Laboratuvar: Statik Analiz Kullanımı

Gerekli tahmini süre: 30 dakika

Statik Analiz Kullanımı için uygulamalı laboratuvara hoş geldiniz! Statik analiz, yürütmeden önce kaynak kodunu otomatik olarak inceleyen bir hata ayıklama yöntemidir. Bu laboratuvarda, statik analiz hakkında daha fazla bilgi edinecek ve projeniz için SonarQube'u nasıl kurup yapılandıracağınızı ve kullanacağınızı öğreneceksiniz.

Öğrenme Hedefleri

Bu laboratuvarın sonunda, şunları yapabileceksiniz:

- Statik analiz kullanmanın faydalarını tanımlamak
- SonarQube'u kurmak ve yapılandırmak
- Statik analiz taramaları gerçekleştirmek
- SonarQube'den gelen güvenlik raporlarını yorumlamak

Statik Kod Analizini Anlamak

Statik kod analizi, kaynak kodunu inceleyen bir hata ayıklama yöntemi olup, programı çalıştırmadan kodu incelemek anlamına gelen bir çalışma ortamında gerçekleştirilir. Bu uygulama, kaynak kodu analizi olarak da bilinir.

Test işlemleri geleneksel olarak kodlayıcı tarafından program çalıştırılarak yapılırken, kaynak kodu analizi bir program tamamlanmadan bile yapılabilir. Bu, geliştiricilere hataları erken yakalama avantajı sağlar.

Kaynak kodu analizi araçları, **Statik Uygulama Güvenlik Testi (SAST)** araçları olarak da bilinir, kaynak kodunuzu güvenlik açıklarını bulmak için analiz eder. SAST araçları, IDE'nize eklenebilir. Yazılım geliştirme sürecinde sorunları tespit etmenize yardımcı olabilir ve özellikle geliştirme döngüsünün ilerleyen aşamalarında güvenlik açıklarını bulmaya kıyasla zaman ve çaba tasarrufu sağlayabilir.

Bir sonraki bölümde, sürekli kod kalitesi değerlendirmesi için SonarSource tarafından oluşturulmuş açık kaynaklı bir platform olan **SonarQube**'yi kullanacaksınız. Proje dallarınız ve çekme istekleriniz arasında sürekli kod denetimi sağlamak için mevcut iş akışınıza entegre edilebilir.

Başlarken

CI/CD hattınızda özel bir SonarQube aşaması kurulu olsa da, SonarQube'u yerel olarak çalıştırmayı bilmek faydalıdır. Bunu yaparak, kodu değiştirme, tarama yapma ve sonuçları görüntüleme geri bildirim döngüsünü kısaltabilirsiniz.

Bu laboratuvar boyunca, SonarQube'u kurma ve Docker ile Cloud IDE'deki bir uygulamanın analizini yürütme konusunda rehberlik alacaksınız. Her şey sağ paneldeki terminalde yapılabilir. Docker yüklü olan herhangi bir ortamda bu laboratuvarı kolayca tekrarlayabilmelisiniz, bu da kendi geliştirici çalışma istasyonunuzu içerir.

Bir SonarCube sunucusu kurmak için şunları yapacaksınız:

1. SonarQube ve PostgreSQl'in iletişim kurması için bir Docker ağı oluşturun
2. Docker konteynerinde PostgreSQL'i çalıştırın
3. Docker konteynerinde SonarQube'u çalıştırın
4. SonarQube tarayıcı Docker görüntüsünü indirin
5. `sonar-scanner-cli` Docker konteynerini kullanarak tarama yapmak için bir takma ad oluşturun
6. Bazı kodları tarayın ve sonuçları yorumlayın

Başlamak için **İleri**'ye tıklayın.

Adım 1: PostgreSQL veritabanını kurun

SonarQube, doğru çalışmak için bir veritabanına bağımlıdır. SonarQube, bir veritabanı olmadan bir docker konteynerinde çalışabilse de, konteyner silindiğinde tüm veriler de silinir. Bu, hızlı geliştirme taramaları için uygun olabilir, ancak zamanla bir geçmiş oluşturmak istiyorsanız, yerel geliştirme için bile harici bir veritabanı kullanmak en iyi uygulamadır.

Bu laboratuvar çalışmasında, PostgreSQL veritabanını docker imajını çekerek kullanacağız. Bunu yapmadan önce, veritabanını SonarQube ile ve daha sonra SonarScanner ile bağlamak için bir Docker ağına ihtiyacımız olacak. Bu, ilk göreviniz olacak.

Göreviniz

1. Üst menü çubuğundan `Terminal -> Yeni Terminal` seçeneğini seçerek bir terminal açın.
2. Ardından, terminal kabuğundan **docker ağı** oluşturmak için `docker network` komutunu çalıştırın ve bu ağa `mynet` adını verin.

   ```
   docker network create mynet
   ```

   Oluşturduğunuz docker ağı `mynet` olarak adlandırılacak ve bu, daha sonra oluşturacağınız PostgreSQL konteyneri ile SonarQube konteyneri arasında iletişim kurmak için kullanılacaktır.
3. Son olarak, bir PostgreSQL docker konteyneri oluşturmak için `docker run` komutunu kullanın:

   ```
   docker run --name postgres  -e POSTGRES_USER=root -e POSTGRES_PASSWORD=Test12345  -p 5432:5432 --network mynet -d postgres
   ```

   Konteyneri çalıştırma isteğinde bulunarak, Docker'ın imajı indirmesi gerektiğini anlayacağını göreceksiniz. `postgres` için Docker imajının katmanlarını indirme ve genişletme hakkında birçok çıktı görmelisiniz.

Sonuçlar

Son çıktınız aşağıdaki gibi görünmelidir:

```
theia@theiadocker-rofrano:/home/project$ docker run --name postgres  -e POSTGRES_USER=root -e PO
45  -p 5432:5432 --network mynet -d postgres
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
7a6db449b51b: Pull complete
b4f184bc0704: Pull complete
606a73c0d34a: Pull complete
c39f1600d2b6: Pull complete
31f42f92b0fe: Pull complete
c8b67d2b0354: Pull complete
31107b8480ee: Pull complete
b26434cf8bfa: Pull complete
36220bd76bfa: Pull complete
b79e75c4a0c2: Pull complete
cc1ab699dda5: Pull complete
37312064dd9b: Pull complete
4bce56fcbfe5: Pull complete
Digest: sha256:befb4cdc1d944bd89784b9caa287cf025f0720f9a02436038124163accd177dc
Status: Downloaded newer image for postgres:latest
6010f98f048f2eb59a066b887d6d96b6e60898328793f899c99848f3d613a277
```

Adım 2: SonarQube sunucusunu kurma

PostgreSQL veritabanımız çalıştığına göre, SonarCube sunucusunu oluşturabilir ve veritabanına bağlayabiliriz. SonarQube'u kurmak ve çalıştırmak için uygun Java ortamına sahip olmak zorunda kalmamak için, SonarQube sunucunuzu çalıştırmak için bir Docker konteyneri kullanacaksınız. Neyse ki, kullanabileceğiniz bir tane SonarQube sağlıyor.

Göreviniz

Aşağıdaki `docker run` komutunu kullanarak `9000` portunda bir SonarQube docker konteyneri çalıştırın:

```
docker run -d --name sonarqube -p 9000:9000 -e sonar.jdbc.url=jdbc:postgresql://postgres/postgres -e sonar.jdbc.username=root -e sonar.jdbc.password=Test12345 --network mynet sonarqube
```

Bu komut, SonarQube'un konteyner dışındaki iletişimi için `9000` portunu açmak amacıyla `-p` bayrağını kullanır. Ayrıca, `-e` komutuyla birkaç ortam değişkeni ayarlar. Docker kullanarak sunucuyu masaüstünüzde çalıştırmak, bir geliştirici olarak size muazzam bir özgürlük ve esneklik sağlar.

