THE LINUX FOUNDATION

OPEN SOURCE SUMMIT INDIA

Breaking Valkey on Purpose: Chaos Fuzzing With Agentic AI

Renuka Uttarala (She/Her)

WW Head - Data & AI Services, AWS

How do you test a distributed database designed to survive failures?

Unit tests miss the real bugs — the race conditions,
the moment a primary dies exactly when a replica is mid-sync.

Today we find those bugs — on purpose.

What is Valkey?

Open Source

BSD-3 Clause

Linux Foundation

In-Memory

200+ commands

Microsecond latency

Drop-in Replace

Redis OSS 7.2

API compatible

Community

AWS, Google, Oracle

Ericsson, Snap

Created March 2024 after Redis license change | Vendor neutral | Will remain open-source forever

Valkey at Scale

225+
Contributors

1B+
Requests/Second

150M+
Container Pulls

2.1M+
RPS Single Node (9.1)

41.4%
Memory Savings

230%
Throughput Improvement

Valkey 9.1 (May 2026): Real workload — 156M+ items, nodes from 21.2 GB to 12.5 GB per node

Cluster Mode Architecture

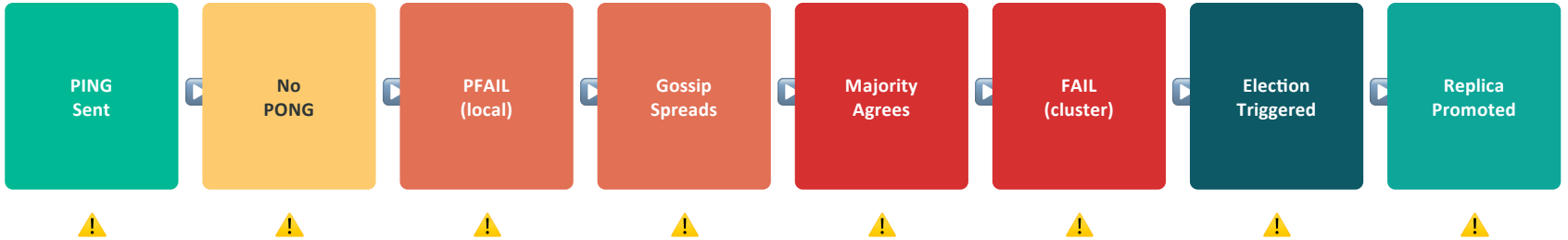
16,384 hash slots distributed across primaries

CLUSTER BUS — Full Mesh Gossip Protocol (TCP)

Membership Discovery • Heartbeats • Failure Agreement • Failover Coordination

Clients (cluster-aware, no proxy)

The Cluster Bus — Failure Detection Flow



Each transition = potential failure point the fuzzer targets

What if two primaries fail simultaneously during vote collection?

What if a node restarts during an election and still believes it's primary?

What if gossip messages arrive out of order after 4 rapid failovers?

Real Bugs Found — The Problem is Real

CRITICAL

Split-Brain

Two nodes both claim primary for same slots → data divergence

HIGH

Stuck Failover

Primary dies, slots never reassigned → partial outage

HIGH

Topology Divergence

Node rejoins as extra primary after SIGKILL → wrong ownership

None of these were caught by 1,000+ traditional integration tests. Found by the fuzzer.

The Gap

Integration Tests

- ✓ Deterministic
- ✓ Single failures
- ✓ Predictable timing
- ✓ Same result every run

Chaos Fuzzer + Agentic AI

Systematic exploration of combinatorial state space

Production

- 🔥 Non-deterministic
- 🔥 Overlapping failures
- 🔥 Real timing
- 🔥 Customers find bugs

What fills this gap? Let me show you.

