Resharding Redis In a Live Environment

JANUARY 15, 2014 BY JASON MCGUIRK 5 COMMENTS

We're about to go live world wide with our first title, HonorBound – and we thought it was a good opportunity to scale up our Redis servers in advance of the (hopefully) coming wave of installs and DAUs.

We currently use Redis as both our primary user store and for our leaderboard, matchmaking and inbox services (all of which are backed by Redis's awesome sorted set functionality). We use a partitioning strategy where all keys for a particular player are routed to the same node – we sometimes refer to this strategy as a prescriptive hash, since the application's data layer is dictating (prescribing) the node where a particular set of keys are stored – rather than leaving it to the Redis bindings (or some other proxy service) to figure it out. This strategy allows us have transactional consistency across a single update (via Redis's multi/exec command) – but at the cost of making it a little bit more challenging to scale up logistically.

Our traffic is still relatively small (Approximately 2,500 DAU) so resharding now provides us an opportunity to prove out both that our data layer can correctly address a larger cluster, and that our battle plan for resharding is proven incase we need to do it in a pinch.

This post details the process we followed (and lessons learned) in scaling up our Redis cluster from a single stack to three stacks. Redis suggests presharding as a strategy (http://redis.io/topics/partitioning) – which in hindsight is an awesome idea, and something we're likely to pursue for subsequent shard-ups.

Step 1: The Plan

Resharding your main data store is a high risk operation. You won't do it frequently and the stakes are high. Set aside some focus time to draft your plan and get buy in from your leadership team.

Channeling Kevin (Mo) Hagan and Nick Tornow from our upshards on Zynga Poker – we setup a master google doc which would outline timeline and status of key tasks.

https://docs.google.com/spreadsheet/ccc?key=0ArfyFdwqD8JcdEQzc0JMLTZJWndsMEh2YWNlVFJuRlE&us

During your planning, establish an outage v cation. Be pessimistic for the time frame required (get buy in for atlea our) and aim for the time when the fewest number of users will b s time was 2 AM PST – where concurrents trickles down to about 1 is important not only to coordinate your team, but also as a remind dow.

From a high level the plan looks something

- Provision your new hardware
- Configure your new hardware
- Slave new nodes to old nodes
- Verify slaving
- Test new configuration
- Bring application down
- Push new configuration
- Bring application up
- Monitor

During this planning – you may identify some unmet dependencies. For us, we had no way of signalling to end users that our application was down due to planned maintenance – which is important to message as distinct from a generic server error message. We pushed a patch to enable handling this

We're Hiring!

JuiceBox Games is looking for talented Game Developers and Artists to join our growing team. Come be part of the midcore revolution!

TAIL

- Thanks for the pizza! April 2, 2015
- They're after me lucky charms March 18, 2015
- What a year for JuiceBox! Looking forward to 2015! January 9, 2015
- 716 December 24, 2014
- HonorBound Forum is live! September 26, 2014
- New Digs July 17, 2014
- Running Redis in a Production Environment April 2, 2014
- Promise to our Players March 24, 2014
- Running Node.js in a Production Game March 21, 2014
- Welcome to HonorBound! February 13, 2014

RSS

RSS - Posts

ABOUT

- app (19)
- etc (6)
- general (20)
- journal (32)
- tech (14)

error code before we began the sharding process and a change to our runtime tool to allow us to bring the application up/down immediately. The goal is to have as much done before you bring the application down for maintenance – give yourself plenty of runway.

