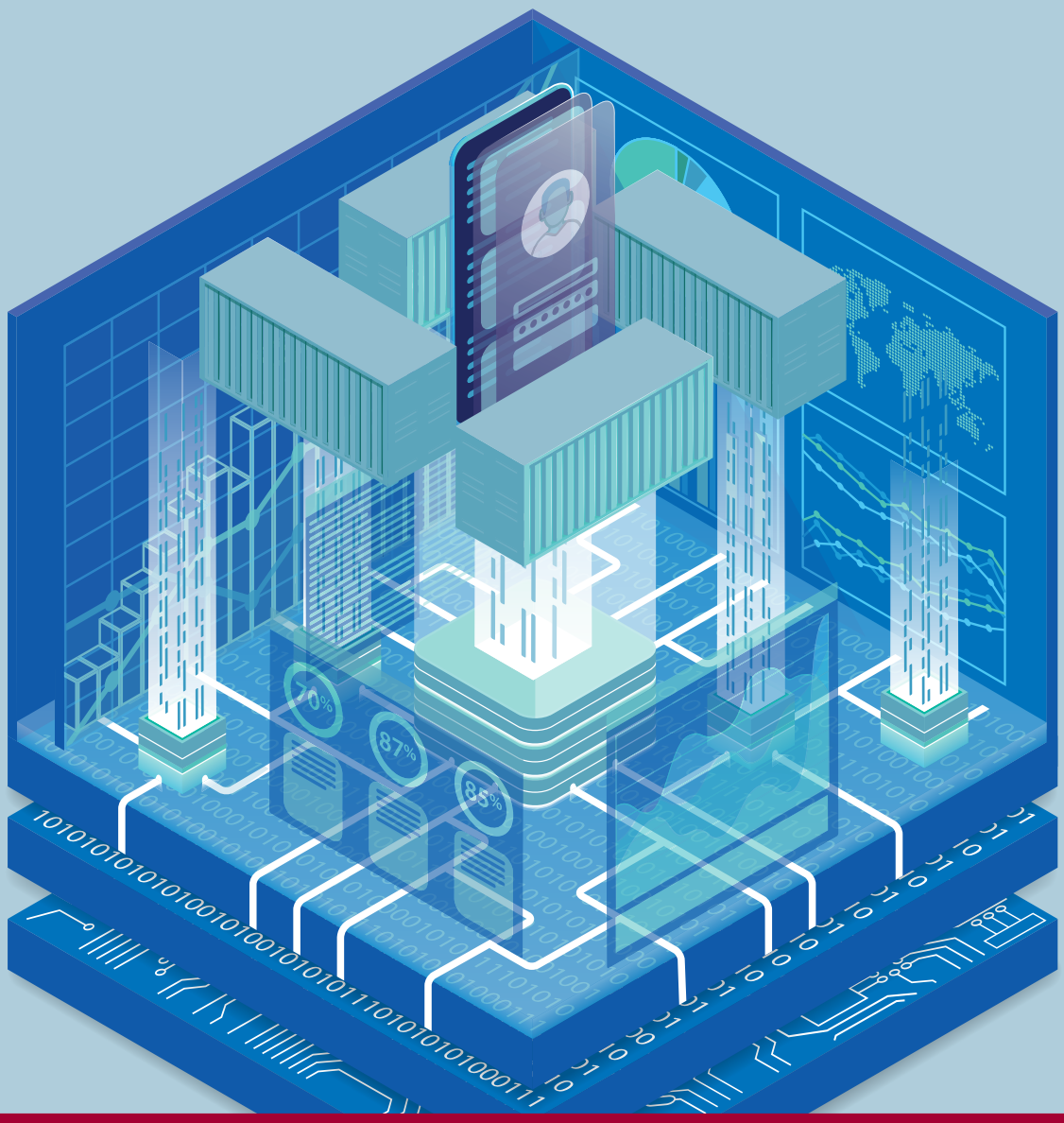

CONTAINER UND SICHERHEIT

| von DR. ROGER FISCHLIN

Sie nutzen eine bekannte Suchmaschine? Oder sehen Filme bei einem populären Video-Streaming-Dienst? Dann haben Sie, vielleicht ohne es zu wissen, bereits Container-Technologie genutzt. Andere Unternehmen, auch die IT der öffentlichen Hand, setzen inzwischen ebenfalls vermehrt auf containerbasierte IT-Lösungen. Doch mit der Container-Technologie gehen neue Bedrohungen für die Informationssicherheit einher.

