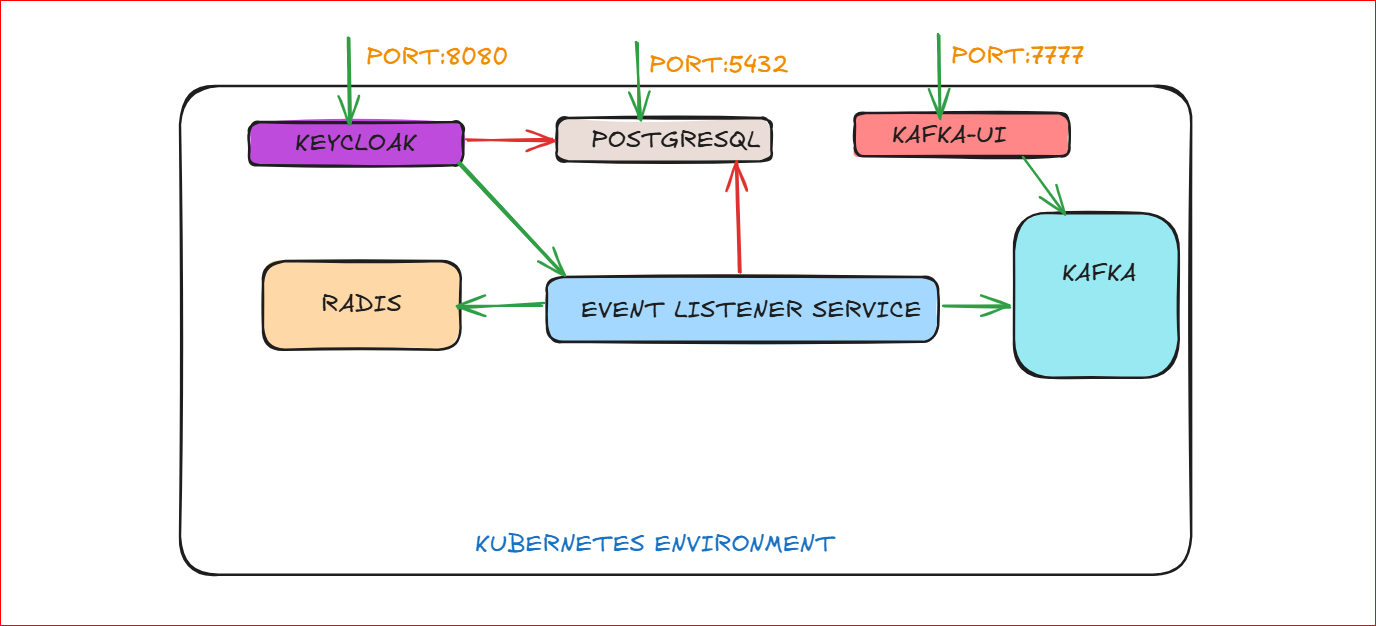
Özet:

