

Kubernetes Hardening Standards

Automation & Engineering team is defining baseline Kubernetes hardening standards.

As a starting point, we are taking our initial baseline from the US Department of Defense who have built a robust framework for building and securing Kubernetes in partnership with the Cybersecurity and Infrastructure Security Agency.

This framework offers a best practice approach that we can adopt and build on, as we become more mature in our containerization use across Tabcorp.



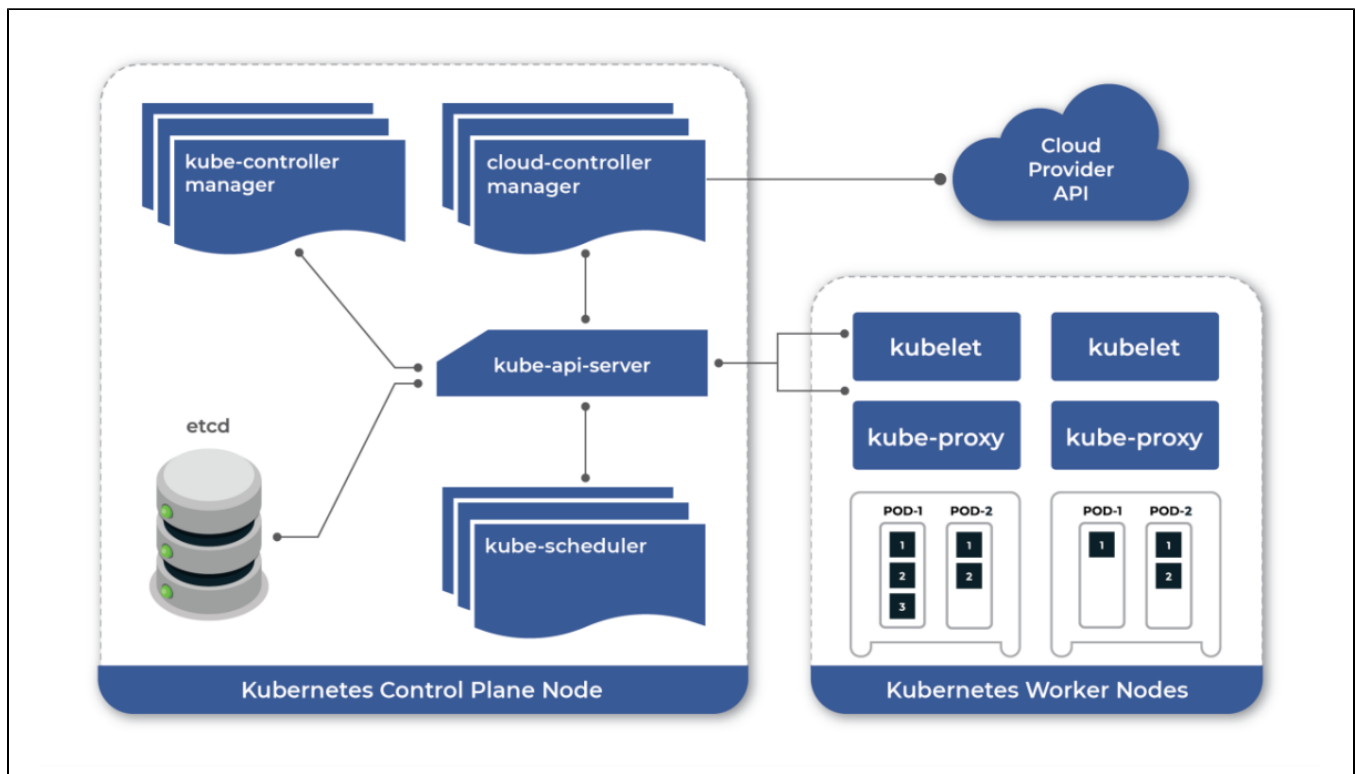
Table of Contents

- [Table of Contents](#)
- [What is Kubernetes?](#)
- [Kubernetes Architecture](#)
 - --List of k8s components--
- [Why Hardening Kubernetes?](#)
- [Kubernetes Pod security](#)
 - Use containers built to run applications as non-root users
 - Where possible, run containers with immutable file systems
 - Scan container images for possible vulnerabilities or misconfigurations
 - Denying container features frequently exploited to breakout, such as hostPID, hostIPC, hostNetwork, allowedHostPath
 - Rejecting containers that execute as the root user or allow elevation to root
 - Hardening applications against exploitation using security services such as SELinux®, AppArmor®, and seccomp
- [Network separation and hardening](#)
 - Lock down access to control plane nodes using a firewall and role-based access control (RBAC)
 - Further limit access to the Kubernetes etcd server
 - Configure control plane components to use authenticated, encrypted communications using Transport Layer Security (TLS) certificates
 - Set up network policies to isolate resources. Pods and services in different namespaces can still communicate with each other unless additional separation is enforced, such as network policies
 - Place all credentials and sensitive information in Kubernetes Secrets rather than in configuration files. Encrypt Secrets using a strong encryption method
- [Authentication and Authorization](#)
 - Disable anonymous authentication (enabled by default)
 - Use strong user authentication / Disable unauthenticated access to the API server
 - Configure Admission controllers
 - Create RBAC policies to limit administrator, user, and service account activity
- [Log auditing](#)
 - Enable audit logging (disabled by default) and configure an auditing policy
 - Persist logs to ensure availability in the case of node, Pod, or container level failure
 - Configure a metrics logger
 - Upgrading and application security practices

What is Kubernetes?

