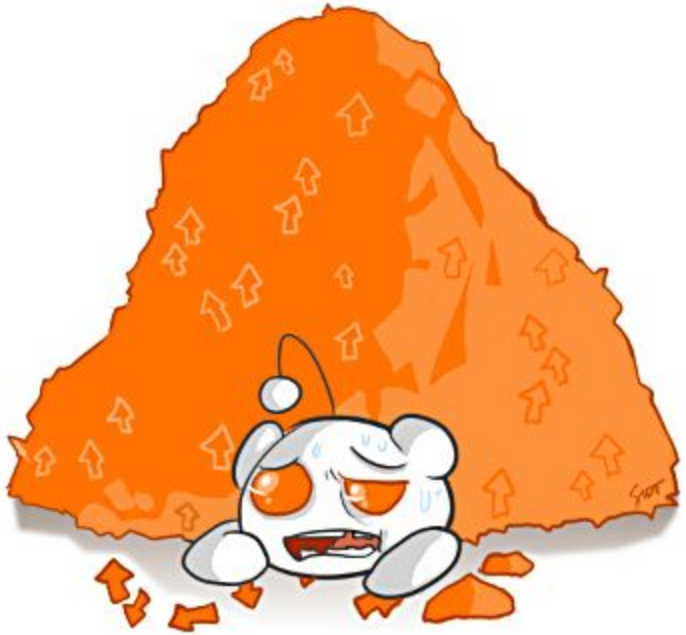
Reddit's Architecture

And how it's broken over the years

Neil Williams
QCon SF, 13 November 2017

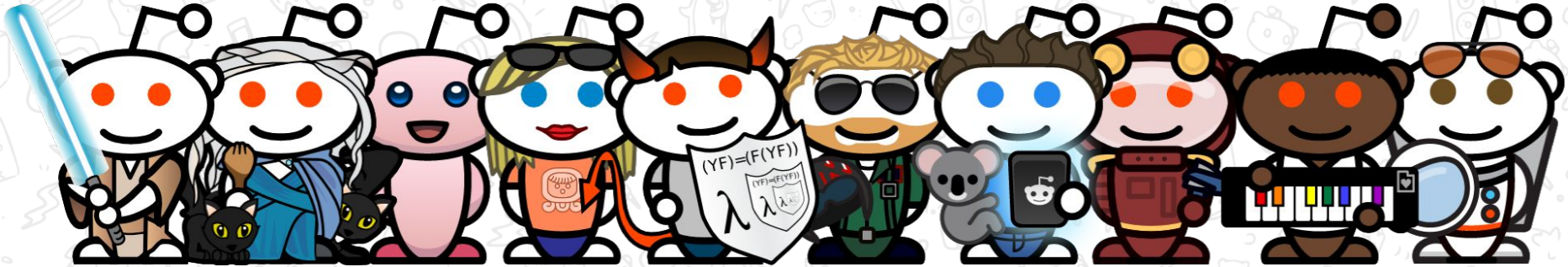


What is Reddit?

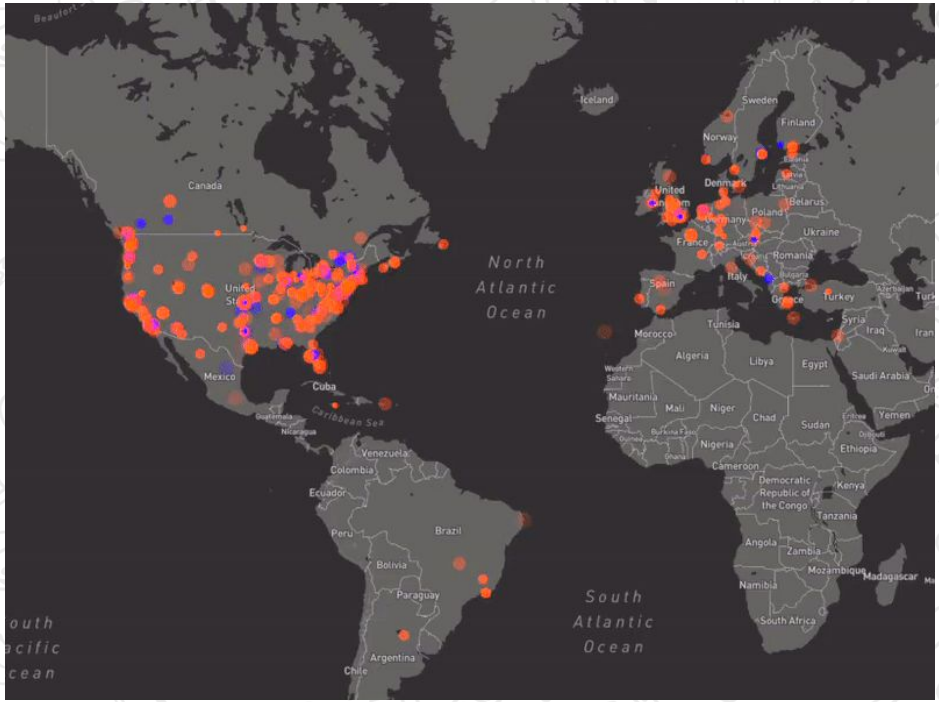
Reddit is the **frontpage** of the internet

A social network where there are tens of thousands of **communities** around whatever **passions or interests** you might have

It’s where people **converse** about the things that are most important to them



Reddit by the numbers



4th/7th Alexa Rank (US/World)

320M MAU

1.1M Communities

1M Posts per day

5M Comments day

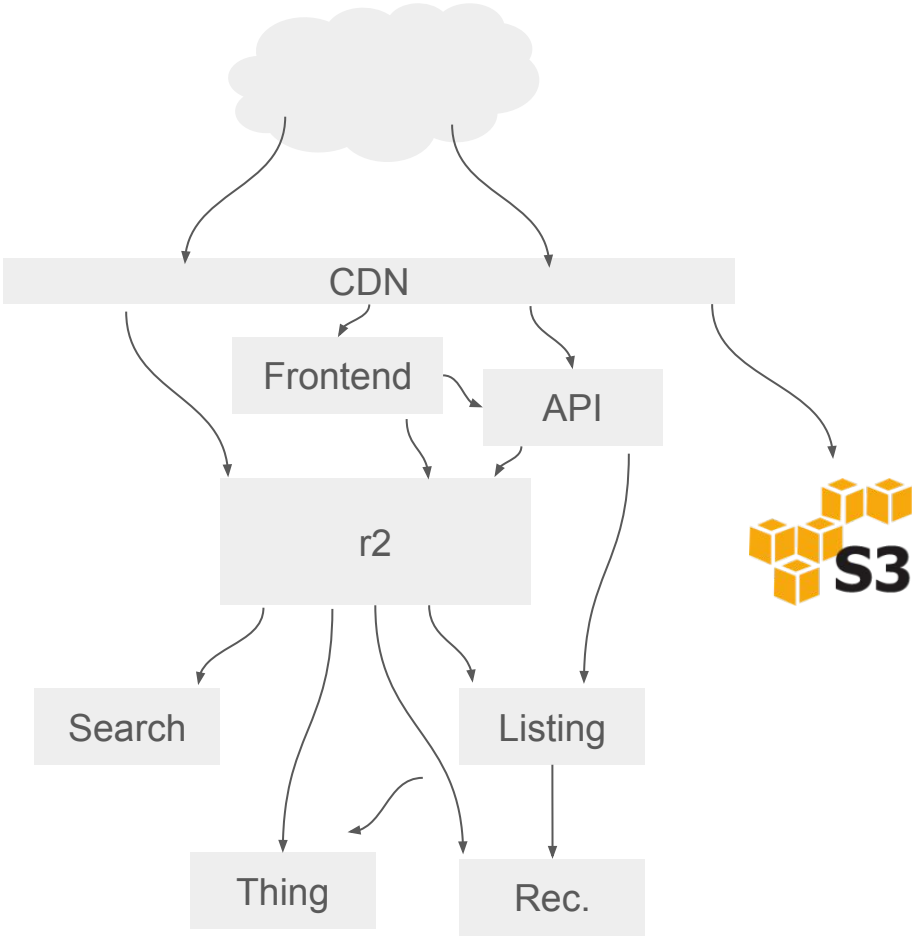
75M Votes per day

70M Searches per Day

Major components

The stack that serves reddit.com.

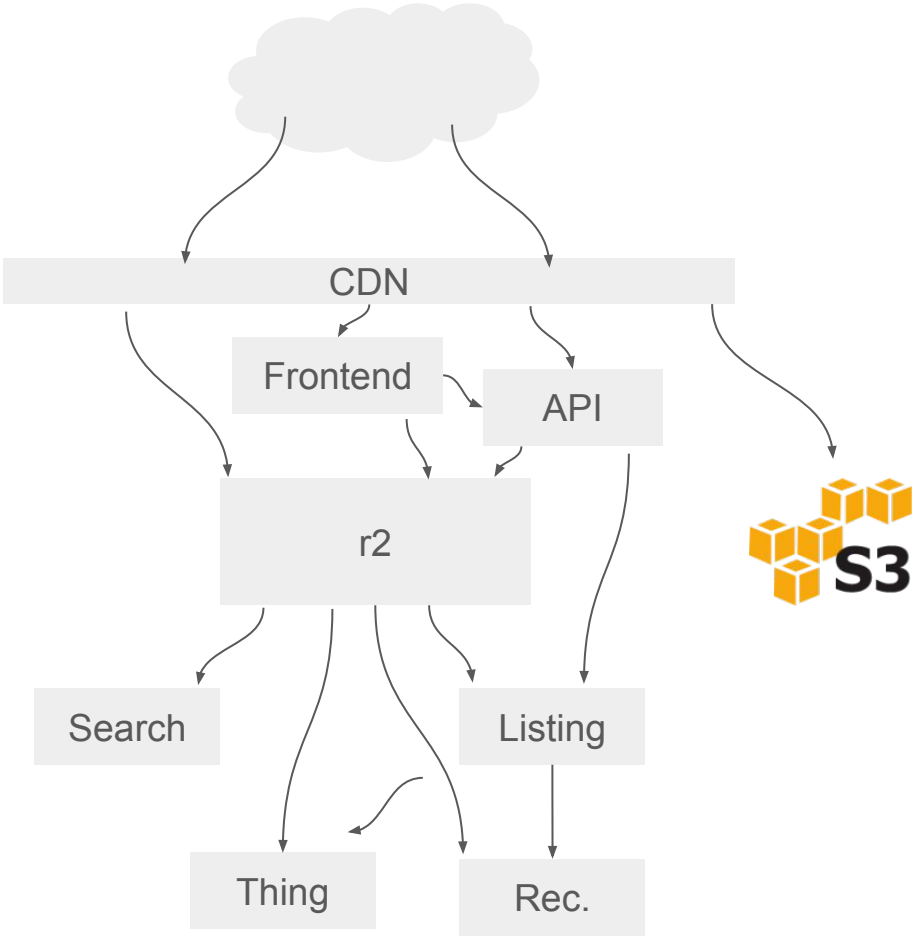
Focusing on just the core experience.



