Scalabilty – a Gentle Intro

HADOOP201 – IXAT SOLUTIONS (ixatsolutions@gmail.com)

Agenda

- A BigData use case - Etsy
- Introduction to the Variables and Factors
- Building our own Scalable Architecture (in incremental steps)
  - Vertical Scaling
  - Vertical Partitioning
  - Horizontal Scaling
  - Horizontal Partitioning
  - … etc
- Platform Selection Considerations
- Tips

ETSY

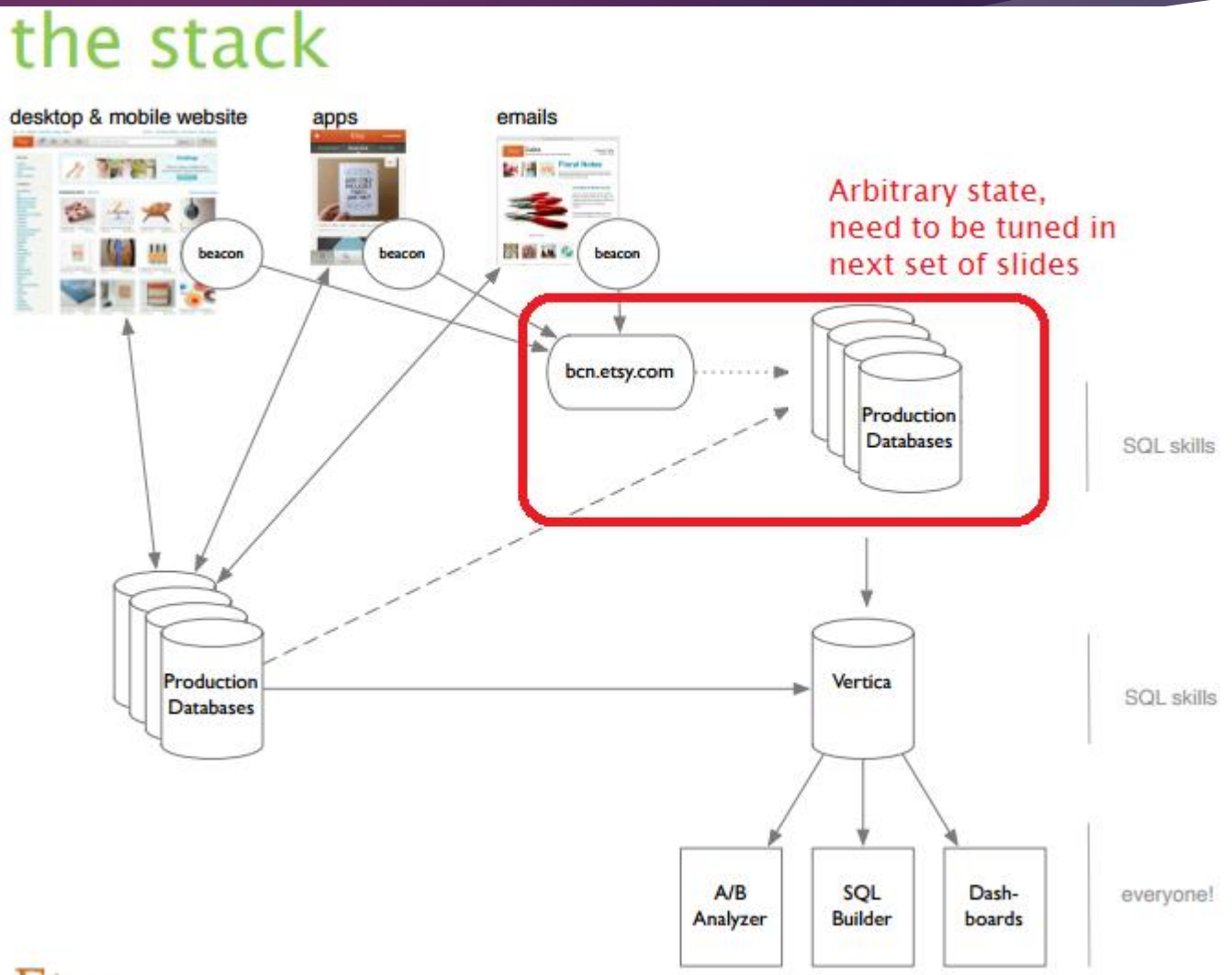
- Etsy is an online market place for handmade stuff
- Some stats
  - 2010
    - 9 Million Members
    - 9.5 Million Items
    - Revenue 314.3M
  - 2015
    - 54 Million Members
    - 35 Million Items
    - Revenue 1.93B (FY2014)

Preferred Architecture

Use Case: Analyzing large volume of log data, without taxing the databases

Scale of log data = 150GB/Day

Typical DB solution would incline to architect the web log analytics system as shown in red border (not preferred)- why? Lets understand in next set of slides



Why is scalability important

Lets Understand something about scalability and other performance variables and typical issues (more detailed at DB level) that are encountered from an online store perspective

- Scalability – The ability to handle sudden and/or seasonal spikes in demand for a business.
- To serve customer better, thus increasing the sale and customer retention(keep customers happy)

