Chaos Engineering in Action: Enhancing Resilience in Strimzi

Maros Orsak
Senior Software Quality Engineer Redhat
Immortal Dota 2

Henrich Zrncik
Software Quality Engineer RedHat
Warlock lvl 70

Quiz
1. Chaos Engineering
2. Target Systems
3. Designing Chaos (Experiments)
4. Demo (Simplified)
5. Conclusion
System’s resilience
System’s resilience

Components

1. Application
2. Components
3. Other Services

Application

Other Services
- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.

"Nonsense...And what's more, it doesn't rhyme. All decent predictions rhyme."
- G. of R.
Chaos Origin
Experimenting on a system in order to build confidence in the system’s capability to withstand turbulent conditions in production.
Experimenting on a system in order to build confidence in the system’s capability to withstand turbulent conditions in production.
Experimenting on a system in order to build confidence in the system’s capability to withstand turbulent conditions in production.
Principles

Minimal blast radius
Principles

- Minimal blast radius
- Run in production
- Hypothesis around steady state
- Wary real world's events
Principles

Minimal blast radius
Run in production
Hypothesis around steady state
Wary real world’s events
Automatized Continuous run
Architectural shift (timeline)

Low complexity

Monolith
Architectural shift (timeline)

Hard to scale horizontally

Monolith
Architectural shift (timeline)

Monolith

Hard to scale horizontally
Architectural shift (timeline)

- Fault tolerant

Monolith
**Architectural shift (timeline)**

- **Monolith**
  - + low complexity
  - - load balancing
  - - hard to scale horizontally
  - - fault tolerant, cost

- **Docker**
  - + portability, testing
  - + - horizontal scaling
  - - complexity increases
  - - management deployments

**Isolation**

**Complexity increases**

**Portability**

**Horizontal scaling**

**Wrapping up**
Architectural shift (timeline)

- **Monolith**
  - + low complexity
  - - load balancing
  - - hard to scale horizontally
  - - fault tolerant, cost

- **Docker**
  - + portability, testing
  - + - horizontal scaling
  - - complexity increases
  - - management deployments

- **Kubernetes**
  - easy to scale horizontally
  - + management of resources
  - - complexity increases again!
Monolith

- low complexity
- load balancing
- hard to scale horizontally
- fault tolerant, cost

Docker

- portability, testing
- horizontal scaling
- complexity increases
- management deployments

Kubernetes

- easy to scale horizontally
- management of resources
- complexity increases again!

Architectural shift (timeline)
Architectural shift (timeline)

Monolith
- low complexity
- load balancing
- hard to scale horizontally
- fault tolerant, cost

Docker
- portability, testing
- horizontal scaling
- complexity increases
- management deployments

Kubernetes
- easy to scale horizontally
- management of resources
- complexity increases again!

fault tolerant

Replicas: 3

Chaos Engineering
Target System
Design Chaos
Demo
Wrapping up
Architectural shift (timeline)

- Monolith
  + low complexity
  - load balancing
  - hard to scale horizontally
  - fault tolerant, cost

- Docker
  + portability, testing
  + horizontal scaling
  - complexity increases
  - management deployments

- Kubernetes
  + easy to scale horizontally
  + management of resources
  - complexity increases again!

Complexity increases again!
**Architectural shift (timeline)**

**Monolith**
- + low complexity
- + load balancing
- - hard to scale horizontally
- - fault tolerant, cost

**Docker**
- + portability, testing
- + horizontal scaling
- - complexity increases
- - management deployments

**Kubernetes**
- + easy to scale horizontally
- + management of resources
- + fault tolerant
- - complexity increases again!

**Operators**
- No single person can grasp the entire system!
**Architectural shift (timeline)**

- **Monolith**
  - + low complexity
  - - load balancing
  - - hard to scale horizontally
  - - fault tolerant, cost

- **Docker**
  - + portability, testing
  - - horizontal scaling
  - - complexity increases
  - - management deployments

- **Kubernetes**
  - + easy to scale horizontally
  - + management of resources
  - + fault tolerant
  - - complexity increases again!

