SonarQube Dökümantasyon

Seyitahmet Genç

Ağustos, 2021

İçindekiler

1 Giriş

Bu belgede SonarQube platformunun ne olduğu, ne amaçlar için kullanıldığı ve SonarQube platformunun Java ve C dilleri ile nasıl kullanılabileceği anlatılmıştır.

2 SonarQube

SonarQube, SonarSource tarafından geliştirilen ve kod kalitesinin sürekli denetilmesi için kullanılan açık kaynak kodlu bir platformdur. Sonar; koddaki hatalar (bugs), kötü kokan kodlar (code smells), güvenlik açıkları (vulnerabilities) ve kod tekrarları (code duplications) hakkında ayrıntılı raporlar oluşturulabilmesini sağlayan statik bir kod analiz aracıdır.

25'ten fazla programlama dilini desteklemektedir. Bu destek yerleşik kural kümeleri (built-in rulesets) ile verilmektedir ve çeşitli eklentiler yardımıyla genişletilebilmektedir.

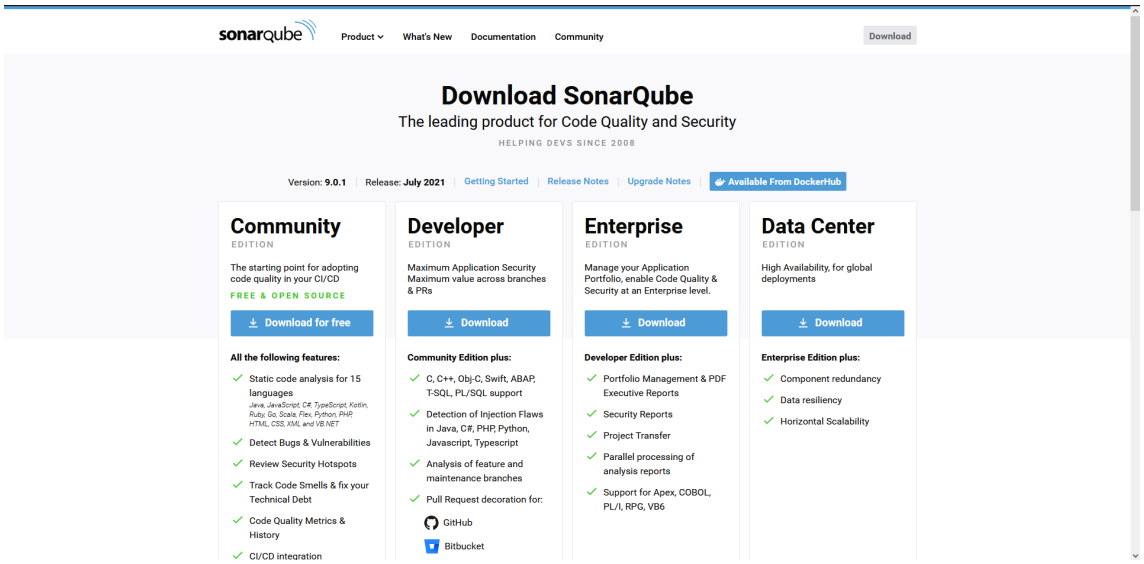
2.1 SonarQube Kullanmanın Faydaları

- **Sürdürülebilirlik**: Yazılan kodun karmaşıklığını, koddaki olası güvenlik açıklarını ve kod tekrarlarını tespit ettiğinden yazılan uygulamaların ömrünün optimize edilebilmesini sağlar.
- **Verimlilik**: Uygulamanın bakım maliyetini ve riskini azaltır; ölçeklenmesini kolaylaştırır. Bu sayede kod üzerinde değişiklik yapmak için harcanan zamanı azaltır.
- **Kaliteli Kod**: Yazılım geliştirme sürecinin çok önemli bir parçası olan kod kalite kontrolünün yapılmasını oldukça kolaylaştırır.
- **Hata Tespiti**: Kodun içerdiği hataların tespitini yapar ve uyarır, böylece yazılımcıların hatayı aramaktansa hatanın çözümü ile uğraşması sağlanarak zamandan tasarruf edilir.
- **Tutarlılık**: Belirlenen kriterlerin ihlal edildiği noktaları göstererek kod tabanındaki tutarlılığın artırılmasını sağlar.
- **Ölçeklenebilirlik**: Proje sayısında herhangi bir kısıtlama bulunmadığından kullanımı küçükten büyüğe her iş için ölçeklenebilirdir.
- **Yazılımcı Becerilerinin Geliştirilmesi**: Kalite problemleri üzerinde sürekli geri dönüş alan yazılımcılar bu problemleri düzelterek kendilerini geliştirebilmektedir.