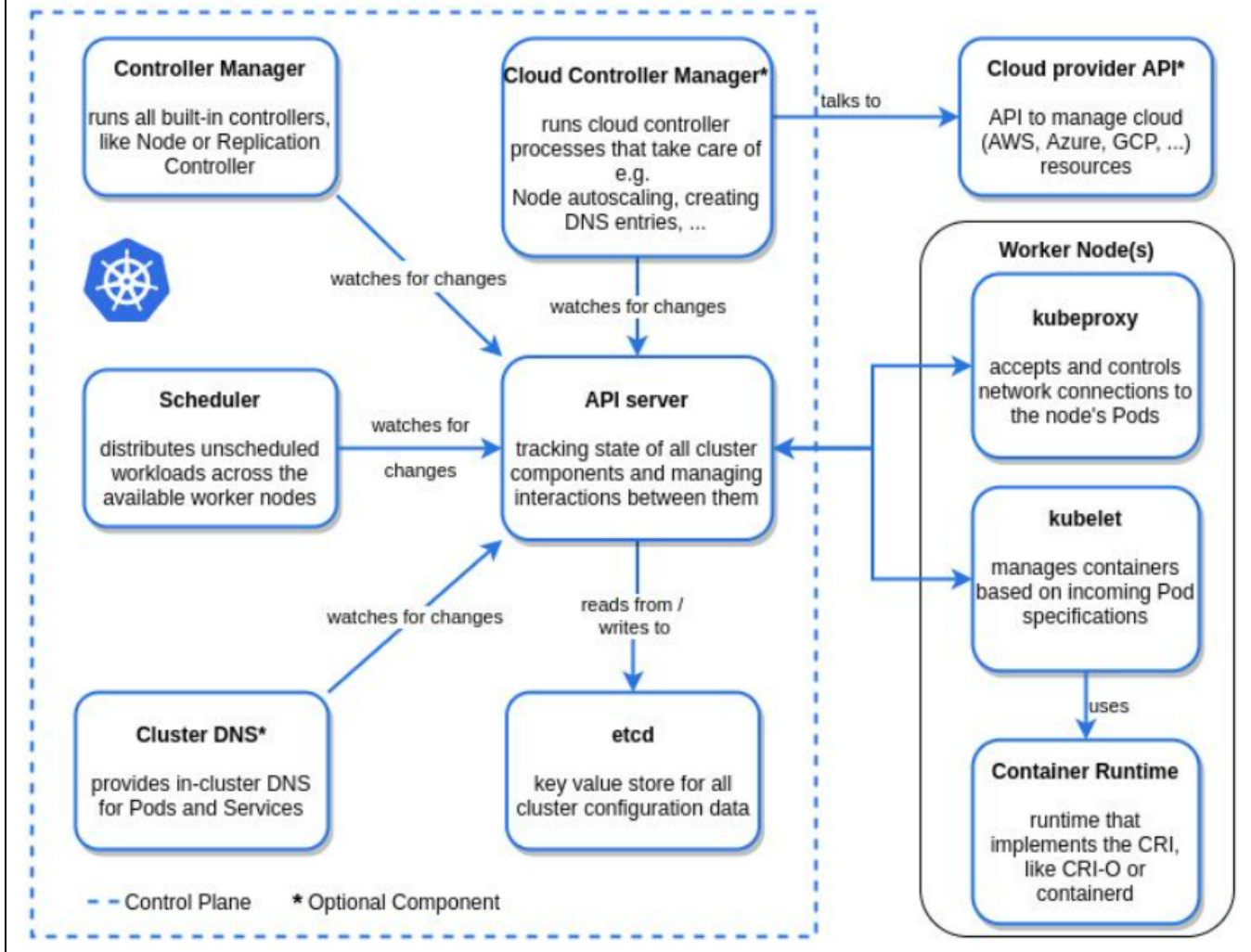
Kubernetes is an open source platform that is used to manage and automate the deployment, scaling and day to day operations of containerized applications. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation.

Kubernetes Architecture



Explained version ..

Kubernetes Architecture



--List of k8s components--

Cluster - when you deploy kubernetes, you get a cluster

Nodes - a set of worker machines that run containerized applications

Pods - components of the application workload

Control Plane - manages the worker Nodes and the Pods in the cluster.

