University of BRISTOL

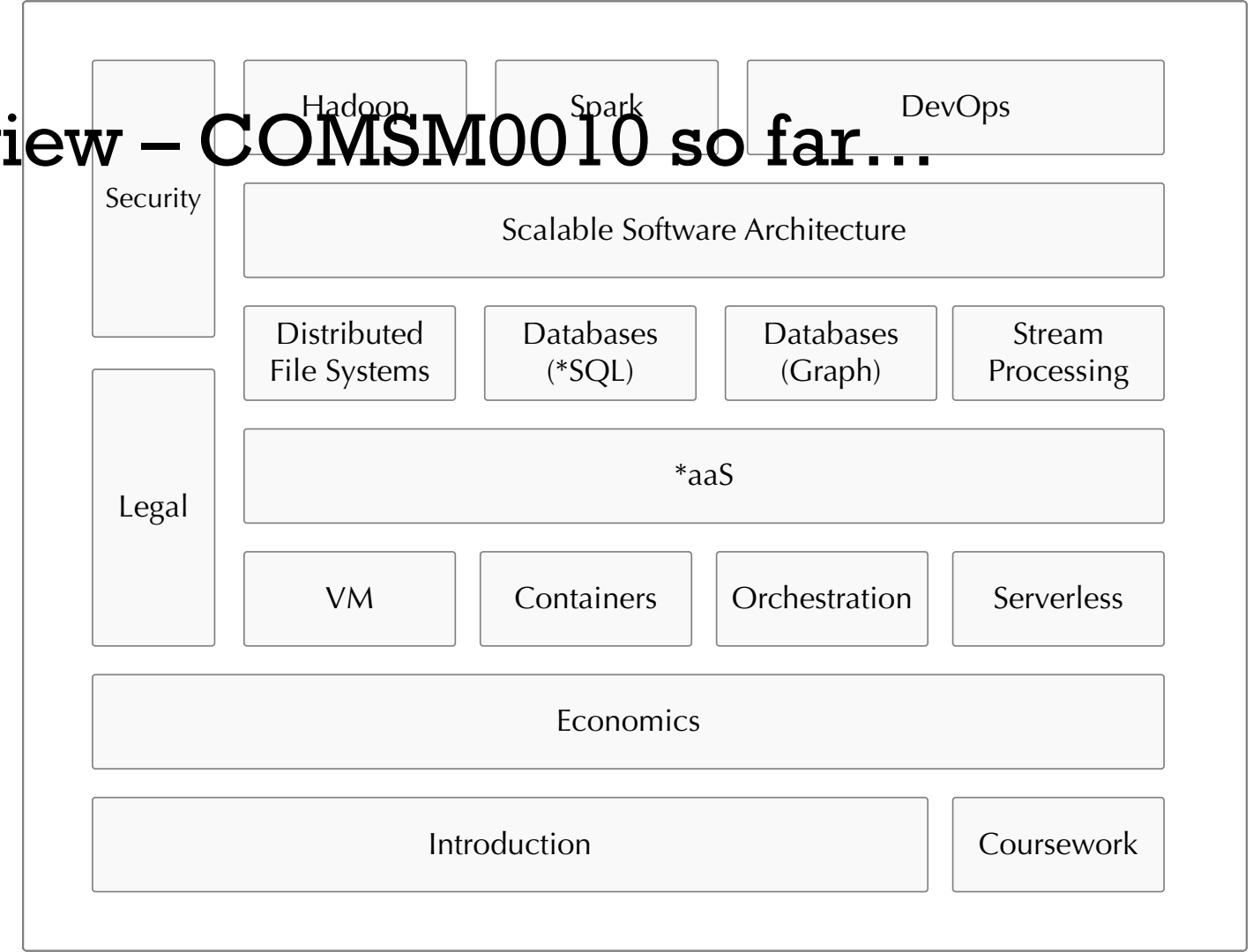
Scalable Cloud Architectures

Dr Dan Schien – COMSM0010

Lecture 11

bristol.ac.uk

Review – COMSM0010 so far…



What is Cloud Native?

- Container based
  - Capabilities
- Elastic
- Microservices
  - Polyglot
- DevOps

https://www.theregister.co.uk/2018/10/19/yale_security_fail/

DATA CENTRE SOFTWARE SECURITY DEVOPS BUSINESS PERSONAL TECH SCIENCE EM

Data Centre ▸ Cloud

Yale Security Fail: 'Unexpected load' caused systems to crash, whacked our Smart Living Home app

All working now says biz. No, no, no, no, say customers, it is NOT!

