Stateful Architectures in Akka

Nick Isaacs
@jorge_distress
Sr. VP Fungineering @
VictorOps

What Is This Talk?

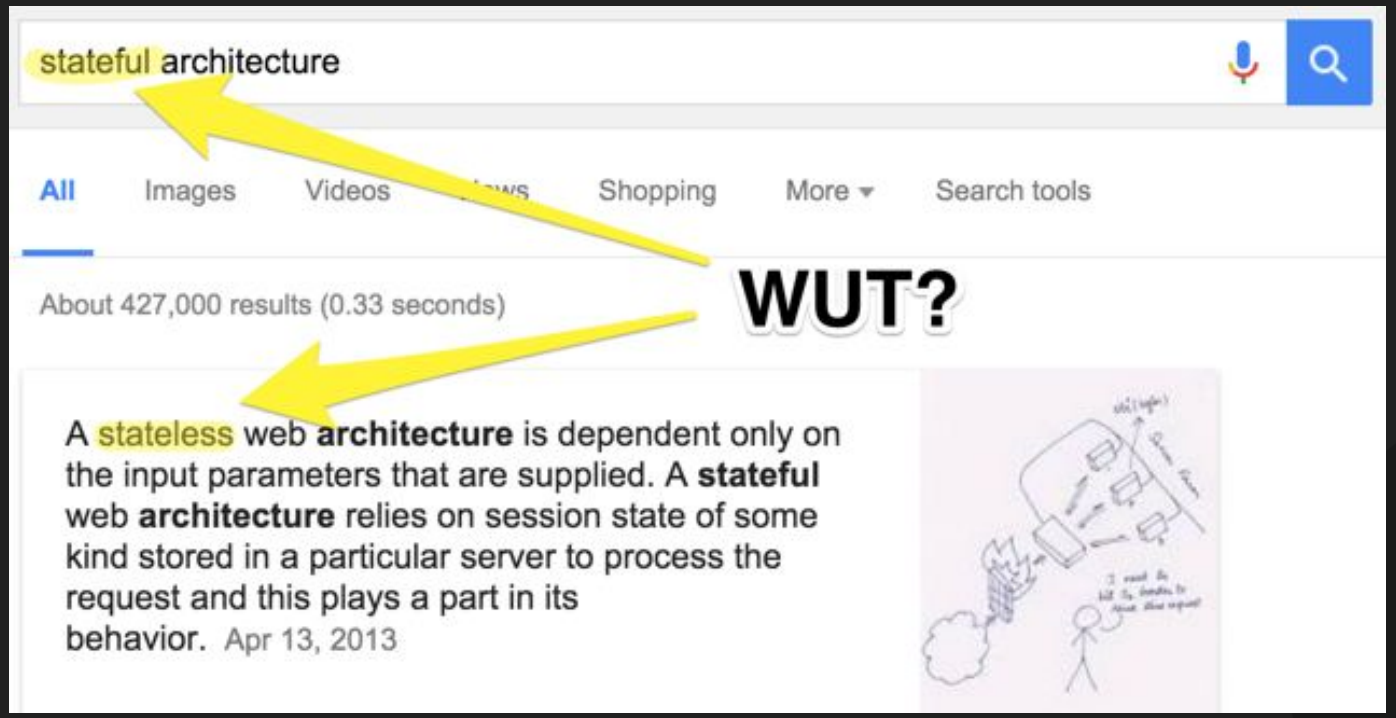
- Anecdotal
- Informational
- A work in progress
- First steps