Other variables

- **Performance** – Optimal utilization of resources
- **Responsiveness** – Time taken per operation
- **Availability** - Probability of the application or a portion of the application being available at any given point in time
- **Downtime Impact** - The impact of a downtime of a server/service/resource - number of users, type of impact etc
- **Cost**
- **Maintenance Effort**

High: scalability, availability, performance & responsiveness

Low: downtime impact, cost & maintenance effort

The Factors

- Platform selection
- Hardware
- Application Design
- Database/Datastore Structure and Architecture
- Deployment Architecture
- Storage Architecture
- Abuse prevention
- Monitoring mechanisms
- … and more

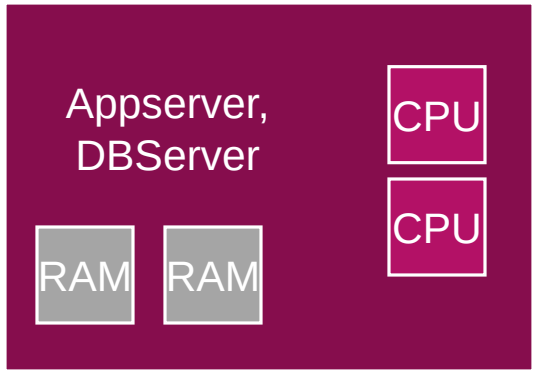
Lets Start …

- We will now build an example architecture for an example app using the following iterative incremental steps –
  - Inspect current Architecture
  - Identify Scalability Bottlenecks
  - Identify SPOFs and Availability Issues
  - Identify Downtime Impact Risk Zones
  - Apply one of -
    - Vertical Scaling
    - Vertical Partitioning
    - Horizontal Scaling
    - Horizontal Partitioning
  - Repeat process

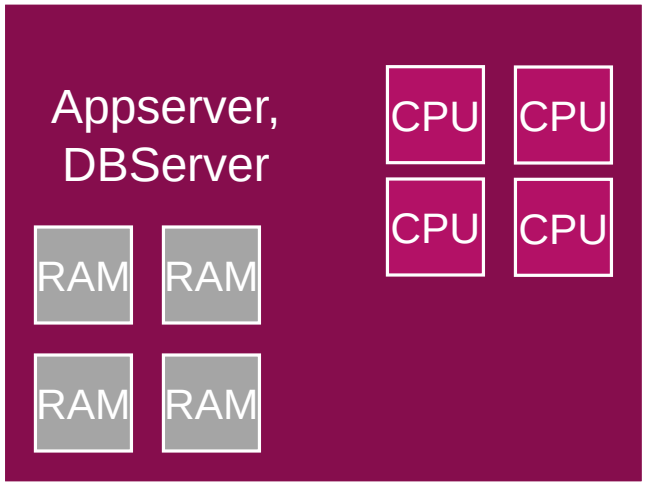
Step 1 – Lets Start …

Appserver & DBServer

Step 2 – Vertical Scaling

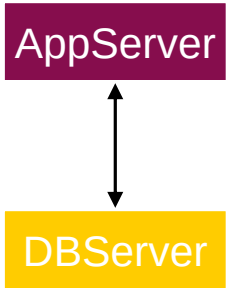


Step 2 - Vertical Scaling



- Introduction
  - Increasing the hardware resources without changing the number of nodes
  - Referred to as "Scaling up" the Server
- Advantages
  - Simple to implement
- Disadvantages
  - Finite limit
  - Hardware does not scale linearly (diminishing returns for each incremental unit)
  - Requires downtime
  - Increases Downtime Impact
  - Incremental costs increase exponentially

Step 3 – Vertical Partitioning (Services)



