

Week 3: NoSQL (Cont.) and Document Databases

DIANE WOODBRIDGE, PH.D



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Objectives

Learn consistency issues on NoSQL.

Learn Map-Reduce concepts.

Overview of document database.

Learn characteristics of MongoDB and do hands-on practice.



NoSQL Interview Questions

What is NoSQL?

Eventual Consistency

Relational Database vs. NoSQL

Map-Reduce

Impedence mismatch

Polygot persistence

Aggregate-oriented database

Key-value database

Document database

Column family database

Graph database

Replication vs sharding

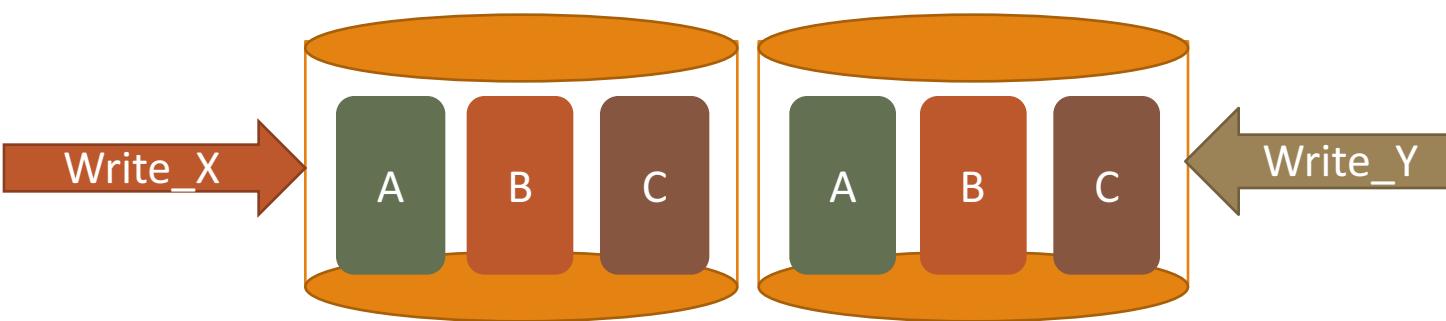
CAP Theorem



Consistency

Update Consistency

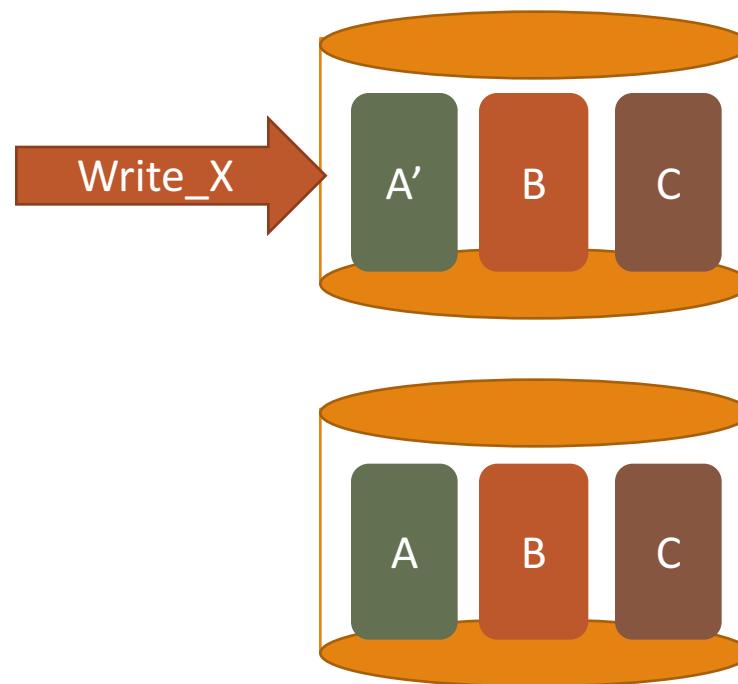
- Write-write conflict
- Consistency control
 - Pessimistic approach – Prevents conflicts (Safety)
 1. Use write locks – One lock at a time.
 2. One will see the updates, before deciding to apply its update.
 - Optimistic approach – Lets conflicts happen, but take care of it. (Responsive)
 1. Conditional updates – Checks whether there is an update right before updating data.
 2. Version control – Save both updates and record that they are in conflict.



Consistency

Update Consistency

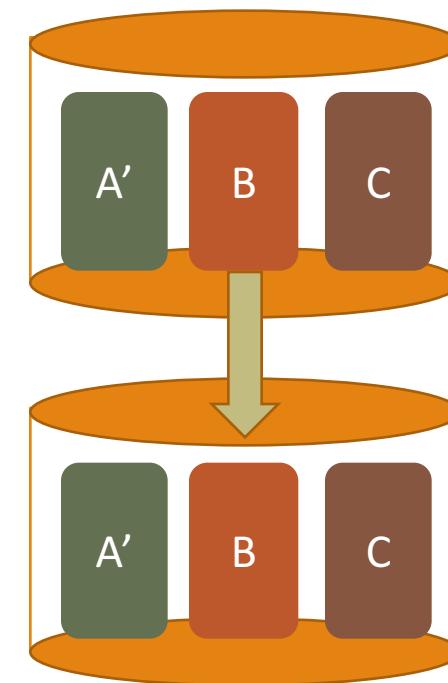
- Write-write conflict
- Consistency control
 - Pessimistic approach – Prevents conflicts (Safety)
 1. Use write locks – One lock at a time.
 2. One will see the updates, before deciding to apply its update.
 - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
 1. Conditional updates – Checks whether there is an update right before updating data.
 2. Version control – Save both updates and record that they are in conflict.



Consistency

Update Consistency

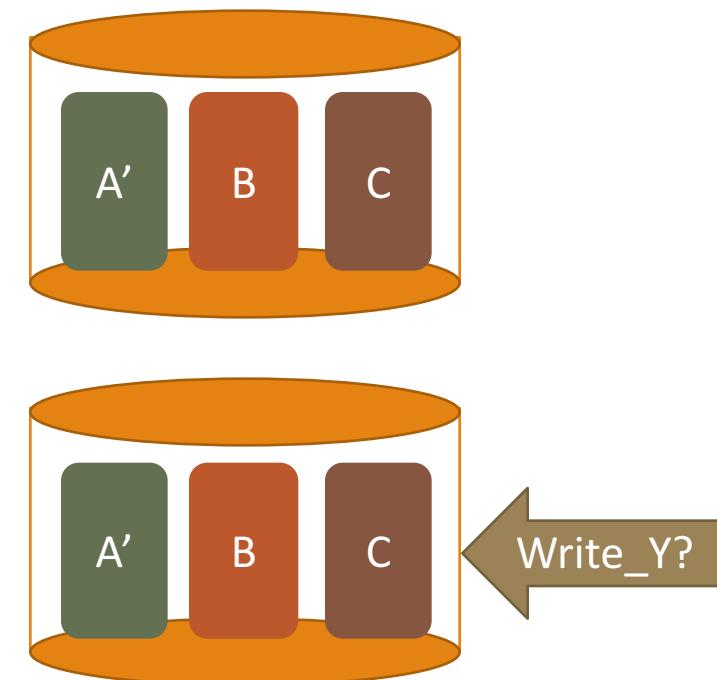
- Write-write conflict
- Consistency control
 - Pessimistic approach – Prevents conflicts (Safety)
 1. Use write locks – One lock at a time.
 2. One will see the updates, before deciding to apply its update.
 - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
 1. Conditional updates – Checks whether there is an update right before updating data.
 2. Version control – Save both updates and record that they are in conflict.



Consistency

Update Consistency

- Write-write conflict
- Consistency control
 - Pessimistic approach – Prevents conflicts (Safety)
 1. Use write locks – One lock at a time.
 2. One will see the updates, before deciding to apply its update.
 - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
 1. Conditional updates – Checks whether there is an update right before updating data.
 2. Version control – Save both updates and record that they are in conflict.

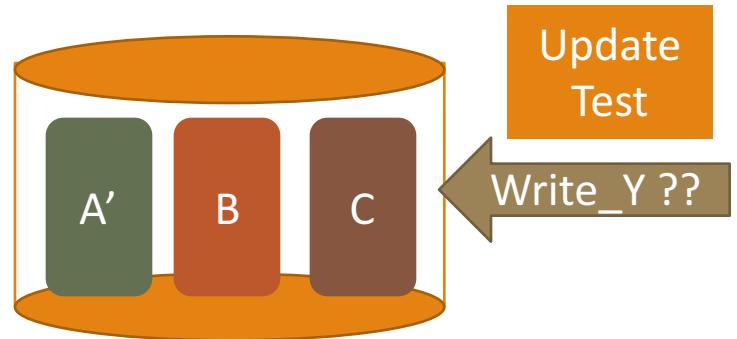


Consistency

Update Consistency

- Write-write conflict
- Consistency control
 - Pessimistic approach – Prevents conflicts (Safety)
 1. Use write locks – One lock at a time.
 2. One will see the updates, before deciding to apply its update.
 - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
 1. Conditional updates – Checks whether there is an update right before updating data.
 2. Version control – Save both updates and record that they are in conflict.

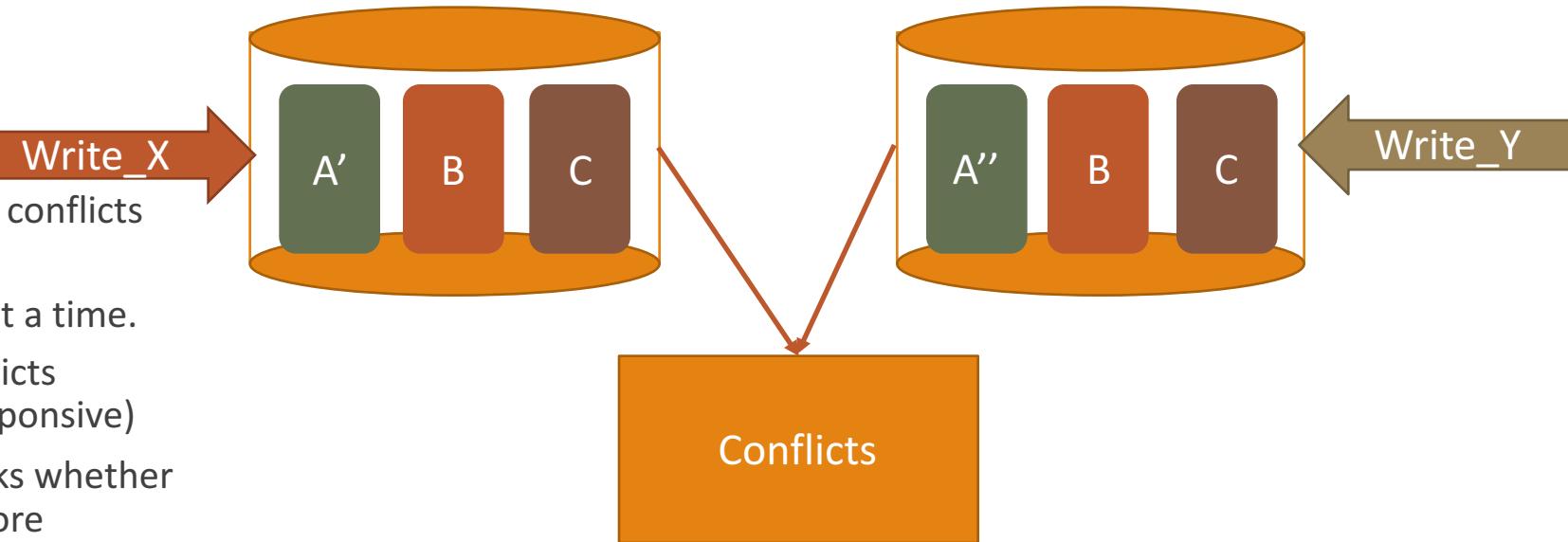
Requires sequential consistency.



Consistency

Update Consistency

- Write-write conflict
- Consistency control
 - Pessimistic approach – Prevents conflicts (Safety)
 1. Use write locks – One lock at a time.
 - Optimistic approach – Lets conflicts happen, but take care of it. (Responsive)
 1. Conditional updates – Checks whether there is an update right before updating data.
 2. Version control – Save both updates and record that they are in conflict.



Consistency

Read Consistency

- Read-write conflict (inconsistent read).
- Replication inconsistency.
- Session inconsistency.

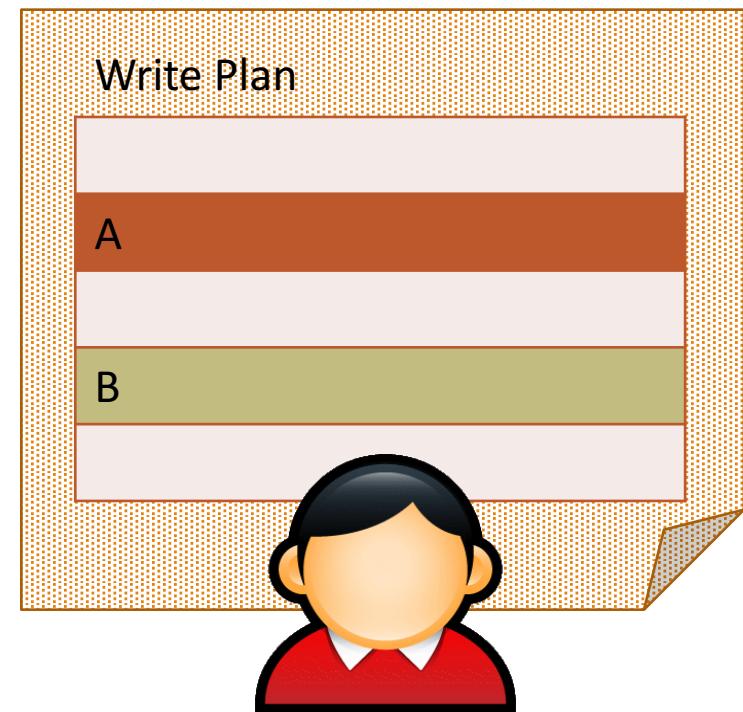


Consistency

Read Consistency

- Read-write conflict (inconsistent read)
 - Logical Consistency
 - Different data items make sense together.
 - RDBMS – Transaction.

Inconsistency window – the length of time that inconsistency exists.

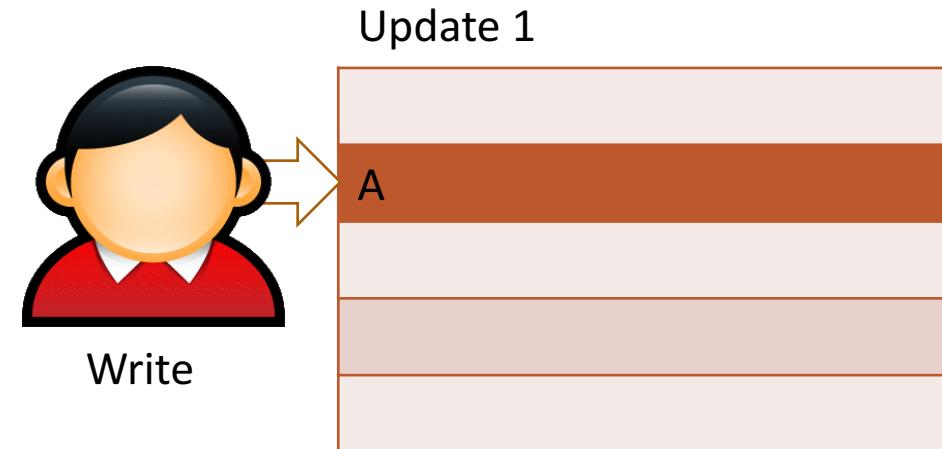


Consistency

Read Consistency

- Read-write conflict (inconsistent read)
 - Logical Consistency
 - Different data items make sense together.
 - RDBMS – Transaction.

Inconsistency window – the length of time that inconsistency exists.



