



# Building declaratively configured Keycloak

Václav Muzikář  
Keycloak Maintainer  
Principal Software Engineer @ Red Hat

# Key points

- Declarative configuration? Doesn't it Keycloak support already?!
- Why all the fuss around it?
- Current limitations.
- How can it be improved?
- A good declarative API? How it could look like in Keycloak?
- What is the Keycloak team brewing around this?
- What about Operator CRs?

Building declaratively configured Keycloak

# Doesn't it Keycloak support already?!



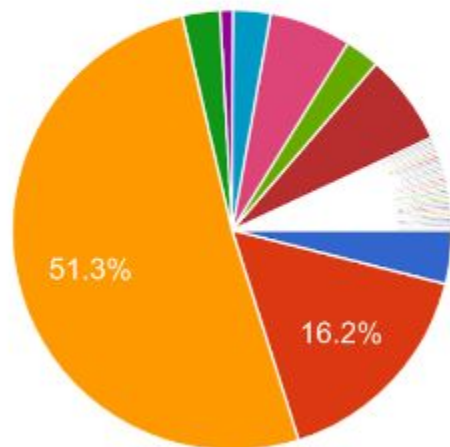
# Declarative configuration: why is it important?

- Configuration as code.
- I want to tell Keycloak my desired configuration state.
  - Not steps how to do it.
- I want to be able to update my desired configuration.
  - Let Keycloak figure out the rest.
- I want my configuration to be portable.
  - I take my config and use it in another Keycloak instance (dev/stage env, ...).
- I want to leverage 3rd party tools to help me manage my configuration.
  - Version control, Kubernetes Custom Resources, templating, ...

# Realm Configuration Management Tools Survey

What's your preferred way to manage Keycloak Realm Configurations?

433 responses



- Keycloak Admin CLI kcadm.sh <https://...>
- Keycloak-Config-CLI <https://github.co...>
- Terraform Provider for Keycloak <https://...>
- Keycloak Ansible <https://github.com/a...>
- Keycloak Pulumi <https://www.pulumi.c...>
- Crossplane Provider for Keycloak <http...>
- Keycloak JSON Import / Export <https://...>
- Keycloak Operator Realm Import via...

▲ 1/5 ▼

<https://www.keycloak.org/2024/09/realm-config-management-tools-survey-results.html>

# What's the issue with that?

- Admin interface – REST API only.
- No real issue with that per se. But...

# A simple Client with a Client Role

```
~/kc/bin (0.926s)
./kcadm.sh create clients -s clientId=my-client
Created new client with id 'b46170d9-3a1d-4e2a-8f04-0dfd2a569284'
```

```
~/kc/bin (0.741s)
./kcadm.sh create clients/b46170d9-3a1d-4e2a-8f04-0dfd2a569284/roles -s name=my-client-role
Created new role with id 'my-client-role'
```

2 requests are needed!?

## Building declaratively configured Keycloak

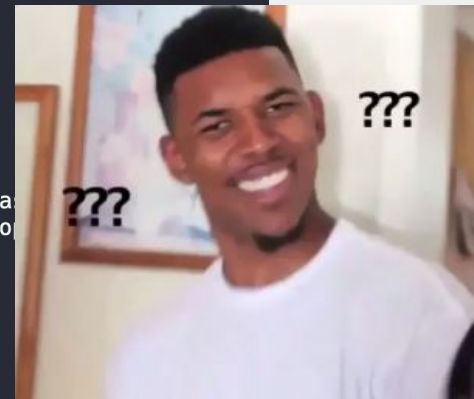
```
~/kc/bin (0.798s)
./kcadm.sh create clients -s clientId=my-client
Created new client with id '6a5ec5f2-e18a-4635-90e5-d3a3adc88f7e'
```

```
~/kc/bin (0.707s)
./kcadm.sh create clients -s clientId=my-client
Client my-client already exists
```



## Building declar

```
~/kc/bin (0.833s)
./kcadm.sh get clients/b46170d9-3a1d-4e2a-8f04-0dfd2a569284
{
  "id" : "b46170d9-3a1d-4e2a-8f04-0dfd2a569284",
  "clientId" : "my-client",
  "surrogateAuthRequired" : false,
  "enabled" : true,
  "alwaysDisplayInConsole" : false,
  "clientAuthenticatorType" : "client-secret",
  "secret" : "MC7tV07sWuhR5fi5UzXhwBNSMRvisAvF",
  "redirectUris" : [ ],
  "webOrigins" : [ ],
  "notBefore" : 0,
  "bearerOnly" : false,
  "consentRequired" : false,
  "standardFlowEnabled" : true,
  "implicitFlowEnabled" : false,
  "directAccessGrantsEnabled" : false,
  "serviceAccountsEnabled" : false,
  "publicClient" : false,
  "frontchannelLogout" : false,
  "protocol" : "openid-connect",
  "attributes" : {
    "realm_client" : "false",
    "client.secret.creation.time" : "1726246769"
  },
  "authenticationFlowBindingOverrides" : { },
  "fullScopeAllowed" : true,
  "nodeReRegistrationTimeout" : -1,
  "defaultClientScopes" : [ "web-origins", "acr", "profile", "roles", "ba
  "optionalClientScopes" : [ "address", "phone", "offline_access", "micro
  "access" : {
    "view" : true,
    "configure" : true,
    "manage" : true
  }
}
```



