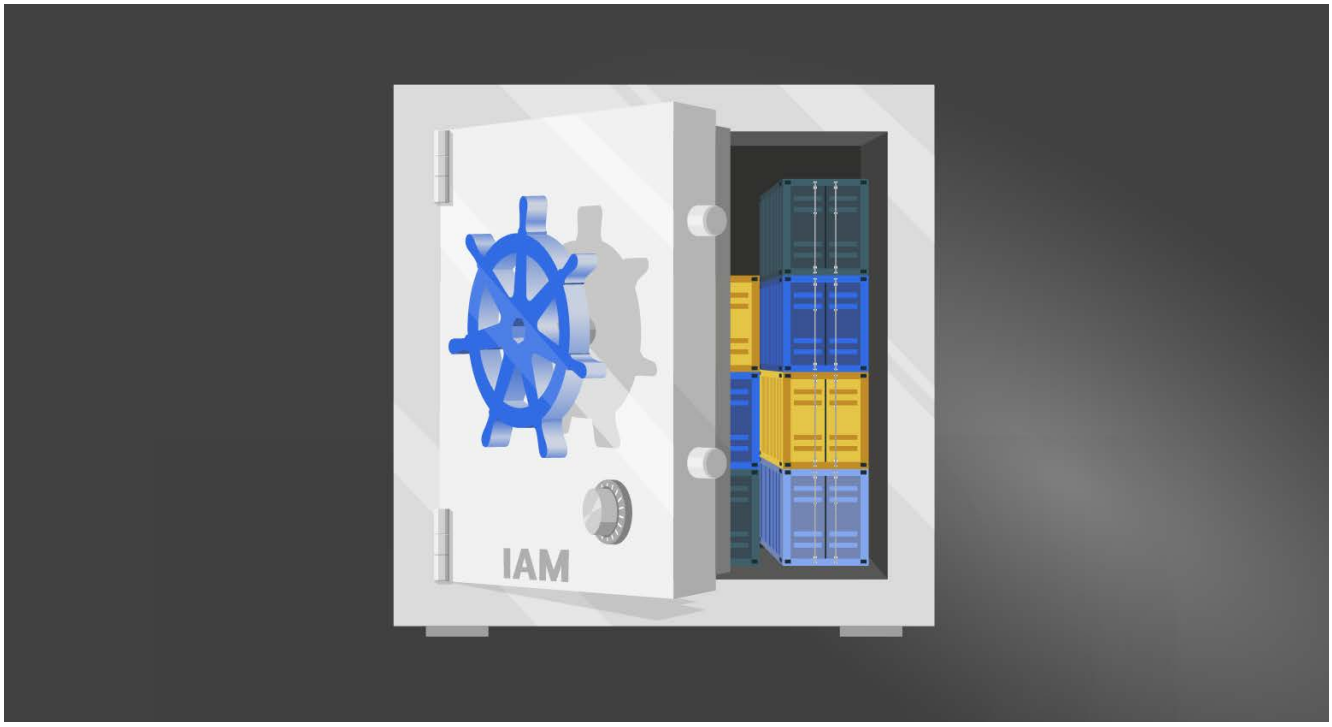


Kubernetes-Tutorial, Teil 2: IAM erklärt, AKS und GKE härten

Bei den großen Anbietern ist Kubernetes stärker in die Managed-Angebote integriert und somit variiert das Vorgehen zur Absicherung auch stärker. Worauf muss man dabei in Azure Kubernetes Service (AKS) und Google Kubernetes Engine (GKE) achten?

Von Christoph Puppe

Der zweite Teil der Tutorialserie betrachtet zunächst die grundlegenden Aspekte Namespaces, Benutzer- und Objektverwaltung in Kubernetes und nimmt sich später die Managed-Kubernetes-Angebote von Azure und Google vor. Auch wenn die im ersten Teil aufgeführten allgemeinen Schritte weiterhin gleich sind, unterscheidet sich die Umsetzung bei den Plattformen doch erheblich.

Namespaces, Benutzer- und Objektverwaltung

Kubernetes verwaltet die Nutzer nicht selbst, nur die Zuordnung von Accounts zu Berechtigungen steht im eigenen Konfig-Store etcd. Dass es nicht nur eine Account-ID gibt, sondern zwei, die auch noch von unterschiedlichen Stellen bereitgestellt und gespeichert werden, trägt nicht gerade zur Übersichtlichkeit bei. Aber Container sind ja nur eine Art, die Software zu paketieren und Prozesse zu starten. Also ist jeder laufende Container auch ein vom Kernel des Node verwalteter normaler Userspace-Prozess.

