Security Overview of Amazon EKS Auto Mode

First published September 18, 2025

Last updated September 18, 2025

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Introduction 2

Benefits 2

AWS Shared Security Responsibility Model 3

EKS control plane and data plane 3

- Amazon EKS control plane 4
  - Kubernetes API data 5
  - EKS Auto Mode capabilities 5
- EKS Auto Mode data plane 6
  - EC2 managed instances 6
  - Instance configuration 7
  - Node role and access entry 7
  - Node operating system 8
  - Node patching 9
  - Compute 10
  - Storage 10
  - Networking 10
  - Node component Kubernetes RBAC 12
- Workloads 12
  - Configuration 12
  - Runtime monitoring 13

Conclusion 13

Contributors 13

Further reading 13

Document revisions 14

Abstract

This paper is intended for existing and potential Amazon EKS customers who are using or considering EKS Auto Mode. It provides a comprehensive security overview of Auto Mode, which is helpful for new adopters and deepens understanding of Auto Mode for current customers.

Introduction

Since its introduction in 2018, Amazon Elastic Kubernetes Service (Amazon EKS) has provided a managed Kubernetes control plane integrated with existing AWS services, where Amazon Web Services (AWS) is responsible for the health, scaling, and patching of the control plane. Amazon EKS Auto Mode represents a significant evolution in Kubernetes infrastructure management, combining secure and scalable cluster infrastructure with integrated Kubernetes capabilities managed by AWS. We have extended the AWS managed portion of the control plane to include the worker nodes, their components and core cluster capabilities.

The result is a production-ready, Kubernetes-conformant cluster that is ready to host workloads out of the box. Customers who have previously used managed node groups (MNG) or Karpenter can transition to EKS Auto Mode, so they can focus on deploying their applications while Auto Mode handles the rest. This makes it an ideal solution for those who want to use Kubernetes without having to manage its underlying complexity.

To make this transition seamless, EKS Auto Mode has been designed to be compatible with existing clusters and their compute management. This allows transitioning the entirety or a subset of workloads to Auto Mode managed compute to minimize disruption.

Benefits

EKS Auto Mode is a new operating model for Amazon EKS. In addition to the reduced operational responsibility, there are several technical changes to further improve the security posture of Auto Mode nodes.

- **EC2 managed instances:** EKS Auto Mode uses Amazon Elastic Compute Cloud (Amazon EC2) managed instances to provide the compute that backs an Auto Mode node. This allows Auto Mode to provide the full breadth of Amazon EC2 capabilities while delegating operational control to Amazon EKS.
- **Minimal container optimized OS:** The operating system used on Auto Mode nodes is a variant of Bottlerocket, which is optimized solely for running containers.
- **Minimal permissions on the node role:** Auto Mode has been designed to require fewer permissions on the node's AWS Identity and Access Management (IAM) role.
- **AWS EKS managed compute, networking, and storage capabilities**: Auto Mode provides managed capabilities for these functions, which shifts the responsibility for health and patching of the components that normally provide them to AWS.
- **Frequent patching**: AWS is responsible for patching the components hosted on AWS infrastructure and the components in the Auto Mode node Amazon Machine Image (AMI).

- **Limited node lifetime:** It's a best practice to treat Kubernetes nodes as ephemeral compute providers. Auto Mode builds upon that by stopping nodes from living longer than 21 days.

AWS Shared Security Responsibility Model

Security and compliance is a shared responsibility between AWS and the customer. The AWS Shared Responsibility Model can help relieve the customer's operational burden because AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the service operates.

With EKS Auto Mode, AWS is responsible for the configuration, patching, and health of the EC2 instances so that customers can focus on the Amazon Virtual Private Cloud (Amazon VPC) and cluster configuration, and the application containers that they are running.

EKS Auto Mode accomplishes this by using EC2 managed instances. Using managed instances, customers can delegate operational control over the instances to the Amazon EKS service. EKS is then responsible for patching the components that are delivered as part of the AMI. This combines with the 21-day maximum node lifetime for Auto Mode, so that nodes are regularly replaced with newer nodes running the most recently released version of the AMI, containing the latest patches.

