FoundryVTT@k8s

Roland Thomas Lichti

Version 1.x, 2025-12-06

Table of Contents

This documentation is also available in PDF format.

Chapter 1. OCI Image

Chapter 2. Helm Chart

Chapter 3. kp-users arc42 architecture documentation

This documentation is also available in PDF format.

3.1. Introduction and Goals

This project provides a container image for Foundry VTT and a helm chart to install it.

Requirements Overview

- [FS01 Tailored to Spring Security]
- [FS02 Data Protection]
- [CO01 Compatible with keycloak]
- [US02 Multi-Language]

Quality Goals

- [RE01 Working hours 24/7]
- [US03 Fast Response Times]
- [MT01 Small Team]

Stakeholders

Role/Name	Contact	Expectations
Paladins Inn	Roland T. Lichti	A small project to provide a place for players to meet and play games together.
Torganized Play	Roland T. Lichti	The first and most important project to use kp-users is the Delphi Council Information System supporting Torganized Play.

3.2. Architecture Constraints

Table 1. Technical Constraints

ID	Description
CT-001	Runtime Environment Kubernetes
	The Target runtime environment is Kubernetes.
CT-002	Publishing Chain
	The pubishing chain is GitHub (with actions) and the containers and helm charts get published via quay.io.
CT-003	Programming Languages
	The main programming language is Java. spring-boot will be used as main framework.

Table 2. Organisational Constraints

ID	Description
CO-001	Kaiserpfalz EDV-Service
	The software will be distributed by Kaiserpfalz EDV-Service.

Table 3. Political Constraints

ID	Description
CP-001	Non Profit
	The software is developed without profit. It is provided via LGPL v3.0 or newer.

