CIS Benchmarks

CIS Amazon Elastic Kubernetes Service (EKS) Benchmark

v1.0.0 - 07-20-2020

Terms of Use

Please see the below link for our current terms of use:

https://www.cisecurity.org/cis-securesuite/cis-securesuite-membership-terms-of-use/

Table of Contents

Overview

This document provides prescriptive guidance for running Amazon Elastic Kubernetes Service (EKS) following recommended security controls. This benchmark only includes controls which can be modified by an end user of Amazon EKS.
To obtain the latest version of this guide, please visit www.cisecurity.org. If you have questions, comments, or have identified ways to improve this guide, please write us at support@cisecurity.org.

Intended Audience

This document is intended for cluster administrators, security specialists, auditors, and any personnel who plan to develop, deploy, assess, or secure solutions that incorporate Amazon EKS using managed and self-managed nodes.
Customers using Amazon EKS on AWS Fargate are not responsible for node management. Hence, this document is not scoped for Amazon EKS on AWS Fargate customers.

Consensus Guidance

This benchmark was created using a consensus review process comprised of subject matter experts. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Each CIS benchmark undergoes two phases of consensus review. The first phase occurs during initial benchmark development. During this phase, subject matter experts convene to discuss, create, and test working drafts of the benchmark. This discussion occurs until consensus has been reached on benchmark recommendations. The second phase begins after the benchmark has been published. During this phase, all feedback provided by the Internet community is reviewed by the consensus team for incorporation in the benchmark. If you are interested in participating in the consensus process, please visit https://workbench.cisecurity.org/.

Typographical Conventions

The following typographical conventions are used throughout this guide:

Convention	Meaning
`Stylized Monospace font`	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
Monospace font	Used for inline code, commands, or examples. Text should be interpreted exactly as presented.
<italic font in brackets>	Italic texts set in angle brackets denote a variable requiring substitution for a real value.
Italic font	Used to denote the title of a book, article, or other publication.
Note	Additional information or caveats

Assessment Status

An assessment status is included for every recommendation. The assessment status indicates whether the given recommendation can be automated or requires manual steps to implement. Both statuses are equally important and are determined and supported as defined below:

Automated

Represents recommendations for which assessment of a technical control can be fully automated and validated to a pass/fail state. Recommendations will include the necessary information to implement automation.

Manual

Represents recommendations for which assessment of a technical control cannot be fully automated and requires all or some manual steps to validate that the configured state is set as expected. The expected state can vary depending on the environment.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1**

  Level 1 profile for running Automated Assessment

- **Level 2**

  Level 2 profile for running Automated Assessment

Acknowledgements

This benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

Special thanks to Authors: Paavan Mistry and Randall Mowen

Contributor/s
Michael Hausenblas
Mark Larinde
Angus Lees
Rory Mccune
Rafael Pereyra
Abeer Sethi
Gert Van Den Berg

Recommendations

1 Control Plane Components

Security is a shared responsibility between AWS and the Amazon EKS customer. The shared responsibility model describes this as security of the cloud and security in the cloud:

Security of the cloud – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and etcd database. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon EKS, see AWS Services in Scope by Compliance Program.

Security in the cloud – Your responsibility includes the following areas.

- The security configuration of the data plane, including the configuration of the security groups that allow traffic to pass from the Amazon EKS control plane into the customer VPC
- The configuration of the worker nodes and the containers themselves
- The worker node guest operating system (including updates and security patches)
    - Amazon EKS follows the shared responsibility model for CVEs and security patches on managed node groups. Because managed nodes run the Amazon EKS-optimized AMIs, Amazon EKS is responsible for building patched versions of these AMIs when bugs or issues are reported and we are able to publish a fix. However, customers are responsible for deploying these patched AMI versions to your managed node groups.
- Other associated application software:
    - Setting up and managing network controls, such as firewall rules
    - Managing platform-level identity and access management, either with or in addition to IAM
- The sensitivity of your data, your company's requirements, and applicable laws and regulations

AWS is responsible for securing the control plane, though you might be able to configure certain options based on your requirements. Section 2 of this Benchmark addresses these configurations.

2 Control Plane Configuration

This section contains recommendations for Amazon EKS control plane logging configuration. Customers are able to configure logging for control plane in Amazon EKS.

2.1 Logging

2.1.1 Enable audit Logs (Manual)

Profile Applicability:

- Level 1

Description:

The audit logs are part of the EKS managed Kubernetes control plane logs that are managed by Amazon EKS. Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations.

Rationale:

Exporting logs and metrics to a dedicated, persistent datastore such as CloudTrail ensures availability of audit data following a cluster security event, and provides a central location for analysis of log and metric data collated from multiple sources.

Audit:

Perform the following to determine if CloudTrail is enabled for all regions:
Via the Management Console

1. Sign in to the AWS Management Console and open the EKS console at https://console.aws.amazon.com/eks
2. Click on Cluster Name of the cluster you are auditing
3. Click Logging
   You will see Control Plane Logging info

```
API Server              Audit                   Authenticator
Enabled/Disabled        Enabled/Disabled        Enabled/Disabled

Controller Manager      Scheduler
Enabled/Disabled        Enabled/Disabled
```

4. Ensure all 5 choices are set to Enabled

Remediation:

Perform the following to determine if CloudTrail is enabled for all regions:
Via The Management Console

1. Sign in to the AWS Management Console and open the EKS console at https://console.aws.amazon.com/eks
2. Click on Cluster Name of the cluster you are auditing
3. Click Logging
4. Select Manage Logging from the button on the right hand side
5. Toggle each selection to the Enabled position.
6. Click Save Changes

Via CLI

```
aws --region "${REGION_CODE}" eks describe-cluster --name "${CLUSTER_NAME}" --query 'cluster.logging.clusterLogging[?enabled==true].types
```

Impact:

Audit logs will be created on the master nodes, which will consume disk space. Care should be taken to avoid generating too large volumes of log information as this could impact the available of the cluster nodes. S3 lifecycle features can be used to manage the accumulation and management of logs over time.

See the following AWS resource for more information on these features:
http://docs.aws.amazon.com/AmazonS3/latest/dev/object-lifecycle-mgmt.html

Default Value:

By default, cluster control plane logs aren't sent to CloudWatch Logs. ... When you enable a log type, the logs are sent with a log verbosity level of 2 . To enable or disable control plane logs with the console. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters .

Amazon EKS Information in CloudTrail CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon EKS, that activity is recorded in a CloudTrail event along with other AWS service events in Event history.

References:

1. https://kubernetes.io/docs/tasks/debug-application-cluster/audit/
2. https://aws.github.io/aws-eks-best-practices/detective/
3. https://docs.aws.amazon.com/eks/latest/userguide/control-plane-logs.html
4. https://docs.aws.amazon.com/eks/latest/userguide/logging-using-cloudtrail.html

CIS Controls:

Version 7

6 Maintenance, Monitoring and Analysis of Audit Logs
Maintenance, Monitoring and Analysis of Audit Logs

3 Worker Nodes

This section consists of security recommendations for the components that run on Amazon EKS worker nodes.

3.1 Worker Node Configuration Files

This section covers recommendations for configuration files on Amazon EKS worker nodes.

3.1.1 Ensure that the kubeconfig file permissions are set to 644 or more restrictive (Manual)

Profile Applicability:

- Level 1

Description:

If `kubelet` is running, and if it is using a file-based kubeconfig file, ensure that the proxy kubeconfig file has permissions of `644` or more restrictive.

Rationale:

The `kubelet` kubeconfig file controls various parameters of the `kubelet` service in the worker node. You should restrict its file permissions to maintain the integrity of the file. The file should be writable by only the administrators on the system.

It is possible to run `kubelet` with the kubeconfig parameters configured as a Kubernetes ConfigMap instead of a file. In this case, there is no proxy kubeconfig file.

Audit:

SSH to the worker nodes
To check to see if the Kubelet Service is running:

```
sudo systemctl status kubelet
```

The output should return `Active: active (running) since..`
Run the following command on each node to find the appropriate kubeconfig file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--kubeconfig /var/lib/kubelet/kubeconfig` which is the location of the kubeconfig file.
Run this command to obtain the kubeconfig file permissions:

```
stat -c %a /var/lib/kubelet/kubeconfig
```

The output of the above command gives you the kubeconfig file's permissions.
Verify that if a file is specified and it exists, the permissions are `644` or more restrictive.

Remediation:

Run the below command (based on the file location on your system) on the each worker node. For example,

```
chmod 644 <kubeconfig file>
```

Impact:

None.

Default Value:

See the AWS EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kube-proxy/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images
Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

3.1.2 Ensure that the kubelet kubeconfig file ownership is set to root:root (Manual)

Profile Applicability:

- Level 1

Description:

If `kubelet` is running, ensure that the file ownership of its kubeconfig file is set to `root:root`.

Rationale:

The kubeconfig file for `kubelet` controls various parameters for the `kubelet` service in the worker node. You should set its file ownership to maintain the integrity of the file. The file should be owned by `root:root`.

Audit:

SSH to the worker nodes
To check to see if the Kubelet Service is running:

```
sudo systemctl status kubelet
```

The output should return `Active: active (running) since..`
Run the following command on each node to find the appropriate kubeconfig file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--kubeconfig /var/lib/kubelet/kubeconfig` which is the location of the kubeconfig file.
Run this command to obtain the kubeconfig file ownership:

```
stat -c %U:%G /var/lib/kubelet/kubeconfig
```

The output of the above command gives you the kubeconfig file's ownership. Verify that the ownership is set to `root:root`.

Remediation:

Run the below command (based on the file location on your system) on the each worker node. For example,

```
chown root:root <proxy kubeconfig file>
```

Impact:

None

Default Value:

See the AWS EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kube-proxy/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

3.1.3 Ensure that the kubelet configuration file has permissions set to 644 or more restrictive (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that if the kubelet refers to a configuration file with the `--config` argument, that file has permissions of 644 or more restrictive.

Rationale:

The kubelet reads various parameters, including security settings, from a config file specified by the `--config` argument. If this file is specified you should restrict its file permissions to maintain the integrity of the file. The file should be writable by only the administrators on the system.

Audit:

First, SSH to the relevant worker node:
To check to see if the Kubelet Service is running:

```
sudo systemctl status kubelet
```

The output should return `Active: active (running) since..`
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Run the following command:

```
stat -c %a /etc/kubernetes/kubelet/kubelet-config.json
```

The output of the above command is the Kubelet config file's permissions. Verify that the permissions are `644` or more restrictive.

Remediation:

Run the following command (using the config file location identied in the Audit step)

```
chmod 644 /etc/kubernetes/kubelet/kubelet-config.json
```

Impact:

None.

Default Value:

See the AWS EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

3.1.4 Ensure that the kubelet configuration file ownership is set to root:root (Manual)

Profile Applicability:

- Level 1

Description:

Ensure that if the kubelet refers to a configuration file with the `--config` argument, that file is owned by root:root.

Rationale:

The kubelet reads various parameters, including security settings, from a config file specified by the `--config` argument. If this file is specified you should restrict its file permissions to maintain the integrity of the file. The file should be writable by only the administrators on the system.

