

CKS CheatSheet

Abdessamad EL AMRANI
Dec 2023

Domain1 : Exam details and general notes	4
Domain2: Cluster Setup	4
CIS Benchmark	4
CIS-CAT tool.....	5
Kube-Bench	5
Network Policy	5
Kube Bench	6
Ingress with TLS	6
Protecting Node Metadata (cloud)	8
Service Account	9
Service Account Token Mounting restricting	10
Protect Dashboard	10
RBAC	11
Detect Modified Binary	11
Domain3: Cluster Hardening	11
Admission control and Request validation	12
Upgrade Procedure	12
Domain4: System Hardening	12
Minimize Host Footprint	13
Minimize IAM Roles	13
Users Management in Linux.....	13
Group Management in Linux.....	14
Firewall Rules in Linux	14
Strace	15
AquaSec Tracee (the strace for containers)	15
SecComp (Limit system calls)	16
AppArmor (Limit system resources access: files, network,..)	17
General Linux AppArmor.....	18
Kubernetes AppArmor.....	19
Linux Capabilities	19
Domain5: Minimizing Microservices Vulnerabilities	20
Intro	20
Security Context	21
PSA (PoD Security Admission)	21
Admission Control	21
Admission Control 1 – DEFAULT ONES EASY to Use--.....	21
Admission Control 2 – Use Own Admission Control server –	22
Admission Control 3 – Default Ones but require config and external webhook --.....	23
Admission Control 4 – Use k8s Ready OPA gatekeeper with rego –	24
Managing Secrets	27
gVisor runtime sandboxing	28

Install gVisor	28
Use gVisor	29
Kata containers.....	30
RunTime Classes	30
Pod to Pod Encryption with mTLS	31
<i>Domain 6. Supply Chain Security.....</i>	<i>31</i>
Minimizing the Base image footprint	31
Picking a Base image small in size.....	31
Using a Multi-Stage Approach for Building Container Images	32
Reducing number of Layers.....	32
Using container image optimization tools	32
Using a private docker repo and image	32
Security the supply chain.....	33
Sign and validate image:.....	33
Enforce image with OPAGatekeeper.....	33
Enforce image with Admission Controller	34
Static analysis of user workload.....	34
Hadolint	34
Kubesec.....	34
Scanning images of known vulnerabilities	34
Trivy	34
<i>Domain 7. Monitoring, Logging, and Runtime Security</i>	<i>34</i>
Falco.....	34
Immutability of contains at runtime.....	35
Using Audit logs to monitor access	35

Domain1 : Exam details and general notes

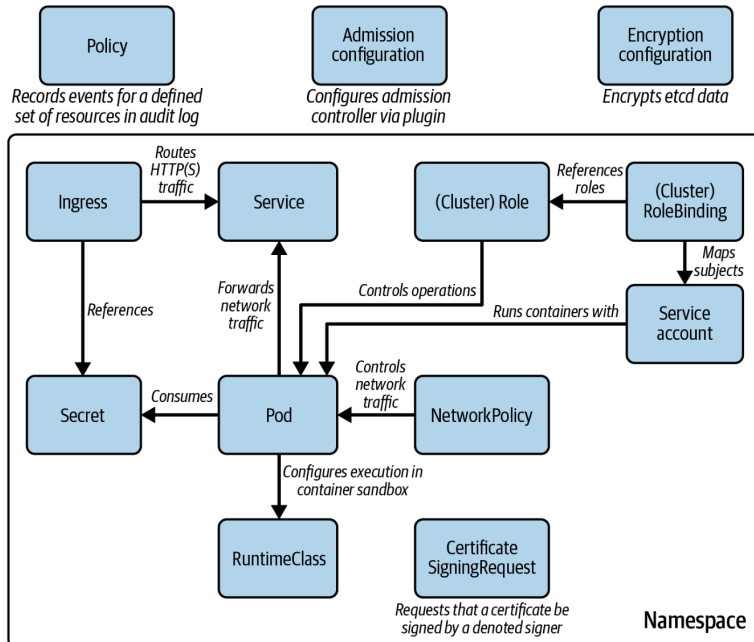


Figure 1-2. Kubernetes primitives relevant to the exam

Domain2: Cluster Setup

CIS Benchmark

§

CIS website provides benchmark for 25+ vendors, for linux, kubernetes, vmware, docker,... ,windows, mac os, AWS, GCP, .. iOS,Android,cisco,juniper,Paloalto,..

CIS-CAT tool

How to run CIS tool in interactive mode (most important thing to remember	<code>./Assessor-CLI.sh -i</code>
--	-----------------------------------

Kube-Bench

From Aqua security, only for Kubernetes, we will see it after..

Network Policy

If Pod restart, will it keep IP ?	No, IP @ of Pod is ephemeral
Whats best way to communicate PoD to PoD	Use Pod TO Service

- The IP address of a Pod is ephemeral. Every restart of a Pod leases a new IP address.
- It's recommended to use Pod-to-Service communication over Pod-to-Pod

NetworkPolicy logic:

- + if empty, its allowing allow
- + if namespace mentioned under metadata, then policy applies to only matched ressources in namespace.
- + the direction mentioned under **policyType**, will change to deny all

Examples:

- + if you mention **- Ingress** element under it, then all Ingress is blocked.
- >> Unless something explicitly allowed under potential **ingress** section in spec.

- + if you mention **- Egress** element under it, then all Egress is blocked.
- >> Unless something explicitly allowed under potential **egress** section in spec.

- + if you mention **- Ingress** and **- Egress** elements under it, then both Ingress and Egress blocked.
- >> Unless something explicitly allowed under potential **ingress** or **egress** section in spec.

+ if you have multiple rules , each rule with '- ' , then that's OR, (like firewall policy), if traffic doesn't match rule, k8s tries the next rule

+ if you have several entries without '- ' , then that's an AND, (like entries of same firewall policy : src ip, dst,..)

Deny all ingress network policy on namespace In this particular example, we disallow all ingress traffic to PoDs in namespace g04. Egress traffic is still permitted.	<pre>apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: default-deny-ingress namespace: g04 spec: podSelector: {} policyTypes: - Ingress</pre>
Deny both ingress and egress	<pre>apiVersion: networking.k8s.io/v1 kind: NetworkPolicy</pre>

§

	<pre> metadata: name: allow-ingress-deny-egress namespace: g04 spec: podSelector: {} policyTypes: - Ingress - Egress </pre>
Deny egress but allow ingress	<pre> apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: allow-ingress-deny-egress namespace: g04 spec: podSelector: {} policyTypes: - Ingress - Egress ingress: - {} </pre>

Kube Bench

Step1 Install kubebench as pod	<pre> \$ kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job-master.yaml job.batch/kube-bench-master created \$ kubectl get pods NAME READY STATUS RESTARTS AGE kube-bench-master-8f6qh 0/1 Completed 0 45s </pre>
Step2 Wait until the Pod transitions into the “Completed” status to ensure that all verification checks have finished. You can have a look at the benchmark result by dumping the logs of the Pod:	<pre> \$ kubectl logs kube-bench-master-8f6qh </pre>
To delete or run a new kube bench analysis after fixing something :	<pre> \$ kubectl delete job kube-bench-master job.batch "kube-bench-master" deleted Then run 1),2),3) again </pre>

Ingress with TLS

STEP 1 : Creating the TLS Certificate and Key	<pre> \$ openssl req -nodes -new -x509 -keyout accounting.key -out accounting.crt -subj "/CN=accounting.tls" Generating a 2048 bit RSA private key++ </pre>
---	--

§

	<pre>writing new private key to 'accounting.key' ----- \$ ls accounting.crt accounting.key</pre>
<u>STEP 2 : Creating the TLS-Typed Secret</u> The secret yaml will look like this :	<pre>\$ kubectl create secret tls accounting-secret -- cert=accounting.crt --key=accounting.key -n t75 secret/accounting-secret created apiVersion: v1 kind: Secret metadata: name: accounting-secret namespace: t75 type: kubernetes.io/tls data: tls.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk... tls.key: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk...</pre>
<u>STEP 3 : Creating the Ingress</u>	<pre>\$ kubectl create ingress accounting-ingress \ --rule="accounting.internal.acme.com/*=accounting- service:80, \ tls=accounting-secret" -n t75 ingress.networking.k8s.io/accounting-ingress created</pre>

Create / Approve CSR in K8S

Create CSR	<pre>Openssl genrsa -out jane.key 2048 Openssl req -new -key user.key -subj "/CN=user" -out user.csr</pre>
Create K8S CSR	<pre>cat user.csr base64</pre> then create a CertificateSigningRequest , and put the base64 encoding inside : <pre>apiVersion: certificates.k8s.io/v1beta1 kind: CertificateSigningRequest metadata: name: jane spec: groups: - system:authenticated usages: - digital signature - key encipherment - server auth</pre>
Check it / approve it	<pre>Kubectrl get csr Kubectrl certificate approve user</pre>

Kubelet Security

Ports that kubelet serves on	10250 : serves full access 10255 : services ready only un-authenticated access
+ How to disable full anonymous access ? + How to configure certificate authentication ? + How to configure authorization to use kube-api? + How to disallow readonly serving port (10255) or other?	kubelet.service <pre>ExecStart=/usr/local/bin/kubelet \\ --anonymous-auth=false \\ --client-ca-file=/path/to/ca.crt \\ --authorization-mode=Webhook --read-only-port=0</pre> kubelet-config.yaml <pre>apiVersion: kubelet.config.k8s.io/v1beta1 kind: KubeletConfiguration authentication: anonymous: enabled: false x509: clientCAFile: /path/to/ca.crt authorization: mode: Webhook readOnlyPort: 0</pre>

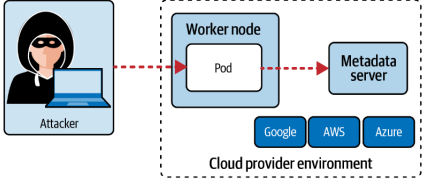
Kubect proxy

How to send api to kube-api server without hassl of adding certs and keys in curl ?	# kubect proxy Starting to serve on 127.0.0.1:8001 # curl http://localhost:8001 -k <pre>{ "paths": ["/api", "/api/v1", "/apis", "/apis/*", "/healthz", "/logs", "/metrics", "/openapi/v2", "/swagger-2.0.0.json"] }</pre>
How to access service in remote k8s with kubect proxy	<pre>curl http://localhost:8001/api/v1/namespaces/default/services/nginx/proxy</pre>

Protecting Node Metadata (cloud)