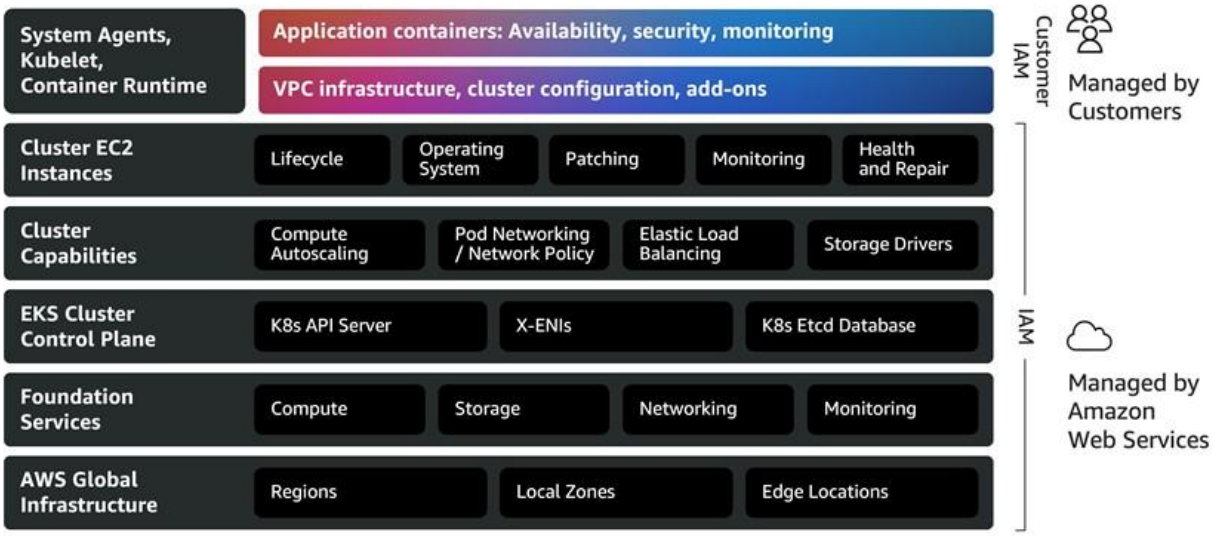

Figure 1: Shared Responsibility Model with EKS Auto Mode

EKS control plane and data plane

Amazon EKS operates a control plane that handles the AWS API calls responsible for high-level cluster management (such as `eks:CreateCluster` and `eks:UpdateClusterConfig`). That control plane is not covered in detail in this document; instead, this document focuses on the cluster-specific Kubernetes control plane and data plane. For information about securing the AWS APIs for cluster management, see the Security best practices in IAM guide.

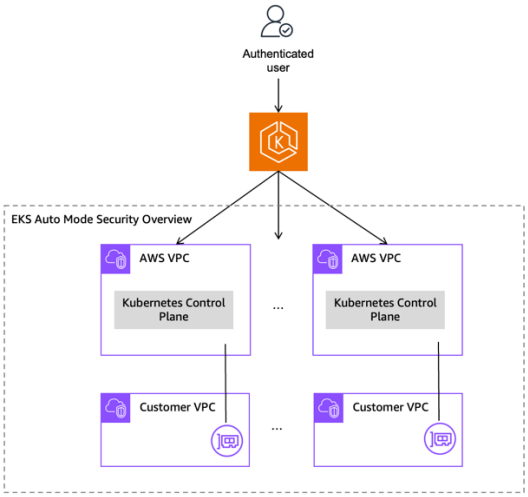

Figure 2: Security Overview covered content

Amazon EKS control plane

The Kubernetes control plane managed by Amazon EKS runs inside an EKS-managed VPC. This control plane is single tenant, meaning that for each EKS cluster there is a unique EKS managed VPC and Kubernetes control plane. The EKS control plane comprises the Kubernetes API server nodes and etcd cluster. Kubernetes API server nodes run components such as the API server, scheduler, and `kube-controller-manager` in an auto-scaling group. EKS runs a minimum of two API server nodes in distinct Availability Zones within an AWS Region. Likewise, for durability, the etcd server nodes also run in an auto-scaling group that spans three Availability Zones. EKS runs a NAT gateway in each Availability Zone, and API servers and etcd servers run in a private subnet. This architecture protects cluster availability so that an event in a single Availability Zone doesn't affect the EKS cluster's availability.

