

# Inhaltsverzeichnis

<b>Sperrvermerk .....</b>	<b>I</b>
<b>Abbildungsverzeichnis.....</b>	<b>III</b>
<b>Tabellenverzeichnis.....</b>	<b>IV</b>
<b>Abkürzungsverzeichnis.....</b>	<b>V</b>
<b>1 Einleitung .....</b>	<b>1</b>
1.1 CI/CD Pipelines .....	1
1.2 Die YAML-Datei.....	4
1.3 Statische Code-Analyse.....	5
1.4 Salesforce .....	5
1.5 Beschreibung des IST-Zustand.....	6
1.6 Beschreibung des SOLL-Zustandes .....	7
<b>2 Konzeptionierung .....</b>	<b>8</b>
2.1 Auswahl des SAST-Services .....	8
2.1.1 GitLab SAST.....	10
2.1.2 Salesforce Code Analyzer .....	11
2.1.3 SonarQube .....	13
2.1.4 Snyk Code.....	15
2.1.5 Vergleich und Entscheidung .....	16
2.2 Konzeptionelle Aufsetzung des Pipeline-Jobs.....	20
<b>3 Implementierung.....</b>	<b>25</b>
3.1 Analyse der derzeitigen CI/CD Pipeline .....	25
3.2 Implementierung des Konzept-Jobs.....	25
<b>4 Fazit .....</b>	<b>27</b>
<b>Literaturverzeichnis.....</b>	<b>VI</b>
<b>Anlagenverzeichnis .....</b>	<b>VIII</b>
<b>Ehrenwörtliche Erklärung .....</b>	<b>XII</b>

## Abbildungsverzeichnis

Abbildung 1: Beispielhafte Vorgänge in einem Runner für eine Pipeline .....	4
Abbildung 2: Beispielausgabe des SFCA in HTML-Format .....	12
Abbildung 3: Architektur eines aufgesetzten SonarQube Systems.....	14
Abbildung 4: Ergebnisse der Umfrage zur Gewichtung der Bewertungskriterien.....	18
Abbildung 5: Stage und Konfiguration des Konzept-Jobs.....	21
Abbildung 6: YAML-Code zur Installation des SAST-Tools mit anpassbarer Version.....	21
Abbildung 7: Vorbereitung der Argumente für den Aufruf des SAST-Tools .....	22
Abbildung 8: Vorbereitung des Outfile-Arguments des SFCA .....	23
Abbildung 9: Vorbereitung des Format-Arguments des SFCA .....	23
Abbildung 10: YAML-Code zur Durchführung des Sicherheitstests mittels SFCA .....	24
Abbildung 11: Upload der Testergebnisse des Sicherheitstests.....	24
Abbildung 12: Die für die Implementation wichtigen teile der existierenden Pipeline .....	25
Abbildung 13: Implementierter Konzept-Job .....	26

## **Tabellenverzeichnis**

Tabelle 1: Software Delivery Performance for 2016 .....	2
Tabelle 2: Manual Work Percentages .....	3
Tabelle 3: Punktevergabe der Kriterien für die Entscheidungsmatrix .....	17
Tabelle 4: Entscheidungsmatrix der SAST-Tools .....	19
Tabelle 5: Bepunktung der Kategorien der Entscheidungsmatrix .....	20

## Abkürzungsverzeichnis

CI/CD.....	Continuous Integration and Continuous Delivery
CRM .....	Customer Relationship Management
DAST.....	Dynamic Security Application Testing
IAST .....	Interactive Security Application Testing
LWC .....	Lightning Web Component
OASIS .....	Organization for the Advancement of Structured Information Standards
OWASP .....	Open Source Application Security Project
RASP.....	Runtime Application Self-Protection
SaaS.....	Software as a Service
SAST .....	Static Application Security Testing
SDLC.....	Software Development Life Cycle
SF.....	Salesforce
SFCA.....	Salesforce Code Analyzer
UI.....	User-Interface
VM.....	Virtual Machine
YAML.....	Yet Another Markup Language

# 1 Einleitung

Durch das stetige Wachstum der IT-Branche entstehen neben immer besseren Softwarelösungen immer komplexere Schwachstellen und Angriffsmöglichkeiten für Cyberkriminelle. Aufgrund dieser Komplexität kann eine späte oder nicht erfolgende Analyse der Sicherheitslücken mit hohen Kosten verbunden sein. Folglich sind IT-Firmen daran bemüht, Schwachstellen so früh wie möglich im Software Development Life Cycle (SDLC) zu erkennen, um eine großflächige Behebung zu garantieren.

Eine Möglichkeit für eine frühe Identifizierung bietet die statische Code-Analyse, welche den Quellcode einer Anwendung auf unsichere Muster analysiert. Diese wird von verschiedenen Dienstleistern, in Form von Static Application Security Testing (SAST), mit unterschiedlicher Ausführung angeboten. Auf Sicherheitsanalysen anderer Art soll in dieser Arbeit nicht eingegangen werden.

Um die Betreuung der SAST-Tools so effizient wie möglich zu gestalten und Sicherheitstests regelmäßig durchzuführen, werden diese in Continuous Integration und Continuous Delivery (CI/CD) Pipelines integriert. So bietet beispielsweise GitLab, ein Versionsverwaltungssystem mit DevOps-Funktionalität, die Möglichkeit, SAST nativ in eine solche Pipeline zu integrieren.

Ziel dieser Arbeit ist es, Salesforce (SF) Projekte zu professionalisieren und deren Sicherheit mittels statischer Code-Analyse zu erhöhen. Dazu werden zunächst verschiedene SAST-Tools, unter Berücksichtigung unternehmensspezifischer Kriterien, gegenübergestellt, mit dem Ziel eins auszuwählen. Anschließend wird dieses evaluiert und in eine GitLab CI/CD Pipeline eingeführt. Eine Nachbetrachtung, ob effektiv Sicherheitslücken behoben werden konnten, müsste über einen längeren Zeitraum erfolgen und soll folglich in dieser Arbeit nicht vorkommen.

## 1.1 CI/CD Pipelines

Pipelines sind automatisierte Prozesse, welche bei Änderungen in einem Versionsverwaltungssystem, selbständig Modultests durchführt oder Verteilungspakete erstellen und sind zentraler Bestandteil von CI/CD<sup>1</sup>. Jez Humble, Autor der [continuousdelivery.com](https://continuousdelivery.com) Website, beschreibt sie als „The key pattern introduced in continuous

---

<sup>1</sup> Vgl. [Jez18]

delivery [...]”.<sup>2</sup> Da CI/CD als Methodik für DevOps gilt, decken sich folglich die technischen relevanten Prinzipien und Ziele von der DevOps Bewegung mit den Gründen eine solche Pipeline zu verwenden. Diese Prinzipien sind Automatisierung, kontinuierliche Verbesserung, ergebnisorientierte Entwicklung und kleine wiederholende SDLC-Zyklen<sup>3</sup> und zielen auf eine schnelle Softwareverteilung, im Fachjargon auch Deployments genannt, Qualität und Anpassungsfähigkeit ab.<sup>4</sup>

Die Praxisanwendung von DevOps bestätigt die Effektivität in der Umsetzung ihrer Ziele und zeigt, dass dessen Nützlichkeit und Wichtigkeit für Unternehmen, die sich zu **High-Performern** zählen wollen, von sehr großer Bedeutung ist. Im Buch Accelerate, welches sich primär mit der Messbarkeit von Softwareverteilung und dessen Antreiber befasst, wird dieser Zusammenhang zwischen der Anwendung der DevOps-Prinzipien und hohen Performern deutlich.

2016	High Performers	Medium Performers	Low Performers
<b>Deployment Frequency</b>	On demand (multiple deploys per day)	Between once per week once per month	Between once per month and once every six months
<b>Lead Time for Changes</b>	Less than one hour	Between one week and one month	Between one month and six months
<b>Mean Time to Restore</b>	Less than one hour	Less than one day	Less than one day
<b>Change Failure Rate</b>	0-15%	31-45%	16-30%

Tabelle 1: Software Delivery Performance for 2016

Mit Änderungen entnommen aus: [FHK18], S. 19

In den Tabellen ist klar erkennbar das Unternehmen, welche Prinzipien wie Automation, also für Routinearbeiten weniger manuelle Arbeit aufwenden (siehe Tabelle 2) oder kleine SDLC-Zyklen besser implementieren und im Umkehrschluss eine höhere Deployment-Frequenz verzeichnen (siehe Tabelle 1), zu High-Performern gehören.

<sup>2</sup> Ebenda

<sup>3</sup> Vgl. [Atl24a]

<sup>4</sup> Vgl. [Atl24b]

<b>Manual Work</b>	<b>High Performers</b>	<b>Medium Performers</b>	<b>Low Performers</b>
<b>Configuration management</b>	28%	47%	46%
<b>Testing</b>	35%	51%	49%
<b>Deployment</b>	26%	47%	43%
<b>Change approval process</b>	48%	67%	59%

Tabelle 2: Manual Work Percentages

Mit Änderungen entnommen aus: [FHK18], S. 214

Der Cybersicherheitskontext der Aufgabenstellung erfordert zusätzlich eine Betrachtung der Motive, Cybersicherheit in eine solche Pipeline zu integrieren.

Wobei sich herausstellt, dass der Begriff DevOps als ursprüngliche reine Zusammenfügung von Developer und Operations nicht ganz sinngemäß ist. Die Integration von Cybersicherheit in CI/CD und somit in Pipelines erhöht die Automation, reduziert das Auftreten großer Nacharbeiten, so ist es besser kleine Schwachstellen gleich zu beheben als nach Fertigstellung große strukturelle Änderungen durchsetzen zu wollen und ist in Verbindung mit den Sicherheitsanforderungen der Moderne eine wichtige Erweiterung.<sup>5</sup> Diesbezüglich wurden mehrere Begriffserweiterungen, wie DevSecOps oder Rugged DevOps vorgeschlagen, welche die Integration von Cybersicherheit in DevOps konkret hervorheben.<sup>6</sup> Aufgrund des Begriffes DevOps als Sammelbegriff für eine Vielzahl an Methoden, Prinzipien und Ziele werden diese nur teilweise durchgesetzt und im Gegenteil wird Cybersicherheit mehr als Bestandteil von DevOps gesehen.<sup>7</sup> Aus diesen Gründen beschäftigt sich die Arbeit mit Pipelines und verwendet diese als Pattern um die CI/CD Prinzipien durchzusetzen. Der Einsatz von Pipelines innerhalb GitLab resultiert aus dem in der dotSource SE bereits etablierten Gebrauch des Versionsverwaltungssystems.

