Architecting Amazon EKS and Bottlerocket for PCI DSS Compliance

November 14, 2023

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers. Additionally, this document is not legal advice and should not be relied on as legal advice. AWS encourages its customers to obtain appropriate advice on their implementation of privacy and data protection environments, and more generally, applicable laws relevant to their business.

This document was developed by AWS Security Assurance Services, LLC (AWS SAS), which is a fully owned subsidiary of Amazon Web Services. AWS SAS is an independent PCI Qualified Security Assessor (QSA) company (QSAC) that provides AWS customers and partners with specific and prescriptive information for achieving PCI DSS compliance in the AWS Cloud.

Contents

Abstract

Companies increasingly use microservices and containers on AWS to support their sensitive data workloads. This paper outlines the best practices that customers should consider when configuring Amazon Elastic Kubernetes Service (Amazon EKS) for their Amazon Elastic Compute Cloud (Amazon EC2) launch type with the Bottlerocket operating system for PCI DSS version 4.0 compliance. This paper covers a subset of PCI DSS controls because some are not applicable to containers and customer environments vary. The intended audience is system architects, developers, security personnel, and risk and compliance personnel who are interested in architecting their AWS Cloud environments for PCI DSS compliance.

Introduction

The Payment Card Industry Data Security Standard (PCI DSS) provides technical and operational guidance on securing payment card processing environments that is applicable to people, processes, and technology. Entities that store, process, or transmit cardholder data (CHD) must validate compliance of their cardholder data environment (CDE) against the PCI DSS controls. Examples of such entities include merchants, payment processors, and service providers.

AWS provides many services that have been attested to align with PCI DSS compliance. For more information, see AWS Artifact, which is a central resource for compliance-related information that provides on-demand access to AWS security and compliance reports and select online agreements. Companies can use these services to help reduce compliance efforts. One area of continued growth is the use of AWS containerized solutions.

If an AWS service is listed as PCI DSS compliant, this does not mean that by default the use of that service makes a customer's environment compliant. Rather, it means that customers have the ability to configure the service to align with PCI DSS requirements. Where customers can access and configure parameters, the customer is responsible for making sure that these parameters are configured to meet applicable compliance requirements. There may be additional AWS services that are not included in the AWS PCI DSS assessment that customers can still use to meet PCI DSS controls.

AWS container solutions include our managed services: Amazon Elastic Container (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS). Each service supports deployment containers on either AWS Fargate or Amazon Elastic Compute Cloud (Amazon EC2). Fargate is a serverless compute engine that removes the need to provision and manage EC2 instances. For EC2 deployment, customers manage the underlying EC2 instances that host the containers.

The benefits of transitioning workloads to container services include operating system independence, speed of deployment, and more efficient use of resources. It's important, as with any cloud workload, to understand how to architect for security when adopting containers. The transient and dynamic nature of container environments creates a continuously changing CDE that can be difficult to assess.

Attack vectors for containerized applications are similar to those faced by non–container-based application deployments with the addition of the container orchestration layer, referred to as the *control plane*. As with other application deployments, we

recommend that you continue to operate within best practices, including adherence to Open Web Application Security Project (OWASP) concerns.

Container functions are typically architected to perform primary tasks, which in turn creates a distributed environment. The services that containers implement become more network interdependent and require scheduling, scaling, and resource management. Unlike virtual machines, containers share the operating system's kernel.

This setup can provide a common point of entry that a threat actor can use to access the containers running on that host. When running multiple containers on a single operating system, the containers might share a common network interface. You can help improve your security by using Bottlerocket in your environment. Bottlerocket is a Linux-based open-source operating system that is purpose-built by AWS for running containers. Bottlerocket includes only the essential software required to run containers, and helps ensure that the underlying software is always secure. With Bottlerocket, you can help reduce maintenance overhead and automate your workflows by applying configuration settings consistently as nodes are upgraded or replaced. In this whitepaper, we will discuss the various solutions that you can build on AWS services to mitigate security risks.

PCI DSS compliance status of AWS services

AWS is a Level 1 PCI DSS Service Provider, which can make it simpler for you to meet your compliance requirements. The scope of the AWS PCI DSS assessment assumes that for each service, data provided by the customer could include primary account numbers (PAN) and sensitive authentication data (SAD), or impact the security of such data. The assessment also includes all physical security requirements that apply to AWS data centers that support PCI DSS in-scope services.

The AWS Services in Scope by Compliance Program page lists the AWS services that were included in the annual PCI DSS assessments, along with other services by compliance programs. At AWS, we work to continually maintain our service's alignment with compliance standards and verify that new features are assessed to determine if they can inherit the compliance status of the parent. We routinely add new services to the AWS portfolio, and as these are added to our assessment, they will also appear in the list of Services in Scope. Customers can access AWS compliance documentation through the AWS Management Console by using AWS Artifact.

AWS Shared Responsibility Model

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

The following figure provides an overview of the shared responsibility model. The line of responsibility might vary depending upon the implemented AWS service.

