AWS Whitepaper

Architecting for HIPAA Security and Compliance on Amazon EKS

Architecting for HIPAA Security and Compliance on Amazon EKS: AWS Whitepaper

Table of Contents

Architecting for HIPAA Security and Compliance on Amazon EKS

Publication date: **March 27, 2022** (*Document history*)

Abstract

This whitepaper extends the technical and configuration-related information for Amazon EKS provided in the Architecting for HIPAA Security and Compliance on Amazon Web Services whitepaper, and outlines how customers may use AWS services to run regulated containerized workloads in accordance with their U.S. Health Insurance Portability and Accountability Act (HIPAA) requirements.

This whitepaper focuses on the considerations pertaining to the HIPAA Privacy and Security Rules for protecting Protected Health Information (PHI); technical and configuration information regarding encrypting data in transit and at-rest; and, how Amazon Elastic Kubernetes Service (Amazon EKS) features can be used to run Kubernetes applications containing Protected Health Information (PHI). AWS does not provide legal or compliance advice. We recommend that customers consult their legal counsel if they have legal questions regarding HIPAA compliance. Customers are responsible for making their own independent assessment of the information in this paper and any use of AWS products or services, including whether the information or the AWS services meet their regulatory, compliance, or operational requirements.

Introduction

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) applies to “Covered Entities” and “Business Associates.” HIPAA was expanded in 2009 by the Health Information Technology for Economic and Clinical Health (HITECH) Act.

HIPAA and HITECH establish a set of federal standards intended to protect the security and privacy of PHI. HIPAA and HITECH impose requirements related to the use and disclosure of protected health information (PHI), appropriate safeguards to protect PHI, individual rights, and administrative responsibilities. For more information on HIPAA and HITECH, see Health Information Privacy Home.

Covered Entities and their Business Associates can use the secure, scalable, low-cost IT components provided by Amazon Web Services (AWS) to architect applications in alignment with HIPAA and

HITECH compliance requirements. AWS offers commercial-off-the-shelf infrastructures with industry-recognized certifications and audits such as ISO 27001, FedRAMP, and the Service Organization Control Reports (SOC1, SOC2, and SOC3). AWS services and data centers have multiple layers of operational and physical security to help ensure the integrity and safety of customer data. With no minimum fees, no term-based contracts required, and pay-as-you-use pricing, AWS is a reliable and effective solution for growing healthcare industry applications.

AWS enables covered entities and their business associates subject to HIPAA to securely process, store, and transmit PHI. Additionally, as of July 2013, AWS offers a standardized Business Associate Addendum (BAA) for such customers. Customers who execute an AWS BAA may use any AWS service in an account designated as a HIPAA Account, but they may only process, store and transmit PHI using the HIPAA-eligible services defined in the AWS BAA. For a complete list of these services, see the HIPAA Eligible Services Reference page.

AWS maintains a standards-based risk management program to ensure that the HIPAA-eligible services specifically support HIPAA administrative, technical, and physical safeguards. Using these services to store, process, and transmit PHI helps our customers and AWS to address the HIPAA requirements applicable to the AWS utility-based operating model.

At time of publication, AWS standard BAA requires customers to encrypt PHI stored in, or transmitted using, HIPAA-eligible services in accordance with guidance from the Secretary of Health and Human Services (HHS). Refer to this site as it could be updated, and be made available on a successor (or related) site designated by HHS.

A service listed as HIPAA-eligible does not mean the use of the service by our customers automatically confirms HIPAA-related safeguards are in place. It more appropriately indicates the service has the ability to be configured to meet HIPAA-related safeguards. Where parameters are accessible and configurable by customers, it is the customer's responsibility to ensure they are configured to meet compliance requirements.

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

The benefits of transitioning workloads to container services include solutions independence, deployment speed, and resource efficiency. It's important, as with any cloud workloads, to

understand how to architect for security in containers. The transient and dynamic nature of container environments may make it 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 management layer. As with other application deployments, we recommend that you continue to operate within best practices, including adherence to Open Web Application Security Project's (OWASP) recommendations and best practices.