Der Linux-Kernel verwendet zur Adressierung der Prozesse eine numerische PID (Process ID), die von 1 – dem Init-Prozess – hochzählt. Damit der Kernel weiß, was dieser Prozess darf, hat er zwei weitere numerische IDs, die UID (User ID) und die GID (Group ID), bei der 0 root vorbehalten ist und alle anderen in /etc/passwd respektive /etc/groups stehen. Diese IDs erbt der Prozess vom aufrufenden Prozess und kann dann zu einer

iX-TRACT

- Auch wenn Kubernetes die Benutzerverwaltung nicht selbst macht, gibt es bei der sicheren Konfiguration eines Clusters doch einige Identitäten zu berücksichtigen.
- Bei den großen Hyperscalern gibt es erhebliche Unterschiede beim Absichern der eigenen Kubernetes-Services.
- Dienste für Managed Kubernetes sind bei allen Hyperscalern stark in die Verwaltung integriert und zentral steuerbar.
- Die zum Absichern benötigten Werkzeuge sind entweder schon im Standardumfang enthalten oder lassen sich leicht nachrüsten.

anderen – weniger berechtigten – UID und GID wechseln. So beginnen die von systemd gestarteten Systemdienste zunächst alle mit Root-Rechten und springen dann in weniger berechtigte Rollen wie www-run. Mit der Einführung der User Namespaces hat sich die Zahl der IDs verdoppelt. Nun kann ein Prozess in einem Namespace eine andere UID und GID besitzen als außerhalb. Für die anderen Prozesse auf dem Node hat etwa der Prozess nginx die UID 42, aber innerhalb seines Namenspace – und nur das ist für den Prozess selbst sichtbar – hat er die UID 0. Gleiches gilt auch für das Root-Verzeichnis und die Capabilities, die innerhalb des Namespace anders sein können als außerhalb.

Leider ist die Nutzung des User Namespace nicht Standard und wird zwar von CRI-O, aber nicht von containerd unterstützt. Wer diesen zusätzlichen Schutz nutzen möchte, muss im Manifest des Pods den Eintrag `pod.spec.hostUsers` auf `false` setzen. Nebeneffekt ist dann, dass der Pod weder das Netzwerk des Hosts noch sein IPC und die PID erreichen kann. Sehr nachteilig ist auch, dass der Pod dann keine Volumes mehr nutzen kann.

Es bleibt die Regel, dass die UID und GID des Containers keine Berechtigungen auf dem Node haben dürfen, damit der Container keine Dateien außerhalb seines eigenen Dateisystems lesen oder ändern kann. Ausnahme sind Container, die etwa Pfade auf dem Node nutzen. Für die gilt, dass ihre UID für den Pfad auf dem Node berechtigt sein muss. Beispiele sind alle Nodes, die lokale Pfade mounten, um dort flüchtige Daten (Scratch Space) oder eine von mehreren Datenkopien bei verteilten Dateisystemen zu speichern. Auch die Sicherheitswerkzeuge benötigen Zugang zu lokalen Pfaden.

Identitäten der Pods

Damit Kubernetes alle Objekte im Cluster verwalten kann, benötigen diese einen eindeutigen Namen und einen Unique Identifier (UID), eine zufällige, innerhalb des Clusters einmalige Zeichenfolge, vergleichbar mit der PID. Auch Kubernetes hat Namespaces, die im Gegensatz zu den Linux-cgroup-Namespaces keine Sicherheitsbarriere darstellen, sondern nur Objekte in einer Hierarchie benennen. So heißt ein Pod Ernie im Namespace A ernie.a.svc.cluster.local und ein Pod Ernie im Namespace B ernie.b.svc.cluster.local. Alle Pods können sich gegenseitig im DNS sehen und auch im Netzwerk erreichen, sofern nicht Network Polices, also Firewallregeln im CNI (Container Network Interface), dies verhindern.

Die Pods haben neben dem Namen dann einen Service-Account, was in etwa der UID eines Prozesses im Linux-Kernel entspricht. Ohne explizite Angabe eines Service-Accounts bekommen alle Pods den Default-Account. Hat dieser Account Rechte auf dem K8s-API-Server, kann der Pod damit die aktuelle Konfiguration des Clusters auslesen oder vielleicht sogar ändern. Dies kann der Admin unterbinden, indem entweder das Secret-Token des Accounts nicht im Pod sichtbar ist (siehe Listing 1) oder eine Network Policy die Verbindung vom Pod zum K8S-API-Server verhindert. Am besten nutzt man beides.

Es empfiehlt sich, mindestens für die Pods einer Anwendung einen eigenen Service-Account einzurichten und nur den Pods den Zugriff auf die Token zu erlauben. Noch besser ist es, wenn jedes Deployment einen eigenen Service-Account hat. Dann können manche CNI-Provider wie Calico diese Identitäten für Network Policies nutzen. Dies kann gegen den Namen des Service-Accounts, aber auch gegen Labels auf den Pod oder den Service-Account erfolgen. Hier ist zu beachten, dass dann eingeschränkt sein muss, wer die Labels setzen darf, da

Listing 1: Keine AccountToken in Pods

```
apiVersion: v1
kind: ServiceAccount
metadata:
 name: build-robot
automountServiceAccountToken: false
...
```