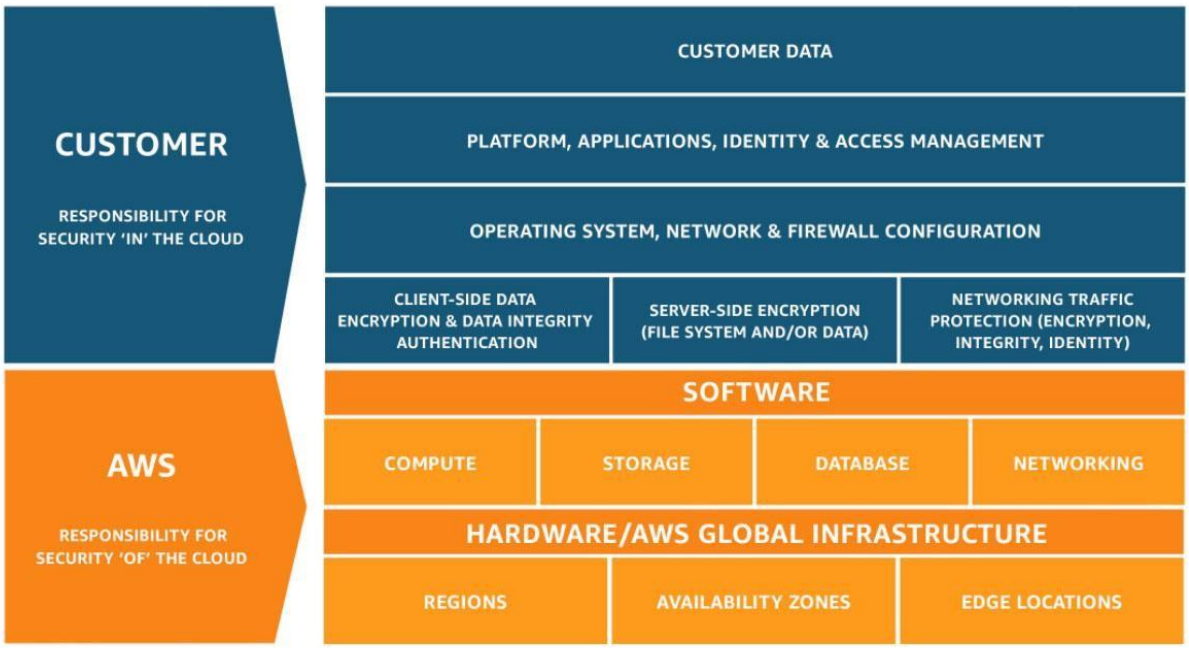

Figure 1: Shared responsibility model

AWS is responsible for the security and compliance **of** the Cloud, which refers to the infrastructure that runs all of the services offered in the AWS Cloud. Cloud security at AWS is the highest priority. AWS customers benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations. This infrastructure consists of the hardware, software, networking, and facilities that run AWS Cloud services.

Customers are responsible for security and compliance **in** the Cloud, which consists of customer-configured systems and services provisioned on AWS. For PCI DSS compliance, the customer responsibility includes all system components, including AWS

resources, in or connected to their CDE. For abstracted services, such as Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB, the customer responsibility includes configuring controls, such as access controls, log settings, data lifecycle policies, and encryption settings.

The division of responsibility depends on the AWS service and implementation that the customer selects. Amazon EKS is a good example: when using EKS, customers can choose a serverless deployment of containers with Fargate, or run containers on EC2 infrastructure that the customer can access.

With Fargate, customers are abstracted from the underlying host and are not responsible for updating or patching the host system. This is in contrast to an EKS deployment that uses EC2 hosts, where the customer must take on a greater level of responsibility, such as controlling host access and applying security patches. Customers that deploy their EC2 instances with the Bottlerocket Amazon Machine Image (AMI) can take advantage of automatic security updates and reduced exposure to security events by including only the essential software to host containers.

If a customer can control a service parameter, then they are responsible for making sure that it is configured to meet PCI DSS requirements.

PCI DSS scope determination and validation

It is critical that you understand the complete flow of CHD within your environment. The CHD flow determines the applicability of the PCI DSS and defines the boundaries and components of a CDE, and therefore, the scope of a PCI DSS assessment. Accurate determination of the PCI DSS scope is key to defining the security posture of the assessed workload and ultimately a successful assessment. You must have a procedure for scope determination that assures its completeness and detects changes or violations of the scope. Typically, the following steps comprise the PCI DSS scope identification:

1. **Identify the CHD flow.** Define the lifecycle of CHD, including the path of consumption or entry of CHD in your environment, the subsequent processing and storing of CHD, and eventually the secure destruction, devaluation, or exit of CHD from your environment.
2. **Identify all in-scope resources in your environment.** Identify the various types of AWS resources involved in receiving, processing, storing, and transmitting the CHD that comprises the CHD flow.