2.2 Örnek Bir Kullanım

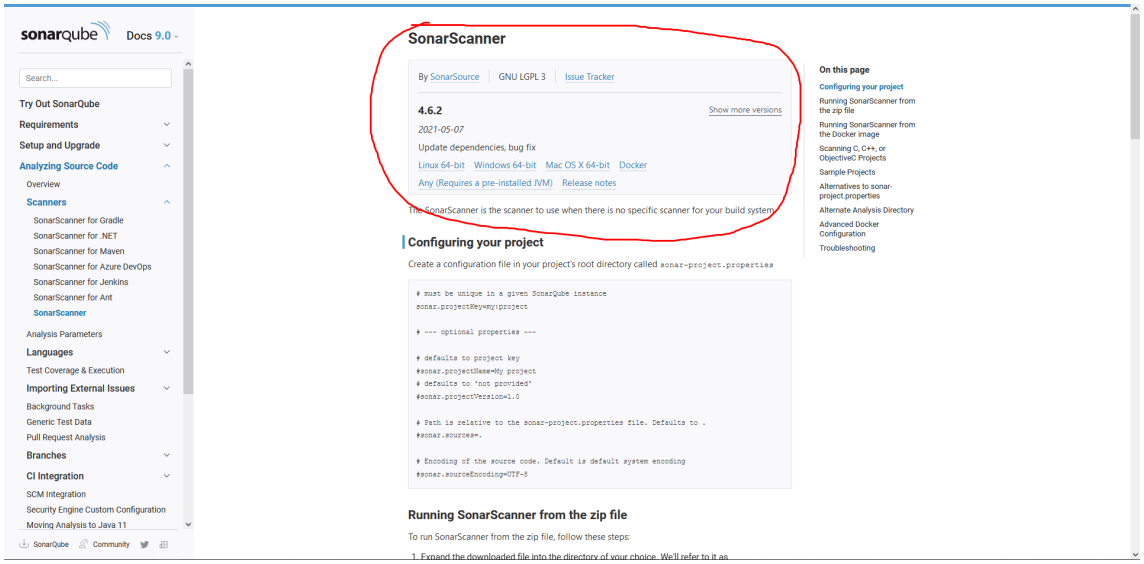
Bu bölümde SonarQube platformunun nasıl kullanılabileceğine dair bir örnek kullanım senaryosu anlatılmıştır.

- Bu adresten SonarQube indirilir. Adresin anlatım tarihindeki görünümü şekil 1 üzerinde gösterilmiştir.



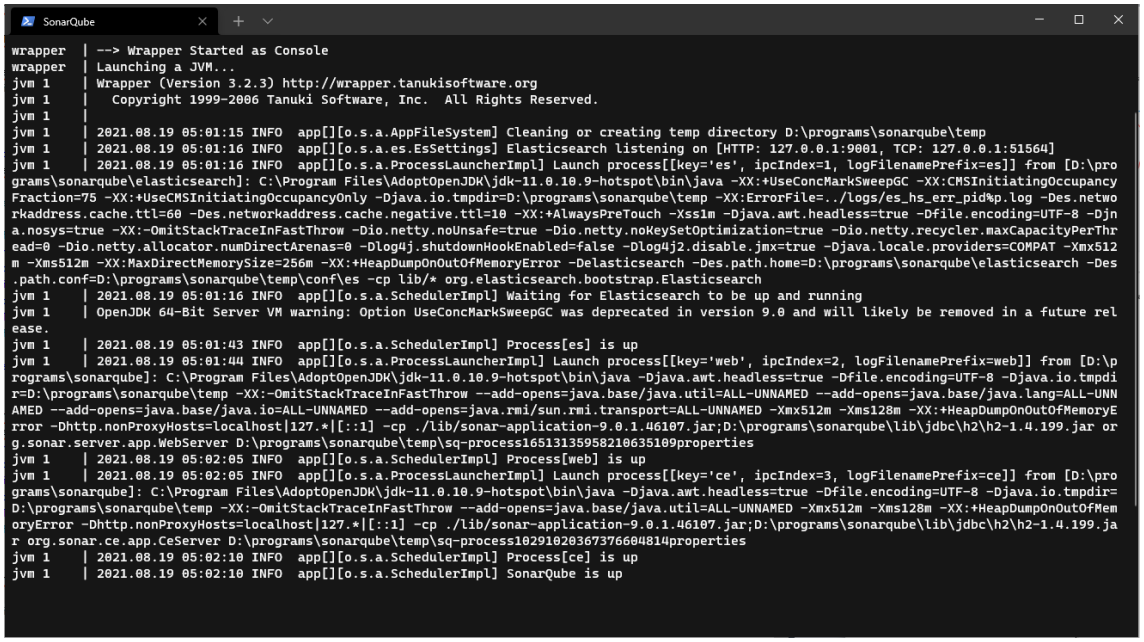
Şekil 1: SonarQube İndirme Sayfası

- Bu adresten SonarScanner de indirilir. Bu program ile analiz işlemi yapılmaktadır. Adresin anlatım tarihindeki görünümü şekil 2 üzerinde görülebilir.



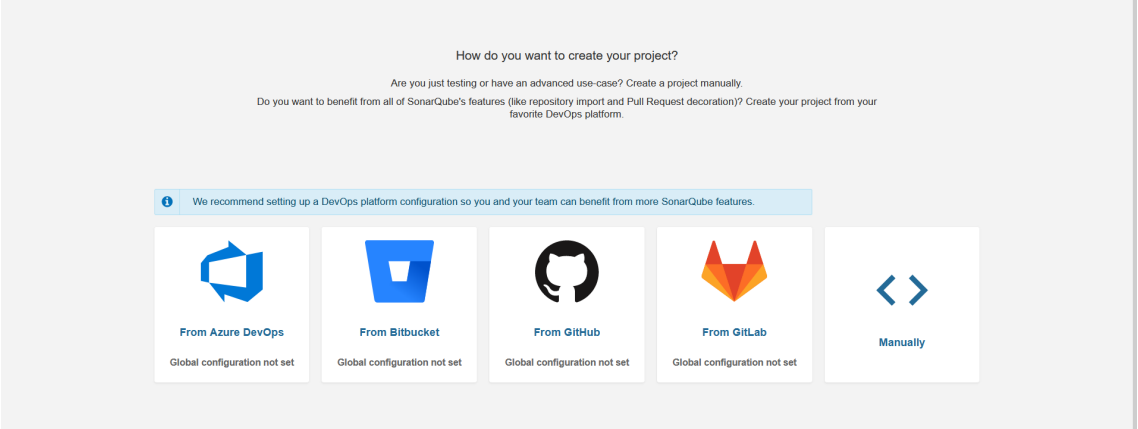
Şekil 2: SonarScanner İndirme Sayfası

- İndirilen SonarQube zip dosyası içerisinde gerekli klasördeki başlatıcı kullanılarak SonarQube başlatılır. Anlatım Windows üzerinden yapıldığından bu senaryodaki dizin "sonarqube bin windows-x86-64" dizini, başlatıcı ise "StartSonar.bat" dosyasıdır. Komutun çalışmasına ait örnek bir çıktı şekil 3 üzerinde verilmiştir.