- **Operators**
  - No single person can grasp the entire system!

where one of such operators is…
Strimzi

Simplifies upgrades of Kafka clusters

Simplifies upgrades of Kafka clusters

Horizontal Scaling

Dynamic configuration

Tracing

Security

Grafana dashboards

Cloud Native Computing Foundation
Too much unknowns...right...let’s break this down...
Apache Kafka

- Messaging system
- Publish subscribe model
- Distributed event streaming platform
- Commit log service
- Fault tolerant

Target System
Design Chaos
Demo
Wrapping up
Messaging system
Commit log service
Publish subscribe model
distributed event streaming platform
Fault tolerant
Apache Kafka
Chaos
Target System
Design Chaos
Demo
Wrapping up
This does not help…
So let’s move to basics of the Kafka…
Apache Kafka

Producers

Kafka broker

Another Kafka broker

Consumers
Apache Kafka

Producer

Kafka broker

Consumer

Kafka

Topics

server.config
Apache Kafka

- Preferred leader
- Quorum
- Controller nodes
- Follower
- Consumer groups
- Leader election
- Kafka Streams
- Kafka Connect
- ZooKeeper-based Kafka
- Replication factor
- Replication factor
- Kafka Mirror Maker
- And more…

Chaos Engineering
Target System
Design Chaos
Demo
Wrapping up
Apache Kafka

Producers

Consumers

Kafka broker

What if we?
encapsulate such system in Kubernetes

Operators managing Kafka ecosystem (Kafka clusters)
Chaos Engineering | Target System | Design Chaos | Demo | Wrapping up

encapsulate such system in Kubernetes Operators managing Kafka ecosystem (Kafka clusters)

Strimzi

Chaos Engineering

Target System

Design Chaos

Demo

Wrapping up
Complexity is really high

Strimzi

Topic and User operator
Cluster operator
source Kafka cluster
Kafka Mirror Maker
target Kafka cluster
Kafka connect
Database

Chaos Engineering
Target System
Design Chaos
Demo
Wrapping up
Production environment for Strimzi and other projects.

thanks to these guys we are able to run Strimzi in testing production environment...
Chaos Experiment - Intuition

Kafka
MirrorMaker
Kafka Clients
Producers
Consumers
Streams
Databases
Kafka A
Kafka B
Kafka C
Kafka Connect
Kafka Clients
Consumers
Design Chaos
Chaos Engineering
Wrapping up
Target System
Demo
1. Observability

"Without observability, you don’t have ‘chaos engineering’. You just have chaos.” Charity M.
Chaos Experiment - Hypothesis

1. Observability
2. Hypothesis (search)
   - Critical components
   - Bottlenecks, network
   - Real world events
## Chaos Experiment - Document

### Component Config:

<table>
<thead>
<tr>
<th>Component</th>
<th>Config</th>
<th>Variables/Param</th>
<th>Chaos</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kafka</td>
<td>Config:</td>
<td>- Kraft</td>
<td>- Pod</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Zookeeper</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Params:</td>
<td>- Replicas</td>
<td>- Network</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Global configs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Ephemeral vs Persistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chaos:</td>
<td>- Pod</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Network</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Topic Operator Config:

<table>
<thead>
<tr>
<th>Topic Operator</th>
<th>Config:</th>
<th>Chaos:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unidirectional</td>
<td>Network ...</td>
</tr>
<tr>
<td></td>
<td>Bidirectional</td>
<td></td>
</tr>
</tbody>
</table>

### Clients Config:

<table>
<thead>
<tr>
<th>Clients</th>
<th>Config:</th>
<th>Chaos:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Producer</td>
<td>Http, Network, DNS</td>
</tr>
<tr>
<td></td>
<td>Consumer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Streams, Http</td>
<td></td>
</tr>
</tbody>
</table>

### Chaos:

<table>
<thead>
<tr>
<th>Bridge Chaos:</th>
<th>Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Http client only</td>
</tr>
</tbody>
</table>

### Infrastructure Chaos:

<table>
<thead>
<tr>
<th>Infrastructure Chaos:</th>
<th>Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td></td>
</tr>
</tbody>
</table>

### Mirror Maker & Kafka Connect

<table>
<thead>
<tr>
<th>Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Different DBs (MySQL, Mongo ...)</td>
</tr>
<tr>
<td>- Connector Type, Tasks/workers</td>
</tr>
</tbody>
</table>

### Zookeeper & Kraft

<table>
<thead>
<tr>
<th>Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Quorum necessary</td>
</tr>
<tr>
<td>- metadata</td>
</tr>
</tbody>
</table>

### Chaos:

<table>
<thead>
<tr>
<th>Variables/Param:</th>
</tr>
</thead>
<tbody>
<tr>
<td>m containers per pod in JVM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Chaos Experiment - Hypothesis

1. Observability
2. Hypothesis (search)
   - Critical components
   - Bottlenecks, network
   - Real world events
1. Observability
2. Hypothesis (formulate)
Chaos Experiment - Hypothesis

1. Observability
2. Hypothesis (formulate)
1. **Observed metrics:**
   a. Incoming traffic metrics
   b. Ready brokers
   c. CPU used, memory