Yine, SonarQube docker imajının katmanlarının indirildiğini ve çıkarıldığını göreceksiniz. Nihai çıktınız aşağıdakine benzer bir şey olmalıdır:

```
theia@theiadocker-rofrano:/home/project$ docker run -d --name sonarqube -p 9000:9000 -e sonar.j
//postgres/postgres -e sonar.jdbc.username=root -e sonar.jdbc.password=Test12345 --network myne
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
9621f1afde84: Pull complete
0da9106727c7: Pull complete
129c5a3f9c32: Pull complete
Digest: sha256:3fa9a76948fab6fafa41950bee256afea943773744723b5e4f38b340643516b9
Status: Downloaded newer image for sonarqube:latest
0be9c3c828e62f5e83d2d236f0fe32ce04e18ef355e5f41008c5680c228bdda8
```

Başarıyı Kontrol Et

Artık hem PostgreSQL hem de SonarQube imajları indirilmiş ve her iki konteyner de başlatılmış olduğuna göre, çalıştıklarını kontrol etmek için `docker ps` komutunu kullanabilirsiniz.

Konteynerleri kontrol etmek için `docker ps` komutunu kullanın:

```
docker ps
```

`postgres` adında bir konteyner ve `sonarqube` adında bir konteyner ile aşağıdaki gibi bir çıktı görmelisiniz:

```
theia@theiadocker-rofrano:/home/project$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS
          NAMES
0be9c3c828e6   sonarqube   "/opt/sonarqube/bin/…"   4 minutes ago   Up 3 minutes   0.0.0.0:900
000/tcp   sonarqube
6010f98f048f   postgres    "docker-entrypoint.s…"   21 minutes ago  Up 20 minutes  0.0.0.0:543
2/tcp     postgres
```

Tebrikler! SonarQube'unuz artık çalışıyor!

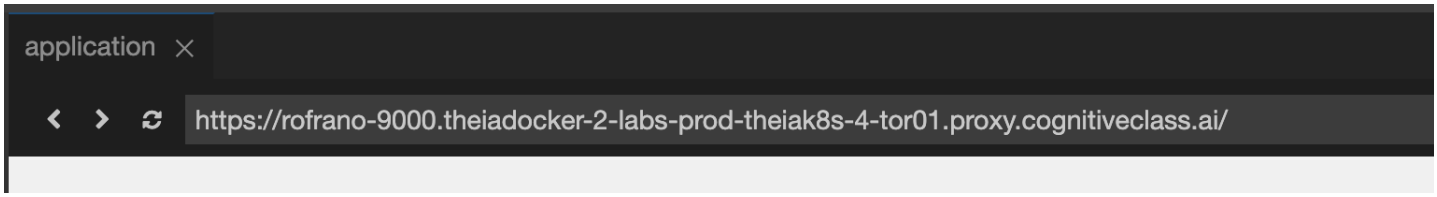
Artık kurulum kısmını tamamladınız ve projeniz için bir tarayıcı token'ı oluşturma aşamasına geçmeye hazırsınız!

Adım 3: SonarQube'a Giriş Yapın

Artık SonarQube'u kullanmaya hazırsınız. Aşağıdaki `[SonarQube UI'yi Başlat]` butonuna tıklayarak web arayüzünü başlatabilirsiniz. SonarQube'un başlaması biraz zaman alabilir.

SonarQube UI'yi Başlat

> Not: SonarQube UI oldukça büyük olduğu için, Cloud IDE yerine bu web sayfasını tarayıcınızda açmak isteyebilirsiniz. Cloud IDE tarayıcısını kullanarak gezinmek zor olabilir. **SonarQube UI'yi Başlat** butonuna tıkladıktan sonra, terminal panelinin sağ üst köşesinde yeni bir sayfa açacak bir ok simgesi olacak.



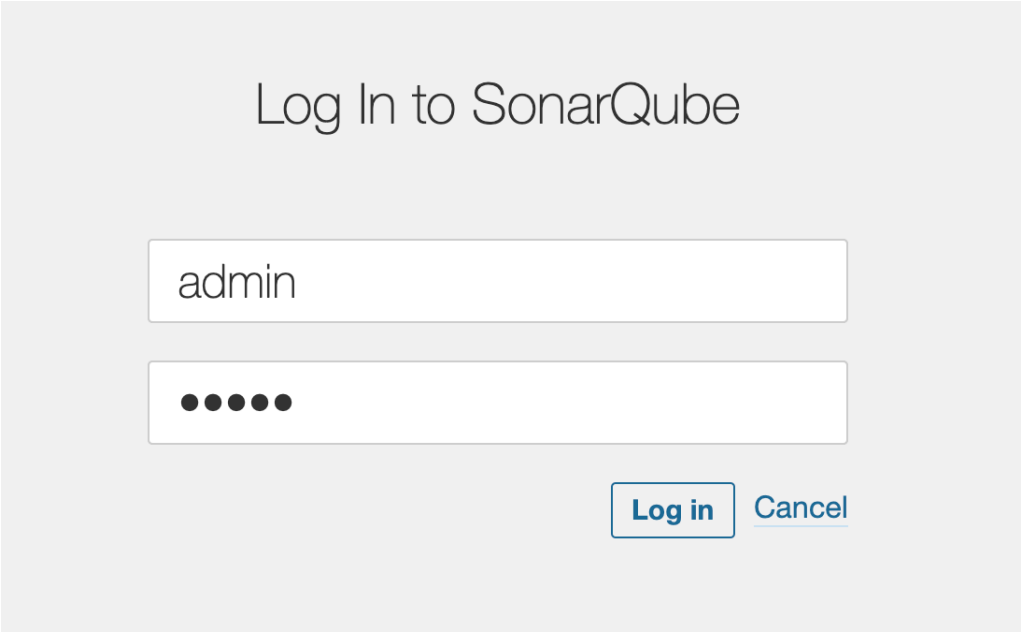
Kimlik bilgilerini değiştirin

SonarQube'a giriş yapmak için aşağıdaki varsayılan kimlik bilgilerini kullanın.

```
username: admin
password: admin
```

Sonra `[Log in]` butonuna tıklayın. Giriş yaptıktan sonra, şifrenizi değiştirmeniz istenecektir.

Not: Yeni şifrenizi kaydetmeyi unutmayın!

