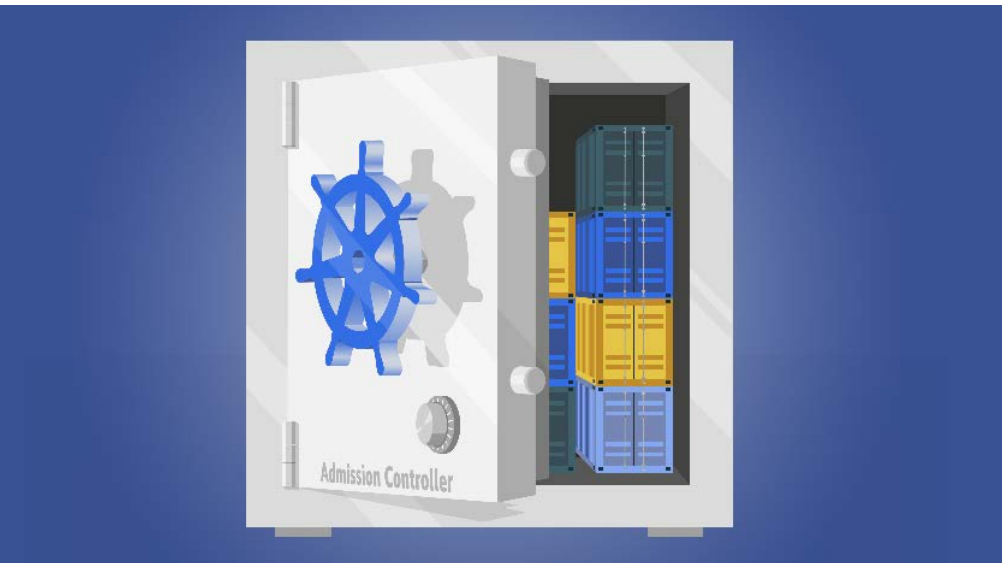


Admission Controller überwachen K8s-API

Jede Anfrage an eine Kubernetes-API wird zuerst authentifiziert und dann autorisiert. Das ist aber nur der erste Schritt in einer komplexen Kette von Bearbeitungsschritten, bevor K8s den Request ausführt. Admission Controller entscheiden, was erlaubt ist und was nicht.

Von Christoph Puppe

Der dritte Teil des Tutorials zu Kubernetes-Sicherheit legt den Fokus auf das Überprüfen von Anfragen an die K8s-API. Jede Anfrage wird erst authentifiziert, dann autorisiert und dann einer Reihe von Kontrollen übergeben, um beispielsweise die Sicherheit der Veränderung zu überprüfen. Diese Admission Controller (AC) genannten Plug-ins überprüfen die Daten und geben entweder Erfolg oder Fehler zurück. Nur eine Anfrage, die alle Überprüfungsschritte erfolgreich durchläuft, wird auch abschließend bearbeitet: K8s schreibt die Daten in die Datenbank etcd und beginnt ab diesem Augenblick, den Stand in der Datenbank mit den laufenden Prozessen zu vergleichen. Sollte es eine Abweichung geben, wird K8s beispielsweise Dienste starten, stoppen, skalieren oder das Netzwerk-Plug-in wird Pakete aufhalten oder weiterleiten.

So wie Kubernetes beim Authentifizieren und Autorisieren die eingebauten Plug-ins nutzt oder optional einen externen Webdienst über einen Webhook anspricht, sind auch die Admission Controller eingebunden. Um die Sache noch interessanter zu machen, kennt K8s zwei unterschiedliche AC-Typen: Validating Admissions können nur entscheiden, ob die Anfrage valide ist, Mutating Admissions können sie zusätzlich verändern.

Grundsätzlich haben Admission Controller zwei Aufgaben, die auch in dieser Reihenfolge zur Ausführung kommen: Sie sollen erstens statische Konfigurationen dynamisch anpassen, beispielsweise einen Sidecar-Container in einen Pod einfügen, sodass das Service Mesh Istio arbeiten kann, und zweitens sollen sie die resultierende Konfiguration mit festgelegten Regeln abgleichen.

