SCALING DATABASES AT ACTIVISION

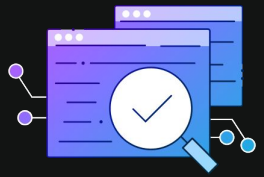
A brief story of how we came to use Vitess/Kubernetes to power some of the biggest entertainment franchises on the planet

ACTIVISION BLIZZARD King

demonware

Greg Smith
Principal Arch.

Vlad Kovacik
Sr. Expert SRE



Background

Challenges

Tech Adoption

Conclusions

BACKGROUND

HISTORY OF DB INFRA

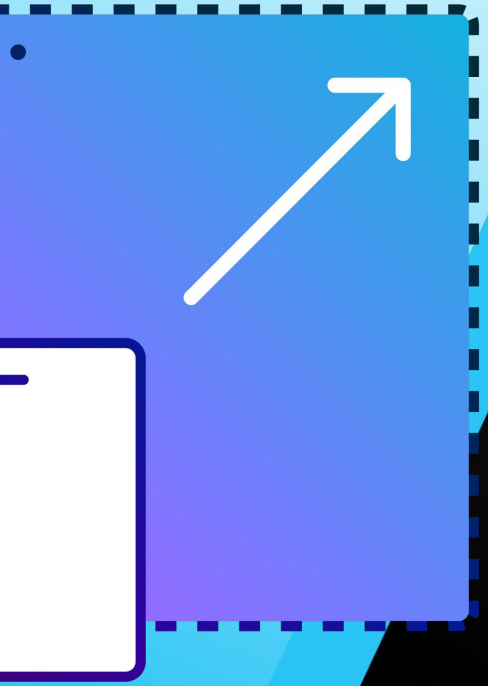
Bare Metal

Hardcoded DB name in application config

Virtual machines / System containers

- KVM/LXC
- Thousands of database VMs
- Proprietary DB discovery and SQL routing solution
- Shards definition in application configuration
- Automation for DB failure remediation

DB CHALLENGES

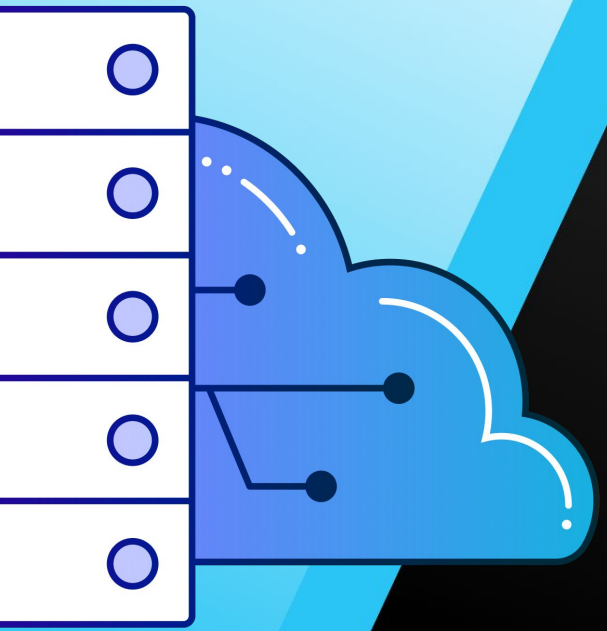


SCALABILITY

- Game launch day traffic is a magnitude higher
- Significant load testing to get "right" shard count
- Shards were overbuilt but never scaled back
- No connection pooling in DB routing layer
- High number of connections (10k+ per DB) were causing MySQL performance and memory issues

OPERATIONAL BURDEN

- Failed replica required On-Call to manually run automation
- Large scale infra failure resulted in hours of manual work
- After such failure, replicas were having more transactions than primaries and had to be replaced
- Only able to scale out by doubling the shards count



SETUP COMPLEXITY

- Service teams own database operations
- Teams needed embedded experts to build and maintain databases
- Database cluster with 100+ shards took 2+ days to build - slow development iterations
- Schema migrations required careful planning and execution in production

TECH ADOPTION

NEXTGEN DB EVALUATION

Requirements

- SQL query compatibility
- Minimal changes to the application
- Runs MySQL in backend
- Kubernetes native
- Provides Kubernetes operator

We evaluated multiple candidates and chose **Vitess**

VITESS ADOPTION

Why did we choose Vitess?

- Near drop-in replacement
- We had MySQL experts on staff
- Success stories of other companies
- Large active community

ADOPTION OVERVIEW

Small Service

- Was able to fine tune workflows
- Build production confidence
- Align technical expectations with platform teams
- See it in action
- Small blast radius

Large Service

- Take work from small service and apply to large
- Prove out scalability
- Have an entire team onboard to the service
- Gain trust within the company

SMALL SERVICE

WHY

- Small and manageable
- Start with fresh data
- Not in the critical login path
- Easy to make architecture changes before launch
- Work with platform integration might prove useful for larger projects
- Good baseline for building a framework for other services

TIMELINE

V0	V1	V2	V3
LXC	Kubernetes	Kubernetes	Kubernetes
In House Orchestration	Helm	Operator	Operator
Ansible			Orchestrator

All backed by local storage

- Eventually have choice of Ceph/Local

PROOF OF CONCEPT

How Did It Go?

