Beyond FaaS
Towards Stateful Serverless

Jonas Bonér
@jboner
"We predict that Serverless Computing will grow to dominate the future of Cloud Computing."

- Berkeley CS Department
FaaS = Function-as-a-Service
FaaS = Function-as-a-Service

FaaS IS VISIONARY
FaaS IS VISIONARY
PAVED THE WAY

FaaS = Function-as-a-Service
FaaS is visionary
Paved the way
Just the first step
FaaS = Function-as-a-Service
SERVERLESS ≠ FAAS
GOOD USE-CASES FOR FAAS?
Good Use-Cases for FaaS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window.
GOOD USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window:

1. Embarrassingly parallel processing tasks—invoked on demand & intermittently, examples include: image processing, object recognition, log analysis
2. Low traffic applications—enterprise IT services, and spiky workloads
3. Stateless web applications—serving static content form S3 (or similar)
4. Orchestration functions—integration/coordination of calls to third-party services
5. Composing chains of functions—stateless workflow management, connected via data dependencies
6. Job scheduling—CRON jobs, triggers, etc.
FAAS: HARD TO BUILD
GENERAL-PURPOSE APPLICATIONS
1. Functions are stateless, ephemeral, short-lived:
   expensive to lose computational context & rehydrate
2. Durable state is always “somewhere else”
3. No co-location of state and processing
4. No direct addressability—all communication over external storage
5. Limited options for managing & coordinating distributed state
6. Limited options for modelling data consistency guarantees
STATE
We Need Serverless Support For...

- Managing in-memory durable session state across individual requests
  E.g. User Sessions, Shopping Carts, Caching
- Low-latency serving of dynamic in-memory models
  E.g. Serving of Machine Learning Models
- Real-time stream processing
  E.g. Recommendation, Anomaly Detection, Prediction Serving
- Distributed resilient transactional workflows
  E.g. Saga Pattern, Workflow Orchestration, Rollback/Compensating Actions
- Shared collaborative workspaces
  E.g. Collaborative Document Editing, Blackboards, Chat Rooms
- Leader election, counting, voting
  ...and other distributed systems patterns/protocols for coordination
1. Stateful long-lived addressable virtual components
   Actors
Technical Requirements

1. Stateful long-lived addressable virtual components
   Actors

2. Options for distributed coordination and communication patterns
   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.
Technical Requirements

1. Stateful long-lived addressable virtual components
   Actors

2. Options for distributed coordination and communication patterns
   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

3. Options for managing distributed state reliably at scale
   Ranging from strong to eventual consistency (durable/ephemeral)
Technical Requirements

1. Stateful long-lived addressable virtual components
   Actors

2. Options for distributed coordination and communication patterns
   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

3. Options for managing distributed state reliably at scale
   Ranging from strong to eventual consistency (durable/ephemeral)

4. Intelligent adaptive placement of stateful functions
   Physical co-location of state and processing, sharding, and sticky routing
1. Stateful long-lived addressable virtual components
   Actors
2. Options for distributed coordination and communication patterns
   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.
3. Options for managing distributed state reliably at scale
   Ranging from strong to eventual consistency (durable/ephemeral)
4. Intelligent adaptive placement of stateful functions
   Physical co-location of state and processing, sharding, and sticky routing
5. Predictable performance, latency, and throughput
   In startup time, communication/coordination, and storage of data
Technical Requirements

1. Stateful long-lived addressable virtual components
   Actors

2. Options for distributed coordination and communication patterns
   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

3. Options for managing distributed state reliably at scale
   Ranging from strong to eventual consistency (durable/ephemeral)

4. Intelligent adaptive placement of stateful functions
   Physical co-location of state and processing, sharding, and sticky routing

5. Predictable performance, latency, and throughput
   In startup time, communication/coordination, and storage of data

6. Ways of managing end-to-end guarantees and correctness
FaaS
Abstracting Over Communication
FaaS
Abstracting Over Communication

MESSAGE IN

DEPLOYMENT

USER FUNCTION
FaaS
Abstracting Over Communication

MESSAGE IN -> USER FUNCTION -> MESSAGE OUT
Faas With CRUD
Not Serverless
In An Ideal World
UNCONSTRAINED DATABASE ACCESS MAKES IT HARD TO AUTOMATE OPERATIONS
Enter Stateful Serverless
1. **Embrace State**—Don’t ignore, hide, or delegate it

Data locality matters. Faster insight into data is a competitive advantage.
1. Embrace State—Don’t ignore, hide, or delegate it
   Data locality matters. Faster insight into data is a competitive advantage.