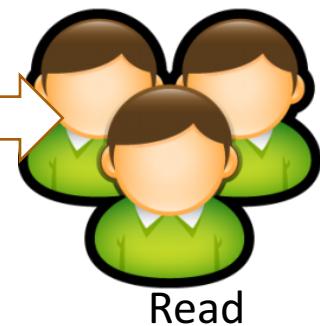
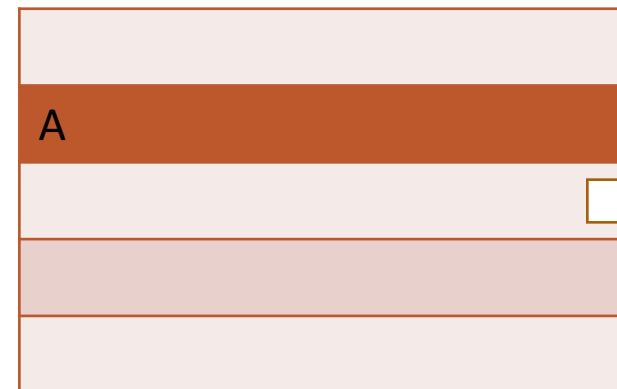
Consistency

Read Consistency

- Read-write conflict (inconsistent read)
 - Logical Consistency
 - Different data items make sense together.
 - RDBMS – Transaction.

Inconsistency window – the length of time that inconsistency exists.

Intermediate

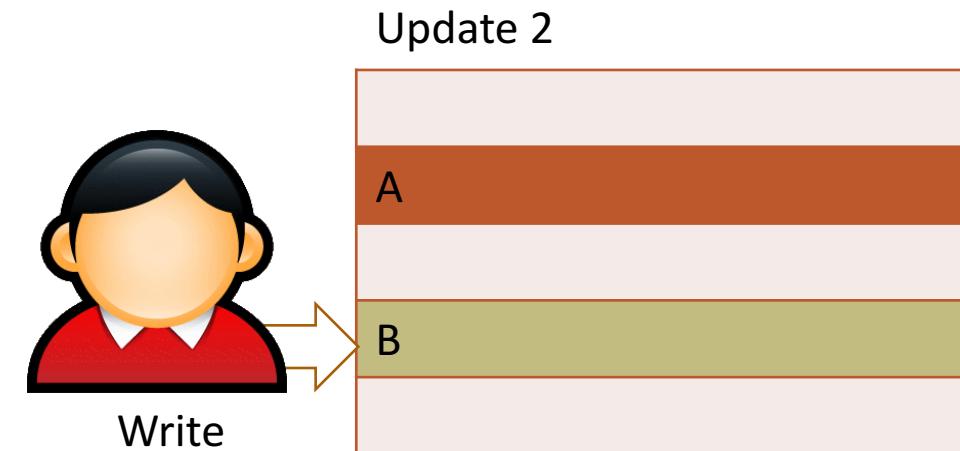


Consistency

Read Consistency

- Read-write conflict (inconsistent read)
 - Logical Consistency
 - Different data items make sense together.
 - RDBMS – Transaction.

Inconsistency window – the length of time that inconsistency exists.

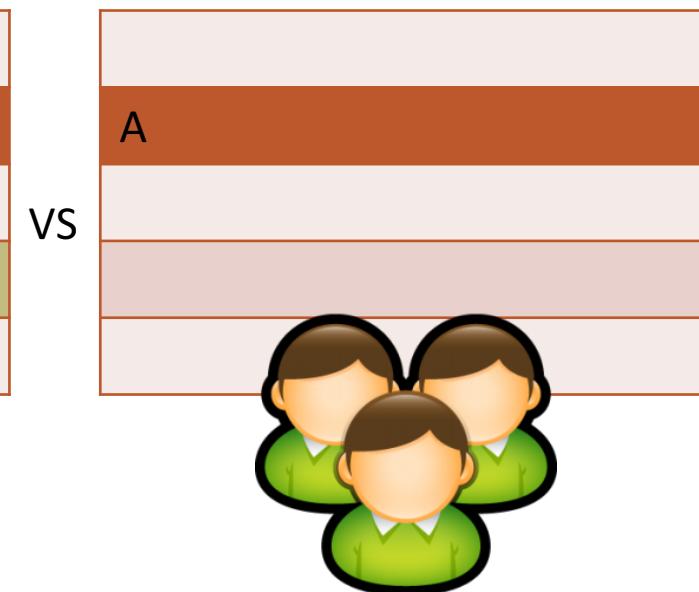
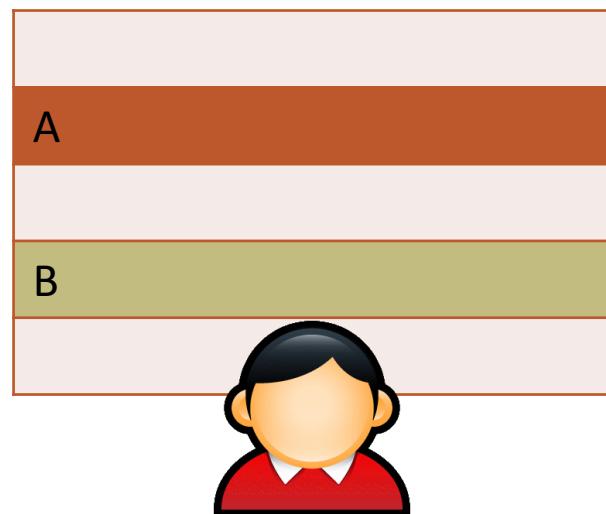


Consistency

Read Consistency

- Read-write conflict (inconsistent read)
 - Logical Consistency
 - Different data items make sense together.
 - RDBMS – Transaction.

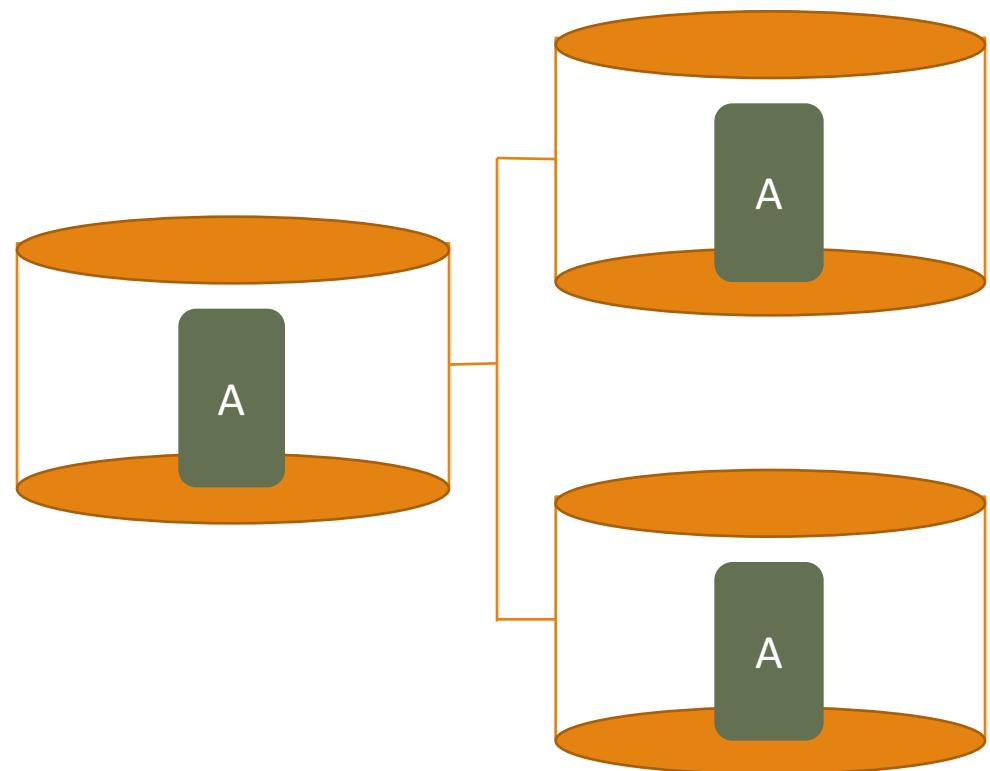
Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

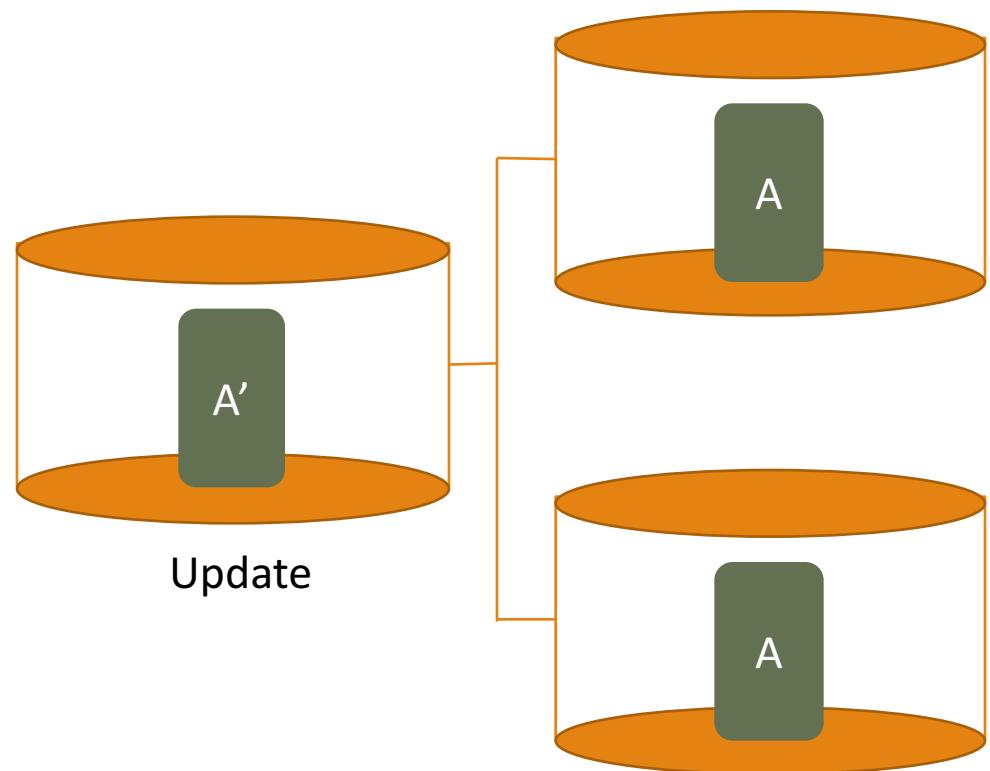
- Replication Consistency
 - The same data has the different value when reading from different replicas.
 - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

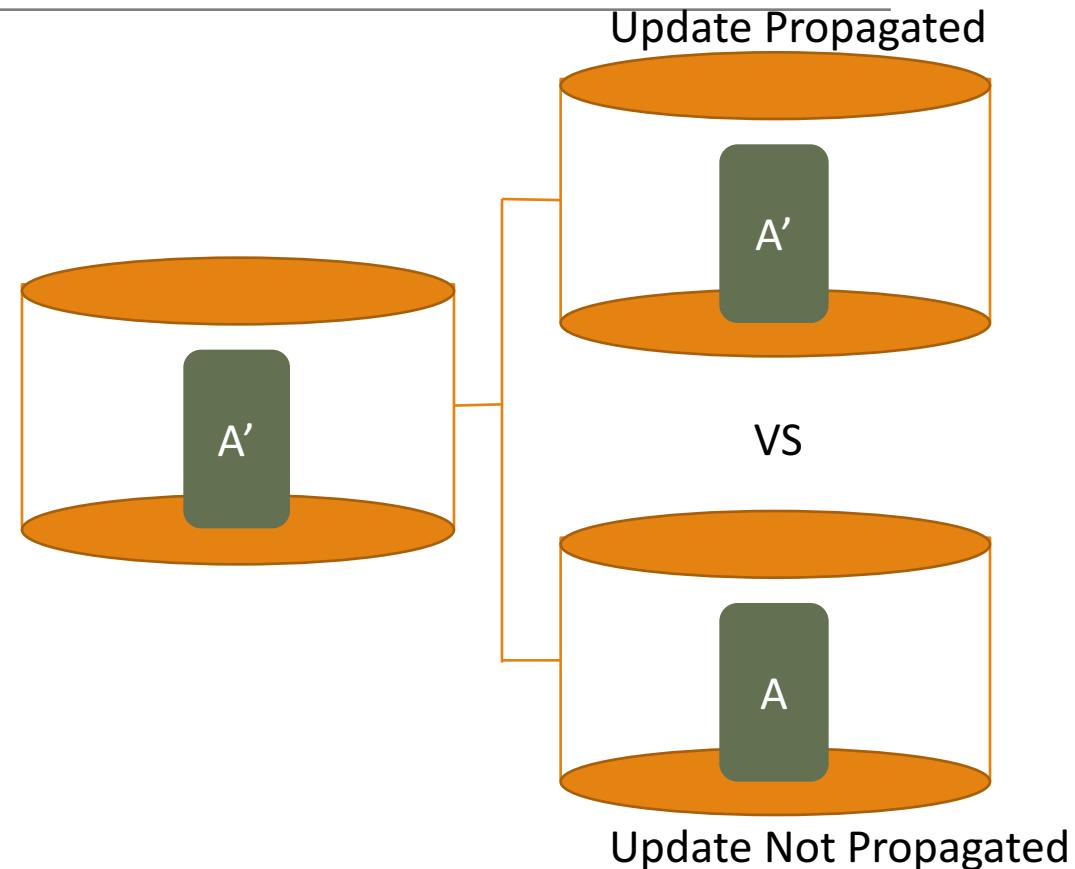
- Replication Consistency
 - The same data has the different value when reading from different replicas.
 - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

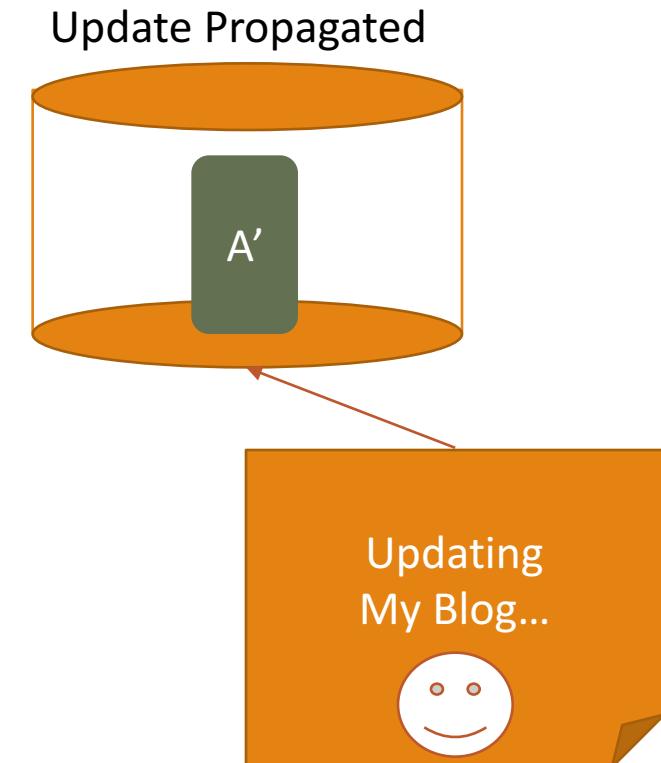
- Replication Consistency
 - The same data has the different value when reading from different replicas.
 - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

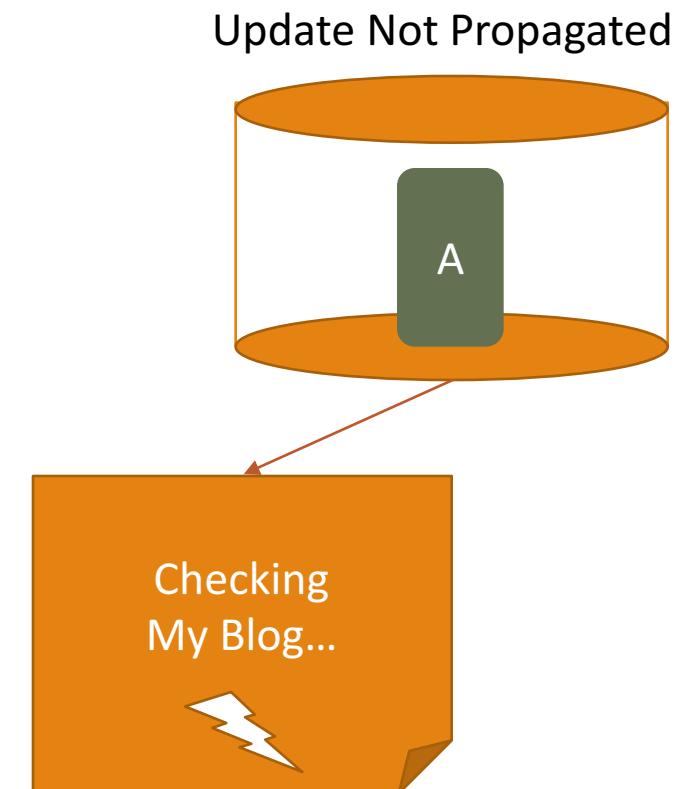
- Session Consistency (Read-your-writes Consistency)
 - Sticky Session (Session affinity): A session is tied to one node and keep session consistency on the node.
 - Version Stamp : Make sure that every interaction with a node returns data with the latest version stamp seen by the session.



