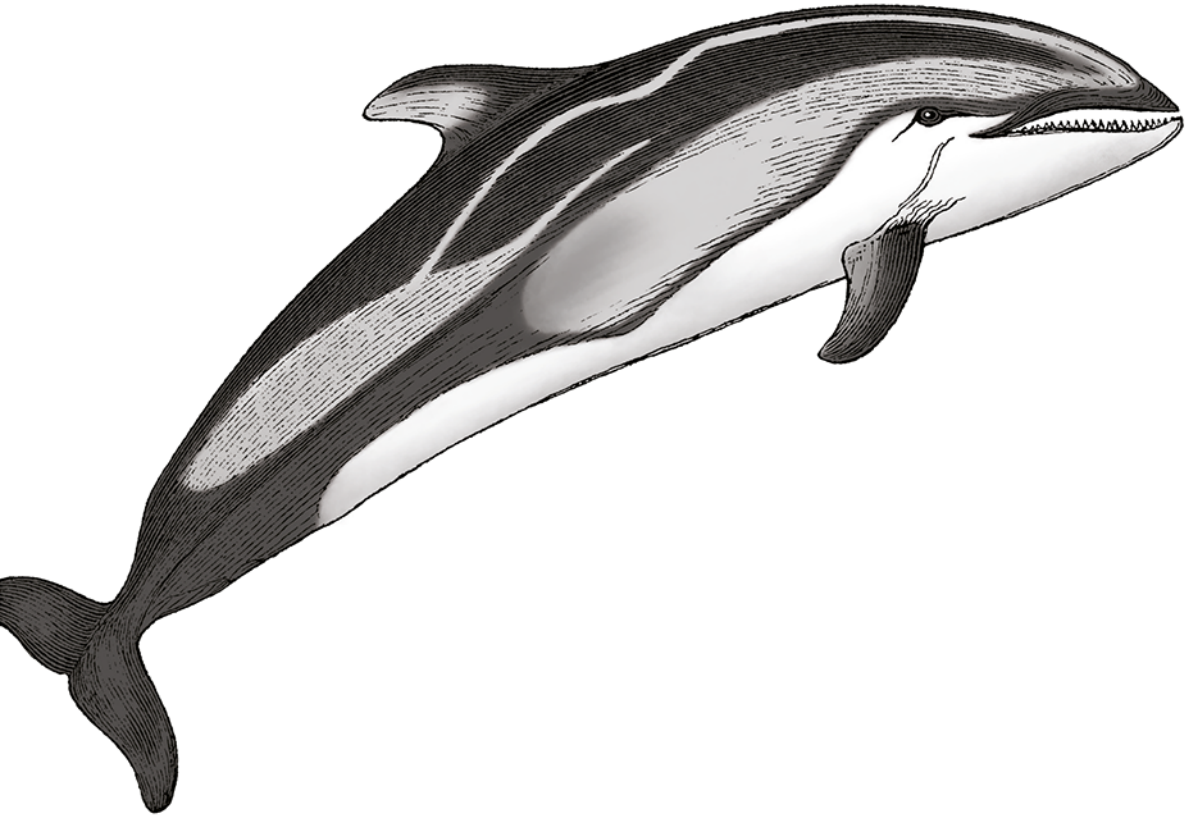
O'REILLY®

Wydanie III

Kubernetes

Tworzenie niezawodnych systemów rozproszonych



Brendan Burns, Joe Beda
Kelsey Hightower, Lachlan Evenson

Helion

Tytuł oryginału: Kubernetes: Up and Running: Dive into the Future of Infrastructure, 3rd Edition

Tłumaczenie: Łukasz Piwko z wykorzystaniem fragmentów poprzedniego wydania w przekładzie Lecha Lachowskiego

ISBN: 978-83-8322-336-0

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/kuber3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/kuber3.zip

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 19.

Zabezpieczanie aplikacji w Kubernetes

Zapewnienie bezpiecznej platformy do wykonywania aplikacji ma krytyczne znaczenie pod kątem szerokiego wykorzystywania Kubernetes w środowisku produkcyjnym. Na szczęście Kubernetes ma wiele różnych API zabezpieczeń, za pomocą których można stworzyć bezpieczne środowisko. To, że jest ich dużo, może się jednak okazać problemem, bo trzeba zadeklarować, których chce się używać. Posługiwanie się tymi API może być niewygodne i skomplikowane, co jeszcze bardziej utrudnia osiągnięcie celów w zakresie bezpieczeństwa.

Aby skutecznie zabezpieczyć kapsuły w Kubernetes, należy na wstępie zrozumieć dwa pojęcia: dogłębna obrona i zasada minimalnych uprawnień. **Dogłębna obrona** to koncepcja polegająca na wykorzystaniu kilku warstw środków bezpieczeństwa w systemach komputerowych zawierających także Kubernetes. **Zasada minimalnych uprawnień** oznacza natomiast, że obciążeniom należy udzielać dostępu tylko do tych zasobów, które są im niezbędne do działania. Obie te koncepcje są powszechnie stosowane w nieustannie zmieniającym się krajobrazie systemów komputerowych.

W tym rozdziale opisujemy interfejsy API Kubernetes zawierające narzędzia bezpieczeństwa, które można stopniowo stosować, aby zabezpieczyć obciążenia na poziomie kapsuły.

Kontekst zabezpieczeń

Podstawę zabezpieczeń kapsuł stanowi kontekst zabezpieczeń (SecurityContext). Jest to zbiór wszystkich pól dotyczących bezpieczeństwa, których można używać zarówno na poziomie kapsuły, jak i specyfikacji kontenera. Oto kilka przykładowych środków bezpieczeństwa dostępnych w kontekście zabezpieczeń:

- kontrola uprawnień i dostępu użytkowników (na przykład ustawianie identyfikatora użytkownika i grupy),
- główny system plików tylko do odczytu,
- zezwalanie na podnoszenie uprawnień,
- przypisania profili i etykiet Seccomp, AppArmor i SELinux,
- wykonywanie aplikacji jako użytkownik z uprawnieniami i bez uprawnień.

Na listingu 19.1 pokazano przykładową kapsułę ze zdefiniowanym kontekstem zabezpieczeń.

