IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

EDGE NETWORKING SYSTEMS LLC,	)	Case No.
Plaintiff,	)	**JURY TRIAL DEMANDED**
v.	)	
AMAZON.COM, INC., AMAZON.COM SERVICES LLC, and AMAZON WEB SERVICES, INC.,	)	
Defendants.	)	

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff Edge Networking Systems LLC ("Edge" or "Plaintiff") for its Complaint against Defendants Amazon.com, Inc., Amazon.com Services LLC, and Amazon Web Services, Inc. (collectively, "Amazon" or "Defendants") for patent infringement, alleges as follows:

THE PARTIES

1. Plaintiff Edge is a Texas limited liability company with its place of business at 7700 Windrose Avenue, Plano, Texas 75024.

2. Defendant Amazon.com, Inc. is a Delaware corporation, with its principal place of business at 410 Terry Avenue North, Seattle, Washington 98109. Amazon.com, Inc. may be served through its registered agent Corporation Service Company, 251 Little Falls Drive, Wilmington, Delaware 19808. "Amazon.com, Inc. sells its products through its internet distribution network . . . ." *Trackthings LLC v. Amazon.com, Inc.*, Case No. 6:21-cv-00720-ADA, Dkt. No. 59 at 1 n.1 (W.D. Tex. May 4, 2022) (Amazon.com, Inc. quoting *Nazomi Commc'ns, Inc. v. Nokia Corp.*, 2010 WL 11509140, at *1-2 (C.D. Cal. Oct. 12, 2010)). Amazon.com, Inc.

employs thousands of employees and has numerous offices and facilities. Amazon.com, Inc. operates this internet distribution network and sells its products within this Judicial District.

3. Defendant Amazon.com Services LLC is a Delaware limited liability company, with its principal place of business at 410 Terry Avenue North, Seattle, Washington 98109. Amazon.com Services LLC is a wholly-owned subsidiary of Amazon.com, Inc. Amazon.com Services LLC is registered to do business in the State of Texas and may be served with process via its registered agent in Texas, Corporation Service Company d/b/a CSC-Lawyers Incorporating Service Company at 211 7th Street, Suite 620, Austin, Texas 78701. Amazon.com Services LLC is "[t]he entity that owns and operates the Amazon.com website." Dkt. No. 15 at 3. Amazon.com Services LLC also leases and operates the Amazon FTW3/FTW4 and STX8 fulfillment centers located within this District. Dkt. No. 15 at 7.

4. Defendant Amazon Web Services, Inc. is a Delaware corporation, with its principal place of business at 410 Terry Avenue North, Seattle, Washington 98109. Amazon Web Services, Inc. is a wholly-owned subsidiary of Amazon.com, Inc. Amazon Web Services, Inc. may be served through its registered agent Corporation Service Company, 251 Little Falls Drive, Wilmington, Delaware 19808.

5. Amazon.com, Inc. is the parent company of Amazon.com Services LLC and Amazon Web Services, Inc. As defined above, Amazon.com, Inc., Amazon.com Services LLC, and Amazon Web Services, Inc. are collectively referred to herein as "Amazon."

6. Amazon conducts business operations within the Eastern District of Texas.

7. Amazon has a regular and established place of business in this District, including at least its Amazon Robotics Fulfillment Center FTW3/FTW4 located at 15201 Heritage Parkway,

Fort Worth, Texas 76177 and at its Sub-Same Day Fulfillment Center STX8 located at 3501 Research Drive, Richardson, exas75082.

JURISDICTION AND VENUE

8. This is an action for patent infringement arising under the patent laws of the United States, 35 U.S.C. §§ 1, *et seq*. This Court has jurisdiction over this action pursuant to 28 U.S.C. §§ 1331 and 1338(a).

9. Amazon is subject to this Court's general and specific jurisdiction pursuant to due process and/or the Texas Long Arm Statute due at least to Amazon's substantial business in the State of Texas and in this District, including through employing more than 89,000 full- and part-time employees at fifty (50) fulfillment and sortation centers, thirty-three (33) delivery stations, an Air Gateway, thirty-six (36) Whole Foods Market locations, three (3) Amazon Hub Locker+ locations, six (6) Prime Now fulfillment centers, an Amazon Pharmacy, a Prime Air drone delivery center, three (3) wind farms, thirteen (13) solar farms, seven (7) on-site solar locations, and an Amazon Original series filmed in the State of Texas: Panic.[1]

10. Venue is proper in this District under 28 U.S.C. §§ 1391(b)-(d) and 1400(b). Defendant Amazon is registered to do business in the State of Texas, has offices in the State of Texas, has transacted and continues to transact business in the Eastern District of Texas, and has committed and continues to commit acts of direct and indirect infringement in the Eastern District of Texas.

11. Amazon has regular and established places of business in this District and has committed and continues to commit acts of infringement in this District including, but not limited

[1] *See Investing in the U.S.*, AboutAmazon.com Website, available at: https://www.aboutamazon.com/investing-in-the-u-s

to: 15201 Heritage Parkway, Fort Worth, Texas 76177; 1101-1107 East 15th Street, Plano, Texas 75074; 1809 West Frankford Road, Suite 160, Carrollton, Texas; 4800 Henrietta Creek Road, Fort Worth, Texas; 3501 Research Drive, Richardson, Texas 75082; 16399 Gateway Path, Frisco, Texas 75033; 1398 Industrial Blvd McKinney, Texas 75069; 3701 Litsey Road, Fort Worth, Texas 76177; 4121 International Pkwy Carrollton, Texas 75007-1907; and 1303 Ridgeview Drive, DDF1, Lewisville, Texas 75057.

12. Amazon has also committed acts of infringement in this District by commercializing, marketing, selling, distributing, testing, and servicing the Accused Products identified herein.

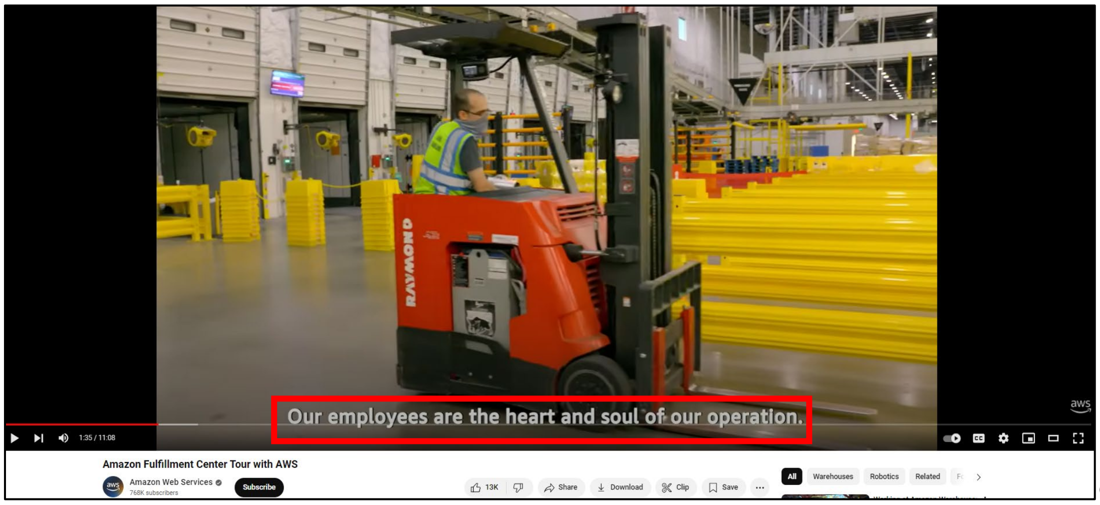
[2]

[2] Amazon FTW3 & FTW4 Fulfillment Centers, located at 15201 Heritage Parkway, Fort Worth, Texas 76177 in the Eastern District of Texas.

13. Amazon Web Services, Inc. ("AWS") operates its business from Amazon's Robotic Fulfillment Centers located within this District, including the FTW3/FTW4 fulfillment center pictured above located at 15201 Heritage Parkway.[3] AWS describes the Amazon Fulfillment Centers as "a symphony of people, AWS, software, and other high-tech components. The result is a coordination between ***our great employees*** and the finely-honed computer systems we've evolved for more than 20 years."[4] Amazon has previously admitted that it employed information technology personal at least at its Fulfillment Center in Denton County, which is in this District[5], and likely employs information technology personnel at other Fulfillment Centers in this District.

14. AWS publicly represents that Fulfillment Center Associates are AWS employees or agents that are conducting the business of AWS.

[3] *See* Amazon Fulfillment Center Tour with AWS, Amazon Web Services YouTube Channel, available at: https://www.youtube.com/watch?v=8nKPC-WmLjU ("Amazon's Fulfillment Centers are wonders of automation, with AWS at their core.").
[4] *Id.* at 1:21 (emphasis added).
[5] *See Jumpsport, Inc. v. Amazon.com, Inc., et al.*, No. 6:17-cv-00666-RWS-JDL, Dkt. 13, ¶ 20 (E.D.T.X. Feb. 12, 2018) ("Amazon admits that it employs information technology personnel at its fulfillment center in Denton County. Amazon denies the remaining allegations in Paragraph 20 of the Complaint."); *see also Vocalife LLC v. Amazon.com, Inc., et al.*, No. 2:19-cv-00123-JRG, Dkt. 226, ¶ 8 (E.D.T.X. July 31, 2020) ("Amazon admits that subsidiaries of Amazon.com, Inc. operate a fulfillment center in this district.").


Our employees are the heart and soul of our operation.
1:35 / 11:08
Amazon Fulfillment Center Tour with AWS
Amazon Web Services
Subscribe
13K
Share
Download
Clip
Save
All
Warehouses
Robotics
Related
[6]

15. In addition to performing core functions of sorting, packing, and shipping products ordered through the Amazon.com website, Fulfillment Center Associates (*i.e.,* AWS employees or agents) actively participate in the operation and improvement of AWS products. For example, when AWS's Machine Learning model has "low confidence" in the classification of a product, "[i]t sends images to people to classify and train more ground truth data for [AWS's] Machine Learning model, which operates in Amazon SageMaker and Amazon SageMaker Ground Truth from AWS."[7]

16. AWS explains that its SageMaker platform "is a fully managed service" that utilizes "human-in-the-loop capabilities" to "[h]arness the power of human feedback across the ML [machine learning] lifecycle to improve the accuracy and relevancy of FMs [foundation models]."[8] Amazon Fulfillment Center Associates fill the "human-in-the-loop" role of providing feedback to the SageMaker platform to improve the accuracy and relevancy of AWS's foundation models.

[6] *See* Amazon Fulfillment Center Tour with AWS at 1:35, Amazon Web Services YouTube Channel, available at: https://www.youtube.com/watch?v=8nKPC-WmLjU
[7] *Id.* at 4:20.
[8] Amazon SageMaker, AWS Website, available at: https://aws.amazon.com/sagemaker/

17. The majority of the square footage ("about 65% of the facility's total square footage") of Amazon Fulfillment Centers are dedicated to robots that operate using AWS's "own created robotic operating software."[9] The Amazon Robotics team has developed "more than a hundred services to support operations and uses AWS extensively in the Amazon Fulfillment Centers."

18. Amazon's Fulfillment Centers, including the FTW3/FTW4 Robotic Fulfillment Centers located in this District, utilize several AWS services and products including but not limited to Amazon Aurora, Amazon Monitron, robotics operating software, Amazon SageMaker, and Industrial Data Fabric. At least Amazon Aurora, Amazon SageMaker, and Industrial Data Fabric support Amazon Elastic Kubernetes Services ("EKS").[10]

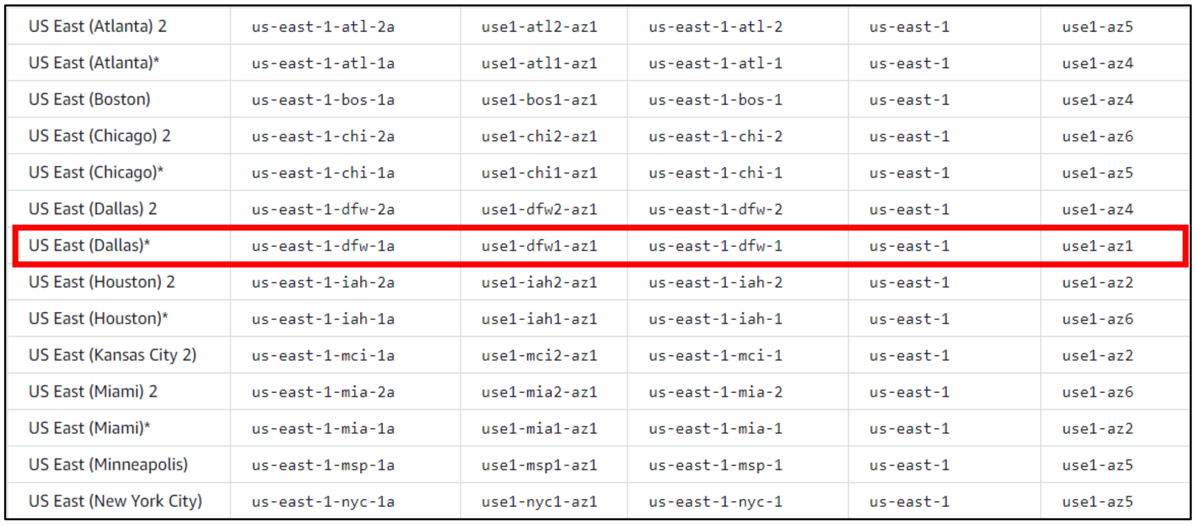
19. AWS relies on both AWS employees, as well as Amazon Fulfillment Center Associates employed by other affiliated subsidiaries of Amazon.com, Inc., who work as agents of AWS to perform the business of AWS from within the Amazon Fulfillment Centers and Amazon

[9] *See* Amazon Fulfillment Center Tour with AWS at 5:18, Amazon Web Services YouTube Channel, available at: https://www.youtube.com/watch?v=8nKPC-WmLjU.

[10] *See* Amazon SageMaker HyperPod introduces Amazon EKS support, AWS Website, available at: https://aws.amazon.com/blogs/aws/amazon-sagemaker-hyperpod-introduces-amazon-eks-support/ ("Today, we are pleased to announce Amazon Elastic Kubernetes Service (EKS) support in Amazon SageMaker HyperPod — purpose-built infrastructure engineered with resilience at its core for foundation model (FM) development."); Scale applications using multi-Region Amazon EKS and Amazon Aurora Global Database: Part 1, AWS Website, available at: https://aws.amazon.com/blogs/database/part-1-scale-applications-using-multi-region-amazon-eks-and-amazon-aurora-global-database/ ("In this two-part series, we show you how you can use Amazon Elastic Kubernetes Service (Amazon EKS) to run your applications in multiple Regions. We use Amazon Aurora Global Database spanning multiple Regions as the persistent transaction data store, and AWS Global Accelerator to distribute traffic between Regions."); Guidance for Industrial Data Fabric with HighByte Intelligence Hub on AWS (3: HighByte Intelligence Hub Industrial DataOps on AWS), AWS Website, available at: https://aws.amazon.com/solutions/guidance/industrial-data-fabric-with-highbyte-intelligence-hub-on-aws/ ("HighByte can be deployed on the edge using an AWS container option, including IoT Greengrass, Amazon Elastic Kubernetes Service (Amazon EKS) Anywhere, and Amazon ECS Anywhere.").

locations within the District. AWS is integral to the successful operation of Amazon's Fulfillment Centers including, but not limited to, the FTW3/FTW4 Fulfillment Centers.[11]

20. AWS maintains a regular and established place of business at 1649 West Frankford Road, Carrollton, Texas 75007. At this location AWS maintains an "AWS Local Zone." An AWS Local Zone is a physical presence that includes AWS hardware in a specific demarcated area along with AWS employees that oversee the hardware. Amazon's own documentation describes an AWS Local Zone thus, "A Local Zone is an extension of an AWS Region in geographic proximity to your users. Local Zones have their own connections to the internet and support AWS Direct Connect, so that resources created in a Local Zone can serve applications that require low latency."[12]



US East (Atlanta) 2	us-east-1-atl-2a	use1-atl2-az1	us-east-1-atl-2	us-east-1	use1-az5
US East (Atlanta)*	us-east-1-atl-1a	use1-atl1-az1	us-east-1-atl-1	us-east-1	use1-az4
US East (Boston)	us-east-1-bos-1a	use1-bos1-az1	us-east-1-bos-1	us-east-1	use1-az4
US East (Chicago) 2	us-east-1-chi-2a	use1-chi2-az1	us-east-1-chi-2	us-east-1	use1-az6
US East (Chicago)*	us-east-1-chi-1a	use1-chi1-az1	us-east-1-chi-1	us-east-1	use1-az5
US East (Dallas) 2	us-east-1-dfw-2a	use1-dfw2-az1	us-east-1-dfw-2	us-east-1	use1-az4
US East (Dallas)*	us-east-1-dfw-1a	use1-dfw1-az1	us-east-1-dfw-1	us-east-1	use1-az1
US East (Houston) 2	us-east-1-iah-2a	use1-iah2-az1	us-east-1-iah-2	us-east-1	use1-az2
US East (Houston)*	us-east-1-iah-1a	use1-iah1-az1	us-east-1-iah-1	us-east-1	use1-az6
US East (Kansas City 2)	us-east-1-mci-1a	use1-mci2-az1	us-east-1-mci-1	us-east-1	use1-az2
US East (Miami) 2	us-east-1-mia-2a	use1-mia2-az1	us-east-1-mia-2	us-east-1	use1-az6
US East (Miami)*	us-east-1-mia-1a	use1-mia1-az1	us-east-1-mia-1	us-east-1	use1-az2
US East (Minneapolis)	us-east-1-msp-1a	use1-msp1-az1	us-east-1-msp-1	us-east-1	use1-az5
US East (New York City)	us-east-1-nyc-1a	use1-nyc1-az1	us-east-1-nyc-1	us-east-1	use1-az5

[13]

[11] *See* AWS for Industrial, AWS Website, available at: https://aws.amazon.com/industrial/ ("Amazon Fulfillment Centers are wonders of innovation and industrial automation. See how robotics, miles of conveyors, complex scanning & sortation equipment, advanced automation and machine learning come together with the help of AWS to get packages to customers in as fast as one day for Prime members.").
[12] *See* How Local Zones Work, AWS Website, available at: https://docs.aws.amazon.com/local-zones/latest/ug/how-local-zones-work.html.
[13] *Id.*

21. This Court has personal jurisdiction over Amazon. Amazon has conducted and continues to conduct business within the State of Texas. Amazon, directly or through subsidiaries or intermediaries (including distributors, retailers, and others), ships, distributes, makes, uses, offers for sale, sells, imports, and/or advertises (including by providing an interactive web page) its products and/or services in the United States and the Eastern District of Texas and/or contributes to and actively induces its customers to ship, distribute, make, use, offer for sale, sell, import, and/or advertise (including the provision of an interactive web page) infringing products and/or services in the United States and the Eastern District of Texas. Amazon, directly and through subsidiaries or intermediaries (including distributors, retailers, and others), has purposefully and voluntarily placed one or more of its infringing products and/or services, as described below, into the stream of commerce with the expectation that those products will be purchased and used by customers and/or consumers in the Eastern District of Texas. These infringing products and/or services have been and continue to be made, used, sold, offered for sale, purchased, and/or imported by customers and/or consumers in the Eastern District of Texas. Amazon has committed acts of patent infringement within the Eastern District of Texas. Amazon interacts with customers in Texas, including through visits to customer sites in Texas. Through these interactions and visits, Amazon directly infringes the patents-in-suit. Amazon also interacts with customers who sell the Accused Products into Texas, knowing that these customers will sell the Accused Products into Texas, either directly or through intermediaries.

22. Amazon has minimum contacts with this District, such that the maintenance of this action within this District would not offend traditional notions of fair play and substantial justice. Thus, the Court therefore has both general and specific personal jurisdiction over Amazon.

PATENTS-IN-SUIT

23. On July 4, 2023, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 11,695,823 (the "'823 Patent") entitled "Distributed Software Defined Networking". A true and correct copy of the '823 Patent is attached hereto as Exhibit A.

24. On January 12, 2021, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 10,893,095 (the "'095 Patent") entitled "Distributed Software Defined Networking". A true and correct copy of the '095 Patent is attached hereto as Exhibit B.

25. On June 16, 2020, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 10,686,871 (the "'871 Patent") entitled "Distributed Software Defined Networking". A true and correct copy of the '871 Patent is attached hereto as Exhibit C.