3.3. System scope and context

3.3.1. Business Context

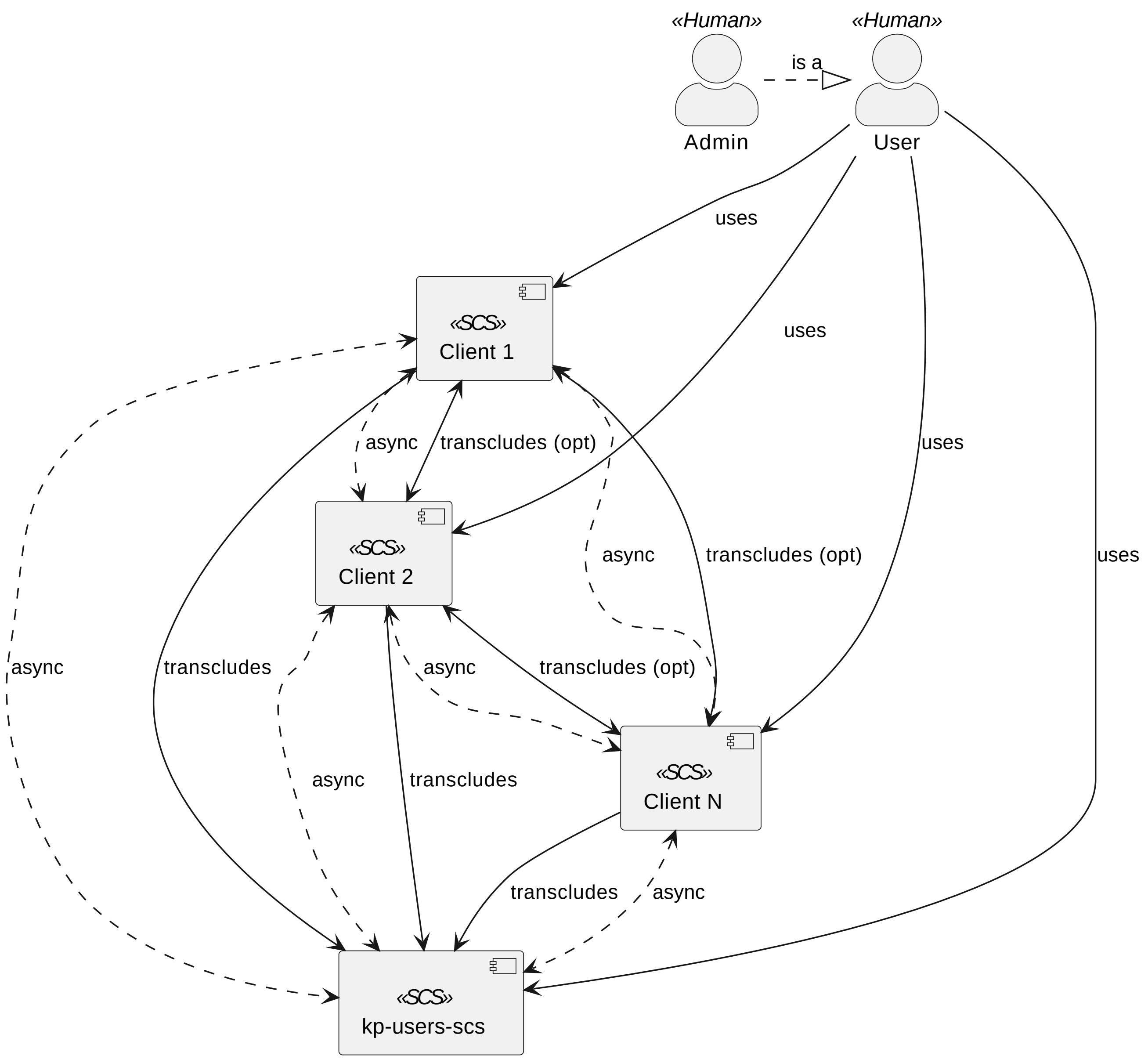


Figure 1. The business context of the DCIS.

