
GitOps for Apache Kafka®
At scale and with ease using Jikkou!
The Opensource Resource as Code Framework for Apache Kafka®
@fhussonnois

Florian Hussonnois

Lead Software Engineer **@Kestra**.

- Worked for 12 years as Consultant & Trainer
- 10+ years of experience with Apache Kafka (version 0.7)
- Open-source projects such as Kafka Connect File Pulse, Jikkou
- Confluent Community Catalyst since 2019

@fhussonnois

The True Story Behind Every Kafka Resources!

It all starts with Kafka's CLI.

```
#!/bin/bash
kafka-topics --bootstrap-server localhost:9092 \
  --create --topic my_topic \
  --partitions 6 \
  --replication-factor 1

# (output)
# Created topic my-topic.
```

in a galaxy far, far away...

Padawan: "Can you please create a new topic for my team?"

Jedi: "Yes, of course! How many partitions do you need?"

Padawan: "...*(silence)*"

What About Configuration ?

Yet another story!

```
#!/bin/bash
# Update to new value
kafka-configs --bootstrap-server localhost:9092 \
  --entity-type topics --entity-name my_topic \
  --alter --add-config retention.ms=17280000

#!/bin/bash
# Reset to default value
kafka-configs --bootstrap-server localhost:9092 \
  --entity-type topics --entity-name my_topic  \
  --alter --delete-config retention.ms
```

in a galaxy far, far away...

Amiral: "Could you update the config of this topic?"

Stormtrooper: "Yes, give me a second, please."

Amiral: "Why did we lose all our data?"

Stormtrooper: "Oops, we forgot a zero in rentention.ms."

What’s the problem ?

With Manual Configuration Management

No versioning

- How to keep track of changes over time?

Non-repeatable

- How to reproduce modifications on multiple environments ?

No traceability

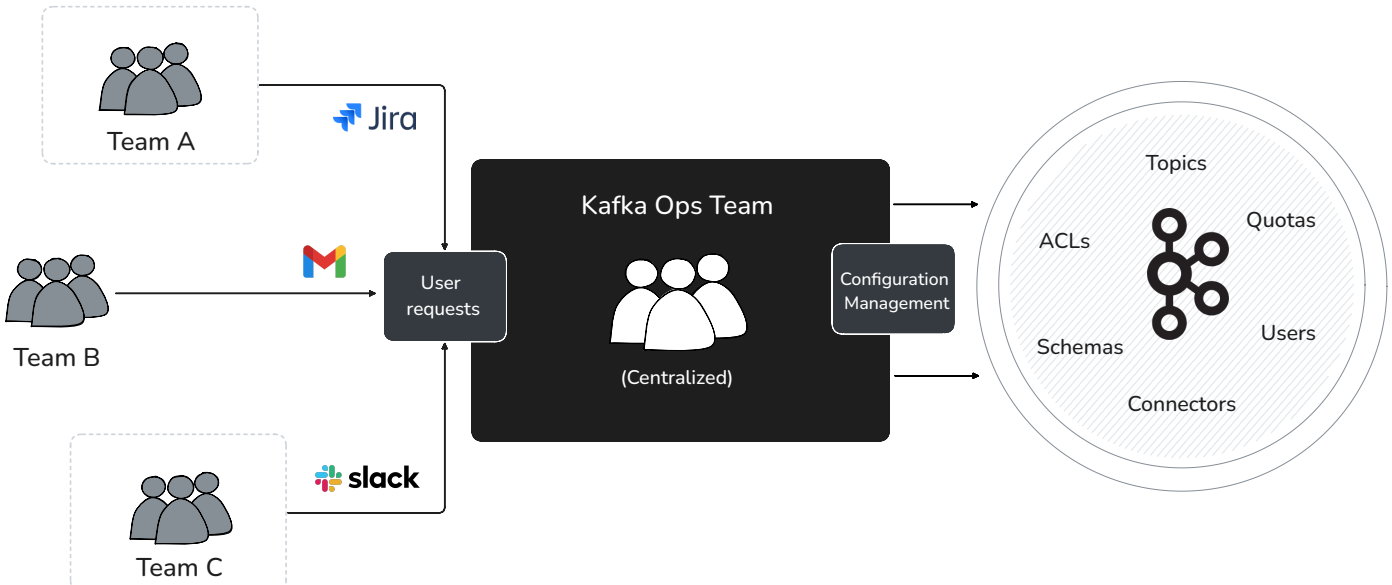
- Who did what and when ?