Vor allem der zweite Punkt ist für die Verbesserung der Clustersicherheit wichtig. Hier ist die Stelle, an der jeder Container, jeder Pod, jede Netzwerkregel vorbeimuss. Und somit ist hier auch die beste Gelegenheit, Regeln nicht nur zu überwachen – was auch möglich ist –, sondern deren Einhaltung zu erzwingen. Übliche Beispiele sind:

- Wurde der Container nicht gescannt?
- Hat er zu viele Verwundbarkeiten?
- Liegt ein privater SSH-Schlüssel im Image?
- Erlaubt die Netzwerk-Policy dem Pod mit dem K8s zu reden?
- Wird dem Default ServiceUser ein RoleBinding zugewiesen?

DevOps verschiebt die Verantwortung für den Betrieb und auch für dessen Sicherheit immer weiter Richtung Entwicklung. Da diese auf Schaubildern meist links zu finden ist, spricht man von Left Shift. In dieser Welt lässt sich all dies auch in der CI/CD, in den Pipelines, im Terraform-Code abfangen. Allerdings funktioniert die Kontrolle nur bei der im Git gespeicherten Konfiguration. Erst ein Admission Controller eröffnet die Chance, alle Requests zu untersuchen. Das Platform Engineering und die IT-Sicherheitsabteilung können hier Regeln festlegen, die die DevOps-Kollegen – also Anwendungsentwickler und -betreiber – nicht ändern können. Diese Admission Controller sind üblicherweise über einen Webhook eingebunden. Dies wird in der Konfiguration definiert. Werden auch die Regeln für die ACs in einem Git verwaltet, ist hier besonders auf die Schreibrechte zu achten.

Die Regeln können entweder mit Gatekeeper aus dem Open Policy Agent (OPA) kommen oder eine der bekannten Cloud Workload Protection Platforms (CWPP) spingt als AC ein. Auch eigene AC sind jederzeit möglich, werden aber wohl eher eine Seltenheit bleiben. Wer dennoch in diese Richtung experimentieren möchte,

iX-TRACT

- Admission Controller (AC) sind ein zentrales Element zum Absichern von Kubernetes.
- Eine Standard-Kubernetes-Distribution bringt bereits die unbedingt notwendigen ACs mit.
- Auch externe Sicherheitssoftware setzt auf eigene Admission Controller.
- Das Erstellen eigener AC ist möglich und nicht schwer.

Tutorialinhalt

Teil 1: Grundlagen der Absicherung

Teil 2: Namespaces, Benutzer- und Objektverwaltung

Teil 3: Admission Controller einsetzen

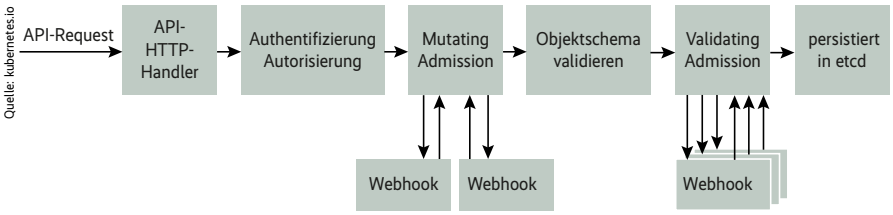
findet in den Listings 1 und 2 erste Eindrücke und in der Onlinedokumentation einen guten Start (siehe ix.de/zcwf).