Control Plane ports	Port range	Purpose
	6643	Kubernetes API server

§

	2379–2380 etcd server client API 10250 Kubelet API 10259 kube-scheduler 10257 kube-controller-manager
Worker Node ports	Port range Purpose 10250 Kubelet API 30000–32767 NodePort Services
Change those ports in kubeapi server	--secure-port ..
Protect metadata server in cloud (aws)	 <pre> apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: default-deny-egress-metadata-server namespace: a12 spec: podSelector: {} policyTypes: - Egress egress: - to: - ipBlock: cidr: 0.0.0.0/0 except: - 169.254.169.254/32 </pre> Figure 2-3. An attacker who gained access to the Pod has access to metadata server

Service Account

How to create token of a service account ?	2 way to create token : New way : <pre>\$ kubectl create token serviceaccountXx eyJhbGciOiJSUzI1NiIsImtpZCI6...</pre> Old way : Create a secret with yaml, with this type : (AFTER CREATING THE SERVICE ACCOUNT) <pre> apiVersion: v1 kind: Secret type: kubernetes.io/service-account-token metadata: name: mysecretname annotations: kubernetes.io/service-account.name: dashboard-sa </pre> Very Old way (before 1.22) Just create service account, token assigned directly.
How to get token of existing token	<pre>k describe secrets -n kubernetes-dashboard admin-user</pre> (if you used get yaml, you need to base64 decode!) <pre>>>>> this only if you used a ticket (old way), otherwise with new way, seems only way is at beginning when created, or to login to POD and dump the the token in /var/run/secrets/kubernetes/token...</pre>

§

Bonus : For this question, the rolebinding itself need to be in kubernetes-dashboard namespace , not default ! Use <code>admin</code> ClusterRole with RoleBinding so that it gives full control over every resource in the role binding's namespace, including the namespace itself.	<pre># k create rolebinding xx --namespace=yyy ...</pre>
--	--

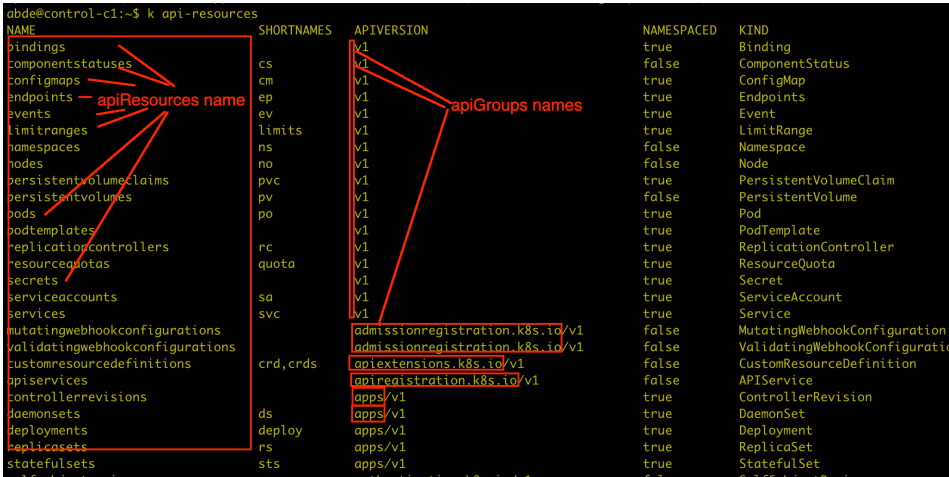
Service Account Token Mounting restricting

When assigning serviceaccount to PoD, the token is automatically mounted in <code>/var/run/secrets/kubernetes.io/serviceaccount/token</code> to disable the automount behavior for a service account :	<pre>apiVersion: v1 kind: ServiceAccount metadata: name: sa-api namespace: k97 automountServiceAccountToken: false</pre> docs tip keywords: you have to remember full <code>'automountServiceAccountToken'</code>
--	---

Protect Dashboard

1) Install k8s dashboard	Get the command (yaml) to apply from kubernetes documentation
2) Option1: create SA & RBAC using <code>cluster-admin</code> role	<pre>apiVersion: v1 kind: ServiceAccount metadata: name: admin-user namespace: kubernetes-dashboard apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: admin-user roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: cluster-admin // default one subjects: - kind: ServiceAccount name: admin-user namespace: kubernetes-dashboard</pre>
2) Option2: create SA & RBAC with custom cluster-role	<pre>apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: annotations: rbac.authorization.kubernetes.io/autoupdate: "true" name: cluster-developer rules: - apiGroups: - '*' resources: - '*' verbs: - get - list - watch - nonResourceURLs: - '*' verbs: - get - list - watch</pre>

§

3) Create the token of the admin or non admin user	<pre>\$ kubectl create token admin-user -n kubernetes-dashboard eyJhbGciOiJIUzI1NiIsImtpZCI6...</pre>
4) Access k8s dashboard,	<pre># kubectl proxy</pre> Then browse to : http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/.
Bonus Question: How to know which apiGroup and apiResource you need to mention in custom clusterrole ?	<pre>>> check default clusterroles as inspiration (k get clusterrole)</pre> <pre>>> k api-resources</pre>  NAME >>> gives you the api resource name APIVERSION >> gives you the api Group, by just removing the v1 at the end. API

RBAC

User in k8s ?	Defined (not created) in kubeconfig , not created in linux fashion..
Inspect permissions of user	<pre>Kubectl auth can-i ... --as john</pre> <pre>Kubectl get pods --as john // simpler ?</pre>

Detect Modified Binary

Detect modified binary	Linux: <pre>echo "\$(cat kubectl.sha256) kubectl" sha256sum --check</pre>
------------------------	---

Domain3: Cluster Hardening

Admission control and Request validation

Configure KubeAPI Server authentication with user/password file : <pre>password1,user1,u0001 password2,user2,u0002 password3,user3,u0003</pre>	--basic-auth-file=user-details.csv (don't forget mount volume where this file is !) To use it in client : <code>curl -v -k https://node:6443/api/v1/pods -u "user1:password1"</code>
Configure KubeAPI Server authentication with token file: <pre>kpoajsdflasjkdfklsjdf,user1,u0001 asdfaoajsdflasjkdfklsjd,user2,u0002 aasdfasdfasdfasdfasdf,user3,u0003</pre>	--token-auth-file=user-token-details.csv (don't forget mount volume where this file is !) To use it in client : <code>curl -v -k https://node:6443/api/v1/pods -header "Authorization: Bearer KpjasdlfkjaAALasdfkdsfp"</code>

Upgrade Procedure

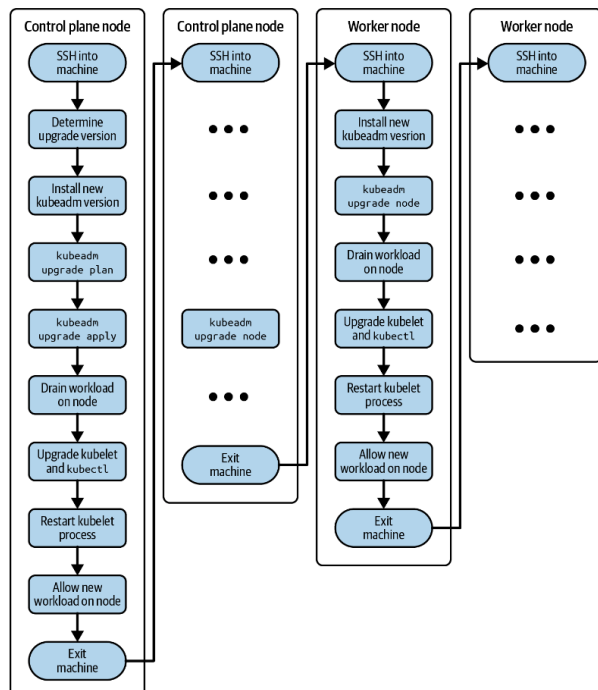


Figure 3-4. Process for a cluster version upgrade

Domain4: System Hardening

§ Minimize Host Footprint

List all services :	# systemctl grep running
Disable service upon system restart	# systemctl disable snapd
List only active services	# systemctl list-units --type service
Remove a package:	# apt purge --auto-remove snapd
List installed packages	# apt list -installed
List all available packets in repos	# apt list (you can also grep)
Show detail of package before installing	# apt show tree <pre> abde@control-c1:/\$ apt show tree Package: tree Version: 1.8.0-1 Priority: optional Section: utils Maintainer: Florian Ernst <florian@debian.org> Installed-Size: 117 kB Depends: libc6 (>= 2.4) Homepage: http://mama.indstate.edu/users/ice/tree/ Tag: implemented-in::c, interface::commandline, role::program, scope::utility, use::browsing, works-with::file Download-Size: 49.3 kB APT-Sources: http://deb.debian.org/debian buster/main amd64 Packages Description: displays an indented directory tree, in color Tree is a recursive directory listing command that produces a depth indented listing of files, which is colorized ala dircolors if the LS_COLORS environment variable is set and output is to tty. </pre>
List info about specific package	# apt policy vim
List details , dependencies,.. of a package	# apt-cache showpkg vim
Upgrade a package	# apt upgrade wget
List all kernel modules	# lsmod
Black list a kernel module	vim /etc/modprobe.d/blacklist.conf
Remove a service	Check 2 places ! /etc/systemd/system/.. (user/local installed services) /lib/systemd/system/.. (packages/system installed services)

Minimize IAM Roles

Users Management in Linux

List users, format : [username:password:UID:GID:comment:home:shell]	# cat /etc/passwd <pre> root:x:0:0:root:/root:/bin/bash nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin ... </pre>
If shell value is : /usr/sbin/nologin , user can't login.	
Switch to other user without inheriting env vars	# su - Ben

§

Run command as user ben	# sudo -u Ben pwd
Delete user including home directory	# sudo userdel -r ben

Group Management in Linux

it's more convenient for a system administrator to group users with similar access requirements to control permissions on an individual user level

List groups, format is : [groupname:passwd:GID: user_list] few notes: + notice groups might be empty, only plugdev group has a member user called packer + User_list separated By comma. (see example of analyticusers group) + The reason /etc/passwd shows only one group for each user is that this file is designed to specify the user's primary group. The primary group is often the group that is used when the user creates new files or directories	<pre># cat /etc/group root:x:0: plugdev:x:46:packer nogroup:x:65534: analyticusers:*:250: _analyticsd, _networkd, _timed ...</pre>
---	--

Firewall Rules in Linux

Enabling the firewall and setting up deny rules for any incoming and outgoing network traffic	<pre># ufw allow ssh # ufw default deny outgoing Default outgoing policy changed to deny # ufw default deny incoming Default incoming policy changed to deny (be sure to update your rules accordingly) # ufw enable Command may disrupt existing ssh connections. Proceed with operation (y n)? y Firewall is active and enabled on system startup</pre>
Allow access to kubeapi	# ufw allow 6443
Allow 1000 to 2000 tcp	# ufw allow 1000:2000/tcp
Allow ssh incoming	# ufw allow 22/tcp
Allow port 9090 from 135.22.65.0/24 to all interfaces	# ufw allow from 135.22.65.0/24 to any port 9090 proto tcp
Display rules by index	<pre># ufw status numbered # ufw status <u>verbose</u></pre>

