CONCISO.

Unlocking the Power of Keycloak

Best Practices for Extension Development

Keycloak DevDay 2024 / Frankfurt, Germany

22.02.2024

sven-torben.janus@conciso.de

@sventorben
@mas.to
.bsky.social

Sven-Torben Janus

Partner, Principal Software Architect

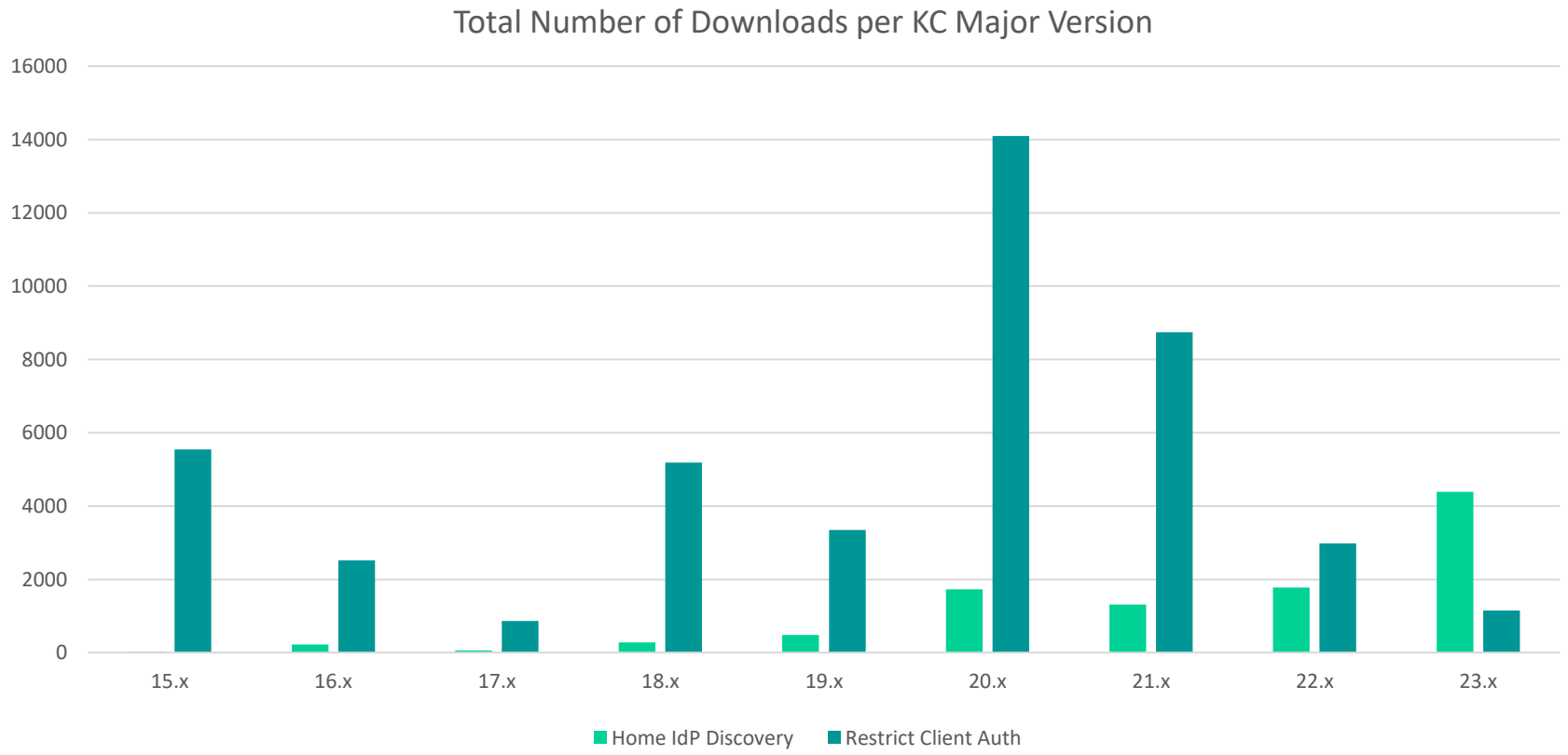
Keycloak Extensions

Home IdP Discovery

Restrict Client Auth

Adoption

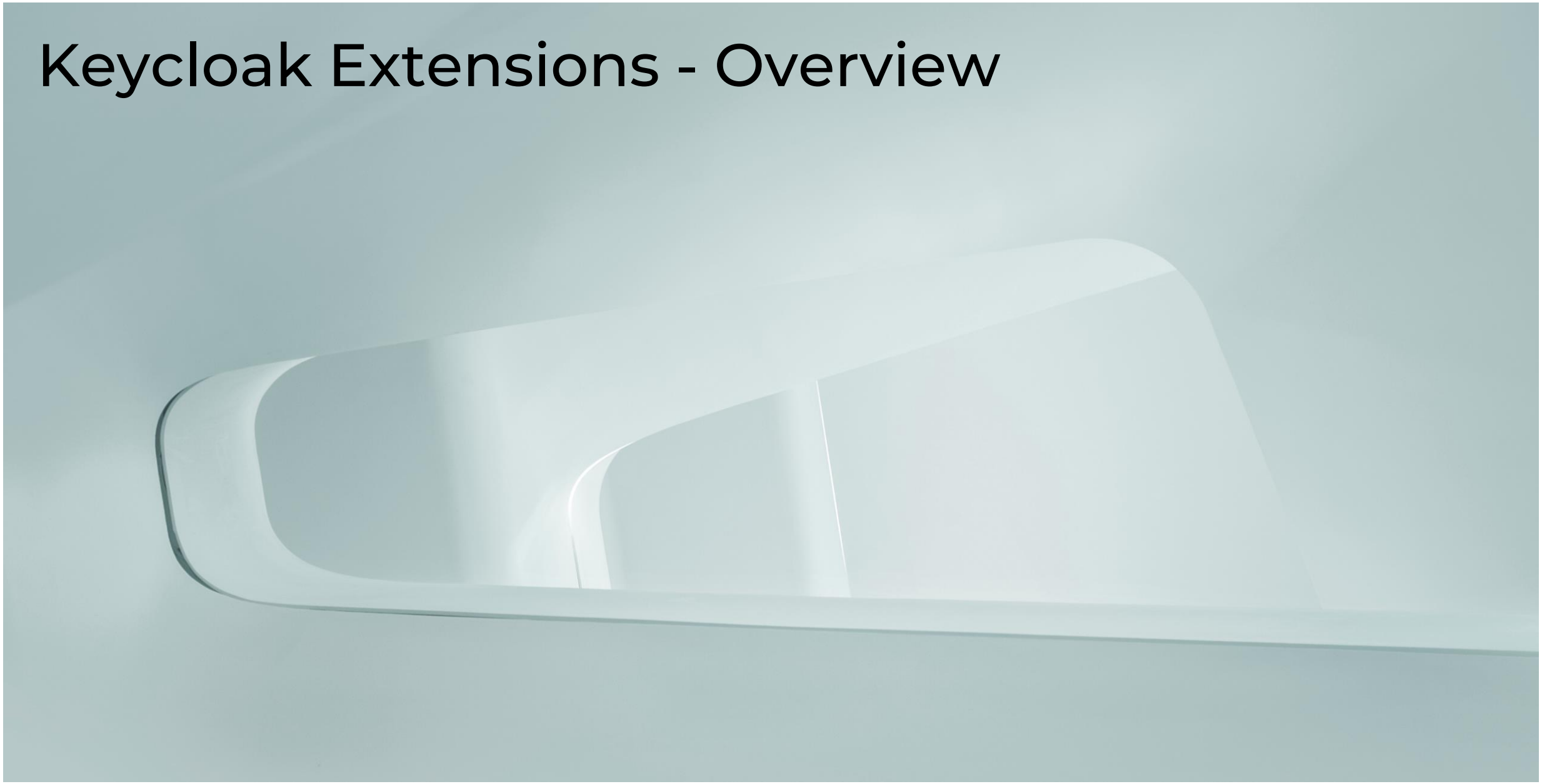
- SovereignCloudStack
- Phase Two
- Several larger enterprises



