WHO AM I

🧑‍💻 I'm **Saifeddine RAJHI**, a **Data Platform Engineer** with a strong focus on **Kubernetes (K8s)** and **AMAZON EKS**.

As an **AWS Community Builder** in the container category, I work extensively with container technologies in the AWS cloud. I am also deeply involved in **DevSecOps** and **Cloud native projects**. I am passionate about **SRE practices** and enjoy sharing my knowledge as a **Tech storyteller**.

CONTACTS

📫 Reach out on:

- **Website:** https://seifrajhi.github.io/
- **X:** https://x.com/RajhiSaifeddine
- **LinkedIn:** https://www.linkedin.com/in/rajhi-saif/
- **Reddit:** https://www.reddit.com/user/ScoreApprehensive992
- **Stackoverflow:** https://stackoverflow.com/users/21897244/saifeddine-rajhi

AMAZON EKS Pocket guide

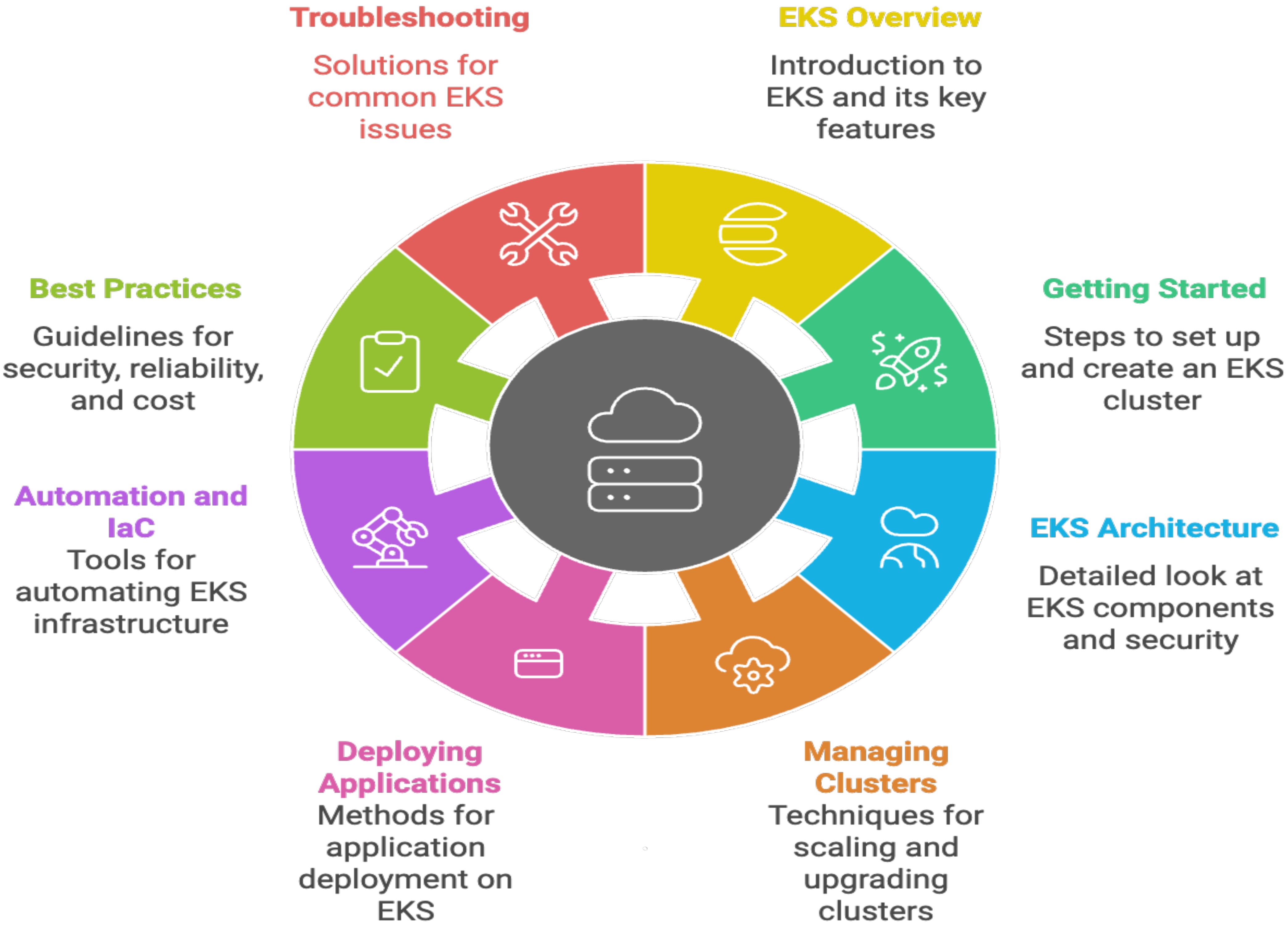
A handy guide covering all essential aspects of AWS EKS

v0.1

Saifeddine RAJHI

Senior Data platform Engineer
AWS Community Builder

TABLE OF CONTENTS



EKS Overview

Introduction to EKS and its key features

Getting Started

Steps to set up and create an EKS cluster

EKS Architecture

Detailed look at EKS components and security

Managing Clusters

Techniques for scaling and upgrading clusters

Deploying Applications

Methods for application deployment on EKS

Automation and IaC

Tools for automating EKS infrastructure

Best Practices

Guidelines for security, reliability, and cost

Troubleshooting

Solutions for common EKS issues

EKS OVERVIEW

Introduction to Amazon EKS:

Amazon Elastic Kubernetes Service (EKS) is a managed Kubernetes service that simplifies the deployment, management, and scaling of containerized applications using Kubernetes on AWS. EKS abstracts the complexities of Kubernetes management, allowing developers to focus on building and running applications.

Key Features and Benefits:

- **Managed Control Plane**: EKS manages the Kubernetes control plane, which includes the etcd database, API server, and controller manager. This ensures high availability and scalability across multiple AWS Availability Zones. The control plane is automatically patched and updated by AWS, reducing the operational burden on users.

- **Integrated with AWS Services**: EKS integrates with various AWS services to enhance functionality and security:

    - **IAM**: Use AWS Identity and Access Management (IAM) for fine-grained access control to Kubernetes resources.
    - **VPC**: EKS clusters run within an Amazon Virtual Private Cloud (VPC), providing isolation and security. You can configure VPC subnets, security groups, and network ACLs to control traffic.
    - **CloudWatch**: Monitor and log cluster activity using Amazon CloudWatch. EKS supports CloudWatch Container Insights for detailed metrics and logs.

- **High Availability**: EKS control plane nodes are distributed across multiple Availability Zones, ensuring resilience against zone failures. The control plane scales horizontally to handle increased load and provides automatic failover.

- **Scalability**: EKS supports the Kubernetes Cluster Autoscaler and AWS Auto Scaling groups. The Cluster Autoscaler adjusts the number of nodes in your cluster based on resource demands, while AWS Auto Scaling ensures that your applications have the right amount of compute capacity.

- **Security**: EKS supports Kubernetes network policies for controlling traffic between pods. It integrates with AWS Key Management Service (KMS) for encrypting secrets and supports AWS PrivateLink for private connectivity to the EKS API server.

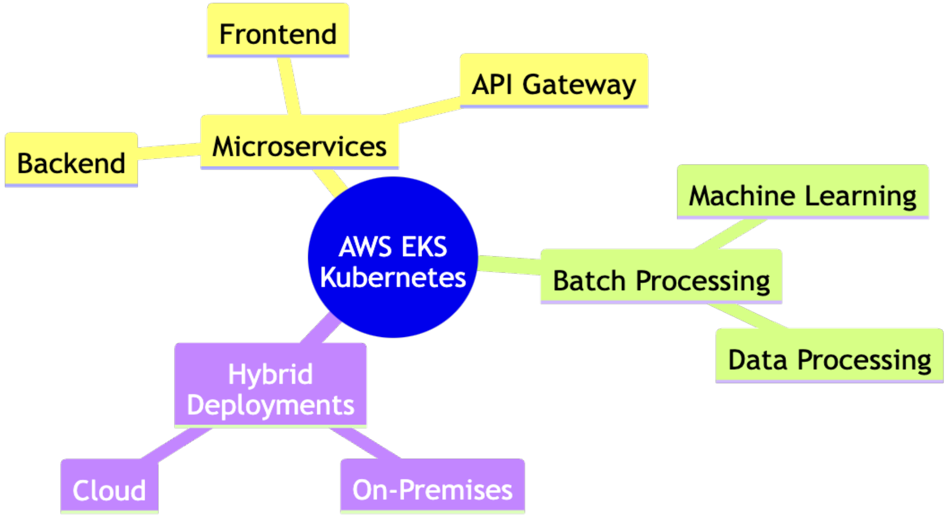
Use Cases:

- **Microservices**: EKS is ideal for microservices architectures, allowing you to deploy, manage, and scale individual services independently. Kubernetes' service discovery and load balancing features simplify communication between microservices.

- **Batch processing**: Use EKS for batch processing workloads. Kubernetes' job and cron job resources allow you to define and manage batch jobs, ensuring efficient use of compute resources.

- **Machine Learning**: Deploy and scale machine learning models on EKS. Kubernetes' support for GPUs and custom resource definitions (CRDs) enables you to run complex ML workloads. Integrate with AWS services like SageMaker for a complete ML pipeline.

- **Hybrid deployments**: EKS supports hybrid cloud deployments, allowing you to run Kubernetes clusters both on-premises and in the cloud. Use AWS Outposts or EKS Anywhere to extend your EKS clusters to your data center.



GETTING STARTED

Prerequisites:

Before you begin setting up Amazon EKS, ensure you have the following:

- **AWS Account**: You need an active AWS account. If you don't have one.
- **IAM Permissions**: Ensure you have the necessary IAM permissions to create and manage EKS clusters. This typically includes permissions for EC2, VPC, IAM, and EKS services.
- **AWS CLI**: Install and configure the AWS Command Line Interface (CLI).
- **kubectl**: Install kubectl, the Kubernetes command-line tool, which you will use to interact with your EKS cluster.
- **eksctl**: Install eksctl, a simple command-line utility for creating and managing EKS clusters.

Setting Up your AWS environment:

1. **Configure AWS CLI**: Use the following command to configure your AWS CLI with your credentials using the command:

```
➢ aws configure
```

2. **Create a VPC**: EKS requires a VPC with specific settings. You can create a VPC using the AWS Management Console, AWS CLI, or CloudFormation. Ensure your VPC has:
   - At least two subnets in different Availability Zones.
   - Proper route tables and internet gateways for public subnets.
   - Security groups configured to allow necessary traffic.

Creating Your First EKS Cluster:

1. **Create an EKS Cluster**: Use eksctl to create your EKS cluster. The following command creates a basic cluster with one node group:

```
➢ eksctl create cluster --name my-cluster --region eu-west-1 --nodegroup-name standard-workers --node-type t3.medium --nodes 3 --nodes-min 1 --nodes-max 4 --managed
```

This command specifies the **cluster name**, **region**, **node group name**, **instance type**, and the **number of nodes**.

2. **Configure kubectl**: After the cluster is created, configure kubectl to use the new cluster:

```
➢ aws eks --region eu-west-1 update-kubeconfig --name my-cluster
```

This command updates your kubeconfig file with the new cluster's information.

3. **Verify Cluster Setup**: Ensure your cluster is up and running by checking the nodes:

```
➢ kubectl get nodes
```

You should see a list of nodes that are part of your EKS cluster.

EKS ARCHITECTURE

Cluster components:

Amazon EKS architecture consists of several key components that work together to manage and run Kubernetes clusters:

1. **Control plane**:
   - **Kubernetes API Server**: Manages the Kubernetes API and serves as the entry point for all administrative tasks.
   - **etcd**: A consistent and highly available key-value store used as Kubernetes' backing store for all cluster data.
   - **Kubernetes Scheduler**: Assigns workloads to specific nodes based on resource availability and other constraints.
   - **Kubernetes Controller Manager**: Runs controller processes that regulate the state of the cluster, such as node lifecycle and replication controllers.

2. **Worker nodes**:
   - **EC2 Instances**: These are the virtual machines that run your containerized applications. They can be managed by EKS or self-managed.
   - **Kubelet**: An agent that runs on each worker node and ensures containers are running in a Pod.
   - **Kube-proxy**: Maintains network rules on nodes and enables communication between Pods.

3. **Add-ons**:
   - **CoreDNS**: Provides DNS-based service discovery within the cluster.
   - **Amazon VPC CNI**: Manages network interfaces for Pods, allowing them to communicate within the VPC.

Networking and Security:

Networking and security are critical aspects of EKS architecture, ensuring secure and efficient communication within and outside the cluster

1. **Networking**:

   - **Amazon VPC**: EKS clusters are deployed within an Amazon Virtual Private Cloud (VPC), providing isolation and control over network configurations.
   - **Subnets**: Clusters can span multiple subnets across different Availability Zones for high availability.
   - **Security Groups**: Act as virtual firewalls to control inbound and outbound traffic to the nodes.
   - **Network Policies**: Kubernetes Network Policies can be used to control traffic flow between Pods.

2. **Security**:

   - **IAM Roles and Policies**: EKS uses AWS Identity and Access Management (IAM) to control access to AWS resources. Each node and service can be assigned specific IAM roles and policies to limit permissions.
   - **Kubernetes RBAC**: Role-Based Access Control (**RBAC**) is used within Kubernetes to define permissions for users and applications.
   - **Encryption**: Data at rest can be encrypted using AWS Key Management Service (KMS). Secrets and configuration data can be stored securely using Kubernetes Secrets.

IAM Roles and Policies:

IAM roles and policies are essential for managing permissions and access control in EKS:

1. **Node IAM Roles**:

   - Each worker node can be assigned an IAM role that grants it permissions to interact with other AWS services, such as pulling container images from Amazon ECR or writing logs to Amazon CloudWatch.