Major components

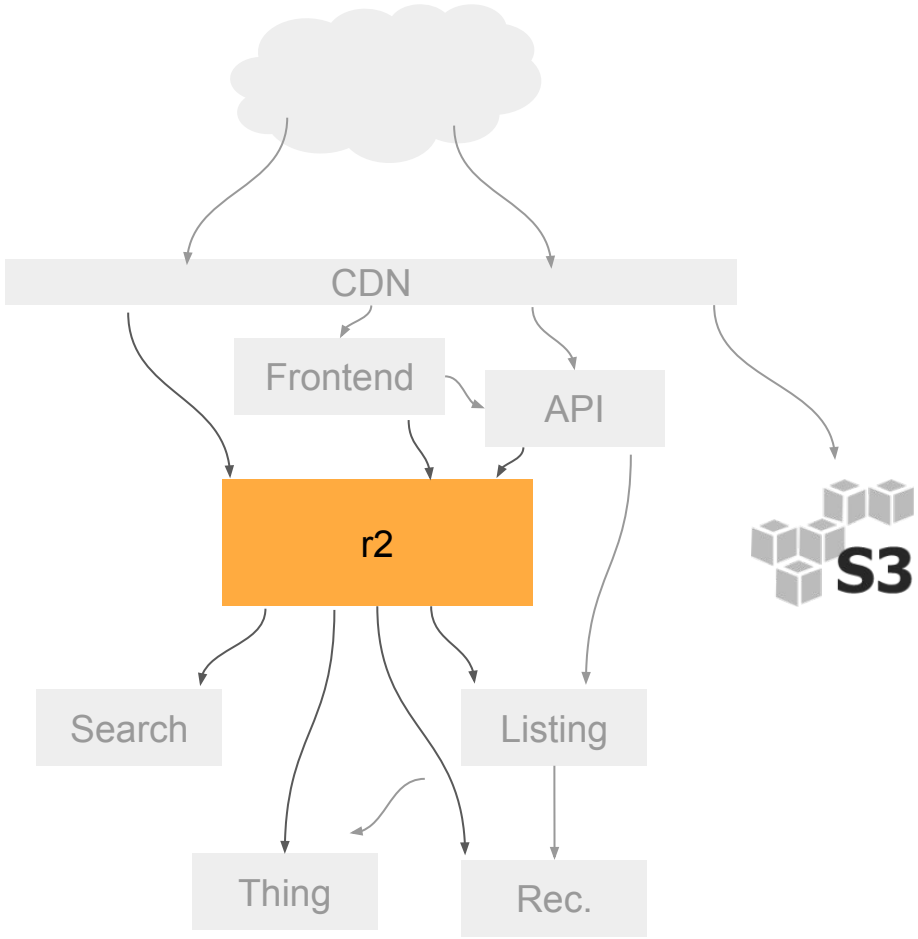
A work in progress.

This tells you as much about the organization as it does our tech.



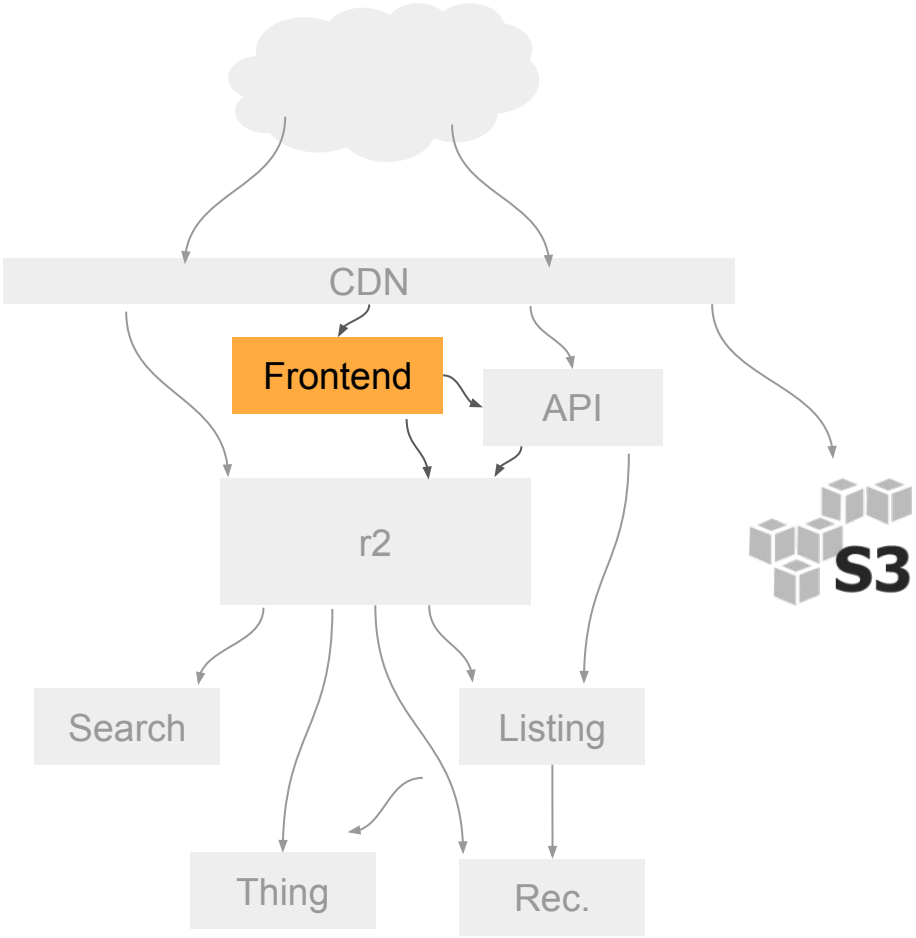
r2: The monolith

The oldest single component of Reddit, started in 2008, and written in Python.



Node.js frontend applications

Modern frontends using shared server/client code.



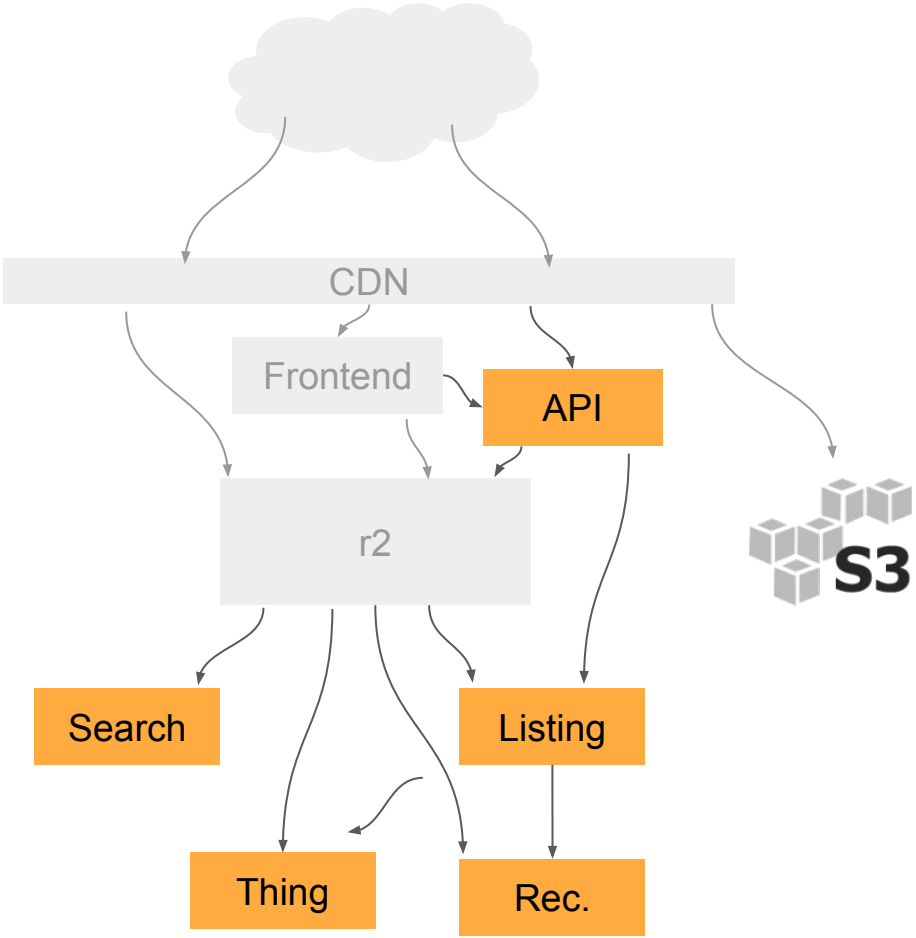
New backend services

Written in Python.

Splitting off from r2.

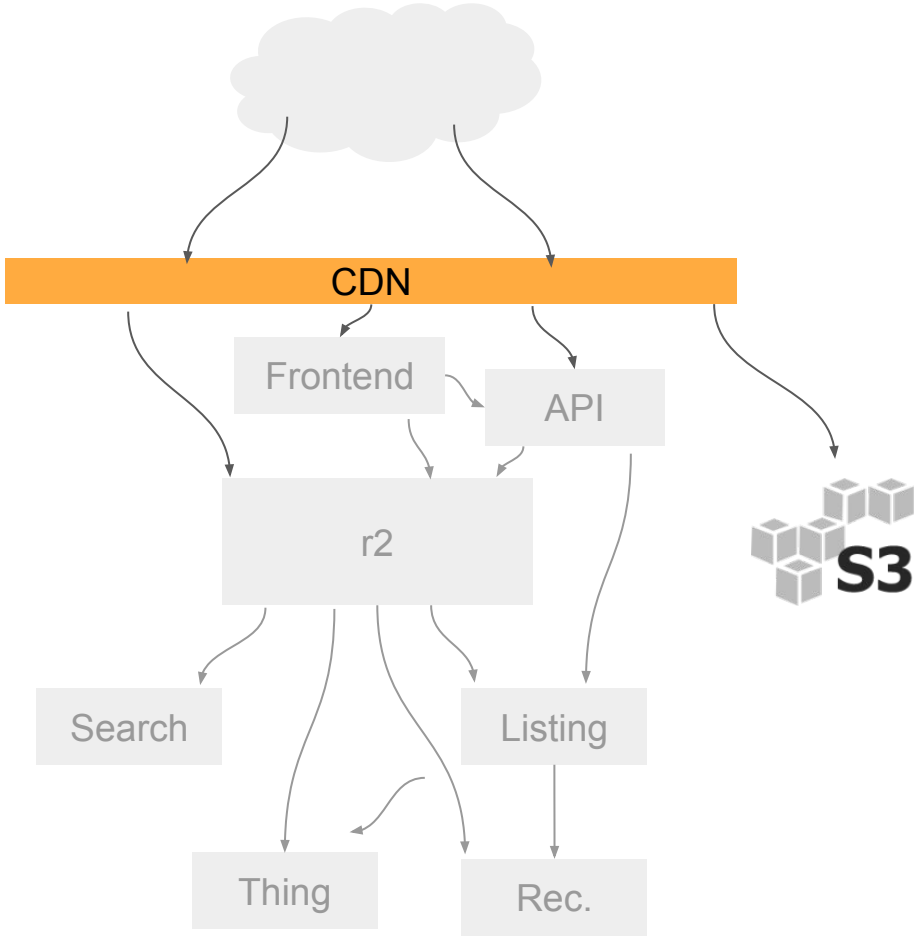
Common library/framework to standardize.

Thrift or HTTP depending on clients.



CDN

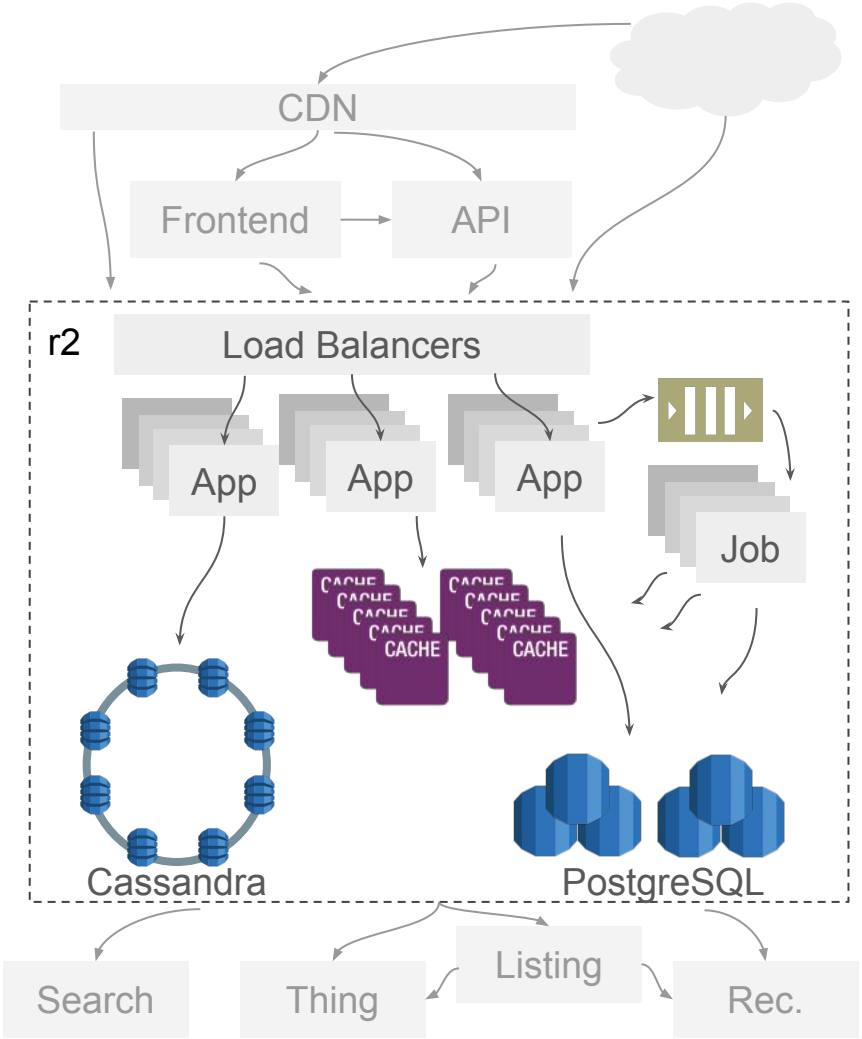
Send requests to distinct stacks depending on domain, path, cookies, etc.



r2 Deep Dive

The original Reddit.

