Deploy and use **Apache Kafka** on **Kubernetes**

Jakub Scholz

Bern, 20.6.2019

About me

Jakub Scholz

- Principal Software Engineer with the Red Hat Messaging and IoT engineering team
- Long-term messaging specialist
- Former messaging team lead at Deutsche Börse

@scholzj

https://github.com/scholzj

https://www.linkedin.com/in/scholzj/

Red Hat

Strimzi

- Open source project licensed under Apache License 2.0
- Focuses on running Apache Kafka on Kubernetes and OpenShift:
  - Container images for Apache Kafka and Apache Zookeeper
  - Operators for managing and configuring Kafka clusters, topics or users
- Provides Kubernetes-native experience for running Kafka on Kubernetes and OpenShift
  - Doesn't manage only Kafka brokers, but also users or topics

Web site: http://strimzi.io/

GitHub: https://github.com/strimzi

Twitter: @strimziio

Strimzi

- Just released: 0.12.0! What's new?
  - Support for Kubernetes 1.9 and 1.10 / OpenShift 3.9 and 3.10 dropped
  - Reduced number of container images
  - Custom Resources improvements
    - Upgraded from v1alpha1 to v1beta1
    - Status subresource for the Kafka resource
    - Improved handling of invalid fields in the Custom Resources
  - Improvements to persistent storage handling
    - Adding / removing JBOD volumes
    - Volume resizing
  - HTTP Bridge

Demo:
Kafka cluster status

What do you use **Kafka** for?

Using Kafka

- Many different use-cases can be addressed by Kafka
  - Microservice integration / messaging
  - Event sourcing
  - Metrics and log collection
  - Stream processing
  - Website activity tracking
  - Storage / Key-value database / Commit log

Events

Key	Jakub	Jakub	Joe	Jakub	Jakub	Joe
Value	Olomouc	Prague	New York	Frankfurt	Prague	Bern

Stream-Table duality

Key	Jakub	Jakub	Joe	Jakub	Jakub	Joe
Value	Olomouc	Prague	New York	Frankfurt	Prague	Bern

Key	Value
Jakub	Prague
Joe	Bern

Compacted topics

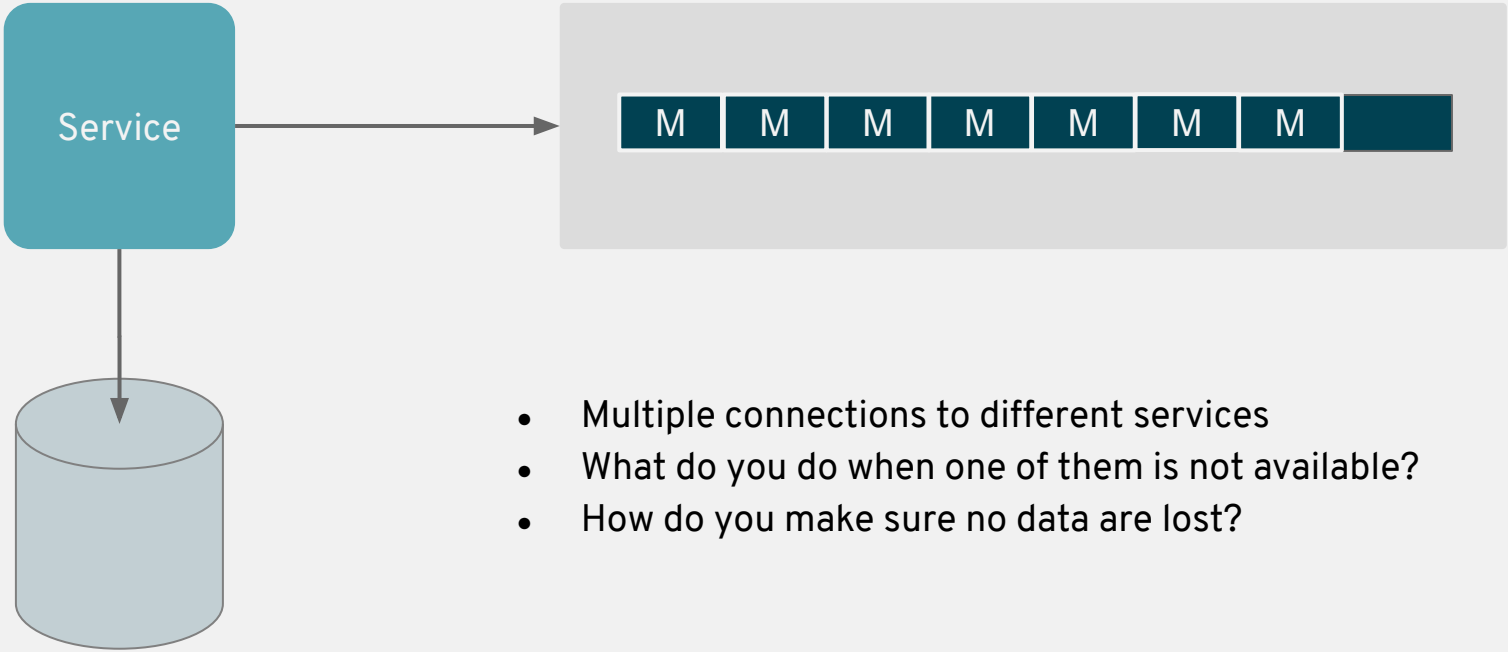
Key	Jack	Jakub	Joe	Jakub	Joe	Joe
Value	London	Prague	Bern	Prague	Boston	Bern