2. **Service Account IAM Roles**:

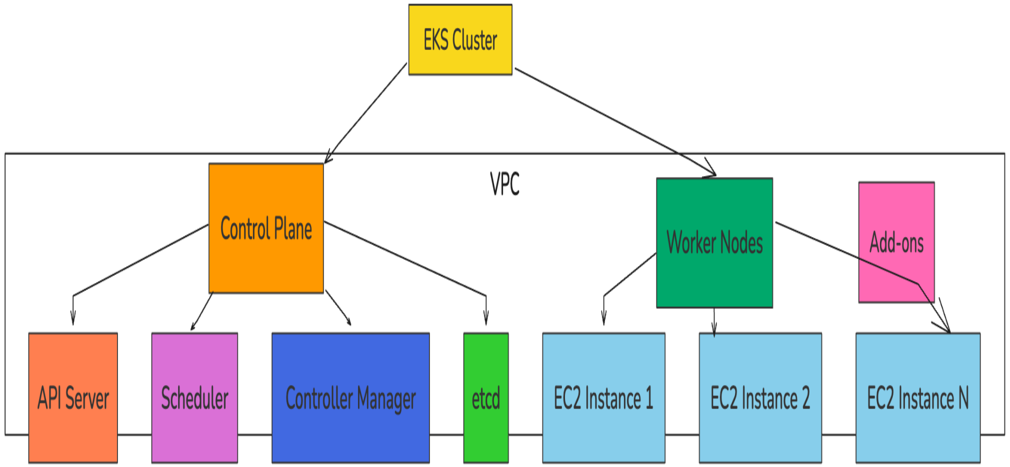
Kubernetes service accounts can be mapped to IAM roles using IAM Roles for Service Accounts (IRSA). This allows fine-grained permissions for Pods, enabling them to securely access AWS resources.

3. **EKS pod Agent identify:**

Amazon EKS Pod Identity provides credentials to your workloads with an additional EKS Auth API and an agent pod that runs on each node.

4. **Cluster IAM Roles**:

   - **Cluster Role**: Grants permissions to the EKS control plane to manage the cluster.
   - **Node Instance Role**: Assigned to EC2 instances to allow them to join the cluster and communicate with the control plane.



MANAGING EKS CLUSTERS

Scaling clusters:

- **Horizontal Pod Autoscaler (HPA)**: Automatically scales the number of pods based on CPU/memory usage or custom metrics.
- **Cluster Autoscaler**: Adjusts the number of nodes in your cluster based on the resource requests of your pods.
- **Karpenter**: An open-source Kubernetes cluster autoscaler built by AWS to improve the efficiency and cost-effectiveness of scaling.

Upgrading Kubernetes versions:

- **Managed Node Groups**: Simplifies the process of upgrading nodes by managing the lifecycle of EC2 instances.
- **In-place Upgrades**: This involves upgrading the control plane and nodes without downtime. AWS manages the control plane upgrades, ensuring minimal disruption. For the data plane (nodes), you can perform rolling updates where nodes are upgraded one at a time, maintaining cluster availability.
- **Version Compatibility**: Ensuring compatibility means verifying that your applications and add-ons work with the new Kubernetes version. This involves checking for deprecated APIs and updating your manifests and configurations accordingly.

Monitoring and Logging:

- **Amazon CloudWatch**: Collects and visualizes metrics and logs from your EKS clusters.
- **AWS X-Ray**: Helps with tracing and debugging applications running on EKS.
- **Prometheus and Grafana**: Popular open-source tools for monitoring and alerting on Kubernetes clusters.

DEPLOYING APPLICATIONS ON EKS

Using kubectl with EKS:

- **kubectl Configuration**: Set up your kubeconfig file to connect kubectl to your EKS cluster.
- **Basic Commands**: Deploy, manage, and troubleshoot applications using kubectl commands.
- **Namespaces**: Organize your resources within the cluster using Kubernetes namespaces.

Deploying with helm:

- **Helm charts**: Use pre-configured templates (charts) to deploy applications.
- **Chart repositories**: Store and share Helm charts using repositories like Helm Hub or your own S3 bucket.
- **Release management**: Manage application releases and rollbacks with Helm.

CI/CD Integration:

- **GitHub actions:** Integrate GHA with EKS for continuous integration and delivery.
- **Jenkins**: Integrate Jenkins with EKS for continuous integration and delivery.
- **GitOps**: Use tools like ArgoCD or Flux to implement GitOps for managing your Kubernetes deployments.

AUTOMATION AND INFRASTRUCTURE AS CODE IAC

Terraform for EKS

- **Provisioning Clusters**: Terraform allows you to define and provision EKS clusters using code. This includes creating VPCs, subnets, security groups, and the EKS cluster itself.
- **Modular Approach**: Use Terraform modules to reuse configurations across different environments, ensuring consistency and reducing errors.
- **State Management**: Terraform keeps track of the state of your infrastructure, making it easier to manage updates and rollbacks.

AWS CloudFormation for EKS:

- **Template-Based Provisioning**: AWS CloudFormation uses JSON or YAML templates to describe and provision all the resources needed for your EKS cluster, including the control plane and node groups.
- **Stack Management**: Manage your infrastructure as a single unit (stack), which can be updated or deleted in a controlled manner.
- **Integration with Other AWS Services**: Easily integrate with other AWS services like IAM, VPC, and CloudWatch for a comprehensive infrastructure setup.

Crossplane for EKS:

- **Kubernetes-Native IaC**: Crossplane extends Kubernetes to manage cloud infrastructure using Kubernetes-style APIs. This allows you to manage AWS resources like RDS, S3, and EKS clusters directly from Kubernetes.
- **Compositions and XRDs**: Define reusable infrastructure blueprints using Crossplane's Compositions and Composite Resource Definitions (XRDs), enabling a high level of abstraction and reuse.

OpenTofu for EKS:

Provisioning Clusters:

- **Infrastructure as Code:** OpenTofu allows you to define and provision EKS clusters using code, like Terraform.
- **Resource Management:** Create and manage resources such as VPCs, subnets, security groups, and the EKS cluster itself.

Modular Approach:

- **Reusable Modules:** Leverage OpenTofu modules to reuse configurations across different environments, reducing duplication and errors.

State Management:

- **State Files:** OpenTofu maintains state files to keep track of your infrastructure, making it easier to manage updates and rollbacks.
- **Remote State Storage:** Store state files remotely for collaboration and backup purposes.

Pulumi for EKS:

Provisioning Clusters:

- **Code-Driven Infrastructure:** Pulumi allows you to define and provision EKS clusters using familiar programming languages like TypeScript, Python, Go, and C#.
- **Resource Management:** Create and manage resources such as VPCs, subnets, security groups, and the EKS cluster itself through code.
- **Rich SDKs:** Utilize Pulumi's rich SDKs to interact with AWS services and manage your infrastructure.

GitOps with Flux and ArgoCD:

- **Declarative Configuration**: GitOps uses Git repositories as the source of truth for declarative infrastructure and application configurations. Flux and ArgoCD continuously reconcile the state of your cluster with the desired state defined in Git.
- **Continuous Delivery**: Automate the deployment of applications and infrastructure changes using CI/CD pipelines integrated with GitOps tools.
- **Multi-Cluster Management**: Manage multiple EKS clusters from a central repository, ensuring consistency and simplifying operations across environments.

BEST PRACTICES FOR EKS

Security:

1. Identity and Access Management (IAM):

IAM is essential for managing access to AWS resources, involving:

- **Authentication**: Verifying the identity of a user or service.
- **Authorization**: Defining what actions an authenticated identity can perform.

Controlling Access to EKS clusters:

- **Authentication Methods**: EKS supports various methods like **Bearer Tokens**, **X.509 certificates**, and **OIDC**. Currently, EKS natively supports **webhook token authentication**, **service account tokens**, and **OIDC authentication**.
- **Webhook Authentication**: Uses bearer tokens generated by the **AWS CLI** or **aws-iam-authenticator** when running kubectl commands. The token is passed to the **kube-apiserver**, which forwards it to the authentication webhook. The webhook validates the token via a **pre-signed URL** embedded in the token's body.

Cluster access manager:

- **Simplifies Identity mapping**: Manages access of AWS IAM principals to EKS clusters, reducing the need to edit the aws-auth ConfigMap.
- **Access entries**: Links an AWS IAM principal to an EKS cluster.
- **Access policies**: Defines the actions an Access Entry can perform in the EKS cluster.

Cluster access recommendations:

Private cluster endpoint: Configure the EKS cluster endpoint to be **private** to restrict access from the internet.

- **CIDR whitelisting**: If the endpoint is public, specify which CIDR blocks can access it.
- **Least Privileged Access**: Grant only necessary permissions to IAM users and roles.
- **Remove Cluster-Admin Permissions**: Avoid giving permanent cluster-admin permissions to the cluster creator.
- **Use IAM Roles for Multiple Users**: Assign roles instead of individual permissions for better management.
- **Regular Audits**: Regularly audit access to the cluster to ensure compliance and security.
- **Avoid Service Account Tokens**: These are long-lived and static. Use IAM roles for better security.

IAM Roles for Service Accounts (IRSA):

- **Assign IAM Roles to Service Accounts**: IRSA is a feature that allows you to assign an IAM role to a Kubernetes service account. It works by leveraging a Kubernetes feature known as **Service Account Token Volume Projection**. When Pods are configured with a Service Account that references an IAM Role, the Kubernetes API server will call the public **OIDC discovery endpoint** for the cluster on startup.

EKS Pod Identities:

- **Assign IAM Roles to Service Accounts**: Allows assigning IAM roles to Kubernetes service accounts without configuring an OIDC identity provider for each cluster.

Identities and Credentials for EKS Pods Recommendations

Update the aws-node Daemonset to Use IRSA

- The aws-node daemonset currently uses a role assigned to EC2 instances to manage IPs for pods. This role includes policies like **AmazonEKS_CNI_Policy** and **EC2ContainerRegistryReadOnly**, which can pose a security risk. Update the aws-node daemonset to use IAM Roles for Service Accounts (IRSA) to mitigate this risk.

Restrict Access to the Instance Profile Assigned to Worker Nodes

- Limit the permissions of the instance profile assigned to worker nodes to only what is necessary.

Scope IAM Role Trust Policy for IRSA Roles

- Scope the trust policy to the service account name, namespace, and cluster. This ensures that only specific service accounts can assume the role, enhancing security.

Use IMDSv2 for Instance Metadata Service

- When your application needs access to the Instance Metadata Service (**IMDS**), use **IMDSv2** and increase the hop limit on EC2 instances to 2. IMDSv2 requires a PUT request to get a session token, adding an extra layer of security.

Disable Auto-Mounting of Service Account tokens

- If your application doesn't need to call the Kubernetes API, set the **automountServiceAccountToken** attribute to **false** in the **PodSpec**. This prevents unnecessary exposure of service account tokens.

Use Dedicated Service Accounts for each application

- Assign a dedicated service account to each application. This applies to both Kubernetes API service accounts and IRSA/EKS Pod Identity service accounts.

Run Applications as non-Root users

- By default, containers run as root, which is not a best practice. Configure your pods to run as **non-root** users by setting the spec.securityContext.runAsUser attribute in the PodSpec.

Grant Least Privileged Access to Applications

- Ensure that applications have only the permissions they need to function. Regularly review and revoke unnecessary permissions.

2. Pod Security:

Overview: Pod security involves configuring your pod specifications to prevent processes running in containers from escaping their isolation boundaries and gaining access to the underlying host.

Linux Capabilities:

- Default Capabilities: Containers run with a set of default Linux capabilities that can allow privilege escalation. These include capabilities like CAP_NET_RAW and CAP_SYS_CHROOT.
- Privileged Pods: Avoid running pods as privileged, as they inherit all root capabilities from the host.

Node Authorization:

- Node Authorization Mode: Authorizes API requests from the kubelet, allowing nodes to perform specific read and write operations.
- Node Restriction Admission Controller: Limits nodes to modifying only their own attributes and pods bound to them, reducing the risk of lateral movement within the cluster.

Pod Security Solutions:

- Pod Security Admission (PSA): Replaces Pod Security Policies (**PSPs**) to enforce security standards.
- Policy-as-Code (**PAC**): Uses dynamic admission controllers to enforce policies written as code. Examples include **OPA/Gatekeeper**, **Kyverno**, **Kubewarden**, and **jsPolicy**.

Recommendations:

- **Use Multiple PSA Modes:** Combine **audit**, **enforce**, and **warn** modes for better user experience and security feedback.
- **Restrict Privileged Containers**: Limit the use of privileged containers to reduce security risks.
- **Avoid Running Processes as Root:** Configure containers to run as non-root users using the **runAsUser** and **runAsGroup** fields in the pod specification.
- **Never Run Docker in Docker**: Avoid running Docker inside containers or mounting the Docker socket, as it can compromise node security.

- **Restrict hostPath Usage:** Limit the use of **hostPath** volumes and configure them as read-only if necessary.
- **Set Resource Requests and Limits**: Define resource requests and limits for each container to prevent resource contention and potential DoS attacks.
- **Disable Privileged Escalation:** Prevent processes from gaining additional privileges by setting **allowPrivilegeEscalation** to **false** in the pod specification.

3. Multi-Tenancy:

Overview: Multi-tenancy involves isolating users or applications running on shared infrastructure. Kubernetes, being a single-tenant orchestrator, uses various constructs to create logical isolation among tenants.

Soft multi-tenancy:

- **Namespaces**: Divide the cluster into logical partitions. Each namespace can have its own **quotas**, **network policies,** and **service accounts.**
- **RBAC (Role-Based Access Control)**: Prevents tenants from accessing or manipulating each other's resources. Roles and role bindings enforce these controls.
- **Quotas and Limit Ranges**: Control the amount of cluster resources each tenant can consume.
- **Network Policies**: Prevent applications in different namespaces from communicating with each other.

Tools for Soft Multi-Tenancy:

- **Kiosk**: An open-source project providing capabilities like account separation, self-service namespace provisioning, and account limits.
- **Loft**: A commercial offering that adds multi-cluster access, sleep mode for inactive deployments, and single sign-on with OIDC providers.

Use Cases:

- **Enterprise Setting**: Tenants are semi-trusted (e.g., employees, contractors). Cluster administrators manage namespaces and policies, possibly delegating some administrative tasks.
- **Kubernetes as a Service (KaaS)**: Tenants interact directly with the Kubernetes API server and manage their own namespaces. Strict network policies and pod sandboxing are often required.
- **Software as a Service (SaaS)**: Each tenant is associated with a specific instance of an application running within the cluster.

Kubernetes Constructs for Multi-Tenancy:

- **Namespaces**: Fundamental for logical isolation.
- **Network Policies**: Control communication between pods.
- **RBAC**: Enforce access controls.
- **Quotas**: Define resource limits.
- **Pod Priority and Preemption**: Ensure higher-priority pods can run by evicting lower-priority ones if necessary.

Hard multi-tenancy:

- **Separate Clusters for Each Tenant**: Provides strong isolation but can be expensive and complex to manage. Each cluster incurs control plane costs and cannot share compute resources, leading to potential resource fragmentation. Managing numerous clusters requires specialized tooling and can become unwieldy. This approach is slower compared to creating namespaces but may be necessary in highly-regulated industries or SaaS environments requiring strong isolation.

