

Conversations with Kafka

Getting to know Kafka better from the Clojure REPL

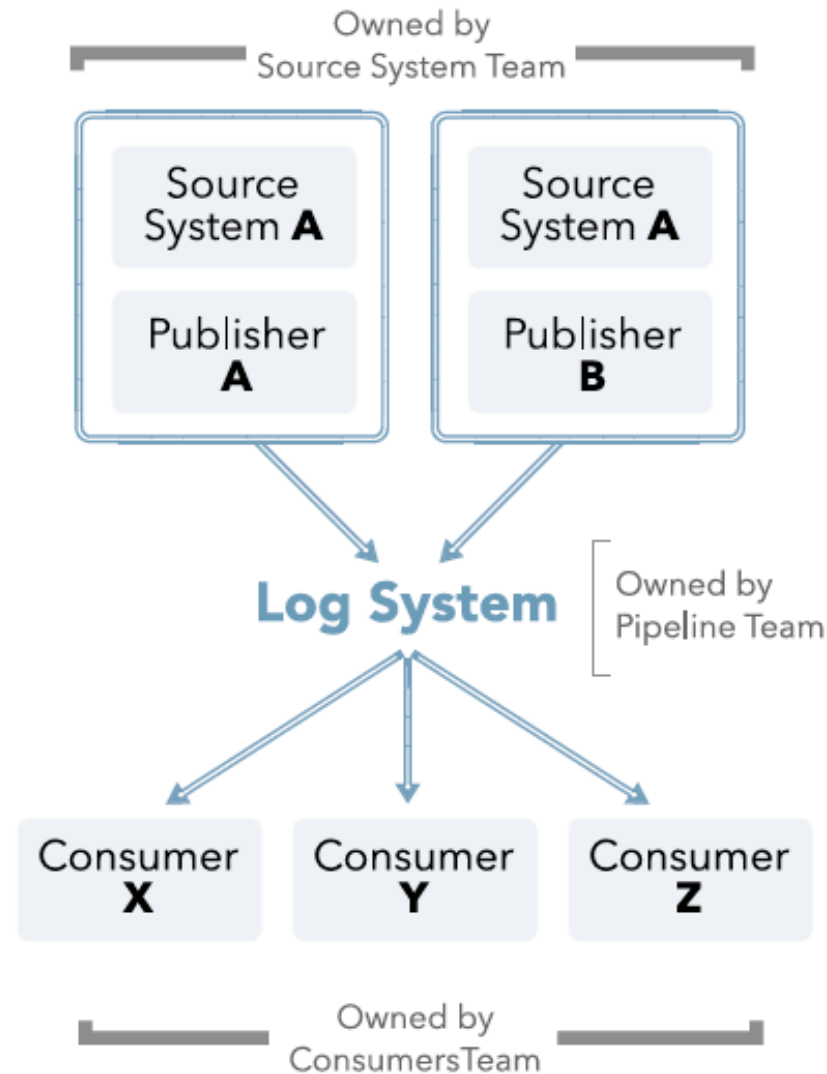
Nacho Muñoz - 25/01/2018



@pimunozg

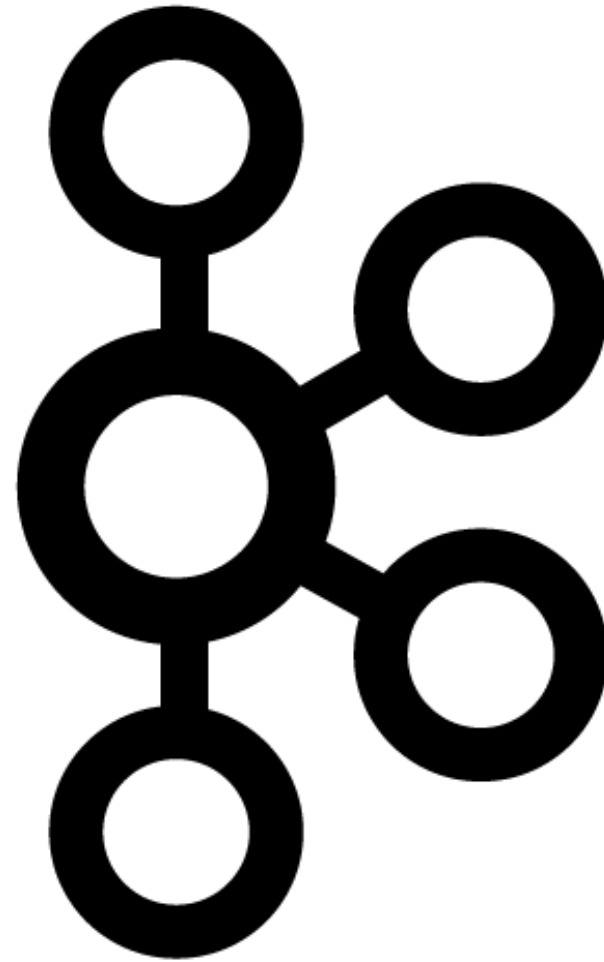
About Kafka

- Distributed commit log
- Pub/Sub messaging system
- Typically used for
 - Activity Tracking
 - Messaging
 - Stream Processing
 - Commit log
 - Metrics and logging



Kafka Protocol

- Binary Protocol
 - Broker - Broker
 - Client - Broker
- All request sent to a broker from a specific client will be processed in the order they were sent.



Request types

- 31 different request types
- Protocol evolution based on:
 - Adding new request types
 - Update existing ones with new information

```
PRODUCE(0, "Produce"),
FETCH(1, "Fetch"),
LIST_OFFSETS(2, "Offsets"),
METADATA(3, "Metadata"),
LEADER_AND_ISR(4, "LeaderAndIsr"),
STOP_REPLICA(5, "StopReplica"),
UPDATE_METADATA_KEY(6, "UpdateMetadata"),
CONTROLLED_SHUTDOWN_KEY(7, "ControlledShutdown"),
OFFSET_COMMIT(8, "OffsetCommit"),
OFFSET_FETCH(9, "OffsetFetch"),
GROUP_COORDINATOR(10, "GroupCoordinator"),
JOIN_GROUP(11, "JoinGroup"),
HEARTBEAT(12, "Heartbeat"),
LEAVE_GROUP(13, "LeaveGroup"),
SYNC_GROUP(14, "SyncGroup"),
DESCRIBE_GROUPS(15, "DescribeGroups"),
LIST_GROUPS(16, "ListGroups"),
SASL_HANDSHAKE(17, "SaslHandshake"),
API_VERSIONS(18, "ApiVersions"),
CREATE_TOPICS(19, "CreateTopics"),
DELETE_TOPICS(20, "DeleteTopics");
```

Request header

API KEY (int16)

API VERSION (int16)

CORRELATION ID (int32)

CLIENT ID (string)

