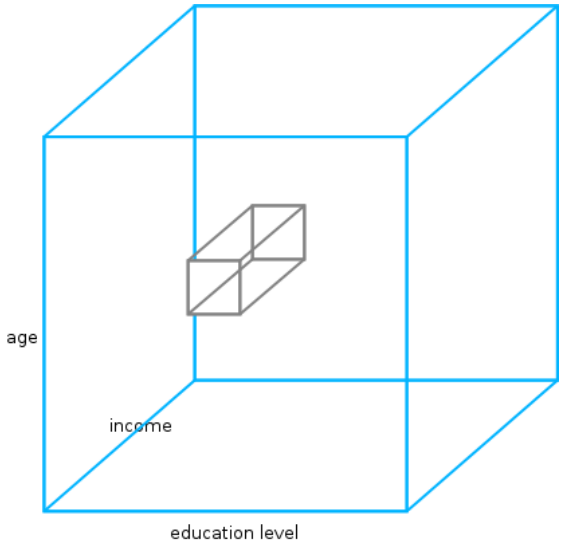
Standards-based drivers

Integrations, partnerships



Spatial Indexes

- Multi Dimensional Analysis
  - Not necessarily Geo-Data but any numeric data
  - Query within a Hyper-Cube
  - Map categories to numbers
- e.g.
  - Income, Age, Education level (Bsc = 4, Msc = 5)
  - Timestamp, Log-Level



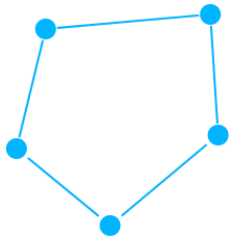
Spatial Indexes

- Geo-Data
  - GeoJSON: the "Open Standard"
  - More complex geometries stored as regions
  - Bounding-Box Queries
  - e.g. all buildings in San Francisco

```
{ "type": "Point", "coordinates": [100.0, 0.0] }
```

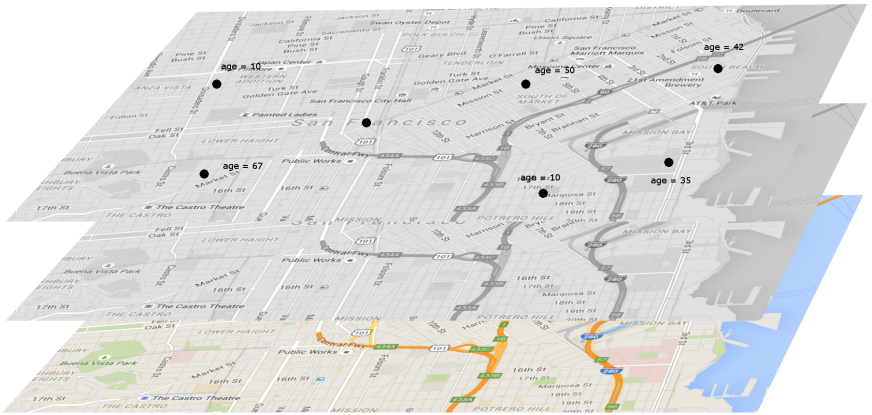
```
{
  "type": "LineString",
  "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
}
```

```
{
  "type": "Polygon",
  "coordinates": [
   [ [100.0, 0.0], [101.0, 0.0],
   [101.0, 1.0], [100.0, 1.0], [100.0, 0.0] ]
   ]
}
```

