**Proof of Concept**

Using Spring Cloud Config to consolidate and serve runtime configuration from different sources simultaneously.

Author Aleksandrs Krivickis

[Alex.a.krivickis@hsbc.com.hk](mailto:Alex.a.krivickis@hsbc.com.hk)

Contents

[Scope 3](#_Toc526423674)

[Design 3](#_Toc526423675)

[Investigation 3](#_Toc526423676)

[Supported backends 3](#_Toc526423677)

[GitHub with Spring Cloud Config Server 4](#_Toc526423678)

[Features 4](#_Toc526423679)

[MySQL with Spring Cloud Config Server 6](#_Toc526423680)

[MySQL(MariaDB) with Spring Cloud Config Server 7](#_Toc526423681)

[MongoDB with Spring Cloud Config Server 9](#_Toc526423682)

[GitHub and MySQL with Spring Cloud Config Server 10](#_Toc526423683)

[GitHub and MySQL with Spring Cloud Config served to a Spring Boot API 11](#_Toc526423684)

[GitHub and MySQL with Spring Cloud Config served to a Spring Boot API through Event Queue 12](#_Toc526423685)

[Investigation summary 12](#_Toc526423686)

[Limitations 12](#_Toc526423687)

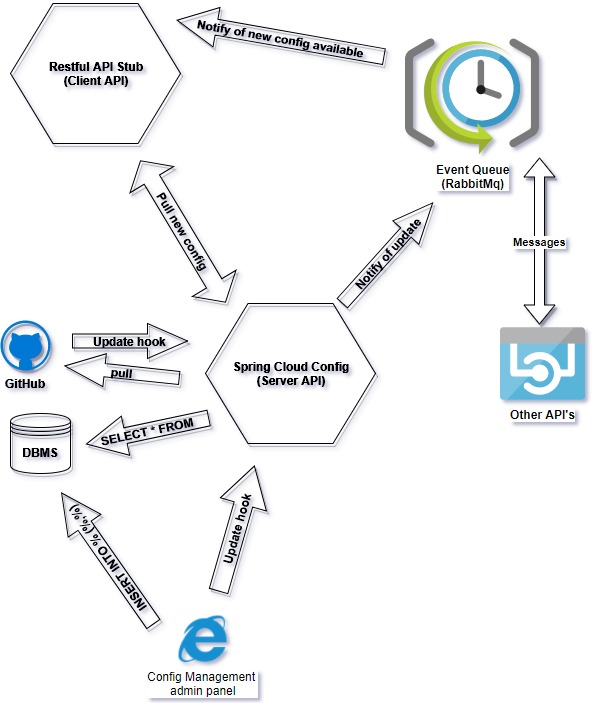
[Conclusions 13](#_Toc526423688)

[References 13](#_Toc526423689)

# Scope

* Find out if it is possible to reliably serve configuration from git and DBMS system simultaneously.
* Check if it is possible to Spring Cloud Config server in combination with Event Queue.
* Implement the system with different DBMS and evaluate them.
* Check if it is possible to store and serve configuration data in JSON format, explore limitations as there might be some in relational DBMS.

# Design



# Investigation

## Supported backends

* Local filesystem
* Git
* MySQL
* MySQL(MariaDB)
* MongoDB
* Cosmos DB

Azure Cosmos DB offers MongoDB API as a service at the protocol level.

* PostgreSQL

## Configuration file/document structure

One of the requirements is to ensure that we have several labels and profiles for the same applications.