3.3.2. Technical Context

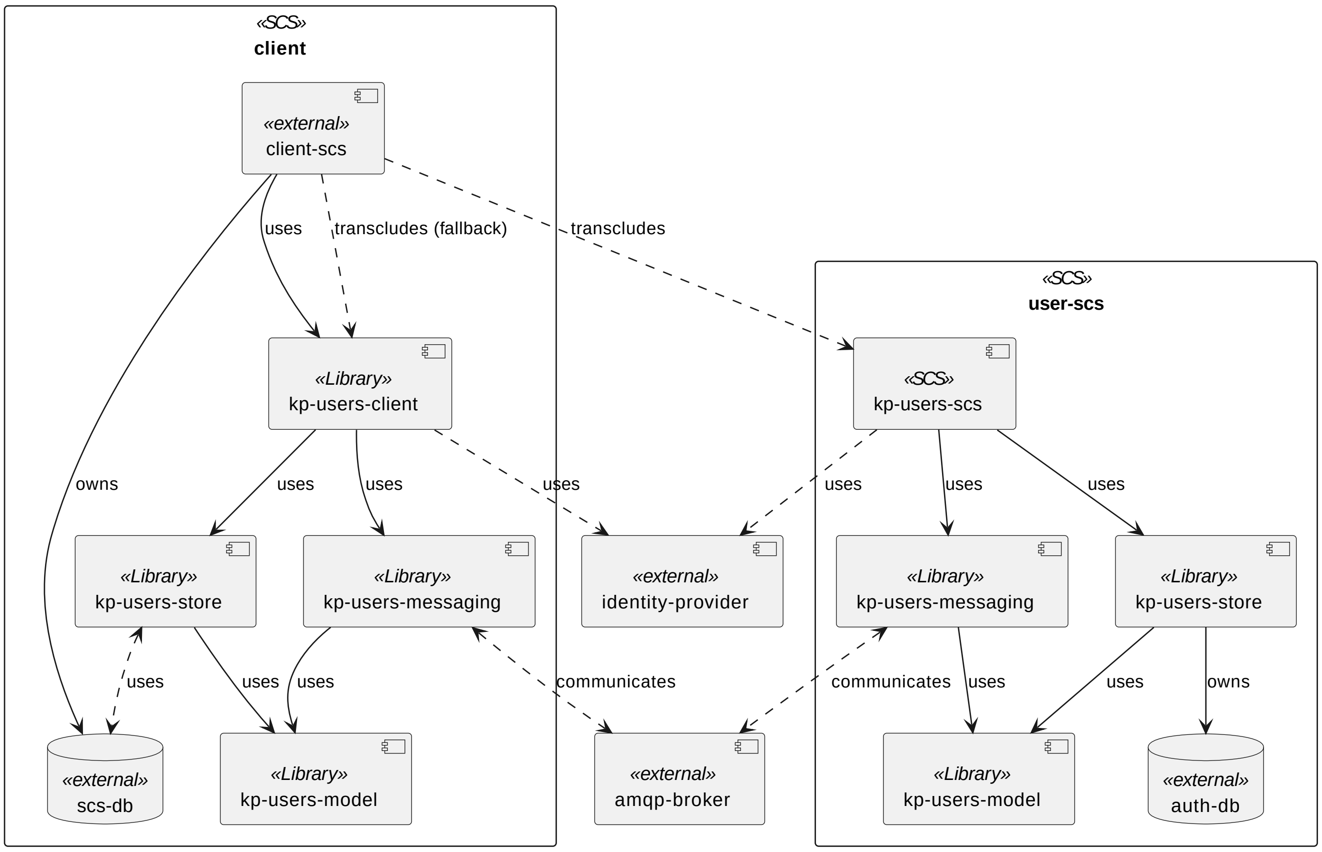


Figure 2. The technical context of the DCIS.

3.4. Solution Strategy

Modularization

The system is composed of multiple components. The components are managed in a multi-module maven project.

The module kp-users-client is the customer facing module. It combines the store and the messaging and will be used by all SCS.

Single Sign On

The system holds no authentication data. It will be connected via OpenIDConnect to external identity providers (like keycloak). The users are specified via their external user. Internally they get an UUID which maps to the (Issuer, User) tuple of the external provider.

Frontend and backend integration

The systems use frontend integration methods for the UI.

Needed communication between the backends (data synchronization, event distribution) are handled via a messaging infrastructure. The broker {madr-003} used will be a rabbitMQ handling AMQP queues and topics.

We heavily rely on self-contained systems {scs} as promoted by INNOQ.

The asynchronous data replication is addressed in section kp-commons:arc42:08_concepts/asynchronus-data-handling.adoc.