* Framework olarak **Spring Boot** kullanıldı.
* Keycloak olay dinleme için bir tane **Keycloak Event Listener SPI** plugin yapıldı.
* Keycloak’tan üretilen olaylar, **Spring Boot** ile yazılan **Event Listener** asında bir servise REST call ile aktarılması yapıldı.
* **Event Listener** servisne gelen olayların, deplonması için ve sıralı olması için **Kafka** kullanıldı ve tek bir partition seçildi.Çünkü kafka sadece partition bazında sıralı garanti eder. Birden fazla partition kullanılmış olsaydı, olayların sadece partition bazında sıralaması garanti edilirdi.
* EventListener servisi ile Kafka arasındaki entegrasyon yapıldı.Eğer Kafka down olursa, **PostgreSQL database’ine** olaylar aktarılması sağlanacaktır.
* Kafka down olduğunda, Kafka’daki verilerin de kaybolmaması için bir Mount tanımı yapıldı.
* Kafka ayağa kalktığında, DB’deki olaylar sırasıyla Kafka’ya aktarımı bir **Job/Scheduled** ile yapıldı. Bu job her 10 saniyede bir çalışacaktır ve Job bitmeyene kadar, sonraki çalışma planına geçmeyecektir.
* Job olayları işlerken, eğer yeni bir event gelirse, bu event Kafka’ya değil DB’ye aktarılacaktır ve db’de kayıtlar sırayla hepsi bitmeden, doğrudan Kafkaya aktarım yapılmayacaktır. **Böylece sıralama garanti edilmiş olacaktır**.
* **Radis** kullanıldı. Radis birden fazla **Event Listener** servisi çalıştığında, hepsi aynı aynda aynı kayıtlar üzerinde çalışma yapmaması için(Job sırasında), **dağıtık bir lock** işlemi yapıldı.
* **Event Listener** servisi down olursa ve veya servis hatalı bir case’e girip işlemi başarılı gerçekleştirmez ise, olaylar bu sefer **Keycloak Event Listener SPI** plugini ile doğrudan DB’ye yazılacaktır.
* Bu süreç PostgreSQL kullanılmadan ve sadece bir Redis cache kuyruğu uzerinden de yapılabilirdi ancak ben burada ikisinin de farklı kullanımlarını göstermek amacıyla kullandım.
* Kafka down durumunda, Event Service Down durumunda, Redis down durumunda ve Postgresql down durumunda test edilmiştir. Herhangi bir sorun görünmemiştir ve veri kaybı yaşanmamış ve sıralamada da sorun görülmemiştir.
* Aşağıda mimari olarak servislerin birbiri ile ilişkisi gösterilmiştir.



Asağıdaki adımları sırayla yaparak, yapılan Keycloak Event Listener projemizin testini sağlayabilirisniz.

1. Dokcer'a Login olmak

**docker login**

1. cmd ile **proje** klasörünün bulunduğu dizinin içine girilmesigerekmektedir: **cd proje**
2. Benim gibi lokalde kubernetes ortamı için **minikube** çalıştırıyorsanız, aşağıdaki komutları lütfen sırayla çalıştırınız:

-ortamın hazırlanması için bu komutu çalıştırın lütfen

**minikube start**

-şimdi bu komutu çalıştırmanız gerekmektedir.

**minikube docker-env**

-”minikube docker-env” komutundan sonra, minikube size şöyle bir komut çalıştırmanızı söyleyecek, bunu cmd'de çalıştırın lütfen:

**@FOR /f "tokens=\*" %i IN ('minikube -p minikube docker-env --shell cmd') DO @%i**

-Ayrı bir cmd açınız ve bu komutu çalıştırınız lütfen. Bu komut ile DB'ye, Keycloak'a ve Kafka-UI'e dışarıdan erişmeniz sağlanacaktır.

**minikube tunnel**

1. Yapılandırılmış Keycloak image'i ve Event Listener Service'in image'lerini oluşturmak için proje klasöründe iken **docker compose build** kullanacağız:

**docker compose build**

1. proje klasöründe iken, cd kubernetes diyerek kubernetes klasöürüne gelip, projemizi kubernetes'e deploy sürecine başlayacağız:

**cd kubernetes**

1. Önce verileri kubernetes ortamında oluşturmamız lazım. Bu veriler içinde horizontalpodautoscaler, secret, configmap, pvc, pv gibi veriler mevcuttur.

**kubectl apply -f data**

1. verileri oluşturduktan sonra servislerimizi oluşturmamız lazım:

**kubectl apply -f service**

1. servisleri oluşturduktan sonra deployment'ları oluşturmamız lazım:

**kubectl apply -f deployment**

1. tüm podlarımız statusü 1 olana kadar bekleyelim lütfen. İlk çalıştırmada, image'leri indirme süreci de olduğundan, container'ların oluşması biraz zaman alabilir.

**kubectl get pods**

1. 6,7,8,9 adımlarımız tamamlandıktan sonra, özet olarak tüm kaynaklara bakabiliriz:

**kubectl get all -n default**

1. Tüm kontainer'lar ayağa kalktıktan sonra, ilk olarak Keycloak'a gidip event listener ayarlarını yapmamız gerekmektedir.