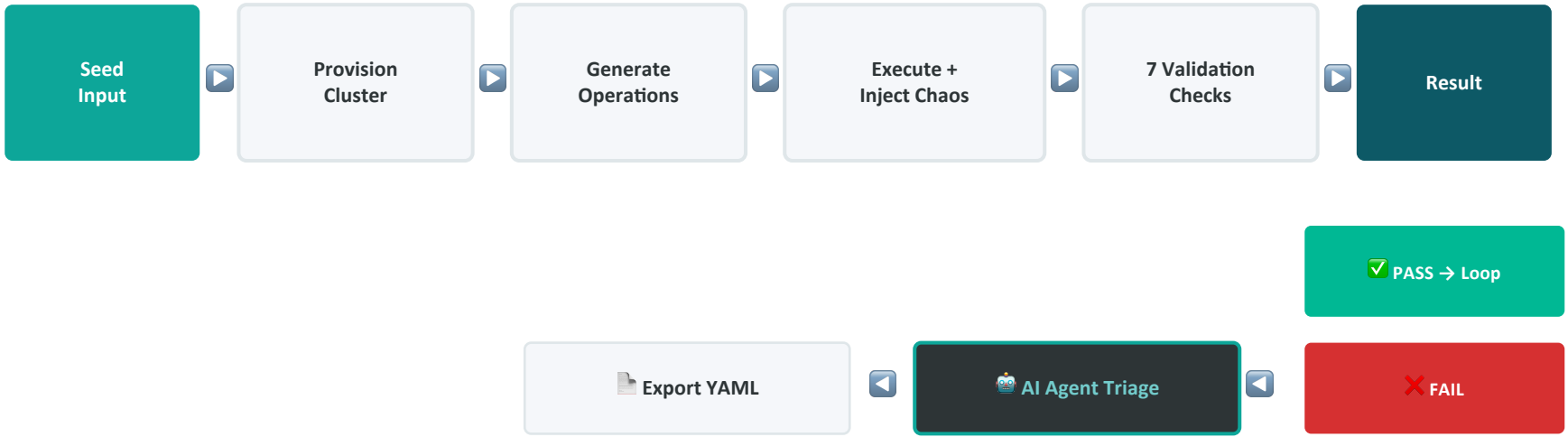
LIVE DEMO

Chaos vs. Valkey: Can it survive?

1	2	3	4	5
✅	✅	🔄	❌	🏆
Simple Failover	**Multi-Chaos**	**Reproducibility**	**Quorum Failure + AI**	**Bug Hall of Fame**
Happy path — 3 shards	SIGTERM + SIGKILL simultaneously	Same seed = same result	Seed 2028421952	Issue #85 lifecycle

`localhost:8502`

How It Works — Fuzzer Architecture



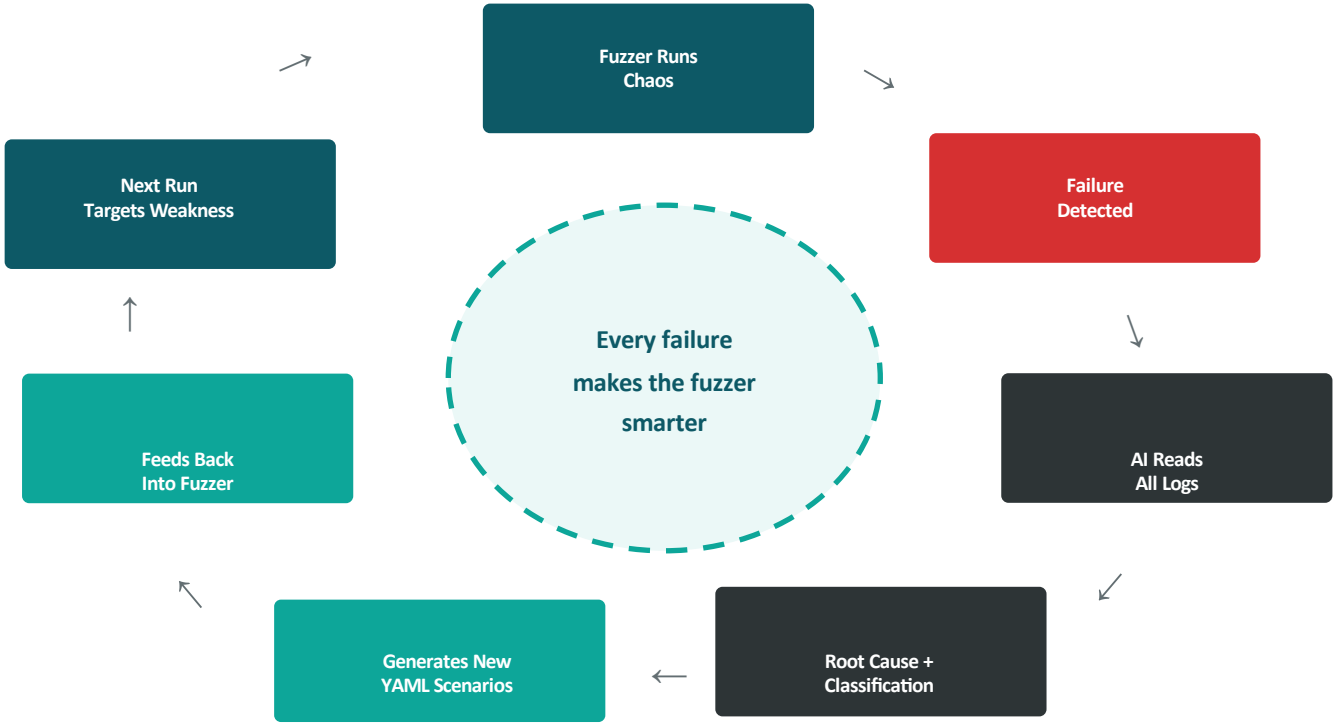
Single integer seed → same cluster topology, same operations, same timing, same result

The Seven Validation Checks

✓	✓	✓	✓	✓	✓	✓
📊	🧩	🏗️	🔗	👁️	💾	📝
Cluster Status	**Slot Coverage**	**Topology**	**Replication**	**View Consistency**	**Data Consistency**	**Log Validation**
All nodes report "ok"	All 16,384 slots assigned	Primary/replica count correct	Replicas connected lag < 3s	No split-brain all nodes agree	100 test keys survive chaos	No PANIC or assertions

