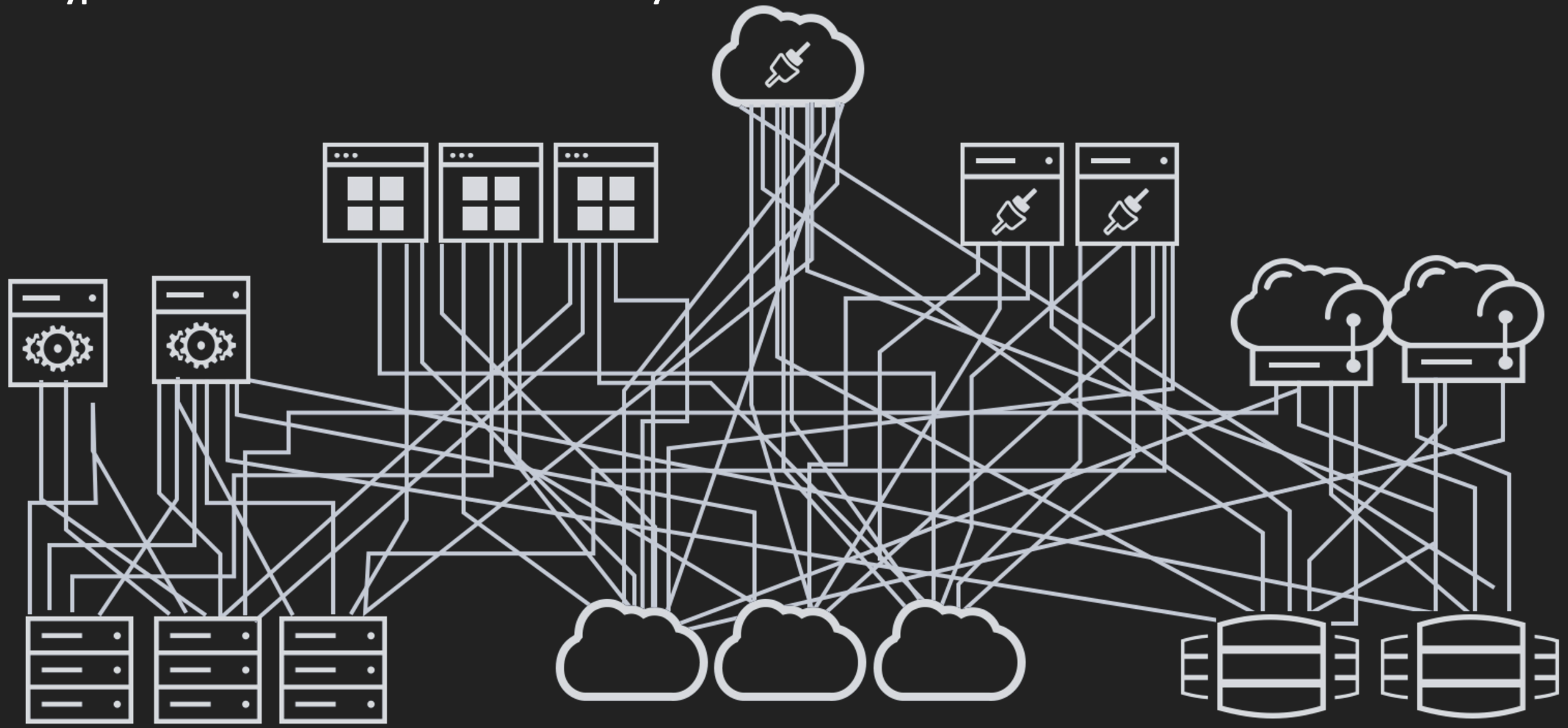
Make communication easier by using a schema registry