Consistency

Read Consistency

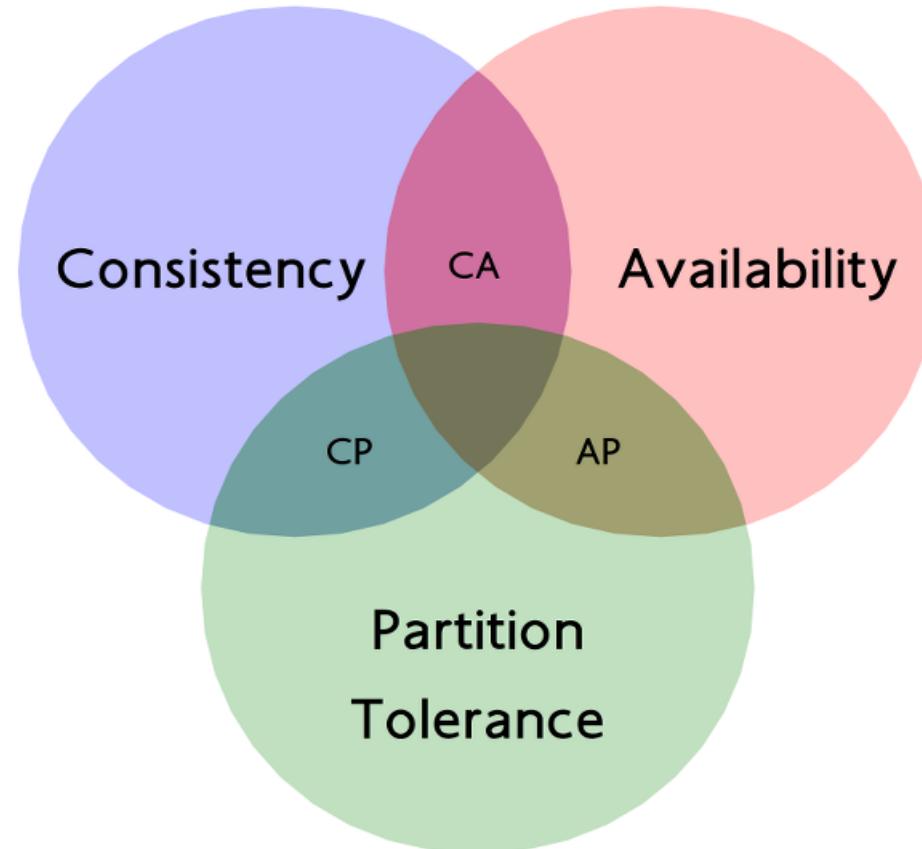
- Session Consistency (Read-your-writes Consistency)
 - Sticky Session (Session affinity) : A session is tied to one node and keep session consistency on the node.
 - Version Stamp : Make sure that every interaction with a node returns data with the latest version stamp seen by the session.



Consistency

Relaxing Consistency/Availability

- CAP Theorem
 - Consistency
 - All nodes have most recent data via eventual consistency.
 - Availability
 - Every request received by a non-failing node must return a response.
 - Partition Tolerance
 - Clusters can survive from communication breakages in the cluster.
- ➔ You can only get two.



<http://blingtechs.blogspot.com/2016/02/cap-theorem.html>

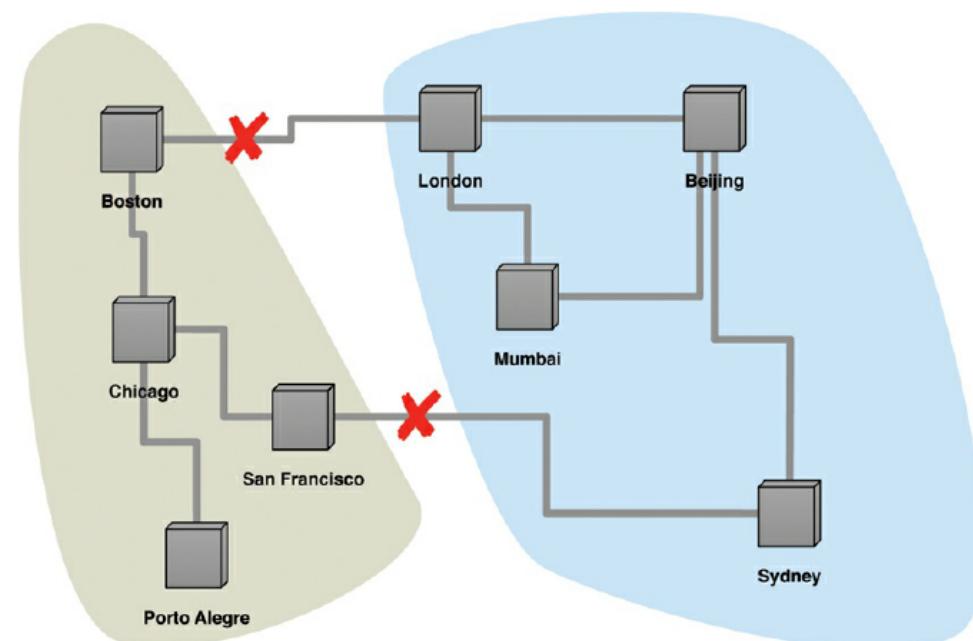


Consistency

Relaxing Consistency/Availability

- CAP Theorem
 - Consistency
 - All nodes have most recent data via eventual consistency.
 - Availability
 - Every request received by a non-failing node must return a response.
 - Partition Tolerance
 - Clusters can survive from communication breakages in the cluster.
- ➔ You can only get two.

ACID addresses an individual node's data consistency.
CAP addresses cluster-wide data consistency .



<http://blingtechs.blogspot.com/2016/02/cap-theorem.html>

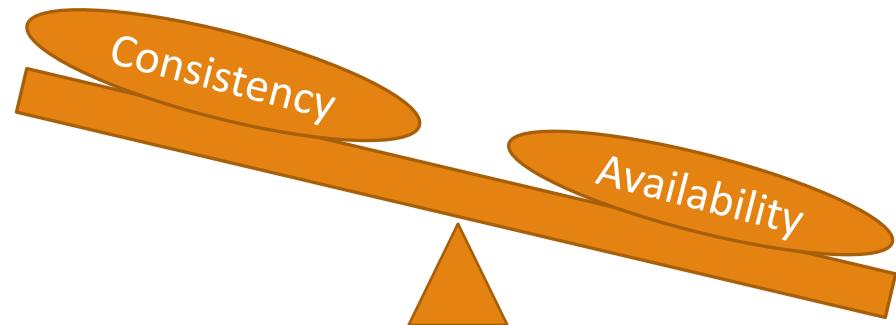


UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Consistency

Relaxing Consistency/Availability

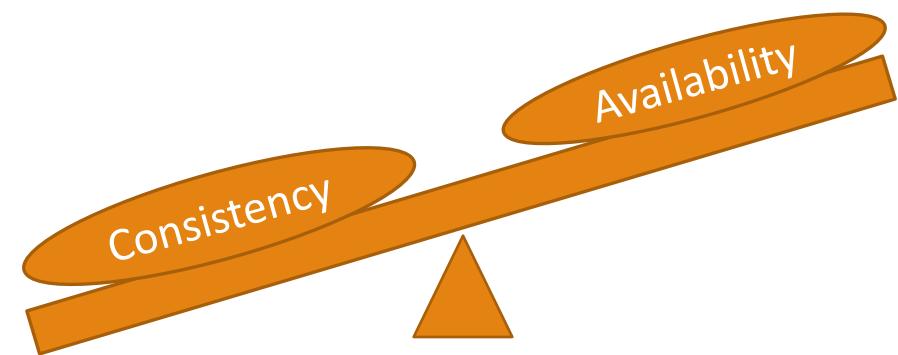
- CAP Theorem and Distributed Database
 - Requirement - Partition-Tolerance *
 - Availability or Consistency??
 - Availability – Shopping
 - Consistency – Stock Market



Consistency

Relaxing Consistency/Availability

- CAP Theorem and Distributed Database
 - Requirement - Partition-Tolerance *
 - Availability or Consistency??
 - Availability – Shopping
 - Consistency – Stock Market



Consistency

Relaxing Consistency/Availability

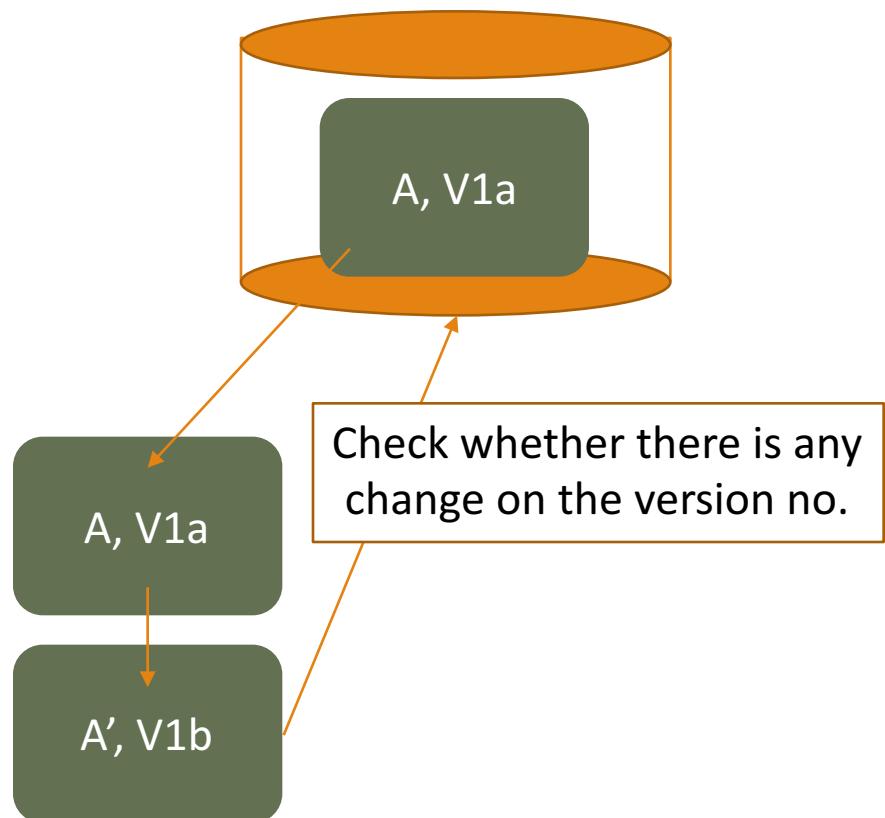
- CAP Theorem and Distributed Database
 - Requirement - Partition-Tolerance *
 - Availability or Consistency??
 - Availability – Shopping
 - Consistency – Stock Market



Consistency

Version Stamps : A field that changes every time when the data value is changed.

- Check updates/reads won't be based on stale data.
 - Example
 - Counter – increment when you update the data. Need a single master.
 - GUID – A large random number (Unique). Can't tell the recentness.
 - Resource content hash - Deterministic. Can't tell the recentness.
 - Timestamp - Clocks have to be kept in sync. Can't take care of too granular updates.
- ➔ Use more than one.



Consistency

Quorum

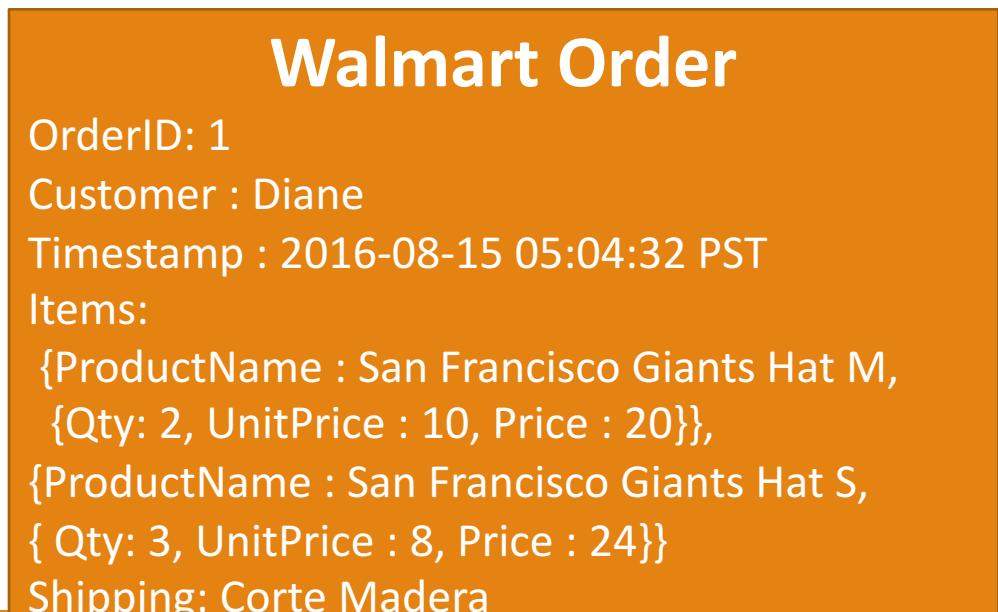
- Write Quorum : How many nodes should acknowledge your writes?
 - $W > N/2$ (W : nodes sending confirmation in write, N – nodes involved in replication (replication factor))
- Read Quorum : How many nodes you need to contact to make sure you have the most up-to-date value?
 - Depending on W .
 - For strong read consistency, $R+W > N$ (R : nodes to be contacted for read consistency)



Map-Reduce Model

Map-Reduce: Allow computations to be parallelized over a cluster.

- Basic Map-Reduce
 - The map-reduce framework plans map tasks to be run on the correct nodes and move data for the reduce function.
 - Map : Return key-value pairs from a single aggregate.
 - Reduce : Return one key-value pair from multiple key-value pairs.



Map-Reduce Model

Map-Reduce: Allow computations to be parallelized over a cluster.

- Basic Map-Reduce
 - The map-reduce framework plans map tasks to be run on the correct nodes and move data for the reduce function.
 - Map : Return key-value pairs from a single aggregate.
 - Reduce : Return one key-value pair from multiple key-value pairs.

