Sharding with Pure FP

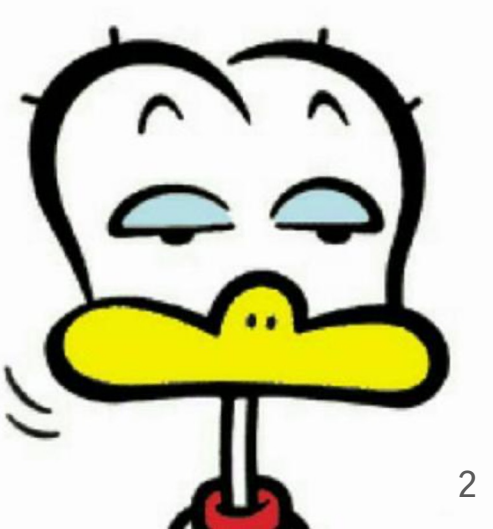
Pierre Ricadat
Functional Scala 2022

DEVSISTERS

Who am I?

Pierre Ricadat

- @ghostdogpr on GitHub/Twitter/Discord/etc
- Early ZIO contributor
- Creator of Caliban (GraphQL library for Scala)
- Creator of Shardcake (Sharding library for Scala)



Who are we?

Devsisters

- Korean gaming company
- Launched Cookie Run: Kingdom in January 2021
- 40+ million kingdoms
- Available worldwide
- Entirely made with functional Scala!