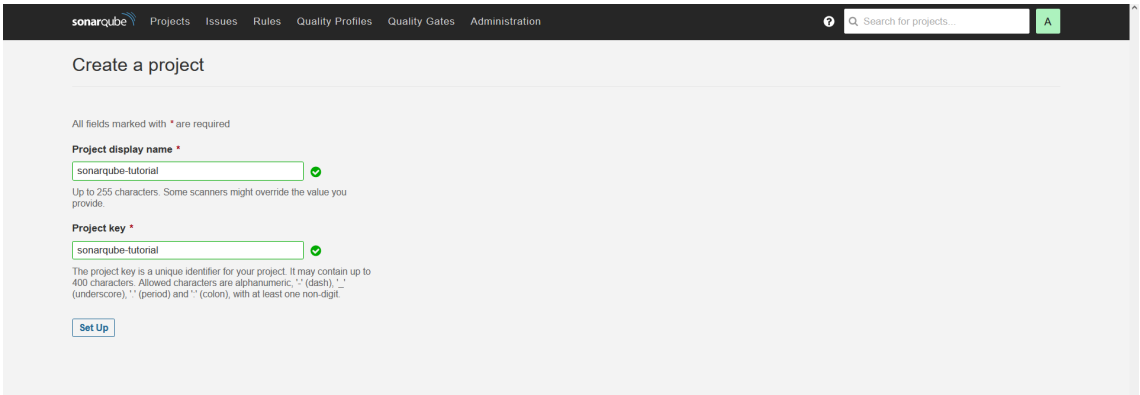
Şekil 3: StartSonar.bat Dosyasının Çalıştırılması

"SonarQube is up" yazısı görüldükten sonra tarayıcı üzerinden "localhost:9000" adresine gidilir. Bu adres ile SonarQube platformunun yönetim sayfasına giriş yapılmaktadır. Varsayılan kullanıcı adı ve şifre "admin"dir. İlk girişte kullanıcının şifreyi değiştirmesi istenmektedir. Giriş yapıldıktan sonra şekil 4 üzerinde görülen gibi bir sayfa kullanıcıyı karşılamaktadır.



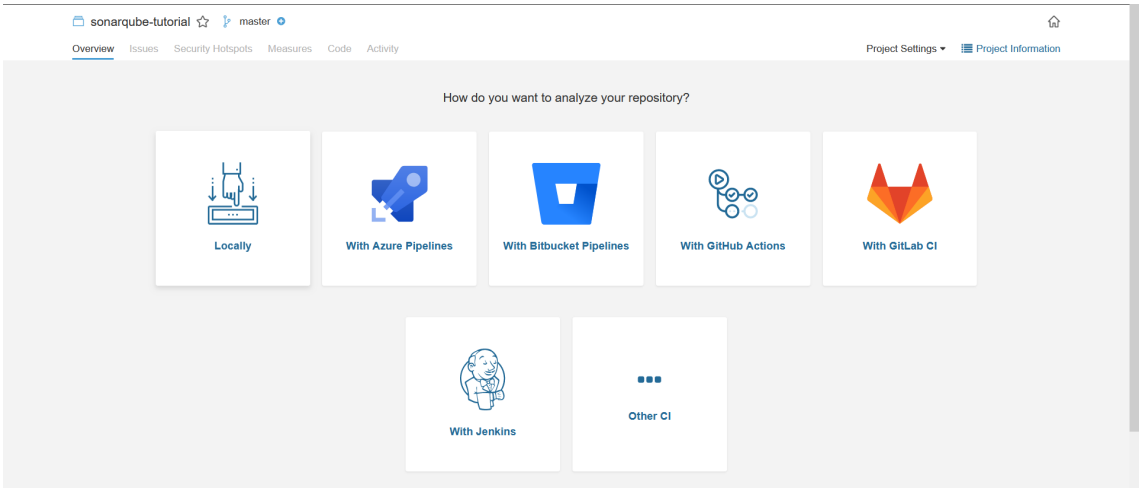
Şekil 4: SonarQube Platformu - Ana Sayfa

- Şekil 4 üzerinde görülen seçeneklerden "Manually" seçilerek yeni bir proje oluşturulur. Proje oluşturulmasında karşılaşılan ilk sayfa şekil 5 üzerinde gösterilmektedir. Sayfada görüldüğü şekilde oluşturulacak projeye bir isim ataması yapılması istenmektedir. Gerekli bilgiler girildikten sonra "Set Up" butonu ile devam edilmelidir.



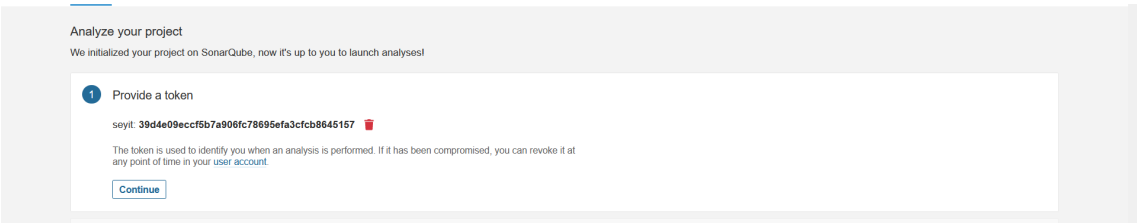
Şekil 5: SonarQube Platformu - Proje Oluşturma

- Proje oluşturulduktan sonra karşılaşılan ekran üzerinden projenin nasıl analiz edileceği seçimi yapılmalıdır. Örnek senaryoda bilgisayar üzerinde bulunan bir kod inceleneceğinden "Locally" seçeneği ile devam edilmelidir. Sayfaya ait örnek bir görünüm şekil 6 üzerinde verilmiştir.



