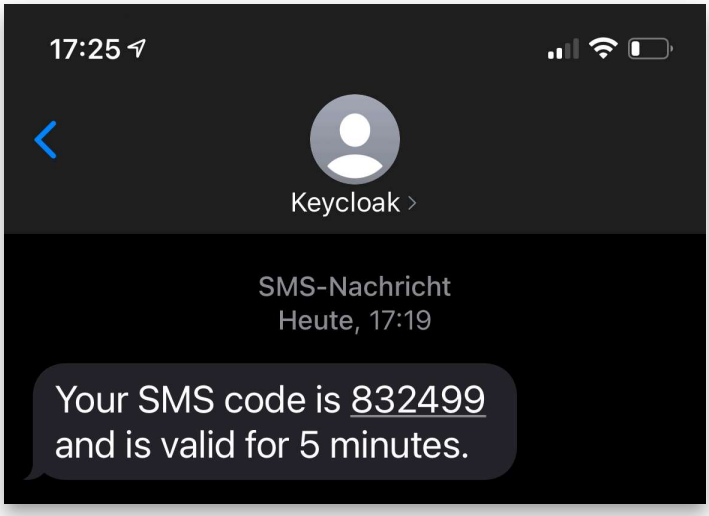
NIKO KÖBLER
CONSULT. DEVELOP. TRAIN.

TWO-FACTOR AUTHENTICATION WITH SMS IN KEYCLOAK

December 23, 2020

Tags: #keycloak #2fa #mfa #authentication #java



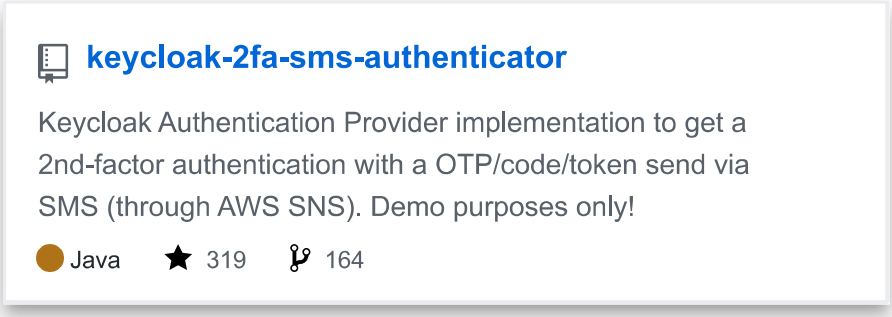
I often get asked by customers or from folks of the community, if there is a built-in option for SMS-based two-/multi-factor-authentication (2FA/MFA) in Keycloak. Unfortunately it is not! Keycloak only ships with a built-in 2FA option for Google Authenticator (and compatible apps). But why is that?

Well, I'm not part of the Red Hat Keycloak team, nor did I asked one of them. But as I do know, is, that there's no "standard" protocol for sending SMS. Yes, of course there are some offical protocols, but nearly every provider I know uses a slightly different one or a completely own (prorietary) API/protocol for this. So, this wouln't make sense to ship something, which isn't useful for most of you, if an extension uses protocol A, but your provider offers only protocol B. That's IMHO the most logical reason for not shipping an SMS-based 2FA authenticator.

But as Keycloak is built up on SPIs and the "Authentication SPI" is one of the most powerful extension points, it's not that hard to implement a 2FA process flow yourself. Using the API and protocol for sending SMS your provider offers to you.

The Authentication SPI is, as already mentioned, very powerful, but at the same time also the most complex SPI in Keycloak, where you can cause harm to your authentication flow. If not implemented properly, you might produce attack vectors to your system and thus make it insecure. So, I really recommend to read the server developer documentation in detail!

For your convenience, I provide a demo implementation of the Authentication SPI on GitHub:



This 2FA authenticator uses AWS SNS to send SMS, as AWS is my default cloud provider and SNS makes it pretty easy to send SNS. For testing reasons and not to spend too much money for useless SMS during development and tests, I implemented also a *"simulation mode"*, where no SMS is actually sent, but the code will be printed to the log output. You can then just copy&paste it from there to the input form.

The actual Authenticator consists of two classes - the factory and the provider itself, as typically for Keycloak SPIs. The `SmsAuthenticatorFactory.java` provides the optional configuration of the authenticator, which is then used in the provider class. The config properties should be self-explaining, as they have a help-text provided.