3. **Categorize the system.** Categorize systems into abstracted and infrastructure services. The scope identification and segmentation of those resources are based on different types of connection, namely infrastructure service (OSI Layer 3–4) connection and abstracted services (OSI layer 7).
4. **Design segmentation boundaries.** Design segmentation boundaries to help ensure that other AWS resources not involved in the CHD flow are segmented from the CDE and thus can be excluded from the PCI DSS scope.

The ephemeral nature of containerized applications provides additional complexities when considering a dynamically changing scope. As a result, you need to maintain an awareness of container configuration parameters to make sure that compliance requirements are addressed throughout each phase of a container lifecycle, as we will illustrate in the following sections. The Architecting for PCI DSS Scoping and Segmentation whitepaper is a detailed resource that you can use to help understand scoping and segmentation within your AWS environment.

Securing an Amazon EKS deployment

The following sections provide guidance on key topics that you should consider when architecting a container-based environment for PCI DSS compliance. These sections comprise the following categories.

- Network segmentation
- Host and container image hardening
- Data protection
- Restricting user access
- Tracking and monitoring access
- Vulnerability scanning and penetration testing

Each section provides an overview of the requirements and best practice recommendations to help you meet your compliance requirements. The guidance is not all inclusive, because each customer's environment is unique. Collectively, the following recommendations provide a defense-in-depth approach to securing a container-based environment.

Network segmentation

Controls within requirement 1 of the PCI DSS call for installing and maintaining network security controls to protect CHD, and require that systems be protected from unauthorized access. Network security controls, in this context, means that inbound and outbound access must be restricted to only approved ports and services. Although the use of network segmentation is not a PCI DSS requirement, its usage is a highly relevant tool to reduce the scope of a customer's environment. Segmentation is often colloquially referred to as *Requirement 0*.

A dedicated AWS account provides the highest level of segmentation boundary that you can achieve on the AWS Cloud. By design, all resources provisioned within an AWS account are logically isolated from resources provisioned in other AWS accounts, even within your own organization in AWS Organizations. By using an isolated account for PCI DSS workloads, you can help establish strong access segmentation when designing your PCI application to run on the AWS Cloud.

Amazon Virtual Private Cloud (Amazon VPC) and subnets provide further logical isolation of CDE-related resources. You can deploy a VPC and subnets that meet the Amazon EKS requirements through manual configuration, or by deploying the VPC and subnets using `eksctl` or an AWS CloudFormation template provided by Amazon EKS. Both `eksctl` and the CloudFormation template create the VPC and subnets with the required configuration.

By default, all pod-to-pod communication is allowed within a Kubernetes cluster. Kubernetes network policies provide a mechanism to restrict network traffic not only between pods but also between pods and external services. Kubernetes network policies operate at layers 3 and 4 of the Open Systems Interconnection (OSI) model. Network policies use pod selectors and labels to identify source and destination pods, but can also include IP addresses, port numbers, protocol numbers, or a combination of these. Project Calico is an open-source policy engine from Tigera that helps with network policy enforcement and works well with Amazon EKS.

In addition to implementing the full set of Kubernetes network policy features, Calico supports extended network polices with a richer set of features, including policy ordering and prioritization, deny rules, and support for layer 7 rules (for example, HTTP, when integrated with service mesh such as AWS App Mesh or Istio).

You can scope Calico policies to namespaces, pods, service accounts, or globally. When you scope policies to a service account, Calico associates a set of inbound and

outbound rules with that service account. With the proper Kubernetes role-based access control (RBAC) rules in place, you can prevent teams from overriding these rules, allowing IT security professionals to safely delegate administration of namespaces. When you first provision an EKS cluster, the Calico policy engine is not installed by default. For instructions on how to install it, see Installing the Calico network policy engine add-on. To review the manifests for installing Calico, see the VPC CNI GitHub repository.

On AWS, security groups act as a virtual firewall and provide stateful inspection. You can use security groups to restrict communications by IP address, port, and protocol. It is important to note that, by default, security groups allow all outbound communications. As a result, you must configure outbound connection rules to meet PCI DSS requirements. You can do this by using security groups or an inline security appliance, such as AWS Network Firewall.

Network Firewall is a fully managed, highly-scalable inline firewall that you can use to help secure traffic moving in and out of your CDE. It supports stateless and stateful firewall rules by using a standard rule format.

You can also use any of the several supported complementary integrations to provide threat intelligence, centralized management, and managed rule sets. You can deploy Network Firewall in a number of different ways to protect east-west (VPC-to-VPC) traffic, as well as north-south (VPC-to-internet and internet-to-VPC) traffic.

Amazon EKS uses VPC security groups to control the traffic between the Kubernetes control plane and the cluster's worker nodes. It also uses security groups to help control the traffic between worker nodes, external IP addresses, and other VPC resources. We strongly recommended that you use a dedicated security group for each control plane (one for each cluster). For information about the minimum and suggested rules for the control plane and node group security groups, see Amazon EKS security group requirements and considerations.

To control communication between services that run within the cluster and services that run outside of the cluster, consider using security groups for pods, which integrate EC2 security groups with Kubernetes pods.