NEUE TECHNOLOGIE, BEKANNTE HERAUSFORDERUNGEN

In den Augen vieler zunächst eine einfache Variante der Rechnervirtualisierung hat die Container-Technologie die Architektur von IT-Fachverfahren und deren Betrieb heute grundlegend verändert: Anwendungen sind kein Monolith mehr, sondern das Zusammenspiel von vielen Mikroservices. Anwendungen werden nicht mehr installiert, sondern Container-Images zentral bereitgestellt. Anwendungen werden nicht mehr gestartet, sondern Container orchestriert. Dies schafft auch eine Voraussetzung für die agile Entwicklung und „Continuous Delivery", um schneller auf Anwenderwünsche reagieren zu können.

Doch mit der Container-Technologie gehen neue Bedrohungen für die Informationssicherheit einher. In diesem Beitrag skizzieren wir diese Technologie, beleuchten die Risiken und erläutern, wie man ihnen entgegenwirkt. Die Grundsätze der Informationssicherheit haben in der Container-Welt Bestand. Es sind die bekannten Herausforderungen für Vertraulichkeit, Integrität und Verfügbarkeit. Doch die Maßnahmen muss man auf die neuen Herausforderungen ausrichten.

DER NEUE ANSATZ: MIKROSERVICES, CONTAINER UND ORCHESTRIERUNG

Das Design von IT-Fachverfahren folgt seit über einem Jahrzehnt der Drei-Schichten-Architektur (Three Tier Architecture):

- Webserver als Frontend beziehungsweise User Interface (Präsentationsschicht)
- Applikationsserver mit der Logik (Mittelschicht)
- Relationale Datenbank als Backend (Persistenzschicht)

Es ist ein lineares Modell: Anwender kommunizieren mit dem Webserver, dieser mit dem Applikationsserver und dieser seinerseits mit der Datenbank. Der Webserver bereitet die Informationen für den Nutzer auf, verarbeitet werden die Daten ausschließlich auf dem Applikationsserver. Für die dauerhafte Datenhaltung (Persistenz) nutzt dieser eine Datenbank.

Der Three-Tier-Ansatz mit einem komplexen Applikationsserver ist ungeeignet für die agile Entwicklung mit ihren fortlaufenden kleinen Anpassungen. Vielmehr ist es wegen der Abhängigkeiten eher erforderlich, Änderungen in größere Releases zu bündeln. Für ein zeitnahes „Continuous Delivery" (CD) entkoppelt man Verfahren heute in Teildienste, sogenannte Mikroservices, mit definierten Schnittstellen. Eine Änderung betrifft meist nur die Implementierung eines Mikroserviceses, nicht das Zusammenspiel mit den anderen Diensten. Die Kapselung mit definierten Schnittstellen erleichtert darüber hinaus, vorhandene Microservices in anderen Anwendungen einzusetzen oder von Dritten angebotene Komponenten zu verwenden.

Das typische Mikroservice-Design besteht aus einem zentralen API-Gateway, auch Edge-Service genannt. Über ihn kommuniziert der Nutzer beziehungsweise der Webserver (allgemeiner das User-Interface) mit den internen Mikroservices. Die zentrale Schnittstelle erlaubt, Authentifizierung usw. für alle Anfragen an Mikroservices einheitlich zu regeln. Ein bekannter Vertreter ist „Zuul" des Streaminganbieters Netflix[1]. Der Streaming-Dienst nutzt das API-Gateway, das in Release 2 Open-Source ist, für seine Angebote. Cineasten erkennen im Namen die Anspielung auf den Film Ghostbusters.

Es liegt nahe, Mikroservice mit Container gleichzusetzen. Doch ein Mikroservice ist vielmehr ein Dienst, der aus mehreren Containern bestehen kann, etwa einem Applikationsserver und einer Datenbank. Das Ziel von Mikroservices sind per se nicht möglichst kleine Dienste, sondern in sich geschlossene Systeme. Wolff charakterisiert dies in seinem Buch „Microservices – Ein Überblick"[2] wie folgt:

- Ein Team sollte den Mikroservice entwickeln und pflegen können.
- Die Funktionalität des Mikroservices sollte für das Team überschaubar sein.
- Ein Mikroservice sollte innerhalb der Anwendung ersetzbar sein.

Schlüssel des Designs mikroservicebasierter Anwendungen ist es, Komponenten ersetzen zu können.. Je kleiner ein Mikroservice ist, desto leichter ist er auszutauschen. Allerdings gibt es praktische Hindernisse:

- Programmlogik kann nur bedingt über Mikroservices hinweg verschoben werden.
- Fachliche Konsistenz der verarbeiteten Informationen lässt sich leichter innerhalb eines Mikroservices sicherstellen.
- Jeder Mikroservice benötigt als unabhängige Komponente eine eigene Umgebung.