---

<sup>5</sup> Vgl. Ebenda, S. 69-71

<sup>6</sup> Vgl. Ebenda, S. 72f

<sup>7</sup> Vgl. [Sca24]

## 1.2 Die YAML-Datei

Eine Pipeline in GitLab besteht aus zwei wesentlichen Komponenten – einer Konfigurationsdatei im Repository und einem Runner.

Die Konfigurationsdatei ist im Yet Another Markup Language (YAML) Format und definiert Phasen, sogenannte Stages, in welchen durch Jobs, Tests am Sourcecode konfiguriert werden. Darüber hinaus bietet die YAML-Datei viele Anpassungsmöglichkeiten, wann und wie getestet und ob eine Output-Datei im Repository hinzugefügt werden. Variablen oder Templates, vorgefertigte YAML-Dateien, können ebenfalls eingebunden und verwendet werden.

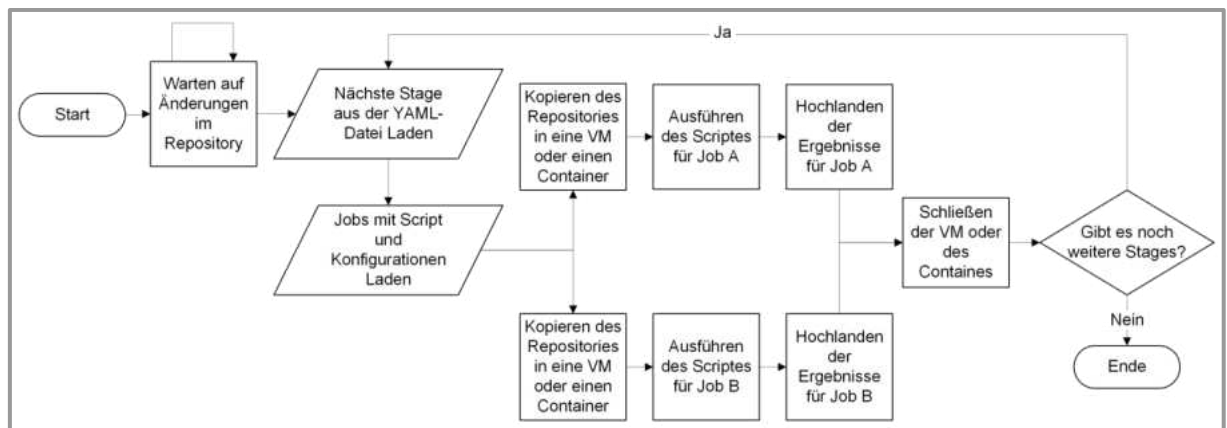


Abbildung 1: Beispielhafte Vorgänge in einem Runner für eine Pipeline

Der Runner ist ein Agent, welcher kontinuierlich auf einem Server läuft und auf Änderungen im Repository wartet. Findet er eine solche Änderung, so führt er die in der YAML-Datei definierte Pipeline aus. Dazu werden zunächst alle Jobs der ersten Stage geladen und anschließend parallel ausgeführt. Der Runner kopiert dazu das Repository für jeden Job in eine eigene Instanz, welche für gewöhnlich eine virtuelle Maschine (VM) oder einen Docker Container ist.

Die YAML-Datei ist in verschiedene Segmente unterteilt. Eines dieser Segmente ist das Script Segment in einem Job Segment. Anhand dieses Segmentes arbeitet der Runner nun Befehle, durch einen vorher definierten Ausführer, auch Executor genannt, wie zum Beispiel Powershell oder CMD.exe ab. Anschließend werden die Ergebnisse in Form von Dateien in GitLab hochgeladen und als Artefakt des jeweiligen Jobs zur Verfügung gestellt.



Dieser Ablauf ist jedoch nur exemplarisch für eine Pipeline. Eine Erläuterung aller Konfigurationsmöglichkeiten innerhalb der Pipeline ist nicht zielführend und ist deshalb nicht Teil der Arbeit. Die YAML-Datei trägt jedoch als zentraler Bestandteil und Anleitung für den Runner eine entscheidende Rolle und wird folglich Teil der Konzeptionierung und Implementation.

### **1.3 Statische Code-Analyse**

Statische Code-Analyse ist ein Verfahren, bei dem Code auf allgemein anerkannte oder selbst erstellte Regeln analysiert. Dadurch das der Source Code an sich und nicht das fertige Programm analysiert wird, muss folglich der Zugriff auf diesen Gewährleistet werden und führt dazu das die statische Code-Analyse auch als White-Box-Testen, als Gegensatz zur Black-Box, eine Box in die nicht hineingeblickt werden kann, bezeichnet wird.<sup>8</sup> Zu den allgemein anerkannten Regeln gehören Codierungsrichtlinien von beispielsweise der Open Source Application Security Project (OWASP) und weiteren Stiftungen, welche sich primär mit Cybersicherheit beschäftigen und von Marktführern wie Salesforce (siehe 1.4) unterstützt werden.

Statische Code-Analyse bietet also eine gute Möglichkeit, wichtige Schwachstellen früh im SDLC zu erkennen, denn durch die Analyse des Codes muss kein fertiges Programm existieren und gilt deswegen als empfohlene erste Maßnahme, um die Sicherheitserweiterung der DevOps-Bewegung umzusetzen.

Die Praxisumsetzung von statischer Code-Analyse geschieht über SAST-Tools, wobei die Begriffe SAST und statische Code-Analyse dasselbe meinen und daher untereinander austauschbar sind. Die Funktionalität, also das Analysieren des Codes auf Schwachstellen, ist deckend.<sup>9</sup>

### **1.4 Salesforce**

Das im Jahr 1999 gegründete Unternehmen Salesforce, entwickelt und stellt allumfassende Cloud basierte Customer Relationship Management (CRM) Lösungen zur Verfügung.<sup>10</sup> Mittlerweile zählt es, durch die Strategie Sales, Marketing, Commerce und IT-Teams unter

---

<sup>8</sup> Vgl. [Lub23]

<sup>9</sup> Vgl. [OWA24]

<sup>10</sup> Vgl. [Sal24b]

einer Plattform zu vereinen als eines der marktführenden Unternehmen auf diesem Gebiet. Dies wird durch eine Vielzahl an Auszeichnungen von Marktanalysen, bei denen Salesforce konstant zu den Top-Kandidaten zählt, unterstrichen.<sup>11</sup>

Aufgrund der hohen Marktpräsenz von Salesforce gibt es viele Kundenanfragen, die eine genauere Anpassung oder Einrichtung von Salesforce auf ihre Anforderungen benötigen. Dies geschieht unter Salesforce mit Apex und Lightning Web Components (LWC). Sie sind Salesforce intern entwickelte Programmiersprachen, welche auf ihre Cloud maßgeschneidert sind, um weitere Funktionalität oder Konfigurationen hinzuzufügen. Diese Dienstleistung ist Teil des dotSource SE Portfolios, weswegen folglich die Ergebnisse dieser Arbeit ein wichtiger Beitrag für die dotSource SE sind.

Des Weiteren resultieren durch den Fokus auf Salesforce konkrete Anforderungen an das zu implementierende SAST-Tool. So ist beispielsweise die Unterstützung der Salesforce relevanten Programmiersprachen deutlich besser zu bewerten als andere Programmiersprachen.

## **1.5 Beschreibung des IST-Zustand**

Das Handeln der programmierenden Teams der dotSource SE innerhalb verschiedener Leistungsbereiche befolgen schon lange die technisch relevanten Methoden, welche aus der DevOps-Bewegung hervorgehen. Einer dieser Leistungsbereiche ist die seit 2019 von der dotSource SE angebotene Apex und LWC-Entwicklung unter Salesforce, bei welcher Salesforce Lösungen maßgeschneidert auf die Anforderungen von Kunden entstehen. In diesem Leistungsbereich wird bereits mit Pipelines und Unit Tests nach jeder Änderung des Codes gearbeitet, um die Funktionalität von bereits entwickeltem Quellcode auch nach zukünftigen Erweiterungen zu gewährleisten.

Aber nicht nur das Handeln ist der dotSource SE von der DevOps-Bewegung geprägt, sondern auch die Unternehmenskultur. So finden sich Prinzipien wie Agilität und Innovation fest, in der dotSource DNA, ein Manuskript mit den 20 Grundsätzen der dotSource SE, verankert. Eine dieser Prinzipien ist die ständige strategische Weiterentwicklung.

---

<sup>11</sup> Vgl. [Sal24a]

## **1.6 Beschreibung des SOLL-Zustandes**

Diesem Prinzip der Weiterentwicklung soll nachgegangen werden und die bestehenden CI/CD-Methoden um Sicherheitsanalysen ergänzt werden. Dazu soll aus den genannten Gründen (siehe 1.1,1.3) ein SAST-Tool in eine bestehende Pipeline integriert werden. Da eine weite Anwendung der Ergebnisse der Arbeit erstrebenswert ist, muss mit den Programmierern im Salesforce Leistungsbereich geklärt werden, welche Eigenschaften das SAST-Tool für sie erfüllen muss. Das folgende Konzept soll begründen, warum sich für jenes Tool entschieden wurde und vermitteln, wie dieses implementiert wird, um die einfache Adaption in Salesforce-Projekten zu begünstigen.

## 2 Konzeptionierung

### 2.1 Auswahl des SAST-Services

Aufgrund der Fülle an existierenden SAST-Tools mit unterschiedlichen Kosten-Modellen und Ausführungen der statischen Code-Analyse sowie Zusatzfunktionen muss eine Auswahl getroffen werden, welches Tool implementiert werden soll. Diese Auswahl ist wichtig, um die weiteren Schritte in der Implementation zu klären, da die meisten Tools unterschiedlich eingebunden werden müssen und um sicherzustellen, dass das Tool mit unseren Anforderungen übereinstimmt.