Recommendations for Tenant Isolation:

- **Sandboxed Execution Environments**: Use technologies like Firecracker to run each container in its own isolated VM.
- **Isolating Tenant Workloads to Specific Nodes**: Use node affinity, taints, and tolerations to schedule tenant-specific workloads on dedicated nodes.

4. Detective Controls:

Auditing and Logging: Collecting and analyzing audit logs is crucial for root cause analysis, attribution, and detecting anomalous behaviors. On EKS, audit logs are sent to Amazon CloudWatch Logs.

Recommendations:

- **Enable Audit Logs**: Ensure that audit logs are enabled as part of the EKS managed Kubernetes control plane logs. This includes logs for the Kubernetes API server, controller manager, scheduler, and audit logs. Instructions can be found here.
- **Utilize Audit Metadata**: Kubernetes audit logs include annotations indicating whether a request was authorized (authorization.k8s.io/decision) and the reason for the decision (authorization.k8s.io/reason). Use these attributes to understand why specific API calls were allowed.
- **Create Alarms for Suspicious Events**: Set up alarms to alert you when there is an increase in 403 Forbidden and 401 Unauthorized responses. Use attributes like host, source IPs, and k8s_user.username to investigate the source of these requests.
- **Analyze Logs with Log Insights**: Use CloudWatch Log Insights to monitor changes to RBAC objects such as Roles, RoleBindings, ClusterRoles, and ClusterRoleBindings. This helps in tracking updates and potential security issues.
- **Audit CloudTrail Logs**: AWS APIs called by pods using IAM Roles for Service Accounts (IRSA) are logged to CloudTrail along with the service account name. If an unauthorized service account appears in the logs, it may indicate a misconfigured IAM role trust policy. CloudTrail is useful for attributing AWS API calls to specific IAM principals.
- **Use CloudTrail Insights**: CloudTrail Insights automatically analyzes write management events and alerts you to unusual activity. This helps identify increases in call volume on write APIs, including those from pods using IRSA to assume an IAM role.

Additional Resources:

- **Sysdig Falco and ekscloudwatch**: As log volume increases, consider using tools like Sysdig Falco and ekscloudwatch. Falco analyzes audit logs for anomalies or abuse, while ekscloudwatch forwards audit log events from CloudWatch to Falco for analysis.
- **SageMaker Random Cut Forest**: Store audit logs in S3 and use the SageMaker Random Cut Forest algorithm to detect anomalous behaviors that warrant further investigation.

5. Network Security:

Network Security Overview: Network security in EKS involves controlling traffic flow and encrypting data in transit. Key mechanisms include network policies, security groups, service meshes, and container network interfaces (CNIs).

Traffic Control:

Network Policies:

- **Purpose**: Restrict network traffic between pods (East/West traffic) and between pods and external services.
- **Implementation**: Use pod, namespace selectors, and labels to define rules. Policies can control both ingress and egress traffic.
- **Amazon VPC CNI Plugin**: Supports native network policy integration. Ensure the VPC CNI add-on is enabled and configured correctly.

Recommendations:

- **Principle of Least Privilege**: Start with a default deny policy and incrementally add rules.
- **Monitoring and Auditing**: Use tools like Network Policy Editor and audit logs to monitor and review policies.
- **Automated Testing**: Regularly test policies in a controlled environment.
- **Prometheus Metrics**: Monitor VPC CNI node agents for health and errors.

Security Groups:

- **Purpose**: Control traffic between the Kubernetes control plane, worker nodes, and external resources.
- **Security Groups for Pods**: Assign security groups to pods for fine-grained control. Ensure proper configuration to avoid scheduling issues.

Best Practices:

- **Layered Security**: Combine security groups and network policies for robust protection.
- **Principle of Least Privilege**: Only allow necessary traffic.
- **Segmentation**: Use network policies to segment applications and reduce the attack surface.

Service Mesh Policy Enforcement:

- **Purpose**: Add capabilities like observability, traffic management, and security at Layer 7.
- **Use Cases**: Advanced traffic control, load balancing, mTLS for secure communication.
- **Integration**: Can be used alongside network policies for comprehensive security.

Third-Party Network Policy Engines:

- **Examples**: Calico, Cilium.
- **Features**: Support for advanced policies, DNS hostname rules, Layer 7 rules, and integration with service meshes.

Migration to Amazon VPC CNI Network Policy Engine:

- **Consistency**: Deploy only one network policy engine to avoid conflicts.
- **Migration Tool**: Use the K8s Network Policy Migrator to convert existing policies to Kubernetes native network policies.
- **Testing**: Test migrated policies in a separate environment before applying them in production.

Network Encryption:

Encryption in Transit:

- **TLS**: Standard for encrypting traffic. Use TLS for secure communications.
- **Nitro Instances**: Automatically encrypt traffic between supported instance types.
- **CNIs**: WeaveNet can encrypt traffic using NaCl and IPsec.
- **Service Mesh**: Linkerd, and Istio support mTLS for encryption.

Ingress Controllers and Load Balancers:

- **Purpose**: Route HTTP/S traffic from outside the cluster to services inside.
- **Encryption**: Use AWS Elastic Load Balancers (ALB, NLB) for SSL/TLS termination. Configure certificates using annotations.

6. Data Encryption and Secrets Management:

Encryption at Rest:

- **AWS Storage Options**: Use EBS, EFS, and FSx for Lustre, all of which support encryption at rest with either a service-managed key or a customer master key (CMK).
- **EBS**: Use the in-tree storage driver or the EBS CSI driver to encrypt volumes.
- **EFS**: Use the EFS CSI driver and configure at-rest encryption before creating a Persistent Volume (PV).
- **FSx for Lustre**: Supports encryption at rest and in transit by default.

Recommendations:

- **Encrypt Data at Rest**: Always encrypt your data to ensure security.
- **Rotate CMKs Periodically**: Configure AWS KMS to automatically rotate your CMKs annually.
- **Use EFS Access Points**: Simplify access to shared datasets with different POSIX file permissions by using EFS access points.

Secrets Management:

- **AWS KMS for Envelope Encryption**: Encrypt Kubernetes secrets with a unique data encryption key (DEK) and a key encryption key (KEK) from AWS KMS.
- **Audit Kubernetes Secrets**: Enable audit logging and create CloudWatch metrics filters to monitor the use of secrets.
- **Rotate Secrets Periodically**: Use external secret stores like Vault or AWS Secrets Manager for automatic rotation.

- **Isolate Secrets with Namespaces**: Use separate namespaces to isolate secrets for different applications.
- **Use Volume Mounts**: Mount secrets as volumes instead of using environment variables to avoid accidental exposure in logs.
- **External Secrets Providers**: Consider using AWS Secrets Manager, Hashicorp Vault, or Bitnami's Sealed Secrets for enhanced security features.

External Secrets Integration:

- **Secret Store CSI Driver**: Fetch secrets from external stores like AWS Secrets Manager, Azure, Vault, and GCP. Supports secret rotation and synchronization with Kubernetes Secrets.
- **AWS Secrets & Configuration Provider (ASCP)**: Integrate AWS Secrets Manager secrets with Kubernetes using the Secret Store CSI Driver.
- **external-secrets**: Copies secrets from external stores to Kubernetes Secrets, allowing management in a Kubernetes-native way.

7. Runtime Security:

Overview: Runtime security involves actively protecting your containers while they are running by detecting and preventing malicious activity. This can be achieved using various Linux kernel mechanisms and tools integrated with Kubernetes.

Security Contexts and Built-in Kubernetes Controls:

- **Privileged Flag**: By default, this is false. Enabling it grants root-like access, which is generally inappropriate for production workloads.
- **Linux Capabilities**: Grant specific privileges to containers without giving full root access. Examples include CAP_NET_ADMIN for network configuration and CAP_SYS_TIME for system clock manipulation.
- **Seccomp**: Restricts the system calls a container can make, reducing the attack surface. Use default profiles or create custom ones with tools like Inspektor Gadget and Security Profiles Operator.
- **AppArmor and SELinux**: Mandatory access control systems that provide fine-grained security policies. AppArmor is supported on Debian/Ubuntu, while SELinux is supported on RHEL/CentOS/Bottlerocket/Amazon Linux 2023.

Recommendations:

- **Use Amazon GuardDuty**: For continuous monitoring and threat detection in EKS environments. GuardDuty analyzes data sources like CloudTrail, VPC flow logs, and Kubernetes audit logs to identify malicious activity.
- **Consider Third-Party Solutions**: If managing seccomp and AppArmor profiles is challenging, use third-party tools that leverage machine learning for runtime security.
- **Add/Drop Linux Capabilities**: Before writing seccomp policies, consider adjusting Linux capabilities to achieve the desired security controls.

8. Protecting the Infrastructure (Hosts):

Overview: Securing the infrastructure that runs your containers is as crucial as securing the containers themselves. This involves using optimized operating systems, keeping systems updated, and employing various security tools and practices.

Recommendations:

Use an OS Optimized for Running Containers:

- **Options**: Consider using Flatcar Linux, Project coreOS, RancherOS, or Bottlerocket (AWS's purpose-built OS for containers). These OS options have reduced attack surfaces and enforced permission boundaries.
- **EKS Optimized AMI**: Use the EKS optimized AMI for Kubernetes worker nodes, which is regularly updated with necessary OS packages and binaries.

Keep Your Worker Node OS Updated:

- Regularly update host OS images with the latest security patches.
- For EKS optimized AMIs, check the CHANGELOG and automate the rollout of updated images.

Treat Infrastructure as Immutable:

- Replace worker nodes instead of performing in-place upgrades. Use tools like eksctl or EKS managed node groups to automate this process.
- For EKS Fargate, AWS handles infrastructure updates, but ensure deployments have multiple replicas to handle rescheduling.

Run kube-bench for CIS Benchmark Compliance:

- Use kube-bench to evaluate your cluster against the CIS benchmarks for Kubernetes. Follow specific instructions for EKS to ensure compliance.

Minimize Access to Worker Nodes:

- Use SSM Session Manager instead of SSH for remote access. This provides better control and audit trails.
- Deploy workers onto private subnets to minimize exposure to the internet.

Run Amazon Inspector:

- Use Amazon Inspector to assess hosts for vulnerabilities and deviations from best practices. Ensure the SSM agent is installed on your instances.

Run SELinux:

- **Configuration**: Use SELinux to enforce mandatory access controls (MAC) for containers. Configure SELinux for Docker to automatically label workloads and isolate containers.
- **Custom Profiles**: Create custom SELinux profiles for different containers/pods as needed.
- **Booleans**: Adjust SELinux Booleans to modify default restrictions based on your needs.

Example SELinux Booleans:

- **container_connect_any**: Allow containers to access privileged ports on the host.
- **container_manage_cgroup**: Allow containers to manage cgroup configuration.
- **container_use_cephfs**: Allow containers to use a Ceph file system.

Relabeling in Kubernetes:

- Use custom MCS labels to run pods and ensure volumes are accessible. Set different MCS labels for strict isolation.

Sample AMIs:

- AWS provides sample AMIs for EKS with SELinux configured on CentOS 7 and RHEL 7, demonstrating implementations for highly regulated environments.

9. Compliance:

Overview: Compliance is a shared responsibility between AWS and its users. AWS manages the security of the cloud infrastructure, while users are responsible for securing their applications and data within the cloud.

AWS Responsibilities:

- **Physical Security**: Data centers and hardware.
- **Virtual Infrastructure**: Amazon EC2 and container runtime (e.g., Docker).

User Responsibilities:

- **Container Image Security**: Ensuring the security of container images.
- **Application Security**: Securing the application running within the containers.

Compliance Programs: AWS container services conform to various compliance programs, including PCI DSS, HIPAA, SOC, ISO, FedRAMP, and more. For the latest compliance status, refer to the AWS Compliance Services in Scope.

Shifting Left:

- **Concept**: Catch policy violations and errors earlier in the software development lifecycle.
- **Benefits**: Allows developers to fix issues before deployment, preventing non-compliant configurations from reaching production.

Policy as Code (PaC):

- **Definition**: Codifying policies to automate security, compliance, and privacy controls.
- **Benefits**: Reuse DevOps and GitOps strategies to manage and apply policies consistently across Kubernetes clusters.

Tools for Policy as Code:

- **OPA (Open Policy Agent)**: An open-source policy engine that can be integrated into CI/CD pipelines to provide early feedback on configurations.
- **Conftest**: Built on top of OPA, it offers a developer-focused experience for testing Kubernetes configurations.
- **Kyverno**: A policy engine designed for Kubernetes, allowing policies to be managed as Kubernetes resources without requiring a new language. Kyverno can validate, mutate, and generate Kubernetes resources and ensure OCI image supply chain security.

10. Incident Response and Forensics:

Overview: Quickly reacting to incidents can minimize damage from breaches. A reliable alerting system is crucial for detecting suspicious behavior. When an incident occurs, decide whether to destroy and replace the affected container or isolate and inspect it for forensic analysis.\

Sample Incident Response Plan:

Identify the Offending Pod and Worker Node:

- **Isolate the Damage**: Identify where the breach occurred and isolate the affected pod and node.
  Using Workload Name: Identify the worker node running the compromised pod.
- **Using Service Account Name**: Identify all pods using the compromised service account and their nodes.

- **Using Vulnerable or Compromised Images**: Identify pods using the compromised image and their nodes.

Isolate the Pod:

- **Network Policy**: Create a policy that denies all ingress and egress traffic to the pod to stop ongoing attacks.

Revoke Temporary Security Credentials:

- **IAM Roles**: Remove IAM roles from the compromised pod or node to prevent further access to AWS resources.

Cordon the Worker Node:

- **Scheduler**: Prevent the scheduler from assigning new pods to the affected node.

Enable Termination Protection:

- **Protection**: Prevent attackers from terminating the affected node to erase evidence.

Label the Offending Pod/Node:

- **Active Investigation**: Label the pod/node to indicate it is under investigation.

Capture Volatile Artifacts:

- **Memory and Processes**: Capture the operating system memory and perform a netstat tree dump.
- **Container State**: Use container runtime capabilities to capture information about running containers.

Redeploy Compromised Pod or Workload Resource:

Fix Vulnerabilities: Roll out fixes and start new replacement pods. Delete vulnerable pods and redeploy workload resources if necessary.

Recommendations:

Review the AWS Security Incident Response Whitepaper:

- **Comprehensive Guide**: Provides detailed recommendations for handling security breaches.

Practice Security Game Days:

- **Kubesploit**: Use this penetration testing framework to simulate real-world attacks and practice response strategies.

Run Penetration Tests:

- **Discover Vulnerabilities**: Periodically test your cluster to find vulnerabilities and misconfigurations. Follow AWS guidelines for conducting penetration tests.

11. Image Security:

Overview: The container image is your first line of defense against attacks. A secure, well-constructed image helps prevent attackers from escaping the container and gaining access to the host.