Neu ist die Entkopplung von Diensten indes nicht. Sie war unter dem Schlagwort „serviceorientierte Architektur" (SOA) bereits um die Jahrtausendwende populär. Die Schnittstellen waren jedoch komplex: Die Services sollten nicht direkt, sondern organisationsübergreifend über einen Enterprise-Service-Bus kommunizieren.

Mit der Container-Technologie ist jetzt der nächste Schritt hin zur Kapselung einer Anwendung auf Basis von Mikroservices möglich.

CONTAINER STATT EINZELNER RECHNER

Ein Container enthält ein Programm mitsamt allen Abhängigkeiten, wie Bibliotheken, Hilfsprogrammen und statischen Daten. Man spricht von Anwendungsvirtualisierung, denn ein Container ist kein vollständiger Rechner mit eigenem Betriebssystem (Kernel). Weil Container in sich abgeschlossen sind, lassen sich Altverfahren (Legacy-Systeme), die bestimmte Versionsstände von Programmen erfordern, so auf modernen Rechnern betreiben.[3]

Die Hauptanwendung von Containern ist indes die Gliederung von Anwendungen in einzelnen Services. Ein (Anwendungs-)Container ist dabei „ein Konstrukt zum Paketieren und Ausführen einer Anwendung oder ihrer Komponenten, die auf einem gemeinsam genutzten Betriebssystem ausgeführt werden".[4]

Bei der Rechnervirtualisierung wird die gesamte Hardware des Rechners auf dem Gastsystem (Wirt) nachgebildet. Dabei erfordert jede virtuelle Maschine ein separates Betriebssystem. Bei Containern separiert man die Ressourcen zwischen Containern dagegen auf Betriebssystemebene des Wirts. Jeder Container hat einen eigenen Blick auf Ressourcen wie Prozesse, Netzwerk, Nutzer und Dateisystem. Container benötigen so im Vergleich zu virtuellen Rechnern deutlich weniger Ressourcen, man kann sie signifikant schneller starten und stoppen. Erkauft wird dieser Vorteil durch einen Rückschritt bei der Virtualisierung und mehr gemeinsamen Ressourcen. Auf virtuelle Rechner wird dennoch nicht verzichtet, oft laufen die Container auf virtuellen Maschinen.

Populär wurde die Container-Technologie mit dem Produkt der Firma „Docker" vor rund fünf Jahren. „Docker" ist zum Gattungsbegriff geworden. Es gibt zwar mittlerweile auch andere Container-Lösungen, sie spielen aber bisher noch nur eine untergeordnete Rolle.

Eine Container-Engine steuert die Container auf einem Rechner. Ein Container wird über ein Image, eine Datei mit einem Speicherabbild, initiiert. Die Engine speichert Änderungen des Containers an dessen Filesystem in einem Overlay-Dateisystem (vergleichbar mit Deltainformationen nach Snapshots). Beim Neustart des Containers sind diese Informationen gelöscht, er startet wieder mit dem Image. Veränderungen oder gar böswillige Manipulationen am Container „überleben" einen Neustart nicht – im Gegensatz zu einer virtuellen Maschine mit ihrem eigenen Dateisystem. Für dauerhaft zu speichernde Informationen (Persistenz) verwenden Container externe Speich-