REQUEST MESSAGE

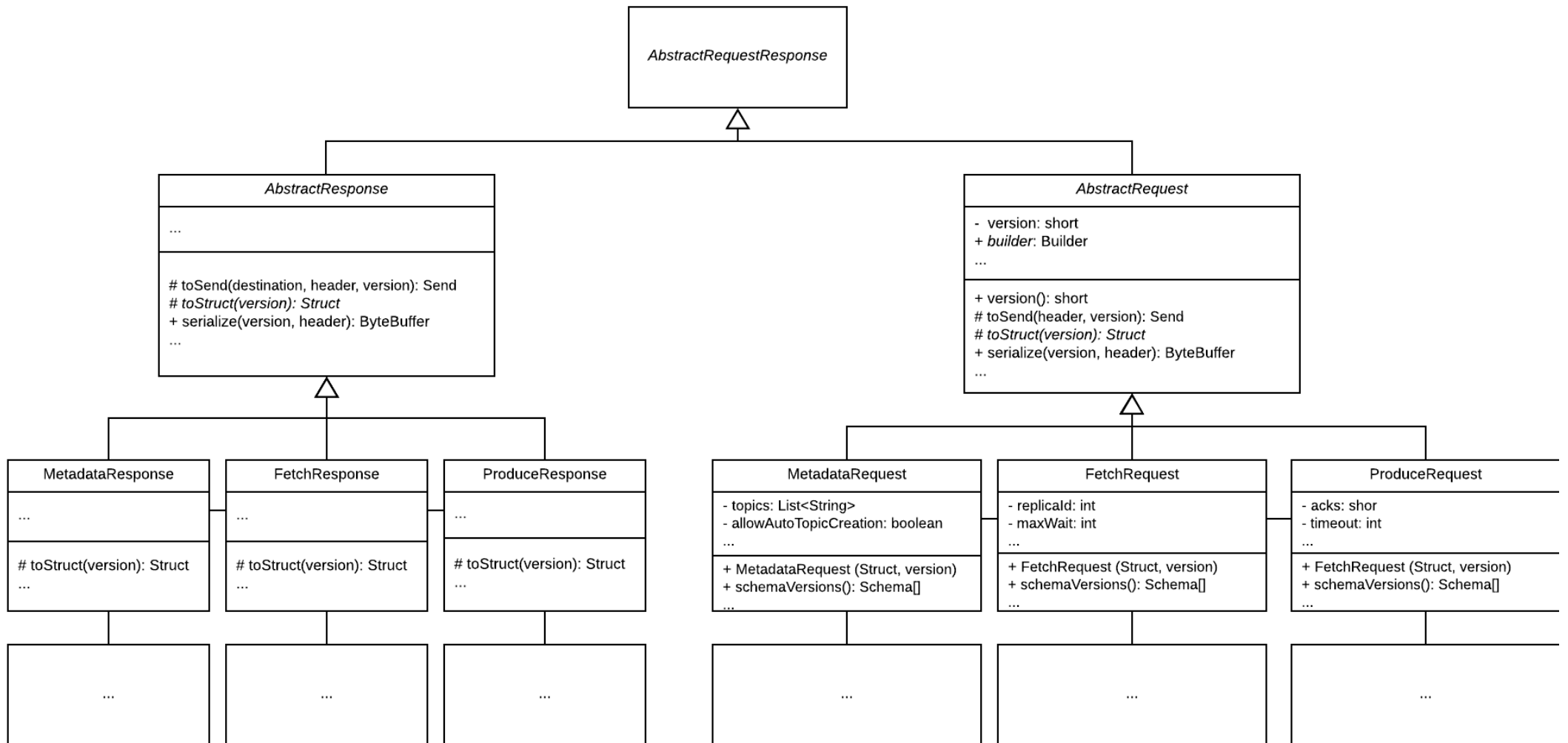
Example: Metadata Request

- Allows the client to get information about:
 - Topics existence
 - Number of partitions for a given topic
 - Associative list of partitions and leaders
 - Current cluster controller
- Any broker can handle it

Kafka protocol implementation

- *AbstractResponse* / *AbstractRequest*
- `toStruct` (protected) / common constructor (*Struct* + version)
- *Schema* to describe fields and types contained in *Struct* instances
- Each request and response types keep track of different versions in a public static field **schemaVersions** facilitating protocol evolution and bidirectional wire compatibility