Listing 19.1. kuard-pod-securitycontext.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: kuard
spec:
    securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
    containers:
      - image: gcr.io/kuar-demo/kuard-amd64:blue
        name: kuard
        securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            privileged: false
        ports:
          - containerPort: 8080
            name: http
            protocol: TCP
```

W tym przykładzie kontekst zabezpieczeń jest zdefiniowany zarówno na poziomie kapsuły, jak i na poziomie kontenera. Wiele środków bezpieczeństwa można stosować na obu tych poziomach. W takim przypadku konfiguracja na poziomie kontenera jest ważniejsza. Przyjrzymy się polom, które zdefiniowaliśmy w tej specyfikacji kapsuły, oraz zastanowimy się, jaki mają wpływ na bezpieczeństwo naszego obciążenia.

`runAsNonRoot`

: Kapsuła lub kontener muszą być uruchomione na koncie użytkownika innego niż `root`. Uruchomienie kontenera jako użytkownik `root` się nie powiedzie. Uruchamianie przy użyciu innego użytkownika niż `root` jest uważane za dobrą praktykę, ponieważ wiele błędów konfiguracji i exploitów wykorzystuje środowisko wykonawcze kontenera z uprawnieniami użytkownika głównego. To pole można ustawić zarówno w kontekście PodSecurityContext, jak i Security Context. Obraz kontenera `kuard` jest skonfigurowany w pliku *Dockerfile* (*https://oreil.ly/4IZI7*) tak, aby był uruchamiany jako użytkownik „nobody”. Uruchamianie kontenerów jako inny użytkownik niż `root` zawsze jest dobrym wyborem. Jeśli jednak uruchamiasz kontener pobrany z innego, który nie ma wprost ustawionego użytkownika, może być konieczne rozszerzenie oryginalnego pliku *Dockerfile*, aby to zmienić. Ta metoda nie zawsze działa, ponieważ aplikacja może mieć jeszcze inne wymagania, które należy uwzględnić.

`runAsUser/runAsGroup`

: To ustawienie zastępuje użytkownika i grupę wykorzystywane do uruchamiania procesu kontenera. Obrazy kontenerów mogą mieć to pole skonfigurowane w pliku *Dockerfile*.

`fsgroup`

: Konfiguruje Kubernetes, aby zmieniał grupę wszystkich plików w woluminie podczas ich montowania w kapsule. Dodatkowo można użyć pola `fsGroupChangePolicy`, by dokładniej skonfigurować ustawienia.

`allowPrivilegeEscalation`
: Określa, czy proces w kontenerze może uzyskać więcej uprawnień niż jego proces nadrzędny. To pole jest popularnym wektorem ataku i należy je ustawić na `false`. Ponadto trzeba mieć świadomość, że będzie ono miało wartość `true`, jeśli pole `privileged` zostanie ustawione na `true`.

`privileged`
: Uruchamia kontener jako uprzywilejowany, co zwiększa jego uprawnienia do poziomu uprawnień hosta.

`readOnlyRootFilesystem`
: Montuje główny system plików kontenera jako tylko do odczytu. Jest to popularny wektor ataku i dobrą praktyką jest włączenie tego ustawienia. Wszystkie dane lub dzienniki wymagające zapisu można zamontować przez wolumin.

Pola pokazane w tym miejscu nie wyczerpują listy wszystkich dostępnych środków bezpieczeństwa, ale wystarczą na początek pracy z kontekstem zabezpieczeń. W dalszej części tego rozdziału opisujemy jeszcze parę innych podobnych pól.

Teraz utworzymy kapsułę przez zapisanie przykładowego kodu w pliku o nazwie *kuard-pod-securitycontext.yaml*. Pokażemy, jak zastosować konfigurację kontekstu bezpieczeństwa do działającej kapsuły. Utwórz kapsułę za pomocą następującego polecenia:

```
$ kubectl create -f kuard-pod-securitycontext.yaml
pod/kuard created
```

Następnie uruchom powłokę w kontenerze `kuard` i sprawdź identyfikatory użytkownika i grupy, pod których kontrolą działają procesy:

```
$ kubectl exec -it kuard -- ash
/ $ id
uid=1000 gid=3000 groups=2000
/ $ ps
PID   USER       TIME    COMMAND
    1 1000         0:00  /kuard
   30 1000         0:00  ash
   37 1000         0:00  ps
/ $ touch file
touch: file: Read-only file system
```

Jak widać, uruchomiona powłoka, `ash`, działa pod kontrolą użytkownika o identyfikatorze (`uid`) 1000 i grupy o identyfikatorze (`gid`) 3000 oraz znajduje się w grupie 2000. Ponadto widzimy, że proces `kuard` działa jako użytkownik 1000 zgodnie z definicją SecurityContext w specyfikacji kapsuły. Potwierdziliśmy też, że nie możemy tworzyć żadnych nowych plików, ponieważ kontener jest tylko do odczytu. Jeśli zastosujesz opisane zmiany, masz wszystko, czego potrzeba na początek.

Teraz wprowadzimy kilka innych środków bezpieczeństwa należących do SecurityContext, które umożliwiają jeszcze dokładniejszą kontrolę dostępu i uprawnień obciążeń. Zaczniemy od środków bezpieczeństwa na poziomie systemu operacyjnego i pokażemy, jak je skonfigurować przez kontekst bezpieczeństwa. Należy pamiętać, że wiele z nich jest zależnych od systemu operacyjnego hosta. To znaczy, że mogą dotyczyć tylko kontenerów działających w systemach Linux, a nie w innych systemach obsługiwanych przez Kubernetes, takich jak Windows. Poniżej znajduje się lista podstawowych zabezpieczeń systemu operacyjnego, które obejmuje SecurityContext.

Umiejętności (ang. *capabilities*)

Umożliwiają dodanie lub usunięcie grup uprawnień, które mogą być potrzebne do działania obciążenia. Na przykład dane obciążenie może definiować konfigurację sieci hosta. Zamiast nadawać uprawnienia kapsule, co w istocie oznacza udzielenie dostępu na poziomie użytkownika `root` hosta, można dodać specjalną „umiejętność" konfiguracji sieci hosta (nazwa tej umiejętności to `NET_ADMIN`). Taki sposób postępowania jest zgodny z zasadą minimalnych uprawnień.

AppArmor

Określa pliki, do których procesy mają dostęp. Profile AppArmor można stosować do kontenerów przez dodanie adnotacji `container.apparmor.security.beta.kubernetes.io/`*<nazwa_kontenera>*`:` *<ref_profilu>* do specyfikacji kapsuły. Akceptowalne wartości *<ref_profilu>* to `runtime/default`, `localhost/`*<ścieżka do profilu>* i `unconfined`. Domyślną wartością jest `unconfined`, która oznacza brak profilu do zastosowania.

Seccomp

Profile Seccomp (ang. *secure computing* — bezpieczne obliczenia) umożliwiają tworzenie filtrów wywołań systemowych, za pomocą których można przepuszczać lub blokować wybrane wywołania systemowe, aby ograniczyć obszar jądra Linuksa dostępny dla procesów w kapsułach.

SELinux

Definiuje środki kontroli dostępu do plików i procesów. Operatory SELinux używają etykiet, które są podzielone na grupy tworzące kontekst bezpieczeństwa (nie mylić z kontekstem zabezpieczeń SecurityContext Kubernetes) ograniczający dostęp do procesu. Domyślnie Kubernetes przydziela losowy kontekst SELinux każdemu kontenerowi, ale można ustawić wybrany przez SecurityContext.

Zarówno AppArmor, jak i Seccomp mogą ustawiać domyślny profil środowiska wykonawczego, który ma być używany. Każde środowisko wykonawcze kontenera ma domyślne profile AppArmor i Seccomp — zostały one starannie skonfigurowane tak, aby ograniczać powierzchnię ataku przez usunięcie wywołań systemowych i dostępu do plików, które są znanymi wektorami ataku lub nie są powszechnie używane przez aplikacje. Te domyślne ustawienia rzadko wpływają na działanie obciążeń i stanowią doskonały punkt startowy.

Aby zademonstrować zastosowanie tych środków bezpieczeństwa do kapsuły, użyjemy narzędzia o nazwie amicontained („Am I contained", *https://oreil.ly/6ubkU*), napisanego przez Jess Frazelle. Zapisz specyfikację kapsuły z listingu 19.2 w pliku o nazwie *amicontained-pod.yaml*. Pierwsza kapsuła nie ma zastosowanego kontekstu SecurityContext i zostanie użyta do pokazania, które środki bezpieczeństwa są domyślnie wykorzystywane w kapsułach. Pamiętaj, że u Ciebie wyniki mogą być odmienne, jeśli używasz innej dystrybucji Kubernetes i innych usług zarządzanych.

Listing 19.2. amicontained-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: amicontained
```

```
spec:
    containers:
      - image: r.j3ss.co/amicontained:v0.4.9
        name: amicontained
        command: [ "/bin/sh", "-c", "--" ]
        args: [ "amicontained" ]
```