§

Delete policy	#ufw delete deny 8080
Delete policy by index	# ufw status numbered (if you forget numbered keyword, just count lines of rules) # ufw delete 5
When you configure ufw before to enable it, how to check the rules	In that case, ufw status will not help because ufw still disable, To check the rules, look at /etc/ufw/user.rules

Strace

Get all sys calls of command 'ls'	# strace ls 100.00 0.000000 94 10 total
Trace calls, and output a summary of all calls and number of times they got called	<pre># strace -c ls % time seconds usecs/call calls errors syscall ----- 0.00 0.000000 0 7 0 read 0.00 0.000000 0 2 0 write 0.00 0.000000 0 11 0 close 0.00 0.000000 0 10 0 fstat 0.00 0.000000 0 17 0 mmap 0.00 0.000000 0 12 0 mprotect 0.00 0.000000 0 1 0 munmap 0.00 0.000000 0 3 0 brk 0.00 0.000000 0 2 0 rt_sigaction 0.00 0.000000 0 1 0 rt_sigprocmask 0.00 0.000000 0 2 0 ioctl 0.00 0.000000 0 8 0 8 access 0.00 0.000000 0 1 0 execve 0.00 0.000000 0 2 0 getdents 0.00 0.000000 0 2 0 2 statfs 0.00 0.000000 0 1 0 arch_prctl 0.00 0.000000 0 1 0 set_tid_address 0.00 0.000000 0 9 0 openat 0.00 0.000000 0 1 0 set_robust_list 0.00 0.000000 0 1 0 prlimit64 ----- 100.00 0.000000 94 10 total</pre>

AquaSec Tracee (the strace for containers)

How it works	For tracee docker to work, it mount following things form host : <table> <tr> <th>Things tracee docker needs to access on host</th><th>Bind mounts</th></tr> <tr> <td>Default workspace</td><td>/tmp/tracee</td></tr> <tr> <td>Kernel headers in host</td><td>/lib/modules</td></tr> <tr> <td>Kernel headers</td><td>/usr/src</td></tr> <tr> <td></td><td></td></tr> </table>	Things tracee docker needs to access on host	Bind mounts	Default workspace	/tmp/tracee	Kernel headers in host	/lib/modules	Kernel headers	/usr/src		
Things tracee docker needs to access on host	Bind mounts										
Default workspace	/tmp/tracee										
Kernel headers in host	/lib/modules										
Kernel headers	/usr/src										
Run tracee as docker, and trace system calls invoked by ls command	<pre>docker run --name tracee --rm --privileged --pid=host \ -v /lib/modules:/lib/modules:ro -v /usr/src:/usr/src:ro \ -v /tmp/tracee:/tmp/tracee aquasec/tracee:0.4.0 tracee</pre>										
Use tracee to trace any new container syscalls	<pre>sudo docker run --name tracee --rm --privileged --pid=host \ -v /lib/modules:/lib/modules:ro -v /usr/src:/usr/src:ro \ -v /tmp/tracee:/tmp/tracee aquasec/tracee:0.4.0 --trace</pre>										
In this case all new containers	container=new										

§

Run Tracee as container in K8S	Find a deployment in internet using image aquasec/tracee:0.4.0
Check results	K logs -f (strace pod)

SecComp (Limit system calls)

Check if Kernel supports SecComp	<code>grep -i seccomp /boot/config-\$(uname -r)</code>
Check if a process is controlled by SecComp Mode 0 : Disabled Mode 1 : Strict Mode 2 : Filtered	<code>grep Seccomp /proc/1/status</code>
Runtimes (docker/containerd..) ship with a default seccomp profile , allowing common syscalls required, and blocking dangerous ones. But k8s doesn't apply it by default, to do so either : + Enable SeccompDefault feature gate (check k8s doc) + Use seccompProfile in PoD <i>In this example, 'echo' related syscalls were allowed by default runtime seccomp profile</i>	<pre>apiVersion: v1 kind: Pod metadata: name: hello-seccomp spec: securityContext: seccompProfile: type: RuntimeDefault containers: - name: hello image: busybox:1.28 command: ["sh", "-c", "echo 'Hello seccomp!' && sleep 1h"]</pre> # kubectl logs hello-seccomp Hello seccomp!
custom seccomp profile : 1) Create directory : 2) Write seccomp file (ex: to prevent mkdir syscalls) ❶ The default action applies to all system calls. Here we'll allow all syscalls using SCMP_ACT_ALLOW. ❷ You can filter for specific architectures the default action should apply to. The definition of the field is optional. ❸ The default action can be overwritten by declaring more fine-grained rules. The SCMP_ACT_ERRNO action will prevent the execution of the mkdir syscall.	/var/lib/kubelet/seccomp/profiles 🔍 seccomp var Search Results About 1,340 results (0.17 seconds) Restrict a Container's Syscalls with seccomp Kubernetes kubernetes.io > docs > tutorials > security > seccomp 31 Oct 2023 ... var/lib/kubelet/seccomp/profiles audit.json fine-grained.json violatic <pre>apiVersion: kind.x-k8s.io/v1alpha4 kind: Cluster nodes: - role: control-plane extraMounts: - hostPath: "/.profiles" containerPath: "/var/lib/kubelet/seccomp/profiles"</pre> #sudo mkdir -p /var/lib/kubelet/seccomp/profiles To find this path in k8s docs, its here :

§

3) Apply the seccomp profile to the container

we point the `seccompProfile` attribute of the security profile to the file `mkdir-violation.json` and set the type to `Localhost`.

```
apiVersion: kind.x-k8s.io/v1alpha4
kind: Cluster
nodes:
- role: control-plane
  extraMounts:
  - hostPath: "/.profiles"
    containerPath: "/var/lib/kubelet/seccomp/profiles"
```

```
{
  "defaultAction": "SCMP_ACT_ALLOW", ❶
  "architectures": [ ❷
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "names": [
        "mkdir"
      ],
      "action": "SCMP_ACT_ERRNO" ❸
    }
  ]
}
```

NOTE: This is a Blacklist kind of profile (Allow all and define what to black list, you can do other way around, with `SCMP_ACT_ERRNO`)