Overview Apicurio Registry

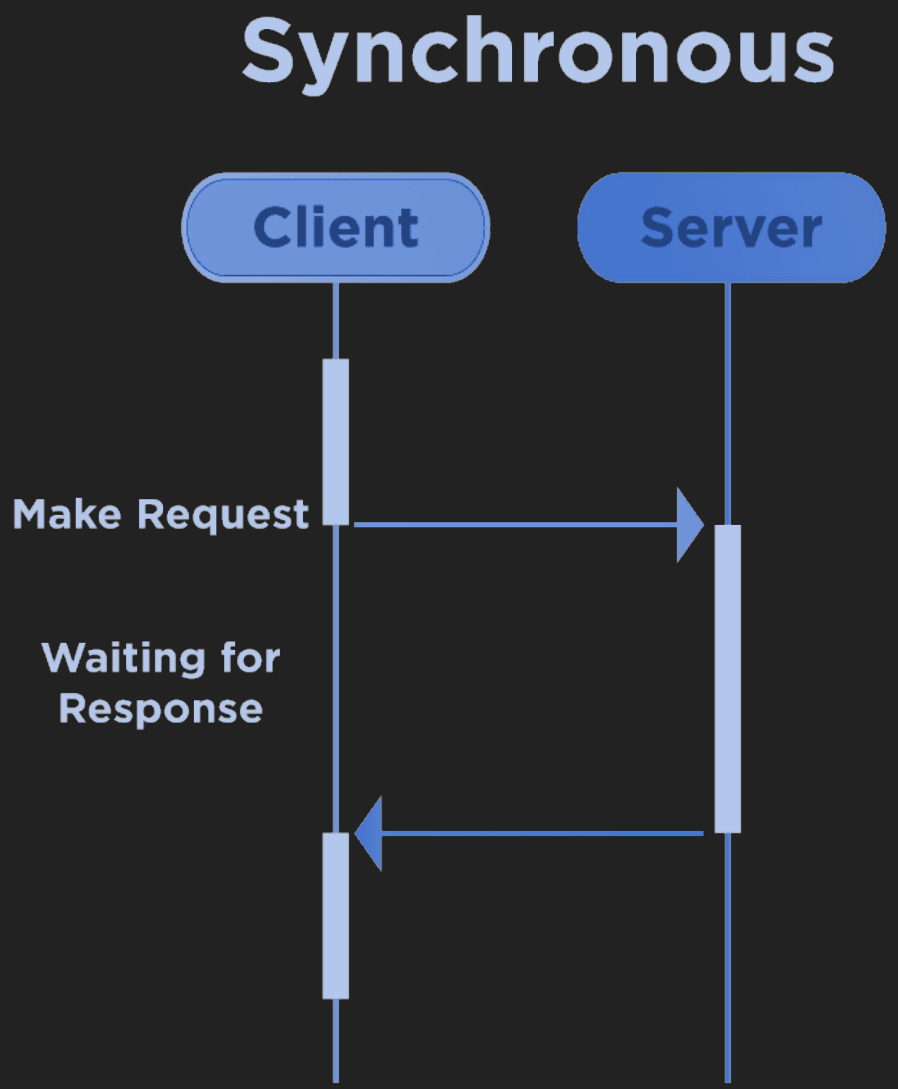
MARCH 2023

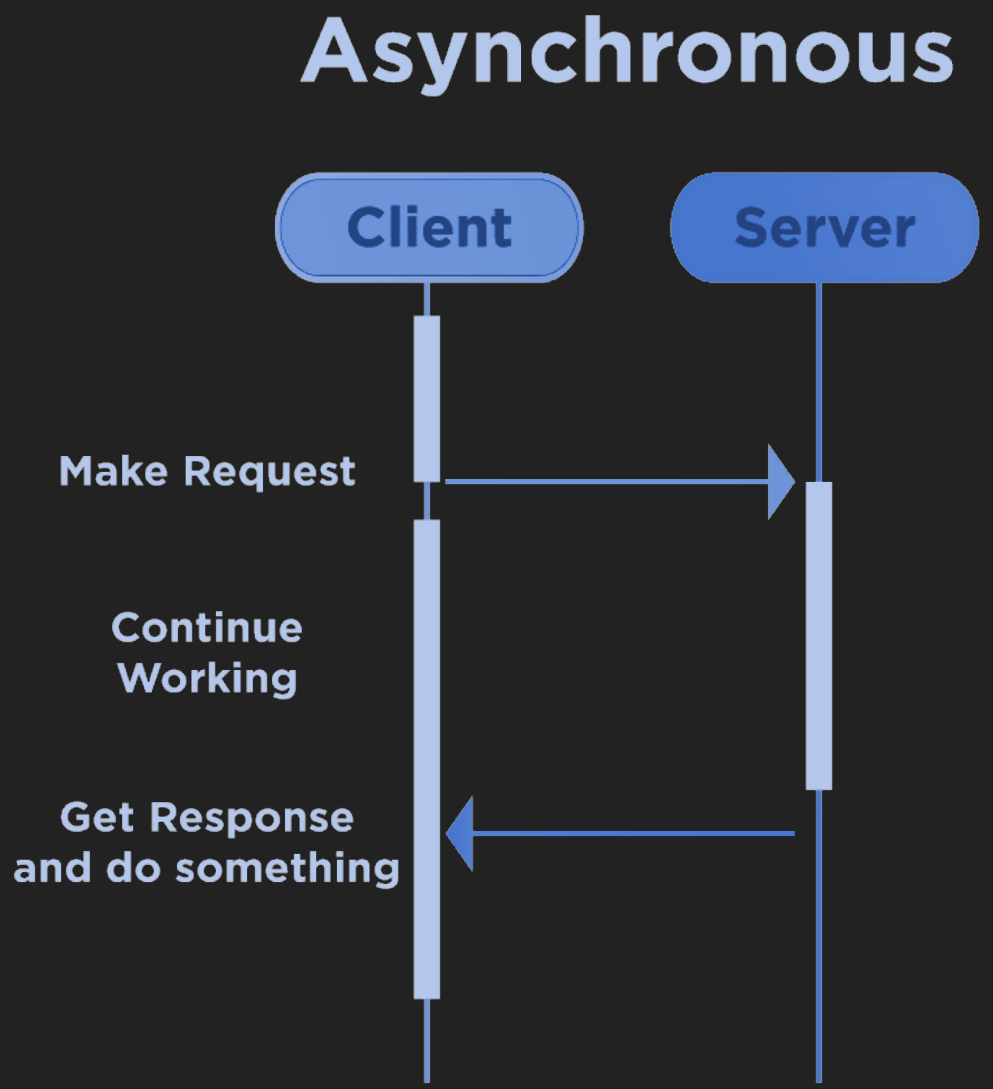
Contents

Typical microservice communication way

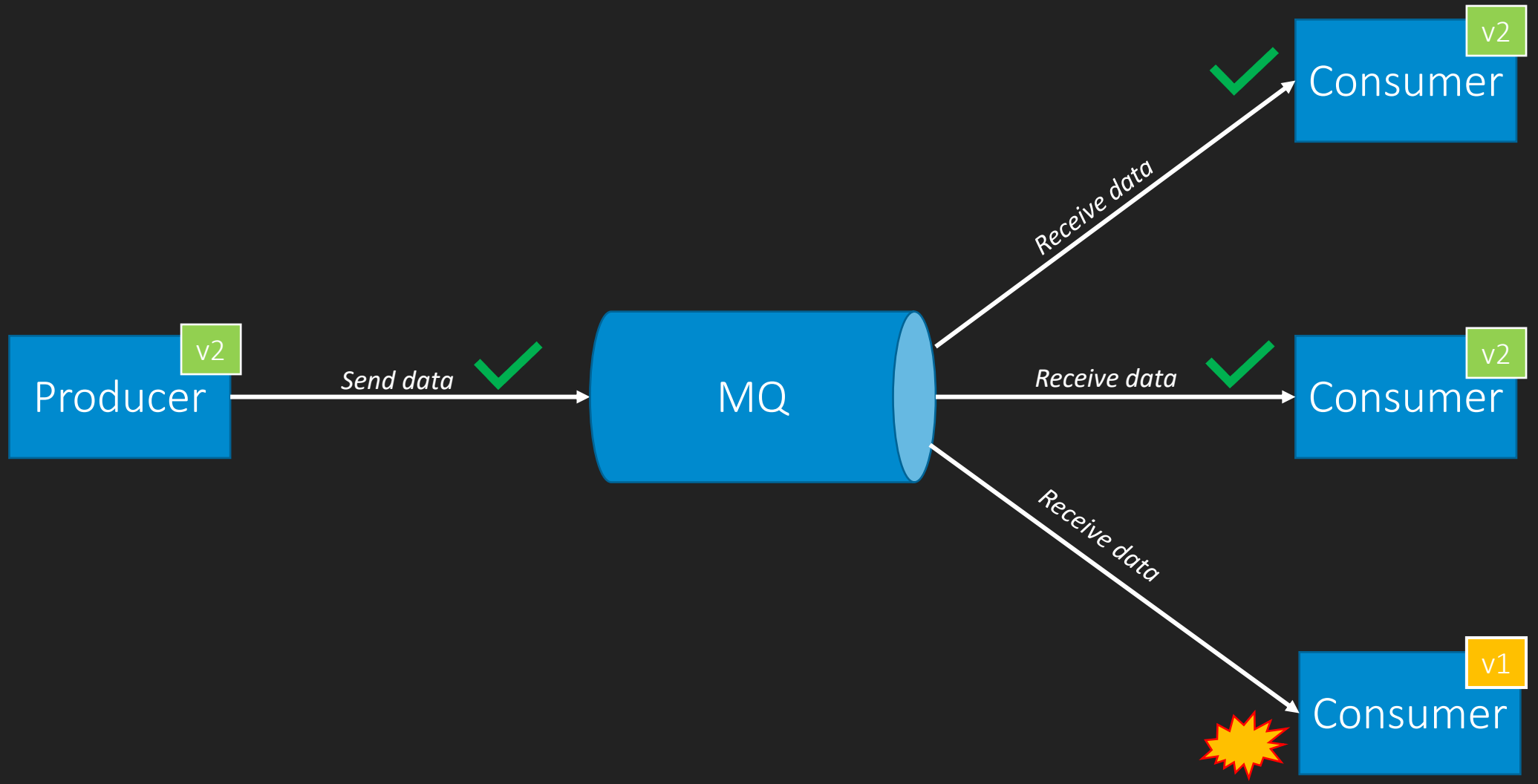


Synchronous vs Asynchronous

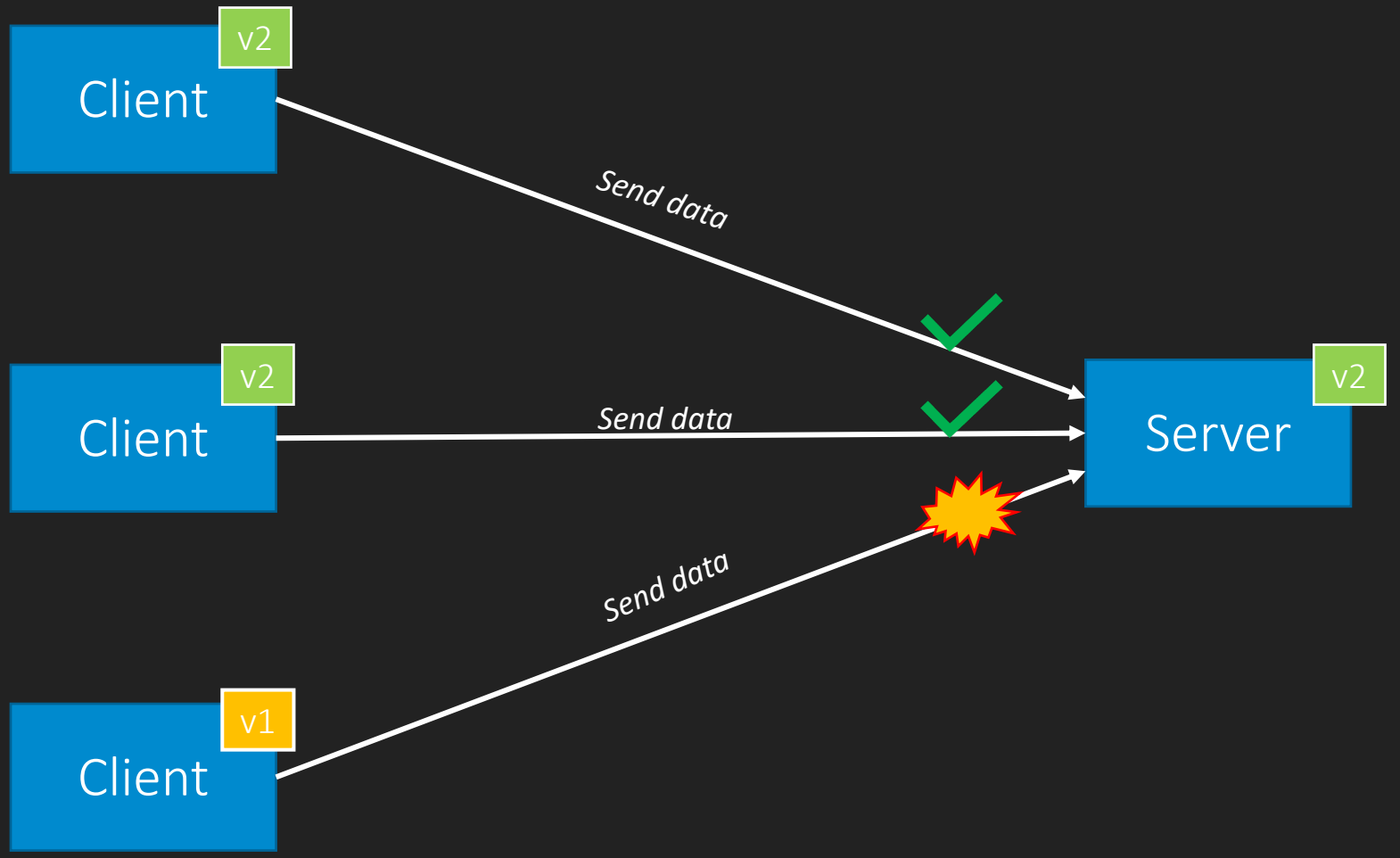




Contract-First API Development