Audit:

First, SSH to the relevant worker node:
To check to see if the Kubelet Service is running:

```
sudo systemctl status kubelet
```

The output should return `Active: active (running) since..`
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Run the following command:

```
stat -c %U:%G /etc/kubernetes/kubelet/kubelet-config.json
```

The output of the above command is the Kubelet config file's ownership. Verify that the ownership is set to `root:root`

Remediation:

Run the following command (using the config file location identified in the Audit step)

```
chown root:root /etc/kubernetes/kubelet/kubelet-config.json
```

Impact:

None

Default Value:

See the AWS EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kube-proxy/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

3.2 Kubelet

This section contains recommendations for kubelet configuration.

Kubelet settings may be configured using arguments on the running kubelet executable, or they may be taken from a Kubelet config file. If both are specified, the executable argument takes precedence.

To find the Kubelet config file, run the following command:

```
ps -ef | grep kubelet | grep config
```

If the `--config` argument is present, this gives the location of the Kubelet config file. This config file could be in JSON or YAML format depending on your distribution.

3.2.1 Ensure that the --anonymous-auth argument is set to false (Automated)

Profile Applicability:

- Level 1

Description:

Disable anonymous requests to the Kubelet server.

Rationale:

When enabled, requests that are not rejected by other configured authentication methods are treated as anonymous requests. These requests are then served by the Kubelet server. You should rely on authentication to authorize access and disallow anonymous requests.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `authentication: anonymous: enabled` set to `false`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet

config file.
Open the Kubelet config file:

```
sudo more /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `"authentication": { "anonymous": { "enabled": false }` argument is set to `false`.

Audit Method 2:

If using the api configz endpoint consider searching for the status of `authentication... "anonymous":{"enabled":false}` by extracting the live configuration from the nodes running kubelet.

Set the local proxy port and the following variables and provide proxy port number and node name;

`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:

If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to false

```
"authentication": { "anonymous": { "enabled": false
```

Remediation Method 2:

If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--anonymous-auth=false
```

Remediation Method 3:

If using the api configz endpoint consider searching for the status of `"authentication.*anonymous":{"enabled":false}"` by extracting the live configuration from the nodes running kubelet.

**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live

Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Anonymous requests will be rejected.

Default Value:

See the EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://kubernetes.io/docs/admin/kubelet-authentication-authorization/#kubelet-authentication
3. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

Version 7

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

3.2.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Automated)

Profile Applicability:

- Level 1

Description:

Do not allow all requests. Enable explicit authorization.

Rationale:

Kubelets, by default, allow all authenticated requests (even anonymous ones) without needing explicit authorization checks from the apiserver. You should restrict this behavior and only allow explicitly authorized requests.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `"authentication": "webhook": "enabled"` set to `true`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
sudo more /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `"authentication": {"webhook": { "enabled": is set to true`.
If the `"authentication": {"mode": {` argument is present check that it is not set to `AlwaysAllow`. If it is not present check that there is a Kubelet config file specified by `--config`, and that file sets `"authentication": {"mode": {` to something other than `AlwaysAllow`.
Audit Method 2:
If using the api configz endpoint consider searching for the status of `authentication... "webhook":{"enabled":true}` by extracting the live configuration from the nodes running

kubelet.
Set the local proxy port and the following variables and provide proxy port number and node name;
`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from "kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to false

```
"authentication"... "webhook":{"enabled":true
```

Remediation Method 2:
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--authorization-mode=Webhook
```

Remediation Method 3:
If using the api configz endpoint consider searching for the status of `"authentication.*webhook":{"enabled":true"` by extracting the live configuration from the nodes running kubelet.
**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from "kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Unauthorized requests will be denied.

Default Value:

See the EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://kubernetes.io/docs/admin/kubelet-authentication-authorization/#kubelet-authentication
3. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

Version 7

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

3.2.3 Ensure that the --client-ca-file argument is set as appropriate (Manual)

Profile Applicability:

- Level 1

Description:

Enable Kubelet authentication using certificates.

Rationale:

The connections from the apiserver to the kubelet are used for fetching logs for pods, attaching (through kubectl) to running pods, and using the kubelet's port-forwarding functionality. These connections terminate at the kubelet's HTTPS endpoint. By default, the apiserver does not verify the kubelet's serving certificate, which makes the connection subject to man-in-the-middle attacks, and unsafe to run over untrusted and/or public networks. Enabling Kubelet certificate authentication ensures that the apiserver could authenticate the Kubelet before submitting any requests.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `"x509": {"clientCAFile:"` set to the location of the client certificate authority file.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
sudo more /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `"x509": {"clientCAFile:"` argument exists and is set to the location of the client certificate authority file.
If the `"x509": {"clientCAFile:"` argument is not present, check that there is a Kubelet config file specified by `--config`, and that the file sets `"authentication": { "x509": {"clientCAFile:"` to the location of the client certificate authority file.

Audit Method 2:

If using the api configz endpoint consider searching for the status of `authentication.. x509":("clientCAFile":"/etc/kubernetes/pki/ca.crt` by extracting the live configuration from the nodes running kubelet.

Set the local proxy port and the following variables and provide proxy port number and node name;

`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:

If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to false

```
"authentication": { "x509": {"clientCAFile:" to the location of the client CA
file.
```

Remediation Method 2:

If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--client-ca-file=<path/to/client-ca-file>
```

Remediation Method 3:

If using the api configz endpoint consider searching for the status of `"authentication.*x509":("clientCAFile":"/etc/kubernetes/pki/ca.crt"` by extracting the live configuration from the nodes running kubelet.

**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
```

```
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

You require TLS to be configured on apiserver as well as kubelets.

Default Value:

See the EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-authentication-authorization/
3. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

3.2.4 Ensure that the --read-only-port is secured (Manual)

Profile Applicability:

- Level 1

Description:

Disable the read-only port.

Rationale:

The Kubelet process provides a read-only API in addition to the main Kubelet API. Unauthenticated access is provided to this read-only API which could possibly retrieve potentially sensitive information about the cluster.

Audit:

If using a Kubelet configuration file, check that there is an entry for `authentication: anonymous: enabled` set to `0`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `--read-only-port` argument exists and is set to `0`.
If the `--read-only-port` argument is not present, check that there is a Kubelet config file specified by `--config`. Check that if there is a `readOnlyPort` entry in the file, it is set to `0`.

Remediation:

If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to false

```
readOnlyPort to 0
```

If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--read-only-port=0
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Removal of the read-only port will require that any service which made use of it will need to be re-configured to use the main Kubelet API.

Default Value:

See the Amazon EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services
Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running
Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

3.2.5 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Manual)

Profile Applicability:

- Level 1

Description:

Do not disable timeouts on streaming connections.

Rationale:

Setting idle timeouts ensures that you are protected against Denial-of-Service attacks, inactive connections and running out of ephemeral ports.

Note: By default, `--streaming-connection-idle-timeout` is set to 4 hours which might be too high for your environment. Setting this as appropriate would additionally ensure that such streaming connections are timed out after serving legitimate use cases.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `streamingConnectionIdleTimeout` is not set to `0`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `streamingConnectionIdleTimeout` argument is not set to `0`.
If the argument is not present, and there is a Kubelet config file specified by `--config`, check that it does not set `streamingConnectionIdleTimeout` to 0.
Audit Method 2:
If using the api configz endpoint consider searching for the status of `"streamingConnectionIdleTimeout":"4h0m0s"` by extracting the live configuration from

the nodes running kubelet.
Set the local proxy port and the following variables and provide proxy port number and node name;
`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to a non-zero value in the format of #h#m#s

```
"streamingConnectionIdleTimeout":
```

Remediation Method 2:
If using a Kubelet config file, edit the file to set `streamingConnectionIdleTimeout` to a value other than 0.
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--streaming-connection-idle-timeout=4h0m0s
```

Remediation Method 3:
If using the api configz endpoint consider searching for the status of `"streamingConnectionIdleTimeout":` by extracting the live configuration from the nodes running kubelet.
**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")
```

```
curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Long-lived connections could be interrupted.

Default Value:

See the EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://github.com/kubernetes/kubernetes/pull/18552

CIS Controls:

Version 6

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

Version 7

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

3.2.6 Ensure that the --protect-kernel-defaults argument is set to true (Automated)

Profile Applicability:

- Level 1

Description:

Protect tuned kernel parameters from overriding kubelet default kernel parameter values.

Rationale:

Kernel parameters are usually tuned and hardened by the system administrators before putting the systems into production. These parameters protect the kernel and the system. Your kubelet kernel defaults that rely on such parameters should be appropriately set to match the desired secured system state. Ignoring this could potentially lead to running pods with undesired kernel behavior.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `protectKernelDefaults` is set to `true`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `--protect-kernel-defaults=true`.
If the `--protect-kernel-defaults` argument is not present, check that there is a Kubelet config file specified by `--config`, and that the file sets `protectKernelDefaults` to `true`.
Audit Method 2:
If using the api configz endpoint consider searching for the status of `"protectKernelDefaults"` by extracting the live configuration from the nodes running kubelet.

Set the local proxy port and the following variables and provide proxy port number and node name;
`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to true

```
"protectKernelDefaults":
```

Remediation Method 2:
If using a Kubelet config file, edit the file to set `protectKernelDefaults` to `true`.
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
----protect-kernel-defaults=true
```

Remediation Method 3:
If using the api configz endpoint consider searching for the status of `"protectKernelDefaults":` by extracting the live configuration from the nodes running kubelet.
**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

You would have to re-tune kernel parameters to match kubelet parameters.

Default Value:

See the EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/

CIS Controls:

Version 6

3 Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers
Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Version 7

5.2 Maintain Secure Images
Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

3.2.7 Ensure that the --make-iptables-util-chains argument is set to true (Automated)

Profile Applicability:

- Level 1

Description:

Allow Kubelet to manage iptables.

Rationale:

Kubelets can automatically manage the required changes to iptables based on how you choose your networking options for the pods. It is recommended to let kubelets manage the changes to iptables. This ensures that the iptables configuration remains in sync with pods networking configuration. Manually configuring iptables with dynamic pod network configuration changes might hamper the communication between pods/containers and to the outside world. You might have iptables rules too restrictive or too open.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `makeIPTablesUtilChains` set to `true`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that if the `makeIPTablesUtilChains` argument exists then it is set to `true`.
If the `--make-iptables-util-chains` argument does not exist, and there is a Kubelet config file specified by `--config`, verify that the file does not set `makeIPTablesUtilChains` to `false`.
Audit Method 2:
If using the api configz endpoint consider searching for the status of `authentication...`

`"makeIPTablesUtilChains":true` by extracting the live configuration from the nodes running kubelet.
Set the local proxy port and the following variables and provide proxy port number and node name;
`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to false

```
"makeIPTablesUtilChains": true
```

Remediation Method 2:
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--make-iptables-util-chains:true
```

Remediation Method 3:
If using the api configz endpoint consider searching for the status of `"makeIPTablesUtilChains": true` by extracting the live configuration from the nodes running kubelet.
**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Kubelet would manage the iptables on the system and keep it in sync. If you are using any other iptables management solution, then there might be some conflicts.

Default Value:

See the Amazon EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

Version 7

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

3.2.8 Ensure that the --hostname-override argument is not set (Manual)

Profile Applicability:

- Level 1

Description:

Do not override node hostnames.

Rationale:

Overriding hostnames could potentially break TLS setup between the kubelet and the apiserver. Additionally, with overridden hostnames, it becomes increasingly difficult to associate logs with a particular node and process them for security analytics. Hence, you should setup your kubelet nodes with resolvable FQDNs and avoid overriding the hostnames with IPs.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `--hostname-override` is not set or does not exist.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that `--hostname-override` argument does not exist.
Note This setting is not configurable via the Kubelet config file.

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to null

```
"hostname-override"
```

Remediation Method 2:
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--hostname-override
```

For all remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Some cloud providers may require this flag to ensure that hostname matches names issued by the cloud provider. In these environments, this recommendation should not apply.