By Paul Kunert 19 Oct 2018 at 16:30 59 SHARE ▼

https://www.theregister.co.uk/2018/10/19/yale_security_fail/

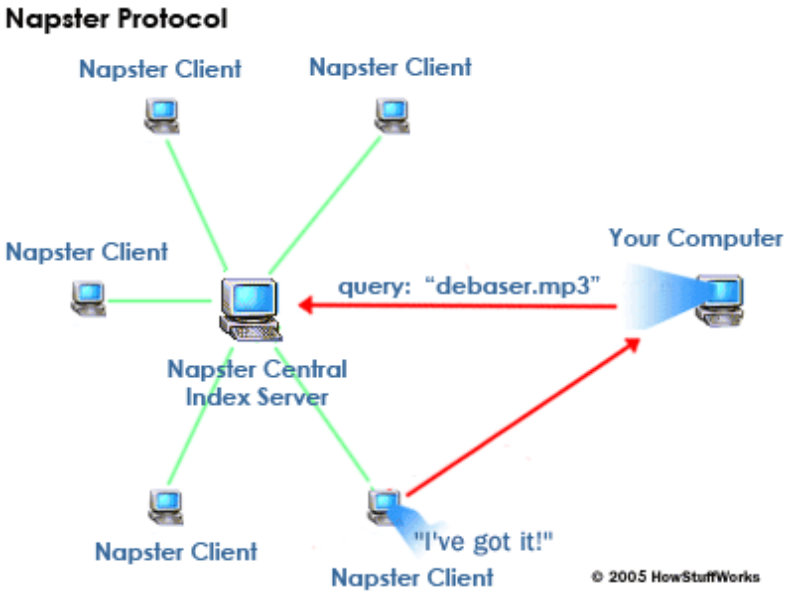
Software Architectures

Definition Software Architecture

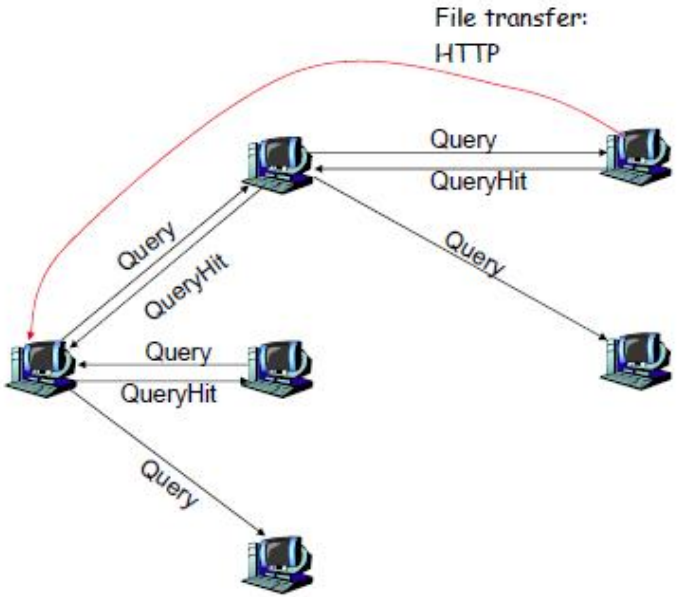
- Software architecture is a set of **structures** needed to **reason** about a system. It includes software components, the **relations** between them and properties of both.
- Architecture might be implicit
  - If you're building any significant system. You cannot not have an architecture. You might just not been have documented or communicated explicitly.

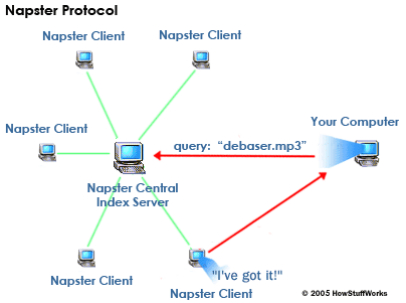
What is the different between Napster and Gnutella?

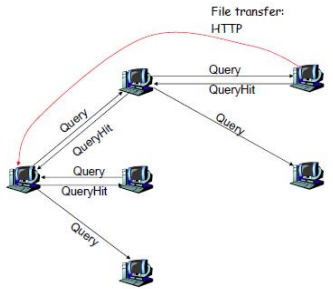
Napster


Napster Protocol
Napster Client
Napster Client
Napster Client
Your Computer
query: "debaser.mp3"
Napster Central
Index Server
Napster Client
"I've got it!"
Napster Client
© 2005 HowStuffWorks

Gnutella


File transfer:
HTTP
Query
QueryHit
Query
QueryHit
Query
Query
QueryHit
Query



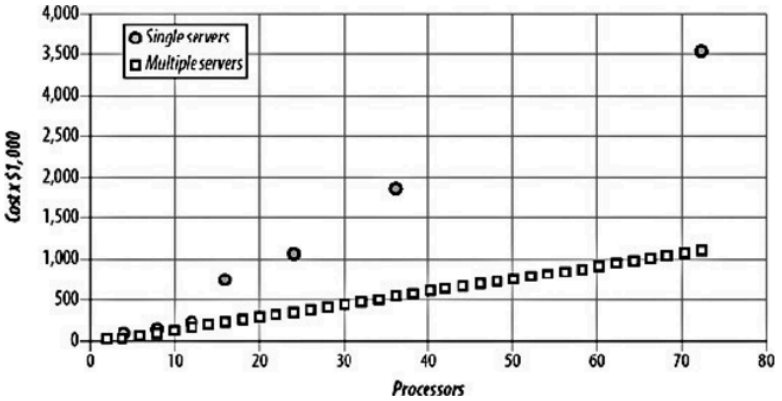


	Napster	Gnutella
Architecture	Centralised Index	Unstructured Using simple flooding
Advantages	+ Highly efficient data lookup + Rapidly adapts to changes in network	+ Entirely decentralized, pure P2P network + Highly resistant to failure
Disadvantages	- Questionable scalability - Vulnerable to censorship, failure, attack	- Search is time-consuming - Network typically scales poorly

Principles of Scaling Systems

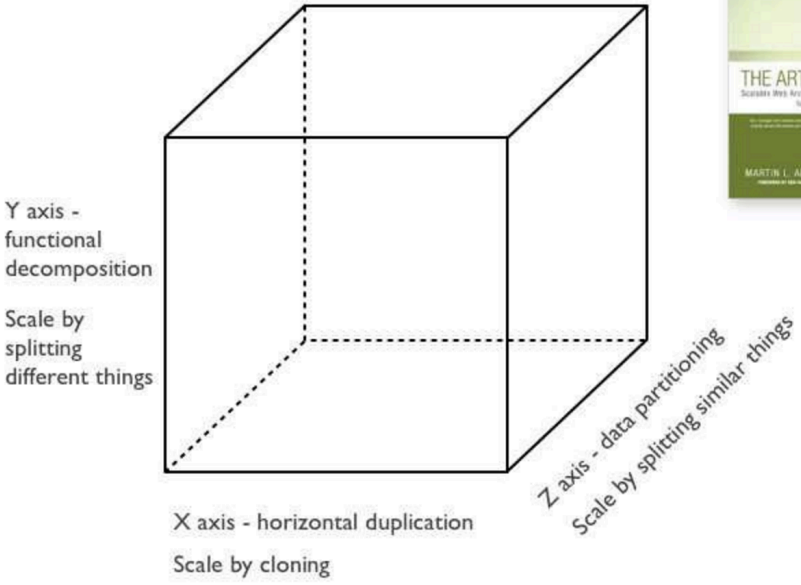
Horizontal vs Vertical Scaling

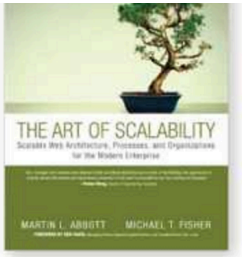
- Vertical (Scale Up) bigger machines
  - no need to change your architecture
- Horizontal (Scale Out) more machines
- Trade-off: Cost of metal vs cost of developers