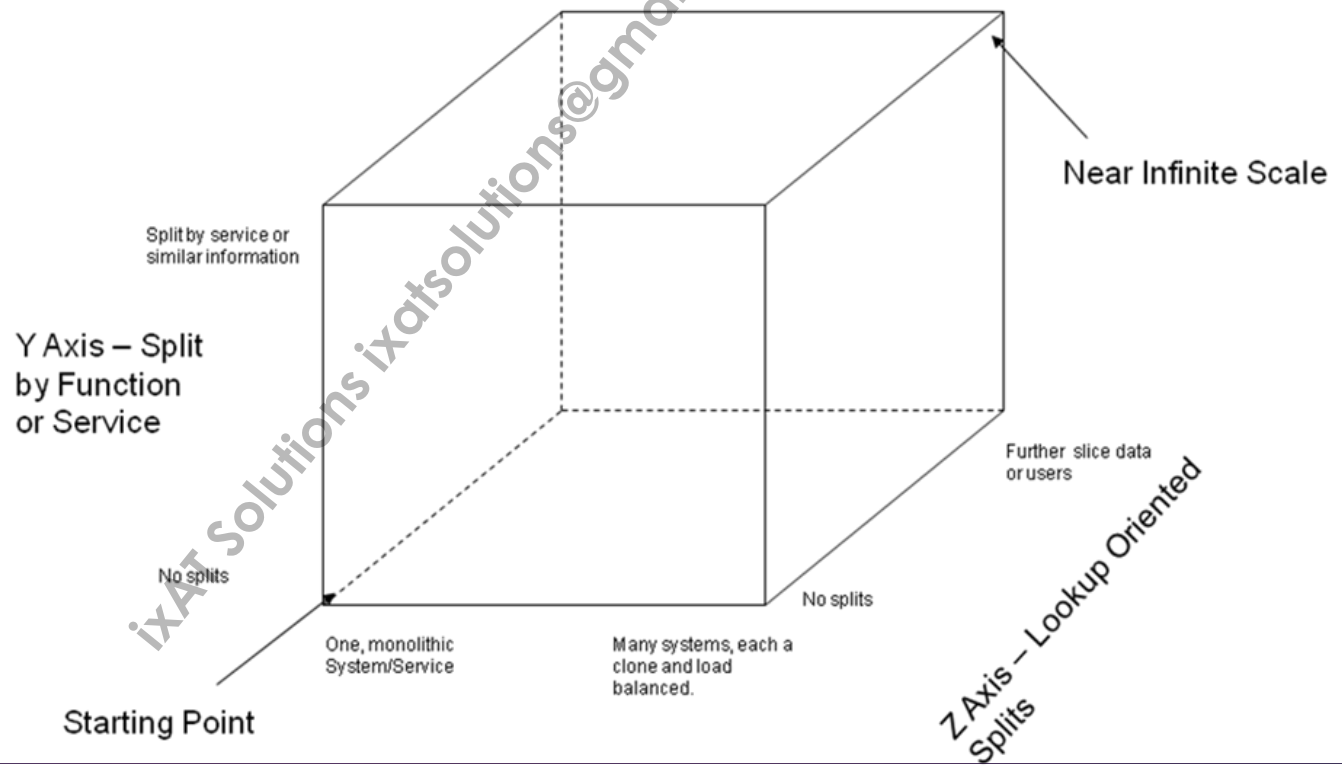
- Introduction
  - Deploying each service on a separate node
- Positives
  - Increases per application Availability
  - Task-based specialization, optimization and tuning possible
  - Reduces context switching
  - Simple to implement for out of band processes
  - No changes to App required
  - Flexibility increases
- Negatives
  - Sub-optimal resource utilization
  - May not increase overall availability
  - Finite Scalability

Understanding Vertical Partitioning

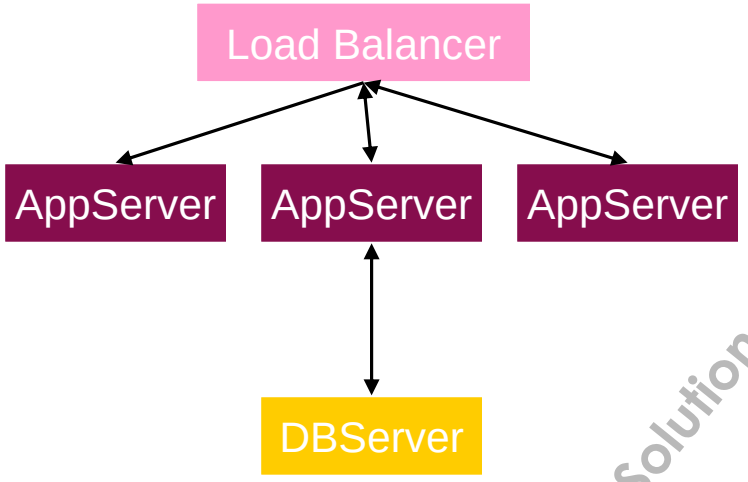
- The term Vertical Partitioning denotes –
  - Increase in the number of nodes by distributing the tasks/functions
  - Each node (or cluster) performs separate Tasks
  - Each node (or cluster) is different from the other
- Vertical Partitioning can be performed at various layers (App / Server / Data / Hardware etc)

AFK Cube

- Abbott, Keeven & Fisher Partners, Partners In Hyper Growth. Top industry consultants in scaling systems. Proposed a standard model depicting how systems can scale



Step 4 – Horizontal Scaling (App Server)



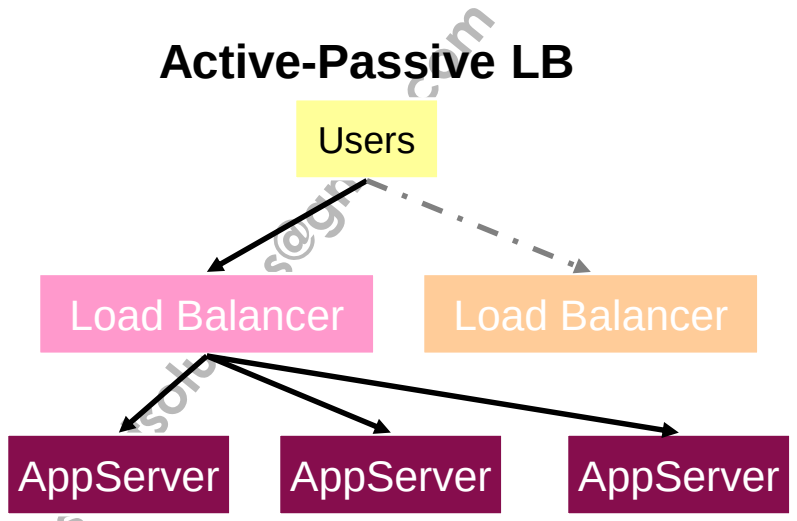
- Introduction
  - Increasing the number of nodes of the App Server through Load Balancing
  - Referred to as “Scaling out” the App Server