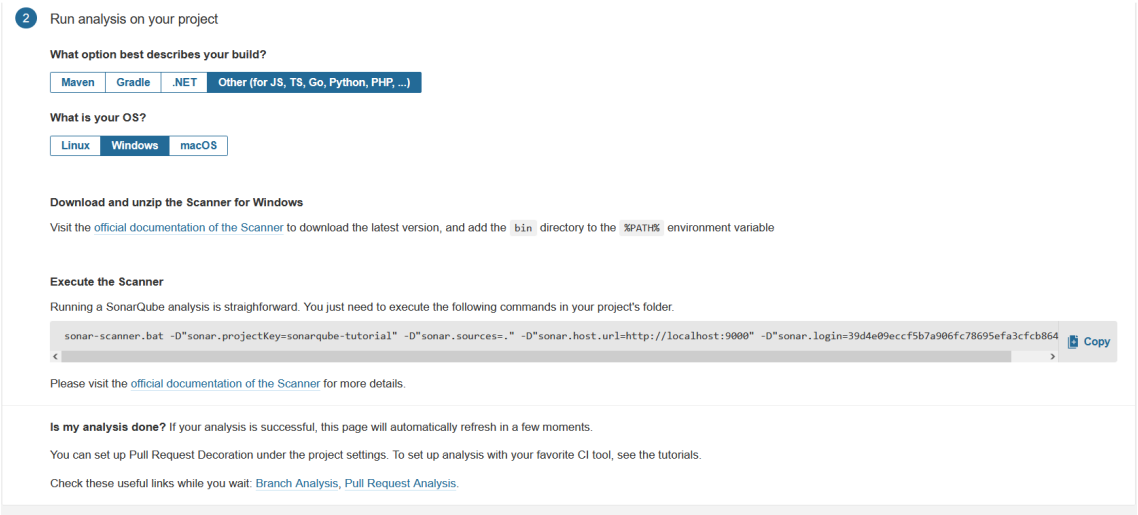
Şekil 6: SonarQube Platformu - Analiz Seçenekleri

- Locally seçeneği seçildikten sonra kullanıcı iki adımlı son bir aşama ile karşılaşmaktadır. Bunlardan ilki, proje analizinde kimlik belirtiminde kullanılan "token" bilgisidir. Tokenler, bir analiz gerçekleştiğinde bu analizi kimin yaptığının ayırt edilebilmesi için kullanılan stringlerdir. Örnek senaryoda SonarQube kullanımına yeni başlandığı için "Generate a token" seçeneği seçilerek yeni bir token oluşturulur ve sonraki aşamaya geçilir. İşleme ait örnek bir ekran görüntüsü şekil 7 üzerinde görülebilir. Oluşturulan tokenin kullanıcıya gösterildiği tek yer burasıdır, aynı tokeni daha sonra kullanabilmek için tokenin kaydedilmesi unutulmamalıdır.



Şekil 7: SonarQube Platformu - Token Oluşturma

Token oluşturulduktan sonra sıradaki aşamaya geçilir. Bu aşamada analiz edilecek proje ile ilgili sorular sorulmaktadır (projenin çalışacağı platform ve işletim sistemi gibi). Sorulan sorular cevaplandıktan sonra, sorulara verilen cevaplar doğrultusunda projenin nasıl analiz edilebileceği kullanıcıya anlatılmaktadır. Örnek kullanım senaryosunda yerel ve küçük bir Java dosyası incelenecektir. Dolayısıyla yapılan seçimler sırasıyla "Other(...) ve "Windows" olmuştur. İşleme ait örnek bir ekran görüntüsü şekil 8 üzerinde verilmiştir.



Şekil 8: SonarQube Platformu - Proje Analizi Yardımı

- Önceki aşamaların tamamlanması ile proje analiz edilmeye hazır hale gelmiştir. Şekil 8 üzerinde görülen "Execute the Scanner" başlığı altında verilen komut projenin olduğu dizin üzerinde kullanılarak proje analiz edilebilmektedir. Örnek kullanım senaryosunda, kullanıcının masaüstü dizinindeki sonarqube dizini içerisindeki InsertionSort.java adlı kodun analizi gerçekleştirilecektir. Analiz edilecek kod şekil 9 üzerinde gösterilmiştir.

```java
// Java program for implementation of Insertion Sort
// https://www.geeksforgeeks.org/insertion-sort/
public class InsertionSort {
  /* Function to sort array using insertion sort */
  void sort(int arr[]) {
    int n = arr.length;
    for (int i = 1; i < n; ++i) {
      int key = arr[i];
      int j = i - 1;

      /*
       * Move elements of arr[0..i-1], that are greater than key, to one position
       * ahead of their current position
       */
      while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;
      }
      arr[j + 1] = key;
    }
  }

  /* A utility function to print array of size n */
  static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
      System.out.print(arr[i] + " ");

    System.out.println();
  }

  // Driver method
  public static void main(String args[]) {
    int arr[] = { 12, 11, 13, 5, 6 };

    InsertionSort ob = new InsertionSort();
    ob.sort(arr);

    printArray(arr);
  }
} /* This code is contributed by Rajat Mishra. */
```