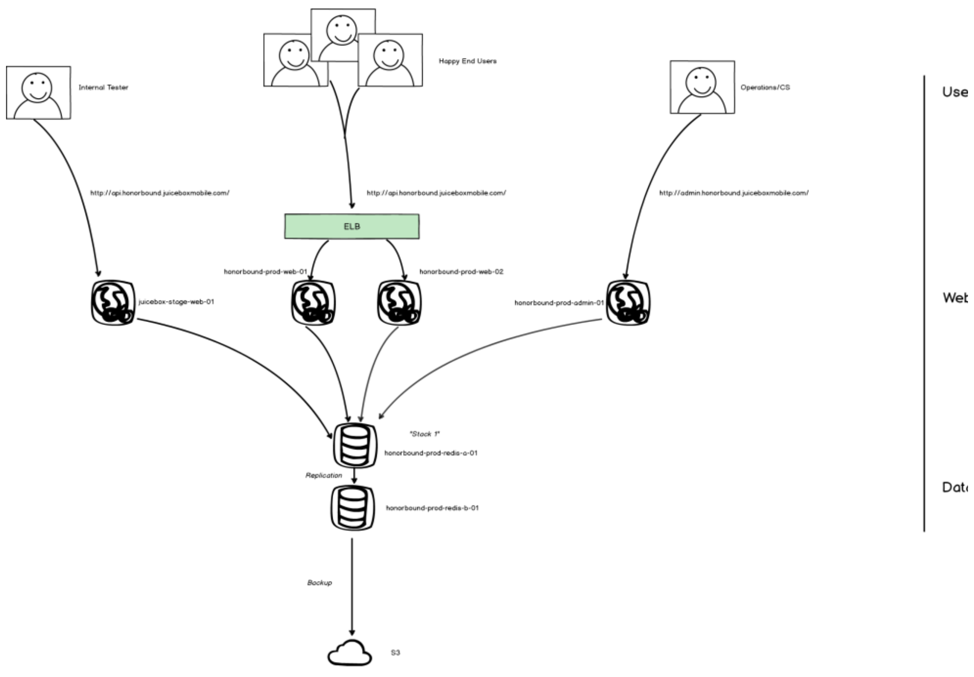
Finally, we also need to develop a contingency plan. This is what we will do if things go horribly sidways and we need to back out the change. Its also useful during this time to establish some sense of criteria for when we would execute a contingency plan and how long we have before the plan is invalid.

Step 2: Provision and configure your new hardware

Depending on how much you've automated your provisioning and configuration, this will likely be the longest time sink. Our cloud vendor is AWS – so spinning up new hardware was a relatively painless process.

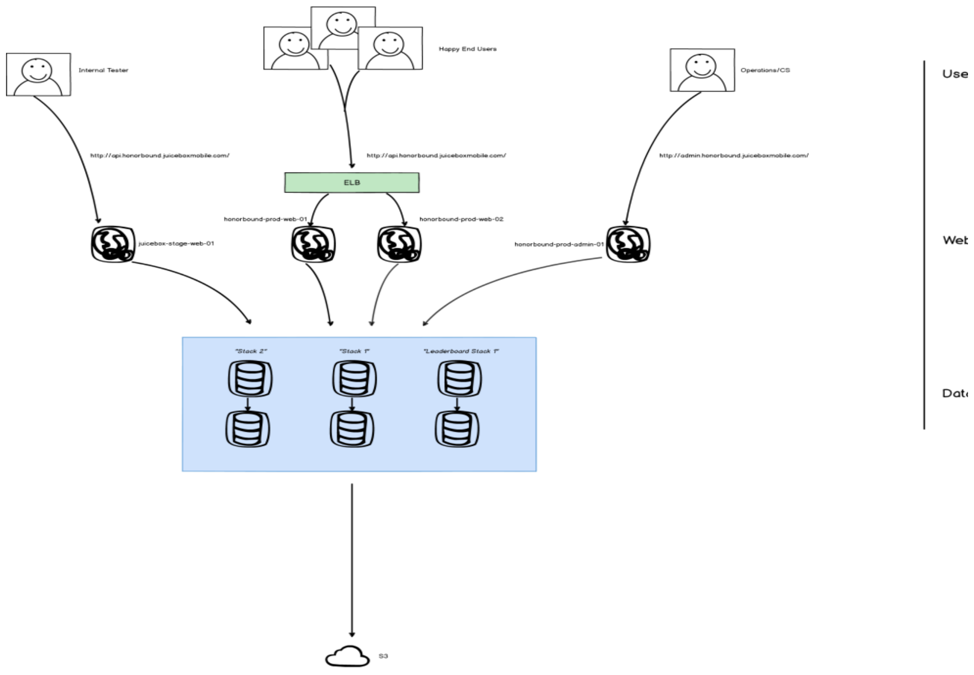
You'll likely want to diagram out your server configuration during each step – for sanity and communication.