Control Plane Components

1. **kube-apiserver** - front end that exposes the Kubernetes API
2. **etcd** - The persistent backing store where all information regarding the state of the cluster is kept. Etcd is not intended to be manipulated directly but should be managed through the API server.
3. **kube-scheduler** - watches for newly created pods and assign them to node to run on
4. **kube-controller-manager** - runs controller processes (Node controller, Job controller, Endpoints controller, Service Account & Token controllers)
5. **cloud-controller-manager** - runs controller processes linking cluster into cloud provider's API (Node controller, Route controller, Service controller)

Node Components - run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1. **kubelet** - An agent that runs on each node in the cluster. It makes sure that containers created by k8s are running in a Pod.
2. **kube-proxy** - A network proxy that runs on each node in the cluster. maintains network rules on nodes and manages traffic from network sessions inside or outside of your cluster.

- **Container runtime** - software that is responsible for running containers.
- **Add-ons** - Add-ons extend the functionality of Kubernetes (Amazon VPC CNI, CoreDNS etc)

- **Web UI (Dashboard)** - web-based UI for users to manage and troubleshoot applications running in the cluster
- **Cluster-level Logging** - saving container logs to a central log store with search/browsing interface.

Amazon VPC CNI - Enable pod networking within your cluster ~v1.10.1-eksbuild.1

CoreDNS- Enable service discovery within your cluster ~v1.8.7-eksbuild.1

kube-proxy- Enable service networking within your cluster ~v1.22.6-eksbuild.1

--Control Plane logging--

API server

Logs pertaining to API requests to the cluster.

Audit

Logs pertaining to cluster access via the Kubernetes API.

Authenticator

Logs pertaining to authentication requests into the cluster.

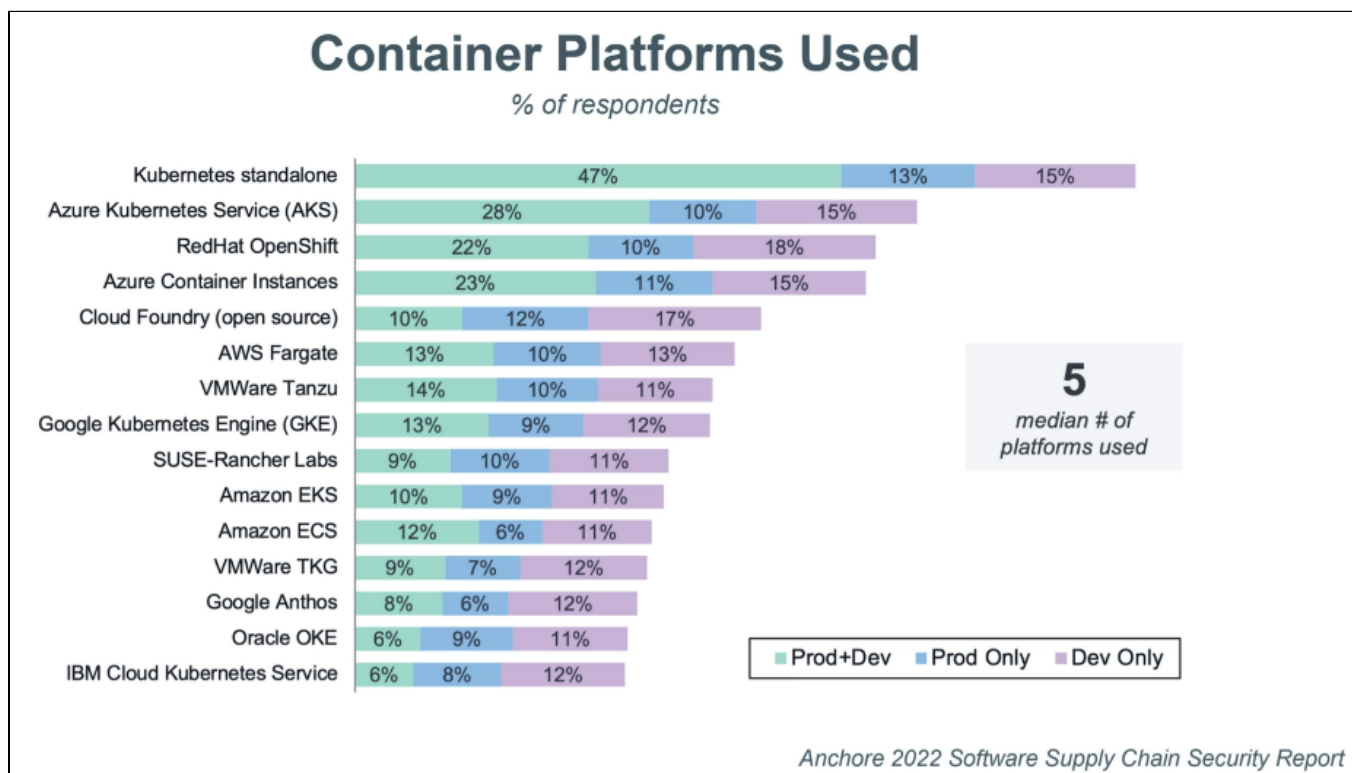
Controller manager

Logs pertaining to state of cluster controllers.