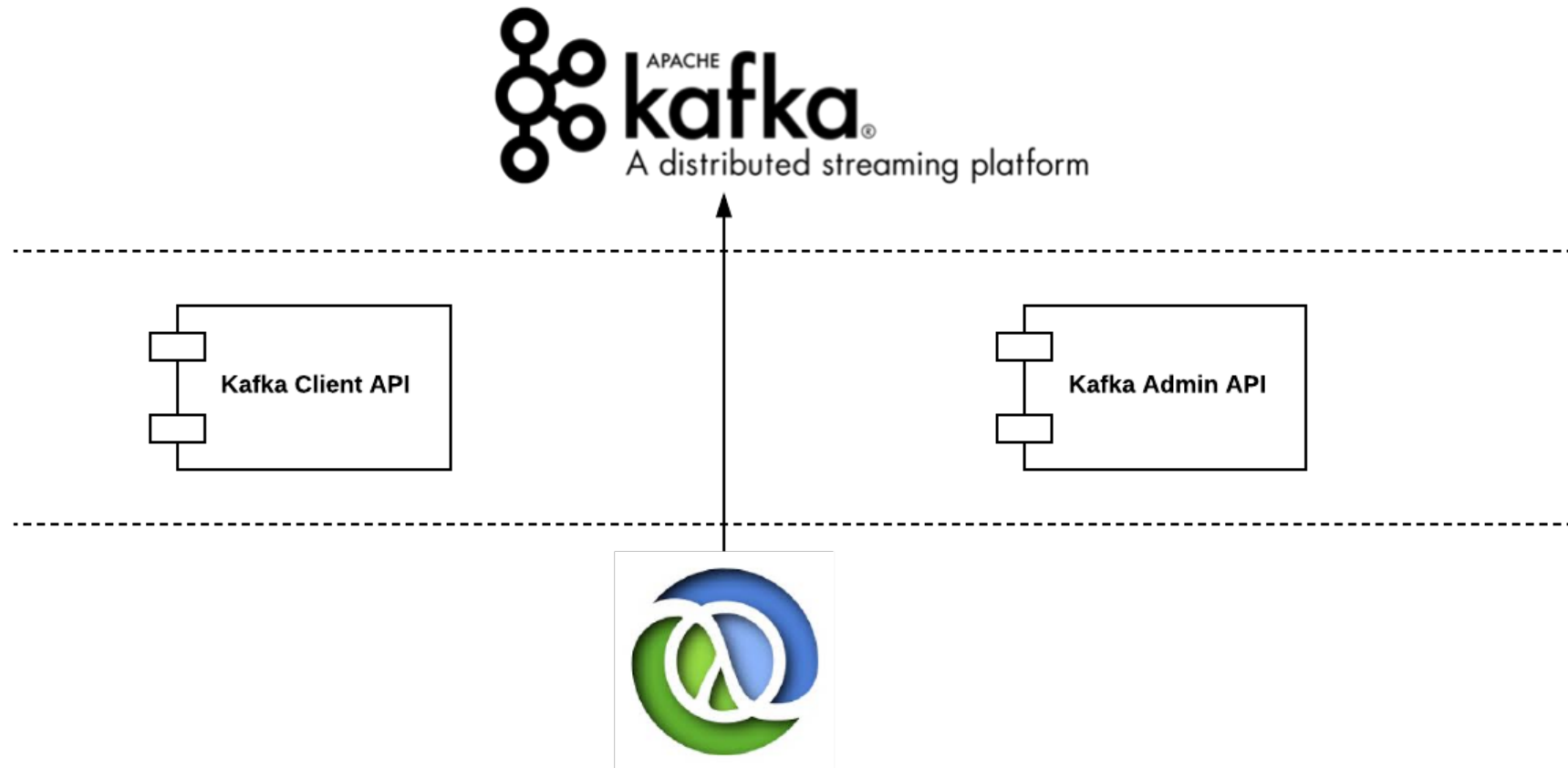
Kafka protocol implementation



Introducing Scoop

- A utility to send low-level request to Kafka
- Responses are translated to Clojure data structures
- Request are expressed as Clojure maps
- Kafka protocol fuzzer when combined with generative testing

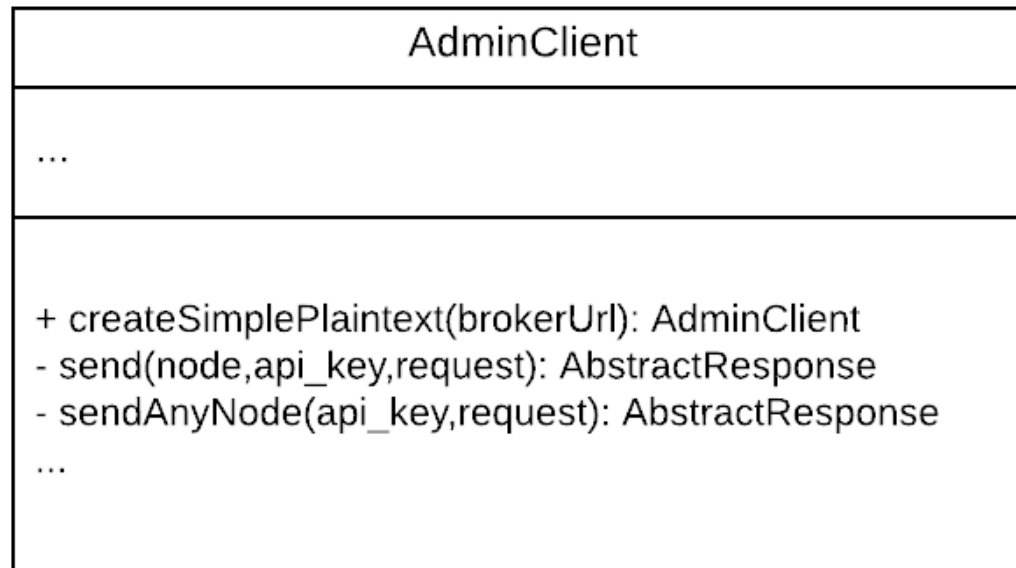
Introducing Scoop



Scoop implementation

1. Protocol primitive level communication with Kafka
2. AbstractResponse translation into Clojure map
3. Create request types specification using clojure.spec
4. Clojure map translation into AbstractRequest
5. Leverage clojure.test.check generators

1. Protocol level communication with Kafka



- Reuse private methods from *AdminClient* to send requests to Kafka
- Expose those methods using Java reflection