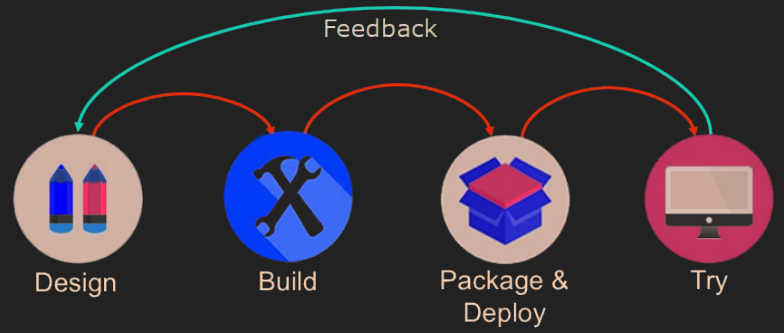


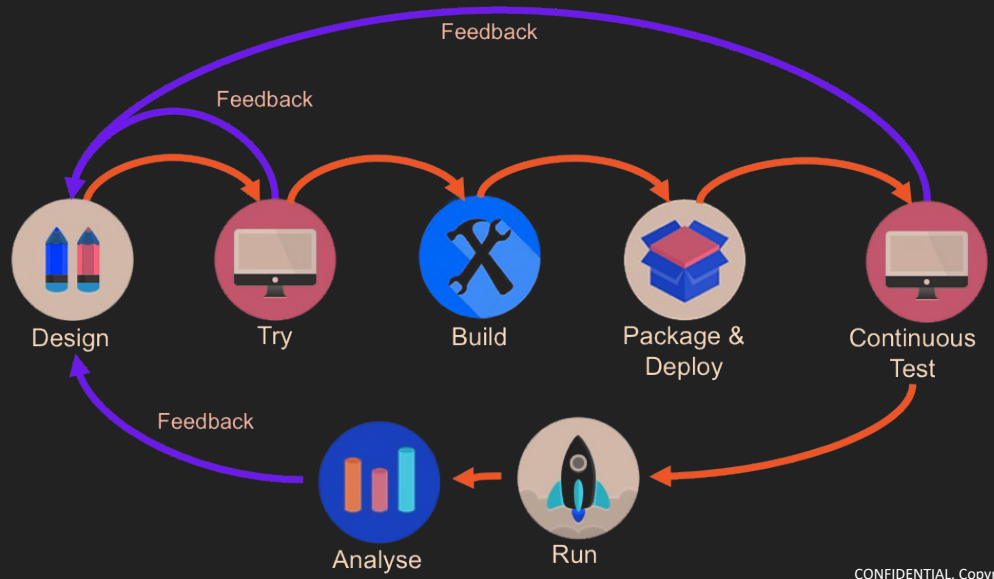
Contract-First API Development



Why Contract-First?

- ✓ Because you want to allow people to work independently
- ✓ Because you want to ensure consistency
- ✓ Because you need strong guarantees about service contracts
