Towards Stateful Serverless

Sean Walsh

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

- BERKELEY CS DEPARTMENT
FaaS
IS VISIONARY
PAVED THE WAY
JUST THE FIRST STEP
FaaS = Function-as-a-Service
SERVERLESS ≠ 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 from 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.
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.


FAAS: HARD TO BUILD GENERAL-PURPOSE APPLICATIONS
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
- Avoiding impedance mismatch
  ...pure domain is rarely in “viewing” condition
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—Useful constraints.
FaaS Is Great At Abstracting Over Communication

(Message In) 

Deployment

(USER FUNCTION)

(Message Out)

(operational concerns implicit)
Not Serverless FaaS
Leaky Abstraction CRUD
The Problem

THE FUNCTION IS A BLACK BOX
The Problem

Unconstrained database access makes it hard to automate operations.
“Freedom is not so much the absence of restrictions as finding the right ones, the liberating restrictions.”

- TIMOTHY KELLER
Faas
Abstracting Over Communication
Stateful Serverless
Abstracting Over State

MESSAGE IN

USER FUNCTION

MESSAGE OUT

STATE IN

STATE OUT
cloudstate is a distributed, clustered and stateful cloud runtime, providing a zero-ops development experience, with polyglot client support, essentially serverless 2.0.

https://cloudstate.io
WHAT IS CLOUDSTATE?

https://cloudstate.io

Overview:

1. Open Source (Apache 2.0) project
2. Makes Stateful Serverless applications easy
3. Reference implementation for a standard (protocol and spec)
4. Let’s you focus on business logic, data model, and workflow
WHAT IS CLOUDSTATE?

https://cloudstate.io

Don’t worry about:

1. Managing: Complexities of Distributed and Concurrent systems
2. Managing: Distributed State—Consistency, Replication, Persistence
3. Managing: Databases, Service Meshes, and other infrastructure
4. Managing: Message Routing, Scalability, Fail-over & Recovery
5. Running & Operating your application
WHAT IS CLOUDSTATE?
https://cloudstate.io

Technical Highlights:

1. Polyglot: Client libs in JavaScript, Java, Go—with upcoming support for Python, .NET, Rust, Swift, Scala

2. PolyState: Powerful state models—Event Sourcing, CRDTs, Key Value

3. PolyDB: Supporting SQL, NoSQL, NewSQL and in-memory replication

4. Leveraging Akka, gRPC, Knative, GraalVM, running on Kubernetes
CLOUDSTATE HELPS YOU WITH
(when being a managed service)

• Pay-as-you-go:
  • On-demand Instance Creation, Passivation, and Failover
  • Autoscaling—up and down

• ZeroOps:
  • Automation of Message Routing and Delivery
  • Automation of State Management
    • Service of Record—In-Memory Cluster Sharding, Co-location of Data & Processing
    • Coordination State—Replication, Consistency
  • Automation of Deployment, Provisioning, Upgrades
AKKA CLUSTER STATE MANAGEMENT

- Actor-based Distributed Runtime
- 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
Cloudstate Uses Better Models For Distributed State

Battle-tested, yet constrained, models like:

Event Sourcing

CRDTs

CQRS (soon!)
Event Log

REPLAY EventS

SAD Path, RECOVER FROM FAILURE
Benefits of Event Sourcing

- One single **SOURCE OF TRUTH** with **ALL HISTORY**
- Allows for **MEMORY IMAGE** (Durable In-Memory State)
- Avoids the **OBJECT-RELATIONAL MISMATCH**
- Allows others to **SUBSCRIBE TO STATE CHANGES**
- Has good **MECHANICAL SYMPATHY** (Single Writer Principle)
Serverless Event Sourcing

Deployment

User Function/Entity

Command In

Event Log In

Reply Out

Events Out
CONFLICT-FREE REPPLICATED DATA TYPES

CRDT DATA TYPES

Counters
Registers
Sets
Maps
Graphs

(that all compose)

Strong Eventual Consistency
Deterministic by Design
Data Types Contain Resolution Logic
Always Converge Correctly, Eventually
Replicated & Decentralized
Highly Available & Very Scalable

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

**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)
\[ a + a = a \]
Serverless CRDTs

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

Message In
Message Out
Snapshot In (By Entity Key)
Snapshot Out (By Entity Key)
3 Tier Architecture

TOTAL RPS 100
ERROR RATE 0.00%

INGRESS

RRS 1100.0%
ERROR RATE 0.00%

RRS 275.0%
ERROR RATE 0.00%

RRS 275.0%
ERROR RATE 0.00%

RRS 275.0%
ERROR RATE 0.00%

RRS 275.0%
ERROR RATE 0.00%

POD-1
POD-2
POD-3
POD-4

DB-1
Reactive Architecture
Example

CRDT Entity

PRESENCE FUNCTION IN A CHAT APP

github.com/cloudstateio/samples-java-chat
@CrdtEntity
public class PresenceEntity {
    private final Vote vote; // Vote CRDT for this user. It's auto replicated
    // and keeps track how each node has voted
    private final String username; // Entity Key (for sharding and routing)

    public PresenceEntity(
        Optional<Vote> vote, CrdtCreationContext ctx, @EntityId String username) { … }

    public static void main(String... args) {
        new CloudState()
            .registerCrdtEntity(…)
            .start();
    }

    // Here we implement the Protobuf Service API, our business logic
    @CommandHandler
    public void connect(StreamedCommandContext<Empty> ctx) {
        vote.vote(true); // Set the user to online
        ctx.onCancel(canceled -> { // Register cancel callback for user disconnect
            vote.vote(false);
        });
        ...
    }

    @CommandHandler
    public OnlineStatus monitor(StreamedCommandContext<OnlineStatus> ctx) {
        ctx.onChange(change -> { // Subscribe to Vote CRDT changes
            ...
        });
        ...
    }
}
Example
Event Sourced Entity

HOME INTERNET DEVICES

github.com/sean-walsh/cloudstate-spring
Join Us

Try Out The Next Generation Stateful Serverless

cloudstate.io