- Slow to start, lots of options for how to run things
- Platform integration was fairly smooth
- Testing, more testing
  - Shard expansion
  - Vertical scaling
  - Recovery and Error handling
- Load testing went well
- Built good relationships with open source community
- Launch was a success

LARGE SERVICE

runs

~100

shards

database size

30+tb

as of 2023

data size

~x2

every 2 years

serves

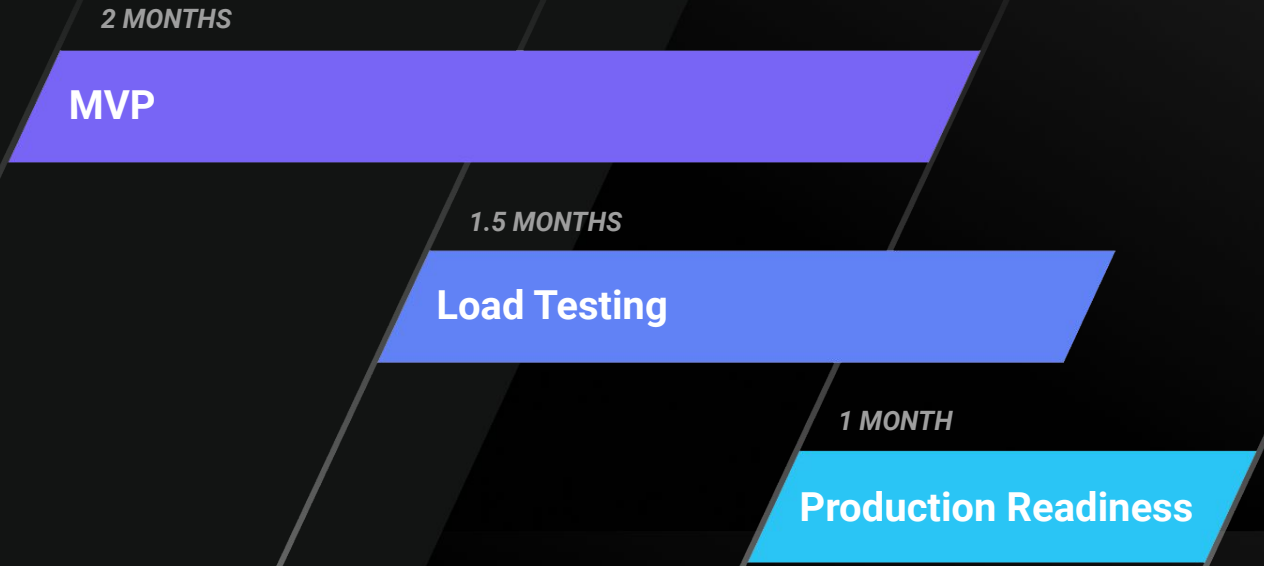
~500k

qps at peak

Inventory Service stores game items owned by player

Business Critical

ADOPTION STAGES



MVP

- Needed criteria to confirm that service works with Vitess
- Fortunately we had good unit tests coverage
- Identified business critical unit tests
- More than 50% of database tests were failing initially
- Failing tests were categorized into a small number of common problems
- Engagement with Vitess community
- All blockers were resolved

INCOMPATIBILITIES & VITESS SPECIFICS

MySQL named locks (GET_LOCK)

PROBLEM
Vitess implementation creates reserved connections to the first Vitess shard which can be scalability bottleneck

SOLUTION
Redesign app to reduce number of locks per sec

MySQL foreign key constraints not fully supported

PROBLEM
Certain Vitess scaling operations fail on FK violation

SOLUTION
Temporarily disable FK checks on the target shards

REPLACE INTO Not Supported

PROBLEM
Query requires all columns including shard key which can't be updated

SOLUTION
Switched to INSERT ON DUPLICATE

LOAD TESTING (LT)

- Player data collocated on shard using PlayerID
- Workload scales well with shards
- LT focused on Vitess components that are on query path: vtgate (proxy), vttablet (mysql sidecar)
- Confirmed that all components were scaling linearly
- vtgate and vttablet take additional CPU resources compared to our previous model
- vttablet requires about the same CPU resources as mysql
- Connection pooling offsets added CPU resources

PRODUCTION READINESS

- MVP was functional and load tests were promising
- We needed to gain more experience
- Chaos testing
- Configuration deployment under load
- Tuned Vitess configuration for resiliency (semi-sync replication, orchestrator, Kubernetes pod-disruption-budget for shards)

MOVING FROM APPLICATION SHARDING TO A SINGLE ENDPOINT

Before

- Shards in application config
- Some queries were "too" shard aware
- Data expiration was executing DELETE with LIMIT on the shards

After

- Single database endpoint
- Optimized queries to help Vitess route them efficiently
- DELETE with LIMIT not supported so we implemented "shard walking"

CONCLUSION

BENEFITS

- Proven method of shard expansion / consolidation
  - Standard tooling provided by Vitess team
- On-call burden greatly reduced
  - Escalations are almost non-existent now
- Database setup using gitops model

Did it work?

YES

- We are building a team around supporting it
- There are approximately 60 separate Vitess clusters currently in Dev/Prod
- Vitess has become the default database solution for new products.

THANK YOU

demonware PlanetScale