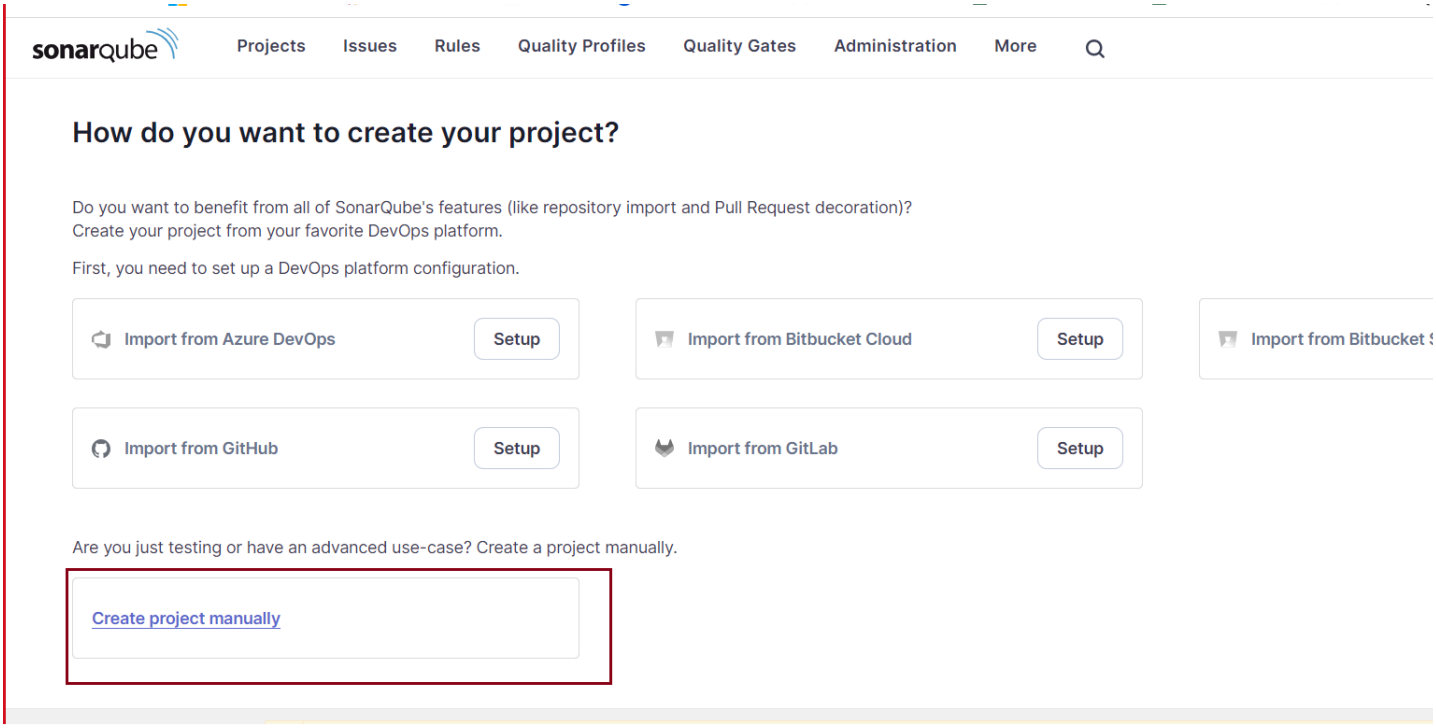


Şifreyi sıfırlama işlemini tamamladıktan sonra, sizi SonarQube'un ana sayfasına yönlendirecektir.

Adım 4: Bir SonarQube Projesi Oluşturun

Kodunuz üzerinde SonarQube tarayıcısını çalıştırmak için önce bir proje token'ı oluşturmanız gerekecek. Bir token oluşturmanın birçok yolu vardır, ancak bu laboratuvar için **manuel** kurulumu kullanacaksınız.

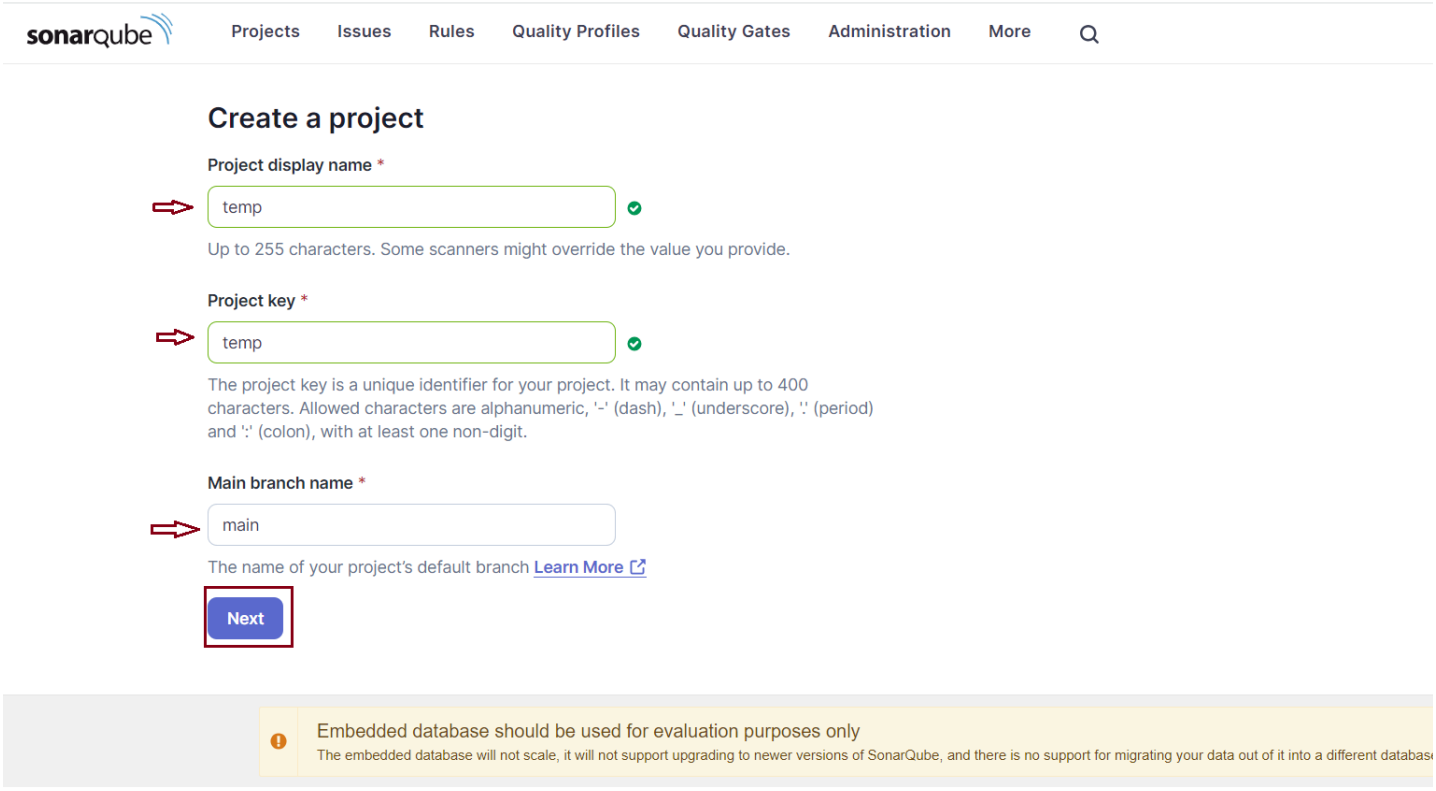
Sol alttaki **Manuel** simgesine tıklayın.