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-seccomp
spec:
  securityContext:
    seccompProfile:
      type: Localhost ❶
      localhostProfile: profiles/mkdir-violation.json ❷
  containers:
  - name: hello
    image: busybox:1.28
    command: ["sh", "-c", "echo 'Hello seccomp!' && sleep 1h"]
    securityContext:
      allowPrivilegeEscalation: false
```

TROUBLE

How to test seccomp is working properly

SHOOTING

Use this container in docker or k8s :
r.j3ss.co/amicontained

And check the logs, it will tell exact if seccomp enabled, how may syscalls blocked/allowed..

How to find syscalls name from id

`grep -w 35 /usr/include/asm/unistd_64.h`

AppArmor (Limit system resources access: files, network,..)

§

AppArmor (Application Armor) is a Linux security module that restricts programs' capabilities by defining per-program profiles. These profiles can limit what system resources, like files and network access, a program can use.

General Linux AppArmor

How to check if system has apparmor enabled Note: if using apparmor in k8s , ensure that all workers are enabled with apparmor	<pre>cat /sys/module/apparmor/parameters/enabled y</pre>
Checks the current status of AppArmor, profiles loaded, and processes confined. Enforce : enforces the rules, reports the violation, and writes it to the syslog. Complain : does not enforce the rules, but writes violations to the log. useful to discover calls made by a program Unconfined: nothing is done for that process.	<pre># sudo aa-status apparmor module is loaded. 31 profiles are loaded. 31 profiles are in enforce mode. /snap/snapd/15177/usr/lib/snapd/snap-confine ... 0 profiles are in complain mode. 14 processes have profiles defined. 14 processes are in enforce mode. /pause (11934) docker-default ... 0 processes are in complain mode. 0 processes are unconfined but have a profile defined.</pre>
Custom AppArmor restricting file write to confined process (1) The identifier after the profile keyword is the name of the profile. (2) Apply to file operations. (3) Deny all file writes.	<pre>#include <tunables/global> profile k8s-deny-write flags=(attach_disconnected) { (1) #include <abstractions/base> file, (2) deny /** w, (3) }</pre>
Load the custom profile to apparmor To load the profile in complain mode, use the -C option.	<pre># sudo apparmor_parser /etc/apparmor.d/k8s-deny-write</pre>
Additional apparmor utilities , This installs aa-genprof, aa-autodep, aa-complain, aa-enforce,...	<pre>(# apt-get install apparmor-utils)</pre>
Use aa-enforce or aa-complain	<pre># aa-enforce /path/to/profile (or just profile name in /etc/apparmor.d) # aa-complain /path/to/profile (or just profile name in /etc/apparmor.d)</pre>
Interactive way to generate a profile	<pre># aa-genprof /path/to/your/program</pre>

§

Automatically generate a profile based on binary analysis by aa-autodep	# aa-autodep /path/to/your/program
---	---

Kubernetes AppArmor

Prerequisites: + **runtime** need support apparmor
 + **worker node** that runs the PoD need to have apparmor installed

Apply AppArmor profile to a container The annotation key needs to use the key in the format <code>container.apparmor.security.beta.kubernetes.io/<container-name></code>. In our case, the container name is hello. The full key is <code>container.apparmor.security.beta.kubernetes.io/hello</code>. The value of the annotation follows the pattern <code>localhost/<profile-name></code>. The custom profile we want to use here is <code>k8s-deny-write</code>	<pre> apiVersion: v1 kind: Pod metadata: name: hello-apparmor annotations: container.apparmor.security.beta.kubernetes.io/hello: \ localhost/k8s-deny-write spec: containers: - name: hello image: busybox:1.28 command: ["sh", "-c", "echo 'Hello AppArmor!' && sleep 1h"] </pre> Annotations are highlighted in green in the original image with circled numbers 1 and 2. The container name 'hello' is circled with number 3.
--	---

Linux Capabilities

Look at this one, even if its run with unconfined seccomp , he didn't had permission to use a date syscall. What blocked it ? >>> linux capability (because date -s requires CAP_SYS_TIME in a containerized env, but by default docker runs with limited capabilities)	<pre> # docker run -it --rm --security-opt seccomp=unconfined docker/whalesay /bin/sh # date -s '19 APR 2012 22:00:00' date: cannot set date: Operation not permitted Thu Apr 19 22:00:00 UTC 2012 </pre>
Same here, the k8s by default runs a pod by default without seccomp (unlike docker), so its like unconfined, and even default is running the pod as root, groupid root, date syscalls still blocked, What blocked it ? >>> linux capability (because date -s requires CAP_SYS_TIME in a containerized env, but by default docker runs with limited capabilities)	<pre> # kubectl run --rm -it ubuntu-sleeper --image=ubuntu -- bash </pre> If you don't see a command prompt, try pressing enter. <pre> root@ubuntu-sleeper:~# date -s '19 APR 2012 22:00:00' date: cannot set date: Operation not permitted Thu Apr 19 22:00:00 UTC 2012 </pre>
How to find linux capability required by some binary	<pre> # getcap /usr/bin/ping </pre> This doesn't work all the time ! (remember 'date' case)
How to find linux capability required by some process	<pre> # getpcaps pidofprocess </pre>

§

Docker runs by default with limited list of linux capabilities , only 14 (and CAP_SYS_TIME not one of them)	<pre>// DefaultCapabilities returns a Linux capabilities list func DefaultCapabilities() []string { return []string{ "CAP_CHOWN", "CAP_DAC_OVERRIDE", "CAP_FSETID", "CAP_FOWNER", "CAP_MKNOD", "CAP_NET_RAW", "CAP_SETGID", "CAP_SETUID", "CAP_SETPCAP", "CAP_NET_BIND_SERVICE", "CAP_SYS_CHROOT", "CAP_KILL", "CAP_AUDIT_WRITE", // ... (the list may continue beyond what is visible in the image) } }</pre>
How to fix the date permission in pod	Add Capability <pre>apiVersion: v1 kind: Pod metadata: name: ubuntu-sleeper spec: containers: - name: ubuntu-sleeper image: ubuntu command: ["sleep", "1000"] securityContext: capabilities: add: ["SYS_TIME"]</pre>
CRITICAL : You can set ‘runAsUser’ at both PoD or Container level (Container overrides PoD level if both existent) HOWEVER , ‘capabilities’ can ONLY best set at CONTAINER level !	

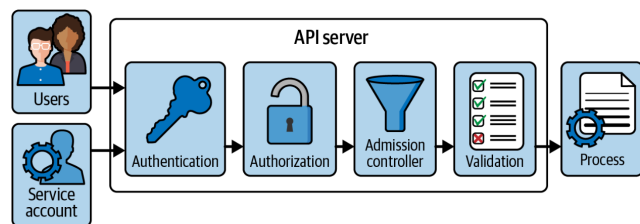
Domain5: Minimizing Microservices Vulnerabilities

Intro

this chapter mainly about Admission Controllers:

§ Processing a Request

Figure 3-1 illustrates the stages a request goes through when a call is made to the API server. For reference, you can find more information in the [Kubernetes documentation](#).



Security Context

By default, containers run with **root privileges**. But that's bad!

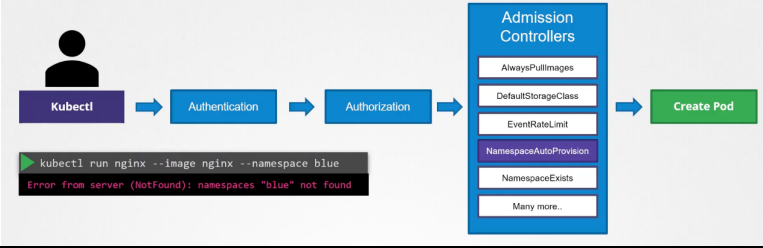
Run container as non root >> this pod will crash since nginx standard image requires root, you can use bitnami/nginx:1.23.1 instead	<pre>apiVersion: v1 kind: Pod metadata: name: non-root-success spec: containers: - image: nginx:1.23.1 name: nginx securityContext: runAsNonRoot: true</pre>
Setting specific user and groupid	<pre>apiVersion: v1 kind: Pod metadata: name: user-id spec: containers: - image: busybox:1.35.0 name: busybox command: ["sh", "-c", "sleep 1h"] securityContext: runAsUser: 1000 runAsGroup: 3000</pre>
Run Pod in privilege mode, which bypass All default seccomp, apparmor, capabilities,..	<pre>apiVersion: v1 kind: Pod metadata: name: privileged spec: containers: - image: busybox:1.35.0 name: busybox command: ["sh", "-c", "sleep 1h"] securityContext: privileged: true</pre>

PSA (PoD Security Admission)

PSA enabled by default in k8s>1.23, its actually an admission control, you will see it enabled in kube-apiserver admission control as ‘PodSecurity’ BUT not applied to PoD or Namespace To Apply to PSA to PoD or Namespace, do following :	<pre>metadata: labels: pod-security.kubernetes.io/{enforce}: {restricted} {audit }: {privileged} {warn }: {baseline}</pre> <table><tr><th>prefix</th><th>mode</th><th>level</th></tr></table>	prefix	mode	level
prefix	mode	level		

Admission Control

Admission Control 1 – Built-In controls direct to use--

Definition	Natif k8s feature that intercepts requests to kubeapi, after they are authenticated and authorized, but before they are persisted in the cluster datastore (etcd). 
Purpose	Enforce additional custom policies or modifications to objects during their creation, modification, deletion, or connection (as in the case of a pod connecting to a network).
Plugins	Admission control is implemented through a series of plugins, each with a specific purpose. These plugins can mutate (modify) or validate objects. Mutating plugins are called first, allowing them to modify objects; following this, validating plugins are called to ensure that the request still complies with the cluster's policy
Find in help all supported admission-control plugins	<pre># kubectl exec kube-apiserver-controlplane -n kube-system -- kube-apiserver -h grep enable-admission-plugins</pre> >>> Notice here we use exec to run the kube-apiserver binary, without it so we don't spawn a new shell, because shell is disabled in kube-apiserver.
How to enable a plugin	<pre>apiVersion: v1 kind: Pod metadata: creationTimestamp: null name: kube-apiserver namespace: kube-system spec: containers: - command: - kube-apiserver - --authorization-mode=Node,RBAC - --advertise-address=172.17.0.107 - --allow-privileged=true - --enable-bootstrap-token-auth=true - --enable-admission-plugins=NodeRestriction,NamespaceLifecycle image: k8s.gcr.io/kube-apiserver-amd64:v1.11.3 name: kube-apiserver</pre>
Type of admission control	1. Mutating admission control 2. Validating admission control
Mutating admission control examples	• DefaultStorageClass: Automatically assigns a default storage class to PersistentVolumeClaim objects if none is specified. • MutatingAdmissionWebhook: Calls an external webhook to mutate the object. • PodNodeSelector: Sets a default node selector if none is provided, based on a namespace annotation.
Validating admission control examples	• LimitRanger: Ensures that all incoming requests to create or modify resources meet the limits defined in the LimitRange object. • ValidatingAdmissionWebhook: Calls an external service to validate the object. • PodSecurityPolicy: Rejects the creation or modification of pods that do not meet the defined security policies.
	Admission

Admission Control 2 – Use Own Admission Control Webhook server –

§

External webhook admission control	1) Deploy Webhook Server Example of webhook server written in GO https://github.com/kubernetes/kubernetes/blob/v1.13.0/test/images/webhook/main.go
Configure it as webhook so API server can point to it :	2) Configure k8s to use it : apiVersion: admissionregistration.k8s.io/v1 kind: MutatingWebhookConfiguration OR ValidatingWebhookConfiguration webhooks: - name: my-webhook.example.com rules: - apiGroups: [""] apiVersions: ["v1"] operations: ["CREATE"] resources: ["pods"] scope: "Namespaced" clientConfig: url: https://my-webhook.example.com:9443/my-webhook-path In this case we point to external URL, but if webhook server is deployed locally in same k8s, and he has a service, we can access it via service as well, check k8s documentation for 'clientConfig' in that case.. Config reference: https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/ Keywords: MutatingWebhookConfiguration

>>> configuring the **MutatingWebhookConfiguration** OR **ValidatingWebhookConfiguration** is enough ! you don't need to do anything else in kube-api server, (no need for any admission control or thing to enable, nor mouting anything)

EXCEPT if authentication is required with the webhook, then you should use 'AdmissionConfiguration' as well (see next admission control 3)

Admission Control 3 – Built-In ones that requires config (and webhook in some cases) --

ImagePolicyWebhook is a good example

Its also interesting, because not only it needs modification , but also authentication

1) Create backend , it should be able to : + handle HTTPS requests. + accept and parse JSON request payloads. + send a JSON response payload	Example of backend : (not required to know for CKS) https://github.com/bmuschko/cks-study-guide/tree/master/ch06/image-validation-webhook for this example, it's the endpoint : https://localhost:8080/validate
2) Create a configuration file for the admission controller so it knows what plugins to use and how it should behave at runtime	<pre>apiVersion: apiserver.config.k8s.io/v1 kind: AdmissionConfiguration plugins:</pre>

	<pre> - name: imagePolicyWebhook ❶ configuration: imagePolicy: kubeConfigFile: /etc/kubernetes/admission-control x/imagepolicywebhook.kubeconfig ❷ allowTTL: 50 denyTTL: 50 retryBackoff: 500 defaultAllow: false ❸ </pre>
3) create image policy config file ❶ The URL to the backend service. Must use the HTTPS protocol.	<pre> apiVersion: v1 kind: Config preferences: {} clusters: - name: image-validation-webhook cluster: certificate-authority: /etc/kubernetes/admission-control/ca.crt server: https://image-validation-webhook:8080/validate ❶ contexts: - context: cluster: image-validation-webhook user: api-server-client name: image-validation-webhook current-context: image-validation-webhook users: - name: api-server-client user: client-certificate: /etc/kubernetes/admission-control/api-server-client.crt client-key: /etc/kubernetes/admission-control/api-server-client.key </pre>
4) configure kube api server to use webhook	<pre> ... spec: containers: - command: - kube-apiserver - --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook - --admission-control-config-file=/etc/kubernetes/admission-control/image-policy-webhook-admission-configuration.yaml ... volumeMounts: ... - name: admission-control mountPath: /etc/kubernetes/admission-control readOnly: true volumes: ... - name: admission-control hostPath: path: /etc/kubernetes/admission-control type: DirectoryOrCreate ... </pre>

Admission Control 4 – Use k8s Ready OPA gatekeeper webhook with rego –

OPA in general (linux)

OPA : open source language and engine(runtime), to define policies in flexible way, and exploit them.

Operates as a server/daemon , can be queried with restful apis,

Policies are written with Rego (OPA language), Rego works seamlessly with JSON

Usefull to enforce Authorization (RBAC,ABAC), Compliance, and Risk management

§

1) Install and run OPA	<pre>curl -L -o opa https://github.com/open-policy-agent/opa/releases/download/v0.11.0/opa_linux_amd64 chmod 755 ./opa ./opa run -s {"addr":["::]:8181"},"insecure_addr":"","level":"info","msg":"First line of log stream.","time":"2021-03-18T02:53:08Z"}</pre>
2) Create Rego policy : Every line inside allow, is a condition, and needs to be true.	<pre>package httpapi.authz # HTTP API request import input default allow = false allow { input.path == "home" input.user == "john" }</pre>
3) Load the policy to OPA	<pre>controlplane ~ → ls -lh -rw-r--r-- 1 root root 114 Mar 9 07:10 sample.rego curl -X PUT --data-binary @sample.rego http://localhost:8181/v1/policies/samplePolicy</pre> <pre>curl -X PUT --data-binary @sample.rego http://localhost:8181/v1/policies/samplepolicy</pre> ↓ Name of Rego file ↓ Given policy name in server
View policies loaded in OPA	<pre>curl http://localhost:8181/v1/policies</pre>
Example of using OPA as rbac service inside python code app:	<pre>@app.route('/home') def hello_world(): user = request.args.get("user") input_dict = { "input": { "user": user, "path": "home", } } rsp = requests.post("http://127.0.0.1:8181/v1/data/httpapi/authz", json=input_dict) if not rsp.json()["result"]["allow"]: return "Unauthorized", 401</pre>

§