2. **Hypothesis:** (Production system) Kafka cluster can withstand failure of 3 brokers without loss of messages or cascading fails.
1. **Observed metrics:**
   a. Incoming traffic metrics
   b. Ready brokers
   c. CPU used, memory

2. **Hypothesis:** (Production system) Kafka cluster can withstand failure of 3 brokers without loss of messages or cascading fails.
Chaos Experiment - Timeline

1. Observability
2. Hypothesis
3. Scale
Chaos Experiment - Timeline

1. Observability
2. Hypothesis
3. Scale

Chaos Engineering

Target System

Design Chaos

Demo

Wrapping up

Brokers up

3

t1

t
Chaos Experiment - Timeline

1. Observability
2. Hypothesis
3. Scale
Chaos Experiment - Timeline

1. Observability
2. Hypothesis
3. Scale
1. Observability
2. Hypothesis
3. Scale
Chaos Engineering

Target System

Design Chaos

Demo

Wrapping up

Chaos Experiment - Timeline

1. Observability
2. Hypothesis
3. Scale

Brokers up

Chaos Engineering

Target System

Design Chaos

Demo

Wrapping up

Chaos Engineering

Target System

Design Chaos

Demo

Wrapping up
<table>
<thead>
<tr>
<th>t</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg/s</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

---

**Chaos Experiment - Scale**

1. Observability
2. Hypothesis
3. Scale
Chaos Experiment - Tools

1. Observability
2. Hypothesis
3. Scale
4. Run & Results
Chaos Engineering

Target System

Design Chaos

Demo

Wrapping up

Chaos Experiment - Tools

1. Observability
2. Hypothesis
3. Scale
4. Run & Results

Define

Evaluate

Execute

Repeat

Graph:
- X-axis: Time (t1, t2, t3, t4, t5, t6)
- Y-axis: Message per Second (Msg/s)
- Legend: Brokers up

Data Points:
- t1: 3, 3
- t2: 2, 2
- t3: 3
- t4: 3
- t5: 3
- t6: 3

Chaos Engineering Tools:
- GREMLIN

Tools Overview:
- Evaluate
- Repeat
- Execute
- Define
Chaos Experiment - Tools

1. Observability
2. Hypothesis
3. Scale
4. Run & Results

kind: PodChaos
metadata:
  name: broker-kill-66
spec:
  action: pod-kill
  mode: one
  selector:
    namespaces: 
      - kafka-main
    labelSelectors: 
      ...

Chaos Engineering
Target System
Design Chaos
Demo
Wrapping up
1. Observability
2. Hypothesis
3. Scale
4. Run & Results

Chaos Experiment - Tools

Chaos Engineering
Target System
Design Chaos
Demo
Wrapping up

Define
Execute
Evaluate
Repeat

kind: PodChaos
metadata:
  name: broker-kill-66
spec:
  action: pod-kill
  mode: one
  selector:
    namespaces:
    - kafka-main
    labelSelectors:
      ...

Brokers up

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg/s</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
1. Observability
2. Hypothesis
3. Scale
4. Run & Results

Chaos Engineering - Target System

Chaos Engineering - Design Chaos

Chaos Engineering - Demo

Chaos Engineering - Wrapping up

Chaos Experiment - Run

112
3
2
3
3
3

Brokers up

Msg/s

3
2
3
2
3
3

The Time Has Come

kind: PodChaos
metadata:
  name: broker-kill-66
spec:
  action: pod-kill
  mode: one
  selector:
    namespaces:
    - kafka-main
    labelSelectors:
      ...
Chaos Experiment - Results

<table>
<thead>
<tr>
<th>type</th>
<th>Traffic_in (msg/s)</th>
<th>replicas_down_to</th>
<th>Duration (m)</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>chaos-66</td>
<td>650 (670 base)</td>
<td>2/3</td>
<td>6</td>
<td>✅</td>
</tr>
</tbody>
</table>

kind: PodChaos
metadata:
  name: broker-kill-66
spec:
  action: pod-kill
  mode: one
  selector:
    namespaces: - kafka-main
    labelSelectors: ...

The Time Has Come
Demo I: Broker(s) failure

**Description:** Fail of critical component Pod(s).

**Observability:** Ensure the availability of metrics for CPU, memory, and traffic in Kafka Pods.
**Demo I: Broker(s) failure**

**Description:** Fail of critical component Pod(s).

**Observability:** Ensure the availability of metrics for CPU, memory, and traffic in Kafka Pods.

**Steady State:** All broker and client replicas are up and ready, with communication throughput stable and free of spikes.
**Demo I: Broker(s) failure**

**Description:** Fail of critical component Pod(s).

**Observability:** Ensure the availability of metrics for CPU, memory, and traffic in Kafka Pods.

**Steady State:** All broker and client replicas are up and ready, with communication throughput stable and free of spikes.