Default Value:

See the Amazon EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://github.com/kubernetes/kubernetes/issues/22063
3. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

3 Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Version 7

5 Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers

Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers

3.2.9 Ensure that the --eventRecordQPS argument is set to 0 or a level which ensures appropriate event capture (Automated)

Profile Applicability:

- Level 2

Description:

Security relevant information should be captured. The `--eventRecordQPS` flag on the Kubelet can be used to limit the rate at which events are gathered. Setting this too low could result in relevant events not being logged, however the unlimited setting of `0` could result in a denial of service on the kubelet.

Rationale:

It is important to capture all events and not restrict event creation. Events are an important source of security information and analytics that ensure that your environment is consistently monitored using the event data.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `eventRecordQPS` set to `5` or a value equal to or greater than 0.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Review the value set for the `--eventRecordQPS` argument and determine whether this has been set to an appropriate level for the cluster. The value of `0` can be used to ensure that all events are captured.
If the `--eventRecordQPS` argument does not exist, check that there is a Kubelet config file specified by `--config` and review the value in this location.
Audit Method 2:

If using the api configz endpoint consider searching for the status of `eventRecordQPS` by extracting the live configuration from the nodes running kubelet.
Set the local proxy port and the following variables and provide proxy port number and node name;
`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to 5 or a value greater or equal to 0

```
"eventRecordQPS": 5
```

Remediation Method 2:
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--eventRecordQPS=5
```

Remediation Method 3:
If using the api configz endpoint consider searching for the status of `"eventRecordQPS"` by extracting the live configuration from the nodes running kubelet.
**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from
"kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

Setting this parameter to `0` could result in a denial of service condition due to excessive events being created. The cluster's event processing and storage systems should be scaled to handle expected event loads.

Default Value:

See the Amazon EKS documentation for the default value.

References:

1. https://kubernetes.io/docs/admin/kubelet/
2. https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/apis/kubeletconfig/v1beta1/types.go
3. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

6 Maintenance, Monitoring, and Analysis of Audit Logs
Maintenance, Monitoring, and Analysis of Audit Logs

Version 7

6 Maintenance, Monitoring and Analysis of Audit Logs
Maintenance, Monitoring and Analysis of Audit Logs

3.2.10 Ensure that the --rotate-certificates argument is not set to false (Manual)

Profile Applicability:

- Level 2

Description:

Enable kubelet client certificate rotation.

Rationale:

The `--rotate-certificates` setting causes the kubelet to rotate its client certificates by creating new CSRs as its existing credentials expire. This automated periodic rotation ensures that the there is no downtime due to expired certificates and thus addressing availability in the CIA security triad.

Note: This recommendation only applies if you let kubelets get their certificates from the API server. In case your kubelet certificates come from an outside authority/tool (e.g. Vault) then you need to take care of rotation yourself.

Note: This feature also require the `RotateKubeletClientCertificate` feature gate to be enabled (which is the default since Kubernetes v1.7)

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `--rotate-certificates` set to `false`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that the `RotateCertificate` argument is not present, or is set to `true`.
If the `--rotate-certificates` argument is not present, verify that if there is a Kubelet config file specified by `--config`, that file does not contain `rotateCertificates: false`.

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to false

```
"RotateCertificate":true
```

Remediation Method 2:
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--RotateCertificate=true
```

Impact:

None

Default Value:

See the Amazon EKS documentation for the default value.

References:

1. https://github.com/kubernetes/kubernetes/pull/41912
2. https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet-tls-bootstrapping/#kubelet-configuration
3. https://kubernetes.io/docs/imported/release/notes/
4. https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/
5. https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

14.4 Encrypt All Sensitive Information in Transit

Encrypt all sensitive information in transit.

3.2.11 Ensure that the RotateKubeletServerCertificate argument is set to true (Manual)

Profile Applicability:

- Level 1

Description:

Enable kubelet server certificate rotation.

Rationale:

`RotateKubeletServerCertificate` causes the kubelet to both request a serving certificate after bootstrapping its client credentials and rotate the certificate as its existing credentials expire. This automated periodic rotation ensures that the there are no downtimes due to expired certificates and thus addressing availability in the CIA security triad.

Note: This recommendation only applies if you let kubelets get their certificates from the API server. In case your kubelet certificates come from an outside authority/tool (e.g. Vault) then you need to take care of rotation yourself.

Audit:

Audit Method 1:
If using a Kubelet configuration file, check that there is an entry for `RotateKubeletServerCertificate` is set to `true`.
First, SSH to the relevant node:
Run the following command on each node to find the appropriate Kubelet config file:

```
ps -ef | grep kubelet
```

The output of the above command should return something similar to `--config /etc/kubernetes/kubelet/kubelet-config.json` which is the location of the Kubelet config file.
Open the Kubelet config file:

```
cat /etc/kubernetes/kubelet/kubelet-config.json
```

Verify that `RotateKubeletServerCertificate` argument exists and is set to `true`.
Audit Method 2:
If using the api configz endpoint consider searching for the status of `"RotateKubeletServerCertificate":true` by extracting the live configuration from the

nodes running kubelet.
Set the local proxy port and the following variables and provide proxy port number and node name;
`HOSTNAME_PORT="localhost-and-port-number"`
`NODE_NAME="The-Name-Of-Node-To-Extract-Configuration" from the output of "kubectl get nodes"`

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from "kubectl get nodes")

curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

Remediation:

Remediation Method 1:
If modifying the Kubelet config file, edit the kubelet-config.json file `/etc/kubernetes/kubelet/kubelet-config.json` and set the below parameter to true

```
"RotateKubeletServerCertificate":true
```

Remediation Method 2:
If using a Kubelet config file, edit the file to set `RotateKubeletServerCertificate to true`.
If using executable arguments, edit the kubelet service file `/etc/systemd/system/kubelet.service.d/10-kubelet-args.conf` on each worker node and add the below parameter at the end of the `KUBELET_ARGS` variable string.

```
--rotate-kubelet-server-certificate=true
```

Remediation Method 3:
If using the api configz endpoint consider searching for the status of `"RotateKubeletServerCertificate":` by extracting the live configuration from the nodes running kubelet.
**See detailed step-by-step configmap procedures in Reconfigure a Node's Kubelet in a Live Cluster, and then rerun the curl statement from audit process to check for kubelet configuration changes

```
kubectl proxy --port=8001 &

export HOSTNAME_PORT=localhost:8001 (example host and port number)
export NODE_NAME=ip-192.168.31.226.ec2.internal (example node name from "kubectl get nodes")
```

```
curl -sSL "http://${HOSTNAME_PORT}/api/v1/nodes/${NODE_NAME}/proxy/configz"
```

For all three remediations:
Based on your system, restart the `kubelet` service and check status

```
systemctl daemon-reload
systemctl restart kubelet.service
systemctl status kubelet -l
```

Impact:

None

Default Value:

See the Amazon EKS documentation for the default value.

References:

1. https://github.com/kubernetes/kubernetes/pull/45059
2. https://kubernetes.io/docs/admin/kubelet-tls-bootstrapping/#kubelet-configuration

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

4 Policies

This section contains recommendations for various Kubernetes policies which are important to the security of Amazon EKS customer environment.

4.1 RBAC and Service Accounts

4.1.1 Ensure that the cluster-admin role is only used where required (Manual)

Profile Applicability:

- Level 1

Description:

The RBAC role `cluster-admin` provides wide-ranging powers over the environment and should be used only where and when needed.

Rationale:

Kubernetes provides a set of default roles where RBAC is used. Some of these roles such as `cluster-admin` provide wide-ranging privileges which should only be applied where absolutely necessary. Roles such as `cluster-admin` allow super-user access to perform any action on any resource. When used in a `ClusterRoleBinding`, it gives full control over every resource in the cluster and in all namespaces. When used in a `RoleBinding`, it gives full control over every resource in the rolebinding's namespace, including the namespace itself.

Audit:

Obtain a list of the principals who have access to the `cluster-admin` role by reviewing the `clusterrolebinding` output for each role binding that has access to the `cluster-admin` role.
kubectl get clusterrolebindings -o=custom-columns=NAME:.metadata.name,ROLE:.roleRef.name,SUBJECT:.subjects[*].name
Review each principal listed and ensure that `cluster-admin` privilege is required for it.

Remediation:

Identify all clusterrolebindings to the cluster-admin role. Check if they are used and if they need this role or if they could use a role with fewer privileges.
Where possible, first bind users to a lower privileged role and then remove the clusterrolebinding to the cluster-admin role :

```
kubectl delete clusterrolebinding [name]
```

Impact:

Care should be taken before removing any `clusterrolebindings` from the environment to ensure they were not required for operation of the cluster. Specifically, modifications should not be made to `clusterrolebindings` with the `system:` prefix as they are required for the operation of system components.

Default Value:

By default a single `clusterrolebinding` called `cluster-admin` is provided with the `system:masters` group as its principal.

References:

1. https://kubernetes.io/docs/admin/authorization/rbac/#user-facing-roles

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.1 Establish Secure Configurations

Maintain documented, standard security configuration standards for all authorized operating systems and software.

4.1.2 Minimize access to secrets (Manual)

Profile Applicability:

- Level 1

Description:

The Kubernetes API stores secrets, which may be service account tokens for the Kubernetes API or credentials used by workloads in the cluster. Access to these secrets should be restricted to the smallest possible group of users to reduce the risk of privilege escalation.

Rationale:

Inappropriate access to secrets stored within the Kubernetes cluster can allow for an attacker to gain additional access to the Kubernetes cluster or external resources whose credentials are stored as secrets.

Audit:

Review the users who have `get`, `list` or `watch` access to `secrets` objects in the Kubernetes API.

Remediation:

Where possible, remove `get`, `list` and `watch` access to `secret` objects in the cluster.

Impact:

Care should be taken not to remove access to secrets to system components which require this for their operation

Default Value:

By default, the following list of principals have `get` privileges on `secret` objects

```
CLUSTERROLEBINDING                                          SUBJECT
TYPE            SA-NAMESPACE

cluster-admin                                               system:masters
Group

system:controller:clusterrole-aggregation-controller  clusterrole-
aggregation-controller  ServiceAccount  kube-system
```

```
system:controller:expand-controller                    expand-controller
ServiceAccount  kube-system

system:controller:generic-garbage-collector            generic-garbage-
collector           ServiceAccount  kube-system

system:controller:namespace-controller                 namespace-controller
ServiceAccount  kube-system

system:controller:persistent-volume-binder             persistent-volume-
binder            ServiceAccount  kube-system

system:kube-controller-manager                         system:kube-controller-
manager      User
```

CIS Controls:

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.1.3 Minimize wildcard use in Roles and ClusterRoles (Manual)

Profile Applicability:

- Level 1

Description:

Kubernetes Roles and ClusterRoles provide access to resources based on sets of objects and actions that can be taken on those objects. It is possible to set either of these to be the wildcard "*" which matches all items.

Use of wildcards is not optimal from a security perspective as it may allow for inadvertent access to be granted when new resources are added to the Kubernetes API either as CRDs or in later versions of the product.