Sicherheitsregeln zur Filterung

Seit K8s in Version 1.24 die PodSecurityPolicy außer Dienst gestellt hat, sind viele Nutzer auf der Suche nach einem Ersatz. Eine Option sind die seit 1.24 mitgelieferten Pod Security Standards. Sie definieren drei Sicherheitslevels und drei Reaktionen, die der mitgelieferte Admission Controller „Pod Security Admission" jeweils auf einen Namespace anwendet. Für viele ist das ausreichend, aber die Levels sind festgelegt und damit nicht sehr flexibel:

- Privileged – ohne Einschränkungen;
- Baseline – geringe Einschränkungen;
- Restricted – starke Einschränkungen gemäß Best Practices.

Als Reaktion ist einstellbar, ob Kubernetes warnen, alarmieren oder verhindern soll. Dies erfolgt über die Labels `pod-security.kubernetes.io/<MODE>: <LEVEL>` und `pod-security.kubernetes.io/<MODE>-version: <VERSION>`.



Jeder externe Request durchläuft in der Kubernetes-API mehrere Stufen, bevor er in etcd gespeichert wird.

Wer mehr möchte, benötigt einen zusätzlichen Admission Controller und Regeln. Zwei Open-Source-Varianten sind weit verbreitet: Kyverno und der Open Policy Agent (OPA) mit Gatekeeper als Admission Controller. Kommerzielle Cloud Workload Protection Platforms (CWPP) sind die kostenpflichtige Alternative, die für das Geld auch einiges an Funktionen mitbringt.

Open Policy Agent

Wer nicht auf ein kommerzielles Produkt wechseln will, hat mit dem OPA schon seit einigen Versionen eine sehr mächtige Alternative. Allerdings war die Benutzung bislang etwas umständlich. Mit dem neuen Gatekeeper als eigenständiger Software hat sich dies geändert: Er dient als einfacher zu installierende und zu pflegende Schnittstelle zum OPA. In OPA sind die Regeln in der eigenen Beschreibungssprache Rego zu hinterlegen. Auf der OPA-Webseite gibt es zudem eine umfangreiche, frei zugängliche Bibliothek, die bei den meisten Anwendungsfällen hilft (siehe ix.de/zcwf). Nur selten wird es nötig sein, eigene Regeln zu schreiben.

Kyverno

Ein anderes Open-Source-Angebot ist Kyverno. Hier gibt es keinen zusätzlichen

Listing 1: Beispiel für einen Admission Controller

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
 - name: EventRateLimit
   path: eventconfig.yaml
...
```

Listing 2: Konfigurationsbeispiel EventRateLimit Controller

```
apiVersion: eventratelimit.admission.k8s.io/v1alpha1

kind: Configuration
limits:
 - type: Namespace
   qps: 50
   burst: 100
   cacheSize: 2000
 - type: User
   qps: 10
   burst: 50
```

Listing 3: Eine Rego-Regel, die Labels erzwingt

```
package k8srequiredlabels

       get_message(parameters, _default) = msg {
         not parameters.message
         msg := _default
       }

       get_message(parameters, _default) = msg {
         msg := parameters.message
       }

       violation[{"msg": msg, "details": {"missing_labels": missing}}] {
         provided := {label | input.review.object.metadata.labels[label]}
         required := {label | label := input.parameters.labels[_].key}
         missing := required - provided
         count(missing) > 0
         def_msg := sprintf("you must provide labels: %v", [missing])
         msg := get_message(input.parameters, def_msg)
       }

       violation[{"msg": msg}] {
         value := input.review.object.metadata.labels[key]
         expected := input.parameters.labels[_]
         expected.key == key
         # do not match if allowedRegex is not defined, or is an empty string
         expected.allowedRegex != ""
         not re_match(expected.allowedRegex, value)
         def_msg := sprintf("Label <%v: %v> does not satisfy allowed regex: %v", [key, value, expected.allowedRegex])
         msg := get_message(input.parameters, def_msg)
       }
```

Listing 4: Aufruf der Rego-Regel mit Parametern

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
 name: all-must-have-owner
spec:
 match:
   kinds:
     - apiGroups: [""]
       kinds: ["Namespace"]
 parameters:
   message: "All namespaces must have an `owner` label that points to your company username"
   labels:
     - key: owner
       allowedRegex: "^[a-zA-Z]+.agilebank.demo$"
```