Demo:
Address book

But what if …

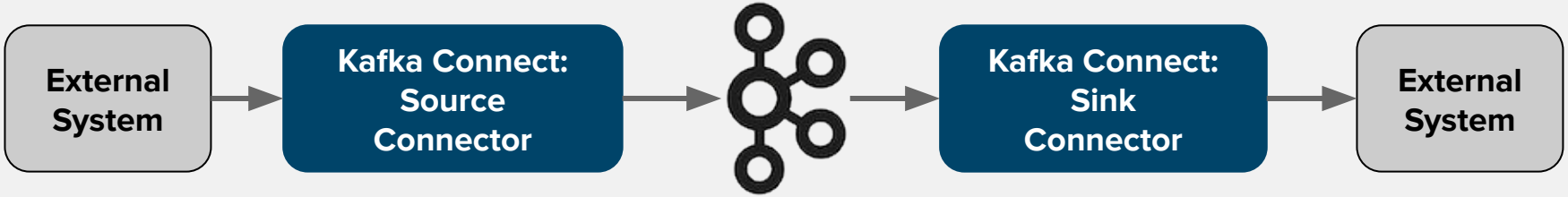
… my app already uses a **database**?

Using database with Kafka



- Multiple connections to different services
- What do you do when one of them is not available?
- How do you make sure no data are lost?