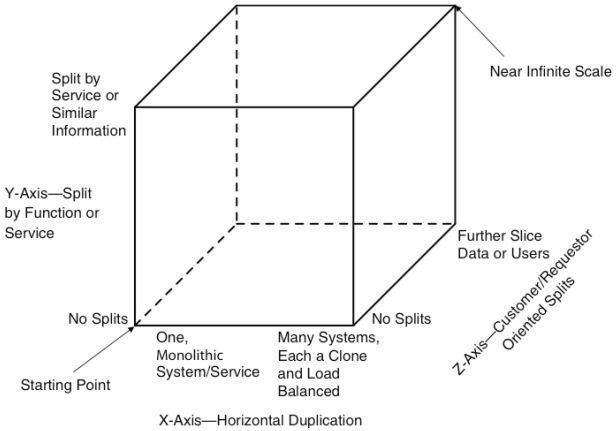
http://www.amazon.com/dp/0596102356

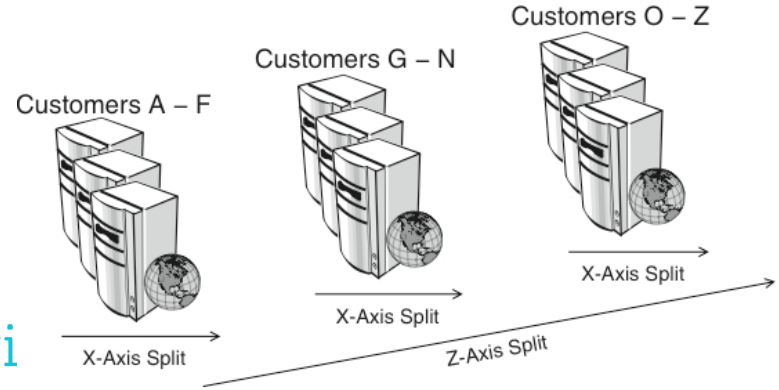
The Scale Cube (of Scaling Out – horizontally only)

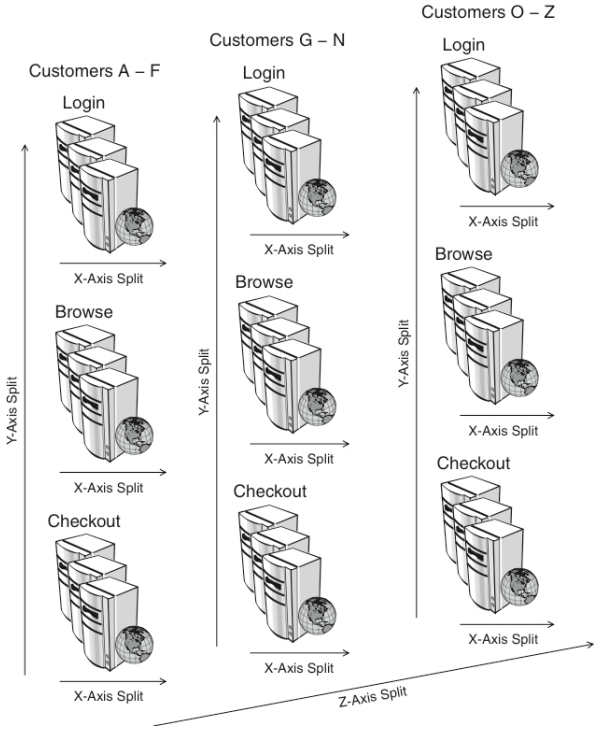




bristc


Split by Service or Similar Information
Near Infinite Scale
Y-Axis—Split by Function or Service
Further Slice Data or Users
No Splits
No Splits
Z-Axis—Customer/Requestor Oriented Splits
One, Monolithic System/Service
Many Systems, Each a Clone and Load Balanced
Starting Point
X-Axis—Horizontal Duplication


Customers A – F
Customers G – N
Customers O – Z
X-Axis Split
X-Axis Split
Z-Axis Split

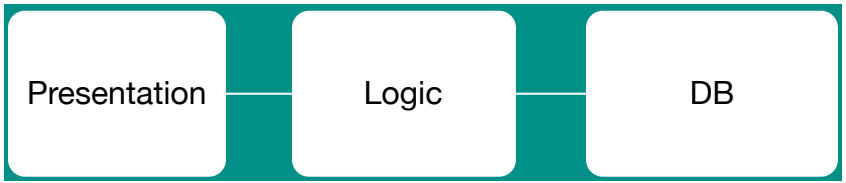

Customers A – F
Customers G – N
Customers O – Z
Login
Browse
Checkout
Y-Axis Split
X-Axis Split
Z-Axis Split