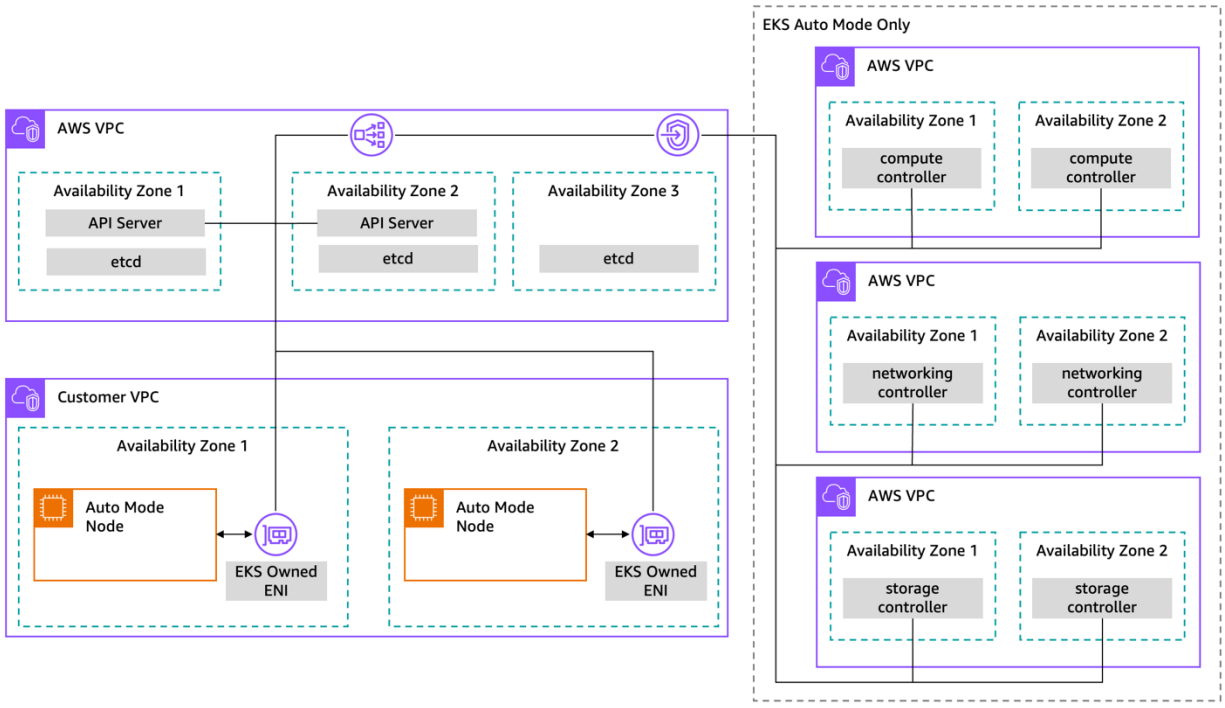

Figure 3: Amazon EKS high level architecture

Kubernetes API data

Amazon EKS provides default envelope encryption for Kubernetes API data in EKS Auto Mode clusters. Envelope encryption protects the data you store with the Kubernetes API server. For example, envelope encryption applies to the configuration of your Kubernetes cluster, such as `ConfigMaps`. Envelope encryption does not apply to data on nodes or Amazon Elastic Block Store (Amazon EBS) volumes. This envelope encryption extends across Kubernetes API data.

This provides a managed, default experience that implements defense-in-depth for your Kubernetes applications and doesn't require action on your part.

Amazon EKS uses AWS Key Management Service (AWS KMS) with Kubernetes KMS provider v2 for this additional layer of security with an AWS owned key and the option for you to bring your own customer managed key (CMK) from AWS KMS.