Here's where we were prior to our upscale:


Happy End Users
Internal Tester
Operations/CS
Use
http://api.honorbound.juiceboxmobile.com/
http://api.honorbound.juiceboxmobile.com/
http://admin.honorbound.juiceboxmobile.com/
ELB
honorbound-prod-web-01
honorbound-prod-web-02
juicebox-stage-web-01
honorbound-prod-admin-01
Web
"Stack 1"
honorbound-prod-redis-a-01
Replication
honorbound-prod-redis-b-01
Dat
Backup
S3

Sharding – Initial State

And here's where we were aiming to get:



Sharding – Final State

We wanted to increase our main storage capacity and move off our leaderboards to a dedicated server – since the memory requirements and storage needs are significantly different.

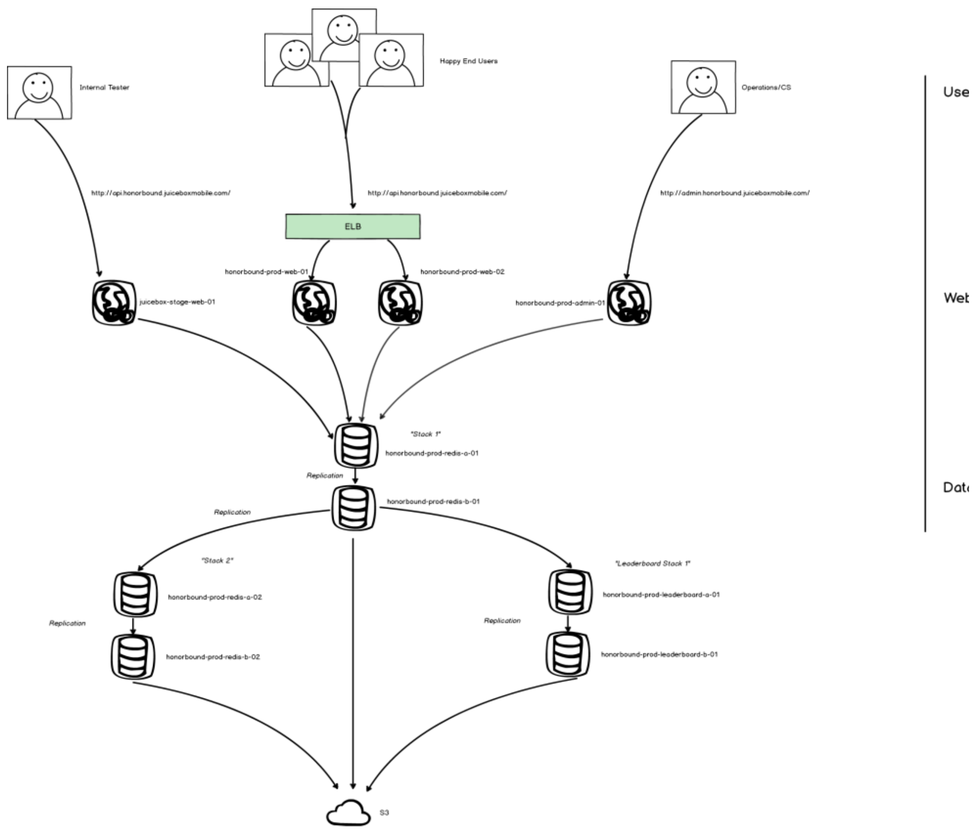
You'll want to kick the tires here when you think your done with a series of sanity checks. This is an important pre-test step that will tell you if your configuration is sane:

- Check connectivity from your web nodes. You'll likely need to open up a port in either your security group or on the local iptables.
- Verify slaving correctly from new masters to new slaves. This can be done by setting a key on the master and getting it on the slave
- Verify your nightly backups are working. For us, we store nightly snapshots of the Redis RDB in S3. We checked to see that the S3 tarballs are making their way up correctly to the right folder
- Verify reporting/alerting is wired up correctly. For us, we use monit and checked to see that the servers could correctly raise an alert if they were stranded.