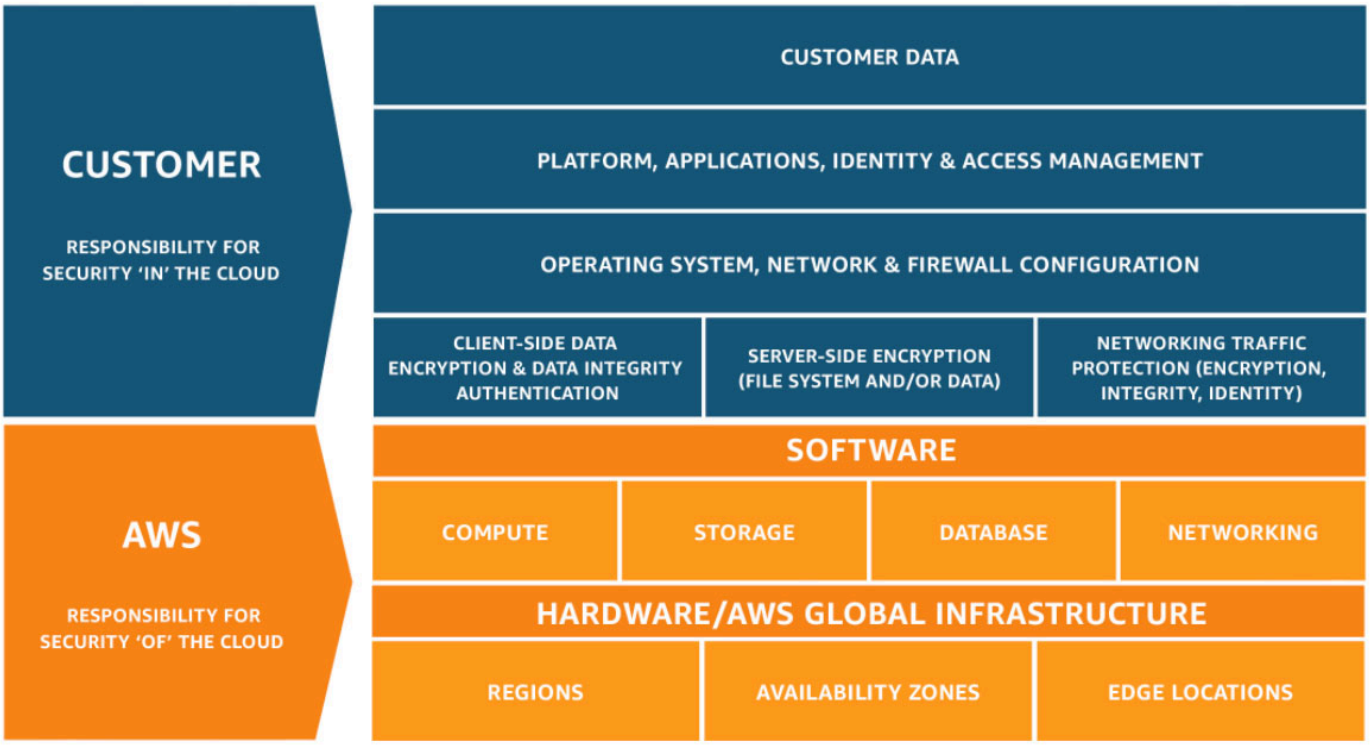
Container functions are typically architected to perform primary tasks, which in turn creates a distributed environment. The services implemented by containers become more network interdependent and necessitate scheduling, scaling, and resource management. Unlike virtual machines, containers share the operating system's kernel. This setup can provide a common point of attack that can be leveraged to access all containers for a given host. When running multiple containers on a single operating system, all of the containers may share a common network interface. In this whitepaper, we will discuss the various architectures that you can build around AWS services to mitigate this security risk.

AWS Shared Responsibility Model

Security and compliance are shared responsibilities between AWS and the customer. Depending on the services deployed, this shared model can help relieve the customer's operational burden. This is 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 customer assumes responsibility and management of the guest operating system (including updates and security patches) and other associated application software, in addition to the configuration of the AWS-provided security group firewall.

We recommend that customers carefully consider the services they choose because their responsibilities vary depending on the services used, the integration of those services into their IT environment, and applicable laws and regulations. It is possible for customers to enhance their security and/or meet their more stringent compliance requirements by leveraging technology such as host-based firewalls, host-based intrusion detection and prevention, encryption, and key management.

The nature of this shared responsibility also provides the flexibility and customer control that permits customers to deploy solutions that meet industry-specific certification requirements.