bri

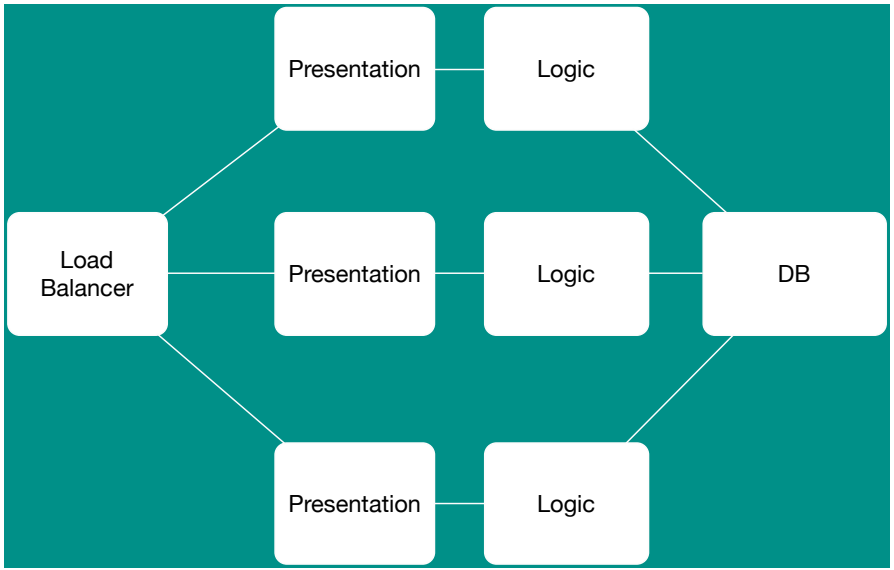
- X axis (horizontal duplication) unbiased cloning of services and data
  - each clone can do the work of all other clones. work is distributed among clones without bias
  - inefficient compared to alternatives
  - easy to do
- Y axis (split by function or service) refers to isolating and making scalable individual responsibility of components
  - needs to be split in the code base
  - more costly than x-axis
- Z axis - partitioning the domain of incoming requests
  - data partitioning, split relative to client
  - improves fault tolerance, cache performance
  - more costly than other two

How to scale out? Layered Architectures

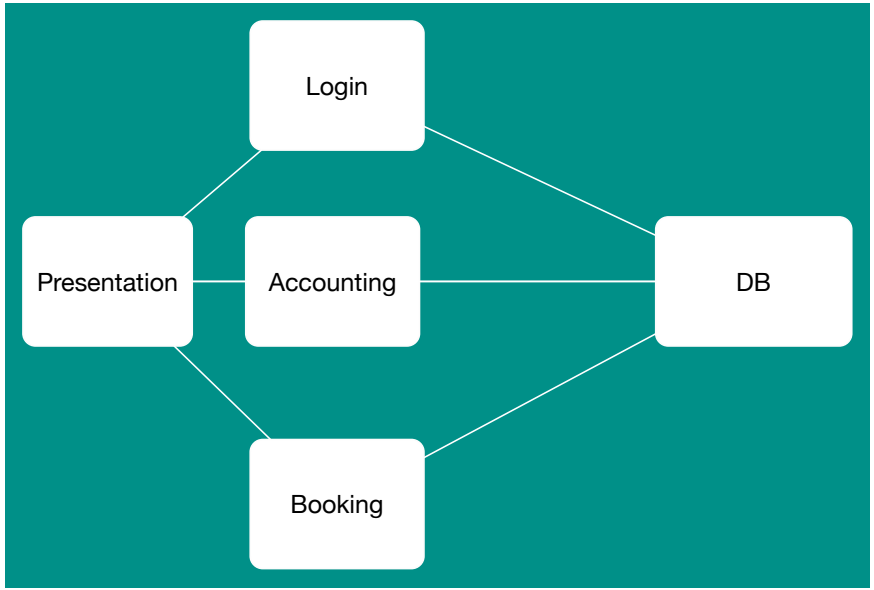
- Consider this simple application, might be a teaching and learning system – such as Blackboard
- Presentation layer provides a mobile API and web UI
- Logic layer has the business logic, book hotels and flights, etc



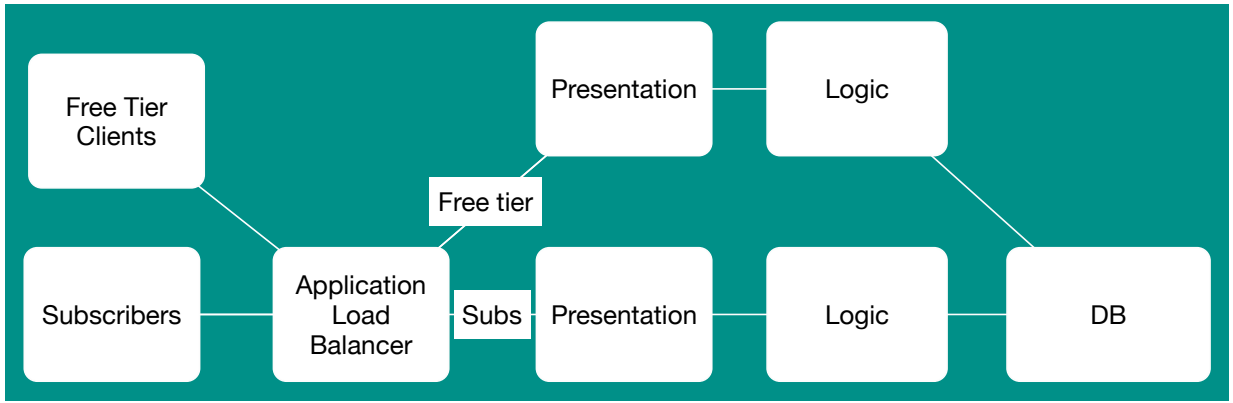
X-Axis scaling
Scale application logic or presentation layers