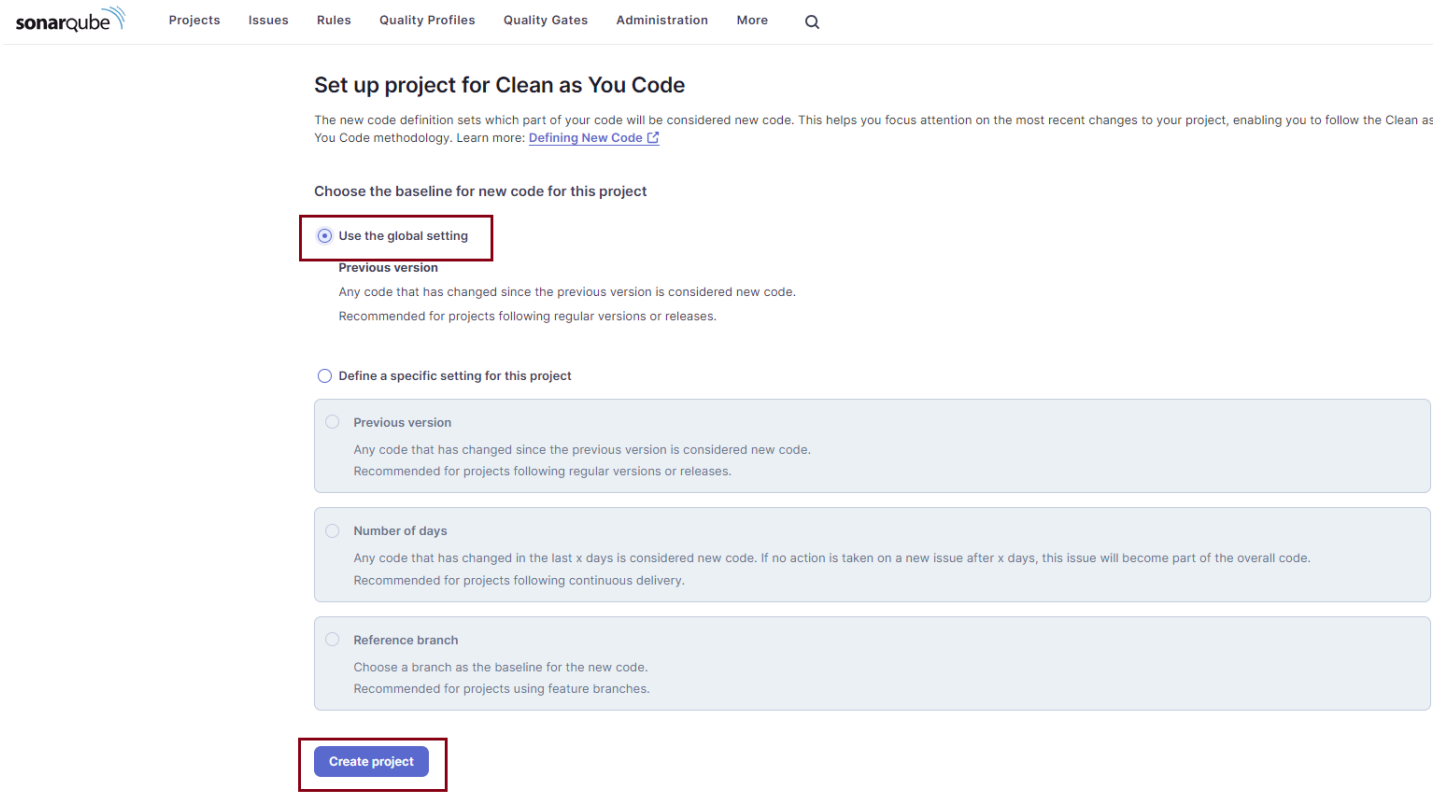
Sonraki sayfada, aşağıdaki adımları izleyerek bir proje oluşturun:

1. Proje görüntü adını `temp` olarak ayarlayın.
2. Proje anahtarını `temp` olarak ayarlayın (bu varsayılan olarak gerçekleşecektir).
3. `main` dalının seçili olduğundan emin olun.
4. Devam etmek için `İleri` butonuna basın.

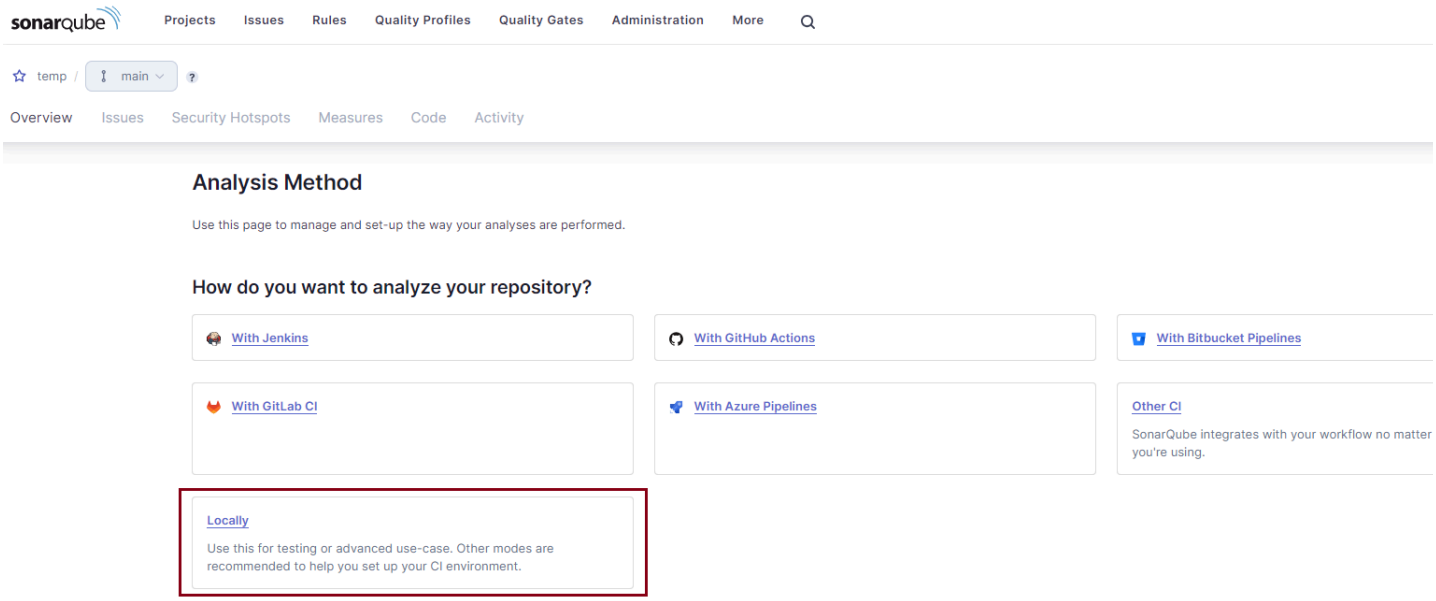
Not: Eğer bir hata mesajı alırsanız, tekrar deneyin.



Lütfen `Küresel ayarı kullan` seçeneğini seçin ve ardından `Proje oluştur` butonuna tıklayın.



Sonraki sayfada, deposunu nasıl analiz etmek istediğinizi soran bölümde `Yerel` seçeneğini seçin.