Şekil 9: Analiz Edilecek Kod

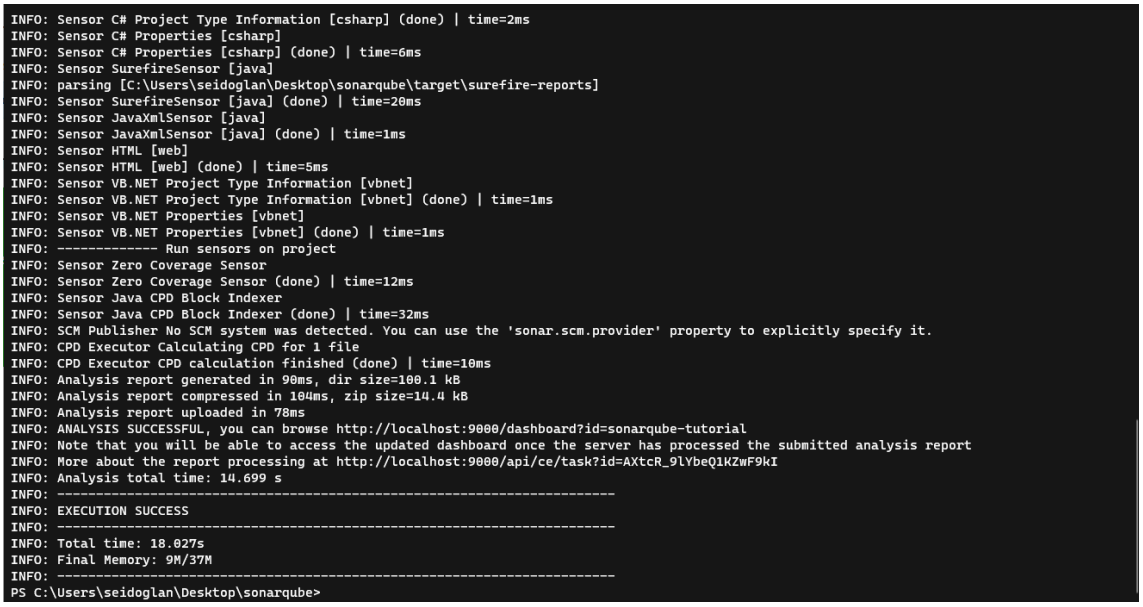
Örnek kullanım senaryosu için verilen kodun analizinde kullanılacak komut şekil 8 üzerinde de görülebildiği gibi **"sonar-scanner.bat**

-D"sonar.projectKey=sonarqube-tutorial"

-D"sonar.sources=."

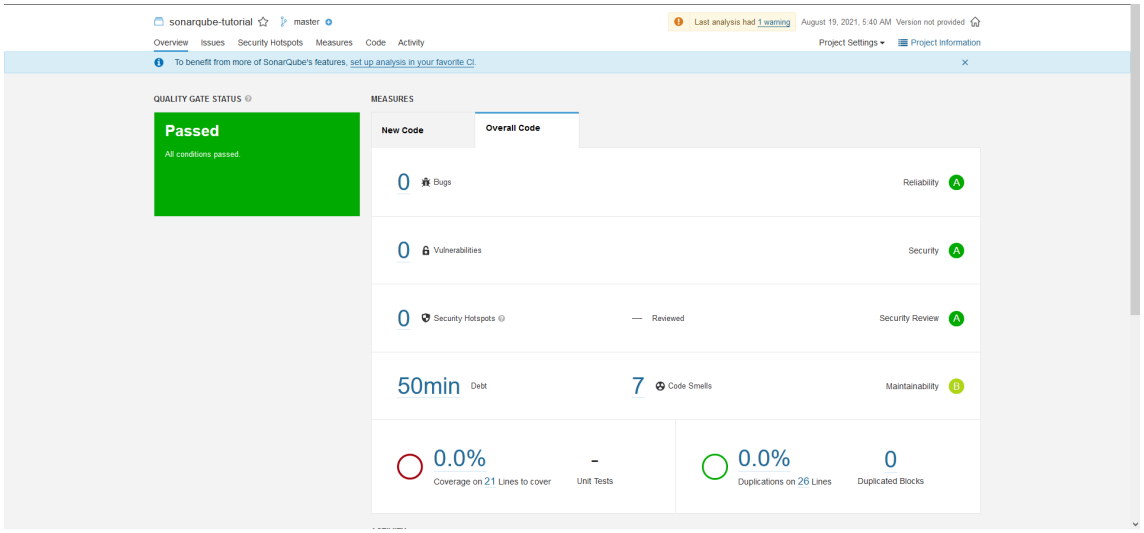
-D"sonar.host.url=http://localhost:9000"

-D"sonar.login=39d4e09eccf5b7a906fc78695efa3cfcb8645157"" komutu olacaktır. Komutun çalıştırılmasına ait örnek bir çıktı şekil 10 üzerinde görülebilmektedir.



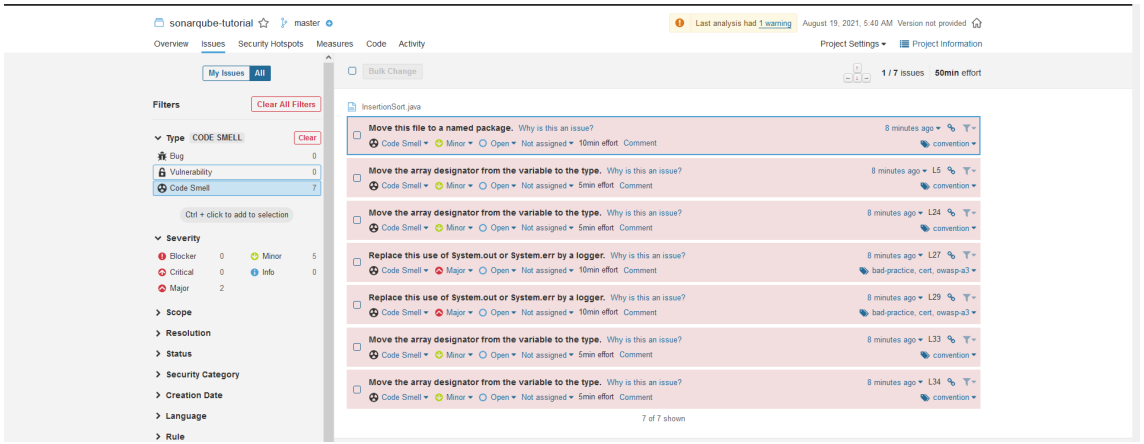
Şekil 10: Analiz İşleminin Gerçekleştirilmesi

Analiz işlemi tamamlandıktan sonra tarayıcı üzerinde kullanılan sayfa yenilenecek ve analiz işleminin sonucunu gösterecektir. Sayfaya ait örnek bir görünüm şekil 11 üzerinde verilmiştir.



