APACHE KAFKA ALS IOT-DATEN-PLATTFORM

Javaforum Stuttgart 05.07.2018

Dr. Ralph Guderlei

eXXcellent Solutions GmbH

Agenda

Wie alles begann

Grundlagen Apache Kafka

Kafka Connect

Kafka Streams

Lessons learned

Technology Advisor & Project Manager
eXXcellent Solutions GmbH, Ulm
ralph.guderlei@exxcellent.de
@rguderlei

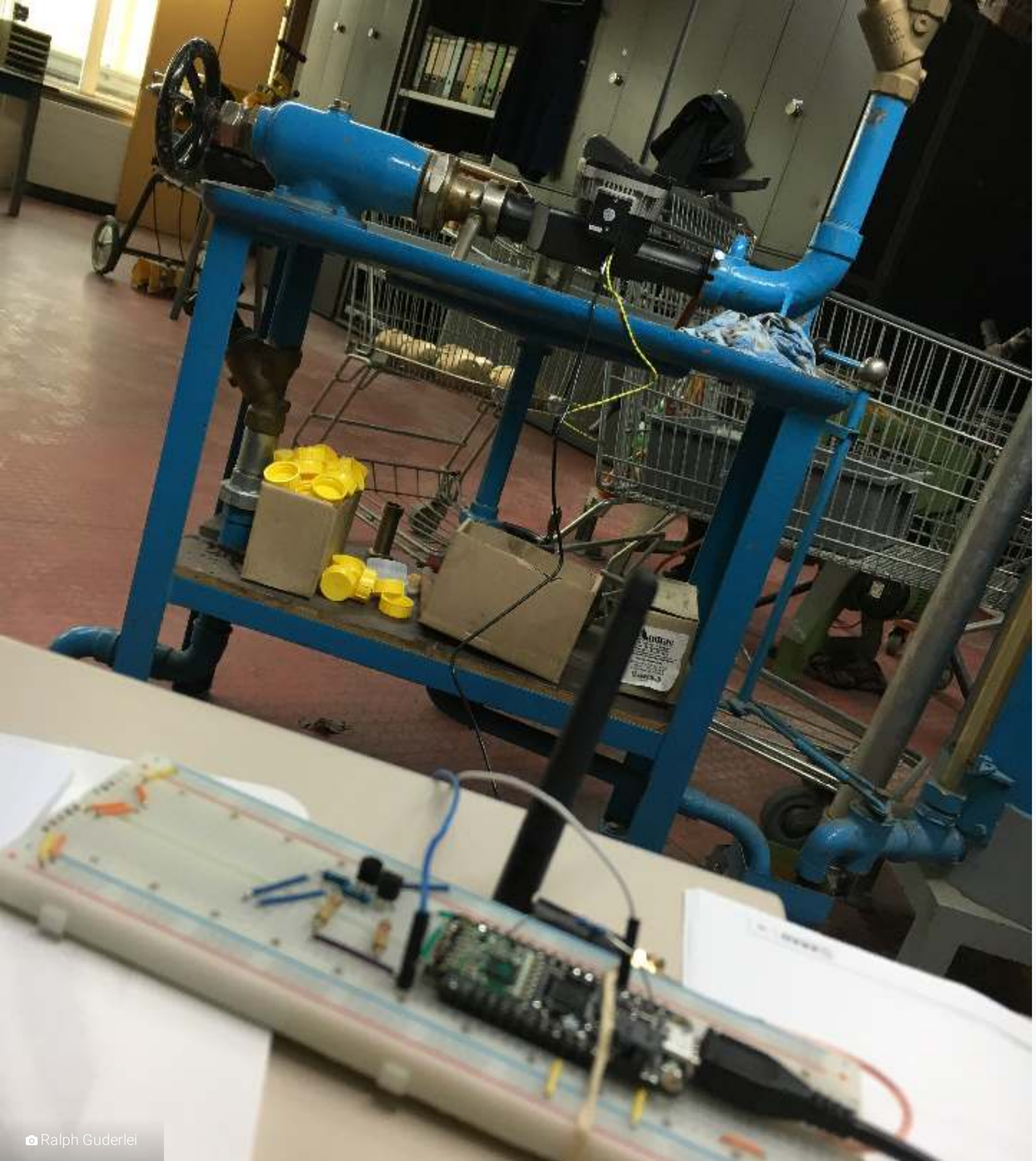

Kelink
communication is everything

Ralph Guderlei

Digitale Stromzähler

Optimierung Abrechnung

Verbesserte Bedarfsprognose



Wasserzähler

Leckage-Erkennung

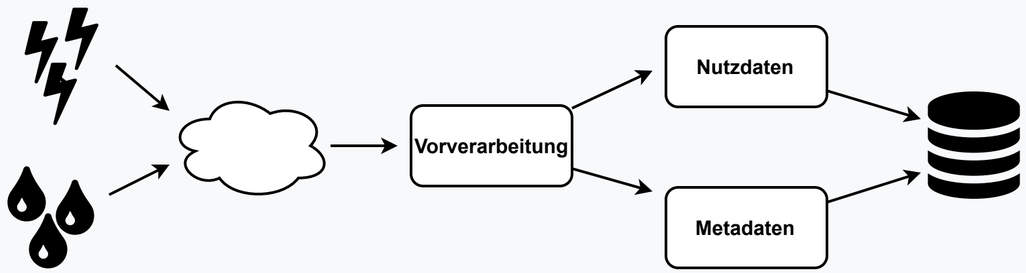
Infrastuktur-Monitoring