You can use EC2 security groups to define rules that allow inbound and outbound network traffic to and from pods that you deploy to nodes running on many EC2 instance types, or on Fargate. For a complete list of supported instances, see the amazon-vpc-resource-controller-k8s GitHub repository. Your nodes must be one of the

supported instance types. Before you deploy security groups for pods, consider the limits and conditions as discussed within Tutorial: Security groups for Pods.

Fargate runs each pod in its own dedicated kernel runtime environment and does not share CPU, memory, storage, or network resources with other pods, which helps ensure improved workload isolation and security. However, because Kubernetes is a single-tenant orchestrator, potential pod-level intercommunication still exists. You should group sensitive workloads that need complete security isolation by using separate EKS clusters.

Optimally, you should group containerized workloads based on data sensitivity levels to facilitate network segmentation. Additionally, Kubernetes namespaces allow for resource segmentation inside the Kubernetes cluster with logical isolation from each other. Namespaces provide scope for pods, services, and deployments in the cluster, so that users that interact with one namespace will not see content in another namespace. However, namespaces within the same cluster do not restrict communication between them. You need network policies for granular control and to restrict such communications between namespaces.

In summary, consider the following when working to isolate containerized application communications:

- Isolate pods on separate nodes based on sensitivity of services, and isolate CDE workloads in a separate cluster with a dedicated security group.
- Use security groups to limit communication between nodes and the control plane, and external communications.
- Implement micro-segmentation with Kubernetes network policies and consider using service mesh, Networking and Cryptography library (NaCl) encryption, and Container Network Interfaces (CNIs), to help limit and secure communications.
- Implement a network segmentation and tenant isolation network policy. Network policies are similar to security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods by using pod selectors and labels. For more information, see Installing the Calico network policy engine add-on.
- Make sure that ingress and egress for your CDE is secured, and that allowed flows are documented.

Host and container image hardening

Requirement 2 of the PCI DSS emphasizes the need to make sure that resources in-scope for the PCI DSS are configured and managed securely. Amazon container services, such as Amazon EKS, are run on container-optimized Amazon Machine Images (AMIs). These operating systems only contain additional libraries that are essential for container deployments, and as a result, help to minimize security risks.

Customers are still responsible for maintaining compliance of all configurations and functions at the operating system, network, and application layers. You should routinely patch operating systems by using AWS Systems Manager and disable or remove nonessential services and libraries. You should also establish configuration standards that are consistent with industry-accepted system hardening guidelines, such as the Center for Internet Security (CIS) Benchmarks for EC2 instance types. For additional AWS secure configuration standards support, see the AWS Security Learning website.

You should limit container builds to only required resources and adopt a model of microservices where a container provides one primary function. Software architects should make sure that images do not rely on outdated software libraries and applications that might contain known vulnerabilities. A best practice is to rebuild container images in the container registry on a periodic basis to help ensure that the latest application versions are in use. The usage of vulnerable libraries could introduce avenues of unauthorized activity that are often overlooked.

When managing containers, you should make sure that they are immutable and not patched in place. You should create trusted base container images that have been assessed and confirmed to use patched libraries and applications. Use a trusted registry to secure container images, such as Amazon Elastic Container Registry (Amazon ECR). Amazon ECR provides image scanning based upon the Common Vulnerabilities and Exposures (CVE) database and can identify common software vulnerabilities.

You can use Amazon Inspector as an automated security assessment service to help improve the security and compliance of applications deployed on AWS. Inspector automatically assesses applications for exposure, vulnerabilities, and deviations from best practices.

In addition, Amazon ECR integrates with AWS Signer to provide a way for customers to sign container images. AWS Signer makes it simpler for you to manage signing keys and certificates because it provides auditable logs and stores signatures alongside images in repositories. Upon signing images, you can choose your preferred

Kubernetes admission controllers such as Gatekeeper or Kyverno to help enforce image verification before deploying images on your clusters.

You are responsible for making sure that your EC2 instances run appropriate anti-malware and file integrity monitoring software when choosing the EC2 launch type for Amazon EKS. We recommend that you use a special purpose operating system such as Bottlerocket that includes a reduced attack surface, a disk image that is verified on boot, and enforced permission boundaries by using SELinux.

Alternatively, consider using the Fargate launch type, which provides on-demand, right-sized compute capacity for containers. With Fargate, you no longer have to provision, configure, or scale groups of virtual machines to run containers. This option removes the need to choose server types, decide when to scale your node groups, or optimize cluster packing. Another advantage of using Fargate is that AWS takes care of hardening, patching, and monitoring the host system and the worker node. For additional considerations for choosing Fargate, see AWS Fargate.

We recommend that you consider implementing policy governance that can enforce security and compliance polices with tools such as Gatekeeper, Open Policy Agent (OPA), and dockerfile-lint.

In summary, consider the following points for host and image hardening:

- Use an operating system that is optimized for running containers.
- Minimize access to worker nodes and deploy the worker nodes in a private subnet.
- Run Inspector to assess hosts for exposure, vulnerabilities, and deviations.
- Use minimal container images, scan images for vulnerabilities regularly, and sign them to help ensure that only approved images are used inside your organization.