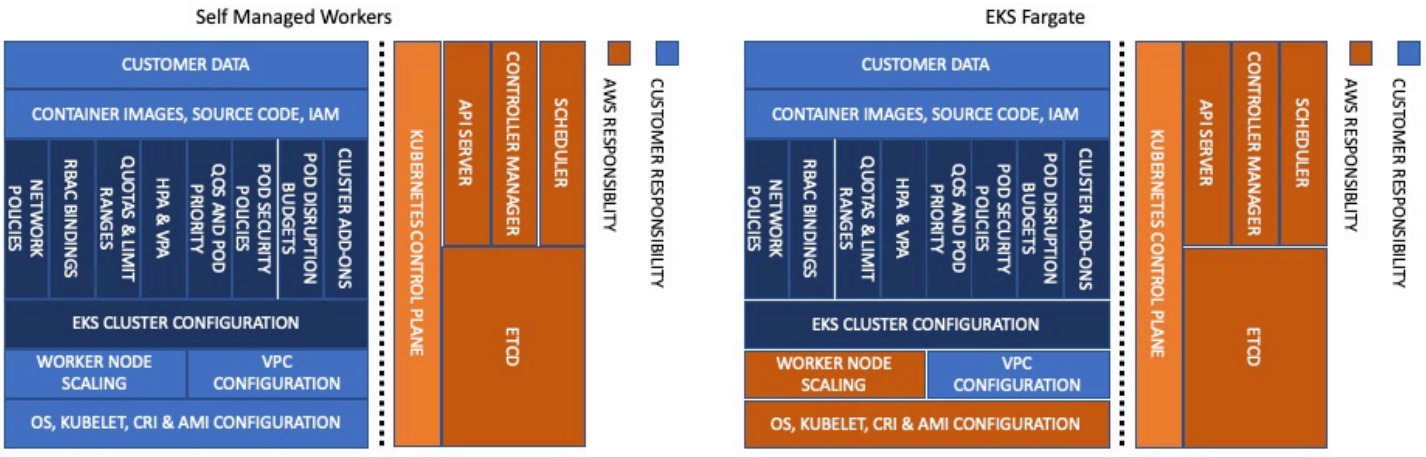


Shared Responsibility Model

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 our highest priority. AWS customers benefit from a data center and network architecture that are built to meet the needs of the most security-sensitive organizations. This infrastructure consists of the hardware, software, networking, and facilities that run AWS Cloud services. Refer to AWS Artifact, available in the console, which is a self-service audit artifact retrieval portal that provides our customers with on-demand access to AWS compliance documentation and AWS agreements.

Customers are responsible for the security and data safeguards **in** the Cloud, which consists of customer configured systems and services provisioned on AWS. For HIPAA-related safeguards, the customer has responsibility for all system components, which includes AWS resources included in, or connected to their containerized workload which process, store, or transmit PHI. For abstracted services, such as Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB, the responsibility includes customer-configurable controls such as access controls, log settings, and encryption settings.

The division of responsibility will depend on the AWS service and implementation that you select Amazon EKS is a good example that allows you to choose a serverless deployment of containers with AWS Fargate, or run containers on Amazon EC2 infrastructure which is accessible by the customer. With AWS Fargate, you are abstracted from the underlying host and are not responsible for updating or patching the host system. An Amazon EKS deployment on Amazon EC2 requires you to take on a greater role of responsibility, such as controlling access and applying security patches.



Shared Responsibility Model using Amazon EKS and self-managed workflows

If you can control a service parameter, you are responsible for ensuring it is configured to meet HIPAA requirements.

You can use AWS Config to monitor configuration data for container-based resources in your AWS account, such as monitoring configuration changes to EKS cluster settings and tracking compliance for cluster configurations. AWS Config provides a detailed view of the configuration of AWS resources in your AWS account, including how resources were configured, how they relate to one another, and how the configurations and relationships change over time.

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 all container configuration parameters to ensure compliance requirements are addressed throughout all phases of a container lifecycle, as will be illustrated in the following sections.

Encryption and protection of PHI in AWS

It is a best practice to encrypt PHI both in transmission (“in transit”) and in storage (“at rest”). For additional guidance, see the Guidance to Render Unsecured Protected Health Information Unusable,Unreadable, or Indecipherable to Unauthorized Individuals (“Guidance”) from the Secretary of Health and Human Services (HHS), which may be updated or may be made available on a successor (or related) site designated by HHS.

AWS offers a comprehensive set of features and services to make key management and encryption of PHI easier to manage and simpler to audit, including the AWS Key Management Service (AWS KMS). With AWS KMS, customers have flexibility in how to encrypt their PHI.

When determining how to implement encryption, you can evaluate and take advantage of the encryption features native to the HIPAA-eligible services, or you can encrypt through other means consistent with the guidance from HHS.

For additional guidance, refer to the Architecting for HIPAA Security and Compliance on Amazon Web Services whitepaper, which provides high-level details about using available encryption features in each of the HIPAA-eligible services and other patterns for encrypting PHI, and how AWS KMS can be used to encrypt the keys used for encryption of PHI on AWS.

Securing an Amazon EKS Deployment

There are generally three types of safeguards: administrative, physical, and technical. AWS provides many physical safeguards per the Shared Responsibility Model, which will not be addressed in the following sections. The following information is provided for key administrative and technical safeguards to consider when architecting a container-based environment. The following sections provide an overview of the best practices. This guidance is not exhaustive and each customer environment is unique.

Identity and Access Management

Identity and Access Management refers to who has access to electronically protected health information (e-PHI) and ensuring that only authorized personnel and components are allowed access to e-PHI data.

User access controls are focused on restricting access to authorized personnel and ensuring appropriate access controls are in place. Access to resources should embrace a least privilege model where access is on a need-to-know basis. This functionality is also known as role-based access.

User access to containers and the underlying host should be authenticated and limited to respective functions. Container images should be run with non-privileged users. For example, container build files that do not contain defined user credentials will run as root by default. This setup means that a compromised container service may extend root privileges to a malicious actor who may use the elevated access to further exploit the underlying host. Unlike AWS Fargate, where no host access is available, Amazon EKS with EC2 launch type provides the option to enable secure shell (SSH) access for underlying system management. Consider disabling the use of the secure shell (SSH) and instead leverage AWS Systems Manager Run Command. With Run Command, there are no SSH keys to manage, and all invoked operations are auditable within AWS CloudTrail. Alternatively, consider preventing any user access to the containers that interact with e-PHI in the production environment and rely on automation for orchestrating the system.