A common problem

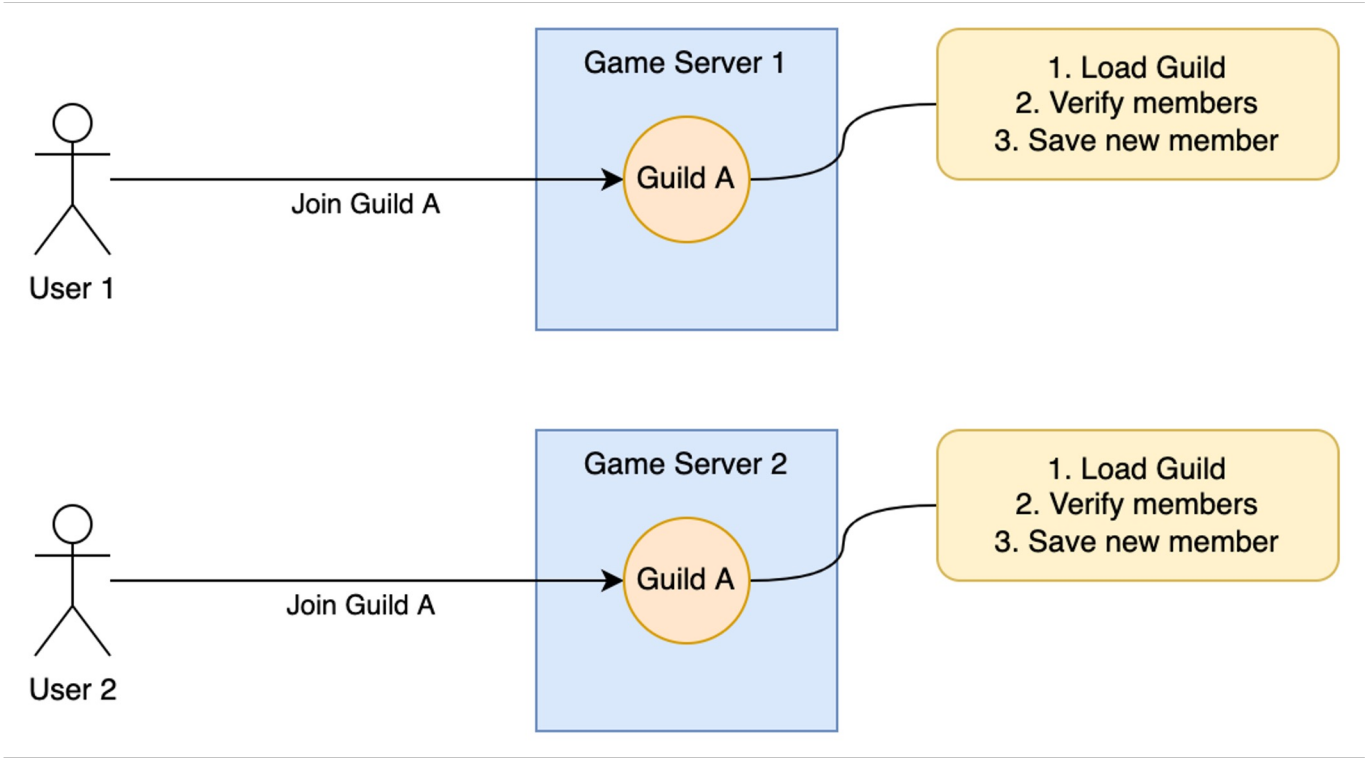
Let’s build a multiplayer game

- Players can create and join guilds
- A guild can only have a limited amount of players
- Ability to scale out game servers to support more users

This is applicable to any system where an entity is accessed concurrently.

Naive concurrency

What happens if both actions happen at the same time?



Solutions to the concurrency problems

Global lock

- Prevent multiple writers modifying the same data at the same time by making them wait on each other
- Commonly implemented in the Database layer
- Can lead to complexity, deadlocks, etc.