Für den letzteren Grund wurden folgende Kriterien festgehalten:

Aus der Anforderung, das Tool in Salesforce-Projekten umzusetzen, müssen die **unterstützten Sprachen** des Tools bewertet werden. Dabei sind Salesforce eigene Sprachen besser zu bewerten. (siehe 1.4)

Ein Nachteil, welcher aus der Analyse des Quellcodes einer Anwendung hervorgeht, ist dass der fehlende Kontext über die tatsächliche Ausführung des Programmes oftmals zu falschen Positiven führt. Eine breite **automatische Erkennung falscher Positive** ist daher eine wünschenswerte Eigenschaft des SAST-Tools.

Um, falls es vom Tool mitgelieferte Regeln gibt, welche nicht unseren Vorstellungen entsprechen, sowie um eigene Regeln umzusetzen, ist die **Anpassbarkeit** des Tools eine weitere zu betrachtende Eigenschaft. Allgemeine Anpassungen, welche nicht im Zusammenhang mit der Erfassung von Sicherheitsschwachstellen oder der Konfiguration von Regeln stehen, fließen dabei nicht in die Bewertung ein.

Um die Schwachstellen effektiv auszuwerten kann, eine **Visualisierung der Ergebnisse** bzw. gefundenen Schwachstellen hilfreich sein und wird demnach ebenfalls bewertet. Es ist eine Einbindung in das GitLab User-Interface (UI) vorgesehen, so sollten bestenfalls in Merge Requests die Schwachstellen aufgelistet werden und die entsprechende Codezeile angezeigt werden.

Die in 1.1 und 1.3 bereits begründete **CI/CD Integration** sollte für das Tool so einfach wie möglich sein und stellt somit ein weiteres Kriterium auf. Genauer gemeint ist der komplette Prozess, welcher nötig ist, um das Tool ohne Konfiguration zu verwenden.

Da im Salesforce-Leistungsbereich mit Kundendaten agiert wird, ist es wichtig, den **Datenspeicherort/Datenschutz** des Tools genauer zu betrachten. Tools, welche Ergebnisse aus der Sicherheitsanalyse in einer Cloud oder auf einem externen Server speichern, sind schlechter zu bewerten als Tools, welche die Ergebnisse in lokalen Dateien oder auf internen Servern speichern.

Die Implementierung des Tools sollte eine positive Auswirkung auf den Profit der Projekte haben. Dazu müssen sich die Ergebnisse aus der Verwendung des Tools mit dessen **Kosten** mindestens decken. Sollten dabei mehrere Tools existieren, die ähnliche Ergebnisse liefern, ist folglich das erschwinglichere vorzuziehen.

Die zu evaluierenden SAST-Tools wurden auf vier mögliche Kandidaten, welche in den nachfolgenden Kapiteln erläutert werden, begrenzt. Folglich ist es möglich, dass es ein Tool gibt, welches sich besser eignen würde, aber in dieser Arbeit nicht betrachtet wird. Diese vier Ursprungskandidaten zeichnen sich dadurch aus, dass sie schon in der bereits **bestehenden Tool-Landschaft**, also teilweise schon in den in der dotSource SE, verwendeten Systemen, bekannt sind bzw. partiell existieren. Was das für jenes Tool genau bedeutet, wird in dem jeweiligen Kapitel geklärt. Diese Eigenschaft wird als ein weiteres Kriterium für die Bewertung genutzt.

Neben den oben genannten Kriterien gibt es weitere, die naheliegend sind. Um diese aber effektiv einschätzen zu können, bedarf es einer Testimplementierung mit nachblickender Analyse. Aufgrund des knappen Zeitrahmens der Arbeit werden diese Kategorien nur oberflächlich bewertet, wurden aber auch in Absprache mit den Programmierern als wichtig festgehalten:

Damit die Anpassung der Regeln oder die Konfiguration des Tools den später verwendenden Programmieren so effizient wie möglich ablaufen können, muss die **Dokumentation** des Tools möglichst genau sein und bildet somit ein weiteres Kriterium. Wie bereits erwähnt, fällt eine qualitative Einschätzung der Dokumentation aufgrund des Zeitrahmens der Arbeit aus. Die Dokumentation wird in dieser Arbeit also nach vorhanden sein bewertet. Eine bessere Dokumentation zeichnet sich durch das zusätzliche Vorhandensein von Kontaktmöglichkeiten, in Form von einem Forum oder einem öffentliches Repository und einer dedizierten Lernplattformen für das SAST-Tool aus.

Auch wenn die Sicherheitsanalyse wichtig ist, sollte sie nicht die Hauptaufgabe eines Mitarbeiters sein. Der **Wartungsaufwand** des SAST-Tools ist folglich so gering wie möglich zu halten, um nicht zu einer möglichen Last zu werden. Ein höherer Wartungsaufwand entsteht, wenn Updates am Tool, eine erhebliche Überarbeitung der Konfiguration benötigen.

### 2.1.1 GitLab SAST

GitLab SAST ist das in dem, in der dotSource SE verwendete, Versionsverwaltungssystem enthaltene SAST-Tool. Es verwendet zur statischen Code-Analyse von Drittanbietern zur Verfügung gestellte Scanner und verpackt sie in ein YAML-Template. Folglich muss für eine CI/CD Integration, dieses Template nur noch in der YAML-Datei aufgerufen werden. Das SAST-Tool unterstützt 26 verschiedene Programmiersprachen und Frameworks, darunter auch Apex. LWC und andere Salesforce-relevante Sprachen werden nicht unterstützt. Der für Apex verwendete Scanner ist PMD (keine ausgeschriebene Bedeutung).

GitLab stellt die Verwendung des SAST-Tools kostenlos allen Nutzern zur Verfügung. Die Anpassungsfähigkeit, Visualisierung und die automatische Erkennung falscher Positive müssen aber durch den Kauf der GitLab Ultimate Lizenz dazu gekauft werden. Diese kostet ca. 99 € pro Entwickler pro Monat. Eine genauere preisliche Festlegung der Lizenz ist nur durch Absprache mit dem GitLab Sales Team möglich.

Die Erkennung falscher positive des GitLab SAST ist für die Auswahl von geringem Ausmaß. Der Grund dafür ist das Apex, die einzig Salesforce relevante, von GitLab SAST analysierte Sprache, nicht in dieser Erkennung inbegriffen ist.<sup>12</sup>

Die Anpassung der Regeln des SAST-Tools geschieht über die darübergerlegte Template-Ebene und erfordert eine GitLab-einzigartige Formatierung. So wird in der YAML-Datei durch eine Variable ein Pfad auf eine Konfigurationsdatei mitgegeben. In dieser finden dann alle Anpassungen der Regeln statt. Für Apex bedeutet das, dass Regeln entweder ganz herausgenommen werden können oder einzelne Teile einer Regel, wie die Beschreibung oder die Risikostufe, verändert werden können. Die Möglichkeit, eigene Regeln zu erstellen, existiert nur für andere Sprachen.<sup>13</sup>

---

<sup>12</sup> Vgl. [Git24a]

<sup>13</sup> Vgl. [Git24b]

Den Datenschutzanforderungen wird GitLab SAST dadurch gerecht, dass alle verwendeten Scanner lokal die Sicherheitstests ausführen bzw. Ergebnisse in Form einer JSON-Datei wiedergeben und nicht extern in einer Cloud verarbeiten oder speichern werden.

Zusammenfassend wäre die Implementation von GitLab SAST trotz Abstrichen bei den unterstützten Sprachen eine gute Lösung. Das Tool hebt sich besonders hervor, durch dessen einfache Integration in die CI/CD Pipeline und einer visuellen Einbindung in das GitLab UI.

### **2.1.2 Salesforce Code Analyzer**

Der Salesforce Code Analyzer (SFCA) ist ein kostenfreies, direkt von Salesforce entwickeltes Tool zur statischen Code-Analyse. Es verwendet, ähnlich zu GitLab SAST, externe Scanner zur Analyse. Dabei deckt sich der Scanner für Apex, für andere Sprachen jedoch nicht. Die Verwendung von ESLint als Scanner für JavaScript führt dazu, dass über ein Plug-in das LWC-Framework auch auf Schwachstellen untersucht werden kann. Zudem arbeitet Salesforce fortwährend an einem eigenen leistungsstarken Scanner, der Salesforce Graph Engine, welche bereits im SFCA verwendet werden kann.<sup>14</sup>

Die Dokumentation für den SFCA existiert zwar größtenteils auf der offiziellen Salesforce Developer Website. Dennoch müssen für die Erstellung eigener Regeln die Dokumentationen der externen Scanner verwendet werden.<sup>15</sup>

Der Scanner lässt sich über das Salesforce Command Line Interface (CLI) installieren und ausführen, benötigt aber zusätzlich Java 17, um zu funktionieren. Eine Integration in eine CI/CD Pipeline ist also, durch Installation der Abhängigkeiten und anschließendes ausführen in dem Job-Skript, umsetzbar<sup>16</sup>

Sollte der SFCA implementiert werden, entstehen keine Kosten, da er Kostenlos mit allen Funktionen verwendbar ist.

SFCA kann Ergebnisse in Sarif-, JSON-, HTML-, JUnit-, XML- und CSV-Dateien sowie im Ausführer direkt als Tabelle ausgeben. Um eine visuelle Einbindung in das GitLab Overlay von externen Scannern zu ermöglichen, ist eine Konfigurationsmöglichkeit in GitLab vorhanden. In der YAML-Datei lassen sich Artefakte der Jobs als Report hinzugefügen, welche dann von

---

<sup>14</sup> Vgl. [Sal24e]

<sup>15</sup> Vgl. [Sal24c]

<sup>16</sup> Vgl. [Sal24d]

GitLab Overlay, je nach Reporttyp visualisiert dargestellt werden. Die infrage kommenden Reporttypen wären dabei der SAST- und der JUnit-Report. Der SAST-Report unterliegt der GitLab Ultimate Lizenz, welche unabhängig von dem verwendeten Scanner gekauft werden müsste, und erforderte eine spezielle Formatierung einer JSON-Datei, welche der JSON-Ausgabe des SFCA nicht entspricht. Ähnliche Formatierungsprobleme existieren Beim JUnit Report, weswegen eine Einbindung derzeit nicht möglich ist.