What CONCISO. builds for customers

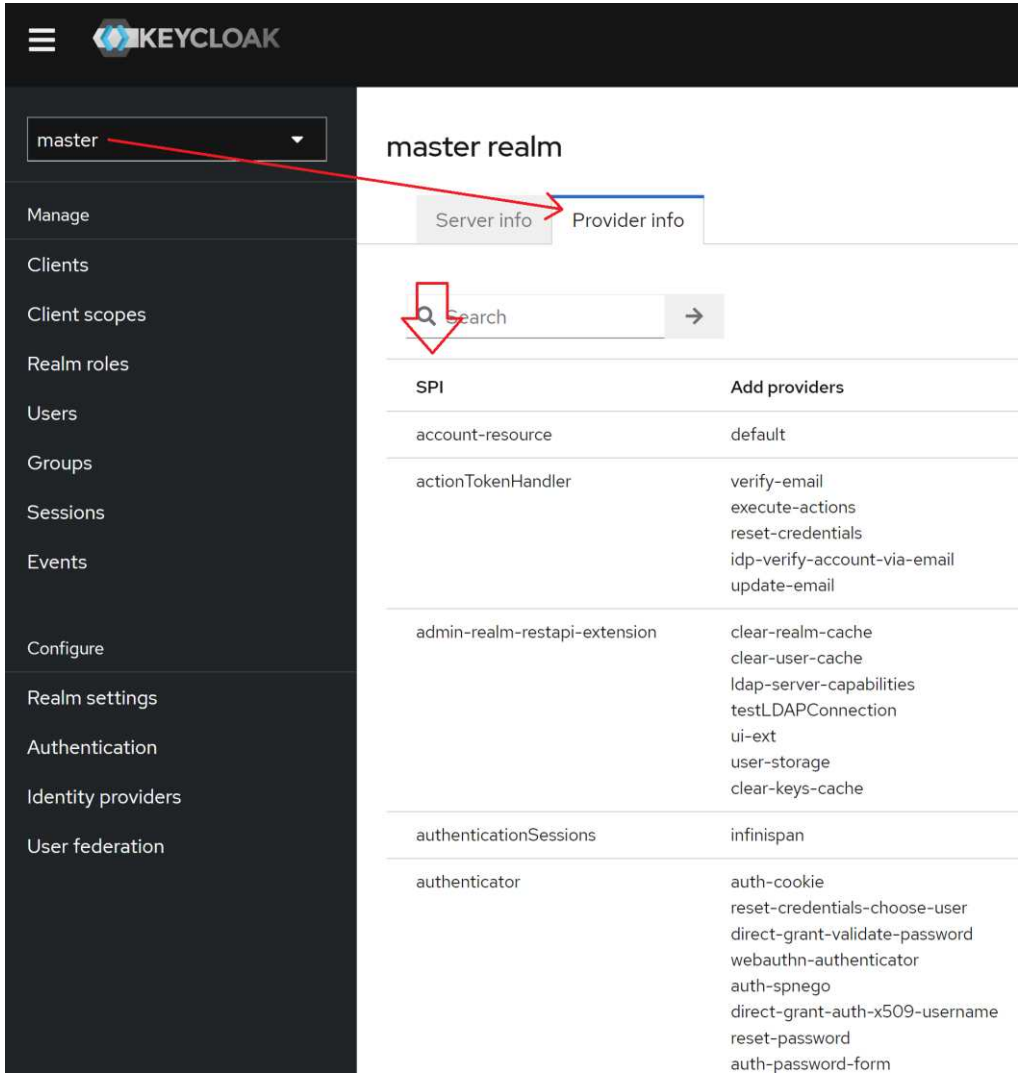
- Custom Themes and Templates
- Authenticators
- User Storage Providers
- Protocol / IdP / LDAP Mappers
- Client Policies (Conditions and Executors)
- Required Actions
- Event Listeners
- Custom REST APIs
- Custom Domain Models / Entities

Keycloak Extensions - Overview

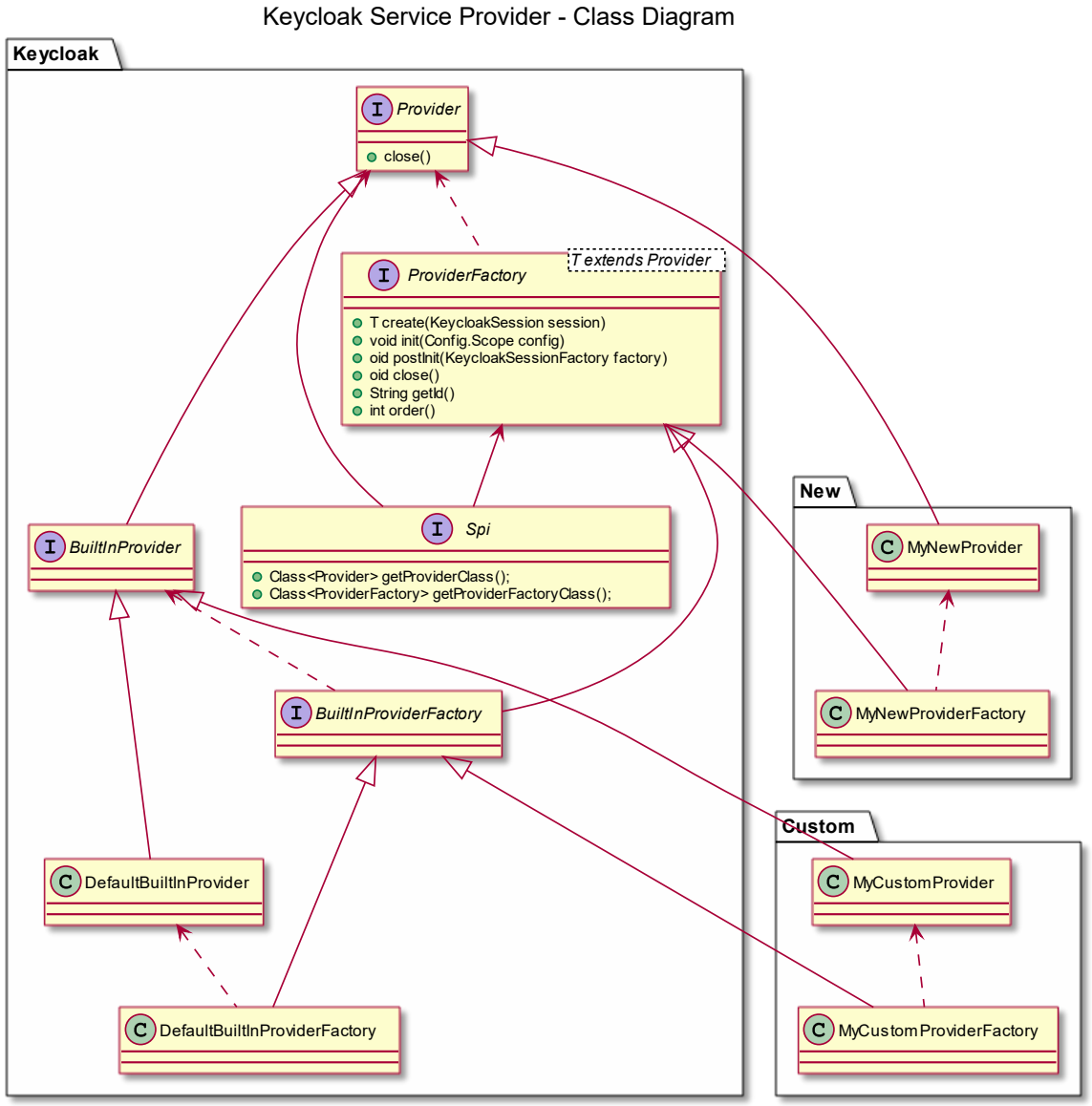


What are Keycloak Extensions?

Keycloak extensions are designed to extend the core functionalities of Keycloak to meet specific, custom requirements that aren't covered by the default installation.



Some UML?!



Additional interfaces

`ConfiguredProvider`

Providers can be configured via Keycloak UI

`ServerInfoAwareProviderFactory`

ProviderFactories can display additional information in provider info page

How to write and activate an extension?

Code the provider factory (Java class)

Code the provider (Java class)

Register the provider in the services manifest (Java SE Service Provider)

Package a JAR file

Copy JAR file to providers folder of your Keycloak instance(s) / image

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build

ARG HIDPD_VERSION=23.0.0

RUN mkdir -p /mnt/rootfs /extensions
RUN dnf install --installroot /mnt/rootfs curl --releasever 9 \
    --setopt install_weak_deps=false --nodocs -y && \
    dnf --installroot /mnt/rootfs clean all && \
    rpm --root /mnt/rootfs -e --nodeps setup

RUN curl -L -s -o /extensions/keycloak-home-idp-discovery.jar \
     https://github.com/sventorben/keycloak-home-idp-
discovery/releases/download/v$HIDPD_VERSION/keycloak-home-idp-
discovery.jar