- ✓ Because you, your team, your customers, and your partners can collaborate
- ✓ Because you can save time by using code generators and testing tooling





Mis-understanding

- Endpoint documentation?
- Data format & validation?
- Infrastructure access?
- Event availability?
- Service providers and consumers compliance?
- Slow time to market?
- Poor quality?

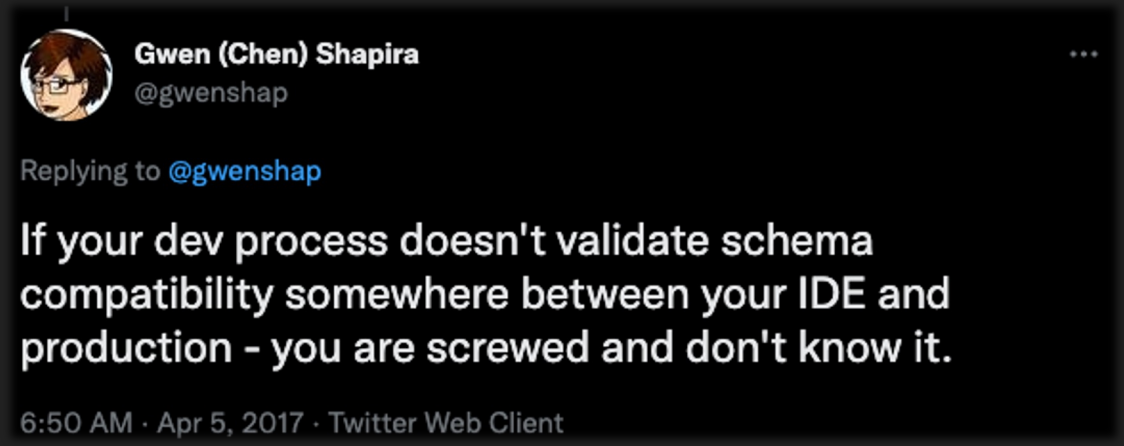
Issues needs to be addressed

- The schemas & API specs are subject to change
- Evolution (and validation) of the schemas & API specs
- Transparency and efficiency
- Central registry where the schemas are stored and accessible
- Prevention of bad data on the consumer side