Aushilfe könnte zukünftig die von vielen SAST-Tools unterstützte Ausgabe im Sarif-Format sein, denn GitLab arbeitet daran, die Sarif-Dateien als Reporttyp hinzuzufügen. Da Sarif ein von der Organization for the Advancement of Structured Information Standards (OASIS) vorgeschlagenes, standardisiertes Ausgabeformat für SAST-Tools sein soll, ist die Einbindung in GitLab eine zukünftige Möglichkeit.

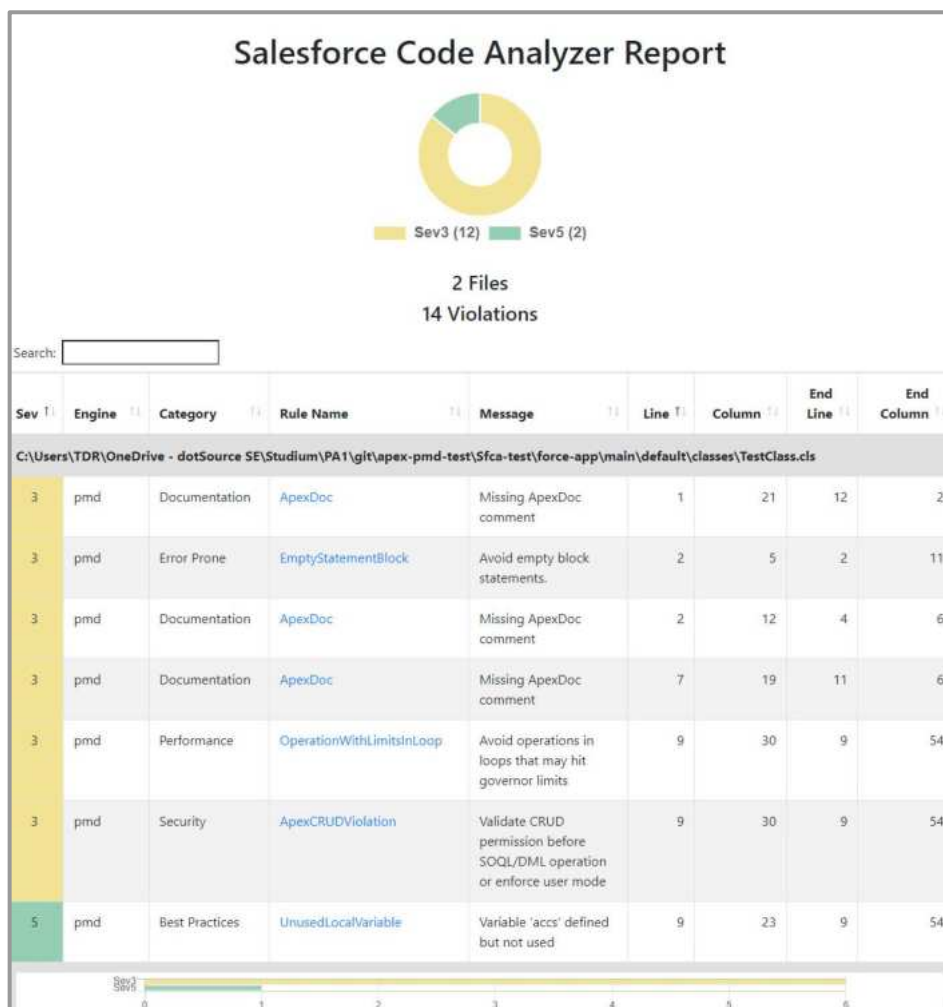


Abbildung 2: Beispielausgabe des SFCA in HTML-Format



Sollten Projekte in der Zwischenzeit dennoch eine Visualisierung erfordern, so kann übergangsweise das HTML-Ausgabeformat des SFCA verwendet werden. Dieses ist menschenlesbarer als eine reine JSON-Formatierung und ermöglicht das Sortieren von erkannten Schwachstellen nach deren Typ, Ort des Auftretens und Regelkategorien. Eine Auswertung könnte somit über den Download des durch den Scanner produzierten Artefaktes und dessen Aufruf in einem Browser erfolgen.<sup>17</sup>

Die Anpassbarkeit des SFCA ist mit unseren Vorstellungen konform. Es können Regeln für PMD und ESLint hinzugefügt und mitgelieferte Regeln ausgeschaltet werden.

Die Implementation des Salesforce Code Analyzer scheint vielversprechend. Der Fokus auf Salesforce-Projekte, Anpassbarkeit, sowie die Kostenlosigkeit sprechen dafür. Eine visuelle Einbindung in GitLab ist zwar derzeit nicht möglich, aber die mitgelieferten Visualisierungsmöglichkeiten bieten eine ausreichende Alternative.

### **2.1.3 SonarQube**

SonarQube ist ein, von SonarSource entwickeltes, umfangreiches Tool zur statischen Sicherheitsanalyse. Dabei ist SonarQube, im Gegensatz zu SonarCloud, die selbstverwaltete Servervariante des Tools, welche somit unseren Datenschutz Anforderungen gerecht wird.

SonarQube wird in mehreren Editionen angeboten. Dazu zählen eine kostenlose und eingeschränkte Community Edition, sowie darauf aufbauende kostenpflichtige Editionen. Das Kostenmodell für die Editionen richtet sich nach der Summe der Anzahl der Codezeilen in allen Projekten in Zusammenhang mit der Editionsstufe. Bei der Aufnahme von SAST-Tools für die Auswahl fiel auf, dass die SonarQube Community Edition bereits firmenintern eingerichtet ist, da diese aber keine Unterstützung der Salesforce sprachen anbietet, kommen nur die Editionen Enterprise oder Data Center infrage. Beide Editionen unterstützen Apex. Es existieren versionsunabhängig keine speziellen LWC-Regeln. Die Enterprise Edition startet bei 21000 \$ pro Jahr für 1 Mio. Codezeilen. Wird die Anzahl an Codezeilen überschritten so erhöhen sich die Kosten pro Jahr.

SonarQube lässt nativ für Apex keine eigenen Regeln zu. Erlaubt aber das Importieren der Ergebnisse anderer Scanner, welche diese Funktionalität besitzen können.

---

<sup>17</sup> Vgl. [Sal24e]

Da diese aber nicht direkt Bestandteil des Tools sind, so wie PMD im GitLab SAST-Tool, wird für die Auswertung angenommen, dass SonarQube keine eigenen Regeln für Apex unterstützt. Die mitgelieferten Regeln von SonarQube können über Qualitätsprofile, welche das Tool als Regelsatz für den Scan verwendet, angepasst werden.

Eine visuelle Einbindung innerhalb von GitLab ist zwar möglich, da SonarQube das Artefakt des Pipeline-Jobs so formatiert, dass es mit dem SAST-Report übereinstimmt, benötigt aber trotzdem GitLab Ultimate, um diese im UI von GitLab zu visualisieren. Eine anderweitige Visualisierung besteht aber über das Web UI des aufgesetzten SonarQube Servers

Die Einbindung in die CI/CD Pipeline ist durch eine Vielzahl von Vorarbeiten geprägt. Das folgende Diagramm visualisiert die SonarQube Architektur.

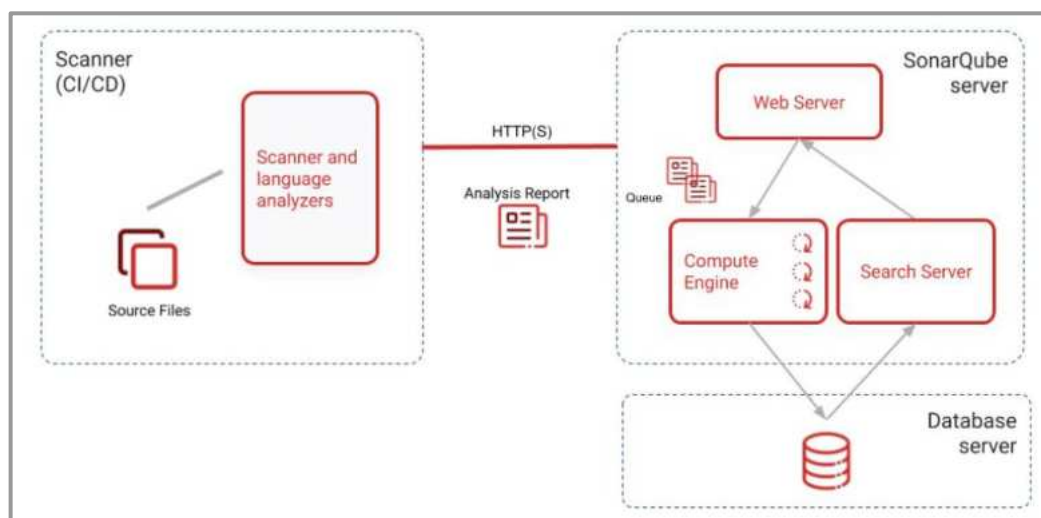


Abbildung 3: Architektur eines aufgesetzten SonarQube Systems

Quelle: [Son24]

Die Integration ist also nur möglich, wenn ein Datenbank-Server der SonarQube-Server und der SonarScanner mit allen Abhängigkeiten installiert werden. Im Vergleich zu den bereits erläuterten SAST fällt dieser Prozess deutlich unständiger aus.

Zusammenfassend wird klar, dass SonarQube sich auf eine unternehmensweite Umsetzung des Tools fokussiert. Der lange Integrationsprozess, die hohen Kosten, die Menge an unterstützten Sprachen, und das komfortable Web UI begründen dies.