Recommendations:

Create Minimal Images:

- **Remove Extraneous Binaries**: Use tools like Dive to inspect and remove unnecessary binaries from the image.
- **De-fang the Image**: Remove SETUID and SETGID bits from binaries to prevent privilege escalation.
- **Multi-Stage Builds**: Use multi-stage builds to create minimal images by excluding build tools and other unnecessary binaries.

Create Software Bill of Materials (SBOMs):

- **Visibility**: Understand the components of your container image.
- **Provenance Verification**: Ensure the origin and integrity of the software artifacts.
- **Trustworthiness**: Verify that the image is safe and suitable for its intended purpose.
- **Tools**: Use Amazon Inspector or Syft from Anchore to generate SBOMs.

Scan Images for Vulnerabilities Regularly:

Image Scanning: Use Amazon ECR to scan images on push or on-demand. Address HIGH or CRITICAL vulnerabilities immediately.

- **Commercial Tools**: Consider tools like Grype, Prisma Cloud, Aqua, openclarity, Trivy, and Snyk for advanced capabilities.

Use Attestations to Validate Artifact Integrity:

- **Attestations**: Cryptographically signed statements that validate the integrity and trustworthiness of an artifact.
- **Tools**: Use AWS Signer or Sigstore Cosign to create and verify attestations.

Create IAM Policies for ECR Repositories:

- **Namespaces**: Group repositories by team using prefixes (e.g., team-a/).
- **Private Endpoints**: Use ECR private endpoints to access the API through a private IP address.
- **Endpoint Policies**: Limit API access to ECR repositories to prevent data exfiltration.

Implement Lifecycle Policies for ECR:

- **Remove Stale Images**: Use lifecycle policies to remove old images with outdated software packages.
- **CI/CD Practices**: Ensure deployments and images are up to date to avoid pull errors.

Create a Set of Curated Images:

- **Vetted Images**: Provide developers with a set of vetted images for different application stacks.
- **Base Images**: Create base images to avoid pulling from Dockerhub, which may contain vulnerabilities.

Add the USER Directive to Dockerfiles:

- **Non-Root User**: Use the USER directive to run containers as non-root users.

Lint Your Dockerfiles:

- **Verification**: Use tools like hadolint to ensure Dockerfiles adhere to best practices.

Build Images from Scratch:

- **Minimal Images**: Create images from scratch to reduce the attack surface.

Sign Your Images, SBOMs, Pipeline Runs, and Vulnerability Reports:

- **Image Signing**: Use AWS Signer or Sigstore Cosign to sign images and attestations, ensuring integrity and trustworthiness.

Image Integrity Verification Using Kubernetes Admission Controller:

Automated Verification: Use dynamic admission controllers to verify image signatures and attestations before deployment.

12. Multi-Account Strategy:

Overview: AWS recommends a multi-account strategy to isolate and manage business applications and data. This approach offers benefits such as increased service quotas, simpler IAM policies, and improved resource isolation.

Benefits of Multi-Account Strategy:

- **Increased Service Quotas**: Each AWS account has its own set of quotas, increasing the overall capacity available.
- **Simpler IAM Policies**: Easier to manage access by granting permissions within specific accounts.
- **Improved Isolation**: Limits the impact of application-related issues to a single account.

Approaches to Multi-Account Strategy:

Centralized EKS Cluster:

- **Cluster Account**: Deploy the EKS cluster in a single AWS account. Use IAM roles for Service Accounts (IRSA) or EKS Pod Identities for temporary AWS credentials.
- **Resource Sharing**: Use AWS Resource Access Manager (RAM) to share VPC subnets with workload accounts.
- **Tenant Isolation**: Align AWS accounts with Kubernetes namespaces for resource isolation.

De-centralized EKS Clusters:

- **Workload Accounts**: Deploy EKS clusters and other AWS resources in respective workload accounts. Each account is independent and managed by specific teams.
- **GitOps**: Manage deployments using Git repositories, allowing each team to deploy changes to their workload clusters.
- **No Cross-Account Access**: Use IRSA or EKS Pod Identities within each account to access AWS resources without cross-account access.

Centralized Networking:

- **Shared VPC Subnets**: Use AWS RAM to share VPC subnets with workload accounts, enabling centralized network management.

Choosing Between Centralized and De-centralized EKS Clusters:

- **Centralized**: Easier management, cost-efficient, but may face service quotas and shared resource risks.
- **De-centralized**: Stronger isolation, better scalability, but requires more resources and complex networking.

Cluster Autoscaling:

1. Scaling with Karpenter:

Overview: Karpenter is an open-source project that automates node lifecycle management for Kubernetes clusters. It efficiently scales and optimizes costs by provisioning and deprovisioning nodes based on pod scheduling needs.

Key Functions:

- **Monitor Unschedulable Pods**: Identifies pods that cannot be scheduled due to resource constraints.
- **Evaluate Scheduling Requirements**: Considers resource requests, node selectors, affinities, tolerations, etc.
- **Provision New Nodes**: Adds nodes that meet the requirements of unschedulable pods.
- **Remove Unneeded Nodes**: Deprovisions nodes when they are no longer required.

Reasons to Use Karpenter:

- **Simplified Node Management**: Eliminates the need for multiple node groups, providing flexibility and diversity with a single NodePool.
- **Improved Scheduling**: Quickly launches nodes and schedules pods, enhancing pod scheduling at scale.
- **Cluster-Aware**: Integrates closely with Kubernetes, offering more flexibility than traditional autoscaling methods.

Recommendations:

Karpenter Best Practices:

- **Use for Dynamic Workloads**: Ideal for clusters with high, spiky demand or diverse compute requirements.
- **Run Controller on EKS Fargate or Node Group**: Ensure the Karpenter controller runs on a stable node, not managed by Karpenter.
- **Exclude Unnecessary Instance Types**: Avoid provisioning instance types that do not fit your workload needs.
- **Enable Interruption Handling for Spot Instances**: Configure Karpenter to handle Spot Instance interruptions and other involuntary events.

Creating NodePools:

- **Multiple NodePools**: Use multiple NodePools for different teams or workloads with varying OS or instance type requirements.
  Mutually Exclusive or Weighted NodePools: Ensure NodePools are either mutually exclusive or weighted for consistent scheduling behavior.
  Scaling with Cluster Autoscaler
  Overview: The Kubernetes Cluster Autoscaler, maintained by SIG Autoscaling, ensures your cluster has enough nodes to schedule pods without wasting resources. It monitors unschedulable pods and underutilized nodes, simulating node additions or removals before applying changes.

Key Concepts:

- **Scalability**: Performance as the cluster grows in pods and nodes.
- **Performance**: Speed of scaling decisions and actions.
- **Availability**: Quick and disruption-free pod scheduling.
- **Cost**: Efficiency in scaling decisions to avoid resource wastage.
- **Node Groups**: Abstract groups of nodes with shared properties.
- **EC2 Auto Scaling Groups**: Implementation of Node Groups on EC2.
- **EC2 Managed Node Groups**: Simplified management with additional features.

2. Operating the Cluster Autoscaler:

- **Deployment**: Installed as a Deployment in your cluster, using leader election for high availability.
- **Version Matching**: Ensure the Cluster Autoscaler version matches the Kubernetes cluster version.
- **Auto Discovery**: Enable auto discovery for easier management.

IAM Role Configuration:

- **Least Privileged Access**: Limit actions to specific Auto Scaling groups to prevent cross-cluster modifications.

Configuring Node Groups:

- **Consistency**: Ensure nodes in a Node Group have identical scheduling properties.

MixedInstancePolicies: Use instance types with similar CPU, memory, and GPU configurations.

- **Node Group Size**: Prefer larger Node Groups over many smaller ones for better scalability.
- **Managed Node Groups**: Use EKS Managed Node Groups for automatic discovery and graceful termination.

Optimizing for Performance and Scalability:

- **Resource Allocation**: Adjust resources provided to the Cluster Autoscaler process.
- **Scan Interval**: Tune the scan interval for the autoscaling algorithm.
  Node Group Management: Balance the number of Node Groups to optimize performance.

AWS EKS Reliability:

Overview: The reliability best practices for EKS are grouped under the following topics:

- ***Applications***
- ***Control Plane***
- ***Data Plane***

What Makes a System Reliable? A reliable system can function consistently and meet demands despite changes in its environment over time. To achieve this, the system must detect failures, automatically heal itself, and scale based on demand.

EKS Reliability Overview:

- **Control Plane**: AWS is responsible for the reliability of the Kubernetes control plane, which is designed to be highly available and fault-tolerant. EKS runs the control plane across three availability zones in an AWS Region, managing the availability and scalability of the Kubernetes API servers and the etcd cluster.
- **Data Plane**: The reliability of the data plane is a shared responsibility between you and AWS. EKS offers three worker node options for deploying the Kubernetes data plane:
  - **Fargate**: The most managed option, handling provisioning and scaling.
  - **Managed Node Groups**: Automates provisioning and lifecycle management of EC2 nodes.
  - **Self-Managed Nodes**: The least managed option, where you are responsible for patching and upgrading the nodes.

Managed Node Groups:

- Automate the provisioning and lifecycle management of EC2 nodes.
- Use the EKS API to create, scale, and upgrade managed nodes.
- Managed nodes run EKS-optimized Amazon Linux 2 EC2 instances and can be customized by enabling SSH access.
- Managed nodes run as part of an EKS-managed Auto Scaling Group that can span multiple Availability Zones.
- EKS automatically tags managed nodes for use with Cluster Autoscaler.

Shared responsibility model:

- **Fargate**: AWS handles most of the management, reducing your responsibility.
- **Managed Node Groups**: AWS builds patched AMIs, but you must deploy these patches.
- **Self-Managed Nodes**: You are responsible for patching and upgrading the AMIs and nodes.

Application reliability:

Running Highly available Applications: Your customers expect your application to be always available, including during changes and traffic spikes. A scalable and resilient architecture keeps your applications running without disruptions. Kubernetes helps by ensuring that once set up, it continuously matches the current state with the desired state.

Recommendations:

1. **Avoid Singleton Pods:**
   - **Risks:** Single Pod applications become unavailable if the Pod is terminated.
   - **Solution:** Use Deployments to ensure redundancy.

2. **Run multiple replicas:**

    - Benefits: Ensures high availability; if one replica fails, others continue to function.
    - Tools: Use Horizontal Pod Autoscaler (HPA) for automatic scaling.

3. **Schedule Replicas Across Nodes:**

    - Importance: Prevents all replicas from being affected if a single node fails.
    - Methods: Use pod anti-affinity or topology spread constraints.

4. **Using Pod Anti-Affinity Rules:**

    - Strategy: Prefer placing pods on separate nodes and AZs.
    - Example: Use **requiredDuringSchedulingIgnoredDuringExecution** for **topologyKey: topology.kubernetes.io/zone**.

5. **Pod Topology Spread Constraints:**

    - **Purpose**: Ensures fault tolerance and availability across different topology domains.
    - **Key Properties**: maxSkew, topologyKey, whenUnsatisfiable, labelSelector.

6. **Run Kubernetes Metrics Server:**

    - Function: Helps scale applications by collecting resource metrics.
    - Note: Metrics server is not a monitoring solution; use tools like Prometheus or CloudWatch for tracking over time.

7. **Horizontal Pod Autoscaler (HPA):**

    - Function: Automatically scales applications based on demand.
    - Metrics: Uses Resource Metrics API and custom/external metrics.

8. **Vertical Pod Autoscaler (VPA):**

    - Function: Adjusts CPU and memory reservations for Pods.
    - Note: May cause temporary unavailability as it recreates Pods.

Updating Applications:

- **Mechanisms for rollback:** Use blue/green deployments to quickly retract changes if needed.
- **Continuous updates:** Kubernetes tools allow for rapid innovation without disrupting availability.

Monitoring Applications:

- **Automated monitoring:** Use tools like Prometheus or CloudWatch Container Insights.
- **Key metrics:** RED method (Requests, Errors, Duration) and USE method (Utilization, Saturation, Errors).
- **Exposing metrics:** Use Prometheus client library for custom metrics.

Centralized Logging:

- **Types of Logs:** Control plane logs (API server, Audit, etc.) and application logs.
- **Tools:** Fluent Bit, Fluentd, CloudWatch Container Insights for log aggregation and storage.

Distributed Tracing:

- **Importance:** Identifies bottlenecks and prevents cascading failures.
- **Methods:** Code-level tracing or using service meshes like **Cilium** and **Istio**.

Tools: AWS X-Ray, Jaeger for both shared library and service mesh implementations.

EKS Control Plane Reliability:

Amazon Elastic Kubernetes Service (EKS): EKS is a managed Kubernetes service that simplifies running Kubernetes on AWS. It ensures compatibility with upstream Kubernetes, automatically managing the availability and scalability of control plane nodes.

EKS Architecture: EKS architecture eliminates single points of failure, ensuring high availability and durability. The control plane runs inside an EKS managed VPC, with components like the API server and etcd cluster distributed across multiple Availability Zones (AZs).

Key Components:

- **Kubernetes API Server Nodes:** Run in an auto-scaling group across multiple AZs.
- **etcd Server Nodes:** Also run in an auto-scaling group across three AZs.
- **NAT Gateway:** Ensures private subnet communication.
- **Managed Endpoint:** Uses NLB to load balance API servers, with ENIs in different AZs for worker node communication.

    Connectivity: You can configure the API server to be reachable via the public internet or through your VPC, ensuring redundant paths for connection.

Recommendations:

1. **Monitor Control Plane Metrics:**

    - **Importance:** Identifies performance issues and maintains availability.
    - **Tools:** Use Prometheus or CloudWatch Container Insights.
    - **Metrics to Monitor:**
        - **API Server**

          Metrics: apiserver_request_total, apiserver_request_duration_seconds, etc.
        - **etcd**

          Metrics: etcd_request_duration_seconds, etcd_db_total_size_in_bytes.

2. **Cluster Authentication:**

    - **Types:** Bearer/service account tokens and IAM authentication.
    - **Management:** Use aws-auth ConfigMap to manage access.
    - **Fallback:** Create a super-admin service account for emergency access.

3. **Admission Webhooks:**

    - **Types:** Validating and mutating admission webhooks.
    - **Configuration:** Avoid “catch-all” webhooks to prevent destabilizing the control plane.

EKS Data Plane reliability:

High availability and resilience: To operate highly available and resilient applications, you need a data plane that can scale and heal automatically. A resilient data plane consists of multiple worker nodes, can grow and shrink with the workload, and recovers from failures.