No human fault-tolerance

- One typo, and boom!

Hard to Scale!

Centralized team can quickly become a bottleneck



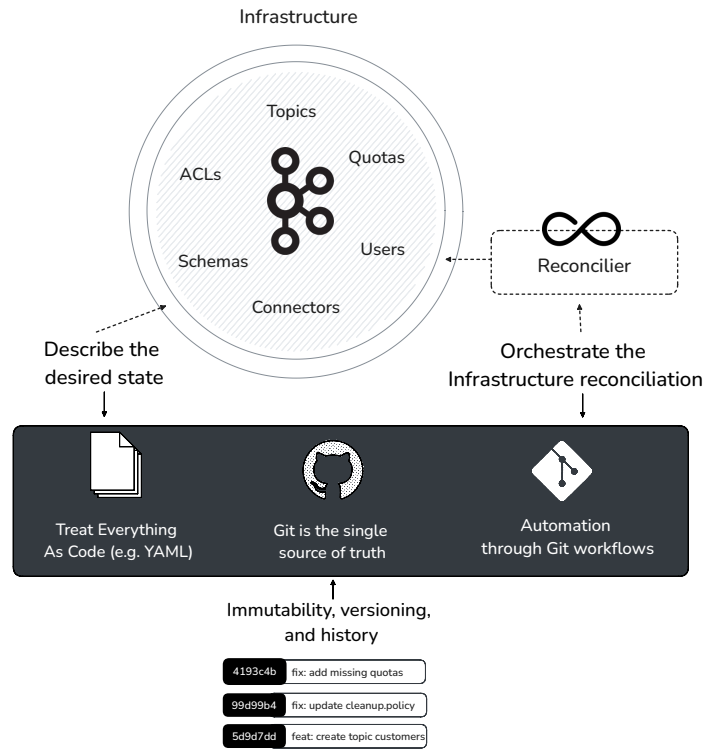
- Creating new topics or requesting access takes hours instead of minutes.

The GitOps and IaC Approach

Everything As Code!

Synchronize the desired states (*plan*) of our resources and configurations with the actual state of our infrastructure from Git.

- Enables a decentralized approach - multiple repositories for each team.
- Shifts responsibilities to development teams.
- Higher autonomy



Existing IaC Solutions

Why they don’t fit ?

- **Terraform** (or OpenTofu)
  - Often preferred by **DevOps** teams, but not as commonly used by development teams.
  - Have to learn HCL - Multiple providers for Kafka
- **Strimzi** (Kubernetes Operators) - Helm
  - Not everyone runs on Kubernetes, or has the ability to deploy an operator.

Not specialized solutions: no control over the pushed configurations.

Introducing Jikkou

The Opensource Resource as Code Framework for Apache Kafka®

What is Jikkou ?

Jikkou is a flexible framework enabling developers and Devops teams to efficiently manage, automate and provision all the resources required for their projects.

- Open Source - **Apache License, Version 2.0**.
- Written in Java.
- Declarative (YAML) - Resource as Code.
- Easy to versioning, Tracking changes over time.
- GitOps - Continuously Reconciliation
- Extensible - Developed for **Apache Kafka®**, but designed for anything!

Jikkou (jikkō/実行), means "execution (e.g. of a plan), carrying out, (putting into) practice in Japanese.

Everything As A Resource

Express the desired state of resources using YAML descriptor files.

```yaml
---
# ./my-topic.yaml
apiVersion: "kafka.jikkou.io/v1beta2"
kind: KafkaTopic
metadata:
  name: 'my-topic'
spec:
  partitions: 3
  replicas: 1
  configs:
    min.insync.replicas: 1
    cleanup.policy: 'delete'
```

- Same resource model as Kubernetes to describe the entities to manage.

Jikkou original goal

A modern and intuitive command line client for Apache Kafka.

Jikkou was initially developed as a simple **Command Line Interface (CLI)** to efficiently ***C**reate*, ***R**ead*, ***U**pdate* , ***D**elete*, and version resources such as Topics, ACLs, and Quotas.

```
# Installing Jikkou CLI using SDKMAN!
$ sdk install jikkou
```

Available as a **native image** (built with GraalVM) or **Java Binary distribution**.