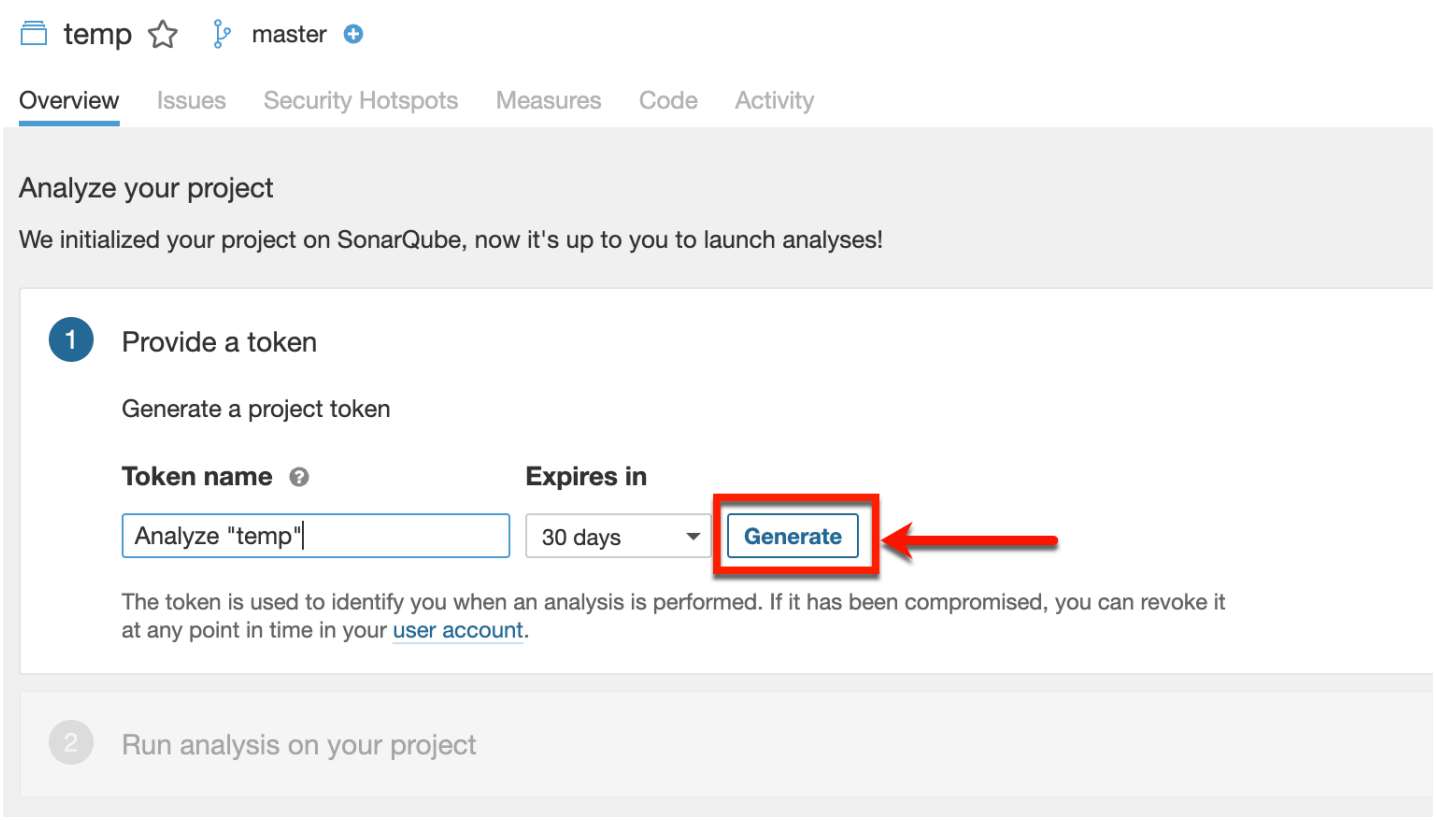


Sonraki adımda, projeniz için bir token oluşturacaksınız.

Adım 5: SonarQube Tarayıcı Token'ı Oluşturma

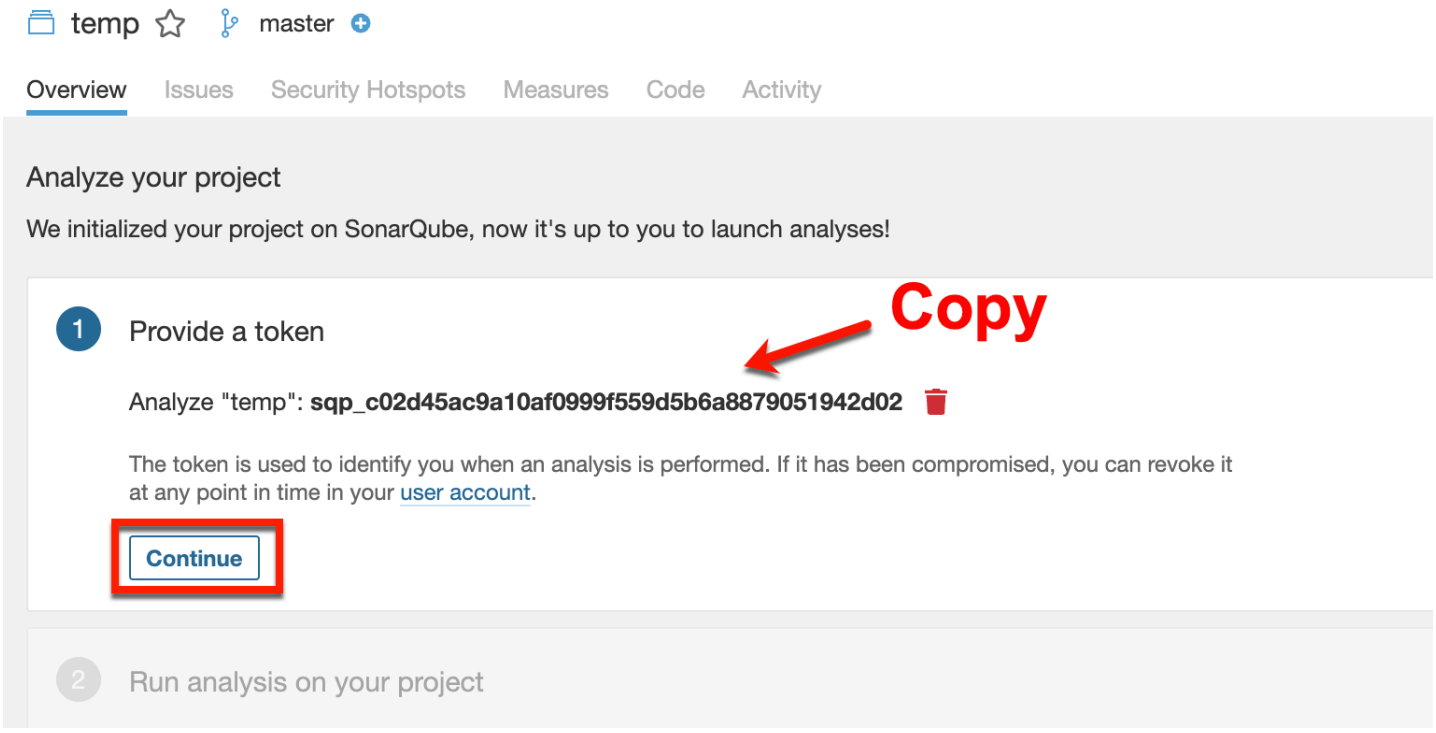
Kodunuzu taramadan önce bir token oluşturmanız gerekecek. Token'ı **Projenizi Analiz Edin** sayfasında **Token sağlayın** adımında oluşturabilirsiniz.

Oluştur butonuna tıklayın.