```
return 'Welcome Home!', 200
```

<https://www.openpolicyagent.org/docs/latest/policy-language/>

play with rego policies: <https://play.openpolicyagent.org/>

K8S Gatekeeper

K8S Gatekeeper is extension to Kubernetes that uses OPA. Gatekeeper allows for defining and enforcing custom policies for any kind of Kubernetes API primitive.

With Gatekeeper for example, you can :

Ensuring that all Service objects need to define a label with the key `team`

Ensuring that all container images defined by Pods need to be pulled from a company-internal registry

Ensuring that Deployments need to control at least three replicas

Install Gatekeeper	<pre># kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml</pre>
Define ‘constraint template’, the constraint template defines the rules with rego and schema of constraint. This example , enforces that a namespace has to have at least one label . 1 Declares the kind to be used by the constraint. 2 Specifies the validation schema of the constraint. In this case, we allow to pass in a property named <code>labels</code> that captures the required label keys. 3 Uses Rego to check for the existence of labels and compares them to the list of required keys. <i>For instance after deploying this ConstraintTemplate and Constraint (See next), then deploying a namespace without label will be rejected..</i> <pre># kubectl create ns governed-ns Error from server (Forbidden): admission webhook "validation.gatekeeper.sh" \ denied the request: [ns-must-have-app-label-key] you must provide labels: {"app"}</pre>	<pre>apiVersion: templates.gatekeeper.sh/v1 kind: ConstraintTemplate metadata: name: k8srequiredlabels spec: crd: spec: names: kind: K8sRequiredLabels ① validation: openAPIV3Schema: ② type: object properties: labels: type: array items: type: string targets: - target: admission.k8s.gatekeeper.sh rego: ③ package k8srequiredlabels violation[{"msg": msg, "details": {"missing_labels": missing}}] { provided := {label input.review.object.metadata.labels[label]} required := {label label := input.parameters.labels[_]} missing := required - provided count(missing) > 0 msg := sprintf("you must provide labels: %v", [missing]) }</pre>
Define ‘constraint’ , is implementation of ‘constraint template’ , notice kind = name of constraint template. Plus it passes the data to it (in this case ‘namespace’ and ‘app’) 1 Uses the kind defined by the constraint template. 2 Defines the API resources the constraint template should apply to.	<pre>apiVersion: constraints.gatekeeper.sh/v1beta1 kind: K8sRequiredLabels ① metadata: name: ns-must-have-app-label-key spec: match: ② kinds: - apiGroups: [""] kinds: ["Namespace"] parameters: ③</pre>

§

③ Declares that the `labels` property expects the key `app` to exist.

sd

```
labels: ["app"]
```

ATTENTION ! :

IN match (2) , in this case we have **kinds** that tells that the type of object the constraint will be applied against, so in this case its namespace.

But if you want to apply the constraints to rather PoDs, you need to change ["Namespace"] to ["PoD"]

If you have other condition, like only config-map inside namespace xxx, then in that case you should have under match both :

```
match: ②
  kinds:
    - apiGroups: [""]
      kinds: ["configmaps"]
  namespaces: ["xxx"]
```

all other options kinds, namespaces,.. are here

The match field

The `match` field defines which resources the constraint will be applied to. It supports the following types of matchers:

- `kinds` accepts a list of objects with `apiGroups` and `kinds` fields that list the groups/kinds of objects to which the constraint will apply. If multiple groups/kinds objects are specified, only one match is needed for the resource to be in scope.
- `scope` determines if cluster-scoped and/or namespace-scoped resources are matched. Accepts `*`, `Cluster`, or `Namespaced`. (defaults to `*`)
- `namespaces` is a list of namespace names. If defined, a constraint only applies to resources in a listed namespace. Namespaces also supports a prefix-based glob. For example, `namespaces: [kube-*)` matches both `kube-system` and `kube-public`.
- `excludedNamespaces` is a list of namespace names. If defined, a constraint only applies to resources not in a listed namespace. ExcludedNamespaces also supports a prefix-based glob. For example, `excludedNamespaces: [kube-*)` matches both `kube-system` and `kube-public`.
- `labelSelector` is the combination of two optional fields: `matchLabels` and `matchExpressions`. These two fields provide different methods of selecting or excluding k8s objects based on the label keys and values included in object metadata. All selection expressions are ANDed to determine if an object meets the cumulative requirements of the selector.
- `namespaceSelector` is a label selector against an object's containing namespace or the object

<https://open-policy-agent.github.io/gatekeeper/website/docs/howto/#the-match-field>

Managing Secrets

Issue :

k8s secrets are stored in plain text in `etcd` !
so access to host & `etcd` is dangerous.
You can read plain text secret with `etcdctl get` :

Solution is next

To encrypt data in ETCD

STEP1: Create a nonce

```
# sudo ETCDCTL_API=3 etcdctl \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
get /registry/secrets/default/app-config | hexdump -C
```

```
00000000 2f 72 65 67 69 73 74 72 79 2f 73 65 63 72 65 74
|/registry/secret|
00000010 73 2f 64 65 66 61 75 6c 74 2f 61 70 70 2d 63 6f
|s/default/app-co|
```

```
# head -c 32 /dev/urandom | base64
W68x1PT/VXcOSEZJvWeIvkGJnGfQNFpvZYfT9e+ZYuY=
```

STEP2: Create encryption config file

- ❶ Defines the API resource to be encrypted in etcd. We are only encrypting Secrets data here.
- ❷ The base64-encoded key assigned to an AES-CBC encryption provider.

STEP3: Refer to encryption config file in kubeapiserver

```

apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets ❶
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: W68x1PT/VXc0SEZJvWeIvkGJnGfQNFpvZYfT9e+ZYuY= ❷
            - identity: {}

# sudo vim /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
    - command:
        - kube-apiserver
        - --encryption-provider-config=/etc/kubernetes/enc/enc.yaml
      volumeMounts:
        - name: enc
          mountPath: /etc/kubernetes/enc
          readOnly: true
      volumes:
        - name: enc
          hostPath:
            path: /etc/kubernetes/enc
            type: DirectoryOrCreate

```

gVisor runtime sandboxing

Kata containers achieves container isolation by running them in a lightweight virtual machine. gVisor takes a different approach than kata container. It effectively implements a Linux kernel that runs on the host system. Therefore, syscalls are not shared anymore across all containers on the host system.

Install gVisor

you don't need to remember installations steps, exam will install it in all hosts if required.

gVisor is a Google-developed tool designed to enhance security in containerized environments by isolating containers from the host kernel.

It functions as a lightweight, user-space kernel, intercepting and handling system calls made by containerized applications.

Key Components:

- Sentry:** A user-space kernel that handles system calls, file system operations, and network operations. It offers a reduced set of functionalities compared to the full Linux kernel, minimizing the attack surface.
- Gofer:** Assists Sentry by managing file-related operations and providing a layer of abstraction over the host file system.

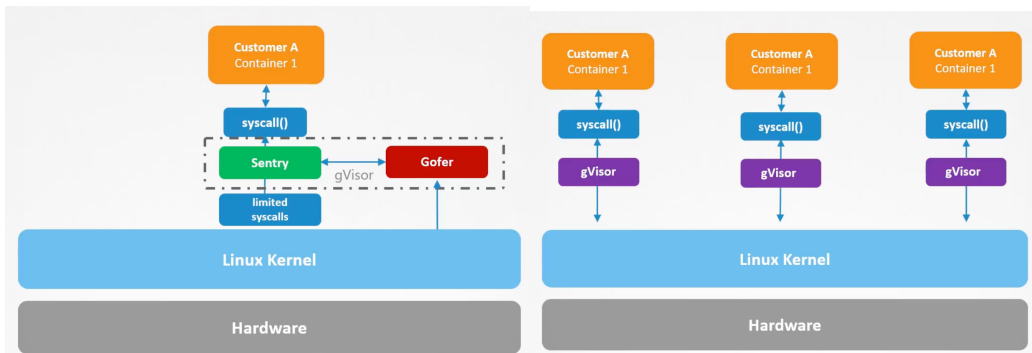
Network Isolation: gVisor implements its own network stack, isolating the container's network operations from the host system.

Security Advantage: By providing a distinct kernel per container, gVisor limits the risk of kernel exploits affecting the host or other containers.

Application Compatibility: gVisor is compatible with most Linux applications but may not support all system calls, leading to potential incompatibilities with some software.

§

Performance Impact: The additional isolation layers in gVisor can lead to a performance overhead, as system calls are handled by the user-space Sentry rather than the host kernel directly.



1) Install dependencies

```
$ sudo apt-get update && \
sudo apt-get install -y \
apt-transport-https \
ca-certificates \
curl \
gnupg
```

2) Get the keys

```
$ curl -fsSL https://gvisor.dev/archive.key | sudo gpg
--dearmor -o /usr/share/\
keyrings/gvisor-archive-keyring.gpg
```

```
$ echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/\
gvisor-archive-keyring.gpg]
https://storage.googleapis.com/gvisor/releases \
release main" | sudo tee
/etc/apt/sources.list.d/gvisor.list > /dev/null
```

3) Install gVisor runtime

```
$ sudo apt-get update && sudo apt-get install -y runsc
```

4) configure containerd with runsc, and restart

```
$ cat <<EOF | sudo tee /etc/containerd/config.toml
version = 2
[plugins."io.containerd.runtime.v1.linux"]
  shim_debug = true
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
  runtime_type = "io.containerd.runc.v2"
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runsc]
  runtime_type = "io.containerd.runsc.v1"
EOF
```

5) restart

```
$ sudo systemctl restart containerd
```

Use gVisor

1) Define a runtime class using runsc handler

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
```