Seed 2028421952: passed 6 of 7 checks — only quorum failed → AI classified as validator timing issue, not Valkey bug

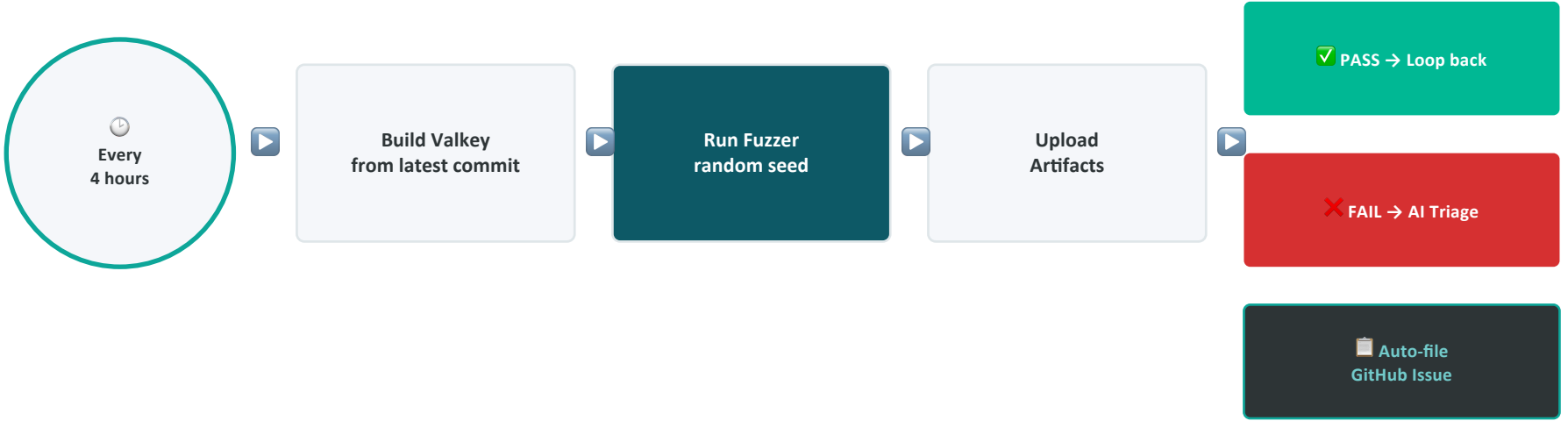
The Agentic AI Loop



Without AI: 500 lines of logs, good luck

With AI: 1 actionable paragraph + root cause + YAML

CI/CD — Continuous Safety Net



Issue #85 was found by the daily fuzzer at 3 AM, March 2026. Nobody was looking. The fuzzer found it.

YAML DSL — From Random to Reproducible


```yaml
scenario:
  name: chaos_demo
  cluster:
    num_shards: 4
    replicas_per_shard: 1

  operations:
    - type: failover
      shard: 1
      delay: 0
    - type: failover
      shard: 2
      delay: 2

  chaos:
    - type: sigterm
      target: replica
      after_operation: 1
    - type: sigkill
      target: primary
      after_operation: 2
```

▶

Git Commit

Permanent regression test
in source control

Bug Report

Attach to GitHub issue
for reproduction

CI/CD Pipeline

Run in automated
continuous testing

Random fuzzer run → one-click export to YAML → permanent regression test

Chaos Engineering & Resiliency Best Practices

What we learned running the valkey-fuzzer against Valkey clusters

Pattern	What the Fuzzer Teaches
Multi-AZ Always On	Single-AZ failure takes out primary + replica — breaks quorum
3+ Shards Minimum	2 shards + partition = no majority vote → stuck failover
Force Failover Testing	Fuzzer found stuck failovers in configs that passed all integration tests
Connection Pool Drain	SIGTERM handling critical — 5-10s drain before switching endpoints
Chaos Test Before Scale	Timing bugs hide at low load, emerge under concurrent operations

What would YOU fuzz next?

Network Partitions

Split the cluster bus

Packet Drops

Partial communication

Resharding

Under active chaos

Scale In/Out

During failures

Memory Pressure

OOM kills

Clock Skew

Between nodes

Get Involved

Try It

pip install valkey-fuzzer

Find bugs in minutes

Break It

Run fuzzer → open issues

when checks fail

Build It

Add operations, chaos types,

and validations

github.com/valkey-io/valkey-fuzzer

Every contribution makes Valkey more reliable for millions of users

Questions?

Key Takeaways

1. Chaos finds bugs tests miss → Seed 3954605966 found Issue #85
2. Every failure is reproducible → Seeds = repeatable science
3. AI closes the loop → Logs → root cause → new scenarios

valkey-fuzzer

github.com/valkey-io/valkey-fuzzer

Connect with me

🔗linkedin.com/in/renuka-uttarala

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT INDIA