S&N INVENT

Security Automation in OpenShift Build Pipelines

Software Security Testing ab dem ersten Tag des Software Development Lifecycles

Lukas Voß

30.04.2020

S&N INVENT

Inhaltsverzeichnis

1 Motivation / Notwendigkeit von Security Automation in Build Pipelines

In der Softwareentwicklung sind aktuell zwei Trends zu verzeichnen.

Der erste Trend betrifft die Sicherheit von Applikationen. Cyber-Risiken werden als „*eine der größten Bedrohungen für den Erfolg der Digitalisierung*“[1] angesehen. Laut BSI waren 70 Prozent der Unternehmen und Institutionen in Deutschland bereits Opfer von Cyber-Angriffen. Die Anzahl der Angriffe nimmt immer weiter zu, ebenso wie die Anzahl und Varianten der verschiedenen Schadprogramme. Aus diesem Grund muss Software hohen Sicherheitsstandards genügen.[2]

Ein weiterer Trend ist, Applikationen containerisiert zu entwickeln. Schlagwörter wie DevOps und CI/CD fallen immer häufiger. Diese Modelle sollen Softwareentwicklung und -betrieb vereinfachen und effizienter gestalten. Container-Plattformen und Microservice-Architekturen unterstützen Unternehmen bei der Umsetzung dieser Modelle.

Diese beiden Trends führen zu einer stetig steigenden Notwendigkeit, Software bereits ab dem ersten Tag der Entwicklung auf Sicherheit zu prüfen. Das Paradigma DevOps wird auf DevSecOps erweitert, um Aufmerksamkeit auf diesen Aspekt zu lenken. Unternehmen können hier viele Kosten einsparen, zum einen, wenn sie Angriffe verhindern und zum anderen, wenn Sicherheitsprobleme frühzeitig behoben werden.

Auf welchen Ebenen sollte die Softwaresicherheit getestet werden und mit welchen Maßnahmen wird sie erhöht? Diese Frage soll in diesem Whitepaper beantwortet werden.

Eine Voraussetzung für sichere Software sollte eine sichere Ausführungsumgebung sein. Container-Plattformen wie Kubernetes können hier bereits viele Vorteile gegenüber anderen Betriebsformen bieten. In diesem Whitepaper wird die auf Kubernetes aufbauende Container-Plattform *Red Hat OpenShift* als Basisinfrastruktur referenziert.

[1] (BSI, 2018), S. 15
[2] vgl. ebd., S. 15 ff.

2 Security in jedem Schritt der Build Pipeline

Der erste Schritt zu einer sicheren Software liegt in der Sicherheit der ausführenden Infrastruktur. *Red Hat OpenShift* bietet hier in der Standardkonfiguration bereits viele Vorteile als Infrastruktur:

- Isolation von Namespaces durch das Software Defined Network (SDN)
- Feingranulare Steuerung über Policies (Service Mesh *Maistra*)
- Isolation von Containern durch die Container Engine
- Nutzer-Rechtesteuerung (RBAC)
- Automatische Patches
- und vieles mehr …

Im zweiten Schritt sollte die Sicherheit der Software selbst betrachtet werden. Eine abgesicherte Infrastruktur alleine kann nicht ausreichen, um eine Software zu schützen, die mit Clients kommuniziert. Verwundbare Software, welche zum Beispiel veraltete Bibliotheken nutzt oder (zu) offene Schnittstellen bereitstellt, kann angegriffen werden. Datenverlust, Datendiebstahl und Überlastung der Anwendung sind nur wenige der möglichen Auswirkungen von Angriffen, welche unter Umständen zu hohen Kosten führen.

Auf welchen Ebenen sollte Software nun getestet werden? Zur Veranschaulichung ist in folgendem Schaubild eine Build Pipeline dargestellt. In der Darstellung wird bewusst auf Pipeline Schritte wie (funktionale) Tests, Integrationstests und Auslieferung auf unterschiedlichen Ausführungsumgebungen verzichtet. Diese Schritte werden nach wie vor ausgeführt. Das Schaubild soll lediglich die erforderlichen Erweiterungen anhand einer einfachen Build Pipeline beschreiben.