Utwórz kapsułę `amicontainer`:

```
$ kubectl apply -f amicontained-pod.yaml
pod/amicontained created
```

Zajrzymy do dzienników kapsuły, aby obejrzeć wyniki działania narzędzia `amicontained`:

```
$ kubectl logs amicontained
Container Runtime: kube
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: docker-default (enforce)
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid
        setpcap net_bind_service net_raw sys_chroot mknod audit_write
        setfcap
Seccomp: disabled
Blocked Syscalls (21):
        SYSLOG SETPGID SETSID VHANGUP PIVOT_ROOT ACCT SETTIMEOFDAY UMOUNT2
        SWAPON SWAPOFF REBOOT SETHOSTNAME SETDOMAINNAME INIT_MODULE
        DELETE_MODULE LOOKUP_DCOOKIE KEXEC_LOAD FANOTIFY_INIT
        OPEN_BY_HANDLE_AT FINIT_MODULE KEXEC_FILE_LOAD
Looking for Docker.sock
```

Z powyższych danych wynika, że został zastosowany domyślny profil środowiska wykonawczego AppArmor. Ponadto widzimy umiejętności, które są dozwolone domyślnie, oraz dowiadujemy się, że wyłączono Seccomp. Na końcu znajduje się lista 21 domyślnie zablokowanych wywołań systemowych. Na tej podstawie możemy zastosować środki bezpieczeństwa Seccomp, AppArmor i Capabilities do specyfikacji kapsuły. Utwórz plik o nazwie *amicontained-pod-securitycontext.yaml* i zawartości przedstawionej na listingu 19.3.

Listing 19.3. amicontained-pod-securitycontext.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: amicontained
    annotations:
        container.apparmor.security.beta.kubernetes.io/amicontained: "runtime/default"
spec:
    securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
        seccompProfile:
            type: RuntimeDefault
    containers:
      - image: r.j3ss.co/amicontained:v0.4.9
```

```
    name: amicontained
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "amicontained" ]
    securityContext:
        capabilities:
            add: ["SYS_TIME"]
            drop: ["NET_BIND_SERVICE"]
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
        privileged: false
```

Najpierw musimy usunąć istniejącą kapsułę `amicontained`:

```
$ kubectl delete pod amicontained
pod "amicontained" deleted
```

Teraz możemy utworzyć nową kapsułę z zastosowanym kontekstem SecurityContext. Deklarujemy, że chcemy wykorzystać domyślne profile środowiska wykonawczego AppArmor i Seccomp. Ponadto dodaliśmy i usunęliśmy po jednej umiejętności:

```
$ kubectl apply -f amicontained-pod-securitycontext.yaml
pod/amicontained created
```

Jeszcze raz zajrzymy do dzienników kapsuły, aby obejrzeć wyniki działania narzędzia `amicontained`:

```
$ kubectl logs amicontained
Container Runtime: kube
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: docker-default (enforce)
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid setpcap
        net_raw sys_chroot sys_time mknod audit_write setfcap
Seccomp: filtering
Blocked Syscalls (67):
        SYSLOG SETUID SETGID SETPGID SETSID SETREUID SETREGID SETGROUPS
        SETRESUID SETRESGID USELIB USTAT SYSFS VHANGUP PIVOT_ROOT SYSCTL ACCT
        SETTIMEOFDAY MOUNT UMOUNT2 SWAPON SWAPOFF REBOOT SETHOSTNAME
        SETDOMAINNAME IOPL IOPERM CREATE_MODULE INIT_MODULE DELETE_MODULE
        GET_KERNEL_SYMS QUERY_MODULE QUOTACTL NFSSERVCTL GETPMSG PUTPMSG
        AFS_SYSCALL TUXCALL SECURITY LOOKUP_DCOOKIE VSERVER MBIND SET_MEMPOLICY
        GET_MEMPOLICY KEXEC_LOAD ADD_KEY REQUEST_KEY KEYCTL MIGRATE_PAGES
        FUTIMESAT UNSHARE MOVE_PAGES PERF_EVENT_OPEN FANOTIFY_INIT
        NAME_TO_HANDLE_AT OPEN_BY_HANDLE_AT SETNS PROCESS_VM_READV
        PROCESS_VM_WRITEV KCMP FINIT_MODULE KEXEC_FILE_LOAD BPF USERFAULTFD
        PKEY_MPROTECT PKEY_ALLOC PKEY_FREE
Looking for Docker.sock
```

Wyzwania związane z kontekstem zabezpieczeń

Jak widać, posługiwanie się kontekstem zabezpieczeń SecurityContext jest dość skomplikowane i nie jest łatwo zastosować podstawowego zestawu środków bezpieczeństwa przez bezpośrednie skonfigurowanie wszystkich pól każdej kapsuły. Tworzenie profili i kontekstów AppArmor, Seccomp i SELinux oraz zarządzanie nimi jest trudne i można przy tym popełnić błąd. Błędy uniemożliwiają aplikacji spełnianie swojej funkcji. Istnieje kilka narzędzi pozwalających na wygenerowanie

z działającej kapsuły profilu Seccomp, który następnie można zastosować przy użyciu SecurityContext. Jednym z takich projektów jest Security Profiles Operator (*https://oreil.ly/grPCN*), który ułatwia generowanie profili Seccomp i zarządzanie nimi. Teraz przyjrzymy się innym interfejsom API zabezpieczeń, które pozwalają na spójne zastosowanie kontekstu SecurityContext w całym klastrze.

Pod Security

Wiesz już, jak zarządzać środkami bezpieczeństwa kapsuł i kontenerów za pomocą SecurityContext, więc teraz pokażemy Ci, jak zastosować zbiór wartości kontekstu zabezpieczeń na większą skalę. Kubernetes ma obecnie wycofywany interfejs API PodSecurityPolicy (PSP), który umożliwia zarówno walidację, jak i mutację. **Walidacja** nie pozwoli na utworzenie zasobów Kubernetes, jeśli nie będą miały zastosowanego określonego kontekstu SecurityContext. **Mutacja** natomiast zmienia zasoby Kubernetes i stosuje określony SecurityContext na podstawie kryteriów pochodzących od PSP. Zważywszy na to, że interfejs PSP jest wycofywany i zostanie usunięty w Kubernetes v1.25, nie opisujemy go szczegółowo, za to więcej miejsca poświęcamy jego następcy — Pod Security. Jedną z najważniejszych różnic między Pod Security a poprzednim API jest to, że Pod Security przeprowadza tylko walidację. Jeśli chcesz dowiedzieć się więcej o mutacji, przeczytaj rozdział 20.

Czym jest Pod Security

Pod Security umożliwia deklarowanie różnych profili bezpieczeństwa dla kapsuł. Są one nazywane standardami Pod Security (ang. *Pod Security Standards*) oraz stosowane na poziomie przestrzeni nazw. Pod Security Standards to kolekcja pól dotyczących bezpieczeństwa w specyfikacji kapsuły (wliczając między innymi SecurityContext) i powiązanych z nimi wartości. Istnieją trzy różne standardy obejmujące zakres od ścisłego do swobodnego poziomu zabezpieczeń. Idea jest taka, że można zastosować ogólny zestaw zabezpieczeń do wszystkich kapsuł w danej przestrzeni nazw. Poniżej znajduje się opis trzech wspomnianych standardów zabezpieczeń:

Baseline
: Najczęściej używana eskalacja uprawnień ułatwiająca wdrażanie.

Restricted
: Wysoce restrykcyjny standard uwzględniający najlepsze praktyki w zakresie bezpieczeństwa. Może spowodować awarię obciążenia.

Privileged
: Otwarty i nieograniczony.

Interfejs Pod Security w Kubernetes v1.23 znajduje się w fazie beta i może ulec zmianom.

Każdy standard Pod Security definiuje listę pól w specyfikacji kapsuły oraz ich dozwolone wartości. Poniżej wymieniliśmy kilka pól obejmowanych przez te standardy:

- `spec.securityContext`
- `spec.containers[*].securityContext`
- `spec.containers[*].ports`
- `spec.volumes[*].hostPath`

Kompletna lista pól ze wszystkich standardów Pod Security znajduje się w oficjalnej dokumentacji (*https://oreil.ly/xPK2p*).

Każdy standard jest stosowany do przestrzeni nazw za pomocą określonego trybu. Są trzy tryby, do których zasady mogą zostać odniesione:

Enforce
: Wszystkie kapsuły łamiące zasady zostaną odrzucone.

Warn
: Kapsuły łamiące zasady nie zostaną odrzucone, ale zostanie wyświetlone ostrzeżenie.

Audit
: Kapsuły łamiące zasady będą generować komunikat audytowy w dzienniku audytów.

Stosowanie standardów Pod Security

Standardy Pod Security stosuje się do przestrzeni nazw za pomocą etykiet w następujący sposób:

- Wymagane: `pod-security.kubernetes.io/<TRYB>: <POZIOM>`.
- Opcjonalne: `pod-security.kubernetes.io/<TRYB>-version: <WERSJA>` (domyślnie najnowsza).

Przestrzeń nazw na listingu 19.4 pokazuje, jak zastosować tryb `enforce` w jednym standardzie (w tym przypadku `baseline`) oraz tryby `audit` i `warn` w innym (`restricted`). Wykorzystanie różnych trybów umożliwia wdrożenie zasad o niższym poziomie zabezpieczeń oraz przeprowadzenie audytu w celu wykrycia, które obciążenia łamią reguły standardu za pomocą bardziej rygorystycznych zasad. Następnie można rozwiązać znalezione problemy z zabezpieczeniami przed wdrożeniem bardziej restrykcyjnego standardu. Ponadto można przypisać tryb do konkretnej wersji, na przykład v1.22. To umożliwia zmianę standardów zasad z każdą wersją Kubernetes oraz pozwala na odniesienie się do określonej wersji. Na listingu 19.4 ustawiliśmy standard `baseline` na tryb `enforce`, a standardy `warn` i `audit` — na tryb `restricted`. Wszystkie tryby są powiązane z wersją v1.22 standardu.

Listing 19.4. baseline-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
    name: baseline-ns
    labels:
        pod-security.kubernetes.io/enforce: baseline
        pod-security.kubernetes.io/enforce-version: v1.22
```

```
        pod-security.kubernetes.io/audit: restricted
        pod-security.kubernetes.io/audit-version: v1.22
        pod-security.kubernetes.io/warn: restricted
        pod-security.kubernetes.io/warn-version: v1.22
```