**Keycloak admin ekranı:** [**http://localhost:8080/admin/master/console/**](http://localhost:8080/admin/master/console/)

**kullanıcı adı:admin**

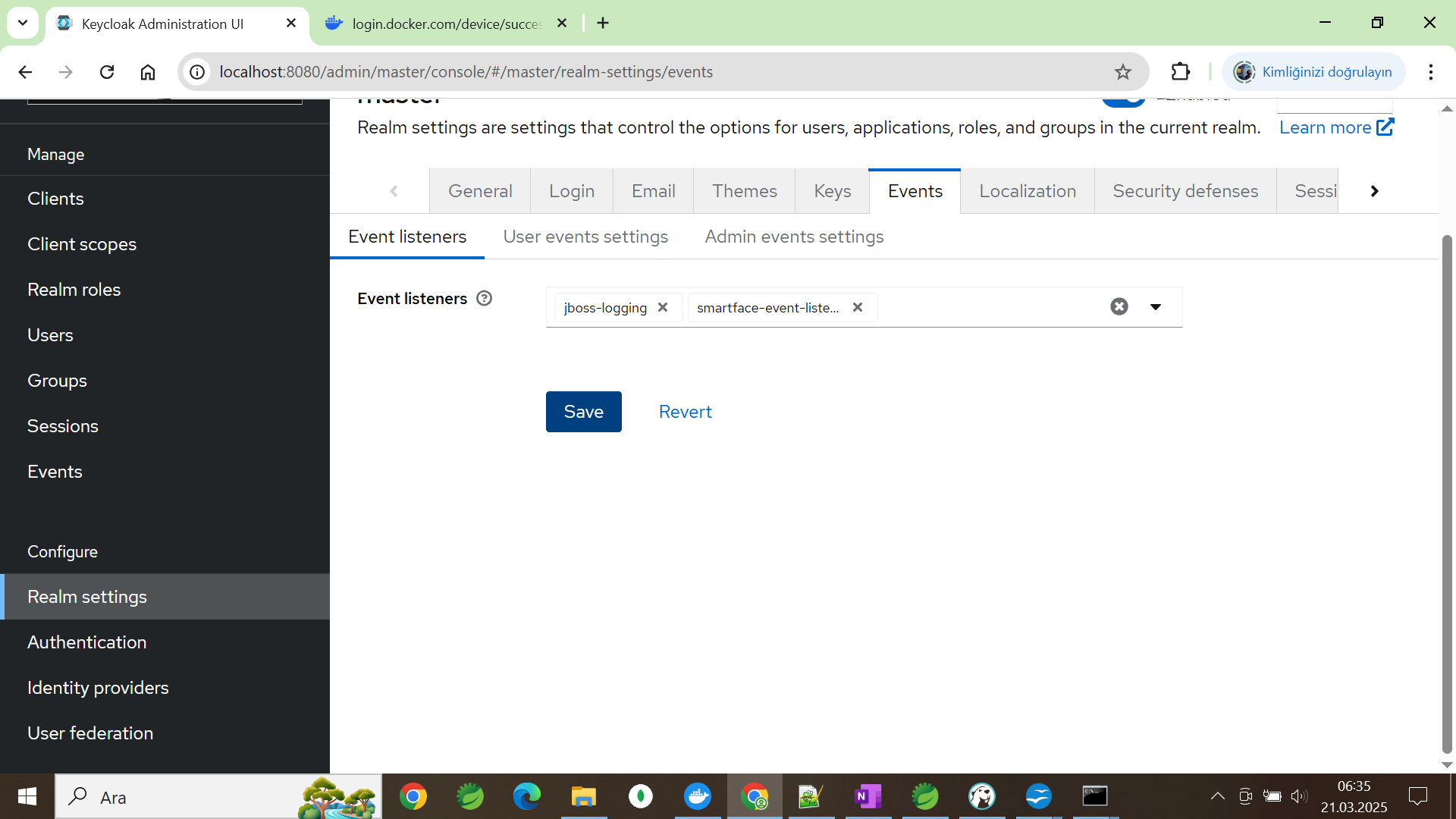
**şifre: admin**

**Giriş yapıldıktan sonra;**

**solda Realm settings'e gidip, Event listeners, User events settings,Admin events settings** bu üç tab'da da değişiklik yapılacaktır.

