12 | 25 Sonderdruck für Adfinis

IT Administrator
Das Magazin für professionelle System- und Netzwerkadministration

Michael Hofer im

Interview

»Administratoren sollten sich vom Namen 'Secret' nicht täuschen lassen«

www.it-administrator.de

interview

»Administratoren sollten sich vom Namen 'Secret' nicht täuschen lassen«

Kubernetes kommt mit unzähligen Komponenten und Ressourcen daher, von denen viele geheime Informationen wie Credentials für den laufenden Betrieb benötigen. Diese Daten händisch zu pflegen ist schon in mittelgroßen Clustern ein unmögliches Unterfangen, das IT-Verantwortliche aus Sicherheitsgründen unbedingt vermeiden sollten. Welche Gefahren für solche Infrastrukturen drohen und wie Secrets-Management-Plattformen helfen, besprachen wir mit Michael Hofer, CTO bei Adfinis.

IT-Administrator: Wie verwalten die Bordmittel einer Kubernetes-Plattform Passwörter, Tokens und Anmeldeinformationen?

Michael Hofer: In seiner einfachsten Form verwaltet Kubernetes Daten wie Passwörter, Tokens und kryptographische Schlüssel – also Secrets – von Applikationen genauso wie Pods, Services und andere typische Ressourcen und Objekte. Hierzu gibt es den Objekt-Typ "Secret", der für solche sensitiven Informationen vorgesehen ist. Beim Erstellen eines Secret-Objekts werden die notwendigen Informationen über das Kubernetes-API übergeben. Dies kann zum Beispiel mittels eines CLI-Befehls beziehungsweise via YAML-Datei geschehen. Im Hintergrund persistiert Kubernetes diese standardmäßig unverschlüsselt in seiner Datenbank "etcd".

Welche Risiken entstehen dadurch?

Wie einleitend gesagt, speichert Kubernetes Secret-Objekte genau wie alle anderen Objekte ab. Administratoren sollten sich also vom Namen "Secret" nicht täuschen lassen. Eine erste Problemstellung ergibt sich dadurch, dass das System die sensitiven Informationen nicht verschlüsselt vorhält. Das birgt zum Beispiel Risiken, sollte jemand unautorisierten Zugang zum Kubernetes-Speicher erhalten. Heute ist Encryption at Rest – gerade für Secrets – eine Minimalanforderung von Organisationen, aber auch in diversen Regulationen. Kubernetes unterstützt dies von Haus aus, wenn weitere Anpassungen erfolgen. Gleichzeitig ist es essentiell, dass Organisationen den Zugang zu den Kubernetes-Secrets umso stringenter kontrollieren und einschränken. Gerade auf einem geteilten Cluster sind zu lasche Rollen- und Zugangsberechtigungen ein kritisches Risiko, da es Personen Zugang ermöglicht, den sie gar nicht haben sollten.

Und wie kann der Einsatz einer Secrets-Managementplattform hier helfen?

Die eben erläuterte Speicherung und der Zugriff auf Secrets sind nur die halbe Miete. Genauso zentral sind die folgenden Problemstellungen: Als Erste ist der Secrets-Sprawl zu nennen, denn viele IT-Umgebungen umfassen unzählige Plattformen, Systeme und Cloudanwendungen. Dies führt zu fragmentierten Anmeldeinformationen, was die Nachvollziehbarkeit und automatisierte Verwaltung erschwert, besonders bei Sicherheitsvorfällen. Auch noch immer ein Thema sind Klartext-Secrets, das auftritt, wenn diese sich nicht sicher speichern lassen. Dann sind technische Teams oft gezwungen, Secrets im Klartext in Quellcode-Repositories, Pipelines oder Konfigurationsdateien abzulegen. Auch ist es entscheidend zu wissen, welche Nutzer oder Systeme auf die sensitiven Daten zugreifen, diese ändern oder löschen. Ohne Auditierbarkeit können IT-Teams betrügerische Aktivitäten nicht überwachen oder Sicherheitsvorfälle nur erschwert mitigieren. Schließlich spielen fehlende Automatisierung und Rotation eine Rolle, denn IT-Verantwortliche sollten Maschinen-Secrets regelmäßig rotieren.

»Idealerweise hat jede Applikation eine eindeutige Maschinenidentität«

Wie greift die Kubernetes-Plattform und deren Anwendungen überhaupt auf geheime Daten zu?

Sobald erstellt, lassen sich die Kubernetes-Secret-Objekte ganz pragmatisch entweder als Datei via Volume Mount oder Umgebungsvariable in einen Pod oder Container einhängen. Anschließend kann der Applikationsprozess innerhalb des Containers darauf zugreifen und diese verwenden. Alternativ erfolgt ein Abruf über das Kubernetes-API, was den Zugriff auf Secrets ermöglicht, deren Namen erst zur Laufzeit bekannt sind. Normalerweise werden Applikationen und alle ihre unzähligen Kubernetes-Ressourcen als Ganzes automatisiert und "as code" ausgerollt. Dafür nutzen Organisationen häufig Helm, Argo CD oder Flux CD. Das bringt uns auch wieder zum Thema Secrets-Management, denn die Frage bleibt, woher kommen nun die Secrets? Erstellen technische Teams diese manuell oder werden sie etwa von Argo CD direkt im Klartext aus einem Git-Repository ausgelesen? Beides ist nicht optimal und besonders Klartext-Secrets in Git-Repositories sind ein No-Go.