In an effort to 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 provides secure storage and transmission of containers to Amazon EKS. The simplified workflow and integration of

Amazon ECR with AWS services also reduces the need for excessively providing credentialed access to container hosts.

With Amazon EKS and its IAM authenticator, users sign into the cluster with an IAM identity – either an IAM user or IAM role. Kubernetes then decides what actions the user can perform via its role-based access control (RBAC). Consider configuring AWS IAM roles to set up authorization at the cluster and infrastructure level. With RBAC, you can set up authorization to the resource level (such as particular pod) or service level (such as pod).

Consider employing least privileged access when creating Roles/RoleBindings for namespace level resources and ClusterRole/ClusterRoleBindings for cluster-level resources. When you create an Amazon 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, and 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.

To summarize, consider the following recommendations for user access:

- Employ least privileged access to AWS resources when creating RoleBindings and ClusterRoleBindings.
- Use IAM Roles when multiple users need identical access to the cluster and IAM Roles for Service Accounts (ISRA) where possible. When using IRSA, it's best to make the role trust policy as explicit as possible by including the service account name.
- Make the Amazon EKS Cluster endpoint private.
- Create the cluster with a dedicated IAM role which should be regularly audited.
- Regularly audit access to the cluster.
- Run the application as a non- root user user.
- Consider treating containers as an immutable infrastructure with no SSH access, and rely on automation to manage your infrastructure.
- Unless there are mechanisms in place to sanitize Application logs to be free of e-PHI, secure and control access to it.
- Prevent access to storage containing e-PHI to Cluster, Node, and Pod Execution roles. If access to e-PHI is granted only on the level of IAM Role for Kubernetes Service Account, it should be easier

to control and audit what components have access to sensitive data, following the least privilege principle, and provide credential isolation.

Important: Even if you assign an IAM role to a Kubernetes service account, the pod still also has the permissions assigned to the Amazon EKS node IAM role unless you block pod access to the IMDS. For more information, see Restricting access to the IMDS and Amazon EC2 instance profile credentials.

Pod security

Pods have a variety of settings that can strengthen or weaken your overall security posture and will also help you to address Access Management concerns, such as Access Control Policy and Privilege Management. We recommend taking steps to prevent a process that's running in a container from escaping the isolation boundaries of Docker and gaining access to the underlying host. You can reject pods with containers configured to run as privileged by using OPA/Gatekeeper and Kyverno policies, which prevents pods from running as privileged or escalating privileges.

With Fargate launch type, you cannot run a privileged container or configure your pod to use `hostNetwork` or `hostPort`. All containers run as root by default. This could be problematic if a malicious actor is able to exploit a vulnerability in the application and get shell access to the running container.

Recommendations for mitigating this risk include:

- Removing the shell from the container image.
- Adding the USER directive to your Dockerfile.
- Run the containers in the pod as a non-root user user.
- Restricting the use of hostPath, or if hostPath is necessary.
- Restricting which prefixes can be used and configure the volume as read-only.
- Setting requests and limits for each container to avoid resource contention and DoS attacks. For example, you can force the use of requests and limits by setting a resource quota on a namespace or by creating a limit range.

You can prevent a container from using privileged escalation by implementing a pod security policy that sets **allow Privileged Escalation to false** or by setting `securityContext.allowPrivilegedEscalation` in the podSpec.

For pods that do not need to access the Kubernetes API, you can disable the automatic mounting of a ServiceAccount token on a pod spec, or for all pods that use a particular ServiceAccount. For pods that do not need to look up or call-in cluster services, you can reduce the amount of information given to a pod. You can set the Pod's DNS policy to not use CoreDNS, and not expose services in the pod's namespace as environment variables. For more information on service links, see the Kubernetes docs on environment variables.

Configuring your images with a read-only root file system may prevent a malicious actor from overwriting a binary on the file system that your application uses. If your application has to write to the file system, consider writing to a temporary directory or attach and mount a volume.

Pod Security Standards (PSSs) are part of the proposal to replace Pod Security Policies (PSPs). They are an attempt to provide a set of standards for pod security that is independent of the enforcement mechanism. The following are useful tools and resources:

- The OPA Gatekeeper policy library
- Common OPA and Kyverno policies for EKS
- Policy based countermeasures for Kubernetes: part 1
- Policy based countermeasures for Kubernetes: part 2

Network segmentation and hardening

For network segmentation and hardening, such as network Routing Control, network Connection Control, and more, you should consider installing and maintaining a firewall to protect PHI data and should ensure that systems be protected from unauthorized access. Firewall, in this context, means inbound and outbound access is restricted to only approved ports and services.