Jikkou CLI

```
CORE COMMANDS:
  apply                    Update the resources as described by the resource definition files.
  create                   Create resources from the resource definition files (only non-existing resources will be cre
  delete                   Delete resources that are no longer described by the resource definition files.
  diff                     Show resource changes required by the current resource definitions.
  get                      Display one or many specific resources.
  prepare                  Prepare the resource definition files for validation.
  update                   Create or update resources from the resource definition files
  validate                 Check whether the resources definitions meet all validation requirements.

SYSTEM MANAGEMENT COMMANDS:
  action                   List/execute actions.
  health                   Print or describe health indicators.

ADDITIONAL COMMANDS:
  api-extensions           Print the supported API extensions
  api-resources            Print the supported API resources
  config                   Sets or retrieves the configuration of this client
  generate-completion      Generate bash/zsh completion script for jikkou.
  help                     Display help information about the specified command
```

Configuration

HOCON

```
# ./application.conf
jikkou {
    extension.providers {
      default.enabled: true
    }

    kafka {
        client {
            bootstrap.servers = "localhost:9092"
            bootstrap.servers = ${?KAFKA_BOOTSTRAP_SERVERS}
        }
    }
}
```

Configure context

```
$ jikkou config set-context localhost \
  --config-file "`pwd`/application.conf
```

Use context

```
$ jikkou config use-context localhost
```

Display current configuration

```
$ jikkou config view
```

See: https://github.com/lightbend/config

Reconcile

Apply resource changes

Multiple reconciliation modes:

- `CREATE`: Create new resource.
- `UPDATE`: Create or update existing resources.
- `DELETE`: Delete existing resource.
- `FULL`: Apply any changes.

NOTE: Jikkou supports a `dry-run` mode.

To only display state changes

```
jikkou diff \
--files ./my-topic.yaml \
--selector "metadata.labels.environment IN (demo)"
--output YAML
```

To only create new resources

```
jikkou create \
--files ./my-topic.yaml \
--selector "metadata.labels.environment IN (demo)"
--output YAML
```