   Bulkhead and Contain. Signal and Die. Supervise and Manage.
1. **Embrace State**—Don’t ignore, hide, or delegate it
   Data locality matters. Faster insight into data is a competitive advantage.

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. **Embrace Uncertainty**—Manage it in the application layer
   End-to-end correctness/stability requires app working in concert w/ infra.
1. **Embrace State**—Don’t ignore, hide, or delegate it
   
   Data locality matters. Faster insight into data is a competitive advantage.

   
   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. **Embrace Uncertainty**—Manage it in the application layer
   
   End-to-end correctness/stability requires app working in concert w/ infra.

4. **Avoid Needless Consistency**
   
   Not all data have need the same guarantees. Start with zero, add as needed.
1. **Embrace State**—Don’t ignore, hide, or delegate it
   Data locality matters. Faster insight into data is a competitive advantage.

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. **Embrace Uncertainty**—Manage it in the application layer
   End-to-end correctness/stability requires app working in concert w/ infra.

4. **Avoid Needless Consistency**
   Not all data have need the same guarantees. Start with zero, add as needed.

5. **Avoid Needless Coordination and Communication**
   Silence is Golden. Favour Eventual Consistency, CALM, CRDTs, ACID 2.0.
1. **Embrace State**—Don’t ignore, hide, or delegate it
   Data locality matters. Faster insight into data is a competitive advantage.

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. **Embrace Uncertainty**—Manage it in the application layer
   End-to-end correctness/stability requires app working in concert w/ infra.

4. **Avoid Needless Consistency**
   Not all data have need the same guarantees. Start with zero, add as needed.

5. **Avoid Needless Coordination and Communication**
   Silence is Golden. Favour Eventual Consistency, CALM, CRDTs, ACID 2.0.

6. **Avoid Coupling in Time and Space**
Stateful Serverless
Abstracting Over State

MESSAGE IN

DEPLOYMENT

USER FUNCTION/ENTITY

MESSAGE OUT
Stateful Serverless
Abstracting Over State

- Message In
- User Function/Entity
- Message Out
- State In

Deployment
Stateful Serverless
Abstracting Over State

MESSAGE IN

MESSAGE OUT

STATE IN

STATE OUT

DEPLOYMENT

USER FUNCTION/ENTITY
Let Us Use Better Models For Distributed State
Let Us Use Better Models
For Distributed State

A few battle-tested, yet constrained, models are:
Let Us Use Better Models For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing
Let Us Use Better Models For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing CRDTs
Let Us Use Better Models For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing  CRDTs  Key Value
Event Sourced Services

HAPPY PATH
Event
Sourced Services

Happy Path
Event Sourced Services

Happy Path
Event
Sourced Services

Happy Path
Event Sourced Services

Happy Path
Event
Sourced
Services

HAPPY PATH
Event Sourced Services

SAD PATH, RECOVER FROM FAILURE
Event
Sourced Services

EVENT LOG

SAD PATH, RECOVER FROM FAILURE
Event Sourced Services

SAD PATH, RECOVER FROM FAILURE
Event Log
REPLAY Events
Sourced Services

COMMAND

EVENT LOG → REPLAY EVENTS

SAD PATH, RECOVER FROM FAILURE
Serverless

Event Sourcing

DEPLOYMENT
Serverless
Event Sourcing

Deployment

USER FUNCTION/ENTITY

EVENT LOG IN
Serverless Event Sourcing

Deployment

User Function/Entity

Command In

Event Log In
Serverless Event Sourcing

- Command In
- Event Log In
- Deployment
- User Function/Entity
- Reply Out
ACID 2.0
ACID 2.0

**Associative**
Batch-insensitive
(grouping doesn’t matter)

\[ a + (b + c) = (a + b) + c \]
ACID 2.0

**Associative**
Batch-insensitive (grouping doesn't matter)

\[a + (b + c) = (a + b) + c\]

**Commutative**
Order-insensitive (order doesn't matter)

\[a + b = b + a\]
ACID 2.0

**Associative**
Batch-insensitive (grouping doesn't matter)
\[ a + (b + c) = (a + b) + c \]

**Commutative**
Order-insensitive (order doesn't matter)
\[ a + b = b + a \]

**Idempotent**
Retransmission-insensitive (duplication does not matter)
\[ a + a = a \]
CONFLICT-FREE REPPLICATED DATA TYPES
CONFLICT-FREE REPLICATED DATA TYPES