Why


Computers
VictorOps
Users

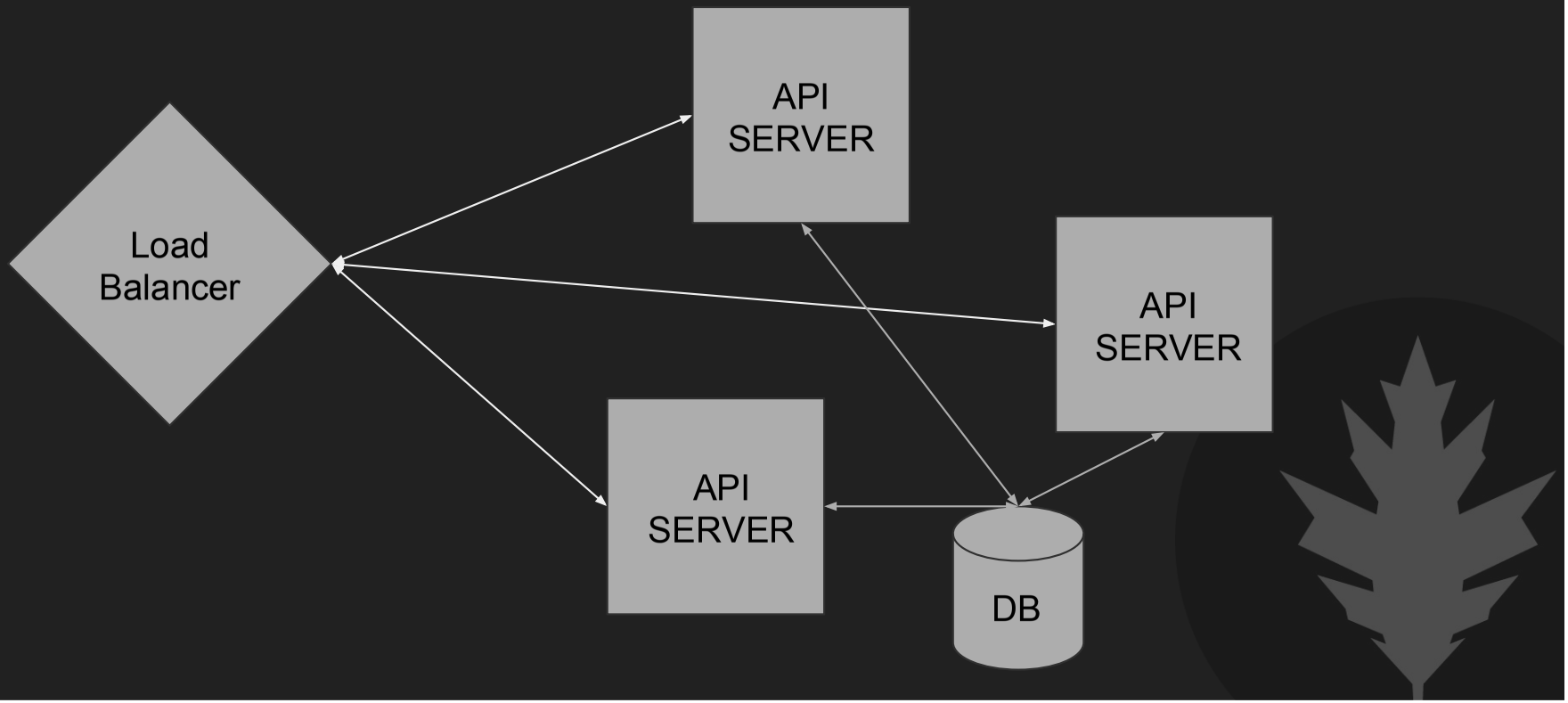
Stateful Architecture

Stateful what?