An individual AWS account provides the highest level of segmentation boundary that can be achieved on AWS. By design, all resources provisioned within an AWS account are logically isolated from resources provisioned in other AWS accounts, even within your own AWS Organizations. Consider using an isolated account for HIPAA workloads to establish access segmentation when designing your PHI data protection to run on the AWS Cloud.

Amazon Virtual Private Cloud (Amazon VPC) and subnets provide further logical isolation of HIPAA-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 Amazon EKS provided AWS CloudFormation template. Both eksctl and the AWS CloudFormation template create the VPC and subnets with the required configuration. For more information, see Creating a VPC for your Amazon EKS cluster.

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.

Network policies can be scoped to namespaces, pods, service accounts, or globally. When policies are scoped to a service account, it associates a set of inbound/outbound rules with that service account. With the proper Kubernetes rule-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, there is no enforcement of network policies. To use network policies, configure the Amazon VPC CNI or an add-on such as Calico.

Within 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 should configure outbound connection rules to conform with HIPAA.

Amazon EKS uses Amazon VPC security groups to control the traffic between the Kubernetes control plane and the cluster's worker nodes. Security groups are also used to control the traffic between worker nodes, external IP addresses, and other VPC resources. It is strongly recommended that you use a dedicated security group for each control plane (one for each cluster). The minimum and suggested rules for the control plane and node group security groups can be found in Amazon EKS security group considerations.

To control communication between services that run within the cluster and services that run outside of the cluster, consider security groups for pods, which integrate Amazon EC2 security groups with Kubernetes pods. You can use Amazon 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 Amazon EC2 instance types. For a complete list of supported instances, see Amazon EC2 supported instances and branch network interfaces. Your nodes should be one of the supported instance types. Before deploying security groups for pods, consider the limits and conditions discussed in the *Amazon EKS User Guide*.

AWS 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 using separate Amazon EKS clusters.

You can only use security groups for pods with pods running on AWS Fargate if your cluster is 1.18 with version eks.7 or later, 1.19 with version eks.5 or later, or 1.20 or later. Before deploying security groups for pods, consider the limits and conditions discussed in the *Amazon EKS User Guide*.

Containerized workloads should be grouped based on data sensitivity levels to more easily 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 interacting with one namespace will not see content in another namespace. However, namespaces within the same cluster don't restrict communication between namespaces. You would need network policies for granular control and to restrict such communications between namespaces.

AWS VPC Flow Logs captures metadata about the traffic flowing through a VPC, such as source and destination IP address and port along with accepted/dropped packets. This information could be analyzed to look for suspicious or unusual activity between resources within the VPC, including Pods. However, since the IP addresses of pods frequently change as they are replaced, Flow Logs may not be sufficient on its own. You should reach out to your account team for more information on 3rd party tools which may make it easier to decipher the traffic flows between pods.

It is a best practice to encrypt PHI both at-rest and in-transit. There are multiple methods (or a combination of them) to encrypt data while it is in transit. Traffic exchanged between the following Nitro instance types - C5n, G4, I3en, M5dn, M5n, P3dn, R5dn, and R5n - is automatically encrypted by default. When there's an intermediate hop, like a transit gateway or a load balancer, the traffic is not encrypted. Encryption in transit can also be implemented with a service mesh like App Mesh. AppMesh supports mTLS with X.509 certificates or Envoy's Secret Discovery Service(SDS).

Ingress controllers are a way for you to intelligently route HTTP/S traffic that emanates from outside the cluster to services running inside the cluster. Oftentimes, these Ingresses are fronted by a layer 4 load balancer, like the Classic Load Balancer or the Network Load Balancer (NLB). Encrypted traffic can be terminated at different places within the network, such as at the load balancer, at the ingress resource, or the Pod.

To summarize, consider the following when working to isolate containerized application communications:

- Isolate pods on separate nodes based on sensitivity of services and isolate HIPAA- related workloads in a separate cluster with a dedicated Security group.
- Use AWS security groups to limit communication between nodes and control plane and external communications.
- Implement micro-segmentation with Kubernetes network policies and consider usage of service mesh, Networking and Cryptography library (NaCl) encryption and Container Network Interfaces (CNIs) to limit and secure communications.
- Implement a network segmentation and tenant isolation network policy. Network policies are similar to AWS 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 using pod selectors and labels. For more information, see Configuring your cluster for Kubernetes network policies.
- Encrypt data in transit using Container Network Interfaces, Service mesh etc. and use Ingress Controllers and Load Balancers to terminate your SSL connection and intelligently route HTTP/S traffic as dictated by your organization's network security policy and regulatory compliance.

Host and image hardening

Consider using the container image as your first line of defense against an attack. An insecure, poorly constructed image can allow a malicious actor to escape the bounds of the container and gain access to the host. Once on the host, a malicious actor can gain access to sensitive information or move laterally within the cluster or with your AWS account. AWS offers container services to help secure underlying infrastructure, like Amazon EKS, which is run on container optimized Amazon Machine Images (AMI). These operating systems only contain additional libraries that are essential for container deployments, and as a result, help to minimize attack vectors.