CRDT

ACID 2.0
Strong Eventual Consistency
Replicated & Decentralized
Always Converge Correctly
Monotonic Merge Function
Highly Available & Scalable

Convergent & Commutative Replicated Data Types - Shapiro et. al. 2011
CONFLICT-FREE REPLICATED DATA TYPES

CRDT DATA TYPES

ACID 2.0
Strong Eventual Consistency
Replicated & Decentralized
Always Converge Correctly
Monotonic Merge Function
Highly Available & Scalable

Counters
Registers
Sets
Maps
Graphs

(that all compose)

Convergent & Commutative Replicated Data Types - Shapiro et. al. 2011
Serverless CRDTs
Serverless CRDTs

Deployment

User Function/Entity

States/Deltas In
Serverless CRDTs

Deployment

Message In

User Function/Entity

States/Deltas In
Serverless CRDTs

Message In
User Function/Entity
Message Out
States/Deltas In
Deployment
States/Deltas Out
Serverless CRUD

Deployment

User Function/Entity

Snapshot In (By Entity Key)
Serverless CRUD

- Message In
- Snapshot In (By Entity Key)

Deployment

User Function/Entity
Serverless CRUD

MESSAGE IN

MESSAGE OUT

SNAPSHOT IN (BY ENTITY KEY)

USER FUNCTION/ENTITY

DEPLOYMENT
Serverless CRUD

- Message In
- Message Out
- Snapshot In (By Entity Key)
- Snapshot Out (By Entity Key)
Introducing cloudstate
WHAT IS CLOUDSTATE?

https://cloudstate.io
WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:
WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
WHAT IS CLOUDSTATE?
https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages
Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

WHAT IS CLOUDSTATE?

https://cloudstate.io
WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

• Polyglot: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works
WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

• Polyglot: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works
• PolyState: Powerful state model—support for Event Sourcing, CRDTs, Key Value

What Is CloudState?
https://cloudstate.io
WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project — defining a specification, protocol, and TCK
2. Reference Implementation — backend + client APIs in different languages

Highlights:

• Polyglot: Client libs in JavaScript, Java, Go — Python, .NET, Swift, Scala in the works
• PolyState: Powerful state model — support for Event Sourcing, CRDTs, Key Value
• PolyDB: Supporting SQL, NoSQL, NewSQL, and in-memory replication
Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

- **Polyglot**: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works
- **PolyState**: Powerful state model—support for Event Sourcing, CRDTs, Key Value
- **PolyDB**: Supporting SQL, NoSQL, NewSQL, and in-memory replication
- **Open Source** leveraging Akka, gRPC, Knative, GraalVM, running on Kubernetes

___

WHAT IS CLOUDSTATE?

https://cloudstate.io
SERVING OF STATEFUL FUNCTIONS
SERVING OF STATEFUL FUNCTIONS

KUBERNETES POD

KUBERNETES POD

KUBERNETES POD
SERVING OF STATEFUL FUNCTIONS

- USER FUNCTION (JavaScript, Go, Java, …)
  - Kubernetes Pod

- USER FUNCTION (JavaScript, Go, Java, …)
  - Kubernetes Pod

- USER FUNCTION (JavaScript, Go, Java, …)
  - Kubernetes Pod
SERVING OF STATEFUL FUNCTIONS

- **CloudState Proxy**
- **User Function** (JavaScript, Go, Java, ...)
- **Kubernetes Pod**
- **User Function** (JavaScript, Go, Java, ...)
- **Kubernetes Pod**
- **User Function** (JavaScript, Go, Java, ...)
- **Kubernetes Pod**
SERVING OF STATEFUL FUNCTIONS

CLOUDSTATE PROXY

USER FUNCTION
(JavaScript, Go, Java, ...)
KUBERNETES POD

USER FUNCTION
(JavaScript, Go, Java, ...)
KUBERNETES POD

USER FUNCTION
(JavaScript, Go, Java, ...)
KUBERNETES POD
SERVING OF STATEFUL FUNCTIONS

CLOUDSTATE PROXY

GRPC

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD
SERVING OF STATEFUL FUNCTIONS

- Cloudstate Proxy
- GRPC
- User Function (JavaScript, Go, Java,…)
- Kubernetes Pod
- User Function (JavaScript, Go, Java,…)
- Kubernetes Pod
- User Function (JavaScript, Go, Java,…)
- Kubernetes Pod
- Datastore (Cassandra, Postgres, Spanner,…)

DATASTORE

KUBERNETES POD
POWERED BY AKKA CLUSTER SIDECARs

CLOUDSTATE PROXY

USER FUNCTION
(JavaScript, Go, Java, ...)