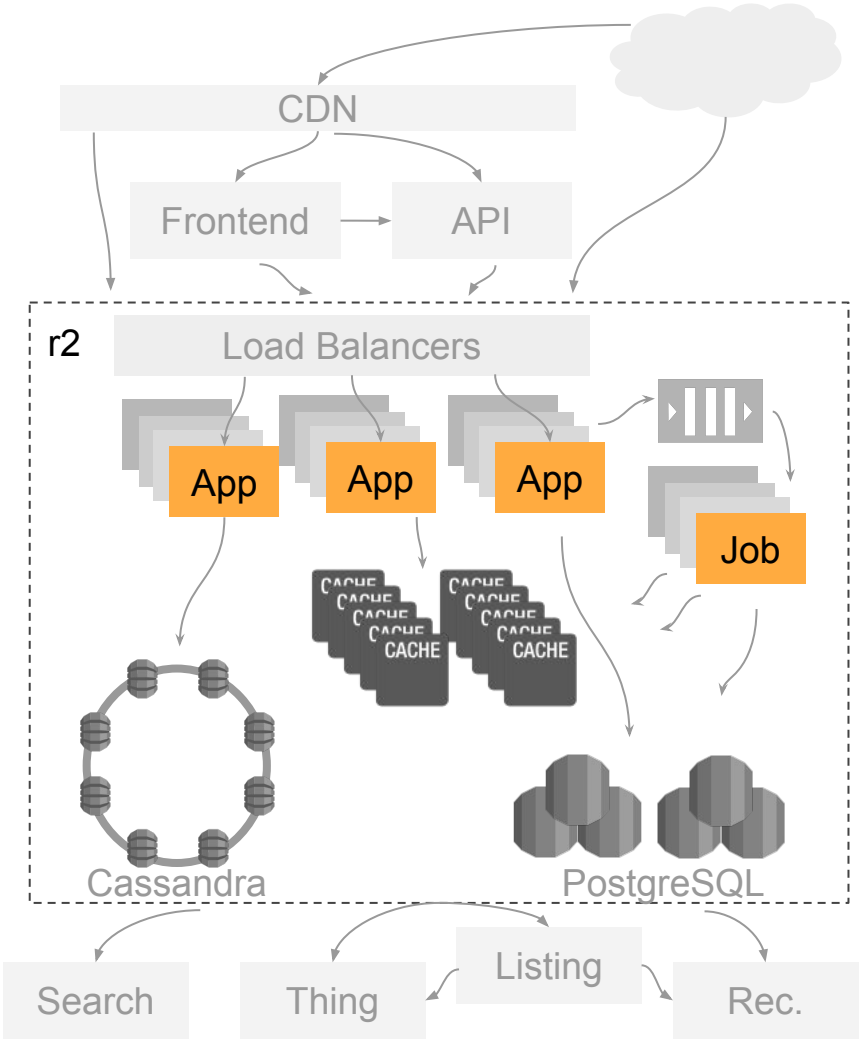
Much more complex than any of the other components.



r2: Monolith

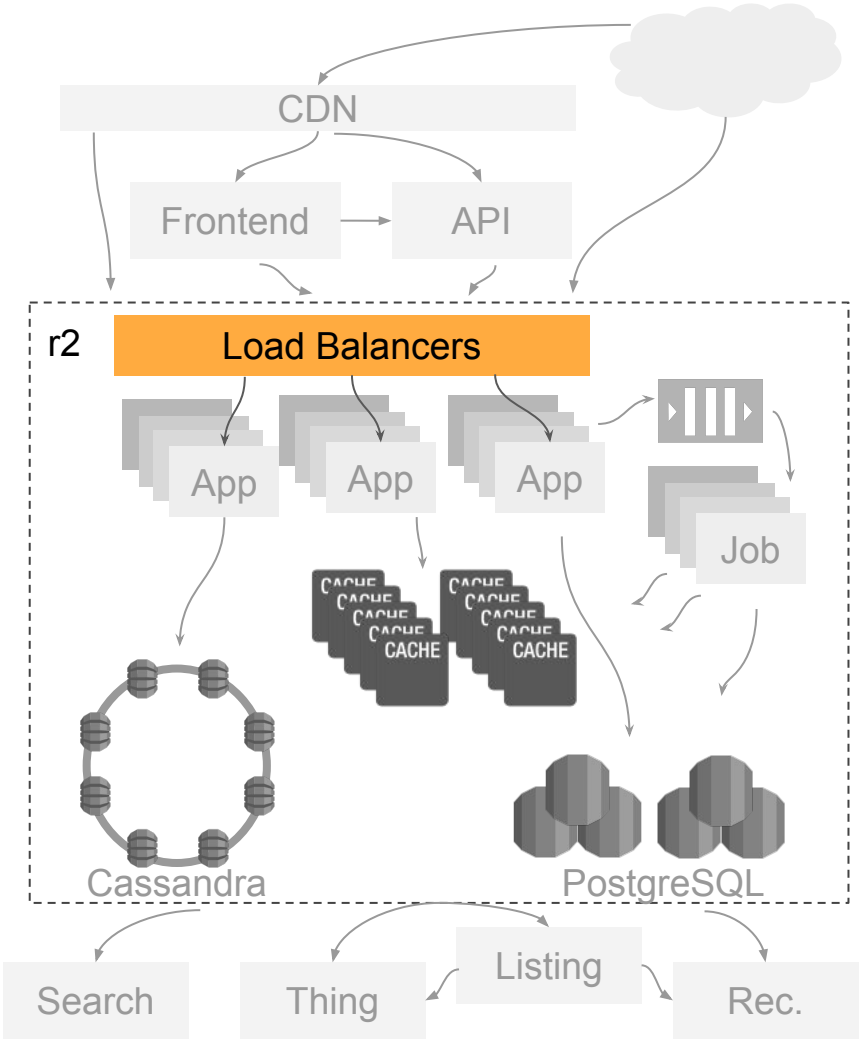
Monolithic Python application.

Same code deployed to all servers,
but servers used for different tasks.



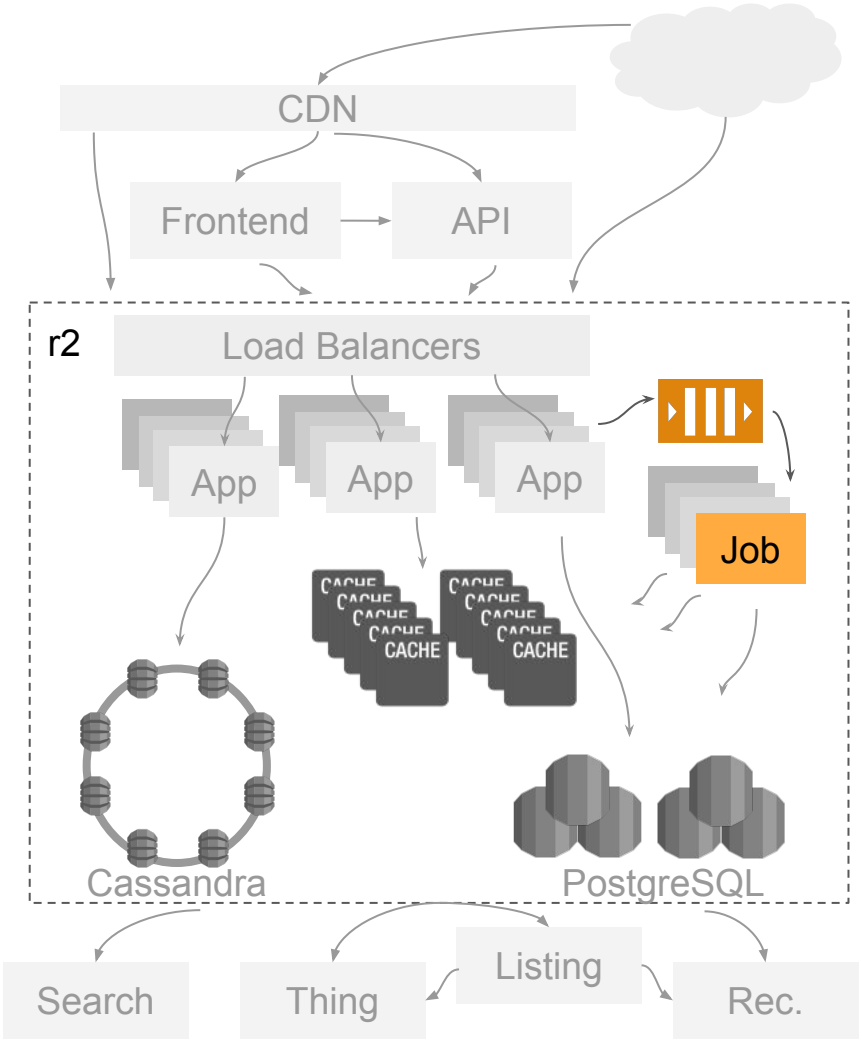
r2: Load Balancers

Load balancers route requests to distinct “pools” of otherwise identical servers.



r2: Job Queues

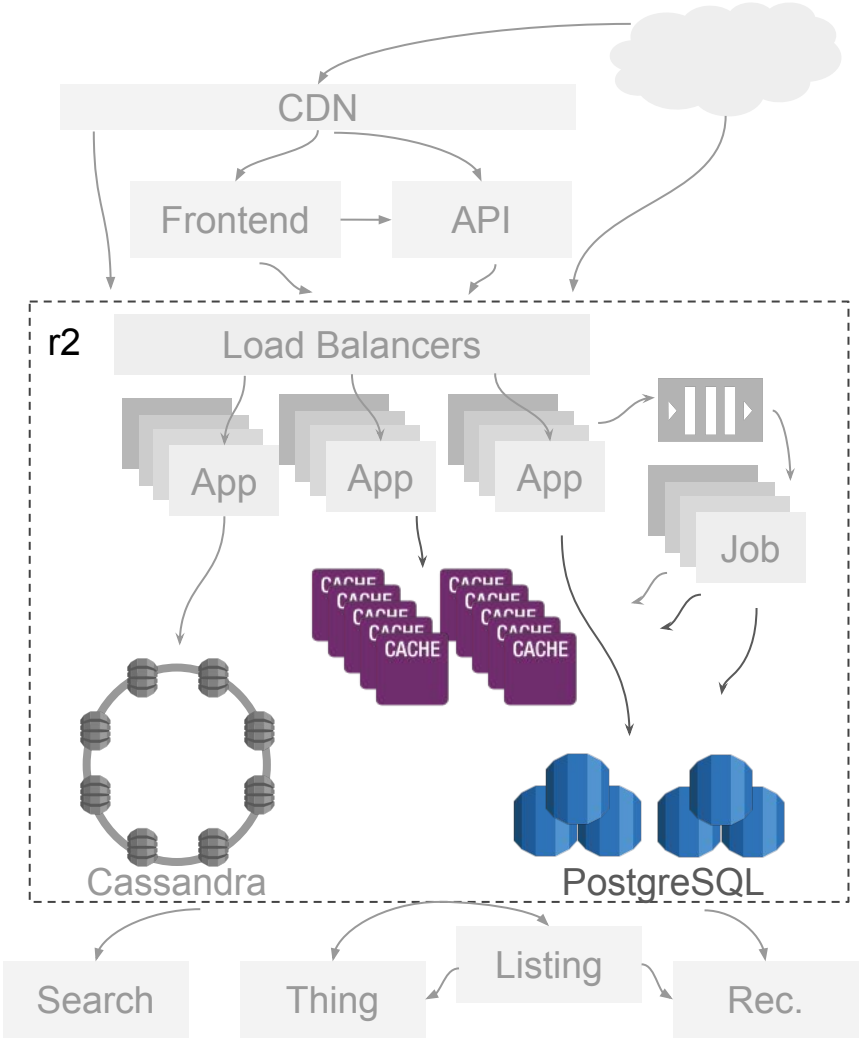
Many requests trigger asynchronous jobs that are handled in dedicated processes.