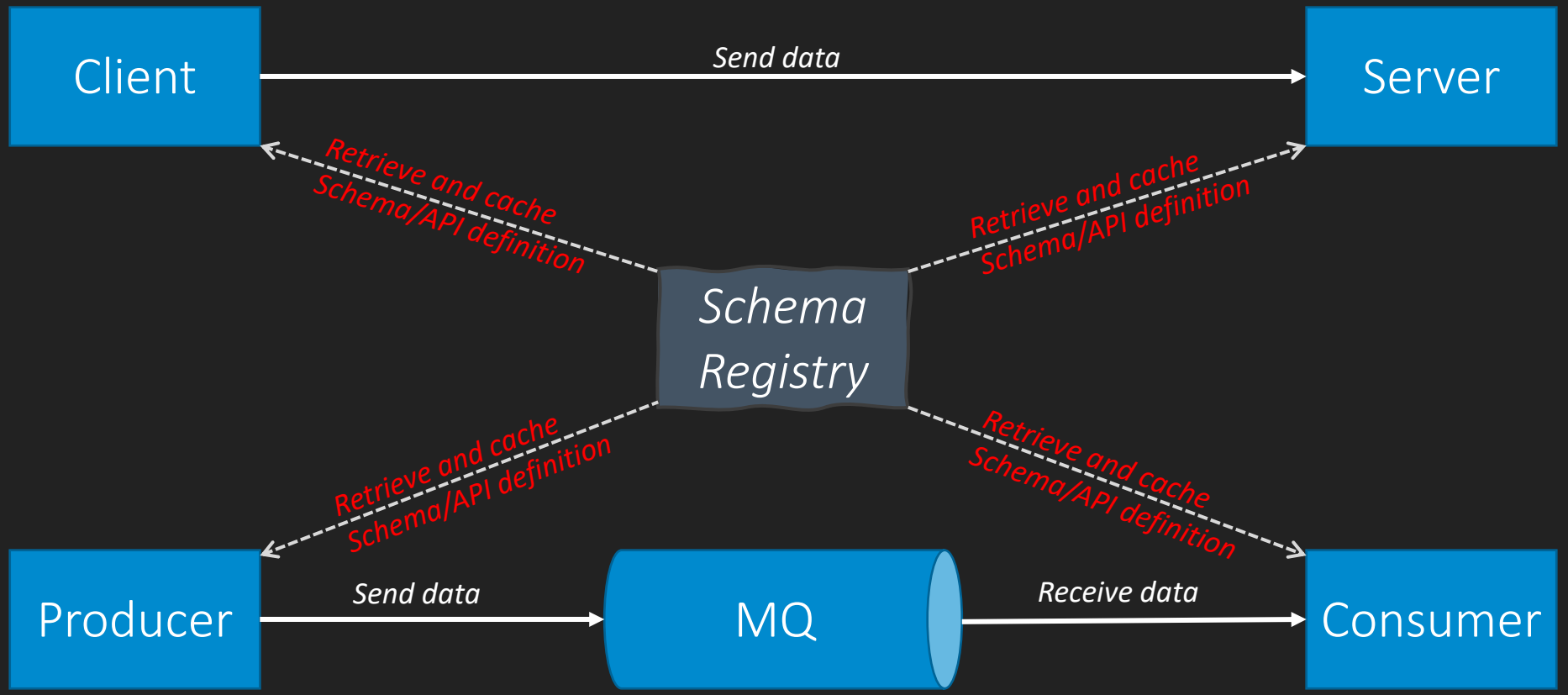
Fact: a message broker usually is not aware of schema formats

Good practices

- Add schema (evolution) validation to CI/CD pipelines
- Create a new resource if you need to break compatibility
- Enable schema validation on the consumer side



Using the Schema Registry in the communication

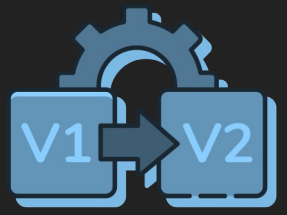


What are the required capabilities of the registry?