26. Edge is the sole and exclusive owner of all right, title, and interest in the '823 Patent, the '095 Patent, and the '871 Patent (collectively, the "Patents-in-Suit" or "Asserted Patents"), and holds the exclusive right to take all actions necessary to enforce its rights to the Patents-in-Suit, including the filing of this patent infringement lawsuit. Edge also has the right to recover all damages for past, present, and future infringement of the Patents-in-Suit and to seek injunctive relief as appropriate under the law.

27. Edge has at all times complied with the marking provisions of 35 U.S.C. § 287 with respect to the Patents-in-Suit. Upon information and belief, prior assignees and licensees have also complied with the marking provisions of 35 U.S.C. § 287.

FACTUAL ALLEGATIONS

28. The Patents-in-Suit generally relate to technology for an end-to-end architecture for enabling secure and flexible programmability across a network with full lifecycle management of services and infrastructure applications, as well as harmonizing application deployment across a

network independent of the hardware vendor. The technology described in the Patents-in-Suit was developed by Pouya Taaghol and Vivek Ramanna. For example, this technology is implemented by Amazon in its Amazon Elastic Kubernetes Service ("EKS") Web Services (the "Accused Products").

29. Amazon has infringed and is continuing to infringe the Patents-in-Suit by making, using, selling, offering to sell, and/or importing, and by actively inducing others to make, use, sell, offer to sell, and/or importing, products including, but not limited, the Amazon EKS Web Service, and other infringing technology.

COUNT I
(Infringement of the '823 Patent)

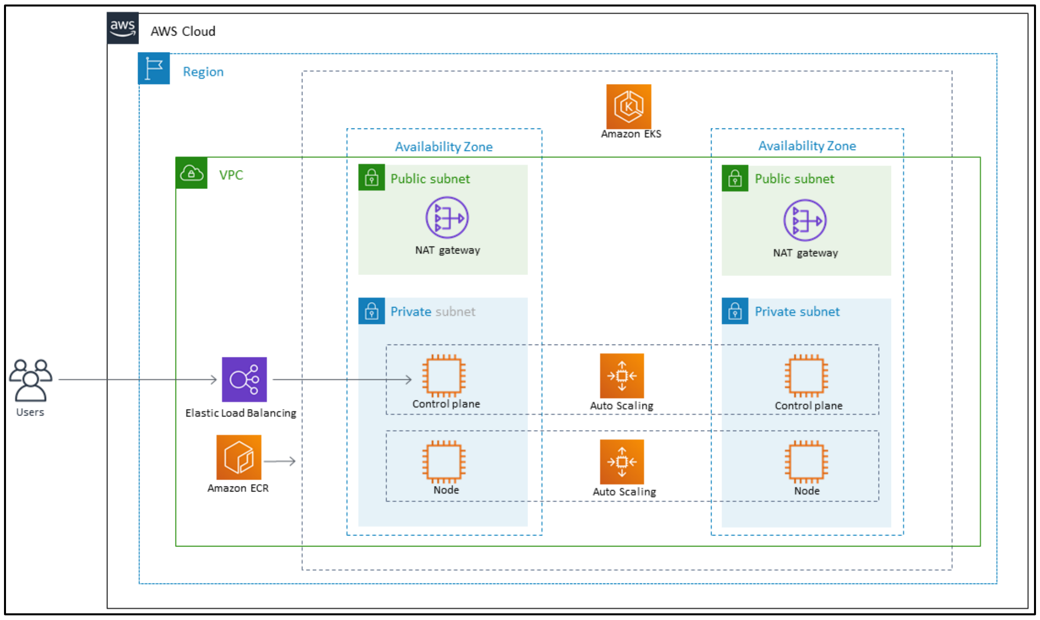
30. Paragraphs 1 through 29 are incorporated by reference as if fully set forth herein.

31. Plaintiff has not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the '823 Patent.

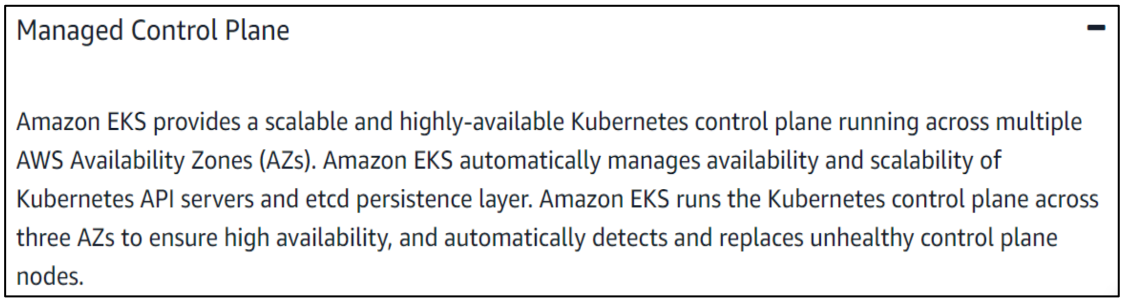
32. Defendants have and continue to directly infringe the '823 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, offering to sell, selling, and/or importing into the United States products that satisfy each and every limitation of one or more claims of the '823 Patent. Such infringing products include Amazon EKS Web Services.

33. For example, Defendants have and continue to directly infringe at least claim 1 of the '823 Patent by making, using, offering to sell, selling, and/or importing into the United States products that include and/or support Amazon EKS Web Services.

34. Amazon EKS Web Services is a system comprising a programmable network device adapted to host a plurality of network device applications (*e.g.*, a managed Kubernetes based service).



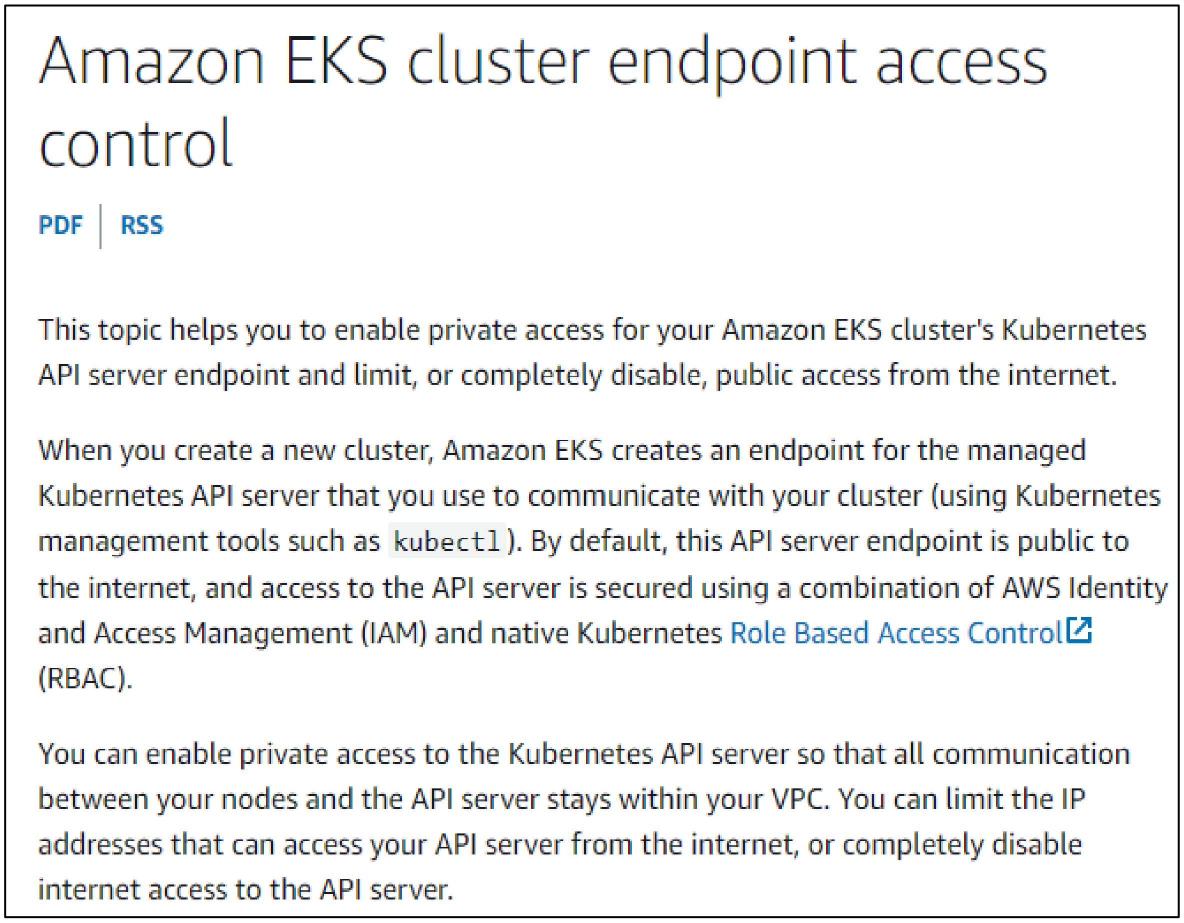
35. Amazon EKS Web Services comprises a programmable cloud device adapted to host a plurality of cloud applications, wherein the plurality of network device applications and the plurality of cloud applications are in secure communication with each other to form distributed applications. For example, the programmable network device can be the first AWS Availability Zone within a managed control plane.


Managed Control Plane

Amazon EKS provides a scalable and highly-available Kubernetes control plane running across multiple AWS Availability Zones (AZs). Amazon EKS automatically manages availability and scalability of Kubernetes API servers and etcd persistence layer. Amazon EKS runs the Kubernetes control plane across three AZs to ensure high availability, and automatically detects and replaces unhealthy control plane nodes.[14]

36. By way for further example, the programmable network device can be the Endpoint for the managed Kubernetes API server.

14 https://aws.amazon.com/eks/features/#:~:text=Amazon%20EKS%20provides%20a%20scalable,servers%20and%20etcd%20persistence%20layer



Amazon EKS cluster endpoint access control

PDF | RSS

This topic helps you to enable private access for your Amazon EKS cluster's Kubernetes API server endpoint and limit, or completely disable, public access from the internet.

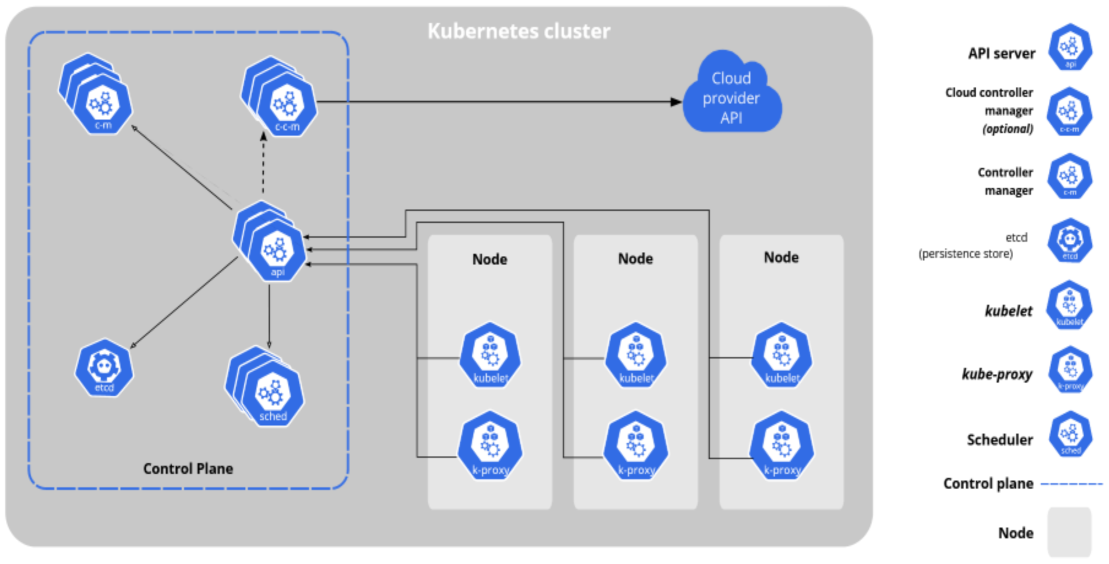
When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). By default, this API server endpoint is public to the internet, and access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC).

You can enable private access to the Kubernetes API server so that all communication between your nodes and the API server stays within your VPC. You can limit the IP addresses that can access your API server from the internet, or completely disable internet access to the API server.[15]

37. By way of further example, the programmable network device can be a kube-proxy and/or kubelet of a node within the first availability zone.

[15] https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

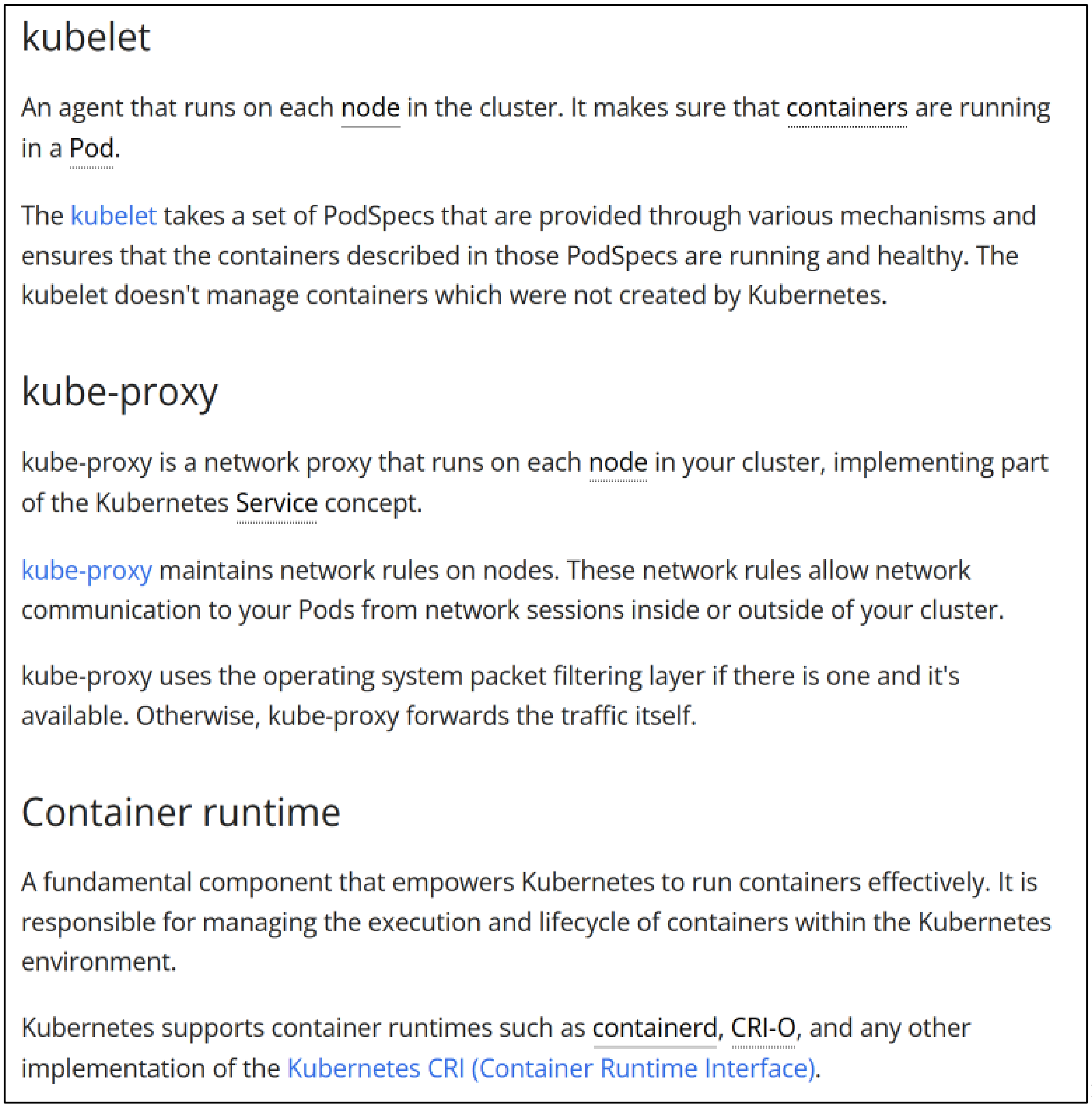
This document outlines the various components you need to have for a complete and working Kubernetes cluster.



The components of a Kubernetes cluster[16]

38. Amazon EKS Web Services comprises a programmable cloud device adapted to host a plurality of cloud applications, wherein the plurality of network device applications and plurality of cloud applications device form unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware. For example, the cloud device (*e.g.*, a node of the second EKS Availability Zone) can be the entire second EKS Availability Zone itself, and/or a Kubernetes container runtime of the node associated with the aforementioned kubelet and/or kube-proxy.

[16] https://kubernetes.io/docs/concepts/overview/components/



kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

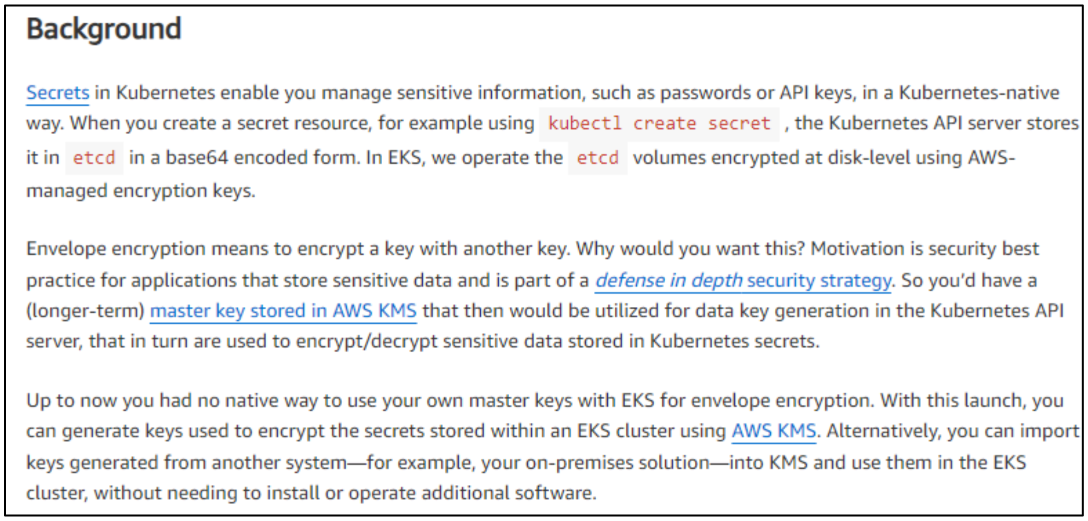
Container runtime

A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).[17]

39. By way of further example, the network applications in both the programmable network device and the programmable cloud device can be in secure communication with each other, forming a distributed application. The secure distributed application is evidenced by Amazon EKS's use of Secrets and end-to-end encryption.

[17] https://kubernetes.io/docs/concepts/architecture/

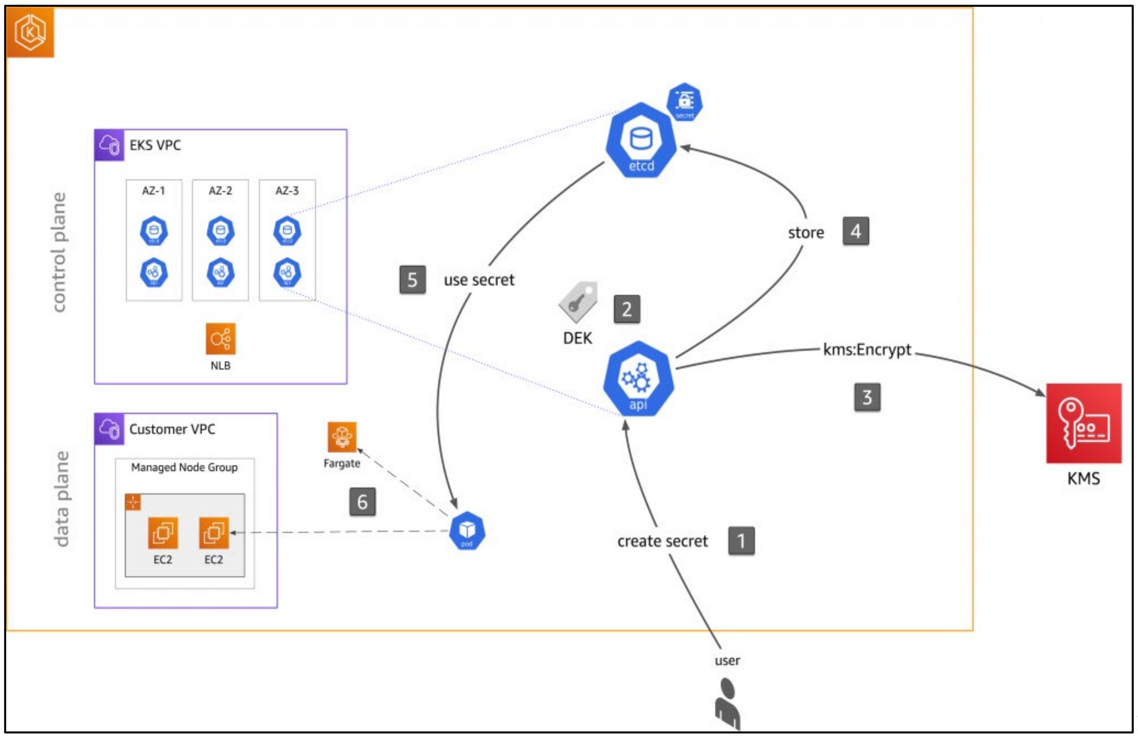


Background

Secrets in Kubernetes enable you manage sensitive information, such as passwords or API keys, in a Kubernetes-native way. When you create a secret resource, for example using `kubectl create secret` , the Kubernetes API server stores it in `etcd` in a base64 encoded form. In EKS, we operate the `etcd` volumes encrypted at disk-level using AWS-managed encryption keys.