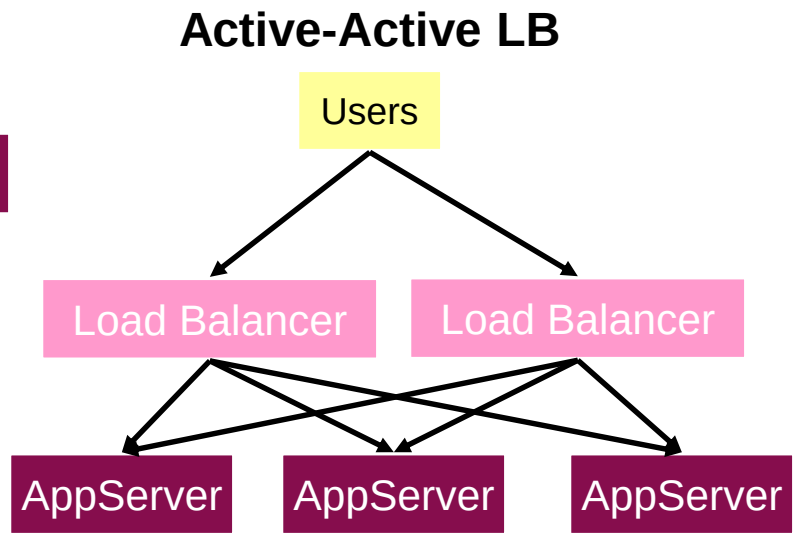
Understanding Horizontal Scaling

- The term Horizontal Scaling denotes –
  - Increase in the number of nodes by replicating the nodes
  - Each node performs the same Tasks
  - Each node is identical
  - Typically the collection of nodes maybe known as a cluster (though the term cluster is often misused)
  - Also referred to as “Scaling Out”
- Horizontal Scaling can be performed for any particular type of node (AppServer / DBServer etc)
- X-Axis scale as per AFK Cube

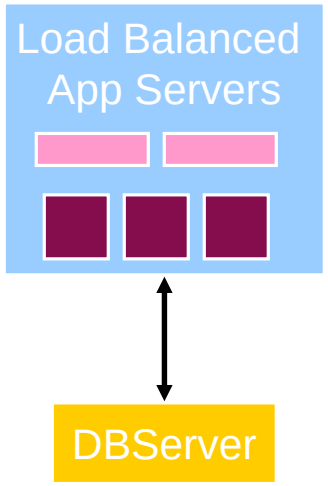
Load Balancer – Removing SPOF

- In a Load Balanced App Server Cluster the LB is an SPOF
- Setup LB in Active-Active or Active-Passive mode
  - Note: Active-Active nevertheless assumes that each LB is independently able to take up the load of the other
  - If one wants ZERO downtime, then Active-Active becomes truly cost beneficial only if multiple LBs (more than 3 to 4) are daisy chained as Active-Active forming an LB Cluster



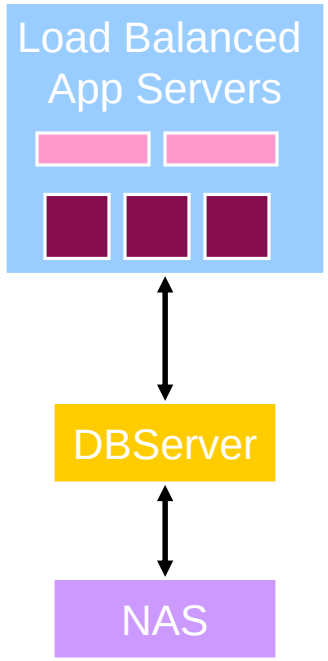


Step 4 – Horizontal Scaling (App Server)



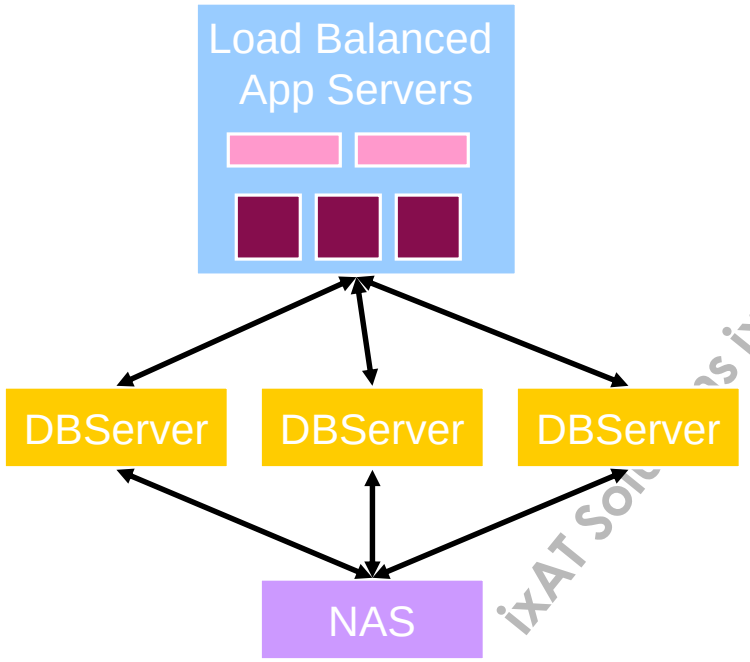
- Our deployment at the end of Step 4
- Positives
  - Increases Availability and Scalability
  - No changes to App required
  - Easy setup
- Negatives
  - Finite Scalability