Zusammen mit der, für unsere Anwendungszwecke, unzureichende Unterstützung der Salesforce Plattform, lässt sich sagen, dass eine Implementierung von SonarQube nicht erstrebenswert ist.<sup>18</sup>

#### **2.1.4 Snyc Code**

Bei Rücksprache mit dem firmeninternen Cloud-Services-Team der dotSource SE, welche SAST-Beratung für Kunden anbietet, ist Snyc Code als weiteres mögliches Tool aufgekommen. Snyc Code ist das SAST-Tool der Snyc Software as a Service (SaaS) Plattform und folglich Cloud basiert. Es unterstützt ebenfalls Apex, LWCs aber nicht.

Es gibt eine kostenlose Version mit einer begrenzten Anzahl an Sicherheitstests und 2 kostenpflichtige, ohne Begrenzung. Die Version, die für die Auswahl infrage kommen würde, ist die Team Version, da sie sich zur am stärksten bepreisten Enterprise Version, außerhalb von Early Access Funktionen und den Kosten, für den SAST-Teil der Plattform, in Bezug auf eine bessere Erkennung von Sicherheitsschwachstellen, nicht ausschlaggebend unterscheidet. Die Team Version ist erhältlich für 25 \$ pro Monat pro Entwickler. Unabhängig von den Versionen werden die Tests auf der Cloud ausgeführt und die Ergebnisse gespeichert.<sup>19</sup>

Ähnlich zu SonarQube verfügt es über eigenes Web UI, für Konfiguration und Visualisierung. Wobei dieses aber nicht auf einem lokalen Server liegt, sondern auf einer externen Cloud, auf welcher auch die Tests durchgeführt werden.

Dadurch ist die CI/CD Integration im Vergleich zu SonarQube, durch die Cloudbasiertheit etwas weniger aufwendig. So muss in einem Job in der YAML-Datei nur Node.js und darüber die Snyc CLI installiert werden. Anschließend reicht ein Aufruf des Befehls, welcher das SAST-Tool startet.<sup>20</sup>

Die Verwendung der Cloud birgt aber auch Nachteile in sich. Bei der Analyse auf einer externen Plattform, verlässt der Quellcode die dotSource SE, was für eine Digitalagentur bedeutet, dass die Verwendung des externen Tools mit dem Kunden abgesprochen werden muss.

---

<sup>18</sup> Vgl. [Sny24d]

<sup>19</sup> Vgl. [Sny24c]

<sup>20</sup> Vgl. [Sny24a]

Dieser Fall sollte vermieden werden – Snyk bietet aber keine lokale Variante des Tools an.<sup>21</sup>

Eine visuelle Auswertung in GitLab durch eine CI/CD Integration ist unter den gleichen Umständen wie bei dem Salesforce Code Analyzer und SonarQube nicht möglich. Eine Übergangslösung bietet dabei das Web UI oder die Ausgabe in einem HTML-Artefakt. Zusätzlich können durch den Erwerb der Enterprise Lizenz im Web UI Zusammenfassungen angezeigt werden lassen.

Die Dokumentation hebt sich gegenüber den anderen Tools, welche nur die Dokumentation und Kontaktmöglichkeiten anbieten, heraus. Snyk bietet eine Lernplattform an, auf welcher Erklärvideos zu finden sind.<sup>22</sup>

Snyk bietet sich an sich, durch die Visualisierung über ein Web UI und die besonders ausgiebige Dokumentation, als mittelmäßige Alternative zu GitLab SAST an. Eine Implementation für Dienstleistungsanbieter, wird aber durch die Cloudbasiertheit stark eingeschränkt und bietet sich daher eher für firmeninterne, aber nicht für Kundenprojekte an.

### **2.1.5 Vergleich und Entscheidung**

Nachdem nun mögliche SAST-Tools vorgestellt wurden, soll aus diesen eins für den weiteren Verlauf der Arbeit, sprich die konzeptionelle Erstellung eines Jobs in der YAML-Datei und dessen Integration in ein Salesforce-Projekt, ausgewählt werden.

Um diese Auswahl statistisch darzustellen, wird eine gewichtete Entscheidungsmatrix verwendet. Eine solche Auswertung hat den Vorteil, dass es zwischen Optionen, welche dieselben Gesamt-Bewertungspunkte haben, trotzdem eine klare bessere Option gibt. Das geht aus der Gewichtung der Kriterien hervor. Trotzdem wird aber nicht ausgeschlossen, dass Tools, welche in einer niedrig gewichteten Kategorie, um ein Vielfaches besser sind als ihre Konkurrenten und überhand gewinnen können.

Ausschlaggebend war, für die Entscheidungsmatrix, dass die Tools, in der gleichen Kategorie, nach den gleichen Anforderungen bewertet werden.

---

<sup>21</sup> Vgl. [Sny24b]

<sup>22</sup> Vgl. [Sny24e]

Die Kategorien wurden nach dem folgenden Schema bepunktet:

Kategorie	Grund (Punkte)	Maximal mögliche Punktzahl
<b>Unterstützte sprachen</b>	Apex (+2) LWC (+2) Visualforce (+0,5) andere (+1)	5,5
<b>Automatische Erkennung falscher Positive</b>	Apex (+2) LWC (+2) Visualforce (+0,5) andere (+1)	5,5
<b>Anpassbarkeit</b>	Erstellen eigener Regeln (+1) Anpassen bestehender Regeln (+0,5) Herausnehmen bestehender Regeln (+1)	2,5
<b>Visualisierung</b>	Einbindung in das GitLab UI (+3) oder HTML-Ausgabe (+1) GitLab unabhängiges WebUI (+1)	3
<b>CD/CI Integration</b>	rangmäßig nach Aufwand (Kleinster zuerst): 1. (+4) 2. (+3) 3. (+2) 4. (+1)	4
<b>Wartungsaufwand</b>	Automatische Aktualisierung der mitgelieferten Regeln bei Update (+2)	2
<b>Datenspeicherort/ Datenschutz</b>	Intern durchgeführter Test (+2) Intern Speicherung der Testergebnisse (+1)	3
<b>Kosten</b>	rangmäßig nach Kosten (Kleinste zuerst): 1. (+4) 2. (+3) 3. (+2) 4. (+1)	4
<b>Dokumentation</b>	Dokumentation Vorhanden (+1) Kontaktmöglichkeit Vorhanden (+1) Lernplattform Vorhanden (+1) benötigt Dokumentation dritter (-1)	3
<b>Bestehende Tool-Landschaft &amp; Integrationen</b>	Bestandteil eines bereits Verwendeten Tools (+2) oder nur Bekannt oder andere Version benötigt (+1)	2

Tabelle 3: Punktevergabe der Kriterien für die Entscheidungsmatrix

Die Gewichtung der Kategorien wurde durch eine anonyme Umfrage festgehalten. In der Umfrage konnten die Salesforce-Programmierer innerhalb der dotSource SE, die oben genannten Kategorien von null bis fünf, anhand ihrer Wichtigkeit bewerten.

Wurde ein Kriterium mit fünf von einem Teilnehmer bewertet, so fließen diese Punkte doppelt ein, da diese Option ein Knockout-Kriterium, sprich das Tool muss diese Anforderungen erfüllen, darstellte.

Es nahmen sieben Teilnehmer an der Umfrage teil. Nach der Umfrage wurden die Punkte der Kategorien als relativer Prozentsatz zu den Gesamtpunkten festgehalten. Die entstandenen Prozente dienen als Gewichtung für die Entscheidungsmatrix:

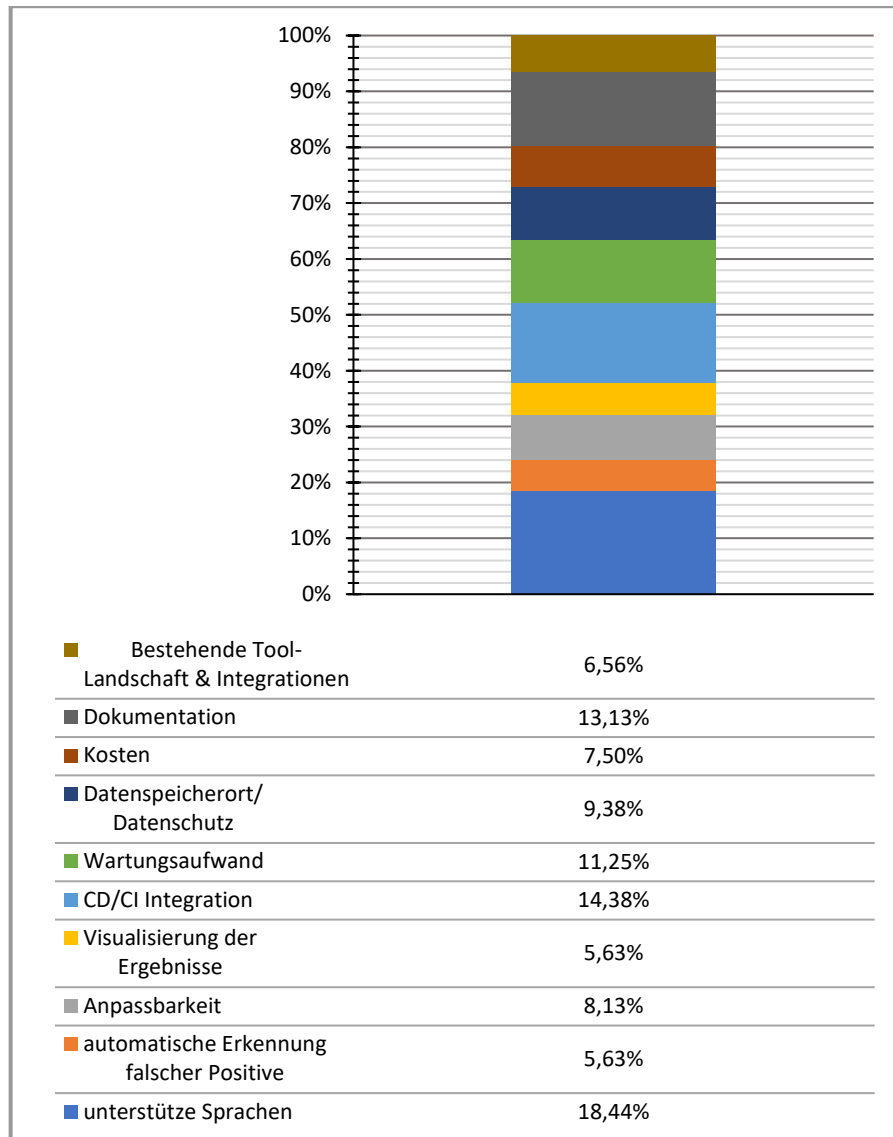


Abbildung 4: Ergebnisse der Umfrage zur Gewichtung der Bewertungskriterien

Das Gesamtergebnis für jedes Tool setzt sich somit aus der Summe der Produkte, der jeweiligen Punkte und Gewichtung in einer Kategorie zusammen.