FROM quay.io/keycloak/keycloak:23.0.6

COPY --from=ubi-micro-build --chown=keycloak:keycloak /extensions/
/opt/keycloak/providers/
```

Extension versioning

The most asked question?

Is version A.B.C of this extension compatible with Keycloak version X.Y.Z?

Tip: Align the major version of your extension with Keycloak‘s major version.

Extension version: **23**.0.2
Compatible KC versions: **23**.0.0, **23**.0.1, **23**.0.2, **23**.0.3 …

Second most asked question?

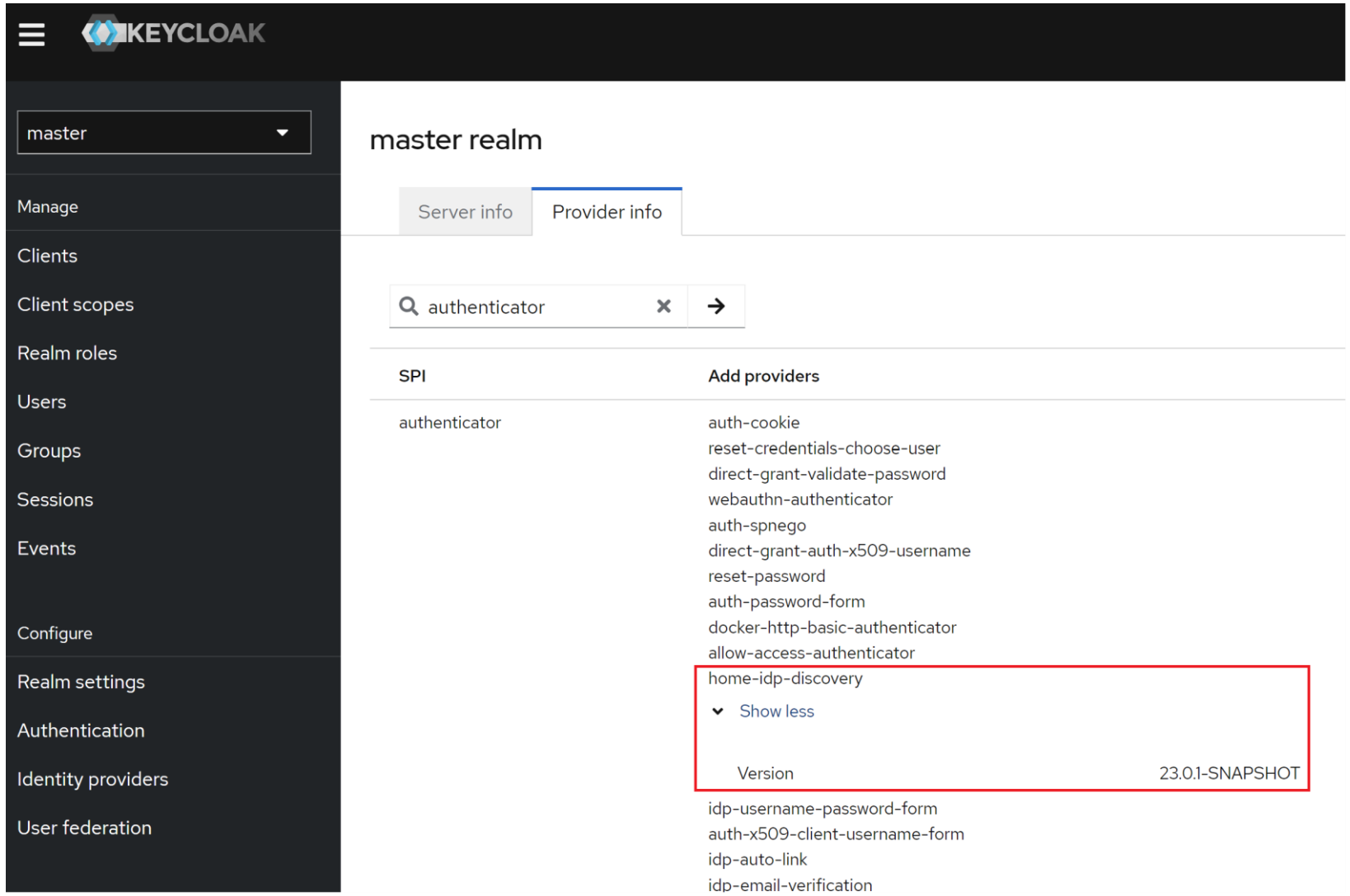
Them: I have upgraded from Keycloak version X to **Y**. Your extension is not working anymore.

Me: Have you upgraded to latest version of my extension: **Y**.a.b?

Them: Yes, of course …

Turns out they didn‘t!

Tip: Make your extension‘s version visible on Keycloak‘s UI and API!



Tip: Make your extension‘s version accesible via Keycloak‘s UI and API!

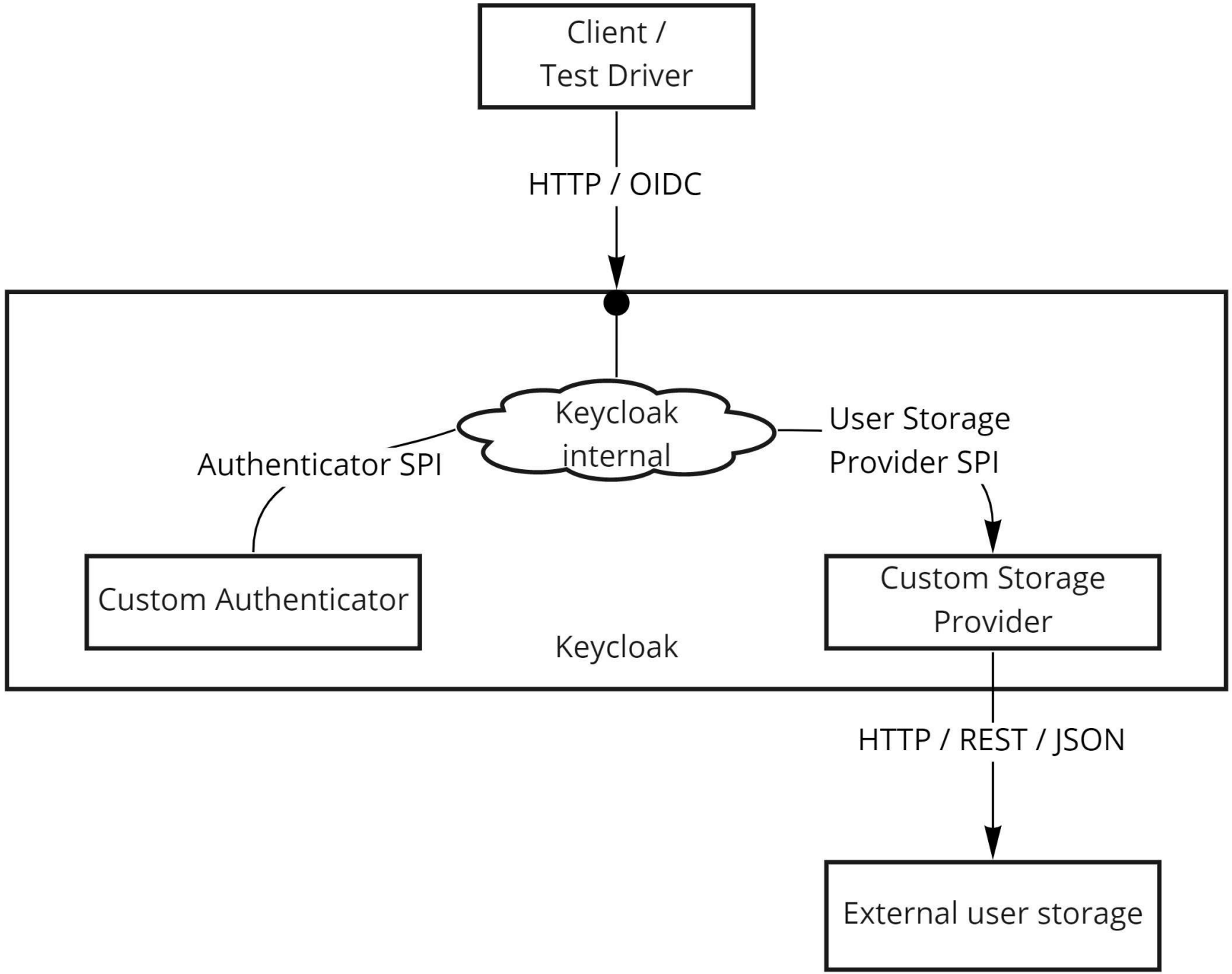
Implement the **ServiceInfoAwareProvider** interface!

```java
@Override
public Map<String, String> getOperationalInfo() {
    String version = getClass().getPackage().getImplementationVersion();
    if (version == null) {
        version = "dev-snapshot";
    }
    return Map.of("Version", version);
}
```

You may also want to add provider factory configuration values.

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.3.0</version>
    <configuration>
        <archive>
            <manifest>
                <addDefaultImplementationEntries>
                    true
                </addDefaultImplementationEntries>
            </manifest>
        </archive>
    </configuration>
</plugin>
```

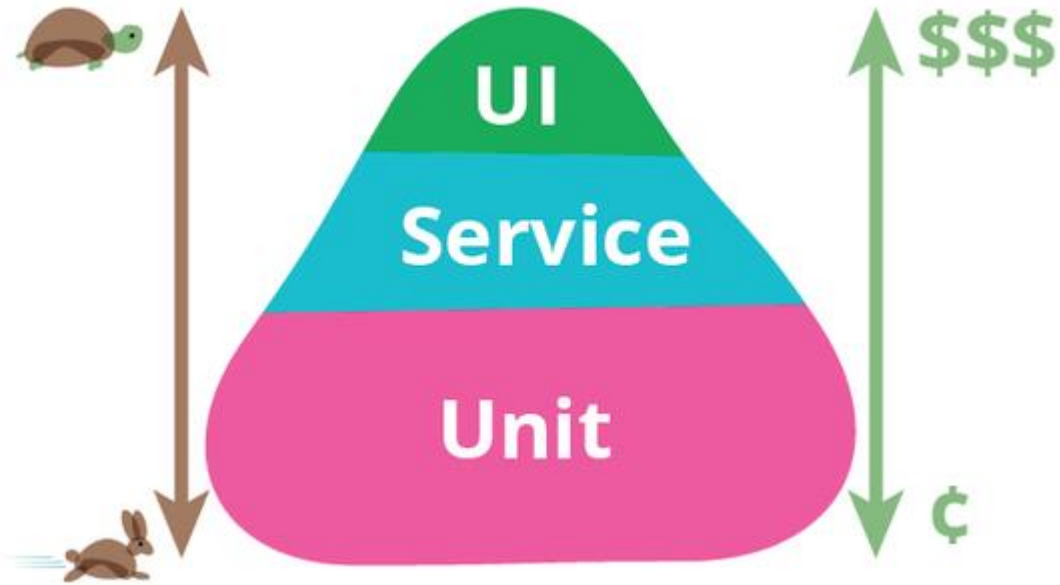
The Safety Net


Client /
Test Driver
HTTP / OIDC
Keycloak
internal
Authenticator SPI
User Storage
Provider SPI
Custom Authenticator
Custom Storage
Provider
Keycloak
HTTP / REST / JSON
External user storage