Yeni oluşturduğumuz olay dinleyicimizi burada seçip, kaydediyoruz:

smartface-event-listener

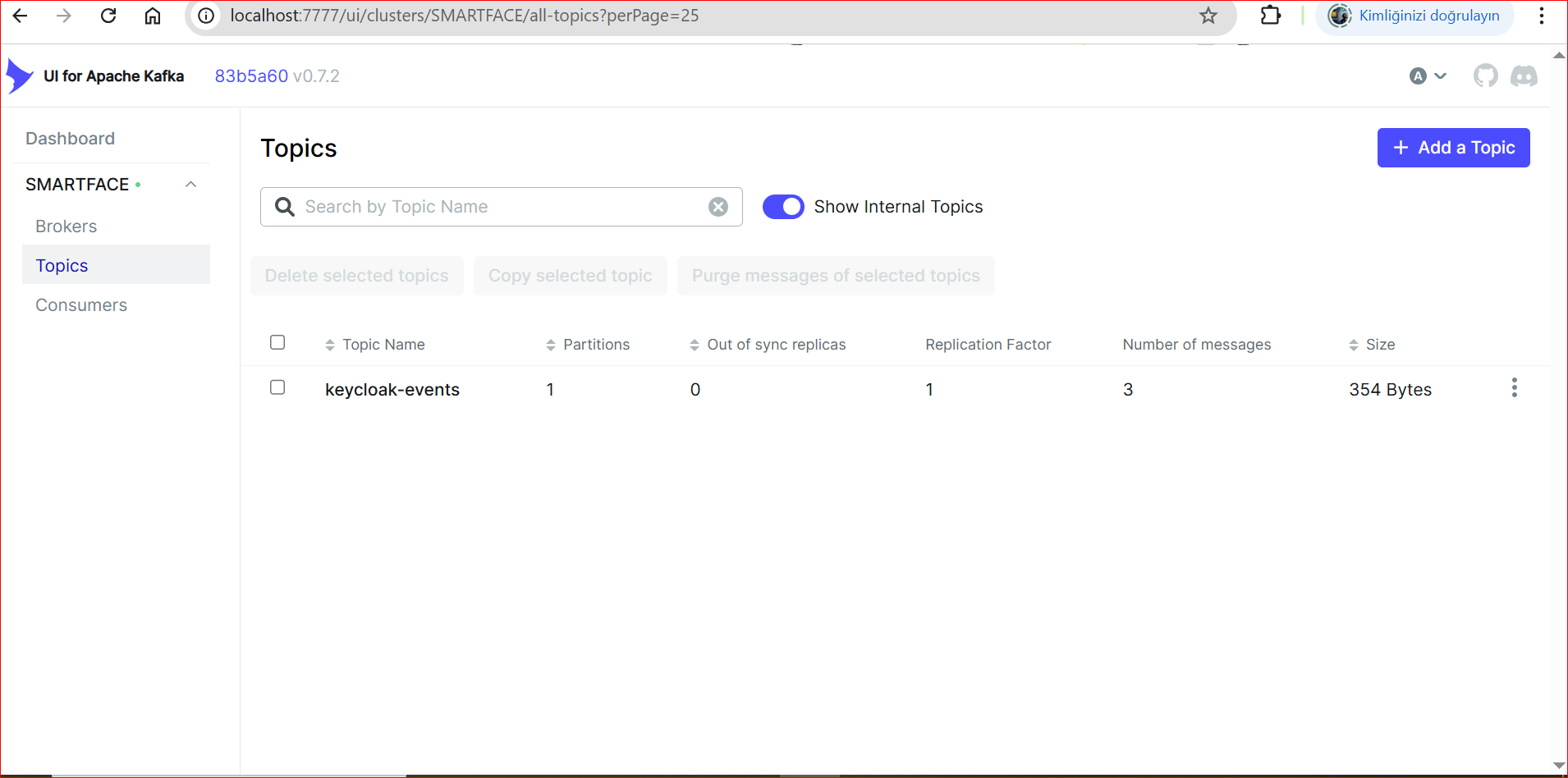


1. Şimdi postgresql db ve kafka'ya bağlanıp verilerin eklenip eklenmediğine bakalım.