EKS Auto Mode capabilities

When EKS Auto Mode is enabled for a cluster, an additional set of control plane capabilities are also enabled. In a standard Amazon EKS cluster, the components that perform auto-scaling, manage Elastic Network Interfaces (ENIs) and Amazon EBS devices run as Kubernetes Pods on nodes in the cluster. With Auto Mode, AWS manages these components and runs them outside of the cluster. This shifts the operational responsibility for the health and patching of these components to AWS and moves the components that require sensitive permissions to operate outside of the cluster.

To manage compute, networking, and storage, EKS Auto Mode requires additional permissions beyond those required in a non-Auto Mode cluster. These are normally provided by adding a set of policies to the cluster IAM role. However, there is no requirement to attach these specific policies to the cluster role. Custom policies can be used provided they grant sufficient permissions to Auto Mode components.

The cluster IAM role used by EKS Auto Mode is a service role. A service role is an IAM role that a service assumes to perform actions on your behalf. Service control policies (SCPs) apply to the actions performed by these roles and can be used to further restrict Auto Mode capabilities, such as limiting the instance types that can be launched. This differs from a service-linked role (SLR), which is a type of role that is linked to an AWS service and is not restricted by SCPs. Auto Mode minimizes its use of SLRs so that SCPs are respected when possible.

Note: Additional guidance for adjusting SCPs to allow Auto Mode to function can be found in Update organization controls for EKS Auto Mode.

EKS Auto Mode data plane

The EKS Auto Mode data plane consists primarily of the Auto Mode nodes, the compute that your workloads run directly on. The data plane is built to give you full flexibility and control with respect to the types of workloads that can be run and the instances that they run on while still delegating operational responsibility for the scaling and health of that data plane to AWS.

EC2 managed instances

EKS Auto Mode uses EC2 managed instances to provide the compute that backs an Auto Mode node. These nodes have built-in IAM-enforced restrictions that block operations on the EC2 instances that could compromise the ability of AWS to operate the nodes. For example, it's not possible to change the instance profile of a node or attach or detach ENIs. Instead, the instance role is controlled using the NodeClass and ENI management is performed by a networking capability that is managed by AWS and hosted on AWS infrastructure. These restrictions are applied regardless of the IAM identity and its permissions. Even the AWS account root user is unable to circumvent these constraints.

The IAM-enforced restrictions extend past the EC2 instance itself. It also includes the Amazon EBS volumes that are attached to the instance at launch, ENIs, and the launch templates used for launching those managed instances.

EC2 managed instances does not provide Amazon EKS additional permissions to EKS Auto Mode nodes. The permissions that Amazon EKS uses to manage those instances are still granted only by the cluster service role and EKS SLR.

By building on top of EC2 managed instances, the EC2 features that customers are familiar with work as expected. With EKS Auto Mode, customers can continue to use capacity reservations and savings plans. Auto Mode allows full control over the instance types that are launched, providing access to the broad range of EC2 instance types including accelerated types for machine learning inferencing and training use cases.

Instance configuration

EKS Auto Mode enforces a few best practices related to security during instance launch. Because the instances are EC2 managed instances, they cannot be changed at runtime. This includes the configuration for Instance Metadata Service (IMDS) and encryption of the root and data Amazon EBS volumes.

IMDS is configured to use IMDSv2 (token required) with a hop limit of one, which is the maximum number of hops that the metadata token can travel. This blocks non-host-network Pods from accessing IMDS, through which it could access the node's IAM credentials.

On EKS Auto Mode nodes, the root and data Amazon EBS volumes are encrypted and configured to be deleted upon termination of the instance. Optionally the NodeClass `ephemeralStorage.kmsKeyID` setting can be used to specify the encryption key to be used.

Node role and access entry

EKS access entry is the recommended mechanism to grant an IAM principal access to the Kubernetes API. Each access entry has a type, Kubernetes username, and list of Kubernetes groups. Depending on the access entry type, the username and groups might not be configurable. Some access entry types can optionally have an association created with access policies. These access policies provide further permissions to the IAM principal, beyond what might be granted based on the Kubernetes username and group.