KeycloakSession – the main integration point

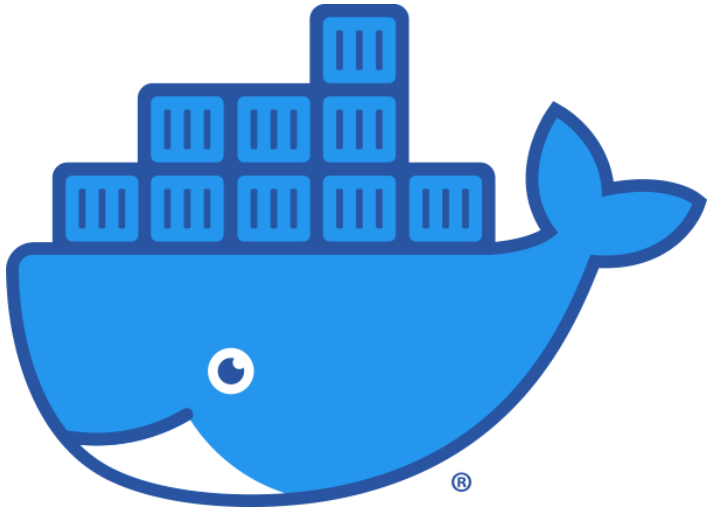
```
public interface KeycloakSession extends AutoCloseable {
    KeycloakContext getContext();
    KeycloakTransactionManager getTransactionManager();
    KeycloakSessionFactory getKeycloakSessionFactory();
    RealmProvider realms();
    ClientProvider clients();
    ClientScopeProvider clientScopes();
    GroupProvider groups();
    RoleProvider roles();
    UserSessionProvider sessions();
    UserLoginFailureProvider loginFailures();
    AuthenticationSessionProvider authenticationSessions();
    UserProvider users();
    KeyManager keys();
    ThemeManager theme();
    TokenManager tokens();
    ClientPolicyManager clientPolicy();
    <T extends Provider> T getProvider(Class<T> clazz);
    //...
}
```

Testpyramid

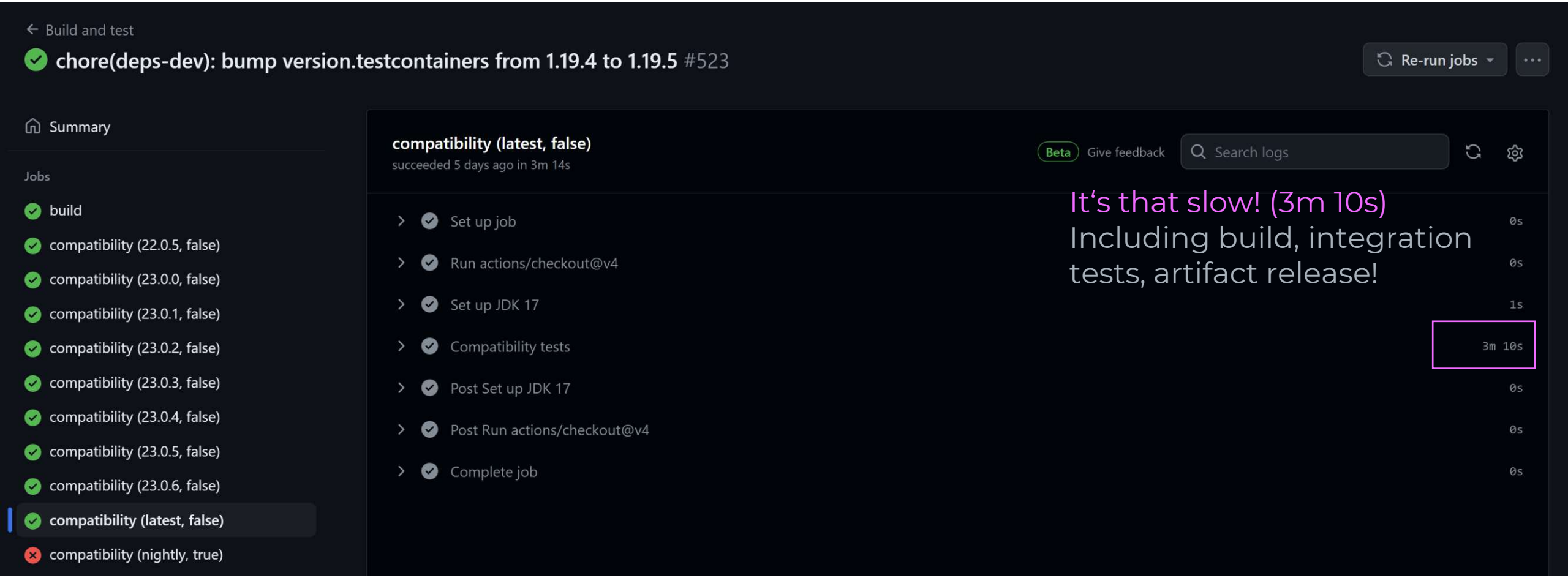


Source: Fowler - https://martinfowler.com/bliki/TestPyramid.html

Source: Wikipedia / Lawrence Livermore National Laboratory



Speed


Build and test
chore(deps-dev): bump version.testcontainers from 1.19.4 to 1.19.5 #523
Re-run jobs
Summary
Jobs
build
compatibility (22.0.5, false)
compatibility (23.0.0, false)
compatibility (23.0.1, false)
compatibility (23.0.2, false)
compatibility (23.0.3, false)
compatibility (23.0.4, false)
compatibility (23.0.5, false)
compatibility (23.0.6, false)
compatibility (latest, false)
compatibility (nightly, true)
compatibility (latest, false)
succeeded 5 days ago in 3m 14s
Beta
Give feedback
Search logs
Set up job 0s
Run actions/checkout@v4 0s
Set up JDK 17 1s
Compatibility tests 3m 10s
Post Set up JDK 17 0s
Post Run actions/checkout@v4 0s
Complete job 0s
It's that slow! (3m 10s)
Including build, integration tests, artifact release!

Testcontainers to the rescue

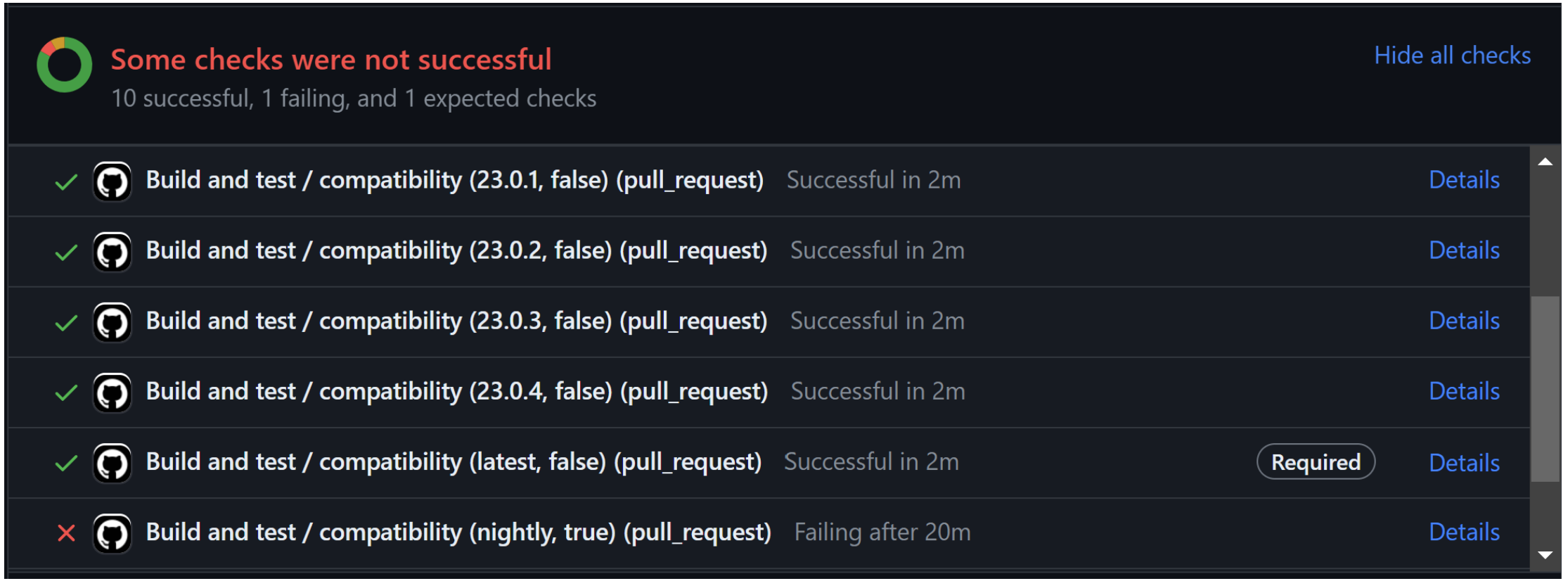
“Testcontainers is a testing library that provides easy and lightweight APIs for bootstrapping integration tests with real services wrapped in Docker containers.