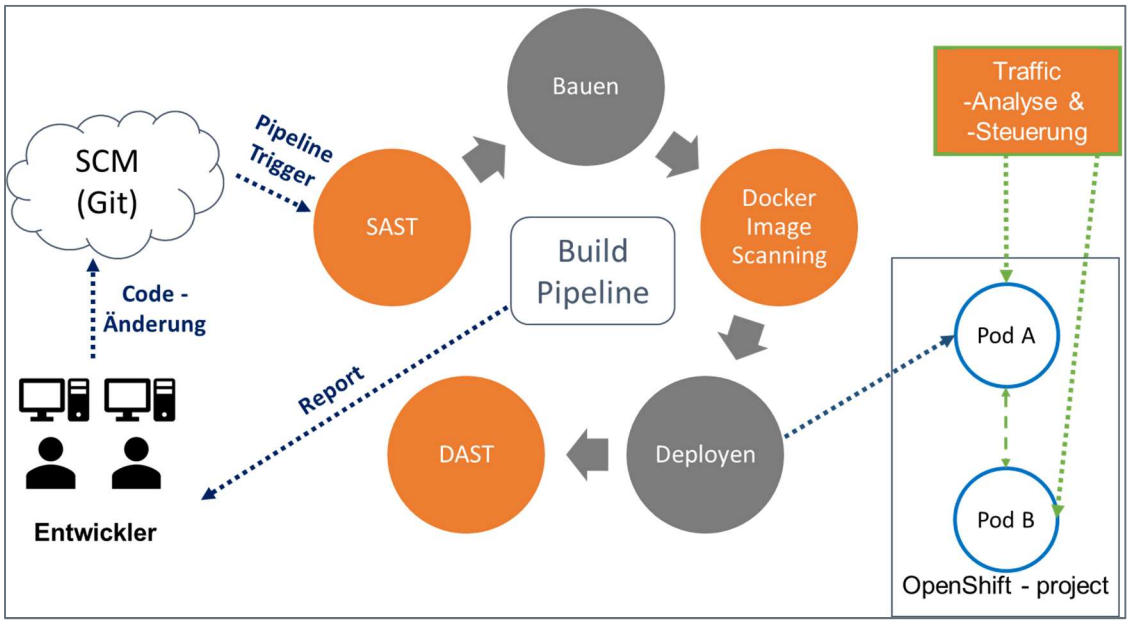

Abbildung 1: Ziel - Build Pipeline

Das Ziel soll sein, dass ein Entwickler kurz nach dem Hochladen einer Code-Änderung eine Rückmeldung erhält, falls ein neues Sicherheitsproblem in der Software gefunden wurde, um

sich unmittelbar mit dem Problem auseinandersetzen zu können. Das ist wichtig, damit sich die Sicherheitslücke in der weiteren Entwicklung nicht fortpflanzen kann.

Software kann auf verschiedenen Ebenen getestet werden. Diese sind der Sourcecode (Static Application Security Testing), das ausführende Docker Image (Docker Image Scanning) und die laufende Anwendung (Dynamic Application Security Testing). Bei der laufenden Anwendung kann zusätzlich eine ständige Netzwerkverkehranalyse und -steuerung durchgeführt werden, um die Sicherheit weiter zu erhöhen oder Angriffe zu identifizieren. Die verschiedenen Möglichkeiten werden nachfolgend weiter ausgeführt.

2.1 Static Application Security Testing (SAST)

Die erste Überprüfung auf Schwachstellen einer Anwendung kann am Programmcode vorgenommen werden. Das Static Application Security Testing, kurz SAST, beschäftigt sich mit eben dieser Analyse des Programmcodes einer Software. Durch diese Art der Sicherheitsanalyse ist es möglich, Sicherheitslücken aufgrund von Programmierfehlern oder eingebundenen Bibliotheken zu lokalisieren, welche Verwundbarkeiten beinhalten.