Listing 5: Beispielregel für Kyverno

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-labels
spec:
  validationFailureAction: enforce
  rules:
  - name: check-for-labels
    match:
      any:
      - resources:
          kinds:
          - Pod
    validate:
      message: "label 'app.kubernetes.io/name' is required"
      pattern:
        metadata:
          labels:
            app.kubernetes.io/name: "?*"
```

Dienst wie den OPA bei Gatekeeper. Alle Regeln liegen in der Konfiguration. Listing 5 zeigt ein einfaches Beispiel.

Der Vergleich der Listings zeigt gut, warum Kyverno bei vielen beliebt ist, auch wenn die OPA-Gatekeeper-Kombination sehr viel öfter anzutreffen ist und vor allem bei den großen Anwendern zum Einsatz kommt. Auch ist aus den Listings zu ersehen, warum kommerzielle Produkte hier durchaus ihren Mehrwert haben, denn sie verpacken die gleiche Funktionalität hinter einer API und einem Web-UI, das auch für Menschen bedienbar ist, die nicht den ganzen Tag Rego schreiben oder debuggen wollen.

Jedes Feature ist eine Flanke

Nicht unerwähnt sollte bleiben, dass auch Angreifende einen Admission Controller im K8s unterbringen könnten. Allerdings ist die Wahrscheinlichkeit recht gering, da das einen Neustart des API-Servers erfordert. Wer aber Zugriff auf die Boot-Images der Control Plane hat, könnte so alle auf diesen Images basierenden K8s-Cluster unter seine Kontrolle bringen. Abgesehen von diesem vielleicht etwas theoretischen Szenario sind manche Admission Controller selbst ja auch nur Webserver, die aus einem Image starten. Auch hier kann ein Angreifender ansetzen und Code einschmuggeln.

Der dritte Weg, dieses Feature zu missbrauchen, sind die Regeln selbst. Wer schreibenden Zugriff auf das Regelwerk hat, kann seine Zugriffsrechte entweder erhöhen oder persistieren. Bei einem Mutating Admission Controller ist es auch einfach, jegliche Anfrage an den Kubernetes-API-Server beliebig zu verändern.

Wer dies verhindern möchte, muss die Schreibrechte auf die relevanten Images, die Regeln im Git und auch die Labels in Kubernetes selbst stark reduzieren und jede Veränderung überwachen.

Fazit

Admission Controller sind ein flexibler Weg, Anforderungen an die Pods und die Konfiguration der Anwendungen im Cluster sowie des Clusters selbst durchzusetzen. Hierfür stehen mit Kyverno, OPA und der mitgelieferten Pod Security Admission gleich drei Open-Source-Varianten zur Verfügung. Auch der Markt kommerzieller CWPP-Produkte bietet reichhaltige Auswahl. Eine gute Absicherung wird an allen Stellen ansetzen, also im Git bereits nach Fehlern im Code der Anwendungen und im Code der Infrastruktur suchen und auch die Images schon beim Erstellen auf Sicherheitsprobleme testen. Die Regelwerke werden schnell komplex. Automatisierung kann hier unterstützen.

Jeder, der das Thema Sicherheit ernst nimmt, sollte die zusätzlichen Kontrollmöglichkeiten eines Admission Controllers nutzen. Das erfordert es, einige Arbeit in möglichst präzise Regeln zu stecken. Auch hier können Automatismen helfen. Entweder erstellen sie die Regeln aus dem Code oder sind als Selbst-lern-Option in den CWPP vorhanden. Wer dies in seinen Bereitstellungsprozess fest einbaut, kann seine Umgebung sehr viel besser absichern, als dies früher möglich war. (avr@ix.de)

Quellen

Weitere Informationen zur Dokumentation der Admission Controller und zur OPA-Rego-Bibliothek unter ix.de/zcwf

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.