**SMS based 2FA**

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](https://github.com/dasniko/keycloak-2fa-sms-authenticator/blob/master/src/main/java/dasniko/keycloak/authenticator/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.

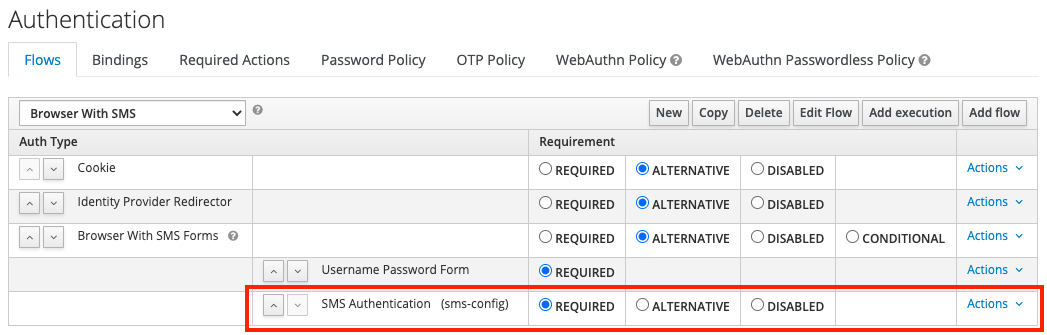
The [SmsAuthenticator.java](https://github.com/dasniko/keycloak-2fa-sms-authenticator/blob/master/src/main/java/dasniko/keycloak/authenticator/SmsAuthenticator.java) is the class with all your (business) logic. As described in the docs, the authenticate() method is being called if there is no already 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.

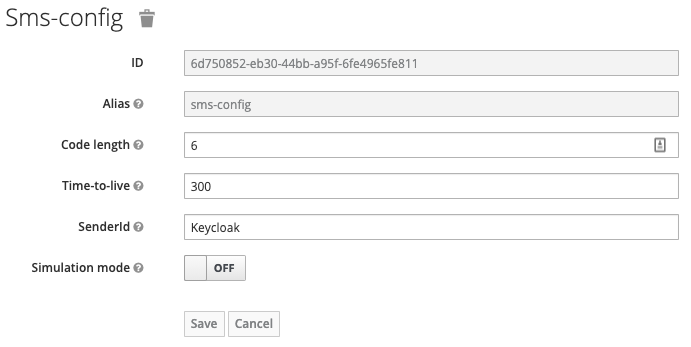
The code will then be sent through your SMS provider (AWS SNS in my case), using the [SmsServiceFactory](https://github.com/dasniko/keycloak-2fa-sms-authenticator/tree/master/src/main/java/dasniko/keycloak/authenticator/gateway) class. This factory returns an implementation of the [SmsService] instance, 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.

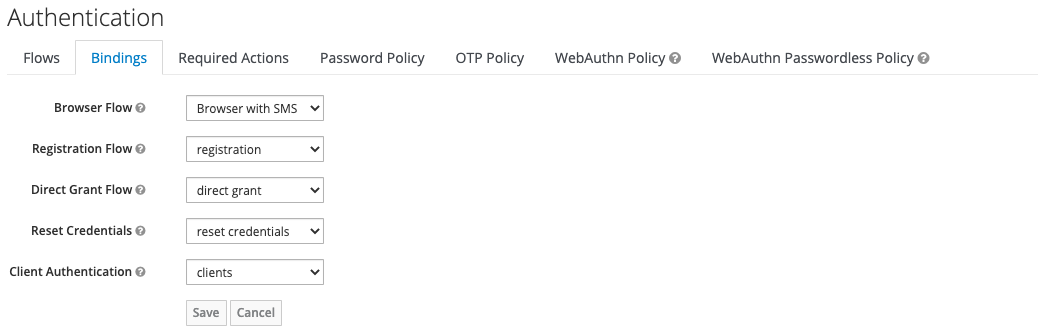
The text of the SMS message is taken from the [included theme resources](https://github.com/dasniko/keycloak-2fa-sms-authenticator/tree/master/src/main/resources/theme-resources). There’s a messages folder for all the localized messages and a templates folder for the form, which will be displayed right 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”