Provisioning and validating the four new servers took roughly half a man day – but should go substantially faster if you've automated server setup with something fancy like Chef.

Step 3: Slave up your nodes

So the next step after we've setup our new server nodes is to slave them up to our existing nodes. This allows for existing records to appear in their new location properly. Our intermediate configuration looked something like this


Happy End Users
Internal Tester
Operations/CS
Use
http://api.honorbound.juiceboxmobile.com/
http://api.honorbound.juiceboxmobile.com/
http://admin.honorbound.juiceboxmobile.com/
ELB
honorbound-prod-web-01
honorbound-prod-web-02
juicebox-stage-web-01
honorbound-prod-admin-01
Wet
"Stack 1"
honorbound-prod-redis-a-01
Replication
honorbound-prod-redis-b-01
Dat
Replication
"Stack 2"
honorbound-prod-redis-a-02
Replication
honorbound-prod-redis-b-02
"Leaderboard Stack 1"
honorbound-prod-leaderboard-a-01
Replication
honorbound-prod-leaderboard-b-01
S3

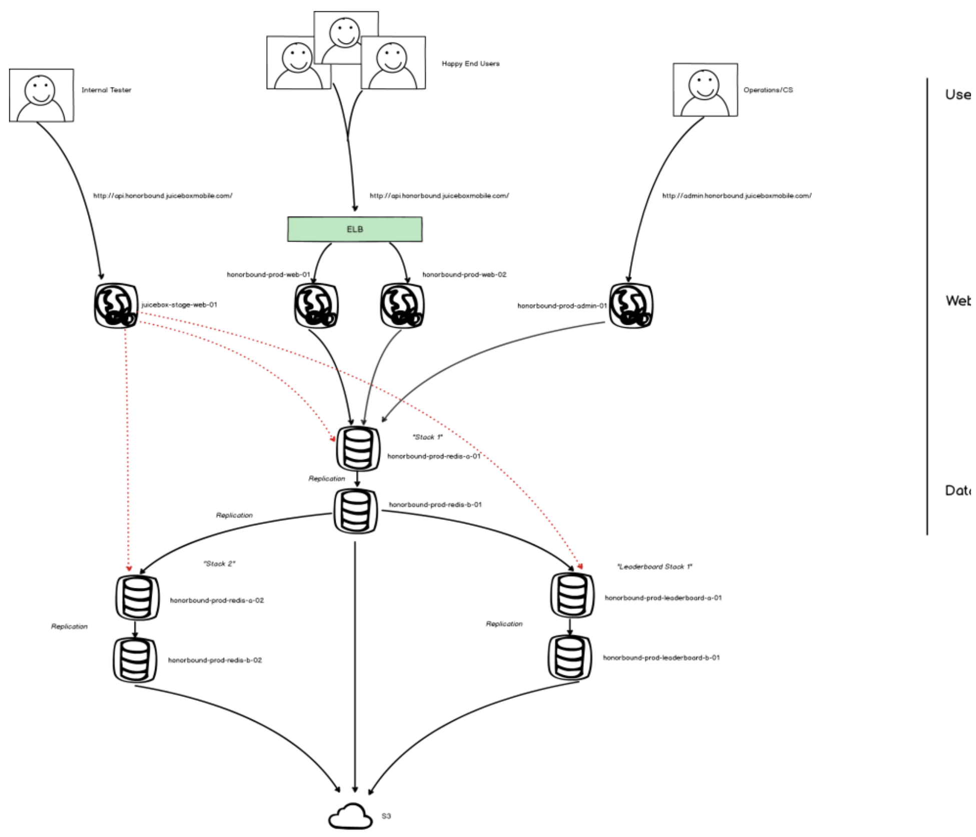
Sharding Configuration – New boxes are slaved

.

Importantly, you'll want to slave up your masters to the existing stack slaves. Starting a fresh slave can be momentarily traumatic to a Redis box as it will need to fetch records from cold storage – so slaving to a slave will limit the impact of these new slaves on your live site. Validate that the records are flowing by checking the key count between slaves – this can be done with the redis info command.