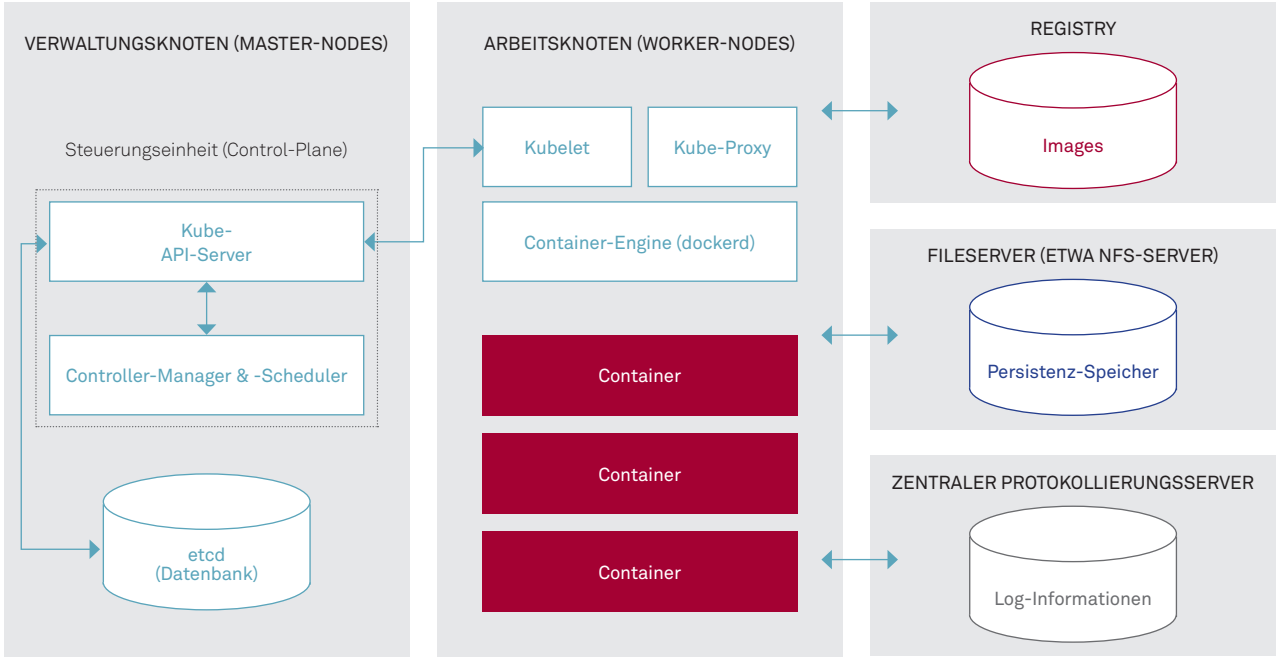

Abbildung 1: Aufbau Kubernetes-Cluster

erorte (Persistent-Volumes, PV) wie NFS-Shares oder nutzen externe Datenbanken. Logeinträge leitet ein Dienst an einen zentralen Protokollierungsserver weiter.

Die Images werden bei einer Docker-Installation in einer webbasierten Registry vorgehalten, um bei Bedarf neue Container zu initiieren. Images sind portabel, von anderen erstellte Images sind im Grundsatz auf vergleichbaren Betriebssystemen funktionsfähig. Eine große Auswahl von Images zu freier Software gibt es im Internet, beispielsweise im „Docker Hub". Allerdings ist die Qualität der Images nicht immer überzeugend, und es besteht die Gefahr verborgener Programme (Schadsoftware, Kryptominer usw.), weshalb Docker in seiner Bibliothek „Docker Store" nur qualitätsgesicherte Images anbietet.

Wer Container auf einem einzelnen Rechner laufen lässt, nutzt im Wesentlichen nur die Trennung der Anwendungen und das einfache Update mittels aktualisierter Images. Das Potenzial der Technologie für ausfallsichere und skalierbare Verfahren zeigt sich erst, wenn man für die Container einen Pool von Servern nutzt. Einen solchen nennt man Cluster, dessen Mitglieder Knoten (Nodes). Die Steuerung (Orchestrierung) der Container ist händisch nicht mehr zu bewerkstelligen. Vielmehr übernimmt ein Scheduler die Verteilung der Container auf den Knoten. Er ist Teil eines Orchestrators, der autonom den Betrieb der Container auf dem Cluster steuert.

Der heute zweifellos populärste Orchestrator, Kubernetes, stammt von Google. Ursprünglich unter dem Namen Borg vor 15 Jahren für den reibungslosen Betrieb ihrer Suchmaschine entwickelt ist Kubernetes heute das Tool für die Container-Orchestrierung, die auf Docker oder anderen Container-Lösungen aufbaut. Für tiefergehende Informationen wird auf das Kubernetes-Einführungstutorial verwiesen.[5]

SICHERHEIT

Übersicht

Die Container-Technologie bietet, wie bereits erwähnt, zahlreiche Vorteile, birgt gleichwohl auch Risiken für die Informationssicherheit. Die Container-Virtualisierung bedeutet letztlich einen Schritt zurück von der Rechnervirtualisierung. Die wesentlichen Ereignisse, die man bei Anwendung der Container-Technologie verhindern beziehungsweise deren Auswirkungen man einschränken muss, sind:

- **Trojaner im Image:** Es wird unbemerkt ein bösartiges (malicious) Image für einen Container genutzt, der so unerwünschte Aktivitäten entfaltet.
- **Schwachstelle im Image:** Es wird unbemerkt ein Image mit veralteter Software für einen Container genutzt, dessen Schwachstelle einem Angreifer erlaubt, den Dienst beziehungsweise den Container anzugreifen.
- **Container-Breakout:** Ein im Container laufendes Programm bricht aus seiner Umgebung aus und greift – unter Umgehung der Schutzmechanismen – andere Container, deren Daten (auf Persistent-Volumes oder in Datenbanken), den Knoten oder gar das gesamte Cluster an.
- **Orchestrator-Angriff:** Ein Unberechtigter manipuliert den Orchestrator oder Teile davon und greift so die Anwendungen des Clusters an.