Using Testcontainers, you can write tests talking to the same type of services you use in production without mocks or in-memory services.”

Keycloak Testcontainer by Niko Köbler

```
KeycloakContainer kc = new KeycloakContainer(fullImage)
    .withImagePullPolicy(pullPolicy);

if (useLibrary) {
    kc = kc.withProviderLibsFrom(List.of(new File("target/keycloak-restrict-client-auth.jar")));
} else {
    kc = kc.withProviderClassesFrom("target/classes");
}

kc.getKeycloakAdminClient()
    .realm("test-realm")
    .users()
    .get("139020a3-4459-43b1-a92f-d90e5cf093a1")
    .logout();
```

Tip: Test all changes (features, fixes, dependency updates, ...) with supported Keycloak versions



Tip: Test all changes, new features, fixes with supported Keycloak versions

```yaml
compatibility:
  runs-on: ubuntu-latest
  strategy:
    fail-fast: false
    matrix:
      keycloak_version: [ 21.0.2, 21.1.2, 22.0.5, 23.0.0, 23.0.1, 23.0.2, 23.0.3, 23.0.4, 23.0.5, latest ]
      experimental: [false]
      include:
          - keycloak_version: nightly
            experimental: true
  continue-on-error: ${{ matrix.experimental }}
  steps:
    ...
    - name: Compatibility tests
      run: mvn -B -U clean test-compile failsafe:integration-test failsafe:verify --file pom.xml \
          -Dkeycloak.version=${{ matrix.keycloak_version }}
```

Tip: Test all new Keycloak versions with all extension versions

Extension \ Keycloak	21.0.2	21.1.2	22.0.5	23.0.6	latest	nightly
21.3.0	ok	ok	fail	fail	fail	fail
22.0.0	fail	fail	ok	ok	ok	fail
22.1.0	fail	fail	ok	ok	ok	fail
23.0.0	fail	fail	ok	ok	ok	fail

```yaml
compatibility:
  runs-on: ubuntu-latest
  strategy:
    fail-fast: false
    matrix:
      keycloak_version: [ 21.0.2, 21.1.2, 22.0.5, 23.0.6, latest, nightly ]
      extension_version: [ 21.3.0, 22.0.0, 22.1.0, 23.0.0 ]
  steps:
    - uses: actions/checkout@v4
      with:
          ref: v${{ matrix.extension_version }}
    ...
    - name: Download extension
      run: curl -L -o target/keycloak-restrict-client-auth.jar https://github.com/sventorben/keycloak-restrict-client-auth/releases/download/v${{ matrix.extension_version }}/keycloak-restrict-client-auth.jar
    - name: Compatibility tests
      run: mvn -B -U clean test-compile failsafe:integration-test failsafe:verify --file pom.xml \
           -Dkeycloak.version=${{ matrix.keycloak_version }} -DuseLibrary=true
```

How to debug it?

```yaml
version: '3'
services:
  keycloak:
    container_name: keycloak
    hostname: keycloak
    image: quay.io/keycloak/keycloak:23.0.6
    environment:
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
      DEBUG_PORT: '*:8787'
      DEBUG: 'true'
    command: ['start-dev']
    ports:
      - 8080:8080
      - 8787:8787
```

Equivalent command line arguments for remote VM:

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:8787
```

Stability and backwards compatibility

Ensure Stability by Encapsulation

Wrap configuration models

```
final class HomeIdpDiscoveryAuthenticator implements Authenticator {

    @Override
    public void authenticate(AuthenticationFlowContext authenticationFlowContext) {

        var config = new HomeIdpDiscoveryConfig(authenticationFlowContext.getAuthenticatorConfig());

        if (config.shouldBypassLoginPage()) {
            // ...
        }
```

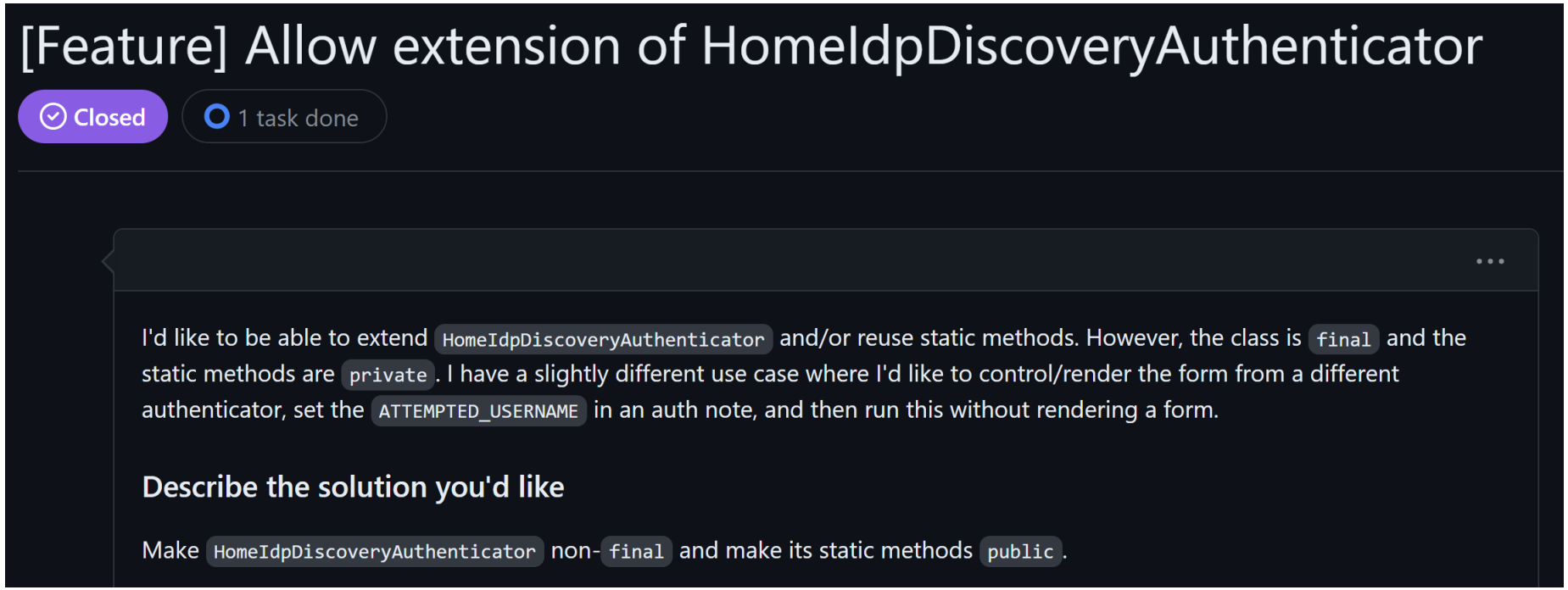
Ensure Stability by Encapsulation

Wrap context-related code

```
final class HomeIdpDiscoveryAuthenticator implements Authenticator {

    @Override
    public void authenticate(AuthenticationFlowContext authenticationFlowContext) {
        HomeIdpAuthenticationFlowContext context = new HomeIdpAuthenticationFlowContext(authenticationFlowContext);
        // ...
        context.rememberMe().remember(username);
```