r2: Things

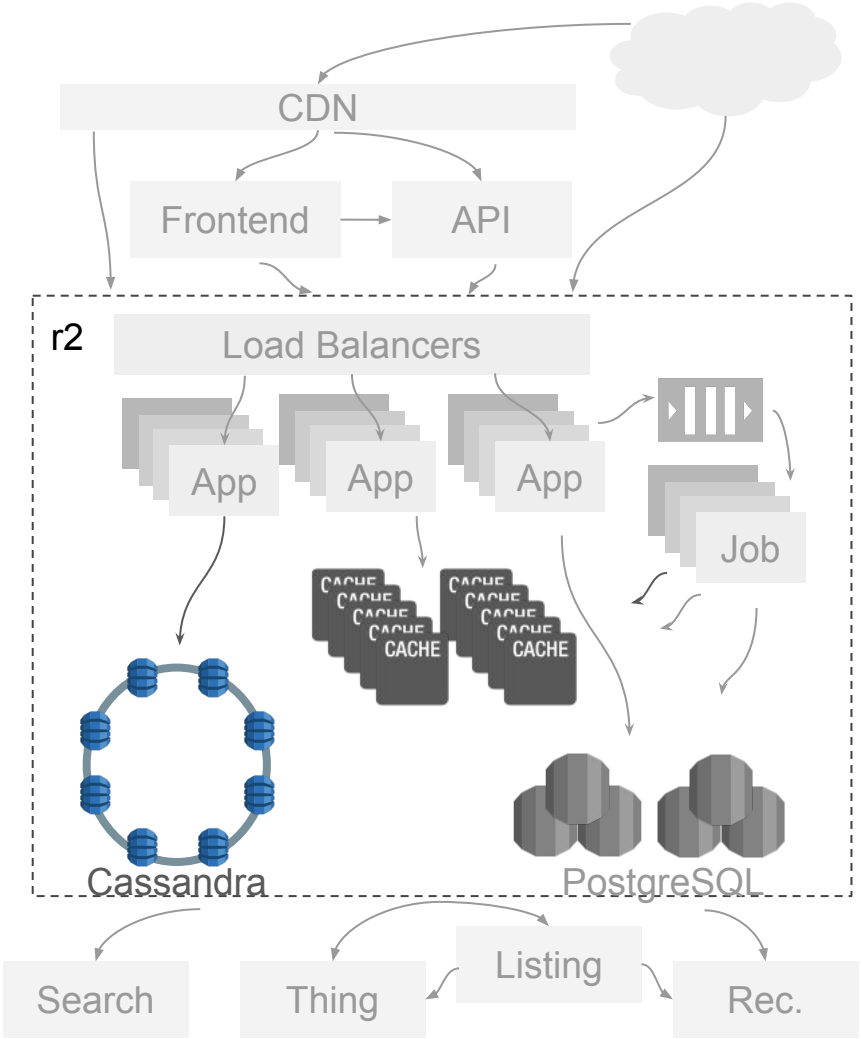
Many core data types are stored in the **Thing** data model.

This uses PostgreSQL for persistence and memcached for read performance.



r2: Cassandra

Apache Cassandra is used for most newer features and data with heavy write rates.

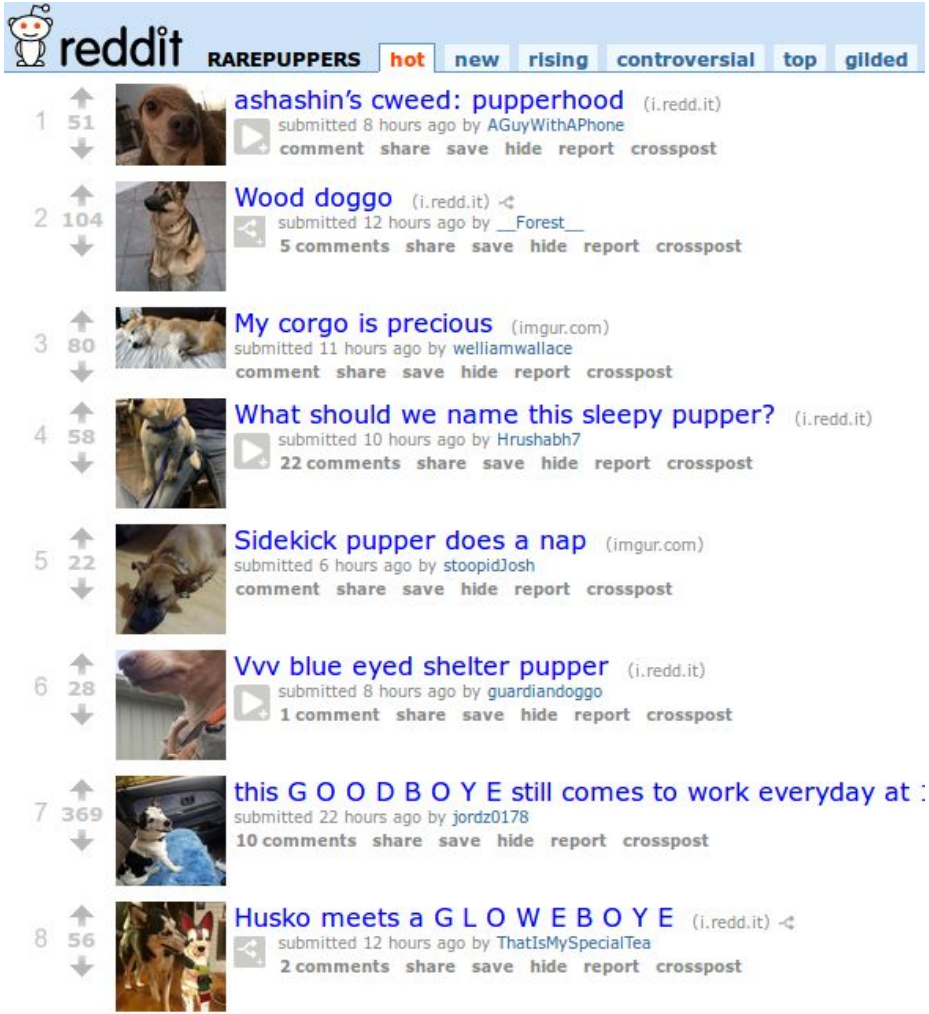


Listings

Listings

The foundation of Reddit: an ordered list of links.

Naively computed with a SQL query:
SELECT * FROM links ORDER BY hot(ups, downs);

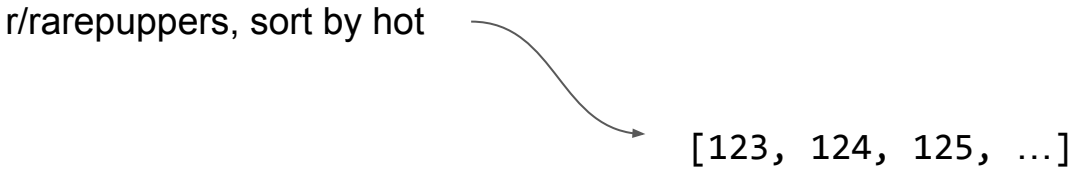


Cached Results

Rather than querying every time, we cache the list of Link IDs.

Just run the query and cache the results.

Invalidate cache on new submissions and votes.



Cached Results

Easy to look up the links by ID once listing is fetched.

r/rarepuppers, sort by hot

`[123, 124, 125, ...]`

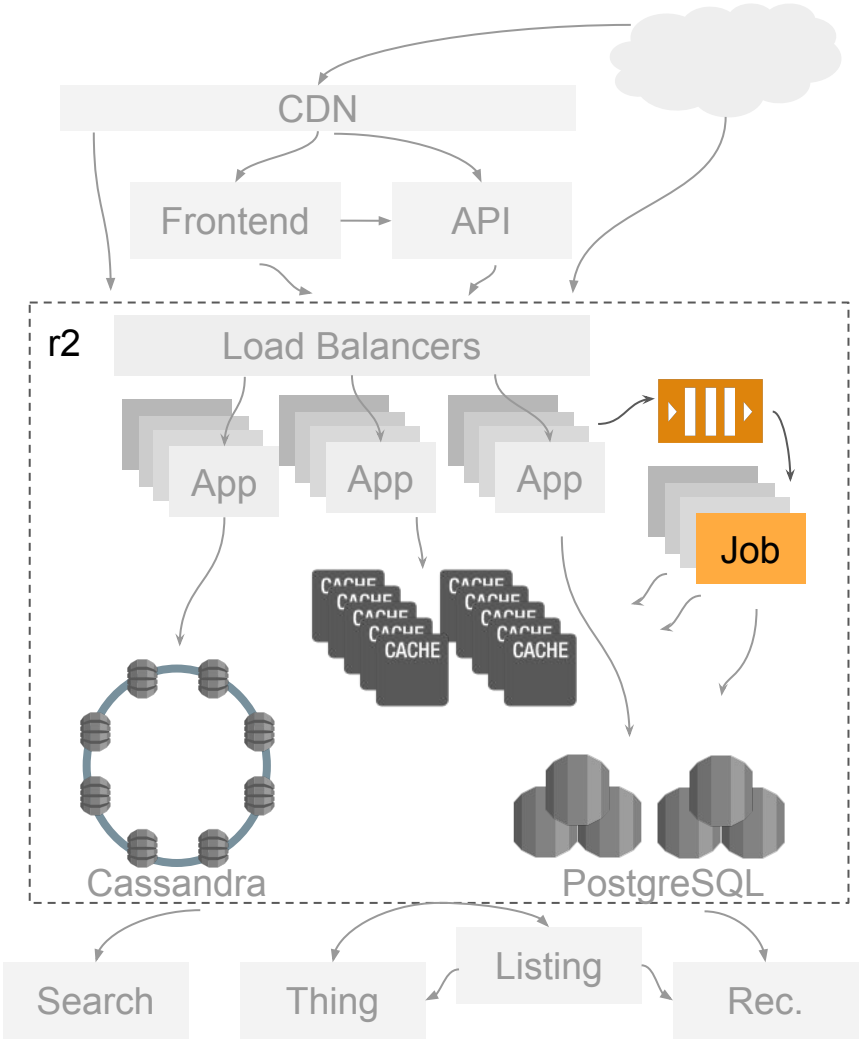
Link #123: title=doggo
Link #124: title=pupper does a nap

Vote Queues

Votes invalidate a lot of cached queries.

Also have to do expensive anti-cheat processing.

Deferred to offline job queues with many processors.



Mutate in place

Eventually, even running the query is too slow for how quickly things change.

Add sort info to cache and modify the cached results in place.