# Client Protocol Mapper

POST /admin/realms/{realm}/clients/{client-uuid}/protocol-mappers/models

Create a mapper

Parameters

Path Parameters

Name	Description	Default	Pattern
<b>realm</b> <i>required</i>	realm name (not id!)	null	
<b>client-uuid</b> <i>required</i>	id of client (not client-id!)	null	

Body Parameter

Name	Description	Default	Pattern
<b>ProtocolMapperRepresentation</b> <i>optional</i>	<a href="#">ProtocolMapperRepresentation</a>		

Responses

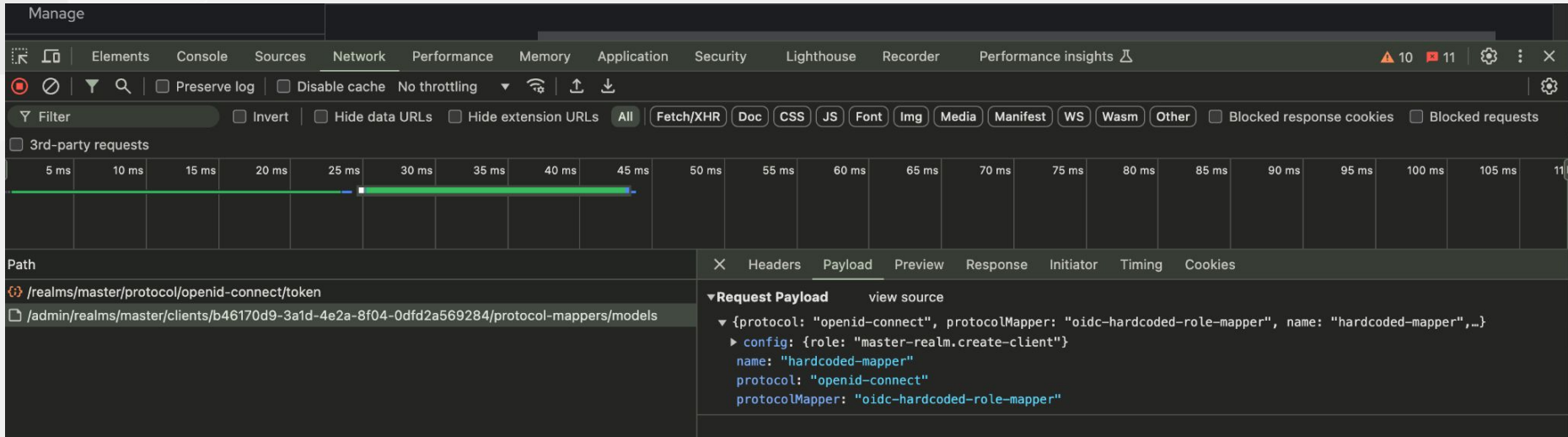
Code	Message	Datatype
200	OK	<<>>

# Client Protocol Mapper

## ProtocolMapperRepresentation

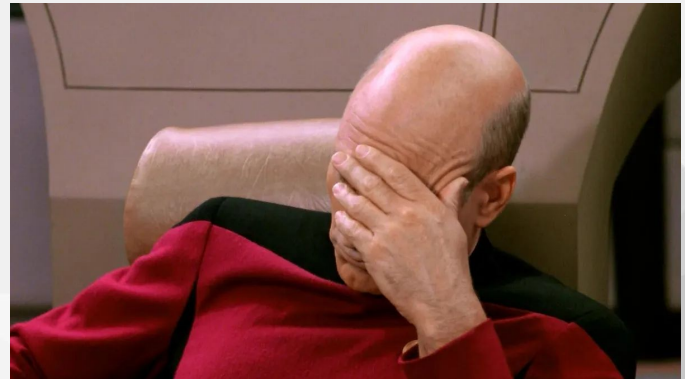
Name	Type	Format
<b>id</b> <i>optional</i>	String	
<b>name</b> <i>optional</i>	String	
<b>protocol</b> <i>optional</i>	String	
<b>protocolMapper</b> <i>optional</i>	String	
<b>consentRequired</b> <i>optional</i>	Boolean	
<b>consentText</b> <i>optional</i>	String	
<b>config</b> <i>optional</i>	Map of <a href="#">[string]</a>	