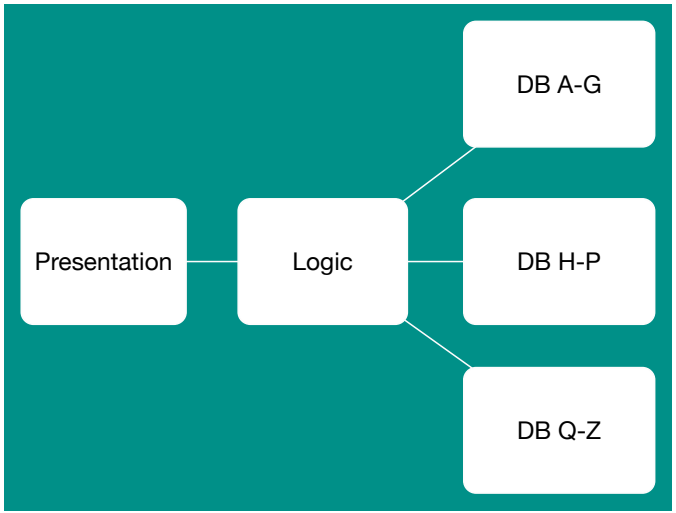
Y-Axis: decompose by function



Z-Axis scaling 1: partition per request type



Z-Axis 2: Scaling by partitioning the data



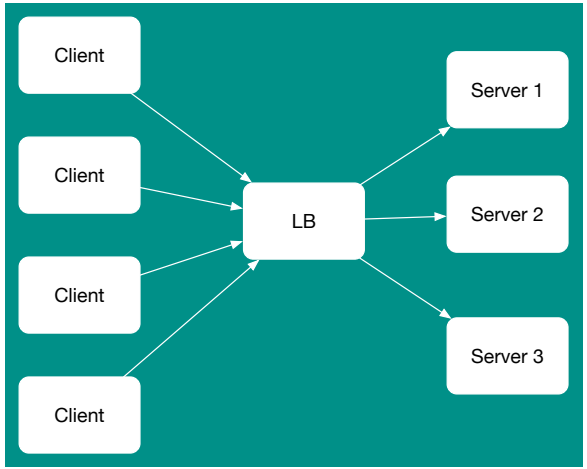
Architectural components and patterns for scalable systems

Decoupled Components

- Independent scalability of components
- Preliminary for Fault Tolerance and High Availability
- Essential requirement for the use of these principles: APIs defined for all components (e.g. HTTP REST, RPC or typed messages)
- Four important mechanisms to **decouple**
  - Load Balancers
  - Message Queues
  - Message Topics
  - Service Registries

Load Balancers

- distributing requests
- manage availability -> HA
- Perform health check
- Session affinity / Sticky Sessions
- list of backend servers
- load balancing policy
- SSL
- Alternative: DNS or Router based



Load Balancers

- Hides the server from client access

Airbnb[11]
Amazon.com[8]
Ancestry.com[12][13]
The A.V. Club[14]
BBC[13]
The Boston Globe[11]
Box[15]
Business Insider[13]
CNN[13]
Comcast[16]
CrunchBase[13]
DirecTV[13]
The Elder Scrolls Online[13][17]
Electronic Arts[16]
Etsy[11][18]
FiveThirtyEight[13]
Fox News[19]
The Guardian[19]
GitHub[11][16]
Grubhub[20]
HBO[13]
Heroku[21]
HostGator[13]
iHeartRadio[12][22]
Imgur[23]
Indiegogo[12]
Mashable[24]
National Hockey League[13]
Netflix[13][19]
The New York Times[11][16]
Overstock.com[13]
PayPal[18]
Pinterest[16][18]
Pixlr[13]
PlayStation Network[16]
Qualtrics[12]
Quora[13]
Reddit[12][16][18]
Roblox[25]
Ruby Lane[13]
RuneScape[12]
SaneBox[21]
Seamless[23]
Second Life[26]
Shopify[11]
Slack[23]
SoundCloud[11][18]
Squarespace[13]
Spotify[12][16][18]
Starbucks[12][22]
Storify[15]
Swedish Civil Contingencies Agency[27]
Swedish Government[27]
Tumblr[12][16]
Twilio[12][13]
Twitter[11][12][16][18]
Verizon Communications[16]
Visa[28]
Vox Media[29]
Walgreens[13]
The Wall Street Journal[19]
Wikia[12]
Wired[15]
Wix.com[30]
WWE Network[31]
Xbox Live[32]
Yammer[23]
Yelp[13]
Zillow[13]

https://en.wikipedia.org/wiki/2016_

```
// root      xc3511      // root      vizxv          // root      admin
// admin     admin       // root      888888         // root      xmhdipc
// root      default     // root      juantech       // root      123456
// root      54321       // support   support        // root      (none)
// admin     password    // root      root           // root      12345
// user      user        // admin     (none)         // root      pass
// admin     admin1234   // root      1111           // admin     smcadmin
// admin     1111        // root      666666         // root      password
// root      1234        // root      klv123         // Administrator admin
// service   service     // supervisor supervisor    // guest     guest
// guest     12345       // guest     12345          // admin1    password
// administrator 1234    // 666666    666666         // 888888    888888
// ubnt      ubnt        // root      klv1234        // root      Zte521
// root      hi3518      // root      jvbzd          // root      anko
// root      zlxx.       // root      7ujMko0vizxv   // root      7ujMko0admin
// root      system      // root      ikwb           // root      dreambox
// root      user        // root      realtek        // root      00000000
// admin     1111111     // admin     1234           // admin     12345
// admin     54321       // admin     123456         // admin     7ujMko0admin
// admin     1234        // admin     pass           // admin     meinsm
// tech      tech
```

Dyn DoS Attack

- Oct 2016, Much of Eastern US Internet
- Mirai botnet
- IoT
  - 600,000 devices
- Two previous attacks
  - OVH (1.1 Tb/s)
  - Krebs on Security (623 Gb/s)
    - Akamai dropped site