Kategorie	Gewichtung	GitLab SAST (Ultimate)	Salesforce Code Analyzer	Snyk (Team)	SonarQube (Enterprise)
<b>Unterstützte Sprachen</b>	18,44%	3	5,5	3	3
<b>False Positive Detection</b>	5,63%	1	0	0	0
<b>Anpassbarkeit</b>	8,13%	1,5	2	1	1
<b>Visualisierung</b>	5,63%	3	1	2	1
<b>CD/CI Integration</b>	14,38%	4	3	2	1
<b>Wartungsaufwand</b>	11,25%	2	2	2	2
<b>Datenspeicherort</b>	9,38%	3	3	1	3
<b>Kosten</b>	7,50%	2	4	3	1
<b>Dokumentation</b>	13,13%	2	1	3	2
<b>Bestehende Tool-Landschaft &amp; Integrationen</b>	6,56%	2	2	1	1
<b>Ergebnis</b>	100%	2,525	2,7328125	2,0375	1,74375

Tabelle 4: Entscheidungsmatrix der SAST-Tools

Der Salesforce Code Analyzer unterstützt am meisten Salesforce-relevante Sprachen und sticht dadurch gegenüber den anderen Tools, welche alle nur Apex unterstützen, in dieser Kategorie klar hervor.

Eine automatische Erkennung falscher Positive existiert nur in GitLab Ultimate, aber nur für andere weniger relevante Sprachen. In den Kategorien Visualisierung und CI/CD Integration liegt das SAST-Tool von GitLab in Führung, was naheliegend ist, da das Tool von dem Versionsverwaltungssystem kommt, in welches integriert wird.

Der Wartungsaufwand fällt bei allen Tools ähnlich aus. So werden die mitgelieferten Regeln immer von dem jeweiligen Tool bei einem Update angepasst und erfordern keine maßgebliche Überarbeitung. Wobei die Bewertung sich verändern könnte, wenn man sie über einen längeren Zeitraum testweise implementiert und analysiert.

Den Datenschutzanforderungen wird nur Cloud-Plattform Snyk nicht gerecht, da Tests über die Cloud ausgeführt werden.

Die Kosten sind bei dem Salesforce Code Analyzer am geringsten, weil dieser kostenlos verwendbar ist. Snyk überwiegt jedoch in der Kategorie der Dokumentation, durch eine eigene Lernplattform.

Die Bewertung der Kategorie bestehende Tool-Landschaft und Integrationen fällt bei Snyk und SonarQube schlechter aus. Bei beiden ist eine Reimplementation oder eine Initialaufsetzung der Plattform nötig. Sie sind daher nicht vollständig Teil der bestehenden Tool-Landschaft.

Zusammenfassend lässt sich sagen, dass der Salesforce Code Analyzer als SAST-Tool sich für unsere Verwendungszwecke am meisten anbietet. In den nachfolgenden Kapiteln wird für dieses Tool ein Pipeline-Job erstellt und anschließend in ein Projekt implementiert.

## 2.2 Konzeptionelle Aufsetzung des Pipeline-Jobs

Um einen Pipeline-Job anzulegen und diesen auf Funktionalität zu überprüfen, wurde ein Test-Repository angelegt. Auf diesem Repository wurde in einem initialen Commit die Standardordnerstruktur eines Salesforce-Projektes hinzugefügt. Anschließend wurde eine leere YAML-Datei und ein GitLab Runner, mit Docker Ausführer, auf einer lokalen Maschine installiert und eingerichtet. Bestandteil des verwendeten `salesforce/cli:latest-full` Docker Images ist die Salesforce CLI und die Java-Version OpenJDK 11, was alle für den Salesforce Code Analyzer benötigten Abhängigkeiten abdeckt. Die Installation dieser erfolgt außerdem automatisch über Docker. Zudem werden die installierten Dateien nach Schließung des Containers von Docker automatisch bereinigt. Der Pipeline-Job und dessen Implementation wird somit, durch die Verwendung von Docker, um ein Vielfaches vereinfacht.

Argument (Kürzel)	Funktion
<b>--target (-t)</b>	Anpassung der Dateien welche im Sicherheitstest enthalten werden sollen. Pfade müssen als relative Pfade im Glob Pattern hinzugefügt werden.
<b>--engines (-e)</b>	Spezifikation der externen Scanner, welche beim Test verwendet werden sollen
<b>--category (-c)</b>	Anpassung der Regelkategorien, auf welche beim Test geprüft werden soll
<b>--severity_threshold (-s)</b>	Definiert ein Sicherheitsrisikolevel ab welcher der Sicherheitstest als fehlgeschlagen gelten soll. Wird verwendet, um in der Pipeline anzuzeigen das ein hohen Sicherheitsrisiko gefunden wurde und der Quellcode somit nicht in den Haupt-Branch gepushed werden sollte.
<b>--format (-f)</b>	Ansteuern des Ausgabe formates der test ergebnisse
<b>--outfile (-o)</b>	Gibt an in welche Datei die Ausgabe, in dem gewünschten Format, geschrieben werden soll

Tabelle 5: Bepunktung der Kategorien der Entscheidungsmatrix



Die Ausführung des Scanners wird über Argumente auf der Befehlszeile gesteuert. Ein Teil dieser sollen über CI/CD Variablen innerhalb des GitLab UI angesteuert und bearbeitet werden können. Zusätzlich werden in der YAML-Datei Variablen verwendet, um die Benutzerfreundlichkeit der Pipeline zu verbessern. In der Tabelle 5 wird eine Übersicht über die Funktionalität der einzelnen Argumente gegeben.

Die YAML-Datei wurde wie folgt aufbereitet:

```
1. stages:
2.   - test
3.
4. SAST-job:
5.   image: salesforce/cli:latest-full
6.   stage: test
7.   variables:
8.     SFCA_VERSION:
```

Abbildung 5: Stage und Konfiguration des Konzept-Jobs

Die ersten Zeilen dienen der Konfiguration des Pipeline-Jobs. Pipeline Jobs sind immer einer Stage zuzuweisen, weswegen wir diese ebenfalls anlegen. Im Image definieren wir das oben begründete, von Docker verwendete Image. Zudem findet sich hier eine Variable für die nachfolgende Installation des SAST-Tools.

```
9.   script:
10.    # INSTALLING SALESFORCE CODE-ANALYSER
11.    - sf plugins install @salesforce/sfdx-scanner@$SFCA_VERSION
12.
```

Abbildung 6: YAML-Code zur Installation des SAST-Tools mit anpassbarer Version

Der Salesforce code Analyzer, als Salesforce internes Produkt, ist ein Plug-in der SF CLI und ist demnach auch so zu installieren. Sollte in Projekten eine gewisse Version des Tools gefordert sein, kann diese in Zeile 8 über die \$SFCA\_VERSION Variable, definiert werden. Wird sie leer gelassen, wird die letzte stabile Version des Tools installiert. Nachfolgend werden die Argumente des SFCA vorbereitet. Die Formatierung der Codezeilen entspricht dabei der eines Linux Terminals, da das Docker Image auf Ubuntu basiert.

```

13. # PREPARING VARIABLES
14. # categories argument
15. - >
16.   if [ "$CATEGORIES" != "" ]; then
17.     export CATEGORIES="-c=$CATEGORIES"
18.   fi
19.
20. # severity threshold argument
21. - >
22.   if [ "$SEVERITY_THRESHOLD" != "" ]; then
23.     export SEVERITY_THRESHOLD="-s=$SEVERITY_THRESHOLD"
24.   fi
25.
26. #engine argument
27. - >
28.   if [ "$ENGINES" != "" ]; then
29.     export ENGINES="-e=$ENGINES"
30.   fi
31.
32. # target argument
33. - >
34.   if [ "$TARGETS" != "" ]; then
35.     export TARGETS="-t='$TARGETS'"
36.   fi
37.

```

Abbildung 7: Vorbereitung der Argumente für den Aufruf des SAST-Tools

Die Argumente dürfen beim Ausführen des Tools nicht ohne Spezifikation verwendet werden, ansonsten wirft es einen Fehler. Deswegen wird das Kürzel nur zur Variable hinzugefügt, wenn die Variable nicht leer ist. Wenn die Variable leer ist, wird das Kürzel nicht hinzugefügt, der dahinterliegende Wert bleibt somit leer und spielt für die Ausführung des SAST-Tools keine Rolle. Somit wird, wenn beispielsweise in der Variable \$CATEGORIES der Wert Security hinterlegt ist, daraus -c=Security.

Die Verwaltung der Variablen \$CATEGORIES, \$TARGETS, \$ENGINES, \$FORMAT, \$OUTFILE\_NAME, \$ADDITIONAL\_ARGS und \$SEVERITY\_THRESHOLD geschieht im GitLab UI. Die Einbindung in die Jobs erfolgt durch GitLab automatisch. Eine solche Konfiguration hat zum Vorteil, dass die Bearbeitung keinen neuen Commit im Repository benötigt und macht sie benutzerfreundlicher.

