Week 3: NoSQL (Cont.) and Document Databases

DIANE WOODBRIDGE, PH.D

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?

Relational Database vs. NoSQL

Impedence mismatch

Polygot persistence

Aggregate-oriented database

Key-value database

Document database

Column family database

Graph database

Replication vs sharding

CAP Theorem

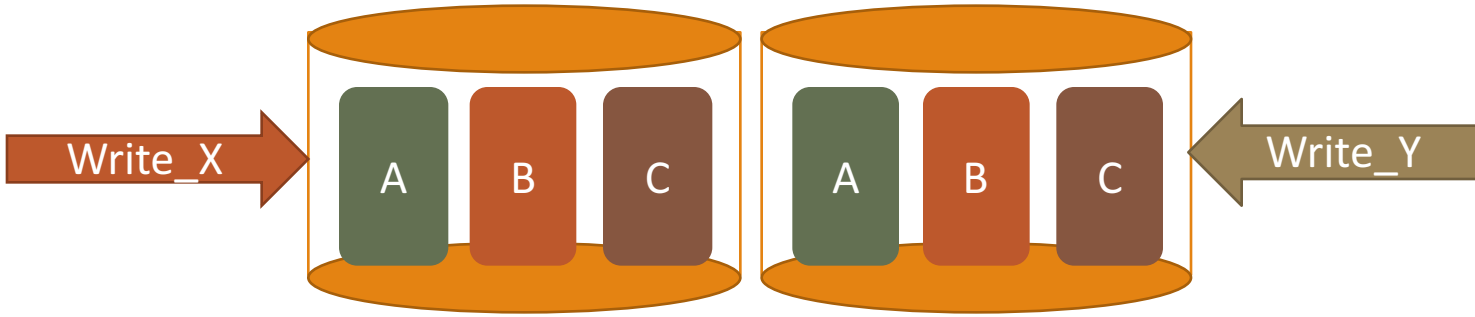
Eventual Consistency

Map-Reduce

Consistency

Update Consistency

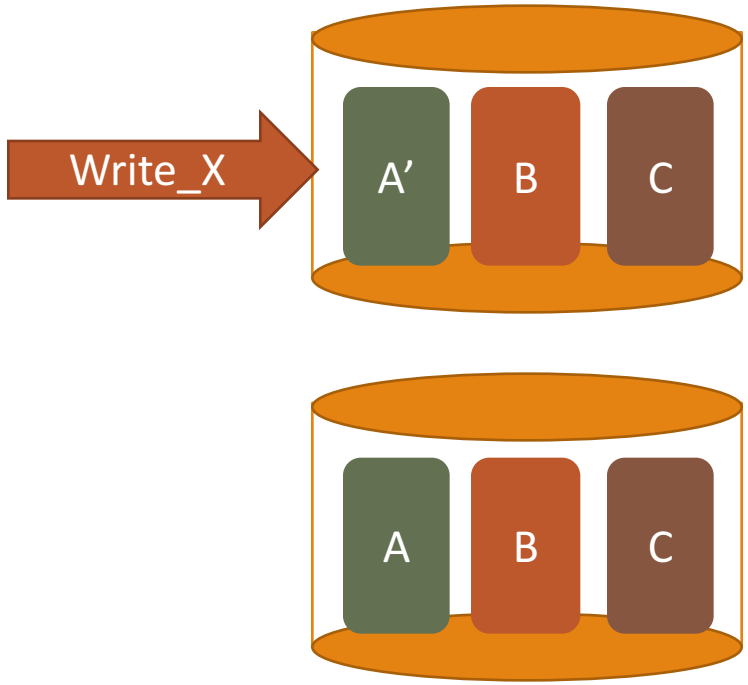
- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but take care of it. (Responsive)
    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

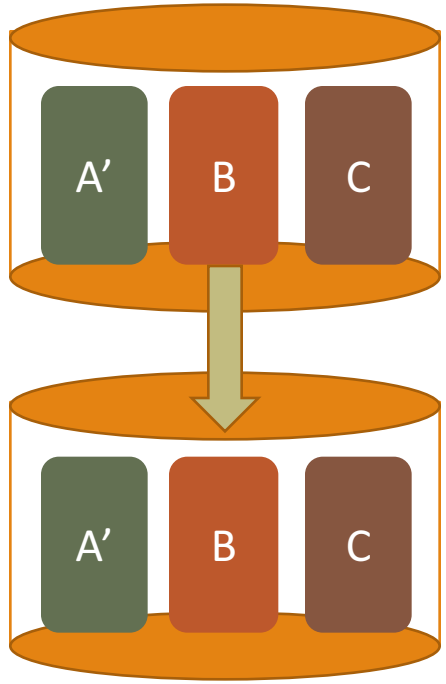
- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
    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

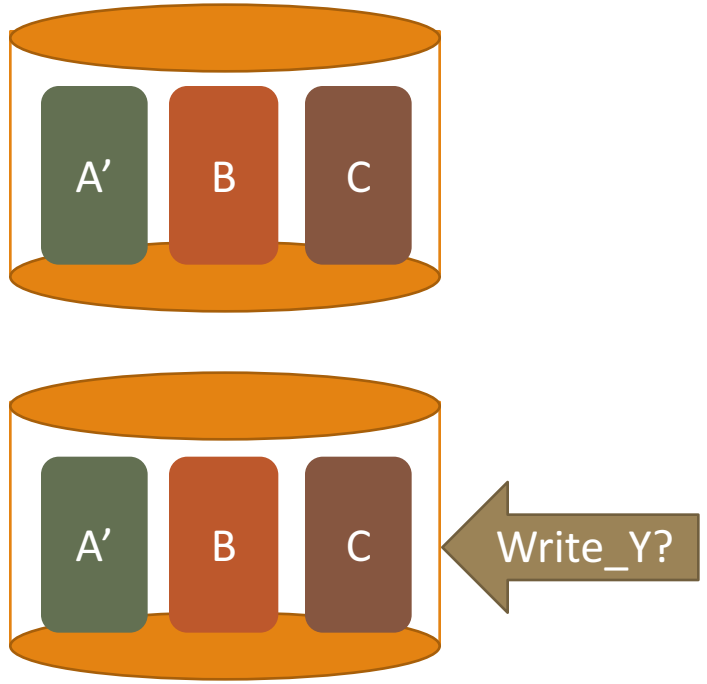
- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
    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.

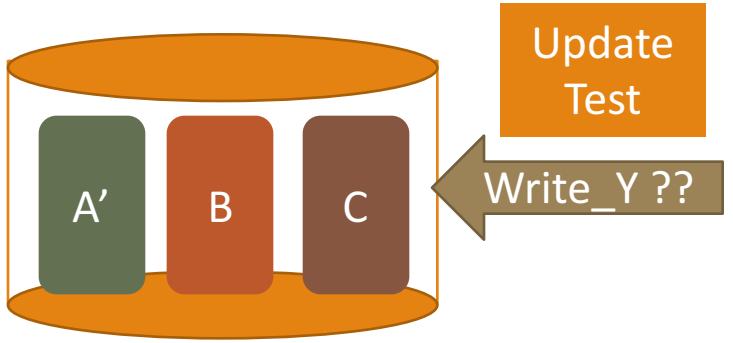


Consistency

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.

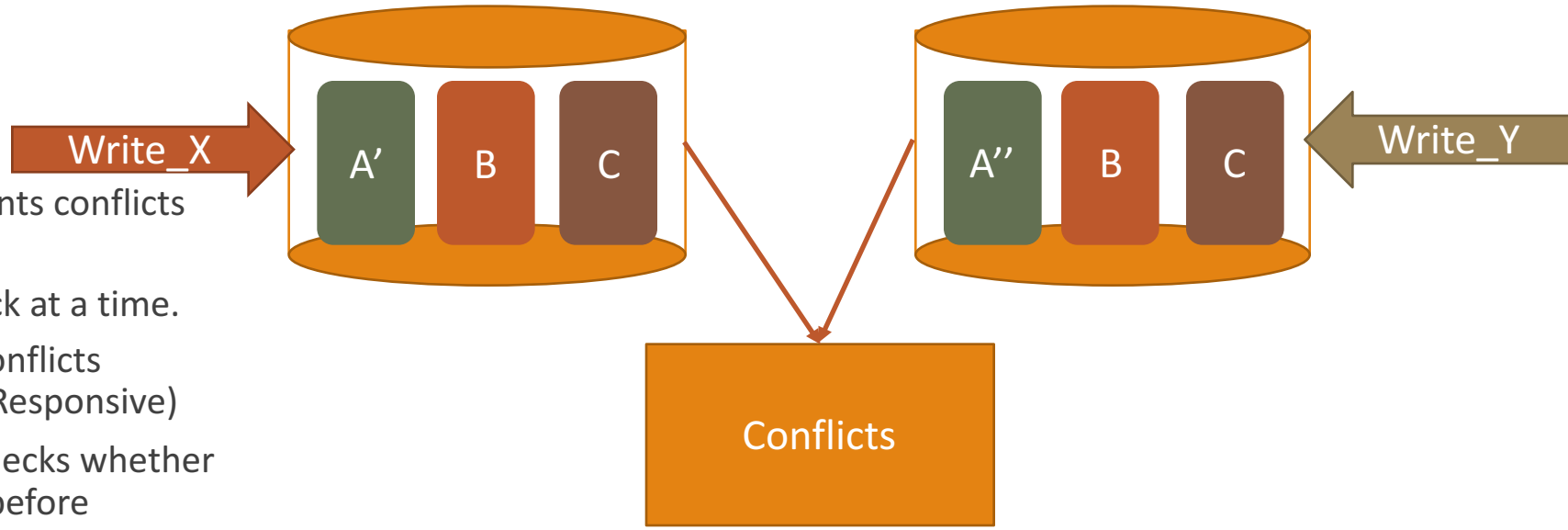
Requires sequential consistency.



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

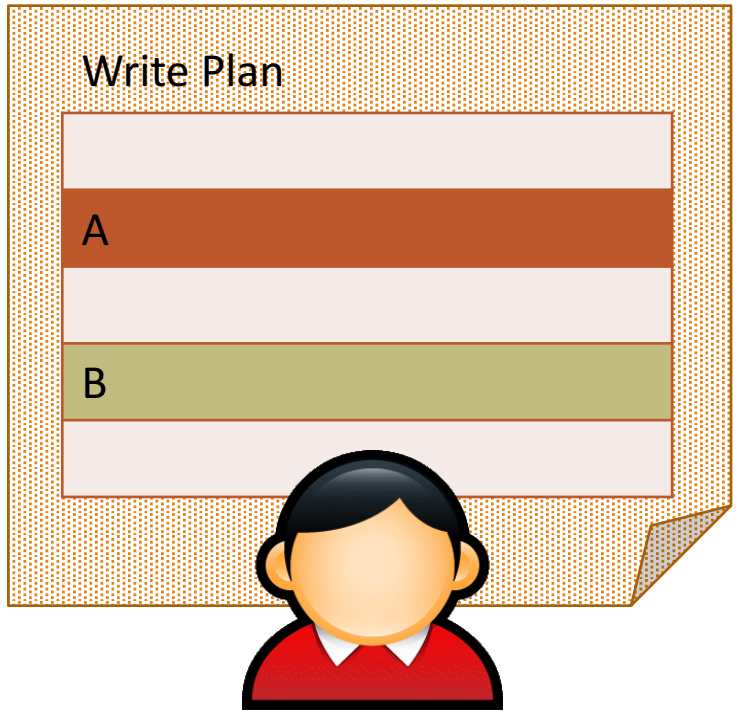
- Read-write conflict (inconsistent read).
- Replication inconsistency.
- Session inconsistency.

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.

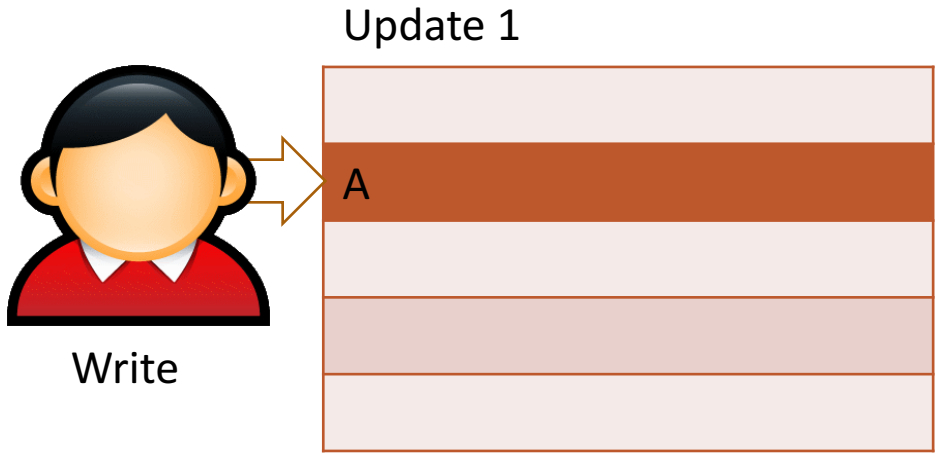


Consistency

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.

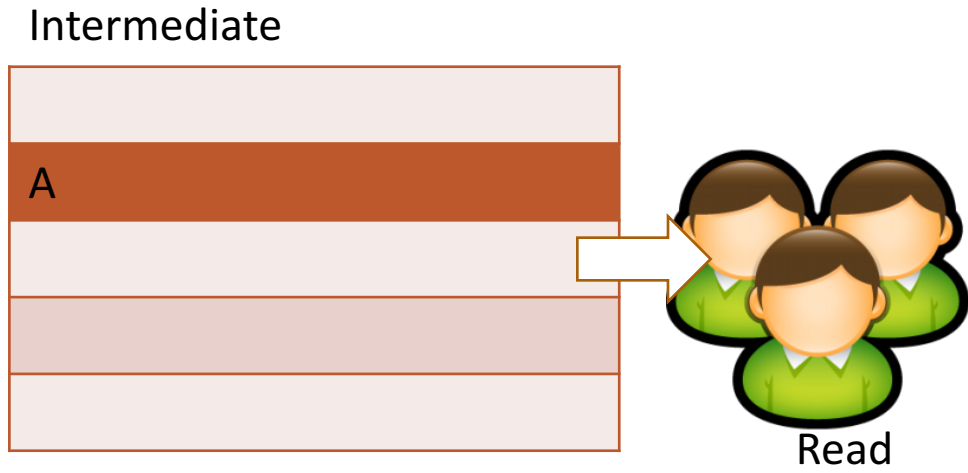


Consistency

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.