The standard EKS Auto Mode node access entry is of type `EC2`, which has a Kubernetes username of `system:node:{{SessionName}}` and is in the `system:nodes` group with the `AmazonEKSAutoNodePolicy` access policy attached. When using an EC2 instance profile to assign an IAM role to an EC2 instance, the `SessionName` is automatically set to the instance ID, leading to a Kubernetes username of `system:node:i-1234567890abcdef0` which corresponds to a Kubernetes node name of just the instance ID, `i-1234567890abcdef0`.

The default Amazon EKS Auto Mode IAM role uses a new AmazonEKSWorkerNodeMinimalPolicy policy. This policy removes nine different permissions from the previous AmazonEKSWorkerNodePolicy , retaining only the permissions required for EKS Auto Mode nodes to operate. The AmazonEC2ContainerRegistryPullOnly policy, while generally useful, was also created while building Auto Mode to further reduce the number of Elastic Container Registry (ECR) permissions made available to nodes compared to the existing AmazonEC2ContainerRegistryReadOnly policy. Lastly, Auto Mode nodes use the EC2 instance ID as the Kubernetes node name. Because the instance ID is reliably determined through IMDS, the node role no longer needs permissions to call `ec2:DescribeInstances` to discover the private DNS name.

Node operating system

The operating system for EKS Auto Mode nodes is a custom variant of Bottlerocket. Bottlerocket was selected as the underlying operating system for Auto Mode nodes because it is optimized and built specifically for running containers and has several security enhancements over a general-purpose operating system. It enforces cryptographic integrity checks for the root file system and mandatory access controls using SELinux to reduce the attack surface in the event of container escape. The reduced number of packages in Bottlerocket minimizes the surface area for potential security issues and reduces the effort required by many compliance programs to keep hosts updated with the latest security patches.

In Bottlerocket, most non-privileged pods will automatically have their own SELinux multi-category security (MCS) label applied to them. This MCS label is unique to each Pod and is designed to protect against a process in one pod manipulating a process in another Pod or on the host. Even if a labeled Pod runs as root and has access to the host filesystem, it will be unable to manipulate files, make sensitive system calls on the host, or access the container runtime.

The EKS Auto Mode Bottlerocket variant hardens the standard Bottlerocket configuration by disabling features like host containers, which while useful in the standard Bottlerocket distribution are not used in Auto Mode. In addition, remote access services like SSH and the AWS Systems Manager agent are not available on Auto Mode nodes. While direct remote access isn't allowed, it is still possible to troubleshoot the node in multiple ways.

- NodeDiagnostic resource – The NodeDiagnostic custom resource definition (CRD) is a Kubernetes-native method of fetching system logs and information from an EKS Auto Mode node. The collected logs are uploaded automatically to an Amazon Simple Storage Service (Amazon S3) bucket. By design, a pre-signed Amazon S3 URL is used, which enables collecting logs from nodes without requiring that S3 permissions be added to the node role. The ability to collect logs is controlled by limiting access to create the NodeDiagnostic object through standard Kubernetes role-based access control (RBAC).
- Console output logs –Auto Mode periodically writes system information to the Amazon EC2 console, which can be useful for debugging issues related to permissions or network configuration issues that stop the node from joining the cluster.
- Debug containers – Because Auto Mode is Kubernetes conformant, standard debug containers can be used to inspect and fetch system logs on the node.

Note: EKS Auto Mode nodes are Kubernetes conformant and because of this it's possible to run Pods on Auto Mode nodes that provide an SSH service or run the SSM agent. In this case, the remote access session is to the Pod itself and resides within the container boundary.

The EKS Auto Mode variant of Bottlerocket is built on the core open source Bottlerocket distribution but adds several Auto Mode specific packages to handle things like the OS level configuration of network interfaces that have been attached to the instance by the AWS managed networking component. When launching instance types with Neuron or NVIDIA accelerators, a specific version of the Auto Mode

operating system is used that contains the appropriate drivers and Kubernetes device plugins to make these nodes compatible with accelerated workloads without requiring further software installation or configuration.