Stateless Architectures

A typical load balanced API server cluster



Stateless Request Flow

1. Hit load balancer
2. Hit API box
3. Lookup state from persistence layer
4. ???
5. Respond
6. Profit

Stateless Request Flow

1. Hit load balancer
2. Hit API box
3. Lookup state from persistence layer
4. ???
5. Respond
6. Profit

Stateful Request Flow

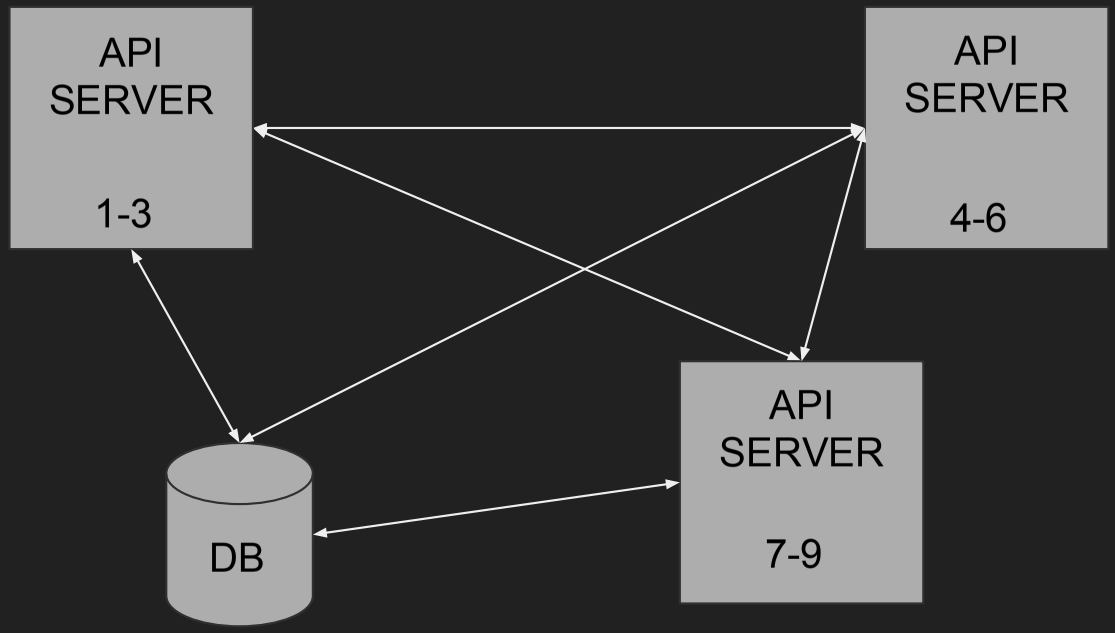
1. Hit load balancer
2. Hit API box
3. ~~Lookup state from persistence layer~~
4. ???
5. Respond
6. Profit

Strategies

- Ephemeral applications (avoid state)
- Caching
- Sharding

Stateful Architecture

A stateful API cluster. Each node has a subset of the application domain.



Sharding Applications

- Keeping data/logic in a known place
- Membership
- Consistent hashing
- Routing
- Coordination

What is Akka?

- Actor implementation on the JVM
- Actors send and receive messages
- Actors can “become” a new state between messages
- Actors are “islands of state”
- You cannot directly access an Actor’s state
- Many great intro talks/posts available

Akka to the Rescue

- Fantastic abstraction of state
- Persistent actors
- Rich message delivery semantics
- Simple message passing across the network
- Failure is a first class citizen

A Little Example

- TemperatureActor
- All of the state is managed in memory
- Fault tolerant, replayable log of our current state

https://github.com/nicky-isaacs/SCALE-14x-demo

1. Actor comes to life, time to read back the state

```
override def receiveRecover: Receive = {
    case SnapshotOffer(_, TempuratureSnapshot(t)) =>
        this.currentTempurature = Some(t)

    case cmd :TempuratureCmd =>
        updateState(cmd)
}
```

2. Actor is recovered, time to work

```
override def receiveCommand: Receive = {
    case GetTempurautre =>
        this.currentTempurautre match {
            case Some(t) =>
                sender() ! t

            case _ =>
                sender ! AkkaFalilure(new TempuratureException("No tempurature to get"))
        }

    case cmd: TempuratureCmd =>
        persist(cmd){ persistedCmd =>
            if (shouldSnapshot()) {
                this.currentTempurature foreach { t =>
                    snapshotAndSweep(TempuratureSnapshot(t))
                }
            }

            updateState(persistedCmd) match {
                case Success(_) =>
                    sender() ! AkkaSuccess(())

                case Failure(err) =>
                    sender() ! AkkaFailure(err)
            }
        }
}
```

Finite State Machine (thanks Akka)

- Recovering
  - No messages from the outside world
  - Reading snapshots and journaled messages
  - Getting back to last known state

- Receiving
  - Ready to respond to the world
  - State changes go to disk asynchronously
  - writes are all in memory, performance++

How We Cluster Today

- Stateful sharding of applications
- Zookeeper based membership
- Consistently hashed by organization
- Single leader

Where We Could Go

- Leaderless clusters
- Gossip membership protocol
- Replication of work as cluster scales

Extra Credit

- Uber Ringpop
  - “application-layer sharding for Node.js applications”
- Akka Clustering
- Dynamo paper

“I regret nothing. The end.”

- Ron Swanson