3.5. Building Block View

Whitebox Overall System

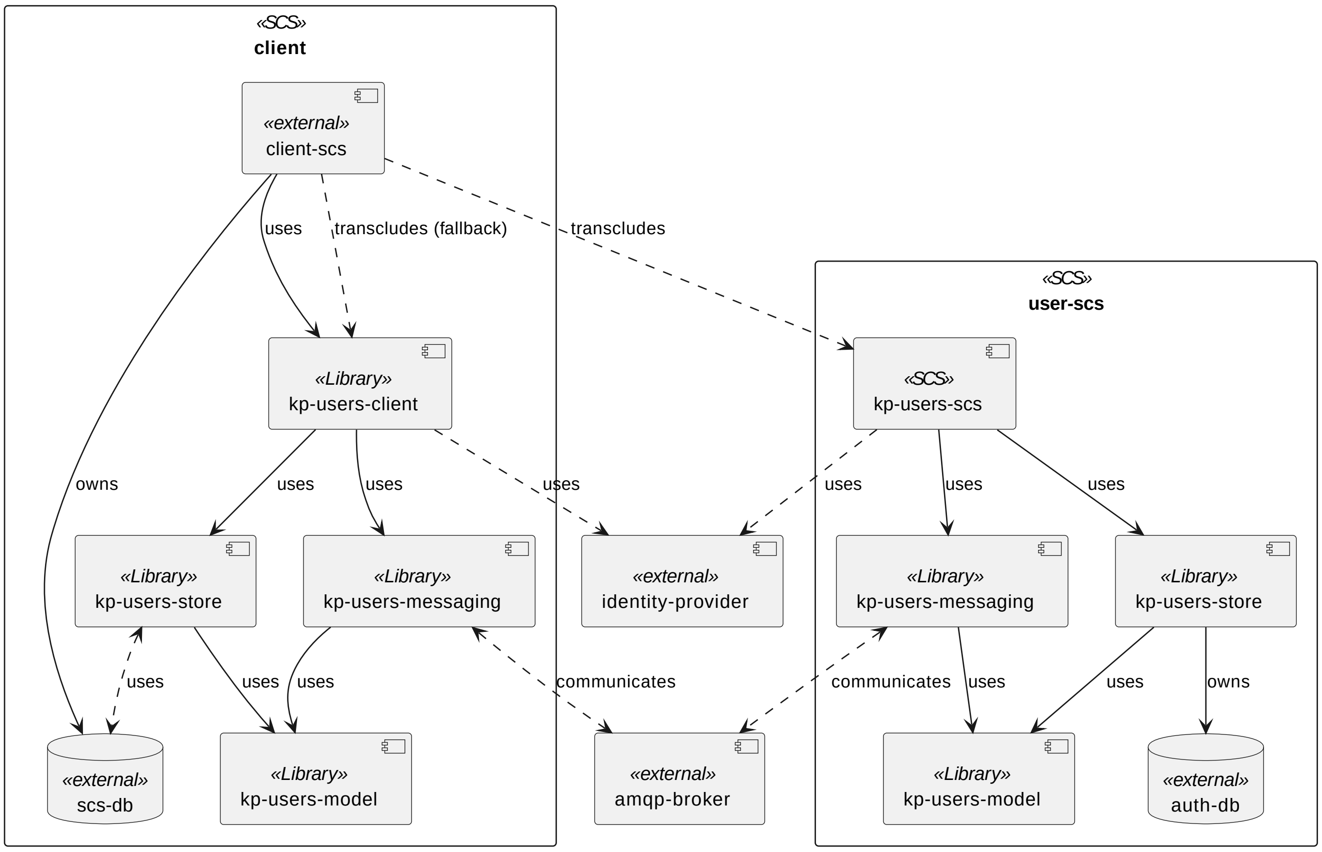


Figure 3. Architectural Overview

Motivation

The system tries to separate the concerns of messaging, data storage and data model.

Contained Building Blocks

kp-users-model [Level 1 | Level 2 | Level 3]

- the data model

kp-users-store [Level 1 | Level 2 | Level 3]

- the data store

kp-users-messaging [Level 1 | Level 2 | Level 3]

- the messaging system between the SCS

kp-users-client [Level 1 | Level 2 | Level 3]

- the client to be used in all SCS
- integration into Spring Boot Security as AuthenticationProvider

kp-users-scs [Level 1 | Level 2 | Level 3]

- management UI
- authoritative data source for the system.

Important Interfaces

IDs

IDs throughout the system are UUID, not the simple numbers used by other systems. Reason is, that the ID should be generated on first creation of an object and UUID is a nice way to handle that distribution.