2) Call it in PoD

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  runtimeClassName: gvisor
  containers:
  - name: nginx
    image: nginx:1.23.2
```

Check with the sandboxing vm starting up : *\$ kubectl exec nginx -- dmesg*

Kata containers

Kata Containers uses a unique approach to container sandboxing, different from gVisor.

It isolates each container in its own virtual machine (VM), providing dedicated kernels.

This method prevents problems arising from containers directly communicating with the same OS kernel.

Containers are isolated; if one crashes or is abused, it doesn't affect the entire system.

Concerns include the time and resources needed to create a VM for each container.

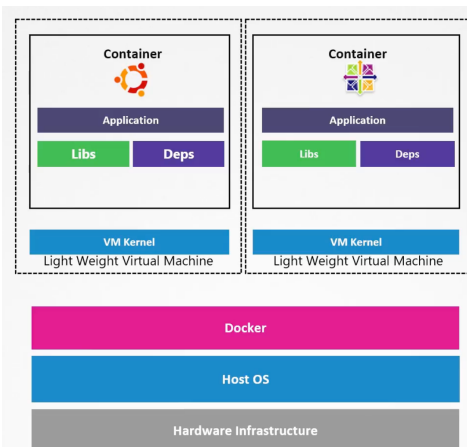
Kata Containers' VMs are lightweight, focusing on performance, but still incur a performance penalty compared to traditional containers.

Kata Containers require more memory and compute resources.

They need hardware virtualization support, limiting compatibility with typical cloud service providers.

Nested virtualization is not widely supported by cloud providers and often has poor performance.

Optimal performance of Kata Containers can be achieved on dedicated physical or bare metal servers in the cloud.

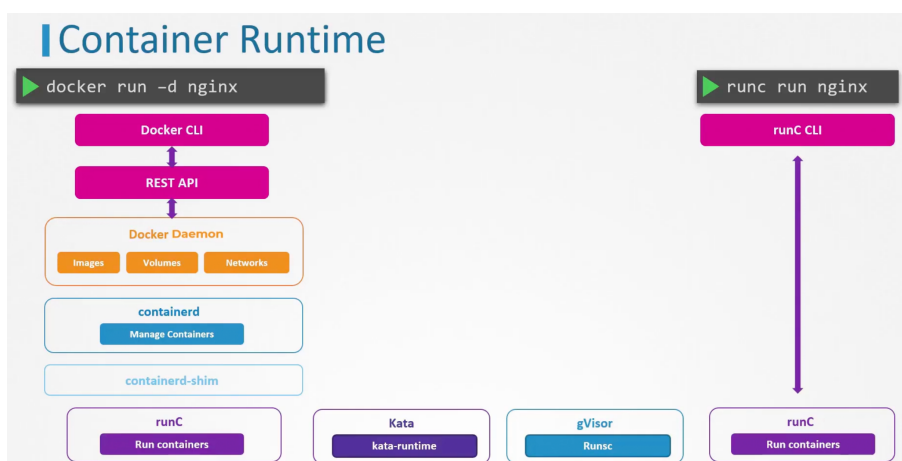


RunTime Classes

A runtime class in Kubernetes (K8s) and Docker context refers to a feature in Kubernetes that allows you to configure and **select container runtimes** for pods.

It's a k8s resource called **RuntimeClass**, (see gVisor example)

You can bypass docker and act on runtime directly (see example on right), but you lose all docker & containerd management benefits



§

Question: which runtime is used by containerd ? >>>> runC !

Pod to Pod Encryption with mTLS

every Pod can talk to any other Pod by targeting its virtual IP address unless you put a more restrictive network policy in place. The communication between two Pods is unencrypted by default.

mTLS = mutual TLS, just both sides authenticate other side,

options to implement mTLS easily without hassle of certificates is

- + using Services Meshes tools **Linkerd**, **Istio**.. *istio is implemented as sidecar container, that takes the packet from main container, encrypt and send, and on other side, decrypt it , and pass it to dest container*

- + simpler VPNs that doesn't require certs like **Wireguard** (supported by CNIs **Calico** and **Cilium**)

Domain 6. Supply Chain Security

Minimizing the Base image footprint

Picking a Base image small in size

Pick rather a small base container image , for instance ***alpine 3.17.0*** (**7Mb only**)

Reduce footprint further with **distroless image** , which contains no shell and really minimum , for instance google: ***gcr.io/distroless/static-debian11*** (**2Mb only !**)

>> how to t-shoot **distroless images** ? use **ephemeral containers** , check k8s documentation

Extra info: parent image and base image:

FROM scratch

ADD rootfs.tar.xz /

CMD ["bash"]

FROM debian:buster-slim

ENV HTTPD_PREFIX /usr/local/apache2

ENV PATH \$HTTPD_PREFIX/bin:\$PATH

WORKDIR \$HTTPD_PREFIX

Content trimmed for brevity

FROM httpd

COPY index.html htdocs/index.html

Using a Multi-Stage Approach for Building Container Images

The idea of a <u>multi-stage build in Docker</u> is : >> separate the build stage from the runtime stage, so building dependencies are removed from final container image, so its smaller. example : stage 1: build the app from <i>golang</i> image stage 2: copy the the binary into <i>alpine</i> image so final image is a light alpine image with the executable.	Example 6-2. Building and testing a Go program as part of a multi-stage Dockerfile <pre>FROM golang:1.19.4-alpine AS build RUN apk add --no-cache git WORKDIR /tmp/go-sample-app COPY go.mod . COPY go.sum . RUN go mod download COPY . . RUN CGO_ENABLED=0 go test -v RUN go build -o ./out/go-sample-app . FROM alpine:3.17.0 RUN apk add ca-certificates COPY --from=build /tmp/go-sample-app/out/go-sample-app /app/go- CMD ["/app/go-sample-app"]</pre>
---	---

Reducing number of Layers

Instructions like FROM, COPY, RUN and CMD are layers , The more layers, slower is the image. >> solution is execute multiple commands in a row.	<pre>FROM ubuntu:22.10 RUN apt-get update -y && apt-get upgrade -y && apt-get install</pre>
---	---

Using container image optimization tools

DockerSlim : optimize image and dependencies

Dive : optimize image layers, or unnecessary layers.

Using a private docker repo and image

Docker use case	1) Login to repo via docker cli docker login private-registry.io 2) Use the image to run container docker run private-registry.io/apps/internal-app
Kubernetes use case	1) Create a secret docker registry <pre>kubectl create secret docker-registry regcred \ --docker-server= \ --docker-username= \ --docker-password= \ --docker-email=</pre> !! attention, when creating docker-registry via imperative way, use -docker-username Not -username ... and so on

§

	2) Use it in PoD definition <pre> nginx-pod.yaml apiVersion: v1 kind: Pod metadata: name: nginx-pod spec: containers: - name: nginx image: private-registry.io/apps/internal-app imagePullSecrets: - name: regcred </pre>
--	---

Security the supply chain

Sign and validate image:

Signing container images (sha256 signature),	docker trust sign
can be used in k8s pod definition to validate the image	<pre> apiVersion: v1 kind: Pod metadata: name: alpine-valid spec: containers: - name: alpine image: alpine@sha256:c0d488a800e4127c334ad20d61d7bc21b4097540327217dfab52262adc02380c command: ["/bin/sh"] args: ["-c", "while true; do echo hello; sleep 10; done"] </pre>

Enforce image with OPAGatekeeper

‘ConstraintTemplate’ that enforces container image <pre> apiVersion: templates.gatekeeper.sh/v1 kind: ConstraintTemplate metadata: name: k8sallowedrepos annotations: metadata.gatekeeper.sh/title: "Allowed Repositories" metadata.gatekeeper.sh/version: 1.0.0 description: >- Requires container images to begin with a string from the specified list. spec: crd: spec: names: kind: K8sAllowedRepos validation: openAPIV3Schema: type: object properties: repos: description: list of prefixes a container image is allowed to have. </pre>	‘Constraint’ that passes data which is PoD and repo is gcr.io <pre> apiVersion: constraints.gatekeeper.sh/v1beta1 kind: K8sAllowedRepos metadata: name: repo-is-gcr spec: match: kinds: - apiGroups: [""] kinds: ["Pod"] parameters: repos: - "gcr.io/" </pre>
---	--

§

```

        type: array
        items:
          type: string
targets:
- target: admission.k8s.gatekeeper.sh
 rego: |
    package k8sallowedrepos
    violation[{"msg": msg}] {
      container := input.review.object.spec.containers[_]
      satisfied := [good | repo = input.parameters.repos[_] ; \
        good = startswith(container.image, repo)]
      not any(satisfied)
      msg := sprintf("container <%v> has an invalid image repo <%v>, allowed
        repos are %v", [container.name, container.image,
        input.parameters.repos])
    }

```

Enforce image with Admission Controller

>> here we are using the default ImagePolicyWebhook, so following configuration is the way to use it .

Check config in previous chapter :

"Admission Control 3 – Default Ones but require config and external webhook –"

Static analysis of user workload

Hadolint

Kubesecc

Scanning images of known vulnerabilities

Trivy

Trivy official site	https://aquasecurity.github.io/trivy/latest/
Pull image and scan it	# crictl pull python:3.10.0a4-alpine # trivy image python:3.10.0a4-alpine
Filter only HIGH severity	# trivy image python:3.6.12-alpine3.11 --severity="HIGH"
Scan tar file	# trivy image --input alpine.tar x
Output to file or json	--format json --output file.txts

Domain 7. Monitoring, Logging, and Runtime Security

Falco

Set output is std output or file	/etc/falco/falco.yaml
Custom rules	/etc/falco/falco_rules.yaml
K8s audit rules	/etc/falco/k8s_audit_rules.yaml

§

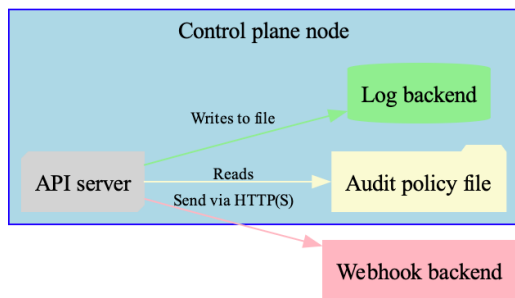
Check falco logs	sudo journalctl -fu falco
Falco Rule	<pre>- rule: access_camera desc: a process other than skype/webex tries to access the camera condition: evt.type = open and fd.name = /dev/video0 and not proc.name in skype, webex) output: Unexpected process opening camera video device (command=%proc.cmdline) priority: WARNING</pre>
Override existing rule	Instead of modifying the rule definition directly in <code>/etc/falco/falco_rules.yaml</code> , I'd suggest you redefine the rule in <code>/etc/falco/falco_rules.local.yaml</code>
Always restart falco after a change	sudo systemctl restart falco
If the same rule is configured on all rule files defined in the <code>rules_file</code> list, which one will take precedence?	The Rule defined in the file that comes last in the list

Immutability of contains at runtime

Ways to add immutability of container	• Use distroless • Use configmap or secrets , and not local attributes • Use pod securityContext : <code>readOnlyRootFilesystem</code> >> its in container level not pod /speclevel >> to find it in documentation, look for 'security context in PoD'
What if app needs some parts to be writable: example: nginx >> in this example, /var/run only is writable , anything other not.	<pre>apiVersion: v1 kind: Pod metadata: name: nginx spec: containers: - name: nginx image: nginx:1.21.6 securityContext: readOnlyRootFilesystem: true volumeMounts: - name: nginx-run mountPath: /var/run volumes: - name: nginx-run emptyDir: {}</pre>

Using Audit logs to monitor access

Kubeapi architecture and audit logs format	• What event occurred? • Who triggered the event? • When was it triggered? • Which Kubernetes component handled the request?
--	---



1) Create Audit Policy File (see pic above)

Level	Effect
None	Do not log events matching this rule.
Metadata	Only log request metadata for the event.
Request	Log metadata and the request body for the event.
RequestResponse	Log metadata, request, and response body for the event.

- ❶ Prevents generating logs for all requests in the RequestReceived stage
- ❷ Logs Pod changes at RequestResponse level
- ❸ Logs specialized Pod events, e.g., log and status requests, at the Metadata level