Rationale:

The principle of least privilege recommends that users are provided only the access required for their role and nothing more. The use of wildcard rights grants is likely to provide excessive rights to the Kubernetes API.

Audit:

Retrieve the roles defined across each namespaces in the cluster and review for wildcards

```
kubectl get roles --all-namespaces -o yaml
```

Retrieve the cluster roles defined in the cluster and review for wildcards

```
kubectl get clusterroles -o yaml
```

Remediation:

Where possible replace any use of wildcards in clusterroles and roles with specific objects or actions.

CIS Controls:

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

4.1.4 Minimize access to create pods (Manual)

Profile Applicability:

- Level 1

Description:

The ability to create pods in a namespace can provide a number of opportunities for privilege escalation, such as assigning privileged service accounts to these pods or mounting hostPaths with access to sensitive data (unless Pod Security Policies are implemented to restrict this access)

As such, access to create new pods should be restricted to the smallest possible group of users.

Rationale:

The ability to create pods in a cluster opens up possibilities for privilege escalation and should be restricted, where possible.

Audit:

Review the users who have create access to pod objects in the Kubernetes API.

Remediation:

Where possible, remove `create` access to `pod` objects in the cluster.

Impact:

Care should be taken not to remove access to pods to system components which require this for their operation

Default Value:

By default, the following list of principals have `create` privileges on `pod` objects

```
CLUSTERROLEBINDING                                      SUBJECT
TYPE            SA-NAMESPACE

cluster-admin                                           system:masters
Group

system:controller:clusterrole-aggregation-controller  clusterrole-
aggregation-controller  ServiceAccount  kube-system
```

```
system:controller:daemon-set-controller                daemon-set-controller
ServiceAccount  kube-system

system:controller:job-controller                       job-controller
ServiceAccount  kube-system

system:controller:persistent-volume-binder             persistent-volume-
binder            ServiceAccount  kube-system

system:controller:replicaset-controller                replicaset-controller
ServiceAccount  kube-system

system:controller:replication-controller               replication-controller
ServiceAccount  kube-system

system:controller:statefulset-controller               statefulset-controller
ServiceAccount  kube-system
```

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.1.5 Ensure that default service accounts are not actively used. (Manual)

Profile Applicability:

- Level 1

Description:

The `default` service account should not be used to ensure that rights granted to applications can be more easily audited and reviewed.

Rationale:

Kubernetes provides a `default` service account which is used by cluster workloads where no specific service account is assigned to the pod.

Where access to the Kubernetes API from a pod is required, a specific service account should be created for that pod, and rights granted to that service account.

The default service account should be configured such that it does not provide a service account token and does not have any explicit rights assignments.

Audit:

For each namespace in the cluster, review the rights assigned to the default service account and ensure that it has no roles or cluster roles bound to it apart from the defaults.
Additionally ensure that the `automountServiceAccountToken: false` setting is in place for each default service account.

Remediation:

Create explicit service accounts wherever a Kubernetes workload requires specific access to the Kubernetes API server.
Modify the configuration of each default service account to include this value

```
automountServiceAccountToken: false
```

Automatic remediation for the default account:

```
kubectl patch serviceaccount default -p $'automountServiceAccountToken:
false'
```

Impact:

All workloads which require access to the Kubernetes API will require an explicit service account to be created.

Default Value:

By default the `default` service account allows for its service account token to be mounted in pods in its namespace.

References:

1. https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
2. https://aws.github.io/aws-eks-best-practices/iam/#disable-auto-mounting-of-service-account-tokens

CIS Controls:

Version 7

4.3 Ensure the Use of Dedicated Administrative Accounts

Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.1.6 Ensure that Service Account Tokens are only mounted where necessary (Manual)

Profile Applicability:

- Level 1

Description:

Service accounts tokens should not be mounted in pods except where the workload running in the pod explicitly needs to communicate with the API server

Rationale:

Mounting service account tokens inside pods can provide an avenue for privilege escalation attacks where an attacker is able to compromise a single pod in the cluster.

Avoiding mounting these tokens removes this attack avenue.

Audit:

Review pod and service account objects in the cluster and ensure that the option below is set, unless the resource explicitly requires this access.

```
automountServiceAccountToken: false
```

Remediation:

Modify the definition of pods and service accounts which do not need to mount service account tokens to disable it.

Impact:

Pods mounted without service account tokens will not be able to communicate with the API server, except where the resource is available to unauthenticated principals.

Default Value:

By default, all pods get a service account token mounted in them.

References:

1. https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2 Pod Security Policies

A Pod Security Policy (PSP) is a cluster-level resource that controls security settings for pods. Your cluster may have multiple PSPs. You can query PSPs with the following command:

```
kubectl get psp
```

PodSecurityPolicies are used in conjunction with the PodSecurityPolicy admission controller plugin.

4.2.1 Minimize the admission of privileged containers (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers to be run with the `securityContext.privileged` flag set to `true`.

Rationale:

Privileged containers have access to all Linux Kernel capabilities and devices. A container running with full privileges can do almost everything that the host can do. This flag exists to allow special use-cases, like manipulating the network stack and accessing devices.

There should be at least one PodSecurityPolicy (PSP) defined which does not permit privileged containers.

If you need to run privileged containers, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether privileged is enabled:

```
kubectl get psp -o json
```

Verify that there is at least one PSP which does not return `true`.

```
kubectl get psp <name> -o=jsonpath='{.spec.privileged}'
```

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.privileged` field is omitted or set to `false`.

Impact:

Pods defined with `spec.containers[].securityContext.privileged: true` will not be permitted.

Default Value:

By default, when you provision an EKS cluster, a pod security policy called `eks.privileged` is automatically created. The manifest for that policy appears below:

```
apiVersion: extensions/v1beta1

kind: PodSecurityPolicy

metadata:

  annotations:

    kubernetes.io/description: privileged allows full unrestricted access to
pod features,

      as if the PodSecurityPolicy controller was not enabled.

    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'

  labels:

    eks.amazonaws.com/component: pod-security-policy

    kubernetes.io/cluster-service: "true"

  name: eks.privileged

spec:

  allowPrivilegeEscalation: true

  allowedCapabilities:

  - '*'

  fsGroup:
```

```
    rule: RunAsAny
  hostIPC: true
  hostNetwork: true
  hostPID: true
  hostPorts:
  - max: 65535
    min: 0
  privileged: true
  runAsUser:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - '*'
```

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy/#enabling-pod-security-policies
2. https://aws.github.io/aws-eks-best-practices/pods/#restrict-the-containers-that-can-run-as-privileged

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.2 Minimize the admission of containers wishing to share the host process ID namespace (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers to be run with the `hostPID` flag set to true.

Rationale:

A container running in the host's PID namespace can inspect processes running outside the container. If the container also has access to ptrace capabilities this can be used to escalate privileges outside of the container.

There should be at least one PodSecurityPolicy (PSP) defined which does not permit containers to share the host PID namespace.

If you need to run containers which require hostPID, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether privileged is enabled:

```
kubectl get psp <name> -o=jsonpath='{.spec.hostPID}'
```

Verify that there is at least one PSP which does not return true.

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.hostPID` field is omitted or set to false.

Impact:

Pods defined with `spec.hostPID: true` will not be permitted unless they are run under a specific PSP.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.3 Minimize the admission of containers wishing to share the host IPC namespace (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers to be run with the `hostIPC` flag set to true.

Rationale:

A container running in the host's IPC namespace can use IPC to interact with processes outside the container.

There should be at least one PodSecurityPolicy (PSP) defined which does not permit containers to share the host IPC namespace.

If you have a requirement to containers which require hostIPC, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether privileged is enabled:

```
kubectl get psp <name> -o=jsonpath='{.spec.hostIPC}'
```

Verify that there is at least one PSP which does not return true.

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.hostIPC` field is omitted or set to false.

Impact:

Pods defined with `spec.hostIPC: true` will not be permitted unless they are run under a specific PSP.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.4 Minimize the admission of containers wishing to share the host network namespace (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers to be run with the `hostNetwork` flag set to true.

Rationale:

A container running in the host's network namespace could access the local loopback device, and could access network traffic to and from other pods.

There should be at least one PodSecurityPolicy (PSP) defined which does not permit containers to share the host network namespace.

If you have need to run containers which require hostNetwork, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether privileged is enabled:

```
kubectl get psp <name> -o=jsonpath='{.spec.hostNetwork}'
```

Verify that there is at least one PSP which does not return true.

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.hostNetwork` field is omitted or set to false.

Impact:

Pods defined with `spec.hostNetwork: true` will not be permitted unless they are run under a specific PSP.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.5 Minimize the admission of containers with allowPrivilegeEscalation (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers to be run with the `allowPrivilegeEscalation` flag set to true.

Rationale:

A container running with the `allowPrivilegeEscalation` flag set to `true` may have processes that can gain more privileges than their parent.

There should be at least one PodSecurityPolicy (PSP) defined which does not permit containers to allow privilege escalation. The option exists (and is defaulted to true) to permit setuid binaries to run.

If you have need to run containers which use setuid binaries or require privilege escalation, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether privileged is enabled:

```
kubectl get psp <name> -o=jsonpath='{.spec.allowPrivilegeEscalation}'
```

Verify that there is at least one PSP which does not return true.

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.allowPrivilegeEscalation` field is omitted or set to false.

Impact:

Pods defined with `spec.allowPrivilegeEscalation: true` will not be permitted unless they are run under a specific PSP.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.6 Minimize the admission of root containers (Automated)

Profile Applicability:

- Level 2

Description:

Do not generally permit containers to be run as the root user.

Rationale:

Containers may run as any Linux user. Containers which run as the root user, whilst constrained by Container Runtime security features still have a escalated likelihood of container breakout.

Ideally, all containers should run as a defined non-UID 0 user.

There should be at least one PodSecurityPolicy (PSP) defined which does not permit root users in a container.

If you need to run root containers, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether running containers as root is enabled:

```
kubectl get psp <name> -o=jsonpath='{.spec.runAsUser.rule}'
```

Verify that there is at least one PSP which returns `MustRunAsNonRoot` or `MustRunAs` with the range of UIDs not including 0.

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.runAsUser.rule` is set to either `MustRunAsNonRoot` or `MustRunAs` with the range of UIDs not including 0.

Impact:

Pods with containers which run as the root user will not be permitted.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy/#enabling-pod-security-policies

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.7 Minimize the admission of containers with the NET_RAW capability (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers with the potentially dangerous NET_RAW capability.

Rationale:

Containers run with a default set of capabilities as assigned by the Container Runtime. By default this can include potentially dangerous capabilities. With Docker as the container runtime the NET_RAW capability is enabled which may be misused by malicious containers.

Ideally, all containers should drop this capability.

There should be at least one PodSecurityPolicy (PSP) defined which prevents containers with the NET_RAW capability from launching.

If you need to run containers with this capability, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether NET_RAW is disabled:

```
kubectl get psp <name> -o=jsonpath='{.spec.requiredDropCapabilities}'
```

Verify that there is at least one PSP which returns NET_RAW or ALL.

Remediation:

Create a PSP as described in the Kubernetes documentation, ensuring that the `.spec.requiredDropCapabilities` is set to include either `NET_RAW` or `ALL`.

Impact:

Pods with containers which run with the NET_RAW capability will not be permitted.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy/#enabling-pod-security-policies
2. https://www.nccgroup.trust/uk/our-research/abusing-privileged-and-unprivileged-linux-containers/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.8 Minimize the admission of containers with added capabilities (Automated)

Profile Applicability:

- Level 1

Description:

Do not generally permit containers with capabilities assigned beyond the default set.