Scheduler

Logs pertaining to scheduling decisions.

Why Hardening Kubernetes?



Although containerization and operating a microservice architecture can offer improved flexibility and resilience, it does introduce some complexities in securely managing the platform and its underlying infrastructure.

As part of designing any system, you need to think about its security implications and the practices that can affect your security posture.

-For example, you need to [control who can perform actions against a set of resources](#).

-You also need the ability to [quickly identify security incidents](#), [protect your systems and services from unauthorized access](#), and [maintain the confidentiality and integrity of data through data protection](#).

This guidance is designed to mitigate the risks of using this technology and ensure applications are safe and secure throughout their lifecycle.

Containers deployments in Kubernetes clusters create both familiar and new security challenges. Given the ephemeral nature of containers, the speed and agility goals of microservices architecture, a preliminary detection of potential risks, and early discovery of viable threats yield the best security outcomes. Successfully addressing the Kubernetes security challenges requires integrating security into each phase of the container lifecycle: build, deploy, and run.

Kubernetes security is important throughout the container lifecycle due to the distributed, dynamic nature of a Kubernetes cluster. Different security approaches are required for each of the three phases of an application lifecycle: build, deploy, and runtime.

CVE (publicly disclosed cybersecurity vulnerabilities) search - https://cve.mitre.org/cve/search_cve_list.html to list latest 'kubernetes' based vulnerabilities.

Let's explore how to:

- * Stay on top of ongoing Kubernetes hygiene by hardening your nodes, employing best practices
- * Implement role-based access control of users
- * Manage Kubernetes Secrets
- * Thwart an attack
- * Manage your images and secrets

Kubernetes Pod security

NOTE



Kubernetes 1.21 starts the deprecation process for PodSecurityPolicy. As with all feature deprecations, PodSecurityPolicy will continue to be fully functional for several more releases. The current plan is to remove PSP from Kubernetes in the 1.25 release.

Refer to OPA-Gatekeeper as recommended solution below 7.

1. Use containers built to run applications as non-root users

How?



define pod security context- pods/security/security-context.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-**
spec:
  securityContext:
    runAsUser: 1009
    runAsGroup: 3000
    fsGroup: 2009
```

fsGroup setting defines a group which Kubernetes will change the permissions of all files in volumes to when volumes are mounted by a pod

2. Where possible, run containers with immutable file systems

How?



#run containers with immutable file systems #gateway\charts\metrics-server\values.yaml