```

apiVersion: audit.k8s.io/v1
kind: Policy
omitStages:
  - "RequestReceived" ❶
rules:
  - level: RequestResponse ❷
    resources:
      - group: ""
        resources: ["pods"]
  - level: Metadata ❸
    resources:
      - group: ""
        resources: ["pods/log", "pods/status"]
  
```

2) Configure KubeApi Server:

--audit-policy-file: points to audit policy file

--audit-log-path: points to audit output log file

- ❶ Provides the location of the policy file and log file to the API server process.
- ❷ Mounts the policy file and the audit log directory to the given paths.
- ❸ Defines the Volumes for the policy file and the audit log directory.

To use a webhook instead :

--audit-webhook-config-file

```

...
spec:
  containers:
    - command:
      - kube-apiserver
      - --audit-policy-file=/etc/kubernetes/audit-policy.yaml ❶
      - --audit-log-path=/var/log/kubernetes/audit/audit.log ❶
    ...
    volumeMounts:
      - mountPath: /etc/kubernetes/audit-policy.yaml ❷
        name: audit
        readOnly: true
      - mountPath: /var/log/kubernetes/audit/ ❷
        name: audit-log
        readOnly: false
    ...
  volumes:
    - name: audit ❸
      hostPath:
        path: /etc/kubernetes/audit-policy.yaml
        type: File
    - name: audit-log ❸
      hostPath:
        path: /var/log/kubernetes/audit/
        type: DirectoryOrCreate
  
```

ATTENTION, DON'T FORGOT THE MOUNTVOLUME AND VOLUME !!!

ANNEXE

Deploy amicontained to check restrictions (seccomp, apparmor, ..) in a k8s with containerd runtime :

To successfully deploy and run the **amicontained** container in your Kubernetes cluster with **containerd** as the container runtime and to enable debug logging, follow these summarized steps:

Preparing the amiContained Image

1. **Clone the Repository and Build the Image:** First, clone the **amicontained** GitHub repository and build the image using **buildah**.

bashCopy code

```
clone
```

```
cd
```

2. **Tag the Image for Local Use:** Tag the built image for easier local use.

bashCopy code

3. **Export and Import the Image to containerd:** Export the image from **buildah** to a tarball and then import it into **containerd**.

bashCopy code

Deploying the Pod in Kubernetes

1. **Create a Pod Definition File:** Make a YAML file (**audit2.yaml**) to define the **amicontained** pod. Include the **-d** flag in **args** for debug logging.

yamlCopy code

```
apiVersion: v1 kind: Pod metadata: labels: run: amicontained name: amicontained spec: securityContext:
seccompProfile: type: Localhost localhostProfile: profiles/audit.json containers: - image:
localhost/amicontained:latest name: amicontained imagePullPolicy: IfNotPresent args: "-d"
```

2. **Apply the Pod Definition:** Deploy the pod to your Kubernetes cluster.

bashCopy code

3. **Verify Pod Status:** Ensure the pod is running correctly.

bashCopy code

4. **View Logs for Debug Information:** Check the logs to see the debug output from **amicontained**.

bashCopy code

§

Summary of Debug Log Output

Upon running `amicontained` with the `-d` flag, you will receive detailed information about the container's runtime environment. This includes:

- **Container Runtime Detection:** Verifies if a known container runtime is detected.
- **Namespace Capabilities:** Lists the namespaces the container has access to (e.g., `pid`, `user`).
- **AppArmor Profile:** Shows the active AppArmor profile.
- **Capabilities:** Details the bounding capabilities available to the container.
- **Seccomp Status:** Indicates if Seccomp is enabled and lists allowed syscalls.

This structured approach allows you to build and deploy the `amicontained` tool within a Kubernetes cluster managed by `containerd`, enabling detailed introspection of the container's runtime environment with enhanced debugging capabilities.

Default kube-system pods not coming up after modifying yaml :

- + if configured as service, look at
#ps aux | grep kube-apiserver or scheduler or other pod,
- + look also at
#journalctl -fu pod-name
- + check if pod and insisde containers are up or not , for containerd :
#crictl pods
#crictl ps
- + if container is not up (absent in list of `crictl ps`) , then look at containerd logs :
#crictl ps container-id
cat /var/log/containerd/containers/..kube-apiserver..
cat /var/log/containerd/pods/..kube-api...

Label an object = to :

We want to apply pod security on namespace `alpha`. To achieve that, add the following label to the namespace `alpha`

```
pod-security.kubernetes.io/warn=baseline
```

Note: Namespace `alpha` is already created.

Check

☐ Is namespace `alpha` labeled as per the task statement?

```
uid: 9377d26e-837b-4eb0-9a39-c116de5ccf5a
spec:
  finalizers:
    - kubernetes
  status:
    phase: Active

controlplane ~ ~$ k label namespaces alpha pod-security.kubernetes.io/warn=baseline
namespace/alpha labeled

controlplane ~ ~$ k get ns alpha -o yaml
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2024-03-08T17:58:41Z"
  labels:
    kubernetes.io/metadata.name: alpha
    pod-security.kubernetes.io/warn: baseline
  name: alpha
  resourceVersion: "1028"
  uid: 9377d26e-837b-4eb0-9a39-c116de5ccf5a
spec:
  finalizers:
    - kubernetes
  status:
    phase: Active

controlplane ~ ~$
```

Another use case of admissionconfiguration that can be tuned (this use case is similar to Admission contrl 3 subchapter)

Here its about the PodSecurity default admission control

```
controlplane ~ → cat admission-configuration.yaml
apiVersion: admissionregistration.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: PodSecurity
  configuration:
    apiVersion: pod-security.admission.config.k8s.io/v1
    kind: PodSecurityConfiguration
    defaults:
      enforce: baseline
      enforce-version: latest
      audit: restricted
      audit-version: latest
      warn: restricted
      warn-version: latest
    exemptions:
      usernames: []
      runtimeClassNames: []
      namespaces: [my-namespace]
```

```
controlplane ~ → █
```

Where envfrom secret in documentation ?

Search this site

- Documentation
- Getting started
- Concepts
- Tasks
 - Install Tools
 - Administer a Cluster

Kubernetes Documentation / Tasks / Inject Data Into Applications
/ Distribute Credentials Securely Using Secrets

Distribute Credentials Securely Using Secrets

This page shows how to securely inject sensitive data, such as passwords and encryption keys, into Pods.

- Use `envFrom` to define all of the Secret's data as container environment variables. The key from the Secret becomes the environment variable name in the Pod.

```

apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret
spec:
  containers:
  - name: envs-test-container
    image: nginx
    envFrom:
    - secretRef:
        name: test-secret

```

§

Rollback deployment to old revision

k rollout history deployment web

k rollout **undo** deployment web **--to-revision=1**

Docker registry secret

We don't use `--from-literal` like secret, but specific `--docker-server` , `--docker-username` ,....

You can also use `.json` (see documentation)

```
kubectrl create secret docker-registry regcred --docker-server=<your-registry-server> --docker-username=<your-name> --docker-password=<your-pword> --docker-email=<your-email>
```

APP-ARMOR

+ you can load the profile in any place, doesn't matter the location (unlike seccomp)

+ remember , after you load with `apparmor_parser` , in add annotations under metadata, and localhost/ before profile name, and container name not pod !

DEPLOYMENT and ANNOTATIONS , NODESELECTOR,..

Those go under template (nested part), not Deployment top part

§

```

# 9_deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: apparmor
  name: apparmor
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apparmor
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: apparmor
      annotations:
        container.apparmor.security.beta.kubernetes.io/c1: localhost/very-secure # add
    spec:
      nodeSelector: # add
      security: apparmor # add
      containers:
        - image: nginx:1.19.2
          name: c1 # change
          resources: {}

```

Find K8S Components parameters : kubelet, kube-apiserver , ..

For instance notice this parameter relation with type of service clusterIP or nodeport, need to be deleted if you switch from nodeport to clusterIP

8

The screenshot shows the Kubernetes documentation page for the kube-apiserver command-line tools. The left sidebar contains a navigation menu with the following items: Documentation, Getting started, Concepts, Tasks, Tutorials, Reference, Glossary, API Overview, API Access Control, Well-Known Labels, Annotations and Taints, Kubernetes API, Instrumentation, Kubernetes Issues and Security, Node Reference Information, Networking Reference, Setup tools, Command line tool (kubectl), and Component tools. The 'Component tools' item is highlighted with a red box. The main content area lists the following flags:

- kubelet-client-certificate string**: Path to a client cert file for TLS.
- kubelet-client-key string**: Path to a client key file for TLS.
- kubelet-preferred-address-types strings**: Default: "Hostname,InternalDNS,InternalIP,ExternalDNS,ExternalIP". List of the preferred NodeAddressTypes to use for kubelet connections.
- kubelet-timeout-duration**: Default: 5s. Timeout for kubelet operations.
- kubernetes-service-node-port int**: If non-zero, the Kubernetes master service (which apiserver creates/maintains) will be of type NodePort, using this as the value of the port. If zero, the Kubernetes master service will be of type ClusterIP.
- lease-reuse-duration-seconds int**: Default: 60. The time in seconds that each lease is reused. A lower value could avoid large number of objects reusing the same lease. Notice that a too small value may cause performance problems at storage layer.
- livez-grace-period duration**: This option represents the maximum amount of time it should take for apiserver to complete its startup sequence and become live. From apiserver's start time to when this amount of time has elapsed, /livez will assume that unfinished post-start hooks will complete successfully and therefore return true.

HOW TO HAVE ACCESS to restricted things :

+ peak to available resource with all , and try with namespace specific, as all might not give results even if available in namespace

+ if you have access to pod , that's a big win, you can use it :

>> ssh , and look into mounted secret

>> ssh , and look into variable env

>> ssh , use his service account token , to send api queries from the pod using to the api server, to reveal other infos like secret :

(--insecure to avoid curl cert check , just same as -k)

§

```
on 12
```

config use-context restricted@infra-prod

ute a possible permission escape in Namespace restricted. The s user restricted which has only limited permissions and shouldn't values.

3-key values of the Secrets secret1, secret2 and secret3 in d. Write the decoded plaintext values into files t1, /opt/course/12/secret2 and /opt/course/12/secret3.