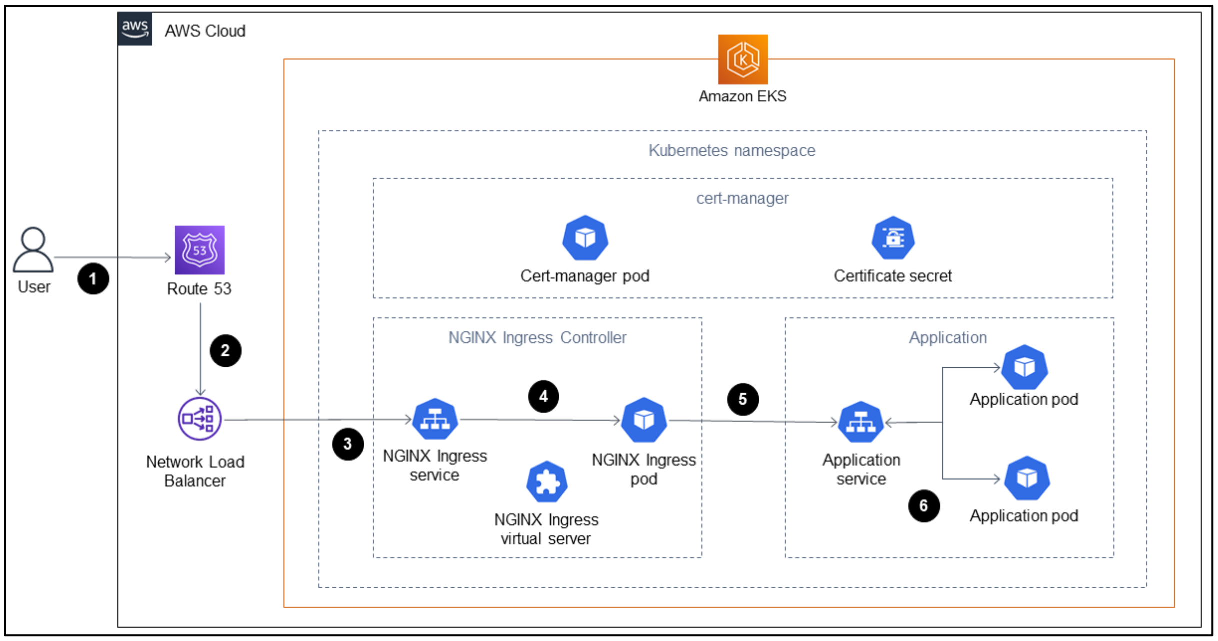
Envelope encryption means to encrypt a key with another key. Why would you want this? Motivation is security best practice for applications that store sensitive data and is part of a *defense in depth* security strategy. So you'd have a (longer-term) master key stored in AWS KMS that then would be utilized for data key generation in the Kubernetes API server, that in turn are used to encrypt/decrypt sensitive data stored in Kubernetes secrets.

Up to now you had no native way to use your own master keys with EKS for envelope encryption. With this launch, you can generate keys used to encrypt the secrets stored within an EKS cluster using AWS KMS. Alternatively, you can import keys generated from another system—for example, your on-premises solution—into KMS and use them in the EKS cluster, without needing to install or operate additional software.

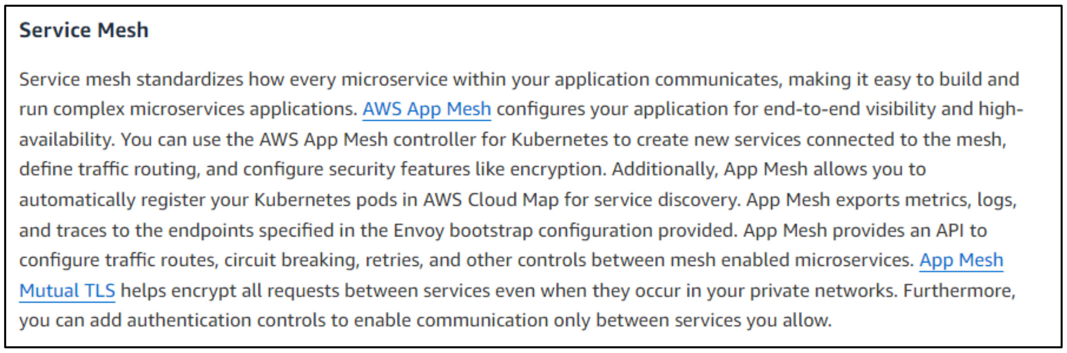


[18]

18 https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/



[19]

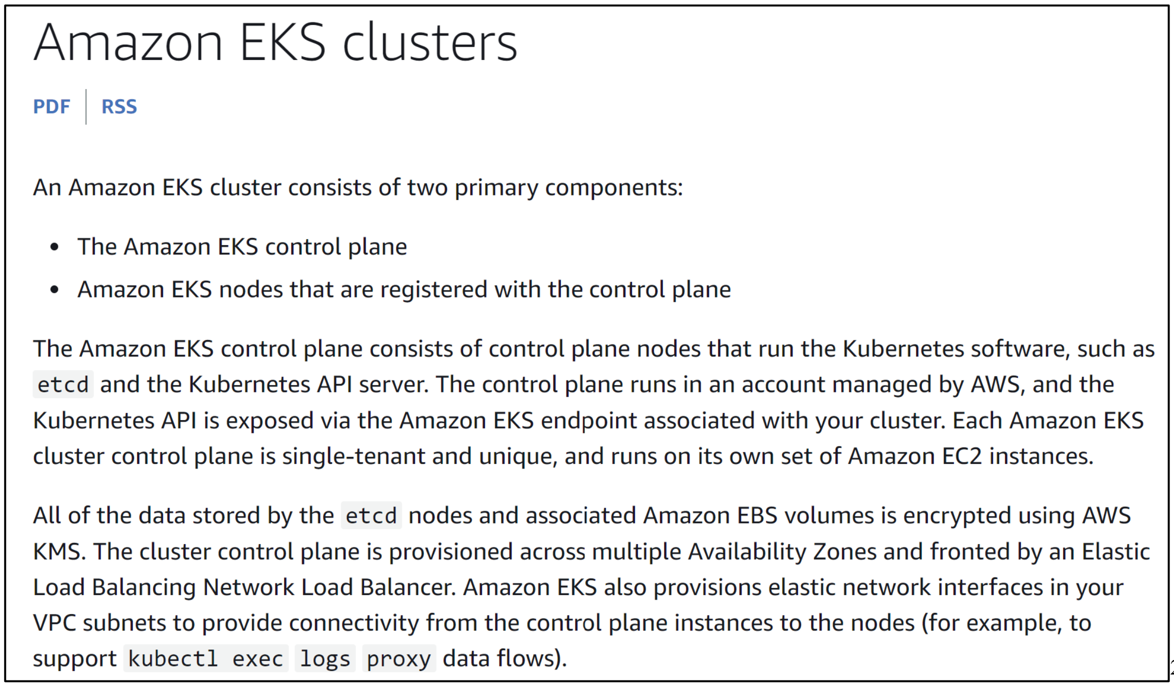


Service Mesh

Service mesh standardizes how every microservice within your application communicates, making it easy to build and run complex microservices applications. AWS App Mesh configures your application for end-to-end visibility and high-availability. You can use the AWS App Mesh controller for Kubernetes to create new services connected to the mesh, define traffic routing, and configure security features like encryption. Additionally, App Mesh allows you to automatically register your Kubernetes pods in AWS Cloud Map for service discovery. App Mesh exports metrics, logs, and traces to the endpoints specified in the Envoy bootstrap configuration provided. App Mesh provides an API to configure traffic routes, circuit breaking, retries, and other controls between mesh enabled microservices. App Mesh Mutual TLS helps encrypt all requests between services even when they occur in your private networks. Furthermore, you can add authentication controls to enable communication only between services you allow.

[20]

[19] https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/set-up-end-to-end-encryption-for-applications-on-amazon-eks-using-cert-manager-and-let-s-encrypt.html
[20] https://aws.amazon.com/eks/features/



Amazon EKS clusters

PDF | RSS

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS nodes that are registered with the control plane

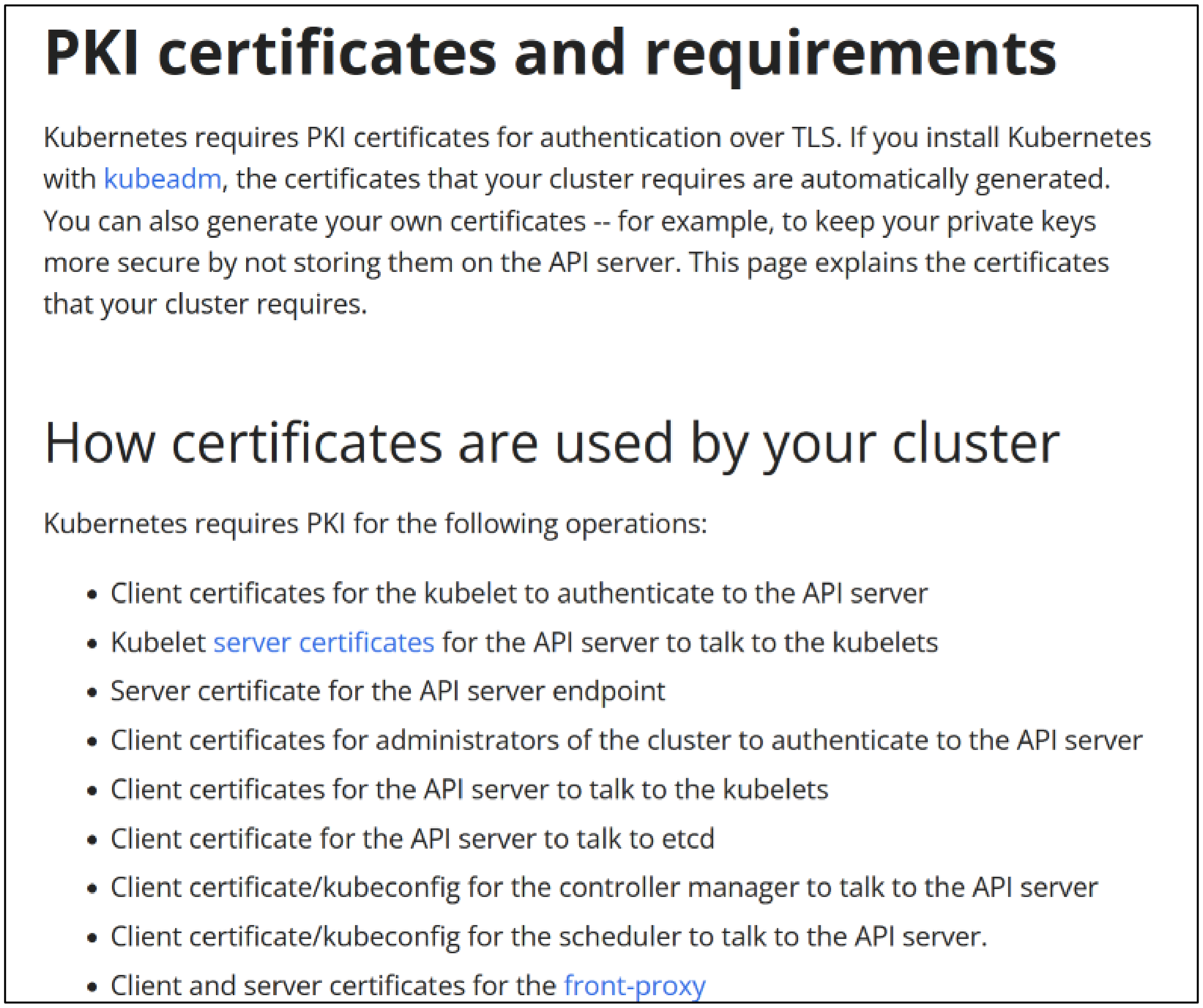
The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as `etcd` and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).

[21]

40. For example, Kubernetes implements PKI between the aforementioned programmable network device and cloud device, implementing a virtual fabric to form a distributed application running between server components.

[21] https://docs.aws.amazon.com/eks/latest/userguide/clusters.html



PKI certificates and requirements

Kubernetes requires PKI certificates for authentication over TLS. If you install Kubernetes with kubeadm, the certificates that your cluster requires are automatically generated. You can also generate your own certificates -- for example, to keep your private keys more secure by not storing them on the API server. This page explains the certificates that your cluster requires.

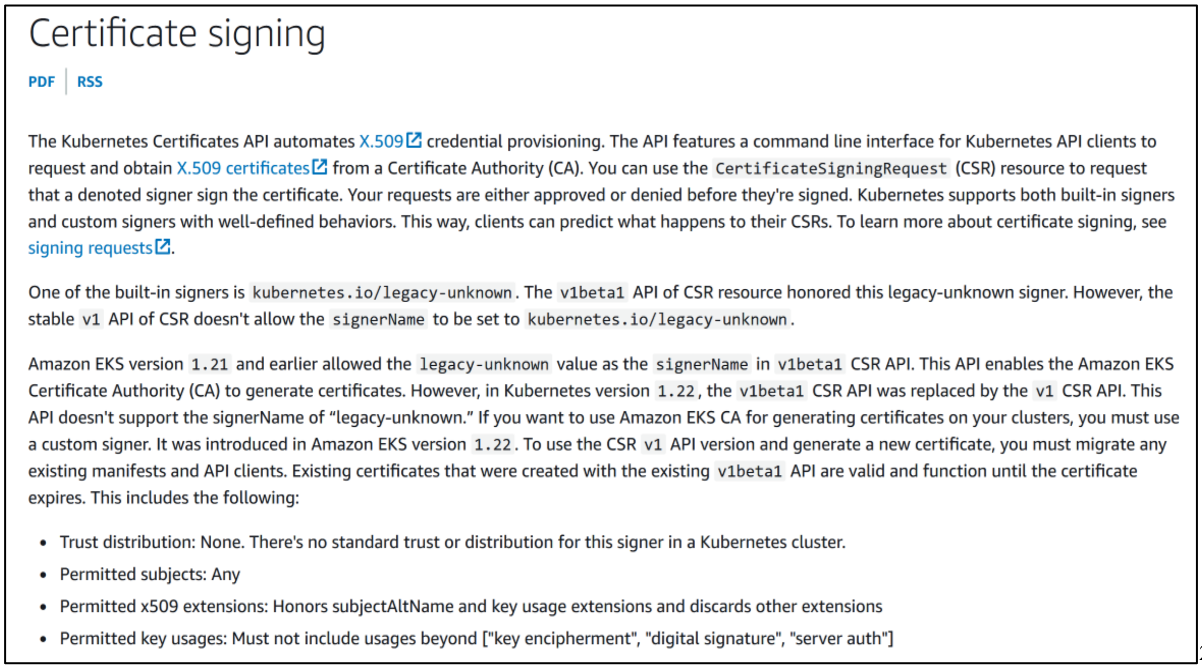
How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Kubelet server certificates for the API server to talk to the kubelets
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

[22]

[22] https://kubernetes.io/docs/setup/best-practices/certificates/


Certificate signing

PDF | RSS

The Kubernetes Certificates API automates X.509 credential provisioning. The API features a command line interface for Kubernetes API clients to request and obtain X.509 certificates from a Certificate Authority (CA). You can use the `CertificateSigningRequest` (CSR) resource to request that a denoted signer sign the certificate. Your requests are either approved or denied before they're signed. Kubernetes supports both built-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see signing requests.

One of the built-in signers is `kubernetes.io/legacy-unknown`. The `v1beta1` API of CSR resource honored this legacy-unknown signer. However, the stable `v1` API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

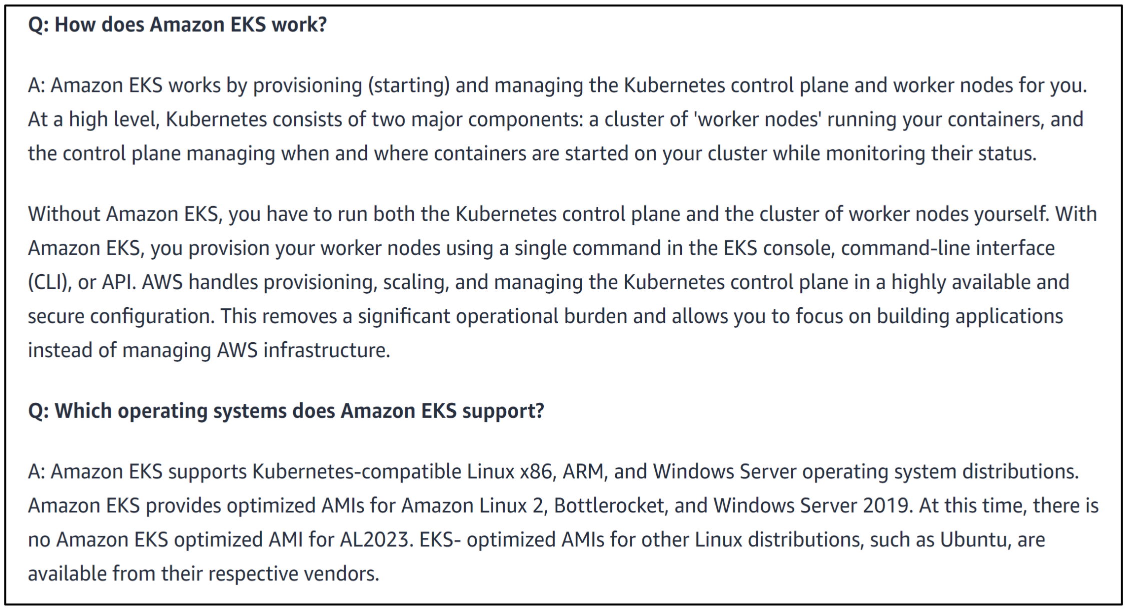
Amazon EKS version `1.21` and earlier allowed the `legacy-unknown` value as the `signerName` in `v1beta1` CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version `1.22`, the `v1beta1` CSR API was replaced by the `v1` CSR API. This API doesn't support the signerName of "legacy-unknown." If you want to use Amazon EKS CA for generating certificates on your clusters, you must use a custom signer. It was introduced in Amazon EKS version `1.22`. To use the CSR `v1` API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the existing `v1beta1` API are valid and function until the certificate expires. This includes the following:

- Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
- Permitted subjects: Any
- Permitted x509 extensions: Honors subjectAltName and key usage extensions and discards other extensions
- Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]

[23]

41. For example, the network applications in both the programmable network device and the programmable cloud device for unified capabilities (*e.g.*, Kubernetes applications) enabling a plurality of upper layer APIs (*e.g.*, control plane APIs) to program the applications independent of network device and cloud device hardware (*e.g.*, through virtual machines implemented across the cloud and network devices). By way of further example, the Network Device and Cloud Device are each powered by a sandboxing operating system (*e.g.*, a server-based operating system running virtual machines partitioned from each other), which facilitate deployment (*e.g.*, deploying an application through Kubernetes hosting models, including during initial deployment and updates). For example, updates to applications through deployments facilitate upgrades of the network applications without interruption to the service provided by those applications.

[23] https://docs.aws.amazon.com/eks/latest/userguide/cert-signing.html



Q: How does Amazon EKS work?