Customers are still responsible for maintaining compliance of all configurations and functions at the operating system, network, and application layers. Operating systems should be routinely patched through the use of AWS Systems Manager whereas non-essential services and libraries should be disabled or removed. Configuration standards should be established that are consistent with industry-accepted system hardening guidelines, such as the Center for Internet Security (CIS) Benchmarks for EC2 instance types. Additional AWS secure configuration standards support is available on the AWS Security Learning website.

Consider using a special purpose operating system (OS) like Bottlerocket that includes a reduced attack surface, a disk image that is verified on boot, and enforced permission boundaries using SELinux.

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

When managing containers, they should be immutable and not patched in-place. You should consider creating trusted base container images which 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 (CVEs) database and can identify common software vulnerabilities.

Amazon Inspector can be leveraged as an automated security assessment service that helps improve the security and compliance of applications deployed on AWS. Amazon Inspector automatically assesses applications for exposure, vulnerabilities, and deviations from best practices.

Customers are responsible for ensuring that their Amazon EC2 instances run appropriate anti-virus and file integrity monitoring software when choosing the EC2 launch type for Amazon EKS. Many container vendors provide solutions optimized for container usage to address these requirements.

Alternatively, consider using the AWS Fargate launch type which provides on-demand, right-sized compute capacity for containers. With AWS Fargate, you no longer have to provision, configure, or scale groups of virtual machines to run containers. This 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 hardening, patching and monitoring the host system, and the worker node is taken care by AWS. Additional considerations for choosing Fargate can be found in the *Amazon EKS User Guide*.

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

To summarize, consider the following points for host and image hardening:

- Use an OS optimized for running containers.
- Minimize access to worker nodes and deploy the worker nodes in private subnet.
- Run Amazon Inspector to assess hosts for exposure, vulnerabilities, and deviations.
- Use minimal container images and scan images for vulnerabilities regularly.

Data protection and Secrets Management

It is essential to safeguard your data, and AWS provides a number of HIPAA eligible services and features to assist with safeguarding Protected Health Information (PHI) while at rest and in transit.

Regulated workloads generally should safeguard data at rest. Storage of data should be 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.

It is advisable to secure sensitive data and environment variables, such as database connection strings, that are contained within container build files. Many AWS services integrate with the AWS Key Management Service (AWS KMS), a HIPAA eligible 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 can be used to secure sensitive data within container build files. AWS 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 needs. Data stored within AWS Systems Manager Parameter Store can be encrypted using AWS KMS.

Similar to AWS Systems Manager Parameter Store, data secured within AWS Secrets Manager also leverages the AWS KMS. AWS Secrets Manager provides additional capabilities that include random password generation and automatic password rotation. AWS KMS is a HIPAA eligible service that is integrated with many AWS services. Users can create and manage cryptographic key material as well as control who can access and use the encryption keys.

For data in transit, it is a best practice that Protected Health 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 Transport Layer Security (TLS). Policies can be applied to the services to enforce support of only TLS 1.2 or greater.

Amazon API Gateway and Application Load Balancer also support use of the integrated AWS WAF (Web Application Firewall) to secure communications at the application-layer. AWS WAF helps protect applications and APIs against common web exploits like those identified within the OWASP Top 10.

Traffic exchanged between the Nitro instance types C5n, G4, I3en, M5dn, M5n, P3dn, R5dn, and R5n, is automatically encrypted by default except when there's an intermediate hop, such as an AWS Transit Gateway or a load balancer. In these instances, the traffic is not encrypted.

Encryption in transit for inter-pod communication can also be implemented with a service mesh like AWS App Mesh with support for mTLS.

Inbound traffic controllers are a way for you to intelligently route HTTP/S 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 the Network Load Balancer. An inbound traffic controller can be configured to terminate SSL/TLS connections.

Kubernetes secrets are used to store sensitive information, such as user certificates, passwords, or API keys. They are persisted in etcd as base64 encoded strings. On EKS, the EBS volumes for etcd nodes are encrypted with EBS encryption. Consider using AWS KMS for envelope encryption of Kubernetes secrets. With the KMS plugin for Kubernetes, all Kubernetes secrets are stored in etcd in ciphertext instead of plain text and can only be decrypted by the Kubernetes API server. For more information, see using EKS encryption provider support for defense in depth.

On EKS, you can turn on audit logging and create a CloudWatch metrics filter and alarm to alert you when a secret is used. Kubernetes doesn't automatically rotate secrets. If you need to rotate secrets, consider using an external secret store, such as AWS Secrets Manager. AWS offers two services to manage secrets and parameters conveniently in your code. AWS Secrets Manager allows you to easily rotate, manage, and retrieve database credentials, API keys, certificates, and other secrets throughout their lifecycle. AWS Systems Manager Parameter Store provides hierarchical storage for configuration data. The AWS provider for the Secrets Store CSI Driver allows you to make secrets stored in Secrets Manager and parameters stored in Parameter Store appear as files mounted in Kubernetes pods.