Rationale:

Containers run with a default set of capabilities as assigned by the Container Runtime. Capabilities outside this set can be added to containers which could expose them to risks of container breakout attacks.

There should be at least one PodSecurityPolicy (PSP) defined which prevents containers with capabilities beyond the default set from launching.

If you need to run containers with additional capabilities, this should be defined in a separate PSP and you should carefully check RBAC controls to ensure that only limited service accounts and users are given permission to access that PSP.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

Verify that there are no PSPs present which have `allowedCapabilities` set to anything other than an empty array.

Remediation:

Ensure that `allowedCapabilities` is not present in PSPs for the cluster unless it is set to an empty array.

Impact:

Pods with containers which require capabilities outwith the default set will not be permitted.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy/#enabling-pod-security-policies
2. https://www.nccgroup.trust/uk/our-research/abusing-privileged-and-unprivileged-linux-containers/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.2.9 Minimize the admission of containers with capabilities assigned (Manual)

Profile Applicability:

- Level 2

Description:

Do not generally permit containers with capabilities

Rationale:

Containers run with a default set of capabilities as assigned by the Container Runtime. Capabilities are parts of the rights generally granted on a Linux system to the root user.

In many cases applications running in containers do not require any capabilities to operate, so from the perspective of the principal of least privilege use of capabilities should be minimized.

Audit:

Get the set of PSPs with the following command:

```
kubectl get psp
```

For each PSP, check whether capabilities have been forbidden:

```
kubectl get psp <name> -o=jsonpath='{.spec.requiredDropCapabilities}'
```

Remediation:

Review the use of capabilites in applications runnning on your cluster. Where a namespace contains applicaions which do not require any Linux capabities to operate consider adding a PSP which forbids the admission of containers which do not drop all capabilities.

Impact:

Pods with containers require capabilities to operate will not be permitted.

Default Value:

By default, PodSecurityPolicies are not defined.

References:

1. https://kubernetes.io/docs/concepts/policy/pod-security-policy/#enabling-pod-security-policies
2. https://www.nccgroup.trust/uk/our-research/abusing-privileged-and-unprivileged-linux-containers/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Version 7

5.2 Maintain Secure Images

Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

4.3 CNI Plugin

4.3.1 Ensure latest CNI version is used (Manual)

Profile Applicability:

- Level 1

Description:

There are a variety of CNI plugins available for Kubernetes. If the CNI in use does not support Network Policies it may not be possible to effectively restrict traffic in the cluster.

Rationale:

Kubernetes network policies are enforced by the CNI plugin in use. As such it is important to ensure that the CNI plugin supports both Ingress and Egress network policies.

Audit:

Review the documentation of CNI plugin in use by the cluster, and confirm that it supports network policies.

Remediation:

As with RBAC policies, network policies should adhere to the policy of least privileged access. Start by creating a deny all policy that restricts all inbound and outbound traffic from a namespace or create a global policy using Calico.

Impact:

None.

Default Value:

This will depend on the CNI plugin in use.

References:

1. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/
2. https://aws.github.io/aws-eks-best-practices/network/

Notes:

One example here is Flannel (https://github.com/coreos/flannel) which does not support Network policy unless Calico is also in use.

CIS Controls:

Version 7

18.4 Only Use Up-to-date And Trusted Third-Party Components

Only use up-to-date and trusted third-party components for the software developed by the organization.

4.3.2 Ensure that all Namespaces have Network Policies defined (Automated)

Profile Applicability:

- Level 2

Description:

Use network policies to isolate traffic in your cluster network.

Rationale:

Running different applications on the same Kubernetes cluster creates a risk of one compromised application attacking a neighboring application. Network segmentation is important to ensure that containers can communicate only with those they are supposed to. A network policy is a specification of how selections of pods are allowed to communicate with each other and other network endpoints.

Network Policies are namespace scoped. When a network policy is introduced to a given namespace, all traffic not allowed by the policy is denied. However, if there are no network policies in a namespace all traffic will be allowed into and out of the pods in that namespace.

Audit:

Run the below command and review the `NetworkPolicy` objects created in the cluster.

```
kubectl get networkpolicy --all-namespaces
```

Ensure that each namespace defined in the cluster has at least one Network Policy.

Remediation:

Follow the documentation and create `NetworkPolicy` objects as you need them.

Impact:

Once network policies are in use within a given namespace, traffic not explicitly allowed by a network policy will be denied. As such it is important to ensure that, when introducing network policies, legitimate traffic is not blocked.

Default Value:

By default, network policies are not created.

References:

1. https://kubernetes.io/docs/concepts/services-networking/networkpolicies/
2. https://octetz.com/posts/k8s-network-policy-apis
3. https://kubernetes.io/docs/tasks/configure-pod-container/declare-network-policy/

CIS Controls:

Version 6

14.1 Implement Network Segmentation Based On Information Class

Segment the network based on the label or classification level of the information stored on the servers. Locate all sensitive information on separated VLANS with firewall filtering to ensure that only authorized individuals are only able to communicate with systems necessary to fulfill their specific responsibilities.

Version 7

14.1 Segment the Network Based on Sensitivity

Segment the network based on the label or classification level of the information stored on the servers, locate all sensitive information on separated Virtual Local Area Networks (VLANs).

14.2 Enable Firewall Filtering Between VLANs

Enable firewall filtering between VLANs to ensure that only authorized systems are able to communicate with other systems necessary to fulfill their specific responsibilities.

4.4 Secrets Management

4.4.1 Prefer using secrets as files over secrets as environment variables (Manual)

Profile Applicability:

- Level 2

Description:

Kubernetes supports mounting secrets as data volumes or as environment variables. Minimize the use of environment variable secrets.

Rationale:

It is reasonably common for application code to log out its environment (particularly in the event of an error). This will include any secret values passed in as environment variables, so secrets can easily be exposed to any user or entity who has access to the logs.

Audit:

Run the following command to find references to objects which use environment variables defined from secrets.

```
kubectl get all -o jsonpath='{range .items[?(@..secretKeyRef)]} {.kind}
{.metadata.name} {"\n"}{end}' -A
```

Remediation:

If possible, rewrite application code to read secrets from mounted secret files, rather than from environment variables.

Impact:

Application code which expects to read secrets in the form of environment variables would need modification

Default Value:

By default, secrets are not defined

References:

1. https://kubernetes.io/docs/concepts/configuration/secret/#using-secrets

Notes:

Mounting secrets as volumes has the additional benefit that secret values can be updated without restarting the pod

CIS Controls:

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

14.8 Encrypt Sensitive Information at Rest
Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.

4.4.2 Consider external secret storage (Manual)

Profile Applicability:

- Level 2

Description:

Consider the use of an external secrets storage and management system, instead of using Kubernetes Secrets directly, if you have more complex secret management needs. Ensure the solution requires authentication to access secrets, has auditing of access to and use of secrets, and encrypts secrets. Some solutions also make it easier to rotate secrets.

Rationale:

Kubernetes supports secrets as first-class objects, but care needs to be taken to ensure that access to secrets is carefully limited. Using an external secrets provider can ease the management of access to secrets, especially where secrests are used across both Kubernetes and non-Kubernetes environments.

Audit:

Review your secrets management implementation.

Remediation:

Refer to the secrets management options offered by your cloud provider or a third-party secrets management solution.

Impact:

None

Default Value:

By default, no external secret management is configured.

CIS Controls:

Version 7

14.8 Encrypt Sensitive Information at Rest
Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.

4.5 Extensible Admission Control

4.5.1 Configure Image Provenance using ImagePolicyWebhook admission controller (Manual)

Profile Applicability:

- Level 2

Description:

Configure Image Provenance for your deployment.

Rationale:

Kubernetes supports plugging in provenance rules to accept or reject the images in your deployments. You could configure such rules to ensure that only approved images are deployed in the cluster.

Audit:

Review the pod definitions in your cluster and verify that image provenance is configured as appropriate.

Remediation:

Follow the Kubernetes documentation and setup image provenance.

Impact:

You need to regularly maintain your provenance configuration based on container image updates.

Default Value:

By default, image provenance is not set.

References:

1. https://kubernetes.io/docs/admin/admission-controllers/#imagepolicywebhook
2. https://hub.docker.com/r/dnurmi/anchore-toolbox/
3. https://github.com/kubernetes/kubernetes/issues/22888

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

Version 7

18 Application Software Security
Application Software Security

4.6 General Policies

These policies relate to general cluster management topics, like namespace best practices and policies applied to pod objects in the cluster.

4.6.1 Create administrative boundaries between resources using namespaces (Manual)

Profile Applicability:

- Level 1

Description:

Use namespaces to isolate your Kubernetes objects.

Rationale:

Limiting the scope of user permissions can reduce the impact of mistakes or malicious activities. A Kubernetes namespace allows you to partition created resources into logically named groups. Resources created in one namespace can be hidden from other namespaces. By default, each resource created by a user in an Amazon EKS cluster runs in a default namespace, called `default`. You can create additional namespaces and attach resources and users to them. You can use Kubernetes Authorization plugins to create policies that segregate access to namespace resources between different users.

Audit:

Run the below command and review the namespaces created in the cluster.

```
kubectl get namespaces
```

Ensure that these namespaces are the ones you need and are adequately administered as per your requirements.

Remediation:

Follow the documentation and create namespaces for objects in your deployment as you need them.

Impact:

You need to switch between namespaces for administration.

Default Value:

By default, Kubernetes starts with two initial namespaces:

1. `default` - The default namespace for objects with no other namespace
2. `kube-system` - The namespace for objects created by the Kubernetes system
3. `kube-public` - The namespace for public-readable ConfigMap
4. `kube-node-lease` - The namespace for associated lease object for each node

References:

1. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
2. http://blog.kubernetes.io/2016/08/security-best-practices-kubernetes-deployment.html

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

Version 7

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

4.6.2 Apply Security Context to Your Pods and Containers (Manual)

Profile Applicability:

- Level 2

Description:

Apply Security Context to Your Pods and Containers

Rationale:

A security context defines the operating system security settings (uid, gid, capabilities, SELinux role, etc..) applied to a container. When designing your containers and pods, make sure that you configure the security context for your pods, containers, and volumes. A security context is a property defined in the deployment yaml. It controls the security parameters that will be assigned to the pod/container/volume. There are two levels of security context: pod level security context, and container level security context.

Audit:

Review the pod definitions in your cluster and verify that you have security contexts defined as appropriate.