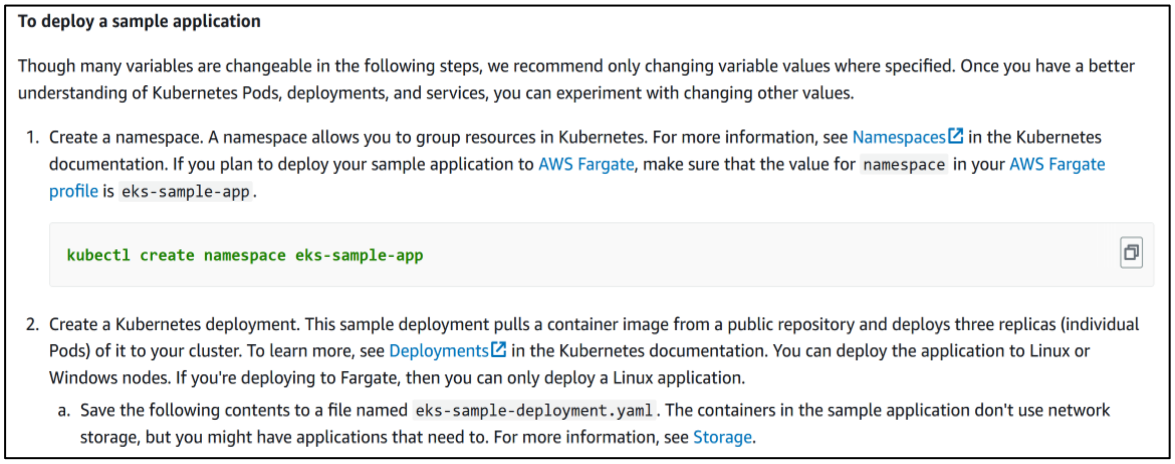
A: Amazon EKS works by provisioning (starting) and managing the Kubernetes control plane and worker nodes for you. At a high level, Kubernetes consists of two major components: a cluster of 'worker nodes' running your containers, and the control plane managing when and where containers are started on your cluster while monitoring their status.

Without Amazon EKS, you have to run both the Kubernetes control plane and the cluster of worker nodes yourself. With Amazon EKS, you provision your worker nodes using a single command in the EKS console, command-line interface (CLI), or API. AWS handles provisioning, scaling, and managing the Kubernetes control plane in a highly available and secure configuration. This removes a significant operational burden and allows you to focus on building applications instead of managing AWS infrastructure.

Q: Which operating systems does Amazon EKS support?

A: Amazon EKS supports Kubernetes-compatible Linux x86, ARM, and Windows Server operating system distributions. Amazon EKS provides optimized AMIs for Amazon Linux 2, Bottlerocket, and Windows Server 2019. At this time, there is no Amazon EKS optimized AMI for AL2023. EKS- optimized AMIs for other Linux distributions, such as Ubuntu, are available from their respective vendors.

[24]



To deploy a sample application

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes Pods, deployments, and services, you can experiment with changing other values.

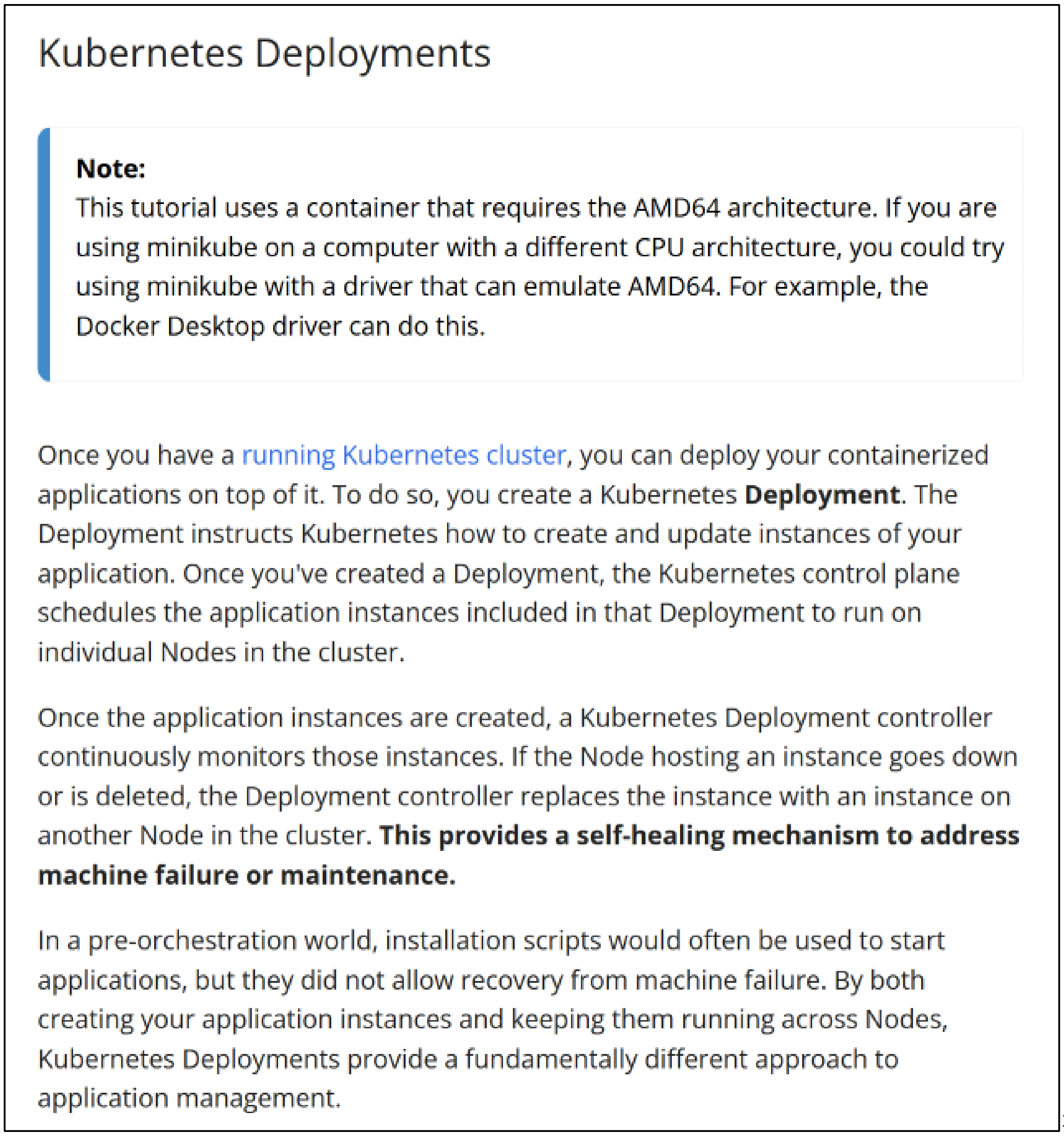
1. Create a namespace. A namespace allows you to group resources in Kubernetes. For more information, see Namespaces in the Kubernetes documentation. If you plan to deploy your sample application to AWS Fargate, make sure that the value for `namespace` in your AWS Fargate profile is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

2. Create a Kubernetes deployment. This sample deployment pulls a container image from a public repository and deploys three replicas (individual Pods) of it to your cluster. To learn more, see Deployments in the Kubernetes documentation. You can deploy the application to Linux or Windows nodes. If you're deploying to Fargate, then you can only deploy a Linux application.
   a. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see Storage.

[25]

[24] https://aws.amazon.com/eks/faqs/
[25] https://docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html


Kubernetes Deployments

Note:
This tutorial uses a container that requires the AMD64 architecture. If you are using minikube on a computer with a different CPU architecture, you could try using minikube with a driver that can emulate AMD64. For example, the Docker Desktop driver can do this.

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment**. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

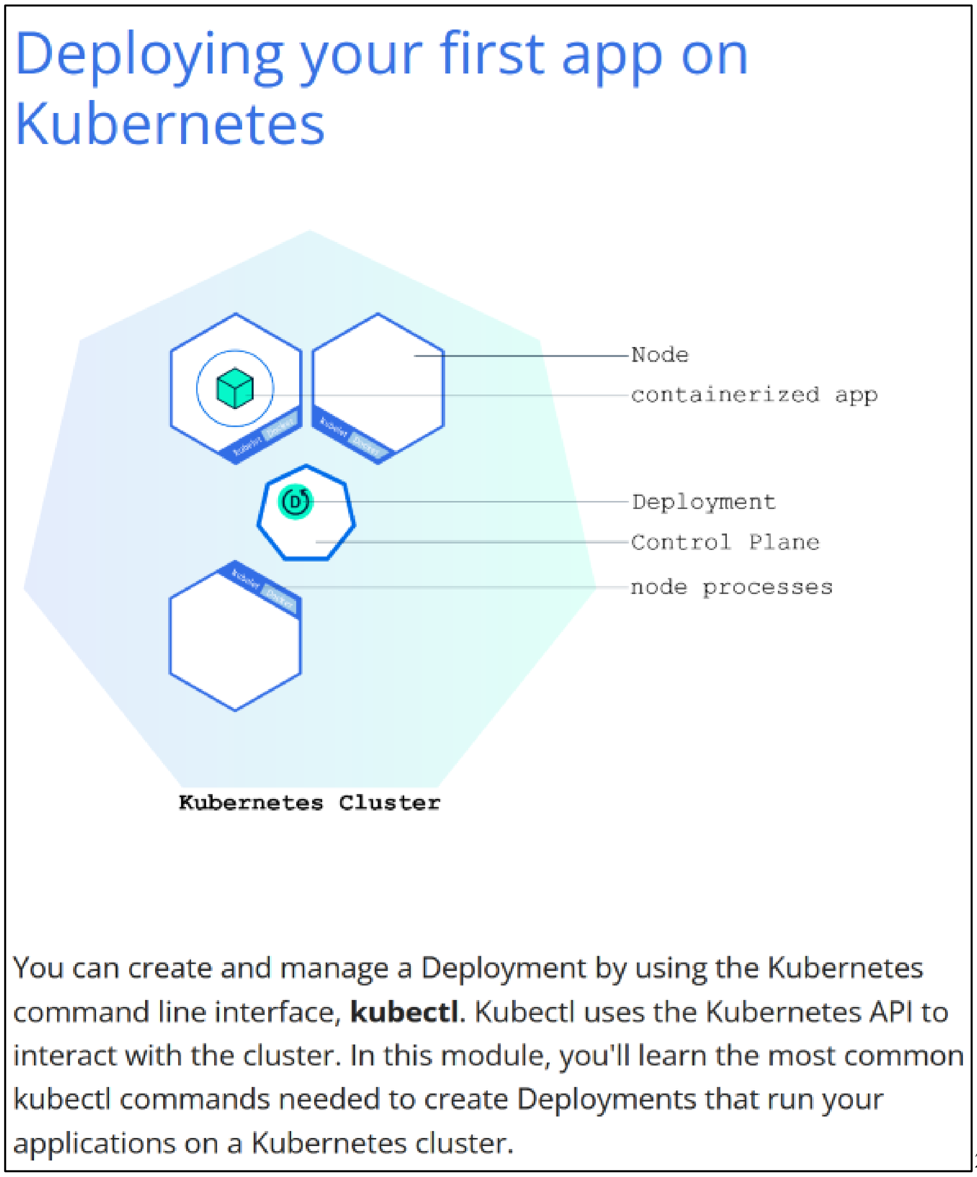
Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**

In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to application management.

[26]

42. For example, the network device and programmable cloud device, as discussed above, can each facilitate deployment of the plurality of network applications (*e.g.*, across nodes, node processes, and control planes).

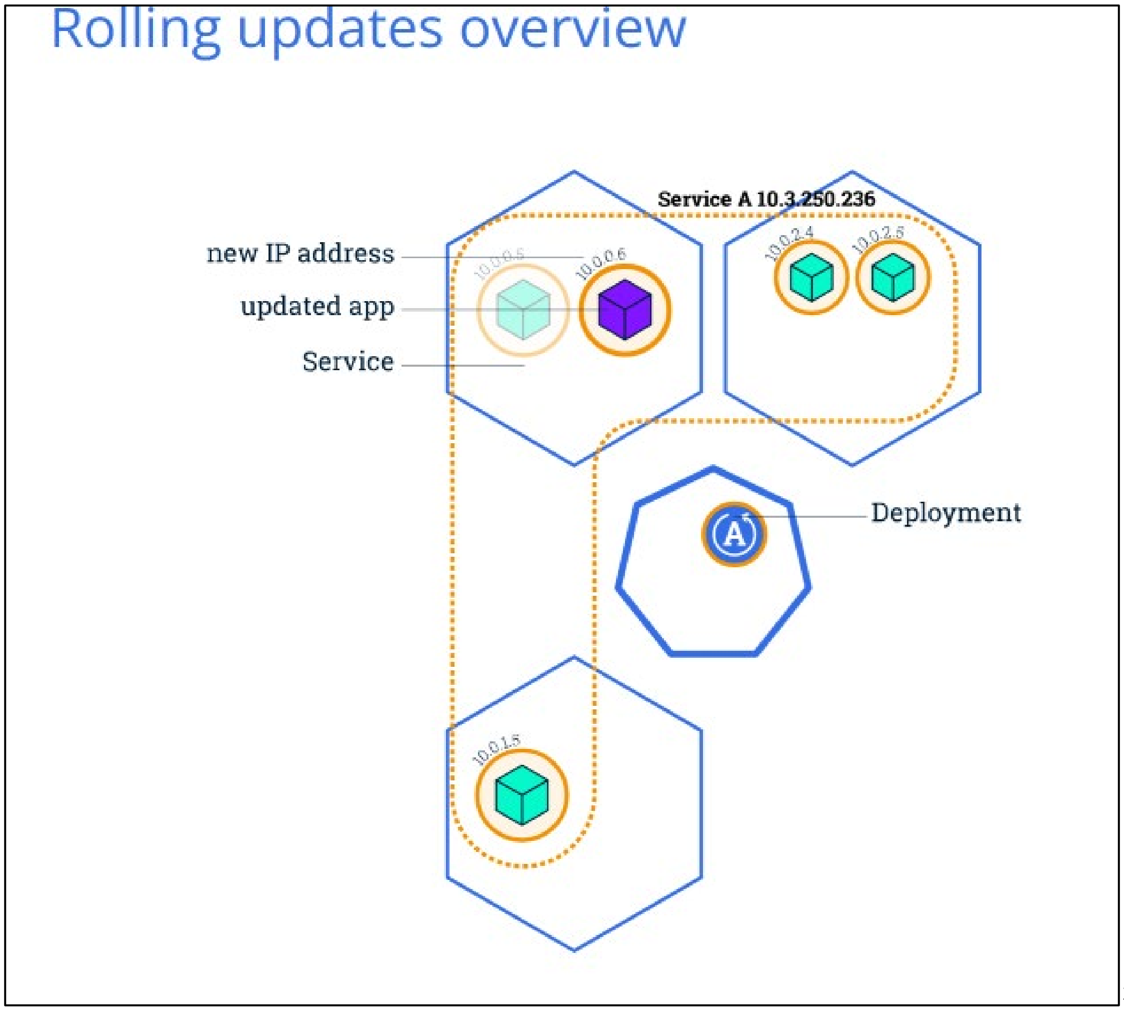
[26] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/



Deploying your first app on Kubernetes

You can create and manage a Deployment by using the Kubernetes command line interface, **kubectl**. Kubectl uses the Kubernetes API to interact with the cluster. In this module, you'll learn the most common kubectl commands needed to create Deployments that run your applications on a Kubernetes cluster.[27]

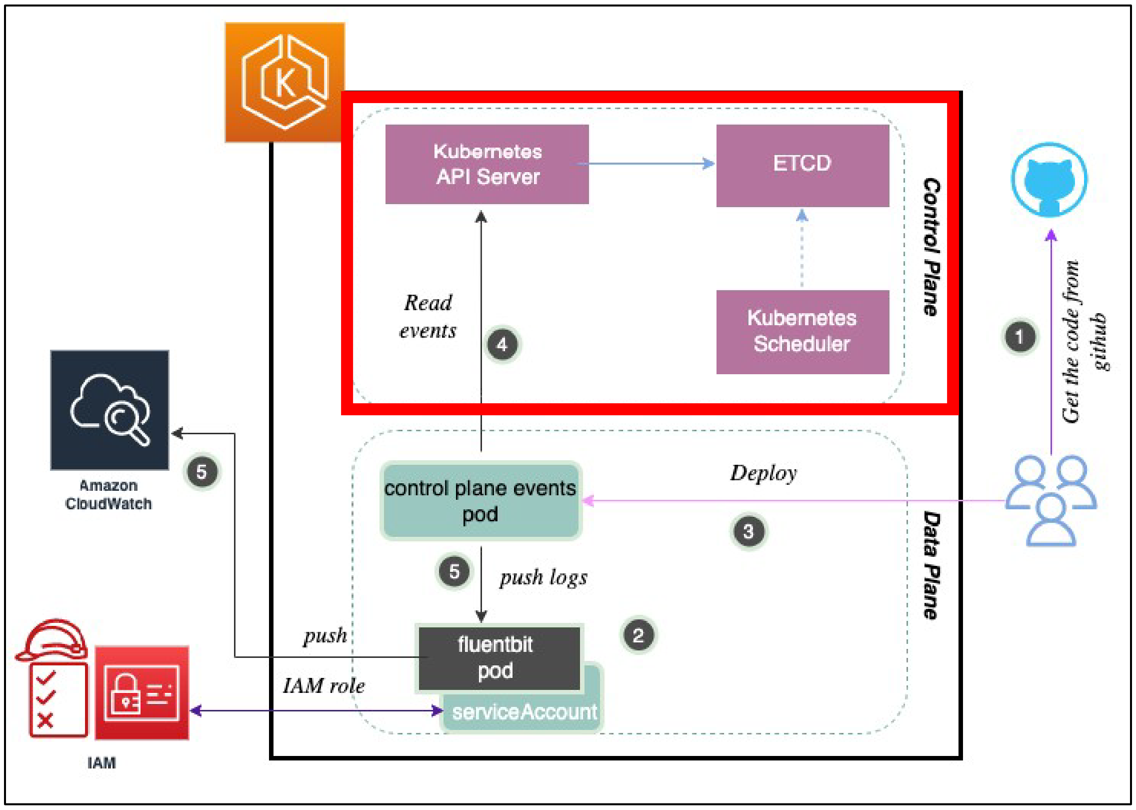
[27] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

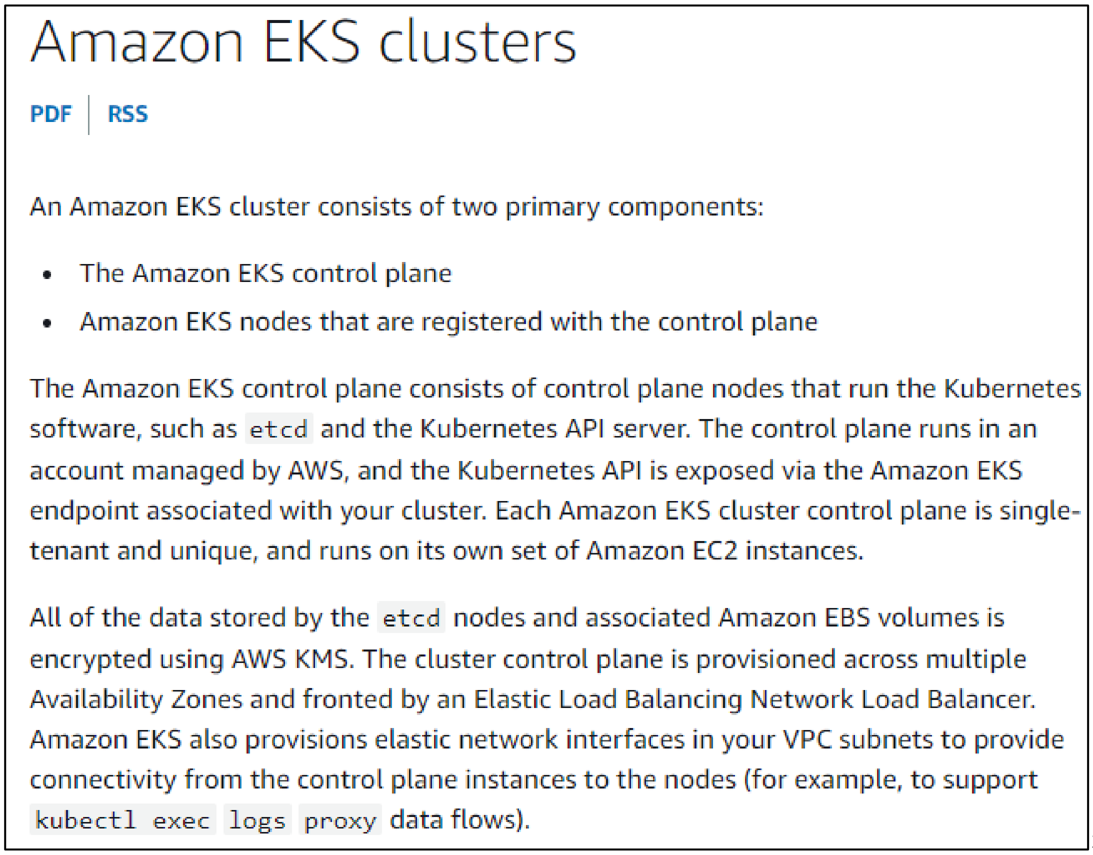


[28]

43. Amazon EKS Web Services includes an application management portal that can be coupled to both the network device and the cloud device, which is capable of managing upgrades with substantially no interruption. The application management portal includes the Control Plane of a managed node group, which includes the dashboard, API server, controllers, and schedulers associated with that Control Plane.

[28] https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/


Kubernetes API Server
ETCD
Kubernetes Scheduler
Control Plane
Read events
4
1
Get the code from github
Amazon CloudWatch
5
control plane events pod
Deploy
3
Data Plane
5
push logs
2
push
fluentbit pod
IAM role
serviceAccount
IAM



Amazon EKS clusters

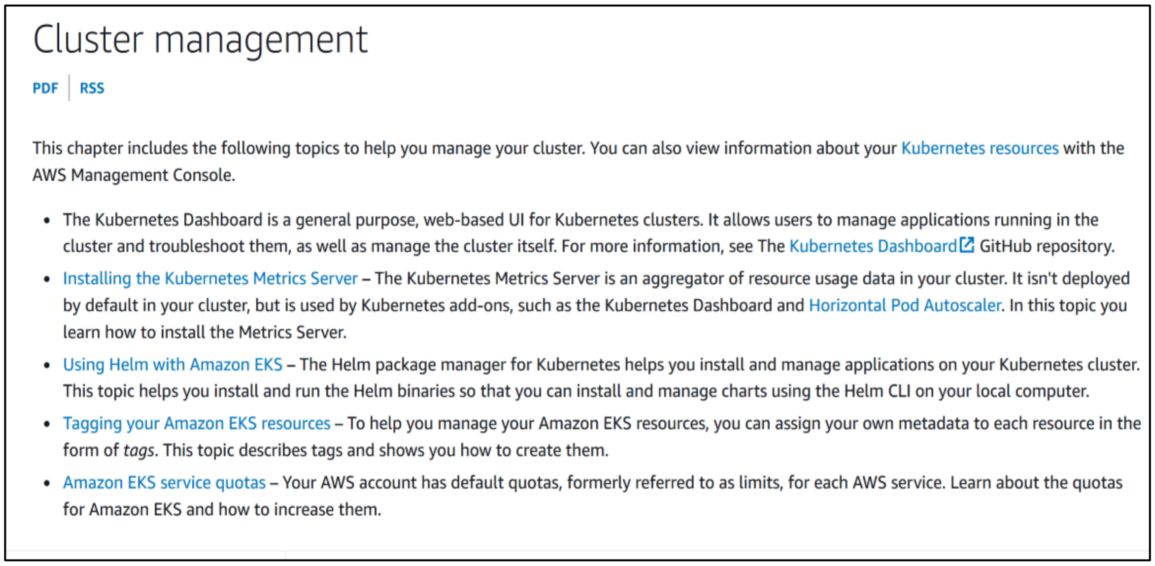
PDF | RSS

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as `etcd` and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).[29]



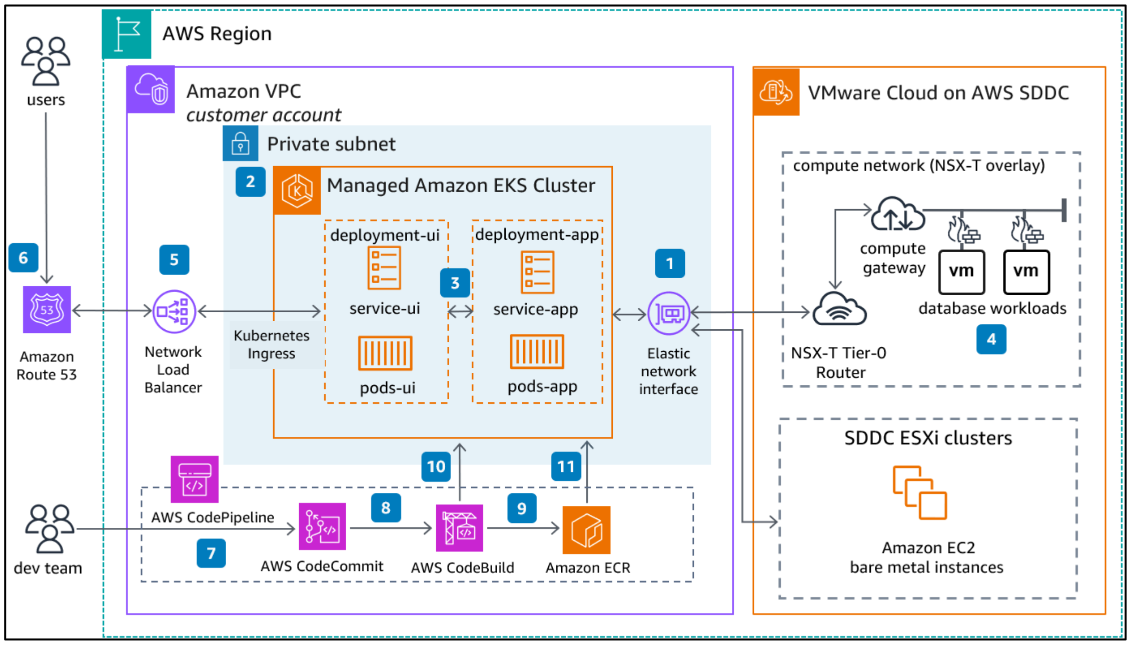
Cluster management

PDF | RSS

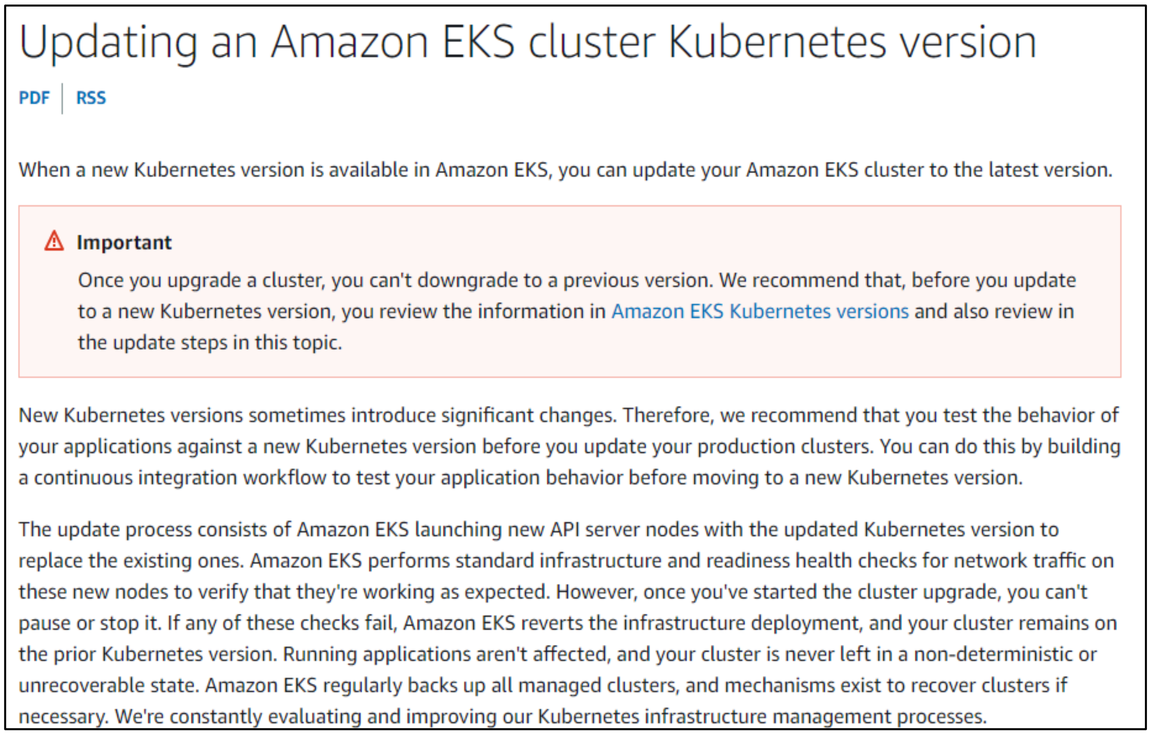
This chapter includes the following topics to help you manage your cluster. You can also view information about your Kubernetes resources with the AWS Management Console.

- The Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself. For more information, see The Kubernetes Dashboard GitHub repository.
- Installing the Kubernetes Metrics Server – The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster. It isn't deployed by default in your cluster, but is used by Kubernetes add-ons, such as the Kubernetes Dashboard and Horizontal Pod Autoscaler. In this topic you learn how to install the Metrics Server.
- Using Helm with Amazon EKS – The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local computer.
- Tagging your Amazon EKS resources – To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.
- Amazon EKS service quotas – Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Learn about the quotas for Amazon EKS and how to increase them.[30]

[29] https://docs.aws.amazon.com/eks/latest/userguide/clusters.html
[30] https://docs.aws.amazon.com/eks/latest/userguide/eks-managing.html



31



Updating an Amazon EKS cluster Kubernetes version

PDF | RSS

When a new Kubernetes version is available in Amazon EKS, you can update your Amazon EKS cluster to the latest version.

Important

Once you upgrade a cluster, you can't downgrade to a previous version. We recommend that, before you update to a new Kubernetes version, you review the information in Amazon EKS Kubernetes versions and also review in the update steps in this topic.

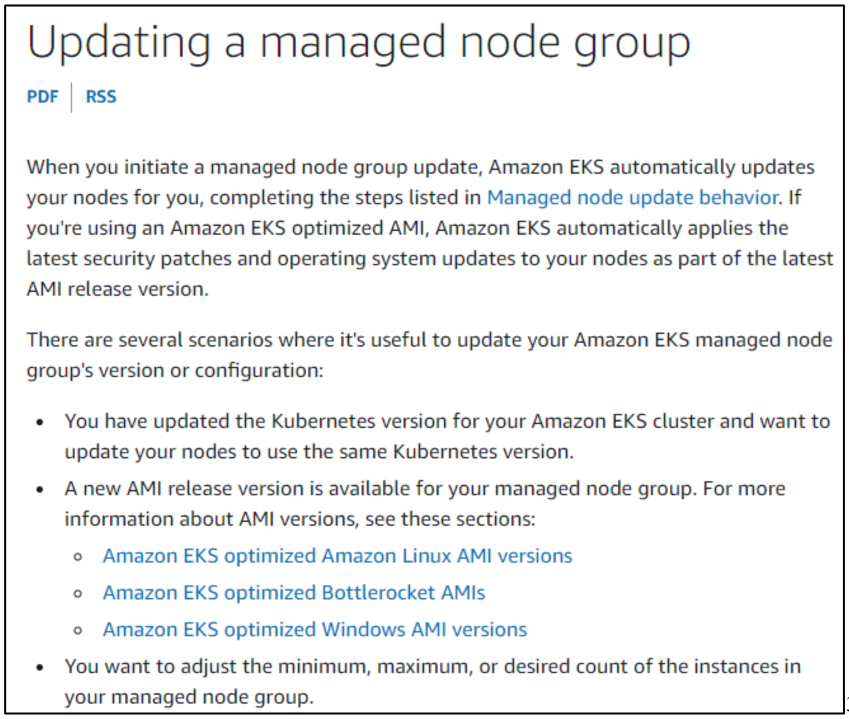
New Kubernetes versions sometimes introduce significant changes. Therefore, we recommend that you test the behavior of your applications against a new Kubernetes version before you update your production clusters. You can do this by building a continuous integration workflow to test your application behavior before moving to a new Kubernetes version.

The update process consists of Amazon EKS launching new API server nodes with the updated Kubernetes version to replace the existing ones. Amazon EKS performs standard infrastructure and readiness health checks for network traffic on these new nodes to verify that they're working as expected. However, once you've started the cluster upgrade, you can't pause or stop it. If any of these checks fail, Amazon EKS reverts the infrastructure deployment, and your cluster remains on the prior Kubernetes version. Running applications aren't affected, and your cluster is never left in a non-deterministic or unrecoverable state. Amazon EKS regularly backs up all managed clusters, and mechanisms exist to recover clusters if necessary. We're constantly evaluating and improving our Kubernetes infrastructure management processes. 32

---

31 https://docs.aws.amazon.com/architecture-diagrams/latest/modernize-applications-with-microservices-using-amazon-eks/modernize-applications-with-microservices-using-amazon-eks.html

32 https://docs.aws.amazon.com/eks/latest/userguide/update-cluster.html



Updating a managed node group

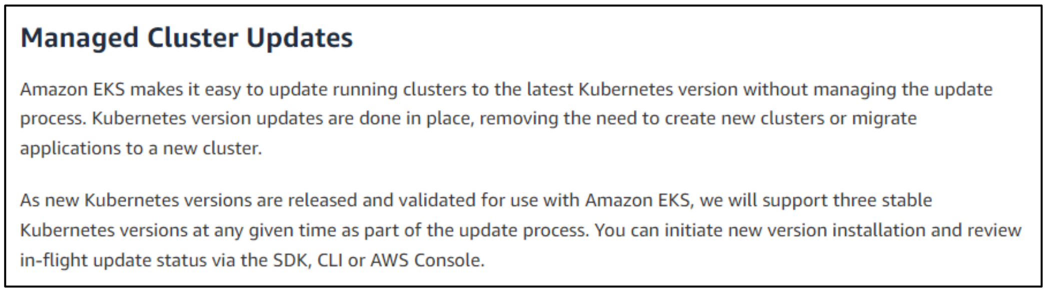
PDF | RSS

When you initiate a managed node group update, Amazon EKS automatically updates your nodes for you, completing the steps listed in Managed node update behavior. If you're using an Amazon EKS optimized AMI, Amazon EKS automatically applies the latest security patches and operating system updates to your nodes as part of the latest AMI release version.

There are several scenarios where it's useful to update your Amazon EKS managed node group's version or configuration:

- You have updated the Kubernetes version for your Amazon EKS cluster and want to update your nodes to use the same Kubernetes version.
- A new AMI release version is available for your managed node group. For more information about AMI versions, see these sections:
  - Amazon EKS optimized Amazon Linux AMI versions
  - Amazon EKS optimized Bottlerocket AMIs
  - Amazon EKS optimized Windows AMI versions
- You want to adjust the minimum, maximum, or desired count of the instances in your managed node group.

[33]



Managed Cluster Updates

Amazon EKS makes it easy to update running clusters to the latest Kubernetes version without managing the update process. Kubernetes version updates are done in place, removing the need to create new clusters or migrate applications to a new cluster.

As new Kubernetes versions are released and validated for use with Amazon EKS, we will support three stable Kubernetes versions at any given time as part of the update process. You can initiate new version installation and review in-flight update status via the SDK, CLI or AWS Console.

[34]

44. Defendants have and continue to indirectly infringe one or more claims of the ’823 Patent by knowingly and intentionally inducing others, including Amazon customers and end-users, to directly infringe, either literally or under the doctrine of equivalents, by making, using, offering to sell, selling, and/or importing into the United States products that include the infringing technology.

[33] https://docs.aws.amazon.com/eks/latest/userguide/update-managed-node-group.html
[34] https://aws.amazon.com/eks/features/

45. Defendants, with knowledge[35] that these products, or the use thereof, infringe the ’823 Patent at least as of the date of this Complaint, knowingly and intentionally induced, and continue to knowingly and intentionally induce, direct infringement of the ’823 Patent by providing these products to end-users for use in an infringing manner. Alternatively, on information and belief, Defendants have adopted a policy of not reviewing the patents of others, including specifically those related to Defendants’ specific industry, thereby remaining willfully blind to the Patent-in-Suit at least as early as the issuance of the Patents-in-Suit.

46. Defendants have induced infringement by others, including end-users, with the intent to cause infringing acts by others or, in the alternative, with the belief that there was a high probability that others, including end-users, infringe the ’823 Patent, but while remaining willfully blind to the infringement. Defendants have and continue to induce infringement by their customers and end-users by supplying them with instructions on how to operate the infringing technology in an infringing manner, while also making publicly available information on the infringing technology via Defendants’ website, product literature and packaging, and other publications.

[35] The ’095 Patent Family was cited as relevant prior art against Amazon Technologies, Inc.’s U.S. Patent Application No. 16/129,632, which was filed on September 12, 2018, and issued as U.S. Patent No. 11,108,687 on August 31, 2021. Upon information and belief, Amazon Technologies, Inc. is a wholly-owned subsidiary of Amazon.com, Inc. Upon information and belief, Amazon Technologies, Inc. is a Nevada corporation formed solely to hold and assert intellectual property rights on behalf of Amazon.com, Inc. and its other subsidiaries. Amazon Technologies, Inc. holds at least 18,743 patents and/or patents applications in the United States alone. *See* https://onscope.com/ipowner/en/owner/profile/1058-amazon-technologies-inc.html; *see also* *Amazon.com, Inc., et al. v. Code Connect Solutions LLC, et al.*, No 5:23-cv-00650-FL, Dkt. 1, ¶ 6 (E.D.N.C. Nov. 9, 2023) (“Amazon Technologies, Inc. (‘Amazon Technologies’) is a Nevada corporation with its principal place of business in Seattle, Washington. Amazon Technologies is a subsidiary of Amazon and is the registered owner of certain intellectual property rights associated with Amazon and affiliated businesses.”). Therefore, Defendants had knowledge of the Patents-in-Suit at least as of the publication of their own patent applications.

47. Plaintiff has suffered damages as a result of Defendants' direct and indirect infringement of the '823 Patent in an amount to be proven at trial.

48. Plaintiff has suffered, and will continue to suffer, irreparable harm as a result of Defendants' infringement of the '823 Patent, for which there is no adequate remedy at law, unless Defendants' infringement is enjoined by this Court.

<u>COUNT II</u>
(Infringement of the '095 Patent)

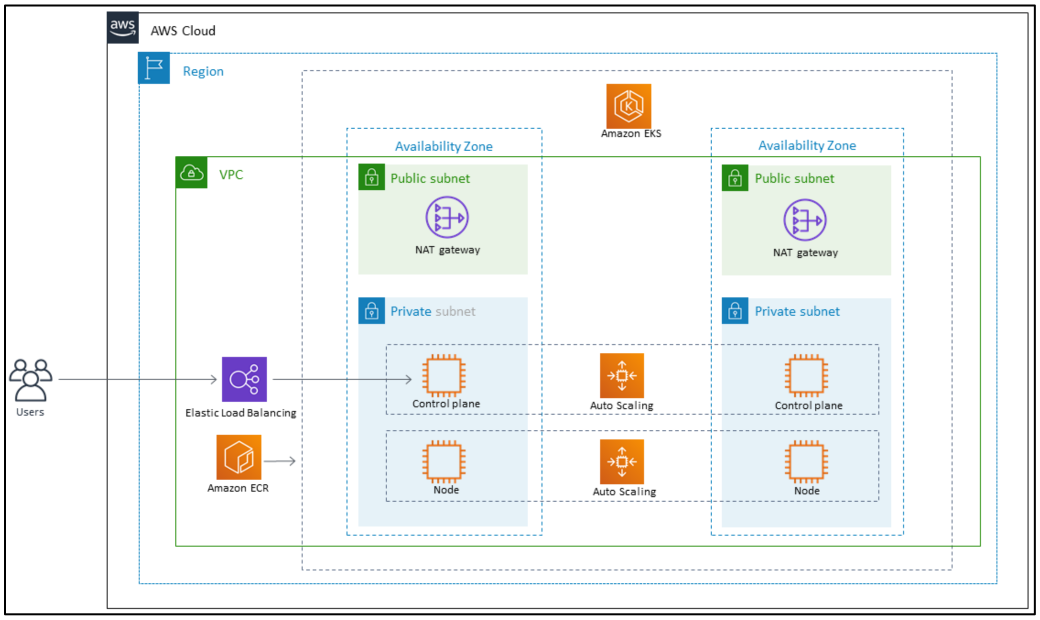
49. Paragraphs 1 through 29 are incorporated by reference as if fully set forth herein.

50. Plaintiff has not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the '095 Patent.

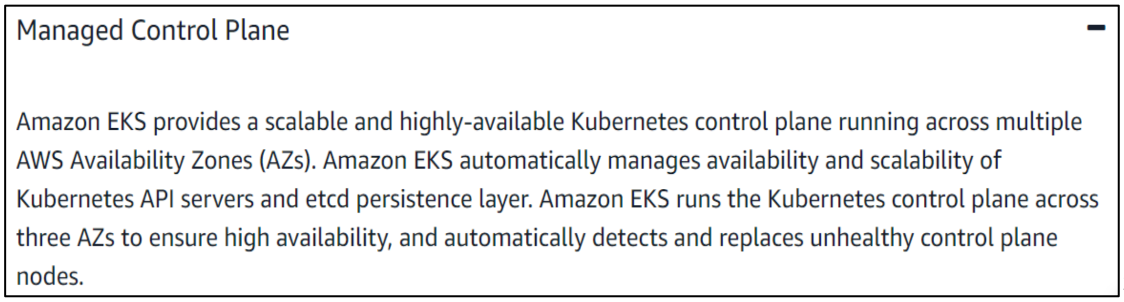
51. Defendants have and continue to directly infringe the '095 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, testing, offering to sell, selling, and/or importing into the United States products that satisfy each and every limitation of one or more claims of the '095 Patent. Such infringing products include Amazon EKS Web Services.

52. For example, Defendants have and continue to directly infringe at least claim 1 of the '095 Patent by making, using, testing, offering to sell, selling, and/or importing into the United States products that include and/or support Amazon EKS Web Services.

53. Amazon EKS Web Services comprises a system for processing data packets in a network.



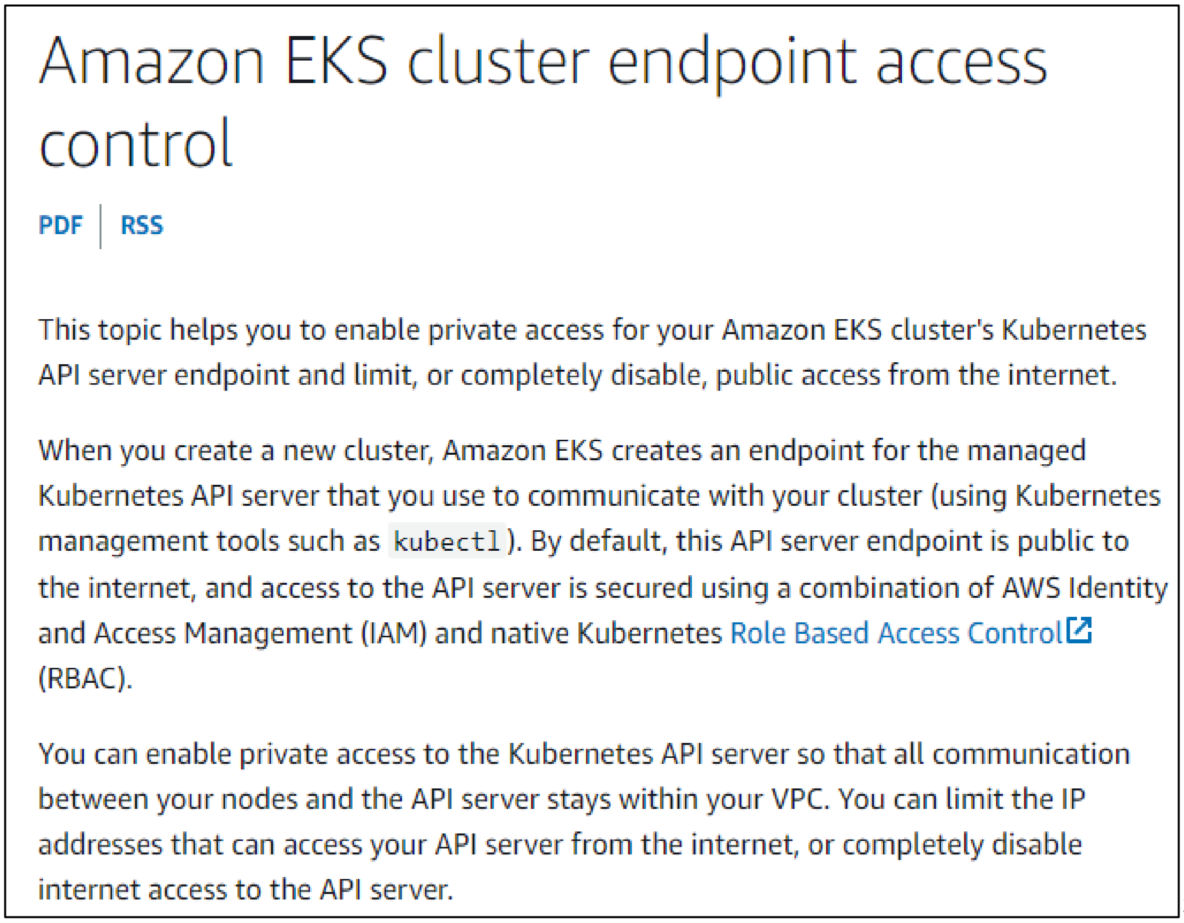
54. Amazon EKS Web Services comprises a programmable network device hosting a plurality of first network applications.


Managed Control Plane

Amazon EKS provides a scalable and highly-available Kubernetes control plane running across multiple AWS Availability Zones (AZs). Amazon EKS automatically manages availability and scalability of Kubernetes API servers and etcd persistence layer. Amazon EKS runs the Kubernetes control plane across three AZs to ensure high availability, and automatically detects and replaces unhealthy control plane nodes.[36]

55. For example, the programmable network device can be the Endpoint for the managed Kubernetes API server.

36 https://aws.amazon.com/eks/features/#:~:text=Amazon%20EKS%20provides%20a%20scalable,servers%20and%20etcd%20persistence%20layer



Amazon EKS cluster endpoint access control

PDF | RSS

This topic helps you to enable private access for your Amazon EKS cluster's Kubernetes API server endpoint and limit, or completely disable, public access from the internet.

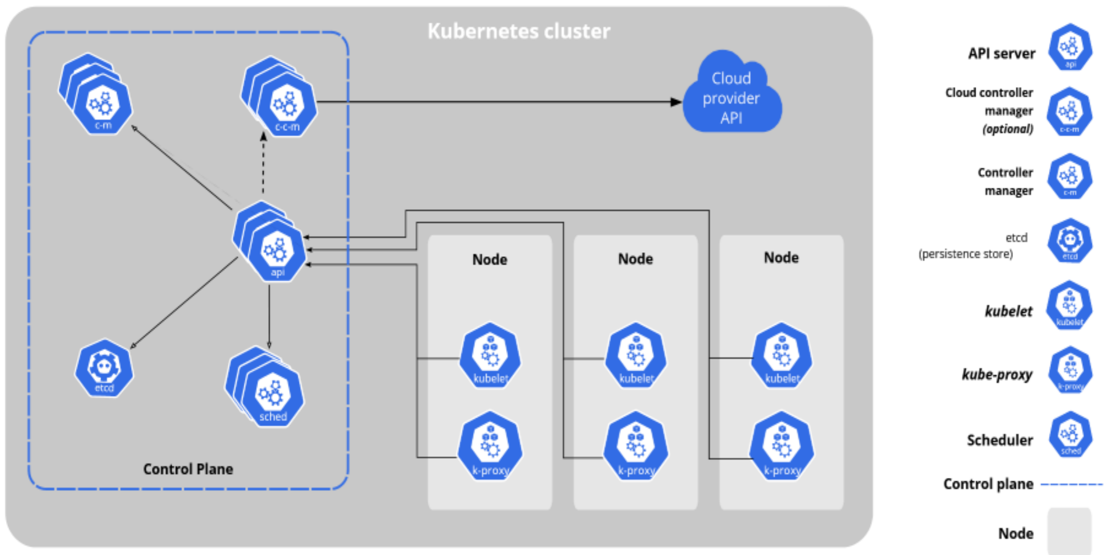
When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). By default, this API server endpoint is public to the internet, and access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC).

You can enable private access to the Kubernetes API server so that all communication between your nodes and the API server stays within your VPC. You can limit the IP addresses that can access your API server from the internet, or completely disable internet access to the API server.[37]

56. By way of further example, the programmable network device can be a kube-proxy and/or kubelet of a node within the first availability zone.

[37] https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

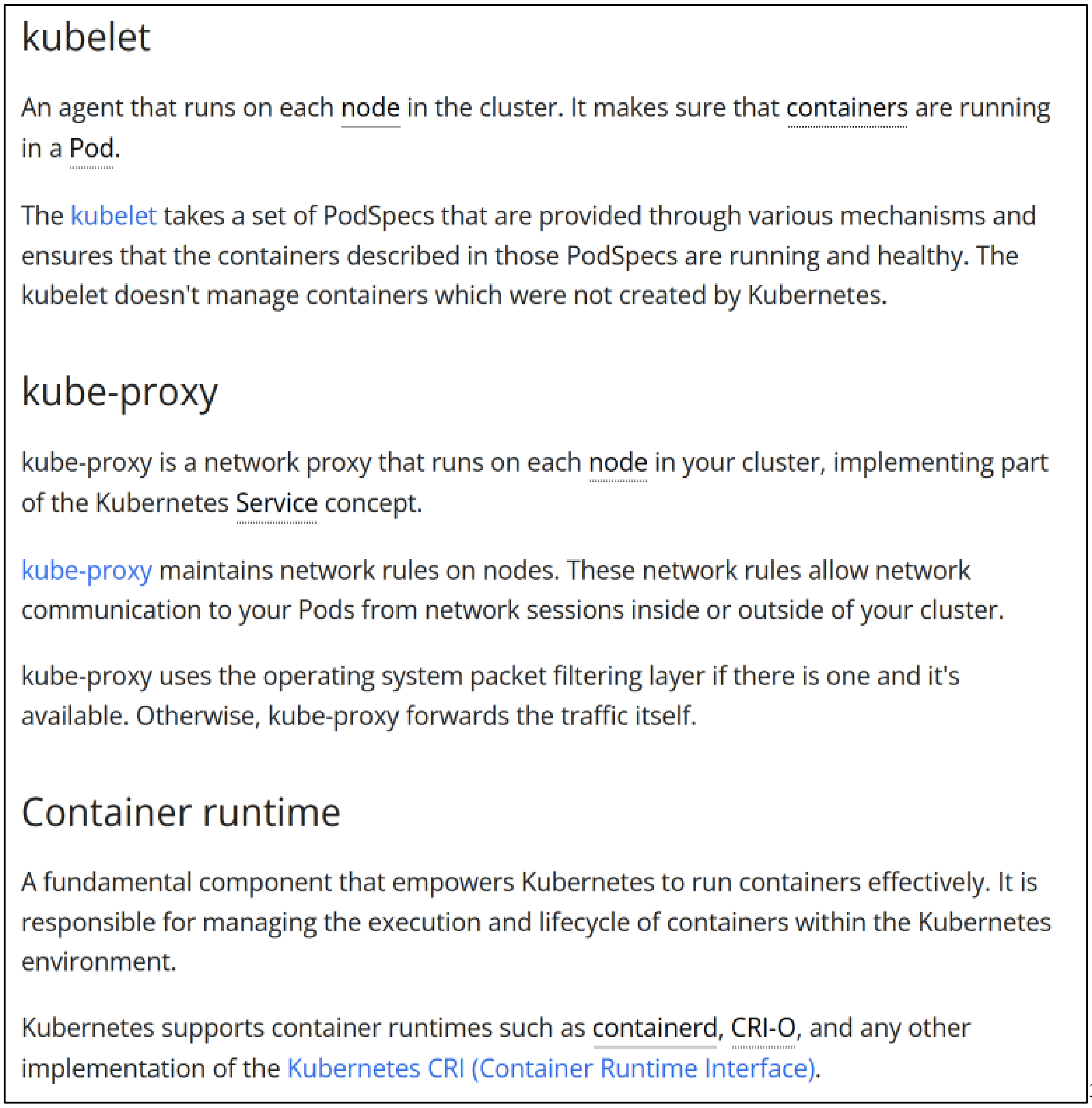
This document outlines the various components you need to have for a complete and working Kubernetes cluster.



The components of a Kubernetes cluster[38]

57. Amazon EKS Web Services comprise a programmable cloud device hosting a plurality of second network applications. For example, the cloud device (*e.g.*, a node of the second EKS Availability Zone) can be the entire second EKS Availability Zone itself, and/or a Kubernetes container runtime of the node associated with the aforementioned kubelet and/or kube-proxy.

[38] https://kubernetes.io/docs/concepts/overview/components/



kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

Container runtime

A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

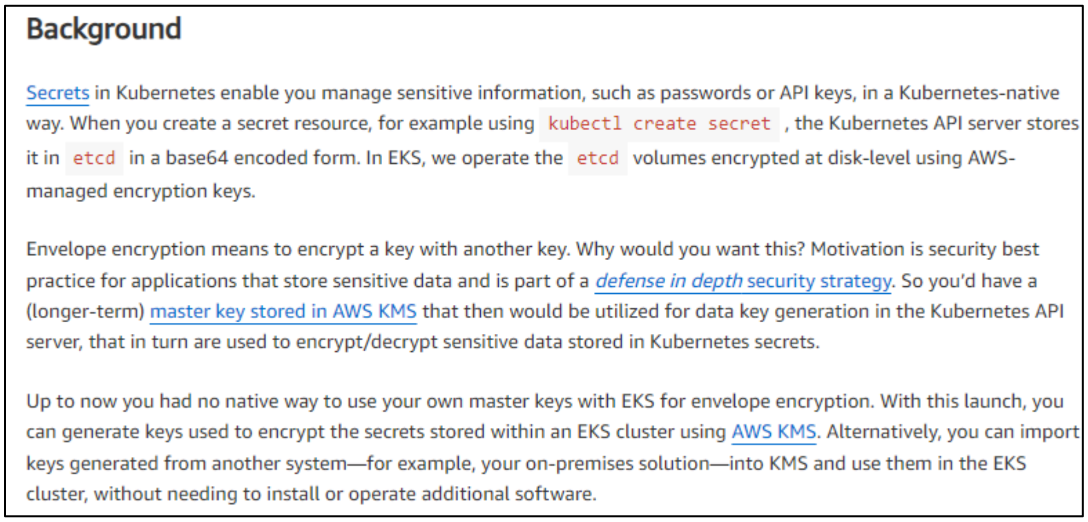
Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

[39]

58. Amazon EKS Web Services comprise a programmable cloud device hosting a plurality of first and second network applications, wherein at least one of the plurality of first network applications in the programmable network device and at least one of the plurality of second network applications in the programmable cloud device are in secure communication with each other to form a distributed application. For example, the network applications in both the

[39] https://kubernetes.io/docs/concepts/architecture/

programmable network device and the programmable cloud device can be in secure communication with each other, forming a distributed application. The secure distributed application is evidenced by Amazon EKS's use of Secrets and end-to-end encryption.

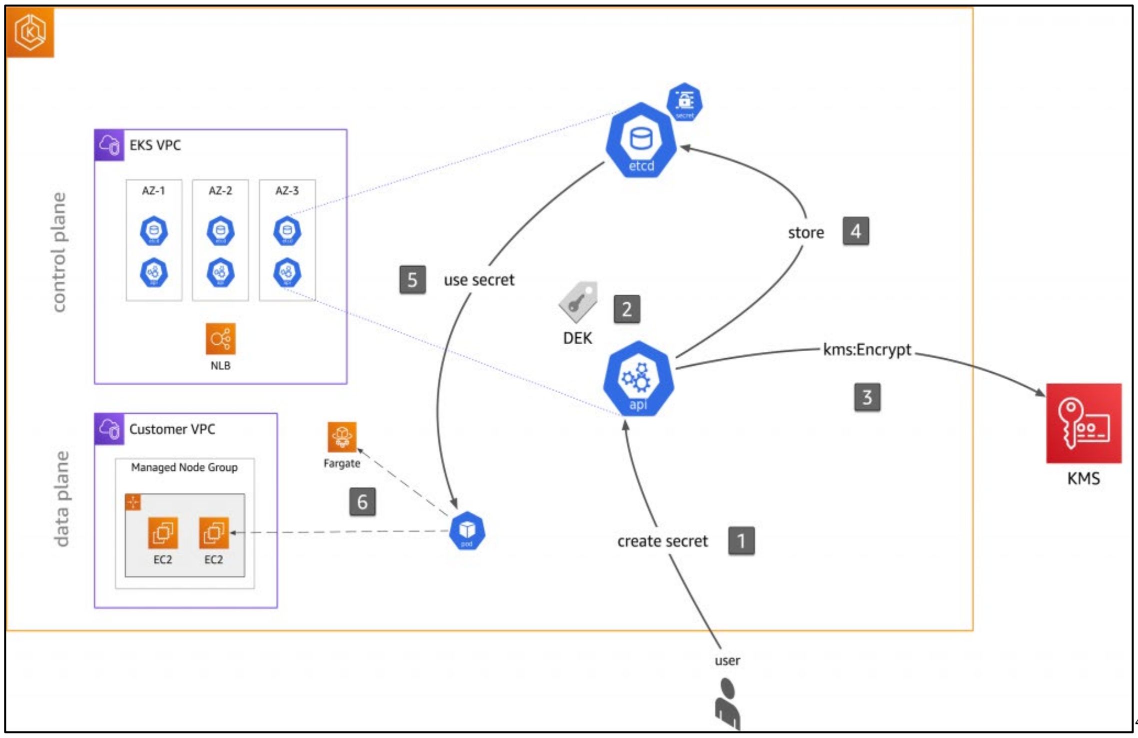


Background

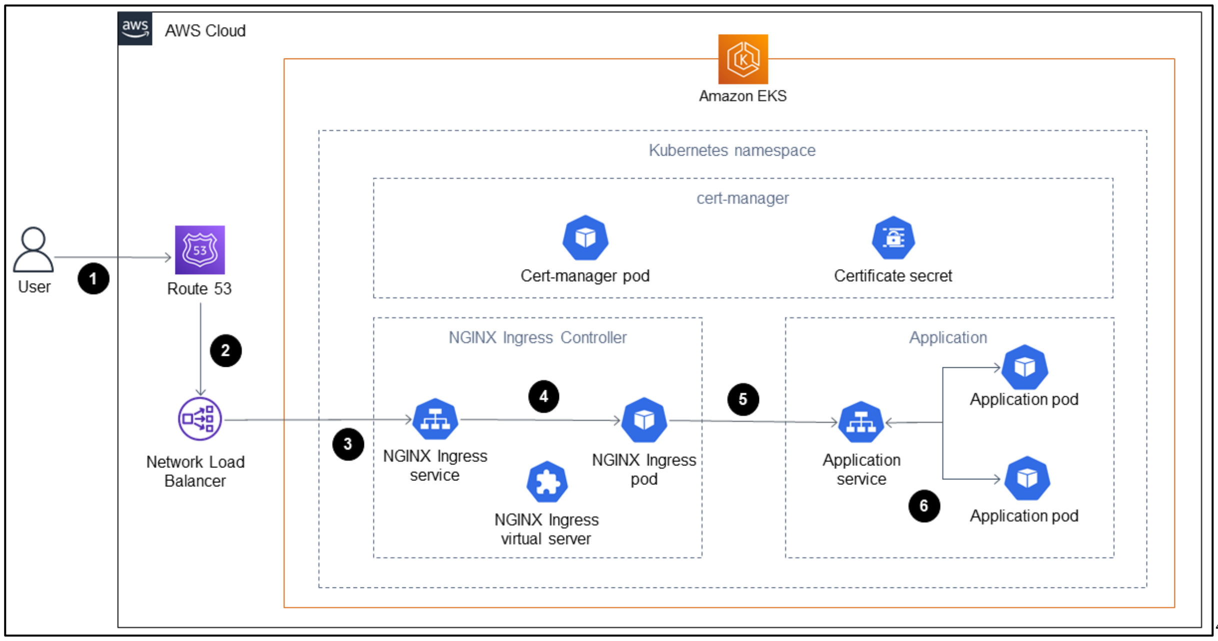
Secrets in Kubernetes enable you manage sensitive information, such as passwords or API keys, in a Kubernetes-native way. When you create a secret resource, for example using `kubectl create secret` , the Kubernetes API server stores it in `etcd` in a base64 encoded form. In EKS, we operate the `etcd` volumes encrypted at disk-level using AWS-managed encryption keys.

Envelope encryption means to encrypt a key with another key. Why would you want this? Motivation is security best practice for applications that store sensitive data and is part of a *defense in depth* security strategy. So you'd have a (longer-term) master key stored in AWS KMS that then would be utilized for data key generation in the Kubernetes API server, that in turn are used to encrypt/decrypt sensitive data stored in Kubernetes secrets.