Connecting Kafka



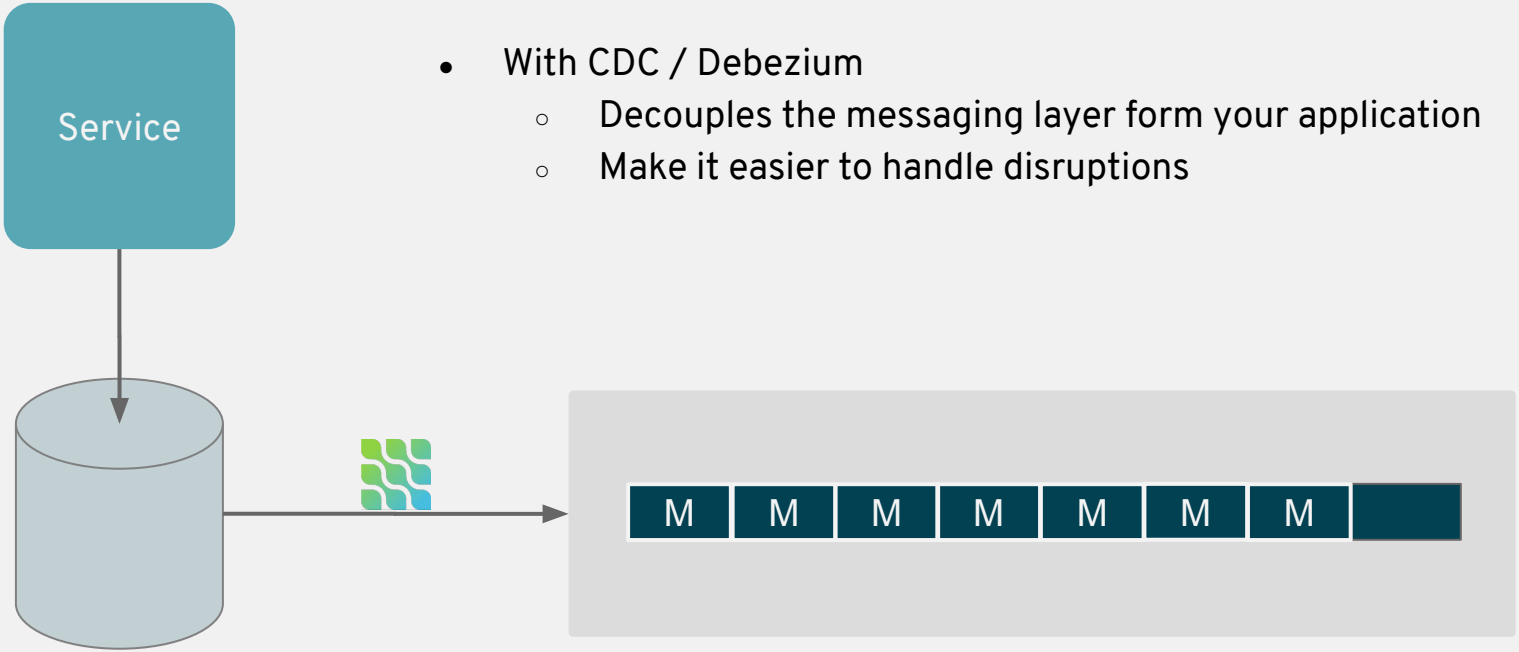
Connecting Kafka

- Framework for integrating Kafka with other systems
  - Source connectors for getting data into Kafka
  - Sink connectors to ship data from Kafka somewhere else
- Many 3rd party connectors available
  - Messaging systems (JMS, Amazon SNS/SQS, WebSphere MQ, ...)
  - Databases / Change Data Capture (JDBC, Debezium, ...)
  - Data Storage (Amazon S3, HDFS, Casandra, ...)
  - and more
- You can use Connect API to write your own connectors
  - Implement few simple methods to create your own connectors

CDC

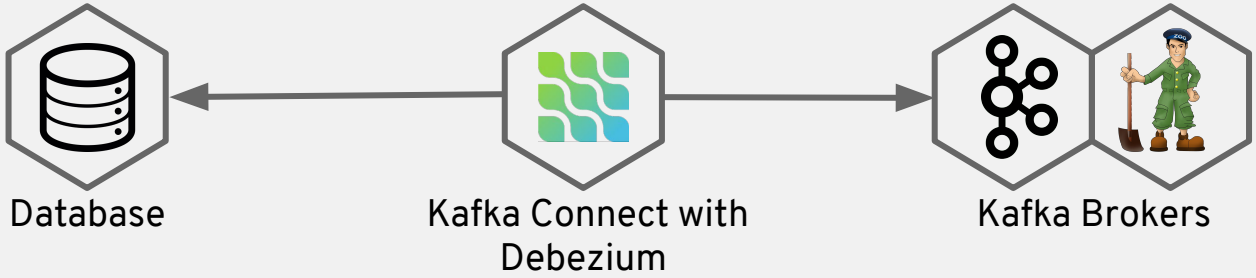
- Change Data Capture (CDC)
- Captures the data directly from database
  - Doesn’t require your application to send / receive data to Kafka
  - Instead a CDC application captures them from database
- Debezium is one of the implementations of this pattern
  - Debezium connects to selected database, reads its transaction log and publishes it as Kafka messages
  - Supported databases are MySQL, PostgreSQL, MongoDB and SQL Server
    - Additional plugins might be needed to access the DB and its logs
  - The Kafka messages can be send for example in JSON format
  - https://debezium.io/

Debezium



- With CDC / Debezium
  - Decouples the messaging layer form your application
  - Make it easier to handle disruptions

Debezium



Debezium

- What makes CDC / Debezium so great?
  - Makes it easy to integrate DB based applications into Kafka
    - No need to change your application
    - No need to write data to DB and send to Kafka
    - Easier to handle errors and (un)availability of DB or Kafka
  - Use cases (for example)
    - Microservice integration
    - Data replication

Demo:
Address book with DB

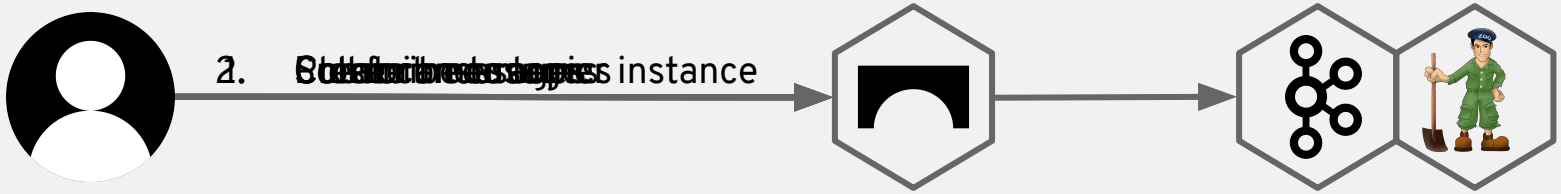
HTTP Bridge

- HTTP Bridge is new component in 0.12.0
- Still a bit *work in progress*
- You can download it and use it on your own or deploy it using Strimzi operators
- Can be used to access Kafka using HTTP REST API
  - Tries to maintain compatibility with the Confluent REST Proxy
  - But implements only selected features
  - Consumers are stateful, producers are stateless
  - Supports JSON and Binary payloads (=> Anything encoded in Base64)
- AMQP Bridge (part of the same upstream code) is not exposed by operators!