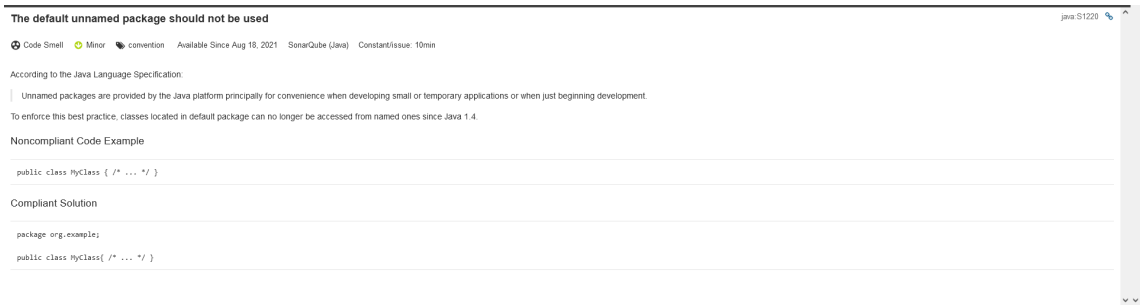
Şekil 11: SonarQube Platformu - Analiz Sonuçlarının Görüntülenmesi

Şekil 11 üzerinde görüldüğü gibi analiz tamamlandıktan sonra koddaki hatalar (bugs), kötü kokan kodlar (code smells) ve güvenlik açıkları (vulnerabilities) görülebilmektedir. Örnek senaryodaki analizde kodun 7 adet kötü kokan kısmı olduğu görülebilmektedir. Bunların incelenebilmesi için şekil 11 üzerinde görülen mavi yazılı "7"ye basılması gerekmektedir. Bu işlemden sonra karşılaşılan sayfaya ait örnek bir görünüm şekil 12 üzerinde görülebilmektedir.



Şekil 12: SonarQube Platformu - Koddaki Hataların Görüntülenmesi

Şekil 12 üzerinde görüldüğü gibi analiz sonucunda bulunan hatalar buradan görülebilmektedir. SonarQube ile bu hataların ne olduklarının yanı sıra bunların neden bir yanlış kabul edildiği, hangi türde bir yanlış olduğu ve bu yanlışın ne boyutta bir yanlış olduğu gibi bilgiler görülebilmektedir. Örneğin bulunan hataların ilki incelenecek olursa, bunun bir code smell olduğu, minor seviyede bir hata olduğu ve hatanın convention hatası olduğu görülebilmektedir. Hata üzerinde bulunan "Why is this an issue?" linki ile bunun neden bir hata olduğuna dair açıklama görüntülenebilmektedir. İlk hataya dair yapılan açıklama şekil 13 üzerinde gösterilmiştir.



Şekil 13: SonarQube Platformu - Hatanın Neden Hata Sayıldığının Görüntülenmesi

Bu adımların tamamlanması ile birlikte örnek bir kullanım durumunun incelenmesi de tamamlanmış bulunmaktadır. Bu bölümde SonarQube kullanımına dair basit bir örnek ile platformun ne olduğu, platform üzerinde nasıl proje oluşturulabileceği, tokenlerin ne olduğu ve ne amaçla kullanıldığı, projenin nasıl analiz edilebileceği, analiz sonuçlarının nasıl görüntülenebileceği ve incelenebileceği gibi bilgiler anlatılmış ve platforma basit bir giriş yapılmıştır.

3 SonarQube Platformunda Java

Java programlama dilinde yazılan programların build edilmesini otomatikleştiren Maven, Gradle gibi yardımcı araçlar yaygın bir şekilde kullanıldığından SonarQube kullanımı daha çok bu platformlar üzerinden yapılmaktadır.

3.1 SonarQube Platformunda Java - Maven

Maven kullanılarak geliştirilme yapılan bir Java projesini SonarQube platformu ile analiz etmek için yapılması gerekenler:

- **1.** "sonar-maven-plugin" ve "jacoco-maven-plugin" eklentilerinin "pom.xml" dosyası içerisinde gerekli yerlere eklenmesi gerekmektedir. İşleme ait örnek bir ekran görüntüsü şekil 14 üzerinde gösterilmektedir.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
        </plugin>
        <plugin>
            <groupId>org.sonarsource.scanner.maven</groupId>
            <artifactId>sonar-maven-plugin</artifactId>
            <version>3.6.0.1398</version>
        </plugin>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.4</version>
        </plugin>
    </plugins>
</build>
```

Şekil 14: Maven - pom.xml Örnek Dosya - Eklentiler

- **2.** SonarQube sunucusuna istatistikleri gönderecek profilin oluşturulması gerekmektedir. Oluşturulmuş örnek bir profile ait ekran görüntüsü şekil 15 üzerinde verilmiştir.