Data protection

The PCI DSS controls within requirements 3 and 4 are focused on the need to protect sensitive data while at rest and in transit. AWS provides a number of PCI DSS compliant services and features to assist with these compliance efforts.

Workloads that contain sensitive data, such as CHD, should secure all storage of data. You should store data on secure file stores or databases and not on the underlying

container host. System architects should be mindful of volume mounts and sharing of data between containers, such as host file systems and temporary storage.

Make sure to secure sensitive data and environment variables, such as database connection strings that are contained within container build files. Many AWS services integrate with AWS Key Management Service (AWS KMS), a PCI DSS compliant service that provides encryption key management functionality including secure encryption key storage, access controls, and annual rotation.

AWS Secrets Manager and AWS Systems Manager Parameter Store are two services that you can use to help secure sensitive data within container build files. Systems Manager Parameter Store provides secure, hierarchical storage of data with no servers to manage. You can establish granular access and audit controls to help ensure appropriate restrictions are in place to meet compliance requirements. You can encrypt data stored within Systems Manager Parameter Store by using AWS KMS.

Like Systems Manager Parameter Store, Secrets Manager uses AWS KMS to help secure data. Secrets Manager provides additional capabilities that include random password generation and automatic password rotation. AWS KMS is a PCI DSS compliant service that is integrated with many other AWS services. Users can create and manage cryptographic key material, and control who can access and use the encryption keys.

For data in transit, PCI DSS requires that sensitive information be encrypted during transmission over open, public networks. Customers are responsible for configuring strong cryptography and security controls.

AWS provides multiple services, such as Amazon API Gateway and Application Load Balancer, that support the use of TLS. You can apply policies to the services to help enforce the use of strong cryptography.

API Gateway and Application Load Balancer also support use of the integrated AWS WAF to secure communications at the application layer. AWS WAF helps protect applications and APIs against common web exploits, such as those identified within the OWASP Top 10.

Traffic exchanged between Nitro System instance types is automatically encrypted by default, except when there is an intermediate network boundary, such as AWS Transit Gateway or a load balancer. In these cases, the traffic is not encrypted.

You can also implement encryption in transit for inter-pod communication by using a service mesh, such as AWS App Mesh with support for mTLS.

You can use inbound traffic controllers to intelligently route HTTP and HTTPS traffic that emanates from outside the cluster to services running inside the cluster. Often, inbound traffic is fronted by a layer 4 load balancer, such as the Classic Load Balancer or Network Load Balancer. You can configure an inbound traffic controller to end SSL and TLS connections.

In summary, consider the following points for data protection:

- Use AWS KMS for service-managed encryption keys and AWS managed customer managed keys, and rotate your customer managed keys periodically.
- Enable support for strong encryption in transit.
- Use envelope encryption of Kubernetes secrets in Amazon EKS to add a customer-managed layer of encryption for application secrets or user data that is stored within a Kubernetes cluster.

Restricting user access

The controls within requirements 7 and 8 of the PCI DSS are focused on restricting access to authorized personnel and ensuring that appropriate access controls are in place. Access to resources should embrace a least privilege model where access is on a need-to-know basis. User access to containers and the underlying host should be authenticated with strong authentication requirements that align with the PCI DSS.

You should run container images with non-privileged user accounts. For instance, container build files that do not contain defined user credentials will run as the root account by default. This setup means that a compromised container service might extend root privileges to a threat actor who could use the elevated access to further affect the underlying host.

Unlike Fargate where no host access is available, Amazon EKS with the EC2 launch type provide the option to enable secure shell (SSH) access for underlying system management. With container-optimized operating systems such as Bottlerocket, you can improve your security posture immediately because all shells, interpreters, and package managers are removed from the image by default. Bottlerocket provides the control container and admin container—both of which run outside of the orchestrator in a separate instance of containerd. The control container runs the Amazon SSM agent that you can use to run commands or start shell sessions, on Bottlerocket instances in Amazon EC2. The admin container has an SSH server that you can use to log in as `ec2-user` by using your EC2-registered SSH key.

To help create and establish secure container images, restrict all access to container images. Container deployments should use a private container registry that restricts access and write permissions, such as Amazon ECR, which integrates with AWS Identity and Access Management (IAM) for access controls. Amazon ECR is a scalable container repository that helps provide secure storage and transmission of container images. The simplified workflow and integration of Amazon ECR with AWS services reduces the need for access to the underlying hosts, making it simpler to implement the principle of least privilege.

With Amazon EKS and its required IAM authenticator, users sign in to the cluster with an IAM identity—either an IAM user or IAM role. Kubernetes then determines what actions the user can perform through its role-based access control (RBAC). You must configure IAM roles to set up authorization at the cluster and infrastructure level. With RBAC, you can set up authorization to the resource level (for example, a particular pod) or service level (for example, a pod). Employ least privileged access when you create `Roles` or `RoleBindings` for namespace level resources and `ClusterRole` or `ClusterRoleBindings` for cluster-level resources.