Apply

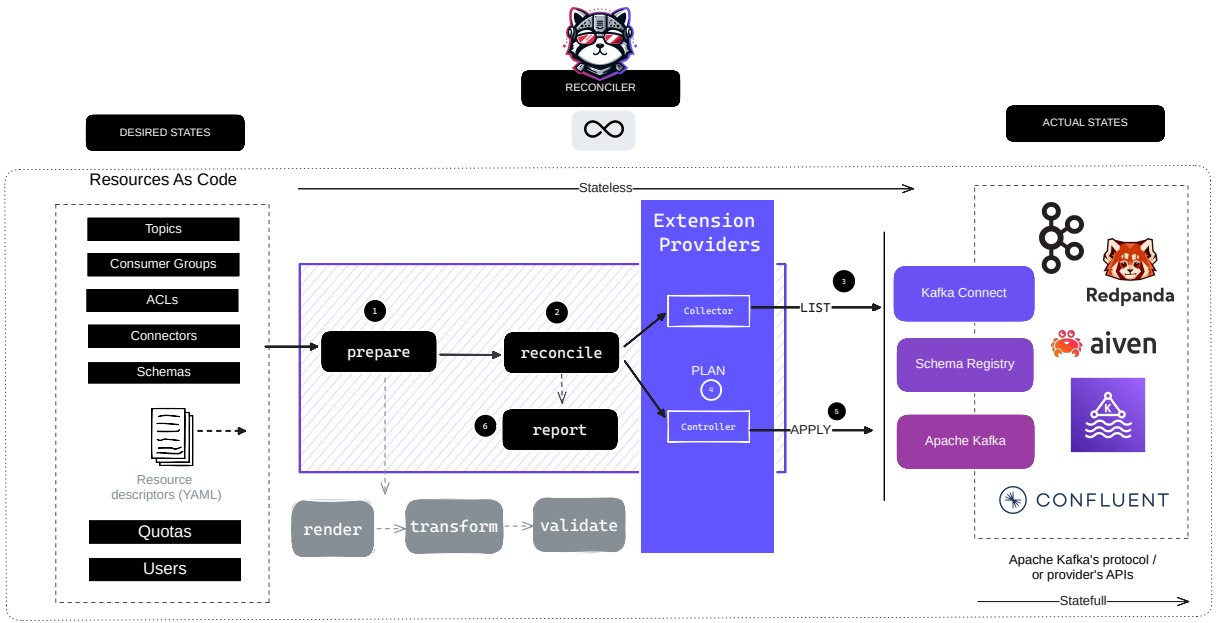
Resource State Changes

(output)

```yaml
---
kind: "ApiChangeResultList"
apiVersion: "core.jikkou.io/v1"
metadata:
  labels: {}
  annotations:
    jikkou.io/changes-count: 1
dryRun: false
results:
- end: "2023-12-20T00:00:00.000000Z"
  status: "CHANGED"
  description: "Create topic 'my-topic' (partitions=3, replicas=1, configs=[cleanup.policy=delete,min.insync.replicas=1
  changed: true
  failed: false
  change:
    apiVersion: "kafka.jikkou.io/v1beta2"
    kind: "KafkaTopicChange"
    metadata:
      name: "my-topic"
```

How It Works ?

Leverage your system as the Source of Truth!



Jikkou adopts a stateless approach and does not store any state internally.

- Seamless integration with other solutions.

Templating

Jikkou provides a simple templating mechanism based-on Jinjava, a Jinja template engine for Java.

Template Resource File

```
# ./kafka-topics.tpl
---
apiVersion: 'kafka.jikkou.io/v1beta2'
kind: 'KafkaTopicList'
items:
{% for country in values.countryCodes %}
- metadata:
    name: "{{ values.topicPrefix}}-iot-events-{{ country }}"
  spec:
    partitions: {{ values.topicConfigs.partitions }}
    replicas: {{ values.topicConfigs.replicas }}
    config:
      retention.ms: 3600000
      max.message.bytes: 20971520
{% endfor %}
```

Data Values File

```
# ./values.yaml
---
topicConfigs:
  partitions: 4
  replicas: 1
topicPrefix: "{{ system.env.TOPIC_PREFIX | default('test', true)
countryCodes:
  - fr
  - uk
  - it
```

Command

```
#!/bin/bash
jikkou prepare \
--files kafka-topics-template.tpl \
--values-files kafka-topics-values.yaml
```

Validate

Ensure that inbound resources conform to specific rules or constraints.

Example: Validate that topic names match a specific regex.

```
# ./application.conf
jikkou {
  validations = [
    {
      name = "topicMustHaveValidName"
      type = io.streamthoughts.jikkou.kafka.validation.TopicNameRegexValidation
      priority = 100
      config = {
        topicNameRegex = "[a-zA-Z0-9\\._\\-]+"
      }
    }
}
```

Transform

Transform, enrich, or filter inbound resources

Example: Enforce a minimum value for the replication factor of kafka topics.

```
# ./application.conf
jikkou {
  transformations: [
    {
      type = io.streamthoughts.jikkou.kafka.transform.KafkaTopicMinReplicasTransformation
      priority = 100
      config = {
        minReplicationFactor = 3
      }
    }
  ]
}
```

Actions

Allow a user to execute a specific and one-shot operation on resources.

Using Jikkou CLI:

```
$ jikkou action <ACTION_NAME> execute [<options>]
```

Built-in actions:

- `KafkaConsumerGroupsResetOffsets`
  - Reset offsets of consumer group to earliest, lastest, a specific offset or from datetime.
- `KafkaConnectRestartConnectors`
  - Restarts all or just the failed Connector and Task instances for one or multiple named connectors.

"Ok, it’s very cool! But…

How to use Jikkou if you don’t have a direct public access to the your Kafka cluster?"