Einen Einblick in das GitLab UI zum Verwalten der Variablen findet sich im Anhang der Arbeit.

```

38. # prework for outfile argument
39. - >
40.   case "$FORMAT" in
41.     "junit")
42.       export OUTFILE_EXT="xml" ;;
43.     "table")
44.       export OUTFILE_EXT="" ;;
45.     *)
46.       export OUTFILE_EXT=$FORMAT ;;
47.   esac
48.
49. - mkdir sfcao
50.
51. # outfile argument
52. - >
53.   if [ "$OUTFILE_EXT" != "" ]; then
54.     export OUTFILE="-o=sfcao/$OUTFILE_NAME.$OUTFILE_EXT"
55.   fi
56.

```

Abbildung 8: Vorbereitung des Outfile-Aruments des SFCA

Die Ausgabe des Scanners wird anders vorbereitet. Da es Ausgabeformate gibt, wo sich das Format der Datei von der Dateierweiterung unterscheidet, wird eine variable \$OUTFILE\_EXT erstellt. Das JUnit-Format wird beispielsweise in eine XML-Datei geschrieben und bekommt durch die Variable diese Dateierweiterung zugeteilt. Bei dem Table-Ausgabeformat werden die Testergebnisse direkt in das Terminal des Pipeline-Jobs geschrieben und es entsteht keine separate Datei, weswegen die Dateierweiterung leer gesetzt wird. Für alle anderen Fälle, bei denen das Format mit der Dateierweiterung übereinstimmt, wird die \$OUTFILE\_EXT Variable mit der \$FORMAT Variable gleichgesetzt.

Die Bereitstellung des Artefaktes, also der Ausgabe im UI einer Merge Request, erfordert, dass der Pfad (siehe Zeile 72.), auf welcher der Runner die Artefakte vermutet, keine CI/CD Variable enthält. Deswegen wird ein Ordner mit festgelegtem Namen erstellt, welcher als Pfad verwendet werden kann.

Das Outfile-Argument setzt sich aus dem Kürzel, dem Ordner, der im UI konfigurierten Benennung und der automatisch erstellten Dateierweiterung zusammen. Eine Überprüfung, ob die Dateierweiterung leer ist, ist aufgrund des Table-Formats erforderlich. Die \$OUTFILE Variable bleibt im Fall, dass keine Dateierweiterung existiert, somit ebenfalls leer und wird vom Scanner nicht betrachtet. Somit kann die Ausgabe in eine Datei vermieden werden.

```

57. # format argument
58. - >
59.   if [ "$FORMAT" != "" ]; then
60.     export OUTFILE_FORMAT="-f=$FORMAT"
61.   fi
62.

```

Abbildung 9: Vorbereitung des Format-Aruments des SFCA

Das Format-Argument wird ähnlich zu den Argumenten in den Zeilen 13 bis 37 auf Leerheit überprüft, um einen fehlschlagenden Test aufgrund von fehlender Spezifikation zu vermeiden.

```
63. # RUNNING SECURITY SCAN
64. - sf scanner run $FORMAT $OUTFILE $TARGETS $ENGINES $CATEGORIES $SEVERITY_THRESHOLD
   $ADDITIONAL_ARGS
65.
```

Abbildung 10: YAML-Code zur Durchführung des Sicherheitstests mittels SFCA

Der Sicherheitstest kann nun ausgeführt werden. Die Variable `$ADDITIONAL_ARGS` wird nicht verändert und wird wie konfiguriert in der Ausführung verwendet. Über diese lassen sich weitere Konfigurationen realisieren, welche Bestandteil des Scanners sind. Tatsächlich könnten alle anderen Argumente ebenfalls über diese Variable eingerichtet werden. Dies erfordert aber Vorwissen zu dem Befehlsaufruf des SAST-Tools. Um dies zu vermeiden und eine schnelle Adaption zu begünstigen, wurden die wichtigsten Argumente wie beschrieben aufbereitet.

```
66. #UPLOADING ARTIFACTS
67. artifacts:
68.   expose_as: "SAST scan report"
69.   when: always
70.   paths: ["sfcao/"]
71.
```

Abbildung 11: Upload der Testergebnisse des Sicherheitstests

Als letzter Bestandteil des konzeptionierten Pipeline-Jobs werden alle Dateien in dem Ausgabeordner als Artefakt hochgeladen und zu einer Merge Request hinzugefügt. Die Artefakte können dann auf der Seite der Merge Request heruntergeladen werden. Der damit entstandene Job realisiert somit die statische Code-Analyse in einem Repository für ein Salesforce-Projekt.

Der Job muss nur noch auf die projektspezifischen Anforderungen konfiguriert werden. Zu beachten ist jedoch, dass, sollte ein anderer Runner mit einem potenziell anderen Executer verwendet werden, die Befehle dementsprechend angepasst werden müssen.

## 3 Implementierung

### 3.1 Analyse der derzeitigen CI/CD Pipeline

Zum Schluss der Arbeit soll der im Konzept entstandene Pipeline-Job in ein Salesforce-Projekt implementiert werden. Damit die Implementation so reibungslos wie möglich verlaufen kann, wurde im ersten Schritt die in der derzeitigen Pipeline für die Implementation relevanten Bestandteile identifiziert. Dabei wurden die Stages im Job, sowie der Docker Runner, der Pipeline, mit dem `salesforce/cli:latest-full` Image, festgehalten.

```
...  
25. image: "salesforce/cli:latest-full"  
...  
37. stages:          # List of stages for jobs, and their order of execution  
38.   - preliminary-testing  
39.   - create-scratch-org  
40.   - test-scratch-org  
41.   - package  
42.   - staging  
43.   - production  
...
```

Abbildung 12: Die für die Implementation wichtigen teile der existierenden Pipeline

Das Docker Image wird standardmäßig für alle Jobs verwendet, weil das Image Keyword keinen Job untergeordnet ist.

### 3.2 Implementierung des Konzept-Jobs

Die Stage, in welcher die statische Code-Analyse durchgeführt werden soll, ist die `preliminary-testing` Stage. Dazu musste die Zeile 8 im Konzept, auf die Stage gesetzt werden.

Die im Konzept Job wurden Variablen im GitLab UI definiert. Dies sollte, für die erste Implementation nicht geschehen. Die Variablen wurden deswegen direkt in der Konfiguration des Jobs verankert.

Außerdem wird, da das Salesforce Docker Image standardmäßig verwendet, die Zeile 5 aus dem Konzept Job entfernt.

Neben diesen kleinen Veränderungen, an der Konfiguration, konnte der Skript-Teil des Konzept-Jobs ohne eingriffe übernommen werden.

```
...
53. SAST-job:
54.   stage: preliminary-testing
55.   variables:
56.     SFCA_VERSION:
57.     CATEGORIES:
58.     ENGINES:
59.     TARGETS:
60.     OUTFILE_FORMAT:
61.     OUTFILE_NAME:
62.     SEVERITY_THRESHOLD:
63.     ADDITIONAL_ARGS:
64.
65-126. Script und Upload der Artefakte wie im Konzept-Job
...
```

Abbildung 13: Implementierter Konzept-Job

Das SAST-Tool wurde somit implementiert und eine projektspezifische Konfiguration des Tools kann nun stattfinden.

## 4 Fazit

Das Ziel dieser Arbeit lag darin, die statische Code-Analyse in einer CD/CI Pipeline für Salesforce-Projekte zu implementieren und somit exemplarisch für zukünftige Projekte zugänglich zu machen. Dazu musste ein Tool, welches diese umsetzt, ausgewählt und implementiert werden. Die Auswahl wurde zielführend durch eine Klare Festlegung der Kategorien und Zusammenarbeit mit dem Programmieren innerhalb der dotSource SE getroffen.

Der entstandene Konzept-Job für eine Pipeline, welche mit Docker arbeitet, bietet eine schnelle Möglichkeit die Sicherheitstests durchzuführen.

Zukünftig soll das Konzept in einem separaten Repository weiterentwickelt werden, um die Implementation mittels eines YAML-Templates ähnlich zum SAST-Tool von GitLab zu ermöglichen. Auch die Visualisierung wird sich in der kommenden Zeit verändern, denn die HTML-Ausgabe des Salesforce Code Analyzers ist anpassbar.

Die dotSource SE tut somit einen wichtigen ersten Schritt um die verwendeten DevOps Methoden und Kapazitäten, im immer wichtig werdenden Thema der Cybersicherheit, zu erweitern. Dabei sind SAST-Tools, aber nur ein kleiner Teil der Möglichkeiten, um die Sicherheit des Quellcodes zu verbessern. Neben diesen Tools existieren Dynamic Security Application Testing (DAST) Interactive Application Security Testing (IAST) und Runtime Application Self-Protection (RASP) Tools, welche ebenfalls zukünftig implementiert und ihre Funktionalität erhoben werden müssen.

## Literaturverzeichnis

- [Atl24a] Atlassian: DevOps-Prinzipien | Atlassian, 2024,  
<https://www.atlassian.com/de/devops/what-is-devops>, Abgerufen am: 12.03.2024
- [Atl24b] Atlassian: Was ist DevOps? | Atlassian, 2024,  
<https://www.atlassian.com/de/devops>, Abgerufen am: 12.03.2024
- [FHK18] Forsgren, N.; Humble, J.; Kim, G.: Accelerate, First edition, IT Revolution, Portland, Oregon, 2018
- [Git24a] GitLab: Static Application Security Testing (SAST) | GitLab, 2024,  
[https://docs.gitlab.com/ee/user/application\\_security/sast/](https://docs.gitlab.com/ee/user/application_security/sast/), Abgerufen am: 21.03.2024
- [Git24b] GitLab: Customize rulesets | GitLab, 2024,  
[https://docs.gitlab.com/ee/user/application\\_security/sast/customize\\_rulesets.html](https://docs.gitlab.com/ee/user/application_security/sast/customize_rulesets.html),  
Abgerufen am: 22.04.2024
- [Jez18] Jez Humble: Patterns - Continuous Delivery, 2018,  
<https://continuousdelivery.com/implementing/patterns/>, Abgerufen am: 12.03.2024
- [Lub23] Luber, S.: Was ist SAST?, Security-Insider, 2023
- [OWA24] OWASP Foundation: Source Code Analysis Tools | OWASP Foundation, 2024,  
[https://owasp.org/www-community/Source\\_Code\\_Analysis\\_Tools](https://owasp.org/www-community/Source_Code_Analysis_Tools), Abgerufen am: 22.03.2024
- [Sal24a] Salesforce: Analyst Reports, 2024,  
<https://www.salesforce.com/company/recognition/analyst-reports/>, Abgerufen am: 21.03.2024
- [Sal24b] Salesforce: Salesforce develops the technology, the partnerships, and the communities that help companies connect with customers, 2024,  
<https://www.salesforce.com/company/our-story/>, Abgerufen am: 22.03.2024
- [Sal24c] Salesforce: Salesforce Developers, 2024,  
<https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/custom-config.html>, Abgerufen am: 22.04.2024
- [Sal24d] Salesforce: Salesforce Developers, 2024,  
<https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/setup.html>, Abgerufen am: 22.04.2024