Öncelikle **kafka ayakta olduğunda** deneme yapacağız.

**Kafka-ui: http://localhost:7777**

Keycloak admin ekranına girip çıkınca :

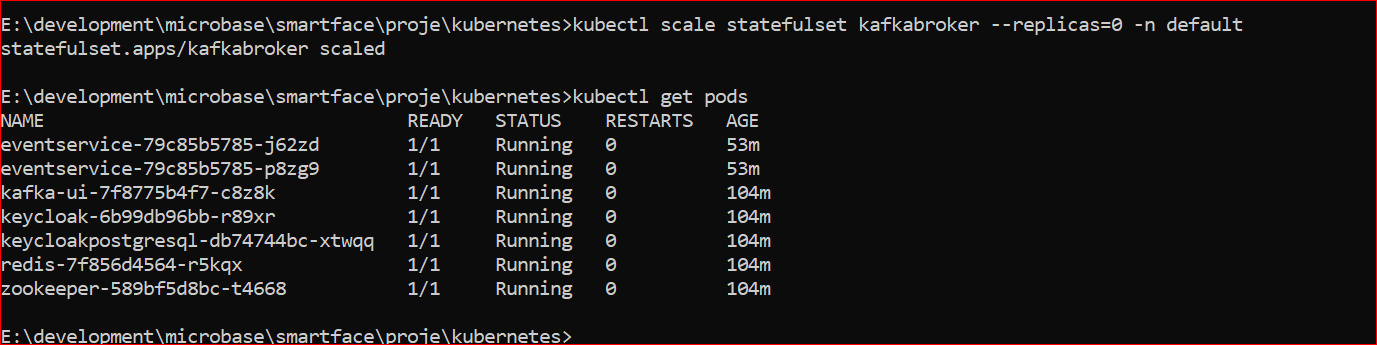


şeklinde bir görünüm elde ettik. 3 tane Event kafkaya başarılı eklendi.

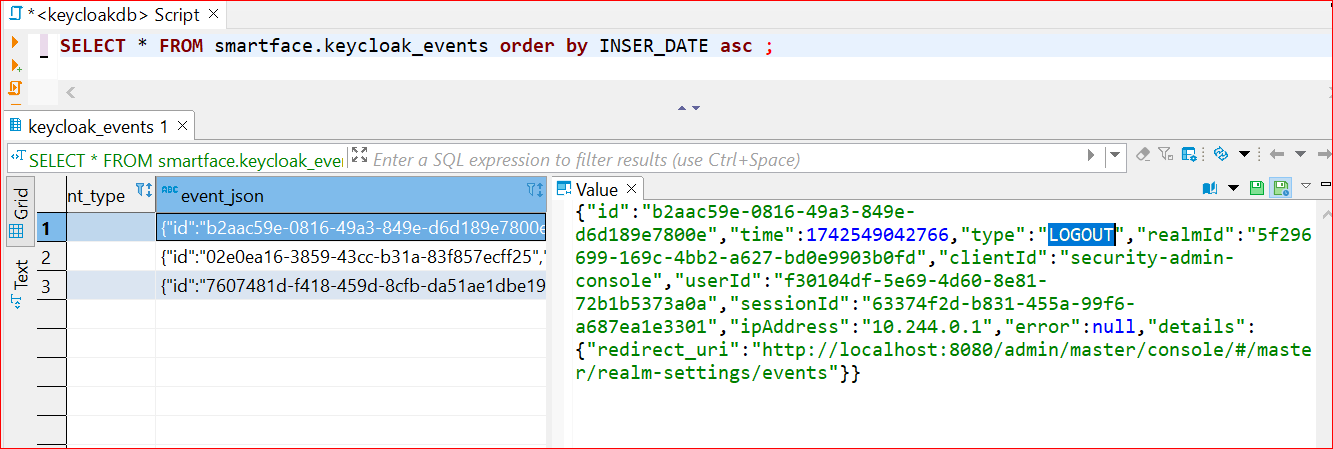
1. **Şimdi kafka'yı down edeceğiz** ve kayıtları keycloak'ın da bağlı olduğu postgresql db'ye aktarımın sağlayacağız.