Listing 2: Calico-Policy-Beispiel

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: demo-calico
  namespace: prod-engineering
spec:
  ingress:
    - action: Allow
      source:
        serviceAccounts:
          names:
            - api-service
            - user-auth-service
  selector: 'app == "db"'
```

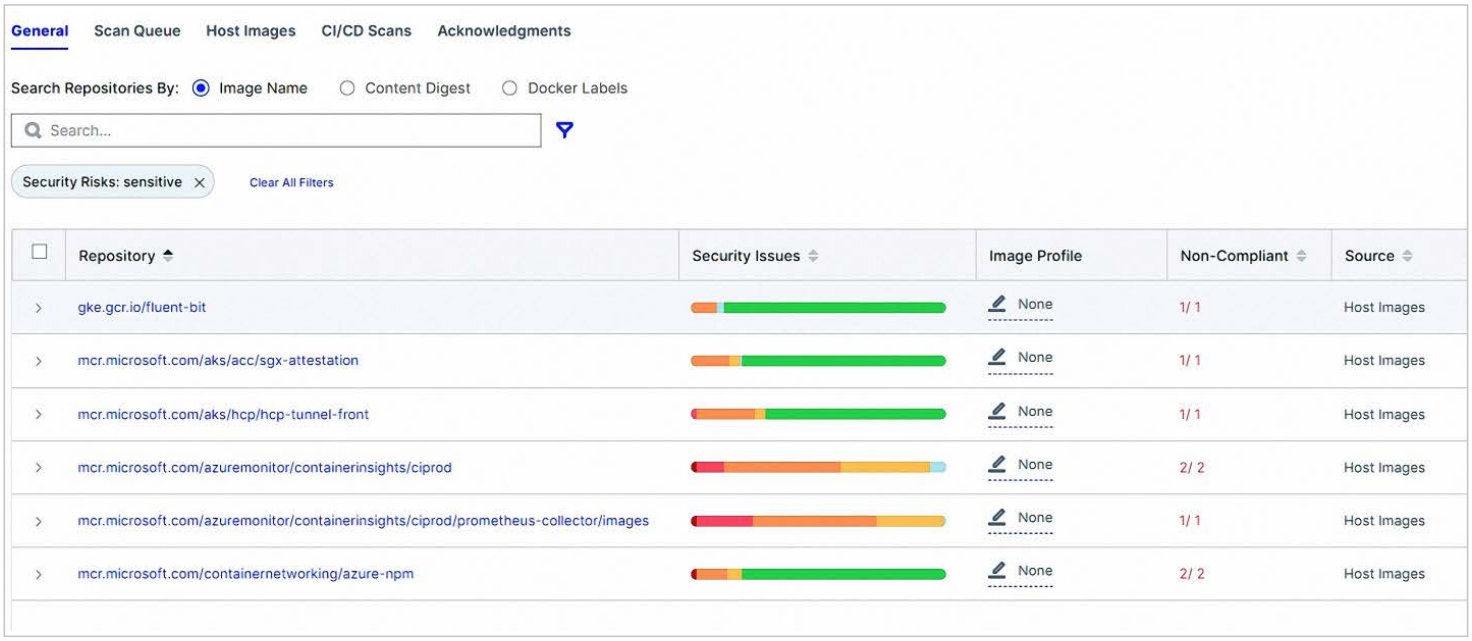
Listing 3: Beispiel Rollendefinition

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 namespace: default
 name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
 resources: ["pods"]
 verbs: ["get", "watch", "list"]
```