```
File Edit View Terminal Tabs Help
k8s@terminal: ~
k8s@terminal: ~
k8s@terminal: ~

{
  "password": "Yw4tYlhemluZwo=",
  "type": "Opaque"
},
{
  "metadata": {
    "name": "secret3",
    "namespace": "restricted",
    "uid": "3887bcd2-c47e-4162-9666-64c5367cd2c1",
    "resourceVersion": "4582",
    "creationTimestamp": "2024-01-06T10:39:10Z",
    "annotations": {
      "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\":\"v1\",\"data\":{\"password\":\"cEVuRXRSYVRpT24tdEVzVGVSg==\"},\"kind\":\"Secret\",\"metadata\":{\"annotations\":{\"password\":\"Yw4tYlhemluZwo=\", \"type\":\"Opaque\"}}\"}"
    }
  },
  "data": {
    "password": "cEVuRXRSYVRpT24tdEVzVGVSg=="
  },
  "type": "Opaque"
}
}

# echo 'cEVuRXRSYVRpT24tdEVzVGVSg==' | base64 -d
pEnEtRaTiOn-tEsTeR
# curl -X GET https://192.168.100.21:6443/api/v1/namespaces/restricted/secrets --header "Authorization: Bearer $(cat /run/secrets/kubernetes.io/serviceaccount/token)"
curl: (60) SSL certificate problem: unable to get local issuer certificate
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
# curl -X GET https://192.168.100.21:6443/api/v1/namespaces/restricted/secrets --header "Authorization: Bearer $(cat /run/secrets/kubernetes.io/serviceaccount/token)" --insecure
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "20312"
  }
}
```

HOW TO HAVE ACCESS to restricted things :

How to find syscalls of a container quickly,

Instead of seccomp, remember a container is a just a process, find it, and run strace on it.

How to find a syscalls id easily ?

```
# sudo apt-get install auditd
# ausyscall kill
→ kill is syscallid 62
```

STRACE a Container !

There are *Pods* in Namespace **team-yellow**. A security investigation noticed that some processes running in these *Pods* are using the Syscall **kill**, which is forbidden by a Team Yellow internal policy.

Find the offending *Pod(s)* and remove these by reducing the replicas of the parent *Deployment* to 0.

Answer:

Syscalls are used by processes running in Userspace to communicate with the Linux Kernel. There are many available syscalls: <https://man7.org/linux/man-pages/man2/syscalls.2.html>. It makes sense to restrict these for container processes and Docker/Containerd already restrict

§

some by default, like the `reboot` Syscall. Restricting even more is possible for example using `Seccomp` or `AppArmor`.

But for this task we should simply find out which binary process executes a specific Syscall. Processes in containers are simply run on the same Linux operating system, but isolated. That's why we first check on which nodes the *Pods* are running:

```
→ k -n team-yellow get pod -owide
NAME                                ... NODE      NOMINATED NODE   ...
collector1-7585cc58cb-n5rtd 1/1 ... cluster1-node1 <none> ...
collector1-7585cc58cb-vdlp9 1/1 ... cluster1-node1 <none> ...
collector2-8556679d96-z7g7c 1/1 ... cluster1-node1 <none> ...
collector3-8b58fdc88-pjg24 1/1 ... cluster1-node1 <none> ...
collector3-8b58fdc88-s9ltc 1/1 ... cluster1-node1 <none> ...
```

All on `cluster1-node1`, hence we ssh into it and find the processes for the first *Deployment* `collector1`.

```
→ ssh cluster1-node1

→ root@cluster1-node1:~# crictl pods --name collector1
POD ID      CREATED      STATE      NAME                                ...
21aacb8f4ca8d 17 minutes ago Ready      collector1-7585cc58cb-vdlp9 ...
186631e40104d 17 minutes ago Ready      collector1-7585cc58cb-n5rtd ...

→ root@cluster1-node1:~# crictl ps --pod 21aacb8f4ca8d
CONTAINER ID  IMAGE          CREATED      ...   POD ID
9ea02422f8660 5d867958e04e1 12 minutes ago ...   21aacb8f4ca8d

→ root@cluster1-node1:~# crictl inspect 9ea02422f8660 | grep args -A1
"args": [
  "/collector1-process"
```

1. Using `crictl pods` we first searched for the *Pods* of *Deployment* `collector1`, which has two replicas
2. We then took one pod-id to find it's containers using `crictl ps`
3. And finally we used `crictl inspect` to find the process name, which is `collector1-process`

We can find the process PIDs (two because there are two *Pods*):

```
→ root@cluster1-node1:~# ps aux | grep collector1-process
root   35039  0.0  0.1 702208 1044 ?    Ssl 13:37   0:00 ./collector1-process
root   35059  0.0  0.1 702208 1044 ?    Ssl 13:37   0:00 ./collector1-process
```

Using the PIDs we can call `strace` to find Syscalls:

```
→ root@cluster1-node1:~# strace -p 35039
strace: Process 35039 attached
futex(0x4d7e68, FUTEX_WAIT_PRIVATE, 0, NULL) = 0
kill(666, SIGTERM) = -1 ESRCH (No such process)
epoll_pwait(3, [], 128, 999, NULL, 1) = 0
kill(666, SIGTERM) = -1 ESRCH (No such process)
epoll_pwait(3, [], 128, 999, NULL, 1) = 0
kill(666, SIGTERM) = -1 ESRCH (No such process)
epoll_pwait(3, ^Cstrace: Process 35039 detached
<detached ...>
...
```

First try and already a catch! We see it uses the forbidden Syscall by calling `kill(666, SIGTERM)`.

§

AUDIT LOGS tips

View the logs better with **cat data.log |jq**

Difference between HostPath volume and EmptyDIR volume ? and where they reside in host in reality?

HostPath Volume

- **Usage:** A `HostPath` volume mounts a file or directory from the host node's filesystem into your Pod. This can be used for various purposes, such as running a container that needs access to Docker internals; using it for storage from a development environment; or for storing logs, configuration files, or other data that needs to persist beyond the life of the Pod.
- **Persistence:** The data in a `HostPath` volume is stored on the node's filesystem. Therefore, it persists beyond the lifecycle of the Pod, and can be seen and modified by other Pods running on the same node, assuming they have the correct path to the volume.
- **Use Cases:** It's typically used in scenarios where specific data or directories from the host need to be accessible to the Pod, or for single-node testing.
- **Risks:** Using `HostPath` can expose your cluster to security risks by allowing Pods to access sensitive areas of the host filesystem. It also ties your Pod to a specific node, reducing the portability and flexibility of your deployment.

EmptyDir Volume

- **Usage:** An `EmptyDir` volume is created when a Pod is assigned to a node and exists as long as that Pod is running on that node. As the name implies, it is initially empty. All containers in the Pod can read from and write to the `EmptyDir` volume, allowing data to be shared between containers in the Pod.
- **Persistence:** The contents of an `EmptyDir` volume are deleted permanently when the Pod is removed from the node for any reason, such as Pod termination, deletion, or node failure. It does not persist data across Pod restarts.
- **Use Cases:** `EmptyDir` volumes are useful for temporary storage that needs to be shared between containers running in the same Pod. This could be used for caching data, processing intermediate files, or storing data that is needed for the duration of the Pod's life but not beyond.
- **Characteristics:** Since `EmptyDir` volumes do not require external storage to be attached to the node, they can provide a fast and straightforward way to allocate storage. They can also be mounted on any node, making your Pods more portable.

Summary

In summary, `HostPath` volumes are best used when you need to access or persist data directly on the host node, while `EmptyDir` volumes are suitable for temporary, Pod-scoped storage that can be shared between containers within the same Pod. The choice between them depends on your specific storage needs and the lifecycle requirements of your data.



You

where emptydir is in reality in the host ?



ChatGPT

An `EmptyDir` volume in Kubernetes is created on the host's filesystem, but its exact location depends on the kubelet's configuration on that host. The kubelet is the primary "node agent" that runs on each node in the Kubernetes cluster. It manages the containers' runtime, ensuring that the state of each Pod matches the desired state defined in the Pod specifications.

Location of EmptyDir Volumes

The data for `EmptyDir` volumes is typically stored in a subdirectory of the kubelet's root directory, which defaults to `/var/lib/kubelet` on most systems. Within this directory, each Pod is assigned its own unique subdirectory. The `EmptyDir` volumes for a Pod are then created within that Pod's specific subdirectory. The path might look something like `/var/lib/kubelet/pods/<pod-uid>/volumes/kubernetes.io-empty-dir/<volume-name>`, where `<pod-uid>` is the unique identifier for the Pod, and `<volume-name>` is the name you've given to the `EmptyDir` volume in your Pod specification.

Temporary Nature

Despite being stored on the host, `EmptyDir` volumes are considered ephemeral or temporary storage. When a Pod is removed from a node for any reason (e.g., the Pod is deleted, or the node is restarted), the data in the `EmptyDir` volume is deleted permanently. This cleanup process helps to ensure that temporary data does not fill up the node's disk space.

Configuration and Management

The management and cleanup of these directories are handled automatically by the kubelet. As a user or administrator, you generally don't need to manage these directories manually. If there's a need to inspect the contents of an `EmptyDir` volume, you should do so through Kubernetes APIs or `kubectl` commands rather than accessing the host filesystem directly, to avoid any potential security or integrity issues.

It's also worth noting that the exact path and management of these directories can be affected by changes in kubelet configurations or the use of alternative container runtimes, but the general principle remains the same: `EmptyDir` volumes are stored on the host's filesystem in a directory managed by the kubelet, isolated to the lifecycle of the Pod.

§

Troubleshooting hidden characters or weird errors in k8s yaml files :

In vim, use `:set list` , to discover any hidden characters, like in this case it's a hidden tab !

```
apiVersion: networking.k8s.io/v1$
kind: NetworkPolicy$
metadata:$
  name: pod-access$
  namespace: dev-team$
spec:$
  policyTypes:$
  - Ingress$
  ingress:$
  - from:$
    namespaceSelector:$
    matchLabels:$
    ^I    mylabel: qa$
  - podSelector:$
    matchLabels:$
    env: testing$
$
```