### Theoretical folder structure in git repository:

/{application}/{profile}[/{label}]

/{application}-{profile}.yml

/{label}/{application}-{profile}.yml

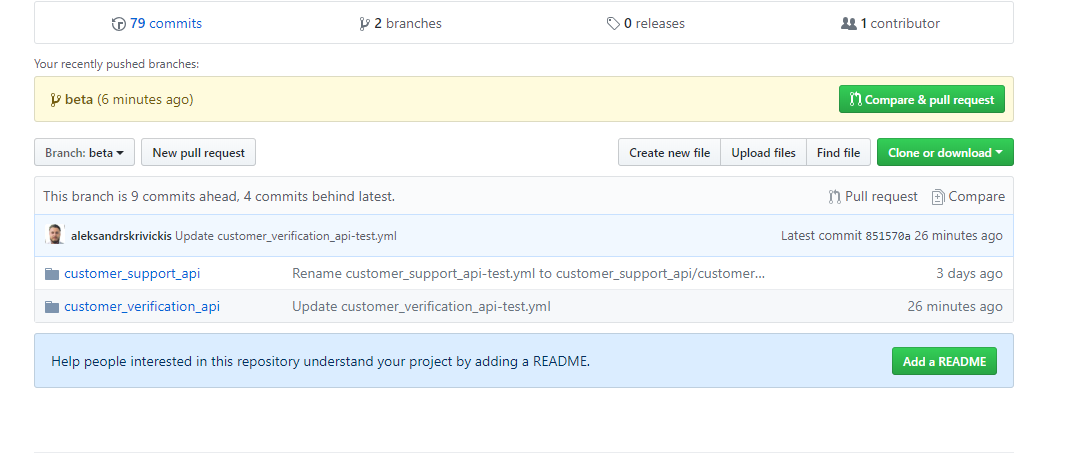
/{application}-{profile}.properties

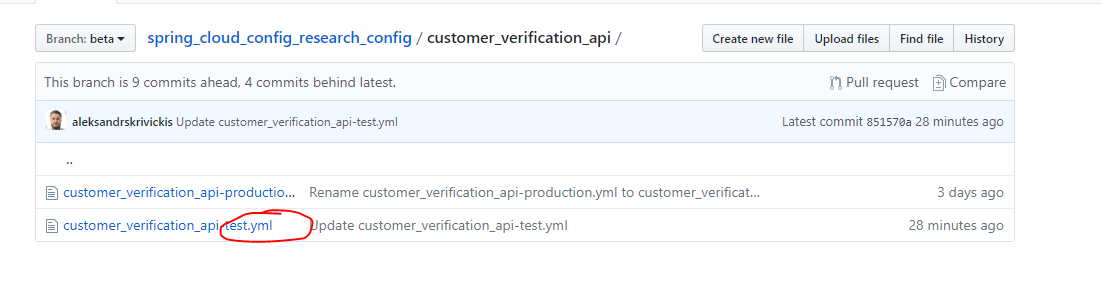
/{label}/{application}-{profile}.properties

### Folder structure example

Let’s imagine that we have an application **customer\_support\_api** and **customer\_verification\_api** with **production** and **test** profiles, all labelled as **latest** and **beta.**

* **Application’s name is defined by a folder’s name on GitHub**
* **Version is defined by the name of branch on Github**
* **Profile is the file’s name**



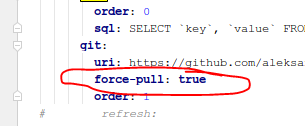


## GitHub with Spring Cloud Config Server

<https://github.com/aleksandrskrivickis/spring_cloud_config_research_config>

It is served in a way which is different from JDBC. For every label on GitHub user has to create separate branch.

Also, there is a feature **force-pull,** that allows to pull git repo and serve fresh config data on-demand. Theoretically there is no need to create a callback/hook from GitHub.