Listing 4: RoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
 name: read-pods
 namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
 name: jane # "name" is case sensitive
 apiGroup: rbac.authorization.k8s.io
roleRef:
 # "roleRef" specifies the binding to a Role / ClusterRole
 kind: Role #this must be Role or ClusterRole
 name: pod-reader # this must match the name of the Role or ClusterRole you wish to bind to
 apiGroup: rbac.authorization.k8s.io
```

Tutorialinhalt



Sensible Daten finden sich bei Google in einem und bei Azure in sieben über die Plattform bereitgestellten Images. Dazu gehören auch einige unnötige Fehlkonfigurationen, so enthält etwa das Azure-Image ciprod:ciprod08102022 in einem Build Tree von fluentd private Schlüssel einer Test-CA und das GCP-Image einen privaten Schlüssel einer snakeoil.ca in /etc/ssl/private/.

sonst Unberechtigte die Firewall-Konfiguration ändern könnten. Listing 2 zeigt eine Calico-Policy für zwei Service-Accounts.

Bei den Hyperscalern ist es möglich, die Service-Accounts aus dem IAM des Hyperscalers zu beziehen. So wird etwa jedes Deployment an einen vom Azure AD verwalteten Service-Account gebunden. Dieser kann dann Berechtigungen für andere Ressourcen im Angebot des Cloud-Anbieters (Cloud-Service-Provider, CSP) erhalten. Durchaus praktisch, aber recht komplex, da der Pod dann die anderen Ressourcen über das Netzwerk erreichen können muss, was den Kubernetes-Dienst zu einem dem Resource-Manager des CSP relativ gleichrangigen Orchestrierungswerkzeug aufwertet.

Authentifizierung über Token, Zertifikate und Plug-ins

Kubernetes kennt drei Möglichkeiten, einen Request zu authentifizieren: Token als Shared Secret, X509-Zertifikate und Authentifizierungs-Plug-ins. Token sind dabei die einfachste Methode. Jeder HTTP-Request kann sie im HTTP-Header als Bearer enthalten und die Verwaltung ist denkbar einfach. Daher sind sie die am meisten genutzte Option. Sie haben aber auch den erheblichen Nachteil, dass jeder, der dieses Geheimnis erfährt, als derjenige agieren kann, für den dieses Token ursprünglich ausgestellt wurde. Wer sich mit AD-Sicherheit beschäftigt, kennt dieses Problem von Kerberos. Der einzige Unterschied ist hier, dass man sich mehr Mühe gibt, die Token vor unbefugtem Zugriff zu schützen, und daher nicht auf jedem Client Token für hoch privilegierte Accounts speichert – oder es zumindest nicht tun sollte.

Grundsätzlich wäre ein X.509-Zertifikat eine grundlegende Verbesserung, sofern der private Schlüssel auch mit einem Passwort verschlüsselt wäre. Was er, wenn sich ein Dienst bei einem anderen Dienst anmeldet, natürlich nicht sein kann. Nun wird aus der eigentlich guten Authentifizierung dann doch wieder nur ein Geheimnis, das der Angreifer ausnutzen kann, sofern er darauf Zugriff erlangt. Im Vergleich zu Token also keine nennenswerte Verbesserung. Die privaten Schlüssel sind also genauso gut zu schützen wie die relativ simplen Token. In der Praxis wird dies daher selten eingesetzt.

Der dritte Weg ist der Austausch des Kubernetes Admission Controllers für die Autorisierung mit einer eigenen Software. Wer dies tut, hat alle Freiheiten, denn für Kubernetes zählt nur der Return-Code des Admission Controllers.

CSP-Verwundbarkeiten

Anbieter	AWS	Azure	GCP
Critical	2	9	0
High	10	7	2
Medium	16	15	12

Mit Admission Controllern wird sich ein Artikel in der nächsten Ausgabe befassen.

Identitäten der Benutzer

Für die Admins der Cluster stehen alle drei Wege offen, eine Anbindung an ein bereits vorhandenes Benutzerverzeichnis wie AD, LDAP, AAD etc. zu realisieren. Sie haben zwei Möglichkeiten. Die erste und in On-Premises-Umgebungen beliebteste ist ein Identity Proxy. Der Nutzer meldet sich dabei an einer Webseite an und im Hintergrund wird seiner Identität ein Token zugeordnet, das Kubernetes als zu diesem Benutzer zugehörig kennt. Wer direkt mit kubectl arbeitet, kennt die Datei ~/.kube/config, die die Token des Nutzers, die Adressen und CA-Zertifikate der K8S-API-Server plus Optionen enthält.

Ob der Benutzer nun ein Mensch ist oder ein externer Dienst wie eine Deployment-Pipeline der CI/CD, macht für Kubernetes erst mal keinen Unterschied. Es sind immer API-Calls an den K8S-API-Server, die ein vom Admission Controller akzeptiertes Secret enthalten müssen. Auf weitere Bootstrap-Optionen wie eine feste Liste gültiger Token gehen wir hier aus Platzgründen nicht weiter ein.

Rollen und Rechte

Wenn der Benutzer authentifiziert ist, kommt die Autorisierung, die für jeden Request festlegt, ob dieser Benutzer auch