Remediation:

As a best practice we recommend that you scope the binding for privileged pods to service accounts within a particular namespace, e.g. kube-system, and limiting access to that namespace. For all other serviceaccounts/namespaces, we recommend implementing a more restrictive policy such as this:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
    name: restricted
    annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames:
'docker/default,runtime/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames:
'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:
'runtime/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:
'runtime/default'
spec:
    privileged: false
    # Required to prevent escalations to root.
    allowPrivilegeEscalation: false
```

```
    # This is redundant with non-root + disallow privilege escalation,
    # but we can provide it for defense in depth.
    requiredDropCapabilities:
    - ALL
    # Allow core volume types.
    volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin are safe to
use.
    - 'persistentVolumeClaim'
    hostNetwork: false
    hostIPC: false
    hostPID: false
    runAsUser:
    # Require the container to run without root privileges.
    rule: 'MustRunAsNonRoot'
    seLinux:
    # This policy assumes the nodes are using AppArmor rather than SELinux.
    rule: 'RunAsAny'
    supplementalGroups:
    rule: 'MustRunAs'
    ranges:
        # Forbid adding the root group.
        - min: 1
        max: 65535
    fsGroup:
    rule: 'MustRunAs'
    ranges:
        # Forbid adding the root group.
        - min: 1
        max: 65535
    readOnlyRootFilesystem: false
```

This policy prevents pods from running as privileged or escalating privileges. It also restricts the types of volumes that can be mounted and the root supplemental groups that can be added.
Another, albeit similar, approach is to start with policy that locks everything down and incrementally add exceptions for applications that need looser restrictions such as logging agents which need the ability to mount a host path.

Impact:

If you incorrectly apply security contexts, you may have trouble running the pods.

Default Value:

By default, no security contexts are automatically applied to pods.

References:

1. https://kubernetes.io/docs/concepts/policy/security-context/
2. https://learn.cisecurity.org/benchmarks
3. https://aws.github.io/aws-eks-best-practices/pods/#restrict-the-containers-that-can-run-as-privileged

CIS Controls:

Version 6

3 Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Version 7

5 Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers

Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers

4.6.3 The default namespace should not be used (Automated)

Profile Applicability:

- Level 2

Description:

Kubernetes provides a default namespace, where objects are placed if no namespace is specified for them. Placing objects in this namespace makes application of RBAC and other controls more difficult.

Rationale:

Resources in a Kubernetes cluster should be segregated by namespace, to allow for security controls to be applied at that level and to make it easier to manage resources.

Audit:

Run this command to list objects in default namespace

```
kubectl get all -n default
```

The only entries there should be system managed resources such as the `kubernetes` service

Remediation:

Ensure that namespaces are created to allow for appropriate segregation of Kubernetes resources and that all new resources are created in a specific namespace.

Impact:

None

Default Value:

Unless a namespace is specific on object creation, the `default` namespace will be used

CIS Controls:

Version 7

5.1 Establish Secure Configurations
Maintain documented, standard security configuration standards for all authorized operating systems and software.

5 Managed services

This section consists of security recommendations for the Amazon EKS. These recommendations are applicable for configurations that Amazon EKS customers own and manage.

5.1 Image Registry and Image Scanning

This section contains recommendations relating to container image registries and securing images in those registries, such as Amazon Elastic Container Registry (ECR).

5.1.1 Ensure Image Vulnerability Scanning using Amazon ECR image scanning or a third party provider (Manual)

Profile Applicability:

- Level 1

Description:

Scan images being deployed to Amazon EKS for vulnerabilities.

Rationale:

Vulnerabilities in software packages can be exploited by hackers or malicious users to obtain unauthorized access to local cloud resources. Amazon ECR and other third party products allow images to be scanned for known vulnerabilities.

Audit:

Please follow AWS ECS or your 3rd party image scanning provider's guidelines for enabling Image Scanning.

Remediation:

To utilize AWS ECR for Image scanning please follow the steps below:
To create a repository configured for scan on push (AWS CLI)

```
aws ecr create-repository --repository-name $REPO_NAME --image-scanning-
configuration scanOnPush=true --region $REGION_CODE
```

To edit the settings of an existing repository (AWS CLI)

```
aws ecr put-image-scanning-configuration --repository-name $REPO_NAME --
image-scanning-configuration scanOnPush=true --region $REGION_CODE
```

Use the following steps to start a manual image scan using the AWS Management Console.

1. Open the Amazon ECR console at https://console.aws.amazon.com/ecr/repositories.
2. From the navigation bar, choose the Region to create your repository in.

3. In the navigation pane, choose Repositories.
4. On the Repositories page, choose the repository that contains the image to scan.
5. On the Images page, select the image to scan and then choose Scan.

Impact:

If you are utilizing AWS ECR

The following are common image scan failures. You can view errors like this in the Amazon ECR console by displaying the image details or through the API or AWS CLI by using the DescribeImageScanFindings API.

UnsupportedImageError You may get an UnsupportedImageError error when attempting to scan an image that was built using an operating system that Amazon ECR doesn't support image scanning for. Amazon ECR supports package vulnerability scanning for major versions of Amazon Linux, Amazon Linux 2, Debian, Ubuntu, CentOS, Oracle Linux, Alpine, and RHEL Linux distributions. Amazon ECR does not support scanning images built from the Docker scratch image.

An UNDEFINED severity level is returned You may receive a scan finding that has a severity level of UNDEFINED. The following are the common causes for this:

The vulnerability was not assigned a priority by the CVE source.

The vulnerability was assigned a priority that Amazon ECR did not recognize.

To determine the severity and description of a vulnerability, you can view the CVE directly from the source.

Default Value:

Images are not scanned by Default.

References:

1. https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning.html

CIS Controls:

Version 7

3 Continuous Vulnerability Management
Continuous Vulnerability Management

3.1 Run Automated Vulnerability Scanning Tools

Utilize an up-to-date SCAP-compliant vulnerability scanning tool to automatically scan all systems on the network on a weekly or more frequent basis to identify all potential vulnerabilities on the organization's systems.

3.2 Perform Authenticated Vulnerability Scanning

Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.

5.1.2 Minimize user access to Amazon ECR (Manual)

Profile Applicability:

- Level 1

Description:

Restrict user access to Amazon ECR, limiting interaction with build images to only authorized personnel and service accounts.

Rationale:

Weak access control to Amazon ECR may allow malicious users to replace built images with vulnerable containers.

Audit:

Remediation:

Before you use IAM to manage access to Amazon ECR, you should understand what IAM features are available to use with Amazon ECR. To get a high-level view of how Amazon ECR and other AWS services work with IAM, see AWS Services That Work with IAM in the IAM User Guide.

Topics

- Amazon ECR Identity-Based Policies
- Amazon ECR Resource-Based Policies
- Authorization Based on Amazon ECR Tags
- Amazon ECR IAM Roles

Amazon ECR Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon ECR supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the IAM User Guide.

Actions

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon ECR use the following prefix before the action: ecr:. For example, to grant someone permission to create an Amazon ECR repository with the Amazon ECR

CreateRepository API operation, you include the ecr:CreateRepository action in their policy. Policy statements must include either an Action or NotAction element. Amazon ECR defines its own set of actions that describe tasks that you can perform with this service.
To specify multiple actions in a single statement, separate them with commas as follows:
`"Action": [ "ecr:action1", "ecr:action2"`
You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Describe, include the following action:
`"Action": "ecr:Describe*"`
To see a list of Amazon ECR actions, see Actions, Resources, and Condition Keys for Amazon Elastic Container Registry in the IAM User Guide.

Resources

The Resource element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.
An Amazon ECR repository resource has the following ARN:
`arn:${Partition}:ecr:${Region}:${Account}:repository/${Repository-name}`
For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.
For example, to specify the my-repo repository in the us-east-1 Region in your statement, use the following ARN:
`"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/my-repo"`
To specify all repositories that belong to a specific account, use the wildcard (*):
`"Resource": "arn:aws:ecr:us-east-1:123456789012:repository/*"`
To specify multiple resources in a single statement, separate the ARNs with commas.
`"Resource": [ "resource1", "resource2"`
To see a list of Amazon ECR resource types and their ARNs, see Resources Defined by Amazon Elastic Container Registry in the IAM User Guide. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Elastic Container Registry.

Condition Keys

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.
If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.
You can also use placeholder variables when you specify conditions. For example, you can

grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the IAM User Guide.
Amazon ECR defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.
Most Amazon ECR actions support the aws:ResourceTag and ecr:ResourceTag condition keys. For more information, see Using Tag-Based Access Control.
To see a list of Amazon ECR condition keys, see Condition Keys Defined by Amazon Elastic Container Registry in the IAM User Guide. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon Elastic Container Registry.

Impact:

Care should be taken not to remove access to Amazon ECR for accounts that require this for their operation.

References:

1. https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning.html#scanning-repository

CIS Controls:

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

5.1.3 Minimize cluster access to read-only for Amazon ECR (Manual)

Profile Applicability:

- Level 1

Description:

Configure the Cluster Service Account with Storage Object Viewer Role to only allow read-only access to Amazon ECR.

Rationale:

The Cluster Service Account does not require administrative access to Amazon ECR, only requiring pull access to containers to deploy onto Amazon EKS. Restricting permissions follows the principles of least privilege and prevents credentials from being abused beyond the required role.

Audit:

Review AWS ECS worker node IAM role (NodeInstanceRole) IAM Policy Permissions to verify that they are set and the minimum required level.
If utilizing a 3rd party tool to scan images utilize the minimum required permission level required to interact with the cluster - generally this should be read-only.

Remediation:

You can use your Amazon ECR images with Amazon EKS, but you need to satisfy the following prerequisites.
The Amazon EKS worker node IAM role (NodeInstanceRole) that you use with your worker nodes must possess the following IAM policy permissions for Amazon ECR.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

Impact:

A separate dedicated service account may be required for use by build servers and other robot users pushing or managing container images.

Default Value:

If you used eksctl or the AWS CloudFormation templates in Getting Started with Amazon EKS to create your cluster and worker node groups, these IAM permissions are applied to your worker node IAM role by default.

References:

1. https://docs.aws.amazon.com/AmazonECR/latest/userguide/ECR_on_EKS.html

CIS Controls:

Version 7

3.2 Perform Authenticated Vulnerability Scanning

Perform authenticated vulnerability scanning with agents running locally on each system or with remote scanners that are configured with elevated rights on the system being tested.

5.1.4 Minimize Container Registries to only those approved (Manual)

Profile Applicability:

- Level 2

Description:

Use approved container registries.

Rationale:

Allowing unrestricted access to external container registries provides the opportunity for malicious or unapproved containers to be deployed into the cluster. Allowlisting only approved container registries reduces this risk.

Audit:

Remediation:

Impact:

All container images to be deployed to the cluster must be hosted within an approved container image registry.

References:

1. https://aws.amazon.com/blogs/opensource/using-open-policy-agent-on-amazon-eks/

CIS Controls:

Version 7

5.2 Maintain Secure Images
Maintain secure images or templates for all systems in the enterprise based on the organization's approved configuration standards. Any new system deployment or existing system that becomes compromised should be imaged using one of those images or templates.

5.3 Securely Store Master Images
Store the master images and templates on securely configured servers, validated with integrity monitoring tools, to ensure that only authorized changes to the images are possible.

5.2 Identity and Access Management (IAM)

This section contains recommendations relating to using AWS IAM with Amazon EKS.

5.2.1 Prefer using dedicated EKS Service Accounts (Manual)

Profile Applicability:

- Level 1

Description:

Kubernetes workloads should not use cluster node service accounts to authenticate to Amazon EKS APIs. Each Kubernetes workload that needs to authenticate to other AWS services using AWS IAM should be provisioned with a dedicated Service account.

Rationale:

Manual approaches for authenticating Kubernetes workloads running on Amazon EKS against AWS APIs are: storing service account keys as a Kubernetes secret (which introduces manual key rotation and potential for key compromise); or use of the underlying nodes' IAM Service account, which violates the principle of least privilege on a multi-tenanted node, when one pod needs to have access to a service, but every other pod on the node that uses the Service account does not.

Audit:

For each namespace in the cluster, review the rights assigned to the default service account and ensure that it has no roles or cluster roles bound to it apart from the defaults. Additionally ensure that the automountServiceAccountToken: false setting is in place for each default service account.

Remediation:

With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. This service account can then provide AWS permissions to the containers in any pod that uses that service account. With this feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs.
Applications must sign their AWS API requests with AWS credentials. This feature provides a strategy for managing credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance’s

role, you can associate an IAM role with a Kubernetes service account. The applications in the pod's containers can then use an AWS SDK or the AWS CLI to make API requests to authorized AWS services.
The IAM roles for service accounts feature provides the following benefits:

- Least privilege — By using the IAM roles for service accounts feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs. You can scope IAM permissions to a service account, and only pods that use that service account have access to those permissions. This feature also eliminates the need for third-party solutions such as kiam or kube2iam.
- Credential isolation — A container can only retrieve credentials for the IAM role that is associated with the service account to which it belongs. A container never has access to credentials that are intended for another container that belongs to another pod.
- Auditability — Access and event logging is available through CloudTrail to help ensure retrospective auditing.

To get started, see Enabling IAM roles for service accounts on your cluster.
For an end-to-end walkthrough using eksctl, see Walkthrough: Updating a DaemonSet to use IAM for service accounts.

References:

1. https://aws.github.io/aws-eks-best-practices/iam/
2. https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html
3. https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts-cni-walkthrough.html

CIS Controls:

Version 7

4.3 Ensure the Use of Dedicated Administrative Accounts

Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.

5.3 AWS Key Management Service (KMS)

This section contains recommendations relating to using AWS KMS with Amazon EKS.

5.3.1 Ensure Kubernetes Secrets are encrypted using Customer Master Keys (CMKs) managed in AWS KMS (Automated)

Profile Applicability:

- Level 1

Description:

Encrypt Kubernetes secrets, stored in etcd, using secrets encryption feature during Amazon EKS cluster creation.

Rationale:

Kubernetes can store secrets that pods can access via a mounted volume. Today, Kubernetes secrets are stored with Base64 encoding, but encrypting is the recommended approach. Amazon EKS clusters version 1.13 and higher support the capability of encrypting your Kubernetes secrets using AWS Key Management Service (KMS) Customer Managed Keys (CMK). The only requirement is to enable the encryption provider support during EKS cluster creation.

Use AWS Key Management Service (KMS) keys to provide envelope encryption of Kubernetes secrets stored in Amazon EKS. Implementing envelope encryption is considered a security best practice for applications that store sensitive data and is part of a defense in depth security strategy.

Application-layer Secrets Encryption provides an additional layer of security for sensitive data, such as user defined Secrets and Secrets required for the operation of the cluster, such as service account keys, which are all stored in etcd.

Using this functionality, you can use a key, that you manage in AWS KMS, to encrypt data at the application layer. This protects against attackers in the event that they manage to gain access to etcd.

Audit:

For Amazon EKS clusters with Secrets Encryption enabled, look for 'encryptionConfig' configuration when you run:

```
aws eks describe-cluster --name="<cluster-name>"
```

Remediation:

Enable 'Secrets Encryption' during Amazon EKS cluster creation as described in the links within the 'References' section.

Default Value:

By default, Application-layer Secrets Encryption is not enabled.

References:

1. https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html
2. https://eksworkshop.com/beginner/191_secrets/

CIS Controls:

Version 7

14.8 Encrypt Sensitive Information at Rest

Encrypt all sensitive information at rest using a tool that requires a secondary authentication mechanism not integrated into the operating system, in order to access the information.

5.4 Cluster Networking

This section contains recommendations relating to network security configurations in Amazon EKS.

5.4.1 Restrict Access to the Control Plane Endpoint (Manual)

Profile Applicability:

- Level 1

Description:

Enable Endpoint Private Access to restrict access to the cluster's control plane to only an allowlist of authorized IPs.

Rationale:

Authorized networks are a way of specifying a restricted range of IP addresses that are permitted to access your cluster's control plane. Kubernetes Engine uses both Transport Layer Security (TLS) and authentication to provide secure access to your cluster's control plane from the public internet. This provides you the flexibility to administer your cluster from anywhere; however, you might want to further restrict access to a set of IP addresses that you control. You can set this restriction by specifying an authorized network.

Restricting access to an authorized network can provide additional security benefits for your container cluster, including:

- Better protection from outsider attacks: Authorized networks provide an additional layer of security by limiting external access to a specific set of addresses you designate, such as those that originate from your premises. This helps protect access to your cluster in the case of a vulnerability in the cluster's authentication or authorization mechanism.
- Better protection from insider attacks: Authorized networks help protect your cluster from accidental leaks of master certificates from your company's premises. Leaked certificates used from outside Amazon EC2 and outside the authorized IP ranges (for example, from addresses outside your company) are still denied access.

Audit:

Input:

```
aws eks describe-cluster \
    --region <region> \
    --name <clustername>
```

Output:

```
        ...
        "endpointPublicAccess": false,
        "endpointPrivateAccess": true,
        "publicAccessCidrs": [
            "203.0.113.5/32"
        ]
        ...
```

Remediation:

Complete the following steps using the AWS CLI version 1.18.10 or later. You can check your current version with aws --version. To install or upgrade the AWS CLI, see Installing the AWS CLI.
Update your cluster API server endpoint access with the following AWS CLI command. Substitute your cluster name and desired endpoint access values. If you set endpointPublicAccess=true, then you can (optionally) enter single CIDR block, or a comma-separated list of CIDR blocks for publicAccessCidrs. The blocks cannot include reserved addresses. If you specify CIDR blocks, then the public API server endpoint will only receive requests from the listed blocks. There is a maximum number of CIDR blocks that you can specify. For more information, see Amazon EKS Service Quotas. If you restrict access to your public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that worker nodes and Fargate pods (if you use them) can communicate with the cluster. Without the private endpoint enabled, your public access endpoint CIDR sources must include the egress sources from your VPC. For example, if you have a worker node in a private subnet that communicates to the internet through a NAT Gateway, you will need to add the outbound IP address of the NAT gateway as part of a whitelisted CIDR block on your public endpoint. If you specify no CIDR blocks, then the public API server endpoint receives requests from all (0.0.0.0/0) IP addresses.
Note
The following command enables private access and public access from a single IP address for the API server endpoint. Replace 203.0.113.5/32 with a single CIDR block, or a comma-separated list of CIDR blocks that you want to restrict network access to.
Example command:

```
aws eks update-cluster-config \
    --region region-code \
    --name dev \
    --resources-vpc-config \
    endpointPublicAccess=true, \
```

```
    publicAccessCidrs="203.0.113.5/32",\
    endpointPrivateAccess=true
```

Output:

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "InProgress",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
```

Impact:

When implementing Endpoint Private Access, be careful to ensure all desired networks are on the allowlist (whitelist) to prevent inadvertently blocking external access to your cluster's control plane.

Default Value:

By default, Endpoint Private Access is disabled.

References:

1. https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

CIS Controls:

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the

principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

5.4.2 Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled (Manual)

Profile Applicability:

- Level 2

Description:

Disable access to the Kubernetes API from outside the node network if it is not required.

Rationale:

In a private cluster, the master node has two endpoints, a private and public endpoint. The private endpoint is the internal IP address of the master, behind an internal load balancer in the master's VPC network. Nodes communicate with the master using the private endpoint. The public endpoint enables the Kubernetes API to be accessed from outside the master's VPC network.

Although Kubernetes API requires an authorized token to perform sensitive actions, a vulnerability could potentially expose the Kubernetes publically with unrestricted access. Additionally, an attacker may be able to identify the current cluster and Kubernetes API version and determine whether it is vulnerable to an attack. Unless required, disabling public endpoint will help prevent such threats, and require the attacker to be on the master's VPC network to perform any attack on the Kubernetes API.

Audit:

Remediation:

Impact:

Configure the EKS cluster endpoint to be private. See Modifying Cluster Endpoint Access for further information on this topic.

1. Leave the cluster endpoint public and specify which CIDR blocks can communicate with the cluster endpoint. The blocks are effectively a whitelisted set of public IP addresses that are allowed to access the cluster endpoint.
2. Configure public access with a set of whitelisted CIDR blocks and set private endpoint access to enabled. This will allow public access from a specific range of public IPs while forcing all network traffic between the kubelets (workers) and the Kubernetes API through the cross-account ENIs that get provisioned into the cluster VPC when the control plane is provisioned.

Default Value:

By default, the Private Endpoint is disabled.

References:

1. https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

CIS Controls:

Version 7

12 Boundary Defense
Boundary Defense

5.4.3 Ensure clusters are created with Private Nodes (Manual)

Profile Applicability:

- Level 1

Description:

Disable public IP addresses for cluster nodes, so that they only have private IP addresses. Private Nodes are nodes with no public IP addresses.

Rationale:

Disabling public IP addresses on cluster nodes restricts access to only internal networks, forcing attackers to obtain local network access before attempting to compromise the underlying Kubernetes hosts.

Audit:

Remediation:

Impact:

To enable Private Nodes, the cluster has to also be configured with a private master IP range and IP Aliasing enabled.

Private Nodes do not have outbound access to the public internet. If you want to provide outbound Internet access for your private nodes, you can use Cloud NAT or you can manage your own NAT gateway.

Default Value:

By default, Private Nodes are disabled.

CIS Controls:

Version 7

12 Boundary Defense
Boundary Defense

5.4.4 Ensure Network Policy is Enabled and set as appropriate (Manual)

Profile Applicability:

- Level 1

Description:

Use Network Policy to restrict pod to pod traffic within a cluster and segregate workloads.

Rationale:

By default, all pod to pod traffic within a cluster is allowed. Network Policy creates a pod-level firewall that can be used to restrict traffic between sources. Pod traffic is restricted by having a Network Policy that selects it (through the use of labels). Once there is any Network Policy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any Network Policy. Other pods in the namespace that are not selected by any Network Policy will continue to accept all traffic.

Network Policies are managed via the Kubernetes Network Policy API and enforced by a network plugin, simply creating the resource without a compatible network plugin to implement it will have no effect. EKS supports Network Policy enforcement through the use of Calico.

Audit:

Remediation:

Impact:

Network Policy requires the Network Policy add-on. This add-on is included automatically when a cluster with Network Policy is created, but for an existing cluster, needs to be added prior to enabling Network Policy.

Enabling/Disabling Network Policy causes a rolling update of all cluster nodes, similar to performing a cluster upgrade. This operation is long-running and will block other operations on the cluster (including delete) until it has run to completion.

If Network Policy is used, a cluster must have at least 2 nodes of type `n1-standard-1` or higher. The recommended minimum size cluster to run Network Policy enforcement is 3 `n1-standard-1` instances.

Enabling Network Policy enforcement consumes additional resources in nodes. Specifically, it increases the memory footprint of the `kube-system` process by approximately 128MB, and requires approximately 300 millicores of CPU.

Default Value:

By default, Network Policy is disabled.

CIS Controls:

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running

Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

9.4 Apply Host-based Firewalls or Port Filtering

Apply host-based firewalls or port filtering tools on end systems, with a default-deny rule that drops all traffic except those services and ports that are explicitly allowed.

5.4.5 Encrypt traffic to HTTPS load balancers with TLS certificates (Manual)

Profile Applicability:

- Level 2

Description:

Encrypt traffic to HTTPS load balancers using TLS certificates.

Rationale:

Encrypting traffic between users and your Kubernetes workload is fundamental to protecting data sent over the web.