Locking required.

```
[(123, 10), (124, 8), (125, 8), ...]
```

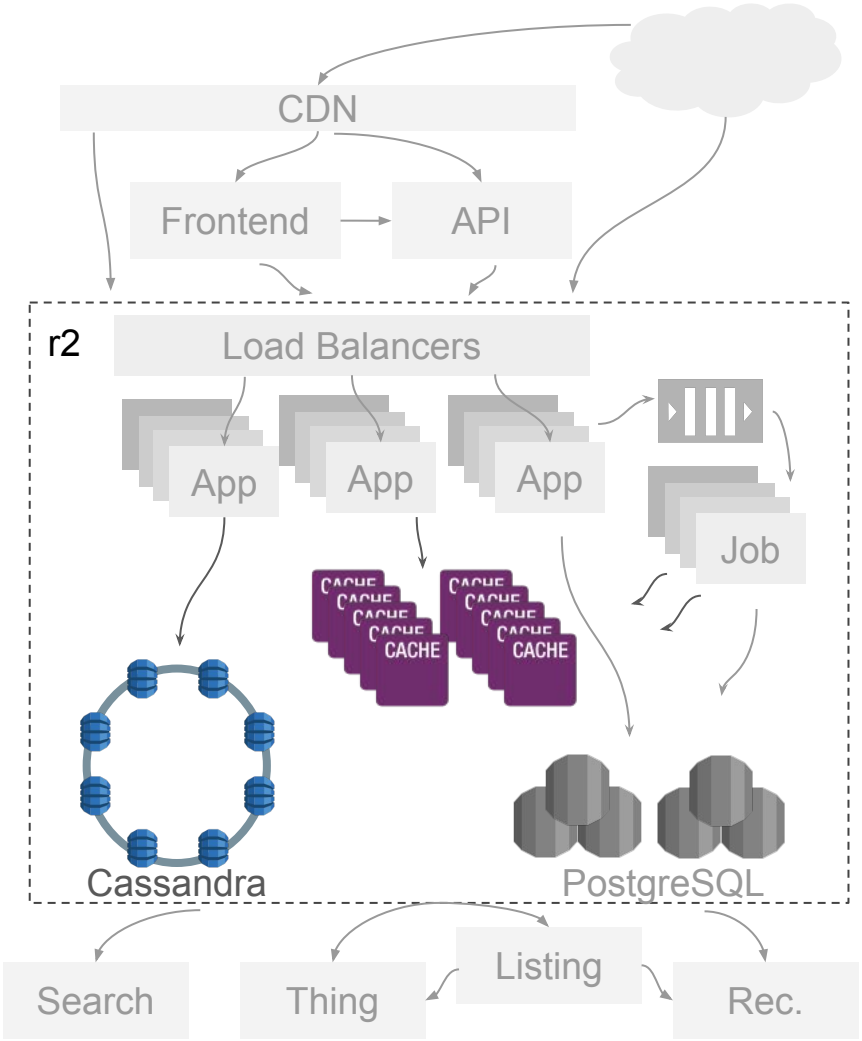
Link #125

```
[(123, 10), (125, 9), (124, 8), ...]
```

“Cache”

This isn’t really a cache anymore:
really a denormalized index of links.

Data is persisted to Cassandra, reads
are still served from memcached.



Vote queue pileups

Mid 2012

We started seeing vote queues fill up at peak traffic hours.

A given vote would wait in queue for hours before being processed. Delayed effects on site noticeable by users.

https://commons.wikimedia.org/wiki/File:Miami_traffic_jam,_I-95_North_rush_hour.jpg

Scale out?

Adding more consumer processes made the problem *worse*.

Observability

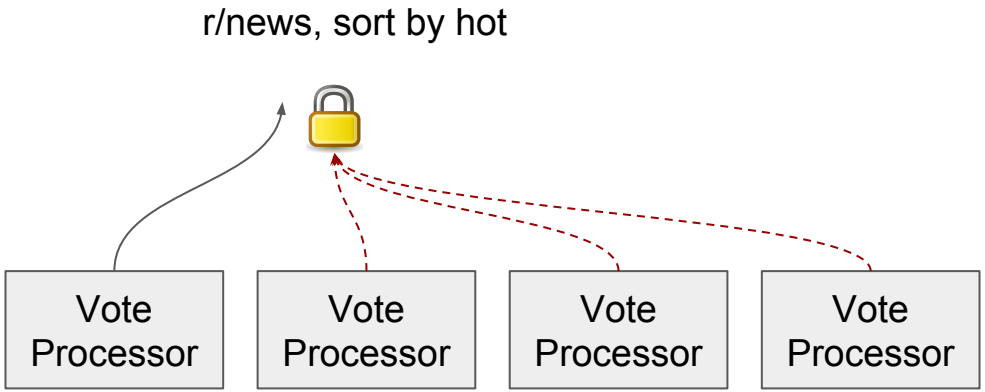
Basic metrics showed average processing time of votes way higher.

No way to dig into anything more granular.

Lock contention

Added timers to various portions of vote processing.

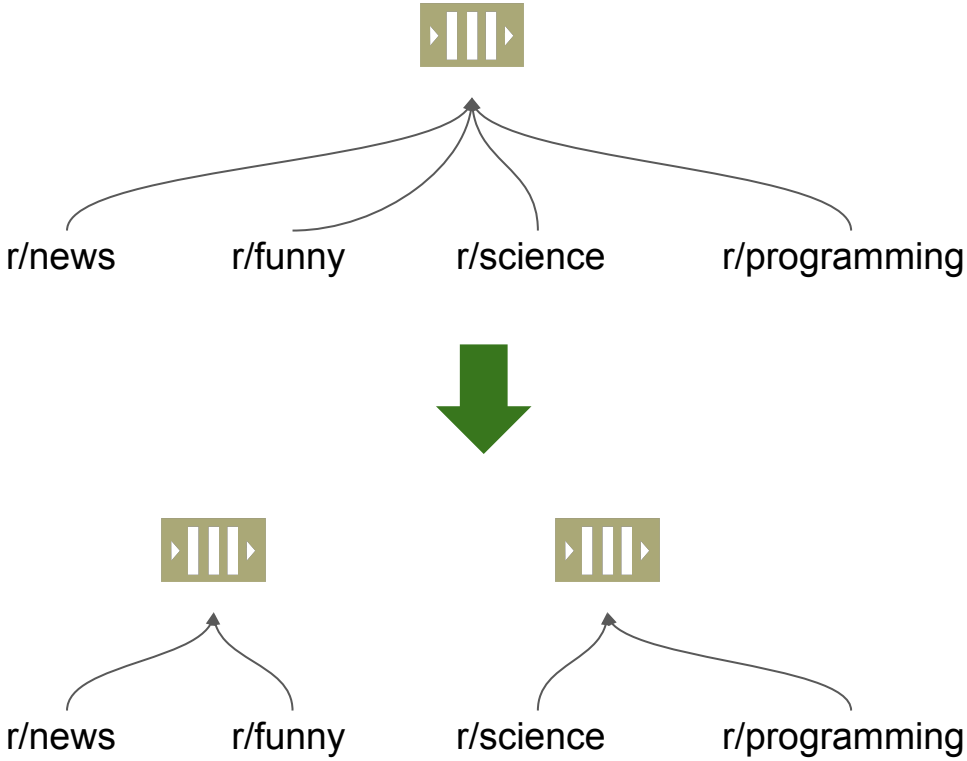
Time spent waiting for the cached query mutation locks was much higher during these pileups.



Partitioning

Put votes into different queues based on the subreddit of the link being voted on.

Fewer processors vying for same locks concurrently.



Smooth sailing!

Slow again

Late 2012

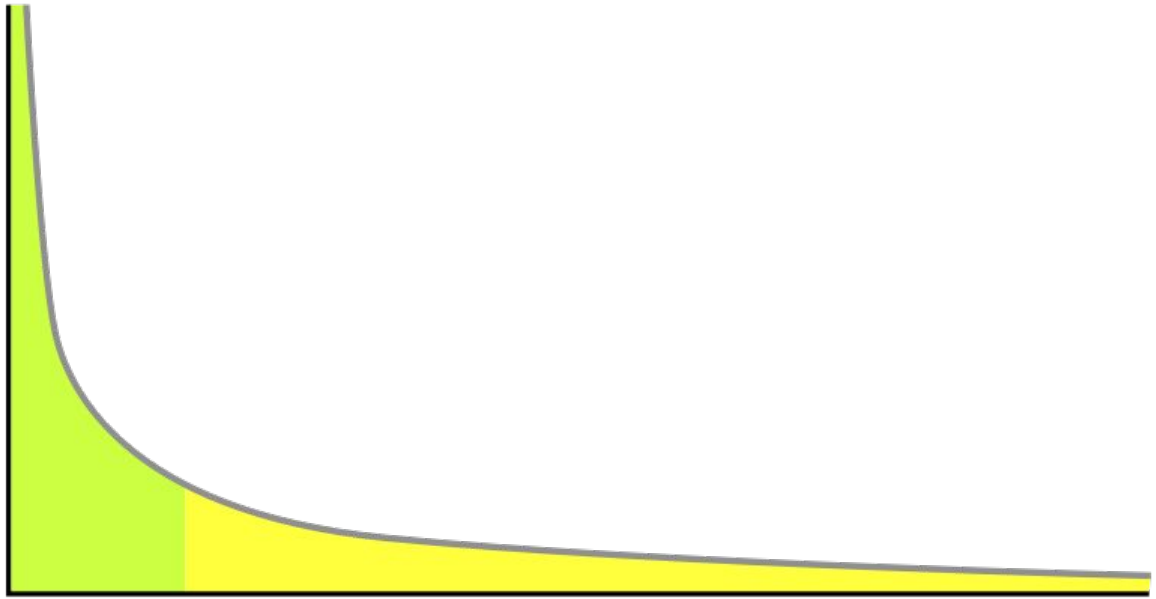
This time, the average lock contention and processing times look fine.

p99

The answer was in the 99th percentile timers.

A subset of votes were performing *very* poorly.

Added print statements to get to the bottom of it.

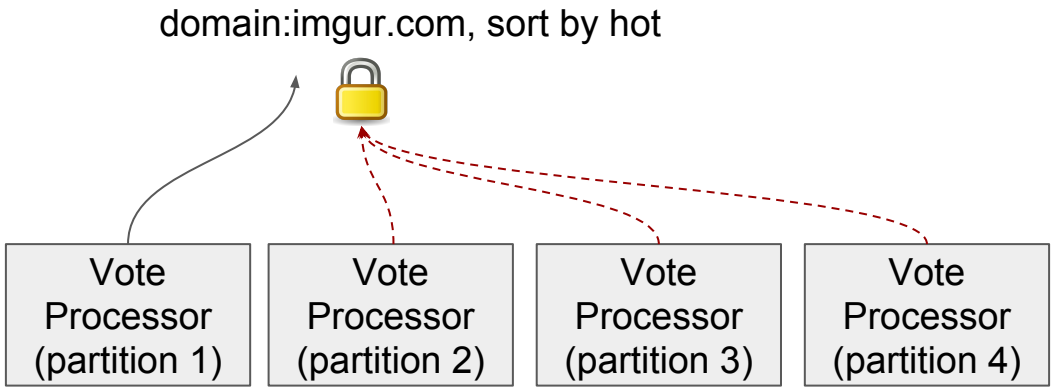


An outlier

Vote processing updated all affected listings.

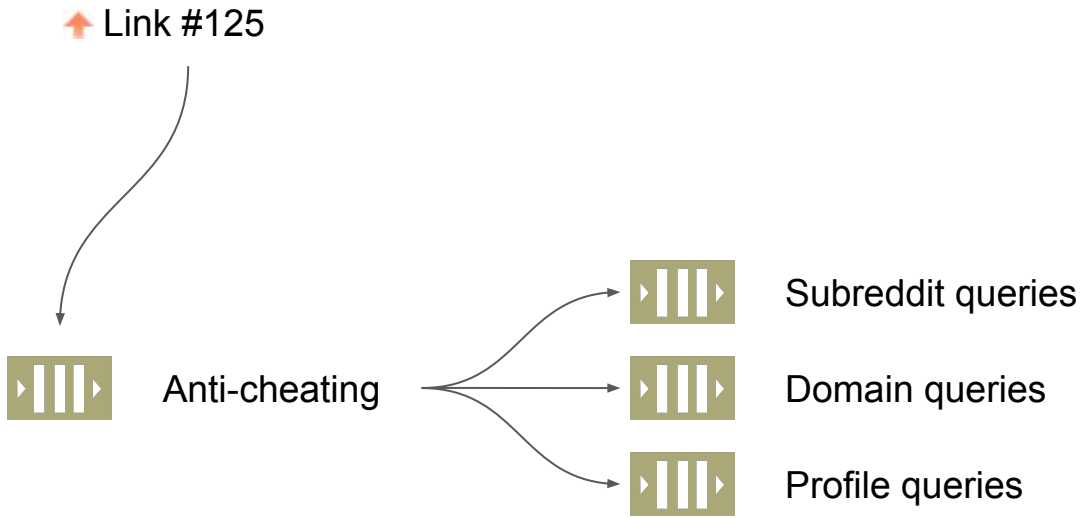
This includes ones not related to subreddit, like the domain of the submitted link.

A very popular domain was being submitted in many subreddits!