**kubectl scale statefulset kafkabroker --replicas=0 -n default**

kullanarak gecici olarak kafkayı down edelim.

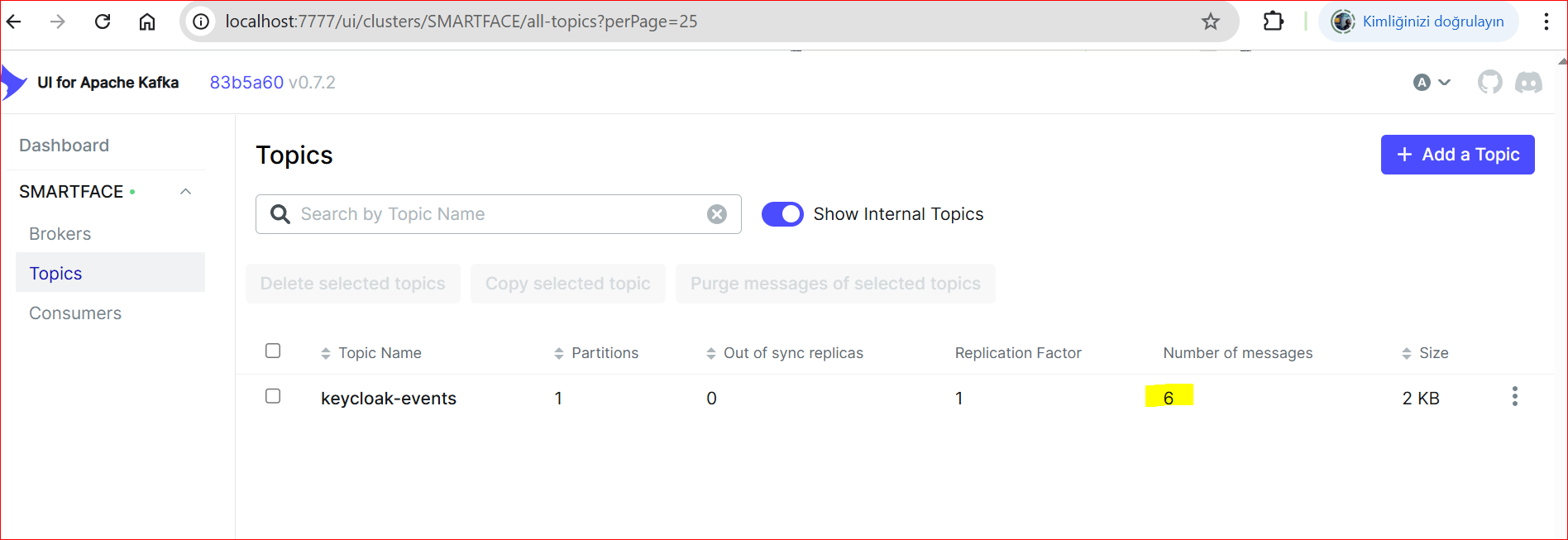
 Kafka'yı kapattık.

Şimdi keycloak admin ekranından login logout olacağız ve kayıtların db'ye aktarıldığını göreceğiz.



Görüldüğü gibi logout ve login olduğumuzda veriler db'ye yazıldı. Ve şimdi Kafka'yı ayağa

kaldırdıktan sonra, bu işlemlerin kafka'ya aktarıldığını ve db'den de kaldırıldığını göreceğiz.



Kafka için mount tanımı yaptığımız için, kafka down olduğunda da önceki 3 kaydını kaybetmedi ve yeni olanlar da eklenerek, 6 tane kayda ulaştı.