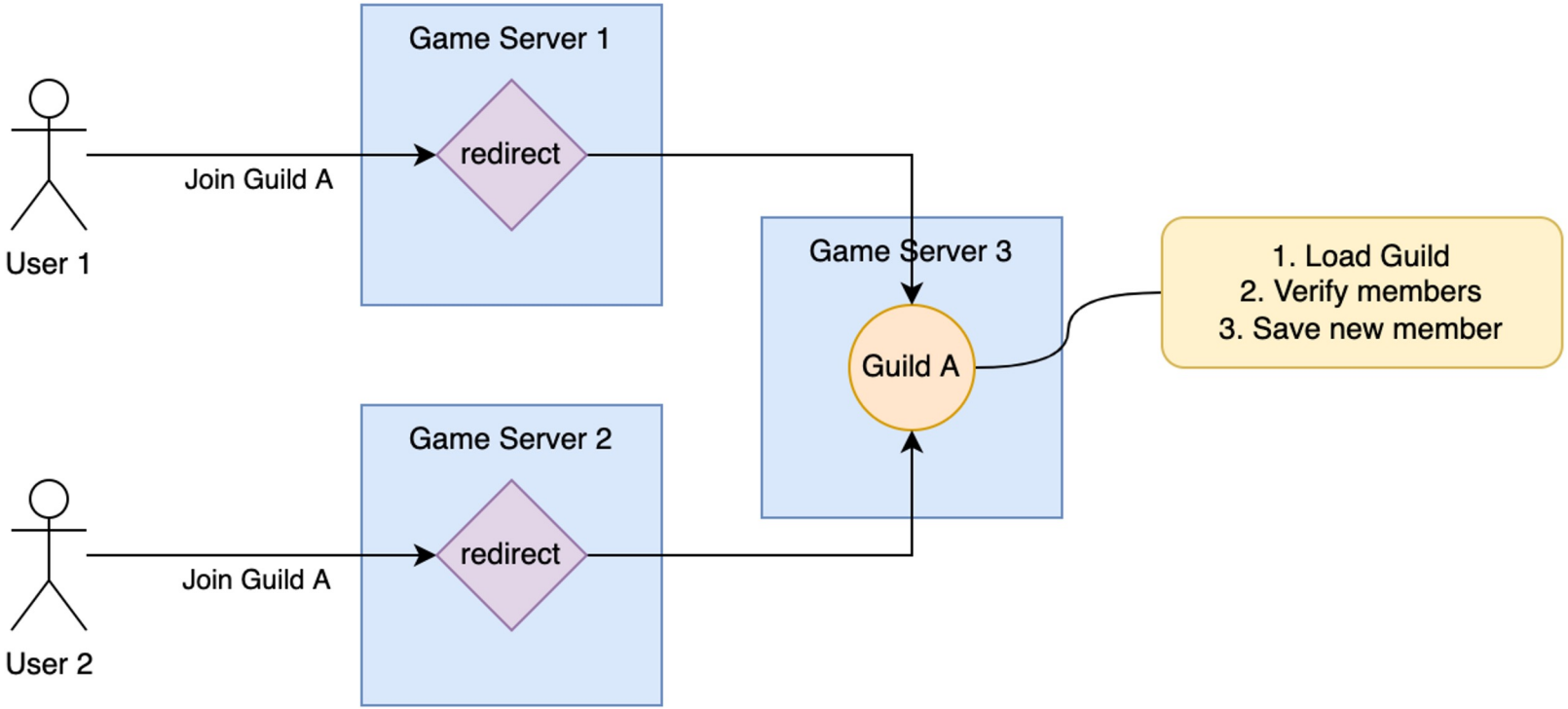
Solutions to the concurrency problems

Single writer

- All requests to modify the same data go to the same place
- Writer simply processes requests sequentially

Single writer solution

All requests go to the same game server



Single writer implementation

Can’t we just use Kafka?

- Kafka consumer groups
- Messages with the same key will go to the same consumer within a group

- Persistence
- Tolerance to lag
- Need for a response (sync vs async workflows)

Entity sharding

Akka Cluster Sharding

- All nodes form a cluster
- Entities (actors) are grouped into shards
- One node host a Shard Coordinator actor in charge of assignments and rebalancing
- Uses Akka remoting for communication between entities

Our experience with Akka

Development

- Easy to get started
- Very good docs

- Akka is not FP–oriented
- Built on top of Future
- Need to mix “pure” and “impure” code when using ZIO

Our experience with Akka

Production

- Cluster is fragile!
- Doesn’t like sudden topology changes
- Doesn’t like small network issues
- Several cases of Split Brain Resolver downing everything
- Timeouts when rebalancing shards
- Difficult to debug/troubleshoot

- PS: one more thing…

Production Pricing

Akka Standard

$1,995 USD

per core [1]

Billed Annually

(Volume discounts available)

$1,995 USD

per core [1]

per core

Let’s make our own!

Design decisions

- No more cluster!

  → Decouple application servers and membership management

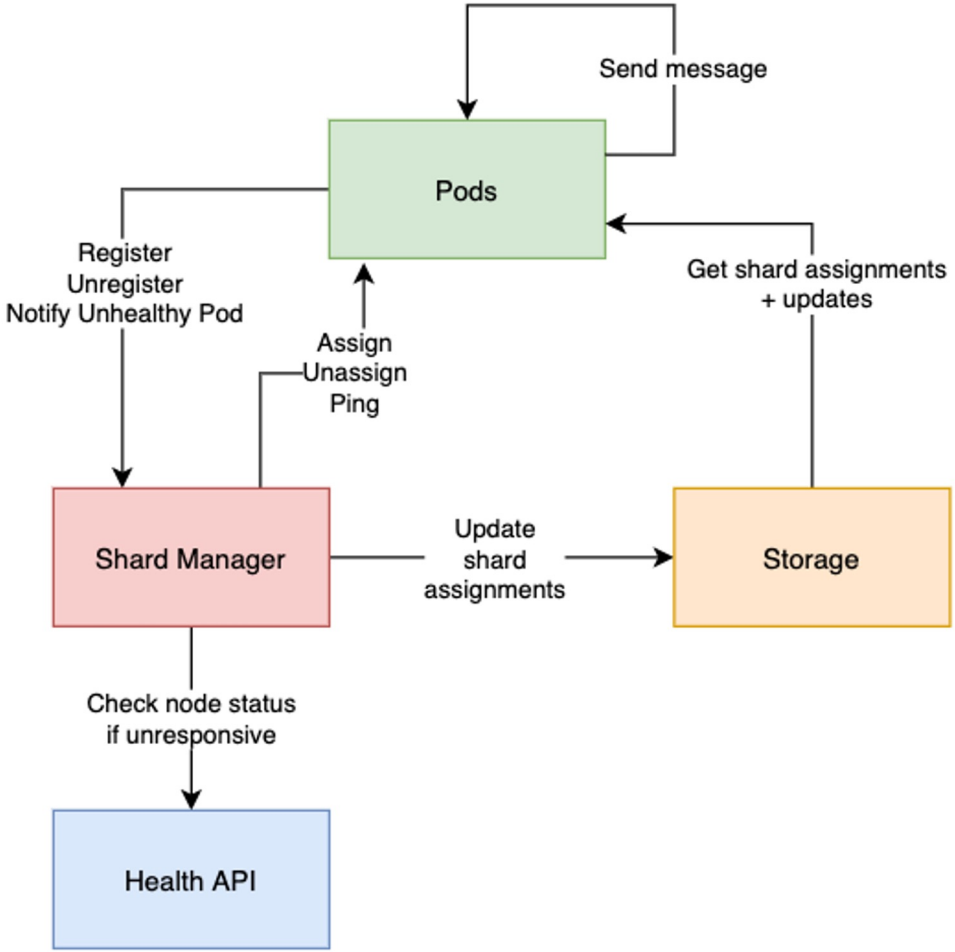
- Simple algorithm for shard assignment and rebalancing