https://en.wikipedia.org/wiki/2016_

https://minecraft.net/en-us/store/

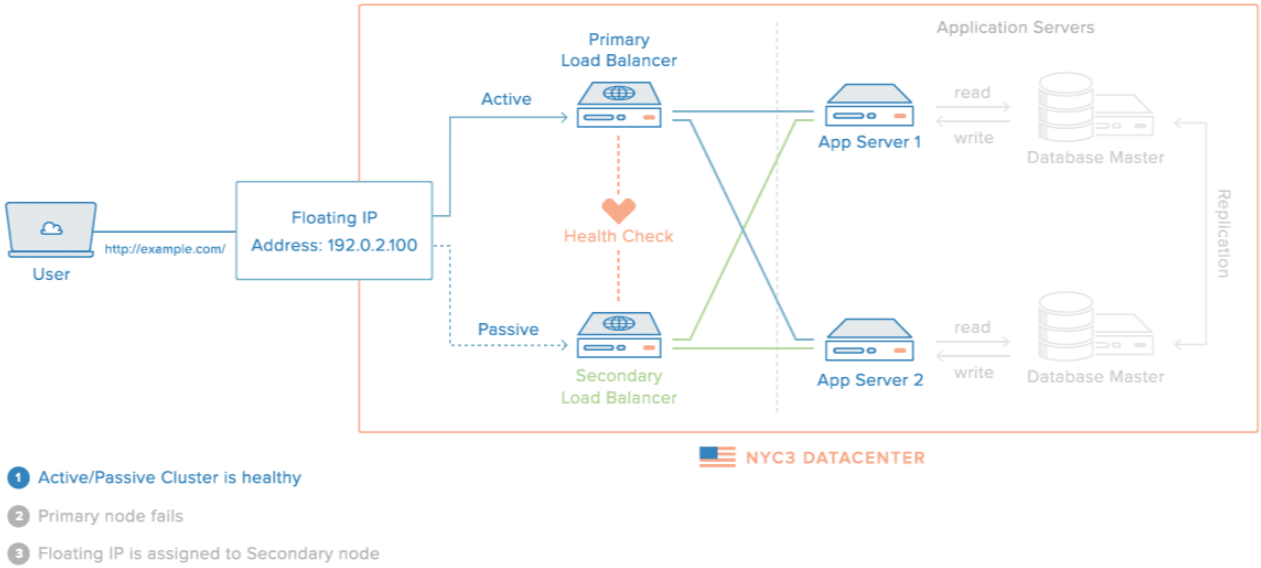
Session affinity / Sticky Sessions

- Sticky LBs
  - always routing a client's requests to the same server instance
  - IP Hash
- Cookies managed by load balancer (duration based)
- Cookies managed by application cookie
- Session Replication - Session-aware clusters
  - Shared session DB or cache
  - Eg. Redis shared
- Session stored with client
  - Exposes application internals

LB Algorithms

- Even Task Distribution / Round Robin
  - Sequential distribution of work among server
  - ignores the difference in the work required
- Weighted Round Robin
- Least Connections

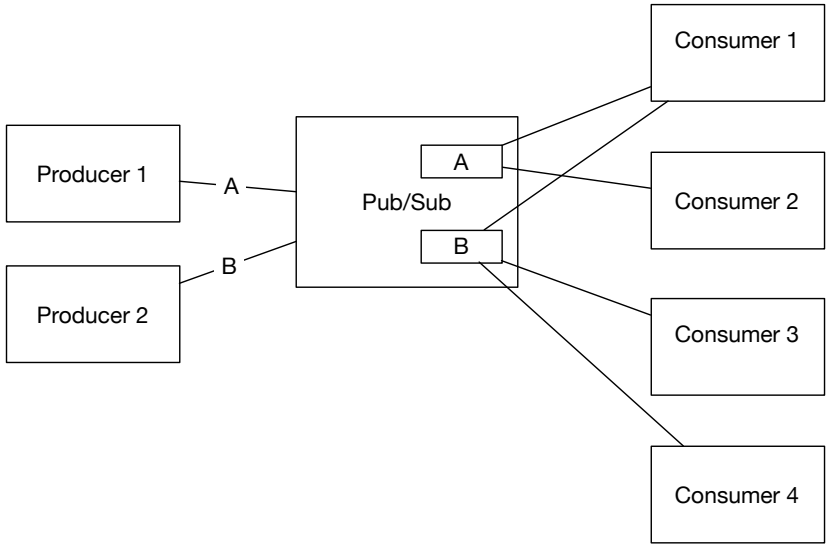
HA LBs



https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

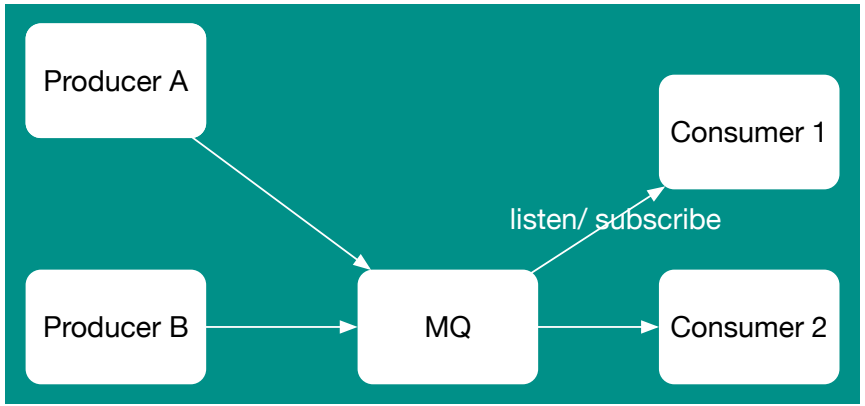
Message Topics

- Messages are immediately pushed to subscribers
- Decouple producers from subscribers
- Concurrent processing
- Independent scalability of components
- Push, One-to-Many distribution

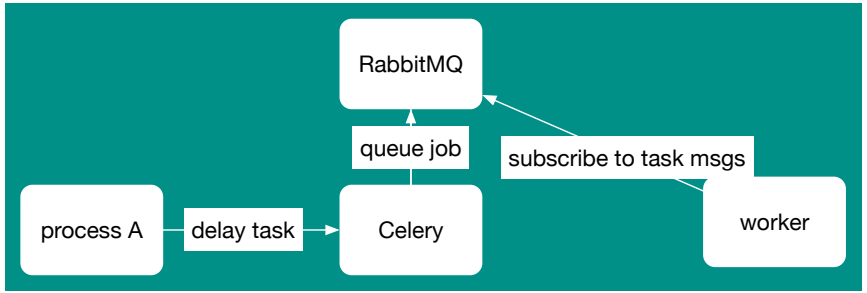


Message Queues

- Asynchronous: Queue it now, run it later. Until a consumer is ready to process
- Decoupling: Separates application logic.
- Scalable: Many workers can process individual jobs in a queue.
- Introduce Latency