Split up processing

Handle different types of queries in different queues so they never work cross-partition.

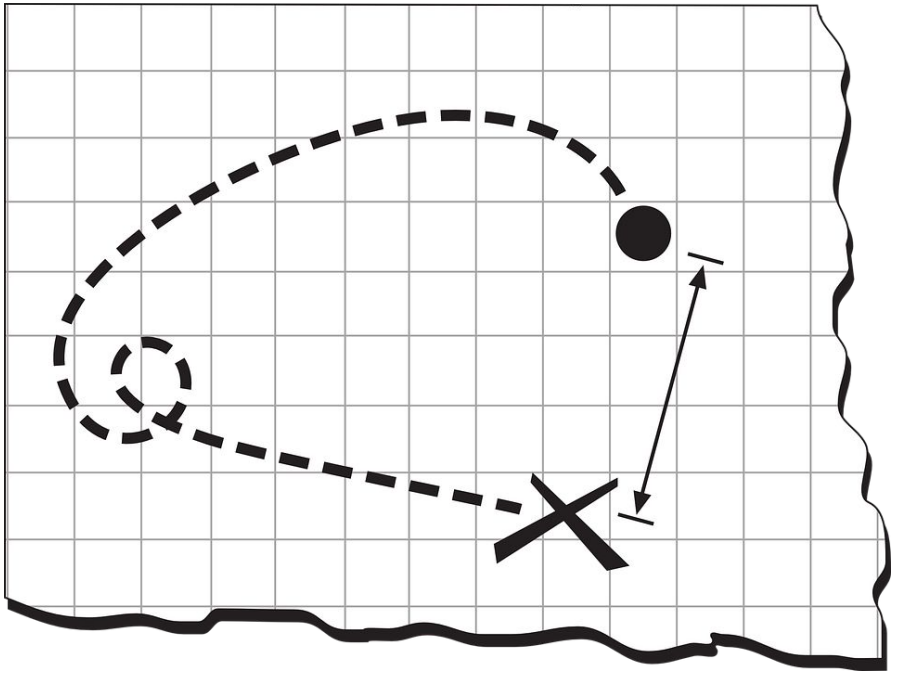


Learnings

Timers give you a cross section.

p99 shows you problem cases.

Have a way to dig into those exceptional cases.



Learnings

Locks are bad news for throughput.

But if you must, use the right partitioning to reduce contention.

Lockless cached queries

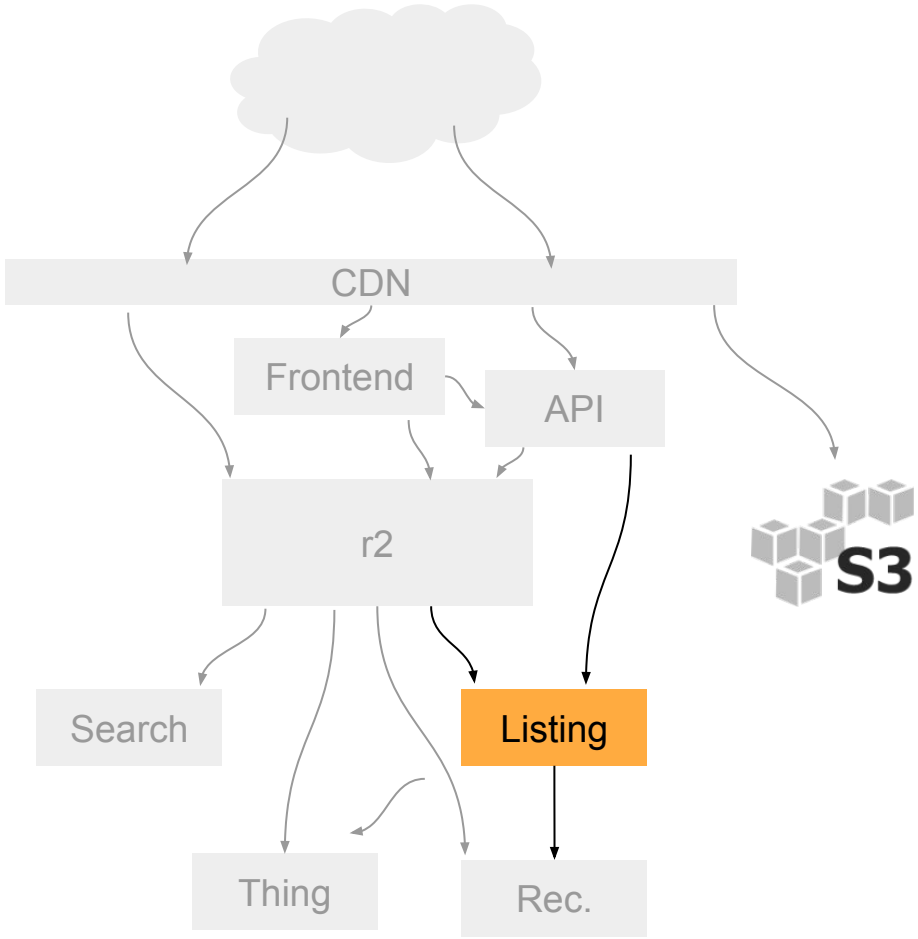
New data model we’re trying out which allows mutations without locking.

More testing needed.

The future of listings

Listing service: extract the basics and rethink how we make listings.

Use machine learning and offline analysis to build up more personalized listings.



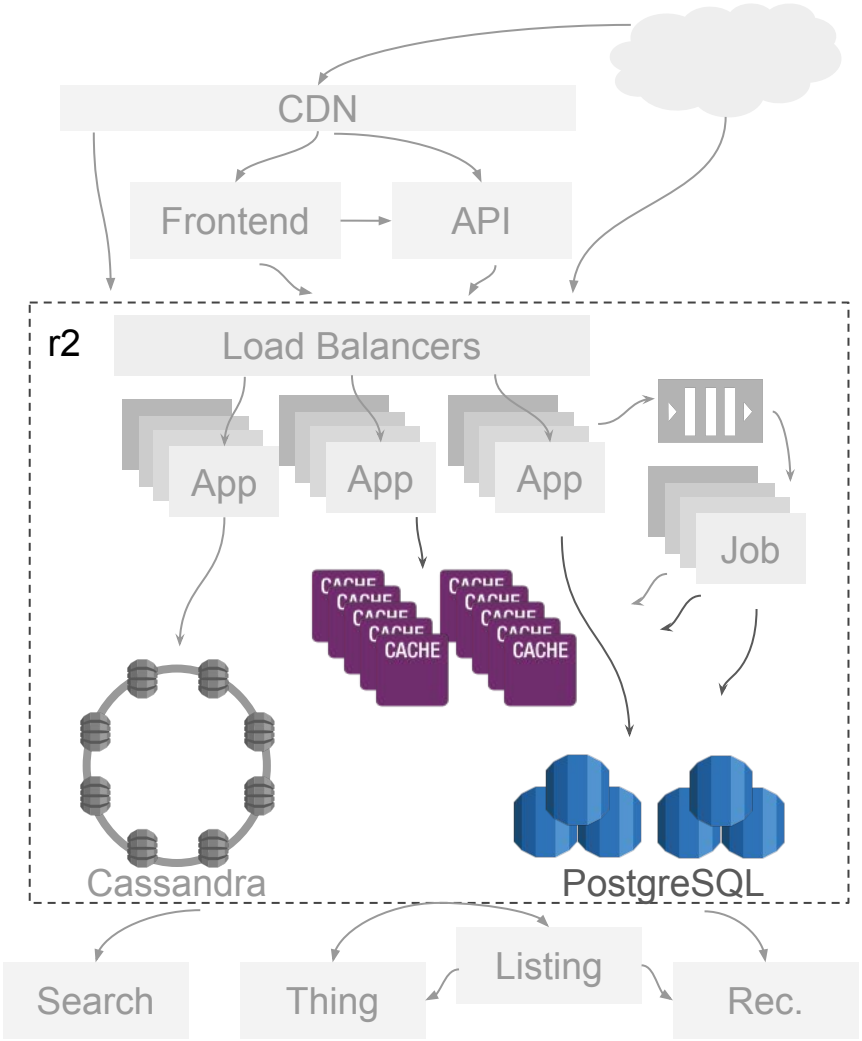
Things

Thing

r2's oldest data model.

Stores data in PostgreSQL with heavy caching in memcached.

Designed to allow extension within a safety net.



Tables

One Thing type per “noun” on the site.

Each Thing type is represented by a pair of tables in PostgreSQL.

Thing

Each row in the *thing* table represents one Thing instance.

The columns in the *thing* table are everything needed for sorting and filtering in early Reddit.

reddit_thing_link

id	ups	downs	deleted
1	1	0	f
2	99	10	t
3	345	3	f

Thing

Many rows in the *data* table will correspond to a single instance of a Thing.

These make up a key/value bag of properties of the thing.

reddit_data_link

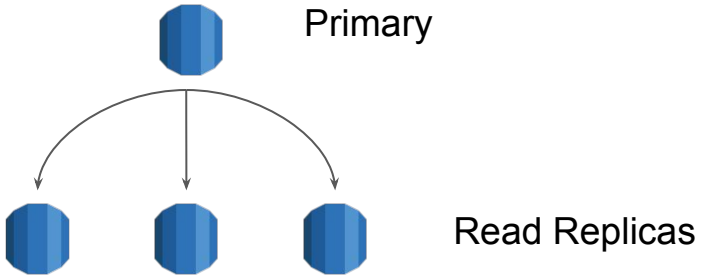
```
thing_id |   key |   value
---------+-------+--------
       1 | title | DAE think
       1 |   url | http://...
       2 | title | Cat
       2 |   url | http://...
       3 | title | Dog!
       3 |   url | http://...
```

Thing in PostgreSQL

Each Thing lives in a database cluster.

Primary that handles writes. A number of read-only replicas.

Asynchronous replication.

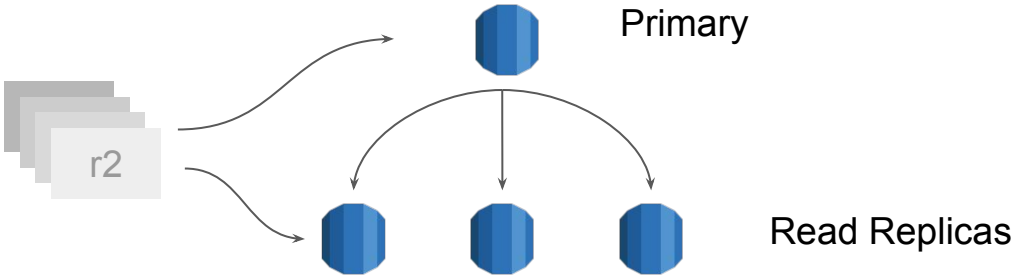


Thing in PostgreSQL

r2 connects directly to databases.

Use replicas to handle reads.

If a database seemed down, remove it from connection pool.

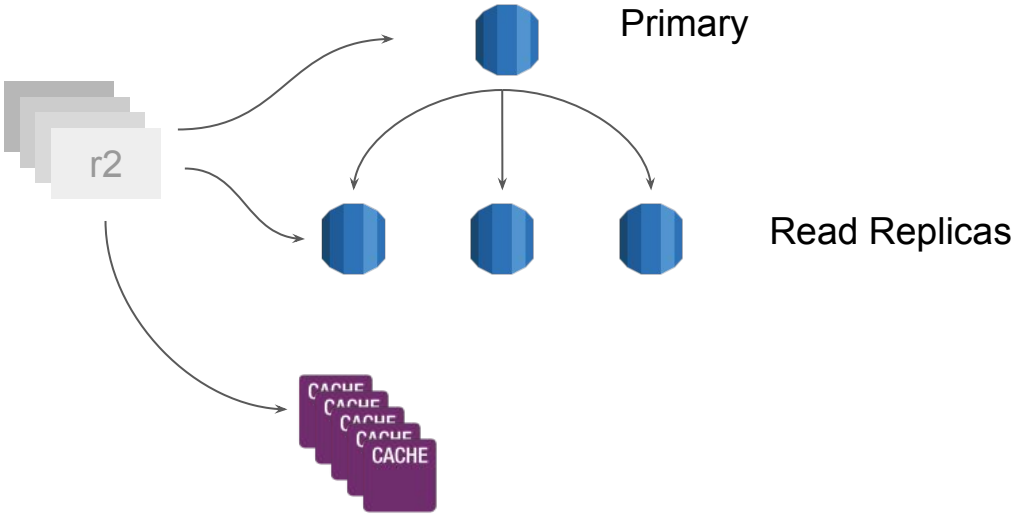


Thing in memcached

Whole Thing objects serialized and added to memcached.

r2 reads from memcached first and only hits PostgreSQL on cache miss.

r2 writes changes directly to memcached at same time it does to PostgreSQL.