**Hypothesis:** Eliminating three out of seven brokers will not result in cascading failures, and user impact will be minimal. Throughput may significantly decrease but should not drop to zero, and the disruption should not last longer than the time required to respawn lost instances (approximately 1 minute).
**Demo I: Broker(s) failure**

**Description:** Fail of critical component Pod(s).

**Observability:** Ensure the availability of metrics for CPU, memory, and traffic in Kafka Pods.

**Steady State:** All broker and client replicas are up and ready, with communication throughput stable and free of spikes.

**Hypothesis:** Eliminating three out of seven brokers will not result in cascading failures, and user impact will be minimal. Throughput may significantly decrease but should not drop to zero, and the disruption should not last longer than the time required to respawn lost instances (approximately 1 minute).
Description: Fail of critical component Pod(s).

Observability: Ensure the availability of metrics for CPU, memory, and traffic in Kafka Pods.

Steady State: All broker and client replicas are up and ready, with communication throughput stable and free of spikes.

Hypothesis: Eliminating three out of seven brokers will not result in cascading failures, and user impact will be minimal. Throughput may significantly decrease but should not drop to zero, and the disruption should not last longer than the time required to respawn lost instances (approximately 1 minute).

Checks:
- All Kafka Pods Ready
- All produced messages consumed
Demo II: Worker node crash

**Description**: Crash of a worker node effect on services (Kafka and Mirror Maker).

**Observability**: Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.
Demo II: Worker node crash

Description: Crash of a worker node effect on services (Kafkas and Mirror Maker).

Observability: Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.
**Demo II: Worker node crash**

**Description:** Crash of a worker node effect on services (Kafkas and Mirror Maker).

**Observability:** Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.

**Steady State:** All services are fully available and ready to accept traffic.
**Demo II: Worker node crash**

**Description:** Crash of a worker node effect on services (Kafkas and Mirror Maker).

**Observability:** Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.

**Steady State:** All services are fully available and ready to accept traffic.

**Hypothesis:** Eliminating one of the Kubernetes worker pools will not bring down any services, even temporarily. The cluster will recover, and within a reasonable time period, all services will return to their full replica count.
**Demo II: Worker node crash**

**Description:** Crash of a worker node effect on services (Kafka and Mirror Maker).

**Observability:** Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.

**Steady State:** All services are fully available and ready to accept traffic.

**Hypothesis:** Eliminating one of the Kubernetes worker pools will not bring down any services, even temporarily. The cluster will recover, and within a reasonable time period, all services will return to their full replica count.
**Demo II: Worker node crash**

**Description:** Crash of a worker node effect on services (Kafkas and Mirror Maker).

**Observability:** Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.

**Steady State:** All services are fully available and ready to accept traffic.

**Hypothesis:** Eliminating one of the Kubernetes worker pools will not bring down any services, even temporarily. The cluster will recover, and within a reasonable time period, all services will return to their full replica count.

**Checks:**

- Verify that all Kafka clusters and accompanying services are ready.
- Ensure all messages produced are successfully consumed from the relevant clusters.
**Demo II: Worker node crash**

**Description:** Crash of a worker node effect on services (Kafka and Mirror Maker).

**Observability:** Ensure the availability of all services across the cluster; monitor for any unexpected events within the cluster.

**Steady State:** All services are fully available and ready to accept traffic.

**Hypothesis:** Eliminating one of the Kubernetes worker pools will not bring down any services, even temporarily. The cluster will recover, and within a reasonable time period, all services will return to their full replica count.

**Checks:**

- Verify that all Kafka clusters and accompanying services are ready.
- Ensure all messages produced are successfully consumed from the relevant clusters.
Embracing Chaos - Benefits

- Confidence in System & Wrinkles prevention
- Misconfigurations
- Experience & new knowledge

- Chaos Engineering
- Target System
- Design Chaos
- Demo
- Wrapping up
Embracing Chaos - Benefits

Confidence in System & Wrinkles prevention

Misconfigurations

Experience & new knowledge
Embracing Chaos - How to?
Embracing Chaos - How to?
Embracing Chaos - How to?

- Know your tools
- Game Days
Embracing Chaos - How to?

Game Days

Know your tools

Chaos Engineering
Target System
Design Chaos
Demo
Wrapping up

Know your tools
Embracing Chaos - How to?

- Game Days
- Start small
- Know your tools
Thank you!