```
void remember(String username) {
    String rememberMe = authenticationFlowContext.getAuthenticationSession().getAuthNote(Details.REMEMBER_ME);
    RealmModel realm = authenticationFlowContext.getRealm();
    boolean remember = realm.isRememberMe() && "true".equalsIgnoreCase(rememberMe);
    if (remember) {
        AuthenticationManager.createRememberMeCookie(username, authenticationFlowContext.getUriInfo(), authenticationFlowContext.getSession());
    } else {
        AuthenticationManager.expireRememberMeCookie(realm, authenticationFlowContext.getUriInfo(), authenticationFlowContext.getSession());
    }
}
```

Explicitly design for extendability or prohibit it



➔ When you follow such requests, classes become part of your public API

➔ You are missing a crucial feature discussion with your users

Explicitly design for extendability or prohibit it

Prohibit subclassing aka „mark all classes final“

Always reduce accessibility as much as possible

- Private fields and methods
- Private constants
- Limit public methods, package protected mostly works!

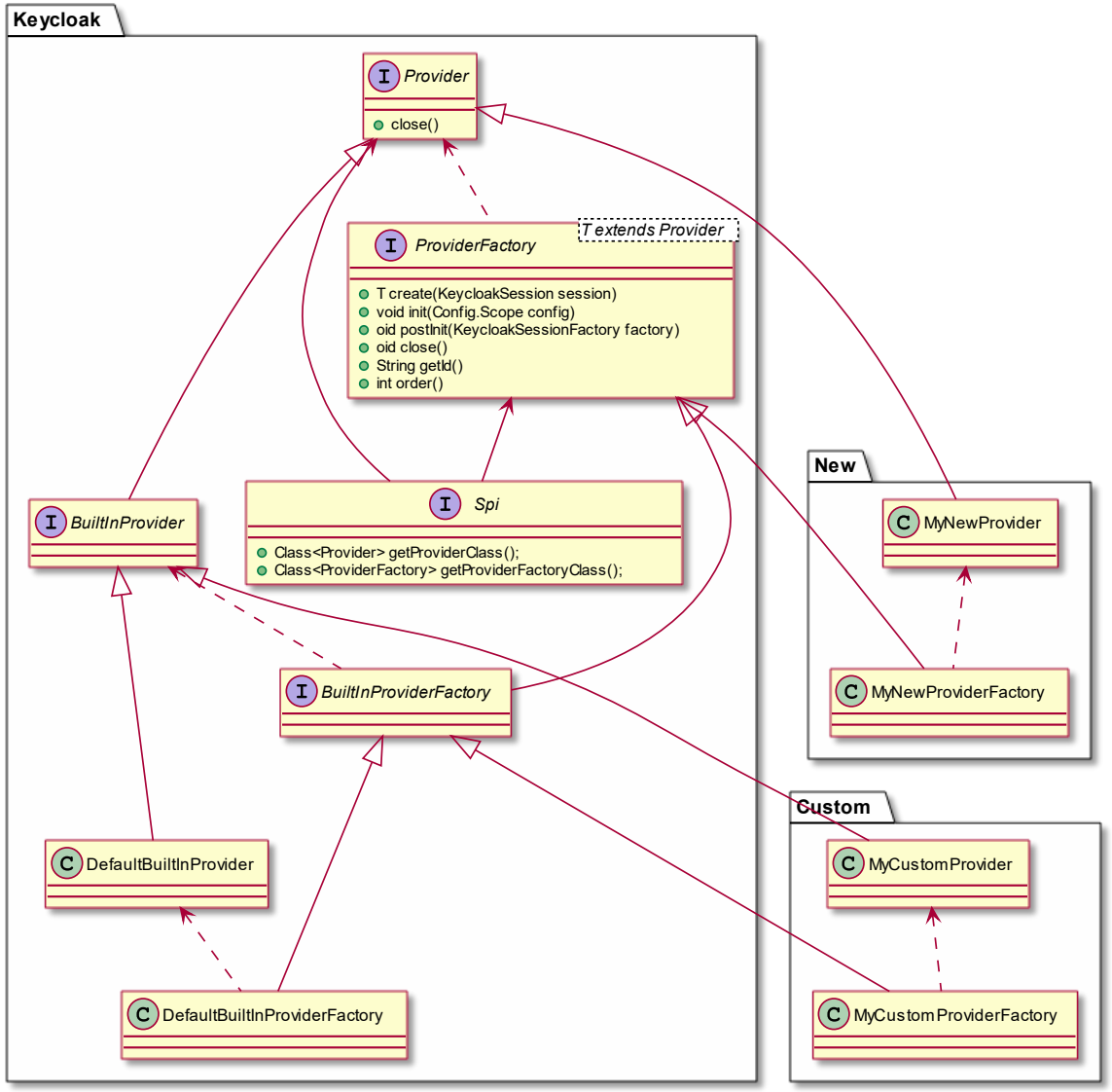
If you need extendability:

- Judiciously choose protected methods as hooks
- Use Service Provider Interfaces (SPI) to provide well defined extension points

Extensions points by SPIs

Extension points by SPIs

Keycloak Service Provider - Class Diagram

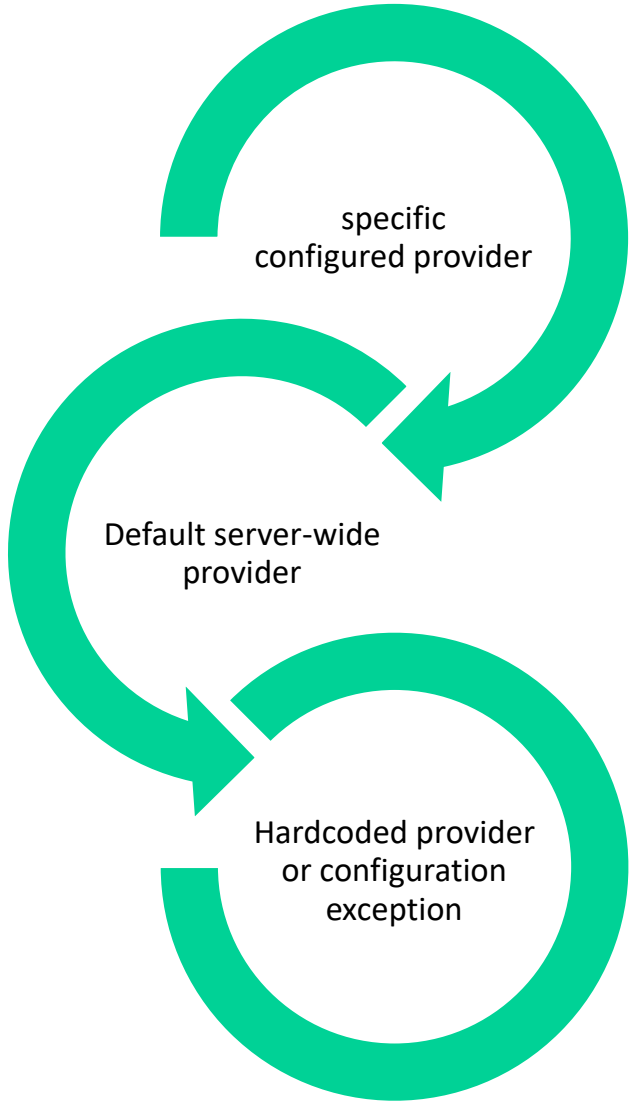



```java
public interface AccessProvider extends Provider {
    boolean isRestricted(ClientModel client);

    boolean isPermitted(ClientModel client, UserModel user);

    void enableFor(ClientModel client);
}

public interface AccessProviderFactory extends ProviderFactory<AccessProvider> {
}
```

```java
final class ClientRoleBasedAccessProviderFactory implements AccessProviderFactory {

    static final String PROVIDER_ID = "client-role";

    @Override
    public AccessProvider create(KeycloakSession session) {
        return new ClientRoleBasedAccessProvider();
    }

    @Override
    public String getId() {
        return PROVIDER_ID;
    }

    // ...
}

final class ClientRoleBasedAccessProvider implements AccessProvider {
    // ...
}
```

Lookup custom provider

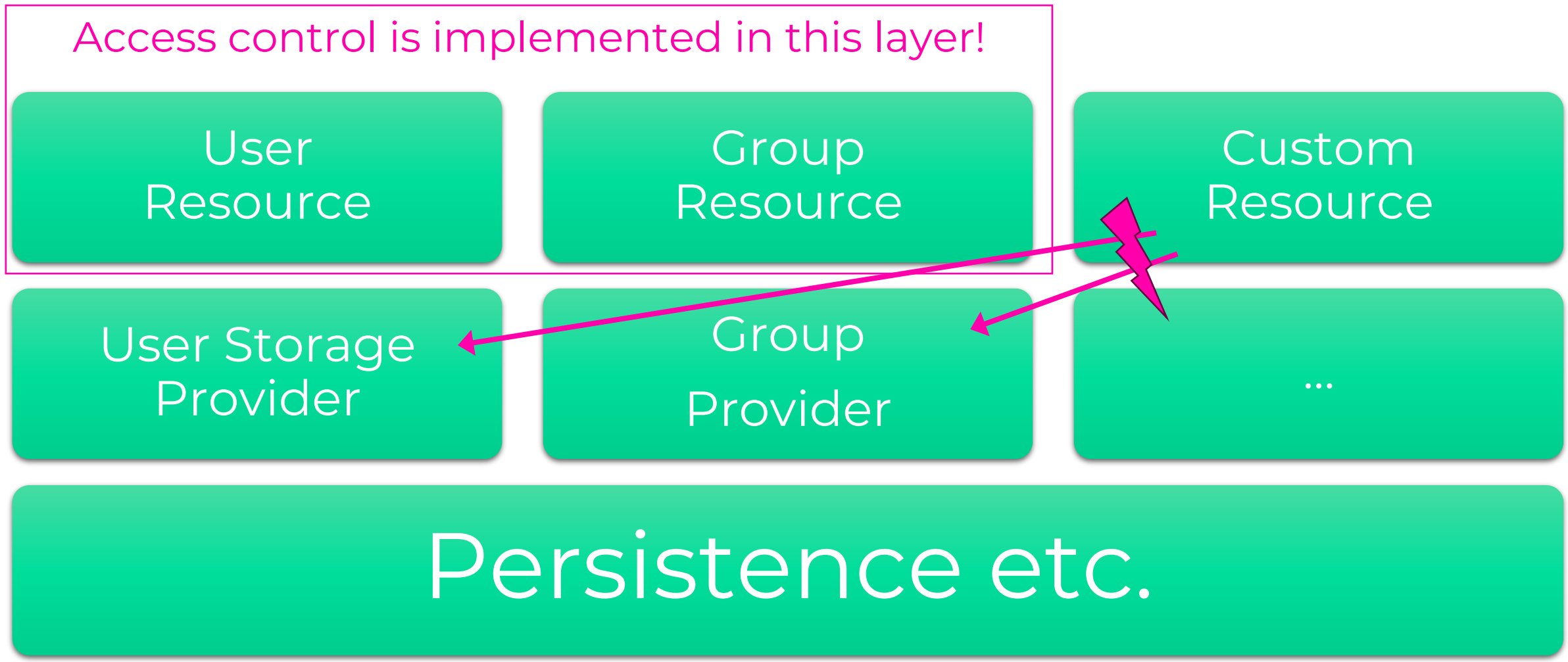
```
private AccessProvider getAccessProvider(KeycloakSession keycloakSession, RestrictClientAuthConfig config) {
    final String accessProviderId = config.getAccessProviderId();