{San Francisco Giants Hat M, {Qty:2, Price : 20}}

{San Francisco Giants Hat S, {Qty:3, Price : 24}}

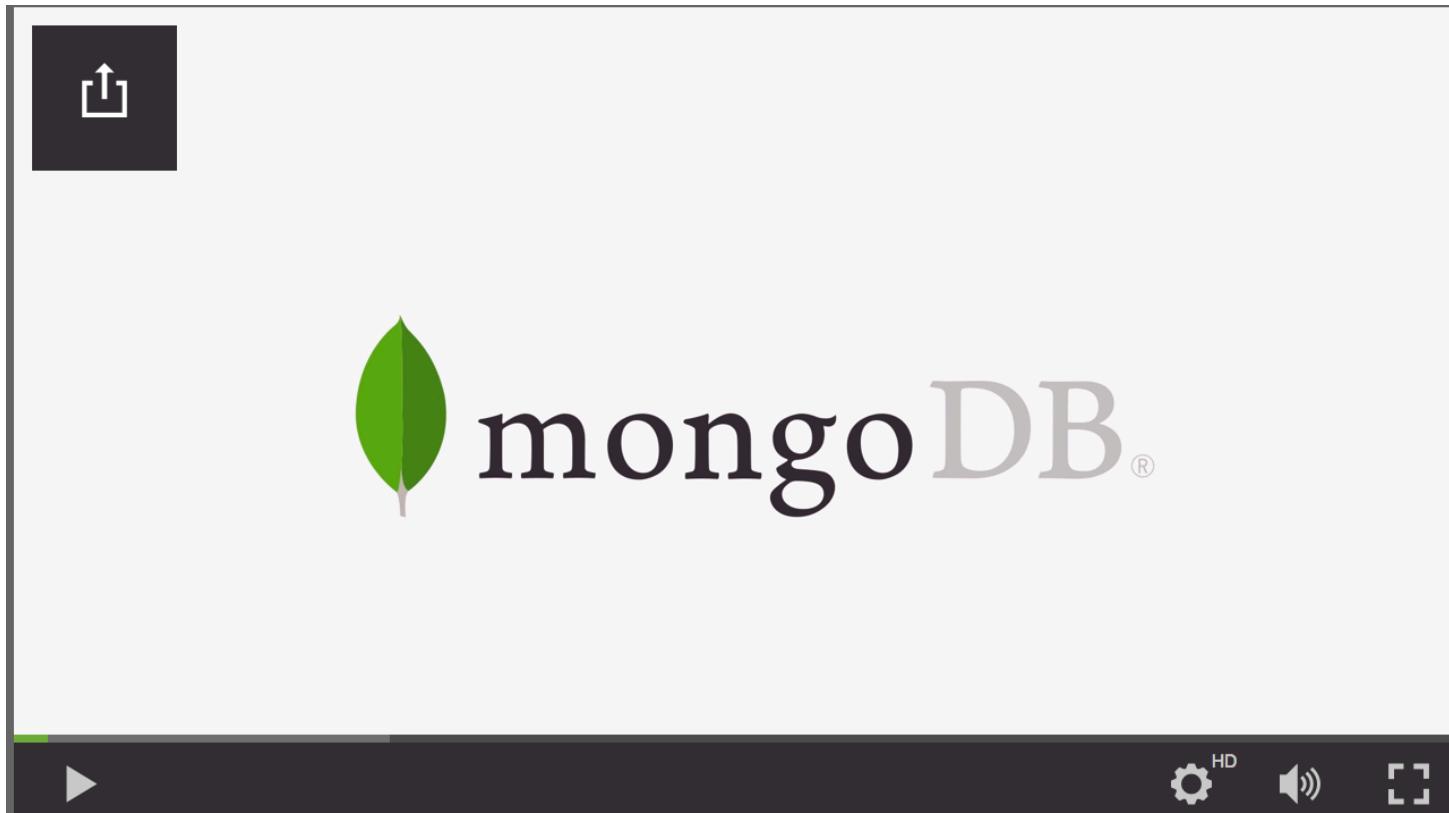
Reduce

{San Francisco Giants Hat, {Qty:6, Price : 54}}

{San Francisco Giants Hat M, {Qty:1, Price : 10}}



MongoDB



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

MongoDB Interview Questions

MongoDB's type

MongoDB's characteristics

Alternative databases

Supported programming languages

Index

Aggregation Operations(aggregation pipeline)

Sharding

Replication

GridFS

ObjectId

Consistency



Document Databases

Document Databases: For storing and retrieving documents with self-contained schema including XML, JSON, etc.

- Store documents in the value part of the key-value store.

```
<?xml version="1.0" standalone="yes"?>
<BankAccount>
    <Number>1234</Number>
    <Type>Checking</Type>
    <OpenDate>11/04/1974</OpenDate>
    <Balance>25382.20</Balance>
    <AccountHolder>
        <LastName>Singh</LastName>
        <FirstName>Darshan</FirstName>
    </AccountHolder>
</BankAccount>
```

```
{
    "empid": "SJ011MS",
    "personal": {
        "name": "Smith Jones",
        "gender": "Male",
        "age": 28,
        "address": {
            "streetaddress": "7 24th Street",
            "city": "New York",
            "state": "NY",
            "postalcode": "10038"
        }
    },
    "profile": {
        "designation": "Deputy General",
        "department": "Finance"
    }
}
```

www.kodingmadesimple.com



UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Document Databases

- Schema of the data can differ across documents, but these documents can still belong to the same collection.
 - Some attributes do not exist in another document.
 - New attributes can be created without defining them in the existing documents.

```
{  
  _id : 1234  
  firname : Diane,  
  lastname : Woodbridge,  
  address :  
    {state: CA, city: Corte Madera},  
  Interest : Books  
}
```

```
{  
  _id :2345  
  firname : Diane,  
  lastname : Woodbridge,  
  order :  
    {"Chodorow, Kristina. MongoDB: the definitive guide. " O'Reilly Media, Inc.", 2013."}  
  type : e-copy  
}
```



Document Databases

Types

44 systems in ranking, October 2016

Rank			DBMS	Database Model	Score		
Oct 2016	Sep 2016	Oct 2015			Oct 2016	Sep 2016	Oct 2015
1.	1.	1.	MongoDB 	Document store	318.80	+2.81	+25.54
2.	2.	↑ 3.	Couchbase 	Document store	29.30	+0.77	+3.11
3.	3.	↑ 4.	Amazon DynamoDB 	Document store	28.98	+1.56	+7.65
4.	4.	↓ 2.	CouchDB	Document store	22.18	+0.71	-4.69
5.	5.	5.	MarkLogic	Multi-model 	10.30	+0.15	-1.04
6.	6.	↑ 8.	OrientDB 	Multi-model 	6.25	-0.15	+1.32
7.	7.	↑ 10.	RethinkDB	Document store	5.23	+0.21	+1.88
8.	8.	↓ 7.	Cloudant	Document store	4.52	+0.04	-0.67
9.	9.	↓ 6.	RavenDB	Document store	4.07	-0.34	-1.75
10.	10.	↓ 9.	GemFire	Document store	3.14	-0.25	-1.56

<http://db-engines.com/en/ranking/document+store>



MongoDB

Terminology

- Documents
- Collections
- Databases

RDBMS	MongoDB
database	database
table	collection
row	document
column	field

<https://docs.mongodb.com/v3.2/reference/sql-comparison/>



Why MongoDB?

 mongoDB. | FOR GIANT IDEAS

Try MongoDB for

SOLUTIONS CLOUD CUSTOMERS RESOURCES ABOUT US



Expressive Query Language &
Secondary Indexes



Strong Consistency



Enterprise Management & Integration



Flexibility



Scalability & Performance



Always On, Global Deployments



Why MongoDB?

- Easy of use
 - No defined schema : Key/values are not fixed types/sizes.
- Easy scaling
 - MongoDB takes care of
 - Balancing data.
 - Loading data across a cluster.
 - Redistributing documents automatically.
 - Routing user request to the correct machines.





Why MongoDB?

- Many features
 - Creating, reading, updating, and deleting (CRUD) data.
 - Indexing : Supports secondary indexes, allowing fast queries.
 - Aggregation pipeline : Allow you to build complex aggregations from simple pieces.
 - Special collection types : Time-to-live collections(session), fixed-size collections.
 - File storage : Stores large files and file meta data (GridFS).
- Supported drivers
 - C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go and Erlang

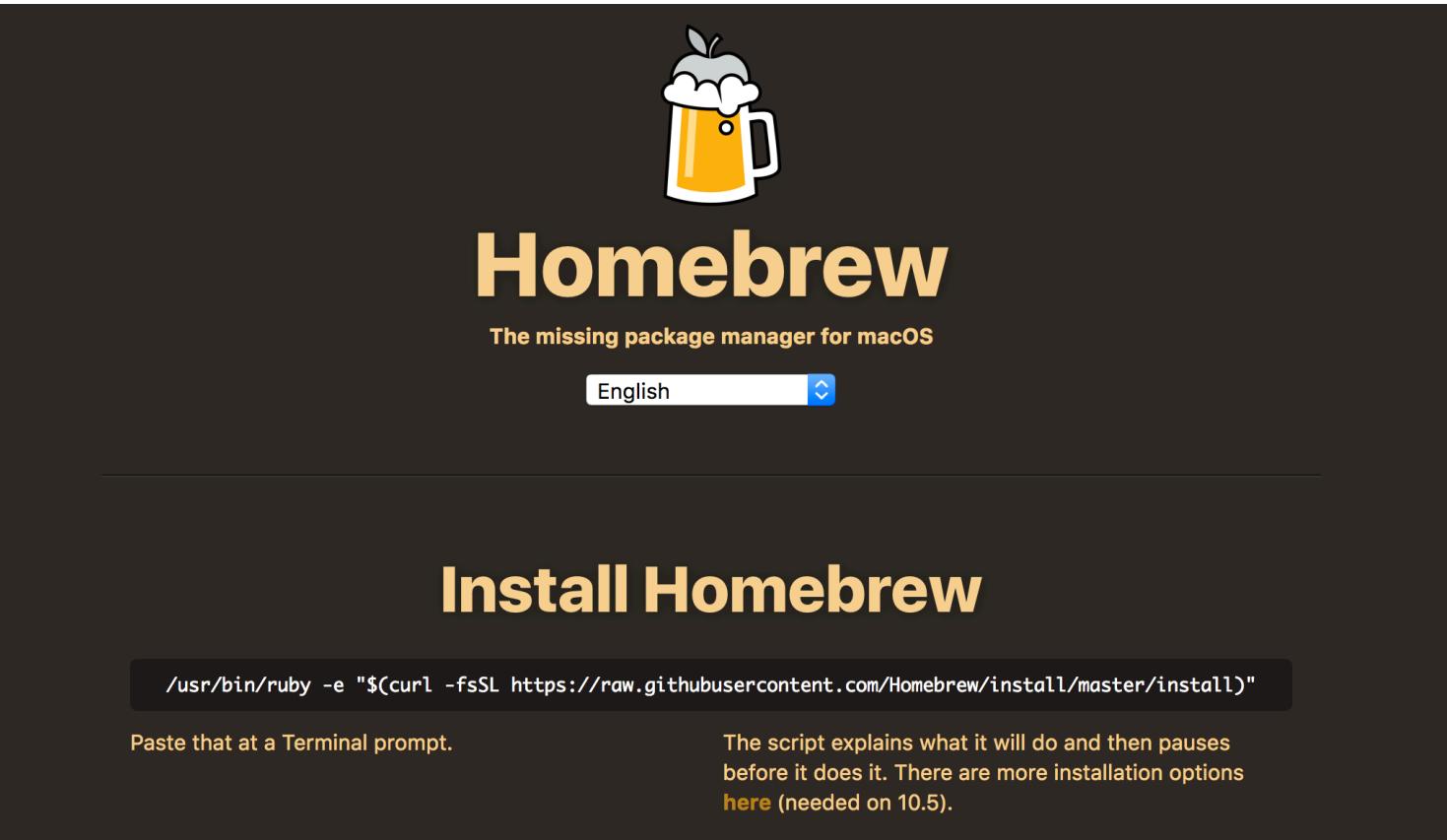
<https://docs.mongodb.com/ecosystem/drivers/>



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

MongoDB

Not installed Homebrew??

A screenshot of the Homebrew website. The page has a dark background. At the top center is a white icon of a mug of beer with foam and an apple on top. Below the icon, the word "Homebrew" is written in a large, bold, orange sans-serif font. Underneath "Homebrew", the text "The missing package manager for macOS" is written in a smaller, orange sans-serif font. Below this text is a language selection dropdown menu showing "English" with a downward arrow. A horizontal line separates the header from the main content area. In the main content area, the word "Install Homebrew" is centered in a large, bold, orange sans-serif font. Below this, a code block contains the command "/usr/bin/ruby -e "\$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)". To the left of the command, the text "Paste that at a Terminal prompt." is written in a small, orange sans-serif font. To the right of the command, the text "The script explains what it will do and then pauses before it does it. There are more installation options here (needed on 10.5)." is written in a small, orange sans-serif font.

Install Homebrew

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Paste that at a Terminal prompt.

The script explains what it will do and then pauses before it does it. There are more installation options [here](#) (needed on 10.5).

<http://brew.sh/>



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

MongoDB

Install MongoDB

```
brew update
```

```
brew install mongodb
```

```
ML-ITS-603436:MSAN697 dwoodbridge$ brew install mongodb
==> Auto-updated Homebrew!
Updated 1 tap (homebrew/core).
==> New Formulae
archi-steam-farm      hesiod          pyinvoke        terragrunt
consul-backinator     libsquish        questdb        ttyd
csvtomm               nvc              rmlint         typescript
dbhash                 osrm-backend    sql-translator  yarn
hana                  piknik          svgcleaner
==> Updated Formulae
abi-compliance-checker glib ✓          oysttyer
ace                   gnome-builder   pazpar2
afl-fuzz              gnome-themes-standard  ncan dnsproxy
```

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>



MongoDB

Create /data/db

Start the server : Run mongod executable.