das Recht für diese Aktion auf dieser Ressource hat. Kubernetes nutzt hier das Konzept der RoleBindings. Diese ordnen einem Benutzer eine Rolle zu. Dies geschieht entweder innerhalb eines Namespace oder für den gesamten Cluster, dann als ClusterRoleBinding.

Zuerst benötigt man die Rolle. Diese definiert, welche Aktionen auf welcher Ressource erlaubt sind. Nur über diese Verbindung erhält ein Benutzer die notwendigen Rechte zugewiesen. Alle Rechte addieren sich und es gibt keine Verbote, nur Erlaubnisse.

Neben den bereits vordefinierten Rollen sind beliebige weitere Rollen definierbar. Die eingebauten Rollen sind nur innerhalb eines Namespace gültig. Der cluster-admin ist dabei die Ausnahme. Wer diese Rolle als ClusterRoleBinding hat, ist auf dem gesamten Cluster autorisiert, alle Aktionen durchzuführen. Kubernetes bringt diese Rollen für Benutzer mit:

- cluster-admin (root user, keine Einschränkungen);
- admin (root, darf aber den Namespace selbst nicht editieren);
- edit (Objekte editieren, aber keine Rollen oder RoleBindings ändern, kann als jeder Service-Account im Namespace agieren);
- view (nur Lesen der meisten Ressourcen im Namespace, bis auf Roles, RoleBindings und Secrets).

Zusätzlich gibt es Rollen für Komponenten. Da sie nicht für Benutzer gedacht sind, bleiben sie hier außen vor. Listing 3 zeigt ein Beispiel für eine Rolle mit lesendem Zugriff auf die Liste der Pods.

Im Managed-Betrieb ist manches anders

Für viele dieser Einstellungen ist bei Managed-Kubernetes-Angeboten der Cloud-Anbieter zuständig. Im Modell der geteilten Verantwortung zwischen CSP und Kunden liegt ein großer Teil der Sicherheit in der Verantwortung des Anbieters. Er muss sicherstellen, dass ein Kunde nicht auf die Daten der anderen Kunden zugreifen kann (Multi Tenancy). Das geht nur, wenn die angebotenen Dienste und die Infrastruktur keine Verwundbarkeiten haben. Daher ist sowohl bei der Wahl des CSP als auch bei der Auswahl jeglicher Software ein Blick in die Vergangenheit angeraten. Jede Softwareschwachstelle wird in der CVE-Datenbank der Mitre (Common Vulnerabilities and Exposures) einer Nummer zugeordnet. Auch wird der Schweregrad der Verwundbarkeit im Common Vulnerability Scoring System (CVSS) festgelegt und gelistet. So lassen sich Softwarehersteller und ihre Produkte schnell vergleichen.

Für den Kunden ist der Vergleich der Anbieter – und in späteren Teilen des Tutorials dann auch der Distributionen für den Eigenbetrieb – der erste Schritt zur Absicherung. Dieser Teil will Leser in die Lage versetzen, den Betrieb von Containern abzusichern und die Entscheidung über einen Managed Service oder eine Distribution zu treffen. Ist das erfolgt, müssen die Kunden ihren Teil der Verantwortung übernehmen und für eine sichere Konfiguration sorgen. Die zwei Tabellen weiter unten führen dabei die Punkte auf, die

- vom CSP bereits erfüllt sind,
- die Kunden mit Bordmitteln umsetzen können und
- für die zusätzliche Software erforderlich ist.

Dieser Abschnitt enthält nur die To-do-Listen. In weiteren Artikeln werden ausgewählte Themen noch mit mehr Tiefgang beleuchtet.

Vergleich der Anbieter

Für Verwundbarkeiten der Cloud-Anbieter gibt es leider keine CVE-Nummern. Daher erfordert ein CSP-Vergleich ent-

Genutzte Demo-Cluster

Google Cloud Platform

Google hat Kubernetes erfunden und der Welt geschenkt. Auch heute noch stammt über ein Drittel der Codeänderungen von Google (siehe ix.de/zr6n). Es ist somit zu erwarten, dass GCPs Angebot für Managed Kubernetes besonders ausgereift ist. Zumindest bietet GCP neben den normalen Clustern und sehr viel Automatisierung auch einen Dienst namens Autopilot, der die Verwaltung noch weiter vereinfacht. Autopilot nimmt den Kunden die Verantwortung für den Kubernetes-Betrieb fast vollständig ab.

Der erste Demo-Cluster für die Untersuchung der Basissicherheit ist komplett mit den Standardeinstellungen des Web-UI für einen Autopilot „public cluster“ entstanden. Für den produktiven Betrieb sollte es allerdings ein „private cluster“ sein.

Die Analyse der Images und ihrer Sicherheit erforderte einen neuen Cluster ohne die Autopilot-Option, da die darin enthaltenen Gatekeeper-Richtlinien jegliche Analysesoftware verhindern. Die Statistik der Verwundbarkeiten ist aus den Images entstanden, die in diesem Cluster auf den Worker Nodes lagen. Die Master Nodes lassen keine Deployments zu, deshalb sind weder die Images noch die Details der Konfiguration zugänglich.

Von AquaSec Enforcer gefundene Verwundbarkeiten in den Demo-Clustern

	Critical	High	Medium	Low
Azure	240	1111	630	85
GCP	2	44	14	1

Azure