Diese Szenarien sind durchaus real. Sicherheitsexperten entdeckten 2018 mehrere Images auf „Docker Hub", die einen Cryptominer enthielten und millionenfach heruntergeladen wurden. Ein Jahr später wurde eine Schwachstelle in einer von Docker genutzten Komponente bekannt, die es ermöglichte, Root-Zugriff auf den Knoten aus einem mit privilegierten Rechten laufenden Container in Verbindung zu erlangen.[6] In der Fachliteratur werden als Sicherheitsstandards für Container und Orchestrierung üblicherweise zwei Quellen genannt:

- NIST Standard SP 800-190 „Application Container Security Guide"[7]
- CIS-Prüflisten für Docker und Kubernetes[8]

Die Dokumente sind kostenlos im Internet erhältlich, die Listen des Centers for Internet Security (CIS) jedoch erst nach Registrierung. NIST beschreibt Risiken für die Komponenten einer Container-Plattform und gibt abstrakt und produktunabhängig Maßnahmenempfehlungen. CIS hat für Docker und Kubernetes detaillierte Hinweise für sichere Einstellungen zusammengestellt, ergänzt um die Prüfbefehle. Das BSI arbeitet aktuell an einem IT-Grundschutz-Baustein, dessen zweiter Community Draft im Mai die Kommentierungsphase abgeschlossen hat.

Generell gilt: Mit Containern ändert sich die Art und Weise grundlegend, wie Anwendungen betrieben werden. NIST betont daher, Betriebskultur und Abläufe auf den sicheren Betrieb von containerisierten Anwendungen anzupassen. Dies gilt insbesondere für den Deployment-Prozess, denn technisch ist ein falsches oder gar manipuliertes Image schnell in die Produktion ausgerollt, wie die beiden Beispiele oben zeigen.

Containerisierte Anwendung

Voraussetzung für den sicheren Betrieb einer containerisierten Anwendung ist, dass diese die Orchestrierung unterstützt, so-