HTTP Bridge

- Use cases
  - Applications which do not speak Kafka (e.g. because of language etc.)
  - Web applications
  - Access from application outside of of Kubernetes cluster / public internet
  - IoT use cases (collecting telemetry etc.)
- Security and exposing
  - Easier than with Kafka because it is just regular HTTP REST API
  - We have chosen different approach then with Kafka
  - Users can configure it on their own using things like Ingress, API Gateways, OAuth Proxies etc.

HTTP Bridge



Red Hat

Demo:
HTTP Bridge

Red Hat

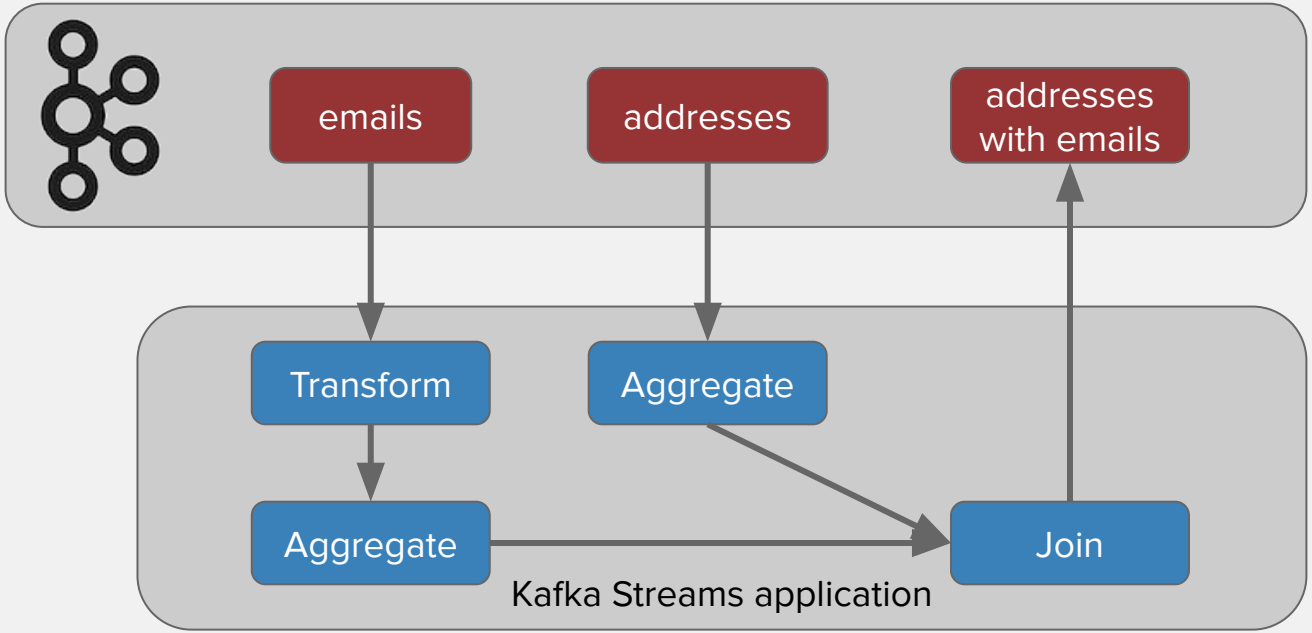
Now we have two topics … one with the emails and one with cities!?

Let’s use some **stream processing** to fix it!

Kafka Streams

- Client library for building streaming applications
  - Just a library, not a framework
  - Perfect integration with Apache Kafka
- Provides a DSL for
  - Data transformation
  - Data aggregation
  - Joining of streams
  - Windowing

Kafka Streams



Demo:
Kafka Streams

Kafka Streams

- Running Kafka Streams on Kubernetes
  - Stateless operations (such as transformations) are easy to run as a Deployment
  - Stateful applications rely on local cache for performance
    - Can run as Deployments
    - Often run as StatefulSets with persistent volumes
    - When the pod crashes or restarts, it can quickly recover big part of the data from the local cache => no need to re-read them from Kafka

Slides and demo source codes:
http://jsch.cz/bernmeetup2019

Red Hat

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos