

# CRQ000001939084: AWS Mendix : EKS : Kubernetes 1.31 & 1.32 version upgrade

- Summary
  - [Amazon EKS Kubernetes release calendar](#)
- Steps
  - Prerequisites
    - [Free IPs](#)
  - [Kubernetes 1.31 version upgrade](#)
    - [Pre-Upgrade Process](#)
      - [Update EKS Master Node Version](#)
      - [Check AWS CDK Compatibility](#)
      - [Update AWS CDK Version](#)
      - [Install Dependencies](#)
      - [Synthesize the CDK Application](#)
      - [Check Infrastructure Differences](#)
  - [Upgrade control plane](#)
    - [Execute Azure DevOps Pipeline](#)
    - [Verify Pipeline Execution](#)
  - [Upgrade EKS core addons](#)
    - [Upgrade CoreDNS](#)
    - [Upgrade kube-proxy](#)
    - [Upgrade aws-node](#)
    - [Verification of All Upgrades](#)
  - [Upgrade other addons](#)
    - [Karpenter](#)
    - [external-dns](#)
    - [aws-ebs-csi-driver](#)
    - [aws-efs-csi-driver](#)
    - [aws-load-balancer-controller](#)
    - [blueprints-addon-metrics-server](#)
    - [aws-node-termination-handler](#)
    - [Common Validation Steps before executing maas pipeline for All other Add-ons upgrade](#)
  - [Upgrade managed nodes](#)
    - [Upgrade Node Groups](#)
    - [Upgrade Karpenter managed nodes](#)
  - [Post-Upgrade Verification](#)
  - [Kubernetes 1.32 version upgrade](#)
    - [Pre-Upgrade Process](#)
      - [Update EKS Master Node Version](#)
      - [Check AWS CDK Compatibility](#)
      - [Update AWS CDK Version](#)
      - [Install Dependencies](#)
      - [Synthesize the CDK Application](#)

- Check Infrastructure Differences
- Upgrade control plane
  - Execute Azure DevOps Pipeline
  - Verify Pipeline Execution
- Upgrade EKS core addons
  - Upgrade CoreDNS
  - Upgrade kube-proxy
  - Upgrade aws-node
  - Verification of All Upgrades
- Upgrade other addons
  - Karpenter
  - external-dns
  - aws-ebs-csi-driver
  - aws-efs-csi-driver
  - aws-load-balancer-controller
  - blueprints-addon-metrics-server
  - aws-node-termination-handler
  - Common Validation Steps before executing maas pipeline for All other Add-ons upgrade
- Upgrade managed nodes
  - Upgrade Node Groups
  - Upgrade Karpenter managed nodes
- Post-Upgrade Verification
- Implementation
- Lesson learnt from previous upgrade
- Configuration Changes

⚠ This wiki outlines the step-by-step process which was followed for upgrading AWS EKS Kubernetes version from 1.30 to 1.31 & 1.32. This is not intended to be followed as such as the steps may vary based on the issues that may be encountered following these steps and also based on the Kubernetes version from and to which the upgrade is been done. Working experience on AWS and Kubernetes is desired for executing these steps.

Always refer to the official documentation and do detailed analysis before upgrading different versions.

⚠ Make sure to adapt the commands based on the environment they are executed in

## Summary

In order to upgrade the AWS EKS cluster to the supported version we need to upgrade the EKS Kubernetes version before the end of support of the current version, the AWS EKS Kubernetes version across all the clusters in Dev, QA and Production needs to be upgraded

to the next higher version. Upgrade till production needs to be completed before Jul 23, 2025 .

Amazon EKS Kubernetes release calendar

As per [doc](#),

Kubernetes version	Upstream release	Amazon EKS release	End of standard support	End of extended support
1.33	April 23, 2025	May, 2025	July, 2026	July, 2027
1.32	December 11, 2024	January 23, 2025	March 23, 2026	March 23, 2027
1.31	August 13, 2024	September 26, 2024	November 26, 2025	November 26, 2026
1.30	April 17, 2024	May 23, 2024	July 23, 2025	July 23, 2026
1.29	December 13, 2023	January 23, 2024	March 23, 2025	March 23, 2026
1.28	August 15, 2023	September 26, 2023	November 26, 2024	November 26, 2025
1.27	April 11, 2023	May 24, 2023	July 24, 2024	July 24, 2025
1.26	December 9, 2022	April 11, 2023	June 11, 2024	June 11, 2025

Steps

Prerequisites

Free IPs

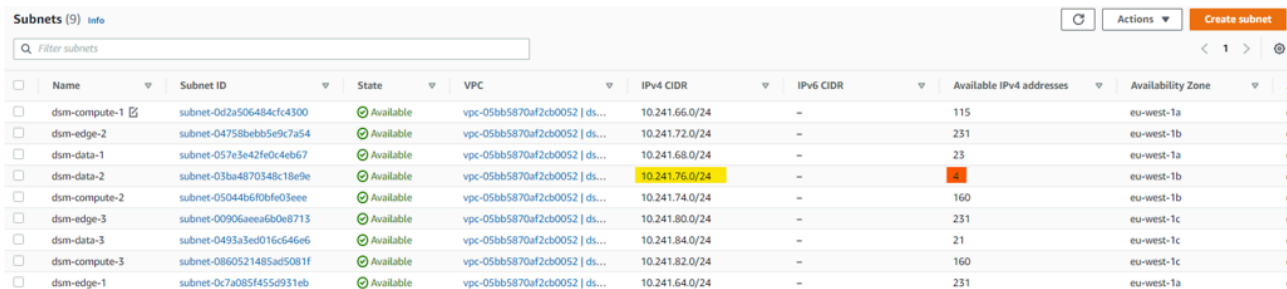


As per [AWS document](#), To update the cluster, Amazon EKS requires up to five free IP addresses from the subnets that you specified when you created your cluster. Amazon EKS creates new cluster elastic network interfaces (network interfaces) in any of the subnets that you specified. The network interfaces may be created in different subnets than your existing network interfaces are in, so make sure that your security group rules allow [required cluster communication](#) for any of the subnets that you specified when you created your cluster. If any of the subnets that you specified when you created the cluster don't exist, don't have enough free IP addresses, or don't have security group rules that allows necessary cluster communication, then the update can fail.

Below is an error you may get if you don't have enough free IP addresses when upgrading the cluster

```
1 2022-10-20 04:43:53 [!] NOTE: cluster VPC (subnets, routing & NAT Gateway) configuration
  changes are not yet implemented
2 2022-10-20 04:43:54 [FÄ] will upgrade cluster "dsm-mendix-alb" control plane from current
  version "1.30" to "1.31"
3 Error: operation error EKS: UpdateClusterVersion, https response error StatusCode: 400,
  RequestID: b915e60a-65eb-4c02-84b7-a2b6ae918bcd, InvalidRequestException: Provided subnets
  subnet-03ba4870348c18e9e Free IPs: 0 , need at least 5 IPs in each subnet to be free for this
  operation
```

and from the subnets in the below screenshot, you can see there is only 4 free IP address available for the subnet **10.241.76.0/24**



Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4 addresses	Availability Zone
dsm-compute-1	subnet-0d2a506484cf4300	Available	vpc-05bb5870af2cb0052   ds...	10.241.66.0/24	-	115	eu-west-1a
dsm-edge-2	subnet-04758beb5e9c7a54	Available	vpc-05bb5870af2cb0052   ds...	10.241.72.0/24	-	231	eu-west-1b
dsm-data-1	subnet-057e3e42fe0c4eb67	Available	vpc-05bb5870af2cb0052   ds...	10.241.68.0/24	-	23	eu-west-1a
dsm-data-2	subnet-03ba4870348c18e9e	Available	vpc-05bb5870af2cb0052   ds...	10.241.76.0/24	-	4	eu-west-1b
dsm-compute-2	subnet-05044b6f0bfe03eee	Available	vpc-05bb5870af2cb0052   ds...	10.241.74.0/24	-	160	eu-west-1b
dsm-edge-3	subnet-00906aeaa6b0e8713	Available	vpc-05bb5870af2cb0052   ds...	10.241.80.0/24	-	231	eu-west-1c
dsm-data-3	subnet-0493a3ed016c646e6	Available	vpc-05bb5870af2cb0052   ds...	10.241.84.0/24	-	21	eu-west-1c
dsm-compute-3	subnet-0860521485ad5081f	Available	vpc-05bb5870af2cb0052   ds...	10.241.82.0/24	-	160	eu-west-1c
dsm-edge-1	subnet-0c7a085f45d931eb	Available	vpc-05bb5870af2cb0052   ds...	10.241.64.0/24	-	231	eu-west-1a

We can check for any application containers that are in this subnet and temporarily scale down them to free up the IP addresses. But in a production environment it is done with caution as only some uncritical applications can be shutdown temporarily.

```
1 kubectl get pods --sort-by=.metadata.creationTimestamp -n mendix -o wide | awk '{print $6}' |
  cut -d. -f1,2,3 | sort | uniq -c
2      43 10.241.66
3      70 10.241.68
4      30 10.241.74
5      83 10.241.76
6      30 10.241.82
7      82 10.241.84
```

After we scaled down some uncritical / redundant apps (always check with concerned developers before stopping if it is for production) we can there are enough free Ips in all subnets to proceed with the upgrade process.

Subnets (9) <a href="#">Info</a>								
<input type="text" value="Filter subnets"/>								
<input type="checkbox"/>	Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4 addresses	Availability Zone
<input type="checkbox"/>	dsm-compute-1	subnet-0d2a506484fc4300	Available	vpc-05bb5870af2cb0052   ds...	10.241.66.0/24	-	115	eu-west-1a
<input type="checkbox"/>	dsm-edge-2	subnet-04758bebb5e9c7a54	Available	vpc-05bb5870af2cb0052   ds...	10.241.72.0/24	-	231	eu-west-1b
<input type="checkbox"/>	dsm-data-1	subnet-057e3e42fe0c4eb67	Available	vpc-05bb5870af2cb0052   ds...	10.241.68.0/24	-	23	eu-west-1a
<input type="checkbox"/>	dsm-data-2	subnet-03ba4870348c18e9e	Available	vpc-05bb5870af2cb0052   ds...	10.241.76.0/24	-	19	eu-west-1b
<input type="checkbox"/>	dsm-compute-2	subnet-05044b6f0bf0e3eee	Available	vpc-05bb5870af2cb0052   ds...	10.241.74.0/24	-	160	eu-west-1b
<input type="checkbox"/>	dsm-edge-3	subnet-00906aeaa6b0e8713	Available	vpc-05bb5870af2cb0052   ds...	10.241.80.0/24	-	230	eu-west-1c
<input type="checkbox"/>	dsm-data-3	subnet-04935ae016c646e6	Available	vpc-05bb5870af2cb0052   ds...	10.241.84.0/24	-	21	eu-west-1c
<input type="checkbox"/>	dsm-compute-3	subnet-0860521485ad5081f	Available	vpc-05bb5870af2cb0052   ds...	10.241.82.0/24	-	160	eu-west-1c
<input type="checkbox"/>	dsm-edge-1	subnet-0c7a085f455d931eb	Available	vpc-05bb5870af2cb0052   ds...	10.241.64.0/24	-	231	eu-west-1a

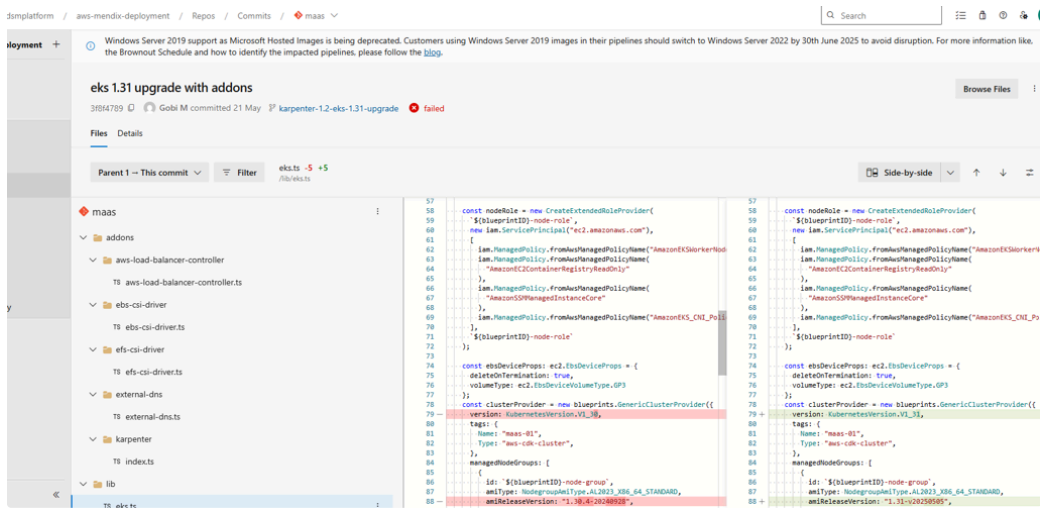
## Kubernetes 1.31 version upgrade

### Pre-Upgrade Process

Below are the steps to upgrade an EKS cluster from version 1.30 to 1.31 using the Azure DevOps [pipeline](#). The upgrade involves changes to the `maas` repository, updating the `aws-cdk` version, and ensuring compatibility with the new EKS version. The plan is to upgrade the Kubernetes version via the Azure DevOps pipeline as the cluster was created using the infrastructure as code defined in the `maas` repository.

#### Update EKS Master Node Version

1. Navigate to the `maas` repository.
- Update the Kubernetes version to `1.31` in the `/lib/eks.ts` file.



#### Check AWS CDK Compatibility

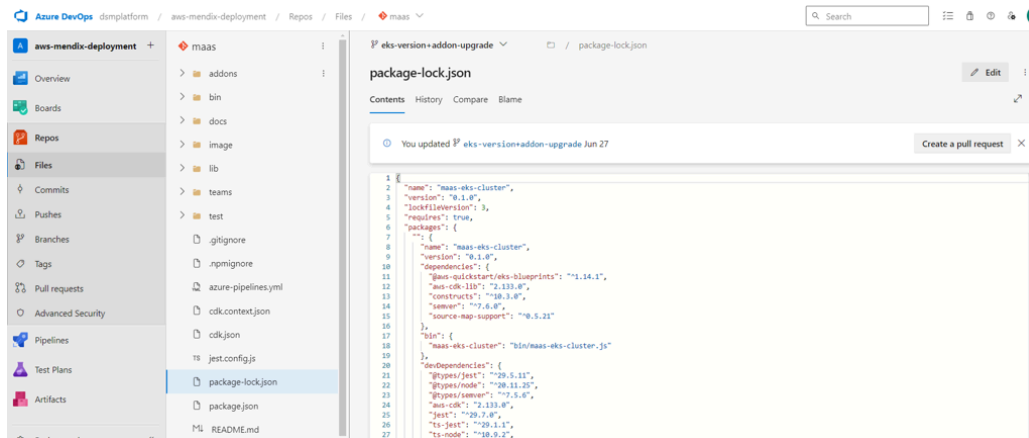
1. Verify if the current `aws-cdk-lib` version in the `package.json` file supports EKS version `1.31`.
- Open the `package.json` file.

◦ Check the version of `aws-cdk-lib` and related dependencies.

- If the current version does not support EKS version **1.31**, proceed to update the **aws-cdk** version.

#### Update AWS CDK Version

1. Update the **aws-cdk** version and other related dependencies and dev-dependencies in the **package.json** file.
  - Save the changes.



#### Install Dependencies

1. Open a terminal.
  - Navigate to the local directory of the **maas** repository.
  - Run the following command to install the updated dependencies:

```
1 npm install
```
  - If there are any errors during the installation, resolve the package versioning issues based on the error messages.
  - Repeat the **npm install** command until all dependencies are properly installed.

#### Synthesize the CDK Application

1. Run the following command to synthesize the CDK application and check for any issues:

```
1 cdk synth
```

- Address any issues that arise during the synthesis.

## Check Infrastructure Differences

1. Run the following command to view the differences between the existing infrastructure and the new changes:

```
1 cdk diff
```

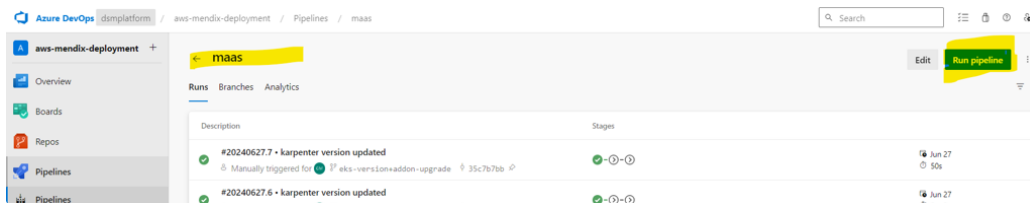
- Review the output to ensure the changes align with the expected upgrades.

## Upgrade control plane

To upgrade the EKS control plane we need ensure the above steps are performed, after preforming the above step we need to execute the maas pipeline to upgrade the control plane.

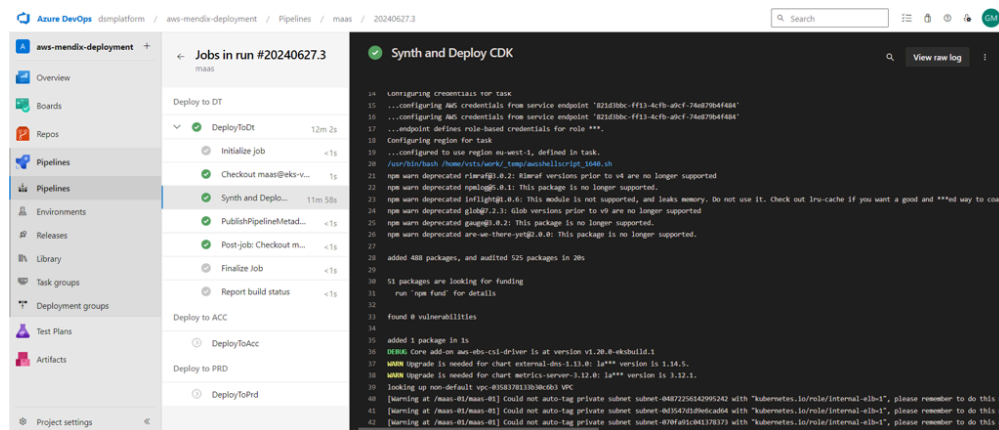
### Execute Azure DevOps Pipeline

- Navigate to the Azure DevOps portal.
- Locate the pipeline associated with the **maas** repository.
- Select the Environment and trigger the pipeline to upgrade the EKS master node to version **1.31**.

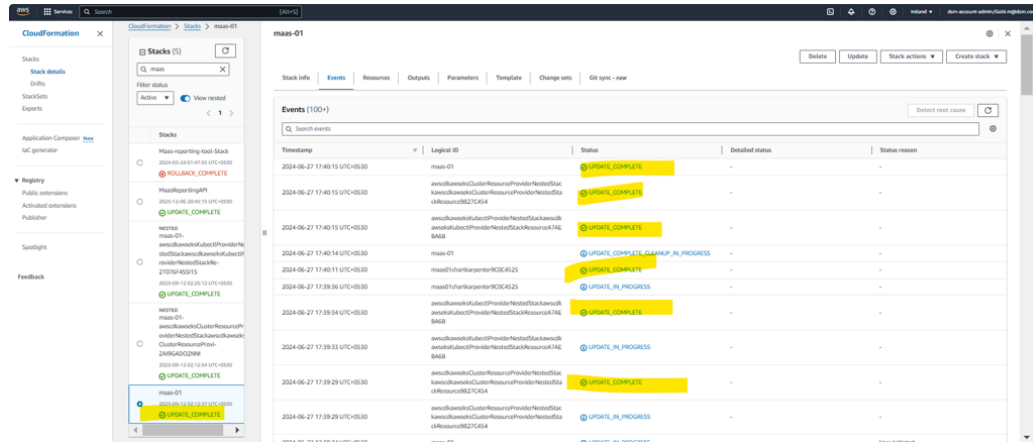


### Verify Pipeline Execution

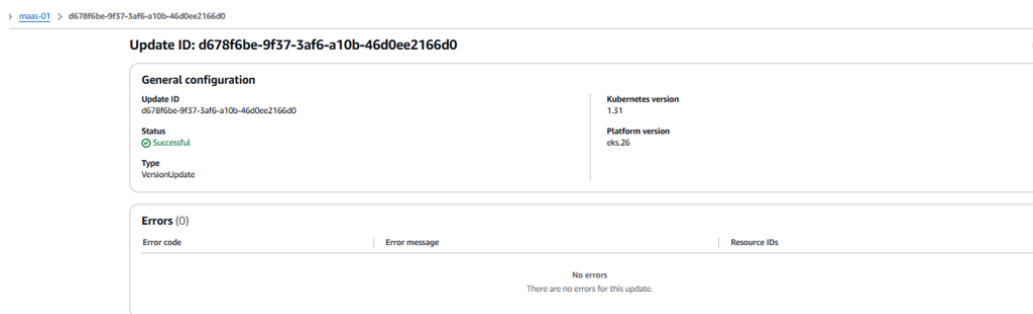
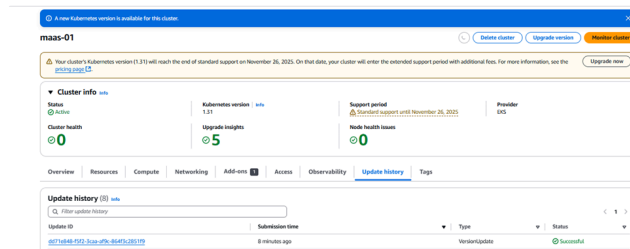
- Once the pipeline execution completes, check the pipeline logs to see the success or failure status of the pipeline.



- Log in to the relevant AWS account and Check the update completion status of the **maas-01** CloudFormation stack.



- Once the control plane upgrade is complete, verify the status from the EKS Console post-upgrade.



## Upgrade EKS core addons

When you provision an EKS cluster you get three add-ons that run on top of the cluster and that are required for it to function properly,





Before upgrading each add-on, refer to the official release documentation to find the compatible version for EKS 1.31. Ensure you update the corresponding code in the Maas repository to reflect the new version. We used Helm to install and manage these add-ons.

This section details the steps to upgrade core add-ons for the EKS cluster: **CoreDNS**, **kube-proxy**, and **aws-node**. The upgrades are performed through azure DevOps pipeline.

### Refer to AWS Documentation for Required Versions

1. Visit the official AWS EKS documentation to find the required versions for each core add-on:
  - [EKS CoreDNS Add-on](#)
  - [EKS kube-proxy Add-on](#)
  - [EKS VPC CNI \(aws-node\) Add-on](#)
  - Check the recommended versions for the EKS version you are upgrading to (in this case, EKS 1.31).

#### Upgrade CoreDNS

- Update the compatible version in the Maas repository `/lib/eks.ts` file. For EKS 1.31, upgrade to CoreDNS version `v1.11.4-eksbuild.2`.

```
1 CoreDNS: 'v1.11.4-eksbuild.2'
```

#### Upgrade kube-proxy

- Update the compatible version in the Maas repository `/lib/eks.ts` file. For EKS 1.31, upgrade to kube-proxy version `v1.30.9-eksbuild.3`.

```
1 kube-proxy: 'v1.30.9-eksbuild.3'
```

#### Upgrade aws-node

- Update the compatible version in the Maas repository `/lib/eks.ts` file. For EKS 1.31, upgrade to aws-node version `v1.11.1-eksbuild.9`.

```
1 aws-node: 'v1.19.4-eksbuild.1'
```

#### Verification of All Upgrades


1. After upgrading each add-on, ensure that all pods are running and there are no issues:

```
1 kubectl get pods -n kube-system
```

By following these steps, you can ensure that the core add-ons are upgraded properly. It is important to keep the add-ons up-to-date to maintain the stability and performance of the EKS cluster. Make sure to monitor the logs and statuses closely to catch and resolve any issues promptly.

## Upgrade other addons

This section details the steps to upgrade various add-ons for the EKS cluster, including **Karpenter**, **external-dns**, **aws-ebs-csi-driver**, **aws-efs-csi-driver**, **aws-load-balancer-controller**, **blueprints-addon-metrics-server**, and **aws-node-termination-handler**. The upgrades are managed through updates to the Maas repository and executed the maas pipeline.

 Before upgrading each add-on, refer to the official release documentation to find the compatible version for EKS 1.31. Ensure you update the corresponding code in the Maas repository to reflect the new version. We used Helm to install and manage these add-ons.

### Karpenter

Karpenter is an open-source cluster autoscaler for Kubernetes clusters. It automatically provisions new nodes in response to unschedulable pods, launching the right compute resources to handle your cluster's applications. Karpenter is designed to leverage the cloud efficiently with fast and simple compute provisioning for Kubernetes clusters. Please refer to the [official Karpenter release documentation](#) to find the compatible version for the upgrade and to check if any prerequisites need to be addressed before upgrading.

**Note:** Karpenter v1.x introduces breaking changes and requires a migration from the `v1beta1` to `v1beta1` APIs. This is not an in-place upgrade.

Please refer to the official Karpenter v1 migration guide and complete the migration before proceeding:

 [v1 Migration](#)

- Update the compatible version in the Maas repository

`/addons/karpenter/karpenter.ts` file. For EKS 1.31, upgrade to version 1.0.6 or later.

```
1 karpenterVersion: '1.0.6'
```

### external-dns

ExternalDNS synchronizes exposed Kubernetes Services and Ingresses with DNS providers. Please refer to the [ExternalDNS Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/external-dns/external-dns.ts` file. For EKS 1.31, upgrade to version 1.15.2.

```
1 externalDnsVersion: '1.15.2'
```

#### aws-ebs-csi-driver

The AWS EBS CSI driver allows dynamic provisioning of Elastic Block Store (EBS) volumes. Please refer to the [AWS EBS CSI Driver Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/ebs-csi-driver/ebs-csi-driver.ts` file. For EKS 1.31, upgrade to version v1.43.0-eksbuild.1.

```
1 ebsCsiDriverVersion: 'v1.43.0-eksbuild.1'
```

#### aws-efs-csi-driver

The AWS EFS CSI driver allows dynamic provisioning of Elastic File System (EFS) volumes. Please refer to the [AWS EFS CSI Driver Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/efs-csi-driver/efs-csi-driver.ts` file. For EKS 1.31, upgrade to version 3.1.7.

```
1 efsCsiDriverVersion: 'v3.1.7'
```

#### aws-load-balancer-controller

The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Kubernetes cluster. Please refer to the [AWS Load Balancer Controller Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/aws-load-balancer-controller/aws-load-balancer-controller.ts` file. For EKS 1.31, upgrade to version 1.13.1.

```
1 loadBalancerControllerVersion: 'v1.13.1',
```

#### blueprints-addon-metrics-server

The Metrics Server collects resource metrics from Kubelets and provides metrics APIs for horizontal scaling.

- This add-on will automatically upgrade based on the EKS version post control plane upgrade.
- Validate by checking the Metrics Server pods in the EKS cluster.

## aws-node-termination-handler

The AWS Node Termination Handler manages termination lifecycle events for EC2 instances in a Kubernetes cluster. Please refer to the [AWS Node Termination Handler Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/aws-node-termination-handler/aws-node-termination-handler.ts` file.
- No version upgrade is needed for this add-on in this document as it's already the latest version.

**Common Validation Steps before executing maas pipeline for All other Add-ons upgrade**

### 1. Validate Code

- Run the `cdk synth` command to ensure the updated code is syntactically correct and there are no issues.

### 2. Execute Pipeline

- Run the Maas pipeline to apply the updates to the EKS cluster.

### 3. Check Pipeline Logs

- Monitor the pipeline logs to ensure successful execution and identify any errors.

### 4. Check CloudFormation Stack Update Status

- Log in to the AWS account and check the `maas-01` CloudFormation stack update completion status.

### 5. Verify Add-on Upgrades

- Log in to the EKS cluster and describe the pods for each add-on to ensure they are running the updated versions.
- Check the logs of the pods to verify they are functioning correctly:

```
1 kubectl get pods -n <namespace>
2 kubectl logs <pod-name> -n <namespace>
```

By following these steps, you can successfully upgrade the versions of various add-ons for your EKS cluster, ensuring they are compatible with EKS 1.31 and functioning properly.

## Upgrade managed nodes

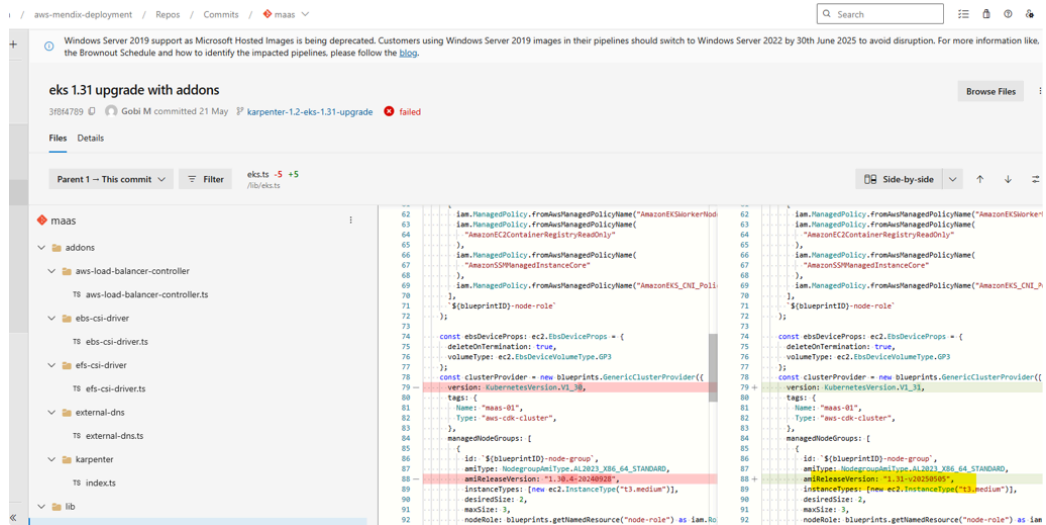
We are using managed node groups.

The way that managed node groups does this is:

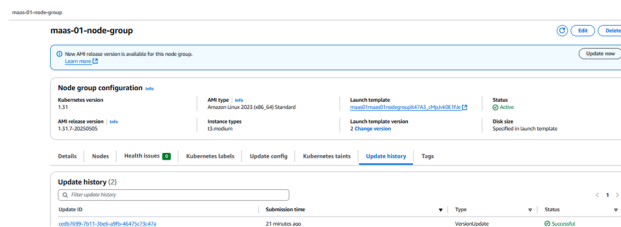
1. Amazon EKS creates a new Amazon EC2 launch template version for the Auto Scaling group associated with your node group. The new template uses the target AMI for the update.
2. The Auto Scaling group is updated to use the latest launch template with the new AMI.
3. The Auto Scaling group maximum size and desired size are incremented by one up to twice the number of Availability Zones in the Region that the Auto Scaling group is deployed in. This is to ensure that at least one new instance comes up in every Availability Zone in the Region that your node group is deployed in.
4. Amazon EKS checks the nodes in the node group for the [eks.amazonaws.com/nodegroup-image](https://eks.amazonaws.com/nodegroup-image) label, and applies a [eks.amazonaws.com/nodegroup=unschedulable:NoSchedule](https://eks.amazonaws.com/nodegroup=unschedulable:NoSchedule) taint on all of the nodes in the node group that aren't labeled with the latest AMI ID. This prevents nodes that have already been updated from a previous failed update from being tainted.
5. Amazon EKS randomly selects a node in the node group and evicts all pods from it.
6. After all of the pods are evicted, Amazon EKS cordons the node. This is done so that the service controller doesn't send any new request to this node and removes this node from its list of healthy, active nodes.
7. Amazon EKS sends a termination request to the Auto Scaling group for the cordoned node.
8. Steps 5-7 are repeated until there are no nodes in the node group that are deployed with the earlier version of the launch template.
9. The Auto Scaling group maximum size and desired size are decremented by 1 to return to your pre-update values.

#### Upgrade Node Groups

1. Navigate to the `maas` repository.
  - Update the `amiReleaseVersion` version for `1.31` in the `/lib/eks.ts` file.
  - Once updated the AMI release version for 1.31 EKS run the pipeline to upgrade the worker node



- **Monitor Progress:** Monitor the upgrade progress through the AWS Management Console and cloudformation template.
- **Verify Post-Upgrade:** Verify that the node group has been successfully upgraded and ensure that your applications are running correctly.



## Upgrade Karpenter managed nodes

1. Navigate to the **maas** repository.
  - Update the **amiSelector** ID for **1.31** in the **/addons/karpenter/index.ts** file.
  - Once updated the AMI release version for 1.31 EKS run the pipeline.
2. Since we have added **do-not-evict** and **do-not-disrupt** annotation for pods karpenter cannot upgrade the nodes for that we need to drain the nodes manually to upgrade the karpenter managed nodes

**Identify Nodes:** Begin by identifying the nodes managed by Karpenter that need to be upgraded. You can use the following command to list nodes:

```
1 kubectl get nodes -l karpenter.sh/provisioner-name=<provisioner-name>
```

**Cordon Nodes:** For each node that needs to be updated, prevent new pods from being scheduled on it by running the `cordon` command:

```
1 kubectl cordon <node-name>
```

Replace `<node-name>` with the name of the specific node.

**Drain Nodes:** After cordoning, drain the node to safely evict running pods. This will ensure pods are rescheduled on other available nodes:

```
1 kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

- The `--ignore-daemonsets` flag prevents issues with daemonset-managed pods.
- The `--delete-emptydir-data` flag clears out temporary data stored in `emptyDir` volumes.

**Monitor Karpenter for New Nodes:** Once the nodes are drained, Karpenter should automatically detect the need for additional capacity and provision new nodes with the latest EKS version. To confirm this, you can monitor the creation of new nodes by running:

```
1 kubectl get nodes -w
```

This command will allow you to watch for new nodes coming online in real-time.

**Verify Node Versions:** After Karpenter provisions the new nodes, check that the new nodes are running the updated EKS version:

```
1 kubectl get nodes -o wide
```

Verify that the version listed for each new node matches the target EKS version.

Following these steps ensures a smooth upgrade for Karpenter-managed nodes in your EKS cluster.

### Post-Upgrade Verification

After upgrading an EKS managed node group, it's crucial to verify the following to ensure the cluster operates smoothly:

- **Node Group Version:** Check that all nodes in the upgraded node group reflect the desired Kubernetes version and AMI release version. This can be verified in the AWS Management Console under EKS Node Groups.

- **Pod Deployment:** Ensure pods are correctly deployed on the updated nodes. Use `kubectl get pods -o wide` to list pods and their nodes. Pods should be scheduled on the nodes that have been upgraded.
- **Pod Logs:** Review logs from pods running on the updated nodes to confirm they are functioning correctly. Use `kubectl logs <pod-name> -n <namespace>` to access specific pod logs.
- **Cluster Health Monitoring:** Monitor the overall health of the EKS cluster using AWS CloudWatch metrics and the Kubernetes dashboard. Look out for any anomalies or issues post-upgrade.
- **Application Validation:** Validate that applications running on the EKS cluster are operating as expected following the node group upgrade. Conduct functional testing of critical services and applications.

## Kubernetes 1.32 version upgrade

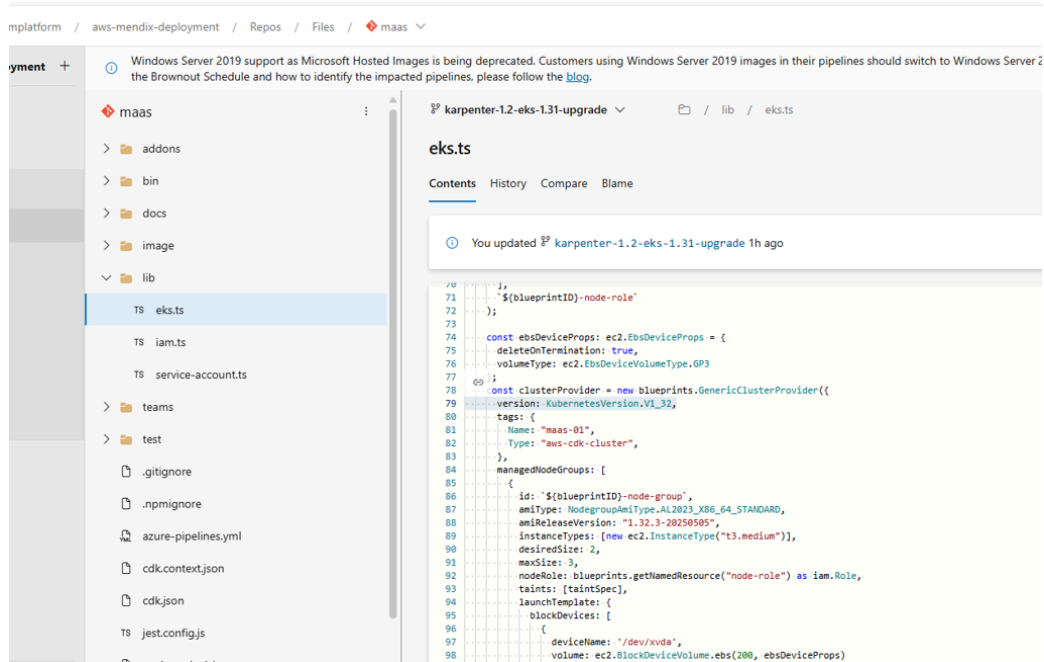
### Pre-Upgrade Process

Below are the steps to upgrade an EKS cluster from version 1.31 to 1.32 using the Azure DevOps [pipeline](#). The upgrade involves changes to the `maas` repository, updating the `aws-cdk` version, and ensuring compatibility with the new EKS version. The plan is to upgrade the Kubernetes version via the Azure DevOps pipeline as the cluster was created using the infrastructure as code defined in the `maas` repository.

### Update EKS Master Node Version

1. Navigate to the `maas` repository.
  - Update the Kubernetes version to `1.32` in the `/lib/eks.ts` file.



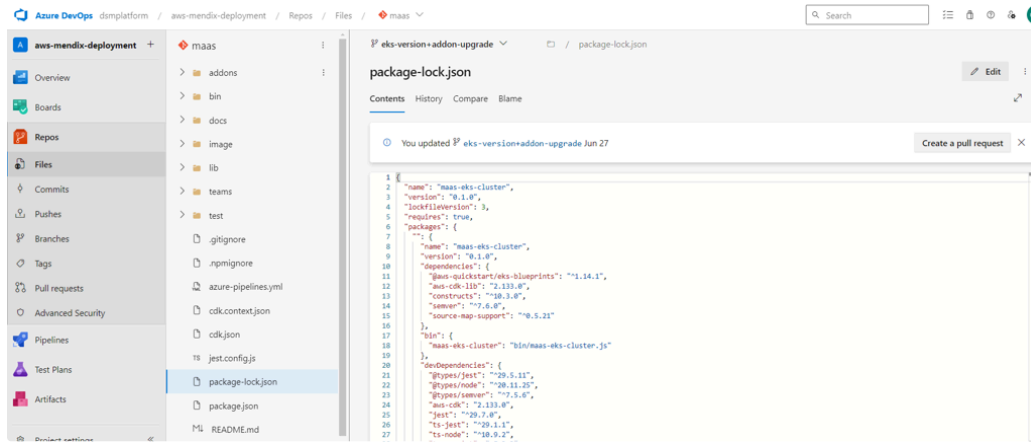


#### Check AWS CDK Compatibility

1. Verify if the current `aws-cdk-lib` version in the `package.json` file supports EKS version `1.32`.
  - Open the `package.json` file.
  - Check the version of `aws-cdk-lib` and related dependencies.
  - If the current version does not support EKS version `1.32`, proceed to update the `aws-cdk` version.

#### Update AWS CDK Version

1. Update the `aws-cdk` version and other related dependencies and dev-dependencies in the `package.json` file.
  - Save the changes.



### Install Dependencies

#### 1. Open a terminal.

- Navigate to the local directory of the **maas** repository.
- Run the following command to install the updated dependencies:

```
1 npm install
```

- If there are any errors during the installation, resolve the package versioning issues based on the error messages.
- Repeat the **npm install** command until all dependencies are properly installed.

### Synthesize the CDK Application

#### 1. Run the following command to synthesize the CDK application and check for any issues:

```
1 cdk synth
```

- Address any issues that arise during the synthesis.

### Check Infrastructure Differences

#### 1. Run the following command to view the differences between the existing infrastructure and the new changes:

```
1 cdk diff
```

- Review the output to ensure the changes align with the expected upgrades.

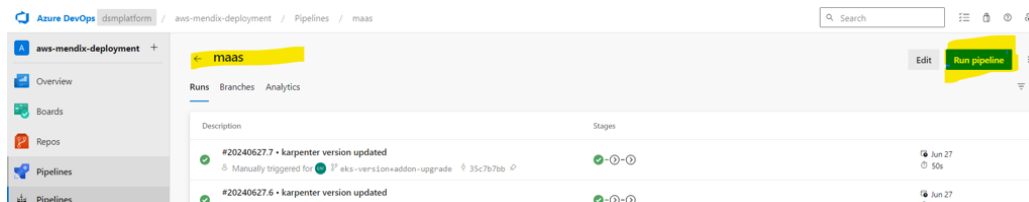
### Upgrade control plane

To upgrade the EKS control plane we need ensure the above steps are performed, after performing the above step we need to execute the maas pipeline to upgrade the control

plane.

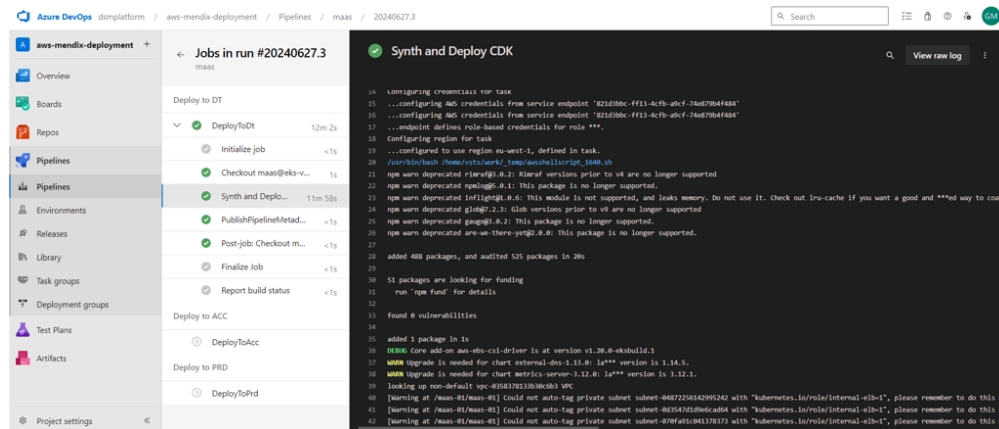
### Execute Azure DevOps Pipeline

- Navigate to the Azure DevOps portal.
- Locate the pipeline associated with the **maas** repository.
- Select the Environment and trigger the pipeline to upgrade the EKS master node to version **1.32**.

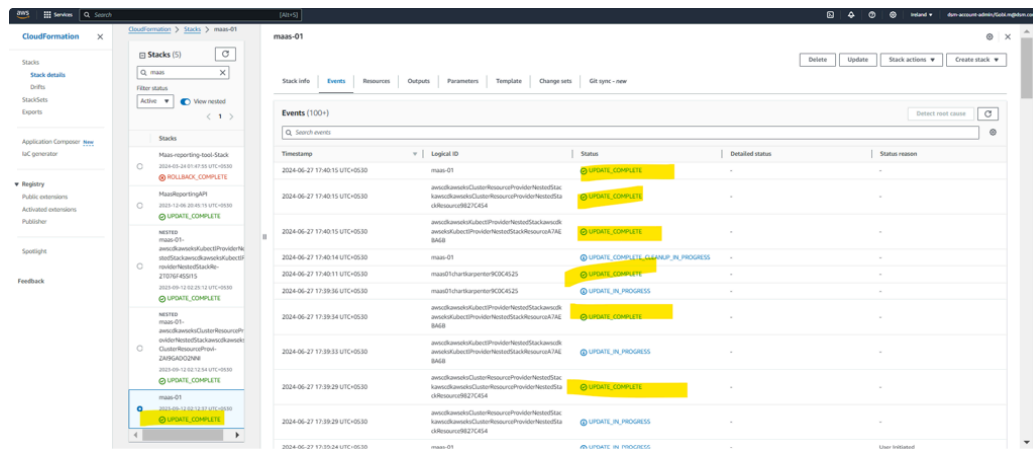


### Verify Pipeline Execution

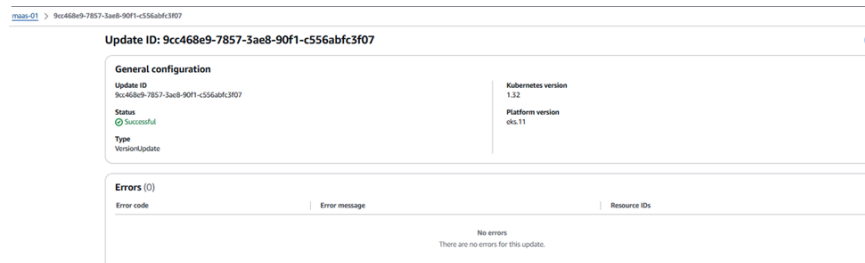
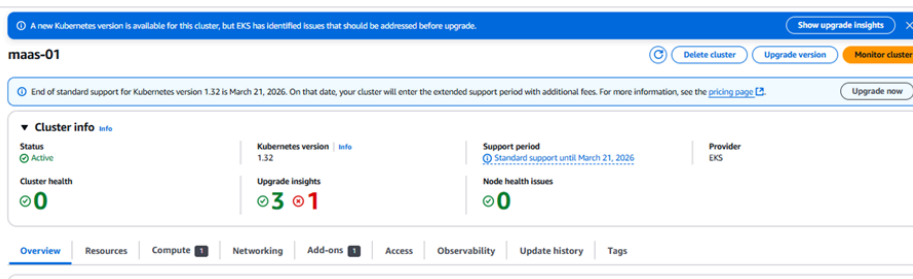
- Once the pipeline execution completes, check the pipeline logs to see the success or failure status of the pipeline.



- Log in to the relevant AWS account and Check the update completion status of the **maas-01** CloudFormation stack.



- Once the control plane upgrade is complete, verify the status from the EKS Console post-upgrade.



## Upgrade EKS core addons

When you provision an EKS cluster you get three add-ons that run on top of the cluster and that are required for it to function properly,

- ⓘ Before upgrading each add-on, refer to the official release documentation to find the compatible version for EKS 1.32. Ensure you update the corresponding code in the Maas repository to reflect the new version. We used Helm to install and manage these add-ons.

This section details the steps to upgrade core add-ons for the EKS cluster: **CoreDNS**, **kube-proxy**, and **aws-node**. The upgrades are performed through azure DevOps pipeline.

### Refer to AWS Documentation for Required Versions

1. Visit the official AWS EKS documentation to find the required versions for each core add-on:
  - [EKS CoreDNS Add-on](#)
  - [EKS kube-proxy Add-on](#)
  - [EKS VPC CNI \(aws-node\) Add-on](#)
  - Check the recommended versions for the EKS version you are upgrading to (in this case, EKS 1.32).

#### Upgrade CoreDNS

- Update the compatible version in the Maas repository `/lib/eks.ts` file. For EKS 1.32, upgrade to CoreDNS version `v1.11.4-eksbuild.10`.