3.5.1. Level 1

3.5.1.1. Level 1: kp-users-scs

Pages

Description	Permissions
`users/`	
Lists all users in the system.	`users:list` `owned`
`users/{userId}`	
Displays the details of a specific user.	`users:read` `owned`

Transcludes

Description	Width	Height	Permissions
`/users/{userId}/card`			
A card displaying the user information.	min: 100px max: 150px	min: 200px max: 300px	`authenticated`
`/users/{userId}/avatar`			
Displays the user’s avatar.	min: 75px max: 75px	min: 75px max: 75px	`authenticated`
`/users/{userId}/petition`			
Displays a link to petition the user. It is an icon.	HTML link		authenticated

3.6. Runtime View

Overview

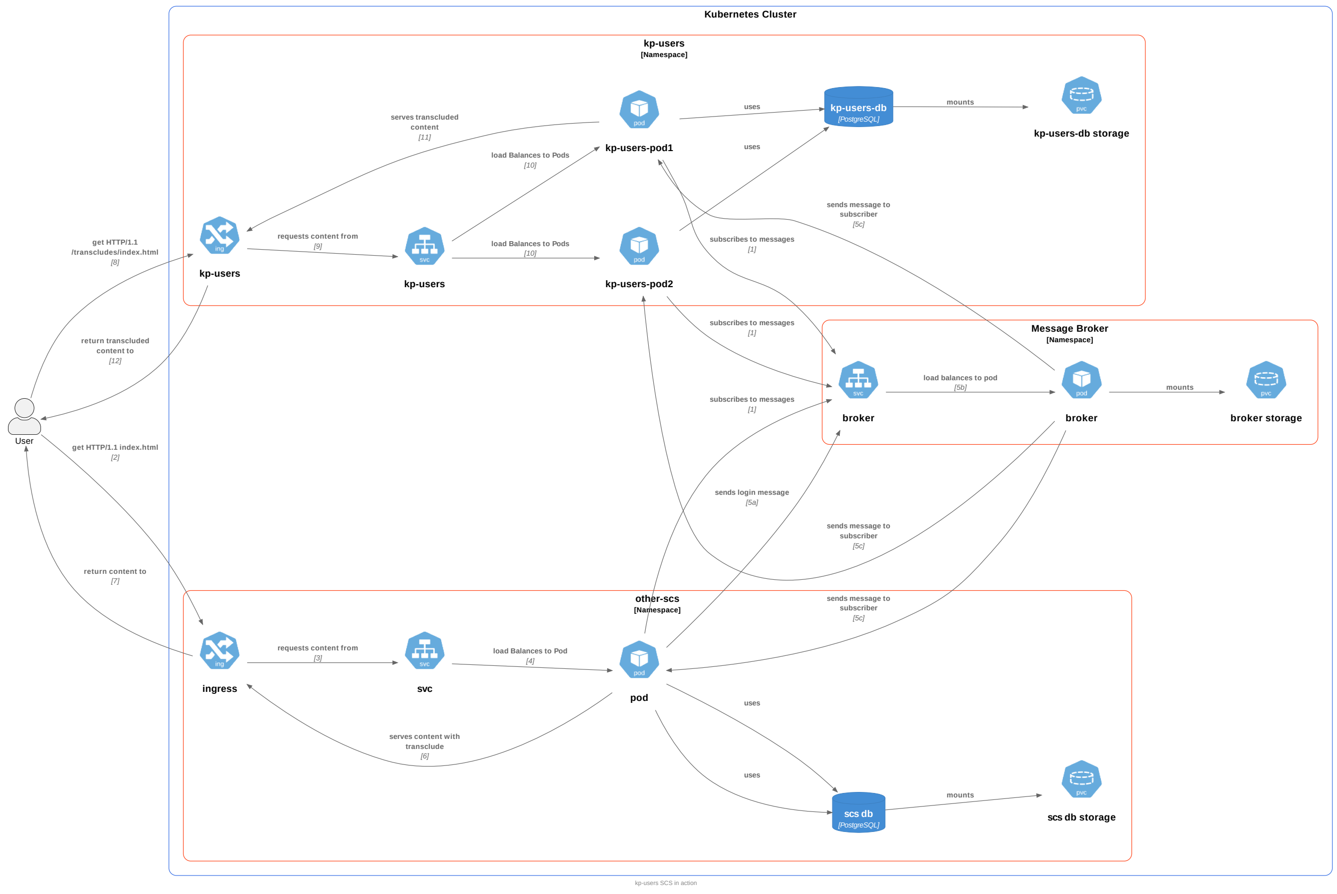


Figure 4. Overview of the usage of the kp-users by other SCS and their users.