Plattform für Smart City

> 150.000 Stromzähler	> 100 Wasserzähler	> 5.000 Parkplätze	> 30.000 Straßenlaternen

Uptime	Latenz	Durchsatz
24/7	< 100ms	> 2500 Msg/s

grobe Systemarchitektur



Technologie

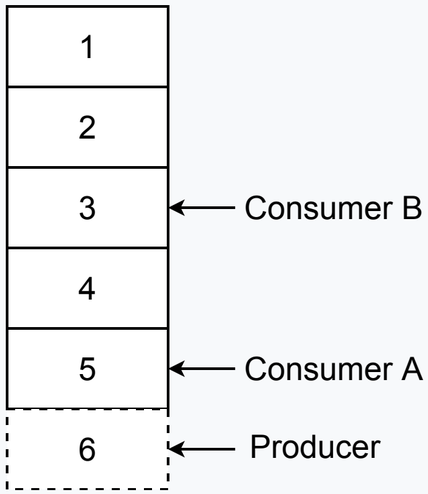
APACHE KAFKA

Pub/Sub, Stream Processing

APACHE CASSANDRA

Verteilte Datenbank

Apache Kafka



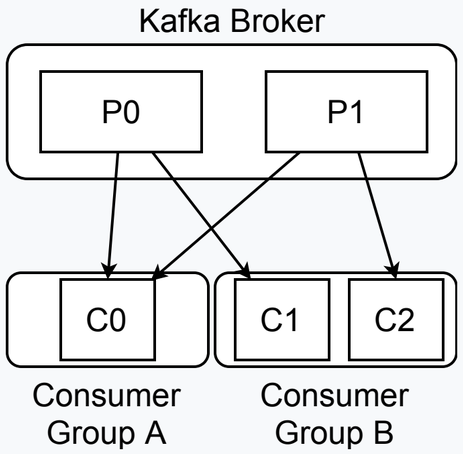
Topics

Geordnete, unveränderliche Sequenz von Nachrichten

Inhalt der Nachrichten beliebig, meist JSON

Persistent

Publish/Subscribe



Partitionen

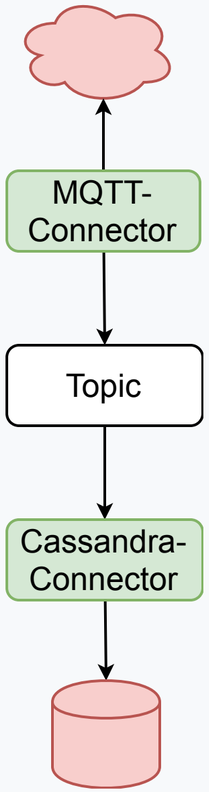
erlauben Verteilung von Daten auf Cluster