### Features

#### Using multiple repositories

spring:

cloud:

config:

server:

git:

uri: https://git/common/config-repo.git

force-pull: **true**

repos:

team-a:

pattern: team-a-\*

uri: http://git/team-a/config-repo.git

force-pull: **true**

team-b:

pattern: team-b-\*

uri: http://git/team-b/config-repo.git

force-pull: **true**

team-c:

pattern: team-c-\*

uri: <http://git/team-a/config-repo.git>

local:

pattern: local\*

uri: file:/home/configsvc/config-repo

#### Git url placeholders:

Spring Cloud Config Server supports a git repository URL with placeholders for the {application} and {profile} (and {label} if you need it, but remember that the label is applied as a git label anyway). So you can easily support a "one repo per application" policy using (for example):

spring:

cloud:

config:

server:

git:

uri: https://github.com/myorg/{application**}**

#### Defining url pattern:

The pattern property in the repo is actually an array, so you can use a YAML array (or [0], [1], etc. suffixes in properties files) to bind to multiple patterns. You may need to do this if you are going to run apps with multiple profiles. Example:

spring:

cloud:

config:

server:

git:

uri: https://github.com/spring-cloud-samples/config-repo

repos:

development:

pattern:

- '\*/development'

- '\*/staging'

uri: https://github.com/development/config-repo

staging:

pattern:

- '\*/qa'

- '\*/production'

uri: https://github.com/staging/config-repo

#### Search path

Spring Cloud Config Server also supports a search path with placeholders for the {application} and {profile} (and {label} if you need it). Example:

spring:

cloud:

config:

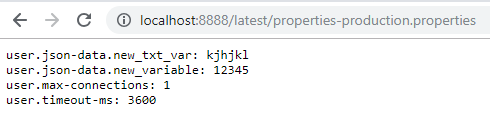
server:

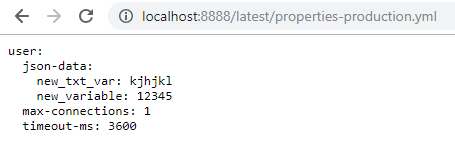
git:

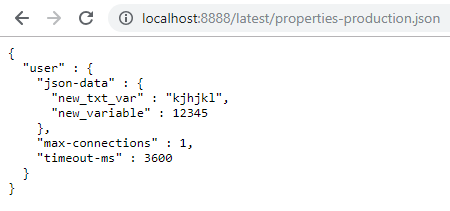
uri: https://github.com/spring-cloud-samples/config-repo

searchPaths: '{application}'

searches the repository for files in the same name as the directory (as well as the top level). Wildcards are also valid in a search path with placeholders (any matching directory is included in the search).



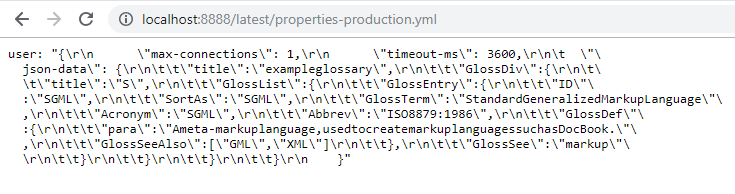


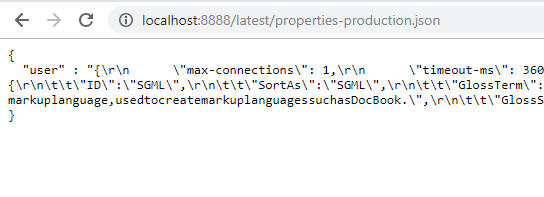


## MySQL with Spring Cloud Config Server

Same limitations and structure of response as with MariaDB. Different driver has to be used for establishing connection.