# Building declaratively configured Keycloak



The screenshot shows the Chrome DevTools Network tab. The top bar includes 'Manage' and various tool panels like Elements, Console, Sources, Network, Performance, Memory, Application, Security, Lighthouse, Recorder, and Performance insights. The Network tab is active, showing a list of requests. The selected request is for the path `/realms/master/protocol/openid-connect/token`. The request payload is expanded, showing a JSON object with the following structure:

```
{protocol: "openid-connect", protocolMapper: "oidc-hardcoded-role-mapper", name: "hardcoded-mapper", ...}
  > config: {role: "master-realm.create-client"}
    name: "hardcoded-mapper"
    protocol: "openid-connect"
    protocolMapper: "oidc-hardcoded-role-mapper"
```



# Using Admin REST API for declarative configuration

- Requires multiple requests to configure a single logical entity.
  - Requests need to be in correct order and reflect results of previous requests like IDs.
  - **The API client needs to understand the API semantics.**
- Poorly documented → weak contract → extremely fragile for external clients.
- Inconsistent.
- No clear distinction between user provided values and default values.
- Not really user friendly structure → hard to maintain.
- Missing validations.
- Resource ownership problem.

But how can we improve that?



Anything beyond this point is  
a sneak peek of WIP and subject to change

# Declarative API in Keycloak

- Backed by a relational DB.
  - File based Map Store was initially also considered.
- Idempotent.
- Stateless from the client perspective.
- Generic, client agnostic.
- User friendly (DTOs, structure, ...).
- Cloud-native friendly.
- Usable at the runtime.
- Clear ownership of declaratively configured resources.



# Files?

- Natural approach: files first.
  - Keycloak loads files and persist configuration in the DB
- When are the files loaded exactly?
  - On startup?
  - Actively watched?
- What if there are more Keycloak instances?
  - What if the files differ between instances?



# REST API? Again?!

- A new API, built from ground up.
- A well-designed REST API ticks all the boxes:
  - Idempotent, stateless, client agnostic.
  - User friendly.
  - Cloud-native friendly.
  - Runtime support.
  - ...
- Something that Keycloak knows very well.
- Current Admin REST API needs a revamp anyway.

# Stateless

- Client should not know anything about current resources state.
  - Should not need to fetch the resource.
- Logical units are a single resource.
  - E.g. Client Roles should be part of Client, not a separate resource.
- Using a single HTTP verb (PUT) for create and update.
  - Ensures idempotency.
  - Overwrites whole resource if exists.

# Access Control

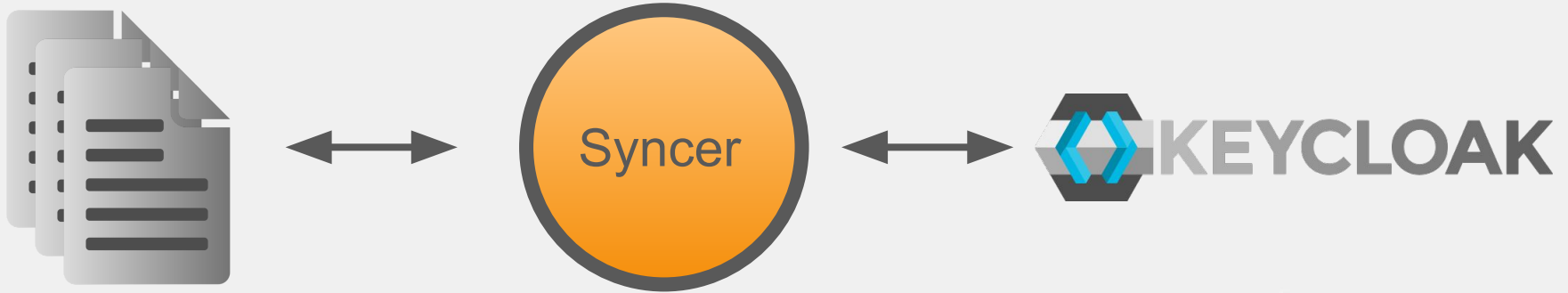
- Declarative config is not the single “config source”.
- Leverage Fine Grain Admin Permissions.
- Declarative config uses a dedicated Service Account for authentication.
- The Service Account is associated with each resource owned by the declarative config.

# Defaults

- Explicit defaults vs. desired state.
- Defaults not always persisted.
- Cleaner GET results.
  - Not strictly needed for declarative config, but good for the overall API design.

# How do files fit into this?

- REST API as an abstraction layer above file system.



# Operator CRs

- Just another syncer.
- CRDs generated from the API representations.
- Initially Client CR only.

## To sum it up...

- A new REST API for native declarative configuration.
- New resource representations.
- Baseline for a new Admin REST API.
- Operator CRs leveraging this.



# What's next

- Prototyping a PoC.
  - Goal is to create a blueprint for an ideal Admin REST API.
- Production ready MVP covering selected use cases for Client.
  - Focused on declarative config and Operator CR.
- Iteratively extending the API.
- Ultimate goal is to allow full migration from the current Admin REST API.

# How to get involved

- <https://github.com/keycloak/keycloak/discussions/33049>
- #keycloak-dev on [slack.cncf.io](https://slack.cncf.io)
- [keycloak-dev@googlegroups.com](mailto:keycloak-dev@googlegroups.com)

Questions?  
Feedback?