Node patching

Auto Mode nodes are updated by replacing the instance with a new instance running the latest Auto Mode AMI. This process allows workloads to gracefully migrate from the unpatched node to the patched node following Pod Disruption Budgets (PDBs) that govern the workload availability in addition to the disruption controls configured at the NodePool level.

The Auto Mode AMIs undergo a rigorous testing process prior to being released. This includes:

- Common vulnerabilities and exposures (CVE) scanning of included components
- Full Kubernetes node conformance tests
- Component functional testing (for example, validating that pods can obtain IAM credentials through EKS Pod Identity)
- Security related testing (for example, testing that the node has only the expected services listening)
- Functional testing of compatibility with both Neuron and NVIDIA accelerators

Auto Mode AMIs are rolled out using standard AWS best practices for safe, hands-off deployments. These deployments are built around an internal AWS construct called a *pipeline*, which automates the build and deployment process and provides automated alarm monitoring, testing, and other validation of safety. The process begins by deploying the newly built and tested AMI to a small subset of EKS Auto Mode clusters in a single Region, with a bake time to detect potential issues. As confidence in the AMI stability grows, it is gradually rolled out to more clusters in larger waves and across more Regions, while reducing the bake time between deployments. There is additional gating included in the deployment pipeline so that by default a new AMI is made available no more than once per week.

The default EKS Auto Mode NodePools allow nodes to be replaced through *drift* after a new AMI has been made available for their EKS cluster. Customers optionally can create their own NodePool disruption windows to control when and how quickly nodes are updated.

The built-in EKS Auto Mode NodePools have a configured node expiration of 14 days, but customers can create their own NodePools to raise or lower this value. To receive patches as soon as they are made available, NodePools should not use the `disruption.budgets[].schedule` setting, which restricts the time windows that a node can be replaced.

If PDBs or NodePool disruption controls do not allow a node to be replaced before the 21-day maximum node lifetime has been reached, the node will be disrupted regardless. This helps make sure that nodes periodically receive security patches and updates, and that a misconfigured PDB or other failing workload can't indefinitely stop a node from being replaced.

Compute

When using EKS Auto Mode, the AWS managed compute capability is responsible for the auto-scaling of Kubernetes worker nodes. Scaling configuration is performed using the standard Karpenter concepts of a NodePool and NodeClass. The NodePool is a standard Karpenter NodePool. The NodeClass is Auto Mode specific, which is the Karpenter mechanism for cloud provider specific extensions.

EKS Auto Mode supports two built-in NodePools, named `system` and `general-purpose` that can optionally be enabled. The `system` NodePool has a `CriticalAddonsOnly` taint and is designed to separate cluster-critical applications from other workloads. The `general-purpose` NodePool has no taints and is designed to run other non-accelerated workloads in your cluster. The built-in NodePools, by virtue of being created and configured using the `eks:CreateCluster` and `eks:UpdateClusterConfig` API calls, allow infrastructure as code (IaC) tooling to create EKS clusters that can run workloads immediately after cluster creation without requiring further interaction with the Kubernetes API to create a NodePool and NodeClass.

Storage

EKS Auto Mode nodes launch with two attached Amazon EBS volumes that share the instance's lifetime. The first is the root volume which contains the Bottlerocket operating system, while the second is the data volume that contains ephemeral data such as Pod logs, container images, and so on. Both volumes are encrypted by default with EKS Auto Mode using an AWS managed key. Optionally, customers can configure a CMK to be used for encryption of these volumes.

The block storage capability of EKS Auto Mode used for persistent volumes backed by Amazon EBS can optionally be configured to encrypt those EBS volumes by default, including with a CMK.

Networking

The managed networking capability of EKS Auto Mode runs on AWS infrastructure and is responsible for two separate activities. First, it handles the lifetime and attachment of ENIs to the managed instance as needed to handle the Pods scheduled to the node. Second, it handles the lifetime and configuration of load balancers that are required to support the `IngressClass` with a controller of type `eks.amazonaws.com/alb`.