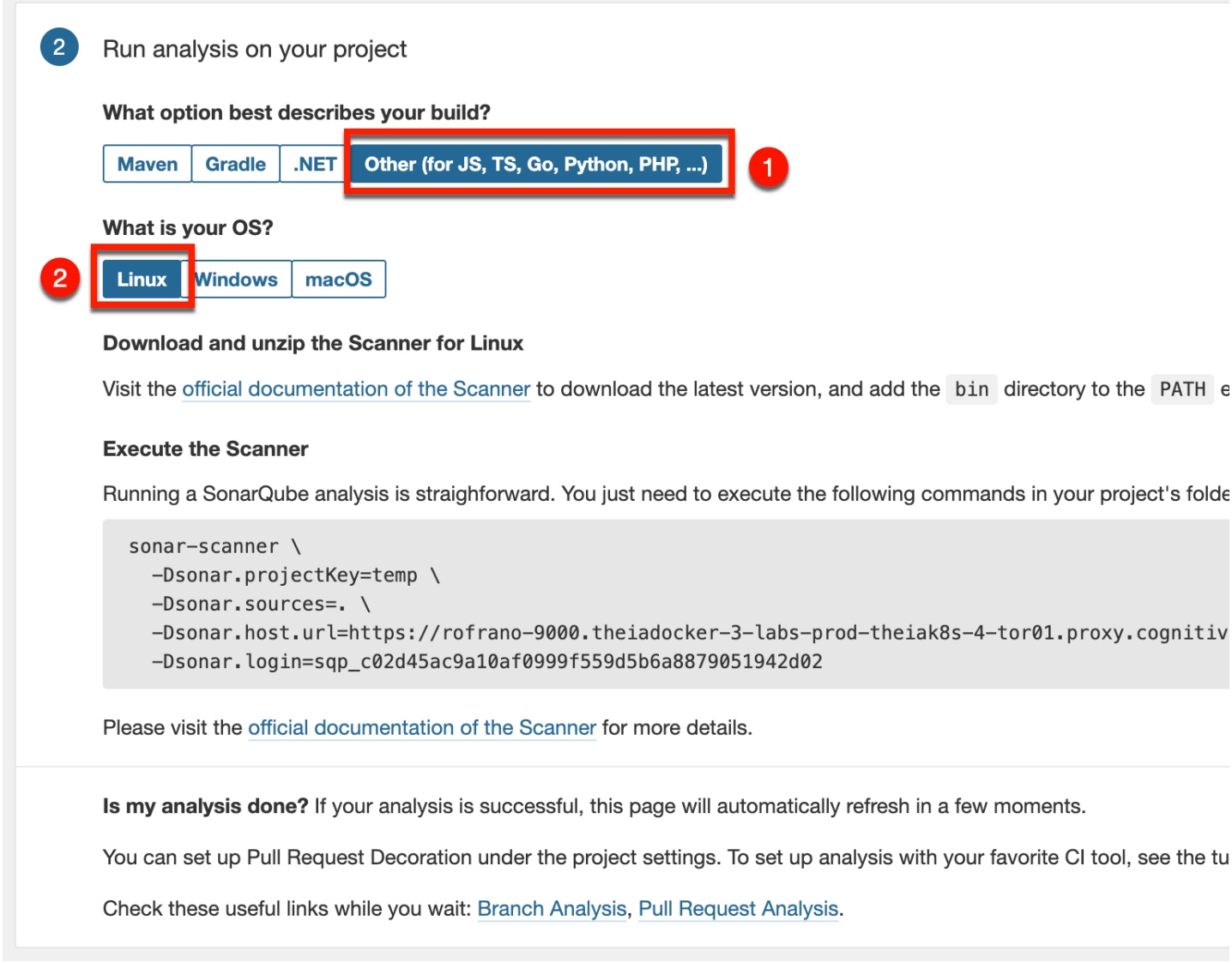


Sonra, oluşturulan token'ı göreceksiniz. Token metnini vurgulayıp kopyalayın ve güvenli bir yere yapıştırın. Daha sonra taramalarınızı göndermek için buna ihtiyacınız olacak.

Ardından **Devam** butonuna tıklayın.



Bu, projenizin yapılandırmasıyla ilgili bazı soruları yanıtlamanız gereken bir sayfaya yönlendirecek; örneğin, kullanılan dil ve işletim sistemi (OS).



Seçimler için şunları seçmelisiniz:

1. Diğer (JS, TS, Go, Python, PHP, …)
2. Linux
3. Kopyala

Önemli!

Bu komutu güvenli bir yere kopyalamak çok önemlidir! Kodunuz üzerinde tarayıcıyı çalıştırabilmeniz için oluşturulan komutu göreceksiniz. Bu komut projenize özeldir.

O komutu bir yere kaydettiğinizden emin olun çünkü gelecekteki bir adımda buna ihtiyacınız olacak!

Adım 6: SonarQube Tarayıcısını Hazırlama

SonarQube sunucusunun, tarama sonuçlarını depolayan ayrı bir sistem olduğunu ve gerçek taramayı gerçekleştiren SonarQube tarayıcısından farklı olduğunu anlamak önemlidir. Şimdiye kadar, analiz sonuçlarını depolamak için bir veritabanı oluşturduk ve kullanıcı arayüzünü sunmak için bir SonarQube sunucusu sağladık.

SonarQube tarayıcısını Cloud IDE'de çalıştırmak için, ya yerel olarak kurabilir ya da docker imajını çekip docker konteynerini çalıştırabilirsiniz. Bu laboratuvar çalışmasında, docker imajını çekecek ve docker konteynerini çalıştıracaksınız.

Göreviniz

1. Öncelikle, `docker pull` komutunu kullanarak Docker hub'dan `sonarsource/sonar-scanner-cli` imajını indirip yerel olarak kullanılabilir hale getireceğiz.

   ```
   docker pull sonarsource/sonar-scanner-cli
   ```

   Not: İmajı çekmezseniz, tarayıcıyı ilk çalıştırdığınızda otomatik olarak çekilecektir. Bunu şimdi yapıyoruz ki daha sonra zaman kazanmış olalım.

2. Terminalde aşağıdaki bash `alias` komutunu çalıştırın; bu, `scanner-cli` docker konteynerini kullanarak tarayıcıyı daha sonra çalıştırmak için `sonar-scanner` adında bir takma ad oluşturur:

   ```
   alias sonar-scanner='docker run --rm -v "$(pwd):/usr/src" sonarsource/sonar-scanner-cli'
   ```

Not: Bu komut, mevcut çalışma dizinini konteyner içinde `/usr/src` konumuna bir hacim olarak bağlıyor; `sonar-scanner` burada kaynak kodunu arıyor. Bunu kendi bilgisayarınızda da ayarlayabilirsiniz.

`sonar-scanner` komutuna geçirdiğiniz herhangi bir argüman, konteyner versiyonuna da iletilecektir. Bu sayede, Docker konteynerlerinde komutları, sanki bilgisayarınıza gerçekten kuruluymuş gibi kolayca çalıştırabilirsiniz.

Artık tarayıcıyı hazırladığımıza göre, üzerinde analiz yapacağımız bir proje edinelim!

Adım 7: Örnek Proje Alma

Taramanız için biraz koda ihtiyacınız var. Güvenlik açıklarını taramak için IBM CI/CD kursundan bir projeyi kullanacağız.

Terminalde, bir CI/CD python projesini GitHub depo adresinden klonlamak için `git clone` komutunu kullanın ve ardından `wtecc-CICD_PracticeCode` proje dizinine `cd` ile geçin.

```
git clone https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git
cd wtecc-CICD_PracticeCode
```

`ls -l` komutunu kullanarak dosyaları görüntüleyin:

```
ls -l
```

Aşağıdakileri görmelisiniz:

```
theia@theiadocker-rofrano:/home/project/wtecc-CICD_PracticeCode$ ls -l
total 44
-rw-r--r-- 1 theia users   491 Sep 12 19:41 Dockerfile
drwxr-sr-x 8 theia users  4096 Sep 12 19:41 labs
-rw-r--r-- 1 theia users 11357 Sep 12 19:41 LICENSE
-rw-r--r-- 1 theia users    72 Sep 12 19:41 Procfile
-rw-r--r-- 1 theia users   957 Sep 12 19:41 README.md
-rw-r--r-- 1 theia users   327 Sep 12 19:41 requirements.txt
drwxr-sr-x 3 theia users  4096 Sep 12 19:41 service
-rw-r--r-- 1 theia users   331 Sep 12 19:41 setup.cfg
drwxr-sr-x 2 theia users  4096 Sep 12 19:41 tests
theia@theiadocker-rofrano:/home/project/wtecc-CICD_PracticeCode$
```

Artık bazı taramalar yapmaya hazırsınız.

Adım 8: Tarayıcıyı Çalıştırma

Artık kod üzerinde statik analiz yapmak için ihtiyacınız olan her şeye sahipsiniz. Terminalde, “Adım 5: SonarQube Token’ı Oluştur”dan kaydettiğiniz komutu çalıştırın. Aşağıdakine benzer görünmelidir:

```
### THIS IS AN EXAMPLE ONLY ### DO NOT PASTE THIS ###
sonar-scanner \
   -Dsonar.projectKey=temp \
   -Dsonar.sources=. \
   -Dsonar.host.url=https://{YOUR SONARQUBE URL} \
   -Dsonar.login={YOUR PROJECT TOKEN}
```

Göreviniz

1. Adım 5’te oluşturduğunuz SonarQube komutunu buraya yapıştırın. Komutunuz, taramanın çalışabilmesi için proje jetonunu da içeren tüm parametreleri içermelidir. Yukarıdaki örnek kodu kullanmayın.
2. Enter tuşuna bastığınızda, tarayıcı mevcut proje dizininizde statik bir analiz başlatır ve bir süre çalışır.