```
<profiles>
    <profile>
        <id>coverage</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.jacoco</groupId>
                    <artifactId>jacoco-maven-plugin</artifactId>
                    <executions>
                        <execution>
                            <id>prepare-agent</id>
                            <goals>
                                <goal>prepare-agent</goal>
                            </goals>
                        </execution>
                        <execution>
                            <id>report</id>
                            <goals>
                                <goal>report</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </profile>
</profiles>
```

Şekil 15: Maven - pom.xml Örnek Dosya - Profil

Bu adımda sunucu URL'si ve token gibi bilgiler, oluşturulan profil içerisinde "<properties>" alanına eklenebilir fakat bu bilgilerin kod tabanı içerisinde gömülü bulunması istenmeyen bir durum olduğundan bu bilgilerin parametre şeklinde "mvn" komutu ile kullanılması daha doğru olacaktır.

- **3.** Bu işlemlerden sonra proje analiz edilmeye hazır hale gelmiştir. Projeyi analiz etmek için komut: **mvn clean verify**

  sonar:sonar

  -Dsonar.host.url=${SONAR_HOST}

  -Dsonar.login=${SONAR_TOKEN}

  -Dsonar.projectKey=${SONAR_PROJECT_KEY} şeklinde kullanılmalıdır. Gerekli bilgiler profil içerisinde verilmiş ise **mvn clean verify -P{profile}** komutu ile ayarlanan profilin analiz edilmesi sağlanabilir.

3.2 SonarQube Platformunda Java - Gradle

Gradle kullanılarak geliştirilme yapılan bir Java projesini SonarQube platformu ile analiz etmek için yapılması gerekenler:

- **1.** gradle.properties dosyası içerisine SonarQube Scanner'in kullanacağı özelliklerin tanımlanması gerekmektedir. Örnek bir tanımlama olarak **systemProp.sonar.host.url=http::/localhost:9000** verilebilir.
- **2.** build.gradle dosyası içerisine gerekli eklentinin eklenmesi gerekmektedir. Örnek bir görüntü şekil 16 üzerinde görülebilmektedir.

```
plugins {
  id "org.sonarqube" version "3.3"
}
```

Şekil 16: Gradle - SonarQube Eklentisinin Eklenmesi

- **3.** Eklentinin eklenmesi ile proje analiz edilmeye hazır hale getirilmiştir. Projenin analiz edilebilmesi için "sonar.login" özelliği bilgisinin ya "gradle.properties" dosyasına eklenmesi ya da komutla birlikte verilmesi gerekmektedir. Komutla birlikte verildiği durumda kullanılacak komut **gradle sonarqube -Dsonar.login=yourToken** şeklinde olmalıdır.

Gradle kullanılarak Java projesinin SonarQube ile analiz edilmesi için bu işlemler çoğu zaman yeterli olmaktadır. Fakat multi-project denilen birden fazla proje içeren Gradle buildleri ya da görevlerin birbirine bağımlı olmasını gerektiren durumlarda yapılan analiz işlemleri için yapılması gerekenler farklılık gösterebilmektedir. Bu gibi durumlar için SonarScanner Gradle Dökümantasyonu incelenebilir.

3.3 SonarQube Platformunda Java - Manuel Analiz

Geliştirilen projede Maven ya da Gradle kullanılmamışsa projenin manuel olarak analiz edilmesi gerekmektedir. Bu işlem için özellikle projeye ait bytecode'ların gerekli parametre ile programa verilmesi gerekmektedir. Bu işlem ve diğer işlemler için kullanılabilecek parametreler şunlardır:

- **sonar.java.binaries (required)**: Compile edilmiş bytecode'ların bulunduğu dizinleri ifade eder. Birden çok dizin ifade edilirken bu dizinler virgül ile ayrılmalıdır.
- **sonar.java.libraries**: JAR ya da zip dosyalarından oluşan ve proje tarafından kullanılan üçüncü parti kütüphanelerin bulunduğu dizinleri ifade eder. Birden çok dizin ifade edilirken bu dizinler virgül ile ayrılmalıdır. Dizin ve dosya isimlerinin belirtilmesinde wildcard karakterleri kullanılabilir.
- **sonar.java.test.binaries**: sonar.java.binaries özelliğine gönderilen bilginin test dosyaları için kullanılan versiyonudur.
- **sonar.java.test.libraries**: sonar.java.libraries özelliğine gönderilen bilginin test dosyaları için kullanılan versiyonudur.

Bunlar dışında Java projelerinde custom kural tanımlamaları için Writing Custom Java Rules 101 sayfası, SonarQube platformunda Java kullanımına dair diğer bilgiler için SonarQube Java Docs sayfası ziyaret edilebilir.

4 SonarQube Platformunda C

SonarQube platformu kullanılarak analiz edilebilen bir diğer popüler dil de C dilidir. Platformun C ile kullanılabilmesi için "Developer Edition" ve üzeri bir sürümün kullanılması gerekmektedir.

SonarQube platformu, C dili için birçok derleyiciyi desteklemektedir. Bu derleyiciler:

- Clang, GCC ve Microsoft C/C++ derleyicilerinin herhangi bir sürümü,
- Linux ve macOS için Intel derleyicisinin herhangi bir sürümü,
- ARM5 ve ARM6 derleyicileri,
- ARM, Atmel AVR32, Renesas H8, Renesas RL78, Renesas RX, Renesas V850, Texas Instruments MSP430 ve 8051 için IAR derleyicileri,
- QNX derleyicileri,
- Windows ve macOS üzerindeki Texas Instruments ARM, C2000, C6000, C7000, MSP430 ve PRU derleyicileri,
- Wind River Diab ve GCC derleyicileri,
- GCC'yi temel alan derleyiciler (Linaro GCC gibi) derleyicileridir.

Platform üzerinde desteklenen dil standartları ise:

- C89, C99, C11, C18, C++03, C++11, C++17
- GCC eklentileri standartlarıdır.

Desteklenen çalıştırma ortamları:

- Microsoft Windows x86-64
- Linux x86-64
- macOS versiyon 10.14.3 ve sonrası x86-64 ortamlarıdır.

SonarQube platformunun C dili ile kullanılabilmesi için **"Build Wrapper"** programı gerekmektedir. Bu program sayesinde platform doğru analiz için gerekli olan tüm konfigürasyonları elde etmektedir (makro tanımları ve include dizinleri gibi). Build Wrapper programı, yazılan programın build sürecini etkilememektedir. Program, SonarQube sunucusundan direkt olarak indirilmelidir, böylece eklenti sürümü ile sürüm uyuşmazlığı sıkıntısı yaşanmaz.

Build Wrapper programının yanında analiz işlemi için SonarScanner programı yine gereklilikler arasında bulunmaktadır.

4.1 SonarQube Platformunda C - Analiz Adımları

- Build Wrapper programı normal clean build sürecine ekstra bir prefix olarak çalıştırılmalıdır. Programın linux, macOS ve Windows işletim sistemlerinde çalıştırılmasına dair bir örnek şekil 17 üzerinde görülebilir.