Pierwsze wdrożenie zasad może być przytłaczającym zadaniem. Na szczęście interfejs Pod Security pozwala na sprawdzenie w łatwy sposób, które istniejące obciążenia łamią zasady standardu, za pomocą polecenia `dry-run`:

```
$ kubectl label --dry-run=server --overwrite ns \
  --all pod-security.kubernetes.io/enforce=baseline
Warning: kuard: privileged
namespace/default labeled
namespace/kube-node-lease labeled
namespace/kube-public labeled
Warning: kube-proxy-vxjwb: host namespaces, hostPath volumes, privileged
Warning: kube-proxy-zxqzz: host namespaces, hostPath volumes, privileged
Warning: kube-apiserver-kind-control-plane: host namespaces, hostPath volumes
Warning: etcd-kind-control-plane: host namespaces, hostPath volumes
Warning: kube-controller-manager-kind-control-plane: host namespaces, ...
Warning: kube-scheduler-kind-control-plane: host namespaces, hostPath volumes
namespace/kube-system labeled
namespace/local-path-storage labeled
```

To polecenie sprawdza wszystkie kapsuły w klastrze Kubernetes pod kątem standardu Pod Security `baseline` i zgłasza w wyniku wszystkie przypadki złamania zasad jako ostrzeżenia.

Obejrzyjmy Pod Security w akcji. Utwórz plik o nazwie *baseline-ns.yaml* o zawartości pokazanej na listingu 19.5.

Listing 19.5. baseline-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
    name: baseline-ns
    labels:
        pod-security.kubernetes.io/enforce: baseline
        pod-security.kubernetes.io/enforce-version: v1.22
        pod-security.kubernetes.io/audit: restricted
        pod-security.kubernetes.io/audit-version: v1.22
        pod-security.kubernetes.io/warn: restricted
        pod-security.kubernetes.io/warn-version: v1.22

        $ kubectl apply -f baseline-ns.yaml
        namespace/baseline-ns created
```

Utwórz plik o nazwie *kuard-pod.yaml* o zawartości pokazanej na listingu 19.6.

Listing 19.6. kuard-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: kuard
    labels:
    app: kuard
spec:
```

```
    containers:
      - image: gcr.io/kuar-demo/kuard-amd64:blue
        name: kuard
        ports:
          - containerPort: 8080
            name: http
            protocol: TCP
```

Utwórz kapsułę i wyświetl wynik za pomocą następującego polecenia:

```
$ kubectl apply -f kuard-pod.yaml --namespace baseline-ns
Warning: would violate "v1.22" version of "restricted" PodSecurity profile:
allowPrivilegeEscalation != false (container "kuard" must set
securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container
"kuard" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or
container "kuard" must set securityContext.runAsNonRoot=true), seccompProfile (pod or
container "kuard" must set securityContext.seccompProfile.type to "RuntimeDefault" or
"Localhost")
pod/kuard created
```

W tych wynikach widać, że kapsuła została pomyślnie utworzona, ale złamała standard `restricted` Pod Security. W wynikach są podane szczegóły, dzięki czemu można poradzić sobie z tym problemem. Ponadto otrzymaliśmy komunikat w dzienniku audytów serwera API, ponieważ skonfigurowaliśmy tryb `audit`:

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"...
```

Użycie interfejsu Pod Security to doskonały sposób na zarządzanie poziomem bezpieczeństwa obciążeń przez zastosowanie zasad w obrębie przestrzeni nazw i umożliwienie tworzenia kapsuł tylko w sytuacji, gdy nie łamią tych zasad. Ten elastyczny API zawiera różne gotowe zasady, od bardzo swobodnych po wyjątkowo restrykcyjne, oraz narzędzia ułatwiające wdrażanie zmian zasad bez ryzyka wywołania awarii obciążeń.

Zarządzanie kontami usług

Konta usług to zasoby Kubernetes nadające tożsamość obciążeniom, które działają w kapsułach. Za pomocą RBAC można kontrolować, przez API Kubernetes, do jakich zasobów dana tożsamość ma dostęp. Więcej informacji na ten temat znajduje się w rozdziale 14. Jeśli Twoja aplikacja nie wymaga dostępu do API Kubernetes, wyłącz dostęp zgodnie z zasadą minimalnych uprawnień. Kubernetes tworzy domyślne konto usług w każdej przestrzeni, które zostaje automatycznie ustawione jako konto usług dla wszystkich kapsuł. Zawiera ono token, który jest montowany w każdej kapsule i używany w celu uzyskiwania dostępu do API Kubernetes. Aby to wyłączyć, należy dodać ustawienie `automountServiceAccountToken: false` do konfiguracji konta usług. Na listingu 19.7 pokazaliśmy, jak to zrobić dla domyślnego konta usług. Czynność tę należy wykonać w każdej przestrzeni nazw.

Listing 19.7. service-account.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
automountServiceAccountToken: false
```

Konta usług są często pomijane przy konfigurowaniu zabezpieczeń kapsuł. Jednak umożliwiają one bezpośrednie używanie API Kubernetes i bez odpowiedniej kontroli RBAC mogą pozwolić hakerowi na włamanie. Dlatego należy wiedzieć, jak ograniczyć dostęp do API przez wprowadzenie prostej zmiany w sposobie posługiwania się tokenami kont usług.

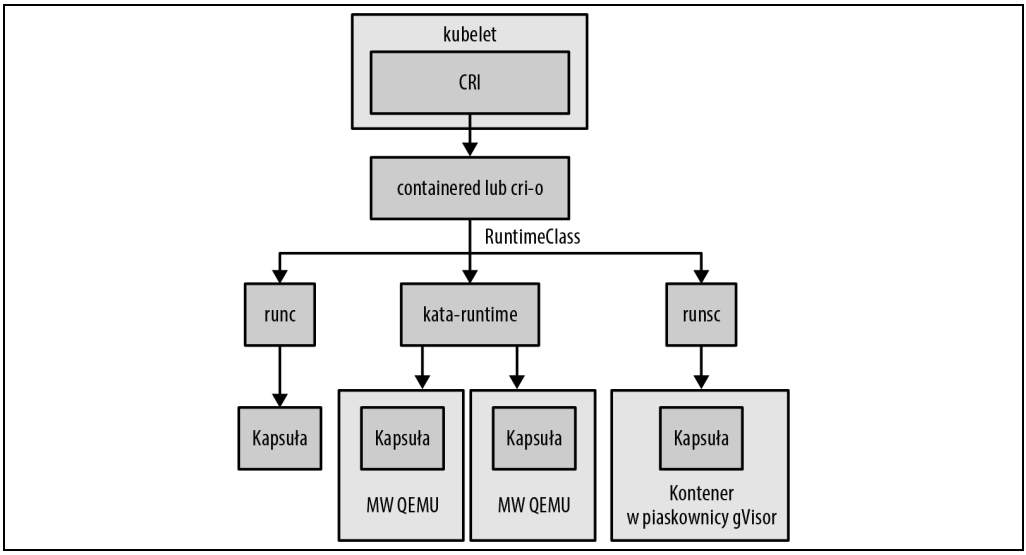
Kontrola dostępu oparta na rolach

W rozdziale na temat zabezpieczeń kapsuł nie możemy pominąć tematu kontroli opartej na rolach Kubernetes (RBAC). Wszystkie potrzebne informacje o RBAC znajdują się w rozdziale 14. i można je wykorzystać w celu uzupełnienia zabezpieczeń swojego obciążenia.