Step 5 – Vertical Partitioning (Hardware)



- Introduction
  - Partitioning out the Storage function using a SAN/NAS
- Positives
  - Allows “Scaling Up” the DB Server
  - Boosts Performance of DB Server
- Negatives
  - Increases Cost

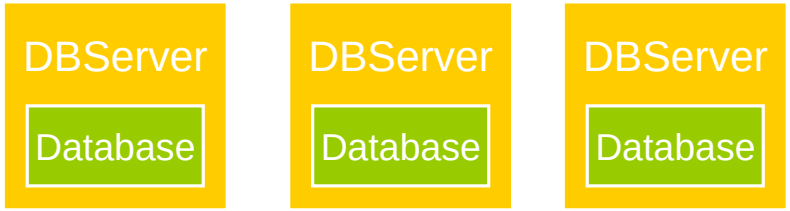
Step 6 – Horizontal Scaling (DB)



- Introduction
  - Increasing the number of DB nodes
  - Referred to as “Scaling out” the DB Server (X-Axis scaling at DB level)
- Options
  - Shared nothing Cluster
  - Real Application Cluster (RAC or Shared Storage Cluster)

Shared Nothing Cluster

- Each DB Server node has its *own complete* copy of the database
- Nothing is shared between the DB Server Nodes
- This is achieved through DB Replication at DB / Driver / App level or through a proxy
- Supported by most RDBMs natively or through 3rd party software



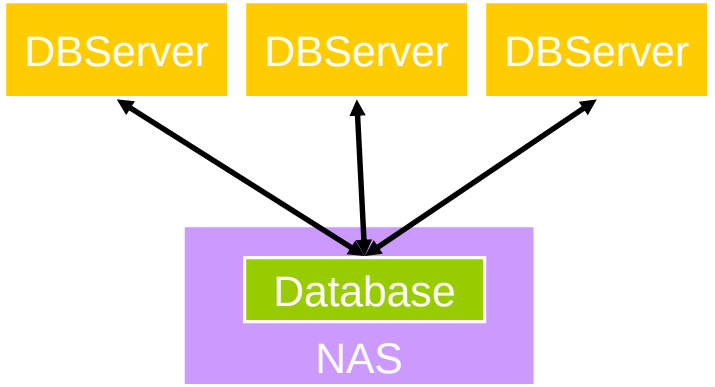
Note: Actual DB files maybe stored on a central NAS/SAN

Replication Considerations

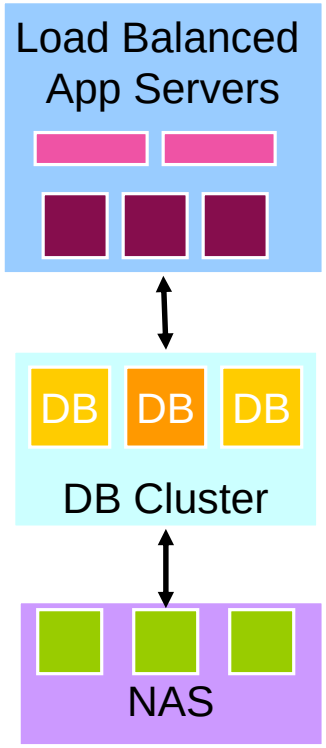
- Master-Slave
  - Writes are sent to a single master which replicates the data to multiple slave nodes
  - Replication maybe cascaded
  - Simple setup
  - No conflict management required
- Multi-Master
  - Writes can be sent to any of the multiple masters which replicate them to other masters and slaves
  - Conflict Management required
  - Deadlocks possible if same data is simultaneously modified at multiple places

Real Application Cluster

- All DB Servers in the cluster share a common storage area on a NAS
- All DB servers mount the same block device

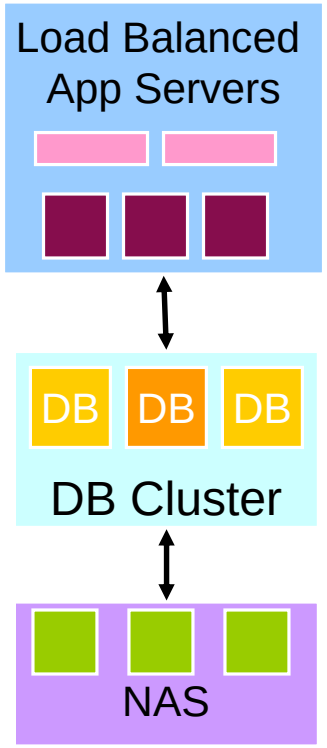


Step 6 – Horizontal Scaling (DB)