Step 4: Test your new configuration

You'll want to test your new configuration with something like a prod stage configuration prior to going live. This is a crucial step, as it exposes obvious failures in your data access patterns or configuration. You'll want to draft a test plan that exposes key data patterns like new installs, multi-gets and multi-writes. You'll also want to make sure your data crosses both the new shards and the old ones and that persistence is maintained properly.


Internal Tester
Happy End Users
Operations/CS
Use
http://api.honorbound.juiceboxmobile.com/
http://api.honorbound.juiceboxmobile.com/
http://admin.honorbound.juiceboxmobile.com/
ELB
honorbound-prod-web-01
honorbound-prod-web-02
juicebox-stage-web-01
honorbound-prod-admin-01
Wet
"Stack 1"
honorbound-prod-redis-a-01
Replication
honorbound-prod-redis-b-01
Dat
Replication
"Stack 2"
"Leaderboard Stack 1"
honorbound-prod-redis-a-02
Replication
honorbound-prod-redis-b-02
honorbound-prod-leaderboard-a-01
Replication
honorbound-prod-leaderboard-b-01
S3

Shard Configuration – Testing

Ideally – you'll want to do this testing well in advance of the maintenance window and use this test to inform a go/no go before people leave for the evening.

Step 5: Bring your application down

So our hardware is ready, we've done our testing and we're at concurrents trough – it's almost time to go. We use this time to do one last preflight check:

- Check that error log velocity is normal – We want to make sure there's nothing exceptional going on in the app before we bring it down.
- Double check S3 backups – we will need these if things go really bad.
- Ensure slaves have caught up – Do another check on keycount and slave delay here. We want to make sure we're as near as possible to the latest in our master data.

Once you've done your preflight check. Bring the application down and verify its down – both in your logs and as an end user.

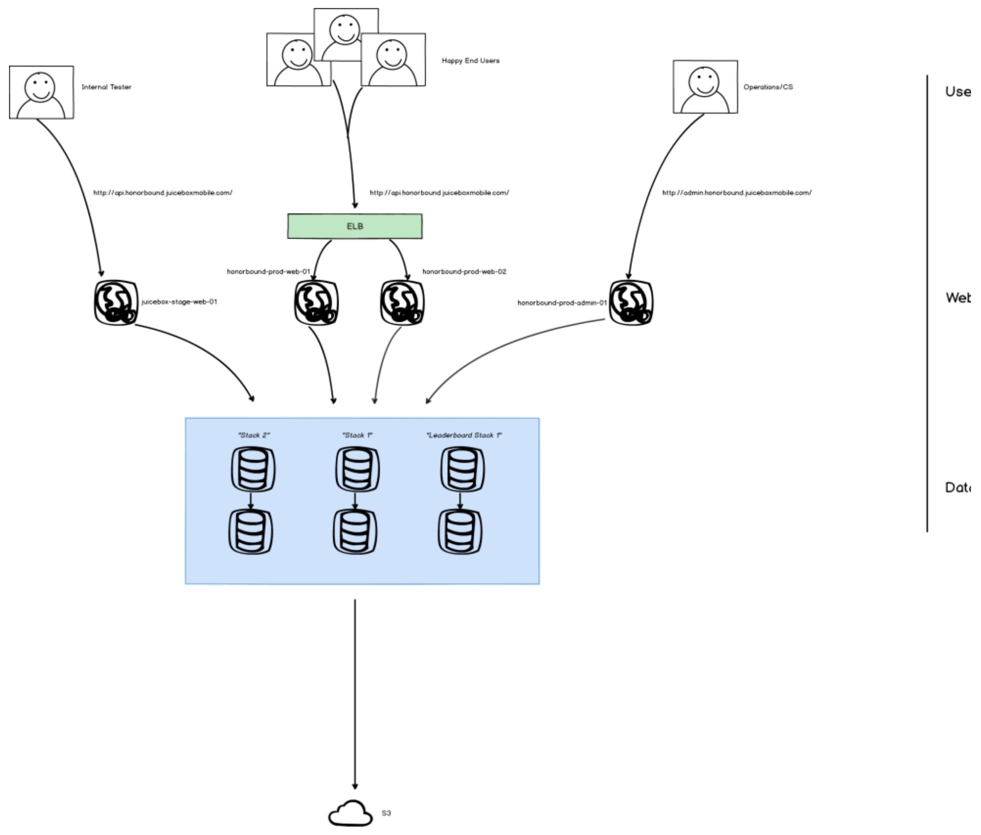
The clock starts here for our outage window, we note the time and check frequently incase we need to abort and bring the application back up

Step 6: Stop Cross Stack Replication

Next we need to stop our cross stack replication from our old slaves to our new masters. We change redis.conf, restart redis and make sure the box is no longer slaving.