    // specific provider configured in authenticator config
    final AccessProvider accessProvider = keycloakSession.getProvider(AccessProvider.class, accessProviderId);
    if (accessProvider != null) {
        return accessProvider;
    }

    // default server-wide provider
    final AccessProvider defaultProvider = keycloakSession.getProvider(AccessProvider.class);
    if (defaultProvider != null) {
        return defaultProvider;
    }

    // fallback to hardcoded provider - could also throw exception here
    return keycloakSession.getProvider(AccessProvider.class, ClientRoleBasedAccessProviderFactory.PROVIDER_ID);
}
```



Keep backwards compatiblity in mind:
What was the behaviour before the concept of „access provider“ existed?

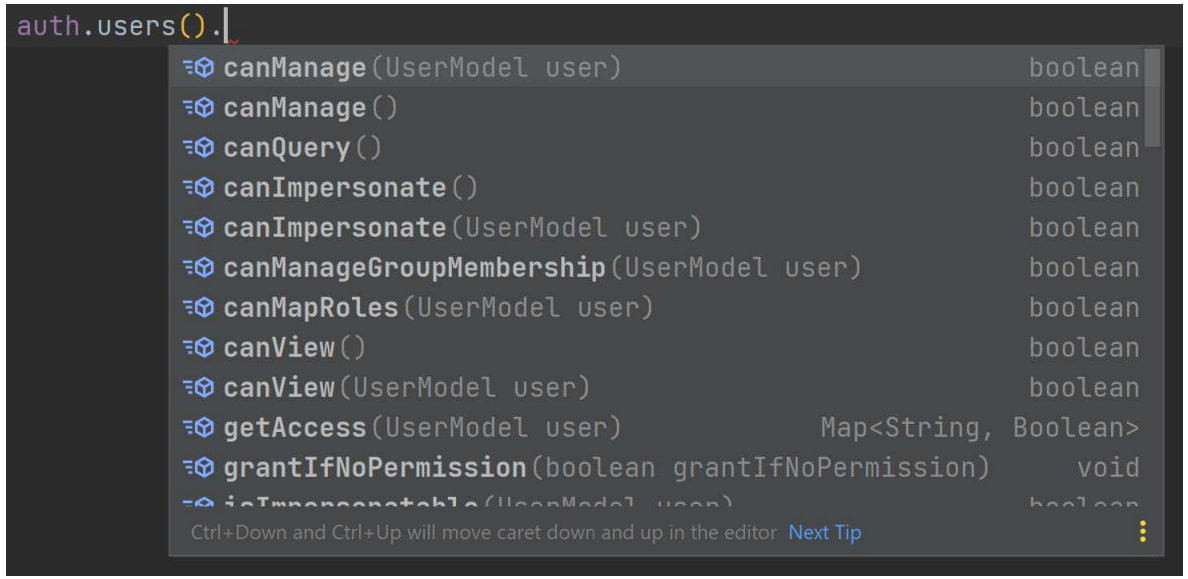
REST APIs

Access control – it‘s not a given!



Access control – it‘s not a given!

```java
public final class CustomRootResourceProvider implements RealmResourceProvider {

    private final KeycloakSession session;

    CustomRootResourceProvider(KeycloakSession session) {
        this.session = session;
    }

    @Override
    public Object getResource() {
        RealmModel realm = session.getContext().getRealm();
        CustomRootResource resource = new CustomRootResource(realm, session);
        ResteasyProviderFactory.getInstance().injectProperties(resource);
        resource.init();
        return resource;
    }

}
```



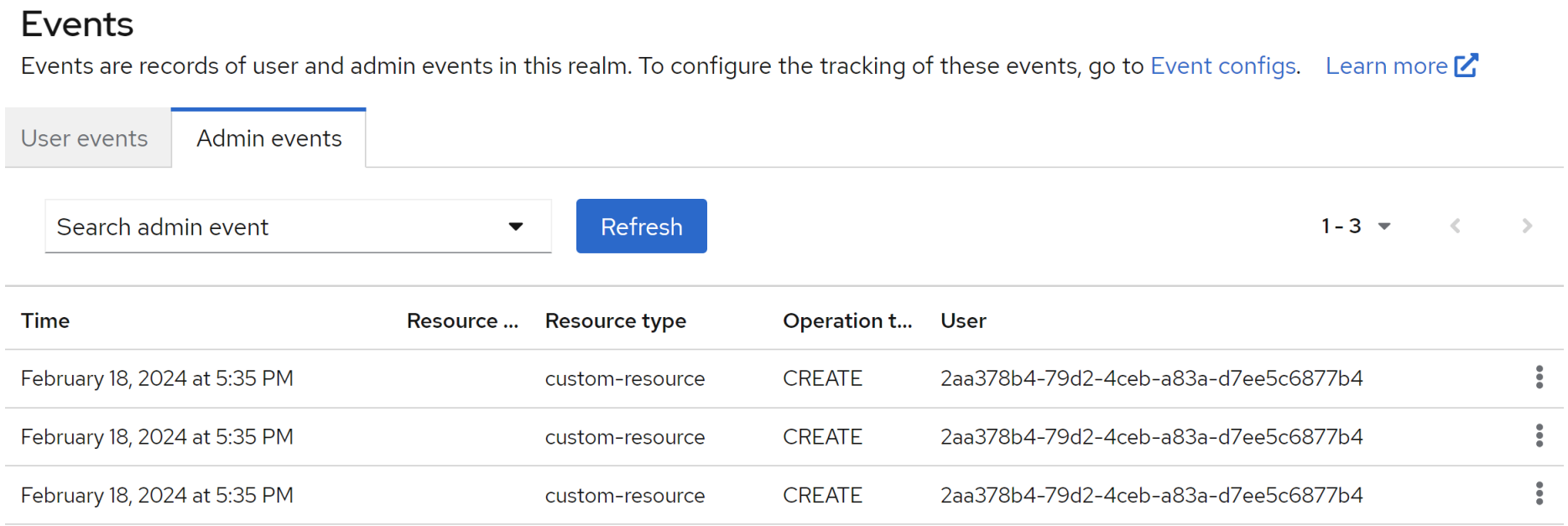

```java
void init() {
    AuthenticationManager.AuthResult authResult = new AppAuthManager.BearerTokenAuthenticator(session).authenticate();
    AdminAuth adminAuth = new AdminAuth(session.getContext().getRealm(), authResult.getToken(), authResult.getUser(), authResult.getClient());
    this.auth = AdminPermissions.evaluator(session, session.getContext().getRealm(), adminAuth);
}
```

Admin Resources – enjoy some free lunch

If you want to extend the Admin API implement **`AdminRealmResourceProvider`**

```
public class CustomAdminRealmResourceProvider implements AdminRealmResourceProvider {
    public CustomAdminRealmResourceProvider() {}

    @Override
    public Object getResource(KeycloakSession keycloakSession,
                              RealmModel realmModel,
                              AdminPermissionEvaluator adminPermissionEvaluator,
                              AdminEventBuilder adminEventBuilder) {
        return new CustomAdminResource(keycloakSession, realmModel, adminPermissionEvaluator, adminEventBuilder);
    }

    @Override
    public void close() {
    }

}
```

Auditability – it‘s not a given either!