  → Easy to tweak as we need

- Rely on Kubernetes as a source of truth for node health

  → Delegate the hard job

Architecture



But...

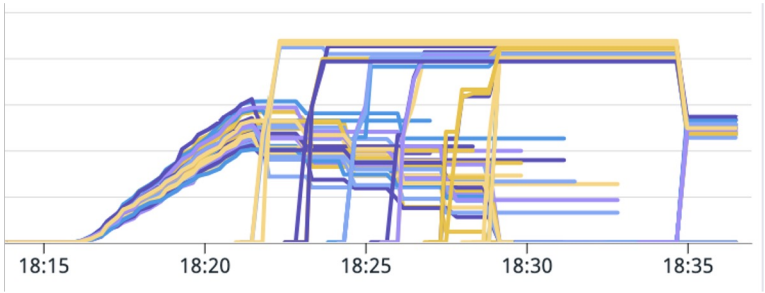
Isn't the Shard Manager a single point of failure?

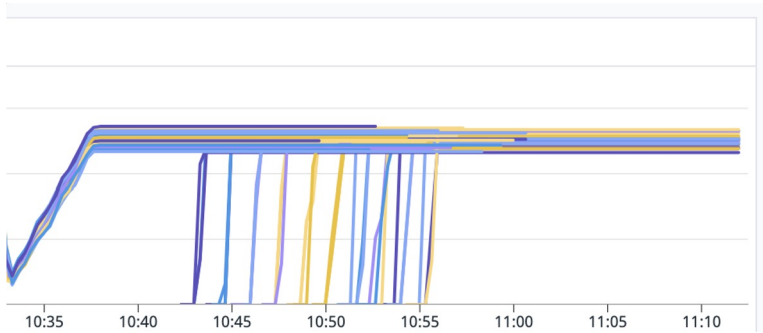
- It is only involved when pods are joining/leaving
- Pods use a locally cached pods<->shards mapping
- Very lightweight (fast to start a new one)

Building confidence

Load testing

- Risky change = need to test well!
- Performance tests: no significant difference with or without Akka
- Rolling update tests: rebalance algorithm improvements





(y axis = amount of entities per node)

Building confidence

Failure testing

- Nodes sudden shutdown
- Network packet drops
- Network packet delays

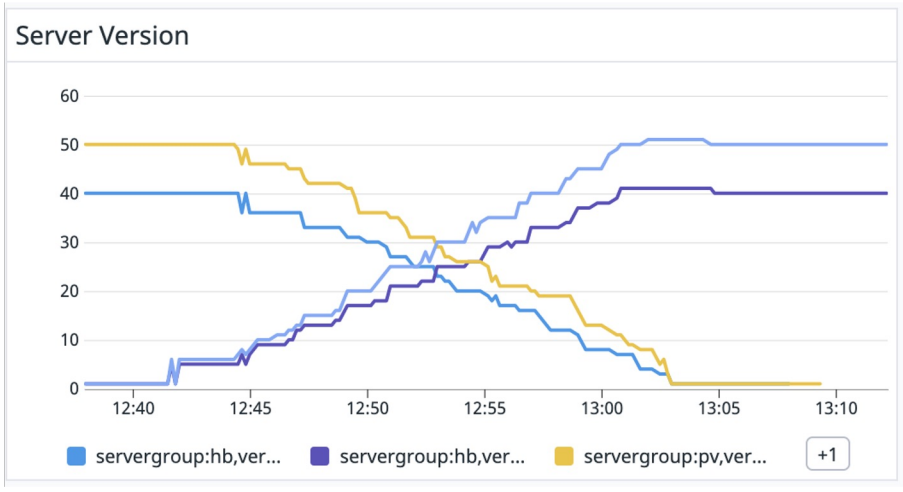
→ the new system is able to recover much faster!