RuntimeClass

Kubernetes współpracuje ze środowiskiem wykonawczym kontenera w systemie operacyjnym węzła przez *Container Runtime Interface* (CRI). Utworzenie i standaryzacja tego interfejsu umożliwiły powstanie ekosystemu środowisk wykonawczych kontenerów. Środowiska te mogą różnić się poziomem izolacji, co daje pewniejsze gwarancje bezpieczeństwa w zależności od sposobu ich implementacji. Projekty takie jak Kata Containers, Firecracker czy gVisor bazują na różnych mechanizmach izolacji, od wirtualizacji zagnieżdżonej, po bardziej zaawansowane filtrowanie wywołań systemowych. Te gwarancje bezpieczeństwa i izolacji dają administratorowi Kubernetes elastyczność w zakresie zezwalania użytkownikom na wybór środowiska wykonawczego na podstawie typu obciążenia. Na przykład jeśli Twoje obciążenie potrzebuje silniejszych gwarancji bezpieczeństwa, możesz wybrać kapsułę używającą innego środowiska wykonawczego kontenerów.

API RuntimeClass wprowadzono, aby umożliwić wybór środowiska wykonawczego kontenerów. Pozwala on użytkownikowi wybrać jedno środowisko z listy środowisk obsługiwanych w klastrze. Na rysunku 19.1 przedstawiono schemat działania API RuntimeClass.



Rysunek 19.1. Diagram RuntimeClass

Różne klasy RuntimeClass muszą być skonfigurowane przez administratora klastra i mogą wymagać, aby obciążenie miało określone selektory `nodeSelector` lub tolerancje (`toleration`) zaplanowane dla odpowiedniego węzła.

Aby użyć interfejsu RuntimeClass, można zdefiniować `runtimeClassName` w specyfikacji kapsuły. Na listingu 19.8 jest pokazana przykładowa kapsuła określająca RuntimeClass.

Listing 19.8. kuard-pod-runtimeclass.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: kuard
    labels:
      app: kuard
spec:
    runtimeClassName: firecracker
    containers:
      - image: gcr.io/kuar-demo/kuard-amd64:blue
        name: kuard
        ports:
          - containerPort: 8080
            name: http
            protocol: TCP
```

Interfejs RuntimeClass umożliwia użytkownikom wybranie różnych środowisk wykonawczych kontenerów, które mogą mieć inne poziomy izolacji zabezpieczeń. Wykorzystanie interfejsu RuntimeClass może pomóc w uzupełnieniu zabezpieczeń obciążeń, zwłaszcza jeśli przetwarzają one wrażliwe informacje lub wykonują niezaufany kod.

Network Policy

Kubernetes ma także interfejs API Network Policy, który umożliwia tworzenie zarówno wejściowych, jak i wyjściowych zasad sieciowych dla obciążeń. Zasady te są konfigurowane przy użyciu etykiet, które pozwalają na wybór kapsuł oraz definiują dopuszczalny sposób komunikacji z innymi kapsułami i punktami końcowymi. Zasada bezpieczeństwa taka jak Ingress nie jest dostarczana z powiązanym kontrolerem Kubernetes. To znaczy, że można tworzyć zasoby Network Policy, ale jeśli nie zainstaluje się kontrolera włączającego się w reakcji na utworzenie tych zasobów, to nie zostaną one wyegzekwowane. Zasoby Network Policy są implementowane przez wtyczki sieciowe, takie jak Calico, Cilium i Weave Net.

Zasób Network Policy znajduje się w przestrzeni nazw oraz jest podzielony na sekcje `podSelector`, `policyTypes`, `ingress` oraz `egress`, z których wymagane jest tylko pole `podSelector`. Jeśli to pole nie zostanie zdefiniowane, polityka odpowiada wszystkim kapsułom w przestrzeni nazw. Może ono także zawierać sekcję `matchLabels`, która działa tak samo, jak zasób `Service`, umożliwiając dodanie zbioru etykiet odpowiadających określonemu zbiorowi kapsuł.

Z używaniem interfejsu Network Policy wiąże się kilka specyficznych kwestii, o których należy wiedzieć. Jeśli kapsule odpowiada jakikolwiek zasób Network Policy, to wszelka komunikacja wejściowa lub wyjściowa musi być wprost zdefiniowana albo zostanie zablokowana. Gdy kapsule

odpowiada kilka zasobów Network Policy, to zasady się sumują. Jeżeli kapsule nie odpowiada żaden zasób Network Policy, to ruch jest dozwolony. Takie rozwiązanie ma na celu ułatwienie wdrażania nowych obciążeń. Jeśli jednak chcesz domyślnie blokować wszelki ruch, utwórz domyślną regułę odmowy, którą można zastosować do przestrzeni nazw. Na listingu 19.9 pokazaliśmy przykład takiej domyślnej reguły.

Listing 19.9. networkpolicy-default-deny.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-ingress
spec:
    podSelector: {}
    policyTypes:
    - Ingress
```

Przeanalizujemy przykładowy zestaw zasad sieciowych, aby pokazać, jak za ich pomocą zabezpieczyć obciążenia. Najpierw utwórz przestrzeń nazw do testów, korzystając z następującego polecenia:

```
$ kubectl create ns kuard-networkpolicy
namespace/kuard-networkpolicy created
```

Przygotuj plik o nazwie *kuard-pod.yaml* o zawartości przedstawionej na listingu 19.10.