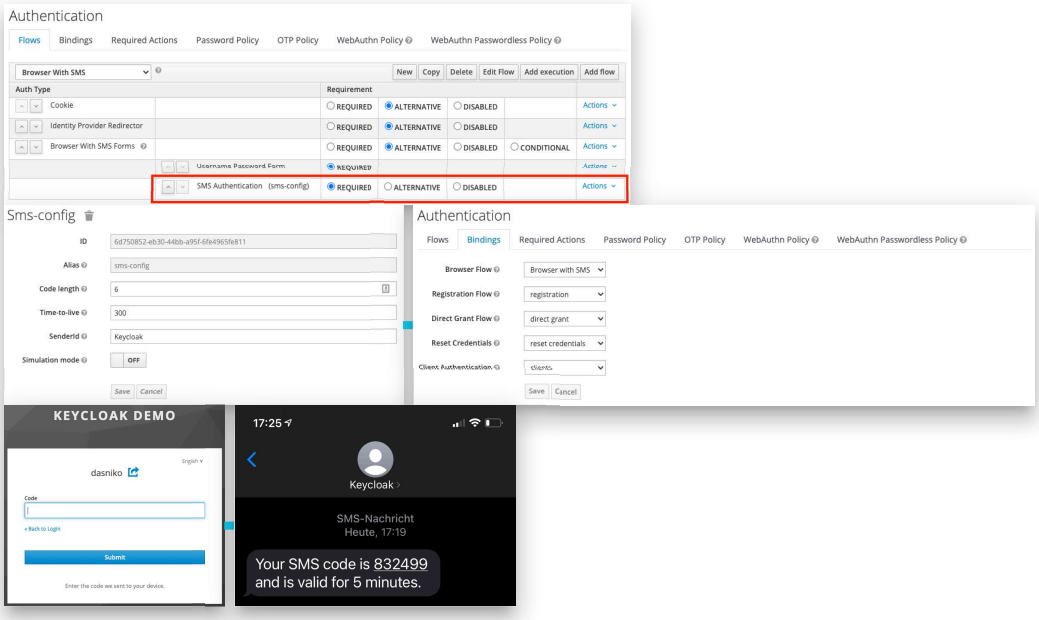
The `SmsAuthenticator.java` is the class with all your (business) logic. As described in the docs, the `authenticate()` method is being called if there is no already started authentication session of this authenticator. As you can see from the code, the current user must have configured an attribute called `mobile_number`, where the SMS will be sent to. With the info from the config (from the factory class, see above), the code is created in length and validity. Defaults are 6 digits in length and 300 seconds (5 mins) validity. The generated code is then stored in the authentication-session-notes. The authentication session is valid during the whole authentication process.

The code will then be sent through your SMS provider (AWS SNS in my case), using the `SmsServiceFactory` class. This factory returns an implementation of the [`SmsService`] instance, which can be a dummy implementation for the simulation mode, or the [`AwsSmsService`]. You can extend it with a provider of your choice, if you want.

The text of the SMS message is taken from the included theme resources. There's a `messages` folder for all the localized messages and a `templates` folder for the form, which will be displayed right after the SMS has been sent. You can adjust/overwrite the messages and template in your own theme, in the common way of Keycloak themeing. There's no special way needed.

After receiving the SMS, entering it into the form and submitting it, the `action()` method of the `SmsAuthenticator` class will be called. The entered code will be compared with the stored code in the authentication session notes and the validity will be checked. If there's too much time gone between creating the code and validating it, the action will fail. If everythin is ok, the user will be authenticated successfully (for this authenticator step).

To be able to use the `2fa-sms-authenticator` after you deployed the JAR profile to the the `/deployments` directory, you'll have to create and configure a Keycloak authentication flow in your realm and use it in a binding. The following images will show you, how this might look like (click to enlarge):



1. Create (copy) a new flow for browser-based authentication and adjust it to your needs. Add an execution for the "SMS

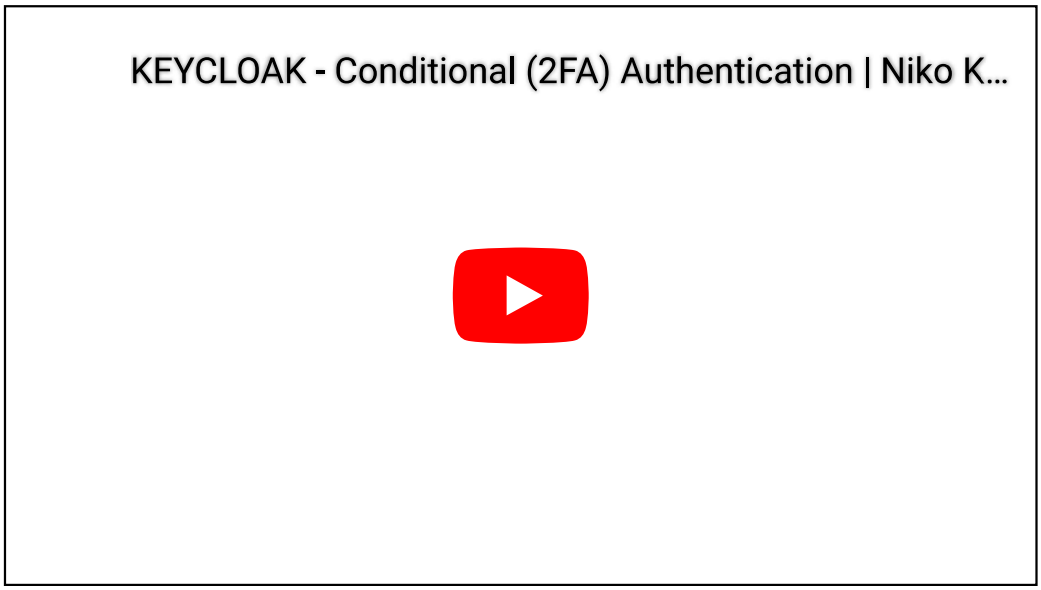
Authentication".
2. Configure the "SMS Authentication" execution step with the values which fits best your needs.
3. Set the new created flow as "Browser Flow" in the Authentication / Bindings tab in the admin console of your realm.
4. The default form to enter the code sent by SMS
5. Example SMS received with configured sender ID "Keycloak"

Have fun with SMS-based 2-factor authentication in Keycloak!

IMPORTANT

Keep in mind, that the 2-fa-sms-authenticator provider is for demo purposes only. There's no warranty! If you want to use it for production purposes, use is at your own risk! You'll most probably have to extend the code with further checks and validations!

KEYCLOAK - 2FA with SMS based OTP text messages...



« Keycloak Session Restrictor - or: HIGHLANDER mode

Keycloak & React.JS & Router Integration How To »

Privacy Policy Terms & Conditions Imprint