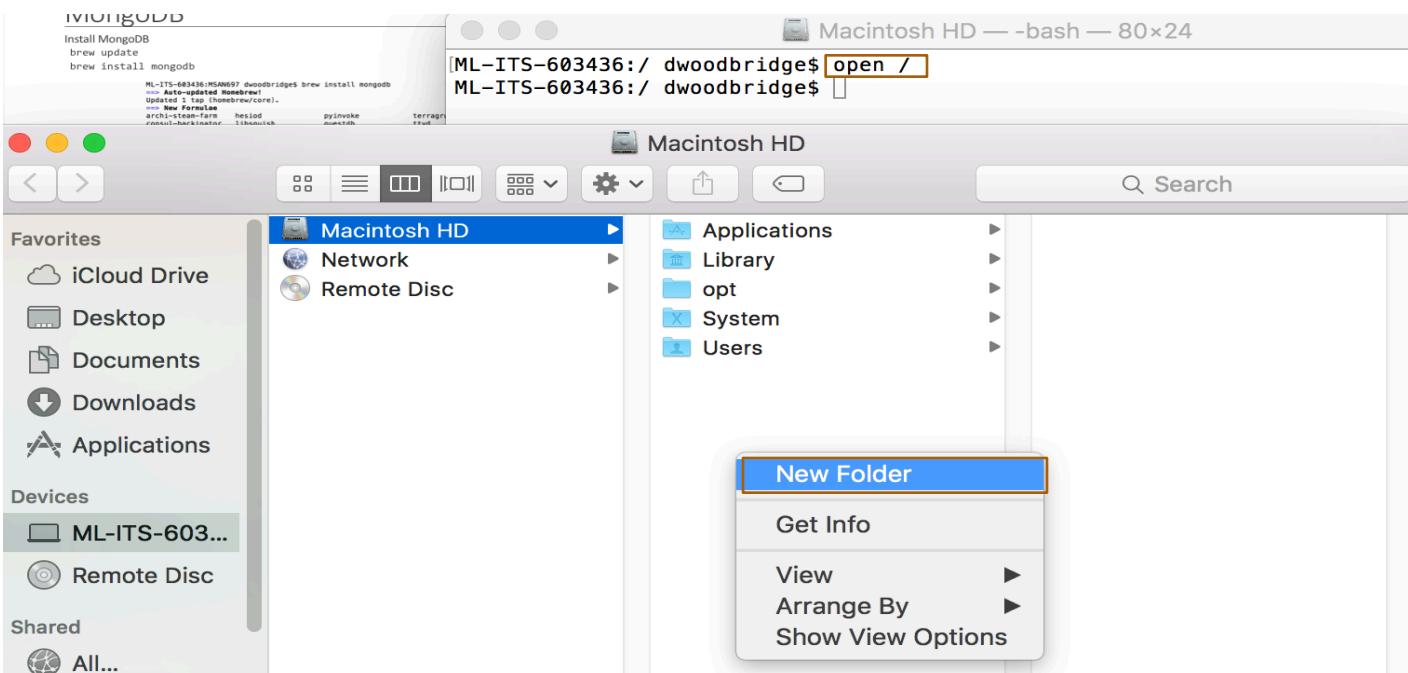
```
|ML-ITS-603436:MSAN697 dwoodbridge$ mongod
2016-10-17T14:46:43.520-0700 I CONTROL [initandlisten] MongoDB starting : pid=9
074 port=27017 dbpath=/data/db 64-bit host=ML-ITS-603436
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] db version v3.2.10
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] git version: 79d9b3ab5ce
20f51c272b4411202710a082d0317
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] OpenSSL version: OpenSSL
1.0.2j 26 Sep 2016
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] allocator: system
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] modules: none
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] build environment:
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] distarch: x86_64
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] target_arch: x86_64
2016-10-17T14:46:43.521-0700 I CONTROL [initandlisten] options: {}
2016-10-17T14:46:43.522-0700 I STORAGE [initandlisten] exception in initAndList
en: 29 Data directory /data/db not found., terminating
2016-10-17T14:46:43.522-0700 I CONTROL [initandlisten] dbexit: rc: 100
```

mongod will use default data directory, /data/db/



MongoDB

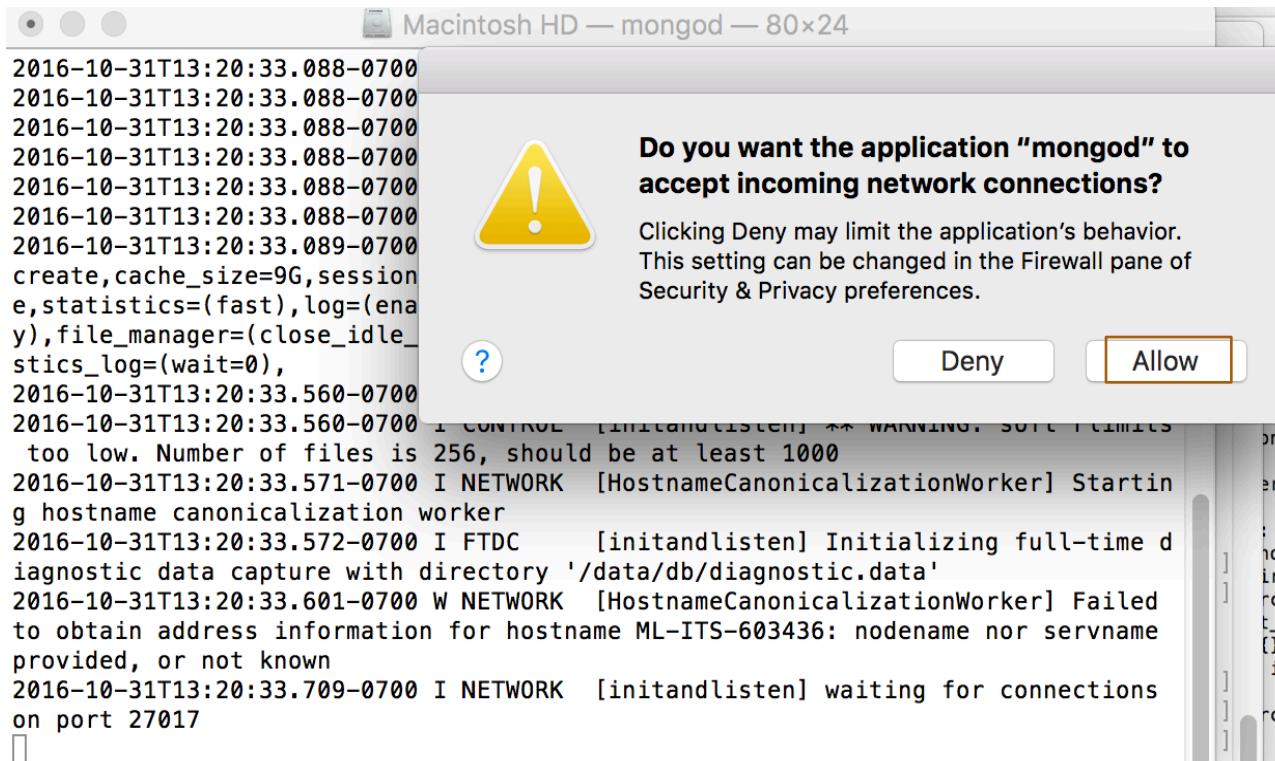
Create /data/db



MongoDB

Start the server : Call **mongod**

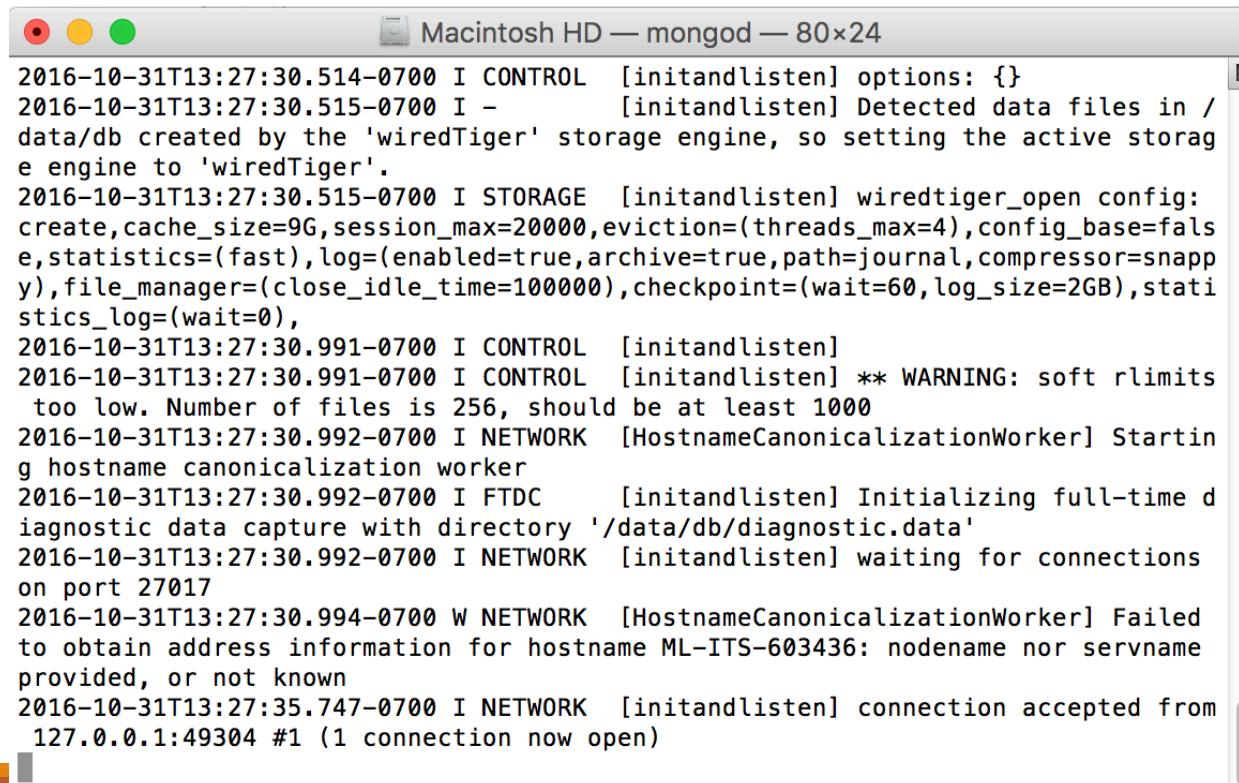
Stop mongod : “**Ctrl-C**”



MongoDB

Start the server : Call **mongod**

Stop mongod : “**Ctrl-C**”



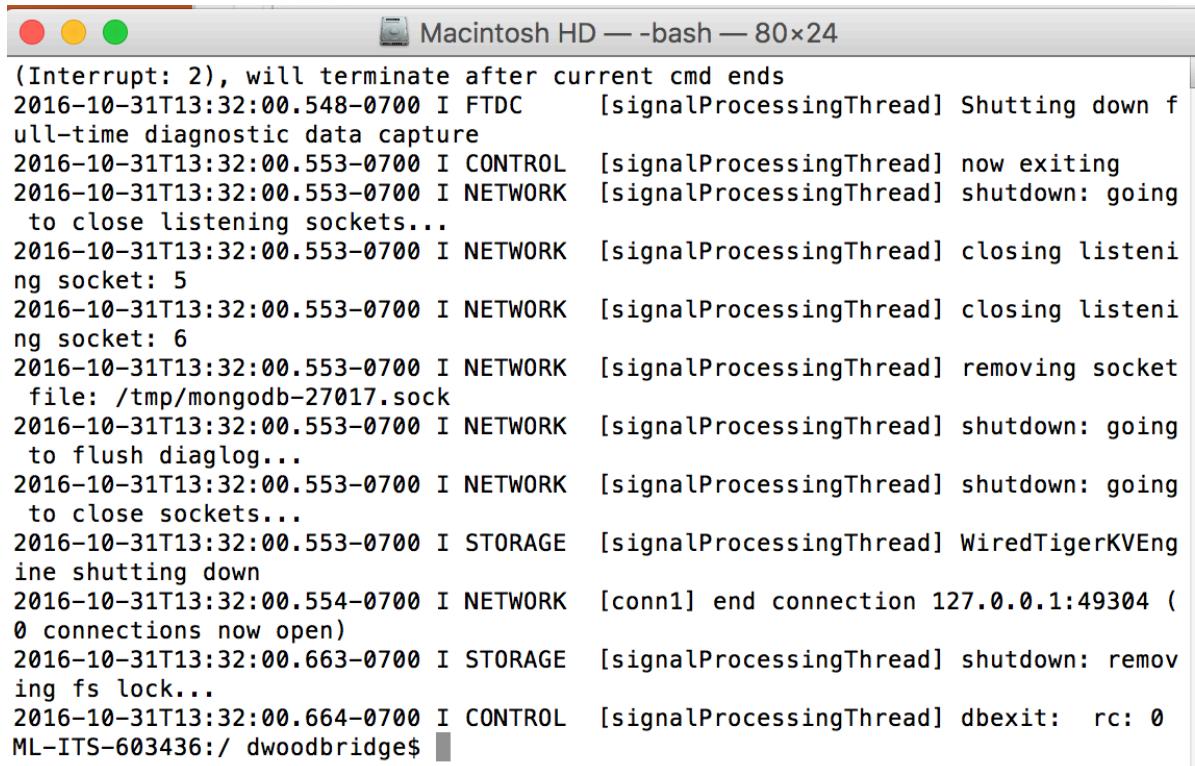
```
Macintosh HD — mongod — 80x24
2016-10-31T13:27:30.514-0700 I CONTROL  [initandlisten] options: {}
2016-10-31T13:27:30.515-0700 I -          [initandlisten] Detected data files in /
data/db created by the 'wiredTiger' storage engine, so setting the active storag
e engine to 'wiredTiger'.
2016-10-31T13:27:30.515-0700 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=9G,session_max=20000,eviction=(threads_max=4),config_base=fals
e,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snapp
y,file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),stati
stics_log=(wait=0),
2016-10-31T13:27:30.991-0700 I CONTROL  [initandlisten]
2016-10-31T13:27:30.991-0700 I CONTROL  [initandlisten] ** WARNING: soft rlimits
too low. Number of files is 256, should be at least 1000
2016-10-31T13:27:30.992-0700 I NETWORK  [HostnameCanonicalizationWorker] Startin
g hostname canonicalization worker
2016-10-31T13:27:30.992-0700 I FTDC      [initandlisten] Initializing full-time d
iagnostic data capture with directory '/data/db/diagnostic.data'
2016-10-31T13:27:30.992-0700 I NETWORK  [initandlisten] waiting for connections
on port 27017
2016-10-31T13:27:30.994-0700 W NETWORK  [HostnameCanonicalizationWorker] Failed
to obtain address information for hostname ML-ITS-603436: nodename nor servname
provided, or not known
2016-10-31T13:27:35.747-0700 I NETWORK  [initandlisten] connection accepted from
127.0.0.1:49304 #1 (1 connection now open)
```



MongoDB

Start the server : Call **mongod**

Stop mongod : “**Ctrl-C**”



A screenshot of a Mac OS X terminal window titled "Macintosh HD — bash — 80x24". The window shows the output of a MongoDB shutdown process. The text is as follows:

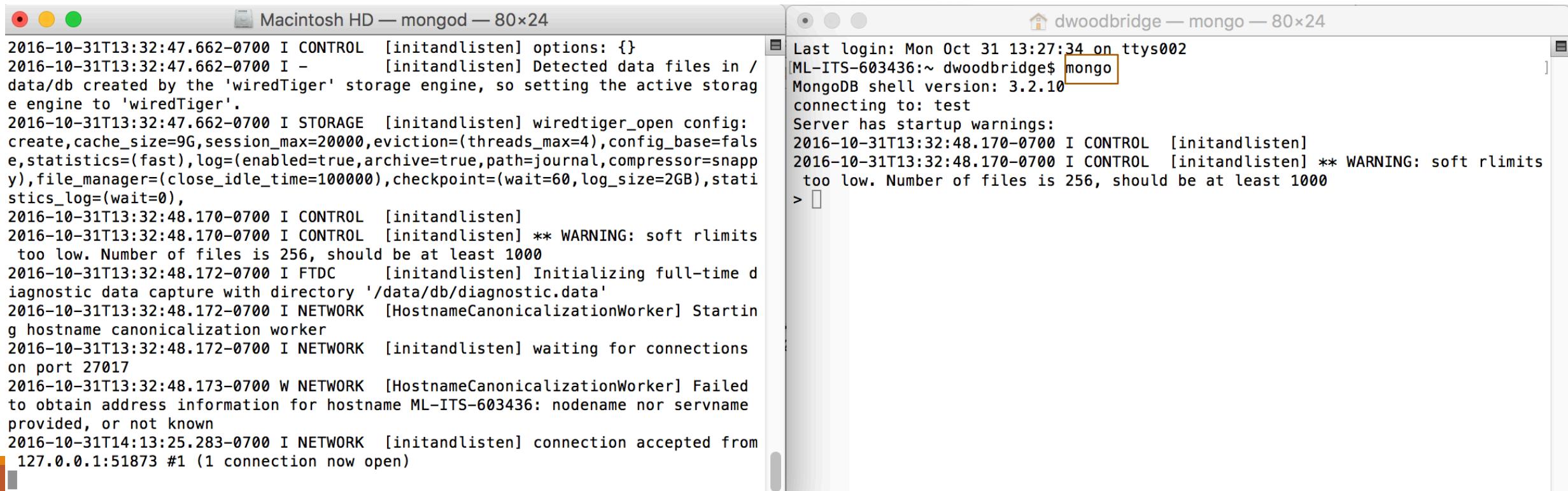
```
(Interrupt: 2), will terminate after current cmd ends
2016-10-31T13:32:00.548-0700 I FTDC      [signalProcessingThread] Shutting down full-time diagnostic data capture
2016-10-31T13:32:00.553-0700 I CONTROL   [signalProcessingThread] now exiting
2016-10-31T13:32:00.553-0700 I NETWORK   [signalProcessingThread] shutdown: going to close listening sockets...
2016-10-31T13:32:00.553-0700 I NETWORK   [signalProcessingThread] closing listening socket: 5
2016-10-31T13:32:00.553-0700 I NETWORK   [signalProcessingThread] closing listening socket: 6
2016-10-31T13:32:00.553-0700 I NETWORK   [signalProcessingThread] removing socket file: /tmp/mongodb-27017.sock
2016-10-31T13:32:00.553-0700 I NETWORK   [signalProcessingThread] shutdown: going to flush diaglog...
2016-10-31T13:32:00.553-0700 I NETWORK   [signalProcessingThread] shutdown: going to close sockets...
2016-10-31T13:32:00.553-0700 I STORAGE   [signalProcessingThread] WiredTigerKVEngine shutting down
2016-10-31T13:32:00.554-0700 I NETWORK   [conn1] end connection 127.0.0.1:49304 (0 connections now open)
2016-10-31T13:32:00.663-0700 I STORAGE   [signalProcessingThread] shutdown: removing fs lock...
2016-10-31T13:32:00.664-0700 I CONTROL   [signalProcessingThread] dbexit: rc: 0
ML-ITS-603436:/ dwoodbridge$
```



MongoDB

Running MongoDB Shell

- While your server is on.
- Run **mongo** executable.