- Our architecture now looks like this
- Positives
  - As Web servers grow, Database nodes can be added
  - DB Server is no longer SPOF
- Negatives
  - Finite limit
  - Replication

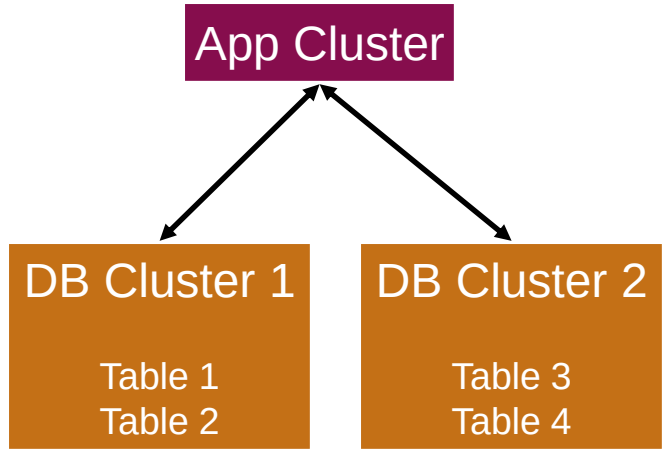
Step 7 – Vertical / Horizontal Partitioning (DB)



- Introduction
  - Increasing the number of DB Clusters by dividing the data
- Options
  - Vertical Partitioning - Dividing tables / columns (Y Axis Scaling)
  - Horizontal Partitioning - Dividing by rows (value) (Z Axis Scaling)

Vertical Partitioning (DB)

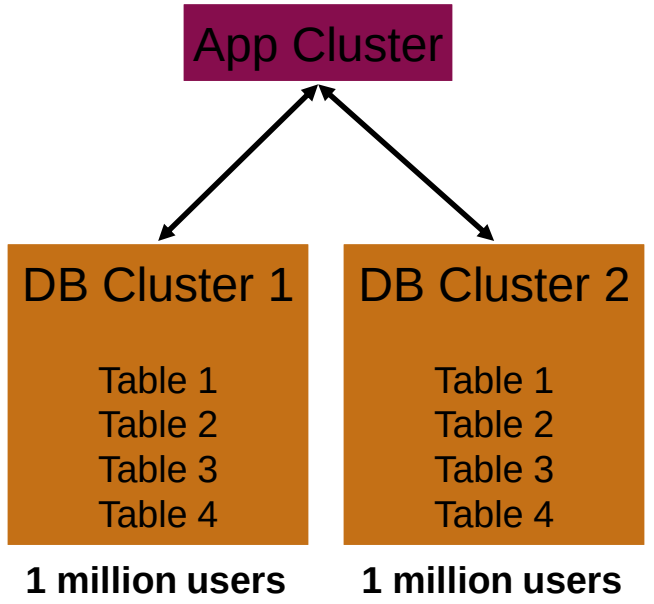
- Take a set of tables and move them onto another DB
  - Eg in a social network - the *users* table and the *friends* table can be on separate DB clusters
  - In case of Etsy, one could move categories of products to different tables into the clusters
- Each DB Cluster has different tables
- Application code or DAO / Driver code or a proxy knows where a given table is and directs queries to the appropriate DB
- Can also be done at a column level by moving a set of columns into a separate table



- Negatives
  - One cannot perform SQL joins or maintain referential integrity (referential integrity is as such over-rated)
  - Finite Limit

Horizontal Partitioning (DB)

- Take a set of rows and move them onto another DB
  - Eg in a social network – each DB Cluster can contain all data for 1 million *users*
- Each DB Cluster has identical tables
- Application code or DAO / Driver code or a proxy knows where a given row is and directs queries to the appropriate DB
- Negatives
  - SQL unions for search type queries must be performed within code