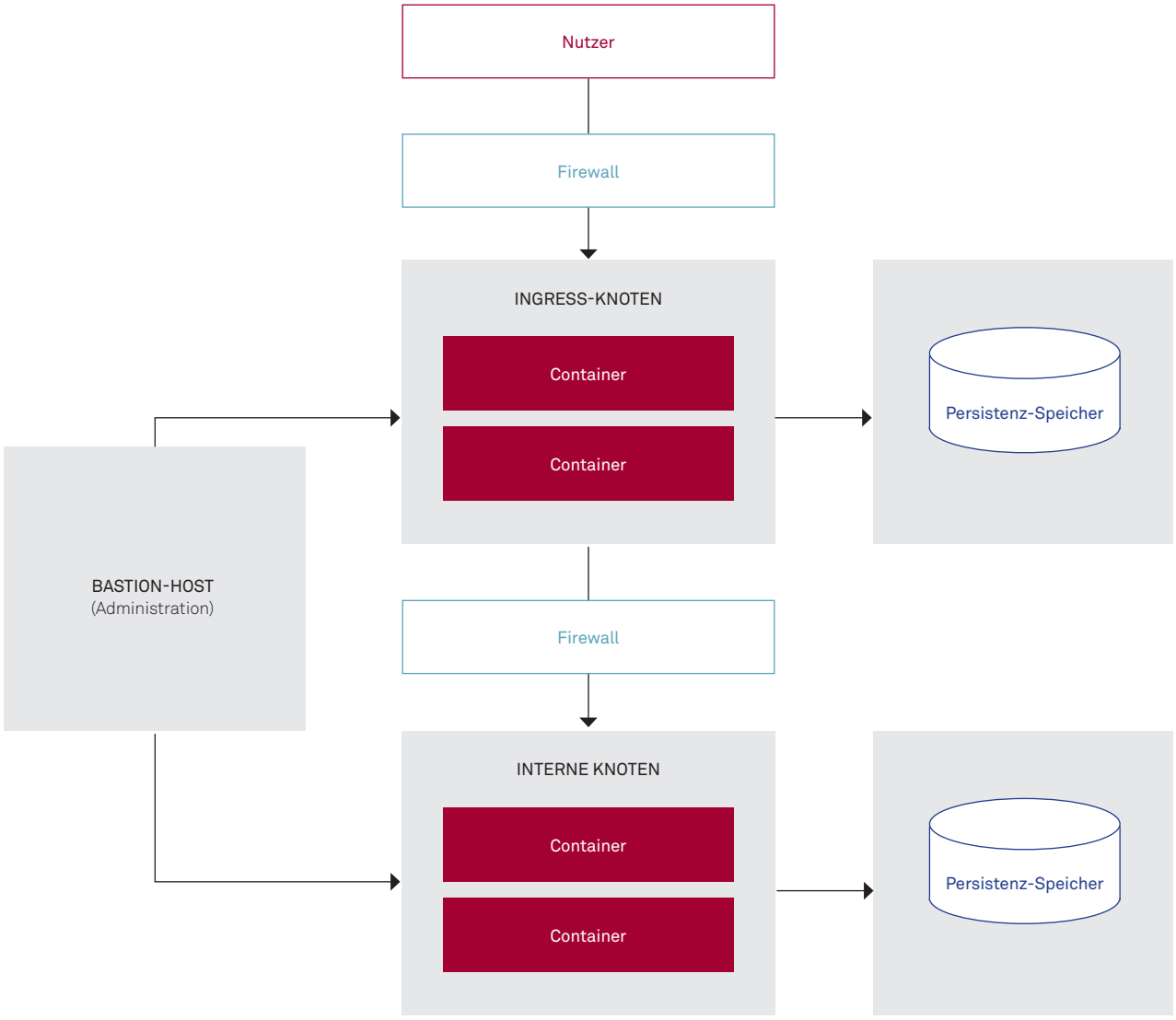

Abbildung 2: Unterteilung der Knoten

wohl vom Design als auch von der Dokumentation (Konfiguration) für den Betrieb.

Die Anwendung muss so gestaltet sein, dass der Orchestrator jeden Container beliebig neu starten kann. Unter Fachleuten ist dieser Punkt für Datenbanken umstritten: Einige argumentieren, heutige Datenbanksysteme können mit Neustarts ohne Weiteres umgehen, andere verweisen auf schlechte Erfahrungen in bestimmten Szenarien. Die Wahrheit liegt wohl dazwischen und hängt von dem Nutzungsprofil ab. Wer auf Nummer sicher gehen will, greift auf klassische Datenbankserver außerhalb des Clusters zurück.

Images

Images sind die neuen Softwarepakete, und so gelten für sie die bekannten Sicherheitsanforderungen. Man darf folglich Images nur aus vertrauenswürdigen Quellen und erst nach Integritätstest nutzen, egal ob man diese direkt verwendet oder als Basis für eigene Images nimmt. Es sollte erwogen werden, Images komplett selbst zu bauen, um unerwünschte Zusatzprogramme, Hintertüren oder Schadcodes von vornherein auszuschließen. Außerdem sollten Images versioniert und entsprechend gekennzeichnet (getaggt) sein, etwa Major Release für Funktionsänderungen und Minor Release für Bugfixes, da-

mit beim Update im laufenden Betrieb keine inkompatiblen Releases von Images zum Einsatz kommen. Die Software in Images muss regelmäßig auf bekannte Schwachstellen geprüft werden. Gleiches gilt für Schadprogramme in Images. Gerade im Kontext „Continuous Delivery" sollten diese Tests als Quality-Gates automatisiert werden. Durchgefallene Images dürfen nicht in die nächste Stufe der Pipeline gelangen. Ein populäres Programm ist „SonarQube"[9], das Komponenten in Images anhand der CVE-Meldungen auf bekannte Schwachstellen abgleicht und die Ergebnisse (Findings) gewichtet.

Ein Image darf nur die erforderlichen Pakete enthalten und keinesfalls Entwickler- oder Debugging-Werkzeuge. Remote-Zugänge (wie SSH) gehören nicht in Images. Der Zugriff in einen Container erfolgt stets über den Gastrechner. Die Software ist nach Best Practices zu konfigurieren. Images dürfen im Grundsatz keine privilegierten Rechte auf dem Gastsystem zur Ausführung benötigen. Genauso wenig wie man geheime Zugangsdaten in Quellcodes schreibt, gehören vertrauliche Informationen im Klartext in Images.

Plattform-Verantwortliche sollten die Anforderungen an Images – vor allem bei Fremdnutzung oder Mehrmandantenplattformen – dokumentieren und deren Einhaltung vor dem Deployment und danach regelmäßig prüfen. Moderne Sicherheitswerkzeuge für Container bieten umfassende automatisierte Tests für Images und ihren Inhalt (Compliance).

Registry

Registrys sind vergleichbar zu Software-Depots, daher gelten für sie die bekannten Regeln. Der Zugriff muss beispielsweise authentifiziert und autorisiert werden. Nur Berechtigte dürfen Images für die Produktion freigeben. Es gilt, den lesenden Zugriff auf relevante Personen und Systeme zu begrenzen – auch, um nicht versehentlich urheberrechtlich geschützte Software zu verbreiten. Der Zugriff auf die Registry sollte gesichert erfolgen, etwa per HTTPS.