The image shows two side-by-side terminal windows. The left window, titled "Macintosh HD — mongod — 80x24", displays the log output of the mongod process. It includes configuration details like storage engine ('wiredTiger'), cache size ('9G'), and log settings ('log_size=2GB'). It also shows startup warnings about soft rlimits being too low (number of files set to 256) and a failed attempt to obtain address information for the hostname. The right window, titled "dwoodbridge — mongo — 80x24", shows the mongo shell running on a Mac OS X system. It connects to the 'test' database and lists startup warnings, including a warning about soft rlimits being too low (number of files set to 256, should be at least 1000).

```
Macintosh HD — mongod — 80x24
2016-10-31T13:32:47.662-0700 I CONTROL [initandlisten] options: {}
2016-10-31T13:32:47.662-0700 I - [initandlisten] Detected data files in /data/db created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTiger'.
2016-10-31T13:32:47.662-0700 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=9G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2016-10-31T13:32:48.170-0700 I CONTROL [initandlisten]
2016-10-31T13:32:48.170-0700 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
2016-10-31T13:32:48.172-0700 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
2016-10-31T13:32:48.172-0700 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-10-31T13:32:48.172-0700 I NETWORK [initandlisten] waiting for connections on port 27017
2016-10-31T13:32:48.173-0700 W NETWORK [HostnameCanonicalizationWorker] Failed to obtain address information for hostname ML-ITS-603436: nodename nor servname provided, or not known
2016-10-31T14:13:25.283-0700 I NETWORK [initandlisten] connection accepted from 127.0.0.1:51873 #1 (1 connection now open)

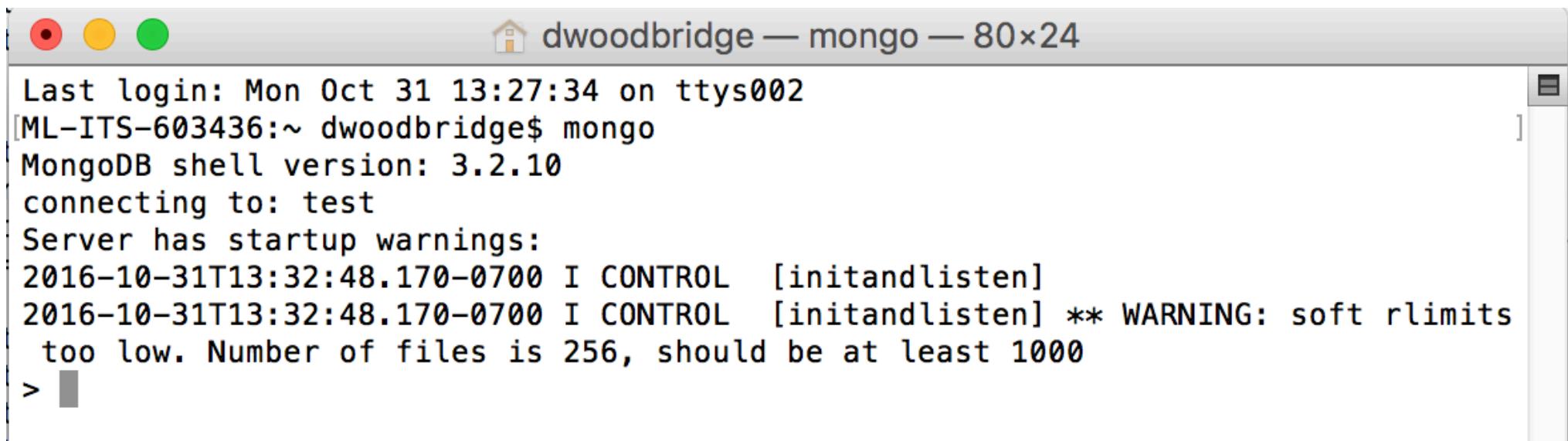
dwoodbridge — mongo — 80x24
Last login: Mon Oct 31 13:27:34 on ttys002
ML-ITS-603436:~ dwoodbridge$ mongo
MongoDB shell version: 3.2.10
connecting to: test
Server has startup warnings:
2016-10-31T13:32:48.170-0700 I CONTROL [initandlisten]
2016-10-31T13:32:48.170-0700 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
> 
```



MongoDB

Running MongoDB Shell

- While your server is on.
- Run **mongo** executable.



A screenshot of a terminal window titled "dwoodbridge — mongo — 80x24". The window shows the following text:

```
Last login: Mon Oct 31 13:27:34 on ttys002
[ML-ITS-603436:~ dwoodbridge$ mongo
MongoDB shell version: 3.2.10
connecting to: test
Server has startup warnings:
2016-10-31T13:32:48.170-0700 I CONTROL  [initandlisten]
2016-10-31T13:32:48.170-0700 I CONTROL  [initandlisten] ** WARNING: soft rlimits
too low. Number of files is 256, should be at least 1000
>
```



MongoDB

Commonly Supported Data Types

- Null- null
- Boolean – true, false
- Number – 64 bit float (default), NumberInt() (4-byte), NumberLong() (8 byte)
- String – UTF-8 characters
- Date – Stored as millisecond since the epoch (No time zone). new Date()
- Regular expression – Use JavaScript’s regular expression syntax.
- Array – [Element1, Element2,...]
- Embedded document – Entire documents can be embedded as value.
- Object ID – 12-Byte ID for documents. “_id”
- Binary data – String of arbitrary bytes.
- Code – JavaScript code.



MongoDB

Commonly Supported Data Types

- Date – Stored as millisecond since the epoch (No time zone). `new Date()`

```
[> new Date()
ISODate("2016-11-01T16:24:34.579Z")
[> a = {"x" : new Date()}
{ "x" : ISODate("2016-11-01T16:24:59.520Z") }
[> new Date("2016-11-02")
TSODate("2016-11-02T00:00:00Z")]
```

<http://www.epochconverter.com/>



MongoDB

Commonly Supported Data Types

- Null- null
- Boolean – true, false
- Number – 64 bit float (default), NumberInt() (4-byte), NumberLong() (8 byte)
- String – UTF-8 characters
- Date – Stored as millisecond since the epoch (No time zone). new Date()
- Regular expression – Use JavaScript’s regular expression syntax.
- Array – [Element1, Element2,...]
- Embedded document – Entire documents can be embedded as value.
- Object ID – 12-Byte ID for documents. “_id”
- Binary data – String of arbitrary bytes.
- Code – JavaScript code.



MongoDB

Commonly Supported Data Types

- Null- null
- Boolean – true, false
- Number – 64 bit float (default), NumberInt() (4-byte), NumberLong() (8 byte)
- String – UTF-8 characters
- Date – Stored as millisecond since the epoch (No time zone). new Date()
- Regular expression – Use JavaScript's regular expression syntax.
- **Array** – [Element1, Element2,...]
- Embedded document – Entire documents can be embedded as value.
- Object ID – 12-Byte ID for documents. “_id”
- Binary data – String of arbitrary bytes.
- Code – JavaScript code.



MongoDB

Commonly Supported Data Types

- Array – [Element1, Element2,...]
 - Used for ordered operations and unordered operations.
 - Can contain different data types.
 - Atomic updates to modify the contents of arrays.

```
> diane = {"favorites" : [7,11]}\n{ "favorites" : [ 7, 11 ] }\n> diane = {"favorites" : [7,11, "Puppies"]}\n{ "favorites" : [ 7, 11, "Puppies" ] }\n> diane = {"favorites" : [7,11, "Puppies", ["Youghurt","Coffee"]]}\n    "favorites" : [\n        7,\n        11,\n        "Puppies",\n        [\n            "Youghurt",\n            "Coffee"\n        ]\n    ]\n}
```

MongoDB

Commonly Supported Data Types

- Null- null
- Boolean – true, false
- Number – 64 bit float (default), NumberInt() (4-byte), NumberLong() (8 byte)
- String – UTF-8 characters
- Date – Stored as millisecond since the epoch (No time zone). new Date()
- Regular expression – Use JavaScript's regular expression syntax.
- Array – [Element1, Element2,...]
- **Embedded document** – Entire documents can be embedded as value.
- Object ID – 12-Byte ID for documents. “_id”
- Binary data – String of arbitrary bytes.
- Code – JavaScript code.



MongoDB

Commonly Supported Data Types

- Embedded document – Entire documents can be embedded as value.

```
> diane = {"favorite" : { "numbers" : [7,11], "animals" : ["Puppies"], "food" :["Yogurt","Coffee"]}}
```

```
{  
    "favorite" : {  
        "numbers" : [  
            7,  
            11  
        ],  
        "animals" : [  
            "Puppies"  
        ],  
        "food" : [  
            "Yogurt",  
            "Coffee"  
        ]  
    }  
}
```



MongoDB

Commonly Supported Data Types

- Embedded document – Entire documents can be embedded as value.

```
> diane = {"name":"Diane MK Woodbridge", "address" : {"street" : "101 Howard", "city" :" San Francisco", "state" : "CA"}  
{  
    "name" : "Diane MK Woodbridge",  
    "address" : {  
        "street" : "101 Howard",  
        "city" : " San Francisco",  
        "state" : "CA"  
    }  
}
```



MongoDB

Commonly Supported Data Types

- Null- null
- Boolean – true, false
- Number – 64 bit float (default), NumberInt() (4-byte), NumberLong() (8 byte)
- String – UTF-8 characters
- Date – Stored as millisecond since the epoch (No time zone). new Date()
- Regular expression – Use JavaScript's regular expression syntax.
- Array – [Element1, Element2,...]
- Embedded document – Entire documents can be embedded as value.
- Object ID – 12-Byte ID for documents. “_id”
- Binary data – String of arbitrary bytes.
- Code – JavaScript code.



MongoDB

Commonly Supported Data Types

- Object ID – 12-byte ID for documents and a default for “`_id`”.
 - Generated by the driver on the client.
 - Every document in MongoDB must have an “`_id`” key.
 - Its value can be any type, but needs to be unique for a collection.
 - Default is `ObjectId`.



MongoDB

MongoDB Shell

- Running a shell
 - Basic math
 - JavaScript
 - Basic CRUD (Create, Read, Update, Delete)
 - Connecting to internal/external database.

```
[> x = 200  
200  
[> x / 23;  
8.695652173913043  
> █
```

```
[> Math.PI  
3.141592653589793  
[>  
[> Math.sin(Math.PI)  
1.2246467991473532e-16  
[>  
[> msan_string = "USF MSAN"  
USF MSAN  
[> msan_string.replace("MSAN", "Master in Analytics");  
USF Master in Analytics
```



MongoDB

Create/Choose a database

- use DB_Name

Check which database you're using.

- db

Access collection from the db variable.

- db.collection_name

```
[> use mydb
switched to db mydb
[> db
mydb
```



MongoDB

CRUD Overview

- Create (Insert)

- insert function add a document to a collection.
- db.collectionName.insert(document)

```
> diane = {"name":"Diane MK Woodbridge", "address" : {"street" : "101 Howard", "city" :" San Francisco", "state" : "CA"}  
{  
    "name" : "Diane MK Woodbridge",  
    "address" : {  
        "street" : "101 Howard",  
        "city" : " San Francisco",  
        "state" : "CA"  
    }  
}
```

```
[> db.friend.insert(diane)  
WriteResult({ "nInserted" : 1 })
```

Insert "Yannet" document into the collection.



MongoDB

CRUD Overview

- Create (Insert)
 - insert function add a document to a collection.
 - db.collectionName.insert(document)

```
> yannet = {"name": "Yannet Interian", "address": {"street": "101 Howard", "city": "San Francisco",  
{  
    "name" : "Yannet Interian",  
    "address" : {  
        "street" : "101 Howard",  
        "city" : "San Francisco",  
        "state" : "CA"  
    }  
}  
> db.friend.insert(yannet)  
WriteResult({ "nInserted" : 1 })  
> db.friend.find()  
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "s  
}
```

MongoDB

CRUD Overview

- Create (Insert)

- Bulk Insert : Takes an array of document.

```
> david = {"name" : "David Guy Brizan", "address" : { "street" : "101 Howard", "city" : "San Francisco", "state" : "CA" } }
{
  "name" : "David Guy Brizan",
  "address" : {
    "street" : "101 Howard",
    "city" : "San Francisco",
    "state" : "CA"
  }
}
> bruce = {"name" : "David Uminsky", "address" : { "street" : "101 Howard", "city" : "San Francisco", "state" : "CA"} }
{
  "name" : "David Uminsky",
  [
    "address" : {
      "street" : "101 Howard",
      [
        "city" : "San Francisco",
        "state" : "CA"
      ]
    }
  ]
}
> db.friend.insert([david, bruce])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
> db.friend.find()
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard", "city" : "San Francisco", "state" : "CA" }, "noKids" : 2 }
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101 Howard", "city" : "San Francisco", "state" : "CA" } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "street" : "101 Howard", "city" : "San Francisco", "state" : "CA" } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "name" : "David Uminsky", "address" : { "street" : "101 Howard", "city" : "San Francisco", "state" : "CA" } }
```



MongoDB

CRUD Overview

- Create (Insert)
 - Import raw data
 - mongoimport

```
[ML-ITS-603436:Week03 dwoodbridge$ mongoimport --db msan697 --collection business --file business.json  
2016-11-02T22:44:33.223-0700      connected to: localhost  
2016-11-02T22:44:33.879-0700      imported 25359 documents  
...  
[ML-ITS-603436:Week03 dwoodbridge$
```

Data From .. <https://github.com/dianewoodbridge/msan697-example>

<https://docs.mongodb.com/getting-started/shell/import-data/>



MongoDB

CRUD Overview

- Create (Insert)
 - Import raw data
 - mongoimport

```
> use msan697
switched to db msan697
> db.business.findOne();
{
    "_id" : ObjectId("581acec18ae04b51d182a130"),
    "address" : {
        "building" : "2780",
        "coord" : [
            -73.98241999999999,
            40.579505
        ],
        "street" : "Stillwell Avenue",
        "zipcode" : "11224"
    },
    "borough" : "Brooklyn",
    "cuisine" : "American",
    "grades" : [
        {
            "date" : ISODate("2014-06-10T00:00:00Z"),
            "grade" : "A",
            "score" : 5
        }
    ],
    "name" : "The Cheesecake Factory"
}
```

<https://docs.mongodb.com/getting-started/shell/import-data/>



MongoDB

CRUD Overview

- Read

- `find(criteria)`
- `findOne(criteria)`

```
[> db.friend.find()
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge",
}
```

```
[> db.friend.find({"address.city":"San Francisco"})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "city" : "San Francisco", "state" : "CA" }, { "name" : "Diane MK Woodbridge", "city" : "San Francisco", "state" : "CA" }, { "name" : "Diane MK Woodbridge", "city" : "San Francisco", "state" : "CA" }
```

```
> db.friend.findOne()
{
  "_id" : ObjectId("581ad92303fdda1ba29ed169"),
  "name" : "Diane MK Woodbridge",
  "address" : {
    "street" : "101 Howard",
    "city" : "San Francisco",
    "state" : "CA"
}
```

Example 1

In the business collection, find all with grade is "C" in "business" collection under the "msan697" database.