Listing 19.10. kuard-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: kuard
    labels:
        app: kuard
spec:
    containers:
      - image: gcr.io/kuar-demo/kuard-amd64:blue
        name: kuard
        ports:
          - containerPort: 8080
            name: http
            protocol: TCP
```

Utwórz kapsułę `kuard` w przestrzeni nazw `kuard-networkpolicy`:

```
$ kubectl apply -f kuard-pod.yaml \
  --namespace kuard-networkpolicy
pod/kuard created
```

Udostępnij kapsułę `kuard` jako usługę:

```
$ kubectl expose pod kuard --port=80 --target-port=8080 \
  --namespace kuard-networkpolicy
pod/kuard created
```

Teraz możesz użyć polecenia `kubectl run`, aby uruchomić kapsułę do przetestowania jako naszego zasobu oraz w celu sprawdzenia dostępu do kapsuły `kuard` bez stosowania jakiejkolwiek polityki Network Policy:

```
$ kubectl run test-source --rm -ti --image busybox /bin/sh \
  --namespace kuard-networkpolicy
If you don't see a command prompt, try pressing enter.
/ # wget -q kuard -O -
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">

  <title><KUAR Demo></title>
...
```

Udało Ci się nawiązać połączenie z kapsułą `kuard` z kapsuły testowej. Teraz zastosuj domyślną zasadę odmowy i ponownie wykonaj test. Utwórz plik o nazwie *networkpolicy-default-deny.yaml* o zawartości pokazanej na listingu 19.11.

Listing 19.11. networkpolicy-default-deny.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-ingress
spec:
    podSelector: {}
    policyTypes:
    - Ingress
```

Zastosuj domyślną sieciową zasadę odmowy:

```
$ kubectl apply -f networkpolicy-default-deny.yaml \
  --namespace kuard-networkpolicy
networkpolicy.networking.k8s.io/default-deny-ingress created
```

W dalszej kolejności przetestuj dostęp do kapsuły `kuard` z kapsuły `test-source`:

```
$ kubectl run test-source --rm -ti --image busybox /bin/sh \
  --namespace kuard-networkpolicy
If you don't see a command prompt, try pressing enter.
/ # wget -q --timeout=5 kuard -O -
wget: download timed out
```

Domyślna zasada odmowy Network Policy spowodowała, że nie masz już dostępu do kapsuły `kuard` z kapsuły `test-source`. Utwórz plik o nazwie *networkpolicy-kuard-allow-test-source.yaml* o treści pokazanej na listingu 19.12.

Listing 19.12. networkpolicy-kuard-allow-test-source.yaml

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
    name: access-kuard
spec:
```

```
    podSelector:
        matchLabels:
            app: kuard
    ingress:
        - from:
            - podSelector:
                matchLabels:
                    run: test-source
```

Zastosuj zasadę Network Policy:

```
$ kubectl apply \
  -f code/chapter-security/networkpolicy-kuard-allow-test-source.yaml \
  --namespace kuard-networkpolicy
networkpolicy.networking.k8s.io/access-kuard created
```

Ponownie zweryfikuj, czy kapsuła `test-source` ma dostęp do kapsuły `kuard`:

```
$ kubectl run test-source --rm -ti --image busybox /bin/sh \
  --namespace kuard-networkpolicy
If you don't see a command prompt, try pressing enter.
/ # wget -q kuard -O -
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">

  <title><KUAR Demo></title>
...
```

Wyczyść przestrzeń nazw przez wykonanie następującego polecenia:

```
$ kubectl delete namespace kuard-networkpolicy
namespace "kuard-networkpolicy" deleted
```

Zastosowanie zasady Network Policy powoduje dodanie kolejnej warstwy zabezpieczeń obciążeń oraz stanowi realizację reguł dogłębnej obrony i minimalnych uprawnień.

Siatka usług

Poziom bezpieczeństwa obciążenia można zwiększyć także za pomocą siatki usług. Siatki takie udostępniają zasady dostępu, które umożliwiają konfigurację rozpoznających protokół reguł opartych na usługach. Na przykład nasza zasada dostępu może deklarować, że usługa A łączy się z usługą B przez port HTTPS 443. Ponadto siatki usług zazwyczaj implementują wzajemny TLS w całej komunikacji między usługami, dzięki czemu jest ona szyfrowana oraz są weryfikowane tożsamości usług. Jeśli chcesz dowiedzieć się więcej na temat siatek usług oraz ich zastosowania w zabezpieczeniach obciążeń, przeczytaj rozdział 15.

Narzędzia do weryfikacji zabezpieczeń

Istnieje kilka narzędzi open source, które umożliwiają wykonywanie pakietów testów zabezpieczeń klastra Kubernetes, aby sprawdzić, czy konfiguracja spełnia określony zestaw wymagań. Jedno

z nich nazywa się kube-bench (*https://oreil.ly/TnUlm*). Za jego pomocą można przeprowadzać testy CIS (*https://oreil.ly/VvUe5*) dla Kubernetes. Narzędzia takie jak kube-bench wykonujące testy CIS nie są specjalnie nastawione na bezpieczeństwo kapsuł, ale potrafią wykryć błędy konfiguracji klastra oraz pomagają w rozwiązywaniu problemów. Narzędzie kube-bench można uruchomić za pomocą poniższego polecenia:

```
$ kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-bench...
job.batch/kube-bench created
```

Następnie można przejrzeć wyniki testu i zalecane rozwiązania w dziennikach kapsuły:

```
$ kubectl logs job/kube-bench
[INFO] 4 Worker Node Security Configuration
[INFO] 4.1 Worker Node Configuration Files
[PASS] 4.1.1 Ensure that the kubelet service file permissions are set to 644...
[PASS] 4.1.2 Ensure that the kubelet service file ownership is set to root ...
[PASS] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to ...
[PASS] 4.1.4 Ensure that the proxy kubeconfig file ownership is set to root ...
[PASS] 4.1.5 Ensure that the --kubeconfig kubelet.conf file permissions are ...
[PASS] 4.1.6 Ensure that the --kubeconfig kubelet.conf file ownership is set...
[PASS] 4.1.7 Ensure that the certificate authorities file permissions are ...
[PASS] 4.1.8 Ensure that the client certificate authorities file ownership ...
[PASS] 4.1.9 Ensure that the kubelet --config configuration file has permiss...
[PASS] 4.1.10 Ensure that the kubelet --config configuration file ownership ...
[INFO] 4.2 Kubelet
[PASS] 4.2.1 Ensure that the anonymous-auth argument is set to false (Automated)
[PASS] 4.2.2 Ensure that the --authorization-mode argument is not set to ...
[PASS] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate...
[PASS] 4.2.4 Ensure that the --read-only-port argument is set to 0 (Manual)
[PASS] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is...
[FAIL] 4.2.6 Ensure that the --protect-kernel-defaults argument is set to ...
[PASS] 4.2.7 Ensure that the --make-iptables-util-chains argument is set to ...
[PASS] 4.2.8 Ensure that the --hostname-override argument is not set (Manual)
[WARN] 4.2.9 Ensure that the --event-qps argument is set to 0 or a level ...
[WARN] 4.2.10 Ensure that the --tls-cert-file and --tls-private-key-file arg...
[PASS] 4.2.11 Ensure that the --rotate-certificates argument is not set to ...
[PASS] 4.2.12 Verify that the RotateKubeletServerCertificate argument is set...
[WARN] 4.2.13 Ensure that the Kubelet only makes use of Strong Cryptographic...