Worker Node options:

- **EC2 instances:** Manage yourself or use EKS managed node groups.
- **Fargate:** Offers isolated compute environments for each Pod, automatically scaling the data plane as Kubernetes scales pods.

Scaling EC2 Worker nodes:

- **Preferred methods:** Kubernetes Cluster Autoscaler, EC2 Auto Scaling groups, or community projects like Atlassian’s Escalator.

Recommendations:

1. **Use EC2 Auto Scaling Groups:**
    - Automatically replace terminated or failed nodes, ensuring cluster capacity.
2. **Use Kubernetes Cluster Autoscaler:**
    - Adjusts data plane size based on pod requirements.
    - Reacts to Pending pods by adding EC2 instances.
    - Terminates underutilized nodes.

3. **Configure Over-Provisioning:**
   - Add more replicas than required to account for scaling delays.
   - Use pause Pods and Priority Preemption to reserve compute capacity.
4. **Multiple Auto Scaling Groups:**
   - Enable --node-group-auto-discovery to find autoscaling groups with a defined tag.
5. **Local Storage:**
   - Set --skip-nodes-with-local-storage to false to allow scaling down nodes with local storage.
6. **Spread Across Multiple AZs:**
   - Run worker nodes and pods in multiple AZs to protect against AZ failures.
   - Use Topology Spread Constraints for Pods in Kubernetes 1.18+.
7. **Ensure EBS Volume Capacity:**
   - Create Auto Scaling Groups for each AZ with enough capacity.
   - Enable --balance-similar-node-groups in Cluster Autoscaler.

Implement QoS:

- **Critical Applications:** Define requests=limits for containers to prevent them from being killed if another Pod requests resources.
- **CPU and Memory Limits:** Implement limits to prevent a container from consuming excessive resources.

Configure and Size Resource Requests/Limits:

- **CPU Limits:** Avoid specifying limits to allow full CPU usage.
- **Non-CPU Resources:** Set requests=limits for predictable behavior.
- **Tools:** Use Vertical Pod Autoscaler, Goldilocks and Parca.

Resource Quotas for Namespaces:

- **Purpose:** Limit aggregate resource consumption in a namespace.
- **ResourceQuota Object:** Limits the quantity of objects and total compute resources.
- **LimitRange Object:** Enforces minimum and maximum resource usage per Pod or Container.

CoreDNS:

- **Function:** Provides name resolution and service discovery.
- **Monitoring:** Use Prometheus to monitor latency, errors, and memory consumption.
- **NodeLocal DNSCache:** Improves DNS performance by running a DNS caching agent on cluster nodes.
- **Cluster-Proportional-Scaler:** Automatically scales CoreDNS based on the number of nodes and CPU cores.

Choosing an AMI for Node Groups:

- **EKS Optimized AMIs:** Use optimized AMIs published by EKS for each Kubernetes version.
- **Deprecated AMIs:** Avoid using deprecated AMIs to prevent security vulnerabilities.

AWS EKS Reliability
Data Plane
Applications
Control Plane

Best Practices for Networking:

Understanding Kubernetes Networking: Efficient operation of your cluster and applications requires a solid understanding of Kubernetes networking. Pod networking, or cluster networking, is central to this. Kubernetes supports Container Network Interface (CNI) plugins for cluster networking.

Amazon VPC CNI Plugin: Amazon EKS officially supports the Amazon Virtual Private Cloud (VPC) CNI plugin for Kubernetes Pod networking. The VPC CNI provides native integration with AWS VPC, operating in underlay mode where Pods and hosts share the same network namespace.

Kubernetes Networking Model: Kubernetes sets specific requirements for cluster networking:

- Pods on the same node must communicate without NAT.
- System daemons on a node must communicate with Pods on the same node.
- Pods using the host network must contact all other Pods on all nodes without NAT.

Container Networking Interface (CNI): Kubernetes supports CNI specifications and plugins to implement its network model. The CNI plugin is enabled by passing the --network-plugin=cni option to kubelet, which reads the CNI configuration from a specified directory.

Amazon VPC CNI: The AWS-provided VPC CNI is the default networking add-on for EKS clusters, installed by default when provisioning EKS clusters. It consists of the CNI binary and the IP Address Management (ipamd) plugin, which manages AWS Elastic Networking Interfaces (ENIs) and maintains a warm pool of IPs for fast Pod startup times.

1- VPC and Subnet Considerations:

Operating an EKS cluster requires knowledge of AWS VPC networking, in addition to Kubernetes networking. Understanding the EKS control plane communication mechanisms is crucial before designing your VPC or deploying clusters into existing VPCs.

EKS Cluster Architecture: An EKS cluster consists of two VPCs:

- **AWS-managed VPC:** Hosts the Kubernetes control plane and does not appear in the customer account.
- **Customer-managed VPC:** Hosts the Kubernetes nodes, containers, and other AWS infrastructure like load balancers. This VPC appears in the customer account and must be created before the cluster.
  Nodes in the customer VPC need to connect to the managed API server endpoint in the AWS VPC to register with the Kubernetes control plane and receive requests to run application Pods. Nodes connect through either an EKS public endpoint or Cross-Account elastic network interfaces (X-ENI) managed by EKS.

EKS Control Plane Communication: EKS offers two ways to control access to the cluster endpoint:

- **Public Endpoint:** Default behavior, where API requests from within the VPC leave the VPC but stay within Amazon’s network.
- **Public and Private Endpoint:** Allows API requests from within the VPC to communicate via X-ENIs, making the API server accessible from the internet.
- **Private Endpoint:** No public access; all traffic to the API server must come from within the VPC or a connected network.

  VPC Configurations: Amazon VPC supports both IPv4 and IPv6 addressing. EKS clusters support IPv4 by default, with IPv6 clusters requiring dual-stack VPCs and subnets. It is recommended to use at least two subnets in different Availability Zones during cluster creation.

Subnet Types:

- **Public Subnets:** Have a route table entry to the internet through an internet gateway, allowing ingress and egress traffic.
- **Private Subnets:** Route traffic to a NAT gateway, allowing egress but blocking ingress traffic from outside the VPC.

In the IPv6 world, all addresses are internet routable, with private subnets supported by egress-only internet gateways (EIGW).

Configuring VPC and Subnets:

1. **Using Only Public Subnets:**
    - Nodes and ingress resources (like load balancers) are created in the same public subnets.
    - Tag the public subnet with kubernetes.io/role/elb to construct internet-facing load balancers.
    - The cluster endpoint can be public, private, or both.

2. **Using Private and Public Subnets:**
    - Nodes are created in private subnets, while ingress resources are in public subnets.
    - Enable public, private, or both access to the cluster endpoint.
    - Node traffic will enter via the NAT gateway or the ENI.

3. **Using Only Private Subnets:**
    - Both nodes and ingress resources are created in private subnets.
    - Use the kubernetes.io/role/internal-elb subnet tag to construct internal load balancers.
    - Accessing the cluster's endpoint requires a VPN connection.
    - Activate AWS PrivateLink for EC2 and all Amazon ECR and S3 repositories.
    - Only the private endpoint of the cluster should be enabled.

Communication Across VPCs:

- **Amazon VPC Lattice:** Connect services across multiple VPCs and accounts without additional connectivity services like **VPC peering**, **AWS PrivateLink**, or **AWS Transit Gateway**.
- **Private NAT Gateway with Custom Networking:** Use for integrating workloads on EKS to solve overlapping CIDR challenges while preserving routable RFC1918 IP addresses.
- **AWS PrivateLink:** Share Kubernetes services and ingress with customer VPCs in separate accounts.

Sharing VPC Across Multiple Accounts:

- **AWS Resource Access Manager (RAM):** Securely share supported AWS resources with individual AWS accounts, organizational units (OUs), or entire AWS Organization.
- **Deploying EKS in Shared Subnets:** Allows central networking teams to control networking constructs while application teams deploy EKS clusters in their respective accounts.

Considerations When Using Shared Subnets:

- **Security Groups:** Control traffic between the Kubernetes control plane and worker nodes, and between worker nodes and other VPC resources.
- **IAM Roles and Policies:** Create within the participant account to grant necessary permissions to Kubernetes clusters managed by EKS.
- **Cross-Account Access:** Use resource-based policies or OIDC provider approach for accessing AWS resources from Kubernetes pods.

Security Groups:

- Control traffic to and from resources associated with them.
- EKS creates a default security group for managed ENIs and nodes.
- Specify your own security groups when creating a cluster for customized traffic control.

Recommendations:

1. **Consider Multi-AZ Deployment:**
    - Deploy EKS clusters to multiple availability zones for automatic failover and high availability.
    - Use node labels in conjunction with Pod topology spread constraints for optimal resource distribution.

2. Amazon VPC CNI

Overview: Amazon EKS uses the Amazon VPC Container Network Interface (VPC CNI) plugin to provide Kubernetes Pods with the same IP address as they have on the VPC network. This ensures seamless communication within the cluster.

Components:

1. **CNI Binary:** Sets up Pod network for communication.
2. **ipamd:** Manages ENIs and maintains a warm pool of IP addresses.

ENI Management:

- **Primary ENI:** Attached to the node at creation.
- **Secondary ENIs:** Added as needed to provide additional IP addresses.
- **Warm Pool:** Pre-allocated IPs for faster Pod startup.

Pod Density:

- **Formula:**

((Number of network interfaces for instance type (number of prefixes per network interface-1)* 16) + 2

- **Example:** For 3 c5.large nodes (3 ENIs, 10 IPs per ENI):
    - Node 1: 2 ENIs, 20 IPs
    - Node 2: 2 ENIs, 20 IPs
    - Node 3: 1 ENI, 10 Ips

IP Address Cool Down:

- 30-second cool down cache to prevent premature recycling of IP addresses.

Recommendations:

1. **Deploy VPC CNI Managed Add-On:**
    - Includes latest security patches and bug fixes.
    - Prevents configuration drift by overwriting settings every 15 minutes.

2. **Migrate to Managed Add-On:**
    - Backup current settings before migration.
    - Use Amazon EKS API, AWS Management Console, or AWS CLI for configuration.

3. **Backup CNI Settings Before Update:**
    - Update add-ons one minor version at a time.
    - Use Helm for managing self-managed add-ons.

4. **Understand Security Context:**
    - CNI binary has privileged access to the node.
    - aws-node Daemonset manages IP addresses and runs in hostNetwork mode.

5. **Use Separate IAM Role for CNI:**
    - Create a new service account with Amazon EKS CNI policy.
    - Block access to instance metadata to minimize security risks.

6. **Handle Liveness/Readiness Probe Failures:**
    - Increase probe timeout values for EKS 1.20+ clusters.
    - Ensure CPU resource requests for aws-node are correctly configured.

Optimizing IP address utilization

Containerized environments are rapidly growing, leading to more worker nodes and pods being deployed. The Amazon VPC CNI plugin assigns each pod an IP address from the VPC's CIDR(s), which can consume a substantial number of IP addresses.

Optimize node-level IP consumption:

- **Prefix Delegation:** Assign IPv4 or IPv6 prefixes to EC2 instances to increase IP addresses per ENI, improving pod density per node.

Mitigate IP Exhaustion:

- **Size VPCs and subnets for growth:** Plan with future scalability in mind to prevent IP exhaustion.
- **Adopt IPv6:** Provides a larger IP address space, allowing for easier scaling.
- **Custom Networking:** Use non-routable Secondary CIDRs to allocate additional IP space for pods.

Recommendations:

1. **Use IPv6 (Recommended):**
   - IPv6 offers a significantly larger IP address space.
   - EKS clusters support both IPv4 and IPv6, with IPv6 providing better scalability.
2. **Optimize IP Consumption in IPv4 Clusters:**
   - **Plan for Growth:** Size your VPCs and subnets to accommodate future needs.
   - **Expand IP Space:** Use Custom Networking with CG-NAT space (e.g., 100.64.0.0/10) to conserve routable IPs.
   - : Set appropriate values to prevent excessive EC2 API calls.
3. **Optimize the IPs Warm Pool:**
   - **Environment Variables:** Adjust WARM_IP_TARGET, MINIMUM_IP_TARGET, and WARM_ENI_TARGET to match expected pod counts.

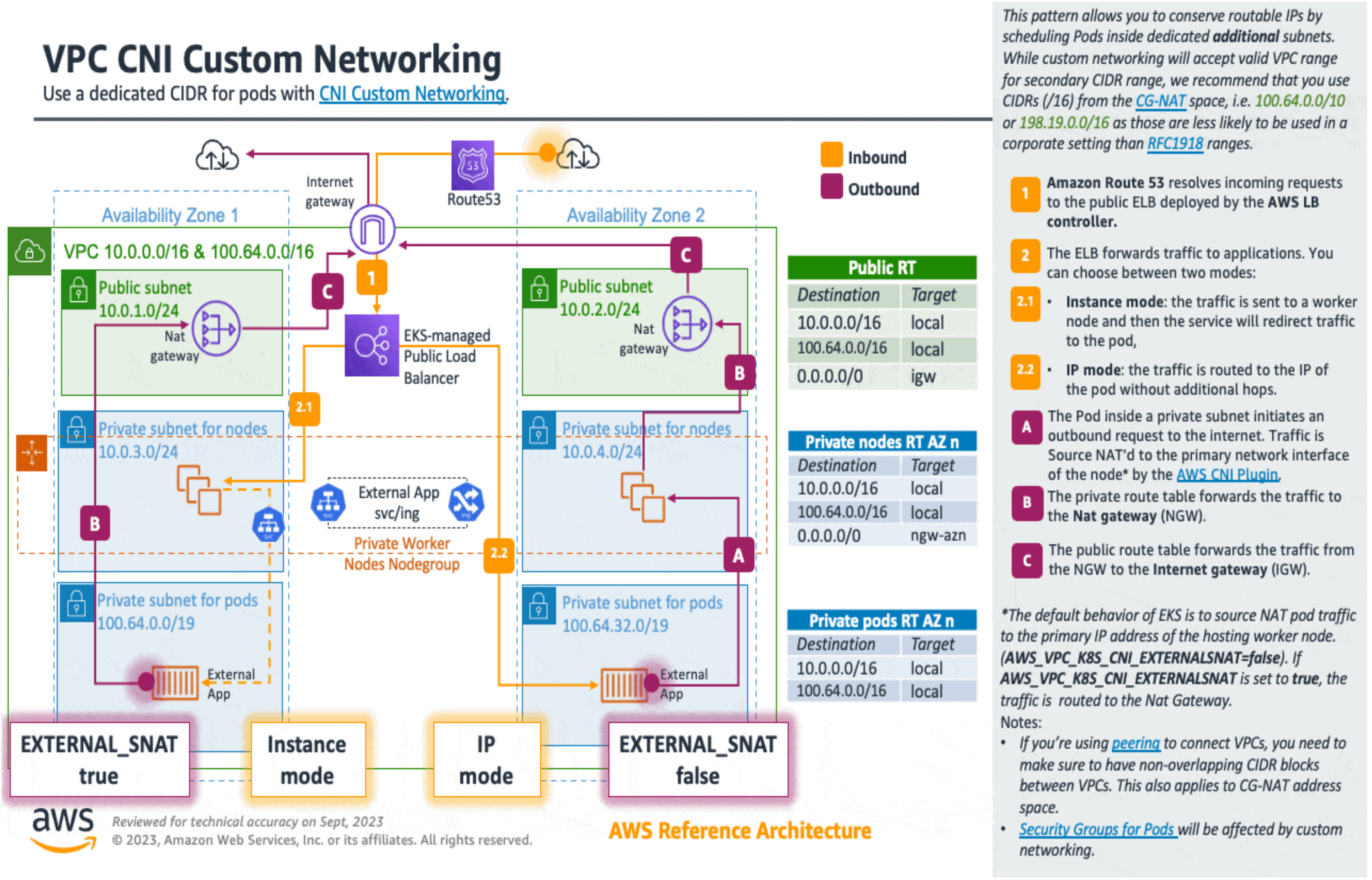
Avoid throttling

Example Configuration:

- **WARM_IP_TARGET:** Number of warm IP addresses to maintain.
- **MINIMUM_IP_TARGET:** Minimum number of IP addresses to allocate.
- **WARM_ENI_TARGET:** Number of warm ENIs to maintain.