Consider using volume mounts instead of environment variables. The values of environment variables can unintentionally appear in logs. Secrets mounted as volumes are instantiated as tmpfs volumes (a RAM backed file system) that are automatically removed from the node when the pod is deleted.

To summarize, consider the following points for data protection and secret Management:

- Make the Amazon EKS Cluster endpoint private.
- Leverage AWS KMS for service-managed encryption keys and avail AWS managed CMKs or rotate your AWS KMS keys periodically.
- Enable support for strong encryption in transit.
- Use envelope encryption of Kubernetes secrets in EKS to encrypt your content with a unique data encryption key (DEK). The DEK is then encrypted using a key encryption key (KEK) from AWS KMS, which can be automatically rotated on a recurring schedule.
- Use volume mounts instead of environment variables.
- Consider using an external secrets provider like AWS Secrets Manager for features such as fine-grained access controls, strong encryption, and automatic rotation of secrets that are not available with Kubernetes Secrets.

Event logging monitoring, auditing and logging

It is a best practice to use event logging mechanisms to track, monitor, and alert on potentially anomalous activities.

In order to achieve this, consider the following:

- Leverage AWS event log services to establish event log monitoring at the network, host, and container level.
- Enable VPC Flow Logs to capture network traffic that details packet information, such as the protocol, port, and source and destination address information.
- Monitor container hosts to ensure health, efficiency, and availability by ensuring 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 dashboard to monitor and alert on all captured event log activity.
- Store the captured event data securely within encrypted Amazon S3 buckets to help you meet your retention needs.

Amazon EKS with Fargate supports a built-in log router, which means there are no sidecar containers to install or maintain. The log router allows you to use the breadth of services at AWS for log analytics and storage. You can stream logs from Fargate directly to Amazon CloudWatch, Amazon OpenSearch Service, and Amazon Data Firehose destinations such as Amazon S3,

Amazon Kinesis Data Streams, and partner tools. Fargate uses a version of Fluent Bit, an upstream compliant distribution of Fluent Bit managed by AWS. For more information, see AWS for Fluent Bit on GitHub.

Finally, we recommend you maintain a holistic view of the environment through the use of AWS tools. Amazon GuardDuty provides threat detection through anomaly detection, machine learning, and threat intelligence of events across AWS data sources, including AWS CloudTrail and VPC Flow Logs. Amazon Athena and Amazon CloudWatch Logs Insights can also be used to query and analyze audit trail logs saved to Amazon S3 from VPC Flow Logs, AWS CloudTrail, and Amazon CloudWatch.

To summarize, 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 Amazon CloudWatch Log Insights.
- Audit your AWS CloudTrail logs.

Incident response and forensics

Your ability to react quickly to an incident can help minimize damage caused from a breach. You should consider having a reliable alerting system that can warn you of suspicious behavior as the first step in an incident response plan. When an incident does arise, you may evaluate whether to destroy and replace the effected container, or isolate and inspect the container. If you choose to isolate the container as part of a forensic investigation and root cause analysis, consider the following:

- Identifying the offending Pod and worker node.
- Isolating the Pod by creating a Network Policy that denies all ingress and egress traffic to the pod.
- Revoking temporary security credentials assigned to the pod or worker node if necessary.
- Cordoning the worker node to informing the kubernetes scheduler to avoid scheduling pods onto the affected node.
- Enabling termination protection on impacted worker node so that the malicious actor cannot erase their misdeeds by terminating an affected node.

- Labeling the offending Pod/Node with a label indicating that it is part of an active investigation to warn cluster administrators not to tamper with the affected Pods/Nodes until the investigation is complete.

For more information, see AWS Security Incident Response. It is helpful to practice security game days to probe different systems for vulnerabilities and defending against them. Kubesploit is a penetration testing framework from CyberArk that you can use to conduct game days. Unlike other tools which scan your cluster for vulnerabilities, kubesploit simulates a real-world attack.

A few other tools that can help are as follows:

Gremlin - a chaos engineering toolkit that you can use to simulate attacks against your applications and infrastructure.

kube-forensics - a Kubernetes controller that triggers a job that collects the state of a running pod and dumps it in an S3 bucket.

Multi-tenancy

Kubernetes is a *single tenant orchestrator*, such as a single instance of the control plane is shared among all the tenants within a cluster. However, there are various Kubernetes objects that you can use to create the semblance of multi-tenancy but, the cluster is the only construct that provides a strong security boundary. If a malicious actor gains access to a host within the cluster, they could retrieve *all* Secrets, ConfigMaps, and Volumes, mounted on that host and could also impersonate the Kubelet, which would allow them to manipulate the attributes of the node and/or move laterally within the cluster.

To mitigate the risk and implement tenant isolation, you can use soft multi-tenancy or hard multi-tenancy.