```
securityContext:
  readOnlyRootFilesystem: true
  runAsNonRoot: true
  runAsUser: 1009
```

#Ensure ECR repositories have immutable tags

- Login into your AWS account
- Navigate to the ECR service at: <https://console.aws.amazon.com/ecr>
- On the ECR main page, pinpoint any Repository name that has Tag immutability set to Disabled.

Repositories (1 of 1)					
Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type
dynamo-build	414398599351.dkr.ecr.us-east-1.amazonaws.com/dynamo-build	03/18/19, 10:33:55 PM	Disabled	Disabled	AES-256

- In the top right corner of this window, select Edit button.
- In the main panel, under Tag immutability select Enabled and select the Save button at the bottom.

Repository access and tags

Repository name
414398599351.dkr.ecr.us-east-1.amazonaws.com/ dynamo-build
A namespace can be included with your repository name (e.g. namespace/repo-name).

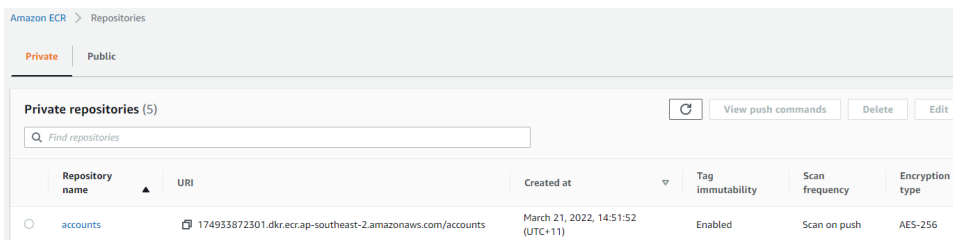
Tag immutability
Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

☒ Enabled

3. Scan container images for possible vulnerabilities or misconfigurations

How?

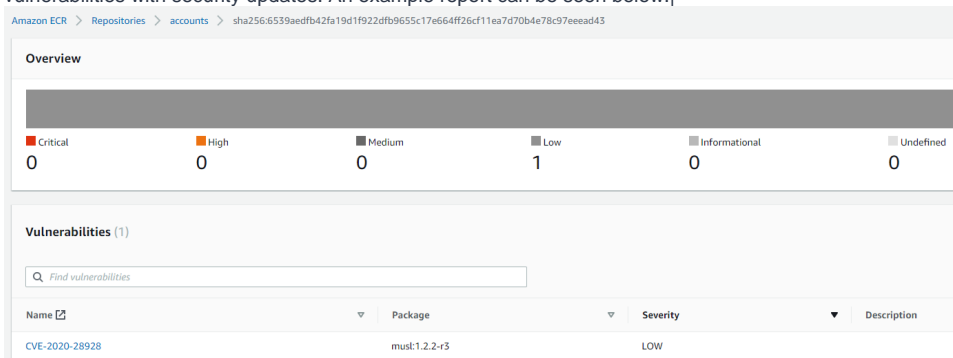
Enable the “scans on push” feature for your repositories to ensure every image automatically goes through a vulnerability scanning. AWS ECR uses Common Vulnerabilities and Exposures (CVEs) databases for findings.



The screenshot shows the Amazon ECR console interface. At the top, there's a breadcrumb trail: Amazon ECR > Repositories. Below this, there are tabs for 'Private' (selected) and 'Public'. A section titled 'Private repositories (5)' contains a search bar and buttons for 'View push commands', 'Delete', and 'Edit'. A table lists the repositories. The first entry is 'accounts' with a URI '174933872301.dkr.ecr.ap-southeast-2.amazonaws.com/accounts', created on March 21, 2022, with tag immutability enabled, scan frequency set to 'Scan on push', and encryption type 'AES-256'.

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
accounts	174933872301.dkr.ecr.ap-southeast-2.amazonaws.com/accounts	March 21, 2022, 14:51:52 (UTC+11)	Enabled	Scan on push	AES-256

After pushing your docker images to the cloud, you could see your scan results on the AWS console in detail. After you get results, you can fix vulnerabilities with security updates. An example report can be seen below:



The screenshot shows the AWS console 'Overview' section for a repository. It displays a bar chart for severity levels: Critical (0), High (0), Medium (0), Low (1), Informational (0), and Undefined (0). Below this, the 'Vulnerabilities (1)' section shows a table with one entry: CVE-2020-28928, package musl:1.2.2-r3, with a LOW severity.

Name	Package	Severity	Description
CVE-2020-28928	musl:1.2.2-r3	LOW	

Enhanced Scanning - Amazon Inspector with Eventbridge notifications, Snyk are currently assessed.

4. Denying container features frequently exploited to breakout, such as hostPID, hostIPC, hostNetwork, allowedHostPath

How?

```
podSecurityPolicy: ..\gateway\charts\kong\values.yaml
```

```
hostNetwork: false
hostIPC: false
hostPID: false
```

5. Rejecting containers that execute as the root user or allow elevation to root

How?

If `runAsNonRoot` is not set to true, containers are allowed to run as the root user.

In your workload configuration, set **securityContext.runAsNonRoot = true**

6. Hardening applications against exploitation using security services such as SELinux®, AppArmor®, and seccomp

How?

#PodSecurityPolicy is deprecated as of Kubernetes v1.21, and will be removed in v1.25

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    # docker/default identifies a profile for seccomp, but it is not particularly tied to the Docker runtime
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default,runtime/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default'
```

7. Pod security policies are supposedly being deprecated and OPA-Gatekeeper can help with a security baseline for infrastructure teams.

Gatekeeper is a customizable admission webhook for Kubernetes that enforces policies executed by the Open Policy Agent (OPA), a policy engine for Cloud Native environments hosted by CNCF.

Few Use Cases:

All namespaces must have a label that lists a point-of-contact

How?

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
 name: k8srequiredlabels
spec:
 crd:
 spec:
 names:
 kind: K8sRequiredLabels

All pods must have an upper bound for resource usage

How?

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
 name: k8scontainerlimits
spec:
 crd:
 spec:
 names:
 kind: K8sContainerLimits