The NodeClass controls the subnets and security groups that are used for Auto Mode nodes and Pods running on those nodes through the `subnetSelectorTerms`, `securityGroupSelectorTerms`, `podSubnetSelectorTerms`, and `podSecurityGroupSelectorTerms`. The `subnetSelectorTerms` and `securityGroupSelectorTerms` settings are required. If only these settings are provided, both the node and Pods will share the same subnets and security groups. The node IP and subsequent Pod IP addresses will be allocated from the primary ENI and additional ENIs will be dynamically created and attached to the node to support Pods as needed.

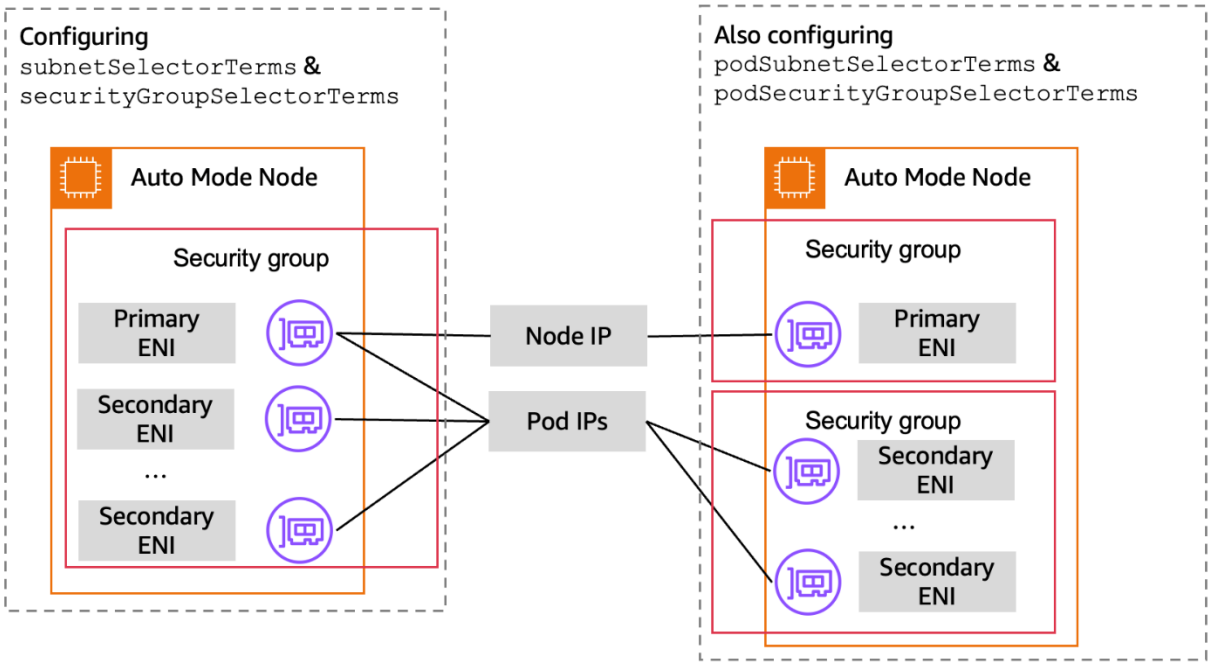

Figure 4: NodeClass IP assignment settings

If `podSubnetSelectorTerms` and `podSecurityGroupSelectorTerms` are also configured, then only the node's IP will come from the primary ENI. Pod IPs will come from secondary ENIs and use the specified security groups. This mode of operations allows segregating the node IP addresses from Pod IP addresses, primarily to allow using separate security groups to control traffic flow for nodes and Pods differently. Because the primary ENI is reserved for only the node IP address, when operating in this configuration the result is reduced Pod density on Auto Mode nodes.

After being enabled on the cluster, Pod-to-Pod traffic can be controlled by using standard Kubernetes NetworkPolicies. These policies are enforced by a networking component on the node using eBPF.