With soft multi-tenancy, you can use native Kubernetes constructs, such as namespaces, roles and role bindings, and network policies, to create logical separation between tenants. RBAC, for example, can prevent tenants from accessing or manipulate each other's resources. Quotas and limit ranges control the amount of cluster resources each tenant can consume while network policies can help prevent applications deployed into different namespaces from communicating with each other.

These controls cannot prevent pods from different tenants from sharing a node. If stronger isolation is required, you can use a node selector, anti-affinity rules, and/or taints and tolerations to

force pods from different tenants to be scheduled onto separate nodes, which are often referred to as *sole tenant nodes*.

Kiosk is an open-source project that can aid in the implementation of soft multi-tenancy. It is implemented as a series of CRDs and controllers. Alternately, you can use commercial solutions like Loft and DevSpace to add additional capabilities for soft multi-tenancy.

Another possible concern in a multi-tenant environment is preventing a malicious actor from gaining access to the underlying host. Consider the controls discussed in Network Segmentation and Host and Image hardening sections in this whitepaper to mitigate this risk.

Hard multi-tenancy can be implemented by provisioning separate clusters for each tenant. While this provides very strong isolation between tenants, this approach may increase costs.

Network intrusion detection

Use of intrusion-detection and/or intrusion-prevention techniques may be used to detect and/or prevent intrusions into the network. It is prudent to monitor of all traffic at the perimeter and critical points of the CDE. With most on-premises environments, this can typically be achieved by using Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) appliances. A similar approach can be used within AWS.

When considering containerized environments, inspection of network traffic can be done at the network layer outside of the container host and within the container management software's virtual container network.

There are several options that can be considered 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 identify threats. It uses machine learning, anomaly detection, and threat intelligence to help identify illicit network activity.

When considering a traditional IDS/IPS solution, Amazon VPC Traffic Mirroring can be configured to route a copy of all network communications to a virtual appliance running on one or more Amazon EC2 instances.

Another common solution is to use a transit network architecture that uses IP routing to ensure that all network traffic crosses a single network. This architecture allows you to use a virtual IDS/IPS device from the AWS Marketplace to inspect all traffic transiting between networks. It is

possible to also use a VPC Gateway to route all traffic to on-premises IDS/IPS infrastructure. Lastly, host-based IDS or IPS solutions can also be used to inspect traffic as it is delivered to an Amazon 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 that 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 will necessitate advanced planning of container deployments and traffic patterns.

Vulnerability scanning and penetration testing

It's a best practice for organizations to engage in vulnerability scanning and penetration testing to regularly test systems and processes to identify vulnerabilities and remediate such findings in a timely manner. It's also a best practice that vulnerability scanning be performed on a quarterly basis, as well as after any significant changes to the environment. Similarly, penetration testing should ideally be performed on an annual basis and after any significant environment changes. Penetration testing of AWS resources may be allowed at any time for certain permitted services. You can consult with your account team or the AWS support policy for additional information on penetration testing. For those service providers who are using network segmentation, testing the effectiveness of segmentation controls is recommended every six months or after any changes to the segmentation controls.

Depending on your environment, the testing may apply to on-premises, cloud resources, and containerized environments. When deploying Amazon EKS on Amazon EC2 instances, you should consider performing vulnerability scanning of the underlying host. You should also consider establishing a process for identifying security vulnerabilities, and assigning a risk ranking to newly discovered security vulnerabilities. Amazon Inspector is a security assessment tool that helps identify vulnerabilities and prioritizes findings by level of severity. Integration of Amazon 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 identifies 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 performing internal and external penetration testing, assessment activities should be done at both a network and application layer and should target the underlying host and containerized applications. Patch container hosts to address vulnerabilities and update container images to mitigate identified container vulnerabilities. You can 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. It can be run using kube-bench, a standard open-source tool for checking configuration using the CIS benchmark on Kubernetes clusters. For more information, see Introducing the CIS Amazon EKS Benchmark.

Conclusion

AWS provides multiple services to support customers' containerized workloads, and customers can configure the services to best meet their data processing needs. Because of this flexibility, organizations should maintain an awareness of all compliance requirements throughout the lifecycle of their container deployments as per the AWS Shared Responsibility Model. Methods of security mitigation outlined within this whitepaper can help you address HIPAA compliance requirements for containerized workloads.

Contributors

The following individuals and organizations contributed to this document:

- Arindam Chatterji, Sr. Solution Architect, US SMB
- Scott Schreckengaust, Sr. Solution Architect, Health Care and Life Sciences (HCLS)
- Abby Palia, Sr. ML Security & Privacy Consultant, AWS ProServ
- Cate Ciccolone, Security Consultant, AWS ProServ
- Megan Yelorda, Sr. Technical Program Manager, AWS ProServ
- Erik Pupo, Sr. Practice Manager-HIPAA, AWS ProServ
- Anton Gridin, Sr. Solution Architect, US SMB Greenfield

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Whitepaper first published.	March 27, 2022

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.

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.