All images must be from an approved/specified repository

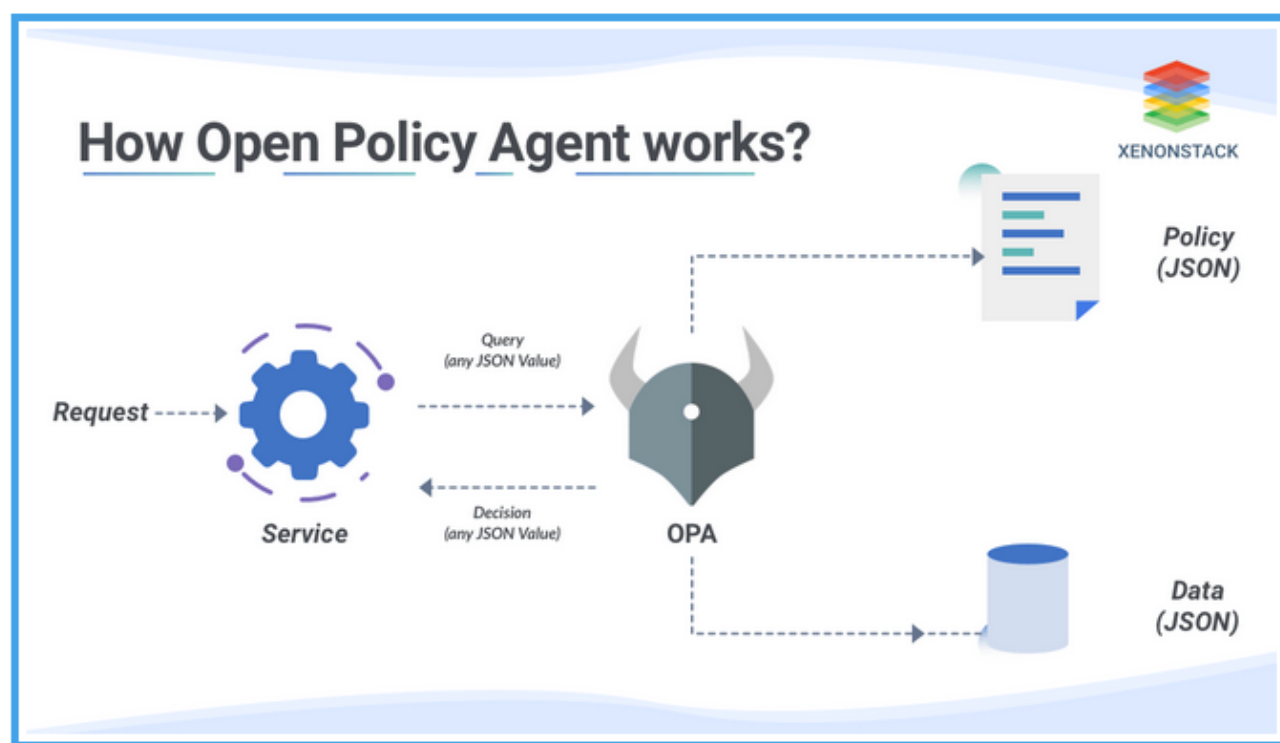
How?

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
 name: k8sallowedrepos
spec:
 crd:
 spec:
 names:
 kind: K8sAllowedRepos
OPA Gatekeeper Templates

How Open Policy Agent works?

After integration of OPA service with your application service. Now each application service request or event will go through OPA service first for policy verification. Steps for policy verification:-

- ❖ A query is sent to OPA service in JSON format.
- ❖ Above query is processed by OPA against a policy written in rego language.
- ❖ After processing output is generated in json format known as a decision.



Few more templates on gateway repo

<https://github.tabcorp.com.au/EnterpriseSystems/gateway/tree/master/charts> ==> /charts/gatekeeper-templates/

Chart.yaml <==Helm chart to setup and deploy opa constraints.

apiVersion: v1

name: gatekeeper-templates

description: Chart to maintain eks cluster opa policies

version: 0.0.1

appVersion: "0.0.1"

disallowanonymous-constraintTemplate.yaml <== Sample constraint template


```

apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sdisallowanonymous
  annotations:
    description: Disallows associating ClusterRole and Role resources to the system:anonymous user and system:unauthenticated group.
spec:
  crd:
    spec:
      names:
        kind: K8sDisallowAnonymous
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            allowedRoles:
              allowedRoles:
                description: >-
                  The list of ClusterRoles and Roles that may be associated
                  with the `system:unauthenticated` group and `system:anonymous`
                  user.
                type: array
                items:
                  type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sdisallowanonymous

        violation[{"msg": msg}] {
          not is_allowed(input.review.object.roleRef, input.parameters.allowedRoles)
          review(input.review.object.subjects[_])
          msg := sprintf("Unauthenticated user reference is not allowed in %v %v ", [input.review.object.kind, input.review.object.metadata.name])
        }

        is_allowed(role, allowedRoles) {
          role.name == allowedRoles[_]
        }

        review(subject) = true {
          subject.name == "system:unauthenticated"
        }

        review(subject) = true {
          subject.name == "system:anonymous"
        }

```

Other templates applied on k8s cluster and its description

SI No	Constraint Templates	Description
1	blocknodeport-constraint.yaml	Disallows all Services with type NodePort. https://kubernetes.io/docs/concepts/services-networking/service/#nodeport
2	cpumemoryquota-constraint.yaml	Requires containers to have memory and CPU limits set and constrains limits to be within the specified maximum values. https://kubernetes.io/docs/concepts/configuration/manage-resources-containers
3	disallowanonymous-constraint.yaml	Disallows associating ClusterRole and Role resources to the system:anonymous user and system:unauthenticated group.
4	imagedigest-constraint.yaml	Requires container images to contain a digest (for example, image@sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2) https://kubernetes.io/docs/concepts/containers/images/
5	k8spspforbiddensysctls-constraint.yaml	Controls the `sysctl` profile used by containers. Corresponds to the `forbiddenSysctls` field in a PodSecurityPolicy. https://kubernetes.io/docs/tasks/administer-cluster/sysctl-cluster/
6	requiredlabels-constraintTemplate.yaml	Requires resources to contain specified labels, with values matching provided regular expressions.
7	uniqueserviceselector-constraint.yaml	Requires Services to have unique selectors within a namespace. Selectors are considered the same if they have identical keys and values. Selectors may share a key/value pair so long as there is at least one distinct key/value pair between them. https://kubernetes.io/docs/concepts/services-networking/service/#defining-a-service

Why OPA is important:

Open Policy Agent is an open-source engine that allows you to write policies declaratively as code and then use those policies as part of a decision-making process. It employs the Rego policy language, which enables you to write policies for various services in the same language.