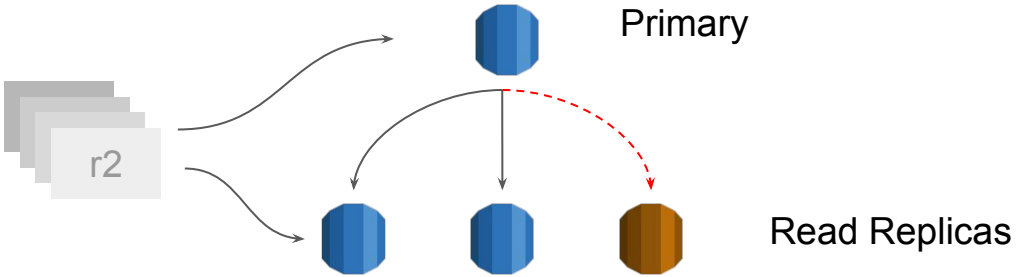


Incident

2011

Alerts indicating replication has crashed on a replica.

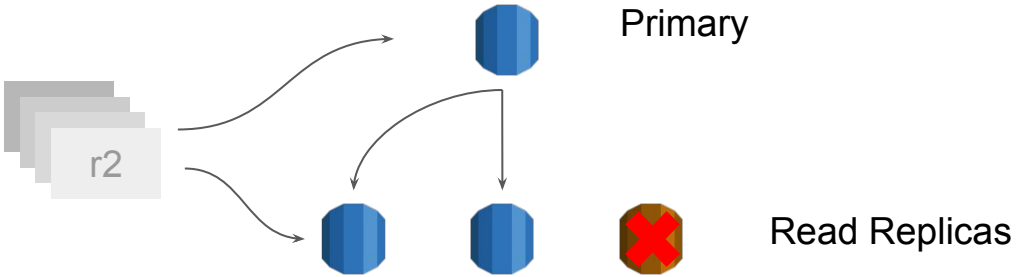
It is getting more out of date as time goes on.



Incident

Immediate response is to remove broken replica and rebuild.

Diminished capacity, but no direct impact on users.



Incident

Afterwards, we see references left around to things that don't exist in the database.

This causes the page to crash since it can't find all the necessary data.

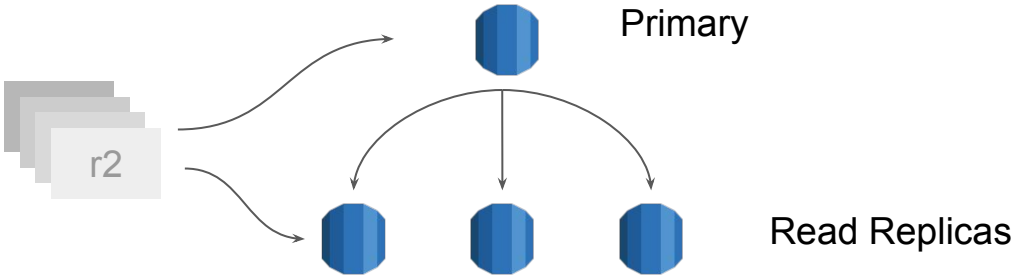
r/example hot links: #1, #2, #3, #4

reddit_thing_link

id	ups	downs	deleted
1	1	0	f
2	99	10	t
4	345	3	f

Incident

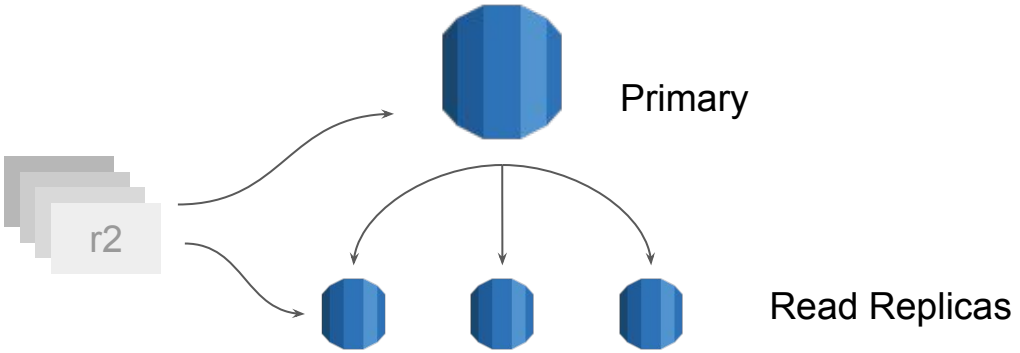
The issue always starts with a primary saturating its disks.



Incident

The issue always starts with a primary saturating its disks.

Upgrade the hardware!

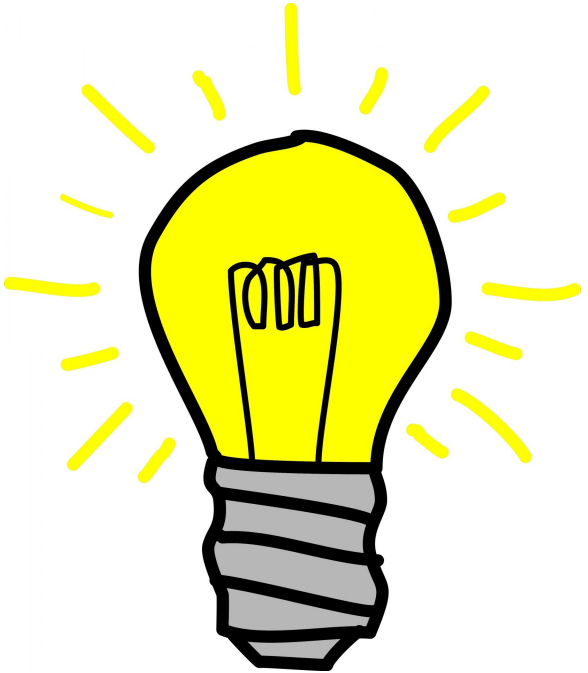


How unsatisfying...

A clue

Primary is bumped offline momentarily during a routine maintenance a few months later.

The old replication problem recurs on a secondary database.



The failover code

List of databases always starts with primary.

```
live_databases = [db for db in databases if db.alive]
primary = live_databases[0]
secondaries = live_databases[1:]

…

if query.type == “select”:
    random.choice(secondaries).execute(query)
elif query.type in (“insert”, “update”):
    primary.execute(query)
```

Oops

The failover code was failing out the primary and writing to a secondary.

```
- live_databases = [db for db in databases if db.alive]
- primary = live_databases[0]
- secondaries = live_databases[1:]

+ primary = databases[0]
+ secondaries = [db for db in databases[1:] if
db.alive]
```

Learnings

Layers of protection help. Security controls can also be availability features.

Learnings

If you denormalize, build tooling to make your data consistent again.

Discovery

New services use service discovery to find databases.

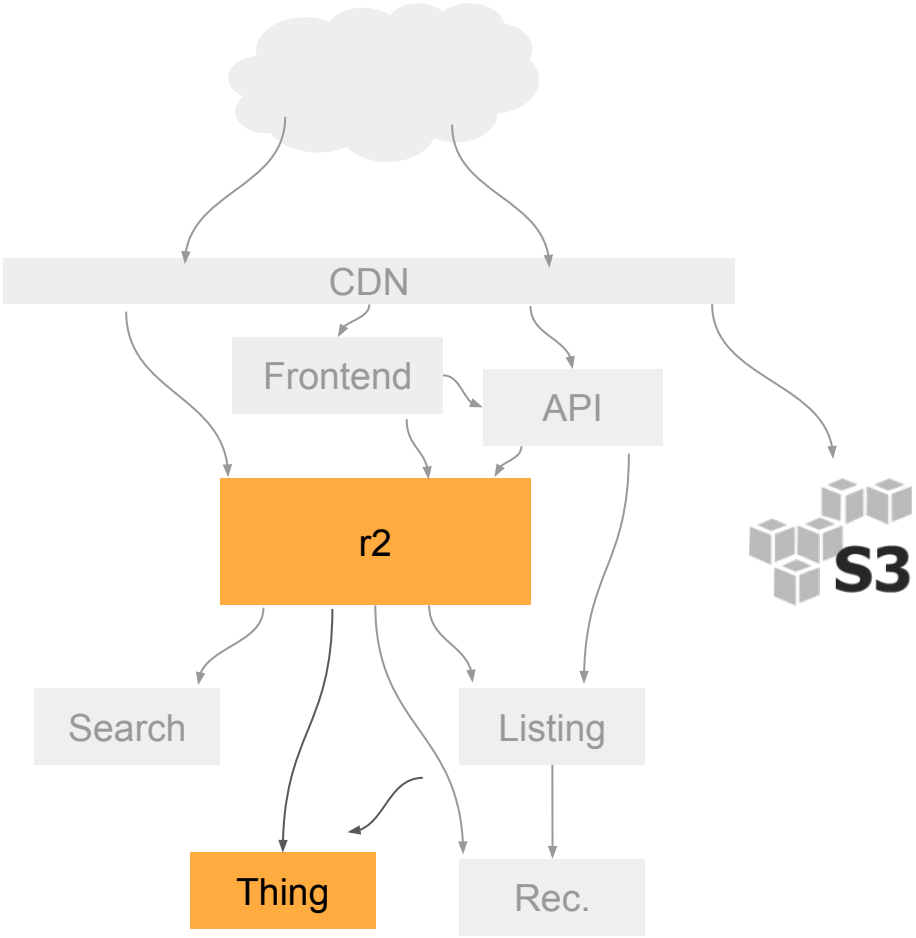
This reduces in-app complexity.

Thing service

Liberating these data models from r2.

This provides access to the data for other services.

Forces untangling complicated legacy code.



Comment Trees

Comment Trees

Tree of comments showing structure of reply threads.

Comment Trees

It’s also possible to link directly to comments deep in tree with context.

Comment Trees

Expensive to figure out the tree metadata in-request, so we precompute and store it.

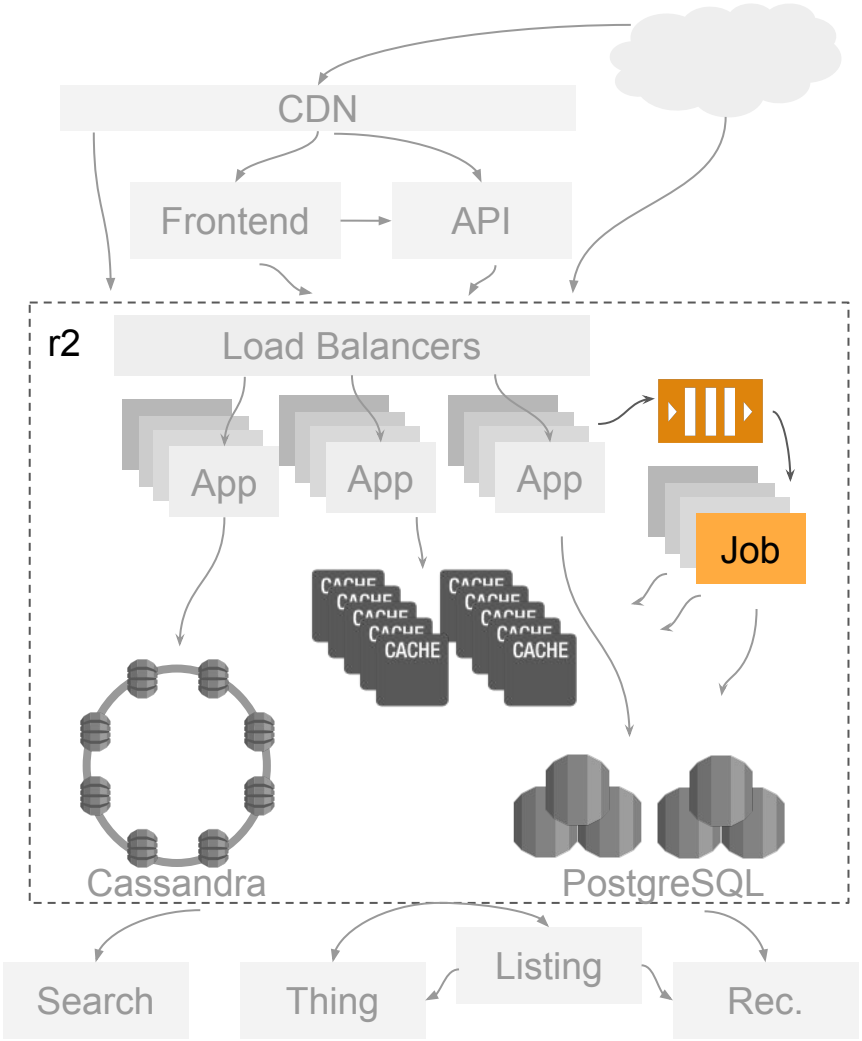
```
children = {
  1: [
     2,
     3,
     4,
     5,
      ...
  ],
  2: [
     6
  ],
  74656: [
     80422
  ],
  ...
}
```

Comment Tree Queues

Updating materialized tree structure is expensive.