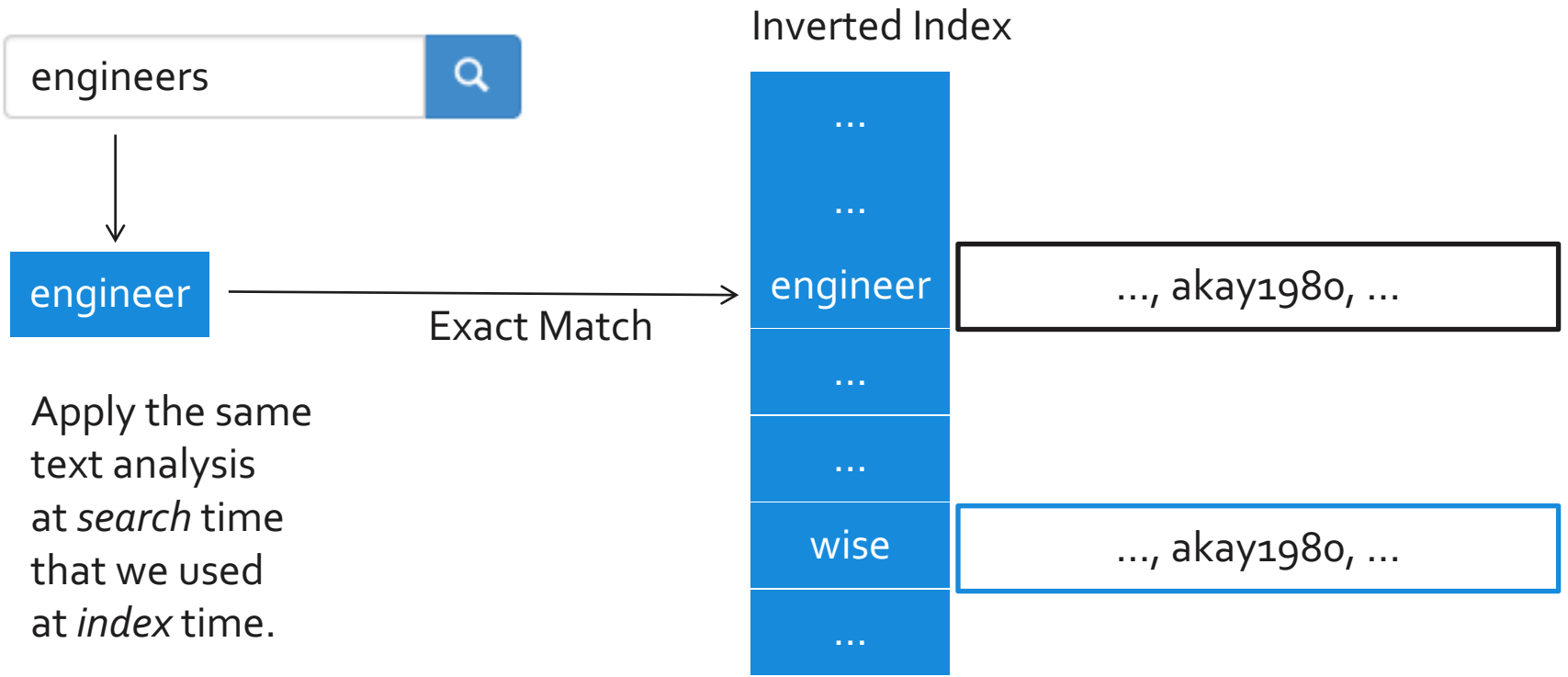


Spatial Indexes

- Combined
  - 2 dimensions for Geo-Data
  - Additional dimensions
  - e.g. all persons with an age greater than 10 in San Francisco



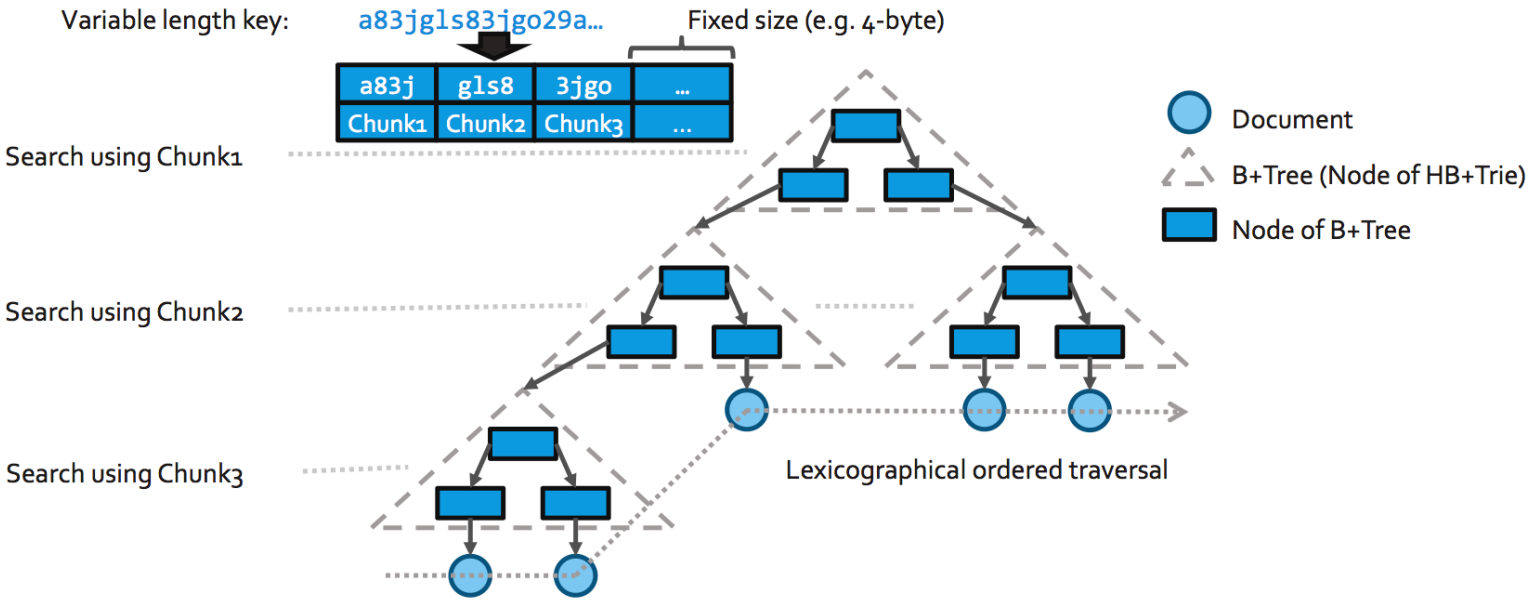
Text Indexes (Developer Preview)