Einen gravierenden Unterschied gibt es freilich: Bei Software-Depots stößt der IT-Betrieb die Installation an, bei Registrys zieht sich Kubernetes automatisch bei jedem Start eines Containers das passende Image. So können sich unerwünschte Images schnell verbreiten, seien es bewusst manipulierte (malicious) oder veraltete Images (Stale Images). Daher müssen Prozesse für das Deployment, Freigabe und Kontrolle der Images in der Registry etabliert werden. Man sollte prüfen, für Produktion und Test getrennte Registrys zu nutzen. Images sollten stets aus der Registry anhand eines bestimmten Funktionsumfangs (beispielsweise Major Release) in Verbindung mit aktuellen Bugfixes (Minor-Release) gezogen werden. Auf keinen Fall darf einfach die letzte verfügbare Version (latest) angefordert werden. Images dürfen nur von einer autorisierten Registry geladen werden, auch nicht aus dem Zwischenspeicher von Kubernetes. Der Zugriff auf andere Registrys sollte etwa durch Firewall-Regeln blockiert werden.

Orchestrator (Kubernetes)

Der Orchestrator muss eine sichere Umgebung für die containerisierten Anwendungen bereitstellen. Dazu gehört die Absicherung der Steuerung, ein Vertrauensverhältnis zwischen den Knoten des Clusters sowie eine Kontrolle der Kommunikation zwischen Containern.

Der Orchestrator steuert das Cluster – wer also den Orchestrator übernimmt, hat die Kontrolle über sämtliche Anwendungen auf der Container-Plattform. Wie üblich muss man daher die administrativen Zugriffe absichern und einschränken. Die von Kubernetes' standardmäßig angebotenen anonymen und ungesicherten Zugänge auf den API-Server sind zu deaktivieren. Kubernetes' einfaches Identitäts- und Berechtigungsmanagement sollte mittels Erweiterungen durch eine rollenbasierte Zugriffskontrolle (RBAC) ersetzt werden. Die Zugriffe der Agenten „Kublet" auf den Arbeitsknoten müssen gleicherweise gesichert werden. Umgekehrt muss man die Möglichkeiten von Kublet, über die zentrale API-Schnittstelle den gesamten Cluster zu steuern, einschränken.

Besonders beim Mehrmandantenbetrieb müssen Mindeststandards für den sicheren Betrieb definiert werden. Kubernetes bietet mit Security Policys die Möglichkeit, Sicherheitsvorgaben der Container-Konfiguration festzulegen, ohne deren Einhaltung der Orchestrator den Container nicht startet.

Das Overlay-Netz muss abgesichert werden. Die Kommunikation zwischen Containern sollte verschlüsselt werden, sei es auf Anwendungs-, Service-Mesh-[10] oder Server-Ebene. Ein weiterer Aspekt ist die Reglementierung des Netzverkehrs. Bei der klassischen Architektur übernehmen Firewalls diese Aufgabe. In Kubernetes' Overlay-Netz gibt es dafür Regeln in Network Policys, vorausgesetzt, das eingesetzte Netzwerk-Plug-in unterstützt diese. Da mit detaillierten Network Policys die Komplexität ansteigt, bieten bessere Sicherheitswerkzeuge an, in einer Lernphase die zulässigen Kommunikationsbeziehungen zu bestimmen und Regelsätze zu generieren. Solche Tools ermöglichen, analog zu Network-Intrusion-Detection-Systemen (NIDS) für klassische Netze, auffälligen Traffic in Overlay-Netzen zu erkennen und automatisiert verdächtige Container zu stoppen.

Die Konfiguration des Orchestrators sollte, etwa mit der CIS-Liste, regelmäßig auf Schwachstellen geprüft werden. Moderne Sicherheitslösungen für Container bieten teilweise an, die Anforderungen automatisiert abzugleichen (Audit).

Container-Laufzeitumgebung (Docker)

Ein Angreifer, der einen Container infiltriert hat, sollte zumindest nicht wieder aus diesem ausbrechen können. Dafür müssen zunächst Schwachstellen in der Container-Laufzeitumgebung vermieden werden. Der Container darf grundsätzlich nicht unter Root-Privilegien laufen oder diese erreichen können. Seine Rechte (zum Beispiel in den Linux Kernel Capabilities) müssen begrenzt sein. Eine Übersicht über die erforderlichen Capabilities findet sich im Whitepaper der „NCC Group“[11]. Der Container darf nicht den globalen Namensraum des Wirts sehen. Fachleute raten, auf dem Betriebssystem ein Security-Modul wie SELinux bei Red-Hat-Linux einzusetzen. Den von Containern ausgehenden Netzwerkverkehr gilt es zu begrenzen, damit ein Hacker abgegriffene Daten nicht nach außen senden kann.