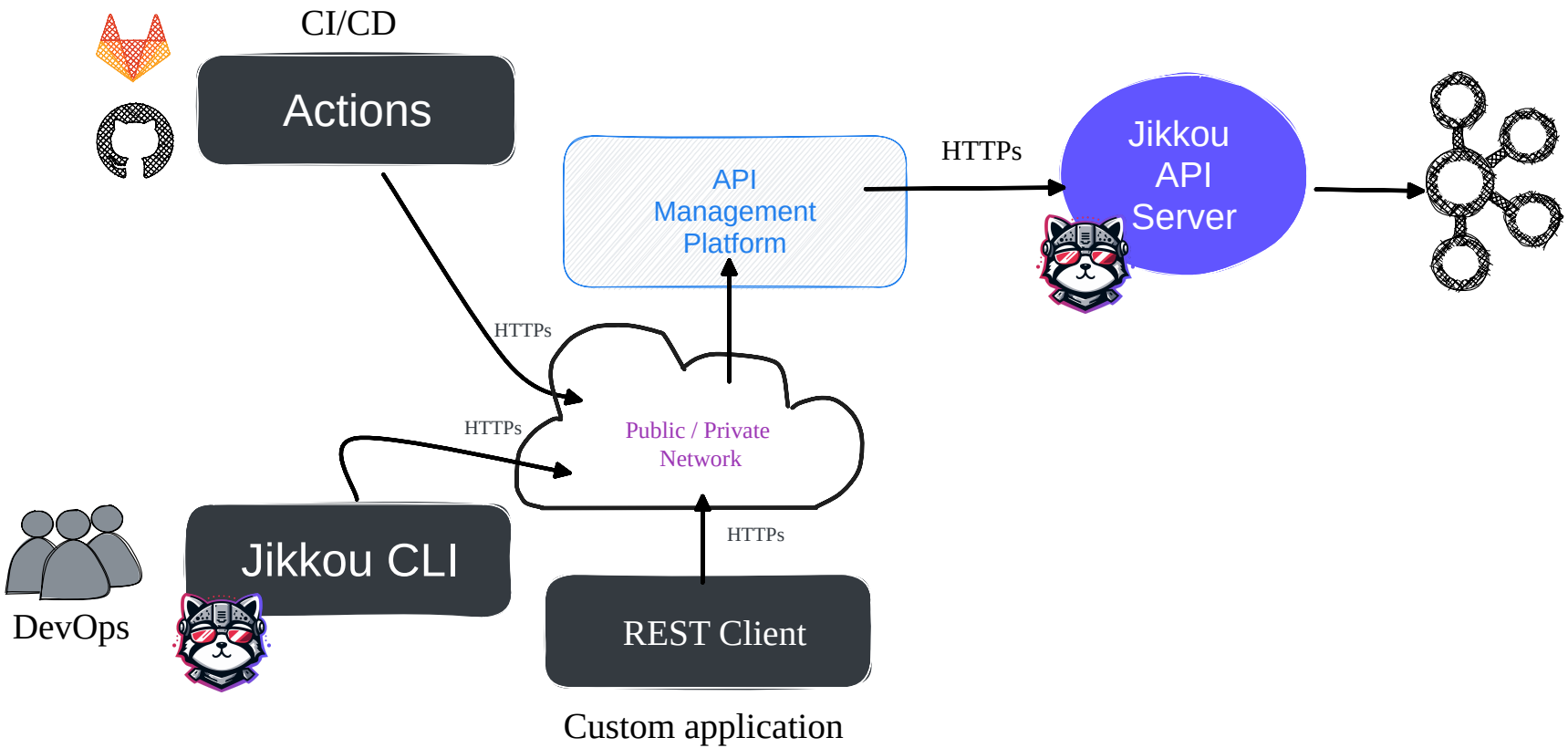
Jikkou API Server

A REST interface that makes it easy to manage and automate all your data platform resources.

```
$ docker run -it --net host streamthoughts/jikkou-api-server:latest
```

Jikkou API Server

Usage



Jikkou API Server

Key Benefits

- **Improved Security** :
  - Can sit behind your API management platform.
  - Provides additional security and auditing capabilities.
- **Centralized Governance** :
  - Acts as a gateway to your data platform.
  - Allows to centralize validation/transformation rules to manage resources.
- **Can be extended** :
  - Supports custom resources and extensions.

REST APIs

Jikkou API Server is built with Micronaut Framework

Listing resources.

`GET /apis/{group}/{version}/{plural}`

Creating resources.

`POST /apis/{group}/{version}/{plural}/reconcile/mode/{mode}{?dry-run}`

Security

- Micronaut Framework supports multiple authentication mechanisms.
- Jikkou API Server with Basic Authentication, JWT, X.509 Certificate, OAuth2, etc.

Jikkou CLI / Proxy Mode

Do not change your developer experience

```
jikkou {
  # Proxy Configuration
  proxy {
    # Specify whether proxy mode is enabled (defaul
    enabled = true
    # URL of the API Server
    url = "http://localhost:28082"
  }
}
```

DEMO

Learn More

Documentations · GitHub · Blog Posts

Thank You

Questions ?

⭐ Star the project on GitHub: https://github.com/streamthoughts/jikkou