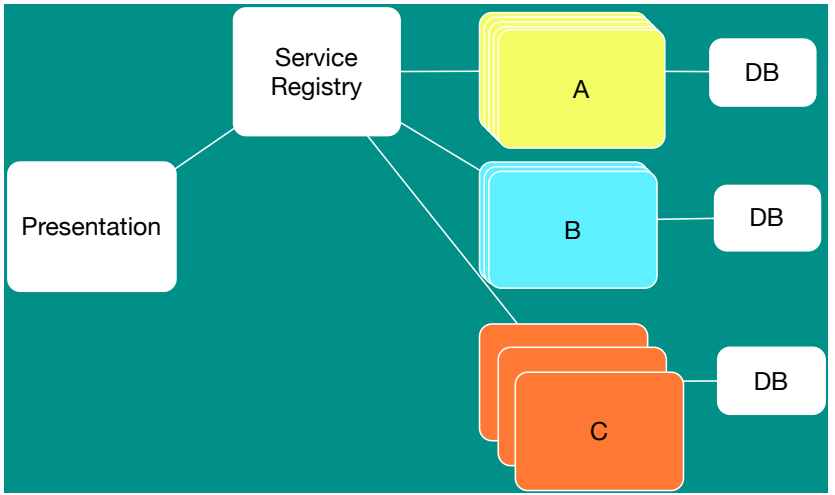
Example: Celery



{"id": "4cc7438e-afd4-4f8f-a2f3-f46567e7ca77", ”
task": "celery.task.PingTask",
"args": [],
"kwargs": {}, "retries": **0**,
"eta": "2009-11-17T12:30:56.527191"}

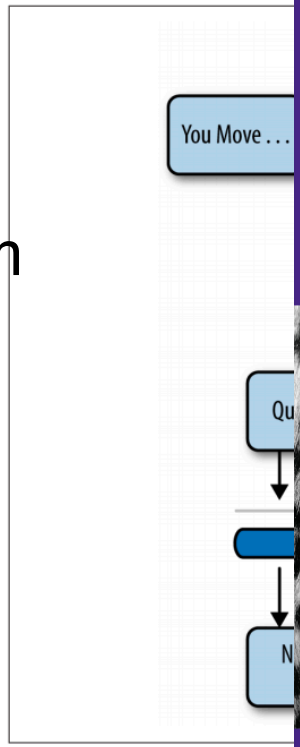
Service Registries

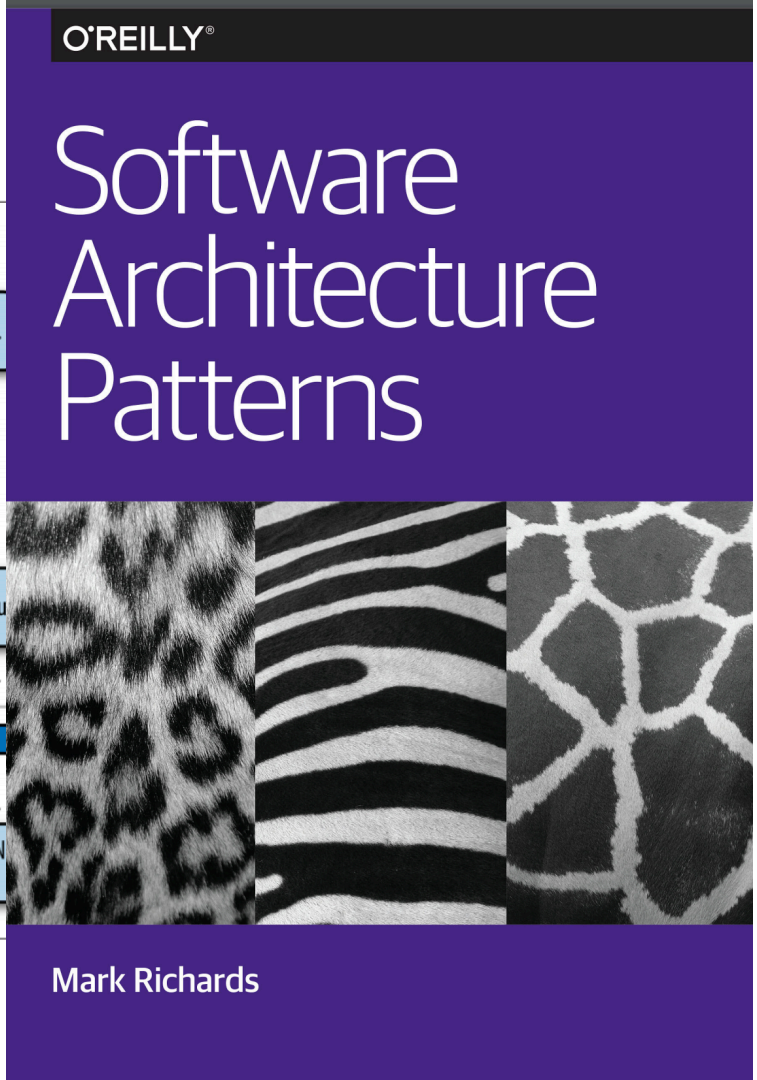
- Resolve Addresses for Names
- Based on HA, transactional data stores
- Examples: Apache Zookeeper, Eureka



Example:

- Customer Processes writes to Pub/Sub system
- Quote Process and Claim Process act concurrently





Automation

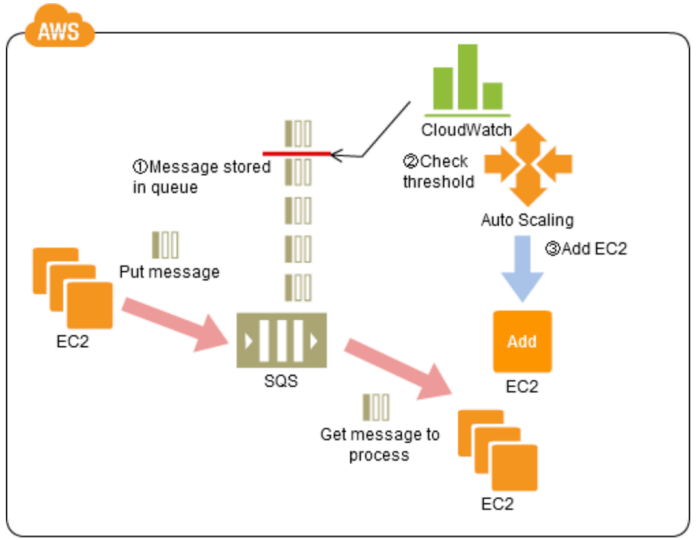
Scaling instances in your Architecture cannot be manual