Die Konfiguration der Container-Laufzeitumgebung sollte etwa mit der CIS-Liste regelmäßig auf Schwachstellen hin geprüft werden. Moderne Sicherheitslösungen für Container bieten teilweise an, die Anforderungen automatisiert abzugleichen (Audit).

Knoten und Betriebssysteme

Die Betriebssysteme für die Knoten müssen gehärtet sein: Plattformfremde Anwendungen oder Endbenutzer-Accounts gehören nicht auf die Server. Das Betriebssystem und dessen Administration müssen auf den Container-Betrieb ausgerichtet sein, zumal Kubernetes tief in die Interna (unter anderem in das Netzwerk) eingreift. Selbstverständlich muss das Betriebssystem regelmäßig auf Schwachstellen geprüft werden, und Patches zeitnah eingespielt werden.

Die Container zu von außen erreichbaren Services sollten in Kubernetes dedizierten Rechnern (Ingress-Knoten) zugewiesen werden. Dieses Pinning erlaubt eine bessere Trennung der Container, auch kann man anhand der IP-Adressen beispielsweise den Zugriff auf Persistent-Volumes knotenfremder Container oder auf die externen Datenbanken unterbinden. Ein Angreifer, der einen von außen erreichbaren Container oder einen Container eines anderen Verfahrens und anschließend den Knoten übernommen hat, kann so nicht auf die Datenablage der internen Container zugreifen. So lässt sich auch eine Drei-Schichten-Architektur realisieren. Alternativen sind, für beide Bereiche verschiedene Cluster mit getrennten Orchestratoren aufzubauen oder die erste Schicht auf separaten Rechnern ohne Orchestrator aufzubauen. Beim Mehrmandantenbetrieb auf dem Cluster kann man mit Pinning für eine bessere Trennung etwa die Knoten den Containern der Anwendungen dediziert zuordnen.

Zwischen den Knoten sollten keine Zugriffe (vor allem nicht über SSH) möglich sein, sondern nur über einen abgesetzten Admin-Rechner als Bastion-Host. So kann ein Angreifer, der einen Knoten infiltriert hat, diesen nicht als Sprungbrett auf andere Knoten des Clusters nutzen (Server Hopping).

AUSBLICK

Die Container-Technologie verändert die Entwicklung und den Betrieb von Anwendungen nachhaltig. Manche der bekannten Sicherheitsmaßnahmen gelten unverändert, andere müssen an die neuen Gegebenheiten angepasst werden. Dies trifft gleichermaßen auf die Sicherheitswerkzeuge zu, die oft noch nicht für die Container-Technologie ausgelegt sind. Die leistungsstarken Tools von Start-ups wie „AquaSec“[12], „Neuvector“[13] oder „Twistlock“[14] zeigen den Weg: Sie setzen stärker auf Verhaltensanalysen und Automatisierung. Die Werkzeuge frieren beispielsweise Container mit verdächtigtem Verhalten ein und starten automatisch einen neuen Container mittels des ursprünglichen Images. ●

1 https://jaxenter.de/zuul-2-gateway-plattform-open-source-71405 (abgerufen am 06.07.2020).
2 Wolff, Eberhard: „Microservices – Ein Überblick“, innoQ Deutschland GmbH, 2018.
3 vgl. .public Ausgabe 01-2020, Tim Pommerening: „Betreutes Wohnen für Altanwendungen“.
4 NIST-Standard SP 800-180.
5 https://kubernetes.io/de/docs/tutorials/kubernetes-basics/ (abgerufen am 06.07.2020).
6 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5736 (abgerufen am 06.07.2020).
7 National Institute of Standards and Technology (NIST): „Application Container Security Guide“, Special Publication 800-190, 2017.
8 Center for Internet Security: „CIS Docker 1.13.0 Benchmark“, v1.0, 2017; Center for Internet Security: „CIS Kubernetes Benchmark“, v1.5.1, 2020.
9 https://www.sonarqube.org/ (abgerufen am 06.07.2020).
10 Ein Service-Mesh ist eine Infrastrukturebene für die sichere und zuverlässige Kommunikation zwischen den Diensten einer Anwendung, die auf dem Netzwerk aufsetzt.
11 NCC Group: „Understanding and Hardening Linux Containers“, Whitepaper, 2019.
12 https://www.aquasec.com/solutions/kubernetes-container-security/ (abgerufen am 06.07.2020).
13 https://neuvector.com/kubernetes-security-solutions/ (abgerufen am 06.07.2020).
14 https://www.twistlock.com/dockerfederalsummit/22/ (abgerufen am 06.07.2020).