```
// example for linux
build-wrapper-linux-x86-64 --out-dir build_wrapper_output_directory
make clean all
// example for macOS
build-wrapper-macosx-x86 --out-dir build_wrapper_output_directory
xcodebuild clean build
// example for Windows
build-wrapper-win-x86-64.exe --out-dir
build_wrapper_output_directory MSBuild.exe /t:Rebuild
```

Şekil 17: Build Wrapper Programı Örnek Kullanım

Build işlemi tamamlandığında **build-wrapper.json** isimli bir dosyanın belirtenen çıktı dizini içerisinde otomatik olarak oluşmuş olması gerekmektedir. Bu dosya, build komutu tarafından build edilen translation üniteleri hakkında bilgiler içermektedir. Derlenmiş translation ünitesi bulunmayan hiçbir dosya SonarQube tarafından analiz edilmemektedir. Aynı şekilde compile edilmeyen kaynak dosyaları ve include edilmeyen header dosyaları da analiz edilmemektedir. Ayrıca Build Wrapper programı projenin build edilmesine etkide bulunmadığından projenin program kullanılmadan tekrar build edilmesine gerek yoktur.

- Projenin kök dizini içerisinde bulunan **sonar-project.properties** dosyası içerisinde **sonar.cfamily.build-wrapper-output** özelliğinin eklenmesi gerekmektedir. Bu özelliğin değerine ise build komutunda belirtilen output dizinin yazılması gerekmektedir. Örnek bir sonar-project.properties dosyası şekil 18 üzerinde verilmiştir.

```
sonar.projectKey=myFirstProject
sonar.projectName=My First C++ Project
sonar.projectVersion=1.0
sonar.sources=src
sonar.cfamily.build-wrapper-output=build_wrapper_output_directory
sonar.sourceEncoding=UTF-8
sonar.host.url=YourSonarCloud/SonarQubeURL
```

Şekil 18: Örnek sonar-project.properties Dosyası

  Projenin içerdiği tüm kaynak kodlarının proje kök dizini içerisinde oluşturulmuş bir alt dizin içerisinde toplanması önerilmektedir, bu sayede alakasız kaynak kodların analiz edilmesi önlenecektir. Şekil 18 üzerinde bu dizin **src** olarak adlandırılmıştır.

- SonarScanner programı projenin kök dizini üzerinde çalıştırılmalıdır. Böylece projenin analizi tamamlanmış olmaktadır.

Bu işlemlerde dikkat edilmesi gereken birkaç husus bulunmaktadır. Bunlardan ilki, Build Wrapper programının topladığı bilgiler arasında mutlak dosya yolları (absolute file paths) bilgilerinin de olmasıdır (kaynak dosyaları, standart header'ler, kütüphaneler gibi). Bu bilgiler daha sonra SonarScanner tarafından kullanılarak bu path'lere erişim sağlanmaktadır. Dolayısıyla bu işlem herhangi bir containerization programı kullanılırken dikkat edilmesi gereken bir işlemdir. Bundan dolayı SonarScanner CLI Docker Image C analizini desteklememektedir.

Bir diğer husus ise Build Wrapper programı tarafından üretilen build-wrapper.log ve build-wrapper-dump.json dosyalarıdır. Bu dosyalar bazı bağlamlarda güvenlik riskleri oluşturabilmektedir. İlk dosya sadece bir log dosyası olmakla ve analiz esnasında kullanılmamakla birlikte ikinci dosya analiz için gereklidir ve silinemez-taşınamaz durumdadır. Bazı durumlarda buna dikkat etmek gerekebilmektedir.

4.2 SonarQube Platformunda C - Çok İş Parçacıklı (Multithreaded) Analiz

Kod analizi sırasında makinenin sahip olduğu tüm çekirdeklerin kullanımı mümkündür. Bunu sağlamak için **sonar.cfamily.threads** özelliğinin ayarlanması gerekmektedir. Varsayılan değeri 1'dir.

- Bu özellik sadece 1 çekirdeğe sahip makinelerde aktifleştirilmemelidir.
- Analiz için en uygun değeri program belirlemez. Bu değerin deneme-yanılma yoluyla kullanıcı tarafından bulunması gerekmektedir.
- Bu özellik single-threaded çalıştırmaya göre daha fazla hafızaya ihtiyaç duymaktadır.
- Eğer 2 çekirdekli bir makine iki farklı analizi aynı anda gerçekleştirmek için ayarlanmışsa, "sonar.cfamily.threads=2" ayarı beklenilen performans kazanımını garanti etmemektedir, hatta varsayılan değerden daha kötü bir performans gösterebilir.
- 64 çekirdekli bir makinenin "sonar.cfamily.threads=64" seçeneği ile çalıştırılması, 32 çekirdekli bir makineye göre çok fazla performans kazanımı sağlayacağını garanti etmez. Performans farkı makineye, projeye ve projenin ayarlanmasına göre değişiklik göstermektedir.

Verilen bu bilgiler dışında C ile testler yapmak, örnek C projesi gibi bilgilere erişmek için SonarQube C/C++/Objective C Dökümantasyon sayfası ziyaret edilebilir.