Anzahl der Partitionen bestimmt Parallelisierungsgrad von Consumern

Reihenfolge der Nachrichten bleibt erhalten

pro Producer und Partition

Kafka Connect



Kafka Connect

- Verbindet Kafka mit anderen Systemen
- Pull von Daten nach Kafka
- Push von Kafka in Drittsysteme
- Große Anzahl bestehender Konnektoren

```java
public class MySinkTask extends SinkTask {
  @Override
  public void start(Map<String, String> props) {
      // ...
  }

  @Override
  public void put(Collection<SinkRecord> records) {
    records.forEach(record -> {
      Struct payload = (Struct) record.value();
      Schema schema = record.valueSchema();
      // ... move data out of kafka
    };
  }

  @Override
  public void stop() {
    // ...
  }
}
```

Kafka Connect

- Konnektoren bestehen aus: Rahmen, Konfiguration und Worker-Tasks
- eigenes Metamodell für Datenstrukturen
- eine Connect-Instanz verwaltet mehrere Konnektoren
- Konnektoren statisch (standalone) oder dynamisch (distributed) konfigurierbar

Kafka Streams

```java
public Topology topology(Properties config) {
  final StreamsBuilder builder = new StreamsBuilder();
  final String IN = config.getProperty("topics.in");
  final String OUT = config.getProperty("topics.out");

  // capture and transform each incoming message
  final KStream<String, DataPoint> values = builder
          .stream(IN, Consumed.with(Serdes.String(),
            new ConnectJsonSerde<>(TTNMessage.class)))
          .flatMapValues(ElectricityMessageParser::parse)

  // send data to topic
  values.to(OUT, Produced.with(Serdes.String(),
    new ConnectJsonSerde<>(DataPoint.class)));
  return builder.build();
}
```

Kafka Streams

- High-level API für Consumer
- Programmiermodel analog zu Streams API
- mehrere Quell- und Ziel-Topics möglich
- einfache Bibliothek
- Deployment als Fat Jar

```
Topology topology =  ...

Properties config = new Properties();
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "test");
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "dummy:1234");
testDriver = new TopologyTestDriver(topology, config);
ConsumerRecordFactory<String, String> recordFactory
  = new ConsumerRecordFactory<>(new StringSerializer(), new StringSerializer());

String payload = ...
testDriver.pipeInput(
  recordFactory.create("ttn", "foo", payload)
);

OutputVerifier.compareKeyValue(
  testDriver.readOutput("ttn-metadata", stringDeserializer,
      new JsonDeserializer<>(TtnConnectionInfo.class)),
    "003E94C6AD72F129",
    new TtnConnectionInfo("exxcellent-uno",
        "2017-04-28T11:30:40.62428417Z",
        "eui-0000024b0b03020e"));
```

Kafka Streams - Testing

- Unit-Tests für Streams
- startet Kafka In-Memory
- erst seit Kafka 1.1.0

```java
final KStream<String, TtnDeviceInfo> uplinkMessages = ...
  .selectKey((k, v) -> v != null ? v.id : "no key")
  .filterNot((k, v) -> Objects.equals(k, "no key"));

final KTable<String, TtnDeviceInfo> storedValues = builder.table(
  TABLE,
  Consumed.with(Serdes.String(), new ConnectJsonSerde<>(TtnDeviceInfo.class))
);

final KStream<String, TtnDeviceInfo> updated = uplinkMessage
  .leftJoin(
    storedValues,
    TtnDeviceInfo::updatePacketsDropped,
    Joined.with(
      Serdes.String(),
      new ConnectJsonSerde<>(TtnDeviceInfo.class),
      new ConnectJsonSerde<>(TtnDeviceInfo.class))
  );
updated.to(TABLE,
  Produced.with(Serdes.String(),
    new ConnectJsonSerde<>(TtnDeviceInfo.class)));
```