Step 7: Push your configuration

The next step is to push your configuration changes live to production. This should be the same configuration you were testing on staging with.



Sharding Configuration – Final Configuration

Double check your app servers have the new code and have correctly restarted with the new code.

Step 8: Bring your app up and test

After we've pushed our code live – it's time to bring our players back. We flip the switch back on and let players back in. Once the application is back up, we execute our test plan again – looking for anything that might have been different between our staging environment and prod.

We note the time here the application came back up to bench it against our plan.

Step 9: Monitor and declare victory

Assuming the test succeeded, we shift focus to intense monitoring – looking particularly for an uptick in types of errors. Note that many more players (those locked out) will be re-entering the application, this will cause a short term shift in your error velocity as your usage pattern shifts. You're looking for a stabilization sometime within the first 10-15 minutes (atleast for us) of turning the application back on.

You're monitoring specifically for new classes of errors that crop up from your new configuration. These are usually new types of "failed to fetch" or "failed to save" records. One thing to note is that your web servers will likely cough up a few connection errors to the new server as the network connection "warms up".

If things are looking good after the first 30 minutes, we send out a victory email to notify the team of the operation completion and the times of app outage – this is relevant for explaining any stats that may have dipped during this time.

Alternatively, if things are NOT looking good. Get ready to execute your contigency plan and backout the changes. Its important to make this call early if its going to be made as the masters will begin to drift and state will be lost on a rollback

Lessons Learned

So our first reshard for HonorBound went *relatively* smoothly. With the following things to improve

- We had an emergency hotfix about 36 hours after the outage – we had a particularly insidious bug in our data layer around multi-gets. Our multi-get code was not correctly spanning all shards and didn't crop up until we had a sufficiently high volume of new records attempting to be fetched. Error volume didn't begin creeping up until approximately 6 hours after the reshard was complete. Prior to doing your first shard, do an audit of your data layer code and make sure all your calls are ready to handle the new configuration.

- Hand configuration of new nodes is time consuming and error prone – this will likely be our last upshard where nodes are provisioned by hand. It's easy to punt on this till later, but misconfiguring a single node could be disastrous.

As always shoot me a note if there's any questions or if there's something we can optimize here 🙂

Related

Running Redis in a Production Environment
In "app"

Unboxing Redis 2.6
In "etc"

Redis Lists as Viral Inboxes
In "tech"

FILED UNDER: APP, ETC, JOURNAL, TECH

« Running a Geolock for a Mobile Game   Mobile Crossplatform Release Management »

Comments

Sean Janis says:
January 15, 2014 at 11:49 pm

On a related note, I'm eagerly awaiting Redis' "true" cluster feature. They've been stalled in an experimental state for a while, but should hopefully reduce the need to manually shard.

Reply

Jason McGuirk says:
January 15, 2014 at 11:51 pm

Yea – I read about that, really cool – I probably wouldn't grab it until the next major version or two after its launched. I have a pretty deep seated mistrust for auto clustering though – but thats just my own baggage :p

Reply

swarajban says:
January 16, 2014 at 3:13 am

My company is dealing with a similar problem by using ketama hashing, no need to pre-shard:
http://www.last.fm/user/RJ/journal/2007/04/10/rz_libketama_-_a_consistent_hashing_algo_for_memcache_clients

Reply

Chris Broglie says:
January 16, 2014 at 5:36 am

Did you guys consider Twemproxy?

Reply

Andrey Smirnov (@smira) says:
March 25, 2014 at 7:24 pm

Have you tried Redis Resharding Proxy (https://github.com/smira/redis-resharding-proxy)?

Reply

Share your thoughts

Enter your comment here...