Backup and Update:

- Always backup CNI settings before updates.
- Review and reapply configuration settings after updates.



VPC CNI Custom Networking

Use a dedicated CIDR for pods with CNI Custom Networking.

Internet gateway
Route53
🟧 Inbound
🟪 Outbound
Availability Zone 1
Availability Zone 2
VPC 10.0.0.0/16 & 100.64.0.0/16
Public subnet 10.0.1.0/24
Nat gateway
Public subnet 10.0.2.0/24
Nat gateway
1
C
C
EKS-managed Public Load Balancer
2.1
B
Private subnet for nodes 10.0.3.0/24
Private subnet for nodes 10.0.4.0/24
External App svc/ing
Private Worker Nodes Nodegroup
B
2.2
A
Private subnet for pods 100.64.0.0/19
External App
Private subnet for pods 100.64.32.0/19
External App
EXTERNAL_SNAT true
Instance mode
IP mode
EXTERNAL_SNAT false

Reviewed for technical accuracy on Sept, 2023

AWS Reference Architecture

Public RT

Destination	*Target*
10.0.0.0/16	local
100.64.0.0/16	local
0.0.0.0/0	igw

Private nodes RT AZ n

Destination	*Target*
10.0.0.0/16	local
100.64.0.0/16	local
0.0.0.0/0	ngw-azn

Private pods RT AZ n

Destination	*Target*
10.0.0.0/16	local
100.64.0.0/16	local

*This pattern allows you to conserve routable IPs by scheduling Pods inside dedicated **additional** subnets. While custom networking will accept valid VPC range for secondary CIDR range, we recommend that you use CIDRs (/16) from the CG-NAT space, i.e. 100.64.0.0/10 or 198.19.0.0/16 as those are less likely to be used in a corporate setting than RFC1918 ranges.*

1 **Amazon Route 53** resolves incoming requests to the public ELB deployed by the **AWS LB controller.**

2 The ELB forwards traffic to applications. You can choose between two modes:

2.1
- **Instance mode**: the traffic is sent to a worker node and then the service will redirect traffic to the pod,

2.2
- **IP mode**: the traffic is routed to the IP of the pod without additional hops.

A The Pod inside a private subnet initiates an outbound request to the internet. Traffic is Source NAT'd to the primary network interface of the node* by the AWS CNI Plugin.

B The private route table forwards the traffic to the **Nat gateway** (NGW).

C The public route table forwards the traffic from the NGW to the **Internet gateway** (IGW).

The default behavior of EKS is to source NAT pod traffic to the primary IP address of the hosting worker node. (AWS_VPC_K8S_CNI_EXTERNALSNAT=false). If AWS_VPC_K8S_CNI_EXTERNALSNAT is set to **true, the traffic is routed to the Nat Gateway.*

Notes:

- *If you're using peering to connect VPCs, you need to make sure to have non-overlapping CIDR blocks between VPCs. This also applies to CG-NAT address space.*
- *Security Groups for Pods will be affected by custom networking.*

Source

Running IPv6 EKS Clusters

Overview: EKS in IPv6 mode addresses the IPv4 exhaustion challenge in large-scale clusters. It provides flexibility to interconnect network boundaries using IPv6 CIDRs, minimizing CIDR overlap issues. Once enabled, IPv6 support is permanent for the cluster's lifetime.

Key features:

- **IPv6 Addresses:** Pods and Services receive IPv6 addresses while maintaining compatibility with IPv4 endpoints.
- **Dual-Stack VPC:** Each VPC gets an IPv4 CIDR block and a /56 IPv6 address prefix. Subnets receive a /64 IPv6 prefix.
- **Internet Routable:** All IPv6 addresses are internet routable by default. Use egress-only internet gateways (EIGW) for private subnets.

IPv6 Cluster setup:

- **Prefix Mode:** Supported only on Nitro-based EC2 instances. Each worker node gets an IPv6 prefix of /80, providing a vast number of IP addresses.
- **Host-Local CNI Plugin:** Allocates non-routable IPv4 addresses for Pods, enabling communication with IPv4 endpoints.

Communication:

- **Pod-to-Pod:** Uses IPv6 addresses.
- **Pod-to-IPv4 Endpoint:** Uses source network address translation (SNAT) to translate IPv4 addresses.
- **Services:** Receive IPv6 addresses from Unique Local IPv6 Unicast Addresses (ULA).

Load balancers:

Dual-Stack: AWS load balancers receive both IPv4 and IPv6 addresses, translating between them as needed.

Recommendations:

1. **Maintain Access to IPv4 EKS APIs:**
   - EKS APIs and cluster endpoints are accessible only via IPv4. Ensure your network supports IPv4 for API access.
2. **Run nodes and pods in private subnets:**
   - Use public load balancers in public subnets to balance traffic to Pods in private subnets.
3. **Use the Latest AWS Load Balancer Controller:**

Ensure compatibility and optimal performance with IPv6 clusters.

Custom networking:

Overview: By default, Amazon VPC CNI assigns Pods an IP address from the primary subnet. If the subnet CIDR is too small, the CNI may not acquire enough secondary IP addresses for your Pods, leading to IP exhaustion. Custom networking addresses this issue by assigning node and Pod IPs from secondary VPC address spaces (CIDR).

How it works:

- **ENIConfig Custom Resource:** Defines alternate subnet CIDR ranges and security groups for Pods.
- **Secondary ENIs:** Created in the subnet defined under ENIConfig, assigning IP addresses to Pods from this range.

Example configuration:

- Use CIDRs from the CG-NAT space (e.g., 100.64.0.0/10) for secondary CIDR ranges.
- Set the environment variable to enable custom networking.
- Define the ENIConfig to specify the subnet and security groups for Pods.

Recommendations:

1. **Use Custom Networking When:**
   - Dealing with IPv4 exhaustion and unable to use IPv6.
   - Security requirements necessitate running Pods on different networks with different security groups.
   - Connecting multiple EKS clusters and on-premises datacenter services.
2. **Avoid Custom Networking When:**
   - Ready to implement IPv6, as it provides a more scalable solution.
   - Exhausted CG-NAT space or unable to link a secondary CIDR with your cluster VPC.
3. **Use Private NAT Gateway:**
   - Enables instances in private subnets to connect to other VPCs and on-premises networks with overlapping CIDRs.
4. **Unique Network for Nodes and Pods:**
   - Deploy nodes and Pods to a subnet from a larger secondary CIDR block for security isolation.

Prefix Mode for Linux

Overview: Amazon VPC CNI assigns network prefixes to EC2 network interfaces to increase the number of IP addresses available to nodes, enhancing pod density per node. This mode is enabled by default on IPv6 clusters and can be configured for IPv4 clusters using version 1.9.0 or later of the Amazon VPC CNI add-on.

Key Features:

- **IPv6 Prefixes:** Assigns a /80 IPv6 prefix to an ENI slot.
- **IPv4 Prefixes:** Assigns /28 (16 IP addresses) IPv4 address prefixes to ENI slots, instead of individual IP addresses.

Benefits:

- **Increased Pod Density:** More IP addresses per ENI, leading to higher pod density per node.
- **Faster Pod Startup:** Pre-allocates prefixes for quicker IP assignment.
- prefix mode functionality.

Configuration Variables:

- **WARM_PREFIX_TARGET:** Number of prefixes to allocate more than current need.
- **WARM_IP_TARGET:** Number of IP addresses to allocate more than current need.
- **MINIMUM_IP_TARGET:** Minimum number of IP addresses to be available at any time.

Recommendations:

1. **Use Prefix Mode When:**
   - Experiencing pod density issues on worker nodes.
   - Using CNI custom networking where the primary ENI is not used for pods.
2. **Avoid Prefix Mode When:**
   - Subnet is highly fragmented with insufficient contiguous IP addresses for /28 prefixes.
   - Security requirements necessitate different security groups for pods.
3. **Use Similar Instance Types in Node Groups:**
   - Ensure consistent maximum pod count by using similar instance types within a node group.
4. **Configure WARM_PREFIX_TARGET:**
   - Set to 1 for a balance of fast pod launch times and minimal unused IP addresses.
   - Adjust WARM_IP_TARGET and MINIMUM_IP_TARGET to conserve IPv4 addresses.
5. **Prefer Prefix Allocation Over New ENI:**
   - Allocating additional prefixes to existing ENIs is faster than attaching new ENIs.
6. **Use Subnet Reservations:**
   - Reserve contiguous IP space within a subnet to avoid fragmentation and ensure successful prefix attachment.

Prefix Mode for Windows:

Overview: In EKS, each Pod on a Windows host is assigned a secondary IP address by default. This can limit the number of Pods you can run on a Windows host. Enabling Prefix Delegation increases pod density by assigning **/28 IPv4** prefixes to **ENI** slots instead of individual secondary IP addresses.

Key Features:

- **Increased pod density:** Assigns /28 IPv4 prefixes to ENI slots, allowing more IP addresses per node.
- **Warm pool:** Maintains a pool of IP addresses for faster pod startup.

Configuration Parameters:

- **warm-prefix-target:** Number of prefixes to allocate more than current need.
- **warm-ip-target:** Number of IP addresses to allocate more than current need.
- **minimum-ip-target:** Minimum number of IP addresses to be available at any time.

Recommendations:

1. **Use prefix delegation when:**
   - Experiencing pod density issues on Windows nodes.
   - Subnets have contiguous blocks of addresses for /28 prefixes.
2. **Avoid prefix delegation when:**
   - Subnets are highly fragmented with insufficient contiguous IP addresses.

   Configure parameters to conserve IPv4 addresses:
   - Adjust **warm-prefix-target**, **warm-ip-target**, and **minimum-ip-target** to balance IP conservation and pod startup latency.
3. **Use subnet reservations:**
   - Reserve contiguous IP space within a subnet to avoid fragmentation and ensure successful prefix attachment.
4. **Replace all nodes when migrating:**
   - Create new node groups rather than rolling replacements to increase available IP addresses.
5. **Run all pods in the same mode:**
   - Avoid running Pods in both secondary IP mode and prefix delegation mode simultaneously to prevent inconsistencies.

Security Groups Per Pod:

Overview: AWS security groups act as virtual firewalls for EC2 instances, controlling inbound and outbound traffic. By default, Amazon VPC CNI uses security groups associated with the primary ENI on the node, meaning all Pods on a node share the same security groups. This can be too coarse-grained for applications with varying security requirements.

Benefits:

- **Network segmentation:** Provides finer control over network access for individual Pods.
- **Improved security:** Allows running applications with different security needs on shared compute resources.

How It Works:

- **Trunk Interface:** A trunk interface (aws-k8s-trunk-eni) is attached to the node.
- **Branch Interfaces:** Branch interfaces (aws-k8s-branch-eni) are created and associated with the trunk interface.
- **SecurityGroupPolicy:** Custom resource used to assign security groups to Pods.

Recommendations:

1. **Disable TCP Early Demux for Liveness Probe:**
   - Ensure kubelet can connect to Pods on branch network interfaces via TCP.
2. **Leverage Existing AWS Configuration:**
   - Reuse existing AWS security group resources to restrict network access to VPC resources.
3. **Configure Pod Security Group Enforcing Mode:**
   - **Strict Mode:** Only branch ENI security groups are enforced, and source NAT is disabled.
   - **Standard Mode:** Both primary and branch ENI security groups are applied.
4. **Use Strict Mode for Isolating Pod and Node Traffic:**
   - Completely separate Pod traffic from node traffic but be aware it increases VPC traffic.

5. **Use Standard Mode for Specific Situations:**

   - **Client Source IP Visibility:** Preserve client source IP for Kubernetes services.
   - **NodeLocal DNSCache:** Support Pods using NodeLocal DNSCache for improved DNS performance.
   - **Kubernetes Network Policy:** Combine with network policies for East/West traffic control.

6. **Identify Incompatibilities:**

   - Note that Windows-based and non-nitro instances do not support security groups for Pods.

Load Balancing:

Overview: Load Balancers distribute incoming traffic across targets in an EKS Cluster, improving application resilience. The AWS Load Balancer Controller manages AWS Elastic Load Balancers for the cluster. It creates Network Load Balancers (NLB) for Kubernetes Services of type LoadBalancer and Application Load Balancers (ALB) for Kubernetes Ingress objects.

Choosing Load Balancer Type:

1. **Application Load Balancer (ALB):**
   - Use for HTTP/HTTPS workloads.
   - Operates at Layer 7 of the OSI model.
   - Configured by the Ingress resource, routing traffic to different Pods.
2. **Network Load Balancer (NLB):**
   - Use for TCP/UDP workloads or when source IP preservation is required.
   - Operates at Layer 4 of the OSI model.
   - Suitable for clients that cannot utilize DNS due to static IPs.

Provisioning Load Balancers:

1. **AWS Load Balancer Controller (Recommended):**
   - Manages AWS load balancers for EKS clusters.
   - Requires installation in the EKS cluster.
   - Uses annotations to configure load balancers.
2. **AWS Cloud Provider Load Balancer Controller (Legacy):**
   - Creates AWS Classic Load Balancers by default.