Deferred to offline job queues.

Process updates in batches to reduce number of distinct changes.



Comment Tree Queues

Updating tree structure is sensitive to ordering.

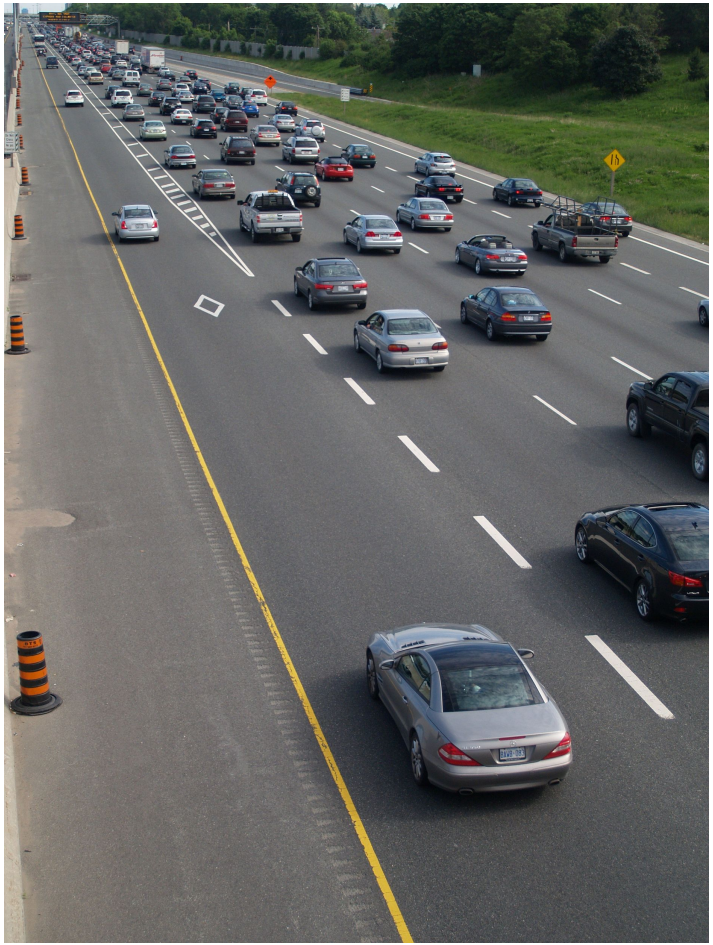
Hard to get into the tree if your parent isn’t there!

Inconsistencies trigger automatic recompute.

Fastlane

Massive threads hog resources. Slow themselves *and* the rest of the site down.

Fastlane is dedicated queue for manually flagged threads to get isolated processing capacity.



https://commons.wikimedia.org/wiki/File:404HOV_lane.png

Incident

Early 2016

Major news event happening. Massive comment thread discussing it actively.

Busy thread is overwhelming processing and slowing down comments across the site.

Incident

We fastlane the news thread to isolate its effects.

Incident

Suddenly, the fastlane queue starts growing exponentially.

Fills available memory on queue broker.

Incident

No new messages can be added to queues now.

Site actions like voting, commenting, and posting links are all frozen.

Self-“healing”

The main queue was backed up.

Switching to fastlane allowed new messages to skip the queue.

Tree is now inconsistent, this causes recompute messages to flood the queue on every pageview.

Start over

We had to restart the queue broker and lose existing messages to get things back to normal.

This then meant a bunch of data structures needed to be recomputed afterwards.

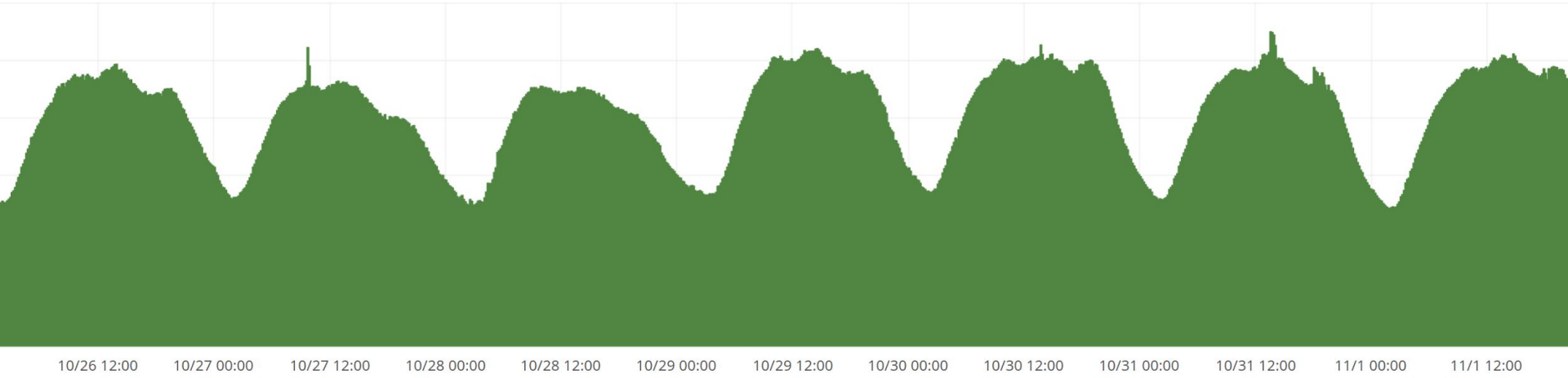
Queue Quotas

We now set maximum queue lengths so that no one queue can consume all resources.

User-visible, but scope of impact limited.

Quotas are important for isolation.

Autoscaler


10/26 12:00
10/27 00:00
10/27 12:00
10/28 00:00
10/28 12:00
10/29 00:00
10/29 12:00
10/30 00:00
10/30 12:00
10/31 00:00
10/31 12:00
11/1 00:00
11/1 12:00

Autoscaler

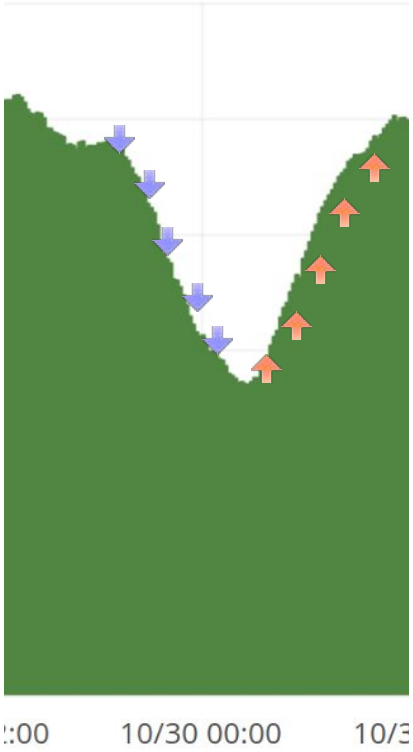
Save money off peak.

Automatically react to higher demand.

Autoscaler

Watch utilization metrics and increase/decrease desired capacity accordingly.

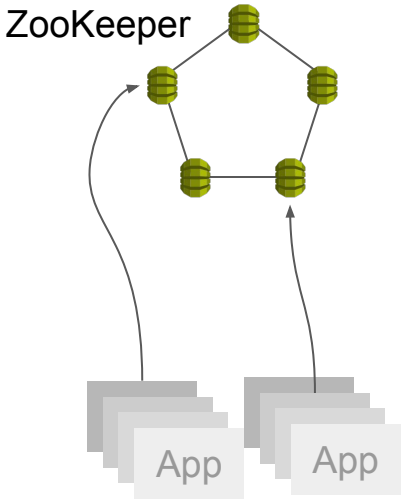
Let AWS's autoscaling groups handle the work of launching/terminating instances.



Autoscaler

Daemon on host registers existence of host.

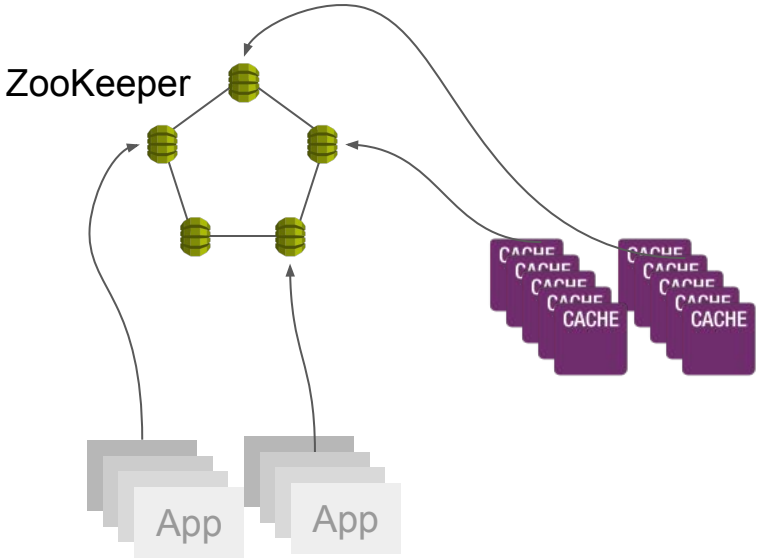
Autoscaler uses this to determine health of hosts.



“Autoscaled” memcached

Cache servers were managed with this system as well.

No scaling out/in but automatic replacement of failed nodes.



Incident

Mid 2016

Migrating entire site from EC2 Classic to VPC.

Last servers to move are the ZooKeeper cluster.

The plan

1. Launch new ZooKeeper cluster in VPC.
2. Stop all autoscaler services.
3. Repoint autoscaler agents on all servers to new cluster.
4. Repoint autoscaler services to new cluster.
5. Restart autoscaler services.
6. Nobody knows anything happened.

The reality

1. ✓ Launch new ZooKeeper cluster in VPC.
2. ✓ Stop all autoscaler services.
3. Start repointing autoscaler agents on all servers to new cluster.

And then suddenly hundreds of servers get terminated, including many caches.

What happened?

Puppet agent ran and re-enabled the autoscaler services.

These services were still pointed at the old ZooKeeper cluster.

Anything migrated to the new cluster was seen as unhealthy and terminated.

Recovery

Realize very quickly why the servers all went down.

Re-launch many servers. This just takes time.

Lost cache servers came back cold. PostgreSQL completely slammed with reads. Have to gently re-warm caches.

Learnings

Tools that take destructive actions need sanity checks.

Learnings

Process improvements needed: peer-reviewed checklists for complex procedures.

Learnings

Stateful services are very different from stateless ones, don’t use the same tooling for them!

Autoscaler v2

The next generation autoscaler uses our service discovery tooling for health checking.

Autoscaler v2

Importantly, it will refuse to take
actions on too many servers at once.

Summary

Remember the human

Observability is key.

People make mistakes. Use multiple layers of safeguards.

Simple and easy to understand goes a long way.

Thanks!

Neil Williams
u/spladug or @spladug

This is just the beginning, come join us!
https://reddit.com/jobs

Infra/Ops team AMA, Thursday in r/sysadmin
https://redd.it/7cl9wv