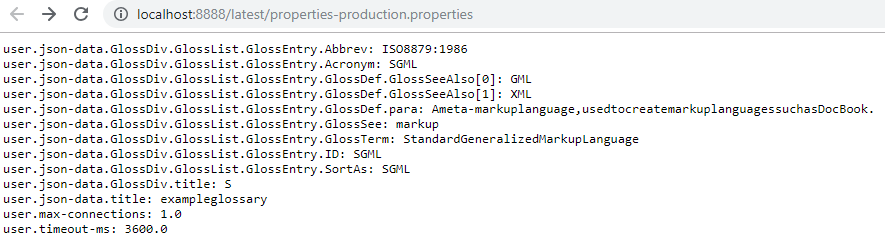


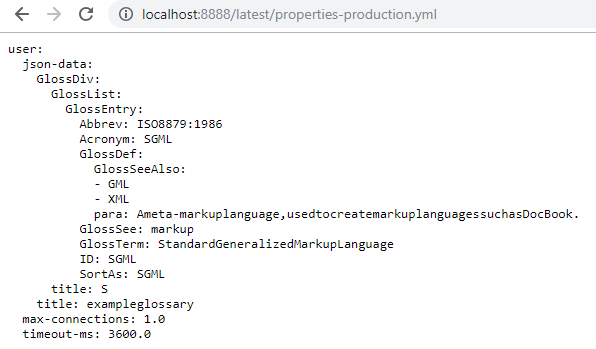




## MongoDB with Spring Cloud Config Server

It is possible to serve MongoDB database in three different formats: properties, yml/yaml, json. In order to display json with indents property “**spring.jackson.serialization.indent\_output=true**” has to be set.

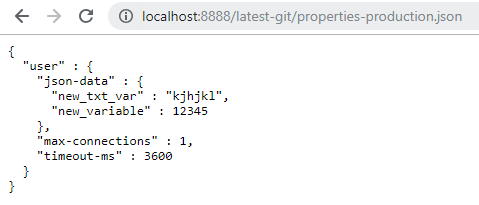


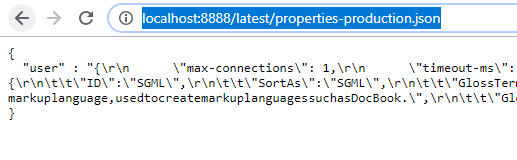




## GitHub and MySQL with Spring Cloud Config Server

It is possible to combine GitHub and MySQL data sources. Unfortunately, there is one limitation – GitHub and MySQL data can’t have the same label. For example: in case if we want to use label “**latest**” to fetch data from MySQL database we can’t create a branch with name “**latest**” as Spring Cloud application will throw an exception and none of data will be available. Unfortunately, to make both data sources work together we have to create an empty branch named “**latest**” and branch named “**latest-git**” in a Git repository. This is the only way for now.



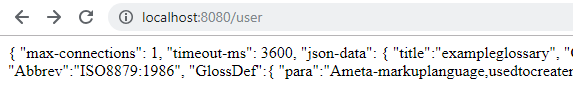


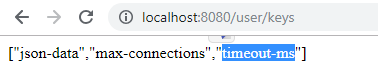
In the next chapter I will try to set-up Spring Cloud Config Client.

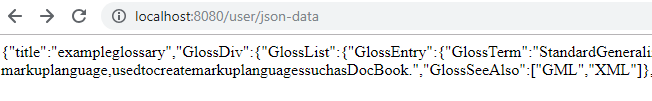
## GitHub and MySQL with Spring Cloud Config served to a Spring Boot API

I have managed to set-up Spring Cloud Config client, that successfully fetches data from Spring Cloud Config Server and makes it available inside of a class. It is possible to access nested json values within the Config Client(API).

@Value(**"${user}"**)  
**private** String **userJson1**;  
@RequestMapping(value = **"/user/{key}"**, method = { RequestMethod.***GET***})  
@ResponseBody  
**public** String getFoosBySimplePathWithPathVariable(  
 @PathVariable(**"key"**) String key) {  
 JSONObject jsonObj = **new** JSONObject(**this**.**userJson1**);  
 **return** jsonObj.get(key).toString();  
}







**From the point of view I currently have, I would implement a converter, which gets root json data and converts it to hashmap of hashmaps. It would allow ease of access to variables. Example from StackOverflow below:**

**public static** Map<String, Object> jsonToMap(JSONObject json) **throws** JSONException {  
 Map<String, Object> retMap = **new** HashMap<String, Object>();  
  