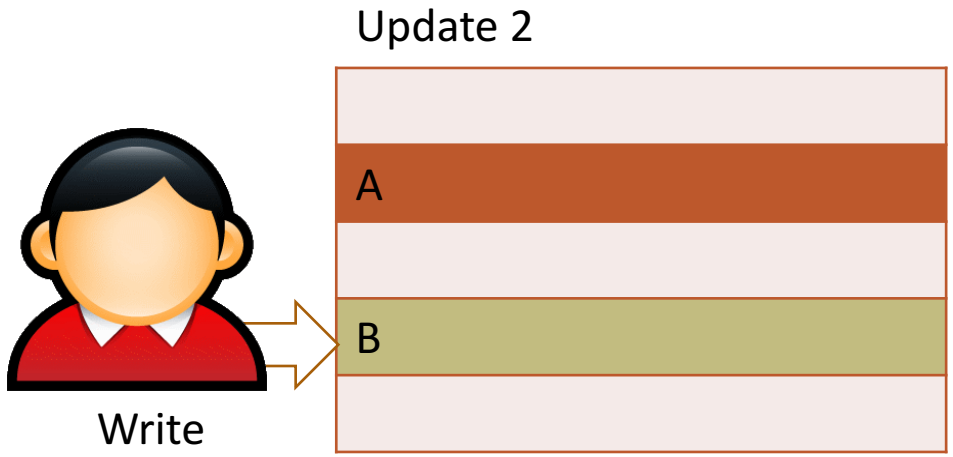


Consistency

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.

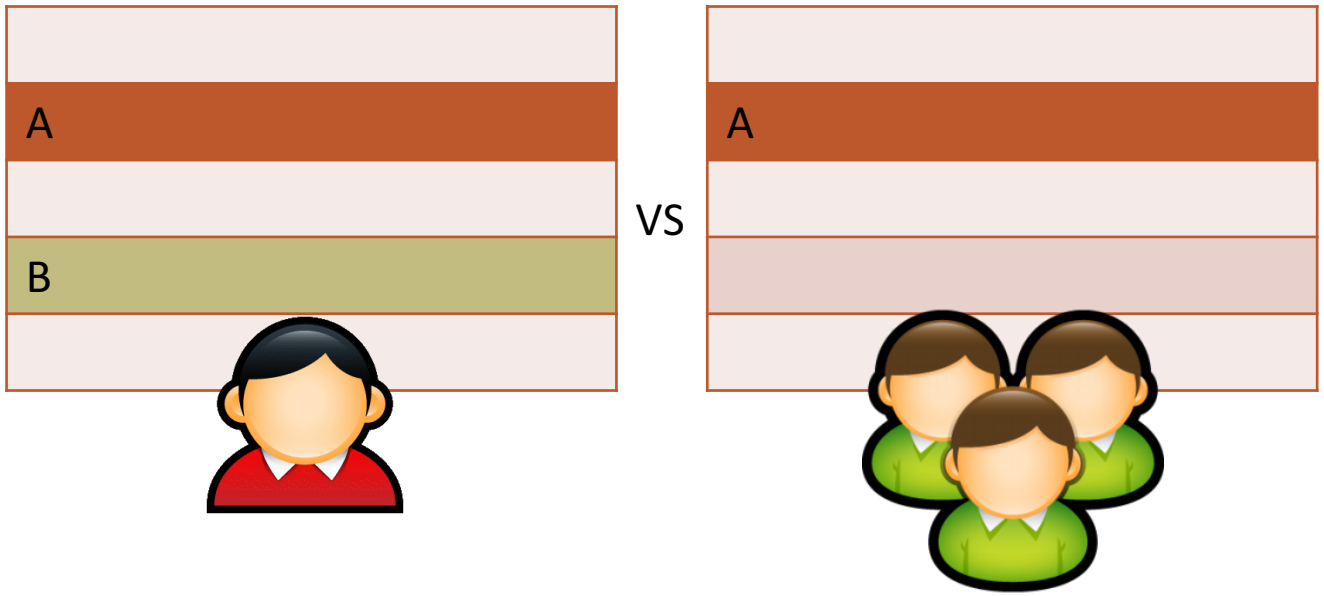


Consistency

Read Consistency

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

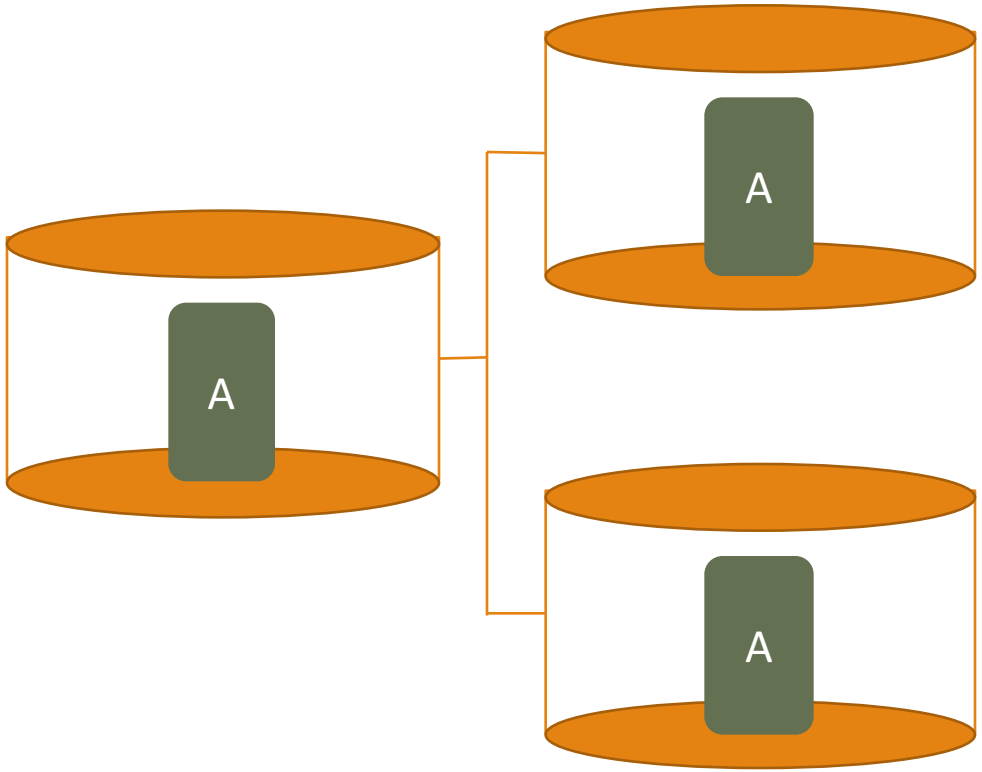
Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

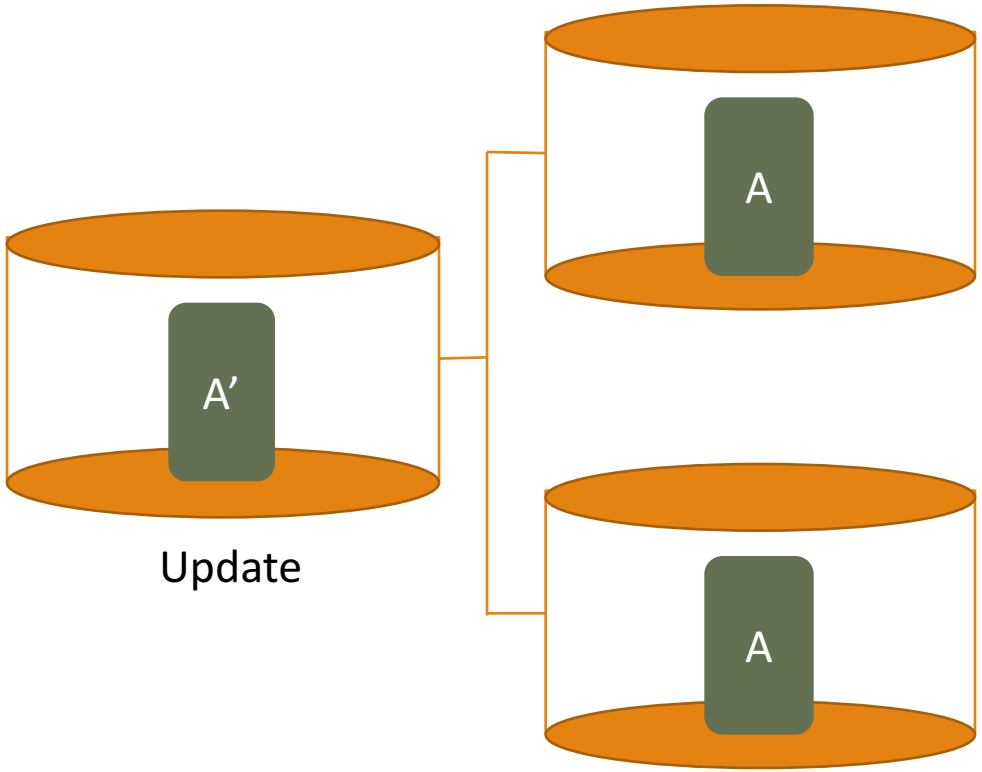
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

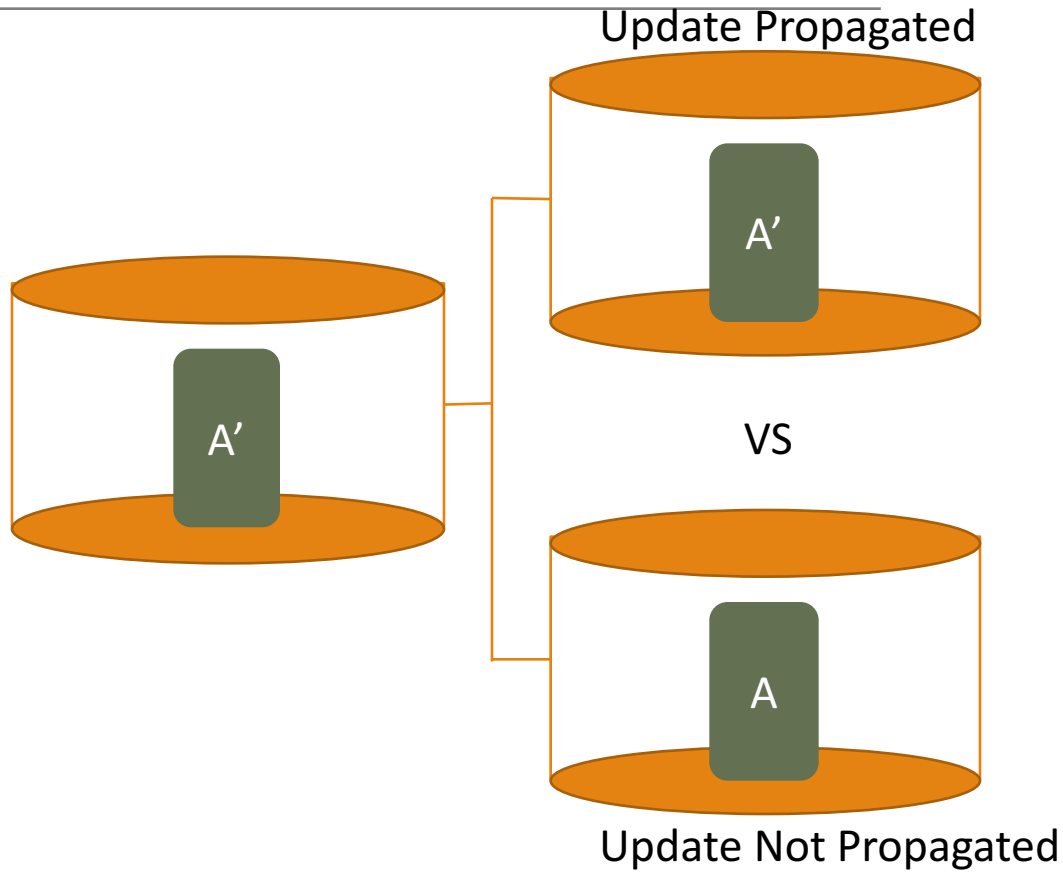
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

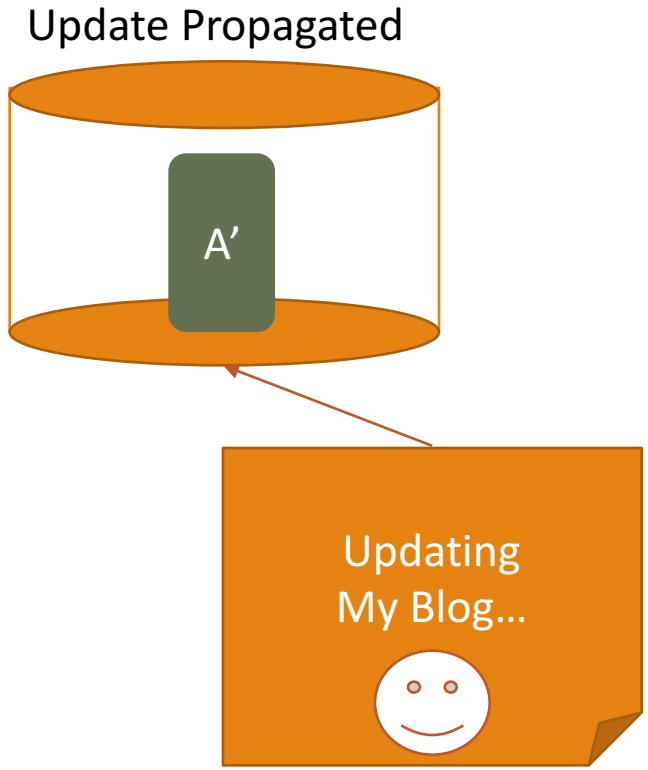
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

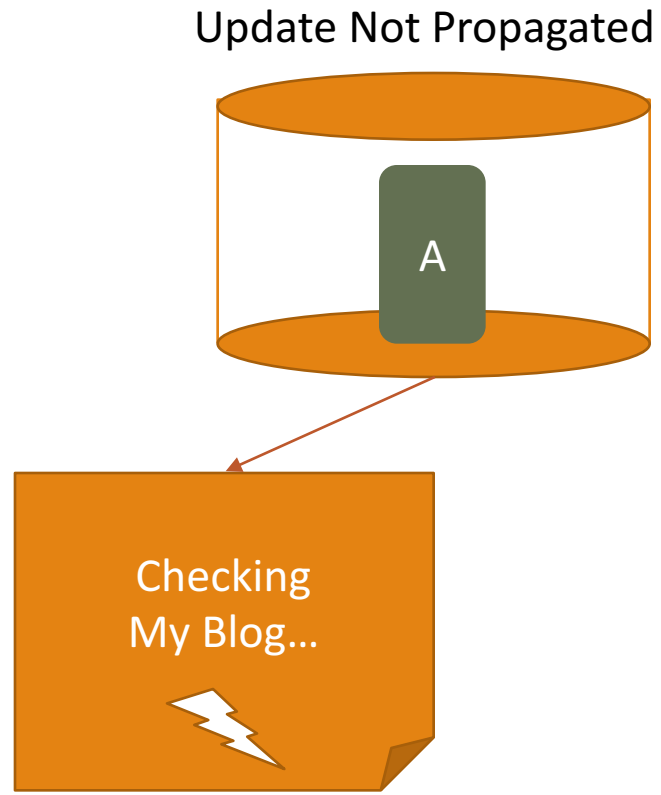
- Session Consistency (Read-your-writes Consistency)
  - Sticky Session (Session affinity): A session is tied to one node and keep session consistency on the node.
  - Version Stamp : Make sure that every interaction with a node returns data with the latest version stamp seen by the session.



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.

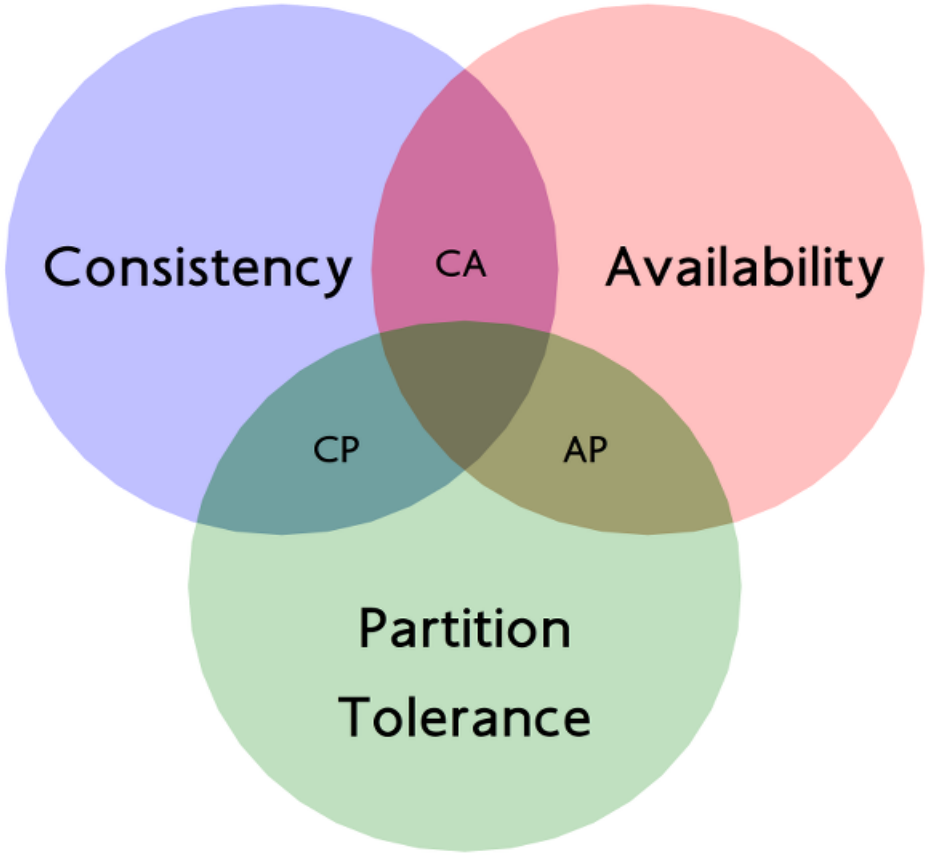


Consistency

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

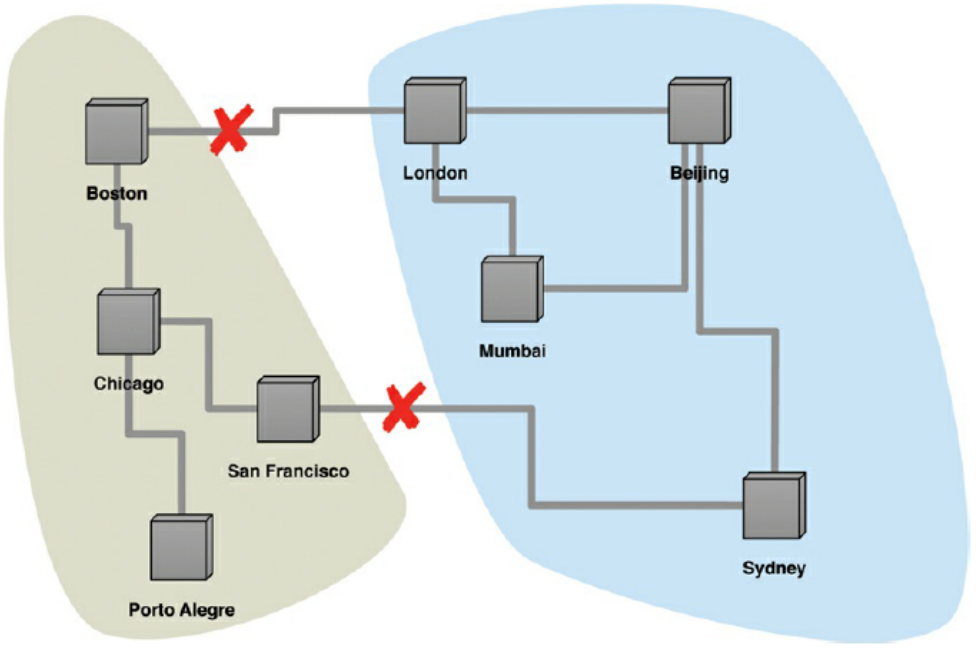
Consistency

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 .

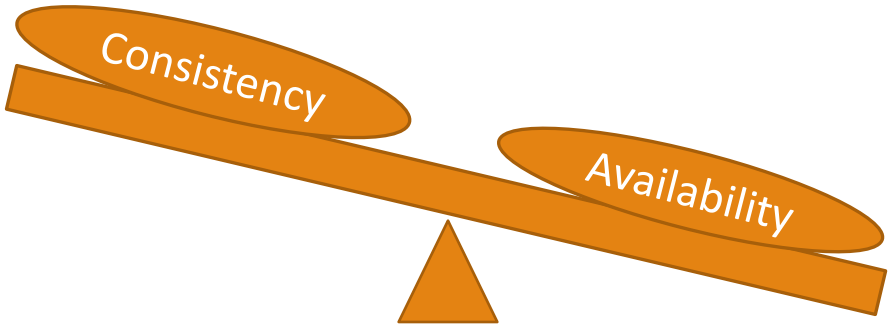


http://blingtechs.blogspot.com/2016/02/cap-theorem.html

Consistency

Relaxing Consistency/Availability

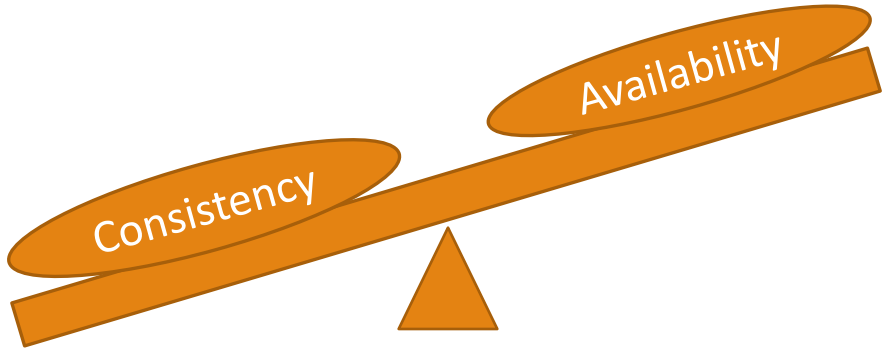
- CAP Theorem and Distributed Database
  - Requirement - Partition-Tolerance *
  - Availability or Consistency??
    - Availability – Shopping
    - Consistency – Stock Market



Consistency

Relaxing Consistency/Availability

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



Consistency

Relaxing Consistency/Availability

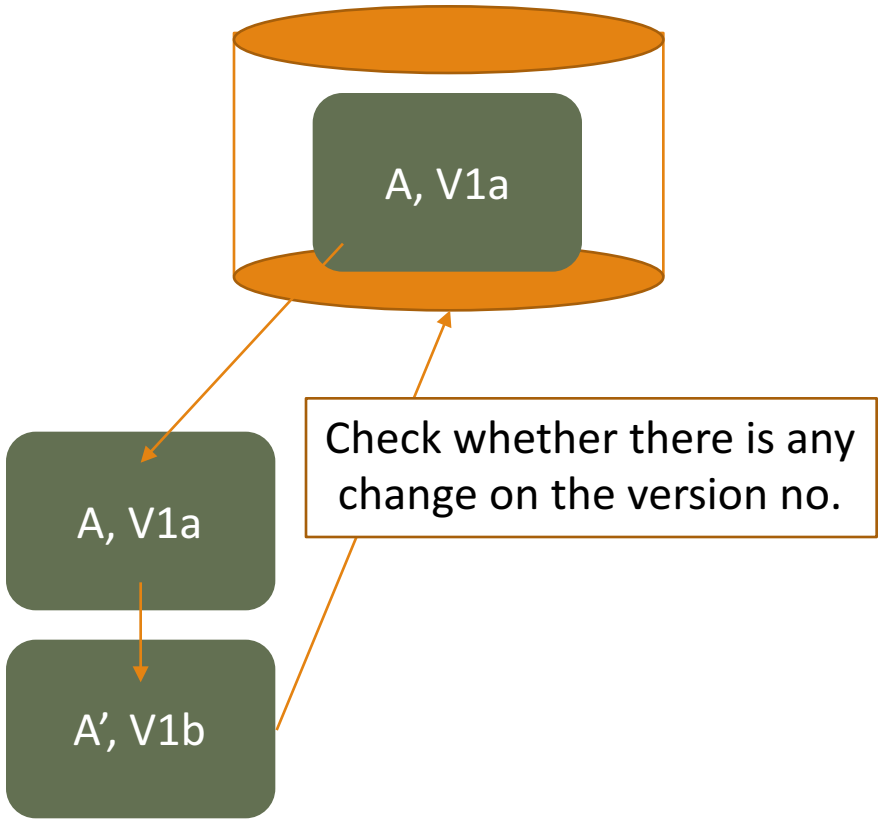
- CAP Theorem and Distributed Database
  - Requirement - Partition-Tolerance *
  - Availability or Consistency??
    - Availability – Shopping
    - Consistency – Stock Market

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.

Walmart Order

OrderID: 1
Customer : Diane
Timestamp : 2016-08-15 05:04:32 PST
Items:
{ProductName : San Francisco Giants Hat M,
{Qty: 2, UnitPrice : 10, Price : 20}},
{ProductName : San Francisco Giants Hat S,
{ Qty: 3, UnitPrice : 8, Price : 24}}
Shipping: Corte Madera

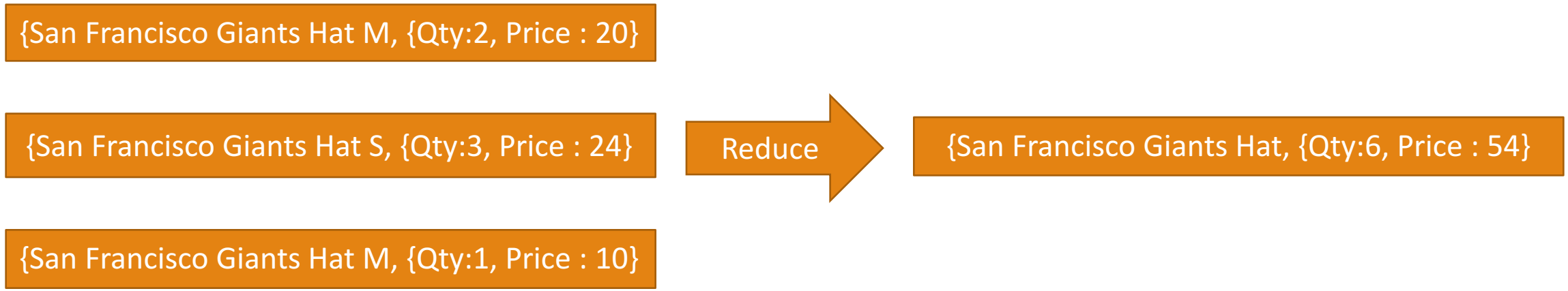
{San Francisco Giants Hat M, {Qty:2, Price : 20}

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

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.