When you create an EKS cluster, the IAM entity user or role (such as a federated user that creates the cluster) is automatically granted `system:masters` permissions in the cluster's RBAC configuration. This access cannot be removed and is not managed through the `aws-auth ConfigMap`. Therefore, it is a best practice to create the cluster with a dedicated IAM role and regularly audit who can assume this role. This role should not be used to perform routine actions on the cluster; instead, additional users should be granted access to the cluster through the `aws-auth ConfigMap` for this purpose. For more information, see Managing users or IAM roles for your cluster.

In summary, consider the following points for user access:

- Employ least privileged access to AWS resources when you create `RoleBindings` and `ClusterRoleBindings`.
- Use IAM roles when multiple users need identical access to the cluster and IAM roles for service accounts where possible.
- Make the endpoint for the Amazon EKS cluster private.
- Create the cluster with a dedicated IAM role, which is regularly audited.
- Regularly audit access to the cluster.
- Run the application as a non-root user.

Tracking and monitoring access

The core control within requirement 10 of the PCI DSS is the need to use logging mechanisms to track, monitor, and alert on potentially anomalous activities. In a dynamic, containerized environment, you must maintain a robust, centralized logging infrastructure and make sure that logs are shipped immediately from the container to a secure store for retention and analysis.

Event logging

Use AWS event log services to establish event log monitoring at the network, host, and container level. Optionally, you can enable VPC Flow Logs to capture network traffic that details packet information, such as the protocol, port, and source and destination address information. Monitor the health, efficiency, and availability of container hosts by making sure that Amazon CloudWatch or Amazon Kinesis agents are enabled and configured.

Enable event logging capabilities within the containerized applications to capture application and container event log data. Use CloudWatch dashboards to monitor and alert on the captured event log activity. Store the captured event data securely within encrypted S3 buckets to help you meet your retention requirements.

Amazon EKS with Fargate supports a built-in log router, which means there are no sidecar containers to install or maintain. With the log router, you can use the breadth of services at AWS for log analytics and storage. You can stream logs from Fargate directly to CloudWatch, Amazon OpenSearch Service, and Kinesis Data Firehose destinations such as Amazon S3, Kinesis Data Streams, and other third-party tools. Fargate uses a version of Fluent Bit, an upstream conformant distribution of Fluent Bit that AWS manages. For more information, see the aws-for-fluent-bit GitHub repository.

To maintain a holistic view of your environment, use AWS tools. You can use Amazon Athena and Amazon CloudWatch Logs Insights to query and analyze audit trail logs that are saved to Amazon S3 from VPC Flow Logs, CloudTrail, and CloudWatch.

We strongly recommended that you use a dedicated audit account, to which access is strictly limited, and in which you store all security and operational logs, including CloudTrail and application logs. Within this account, consider the use of configuration options, such as S3 MFA-delete, S3 file versioning, and S3 Lifecycle Policies, to help ensure that data is retained and cannot be tampered with. Similarly, we recommend that you use AWS Organizations and service control policies to help ensure that critical configuration, such as security logging configuration, cannot be altered or disabled.

Finally, you should consider the use of tools to support the assessment of risk and compliance within your accounts. AWS Audit Manager has support for the latest PCI DSS versions 3.2.1 and 4.0 through a prebuilt framework and offers features such as automated evidence collection and audit-ready reports.

In summary, consider the following points for monitoring and logging:

- Enable EKS cluster audit logs.
- Use Kubernetes audit metadata annotations for authorization history tracking.
- Create alarms for suspicious events.
- Analyze logs with CloudWatch Logs Insights.
- Audit your CloudTrail logs.
- Use Organizations and service control policies to help ensure that security controls cannot be circumvented or disabled.

Network intrusion detection

Controls within requirement 11 of the PCI DSS specify the use of intrusion-detection and intrusion-prevention techniques to help detect and prevent intrusions into the network. The standard requires monitoring of all traffic at the perimeter and critical points of the CDE. With most on-premises environments, the requirements are met by using Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) appliances. You can use a similar approach on AWS. For containerized environments, you can perform inspection of network traffic at the network layer outside of the container host and within the container management software's virtual container network.

There are several options for inspection of network data outside of the container host on AWS. Amazon GuardDuty is a managed service that provides threat detection across multiple AWS data sources to help identify threats. It uses machine learning, anomaly detection, and threat intelligence to help identify illicit network activity based on CloudTrail, VPC Flow Logs, and Amazon Route 53 DNS logs. It also processes features such as EKS Audit Log Monitoring and EKS Runtime Monitoring to detect potentially suspicious activities and threats by using Kubernetes audit logs and operating system-level events, respectively. For more information, see EKS Protection in Amazon GuardDuty.

For a traditional IDS and IPS solution, you can configure Amazon VPC Traffic Mirroring to route a copy of all network communications to a virtual appliance that runs on one or more EC2 instances.