SAST beinhaltet die Analyse von Sourcecode, Bytecode sowie Binärdateien. SAST-Tools versuchen dabei anhand von bekannten Mustern Sicherheitsrisiken aufzudecken. Generierte False-Positives müssen von den tatsächlich gefundenen Sicherheitsrisiken separiert werden, damit das SAST aussagekräftig bleibt.

In einer Build Pipeline sollte SAST nach jeder Codeänderung durchgeführt werden, damit Entwickler neu aufgetretene Fehler unmittelbar beheben können. Zudem sollte bereits die IDE des Entwicklers eine Überprüfung auf Sicherheitsrisiken durchführen.

Beispiele für Tools: SonarQube, OWASP Dependency Check, Veracode, Black Duck

2.2 Docker Image Scanning

Das Docker Image Scanning ist die Analyse der verschiedenen Schichten (Layer) eines Docker Images.

Die Analyse von Docker Images ist die Überprüfung einer Applikation, nachdem sie als Docker Image gebaut, aber noch nicht gestartet wurde. Sie überprüft jede Schicht eines Images auf Schwachstellen. Dabei werden die Binärdateien und die installierten Softwarekomponenten analysiert und gegen eine Datenbank mit bekannten Verwundbarkeiten abgeglichen.

Die nachfolgende Abbildung zeigt das Ergebnis einer Docker Image Analyse in *Red Hat Quay*. Im nächsten Schritt kann ein Entwickler sich mit den Verwundbarkeiten auseinanderzusetzen, um Maßnahmen abzuleiten, wie etwa Bibliotheken in dem Image zu ersetzen oder zu aktualisieren.

Abbildung 2: Docker Image Scanning – Red Hat Quay, Quelle: quay.io

Beispiele für Tools: Anchore Engine, Red Hat Quay, Black Duck OpsSight, Aqua Security, Twistlock

2.3 Dynamic Application Security Testing (DAST)

Das Dynamic Application Security Testing, kurz DAST, erlaubt die Analyse des Laufzeitverhaltens von Software. Die DAST-Tools testen die Anwendung im laufenden Zustand. Dabei spricht man auch von einem Black-Box-Test, da die laufende Anwendung von außen, ohne Einsicht des Programmcodes, getestet wird. Auch DAST verfolgt den Ansatz, Sicherheitsrisiken aufgrund von bekannten Mustern aufzudecken.

Die Analyse von Software erfolgt dabei durch automatisiertes Angreifen und Client-seitige Tests. Diese Angriffe können in dem Versuch bestehen, Schadcode zu injizieren, Speicherüberläufe zu verursachen, unsichere Schnittstellen zu finden oder auf beliebige andere Sicherheitslücken zu testen.

Beispiele für Tools: OWASP Zed Attack Proxy, Arachni, Veracode

2.4 Netzwerkverkehranalyse und -steuerung

Die Überwachung des Netzwerkverkehrs, d. h. das Mitlesen von Netzwerkpaketen, bietet unterschiedliche Analysemöglichkeiten. Zum einen können die gewonnenen Informationen zum Verbessern der Performanz von Webapplikationen genutzt werden, zum anderen können die Informationen auch zur Verbesserung der Sicherheit und Steuerung des Netzwerks genutzt werden.

Die Analyse von Netzwerkverkehr erfolgt dabei in der Regel in Echtzeit und kann von Layer 2 bis Layer 7 implementiert sein. Die Netzwerkverkehranalyse kann auch genutzt werden, um Unregelmäßigkeiten aufzudecken. Für die Analyse und Kontrolle von Netzwerkverkehr kann zum Beispiel ein Service Mesh implementiert werden.