OPA you can enforce custom policies on Kubernetes objects without recompiling or reconfiguring the Kubernetes API server.

Importance:

- OPA is designed to work with any kind of JSON input, meaning it can easily integrate with any tool that produces JSON output.
- it allows you to use a standard policy language across many parts of your system, rather than relying on multiple vendor-specific technologies.
- OPA supports unit-testing, making it easier and faster to iterate your policies with confidence that they won't break.
- With Policy as Code, you can track the standard development lifecycle using PR, CI, etc. and provide policy change history.

<https://www.styra.com/blog/what-is-open-policy-agent/>

Network separation and hardening

1. Lock down access to control plane nodes using a firewall and role-based access control (RBAC)

How?



#Use separate networks for the control plane

2. Further limit access to the Kubernetes etcd server

How?



#it is recommended to grant permission to only those nodes that require access to etcd clusters.

3. Configure control plane components to use authenticated, encrypted communications using Transport Layer Security (TLS) certificates

How?



#The Kubernetes API Server must use **TLS 1.2**, at a minimum, to protect the confidentiality of sensitive data during electronic dissemination.
#You can secure an application running on Kubernetes by creating a secret that contains a TLS (Transport Layer Security) private key and certificate.

#Create TLS certificates using Kubernetes API:

```
apiVersion: v1
data:
  tls.crt: "base64 encoded cert"
  tls.key: "base64 encoded key"
kind: Secret
metadata:
  name: my-tls-secret
  namespace: default
type: kubernetes.io/tls
```

4. **Set up network policies to isolate resources.** Pods and services in different namespaces can still communicate with each other unless additional separation is enforced, such as network policies

How?



By Default: Each pod can talk to any other pod
But: Not every pod needs to talk to all others

Best Practice: Limit the communication with Network Rules (Define Communication rules between Pods)

Control traffic flow at the IP address or port level

Which pods can talk to which

Which pods they can receive traffic from

When writing a NetworkPolicy, you can target a range of ports instead of a single port.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: multi-port-egress
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 32000
      endPort: 32768
```

5. **Place all credentials and sensitive information in Kubernetes Secrets rather than in configuration files. Encrypt Secrets using a strong encryption method**

A note about Secrets!

Remember that secrets encode data in base64 format. Anyone with the base64 encoded secret can easily decode it. As such the secrets can be considered as not very safe.

The concept of safety of the Secrets is a bit confusing in Kubernetes. The [kubernetes documentation](#) page and a lot of blogs out there refer to secrets as a "safer option" to store sensitive data. They are safer than storing in plain text as they reduce the risk of accidentally exposing passwords and other sensitive data.

In my opinion it's not the secret itself that is safe, it is the practices around it.

Secrets are not encrypted, so it is not safer in that sense. However, some best practices around using secrets make it safer. As in best practices like:

- Not checking-in secret object definition files to source code repositories.
- [Enabling Encryption at Rest](#) for Secrets so they are stored encrypted in ETCD.

Also the way kubernetes handles secrets. Such as:

- A secret is only sent to a node if a pod on that node requires it.
- Kubelet stores the secret into a tmpfs so that the secret is not written to disk storage.
- Once the Pod that depends on the secret is deleted, kubelet will delete its local copy of the secret data as well.

Read about the [protections](#) and [risks](#) of using secrets [here](#)

Having said that, there are other better ways of handling sensitive data like passwords in Kubernetes, such as using tools like Helm Secrets, [Hashi Corp Vault](#).

How?

#Kubernetes Secrets is a mechanism that allows for storing secrets within a centralized repository named **etcd**, essentially a key-value store that holds the complete information of the cluster.

#Problem: Etcd stores Secrets in a non-encrypted form – base64, which is an encoding method, not encryption.

#Enable Encryption-at-rest

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
- resources:
  - secrets
  providers:
  - identity: {}
  - aesgcm:
      keys:
      - name: key1
        secret: c2VjcmV0IGlzIHNLy3VyZQ==
      - name: key2
        secret: dGhpcyBpcyBwYXNzd29yZA==
  - aescbc:
      keys:
      - name: key1
        secret: c2VjcmV0IGlzIHNLy3VyZQ==
      - name: key2
        secret: dGhpcyBpcyBwYXNzd29yZA==
  - secretbox:
      keys:
      - name: key1
        secret: YWJjZGVmZ2hpamtsbW5vcHFyc3Rldnd4eXoxMjM0NTY=
```