Artifact Management

Data Formats

Authentication & Authorization



Evolution & Versioning

Auditability

Popular OSS Schema Registry Comparison

VS

Confluent Schema Registry vs Apicurio Registry

Criteria		Apicurio Registry	Confluent Schema Registry
Schema registration, evolution, and validation		✓	✓
Support schemas	Apache Avro	✓	✓
	JSON Schema	✓	✓
	Protobuf	✓	✓
	GraphQL	✓	✗
	OpenAPI	✓	✗
	AsyncAPI	✓	✗
	WSDL	✓	✗
	XML Schema	✓	✗
Authorization		✓	✗
Auditing (tech. event sourcing)		✓	✗

Confluent Schema Registry vs Apicurio Registry

Criteria		Apicurio Registry	Confluent Schema Registry
Technology stack		Java, Quarkus based	Java, based on Confluent's libraries
Licensing		OSS	OSS with paid addons
Persistence	Kafka	✓	✓
	SQL Database	✓	✗
	In-memory	✓	✗
Authentication	HTTP Basic	✓	✓
	OpenID Connect	✓	✗
Authorization	Role-based	✓	OSS ✗ Paid ✓
	Content-based	✓	✗
GUI		✓	OSS ✗ Paid ✓
Multitenancy		✓	✗

APICURIO

Content rule maturity

- ✓ **Validation Rules**
  - The artifact must have valid content, or the server will reject it
  - Can check for valid syntax (SYNTAX_ONLY)
  - Can also check for valid semantics (FULL)

- ✓ **Compatibility Rules**
  - Determines whether an update is allowed based on configured compatibility requirement setting
  - Multiple compatibility options, including Backwards and Forwards compatible
  - Only relevant for updates (checks the new version against the previous version)
  - Controls the evolution of a single Artifact over time

Schema Compatibility (for AVRO & JSON Schema)

Compatibility Type	Changes allowed	Check against which schemas	Upgrade first
BACKWARD	• Delete fields • Add optional fields	Last version	Consumers
BACKWARD_TRANSITIVE	• Delete fields • Add optional fields	All previous versions	Consumers
FORWARD	• Add fields • Delete optional fields	Last version	Producers
FORWARD_TRANSITIVE	• Add fields • Delete optional fields	All previous versions	Producers
FULL	• Add optional fields • Delete optional fields	Last version	Any order
FULL_TRANSITIVE	• Add optional fields • Delete optional fields	All previous versions	Any order
NONE	• All changes are accepted	Compatibility checking disabled	Depends

Content rule maturity

Artifact type	Validity rule	Compatibility rule
Avro	Full	Full
Protobuf	Full	Full
JSON Schema	Full	Full
OpenAPI	Full	None
WSDL	Full	None
XSD	Full	None
AsyncAPI	Syntax Only	None
GraphQL	Syntax Only	None
Kafka Connect	Syntax Only	None

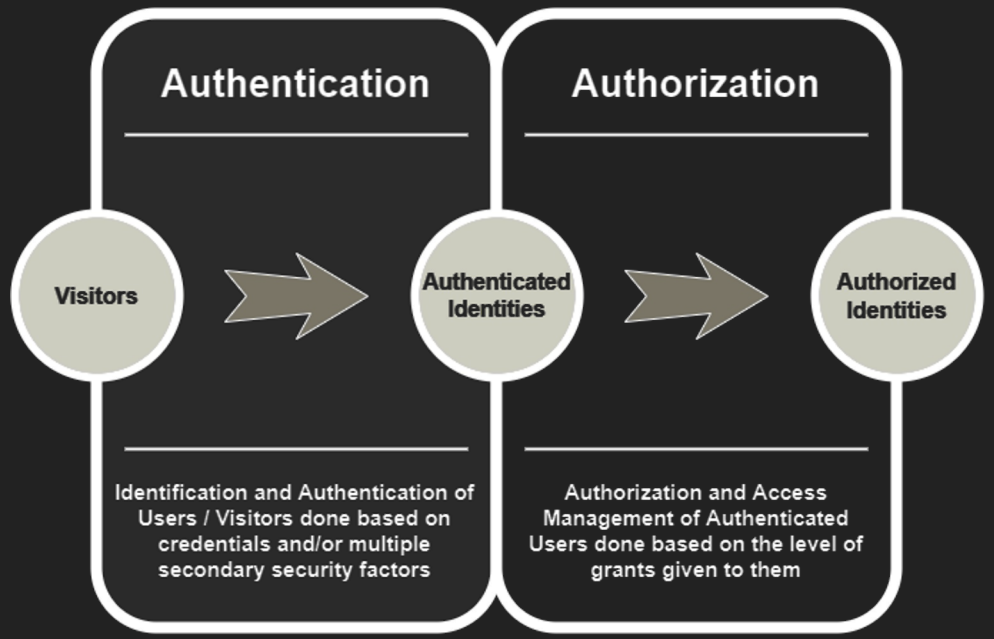
Compatibility tips for AVRO & JSON Schema