Microsoft verfolgt seit mehreren Jahren eine Pro-Linux-Strategie. Neben Linux-Images für virtuelle Maschinen sind heute auch viele andere Open-Source-Champions in der Azure-Welt zu Hause. Azure Kubernetes Service (AKS) bietet ein vollständiges und leicht zu bedienendes Managed Kubernetes, das vor allem mit einer guten Integration in die Verwaltung der Sicherheitsrichtlinien und ins Azure AD glänzt. Richtlinien-Sets können auf den gesamten Tenant, auf Managementgruppen, Subscriptions oder einzelne Cluster wirken. Das Besondere bei Azure ist, dass diese auch Konfigurationen verhindern können und nicht nur Fehler nachträglich beheben.

Der Demo-Cluster wurde auf dem Web-UI mit den Standardeinstellungen des Preset Hardened Access inklusive der vorgeschlagenen Node Pools installiert. Um den Cluster zu erreichen, benötigt man einen Bastion-Host, der aus dem Internet erreichbar ist und dessen Netz über ein Peering mit dem Netz des Clusters verfügt. Dieser Jump Host ist dann der Arbeitsplatz, da auch das Web-UI von Azure keine Details des Clusters mehr anzeigt. Die Installation des lokalen Agenten zur Analyse der Images verlief problemlos.

Empfehlungen zum BSI-IT-Grundschutz-Baustein SYS.1.6 Containerisierung

Anforderung[1]	GCP[2]	Azure[2]
SYS.1.1.A16 Sichere Grundkonfiguration von Servern (S)	COS_containerd ist ein speziell für den Einsatzzweck zusammengestelltes und gehärtetes Read-only-Image.	AKSUbuntu-1804gen2containerd-2022.10.24 ist ein speziell für den Einsatzzweck zusammengestelltes und gehärtetes Image.
DER.1.A5 Einsatz von mitgelieferten Systemfunktionen zur Detektion [Fachverantwortliche] (B)	Ein SIEM ist in GCP integriert, es können aber 3rd-Party-Produkte zum Einsatz kommen.	Ein SIEM ist in Azure integriert, es können aber 3rd-Party-Produkte zum Einsatz kommen.
OPS.1.1.2.A18 Durchgängige Protokollierung administrativer Tätigkeiten (H)	Alle Tätigkeiten auf den Nodes und im Web-UI sind für den Nutzer unveränderlich protokolliert.	Alle Tätigkeiten auf den Nodes und im Web-UI sind für den Nutzer unveränderlich protokolliert.
OPS.1.1.3.A1 Konzept für das Patch- und Änderungsmanagement [Fachverantwortliche] (B)	Autopilot-Cluster patcht GCP ohne Zutun des Nutzers, bei anderen Clustertypen ist dies optional.	Der Nutzer kann bei der Anlage des Clusters die Update-Strategie wählen, Standard ist die beste Option.
SYS.1.6.A5 Separierung der Administrations- und Zugangsnetze bei Containern (B)	Option beim Set-up sind Private Nodes. So sind die Worker und der Master nur aus den eigenen Netzen erreichbar. Empfohlene Option: Zugriffe auf die API mit Token auf das Projekt oder einen IP-Bereich einschränken, auch gestohlene Token sind so nutzlos. Dies bietet derzeit kein anderer Anbieter. Empfohlene Option: Firewallregeln auf dem Kubernetes API Server.	Mit der Option Hardened Access wird ein Private Cluster mit Kubenetes-CNI und -Network-Policies mit Calico ausgewählt. Dies ist auch bei anderen Clusteroptionen die Empfehlung.
SYS.1.6.A6 Verwendung sicherer Images (B)	Die GKE-eigene Registry ist abgekündigt, kann aber nach Verwundbarkeiten scannen, dies ist manuell zu aktivieren. GCP empfiehlt, eine Artifact Registry zu nutzen.	Die von Azure bereitgestellte Registry bietet Scans nach Verwundbarkeiten. Sie kann auch Updates in Base Images erkennen und mit Tasks eigene Images neu bauen.
SYS.1.6.A7 Persistenz von Protokollierungsdaten der Container (B)	Alle Protokolle aller Container sind automatisch im Cloud Logging enthalten und lassen sich in BigTable persistieren.	Bei der Bereitstellung wird automatisch die Funktion Container Insights aktiviert und ein Log Analytics Workspace angelegt. Container-Logs werden im Workspace persistiert.
SYS.1.6.A8 Sichere Speicherung von Zugangsdaten bei Containern (B)	Das Verschlüsseln von Secrets erfolgt automatisch, eigene Schlüssel sind möglich.	Azure Key Vault als Secret Store ist in der Option Hardened Access vorausgewählt und auch für andere Cluster die Empfehlung.
SYS.1.6.A18 Accounts der Anwendungsdienste (S)	siehe APP.4.4.A3	siehe APP.4.4.A3
SYS.1.6.A21 Erweiterte Sicherheitsrichtlinien (H)	nur per 3rd-Party-Software	nur per 3rd-Party-Software
SYS.1.6.A22 Vorsorge für Untersuchungen (H)	nur per 3rd-Party-Software	nur per 3rd-Party-Software
SYS.1.6.A24 Hostbasierte Angriffserkennung (H)	nur per 3rd-Party-Software	Defender for Cloud ist in der Option Hardened Access vorausgewählt.
SYS.1.6.A25 Hochverfügbarkeit von containerisierten Anwendungen (H)	Cluster über drei Zonen einer Region zu verteilen ist Standard bei Autopilot und sollte bei eigenen Konfigurationen auch erfolgen.	Default sind drei Zonen in einer Region. Für den API-Server kann man zwischen 99,95 und 99,5 % Verfügbarkeit wählen. Letzteres ist preiswerter.

[1] (B) Basisanforderung, (H) Anforderung bei erhöhtem Schutzbedarf, (S) Standardanforderung; [2] Erläuterung der Farben siehe Text