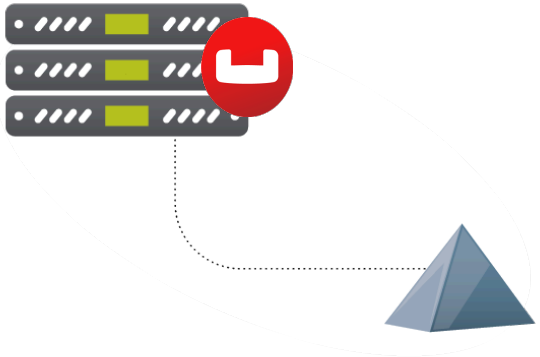
HB+-Trie for GSI

Trie (prefix tree) for which the nodes are B+-Trees

- *HB+-Trie* was originally presented at ACM SIGMOD 2011 Programming Contest, by Jung-Sang Ahn who works at Couchbase
  (http://db.csail.mit.edu/sigmod11contest/sigmod_2011_contest_poster_jungsang_ahn.pdf)



External Identity Management Using LDAP



Centralized identity management

- Define multiple read-only admins and full-admins
- Centralized security policy management for admin accounts for stronger passwords, password rotation, and auto lockouts

Individual accountability; simplified compliance

- Define UIDs in LDAP, and map UIDs to the read-only/full admin role in Couchbase
- Comprehensive audit trails with LDAP UIDs in audit records

Admin Auditing in Couchbase

Rich audit events

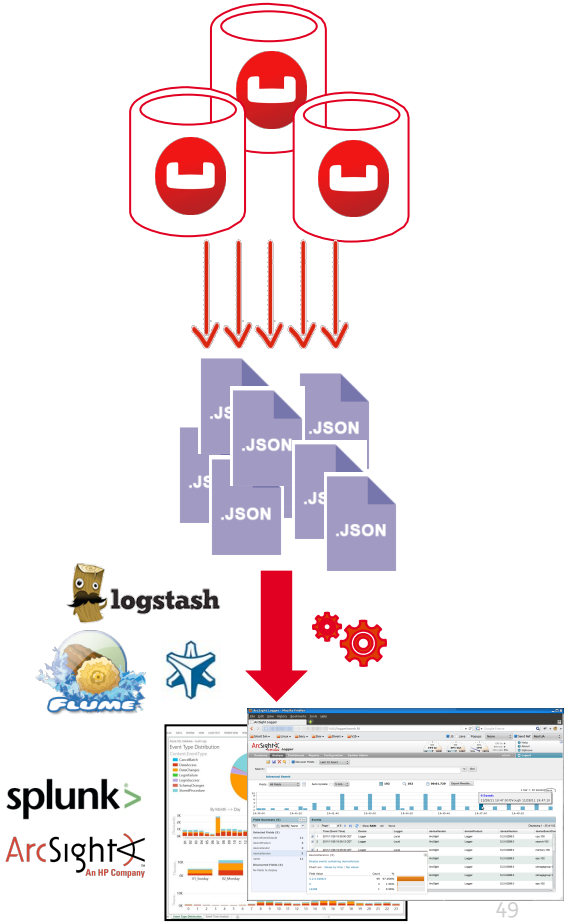
- Over 25+ different, detailed admin audit events
- Auditing for tools including backup

Configurable auditing

- Configurable file target
- Support for time-based log rotation and audit filtering

Easy integration

- JSON format allows for easy integration with downstream systems using flume, logstash, and syslogd



And There's More …

Feature	Description
XDCR Filtering	Key-based filtering on XDCR streams addressing reference data & geo-based replication use cases.
Bloom Filters	Significantly improved latency for low % working set (DGM) situations
Memory Defragmentation	Use memory according to one's needs
Improved Cluster Robustness	Rogue view queries no longer cause cluster manager outage
Breakpad	Compact and readable crash logs for memcached
SDK	First class SDK support across Couchbase Server 4.0 features
SUSE, Oracle Enterprise Linux, CentOS 7, RHEL 7, Ubuntu 14.04	Expand the supported production platforms