KUBERNETES POD

USER FUNCTION
(JavaScript, Go, Java, ...)

KUBERNETES POD

USER FUNCTION
(JavaScript, Go, Java, ...)

KUBERNETES POD

DATASTORE
(Cassandra, Postgres, Spanner, ...)

Powered by Akka Cluster Sidecars
POWERED BY AKKA CLUSTER SIDECARs

CLOUDSTATE PROXY

AKKA SIDECAR

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD

AKKA SIDECAR

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD

AKKA SIDECAR

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD

AKKA SIDECAR

USER FUNCTION
(JavaScript, Go, Java, …)

KUBERNETES POD

DATASTORE
(Cassandra, Postgres, Spanner, …)
POWERED BY AKKA CLUSTER SIDE CARS

Cloudstate Proxy

Akka Sidecar

Akka Sidecar

Akka Sidecar

Akka Sidecar

Kubernetes Pod

GRPC

User Function

User Function

User Function

Kubernetes Pod

Datastore

(Cassandra, Postgres, Spanner, ...)

Powered by Akka Cluster Sidecars

*User Function*

(JavaScript, Go, Java, ...)
POWERED BY AKKA CLUSTER SIDECARS

CLOUDSTATE PROXY

AKKA CLUSTER

AKKA SIDECAR → GRPC → USER FUNCTION (JavaScript, Go, Java, ...)
  ↓  ↓  ↓
KUBERNETES POD

AKKA SIDECAR → GRPC → USER FUNCTION (JavaScript, Go, Java, ...)
  ↓  ↓  ↓
KUBERNETES POD

AKKA SIDECAR → GRPC → USER FUNCTION (JavaScript, Go, Java, ...)
  ↓  ↓  ↓
KUBERNETES POD

DATASTORE (Cassandra, Postgres, Spanner, ...)

Powered by Akka Cluster Sidecars

Cloudstate proxy

Akka Cluster

Akka Sidecar

User Function (JavaScript, Go, Java, ...)

Kubernetes Pod

Datastore (Cassandra, Postgres, Spanner, ...)

gRPC
POWERED BY AKKA CLUSTER SIDECAR

[Diagram of a network architecture involving Cloudstate proxy, Akka cluster, Akka sidecar, gRPC, User function (JavaScript, Go, Java,...), Kubernetes pod, and Datastore (Cassandra, Postgres, Spanner,...)]
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
AKKA CLUSTER STATE MANAGEMENT

• Actor-based Distributed Runtime
• Dynamo-style Node Ring
• Decentralized Masterless P2P
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
**AKKA CLUSTER STATE MANAGEMENT**

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log

**AKKA CLUSTER**

![Diagram of AKKA Cluster State Management](image-url)
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing
Akka Cluster State Management

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

![Diagram of Akka Cluster State Management](image-url)
Akka Cluster State Management

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

Diagram:
- Akka Cluster
- Akka Sidecar
- Event Log
- Key, State
AKKA CLUSTER STATE MANAGEMENT

AKKA CLUSTER

- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

AKKA SIDECAR

EVENT LOG
**AKKA CLUSTER STATE MANAGEMENT**

- **Actor-based Distributed Runtime**
- **Dynamo-style Node Ring**
- **Decentralized Masterless P2P**
- **Epidemic Gossiping, Self-healing**
- **State Sharding & Routing on Entity Key**
- **Forwarding of Requests (if needed)**
- **Co-Location of State & Processing**
- **Backed by Event Log**
- **Automatic Failover, Rehydration, and Rebalancing**

**https://akka.io**
In Summary
In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let’s not stop here
In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let’s not stop here
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development
In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let’s not stop here
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development
4. We can’t ignore/delegate the hardest thing: State
In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let’s not stop here
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development
4. We can’t ignore/delegate the hardest thing: State
5. We think that Cloudstate shows the way