Audit:

Remediation:

References:

1. https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/data-protection.html

CIS Controls:

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

5.5 Authentication and Authorization

This section contains recommendations relating to authentication and authorization in Amazon EKS.

5.5.1 Manage Kubernetes RBAC users with AWS IAM Authenticator for Kubernetes (Manual)

Profile Applicability:

- Level 2

Description:

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster through the AWS IAM Authenticator for Kubernetes. You can configure the stock kubectl client to work with Amazon EKS by installing the AWS IAM Authenticator for Kubernetes and modifying your kubectl configuration file to use it for authentication.

Rationale:

On- and off-boarding users is often difficult to automate and prone to error. Using a single source of truth for user permissions reduces the number of locations that an individual must be off-boarded from, and prevents users gaining unique permissions sets that increase the cost of audit.

Audit:

To Audit access to the namespace $NAMESPACE, assume the IAM role yourIAMRoleName for a user that you created, and then run the following command:

```
$ kubectl get role -n $NAMESPACE
```

The response lists the RBAC role that has access to this Namespace.

Remediation:

Refer to the 'Managing users or IAM roles for your cluster' in Amazon EKS documentation.

Impact:

Users must now be assigned to the IAM group created to use this namespace and deploy applications. If they are not they will not be able to access the namespace or deploy.

Default Value:

RBAC is not enabled by Default.

References:

1. https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

CIS Controls:

Version 7

16.2 Configure Centralized Point of Authentication

Configure access for all accounts through as few centralized points of authentication as possible, including network, security, and cloud systems.

5.6 Other Cluster Configurations

This section contains recommendations relating to any remaining security-related cluster configurations in Amazon EKS.

5.6.1 Consider Fargate for running untrusted workloads (Manual)

Profile Applicability:

- Level 1

Description:

It is Best Practice to restrict or fence untrusted workloads when running in a multi-tenant environment.

Rationale:

AWS Fargate is a technology that provides on-demand, right-sized compute capacity for containers. With AWS Fargate, you no longer have to provision, configure, or scale groups of virtual machines to run containers. This removes the need to choose server types, decide when to scale your node groups, or optimize cluster packing.

You can control which pods start on Fargate and how they run with Fargate profiles, which are defined as part of your Amazon EKS cluster.

Amazon EKS integrates Kubernetes with AWS Fargate by using controllers that are built by AWS using the upstream, extensible model provided by Kubernetes. These controllers run as part of the Amazon EKS managed Kubernetes control plane and are responsible for scheduling native Kubernetes pods onto Fargate. The Fargate controllers include a new scheduler that runs alongside the default Kubernetes scheduler in addition to several mutating and validating admission controllers. When you start a pod that meets the criteria for running on Fargate, the Fargate controllers running in the cluster recognize, update, and schedule the pod onto Fargate.

Each pod running on Fargate has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another pod.

Audit:

Check the existence of Fargate profiles in the Amazon EKS cluster by using:

```
aws --region ${AWS_REGION} eks list-fargate-profiles --cluster-name
${CLUSTER_NAME}
```

Alternatively, to audit for the presence of a Fargate profile node run the following command:

```
kubectl get nodes
```

The response should include a NAME entry starting with "fargate-ip"
for example:
NAME STATUS ROLES AGE VERSION
fargate-ip-192-168-104-74.us-east-2.compute.internal Ready 2m15s v1.14.8-eks

Remediation:

Create a Fargate profile for your cluster
Before you can schedule pods running on Fargate in your cluster, you must define a Fargate profile that specifies which pods should use Fargate when they are launched. For more information, see AWS Fargate profile.
Note
If you created your cluster with eksctl using the --fargate option, then a Fargate profile has already been created for your cluster with selectors for all pods in the kube-system and default namespaces. Use the following procedure to create Fargate profiles for any other namespaces you would like to use with Fargate.
via eksctl CLI
Create your Fargate profile with the following eksctl command, replacing the variable text with your own values. You must specify a namespace, but the labels option is not required.

```
eksctl create fargateprofile --cluster cluster_name --name
fargate_profile_name --namespace kubernetes_namespace --labels key=value
```

via AWS Management Console
To create a Fargate profile for a cluster with the AWS Management Console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster to create a Fargate profile for.
3. Under Fargate profiles, choose Add Fargate profile.
4. On the Configure Fargate profile page, enter the following information and choose Next.

- For Name, enter a unique name for your Fargate profile.
- For Pod execution role, choose the pod execution role to use with your Fargate profile. Only IAM roles with the eks-fargate-pods.amazonaws.com service principal

are shown. If you do not see any roles listed here, you must create one. For more information, see Pod execution role.
- For Subnets, choose the subnets to use for your pods. By default, all subnets in your cluster's VPC are selected. Only private subnets are supported for pods running on Fargate; you must deselect any public subnets.
- For Tags, you can optionally tag your Fargate profile. These tags do not propagate to other resources associated with the profile, such as its pods.

5. On the Configure pods selection page, enter the following information and choose Next.

- list text hereFor Namespace, enter a namespace to match for pods, such as kube-system or default.
- list text here(Optional) Add Kubernetes labels to the selector that pods in the specified namespace must have to match the selector. For example, you could add the label infrastructure: fargate to the selector so that only pods in the specified namespace that also have the infrastructure: fargate Kubernetes label match the selector.

6. On the Review and create page, review the information for your Fargate profile and choose Create.

Impact:

Refer to the 'AWS Fargate considerations' section in the AWS Fargate guide within Amazon EKS User Guide.

Default Value:

By default, AWS Fargate is not utilized.

References:

1. https://docs.aws.amazon.com/eks/latest/userguide/fargate.html

CIS Controls:

Version 7

18.9 Separate Production and Non-Production Systems

Maintain separate environments for production and nonproduction systems. Developers should not have unmonitored access to production environments.

Appendix: Summary Table

Control		Set Correctly	
		Yes	No
1	**Control Plane Components**		
2	**Control Plane Configuration**		
2.1	**Logging**		
2.1.1	Enable audit Logs (Manual)	☐	☐
3	**Worker Nodes**		
3.1	**Worker Node Configuration Files**		
3.1.1	Ensure that the kubeconfig file permissions are set to 644 or more restrictive (Manual)	☐	☐
3.1.2	Ensure that the kubelet kubeconfig file ownership is set to root:root (Manual)	☐	☐
3.1.3	Ensure that the kubelet configuration file has permissions set to 644 or more restrictive (Manual)	☐	☐
3.1.4	Ensure that the kubelet configuration file ownership is set to root:root (Manual)	☐	☐
3.2	**Kubelet**		
3.2.1	Ensure that the --anonymous-auth argument is set to false (Automated)	☐	☐
3.2.2	Ensure that the --authorization-mode argument is not set to AlwaysAllow (Automated)	☐	☐
3.2.3	Ensure that the --client-ca-file argument is set as appropriate (Manual)	☐	☐
3.2.4	Ensure that the --read-only-port is secured (Manual)	☐	☐
3.2.5	Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Manual)	☐	☐
3.2.6	Ensure that the --protect-kernel-defaults argument is set to true (Automated)	☐	☐
3.2.7	Ensure that the --make-iptables-util-chains argument is set to true (Automated)	☐	☐
3.2.8	Ensure that the --hostname-override argument is not set (Manual)	☐	☐
3.2.9	Ensure that the --eventRecordQPS argument is set to 0 or a level which ensures appropriate event capture (Automated)	☐	☐
3.2.10	Ensure that the --rotate-certificates argument is not set to false (Manual)	☐	☐
3.2.11	Ensure that the RotateKubeletServerCertificate argument is set to true (Manual)	☐	☐
4	**Policies**		

4.1	**RBAC and Service Accounts**		
4.1.1	Ensure that the cluster-admin role is only used where required (Manual)	☐	☐
4.1.2	Minimize access to secrets (Manual)	☐	☐
4.1.3	Minimize wildcard use in Roles and ClusterRoles (Manual)	☐	☐
4.1.4	Minimize access to create pods (Manual)	☐	☐
4.1.5	Ensure that default service accounts are not actively used. (Manual)	☐	☐
4.1.6	Ensure that Service Account Tokens are only mounted where necessary (Manual)	☐	☐
4.2	**Pod Security Policies**		
4.2.1	Minimize the admission of privileged containers (Automated)	☐	☐
4.2.2	Minimize the admission of containers wishing to share the host process ID namespace (Automated)	☐	☐
4.2.3	Minimize the admission of containers wishing to share the host IPC namespace (Automated)	☐	☐
4.2.4	Minimize the admission of containers wishing to share the host network namespace (Automated)	☐	☐
4.2.5	Minimize the admission of containers with allowPrivilegeEscalation (Automated)	☐	☐
4.2.6	Minimize the admission of root containers (Automated)	☐	☐
4.2.7	Minimize the admission of containers with the NET_RAW capability (Automated)	☐	☐
4.2.8	Minimize the admission of containers with added capabilities (Automated)	☐	☐
4.2.9	Minimize the admission of containers with capabilities assigned (Manual)	☐	☐
4.3	**CNI Plugin**		
4.3.1	Ensure latest CNI version is used (Manual)	☐	☐
4.3.2	Ensure that all Namespaces have Network Policies defined (Automated)	☐	☐
4.4	**Secrets Management**		
4.4.1	Prefer using secrets as files over secrets as environment variables (Manual)	☐	☐
4.4.2	Consider external secret storage (Manual)	☐	☐
4.5	**Extensible Admission Control**		
4.5.1	Configure Image Provenance using ImagePolicyWebhook admission controller (Manual)	☐	☐
4.6	**General Policies**		
4.6.1	Create administrative boundaries between resources using namespaces (Manual)	☐	☐

4.6.2	Apply Security Context to Your Pods and Containers (Manual)	☐	☐
4.6.3	The default namespace should not be used (Automated)	☐	☐
5	**Managed services**		
5.1	**Image Registry and Image Scanning**		
5.1.1	Ensure Image Vulnerability Scanning using Amazon ECR image scanning or a third party provider (Manual)	☐	☐
5.1.2	Minimize user access to Amazon ECR (Manual)	☐	☐
5.1.3	Minimize cluster access to read-only for Amazon ECR (Manual)	☐	☐
5.1.4	Minimize Container Registries to only those approved (Manual)	☐	☐
5.2	**Identity and Access Management (IAM)**		
5.2.1	Prefer using dedicated EKS Service Accounts (Manual)	☐	☐
5.3	**AWS Key Management Service (KMS)**		
5.3.1	Ensure Kubernetes Secrets are encrypted using Customer Master Keys (CMKs) managed in AWS KMS (Automated)	☐	☐
5.4	**Cluster Networking**		
5.4.1	Restrict Access to the Control Plane Endpoint (Manual)	☐	☐
5.4.2	Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled (Manual)	☐	☐
5.4.3	Ensure clusters are created with Private Nodes (Manual)	☐	☐
5.4.4	Ensure Network Policy is Enabled and set as appropriate (Manual)	☐	☐
5.4.5	Encrypt traffic to HTTPS load balancers with TLS certificates (Manual)	☐	☐
5.5	**Authentication and Authorization**		
5.5.1	Manage Kubernetes RBAC users with AWS IAM Authenticator for Kubernetes (Manual)	☐	☐
5.6	**Other Cluster Configurations**		
5.6.1	Consider Fargate for running untrusted workloads (Manual)	☐	☐

Appendix: Change History

Date	Version	Changes for this version