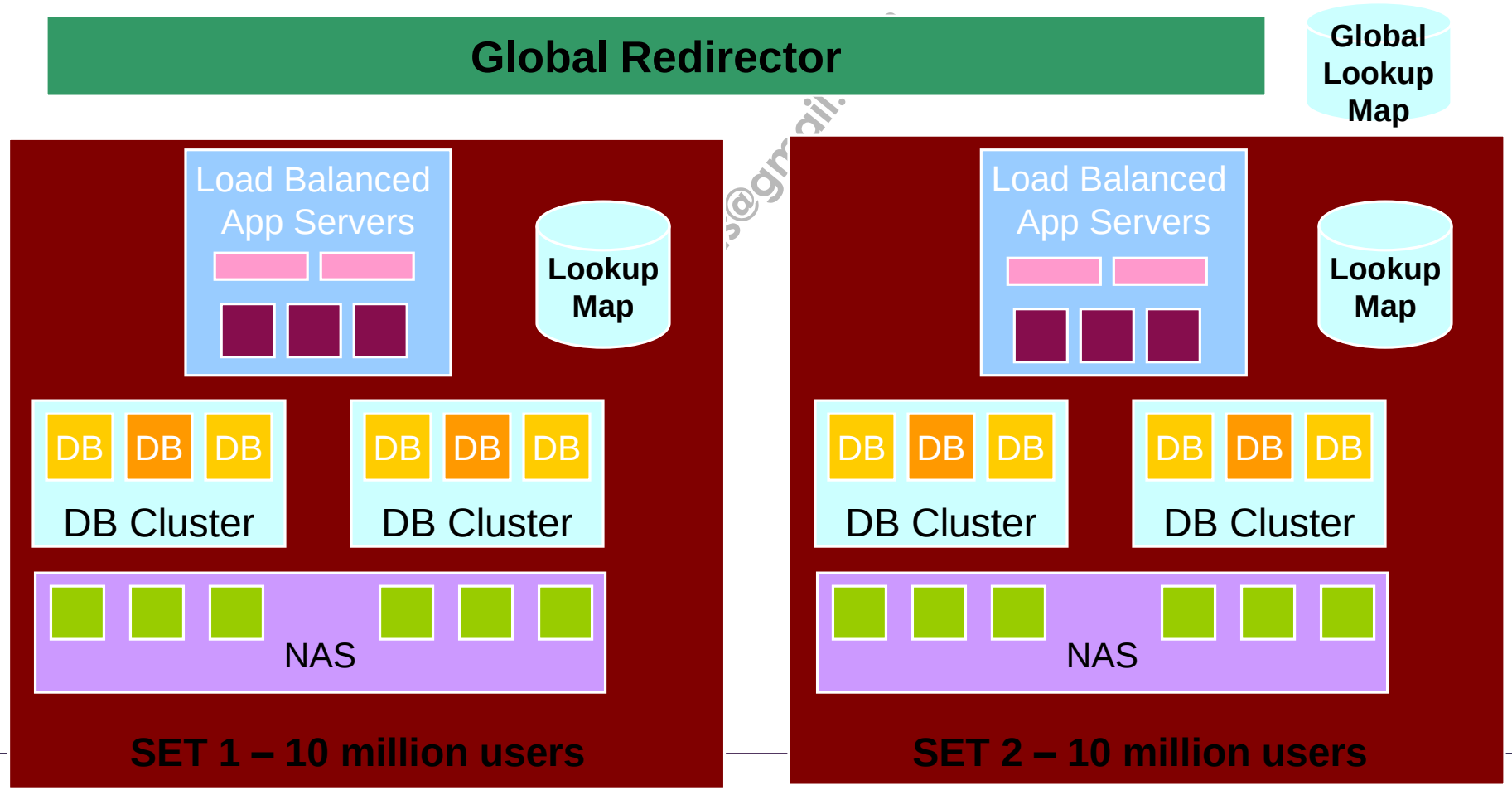


Horizontal Partitioning (DB)

- Techniques
  - FCFS
    - 1st million users are stored on cluster 1 and the next on cluster 2
  - Round Robin
  - Least Used (Balanced)
    - Each time a new user is added, a DB cluster with the least users is chosen
  - Hash based
    - A hashing function is used to determine the DB Cluster in which the user data should be inserted
  - Value Based
    - User ids 1 to 1 million stored in cluster 1 OR
    - all users with names starting from A-M on cluster 1
  - Except for Hash and Value based all other techniques also require an independent lookup map – mapping user to Database Cluster
  - This map itself will be stored on a separate DB (which may further need to be replicated)

Step 8 – Separating Sets

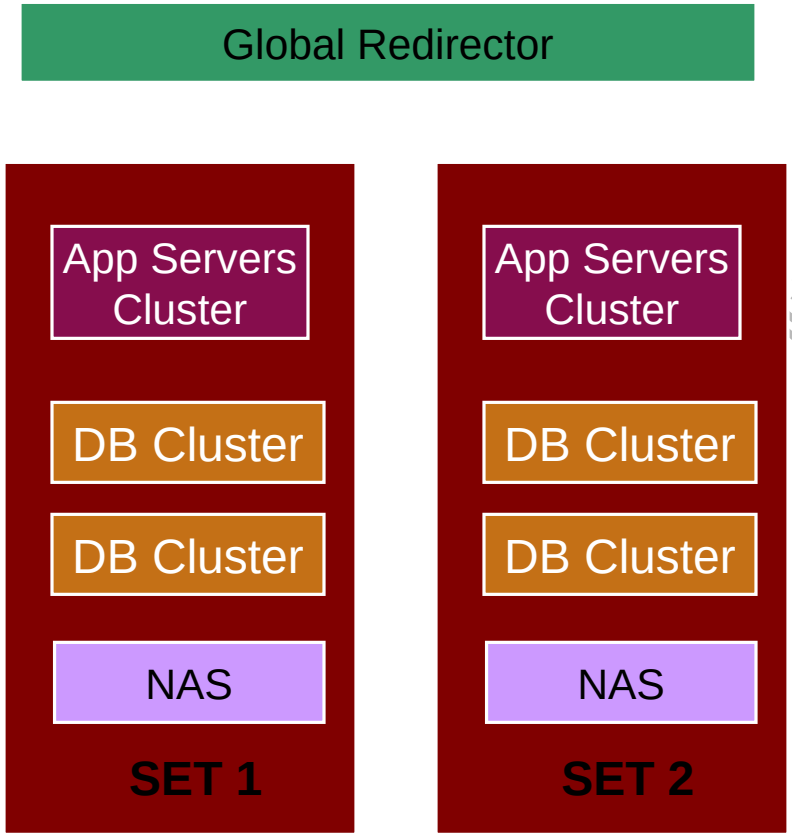
Now we consider each deployment as a single Set serving a [illegible] of users