Choosing Load Balancer Target-Type:

1. **Instance Target-Type:**

   - Registers Worker Node's IP and **NodePort**.
   - Involves multiple hops, potentially increasing latency.
   - Health checks are received by the Worker Node.
2. **IP Target-Type (Recommended):**

   - Directly forwards traffic to the Pod.
   - Simplifies network path, reducing latency.
   - Health checks are directly received by the Pod.

Availability and Pod Lifecycle:

- Ensure application availability during upgrades to avoid downtime.
- Use IP target-type for better monitoring and troubleshooting.

Monitoring EKS Workloads for Network Performance Issues:

Monitoring CoreDNS Traffic for DNS Throttling Issues:

- **Problem:** DNS intensive workloads may experience intermittent CoreDNS failures due to DNS throttling, leading to **UnknownHostException** errors.
- **Solution:** Increase CoreDNS replicas or implement NodeLocal DNSCache. Monitor the **linklocal_allowance_exceeded** metric to identify DNS throttling.

Monitoring DNS Query Delays Using Conntrack Metrics:

- **Metrics to Monitor:**
  - **conntrack_allowance_available:** Number of tracked connections that can be established before hitting the limit.
  - **conntrack_allowance_exceeded:** Number of packets dropped due to exceeded connection tracking limits.

Other Important Network Performance Metrics:

- **bw_in_allowance_exceeded:** Packets queued/dropped due to exceeded inbound bandwidth.
- **bw_out_allowance_exceeded:** Packets queued/dropped due to exceeded outbound bandwidth.
- **pps_allowance_exceeded:** Packets queued/dropped due to exceeded packets-per-second limit.

Capturing Metrics:

- **Elastic Network Adapter (ENA) Driver:** Publishes network performance metrics.
- **CloudWatch Agent:** Publishes metrics to CloudWatch for monitoring.

Recommendations:

1. **Monitor CoreDNS Metrics:**
   - Capture linklocal_allowance_exceeded to identify DNS throttling.
   - Use Amazon Managed Service for Prometheus and Amazon Managed Grafana for visualization.
2. **Monitor Conntrack Metrics:**
   - Track conntrack_allowance_available and conntrack_allowance_exceeded to monitor connection limits.
3. **Publish Metrics to CloudWatch:**
   - Use the CloudWatch agent to publish network performance metrics from ENA driver.
4. **Set Up Alerts:**
   - Use Amazon Managed Service for Prometheus to configure alerting rules and send notifications via Amazon SNS.

EKS Scalability Best Practices:

Overview: Scaling an EKS cluster aims to maximize the workload a single cluster can handle. Using a large EKS cluster can reduce operational load compared to multiple clusters, but it has trade-offs for multi-region deployments, tenant isolation, and cluster upgrades.

Understanding Scaling Dimensions: Scalability differs from performance and reliability, and all three should be considered when planning your cluster and workload needs. EKS can scale to large sizes, but planning is required for clusters beyond 300 nodes or 5000 pods.

Key Areas for Scaling:

1. **Kubernetes Control Plane:**
   - Includes services AWS runs and scales automatically (e.g., Kubernetes API server).
   - AWS is responsible for scaling the Control Plane, but users must use it responsibly.
2. **Kubernetes Data Plane:**
   - Involves AWS resources required for your cluster and workloads, such as EC2 instances, kubelet, and storage.
   - These resources need to scale as your cluster scales.
3. **Cluster Services:**
   - Kubernetes controllers and applications running inside the cluster providing functionality for your workloads.
   - Includes EKS Add-ons and other services or Helm charts for compliance and integrations.
   - These services must scale with your workloads.
4. **Workloads:**
   - The primary reason for having a cluster, which should scale horizontally with the cluster.
   - Integrations and settings in Kubernetes can help the cluster scale.
   - Architectural considerations include namespaces and services.

Kubernetes Control Plane Scalability:

Overview: The Kubernetes control plane includes the API Server, Controller Manager, Scheduler, and other essential components. Key factors impacting scalability are the Kubernetes version, utilization, and node scaling.

Recommendations:

1. **Use EKS 1.24 or Above:**
    - Switches to containerd for better node performance.
2. **Limit Workload and Node Bursting:**
    - Avoid scaling spikes that increase cluster size by double-digit percentages.
3. **Scale Nodes and Pods Down Safely:**
    - Regularly replace long-running instances.
    - Remove underutilized nodes.
    - Use Pod Disruption Budgets to manage node shutdowns.
4. **Use Client-Side Cache with Kubectl:**
    - Reduces API calls and prevents throttling.
5. **Disable Kubectl Compression:**
    - Reduces CPU usage on the client and server.
6. **Shard Cluster Autoscaler:**
    - Run multiple instances for clusters with more than 1000 nodes.
7. **API Priority and Fairness:**
    - Manages inflight requests to prevent overloading the API Server.

Kubernetes Data Plane Scalability:

Overview: Selecting EC2 instance types for clusters with multiple workloads can be challenging. Here are tips to avoid common pitfalls and scale compute effectively.

Recommendations:

1. **Automatic Node Autoscaling:**
    - Use managed node groups or Karpenter for large-scale clusters.
    - Managed node groups offer flexibility with Auto Scaling groups and managed upgrades.
    - Karpenter scales nodes based on workload requirements, avoiding node group quotas.
2. **Use Many Different EC2 Instance Types:**
    - Avoid limiting instance types to prevent availability issues.
    - Karpenter uses a broad set of compatible instance types by default.
3. **Prefer Larger Nodes to Reduce API Server Load:**
    - Fewer, larger nodes reduce the load on the Kubernetes Control Plane.
    - Evaluate node sizes based on workload availability and scale requirements.
4. **Use Similar Node Sizes for Consistent Performance:**
    - Define node size requirements for consistent workload performance.
    - Avoid burstable CPU instances like T series.
5. **Use Compute Resources Efficiently:**
    - Efficient resource usage increases scalability, availability, and performance.
    - Karpenter provisions instances on-demand based on workload needs.
6. **Automate Amazon Machine Image (AMI) Updates:**
    - Keep worker node components up to date with the latest security patches.
    - Use the latest Amazon EKS optimized AMIs for node images.
7. **Use Multiple EBS Volumes for Containers:**
    - Distribute I/O load across multiple EBS volumes to avoid bottlenecks.

EKS Cluster Services scalability:

Overview: Cluster services run inside an EKS cluster to support automation and operation, like services like NTP and syslog on a Linux server. These services, often critical during outages, should run on dedicated compute instances to avoid being impacted by user workloads.

Recommendations:

1. **Run on Dedicated Instances:**
   - Use separate node groups or AWS Fargate to ensure high uptime and reliability.

Scaling CoreDNS

Mechanisms:

1. **Reduce External Queries:**
   - Lower the ndots setting to reduce unnecessary DNS queries.
   - Fully qualify domain requests (e.g., api.example.com.).
2. **Horizontal Scaling:**
   - Add replicas to the CoreDNS deployment.
   - Use NodeLocal DNS or cluster proportional autoscaler.

Additional Tips:

- **Lameduck Duration:** Set to 30 seconds to reduce DNS lookup failures during pod termination.
- **Readiness Probe:** Use /ready instead of /health.

Scaling Kubernetes Metrics Server

Vertical Scaling:

- **Memory and CPU:** Increase resources as the cluster grows.
- **Tools:** Use Vertical Pod Autoscaler (VPA) or Addon Resizer.

Logging and Monitoring Agents

Recommendations:

1. **Use Local Metadata:**
   - Enable Use_Kubelet in Fluent Bit to fetch metadata locally.
   - Set Kube_Meta_Cache_TTL to reduce repeated API calls.
2. **Scaling Options:**
   - **Disable Integrations:** Only if metadata is not critical.
   - **Sampling and Filtering:** Reduce the number of metrics and logs collected to lower API requests and storage costs.

Workloads in EKS

Overview: Workloads that heavily use Kubernetes APIs can limit cluster scalability. Disabling unnecessary features and limiting access can help reduce load and improve security.

Recommendations:

1. **Disable Unused Features:**
   - Disable features like Secrets and ServiceAccounts if not required.
   - Implement least privilege practices.
2. **Use IPv6 for Pod Networking:**
   - Enable IPv6 before provisioning a cluster to avoid IP address exhaustion and improve performance.
3. **Limit Number of Services per Namespace:**
   - Keep services per namespace under 500 to avoid performance impacts and service discovery limits.
4. **Understand Elastic Load Balancer Quotas:**
   - Choose the appropriate load balancer type (NLB or ALB) based on your needs and adjust quotas if necessary.
   - Consider using an ingress controller for multiple services.
5. **Use Route 53, Global Accelerator, or CloudFront:**
   - Expose multiple load balancers as a single endpoint using these services.
6. **Use EndpointSlices Instead of Endpoints:**
   - Enable EndpointSlices for better scalability and performance.
7. **Use Immutable and External Secrets:**
   - Disable auto-mounting service account secrets if not needed.
   - Mark static secrets as immutable to reduce API server load.
8. **Limit Deployment History:**
   - Reduce **revisionHistoryLimit** to clean up older ReplicaSets and lower the total number of objects tracked.

Control Plane Monitoring in EKS:

API Server Monitoring:

Key Considerations:

- **Request Latency:** Measure how long it takes for the API server to service requests.
- **Queue Depth:** Check if the API server queue depth is contributing to latency.
- **API Priority and Fairness (APF):** Ensure queues are set up correctly for your API call patterns.

Metrics to Monitor:

- **API Latency:** Track request duration to understand performance.
- **Total Inflight Requests:** Monitor the number of concurrent requests to see if the API server is oversubscribed.
- **Queue Utilization:** Check for overutilized or underutilized queues to balance load effectively.

Best Practices:

- **Avoid Averaging Metrics:** Separate metrics for each API server to avoid misinterpretation.
- **Redistribute Shares:** Adjust shares in APF to balance load across queues.
- **Monitor etcd Latency:** Correlate API server latency with etcd performance to identify the root cause.

Troubleshooting:

- **Identify Slow Requests:** Use logs to find the source of latency and understand which applications are causing delays.

Kubernetes Upstream SLOs

Overview: Amazon EKS runs the same code as upstream Kubernetes releases and ensures that EKS clusters operate within the SLOs defined by the Kubernetes community. The Kubernetes Scalability Special Interest Group (SIG) defines scalability goals and investigates performance bottlenecks through SLIs and SLOs.

Key Concepts:

- **SLIs (Service Level Indicators):** Metrics used to measure system performance, such as request latency.
- **SLOs (Service Level Objectives):** Expected performance values, such as keeping request latency under 3 seconds.

Kubernetes SLOs:

- **API Request Latency (Mutating):** Latency for processing mutating API calls (e.g., create, delete) should be ≤ 1 second at the 99th percentile over 5 minutes.
- **API Request Latency (Read-Only):** Latency for processing read-only API calls should be ≤ 1 second for single resources and ≤ 30 seconds for multiple resources at the 99th percentile over 5 minutes.
- **Pod Startup Latency:** Time from pod creation to container start should be ≤ 5 seconds at the 99th percentile over 5 minutes.

API Request Latency:

- **Mutating Requests:** Changes to resources must be written to etcd, making these requests more expensive.
- **Read-Only Requests:** Can be served from cache or etcd, with different latency targets for single and multiple resources.

Pod Startup Latency:

- **Measurement:** Time from pod creation to container start, excluding image pulls and init containers.
- **Assumptions:** Worker nodes are ready, and the test is limited to stateless pods.

Kubernetes SLI Metrics:

- **Prometheus Metrics:** Used to track SLIs over time, helping to monitor and investigate performance.

Best Practices:

- **Monitor API Latency:** Regularly check request latency to ensure it meets SLOs.
- **Optimize Queue Utilization:** Balance load across queues to avoid bottlenecks.
- **Correlate with etcd Performance:** Identify if latency issues are due to the API server or etcd.

Best Practices for Cluster Upgrades:

Overview: This guide helps cluster administrators plan and execute their Amazon EKS upgrade strategy, covering self-managed nodes, managed node groups, Karpenter nodes, and Fargate nodes.

Before Upgrading:

1. **Understand Deprecation Policies:** Be aware of upcoming changes that may affect your applications.
2. **Review Kubernetes Change Log:** Check for any breaking changes that may impact your workloads.
3. **Assess Cluster Add-Ons Compatibility:** Ensure add-ons are compatible with the new Kubernetes version.
4. **Enable Control Plane Logging:** Capture logs to monitor for issues during the upgrade.
5. **Test Upgrades:** Use a non-production environment or automated tests to assess compatibility.
6. **Use eksctl for Cluster Management:** Simplify updates with eksctl.
7. **Opt for Managed Node Groups or EKS on Fargate:** Automate worker node upgrades.
8. **Utilize kubectl Convert Plugin:** Convert manifest files between different API versions.

Keeping Your Cluster Up to Date:

1. **Supported Version Policy:** EKS supports three active Kubernetes versions, with a lifecycle of 26 months.
2. **Auto-Upgrade Policy:** Stay in sync with Kubernetes updates to avoid automatic upgrades.
3. **Create Upgrade Runbooks:** Document your upgrade process and perform upgrades at least once a year.
4. **Review the EKS Release Calendar:** Stay informed about new versions and support timelines.
5. **Understand the Shared Responsibility Model:** AWS manages the control plane, but you are responsible for the data plane.

Upgrade Process:

1. **Upgrade Control Plane:** Use the AWS console or CLI to upgrade the control plane.
2. **Review Add-On Compatibility:** Upgrade add-ons and custom controllers as needed.
3. **Update kubectl:** Ensure you are using the latest version.
4. **Upgrade Data Plane:** Upgrade your nodes to match the control plane version.
5. **Use EKS Documentation:** Build a checklist for each upgrade based on the EKS version documentation.
6. **Verify Basic Requirements:** Ensure available IP addresses and IAM roles are in place.

Identifying and Remediating Removed API Usage:

1. **Use Tools to Check API Usage:** Tools like Cluster Insights, kube-no-trouble, and pluto can help identify deprecated APIs.
2. **Update Manifests:** Use kubectl-convert to update manifest files to the new API versions.

Ensuring Workload Availability:

1. **Configure PodDisruptionBudgets and TopologySpreadConstraints:** Ensure workloads remain available during upgrades.
2. **Use Managed Node Groups or Karpenter:** Simplify node upgrades with automated tools.
3. **Confirm Version Compatibility:** Ensure compatibility between managed node groups, self-managed nodes, and the control plane.
4. **Enable Node Expiry and Drift Features:** Automate node upgrades and replacements with Karpenter.

Backup and Recovery:

1. **Backup the Cluster:** Use tools like Velero to backup your cluster before upgrading.
2. **Restore if Needed:** Create new clusters and restore data if an upgrade fails.