```
1 CoreDNS: 'v1.11.4-eksbuild.10'
```

#### Upgrade kube-proxy

- Update the compatible version in the Maas repository `/lib/eks.ts` file. For EKS 1.32, upgrade to kube-proxy version `v1.32.3-eksbuild.7`.

```
1 kube-proxy: 'v1.32.3-eksbuild.7'
```

#### Upgrade aws-node

- Update the compatible version in the Maas repository `/lib/eks.ts` file. For EKS 1.32, upgrade to aws-node version `v1.19.5-eksbuild.1`.

```
1 aws-node: 'v1.19.5-eksbuild.1'
```

#### Verification of All Upgrades

1. After upgrading each add-on, ensure that all pods are running and there are no issues:

```
1 kubectl get pods -n kube-system
```

By following these steps, you can ensure that the core add-ons are upgraded properly. It is important to keep the add-ons up-to-date to maintain the stability and performance of the EKS cluster. Make sure to monitor the logs and statuses closely to catch and resolve any issues promptly.

## Upgrade other addons

This section details the steps to upgrade various add-ons for the EKS cluster, including **Karpenter**, **external-dns**, **aws-ebs-csi-driver**, **aws-efs-csi-driver**, **aws-load-balancer-controller**, **blueprints-addon-metrics-server**, and **aws-node-termination-handler**. The upgrades are managed through updates to the Maas repository and executed the maas pipeline.

- Before upgrading each add-on, refer to the official release documentation to find the compatible version for EKS 1.32. Ensure you update the corresponding code in the Maas repository to reflect the new version. We used Helm to install and manage these add-ons.

### Karpenter

Karpenter is an open-source cluster autoscaler for Kubernetes clusters. It automatically provisions new nodes in response to unschedulable pods, launching the right compute resources to handle your cluster's applications. Karpenter is designed to leverage the cloud efficiently with fast and simple compute provisioning for Kubernetes clusters. Please refer to the [official Karpenter release documentation](#) to find the compatible version for the upgrade and to check if any prerequisites need to be addressed before upgrading.

- Update the compatible version in the Maas repository `/addons/karpenter/karpenter.ts` file. For EKS 1.32, upgrade to version 1.2.2 or later.

```
1 karpenterVersion: '1.2.2'
```

### external-dns

ExternalDNS synchronizes exposed Kubernetes Services and Ingresses with DNS providers. Please refer to the [ExternalDNS Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/external-dns/external-dns.ts` file. For EKS 1.32, upgrade to version 1.15.2.

```
1 externalDnsVersion: 'v1.15.2'
```

#### aws-ebs-csi-driver

The AWS EBS CSI driver allows dynamic provisioning of Elastic Block Store (EBS) volumes. Please refer to the [AWS EBS CSI Driver Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/ebs-csi-driver/ebs-csi-driver.ts` file. For EKS 1.32, upgrade to version v1.43.0-eksbuild.1.

```
1 ebsCsiDriverVersion: 'v1.43.0-eksbuild.1'
```

#### aws-efs-csi-driver

The AWS EFS CSI driver allows dynamic provisioning of Elastic File System (EFS) volumes. Please refer to the [AWS EFS CSI Driver Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/efs-csi-driver/efs-csi-driver.ts` file. For EKS 1.32, upgrade to version 3.1.7.

```
1 efsCsiDriverVersion: 'v3.1.7'
```

#### aws-load-balancer-controller

The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Kubernetes cluster. Please refer to the [AWS Load Balancer Controller Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/aws-load-balancer-controller/aws-load-balancer-controller.ts` file. For EKS 1.32, upgrade to version 1.13.1.

```
1 loadBalancerControllerVersion: 'v1.13.1',
```

#### blueprints-addon-metrics-server

The Metrics Server collects resource metrics from Kubelets and provides metrics APIs for horizontal scaling.

- This add-on will automatically upgrade based on the EKS version post control plane upgrade.
- Validate by checking the Metrics Server pods in the EKS cluster.

#### aws-node-termination-handler

The AWS Node Termination Handler manages termination lifecycle events for EC2 instances in a Kubernetes cluster. Please refer to the [AWS Node Termination Handler Releases](#) to find the compatible version for the upgrade.

- Update the compatible version in the Maas repository `/addons/aws-node-termination-handler/aws-node-termination-handler.ts` file.
- No version upgrade is needed for this add-on in this document as it's already the latest version.

Common Validation Steps before executing maas pipeline for All other Add-ons upgrade

#### 1. **Validate Code**

- Run the `cdk synth` command to ensure the updated code is syntactically correct and there are no issues.

#### 2. **Execute Pipeline**

- Run the Maas pipeline to apply the updates to the EKS cluster.

#### 3. **Check Pipeline Logs**

- Monitor the pipeline logs to ensure successful execution and identify any errors.

#### 4. **Check CloudFormation Stack Update Status**

- Log in to the AWS account and check the `maas-01` CloudFormation stack update completion status.

#### 5. **Verify Add-on Upgrades**

- Log in to the EKS cluster and describe the pods for each add-on to ensure they are running the updated versions.
- Check the logs of the pods to verify they are functioning correctly:

```
1 kubectl get pods -n <namespace>
2 kubectl logs <pod-name> -n <namespace>
```

By following these steps, you can successfully upgrade the versions of various add-ons for your EKS cluster, ensuring they are compatible with EKS 1.32 and functioning properly.

### **Upgrade managed nodes**

We are using managed node groups.

The way that managed node groups does this is:

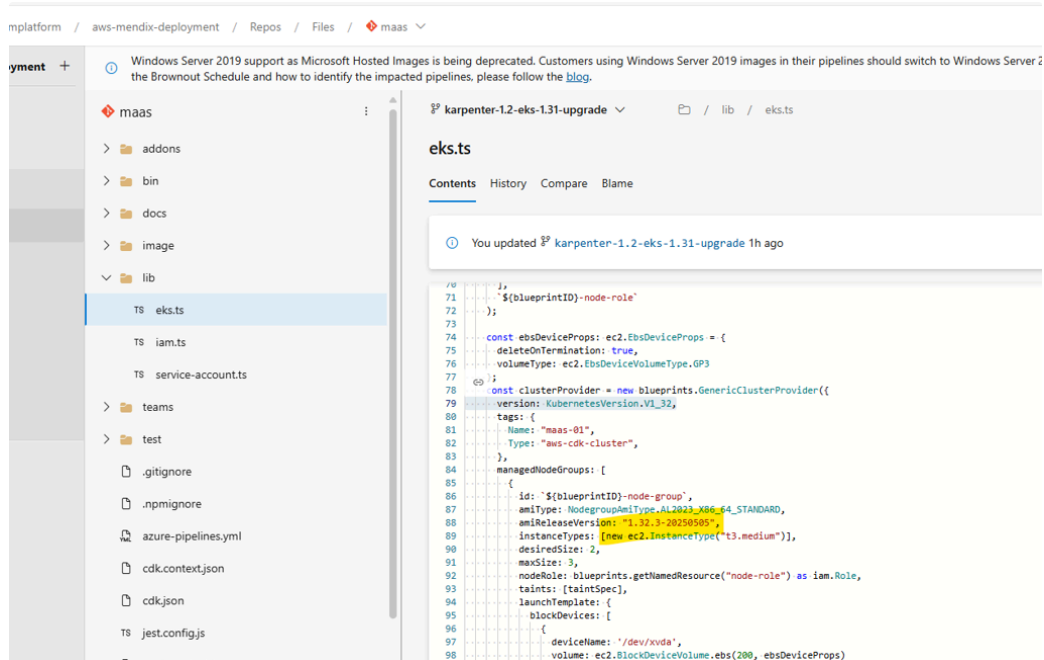
1. Amazon EKS creates a new Amazon EC2 launch template version for the Auto Scaling group associated with your node group. The new template uses the target AMI for the update.
2. The Auto Scaling group is updated to use the latest launch template with the new AMI.



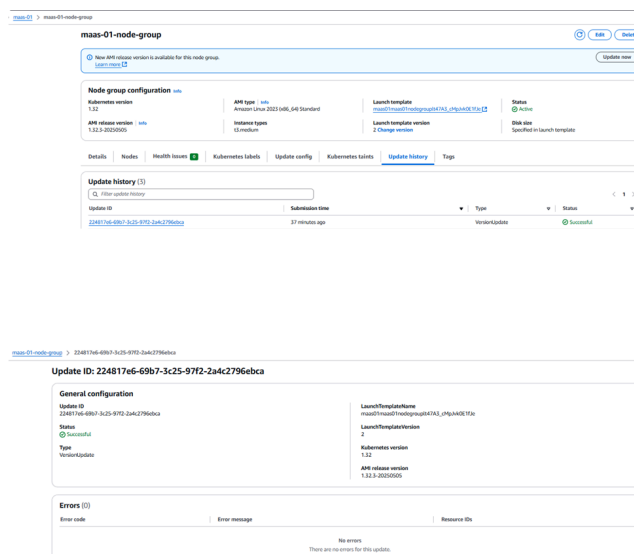
3. The Auto Scaling group maximum size and desired size are incremented by one up to twice the number of Availability Zones in the Region that the Auto Scaling group is deployed in. This is to ensure that at least one new instance comes up in every Availability Zone in the Region that your node group is deployed in.
4. Amazon EKS checks the nodes in the node group for the [eks.amazonaws.com/nodegroup-image](https://eks.amazonaws.com/nodegroup-image) label, and applies a [eks.amazonaws.com/nodegroup=unschedulable:NoSchedule](https://eks.amazonaws.com/nodegroup=unschedulable:NoSchedule) taint on all of the nodes in the node group that aren't labeled with the latest AMI ID. This prevents nodes that have already been updated from a previous failed update from being tainted.
5. Amazon EKS randomly selects a node in the node group and evicts all pods from it.
6. After all of the pods are evicted, Amazon EKS cordons the node. This is done so that the service controller doesn't send any new request to this node and removes this node from its list of healthy, active nodes.
7. Amazon EKS sends a termination request to the Auto Scaling group for the cordoned node.
8. Steps 5-7 are repeated until there are no nodes in the node group that are deployed with the earlier version of the launch template.
9. The Auto Scaling group maximum size and desired size are decremented by 1 to return to your pre-update values.
- 10.

#### Upgrade Node Groups

1. Navigate to the `maas` repository.
  - Update the `amiReleaseVersion` version for `1.32` in the `/lib/eks.ts` file.
  - Once updated the AMI release version for 1.32 EKS run the pipeline to upgrade the worker node



- **Monitor Progress:** Monitor the upgrade progress through the AWS Management Console and cloudformation template.
- **Verify Post-Upgrade:** Verify that the node group has been successfully upgraded and ensure that your applications are running correctly.



## Upgrade Karpenter managed nodes

1. Navigate to the **maas** repository.
  - Update the **amiSelector** ID for **1.32** in the **/addons/karpenter/index.ts** file.

- Once updated the AMI release version for 1.32 EKS run the pipeline.
2. Since we have added `do-not-evict` and `do-not-disrupt` annotation for pods karpenter cannot upgrade the nodes for that we need to drain the nodes manually to upgrade the karpenter managed nodes

**Identify Nodes:** Begin by identifying the nodes managed by Karpenter that need to be upgraded. You can use the following command to list nodes:

```
1 kubectl get nodes -l karpenter.sh/provisioner-name=<provisioner-name>
```

**Cordon Nodes:** For each node that needs to be updated, prevent new pods from being scheduled on it by running the `cordon` command:

```
1 kubectl cordon <node-name>
```

Replace `<node-name>` with the name of the specific node.

**Drain Nodes:** After cordoning, drain the node to safely evict running pods. This will ensure pods are rescheduled on other available nodes:

```
1 kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

- The `--ignore-daemonsets` flag prevents issues with daemonset-managed pods.
- The `--delete-emptydir-data` flag clears out temporary data stored in `emptyDir` volumes.

**Monitor Karpenter for New Nodes:** Once the nodes are drained, Karpenter should automatically detect the need for additional capacity and provision new nodes with the latest EKS version. To confirm this, you can monitor the creation of new nodes by running:

```
1 kubectl get nodes -w
```

This command will allow you to watch for new nodes coming online in real-time.

**Verify Node Versions:** After Karpenter provisions the new nodes, check that the new nodes are running the updated EKS version:

```
1 kubectl get nodes -o wide
```

Verify that the version listed for each new node matches the target EKS version.

Following these steps ensures a smooth upgrade for Karpenter-managed nodes in your EKS cluster.

### Post-Upgrade Verification

After upgrading an EKS managed node group, it's crucial to verify the following to ensure the cluster operates smoothly:

- **Node Group Version:** Check that all nodes in the upgraded node group reflect the desired Kubernetes version and AMI release version. This can be verified in the AWS Management Console under EKS Node Groups.
- **Pod Deployment:** Ensure pods are correctly deployed on the updated nodes. Use `kubectl get pods -o wide` to list pods and their nodes. Pods should be scheduled on the nodes that have been upgraded.
- **Pod Logs:** Review logs from pods running on the updated nodes to confirm they are functioning correctly. Use `kubectl logs <pod-name> -n <namespace>` to access specific pod logs.
- **Cluster Health Monitoring:** Monitor the overall health of the EKS cluster using AWS CloudWatch metrics and the Kubernetes dashboard. Look out for any anomalies or issues post-upgrade.
- **Application Validation:** Validate that applications running on the EKS cluster are operating as expected following the node group upgrade. Conduct functional testing of critical services and applications.

### Implementation

Date	Environment	Notes	Status
May 22, 2025	Dev		DONE
May 30, 2025	QA		DONE
Jun 14, 2025	Prod		PLANNING

### Lesson learnt from previous upgrade

1. [📄 Learning from EKS 1.22 upgrade activity](#)
2. [📄 Lesson learned from EKS 1.23 Dev upgrade activity](#)
3. [📄 Lesson learned from EKS 1.24 Dev & QA upgrade activity](#)

## Configuration Changes

- Update the [cluster creation pipeline](#) config file
- Update the environment variable library in azure devops
- Commit the changes to Azure Repos for Dev/Test but approvals required for QA and Prod.
- [Pipeline](#) gets triggered