Another common solution is to use a transit network architecture that uses IP routing to help ensure that all network traffic crosses a single network. By implementing this architecture, you can use a virtual IDS or IPS device from the AWS Marketplace to inspect all traffic that transits between networks. You can also use a Gateway VPC endpoint to route all traffic to on-premises IDS or IPS infrastructure. Lastly, you can use host-based IDS or IPS solutions to inspect traffic as it is delivered to an EC2 instance.

Inspection of inter-container communications on the virtual container network is another viable option. There are vendors within the AWS Marketplace that provide IDS container solutions, which mostly use a side container to monitor and alert on unusual traffic patterns. Agent-based solutions are also available and use machine learning to detect anomalous communication patterns among the containers.

The security measures put into place will depend heavily on the architecture of the environment. Traffic detection at the network layer requires advanced planning of container deployments and traffic patterns.

Vulnerability scanning and penetration testing

The PCI DSS control requirement 11.3 requires organizations to regularly test systems and processes to identify vulnerabilities and remediate such findings in a timely manner. You should perform vulnerability scanning on a quarterly basis, as well as after significant changes to your environment. Similarly, you should perform penetration testing on an annual basis and after significant environment changes. For certain permitted services, penetration testing of AWS resources is allowed at any time. For further details, see the AWS support policy for penetration testing, or consult with your account team.

Service providers that use network segmentation are required to test the effectiveness of segmentation controls every six months or after any changes to the segmentation controls.

The scope of the assessment activities include the CDE and ancillary systems that are used in support of the CDE. For scope and methodology guidance for penetration testing, see the PCI DSS Information Supplement: Guidance for PCI DSS Scoping and Network Segmentation and Information Supplement: Penetration Testing Guidance. For

scope reduction strategies, see the AWS whitepaper Architecting for PCI DSS Scoping and Segmentation on AWS.

Depending your environment, your test requirements might apply to on-premises, cloud resources, and containerized environments. When you deploy Amazon EKS on EC2 instances, you must perform vulnerability scanning of the underlying host.

According to the PCI DSS requirement, customers are responsible for establishing a process to identify security vulnerabilities, and assigning a risk ranking to newly discovered security vulnerabilities. Inspector is a security assessment tool that helps identify vulnerabilities and prioritizes findings by level of severity. Integration of Inspector within the DevOps process provides for assessment automation to proactively identify vulnerabilities and to check for unintended network accessibility of your nodes.

You can also use container-specific scanning tools to scan container images for vulnerabilities. Container scanning can help identify non-compliant code, vulnerable libraries, and potentially exposed secrets. Security vendors within the AWS Marketplace provide solutions capable of scanning systems, containers, and applications.

When you perform internal and external penetration testing, you should do assessment activities at both the network and application layer, and target the underlying host and containerized applications. Patch container hosts to address vulnerabilities and update container images to mitigate identified container vulnerabilities. Create golden images for containers and securely store them within private container registries, such as Amazon ECR.

The Center for Internet Security (CIS) Kubernetes Benchmark provides guidance for Amazon EKS node security configurations. You can run it by using kube-bench, a standard, publicly available tool for comparing the configuration of your Kubernetes clusters to the CIS benchmark. To learn more, see Introducing the CIS Amazon EKS Benchmark.

Securing a Bottlerocket deployment

Bottlerocket is a special-purpose Linux distribution for hosting containers. AWS designed it with security in mind, with only a required set of services, read-only root file system, and default security settings and enforcement by using SELinux policies.

Due to the unique nature of Bottlerocket compared to general-purpose Linux distributions such as Amazon Linux 2, there are some different sets of considerations

when deploying Bottlerocket as your container host operating system in a PCI DSS environment.

Host and image hardening

Bottlerocket has several protections in place to enhance the default security of the host. First, it has a read-only root filesystem. Filesystem content is protected with cryptographic digest verification by using dm-verity. By default, interaction with the host operating system is limited to capabilities exposed through the apiclient, and no interactive shell is included.

This helps deter a wide swath of exploits by preventing modifications to system files, if an exploit escapes its container isolation. Many exploits assume the availability of an interactive shell to run scripts, which would fail immediately on Bottlerocket. Should an exploit find a way to modify or replace a file on the root file system with a malicious version, this would cause the partition's cryptographic hash to change, and dm-verity would immediately cause the host to restart. This supports a customer's ability to meet PCI DSS requirement 11.5.2 for a change detection mechanism for system components.

In addition to these integrity protections, there are certain operations that some processes should be able to perform, but others should be denied. Bottlerocket uses SELinux in enforcing mode. These policy restrictions prevent most processes from changing the API settings directly, or modifying the container image and the file system layers of other containers.

The use of a read-only root file system means that there are some restrictions when using anti-virus or other host-protection agents, but it also means that security monitoring is most relevant for protecting the applications run in the EKS or ECS cluster that Bottlerocket is a part of. We recommend the use of tools such as KubeArmor to protect these workloads.

Many of these products are designed to be installed directly on the host operating system using a package manager. Because Bottlerocket itself is read-only, there is no package manager available to install additional host level components. Third-party host monitoring and reporting must be able to use a containerized agent to accomplish its goals. You should perform your own targeted risk analysis to determine whether you are confident that Bottlerocket is "not commonly affected by malicious software" due to the out-of-the-box security settings in place and covered in this whitepaper.