Post-Upgrade Steps:

1. **Restart Fargate Deployments:** Redeploy workloads on Fargate nodes after upgrading the control plane.
2. **Evaluate Blue/Green Clusters:** Consider blue/green upgrades for more flexibility, despite higher costs and complexity.
3. **Track Major Changes:** Stay informed about upcoming changes in Kubernetes to prepare in advance.

Specific Guidance on Feature Removals:

1. **Dockershim Removal:** Use Detector for Docker Socket (DDS) to identify dependencies on Dockershim.
2. **PodSecurityPolicy Removal:** Migrate to Pod Security Standards or a policy-as-code solution.
3. **In-Tree Storage Driver Deprecation:** Migrate to Container Storage Interface (CSI) drivers.

Additional Resources:

- **ClowdHaus EKS Upgrade Guidance:** CLI tool for analyzing potential upgrade issues.
- **GoNoGo:** Tool to determine upgrade confidence for cluster add-ons.

Cost Optimization in Amazon EKS:

Overview: Achieving business outcomes at the lowest price point by optimizing Amazon EKS workloads.

General Guidelines:

1. **Independence from Specific Infrastructure:** Ensure workloads are flexible to run on the least expensive infrastructure types.
2. **Optimal Container Instances:** Profile environments and monitor critical metrics like CPU and memory.
3. **AWS Purchasing Options:** Utilize On-Demand, Spot, and Savings Plans for cost-effective resource management.

Cost Optimization Framework:

1. **See Pillar (Measurement and Accountability):**
   - Define and maintain a tagging strategy.
   - Use tools like Kubecost for reporting and monitoring.
   - Enable Cloud Intelligence Dashboards for cost visualization.
2. **Save Pillar (Cost Optimization):**
   - Identify and eliminate waste.
   - Architect for cost efficiency.
   - Choose the best purchasing options.
   - Adapt to evolving AWS services.
3. **Plan Pillar (Planning and Forecasting):**
   - Budget and forecast cloud costs dynamically.
   - Integrate EKS cost management with IT financial planning.
4. **Run Pillar:**
   - Implement ongoing optimization practices.

Expenditure Awareness:

1. **Use Cost Explorer:** Visualize and manage AWS costs and usage.
2. **Tagging of Resources:** Add tags to EKS clusters for cost allocation.
3. **AWS Trusted Advisor:** Use for best practice checks and recommendations.
4. **Kubernetes Dashboard:** Monitor resource usage at cluster, node, and pod levels.

5. **kubectl Commands:** Use kubectl top and kubectl describe for resource usage metrics.
6. **CloudWatch Container Insights:** Collect and summarize metrics and logs from containerized applications.
7. **Kubecost:** Deploy for visibility into Kubernetes cluster costs.

Compute and Autoscaling:

1. **Right-Size Workloads:** Use appropriate requests and limits for containers.
2. **Reduce Consumption:** Adjust workloads based on current requirements.
3. **Reduce Unused Capacity:** Use node autoscalers like Karpenter and Cluster Autoscaler.
4. **Cluster Autoscaler Priority Expander:** Prioritize scaling less expensive node groups.

Networking:

1. **Restrict Traffic to an Availability Zone:** Use topology aware routing and autoscalers.
2. **Pod Assignment and Node Affinity:** Schedule pods to nodes in the same AZ.
3. **Service Internal Traffic Policy:** Restrict pod network traffic to a node.

Storage:

1. **Ephemeral Volumes:** Use for transient local volumes.
2. **Persistent Volumes:** Use for applications needing to preserve data.
3. **Choosing the Right Volume:** Use gp3 for cost-effective block storage.
4. **Monitor and Optimize Over Time:** Use tools like AWS Trusted Advisor and AWS Compute Optimizer.

Observability:

1. **Logging:**
   - **Optimize Control Plane Logs:** Enable necessary log streams and evaluate their necessity.
   - **Stream Logs to S3:** Use CloudWatch Logs subscriptions to stream logs to S3 for cost-efficient storage.
   - **Log Retention:** Customize retention policies based on workload requirements.
   - **Log Storage Options:** Forward logs to S3 for long-term storage and use lifecycle rules to manage costs.

Reduce Log Levels: Adjust log levels to align with the criticality of the workload and environment.

2. **Metrics and Traces:**
   - **Use CloudWatch Container Insights:** Collect and summarize metrics and logs.
   - **Monitor Resource Usage:** Use tools like kubectl and Kubernetes Dashboard for real-time insights.

For more details on EKS Observability, check this documentations.

Best Practices for Windows in Amazon EKS:

Amazon EKS Optimized Windows AMI Management:

- **Components Included:** kubelet, kube-proxy, AWS IAM Authenticator, csi-proxy, containerd.
- **Retrieve AMI ID:** Use AWS Systems Manager Parameter Store API.
- **Custom AMIs:** Use Amazon EC2 Image Builder to create and maintain custom AMIs.
- **Fast Launch:** Enable Fast Launch for custom AMIs to reduce node launch times.
- **Caching Windows Base Layers:** Pre-pull necessary container images to reduce pod startup times.

Configuring gMSA for Windows Pods and Containers:

- **gMSA Accounts:** Use for Windows authentication in Active Directory.
- **Domain-Joined Nodes:** Use the node's AD Computer account for authentication.
- **Non-Domain-Joined Nodes:** Use AWS Secrets Manager and AWS Identity and Access Management (IAM) roles.

Windows Worker Nodes Hardening:

- **OS Hardening:** Apply necessary OS configurations and patches.
- **Windows Server Core:** Use for a smaller attack surface and disk footprint.
- **Avoid RDP Connections:** Use AWS Systems Manager Session Manager for secure access.
- **Amazon Inspector:** Run CIS Benchmark assessments.
- **Amazon GuardDuty:** Monitor for malicious activity.

Container Image Scanning:

- **Third-Party Tools:** Use tools like Anchore, PaloAlto Prisma Cloud, and Trend Micro for Windows container image scanning.

Windows Server Version and License:

- **Windows Server Versions:** Use Windows Server 2019 and 2022 Datacenter editions.
- **Licensing:** Amazon covers licensing costs for Windows Server-based AMIs.

Logging:

- **LogMonitor:** Use to retrieve logs from Windows containers and pipe to STDOUT.
- **DaemonSet for Log Collection:** Use fluentd or fluent-bit to stream logs to destinations like Amazon CloudWatch.
- **Log Retention and Storage:** Customize retention policies and use S3 for long-term storage.
- **General Logging Best Practices:** Log structured entries, centralize logs, keep log verbosity down, and include transaction/request IDs for traceability.

Monitoring:

- **Prometheus:** Use for collecting metrics around containers, pods, nodes, and clusters.
- **WMI Exporter:** Install on Windows nodes to export metrics.
- **ServiceMonitor:** Use to define how groups of Kubernetes services should be monitored.

Windows Networking:

- **Host Compute Service (HCS) and Host Network Service (HNS):** Manage network topology and container implementation.
- **IP Address Management:** Managed by VPC Resource Controller, with limitations on the number of pods per node based on available IPv4 addresses.
- **Direct Server Return (DSR):** Enable to reduce port exhaustion and improve communication speed.
- **Container Network Interface (CNI) Options:** Use AWSVPC CNI or Calico CNI for networking.
- **Network Policies:** Use Project Calico for network policy management.

Avoiding OOM Errors:

- **Memory Management:** Windows uses virtual memory and pagefiles to handle memory allocation, reducing the risk of OOM errors.
- **Reserving System and Kubelet Memory:** Combine flags to manage NodeAllocatable and control memory allocation per pod.

Pod Security Contexts:

- **Pod Security Policies (PSP) and Pod Security Standards (PSS):** Use PSS for enforcing security going forward.
- **Security Context Settings:** Apply privileges, restrict capabilities, and manage user contexts.
- **Windows Security Context Options:** Use windowsOptions for settings like runAsUserName.

Persistent Storage Options:

- **In-Tree vs. Out-of-Tree Volume Plugins:** Use CSI for out-of-tree plugins to manage storage independently of Kubernetes releases.
- **In-Tree Volume Plugin for Windows:** Use awsElasticBlockStore with NTFS as the fsType.

Hardening Windows Container Images:

- **Account Security Policies:** Enforce strong password and lockout policies.
- **Audit Policies:** Enable to monitor user activities and ensure compliance.
- **IIS Security Best Practices:** Implement security best practices for IIS.
- **Principle of Least Privilege:** Apply the least privilege principle to minimize security risks.

ADVANCED EKS FEATURES

Fargate for EKS:

- **On-demand Compute Capacity:** Fargate provides on-demand, right-sized compute capacity for containers, eliminating the need to provision, configure, or scale groups of virtual machines manually.
- **Simplified Management:** With Fargate, you don't need to choose server types, decide when to scale your node groups, or optimize cluster packing.
- **Fargate Profiles:** Control which Pods start on Fargate and how they run using Fargate profiles, which are defined as part of your Amazon EKS cluster.
- **Integration with Kubernetes:** Amazon EKS integrates Kubernetes with Fargate using AWS-built controllers that run as part of the Amazon EKS managed Kubernetes control plane.
- **Pod Scheduling:** When you start a Pod that meets the criteria for running on Fargate, the Fargate controllers recognize, update, and schedule the Pod onto Fargate.

Considerations for Using Fargate:

- **Isolation:** Each Pod running on Fargate has its own isolation boundary, meaning they don't share the underlying kernel, CPU resources, memory resources, or elastic network interface with another Pod.
- **Load Balancers:** Network Load Balancers and Application Load Balancers can be used with Fargate with IP targets only.
- **Service Connectivity:** Fargate exposed services run on target type IP mode, not node IP mode. Connectivity between services running on managed nodes and Fargate can be checked via service name.
- **Profile Matching:** Pods must match a Fargate profile at the time they're scheduled to run on Fargate. Pods that don't match might be stuck as Pending.
- **Limitations:** Fargate does not support Daemonsets, privileged containers, HostPort, or HostNetwork. The default nofile and nproc soft limit is 1024 and the hard limit is 65535. GPUs are not currently available on Fargate.
- **Subnet Requirements:** Pods running on Fargate are only supported on private subnets with NAT gateway access to AWS services.

Scaling with Fargate:

Vertical Pod Autoscaler: Use this to set the initial correct size of CPU and memory for your Fargate Pods.

- **Horizontal Pod Autoscaler:** Use this to scale your Pod deployments. Set the Vertical Pod Autoscaler mode to Auto or Recreate for automatic redeployment with larger CPU and memory combinations.

EKS Add-ons:

- **Operational Capabilities:** Add-ons provide supporting operational capabilities to Kubernetes applications, such as observability agents or Kubernetes drivers.
- **Automatic Installation:** Amazon EKS automatically installs self-managed add-ons like the Amazon VPC CNI plugin for Kubernetes, kube-proxy, and CoreDNS for every cluster.
- **Custom Add-ons:** You can view and install additional add-ons using eksctl, the AWS Management Console, or the AWS CLI. Some add-ons may require IAM permissions and an IAM OpenID Connect (OIDC) provider for your cluster.

Available EKS Add-ons:

- **Amazon VPC CNI Plugin:** Provides native VPC networking for your cluster.
- **CoreDNS:** A flexible, extensible DNS server that serves as the Kubernetes cluster DNS.
- **Kube-proxy:** Maintains network rules on each Amazon EC2 node.
- **Amazon EBS CSI Driver:** Provides Amazon EBS storage for your cluster.
- **Amazon EFS CSI Driver:** Provides Amazon EFS storage for your cluster.
- **Mountpoint for Amazon S3 CSI Driver:** Provides Amazon S3 storage for your cluster.
- **CSI Snapshot Controller:** Enables the use of snapshot functionality in compatible CSI drivers.
- **AWS Distro for OpenTelemetry:** A secure, production-ready distribution of the OpenTelemetry project.
- **Amazon GuardDuty Agent:** A security monitoring service that analyzes and processes foundational data sources.
- **Amazon CloudWatch Observability Agent:** Provides enhanced observability for Amazon EKS.
- **EKS Pod Identity Agent:** Manages credentials for your applications.

Custom AMIs:

- **AmazonLinux2:** The default EKS AMI image.
- **AmazonLinux2023:** A newer version of the Amazon Linux AMI.
- **Ubuntu1804:** Supported for EKS versions up to 1.29.
- **Ubuntu2004:** Supported for EKS versions up to 1.29.
- **Ubuntu2204:** Available for EKS versions 1.29 and later.
- **UbuntuPro2204:** Available for EKS versions 1.29 and later.
- **Bottlerocket:** A container-optimized operating system.
- **WindowsServer2019FullContainer:** A Windows Server 2019 AMI with full container support.
- **WindowsServer2019CoreContainer:** A Windows Server 2019 AMI with core container support.
- **WindowsServer2022FullContainer:** A Windows Server 2022 AMI with full container support.
- **WindowsServer2022CoreContainer:** A Windows Server 2022 AMI with core container support.

Considerations for Custom AMIs:

- Ensure that instances can properly join and function within the Kubernetes cluster when using custom AMIs.

TROUBLESHOOTING AND SUPPORT

Common Issues and Solutions:

- **Pod Scheduling Issues:** Ensure Pods match Fargate profiles or node group configurations.
- **Networking Problems:** Verify VPC configurations, security groups, and network policies.
- **Resource Limits:** Check for CPU, memory, and storage limits in your cluster.

AWS Support Options:

- **AWS Support Plans:** Choose from Basic, Developer, Business, and Enterprise support plans based on your needs.
- **AWS Support Center:** Access support cases, documentation, and AWS Trusted Advisor.
- **AWS Premium Support:** Get 24/7 access to Cloud Support Engineers for critical issues.

Community Resources:

- **AWS Forums:** Engage with the AWS community to ask questions and share knowledge.
- **GitHub Repositories:** Explore and contribute to open-source projects related to EKS.
- **Stack Overflow:** Find answers to common EKS issues from the developer community.

FUTURE OF EKS

Roadmap:

- **Feature Requests:** Submit and vote on feature requests to influence the EKS roadmap.
- **Development Updates:** Stay informed about the latest developments and upcoming releases.
-

Upcoming Features:

- **Enhanced Security:** New features to improve cluster security and compliance.
- **Performance Improvements:** Optimizations for better performance and scalability.
- **New Integrations:** Additional integrations with other AWS services and third-party tools.

For the latest updates and roadmap details, visit the AWS Containers Roadmap on GitHub.

RESOURCES

- https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html
- https://docs.aws.amazon.com/eks/latest/best-practices/introduction.html
- https://docs.aws.amazon.com/eks/latest/APIReference/Welcome.html
- https://www.eksworkshop.com/