#How to ensure all secrets are encrypted

```
$ kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

Authentication and Authorization

1. Disable anonymous authentication (enabled by default)

How?

kubernetes version 1.6 and later allow anonymous authentication by default. We can disable the anonymous authentication by passing the --**anonymous-auth=false** flag

2. Use strong user authentication / Disable unauthenticated access to the API server

How?

Ensure the authentication.k8s.io/v1beta1 API group is enabled in the API server. start the kubelet with the --authentication-token-webhook and --kubeconfig flags.

3. Configure Admission controllers

How?

```
kube-apiserver --enable-admission-plugins=NamespaceLifecycle,LimitRanger ...
```

```
kube-apiserver -h | grep enable-admission-plugins
```

4. Create RBAC policies to limit administrator, user, and service account activity

How?



```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::555555555555:role/devel-worker-nodes-NodeInstanceRole-74RF4UBDUKL6
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
  mapUsers: |
    - userarn: arn:aws:iam::111122223333:user/<username>
      username: <username>
      groups:
        - system:masters
```

Once the user map is added in the configuration we need to create cluster role binding for that user:

```
kubectl create clusterrolebinding ops-user-cluster-admin-binding-<username> --clusterrole=cluster-admin --
user=<username>
```

Replace the placeholder with proper values

Log auditing

1. Enable audit logging (disabled by default) and configure an auditing policy

How?



#Sample audit policy config file to audit only certain events about pods, like pod creation, and deletion.
#Policy file will be saved at "/kube/audit/policy.yaml" on master node filesystem.

```
apiVersion: audit.k8s.io/v1
kind: Policy
# Don't generate audit events for all requests in RequestReceived stage.
omitStages:
  - "RequestReceived"
rules:
  # Log pod changes at Request level
  - level: Request
    resources:
      - group: ""
        resources: ["pods"]
  # Log pod changes at RequestResponse level
  - level: RequestResponse
    resources:
      - group: ""
        resources: ["pods"]
```

2. Persist logs to ensure availability in the case of node, Pod, or container level failure

How?



#logs are usually stored in files under the **/var/log** directory of the Kubernetes master **node server** on which the service runs.

#In Kubernetes, when pods are evicted, crashed, deleted, or scheduled on a different node, the logs from the containers are gone. The system cleans up after itself. Therefore you lose any information about why the anomaly occurred.

##**kubectl logs -p** will fetch logs from existing resources at API level. This means that terminated pods' logs will be unavailable using this command.

#mount the logs directory inside the container to the host machine as well, using the PersistentVolume and PersistentVolumeClaim

#let's say your pods are running in two nodes: mynode-1 and mynode-2

PersistentVolume spec sample.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: myapp-log-pv
spec:
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /var/log/myapp
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - mynode-1
                - mynode-2
```

Your **PersistentVolumeClaim** spec sample.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp-log-pvc
spec:
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  storageClassName: local-storage
  resources:
    requests:
      storage: 2Gi
  volumeName: myapp-log
```

Deployment spec spec sample.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deploy
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp
          image: myrepo/myapp:latest
          volumeMounts:
            - name: log
              mountPath: /var/log
      volumes:
        - name: log
          persistentVolumeClaim:
            claimName: myapp-log-pvc
```

3. Configure a metrics logger

How?



#The Kubernetes Metrics Server collects resource metrics from the kubelet running on each worker node and exposes them in the Kubernetes API server through the Kubernetes Metrics API.

#Metrics Server can be installed either directly from YAML manifest or via the official Helm chart. To install the latest Metrics Server release from the *components.yaml* manifest, run the following command.

```
kubectl get pods --all-namespaces | grep metrics-server
```

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

#Use kubectl get to query the Metrics API

```
kubectl get --raw /apis/metrics.k8s.io/v1beta1/nodes/<NODE_NAME> | jq
```

```
kubectl get --raw /apis/metrics.k8s.io/v1beta1/namespaces/<NAMESPACE>/pods/<POD_NAME> | jq
```

```
kubectl top node
```

```
kubectl top pod --namespace yournamespace
```

4. Upgrading and application security practices

How?



#Immediately apply security patches and updates

#Perform periodic vulnerability scans and penetration tests

#Remove components from the environment when they are no longer needed

Recommendations for **cluster administrators** included:

- Attribute Based Access Controls vs Role Based Access Controls
- RBAC best practices
- Node-host configurations and permissions
- Default settings and backwards compatibility
- Networking
- Environment considerations
- Logging and alerting

Recommendations for **Kubernetes developers** included:

- Avoid hardcoding paths to dependencies
- File permissions checking
- Monitoring processes on Linux
- Moving processes to a cgroup
- Future cgroup considerations for Kubernetes
- Future process handling considerations for Kubernetes