weder viel Recherche oder man nutzt das relativ neue Onlineangebot des Cloud-Sicherheitsanbieters WIZ (siehe ix.de/zr6n). Diese Datenbank wird von einigen Sicherheitsforschenden gepflegt und ist recht vollständig – hat allerdings noch nicht den gleichen Reifegrad erreicht wie das CVE-Pendant. Es fehlt noch ein guter Standard für die Beurteilung des Schweregrades.

Dabei ist natürlich auch zu beachten, dass kleinere Cloud-Anbieter wie Oracle und IBM auch weniger Aufmerksamkeit der Sicherheitsforschenden bekommen und die Marktführer erheblich besser untersucht sind. Für Oracle finden sich insgesamt nur zwei Einträge, Alibaba und IBM tauchen überhaupt nicht auf. Bei den drei Marktführern gibt es jedoch genügend Daten, die erhebliche Abstände zwischen ihnen zeigen. Zumindest für diese drei Cloud-Anbieter ist also ein Ranking möglich, das Google mit großem Vorsprung als Spitzenreiter in der Absicherung der eigenen Dienste ausweist. Dann kommt AWS, das wiederum einigen Vorsprung vor Azure hat, wobei die letzten beiden im Vergleich der drei ziemlich schlecht dastehen.

Angebote der Cloud-Service-Provider absichern

Grundlage für die eigenen Bemühungen zur Absicherung ist die Basissicherheit des Clusters. Diese liegt zu einem guten Teil in der Verantwortung des CSP. Daher ist es wichtig, die Anbieter selbst vergleichen zu können und auch aufzuzeigen, wo der Nutzer das Basisangebot nachbessern muss. Die dazu notwendigen Schritte sind hier kurz angerissen. Um eine Vergleichbarkeit zwischen den Anbietern und dann später im Tutorial auch zwischen den Kubernetes-Distributionen zu ermöglichen, folgt der Artikel den Bausteinen des BSI IT-Grundschutz zum Thema. Wie die Absicherung erfolgt, wird für die hier betrachteten Anbieter und Distributionen in der Tabelle kommentiert. Die Empfehlungen stützen sich dabei auf Untersuchungen mit dem Tool Trivy von AquaSec in der jeweiligen Grundkonfiguration eines frisch installierten Clusters (Stand 1.12.2022). Um die Images auf den Nodes zu erreichen, kam der Enforcer von AquSec zum Einsatz. Die in den Tabellen aufgeführten Besonderheiten der Anbieter erheben allerdings keinen Anspruch auf Vollständigkeit.

Hintergrundinformationen zur Härtungsliste

Die beiden Tabellen „Empfehlungen zum BSI-IT-Grundschutz-Baustein SYS.1.6 Containerisierung“ und „Empfehlungen zum BSI-IT-Grundschutz-Baustein APP.4.4 Kubernetes“ enthalten nur die Anforderungen, die technisch auf dem Cluster umzusetzen sind. Die organisatorischen Anforderungen, zum Beispiel nach Regelungen für den Betrieb von Containern, Freigaben, Aktualisierungen et cetera, liegen in der Verantwortung des Kunden. Auch bleiben hier die Anforderungen unberücksichtigt, die den Betrieb des Servers selbst betreffen. Bei den Managed-Kubernetes-Angeboten ist hier der CSP in der Pflicht und von den Anfor-

Empfehlungen zum BSI-IT-Grundschutz-Baustein APP.4.4 Kubernetes

Anforderung[1]	GCP[2]	Azure[2]
APP.4.4.A1 Planung der Separierung der Anwendungen (B)	Die geplanten Netzzonen und Separierung in Namespaces und Cluster ist umzusetzen.	Die geplanten Netzzonen und Separierung in Namespaces und Cluster ist umzusetzen.
APP.4.4.A3 Identitäts- und Berechtigungsmanagement bei Kubernetes (B)	GKE ist in die Berechtigungsverwaltung von GCP integriert. Alle Rollen und Rechte sind dort zu hinterlegen. Details des Rollen- und Rechtemanagements sind festzulegen und umzusetzen.	AKS bietet lokale Accounts und Azure-Active-Directory-Accounts (AAD) mit zwei RBAC-Optionen. Empfohlen ist AAD RBAC. Details des Rollen- und Rechtemanagements sind festzulegen und umzusetzen.
APP.4.4.A4 Separierung von Pods (B)	Per Default umgesetzt.	Per Default umgesetzt; erlaubt auch Confidential Nodes.
APP.4.4.A5 Datensicherung im Cluster (B)	Backup der Datenspeicher ist separat zu konfigurieren. Es gibt zusätzlich einen GKE-Backup-Service.	AKS empfiehlt Velero, dies ist vom Nutzer einzurichten.
APP.4.4.A7 Separierung der Netze bei Kubernetes (S)	siehe SYS.1.6.A5	siehe SYS.1.6.A5
APP.4.4.A9 Nutzung von Kubernetes Service Accounts (S)	Workload Identity zur automatischen Verwaltung von Accounts ist in Autopilot Default. In anderen Clustern muss es manuell aktiviert werden.	Azure AD Workload Identity ist derzeit im Betastadium. Es ist dennoch den Kubernetes-nativen Service-Accounts ohne Synchronisation vorzuziehen.
APP.4.4.A12 Absicherung der Infrastruktur-Anwendungen (S)	Abhängig von den verwendeten Anwendungen. Cloud Build und andere GCP-Tools ermöglichen die Umsetzung, sofern keine 3rd-Party-Werkzeuge im Einsatz sind. Die Berechtigungen der Pipelines muss der Kunde einstellen.	Abhängig von den verwendeten Anwendungen. ADO und andere Azure-Tools ermöglichen die Umsetzung, sofern keine 3rd-Party-Werkzeuge im Einsatz sind. Die Berechtigungen der Pipelines muss der Kunde einstellen.