Ein Service Mesh ist eine Netzwerkinfrastruktur, welche den Netzwerkverkehr zwischen verschiedenen (Micro-) Services steuert und überwacht. Das Service Mesh kann genaue Regeln durchsetzen, welche Anwendungen miteinander kommunizieren können und über welchen Port. Außerdem kann der Netzwerkverkehr zu Analysezwecken aufgezeichnet werden, um Informationen über Auslastung und Sicherheit zu erhalten. Viele Service Meshs verfügen über weitere Funktionen, wie zum Beispiel ein Load Balancing, welches den Netzwerkverkehr zwischen verschiedenen parallel ausgeführten Versionen einer Anwendung aufteilen kann und die Möglichkeit, in die Netzwerkkommunikation einzugreifen.

Ein solches Service Mesh ist *Maistra*, welches in *OpenShift* über einen Operator installiert werden kann. *Maistra* baut auf die Service Mesh-Lösung *Istio* auf. Microservices kommunizieren nicht mehr direkt, der gesamte Netzwerkverkehr wird über sogenannte Sidecar Proxies umgeleitet. Dazu ist es notwendig, jeden Service (Pod) mit einem eigenen Sidecar Proxy auszustatten. Die Sidecar Proxies werden von dem Istio Control Plane gesteuert und konfiguriert.

Abbildung 3 zeigt, wie die Kommunikation zwischen einem Client und der Anwendung abläuft. Die eingehende sowie ausgehende Kommunikation mit den einzelnen Microservices erfolgt jeweils über die Sidecars, welche als zusätzliche Docker Container in jeden Pod injiziert worden sind. Der Client und der (Micro-)Service merken hier keinen Unterschied gegenüber der Kommunikation ohne Sidecars. Die Sidecars ermöglichen die Aufzeichnung des Netzwerkverkehrs, eine gezielte Umsetzung von Kommunikationsrichtlinien und eine Verschlüsselung der Kommunikation, ohne dass die Microservices stark angepasst werden müssen.