Kafka Streams - Tables

- Spezielle Topics
- Join von Topics
- lokale oder globale Tabellen möglich
- ermöglichen zustandsbehaftete Berechnungen

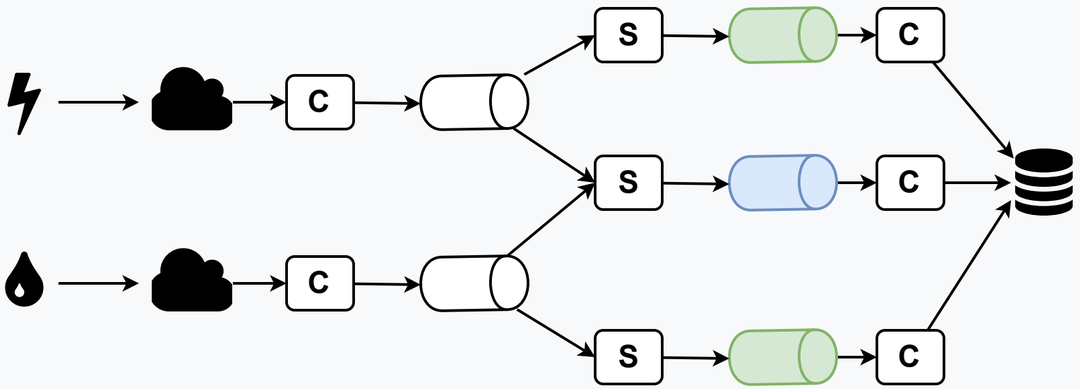
```
// Load pre-trained model
MultiLayerNetwork model = ModelSerializer.restoreMultiLayerNetwork(modelFile);

// capture and transform each incoming message
final KStream<String, Result> values = builder
        .stream(IN, Consumed.with(Serdes.String(),
          new ConnectJsonSerde<>(TTNMessage.class)))
        .mapValues(Extractor::extractFeatures)
        .mapValues(model::output)
        .mapValues(Result::fromPrediction)

// send data to topic
values.to(OUT, Produced.with(Serdes.String(),
  new ConnectJsonSerde<>(Result.class)));
return builder.build();
```