- Elasticity controller
- Metrics- CPU, Mem, Disk
- When exceeded -> start new nodes
- When underused -> stop nodes

Example: Job Observer Pattern

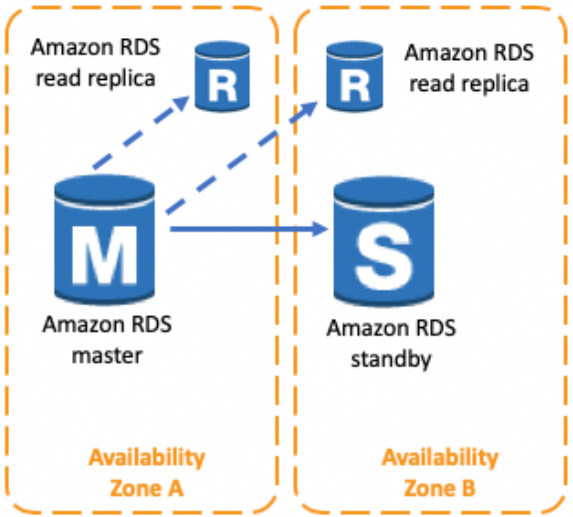
http://en.clouddesignpattern.org/index.php/CDP:Job_Observer_Pattern

- Queue Scaling
- Watch a job queue and scale depending on that



Database Scaling

- Read-replicas
- Database sharding
  - Postgreslq 11 with FDW



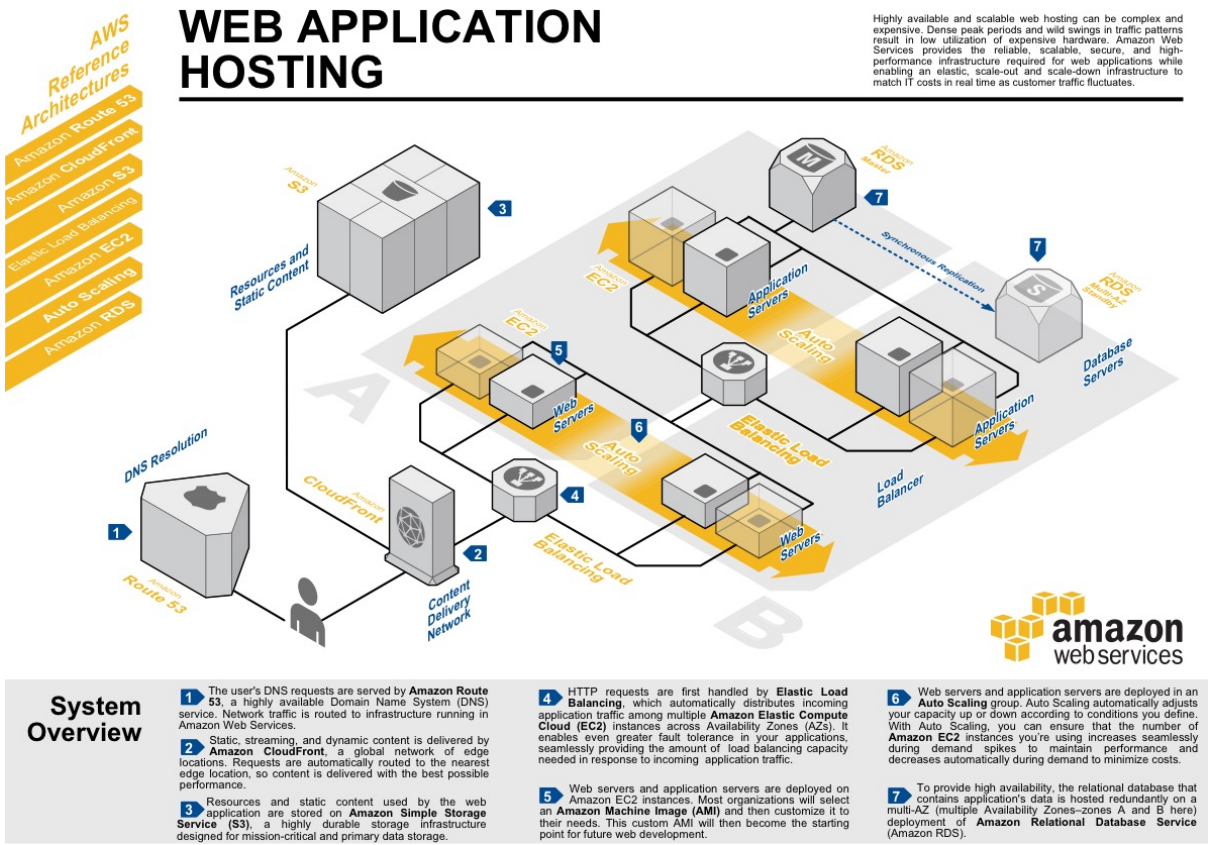

```
CREA                                                    temp integer,
maxte

CREA                                                    temperatures
FOR V                                                   );
CREA                                                    temperatures
FOR V                                                   );
```

```
CREATE SERVER box2 FOREIGN DATA WRAPPER p
dbname 'box2db');
```

```
CREATE FOREIGN TABLE temperatures_2016 PARTITION OF temperatures FOR VALUES FROM ('2016-01-
01') TO ('2017-01-01') SERVER box2;
```

Architectural Patterns



Thursday

- More on Microservices
- Monitoring and Tracing