Bottlerocket uses a full image update method to perform updates. When an update is available, the newer Bottlerocket release image is downloaded and written to a secondary partition. Upgrading to this new release is done by switching the target boot partition and rebooting the host. When deploying as part of an EKS cluster, use the Bottlerocket Update Operator. Like the ECS Updater, the Bottlerocket Update Operator will periodically check for updates and safely migrate workloads to apply upgrades. This can help you meet PCI DSS requirement 6.3.3 for installing security patches and updates on system components.

You can also use Amazon EKS managed node groups or Karpenter to manage your Bottlerocket nodes. These provide mechanisms to add and remove nodes, providing the ability to create fresh Bottlerocket instances to join to your cluster before removing the older versions. You can use either the in-place upgrade (Bottlerocket Update Operator) or replacement (managed node groups) to keep your nodes up to date.

In summary, consider the following points for host and image hardening:

- Eliminate hardening concerns faced with general Linux distributions because they are not applicable when you use Bottlerocket as the host operating system to run your container workloads.
- Use workload-level monitoring tools to protect application level data and processes.
- Use the Bottlerocket Update Operator or managed node groups to keep Bottlerocket cluster nodes updated.
- Use best practices for securing EKS deployments and PCI DSS requirement 2.

Data protection

When troubleshooting issues with the operating system, use the Bottlerocket admin container. This container provides an SSH server that enables public key access and interactive console access.

Note that this SSH connection terminates in the admin container and not on the base Bottlerocket host itself. But there are tools and methods from this privileged container to access and modify the Bottlerocket settings. An administrator can access files, run system commands, and perform troubleshooting at the host level.

The admin container is disabled by default and you should only enable it when needed to perform these troubleshooting tasks.

Bottlerocket enables SELinux in enforcing mode by default. This, along with a set of default policies, limits access to certain file system paths. Containers that run on Bottlerocket are given the `container_t` label be default. The security policy specified in a Kubernetes pod definition allows escalating privilege levels and can give a container process the increasing privileges of the `control_t` or `system_t` labels.

For guidance on how to define a Pod Security Policy that will limit the SELinux and pod privilege levels when using Kubernetes 1.24 and earlier, see the Kubernetes Pod Security Policy. For Kubernetes 1.25 and later, you will need to define a PodSecurity Admission Controller and Admission Webhook to enforce similar restrictions. Pod privilege escalation has valid uses, but if those do not apply to your environment, you should restrict them to avoid granting more permissions than required.

In summary, consider the following points for data protection:

- Limit the use of the Bottlerocket admin container to only enable it when needed, and as needed, and use a Systems Manager Session Manager session rather than SSH to access the console.
- Use pod security settings to limit access from container processes to allow default SELinux policies to enforce protections.
- Use best practices for data protection for Amazon EKS deployments.

Tracking and monitoring access

To maintain your PCI DSS compliance, you must be able to track, monitor, and alert on activities. Bottlerocket logs activities to the system logs by using `journald`. The way that administrators access the logs might differ from the way they access a general-purpose Linux distribution because Bottlerocket doesn't have an interactive console or the ability to install specialized log management packages directly on the operating system.

Bottlerocket nodes in an EKS cluster do not have an out-of-the-box log shipping option, but many logging and event agents can be containerized and run in a privileged container that would have access to host event logs. Fluent Bit is a popular solution that has many plugins for ingesting and sending logging and metrics.

You can also use custom containerized logging solutions with Amazon EKS.

Choose the best logging option that integrates well with your existing or preferred log auditing workflows. The important requirement is to be able to log and monitor the

events for both the host operating system and your container workloads running in the cluster to identify unusual or unexpected events. You must have these audit logs to satisfy PCI DSS requirement 10.2.

In summary, consider the following points for monitoring and logging:

- Use Fluent Bit or a customized event agent to monitor the host operating system and workload events.
- Use best practices for monitoring Amazon EKS deployments.

Conclusion

AWS provides multiple services to support your containerized workloads, and you can configure the services to best meet your data processing needs. Because of this flexibility, you must maintain an awareness of compliance requirements throughout the lifecycle of your container deployments as outlined in the shared responsibility model. Methods of security mitigation outlined in this whitepaper can help you to address PCI DSS compliance requirements for your containerized workloads.

Contributors

The following individuals and organizations contributed to this document:

- Ben Cressey, Principal Engineer, Bottlerocket
- Puneet Guglani, Senior Assurance Consultant, AWS Security Assurance Services
- Sean McGinnis, Senior Software Development Engineer, Bottlerocket
- Jeff Montgomery, Senior Assurance Consultant, AWS Security Assurance Services
- Sai Charan Teja Gopaluni, Senior Specialist Solutions Architect, Containers

Document revisions

Date	Description
March 31, 2023	First publication
November 14, 2023	Added the section “Securing Bottlerocket Deployment”