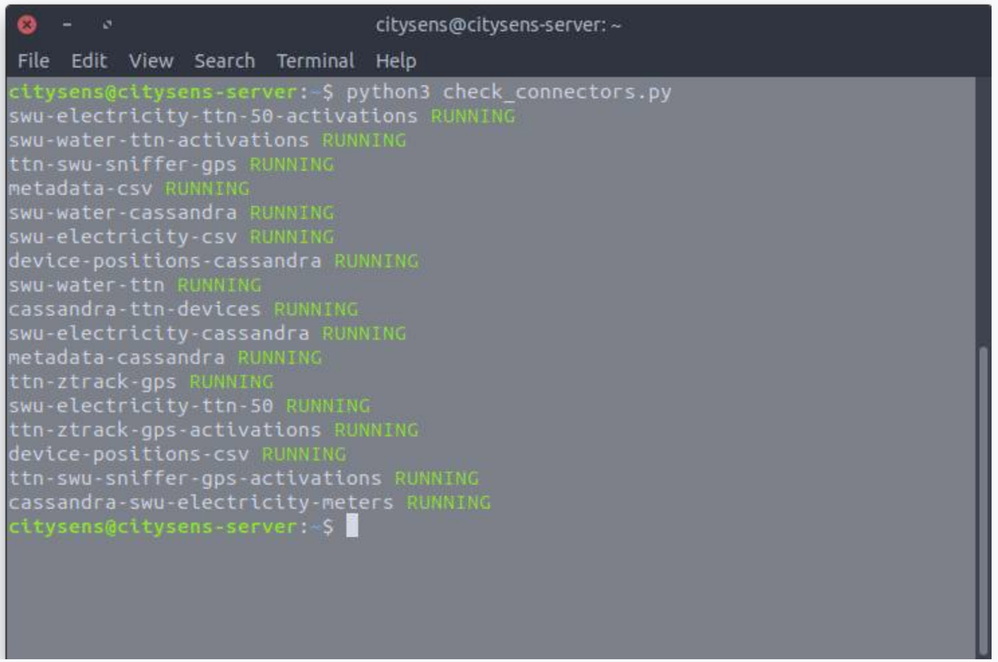
Kafka Streams - ML

- In Streams können beliebige Bibliotheken eingebunden werden
- Deeplearning4J: Import von trainierten Modellen
- Verwendung des geladenen Modells im Stream



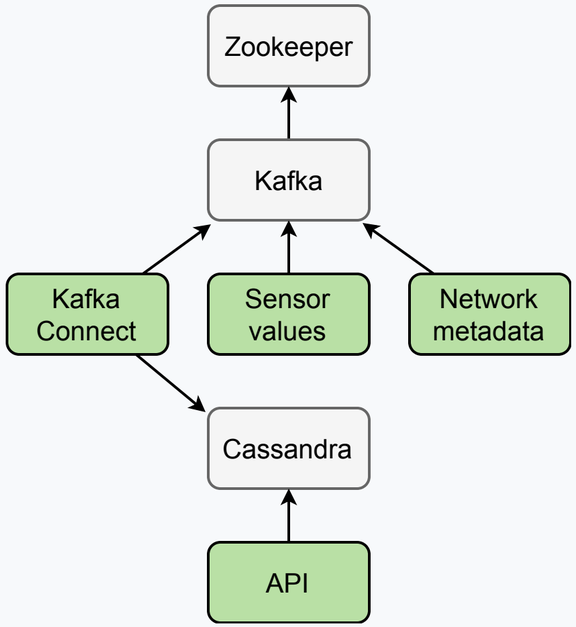
Lessons Learned - Kafka

- immer aktuelle Versionen verwenden
- Kafka < 1.0.0 hat einen Bug beim Deregistrieren von Streaming Apps
- Kafka: wichtigste Metrik Consumer lag
- →Burrow: https://github.com/linkedin/Burrow



Lessons Learned - Kafka Connect

- generischer Ansatz, damit unnötig komplex
- nur rudimentäre Dokumentation
- fertige Konnektoren haben schwankende Qualität
- eigene Tools für Umgang mit Kafka Connect:
  - Skripte zur Inspektion
  - Serialisierung und Deserialisierung von Daten
  - Umgang mit Connect-Datenstrukturen