Wie lassen sich solche Plattformen integrieren?

Beim Einsatz einer Secrets-Managementplattform wie OpenBao oder Public-Cloud-Werkzeugen wie dem AWS Secrets Manager gibt es diverse Ansätze, um Secrets möglichst einfach, transparent und sicher den Applikationen zur Verfügung zu stellen. Dabei ist der Kubernetes Operator wohl die meistgenutzte Option. Dieser stellt sicher, dass alle angeforderten Secrets auf Kubernetes verfügbar und aktuell sind. Im Hintergrund spricht der Operator etwa mit OpenBao oder AWS Secrets Manager, authentisiert sich, liest die geforderten Secrets aus und speichert

diese als Kubernetes-Secrets im Namensraum der Applikation ab. Falls eine Anmeldeinformation rotiert wird, merkt das ein moderner Operator und aktualisiert es automatisch. Mit dem Secrets Store CSI Driver können Apps transparent über das Container Storage Interface verschiedene Speichertypen bestellen. Hier ist es möglich, Secrets-Store-Treiber zu verzahnen, um auf diese Art und Weise auch Secrets anzufordern. Die dritte Variante ist die Sidecar Injection, die ein Deployment um einen Sidecar-Container erweitert. Damit übernimmt ein spezifischer Agent die Aufgabe, ein Secret anzufordern und es in den Applikationscontainer zu injizieren.

Welche Best Practices müssen Administratoren in punkto Integration unbedingt kennen?

Wichtig ist zu beachten, welchen Weg Secrets nehmen, um auf dem Zielsystem, also Kubernetes, und schließlich der Applikation zur Verfügung zu stehen. Welche Komponente hat wie viel Zugang und genießt wie viel Vertrauen? Idealerweise hat jede Applikation eine eindeutige Maschinenidentität und stark eingeschränkte Berechtigungen, um lediglich auf deren Secrets zuzugreifen. Gleichzeitig sollten IT-Verantwortliche bedenken, dass Integrationen wie Operator oder Sidecar Injection weitere Kubernetes-Komponenten hinzufügen. Sprich, diese sind genauso zu warten und zu überwachen.

Löst dies auch die Problematik von RBAC und somit den direkten oder indirekten Zugriff auf geheime Anmeldeinformationen?

Streng genommen nein, denn ein Secret muss immer irgendwie für eine Applikation zugänglich sein, auch wenn dieses nur während der Laufzeit im Memory liegt. Mit genügend Berechtigungen auf eine Kubernetes-Plattform oder deren Namensräume ist ein Geheimnis normalerweise als exponiert zu betrachten. Deshalb sind stringente Zugriffs- und Rollenberechtigungen essentiell für jede Plattform. Gleiches gilt auch für die verschiedenen Integrationsmöglichkeiten: Zu lasche Berechtigungen auf ein Git-Repository und dessen Pipeline steigern genauso das Risiko auf einen Secrets-Leak. Gleichzeitig helfen Secrets-Management-Werkzeuge mit Auditierbarkeit und Rotation dabei, ein Leak zu erkennen und Vertraulichkeit wiederherzustellen.

Schützt eine Secrets-Management-Plattform auch Secrets in CI/CD-Pipelines, um Leaks in Build-Logs, Artefakten oder Images zu verhindern?

Viele Pipelines nutzen schon für den Bau oder spätestens beim Hochladen von Artefakten in ein Artefakt-Repository entsprechende Secrets. Gerade wenn aber Pipelines verantwortlich sind, um alle notwendigen Secrets von OpenBao oder ähnlichen Anwendungen auszulesen und dann auf die Zielsysteme zu verteilen, besteht das Risiko, dass diese leaken. Wichtig ist vor allem, dass nur die verantwortlichen Teams die Pipelines bearbeiten und gegebenenfalls sogar ausführen können.

Welche Optionen stehen dem Admin mit einer Secrets-Managementplattform zur Verfügung, wenn dennoch einmal Anmeldeinformationen öffentlich werden?

Als erste Maßnahme sollte der Administrator das entsprechende Secret revozieren und rotieren. Hierzu ist eine Secrets-Managementplattform bereits eine Hilfe, entweder weil es die Anmeldeinformationen selbst mit einer kurzen Lebenszeit ausgestellt hat oder um wenigstens zu verstehen, wo und wer dieses Secrets konsumiert. Anschließend sollten damit zusammenhängende Sessions und Tokens auf den Zielsystemen invalidiert werden. Im Hintergrund sind sauber aggregierte und indexierte Auditlogs essentiell.

Welche typischen Audit-Problematiken fangen Secrets-Managementplattformen durch ihre Architektur ab?

Externe und interne Audits aus Sicht von Cybersecurity und Compliance stehen für viele Organisationen fast an der Tagesordnung. Eine Plattform wie OpenBao unterstützt die verantwortlichen Teams dabei, klare und nachhaltige Aussagen treffen zu können. Sei es, einen lückenlosen Auditverlauf aufzuzeigen oder eben durch eine saubere und moderne Integration der Konsumenten und Umsysteme. Im erweiterten Kontext von Sicherheitsaudits gilt es auch nicht zu vergessen, wie viel Transparenz und Nachvollziehbarkeit eine Open-Source-Software bietet. So können Organisationen nämlich auch die Software und deren Komponenten auditieren, was im Zuge von Regulationen wie EU Cyber Resilience Act und dem Kontext Supply-Chain-Security essentiell ist.

Wir danken für das Gespräch.