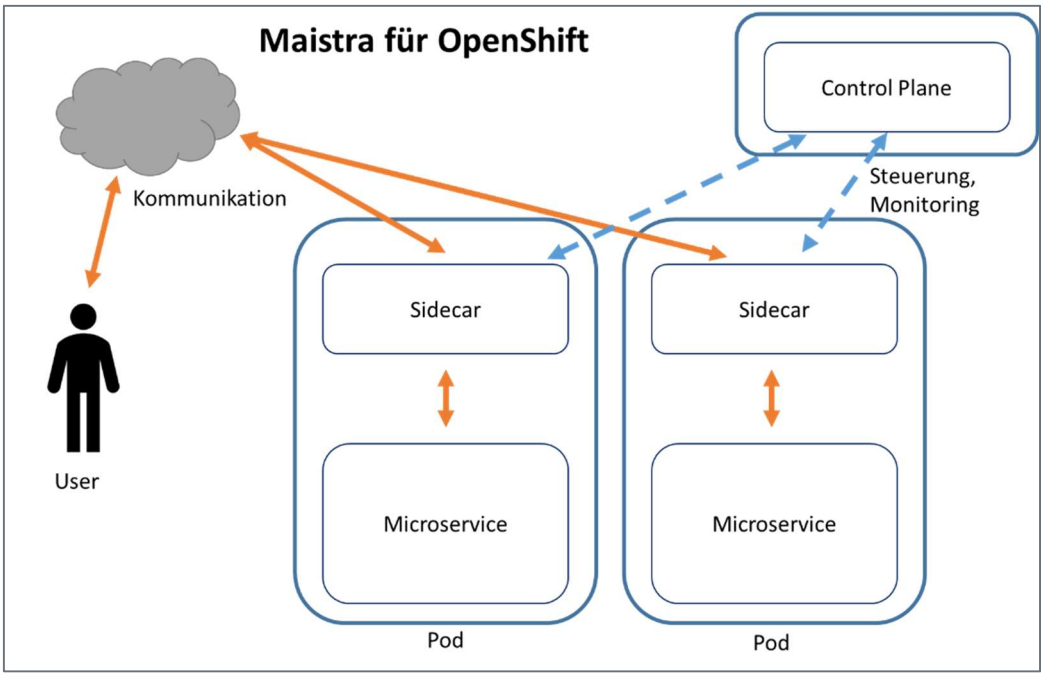

Abbildung 3: Service Mesh Maistra

Eine Alternative zum Service Mesh kann eine Web Application Firewall (WAF) sein. WAFs fungieren als zusätzliche Schicht zwischen Client und Webserver. Der gesamte Netzwerkverkehr wird über die WAF umgeleitet. Diese kann dann Regeln durchsetzen, um schädliche Internetpakete zu unterbinden. Die WAF kann dabei Cookies, URLs und andere Eigenschaften in den Internetpaketen prüfen. Dadurch können viele Bedrohungen abgewendet werden (zum Beispiel XSS-Angriffe, DoS-Angriffe, SQL-Injection, Malware).

Beispiele für Tools: Maistra, NAXSI (WAF)

3 Fazit

Softwaresicherheit kann auf verschiedene Weisen erhöht werden. Hier sind Methoden vorgestellt worden, welche insbesondere in Hinblick auf die Entwicklung von Microservices in einer *OpenShift*-Umgebung ausgewählt wurden.

Sicherheitstests ab dem ersten Tag des Software Development Lifecycles können erhebliche Kosteneinsparungen bedeuten. Nicht nur, dass unter Umständen ein späterer Angriff auf das Produktivsystem mit all seinen Folgen (Reputationsverlust, Datenverlust, Schadenersatzforderungen, Downtime) verhindert werden kann. Zusätzlich werden Kosten eingespart, wenn Fehler so früh wie möglich erkannt und behoben werden: Entwickler wiederholen diesen Fehler wahrscheinlich nicht weiter in ihrem Code, der Fehler wird sich nicht fortpflanzen und Auswirkungen auf andere Softwareabschnitte bleiben aus.

Wichtig ist natürlich nicht nur, dass die Software sicher ist, auch die Infrastruktur muss sicher sein. Dabei unterstützen die zahlreichen Features von *OpenShift*, welche um das Service Mesh *Maistra* oder eine WAF erweitert werden können.

Softwaresicherheit ist nicht unbedingt schwer umzusetzen, wenn man weiß welche Tools wie genutzt werden können und die Entwickler sensibilisiert sind. Ein wesentlicher Aufwandstreiber ist dabei die Definitionsfindung, was im Kontext der Gesamtlösung als sicher anzusehen ist. Am Ende der Definitionsfindung muss den Projektbeteiligten klar sein, welche Maßnahmen aus welchem Grund ergriffen und mit welchen Tools diese Ziele erreicht werden. In jedem Fall lohnt sich der Mehraufwand aber langfristig.

Autor

Lukas Voß

B Sc. Wirtschaftsinformatik und Bankkaufmann

Über S&N Invent

S&N Invent ist ein bundesweit tätiges IT-Unternehmen mit einem umfassenden Leistungsportfolio über die gesamte Wertschöpfungskette des IT-Lifecycles. Wir entwickeln gemeinsam mit unseren Kunden Lösungen, setzen Projekte um und schaffen damit digitale Mehrwerte.

Das Leistungsspektrum reicht von klassischen Mainframe-Architekturen über moderne Java/JEE Web- und Portalarchitekturen bis hin zu neuesten Technologien im Cloud- und Mobile-Bereich. Agile Projekte mit hohem Qualitätsanspruch, gewährleistet durch modernes Continuous Quality Management, sind für uns in zahlreichen Projekten gelebter Standard.

Insgesamt sind in der S&N Group ca. 380 feste Mitarbeiterinnen und Mitarbeiter an acht Standorten in Deutschland sowie dem Nearshore-Standort Budapest beschäftigt. Damit können wir unsere Kunden mit umfassender Kompetenz und regionaler Nähe bestens betreuen. Gleichzeitig sind wir so in der Lage, große und komplexe Projekte in Time und Budget erfolgreich umsetzen zu können.

Abbildungsverzeichnis

Literaturverzeichnis

BSI. (2018). Lagebericht der IT-Sicherheit in Deutschland.