== Remediations node ==
4.2.6 If using a Kubelet config file, edit the file to set protectKernel...
If using command line arguments, edit the kubelet service file
/etc/systemd/system/kubelet.service.d/10-kubeadm.conf on each worker node and
set the below parameter in KUBELET_SYSTEM_PODS_ARGS variable.
--protect-kernel-defaults=true
Based on your system, restart the kubelet service. For example:
systemctl daemon-reload
systemctl restart kubelet.service

4.2.9 If using a Kubelet config file, edit the file to set eventRecordQPS...
If using command line arguments, edit the kubelet service file
/etc/systemd/system/kubelet.service.d/10-kubeadm.conf on each worker node and
set the below parameter in KUBELET_SYSTEM_PODS_ARGS variable.
Based on your system, restart the kubelet service. For example:
systemctl daemon-reload
systemctl restart kubelet.service
...
```

Za pomocą takich narzędzi jak `kube-bench` wykonujących testy CIS można sprawdzić, czy klaster Kubernetes spełnia wymogi bezpieczeństwa. Ponadto dostarczają one podpowiedzi, jak rozwiązać ewentualne problemy.

Bezpieczeństwo obrazów

Innym ważnym aspektem związanym z bezpieczeństwem kapsuł jest zapewnienie bezpieczeństwa kodu i aplikacji w kapsule. Zabezpieczanie kodu aplikacji to skomplikowany temat, którego omówienie wykracza poza ramy tego rozdziału. Podstawowe zasady bezpieczeństwa kontenera obejmują **skanowanie statyczne** rejestru obrazu pod kątem znanych luk w kodzie. Ponadto należy używać narzędzia **skanującego w trakcie działania** obrazu, które będzie znajdowało luki odkryte po jego uruchomieniu oraz będzie szukało potencjalnie szkodliwej działalności, na przykład włamań. Istnieje wiele narzędzi do skanowania, zarówno typu open source, jak i opracowanych przez różne firmy. Ponadto należy minimalizować zawartość obrazu kontenera, aby pozbyć się niepotrzebnych zależności i zredukować szum podczas skanowania. W końcu dbanie o bezpieczeństwo obrazów to kolejny dobry powód, aby zainwestować w ciągłe dostarczanie, które umożliwia szybkie wprowadzanie poprawek i wdrażanie obrazów po wykryciu luk bezpieczeństwa.

Podsumowanie

W tym rozdziale opisaliśmy wiele różnych API i zasobów zabezpieczeń, za pomocą których można zwiększyć poziom bezpieczeństwa obciążeń. Przestrzegając zasad dogłębnej obrony i minimalnych uprawnień, można stopniowo poprawiać bezpieczeństwo klastra Kubernetes. Nigdy nie jest za późno, aby zacząć stosować lepsze zabezpieczenia, a w tym rozdziale znajdują się wszystkie informacje potrzebne do tego, by zrozumieć środki bezpieczeństwa dostępne w Kubernetes.

Poznaj przyszłość infrastruktury!

Oto podstawowe źródło wiedzy na temat systemu Kubernetes, okraszone przykładami, które umożliwiają samodzielne poznawanie tego produktu!

Liz Rice

Kubernetes jest ważnym narzędziem do tworzenia, wdrażania i utrzymywania aplikacji w chmurze. Obecnie pozwala na uzyskiwanie prędkości, zwinności, niezawodności i wydajności na wysokim poziomie, a samo tworzenie i utrzymywanie systemów rozproszonych jest proste, efektywne i satysfakcjonujące. Trzeba tylko zrozumieć, na czym polega abstrakcja kontenerów i interfejsów API orkiestracji kontenerów i poświęcić trochę czasu na zapoznanie się z samym Kubernetesem.

To trzecie wydanie przewodnika autorstwa twórców Kubernetesa. Zostało starannie zaktualizowane i wzbogacone o tak ważne zagadnienia jak bezpieczeństwo, dostęp do Kubernetesa za pomocą kodu napisanego w różnych językach programowania czy tworzenie aplikacji wieloklastrowych. Dzięki książce poznasz podstawy funkcjonowania Kubernetesa, a naukę rozpoczniesz od budowy prostej aplikacji. Później dowiesz się, jak używać narzędzi i interfejsów API do automatyzacji skalowalnych systemów rozproszonych, w tym usług internetowych, aplikacji do uczenia maszynowego czy klastrów komputerów Raspberry Pi. Omówiono tu również zaawansowane zagadnienia, takie jak obiekty specjalne czy siatki usług i system kontroli źródła.

Najciekawsze zagadnienia:

- budowa i działanie Kubernetesa na podstawie prostego przykładu
- specjalne obiekty, takie jak DaemonSet, Job, ConfigMap i tajne dane
- cykl życia kompletnej aplikacji rozproszonej
- zabezpieczanie wdrażanych aplikacji
- aplikacje wieloklastrowe i dostęp do Kubernetesa za pomocą własnego kodu

Brendan Burns kieruje zespołami zajmującymi się technologiami DevOps, open source i mikrousługami w Microsoft Azure. Współtworzył projekt Kubernetes w Google.

Joe Beda jest jednym z twórców Kubernetesa. Jako dyrektor techniczny w VMware wdraża strategię rozwoju tego projektu.

Kelsey Hightower jest doświadczonym liderem zespołów, pracuje dla Google Cloud Platform.

Lachlan Evenson jest głównym menedżerem produktu w zespole obliczeń kontenerowych Microsoft Azure. Tworzy praktyczne poradniki dotyczące Kubernetesa.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-8322-336-0
9 788383 223360
Cena: 69,00 zł