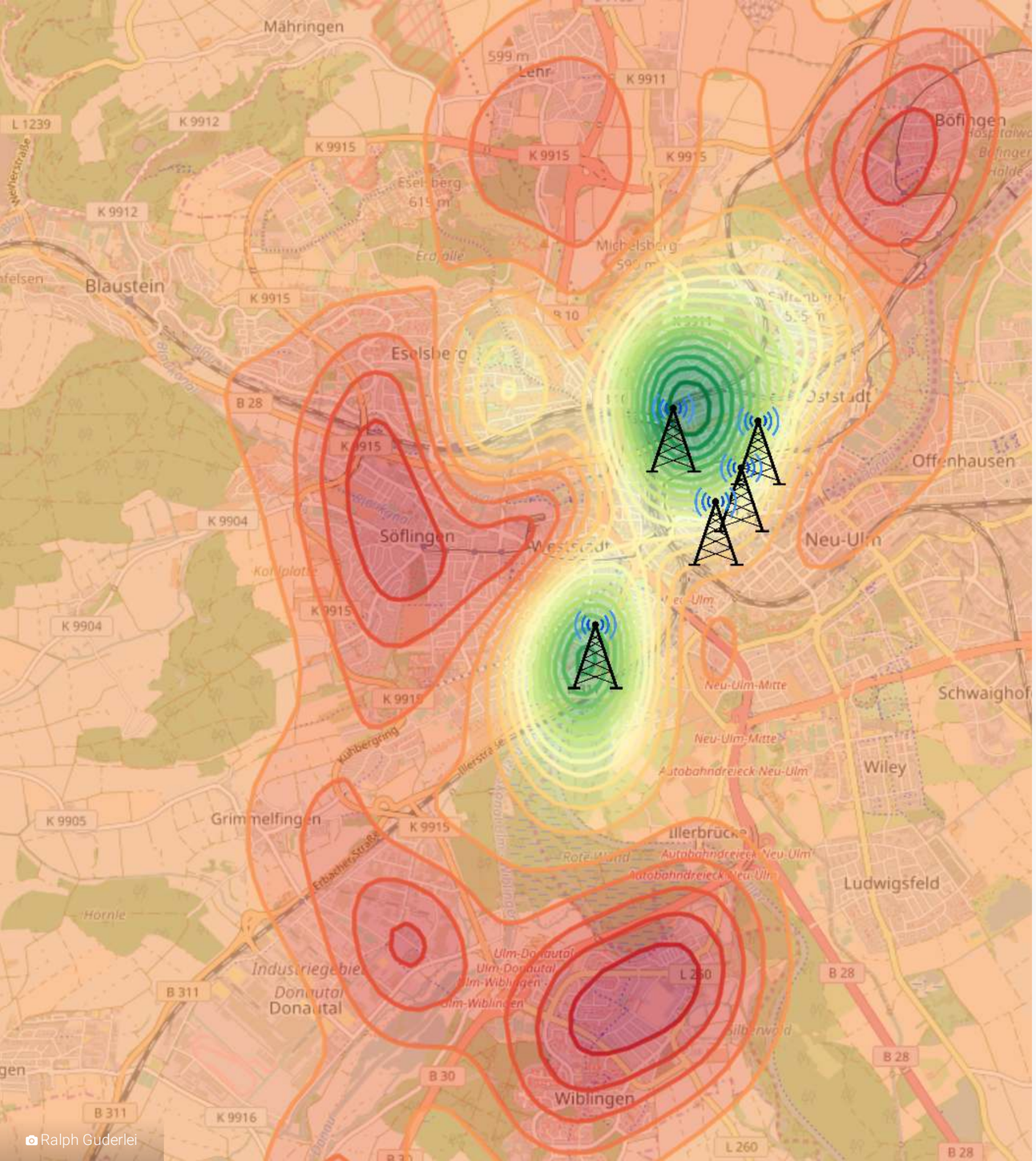


Lessons Learned - Betrieb

- komplexes Gesamtsystem aus vielen Teilen
- Kafka Streams sind Microservices
- Infrastruktur-Automatisierung notwendig (z.B. Docker Compose)
- Monitoring notwendig (z.B. Prometheus/Grafana)

Lessons Learned - Monitoring

- sehr viele Metriken via JMX verfügbar
- Kafka Connect: Konnektoren schwierig zu überwachen
- Konnektoren und Streaming Apps sollten sich bei Fehlern sofort beenden



Ausblick: Analytics

Netzabdeckung

Interpoliertion mittels Kriging/Gauß-Prozess-Regression

Anomalieerkennung

Zeitreihenanylyse von Sensorwerten und Metadaten

Fazit

- Kafka: Komplex, aber gut verwendbar
- Kafka Connect: Gute Idee, moderat gute Umsetzung
- Kafka Streams: sehr einfache Umsetzung von Stream Processing-Aufgaben


The
END