MongoDB

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

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

```
{
_id :2345
firsname : 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 Oct 2016	Rank Sep 2016	Rank Oct 2015	DBMS	Database Model	Score Oct 2016	Score Sep 2016	Score 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

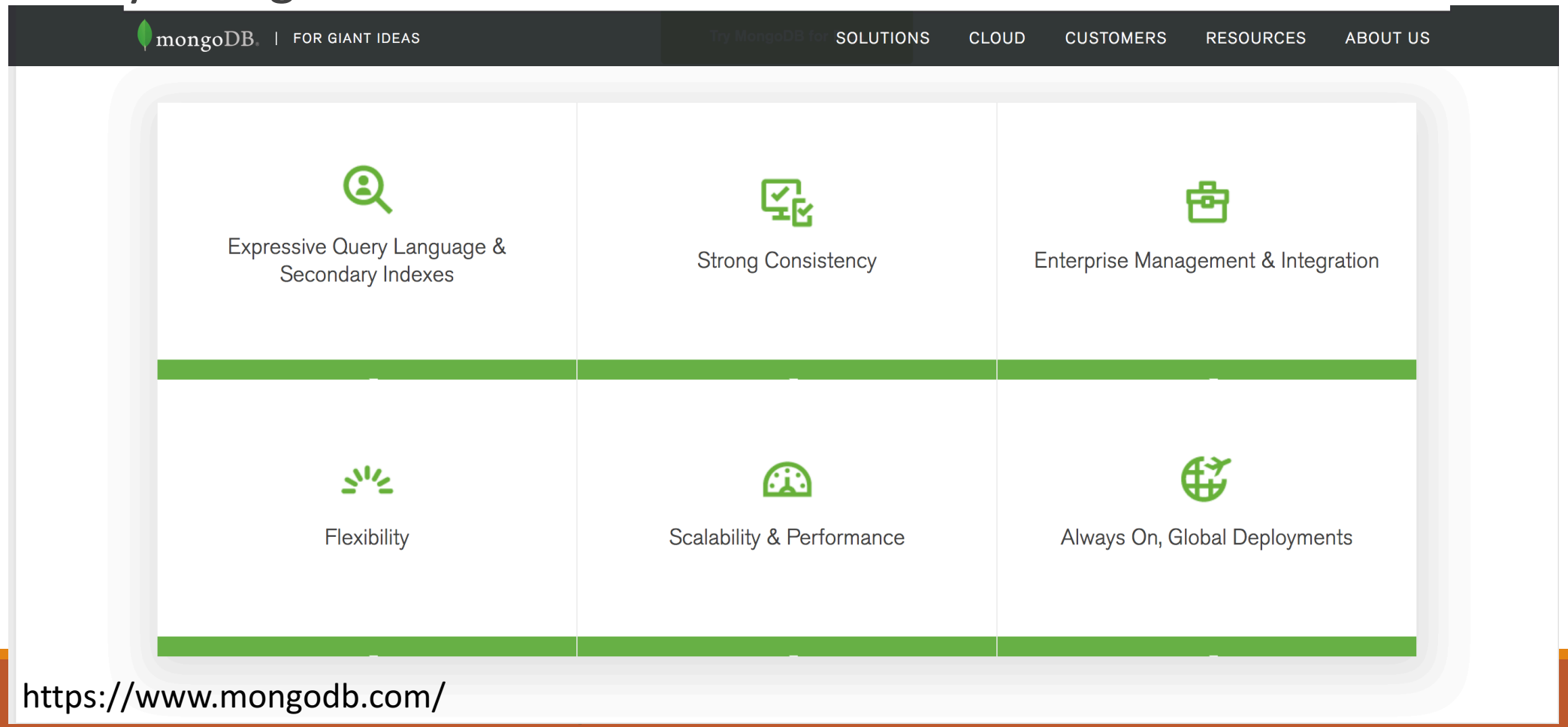
- Documents
- Collections
- Databases

RDBMS	MongoDB
database	database
table	collection
row	document
column	field

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

MongoDB

Why MongoDB?



https://www.mongodb.com/

MongoDB 

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.

MongoDB

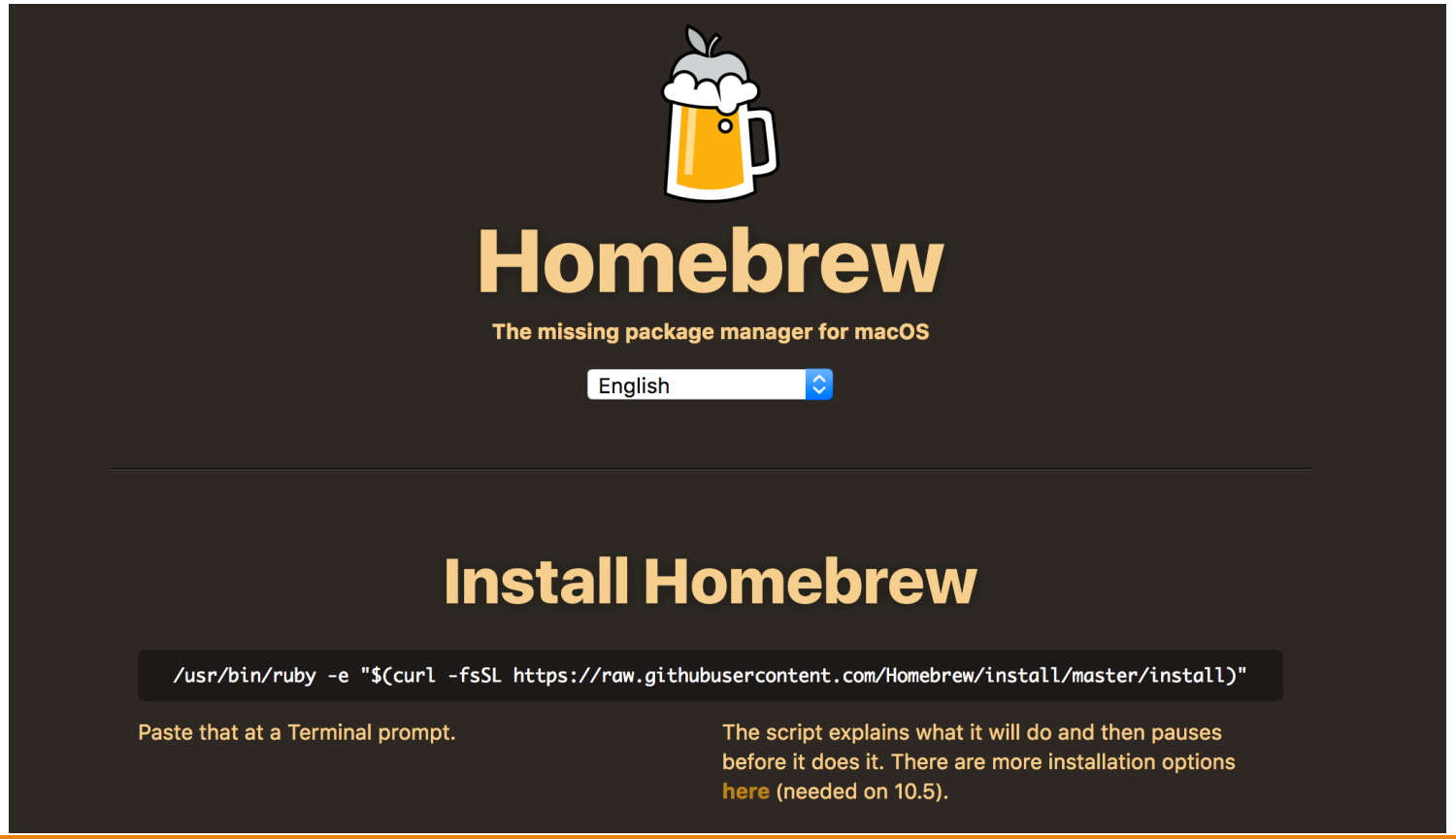
Why MongoDB?

- 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/

MongoDB

Not installed Homebrew??



http://brew.sh/

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
csvtomd             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     pcap_dnsproxy
```

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

MongoDB

Create **/data/db**

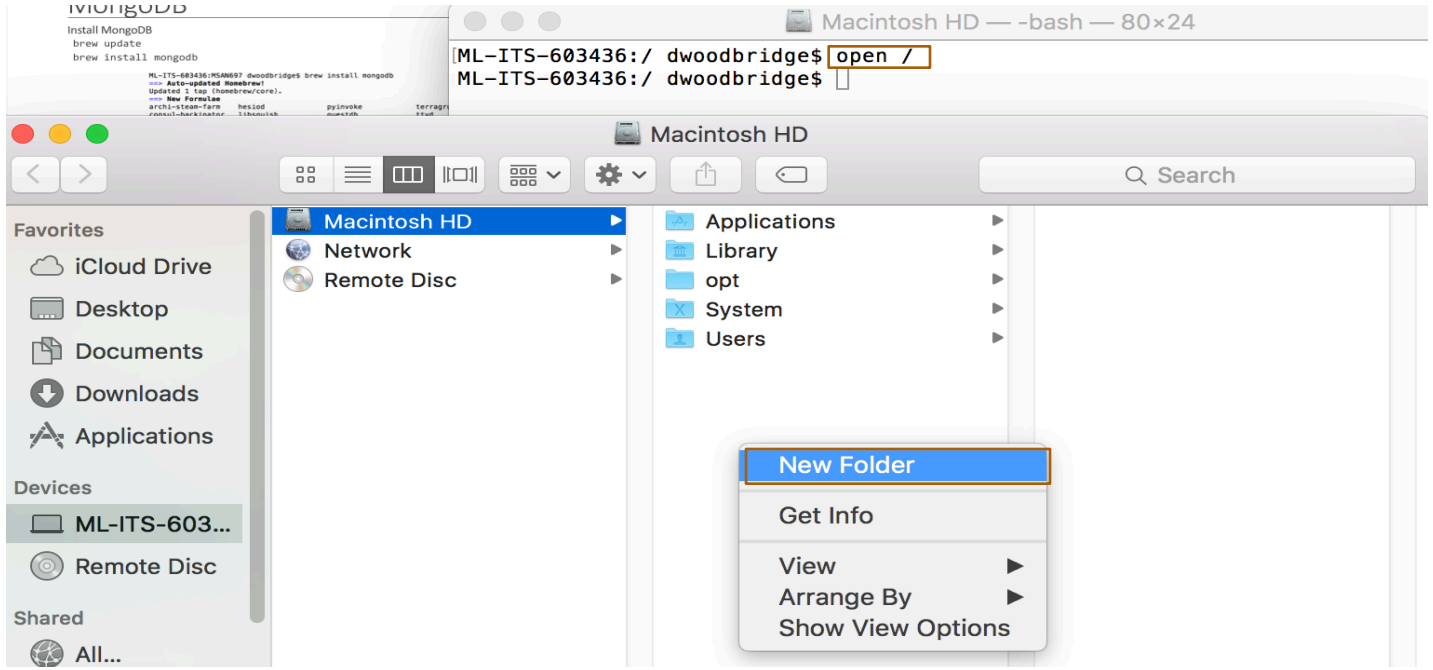
Start the server : Run mongod executable.

```
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

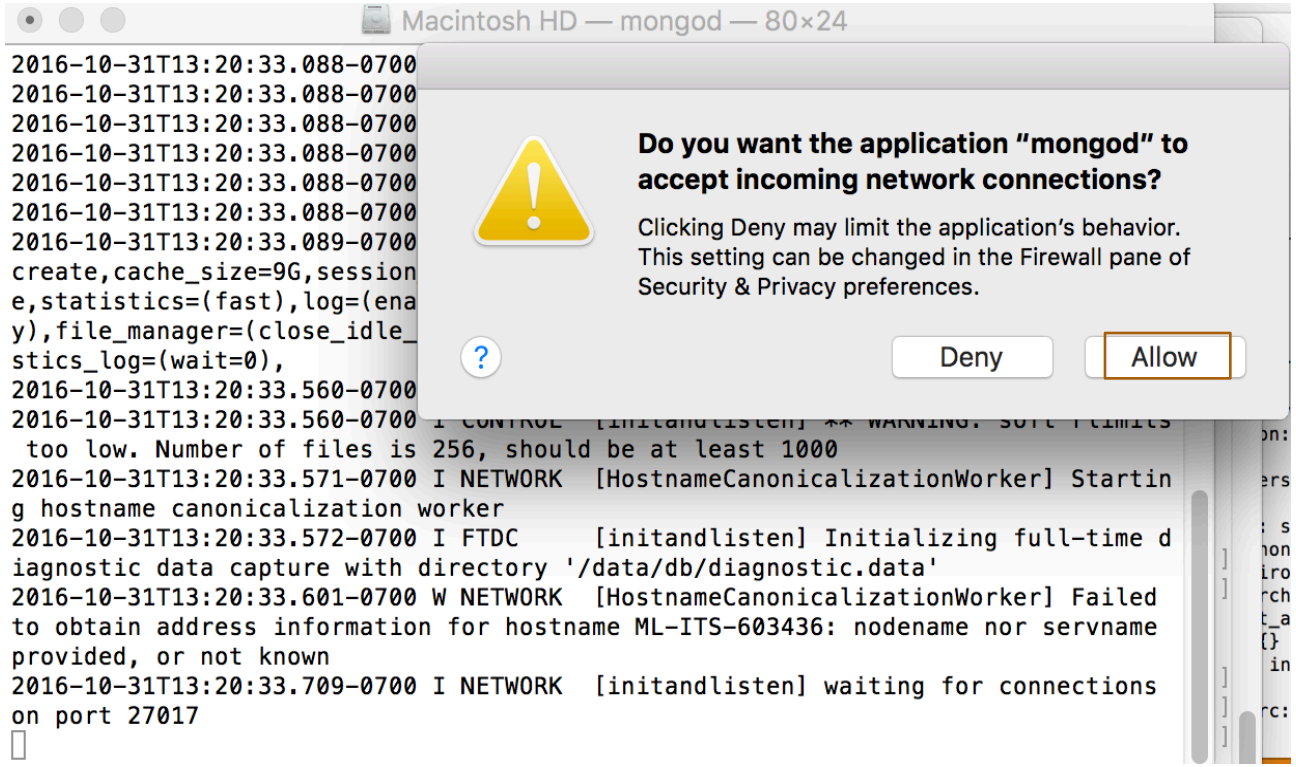
Create **/data/db**



MongoDB

Start the server : Call **mongod**

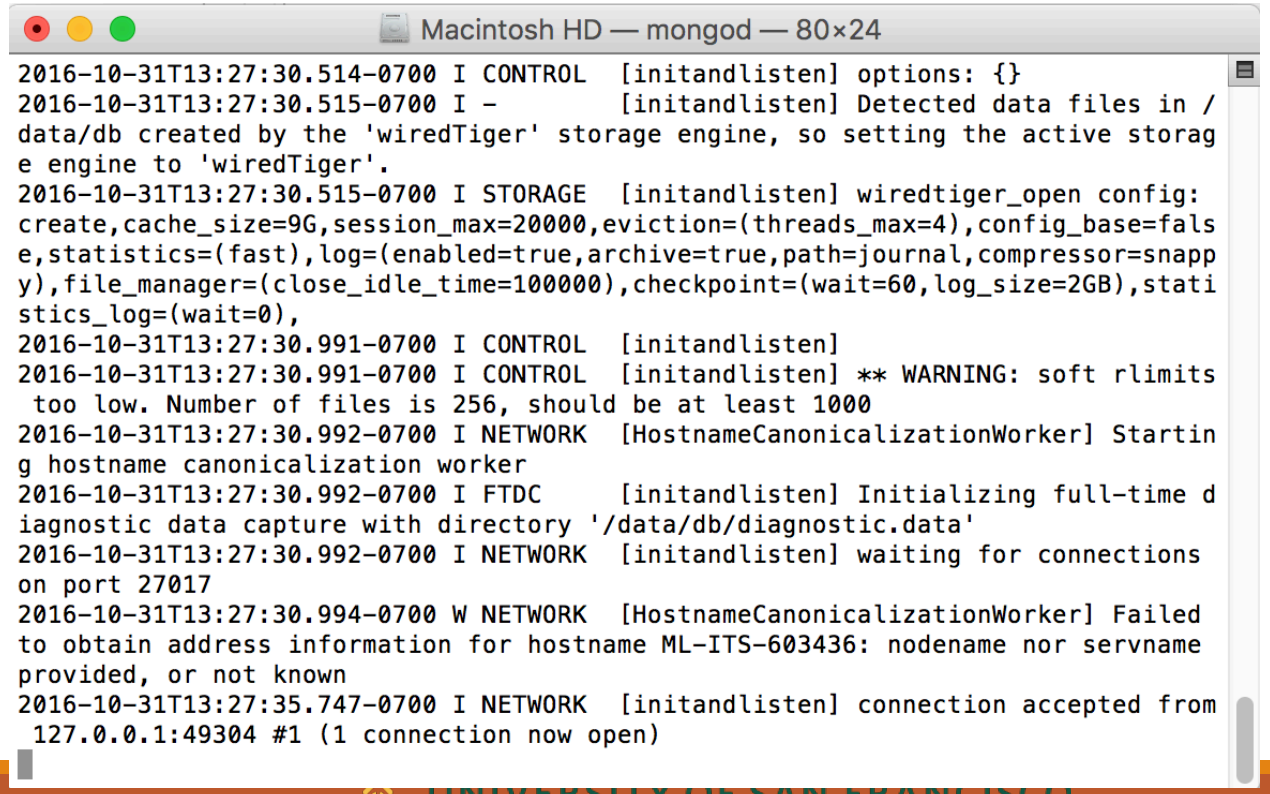
Stop mongod : "`Ctrl-C`"



MongoDB

Start the server : Call **mongod**

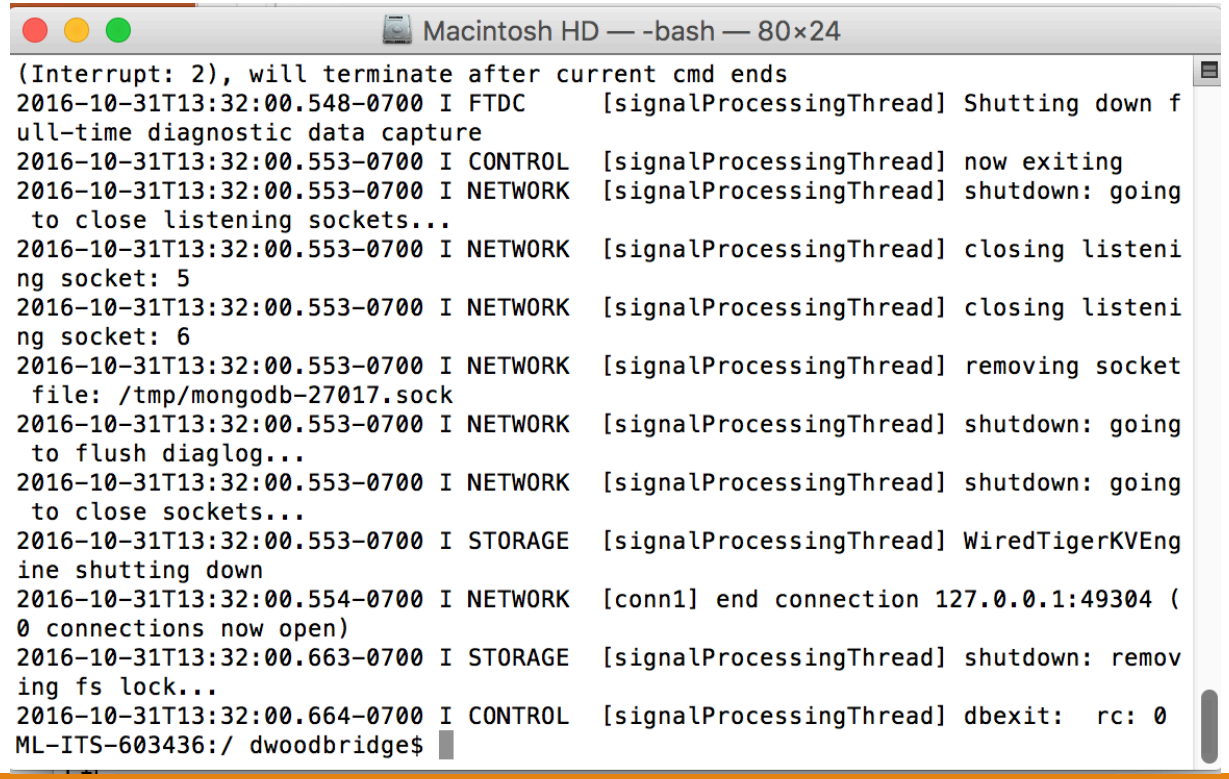
Stop mongod : "`Ctrl-C`"



MongoDB

Start the server : Call **mongod**

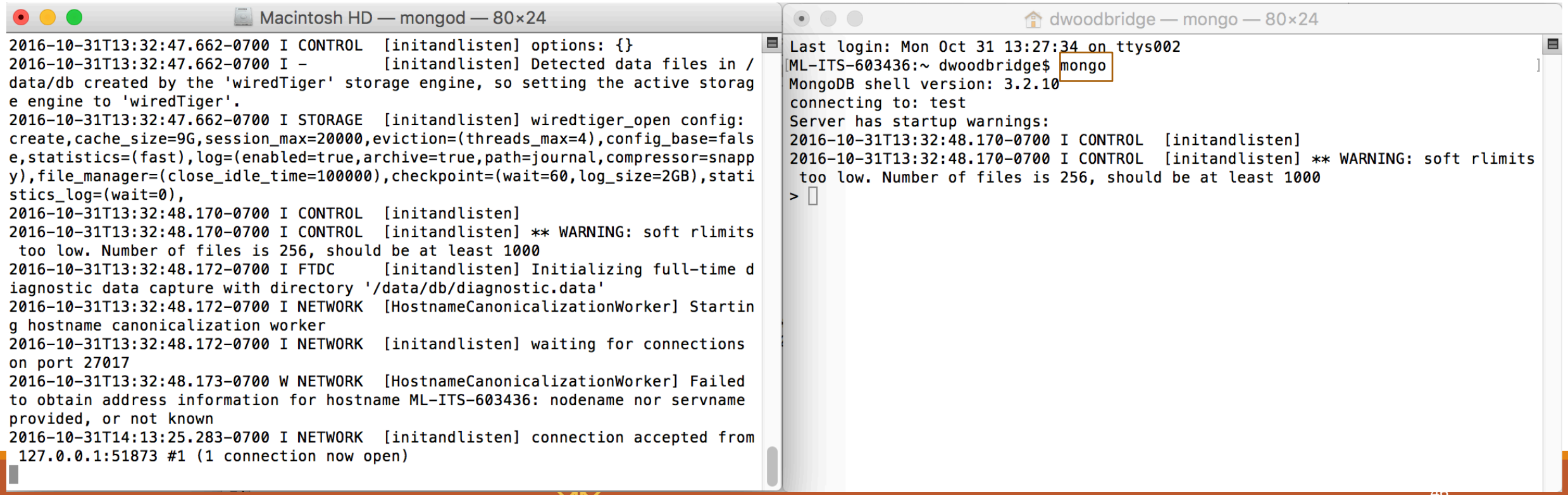
Stop mongod : “Ctrl-C”



MongoDB

Running MongoDB Shell

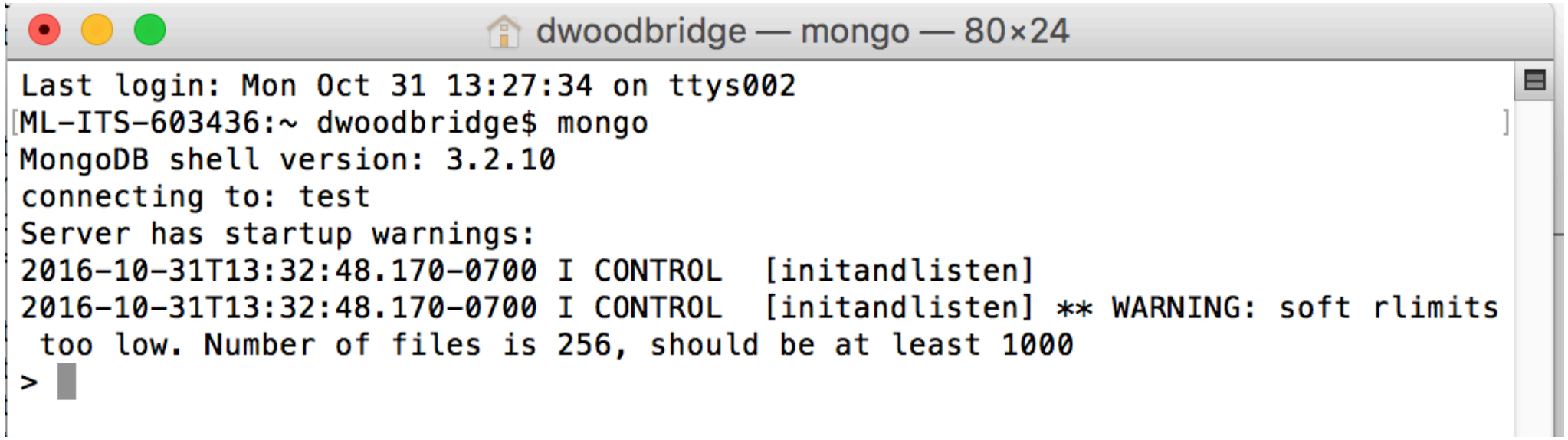
- While your server is on.
- Run **mongo** executable.



MongoDB

Running MongoDB Shell

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



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")
ISODate("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]}
{ "favorites" : [ 7, 11 ] }
> diane = {"favorites" : [7,11, "Puppies"]}
{ "favorites" : [ 7, 11, "Puppies" ] }
> diane = {"favorites" : [7,11, "Puppies",["Yougurt","Coffee"]]}
{
        "favorites" : [
                7,
                11,
                "Puppies",
                [
                        "Yougurt",
                        "Coffee"
                ]
        ]
}
```

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" : "
{
        "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
```

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
                },
                {
                        "date" : ISODate("2013-06-05T00:00:00Z"),
```

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"), "na
", "city" : "San Francisco", "state" : "CA" }, "noK
tent" : "Please cook mac and cheese" }, { "name" :
ntent" : "Want to make sure you're doing fine in yo
going?" } ] }
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "na
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
}
```

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
`$inc`	Increments the value of the field by the specified amount.
`$mul`	Multiplies the value of the field by the specified amount.
`$rename`	Renames a field.
`$setOnInsert`	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.
`$set`	Sets the value of a field in a document.
`$unset`	Removes the specified field from a document.
`$min`	Only updates the field if the specified value is less than the existing field value.
`$max`	Only updates the field if the specified value is greater than the existing field value.
`$currentDate`	Sets the value of a field to current date, either as a Date or a Timestamp.

https://docs.mongodb.com/manual/reference/operator/update/

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", "addres
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" :
```

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

https://docs.mongodb.com/manual/reference/operator/update/

MongoDB

Array modifier

- Modifier

Modifiers

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

https://docs.mongodb.com/manual/reference/operator/update/

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."
                }
```

https://docs.mongodb.com/manual/reference/operator/update/

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?" }
```

https://docs.mongodb.com/manual/reference/operator/update/

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.