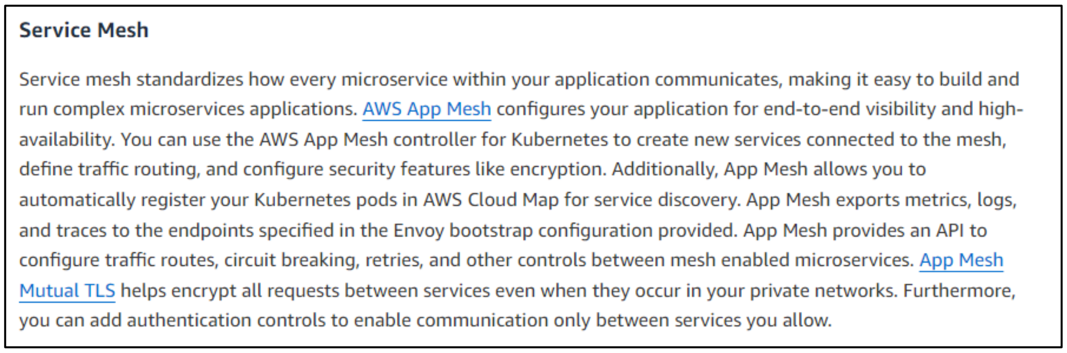
Up to now you had no native way to use your own master keys with EKS for envelope encryption. With this launch, you can generate keys used to encrypt the secrets stored within an EKS cluster using AWS KMS. Alternatively, you can import keys generated from another system—for example, your on-premises solution—into KMS and use them in the EKS cluster, without needing to install or operate additional software.


[40]

[40] https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/

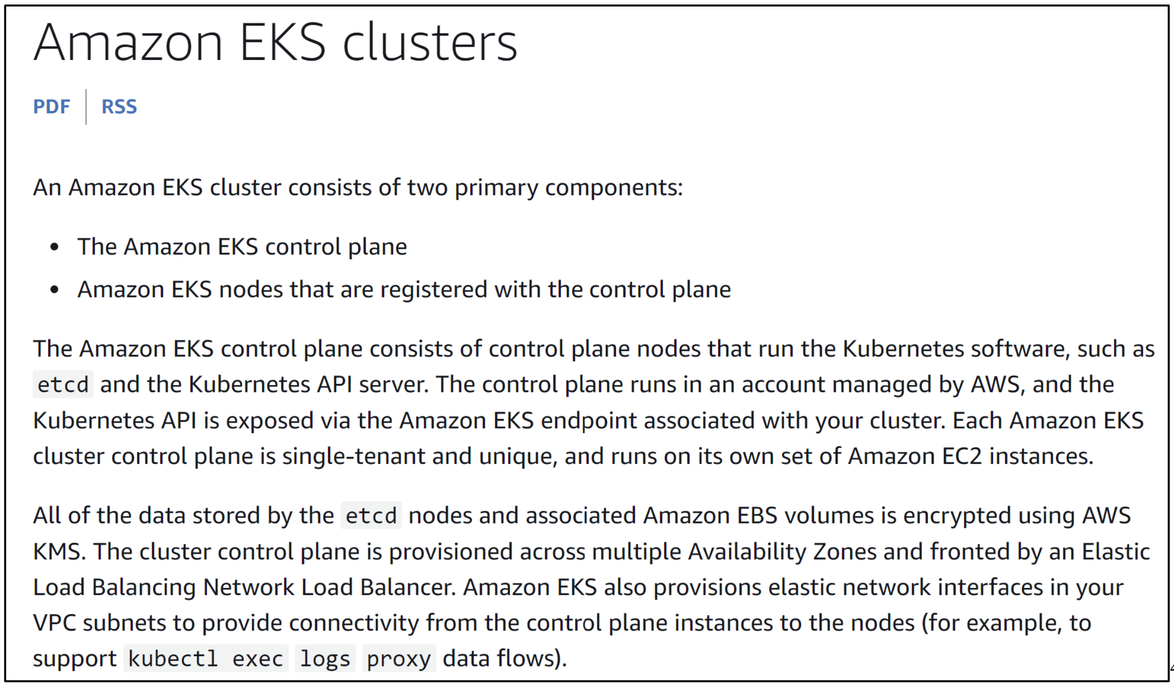


[41]


Service Mesh

Service mesh standardizes how every microservice within your application communicates, making it easy to build and run complex microservices applications. AWS App Mesh configures your application for end-to-end visibility and high-availability. You can use the AWS App Mesh controller for Kubernetes to create new services connected to the mesh, define traffic routing, and configure security features like encryption. Additionally, App Mesh allows you to automatically register your Kubernetes pods in AWS Cloud Map for service discovery. App Mesh exports metrics, logs, and traces to the endpoints specified in the Envoy bootstrap configuration provided. App Mesh provides an API to configure traffic routes, circuit breaking, retries, and other controls between mesh enabled microservices. App Mesh Mutual TLS helps encrypt all requests between services even when they occur in your private networks. Furthermore, you can add authentication controls to enable communication only between services you allow.
[42]

[41] https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/set-up-end-to-end-encryption-for-applications-on-amazon-eks-using-cert-manager-and-let-s-encrypt.html.
[42] https://aws.amazon.com/eks/features/



Amazon EKS clusters

PDF | RSS

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS nodes that are registered with the control plane

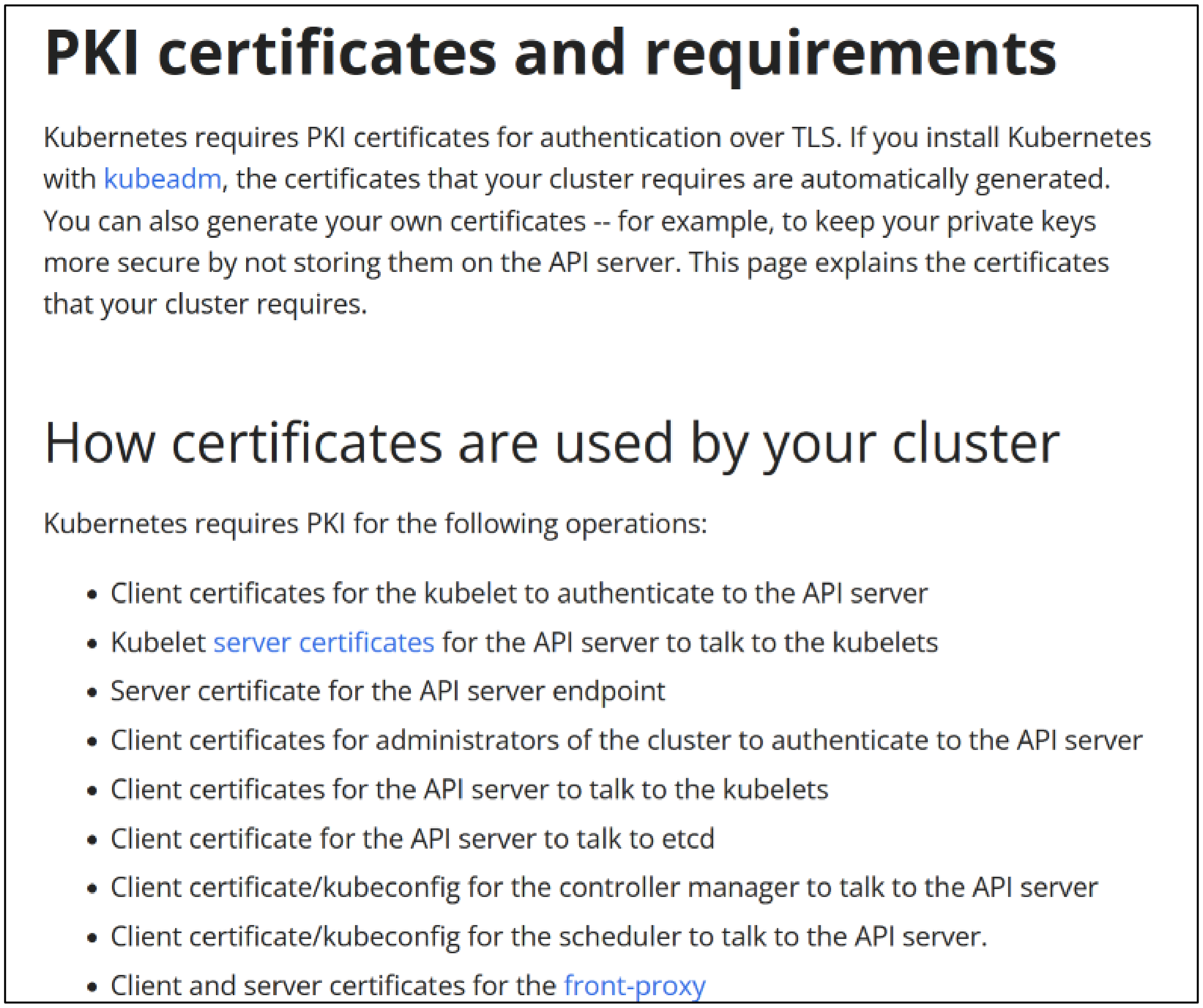
The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as `etcd` and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).

[43]

59. For example, Kubernetes implements PKI between the aforementioned programmable network device and cloud device, implementing a virtual fabric to form a distributed application running between server components.

[43] https://docs.aws.amazon.com/eks/latest/userguide/clusters.html



PKI certificates and requirements

Kubernetes requires PKI certificates for authentication over TLS. If you install Kubernetes with kubeadm, the certificates that your cluster requires are automatically generated. You can also generate your own certificates -- for example, to keep your private keys more secure by not storing them on the API server. This page explains the certificates that your cluster requires.

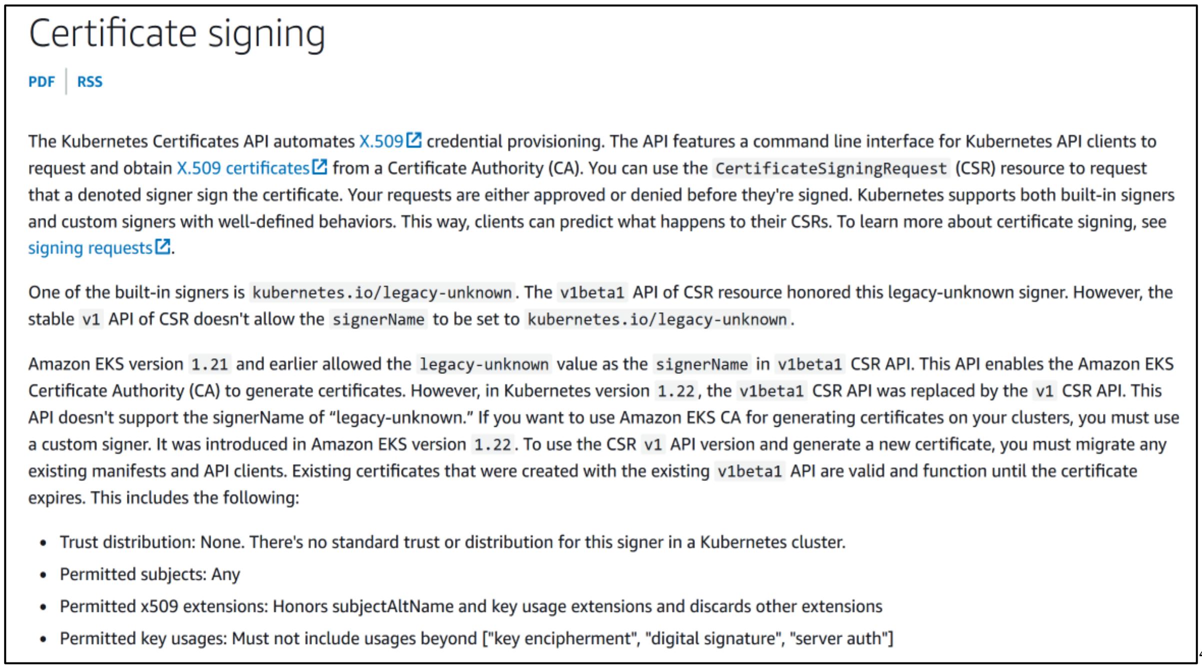
How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Kubelet server certificates for the API server to talk to the kubelets
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

[44]

[44] https://kubernetes.io/docs/setup/best-practices/certificates/



Certificate signing

PDF | RSS

The Kubernetes Certificates API automates X.509 credential provisioning. The API features a command line interface for Kubernetes API clients to request and obtain X.509 certificates from a Certificate Authority (CA). You can use the `CertificateSigningRequest` (CSR) resource to request that a denoted signer sign the certificate. Your requests are either approved or denied before they're signed. Kubernetes supports both built-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see signing requests.

One of the built-in signers is `kubernetes.io/legacy-unknown`. The `v1beta1` API of CSR resource honored this legacy-unknown signer. However, the stable `v1` API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

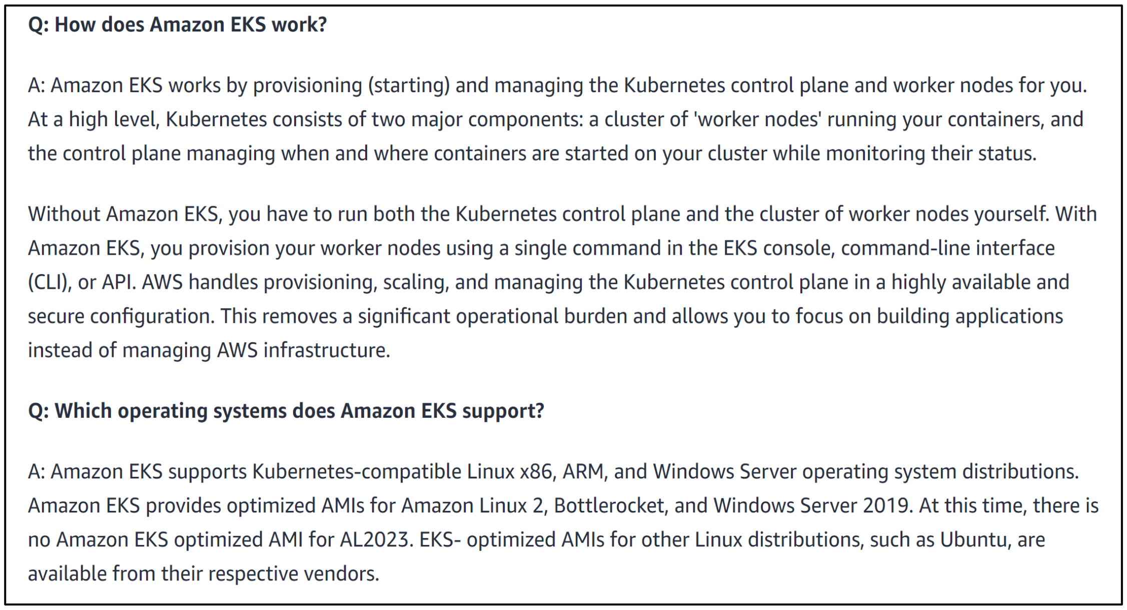
Amazon EKS version `1.21` and earlier allowed the `legacy-unknown` value as the `signerName` in `v1beta1` CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version `1.22`, the `v1beta1` CSR API was replaced by the `v1` CSR API. This API doesn't support the signerName of "legacy-unknown." If you want to use Amazon EKS CA for generating certificates on your clusters, you must use a custom signer. It was introduced in Amazon EKS version `1.22`. To use the CSR `v1` API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the existing `v1beta1` API are valid and function until the certificate expires. This includes the following:

- Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
- Permitted subjects: Any
- Permitted x509 extensions: Honors subjectAltName and key usage extensions and discards other extensions
- Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]

[45]

60. Amazon EKS Web Services comprise the programmable network device and programmable cloud device and are each powered by a sandboxing operating system which facilitates deployment of the plurality of first and second network applications and facilitates upgrades of the first and second network applications with substantially no interruption to operation of the programmable network device and programmable cloud device. For example, the Network Device and Cloud Device are each powered by a sandboxing operating system (*e.g.*, a server-based operating system running virtual machines partitioned from each other), which facilitate deployment (*e.g.*, deploying an application through Kubernetes hosting models, including during initial deployment and updates). By way of further example, updates to applications through deployments facilitate upgrades of the network applications without interruption to the service provided by those applications.

[45] https://docs.aws.amazon.com/eks/latest/userguide/cert-signing.html



Q: How does Amazon EKS work?

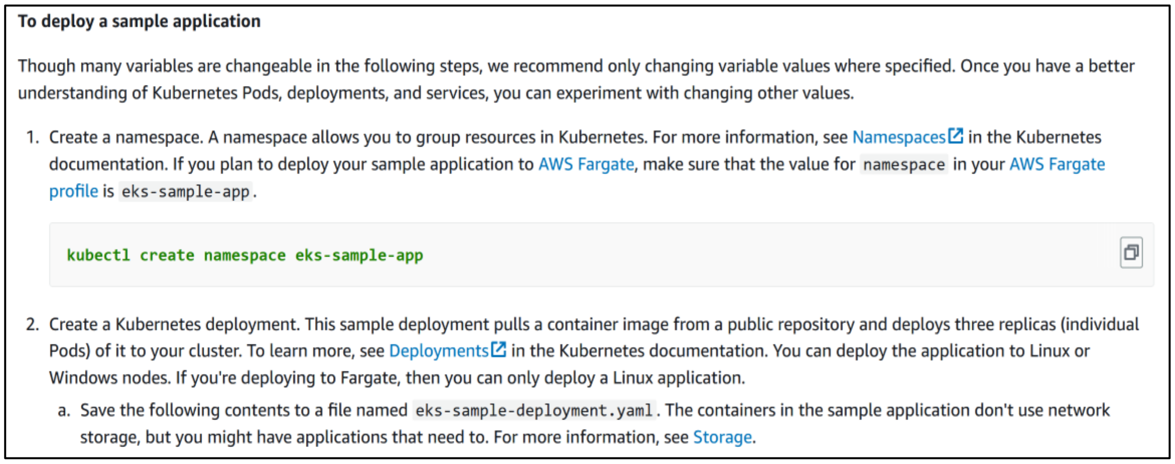
A: Amazon EKS works by provisioning (starting) and managing the Kubernetes control plane and worker nodes for you. At a high level, Kubernetes consists of two major components: a cluster of 'worker nodes' running your containers, and the control plane managing when and where containers are started on your cluster while monitoring their status.

Without Amazon EKS, you have to run both the Kubernetes control plane and the cluster of worker nodes yourself. With Amazon EKS, you provision your worker nodes using a single command in the EKS console, command-line interface (CLI), or API. AWS handles provisioning, scaling, and managing the Kubernetes control plane in a highly available and secure configuration. This removes a significant operational burden and allows you to focus on building applications instead of managing AWS infrastructure.

Q: Which operating systems does Amazon EKS support?

A: Amazon EKS supports Kubernetes-compatible Linux x86, ARM, and Windows Server operating system distributions. Amazon EKS provides optimized AMIs for Amazon Linux 2, Bottlerocket, and Windows Server 2019. At this time, there is no Amazon EKS optimized AMI for AL2023. EKS- optimized AMIs for other Linux distributions, such as Ubuntu, are available from their respective vendors.

[46]



To deploy a sample application

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes Pods, deployments, and services, you can experiment with changing other values.