→ the new system is much less sensitive to network issues!

Production Experience

Since February 24, 2022

- Release went smooth
- No more network-related incidents
- Rolling updates are much faster
- Allowed us to enable auto-scaling → lots of savings



Shardcake

Open source library

Shardcake

Entity Sharding library for Scala

Get Started →

Single writer pattern

Ensure that each entity exists in only one place at the time across all your servers.

Location transparency

Communicate with your remote entities using their ID, regardless of their physical location.

Customization

Bring your own storage, serialization, messaging protocol... or use the ones provided.

Example (1/3)

Define a message type

```
sealed trait GuildMessage

object GuildMessage {
  case class Join(userId: String, replier: Replier[Try[Set[String]]]) extends GuildMessage
  case class Timeout(replier: Replier[Try[Set[String]]])              extends GuildMessage
  case class Leave(userId: String)                                    extends GuildMessage
}

object Guild extends EntityType[GuildMessage]("guild")
```

Example (2/3)

Define a behaviour

```
def behavior(entityId: String, messages: Dequeue[GuildMessage]): RIO[Sharding, Nothing] =
  Ref
    .make(Set.empty[String])
    .flatMap(state => messages.take.flatMap(handleMessage(state, _)).forever)
```

Example (3/3)

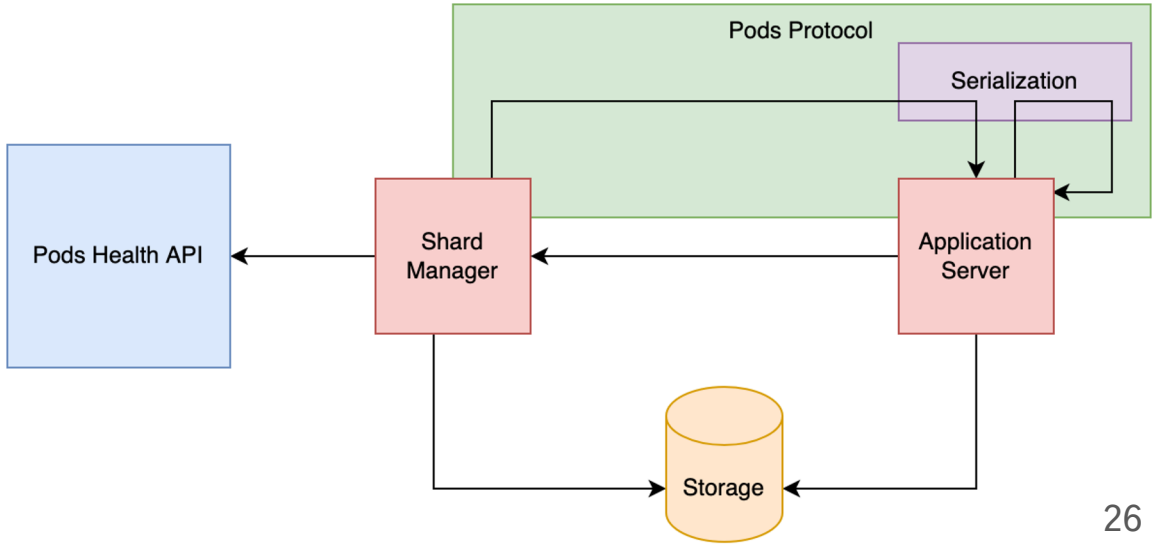
Register the behavior and send messages to entities

```
for {
  _     <- Sharding.registerEntity(Guild, behavior, p => Some(Terminate(p)))
  _     <- Sharding.registerScoped
  guild <- Sharding.messenger(Guild)
  user1 <- Random.nextUUID.map(_.toString)
  user2 <- Random.nextUUID.map(_.toString)
  user3 <- Random.nextUUID.map(_.toString)
  _     <- guild.send("guild1")(Join(user1, _)).debug
  _     <- guild.send("guild1")(Join(user2, _)).debug
  _     <- guild.send("guild1")(Join(user3, _)).debug
  _     <- ZIO.never
} yield ()
```