APP.4.4.A13 Automatisierte Auditierung der Konfiguration (H)	Mit der Container Security API möglich. Die GKE Security Posture liefert auch gute Einsichten in den Zustand der Cluster.	Der Defender for Cloud bietet sehr viele Richtlinien-Sets inklusive CIS-Benchmarks, NIST, HIPPA und andere. Diese sollten Anwender aktivieren.
APP.4.4.A14 Verwendung dedizierter Nodes (H)	Management Nodes sind separat und in der Verwaltung durch GCP, also Shell-Zugriffe für Nutzer nicht möglich. Nodes für Speicher sind nicht notwendig. Die Aufgabe der Bastion Nodes übernehmen die Loadbalancer, etwa mit Cloud Armor als Sicherheits-Gateway.	Management Nodes sind separat und in der Verwaltung durch Azure, also Shell-Zugriffe für Nutzer nicht möglich. Nodes für Speicher sind nicht notwendig. Die Aufgabe der Bastion Nodes übernehmen die Loadbalancer, etwa mit Azure Firewall als Sicherheits-Gateway.
APP.4.4.A16 Verwendung von Operatoren (H)	möglich, aber kein Standard	möglich, aber kein Standard
APP.4.4.A17 Attestierung von Nodes (H)	Shielded Nodes mit vTPM-basiertem Secure Boot und Integrity Monitoring sind auch attestiert. Dies ist bei Autopilot standardmäßig aktiviert.	Auch Azure bietet Confidential- und Secure-Boot-IaaS für Node Pools, was seit Oktober dieses Jahres eine Attestierung des Nodes beinhaltet.
APP.4.4.A18 Verwendung von Mikro-Segmentierung (H)	Verschlüsselung der Datenverbindungen innerhalb des Clusters mit ISTIO funktioniert über die Option Anthos Service Mesh.	Entweder mit dem AKS-Add-on OpenService Mesh oder mit zusätzlichen Tools wie Istio, Linkerd oder Consul, die der Nutzer konfigurieren kann.
APP.4.4.A19 Hochverfügbarkeit von Kubernetes (H)	siehe SYS.1.6.A25	siehe SYS.1.6.A25
APP.4.4.A20 Verschlüsselte Datenhaltung bei Pods (H)	Die Datenhaltung ist per Default verschlüsselt. Optional kann der Nutzer eigene Schlüssel einsetzen.	Die Datenhaltung ist per Default verschlüsselt. Optional kann der Nutzer eigene Schlüssel einsetzen.

[1](B) Basisanforderung, (H) Anforderung bei erhöhtem Schutzbedarf, (S) Standardanforderung; [2]Erläuterung der Farben siehe Text

derungen der Bausteine „SYS.1.1 Allgemeiner Server“ und „SYS.1.3 Server unter Linux“ ist hier nur „SYS.1.1.A16 Sichere Grundkonfiguration von Servern (S)“ relevant. Mit aufgeführt sind drei übergeordnete Anforderungen, die das BSI im Zuge der Modularisierung des IT-Grundschutz-Kompendiums aus den SYS-Bausteinen in die oberen Schichten verlagert hat. Die Kommentare zur Umsetzung sind farblich hervorgehoben. Die Anforderungen werden wie folgt nach der Art der Umsetzung kategorisiert:

- Grün: Die Standardinstallation über die Weboberfläche reicht aus. Es ist keine weitere Konfiguration erforderlich. Sofern eine andere Methode zum Erstellen des Clusters genutzt wird, müssen auch die grün markierten Anforderungen explizit durch den Cluster-Ersteller aktiviert werden.
- Rot: Der Nutzer kann und sollte die Anforderungen mit den Bordmitteln umsetzen, die notwendige Konfiguration ist jedoch bei der Standardinstallation über die Weboberfläche nicht voreingestellt.
- Gelb: Es ist zusätzliche Software erforderlich.

Fazit

Bei beiden in diesem Artikel betrachteten Managed-Kubernetes-Angeboten muss der Kunde noch einiges tun oder zumindest die richtigen Entscheidungen bei der Installation treffen. GCP hat hier insgesamt dank der Autopilot-Option und der sehr viel besseren Qualität der Images des Kubernetes-Dienstes die Nase vorn. So war es in der Demo-Umgebung nicht möglich, die Analysesoftware auf einem Autopilot-Cluster zu installieren, da die Härtung mit Gatekeeper Policies und den für GKE optimierten Read-only Nodes dies schlicht nicht zuließ. Bei der Verwaltung der Cluster mit Richtlinien ist Azure aber ebenfalls sehr gut aufgestellt, da hier direkt aus dem Defender for Cloud eine zentrale Verwaltung und Überwachung erfolgt.

Wichtig ist in beiden Fällen, einen privaten Cluster zu bauen und nur die Dienste freizugeben, die auch von außerhalb erreichbar sein sollen. Dies kann entweder mit Paketfiltern oder mit Berechtigungen auf Basis der Identität der Dienste geschehen. (avr@ix.de)

Quellen

[1] Christoph Puppe; Krähennest in der Wolke; Cloud Security Operations Center im Vergleich; iX 7/2021, S. 88
[2] Christoph Puppe; Organisationen in der Cloud; iX 9/2022, S. 110
[3] Weitere Hintergrundinformationen, siehe ix.de/zr6n

CHRISTOPH PUPPE

ist Managing IT Security Architect und Auditteamleiter für ISO 27001 nach Grundschutz bei SVA System Vertrieb Alexander GmbH, Mitautor des Grundschutz-Kompendiums und ehemaliger Penetrationstester.