Tarama tamamlandığında, çıktının sonu aşağıdaki gibi görünmelidir:

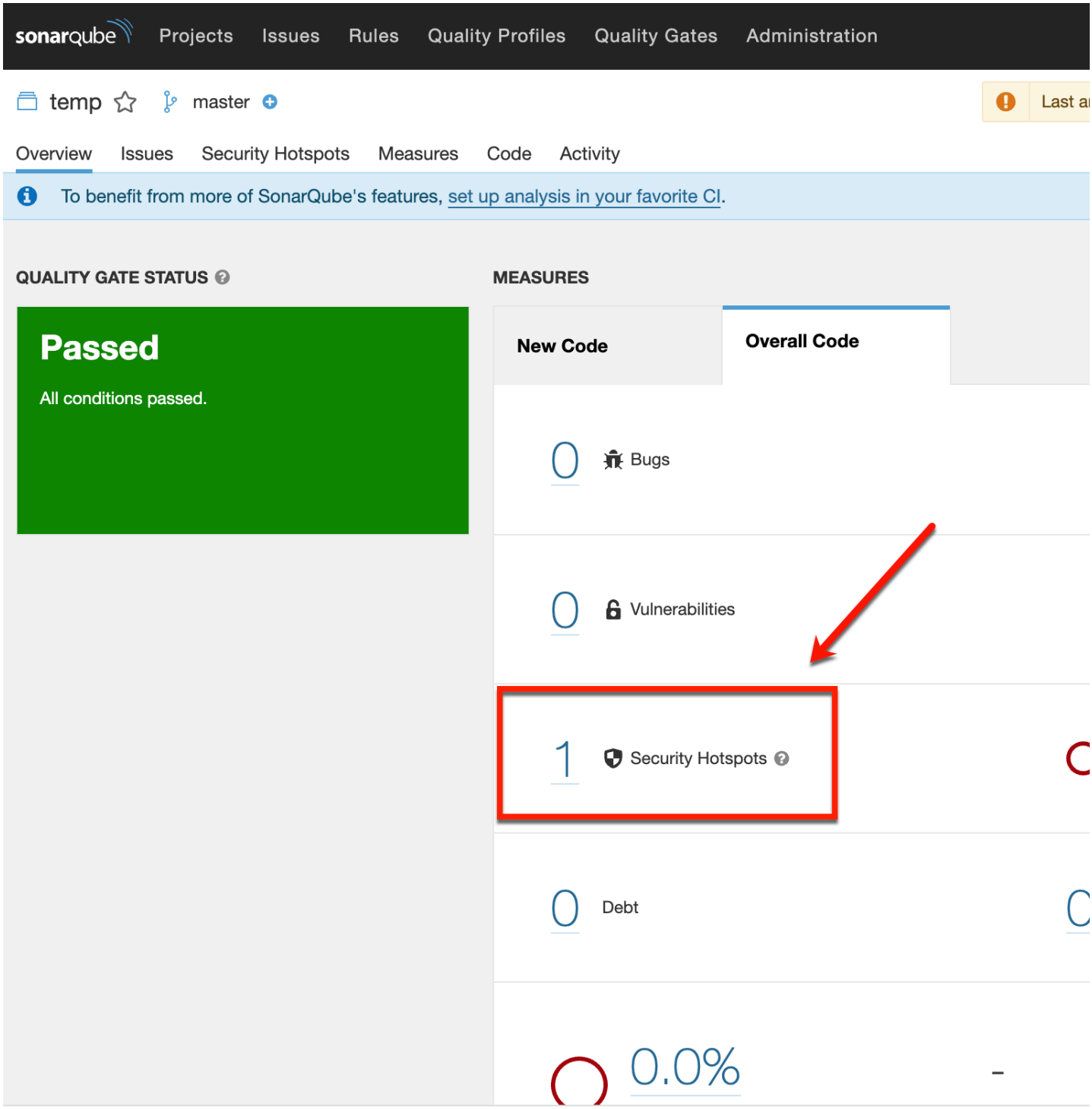
```
roxy.cognitiveclass.ai/dashboard?id=temp
INFO: Note that you will be able to access the updated dashboard once the server has processed
eport
INFO: More about the report processing at https://rofrano-9000.theiadocker-3-labs-prod-theiak8s
class.ai/api/ce/task?id=AYMzJIZaolQa5I6QBptT
INFO: Analysis total time: 9.773 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 30.262s
INFO: Final Memory: 16M/57M
INFO: ------------------------------------------------------------------------
theia@theiadocker-rofrano:/home/project/sampleproject$
```

Adım 9: Tarama sonuçlarını yorumlama

Kod taraması ve analizi tamamlandığında, sonuçları SonarQube arayüzünde görebilirsiniz. Sonuçları yeni bir tarayıcı penceresinde görmek daha kolay olabilir. Sonuçlarla birlikte yeni bir tarayıcı penceresi açmak için `[SonarQube UI'yi Tarayıcıda Başlat]` butonuna basabilirsiniz.

SonarQube UI'yi Tarayıcıda Başlat

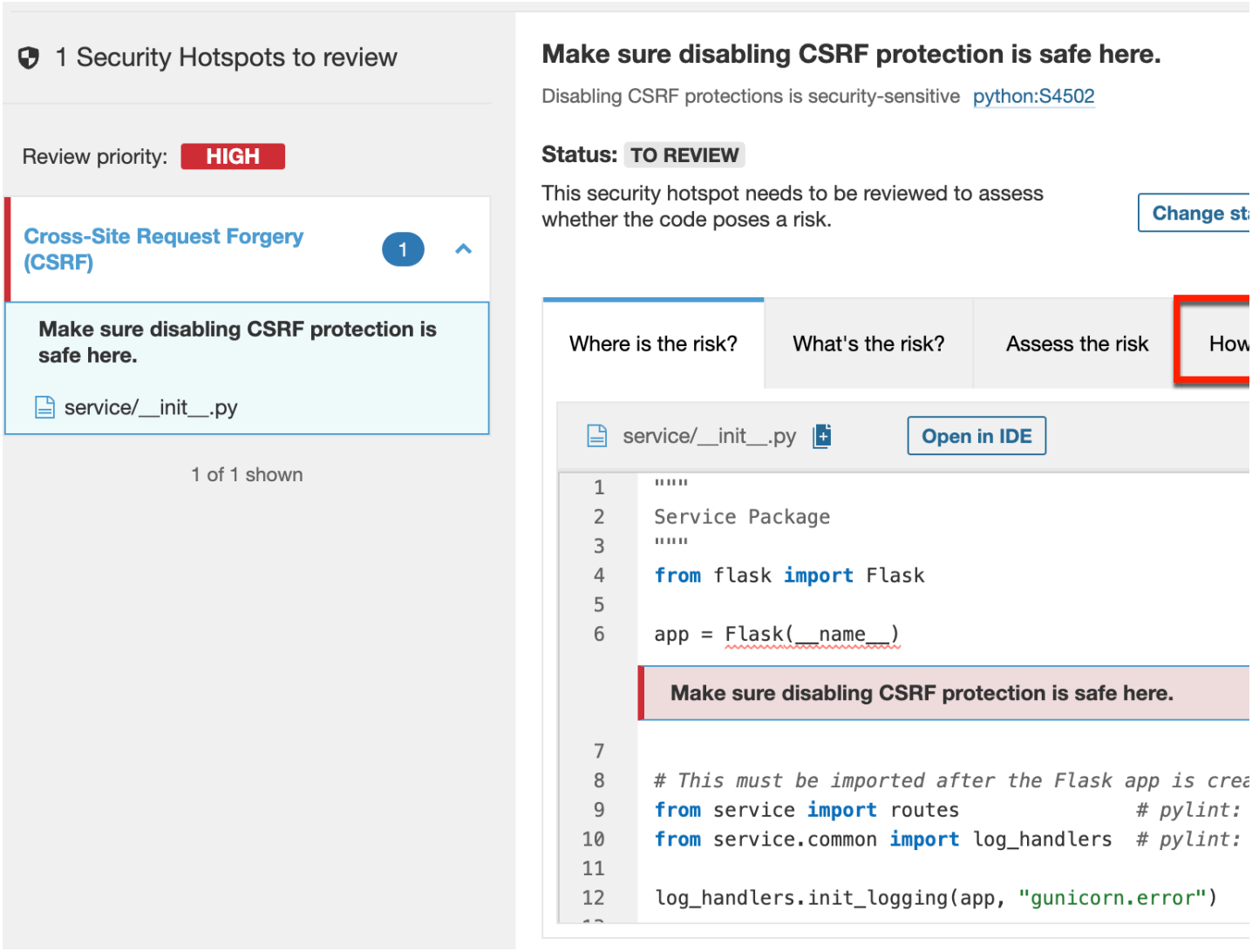
Raporu görüntülediğinizde, genel raporun geçerli olduğunu ancak bir güvenlik maddesinin işaretlendiğini göreceksiniz.