① At system startup the SCS subscribe to the user management topics on the message broker.

② The user accesses the other-scs with a HTTP request.

③ The process gets handled by the ingress controller which forwards the request to the service.

④ The service load "balances" the request to one of the pods (there is only one).

⑤ The handling is done in parallel of serving the request.
5a: The pod sends the login event to the broker.
5b: The broker load balances the request to one of the broker pods.
5c: the event is sent to all subscribers of the topic.

⑥ The pod send the content with a transclude to kp-users to the user (via ingress).

⑦ The user gets the content with the transclude.

⑧ The user requests the transclude from the kp-users SCS.

⑨ The request is handled by the ingress controller which forwards the request to the service.

⑩ The service load balances the request to one of the pods.

⑪ The pod serves the transcluded content.

⑫ The ingress controller returns the transcluded content to the user.

3.6.1. kp-users-client

TBD

3.6.2. kp-users-scs

Handles the user management stuff for the whole system. In addition the SCS handles arbitation and notification of users to external systems like e-mail or discord.

Pages

Table 4. Web based UI of the dcis-users system

Name	Method	URL	Permission
Use case: List users			
List Users	`GET`	`/users/index`	anon
	List all users matching the query parameters.		
Use case: Create user			
Create User	`POST`	`/users/`	• ADMIN • ORGA • self
	Creates a user with the given data.		

REST API

Table 5. REST API of the dcis-users system

Name	Method	URL	Permission
List Users	`GET`	`/users/api/v1index`	anon
	List all users matching the query parameters.		
Create User	`POST`	`/users/api/v1/users/`	• ADMIN • ORGA • self
	Creates a user with the given data.		

Transcludes

Table 6. Transcludes of dcis-users to be included on other pages.

Name	URL	MinX	MinY	MaxX	MaxY
User List	`/users/tc/?id[]=<ID1>&id[]=<ID2>`	100	300	500	400
	Lists users with a link to the users details page.				

Name	URL	MinX	MinY	MaxX	MaxY
User Title	`/users/<ID>/title`	20	20	200	20
	Gives the username to be displayed on pages				
ID-Card	`/users/<ID>/card`	200	300	200	300
	A standardized card for displaying users.				
List Arbitration	`/users/arbitration/<system>/<entity>/<id>/list`	400	100	400	300
	Lists running arbitration for the given entity.				
Contest Entity	`/users/arbitration/<system>/<entity>/<id>/start`	400	300	400	300
	Start an arbitration for this entity.				
Arbitration Card	`/users/arbitration/<id>/card`	200	300	200	300
	A standardized card for displaying a single arbitration.				

Messaging Channels

This is an overview over existing general queues that exist in the whole system.

Table 7. Messaging channels of the dcis-users.

Name	Type	Direction	SCS
dcis.user.log	Queue	Inbound	dcis-users
	New log entries for the users action log		
dcis.user.notification	Topic	Outbound	dcis-users
	Changes to user states are published for recognition by other systems.		
dcis.user.registration	Queue	Inbound	dcis-users
	If users register themselves on other systems they can send the new user into this system.		
dcis.arbitration.start	Queue	Inbound	dcis-users
	Starting an arbitration.		
dcis.arbitration.file	Queue	Inbound	dcis-users
	File Information to an arbitration		
dcis.arbitration.close	Queue	Inbound	dcis-users
	Close arbitration by system.		
dcis.arbitration.notification	Topic	Outbound	dcis-users
	Changes to arbitrations are published for recognition by other systems.		
dcis.user.contact	Queue	Inbound	dcis-users
	Notify a user via e-mail, discord, …		

Scheduled Jobs

Table 8. Scheduled jobs to do data house keeping

Name	Period
Unban Users	daily
	Unban users at end of their banning period.
Block Users	daily
	Block users inactive for more than 2 years.
Delete Users	daily
	Delete user data blocked/marked for deletion more than 3 years.
Purge Logs	yearly
	Purge logfiles after 10 years.