```
new AdminEventBuilder(realm, adminAuth, session, clientConnection)
    .resource("custom-resource")
    .operation(OperationType.CREATE)
    .success();
```

Time-related tasks

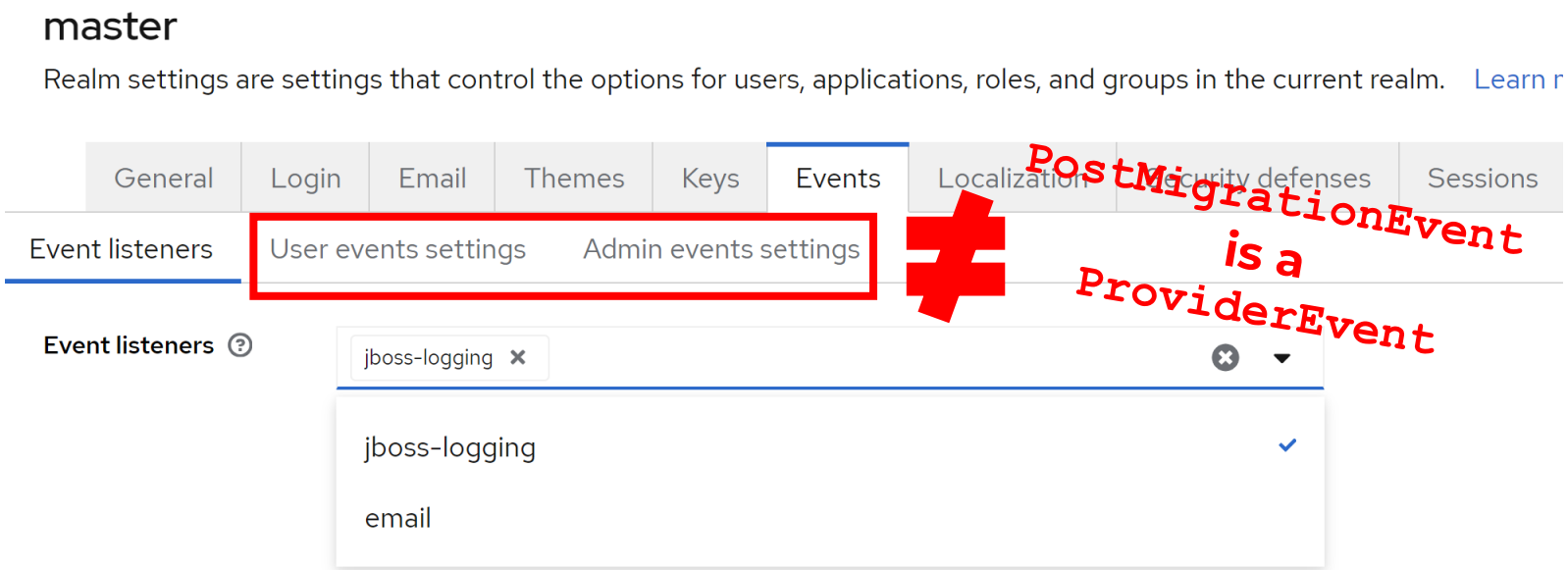
General use-cases

- Syncing users from custom user federations
- Cleanup jobs
  - Remove user accounts
  - Disable user accounts
- Collecting metrics

The EventListenerProvider Anti-Pattern

General pattern that I see in the wild is to implement the

EventListenerProvider



Build a custom SPI

```java
public final class ScheduledTaskSpi implements Spi {

    @Override
    public boolean isInternal() {
        return true;
    }

    @Override
    public String getName() {
        return "scheduled-task";
    }

    @Override
    public Class<? extends Provider> getProviderClass() {
        return ScheduledTaskProvider.class;
    }

    @Override
    public Class<? extends ProviderFactory<ScheduledTaskProvider>> getProviderFactoryClass() {
        return ScheduledTaskProviderFactory.class;
    }
}
```

```java
interface ScheduledTaskProvider extends Provider {
    ScheduledTask getScheduledTask();

    long getIntervall();

    String getTaskName();
}
```

Build a custom SPI

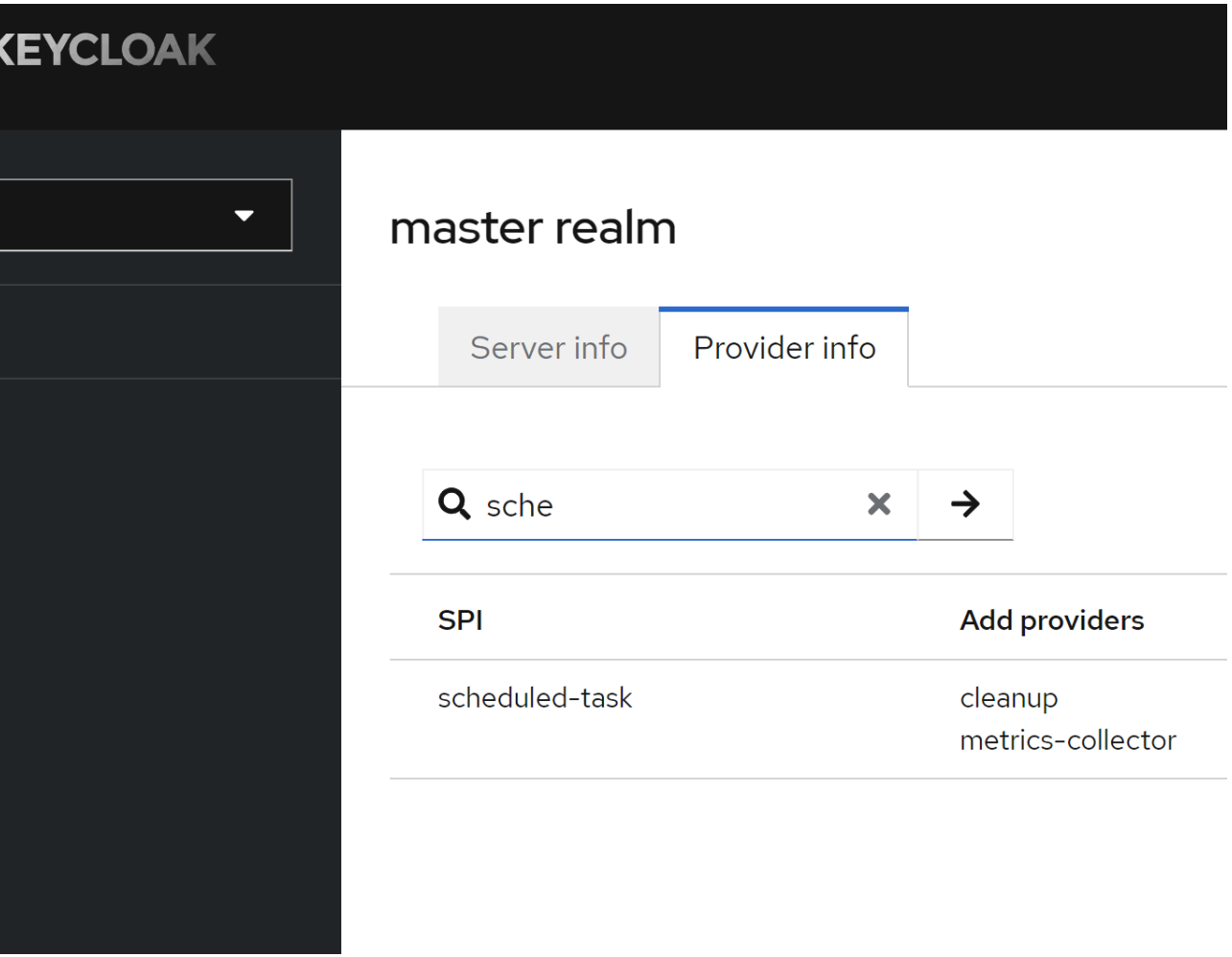
```
public abstract class ScheduledTaskProviderFactory implements ProviderFactory<ScheduledTaskProvider> {
    private KeycloakSessionFactory keycloakSessionFactory;

    @Override
    public final void postInit(KeycloakSessionFactory keycloakSessionFactory) {
        this.keycloakSessionFactory = keycloakSessionFactory;
        keycloakSessionFactory.register((event) -> {
            if (event instanceof PostMigrationEvent) {
                KeycloakSession session = keycloakSessionFactory.create();
                TimerProvider provider = session.getProvider(TimerProvider.class);
                ScheduledTaskProvider stp = create(session);
                provider.scheduleTask(stp.getScheduledTask(), stp.getIntervall(), stp.getTaskName());
            }
        });
    }
    @Override
    public final void close() {
        KeycloakSession session = keycloakSessionFactory.create();
        TimerProvider provider = session.getProvider(TimerProvider.class);
        ScheduledTaskProvider stp = this.create(session);
        provider.cancelTask(stp.getTaskName());
    }
}
```

Register new tasks

```
public final class CleanupScheduledTaskProviderFactory
    extends ScheduledTaskProviderFactory {
    @Override
    public ScheduledTaskProvider create(KeycloakSession
keycloakSession) {
        return new CleanupScheduledTaskProvider();
    }

    @Override
    public void init(Config.Scope scope) {
    }

    @Override
    public String getId() {
        return "cleanup";
    }
}
```

```
public final class CleanupScheduledTaskProvider
    implements ScheduledTaskProvider {

    @Override
    public ScheduledTask getScheduledTask() {
        return (keycloakSession -> {
            // Cleanup here
        });
    }

    @Override
    public long getIntervall() {
        return 1000;
    }

    @Override
    public String getTaskName() {
        return "test";
    }
}
```

Nice job list ;)



How to configure providers?! ☝️

Remeber the
ServerInfoAwareProviderFactory !

Pro Tip: Scheduled jobs within a cluster

How to prevent running the same scheduled task on all cluster nodes?

`ClusterProvider` to the rescue!

```
public final class CleanupScheduledTaskProvider implements ScheduledTaskProvider {
    @Override
    public ScheduledTask getScheduledTask() {
        return (keycloakSession -> {
            ClusterProvider cluster = keycloakSession.getProvider(ClusterProvider.class);
            cluster.executeIfNotExecuted(getTaskName() + "::scheduled", 1000, () -> {
                // Clean up
                return null;
            });
        });
    }
```

Thank you for your attention!

sven-torben.janus@conciso.de

@sventorben
@mas.to
.bsky.social

Join our team!

Join our team!