Güvenlik Sıcak Noktaları yanındaki **1** bağlantısına tıkladığınızda bununla ilgili bazı bilgileri göreceksiniz.

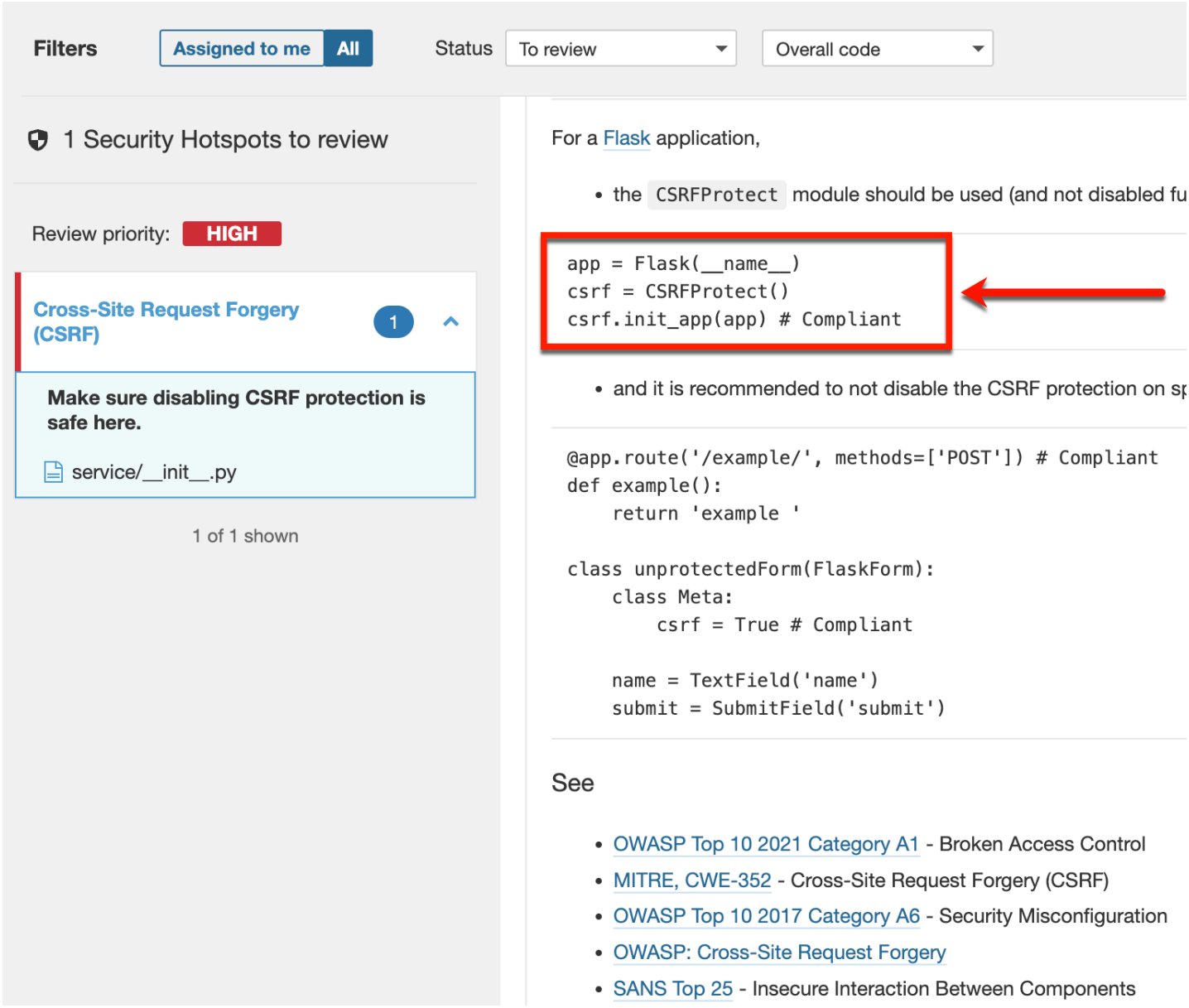
Üst sekmelerde aşağıdaki etiketleri görebilirsiniz:

- **Risk nerede?**
- **Risk nedir?**
- **Riski değerlendirin**
- **Bunu nasıl düzeltebilirim?**



Risk nerede? sekmesinin altında, kullandığımız örnek kodun uygun güvenlik önlemlerini içermediği için potansiyel bir Cross Site Request Forgery (CSRF) riski olduğunu belirtiyor.

Bir sonraki sorunuz "Bunu nasıl düzeltebilirim?" olabilir. Bunu öğrenmek için **Bunu nasıl düzeltebilirim?** sekmesine tıklayabilirsiniz.



Bunu nasıl düzeltebilirim? sekmesi, dikkat etmeniz gereken bazı noktaları veriyor.

Bu uygulama Flask framework'ü kullanılarak yazılmıştır. Flask ile ilgili bölümüne indiğinizde, sorunu düzeltmek için `CSRFProtect` sınıfını nasıl kullanacağınızı tam olarak anlatıyor ve bazı diğer tavsiyeler veriyor.

```
app = Flask(__name__)
csrf = CSRFProtect()
csrf.init_app(app) # Compliant
```

Eğer bu sizin orijinal kodunuz olsaydı, önerilen değişiklikleri uygulamanız ve sorunun düzeldiğinden emin olmak için taramayı tekrar çalıştırmanız gerekirdi.

Sonuç

Tebrikler! Statik analiz üzerine bu laboratuvarı tamamladınız, bu da güvenli uygulama geliştirme sürecinin ayrılmaz bir adımıdır. Artık uygulamalarınızı daha güvenli hale getirmek için onlara statik analiz güvenlik taramaları yapma yolunda ilerliyorsunuz.

Statik analizin bir projedeki zayıflıkları tespit etmek için nasıl kullanılabileceğini anlıyorsunuz. Ayrıca, statik analizler gerçekleştirmek için SonarQube gibi açık kaynak araçlarla nasıl başlayacağınızı da biliyorsunuz.

Sonraki Adımlar

Farklı türdeki zayıflıkları tespit etmek, güvenli uygulama geliştirmedeki ilk adımlardan biridir. Ayrıca, bu zayıflıkların arkasındaki anlamı anlamanız gerekir ki doğru eylemleri alabilin. Öğrenmenin en iyi yolu, uygulamaktır.

Bir sonraki meydan okumanız, geliştirme ortamınızda SonarQube'u kurmak, kodunuza güvenlik taramaları yapmak ve bulduğu sorunları düzeltmektir. Daha güvenli kod yazma yolunda ilerliyorsunuz!

Author(s)

Roxanne Li
John J. Rofrano