3.7. Deployment View

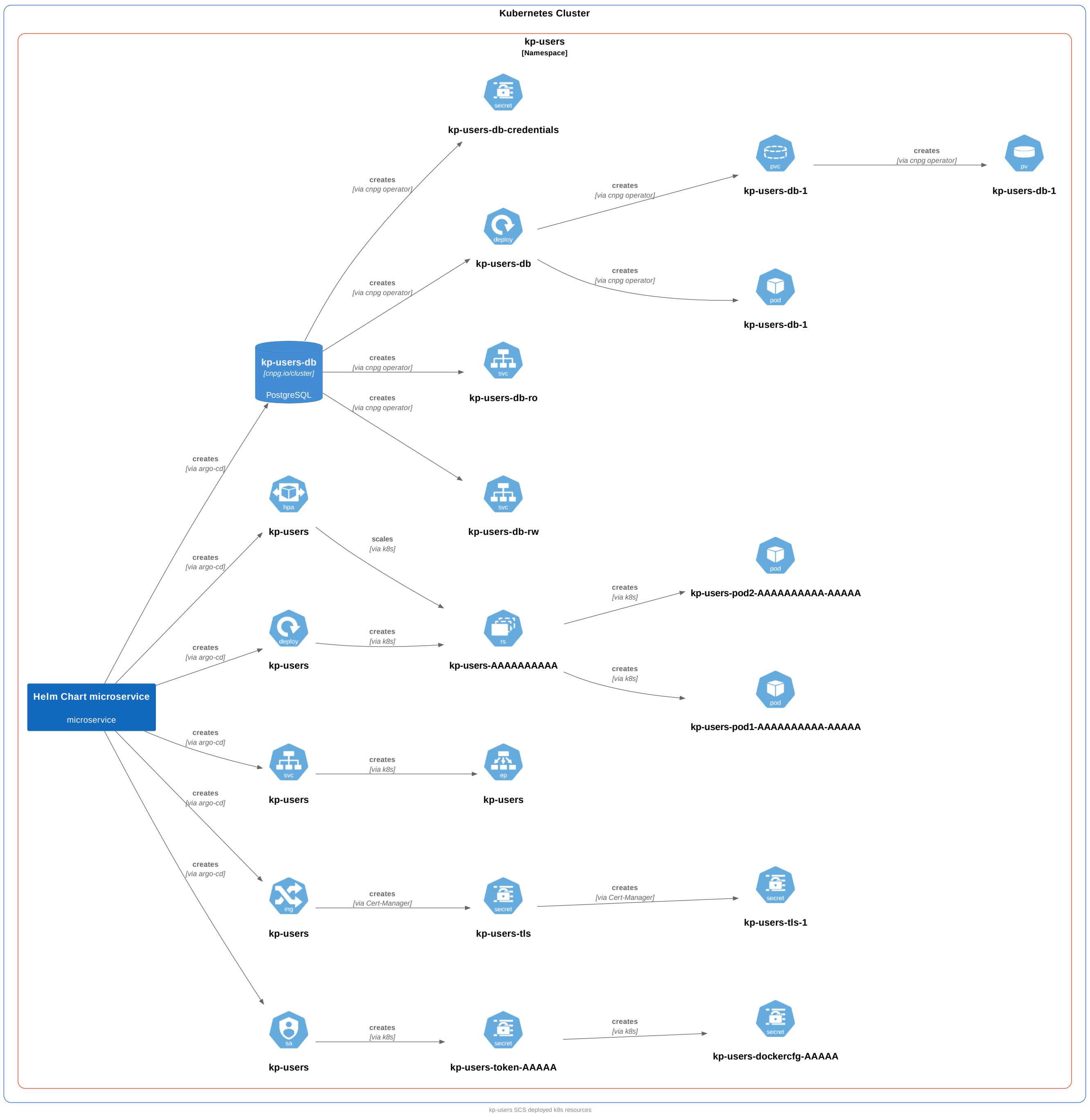


Figure 5. Overview of the deployment of the kp-users SCS within a Kubernetes cluster.