MongoDB

CRUD Overview

- Update
 - Atomic.
 - `update(Arg1, Arg2, Arg3, Arg4, Arg5)`
 - Two required arguments.
 - Arg1: Criteria to find
 - Arg2: new document/change criteria.
 - Three optional arguments.
 - Arg3 : Upsert (default = false)
 - When no document meets the Arg1 criteria, it creates a new document.
 - Arg4 : multiple-update (default = false).
 - Update multiple documents that meet the Arg1 criteria.
 - Arg5 : writeConcern. Error handling.

<https://docs.mongodb.com/v3.2/reference/method/db.collection.update/>

MongoDB

CRUD Overview

- Update
 - Update the entire document.

```
> db.friend.findOne()
{
  "_id" : ObjectId("581ad92303fdda1ba29ed169"),
  "name" : "Diane MK Woodbridge",
  "address" : {
    "street" : "101 Howard",
    "city" : "San Francisco",
    "state" : "CA"
  }
}
```



MongoDB

CRUD Overview

- Update
 - Update the entire document.

```
[> diane.noKids = 2;
2
[> diane
{
    "name" : "Diane MK Woodbridge",
    "address" : {
        "street" : "101 Howard",
        "city" : " San Francisco",
        "state" : "CA"
    },
    "noKids" : 2
}
diane.much.updated / name -> Diane MK Woodbridge
```



MongoDB

CRUD Overview

- Update
 - Update the entire document.

```
[> db.friend.update({"name": "Diane MK Woodbridge"}, diane)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
[> db.friend.find({"name": "Diane MK Woodbridge"})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "st
"noKids" : 2 }
```



MongoDB

CRUD Overview

- Update
 - Update certain portions of a document.
 - Use atomic update modifier.
 - Field modifier
 - Array modifier
 - Operators
 - Modifiers



MongoDB

Field modifier

Name	Description
<code>\$inc</code>	Increments the value of the field by the specified amount.
<code>\$mul</code>	Multiplies the value of the field by the specified amount.
<code>\$rename</code>	Renames a field.
<code>\$setOnInsert</code>	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
<code>\$set</code>	Sets the value of a field in a document.
<code>\$unset</code>	Removes the specified field from a document.
<code>\$min</code>	Only updates the field if the specified value is less than the existing field value.
<code>\$max</code>	Only updates the field if the specified value is greater than the existing field value.
<code>\$currentDate</code>	Sets the value of a field to current date, either as a Date or a Timestamp.

MongoDB

Field modifier

- \$set : replaces the value of a field with the specified value.
 - Use in the Arg2 field. ex. {\$set : {key:value}}
- \$unset : delete a particular field.
 - Use in the Arg2 field. ex. {\$unset : {key:anything}}

```
[> db.friend.update({"name" : "Diane MK Woodbridge"}, {$set:{"noCats":1}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
[> db.friend.find({"name":"Diane MK Woodbridge"})  
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address":  
2, "noCats" : 1 }
```



MongoDB

Field modifier

- \$set : replaces the value of a field with the specified value.
 - Use in the Arg2 field. ex. {\$set : {key:value}}
- \$unset : delete a particular field.
 - Use in the Arg2 field. ex. {\$unset : {key:anything}}

```
> db.friend.update({"name" : "Diane MK Woodbridge"}, {"$unset":{"noCats":1}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.friend.find({"name":"Diane MK Woodbridge"})  
[{"_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" :  
"123 Main Street", "age" : 30, "noCats" : 0}]
```



MongoDB

Field modifier

- {\$inc : {key: value} }
- Useful for updating votes, scores, etc.

```
[> db.friend.update({"name": "Diane MK Woodbridge"}, {$inc : {"noCats": 2}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
[> db.friend.findOne({"name": "Diane MK Woodbridge"})  
{  
    "_id" : ObjectId("581ad92303fdda1ba29ed169"),  
    "name" : "Diane MK Woodbridge",  
    "address" : {  
        "street" : "101 Howard",  
        "city" : "San Francisco",  
        "state" : "CA"  
    },  
    "noKids" : 2,  
    "noCats" : 3  
}
```



MongoDB

CRUD Overview

- Update
 - Update certain portions of a document.
 - Use atomic update modifier.
 - Array modifier
 - Operators : \$, \$addToSet, \$pop, \$pullAll, \$pull, \$pushAll, \$push
 - Modifiers : \$each, \$slice, \$sort, \$position



MongoDB

Array modifier Operators

- Operators

Name	Description
<code>\$</code>	Acts as a placeholder to update the first element that matches the query condition in an update.
<code>\$addToSet</code>	Adds elements to an array only if they do not already exist in the set.
<code>\$pop</code>	Removes the first or last item of an array.
<code>\$pullAll</code>	Removes all matching values from an array.
<code>\$pull</code>	Removes all array elements that match a specified query.
<code>\$pushAll</code>	<i>Deprecated.</i> Adds several items to an array.
<code>\$push</code>	Adds an item to an array.

MongoDB

Array modifier

- Modifier

Modifiers

Name	Description
<code>\$each</code>	Modifies the <code>\$push</code> and <code>\$addToSet</code> operators to append multiple items for array updates.
<code>\$slice</code>	Modifies the <code>\$push</code> operator to limit the size of updated arrays.
<code>\$sort</code>	Modifies the <code>\$push</code> operator to reorder documents stored in an array.
<code>\$position</code>	Modifies the <code>\$push</code> operator to specify the position in the array to add elements.

MongoDB

Array modifier

- Operators

- `$push` : Add elements to the end of an array.
- `{$push : {key : value}}`

```
[> db.friend.update({"name":"Diane MK Woodbridge"},{$push : {"comments": {"name" : "Abigail", "content" : "Please cook mac and cheese"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
[> db.friend.findOne()
{
  "_id" : ObjectId("581ad92303fdda1ba29ed169"),
  "name" : "Diane MK Woodbridge",
  "address" : {
    "street" : "101 Howard",
    "city" : "San Francisco",
    "state" : "CA"
  },
  "noKids" : 2,
  "noCats" : 3,
  "comments" : [
    {
      "name" : "Abigail",
      "content" : "Please cook mac and cheese"
    }
  ]
}
```

MongoDB

Array modifier

- Operators

- \$push : adding elements to the end of an array.

```
[> db.friend.update({"name": "Diane MK Woodbridge"}, {$push : {"comments": {"name" : "Bora", "content" : "Need more water."}}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
[> db.friend.findOne()  
{  
    "_id" : ObjectId("581ad92303fdda1ba29ed169"),  
    "name" : "Diane MK Woodbridge",  
    "address" : {  
        "street" : "101 Howard",  
        "city" : "San Francisco",  
        "state" : "CA"  
    },  
    "noKids" : 2,  
    "noCats" : 3,  
    "comments" : [  
        {  
            "name" : "Abigail",  
            "content" : "Please cook mac and cheese"  
        },  
        {  
            "name" : "Bora",  
            "content" : "Need more water."  
        }  
    ]  
}
```

MongoDB

Array modifier

- Modifier
 - \$each : Modify \$push and \$addToSet to append multiple items for array updates.

```
[> comment_1 = {"name": "David", "content" : "Want to make sure you're doing fine in your class."}
{
    "name" : "David",
    "content" : "Want to make sure you're doing fine in your class."
}
[> comment_2 = {"name": "Yannet", "content" : "How's your class going?"}
{ "name" : "Yannet", "content" : "How's your class going?" }
```

MongoDB

```
[> db.friend.update({"name" : "Diane MK Woodbridge"}, {$push:{"comments" : {$each: [comment_1,comment_2]}}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
[> db.friend.findOne()
```

```
{  
    "_id" : ObjectId("581ad92303fdda1ba29ed169"),  
    "name" : "Diane MK Woodbridge",  
    "address" : {  
        "street" : "101 Howard",  
        "city" : "San Francisco",  
        "state" : "CA"  
    },  
    "noKids" : 2,  
    "noCats" : 3,  
    "comments" : [  
        {  
            "name" : "Abigail",  
            "content" : "Please cook mac and cheese"  
        },  
        {  
            "name" : "Bora",  
            "content" : "Need more water."  
        },  
        {  
            "name" : "David",  
            "content" : "Want to make sure you're doing fine in your class."  
        },  
        {  
            "name" : "Yannet",  
            "content" : "How's your class going?"  
        }  
    ]  
}
```

MongoDB

CRUD Overview

- Delete
 - Permanently deletes documents from the database.
 - `remove(criteria)`
 - `drop()` : Delete entire collection efficiently.

```
> db.mydb.remove({"name" : "Yannet Interian"});
WriteResult({ "nRemoved" : 1 })
> db.mydb.find();
{ "_id" : ObjectId("581a1946ff611b997cb2bde7"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101
, "noKids" : 2 }
```



MongoDB

Use external database.

- mongo hostname:port/dbname
- conn = new Mongo("hostname:port")
db = conn.getDB("dbname")



Example 2

Connect to

- Host name : ec2-54-161-217-247.compute-1.amazonaws.com
- Port : 27017
- Database : msan

And find documents in “class” collection.



References

- Sadalage, Pramod J., and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.
- Chodorow, Kristina. *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2013.
- MongoDB. *MongoDB Documentation*, <https://docs.mongodb.com/>, 2016.



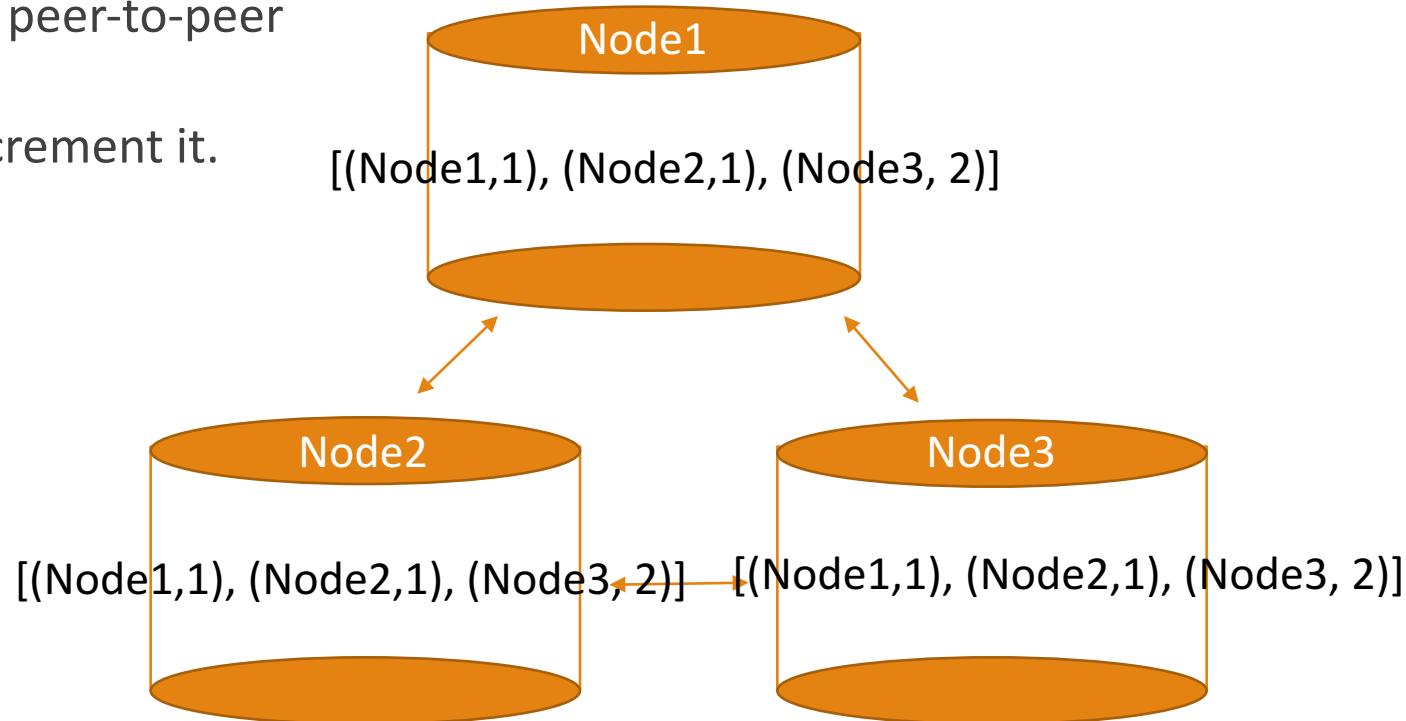
Further Readings



Version Stamps

Vector Stamp

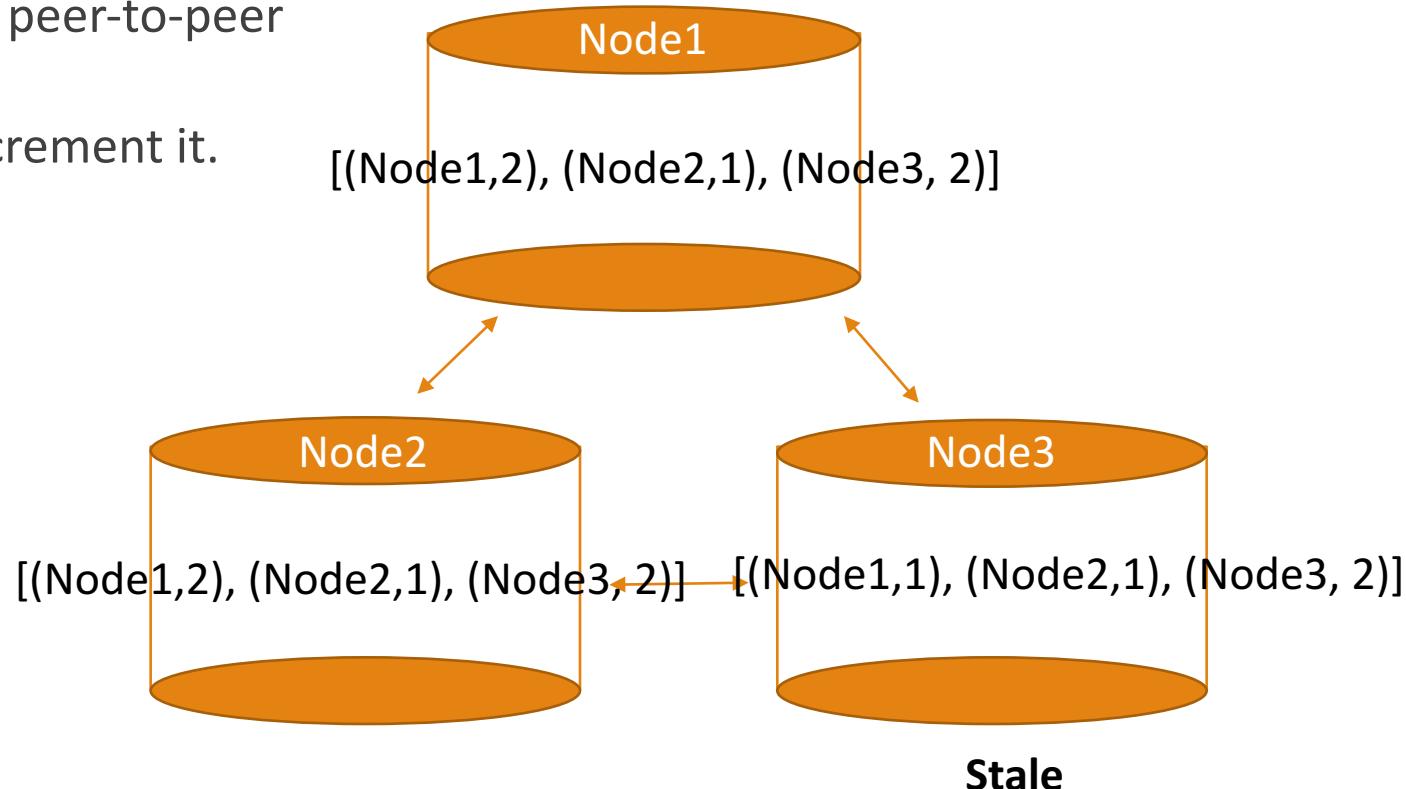
- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
 - Ex. $[(\text{Node1}, 1), (\text{Node2}, 2), (\text{Node3}, 6)]$