Customizability

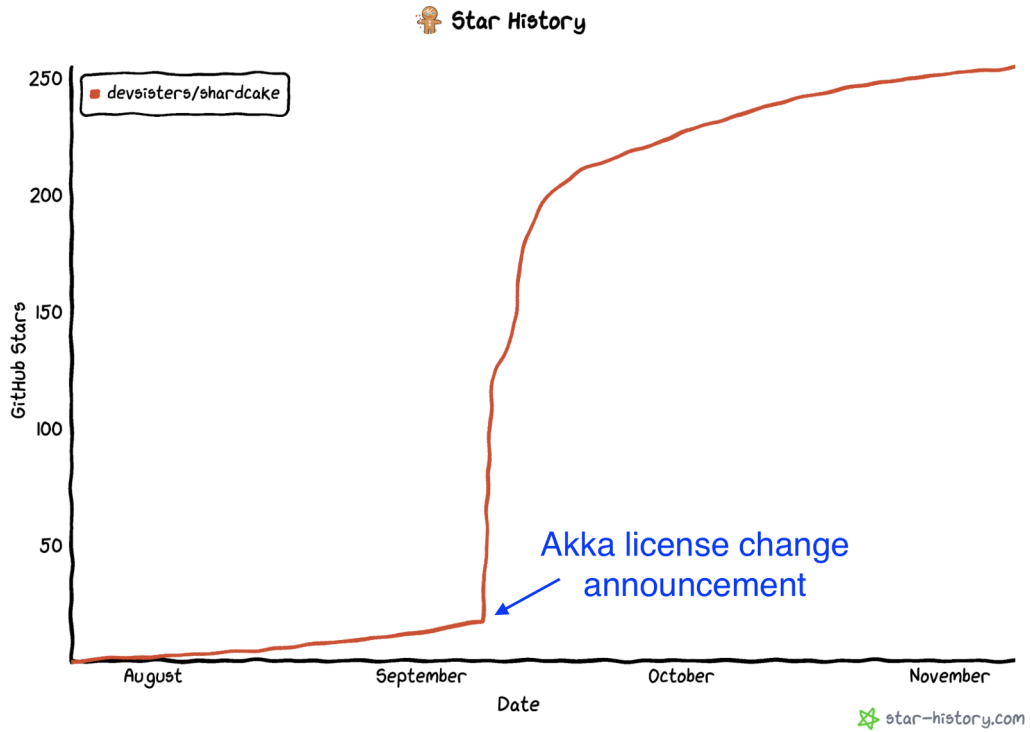
Use the default one or make your own

- Storage (default: Redis)
- Messaging Protocol (default: gRPC)
- Serialization (default: Kryo)
- Health API (default: K8s)



Star History

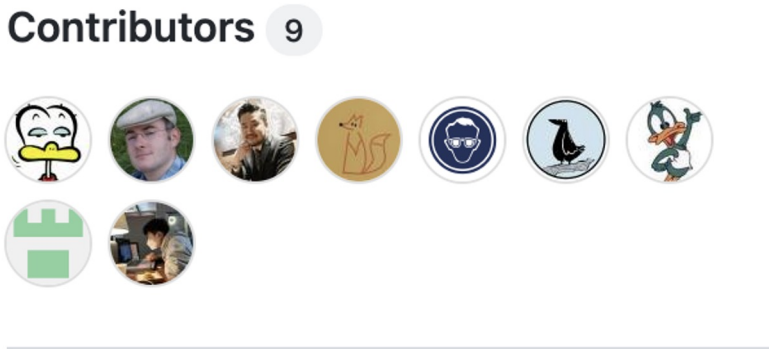
Thanks Lightbend!



Recent additions

From new contributors 🙏

- Distributed PubSub (broadcast messages to all listening pods)
- Improved resilience and logging
- Ability to customize shard ID hash function



Thanks!

Questions?

Find me online at @ghostdogpr

DEVSISTERS