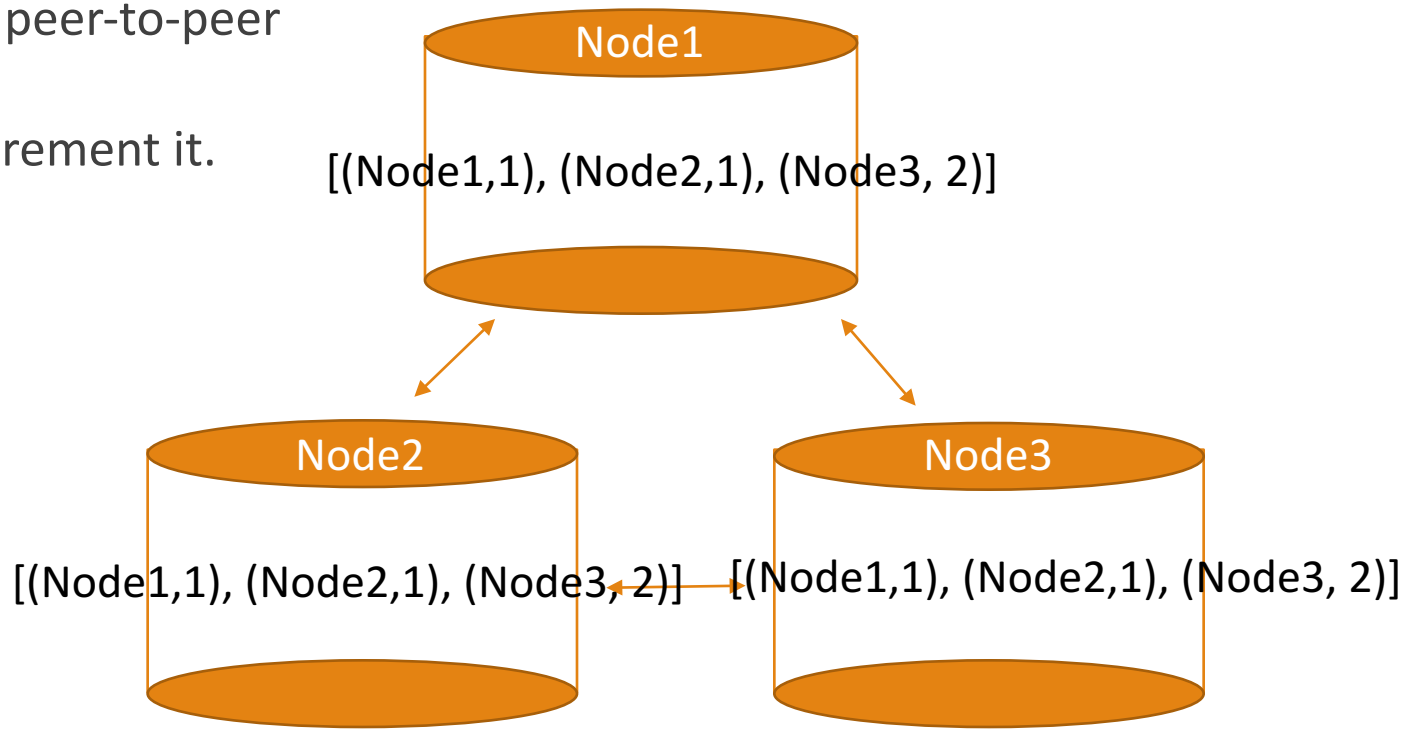
MongoDB. *MongoDB Documentation*, https://docs.mongodb.com/, 2016.

Further Readings

Version Stamps

Vector Stamp

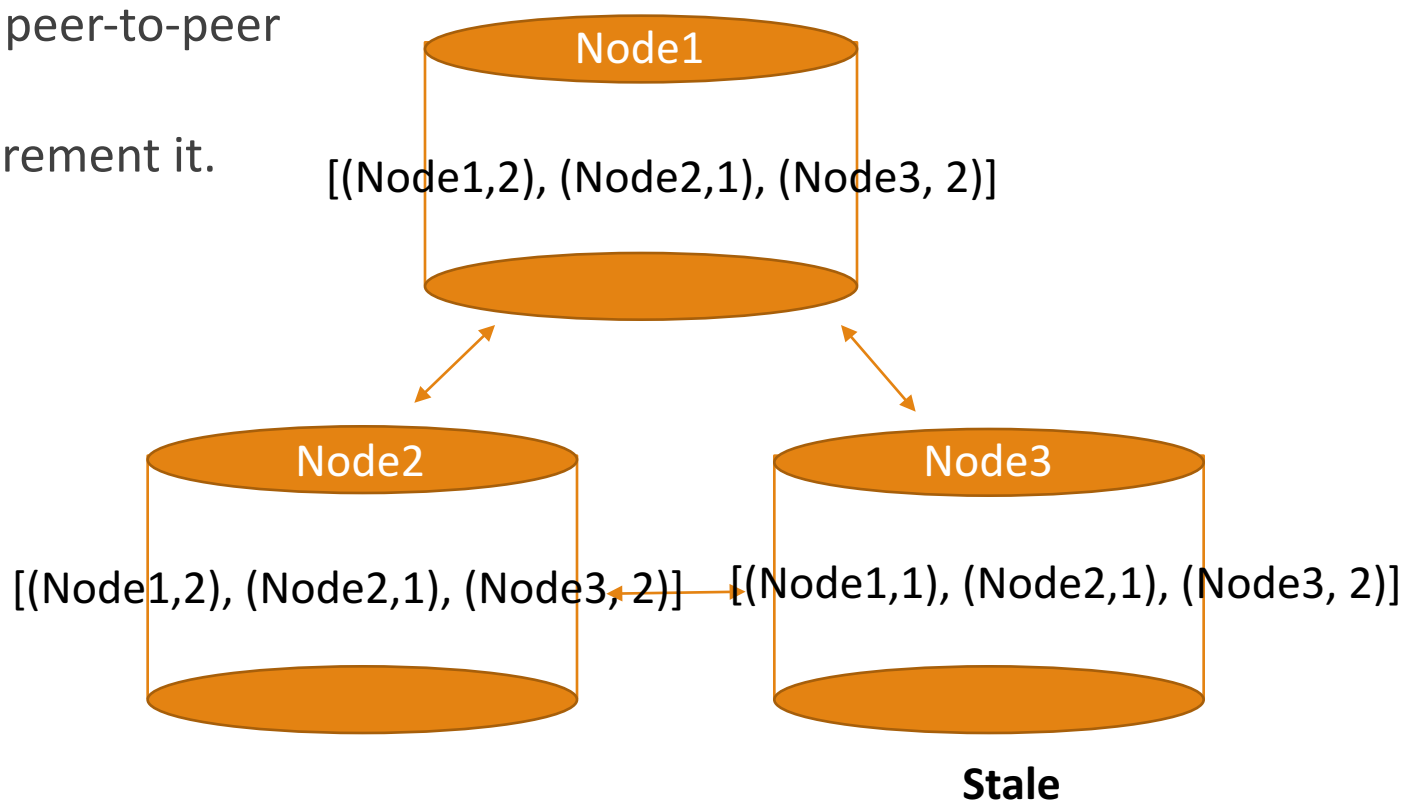
- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
  - Ex. [(Node1, 1), (Node2, 2) , (Node3, 6)]



Version Stamps

Vector Stamp

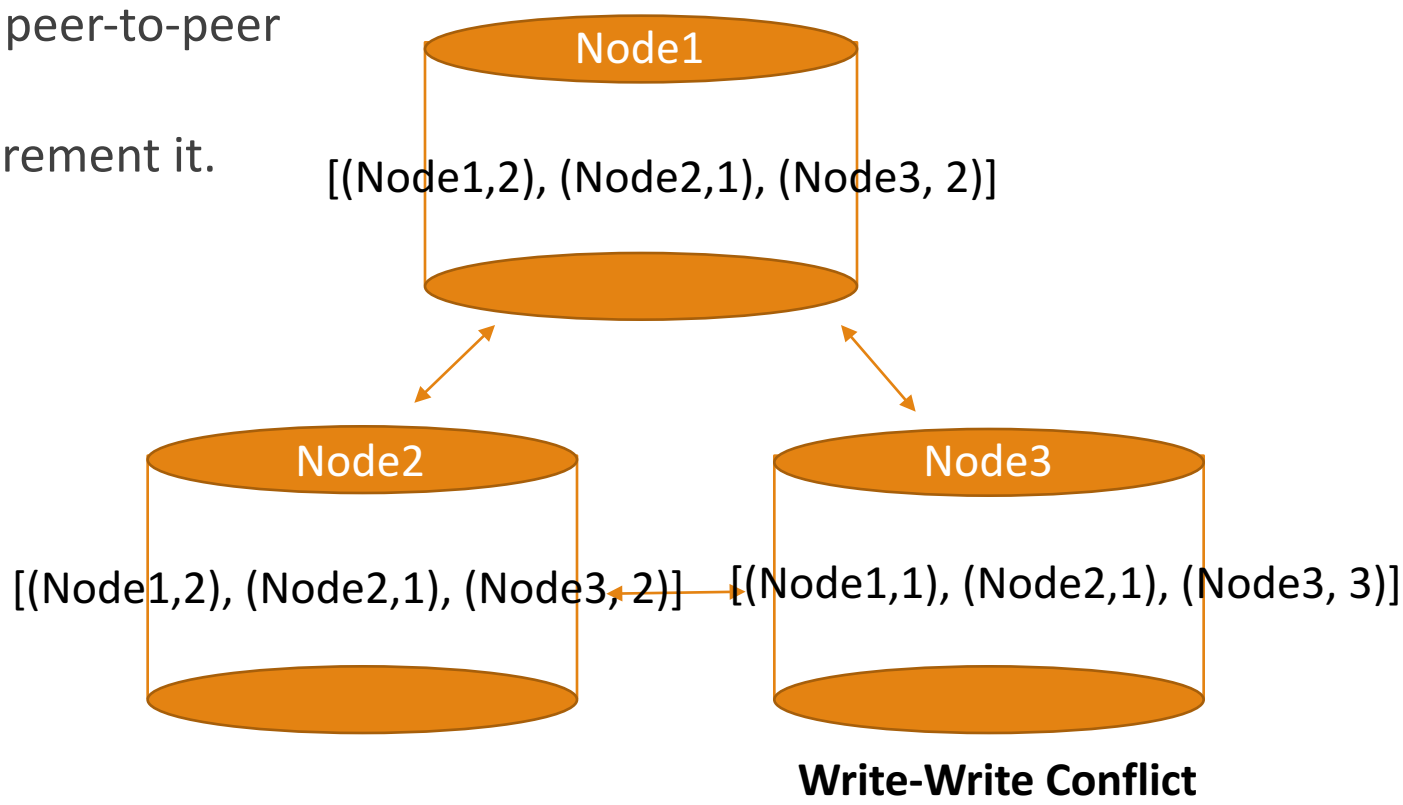
- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
  - Ex. [(Node1, 1), (Node2, 2) , (Node3, 6)]



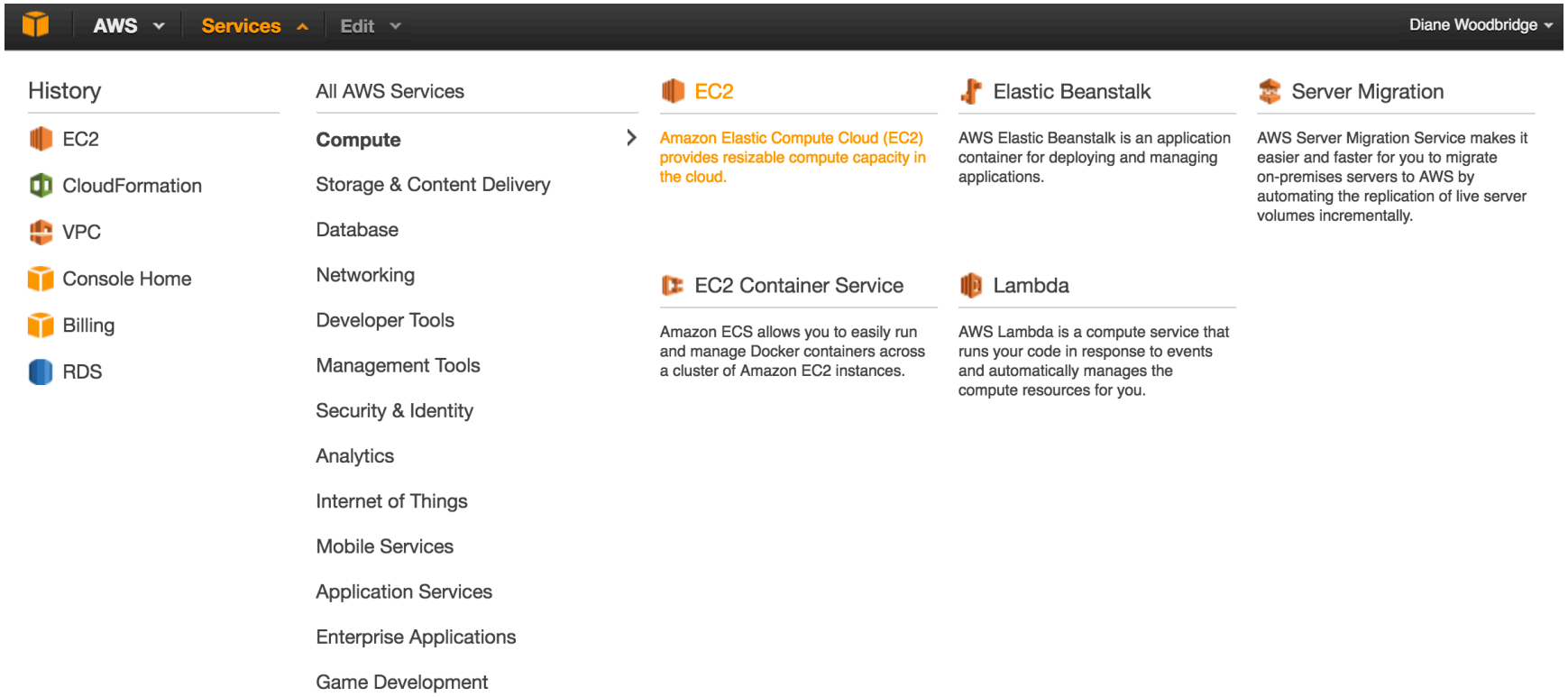
Version Stamps

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. [(Node1, 1), (Node2, 2) , (Node3, 6)]

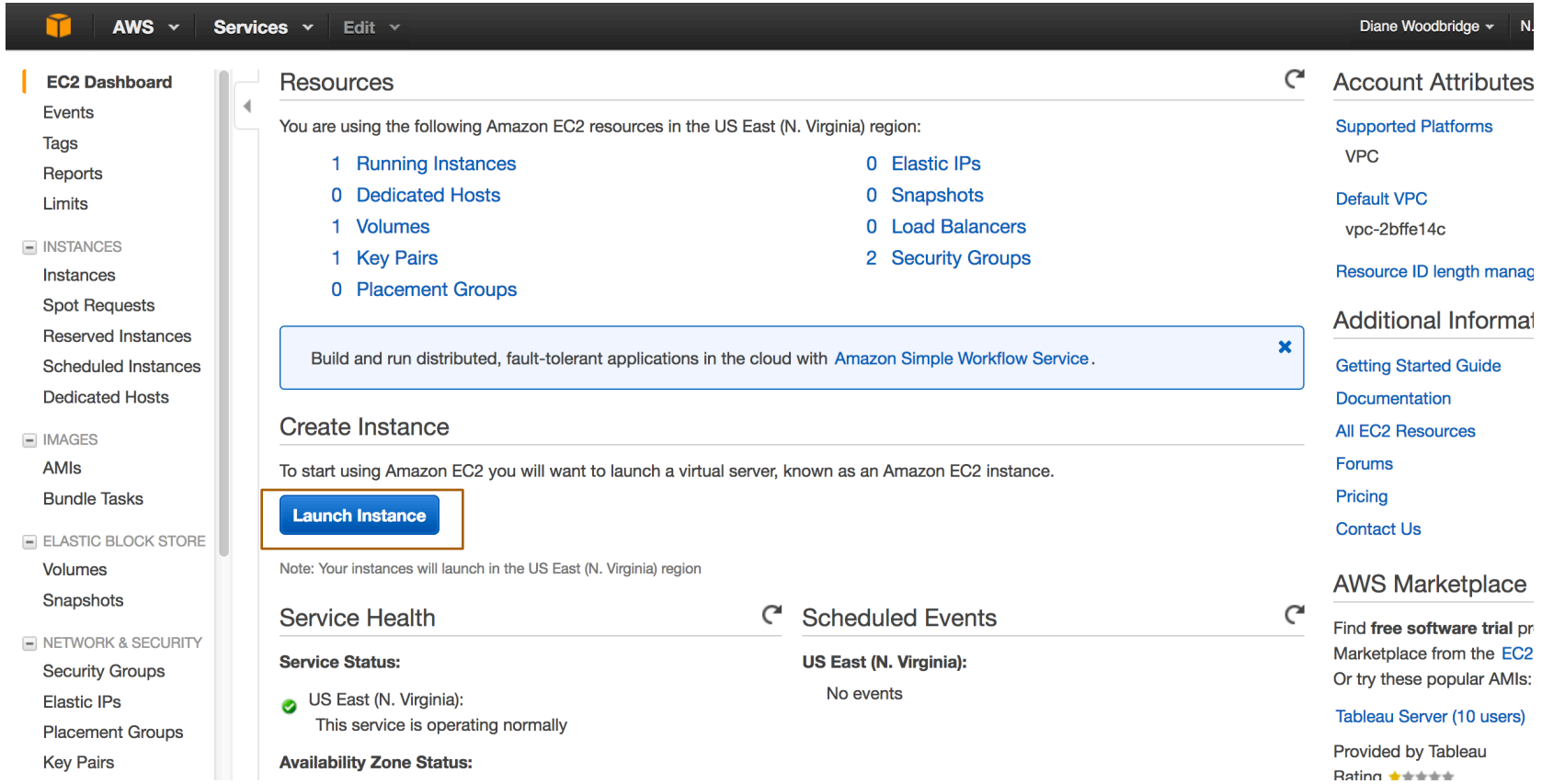


MongoDB on AWS

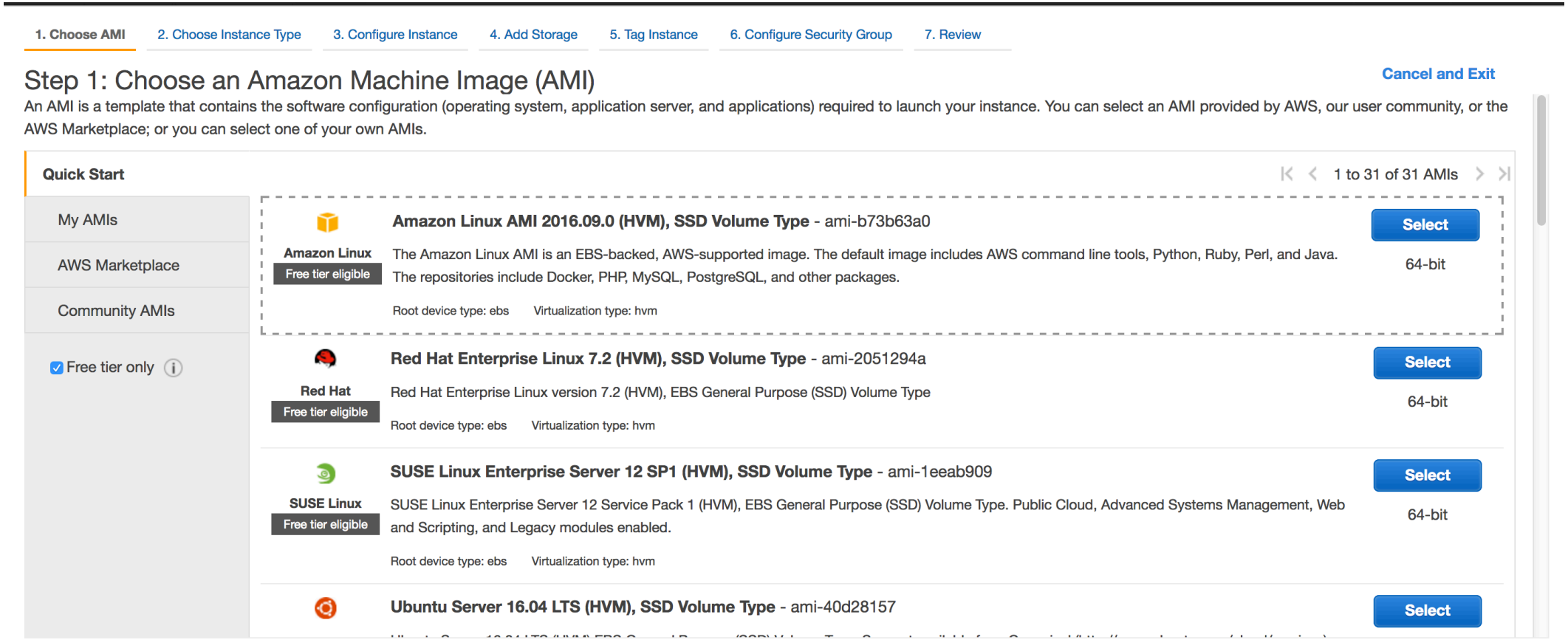


https://aws.amazon.com/blogs/aws/mongodb-on-the-aws-cloud-new-quick-start-reference-deployment/
https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-amazon/

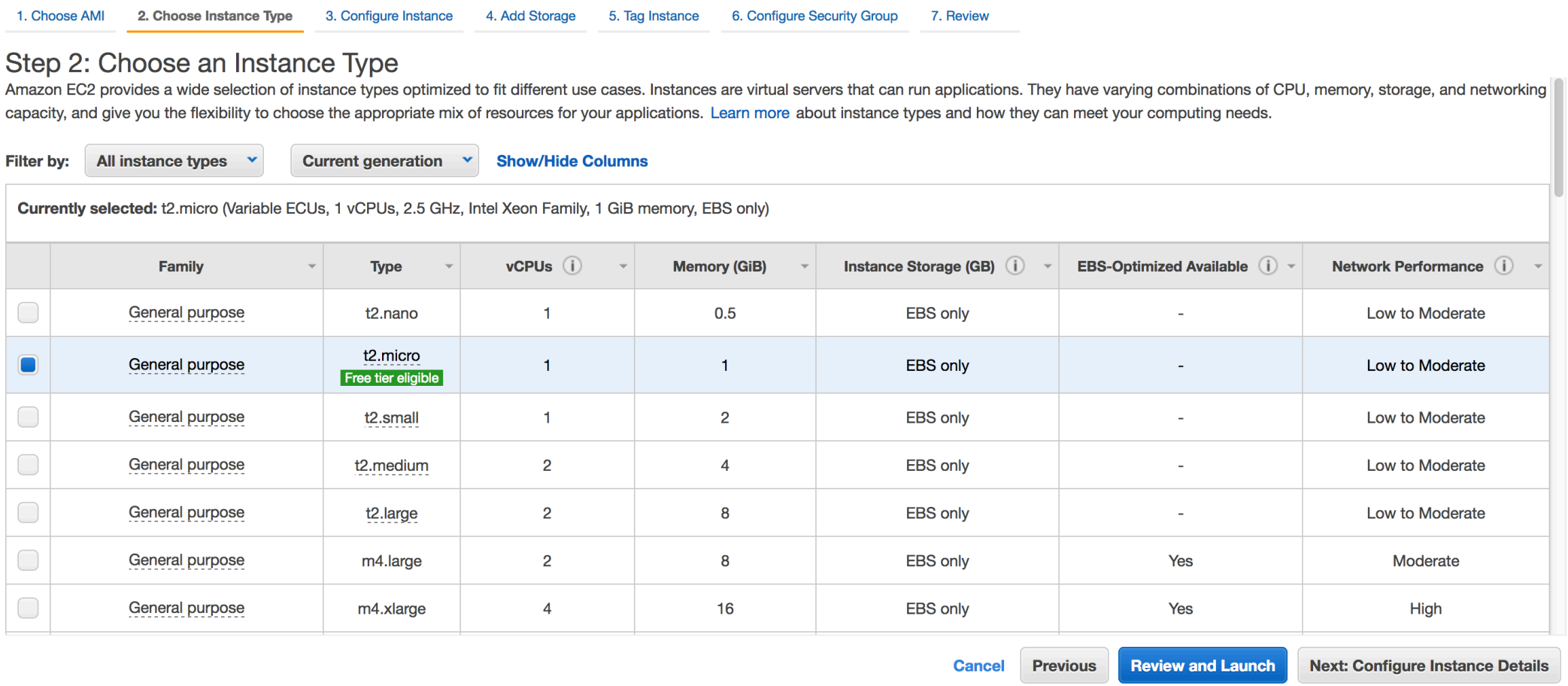
MongoDB on AWS



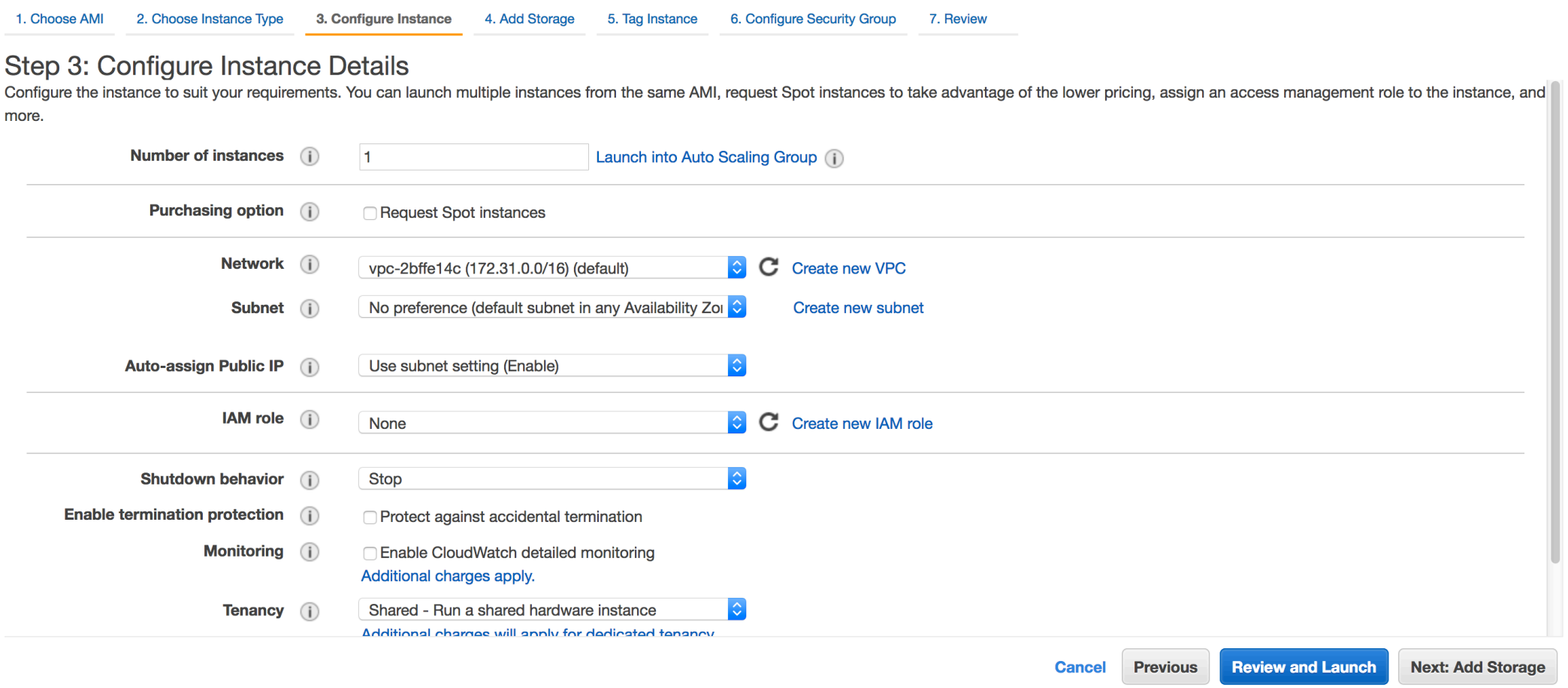
MongoDB on AWS



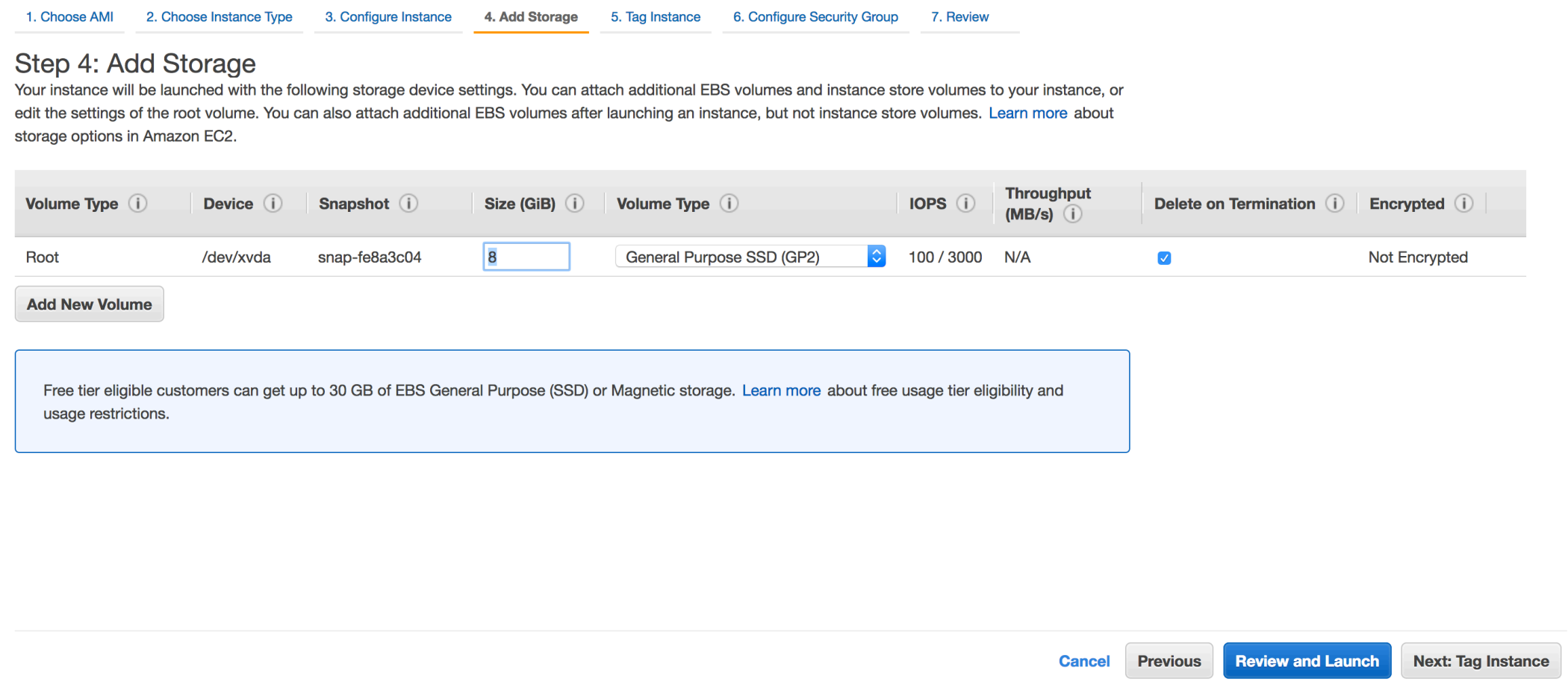
MongoDB on AWS



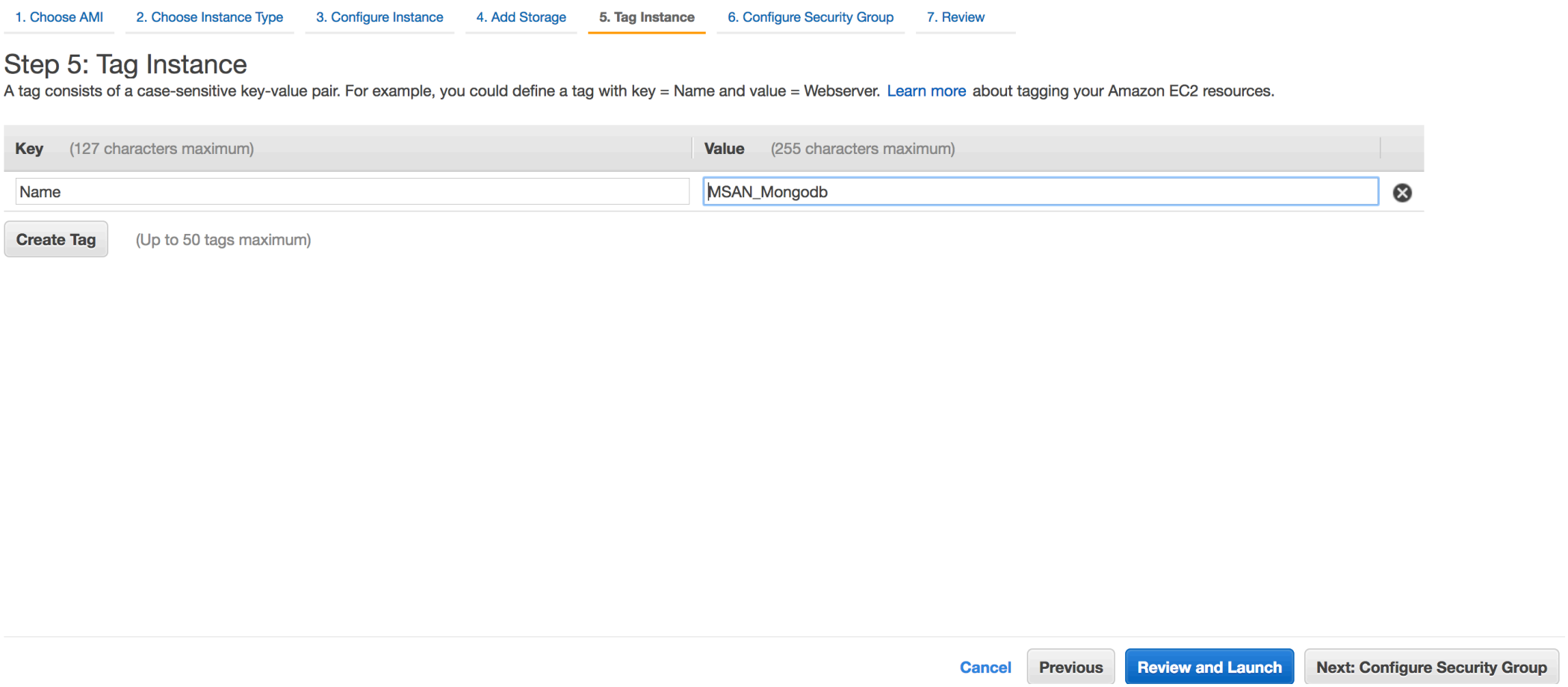
MongoDB on AWS



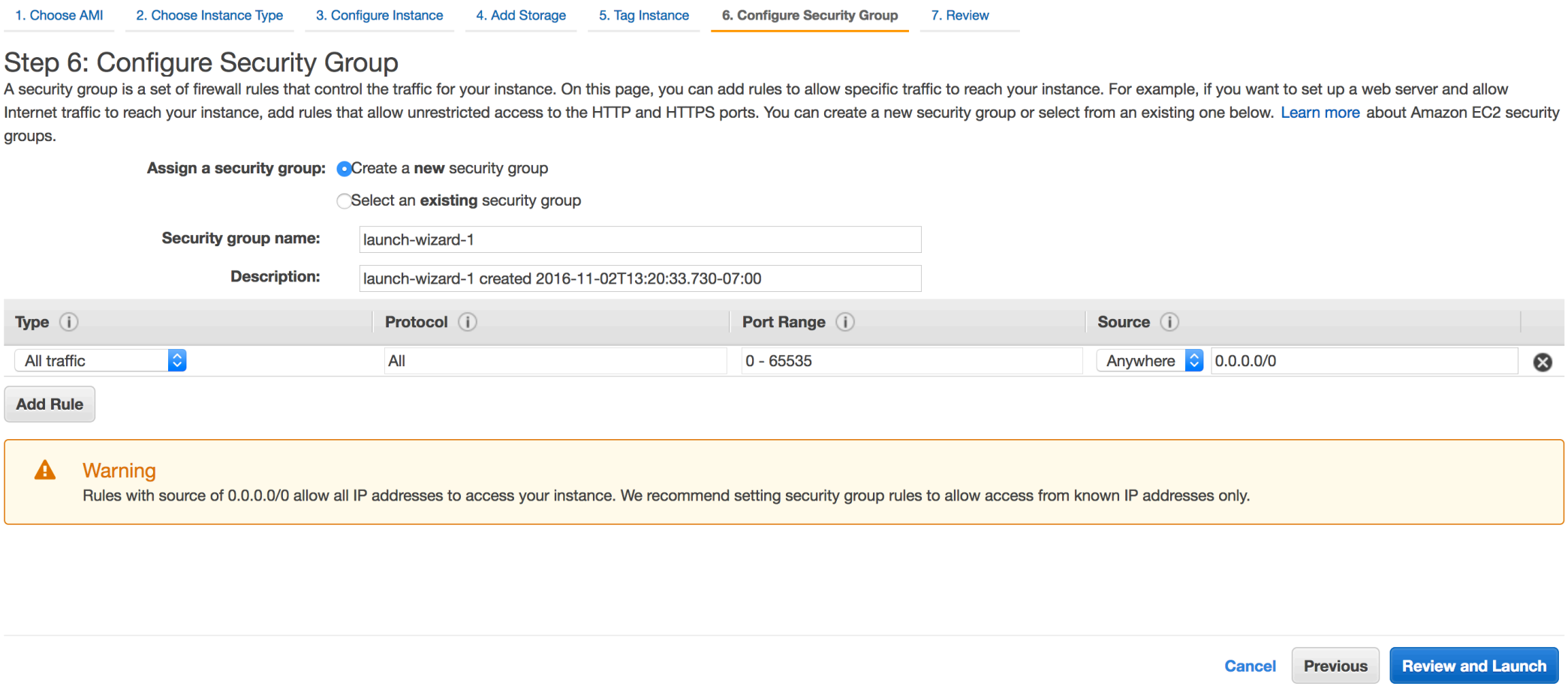
MongoDB on AWS



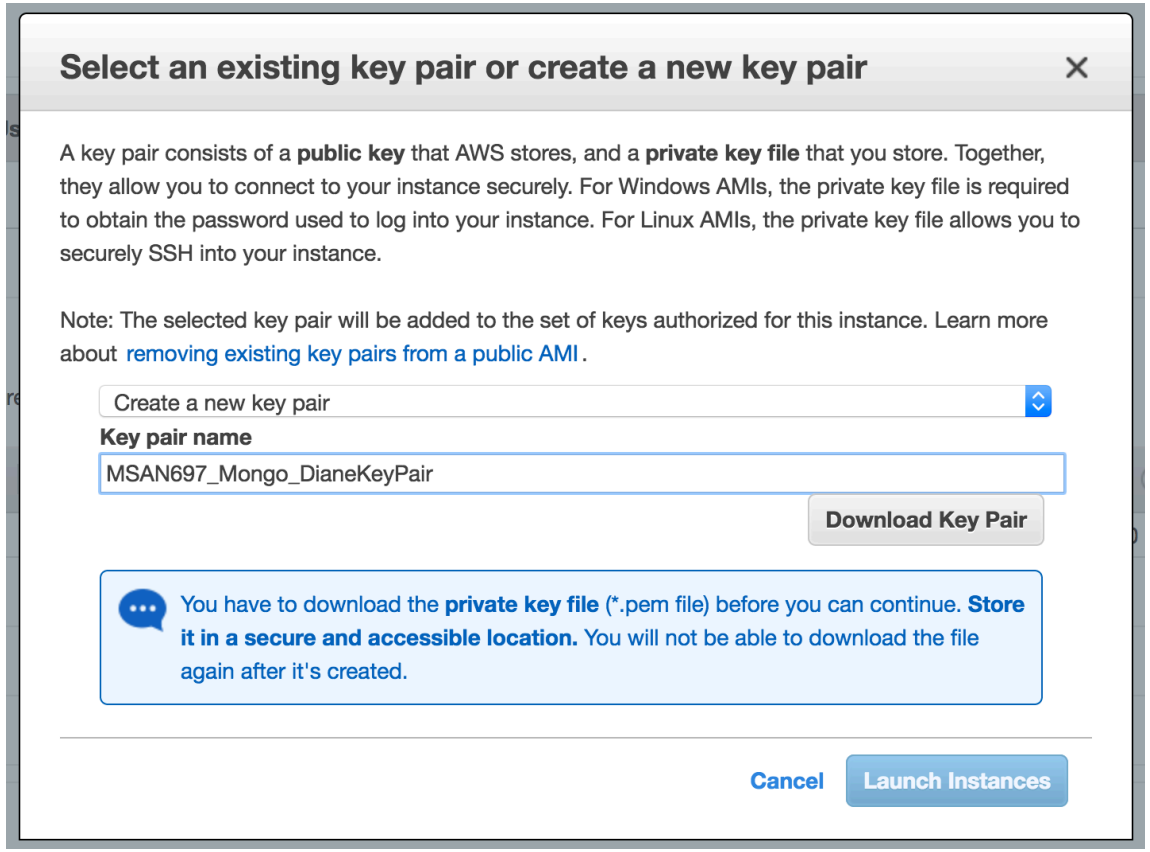
MongoDB on AWS



MongoDB on AWS



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