Version Stamps

Vector Stamp

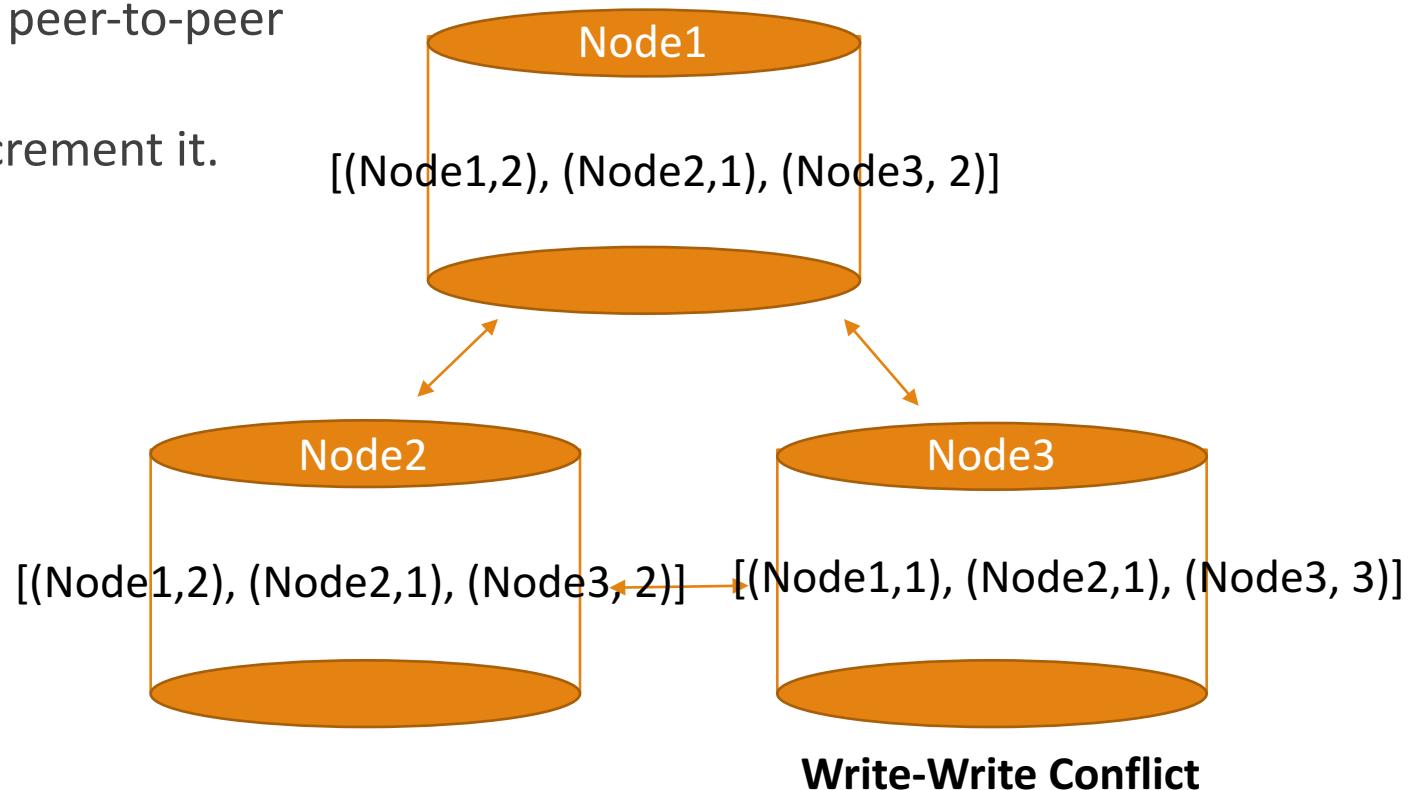
- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
 - Ex. $[(\text{Node1}, 1), (\text{Node2}, 2), (\text{Node3}, 6)]$



Version Stamps

Vector Stamp

- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
 - Ex. $[(\text{Node1}, 1), (\text{Node2}, 2), (\text{Node3}, 6)]$



MongoDB on AWS

The screenshot shows the AWS Services dashboard. At the top, there's a navigation bar with icons for Lambda, AWS, Services, and Edit, and a user profile for Diane Woodbridge. Below the navigation bar, there are two columns of service links:

History	All AWS Services
EC2	Compute
CloudFormation	Storage & Content Delivery
VPC	Database
Console Home	Networking
Billing	Developer Tools
RDS	Management Tools
	Security & Identity
	Analytics
	Internet of Things
	Mobile Services
	Application Services
	Enterprise Applications
	Game Development

In the center, there's a detailed view of the EC2 service. It includes the EC2 logo, a brief description of EC2, and links to EC2 Container Service and Lambda.

EC2
Amazon Elastic Compute Cloud (EC2) provides resizable compute capacity in the cloud.

EC2 Container Service
Amazon ECS allows you to easily run and manage Docker containers across a cluster of Amazon EC2 instances.

Lambda
AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you.

Elastic Beanstalk
AWS Elastic Beanstalk is an application container for deploying and managing applications.

Server Migration
AWS Server Migration Service makes it easier and faster for you to migrate on-premises servers to AWS by automating the replication of live server volumes incrementally.

<https://aws.amazon.com/blogs/aws/mongodb-on-the-aws-cloud-new-quick-start-reference-deployment/>

<https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-amazon/>



MongoDB on AWS

The screenshot shows the AWS EC2 Dashboard. The left sidebar contains navigation links for EC2 Dashboard, Instances, Images, Elastic Block Store, Network & Security, and Service Health. The main content area displays resource usage statistics and a 'Create Instance' section with a 'Launch Instance' button. The 'Service Health' section shows the US East (N. Virginia) region is operating normally. The right sidebar includes account attributes like VPC and Default VPC, additional information links, and an AWS Marketplace section.

Diane Woodbridge | N.

EC2 Dashboard

- Events
- Tags
- Reports
- Limits

INSTANCES

- Instances
- Spot Requests
- Reserved Instances
- Scheduled Instances
- Dedicated Hosts

IMAGES

- AMIs
- Bundle Tasks

ELASTIC BLOCK STORE

- Volumes
- Snapshots

NETWORK & SECURITY

- Security Groups
- Elastic IPs
- Placement Groups
- Key Pairs

Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

1 Running Instances	0 Elastic IPs
0 Dedicated Hosts	0 Snapshots
1 Volumes	0 Load Balancers
1 Key Pairs	2 Security Groups
0 Placement Groups	

Build and run distributed, fault-tolerant applications in the cloud with [Amazon Simple Workflow Service](#).

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US East (N. Virginia) region

Service Health

Service Status:

- US East (N. Virginia): This service is operating normally

Availability Zone Status:

Scheduled Events

US East (N. Virginia):

No events

Account Attributes

Supported Platforms

- VPC

Default VPC

vpc-2bffe14c

Resource ID length manager

Additional Information

[Getting Started Guide](#)

[Documentation](#)

[All EC2 Resources](#)

[Forums](#)

[Pricing](#)

[Contact Us](#)

AWS Marketplace

Find **free software trial products** in the AWS Marketplace from the [EC2](#) Marketplace. Or try these popular AMIs:

- [Tableau Server \(10 users\)](#)

Provided by Tableau

Rating



MongoDB on AWS

[1. Choose AMI](#) [2. Choose Instance Type](#) [3. Configure Instance](#) [4. Add Storage](#) [5. Tag Instance](#) [6. Configure Security Group](#) [7. Review](#)

Step 1: Choose an Amazon Machine Image (AMI)

[Cancel and Exit](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Quick Start

My AMIs

AWS Marketplace

Community AMIs

Free tier only ⓘ

Image	Name	Description	Select	64-bit
	Amazon Linux AMI 2016.09.0 (HVM), SSD Volume Type - ami-b73b63a0	The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.	Select	64-bit
	Red Hat Enterprise Linux 7.2 (HVM), SSD Volume Type - ami-2051294a	Red Hat Enterprise Linux version 7.2 (HVM), EBS General Purpose (SSD) Volume Type	Select	64-bit
	SUSE Linux Enterprise Server 12 SP1 (HVM), SSD Volume Type - ami-1eeab909	SUSE Linux Enterprise Server 12 Service Pack 1 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.	Select	64-bit
	Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-40d28157		Select	

1 to 31 of 31 AMIs



MongoDB on AWS

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types ▾ Current generation ▾ Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate
<input checked="" type="checkbox"/>	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	m4.large	2	8	EBS only	Yes	Moderate
<input type="checkbox"/>	General purpose	m4.xlarge	4	16	EBS only	Yes	High

Cancel Previous **Review and Launch** Next: Configure Instance Details



MongoDB on AWS

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance 6. Configure Security Group 7. Review

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances Launch into Auto Scaling Group [i](#)

Purchasing option [i](#) Request Spot instances

Network [i](#) vpc-2bffe14c (172.31.0.0/16) (default) [C](#) Create new VPC

Subnet [i](#) No preference (default subnet in any Availability Zone) [C](#) Create new subnet

Auto-assign Public IP [i](#) Use subnet setting (Enable)

IAM role [i](#) None [C](#) Create new IAM role

Shutdown behavior [i](#) Stop

Enable termination protection [i](#) Protect against accidental termination

Monitoring [i](#) Enable CloudWatch detailed monitoring
Additional charges apply.

Tenancy [i](#) Shared - Run a shared hardware instance
Additional charges will apply for dedicated tenancy

Cancel

Previous

Review and Launch

Next: Add Storage



MongoDB on AWS

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance 6. Configure Security Group 7. Review

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/xvda	snap-fe8a3c04	8	General Purpose SSD (GP2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted
Add New Volume								

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

Cancel

Previous

Review and Launch

Next: Tag Instance



MongoDB on AWS

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage **5. Tag Instance** 6. Configure Security Group 7. Review

Step 5: Tag Instance

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. [Learn more](#) about tagging your Amazon EC2 resources.

Key	(127 characters maximum)	Value	(255 characters maximum)
Name		MSAN_Mongodb	<input type="button" value="X"/>

Create Tag (Up to 50 tags maximum)

Cancel

Previous

Review and Launch

Next: Configure Security Group



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

MongoDB on AWS

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance **6. Configure Security Group** 7. Review

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: Create a new security group

Select an existing security group

Security group name: launch-wizard-1

Description: launch-wizard-1 created 2016-11-02T13:20:33.730-07:00

Type <small>i</small>	Protocol <small>i</small>	Port Range <small>i</small>	Source <small>i</small>
All traffic	All	0 - 65535	Anywhere <small>▼</small> 0.0.0.0/0 <small>x</small>

Add Rule



Warning

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

[Cancel](#)

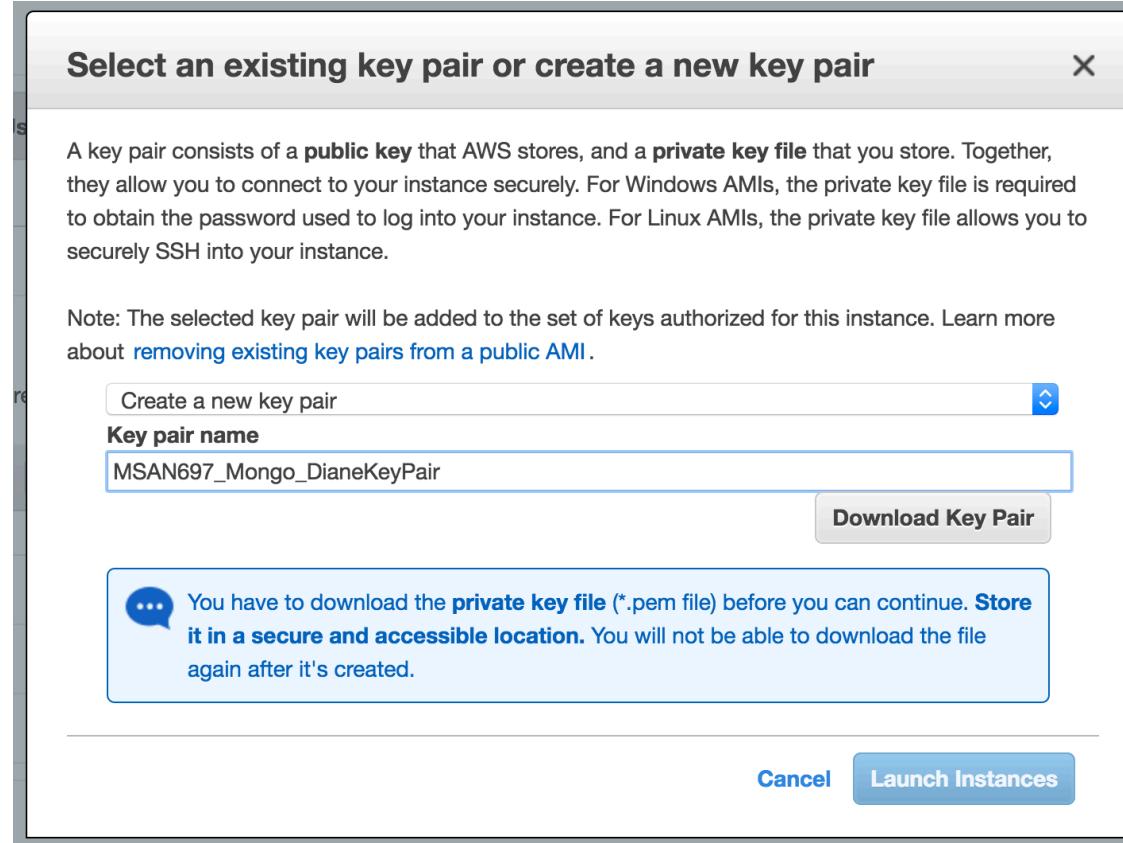
[Previous](#)

Review and Launch



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

MongoDB on AWS



MongoDB on AWS

Place the .pem file somewhere you prefer.

```
sudo vi /etc/yum.repos.d/mongodb-org-2.6.repo
```

Place...

```
[mongodb-org-3.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2013.03/mongodb-
org/3.2/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.2.asc
```

Install Mongo

```
sudo yum install -y mongodb-org
```

<https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-amazon/>

MongoDB on AWS

Install Mongo

```
sudo yum install -y mongodb-org
```

To allow it to be accessed by outside.

comment out `bind_ip` variable at the `/etc/mongodb.conf` file.

Start the service.

```
sudo service mongod start
```

<https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-amazon/>



MongoDB on AWS

Connect To Your Instance

X

I would like to connect with

A standalone SSH client

A Java SSH Client directly from my browser (Java required)

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (MSAN697_Mongo_DianeKeyPair.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 MSAN697_Mongo_DianeKeyPair.pem
```

4. Connect to your instance using its Public DNS:

```
ec2-54-161-217-247.compute-1.amazonaws.com
```

Example:

```
ssh -i "MSAN697_Mongo_DianeKeyPair.pem" ec2-user@ec2-54-161-217-247.compute-1.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

If you need any assistance connecting to your instance, please see our [connection documentation](#).

Close