The EKS Auto Mode NodeClass also offers several settings for more advanced networking use cases:

- `advancedNetworking.httpsProxy` and `advancedNetworking.noProxy` – Controls the `HTTPS_PROXY` and `NO_PROXY` settings for containerd and kubelet.
- `certificateBundles` – Certificate bundles for custom certificate authorities (CA) to be trusted by the node. This is most often used when pulling container images from a private container registry that uses self-signed certificates.
- `advancedNetworking.associatePublicIPAddress` – Controls the setting of the `AssociatePublicIpAddress` property on the launch template used for launching EKS Auto Mode nodes. This setting will need to be set to false if SCPs require it to allow Auto Mode to launch EC2 instances.

Node component Kubernetes RBAC

Several of the built-in node components require access to the Kubernetes API server to function. For example, the DNS component needs to list services and the node monitoring component needs to access NodeDiagnostic resources to respond to log collection requests. This access is provided by the AmazonEKSAutoNodePolicy access policy.

Instead of providing the union of all permissions to the kubelet RBAC identity through this access policy, a more restrictive approach was taken. The components begin by using the kubelet's identity and then use standard Kubernetes impersonation to assume an identity with only the specific permissions that the component needs. After this identity assumption has occurred, the component only has the new permissions. This will be visible in the Kubernetes audit logs by the addition of an `impersonatedUser` property on the audit event:

```
"impersonatedUser": {
    "username": "eks-auto:component-name",
    "groups": [
        "system:authenticated"
    ]
}
```

Workloads

With EKS Auto Mode, customers continue to maintain responsibility for their application containers, including availability, security, and monitoring. Auto Mode provides a solid foundation to build upon, but there are several areas where following EKS best practices can improve the security posture of those workloads.

Configuration

Because EKS Auto Mode nodes are Kubernetes conformant, standard Pod-level configurations work as expected. For example, the Pod `securityContext` field can be used to give additional permissions to Pods and `volumeMounts` can be used to provide access to the host filesystem. Even then, Pods however still face the restrictions provided by SELinux and a read-only root filesystem on the node. You can use Kubernetes policy enforcement tools like Kyverno or OPA Gatekeeper to limit Pod-level configuration within a cluster. Additional guidance for Pod security can be found in the EKS Best Practices guide.

To vend IAM credentials to Pods within a cluster, EKS Auto Mode nodes include built-in support for EKS Pod Identity. When a Pod is launched using a Kubernetes service account that is configured with Pod Identity, the Kubernetes control plane injects a set of environment variables into the Pod. These environment variables cause the AWS SDK to request credentials from the Pod Identity component that Auto Mode has preconfigured on the Node. This process involves the AWS SDK fetching the Pod's

service account token, assigned by the Kubernetes API server, and exchanging it for IAM credentials via the `eks-auth:AssumeRoleForPodIdentity` API. This is the only permission on the managed AmazonEKSWorkerNodeMinimalPolicy policy.

Note: IAM roles for service accounts (IRSA) can also be configured to provide credentials to Pods, while Pod Identity remains the recommended method.

Runtime monitoring

Runtime monitoring observes and analyzes operating system level, networking, and file events to help you detect potential threats in the workloads in your environment. This can include detection of issues such as container breakouts, creation of reverse shells, or elevation of privileges.

Because EKS Auto Mode nodes are fully Kubernetes conformant, runtime monitoring systems that are compatible with Kubernetes nodes should work with Auto Mode nodes. We recommend using Amazon GuardDuty or a third-party solution that is validated to work with Auto Mode for runtime monitoring. The full list of runtime issues that GuardDuty can detect is available in GuardDuty Runtime Monitoring finding types.

Conclusion

Amazon EKS Auto Mode represents a significant evolution in how customers can run Kubernetes on AWS. This whitepaper covers the security related aspects of EKS Auto Mode, including some of the design decisions that customers can make to shift their focus from infrastructure management to application development.

Contributors

Contributors to this document include:

- ***Todd Neal***, Principal Engineer, Amazon EKS

Further reading

For additional information, see:

- Amazon EKS Auto Mode User Guide
- Amazon EKS Best Practices Guide
- Shared Responsibility Model
- Security Best Practices in IAM
- Update organization controls for EKS Auto Mode
- Under the Hood: Amazon EKS Auto Mode

Document revisions

Date	Description
September 18, 2025	First publication