Creating Sets

- The goal behind creating sets is easier manageability
- Each Set is independent and handles transactions for a set of users
- Each Set is architecturally identical to the other
- Each Set contains the entire application with all its data structures
- Sets can even be deployed in separate datacenters
- Users may even be added to a Set that is closer to them in terms of network latency

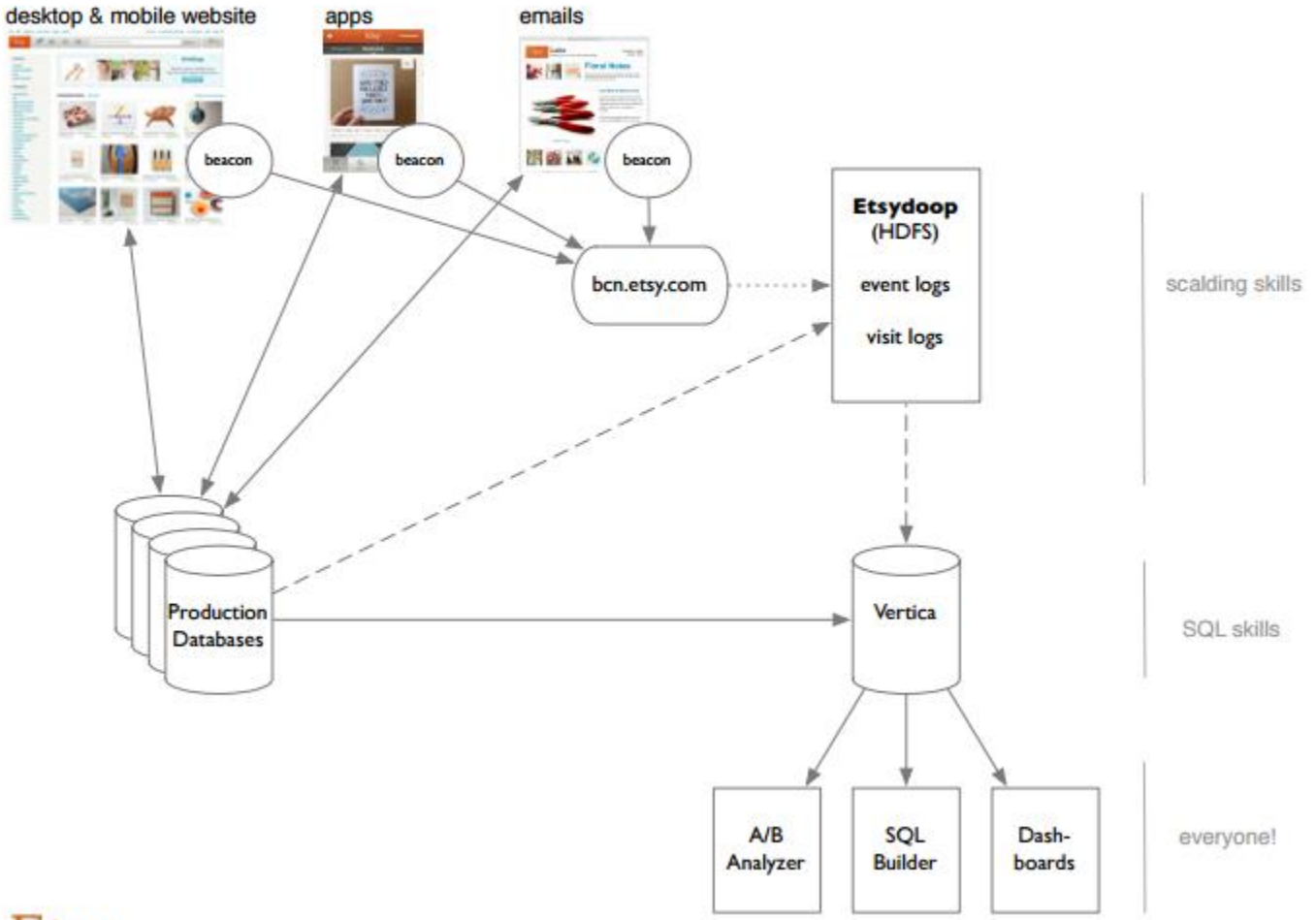
Step 8 – Horizontal Partitioning (Sets)



- Our architecture now looks like this
- Positives
  - Infinite Scalability but at cost
- Negatives
  - Aggregation of data across sets is complex
  - Users may need to be moved across Sets if sizing is improper
  - Global App settings and preferences need to be replicated across Sets
  - Application need to take care of rebalancing partitions
  - Rebalancing requires downtime

Etsy's solution

the stack



Etsy