 **if**(json != JSONObject.***NULL***) {  
 retMap = *toMap*(json);  
 }  
 **return** retMap;  
}  
  
**public static** Map<String, Object> toMap(JSONObject object) **throws** JSONException {  
 Map<String, Object> map = **new** HashMap<String, Object>();  
  
 Iterator<String> keysItr = object.keys();  
 **while**(keysItr.hasNext()) {  
 String key = keysItr.next();  
 Object value = object.get(key);  
  
 **if**(value **instanceof** JSONArray) {  
 value = *toList*((JSONArray) value);  
 }  
  
 **else if**(value **instanceof** JSONObject) {  
 value = *toMap*((JSONObject) value);  
 }  
 map.put(key, value);  
 }  
 **return** map;  
}  
  
**public static** List<Object> toList(JSONArray array) **throws** JSONException {  
 List<Object> list = **new** ArrayList<Object>();  
 **for**(**int** i = 0; i < array.length(); i++) {  
 Object value = array.get(i);  
 **if**(value **instanceof** JSONArray) {  
 value = *toList*((JSONArray) value);  
 }  
  
 **else if**(value **instanceof** JSONObject) {  
 value = *toMap*((JSONObject) value);  
 }  
 list.add(value);  
 }  
 **return** list;  
}

## GitHub and MySQL with Spring Cloud Config served to a Spring Boot API through Event Queue

I have successfully set-up Spring Cloud Config Server(**Publisher**) that allows re-translating and combining information from GitHub and MySQL database to Spring Boot Application(**Subscriber**). It is possible thanks to **Spring Cloud Bus** module, that is using **RabbitMQ** server as a message broker. Also, I have implemented RabbitMQ noise generator to imitate real situation, where API is subscribed to multiple topics.

**Configuration repository has been set-up under following link:**

<https://github.com/aleksandrskrivickis/spring_cloud_config_research_config>

**Proof-of concept project files are based in following repository:**

<https://github.com/aleksandrskrivickis/spring_cloud_config_research>

# Limitations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **MySQL** | **MySQL(MariaDB)** | **MongoDB** | **GitHub** |
| Value column character length | 4,294,967,295 | | N/A | |
| Max document size | N/A | | 16 MB | 100 MB+ |
| Serves JSON | No | | Yes, pretty | Yes, data to be defined in \*.yml or \*.properties format |
| Access nested JSON values |  | | | |
| Triggering Web-hook for config update. | By use of scheduler or by implementing webhook in MySQL database management portal. Additional option WebHooks from Zapier. | | | Supports native WebHooks integration |

# Versions used

org.springframework.boot=2.0.5.RELEASE

spring-cloud-starter-bus-amqp=latest

mysql-connector-java=8.0.12

springfox-swagger2=2.8.0

# Conclusions

* It is possible to combine and efficiently serve configuration from both MySQL and GitHub sources using 2.0.5 Spring Boot Starter kit.
* It is possible to trigger configuration update from outer environment with use of WebHooks.
* Configuration can be stored and retrieved in JSON format with a very high data size limitation threshold.
* Theoretically, it is also possible to serve configuration from Local file system, MongoDB, Microsoft SQL or any other DBMS supported by Oracle JDBC.
* **Next step would be to check if it is possible to use the same combination of technologies on Payme custom StarterKit**

# References

* <https://github.com/spring-cloud-incubator/spring-cloud-config-server-mongodb>
* <https://medium.com/@nani2ratna/spring-cloud-config-server-with-jdbc-55de8f7ec86d>
* <https://cloud.spring.io/spring-cloud-static/Edgware.SR2/single/spring-cloud.html#_spring_cloud_config>
* <https://dev.mysql.com/doc/refman/8.0/en/data-types.html>
* <https://start.spring.io/>
* <http://cloud.spring.io/spring-cloud-config/single/spring-cloud-config.html>
* <https://www.baeldung.com/spring-cloud-bus>
* <https://www.rabbitmq.com/tutorials/tutorial-one-spring-amqp.html>