3.8. Cross Cutting Concepts

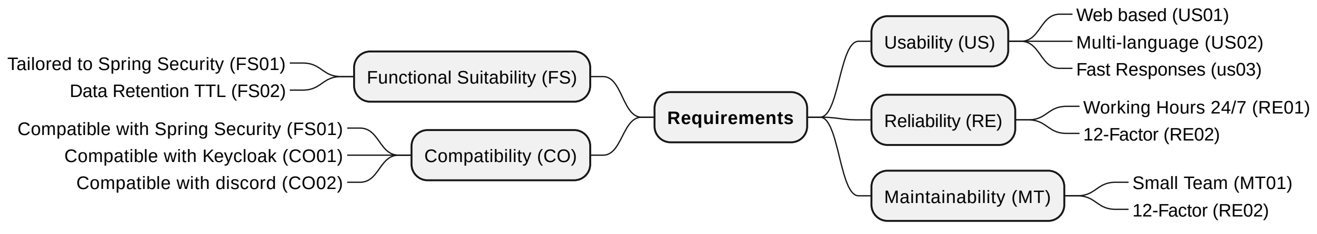
Please refer to the macro architecture documentation for the general kp-commons:arc42:08_concepts/index.adoc sections.

3.9. Architecture Decisions

3.10. Quality Requirements

Requirement	Description
Functional Suitability	• The software **has to** manage user accounts.
Compatibility	• The managed data **has to** integrate into spring-security.
Usability	• The software **has to** be accessible via Web Browsers • The software **has to** be internationalized. There has to be localization for at least *German* and *English* languages.
Reliability	The systems needs to operate for world wide consumption.
Maintainability	The software should be easily maintainable. This includes using widely used frameworks so help can be given quite easily.

Quality Tree


Requirements
Functional Suitability (FS)
Tailored to Spring Security (FS01)
Data Retention TTL (FS02)
Compatibility (CO)
Compatible with Spring Security (FS01)
Compatible with Keycloak (CO01)
Compatible with discord (CO02)
Usability (US)
Web based (US01)
Multi-language (US02)
Fast Responses (us03)
Reliability (RE)
Working Hours 24/7 (RE01)
12-Factor (RE02)
Maintainability (MT)
Small Team (MT01)
12-Factor (RE02)

Quality Scenarios

ID	Scenario
FS01	The system is created for integrating into Spring Security.
FS02	The software needs to conform to the GDPR in Europe.
CO01	The data transfer between keycloak and the system should be possible.
CO02	The system should offer a discord bot to use the data.
US01	The main interface should be web based.
US02	The system **has to** be available at least in **German** and **English** language. Other translations **should** be easily addable.

ID	Scenario
US03	Request have to be answered quickly. The following time percentiles are sufficient:

Percentage of requests	**Response Time**
95%	1,5s
90%	2s
75%	2,5s
50%	4s
25%	5

ID	Scenario
RE01	Torg Eternity is played around the world. So the systems have to work 24/7. There is no maintenance window available where no users would be affected.
RE02	Following the guidelines of 'The Twelve-Factor App' {12factor} support the maintainability and resilience of the system.
MT01	Maintaining the software and the data must be possible for a small team (basically the full army of myself, me, and I).

3.11. Risks and Technical Debts

Table 9. Risks and technical debts

ID	Risk
`TR-001`	**The Team is too small**
	The long development shows that the team is too small for this project.