2. AbstractResponse translation into Clojure map

- Two steps
 1. **response->struct**
 2. **struct->map**
- We use Java reflection again to expose a protected method: *toStruct(version)*
- Once we have an instance of *Struct* we just follow its *Schema* to figure out fields and type of data while extracting the values from the *Struct* itself




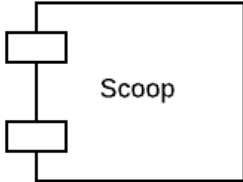
3. Request types specification using clojure.spec

- Send requests to Kafka using plain Clojure maps
- Validate the request before sending it
- Generate valid requests based on the specification
- Take into account request types versioning: one spec per request type and version (e.g. `::metadata-request-v1`, `::metadata-request-v2`, ...)

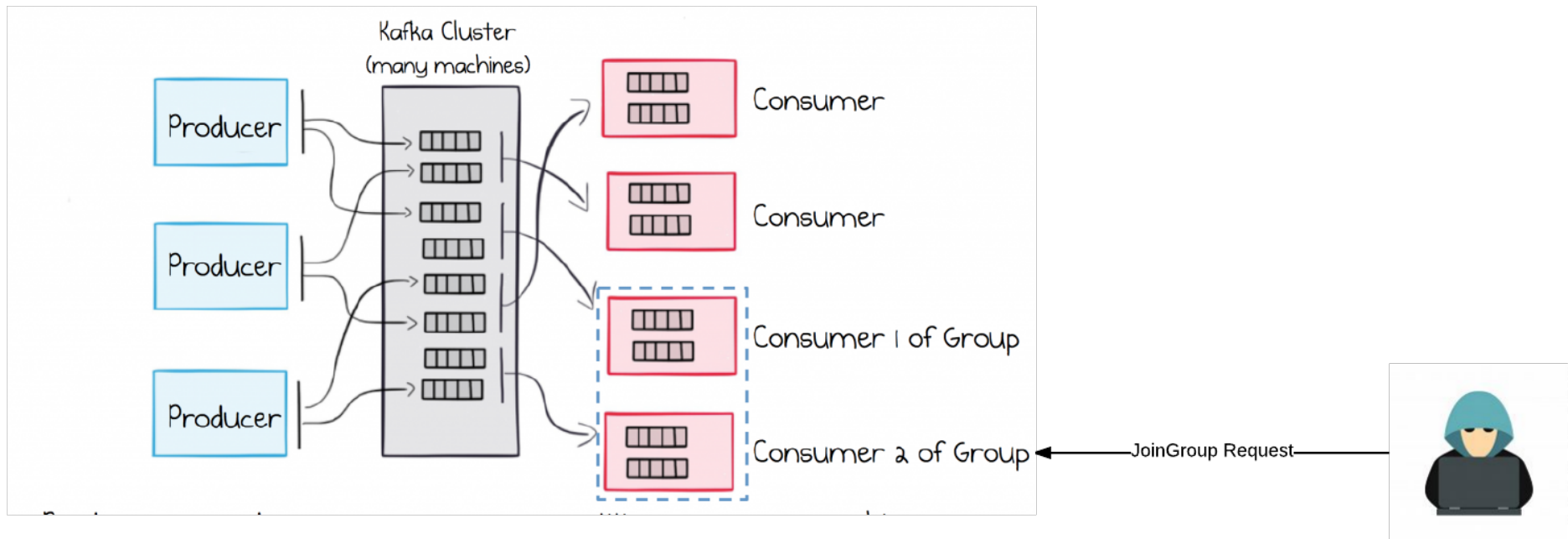
4. Clojure map translation into AbstractRequest

- All the requests types shared a common constructor that receives a *Struct* and a version
- Two steps:
 1. **map->struct** using an equivalent *Schema* than the one expected by the request type
 2. **struct->request** using Java reflection to dynamically invoke the request-type's constructor

Demo

Services overview	Docker	 docker	 APACHE kafka [®] A distributed streaming platform
	Local		 Scoop

Demo



Improvements

- Crash awareness
- Visualise offended request flow in a sequence diagram
- Less reliance on Java reflection

Resources

- Scoop code - <https://github.com/nachomdo/scoop>
- Proto-REPL - <https://github.com/jasongilman/proto-repl>
- Kafka code - <https://github.com/apache/kafka>
- Clojure.Spec - <https://clojure.org/guides/spec>

Thanks!