- [Sal24e] Salesforce: Salesforce Developers, 2024,  
<https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/code-analyzer-command-reference.html>, Abgerufen am: 22.04.2024
- [Sca24] Scaled Agile Framework: DevOps - Scaled Agile Framework, 2024,  
<https://scaledagileframework.com/devops/>, Abgerufen am: 14.03.2024
- [Sny24a] Snyk: Configure Snyk Code | Snyk User Docs, 2024, <https://docs.snyk.io/scan-with-snyk/snyk-code/configure-snyk-code>, Abgerufen am: 22.04.2024
- [Sny24b] Snyk: How Snyk handles your data | Snyk User Docs, 2024,  
<https://docs.snyk.io/working-with-snyk/how-snyk-handles-your-data>, Abgerufen am: 22.04.2024
- [Sny24c] Snyk: Produktvarianten & Preise | Für Teams jeder Größe | Snyk, 2024,  
<https://snyk.io/de/plans/>, Abgerufen am: 22.04.2024
- [Sny24d] Snyk: SonarQube 10.5, 2024, <https://docs.sonarsource.com/sonarqube/latest/>,  
Abgerufen am: 22.04.2024
- [Sny24e] Snyk Learn: Free Interactive Secure Development Training, 2024,  
<https://learn.snyk.io/>, Abgerufen am: 22.04.2024
- [Son24] SonarSource: Installation introduction | SonarQube Docs, 2024,  
<https://docs.sonarsource.com/sonarqube/latest/setup-and-upgrade/install-the-server/introduction/>, Abgerufen am: 09.04.2024

## Anlagenverzeichnis

Anlage 1: Datalierte Entscheidungsmatrix der SAST-Tools.....	IX
Anlage 2: gesamte konzeptionelle YAML-Datei .....	X
Anlage 3: GitLab UI zur Verwaltung von CI/CD Variablen.....	XI

Bewertungskriterien	Gewicht	GitLab SAST (Ultimate)	Werte	Salesforce Code Analyzer	Werte2	Snyk (Team)	Werte3	SonarQube (Enterprise)	Werte4
Unterstützte Sprachen	18,44%	Apex + andere	3	Apex LWC Visualforce + andere	5,5	Apex + andere	3	Apex + andere	3
False Positive Detection	5,63%	Für die Sprachen Go und Ruby	1	-	0	-	0	-	0
Anpassbarkeit	8,13%	Herausnehmen von Regeln Ändern einiger Eigenschaften bestehender Regeln (Bspw. Beschreibung oder Risikostufe)	1,5	herausnehmen von Regeln Eigene Regeln hinzufügender	2	Eigene Regeln hinzufügender	1	herausnehmen von Regeln	1
Visualisierung	5,63%	Visuelle Einbindung im bereits etablierten Versionsverwaltungssystem (Wunschzustand)	3	Eigene Visualisierung durch HTML-Ausgabeformat	1	Eigene Visualisierung durch WebUI oder HTML Ausgabeformat	2	Eigene Visualisierung über WebUI	1
CD/CI Integration	14,38%	über YAML-Datei (mitunter nur 2 Codezeilen)	4	SF CLI Befehle in Job script der YAML-Datei benötigt Bibleothenken welche im Job heruntergeladen werden	3	Snyk Plattform aufsetzen benötigt Bibleothenken welche im Job heruntergeladen werden benötigte setup für das Projekt Snyk CLI Befehle in Job script der YAML-Datei	2	benötigt aufgesetzten SonarQube Server benötigt Bibleothenken welche im Job heruntergeladen werden benötigt Konfiguration für das Projekt Sonar-Scanner CLI Befehle in Job script der YAML-Datei	1
Wartungsaufwand	11,25%	Automatische Aktualisierung der mitgelieferten Regeln bei Update	2	Automatische Aktualisierung der mitgelieferten Regeln bei Update	2	Automatische Aktualisierung der mitgelieferten Regeln bei Update	2	Automatische Aktualisierung der mitgelieferten Regeln bei Update	2
Datenspeicherort	9,38%	lokale analyse und ausgabe	3	lokale analyse und ausgabe	3	Analyse über Cloud Ergebnisse über Cloud und zusätzlich lokal	1	Analyse lokal erfordert aber internen server Ausgabe über internen server	3
Kosten	7,50%	GitLab Ultimate: Ca. 995 Pro Entwickler genauer nach Absprache benötigt für Anpassung, Visualisierung und die Erkennung falscher Positive	2	Kostenlos	4	Snyk Team (nur Snyk Code); 29\$/Monat/Entwickler benötigt für unlimitierte Tests	3	Enterprise Edition: \$21.000/jahr - \$252000/jahr 1m LOC - 100m LOC benötigt für Apex, interne bearbeitung	1
Dokumentation	13,13%	Dokumentation vorhanden Kontaktmöglichkeit vorhanden	2	Dokumentation vorhanden Erfordert externe Dokumentationen zum Anpassen von Regeln Kontaktmöglichkeit vorhanden	1	Dokumentation vorhanden Kontaktmöglichkeit vorhanden Lernplattform vorhanden	3	Dokumentation vorhanden Kontaktmöglichkeit vorhanden	2
Bestehende Tool-Landschaft & Integrationen	6,56%	GitLab intern	2	Salesforce intern	2	Bereits firmenintern bekannt	1	Andere Version als Nötig bereits firmenintern implementiert	1
Ergebnis	100,00%		2,525		2,7328125		2,0375		1,74375




































Anlage 1: detaillierte Entscheidungsmatrix der SAST-Tools

```

1. stages:
2.   - test
3.
4. SAST-job:
5.   image: salesforce/cli:latest-full
6.   stage: test
7.   variables:
8.     SFCA_VERSION:
9.   script:
10.  # INSTALLING SALESFORCE CODE-ANALYSER
11.  - sf plugins install @salesforce/sfdx-scanner@$SFCA_VERSION
12.
13.  # PREPARING VARIABLES
14.  # categories argument
15.  - >
16.    if [ "$CATEGORIES" != "" ]; then
17.      export CATEGORIES="-c=$CATEGORIES"
18.    fi
19.
20.  # severity threshold argument
21.  - >
22.    if [ "$SEVERITY_THRESHOLD" != "" ]; then
23.      export SEVERITY_THRESHOLD="-s=$SEVERITY_THRESHOLD"
24.    fi
25.
26.  #engine argument
27.  - >
28.    if [ "$ENGINES" != "" ]; then
29.      export ENGINES="-e=$ENGINES"
30.    fi
31.
32.  # target argument
33.  - >
34.    if [ "$TARGETS" != "" ]; then
35.      export TARGETS="-t='$TARGETS'"
36.    fi
37.
38.  # prework for outfile argument
39.  - >
40.    case "$FORMAT" in
41.      "junit")
42.        export OUTFILE_EXT="xml" ;;
43.      "table")
44.        export OUTFILE_EXT="" ;;
45.      *)
46.        export OUTFILE_EXT=$FORMAT ;;
47.    esac
48.
49.  - mkdir sfcao
50.
51.  # outfile argument
52.  - >
53.    if [ "$OUTFILE_EXT" != "" ]; then
54.      export OUTFILE="-o=sfcao/$OUTFILE_NAME.$OUTFILE_EXT"
55.    fi
56.
57.  # format argument
58.  - >
59.    if [ "$FORMAT" != "" ]; then
60.      export OUTFILE_FORMAT="-f=$FORMAT"
61.    fi
62.
63.  # RUNNING SECURITY SCAN
64.  - sf scanner run $FORMAT $OUTFILE $TARGETS $ENGINES $CATEGORIES $SEVERITY_THRESHOLD $ADDITIONAL_ARGS
65.
66.  #UPLOADING ARTIFACTS
67.  artifacts:
68.    expose_as: "SAST scan report"
69.    when: always
70.    paths: ["sfcao/"]
71.

```

Anlage 2: gesamte konzeptionelle YAML-Datei

CI/CD Variables </> 7		Reveal values	Add variable
Key ↑	Value	Environments	Actions
<b>ADDITIONAL_ARGS</b>  For additional arguments refer to <a href="https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/code-analyzer-command-reference.html">https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/code-analyzer-command-reference.html</a> Expanded	***** 	All (default) 	 
<b>CATEGORIES</b>  One or more categories of rules to run. Specify multiple values as a comma-separated list. Leave empty for Default. Default is all Expanded	***** 	All (default) 	 
<b>ENGINES</b>  Specifies one or more engines to run. Submit multiple values as a comma-separated list. Expanded	***** 	All (default) 	 
<b>OUTFILE_FORMAT</b>  Set the format of the output. Supported formats include csv, xml, json, html, sarif, table, junit Expanded	***** 	All (default) 	 
<b>OUTFILE_NAME</b>  Set the Name of the artifact file. Expanded	***** 	All (default) 	 
<b>SEVERITY_THRESHOLD</b>  Throws an error when violations are found with equal or greater severity than the provided value. Values are 1 (high), 2 (moderate), and 3 (low). Exit code is the most severe violation. Using this flag also invokes the --normalize-severity flag. Expanded	***** 	All (default) 	 
<b>TARGETS</b>  Specifies the source code location. Can use glob patterns. Specify multiple values as a comma-separated list. Leave empty for Default. Default is ".". Expanded	***** 	All (default) 	 

Anlage 3: GitLab UI zur Verwaltung von CI/CD Variablen