- ✓ Do you want to upgrade the producer without touching consumers?
  - ✓ You want forward compatibility
  - ✓ You can add fields
  - ✓ You can delete fields with defaults
- ✓ Do you want to update the schema in storage but still read old messages?
  - ✓ You want backward compatibility
  - ✓ You can delete field
  - ✓ You can add fields with defaults
- ✓ Both?
  - ✓ You want full compatibility
  - ✓ Default all things!
  - ✓ Except the primary key which you never touch

Authentication & Authorization

Authentication options:

- OpenID Connect with KeyCloak
- HTTP Basic



Authorization options:

- RBAC
- Owner-only

Role	Read artifacts	Write artifacts	Global rules	Summary
ADMIN	Yes	Yes	Yes	Full access to all create, read, update, and delete operations.
DEVELOPER	Yes	Yes	No	Access to create, read, update, and delete operations, except configuring global rules. This role can configure artifact rules.
READ_ONLY	Yes	No	No	Access to read and search operations only. This role cannot configure any rules.

Event sourcing

Apicurio Registry to send events when changes are made to the registry, the event types include:

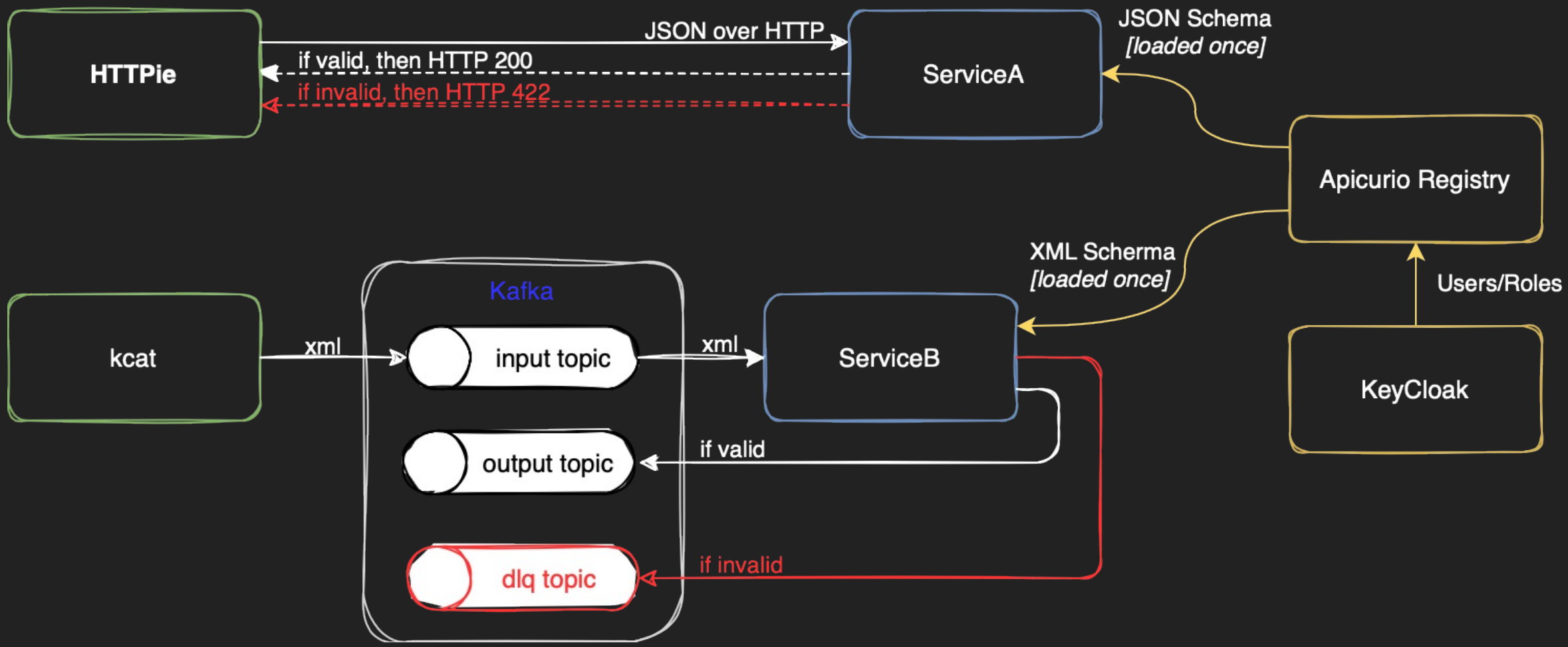
- io.apicurio.registry.artifact-created
- io.apicurio.registry.artifact-updated
- io.apicurio.registry.artifact-rule-created
- io.apicurio.registry.global-rule-created

The currently implemented protocols:

- HTTP
- Apache Kafka


DEMO

Demo



The sources are available by the link: https://github.com/stn1slv/meetup-schema-registry


ANY
QUESTIONS?