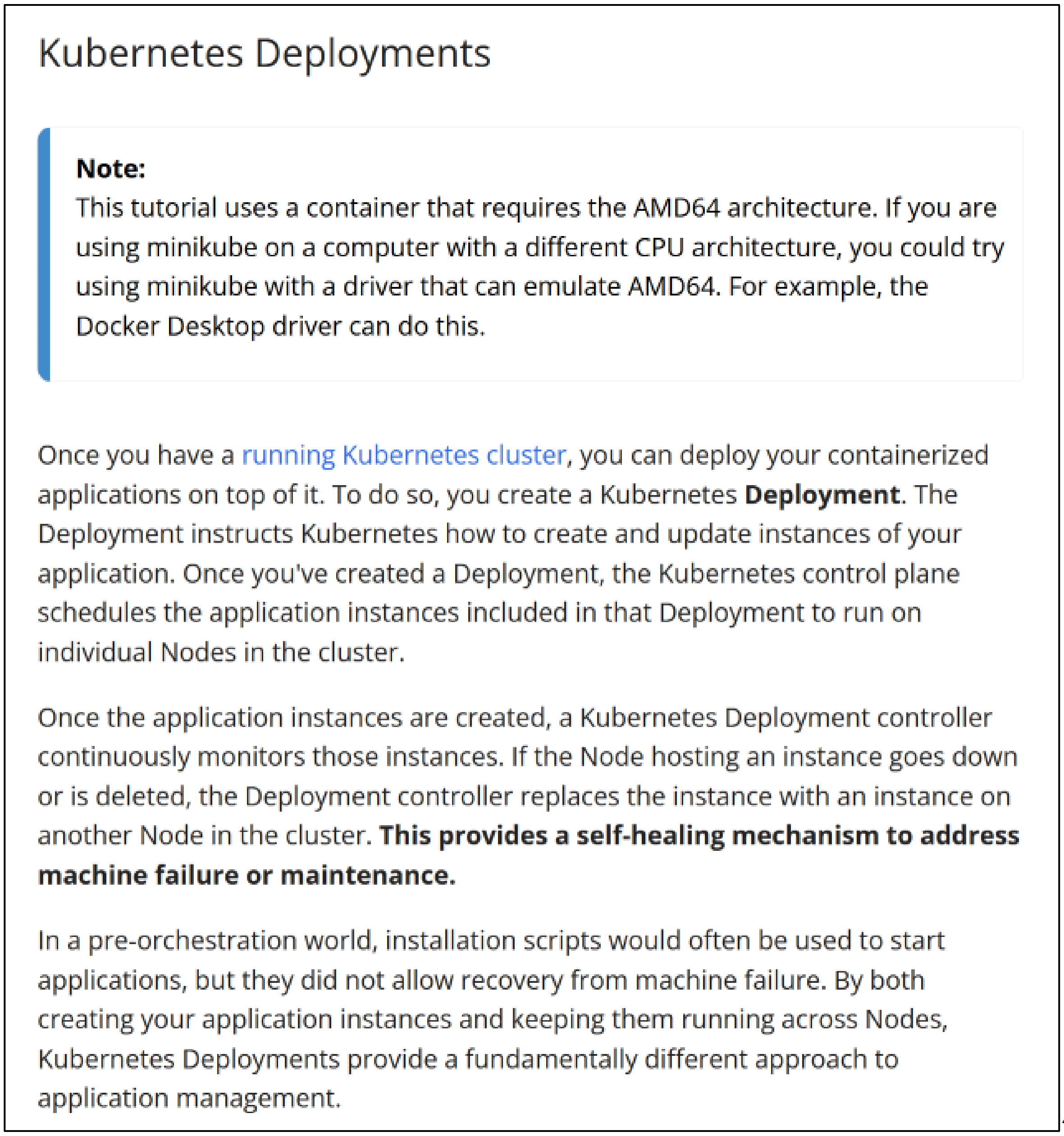
1. Create a namespace. A namespace allows you to group resources in Kubernetes. For more information, see Namespaces in the Kubernetes documentation. If you plan to deploy your sample application to AWS Fargate, make sure that the value for `namespace` in your AWS Fargate profile is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

2. Create a Kubernetes deployment. This sample deployment pulls a container image from a public repository and deploys three replicas (individual Pods) of it to your cluster. To learn more, see Deployments in the Kubernetes documentation. You can deploy the application to Linux or Windows nodes. If you're deploying to Fargate, then you can only deploy a Linux application.
   a. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see Storage.

[47]

[46] https://aws.amazon.com/eks/faqs/
[47] https://docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html



Kubernetes Deployments

Note:
This tutorial uses a container that requires the AMD64 architecture. If you are using minikube on a computer with a different CPU architecture, you could try using minikube with a driver that can emulate AMD64. For example, the Docker Desktop driver can do this.

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment**. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

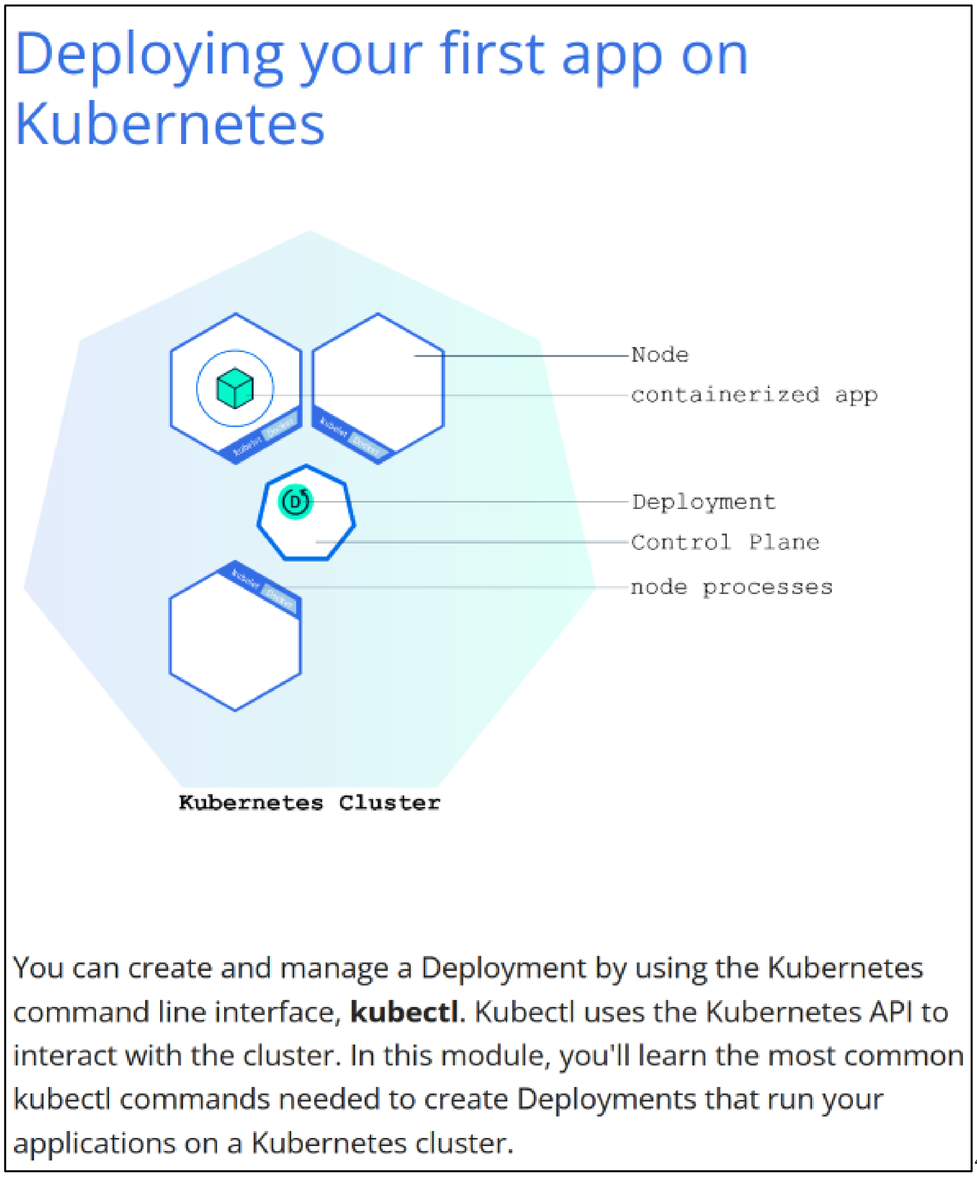
Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**

In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to application management.

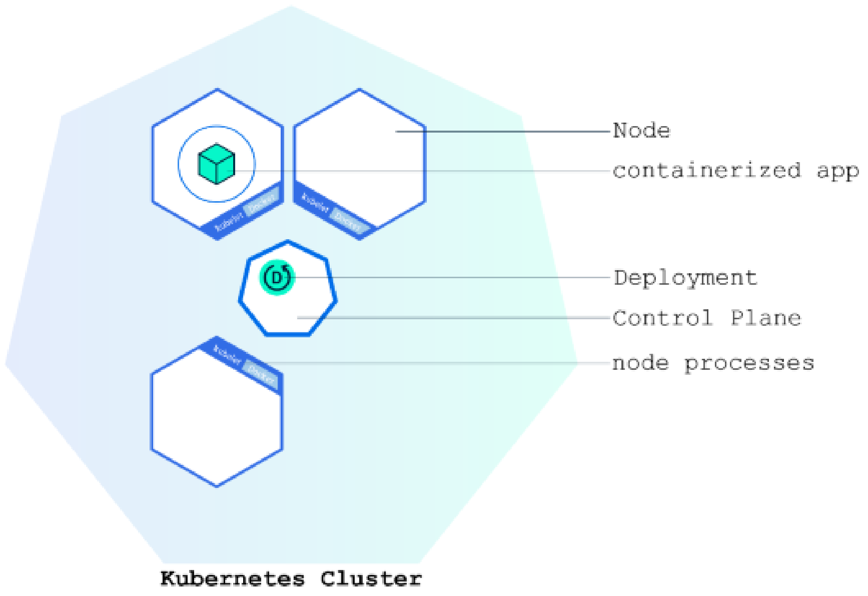
[48]

61. For example, the network device and programmable cloud device, as discussed above, can each facilitate deployment of the plurality of network applications (*e.g.*, across nodes, node processes, and control planes).

[48] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

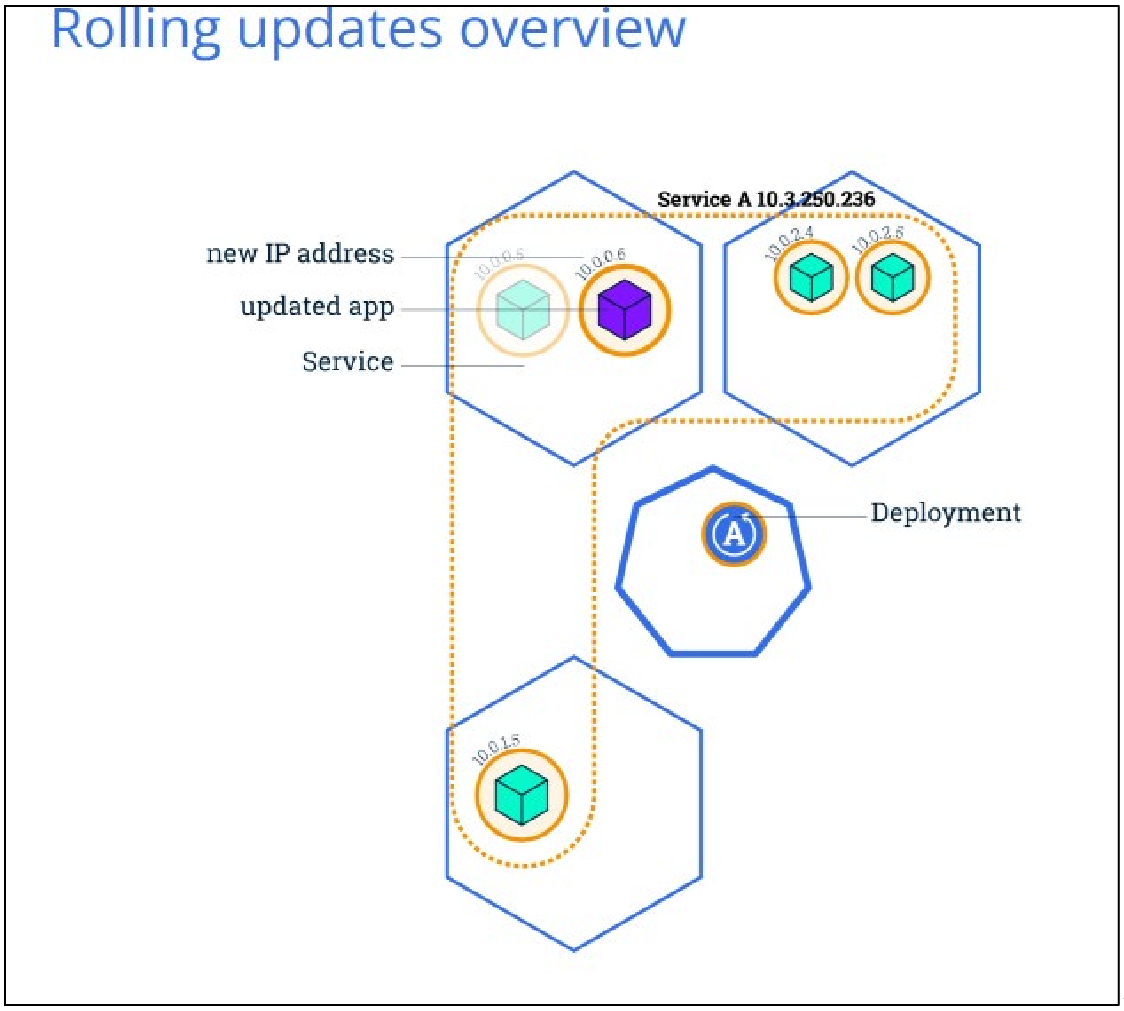


Deploying your first app on Kubernetes



You can create and manage a Deployment by using the Kubernetes command line interface, **kubectl**. Kubectl uses the Kubernetes API to interact with the cluster. In this module, you'll learn the most common kubectl commands needed to create Deployments that run your applications on a Kubernetes cluster.[49]

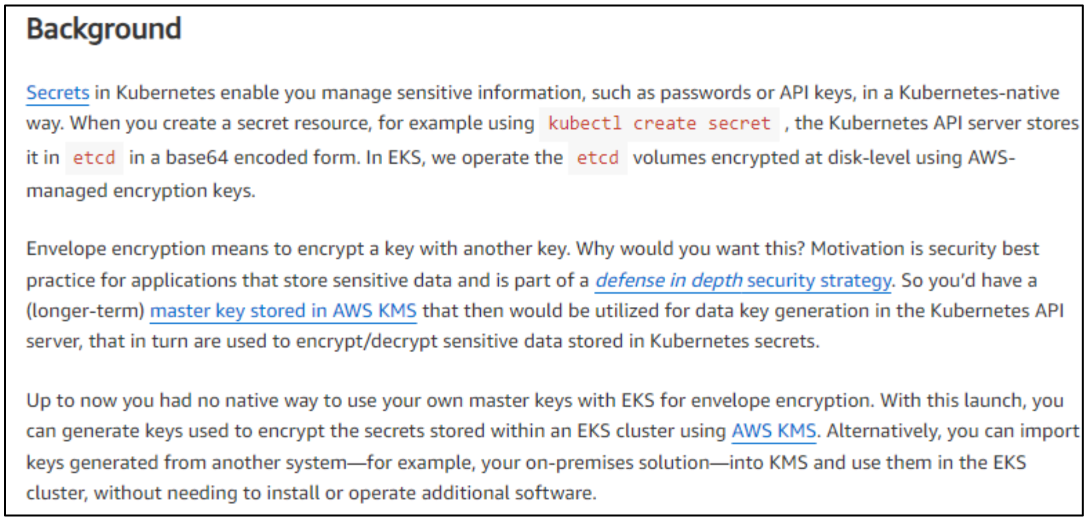
[49] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/



[50]

62. Amazon EKS Web Services comprise a programmable network device and a programmable cloud device, wherein the programmable network device verifies authenticity of the upgrades to the plurality of first network applications and the programmable cloud device verifies authenticity of the upgrades to the plurality of second network applications and wherein the verification is based on unique security keys associated with each of the plurality of first and second network applications. For example, the programmable network device and the programmable cloud device can verify the authenticity to the upgrades to both sets of network applications based on unique security keys.

[50] https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

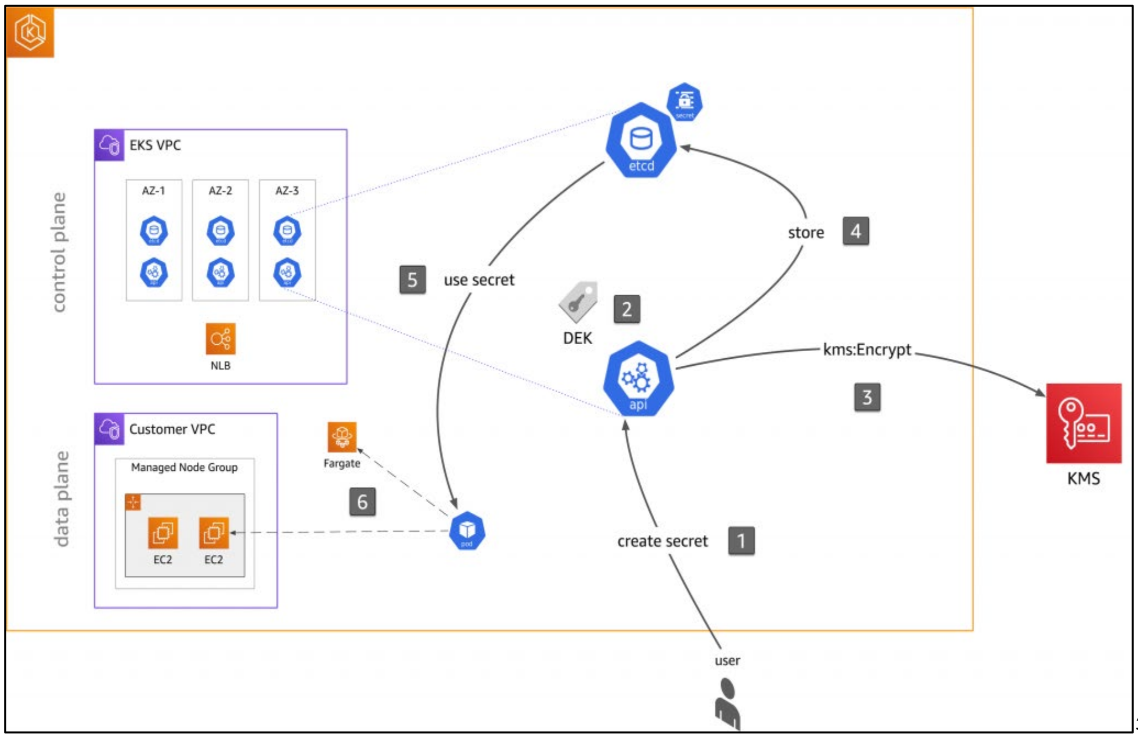


Background

Secrets in Kubernetes enable you manage sensitive information, such as passwords or API keys, in a Kubernetes-native way. When you create a secret resource, for example using `kubectl create secret` , the Kubernetes API server stores it in `etcd` in a base64 encoded form. In EKS, we operate the `etcd` volumes encrypted at disk-level using AWS-managed encryption keys.

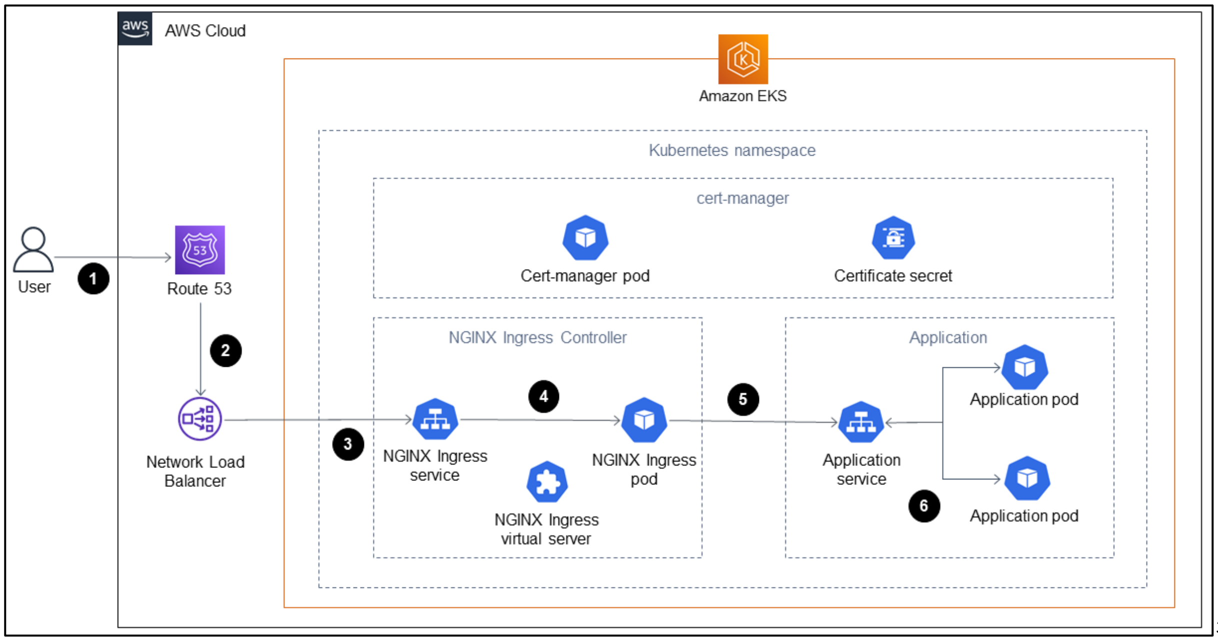
Envelope encryption means to encrypt a key with another key. Why would you want this? Motivation is security best practice for applications that store sensitive data and is part of a *defense in depth* security strategy. So you'd have a (longer-term) master key stored in AWS KMS that then would be utilized for data key generation in the Kubernetes API server, that in turn are used to encrypt/decrypt sensitive data stored in Kubernetes secrets.

Up to now you had no native way to use your own master keys with EKS for envelope encryption. With this launch, you can generate keys used to encrypt the secrets stored within an EKS cluster using AWS KMS. Alternatively, you can import keys generated from another system—for example, your on-premises solution—into KMS and use them in the EKS cluster, without needing to install or operate additional software.

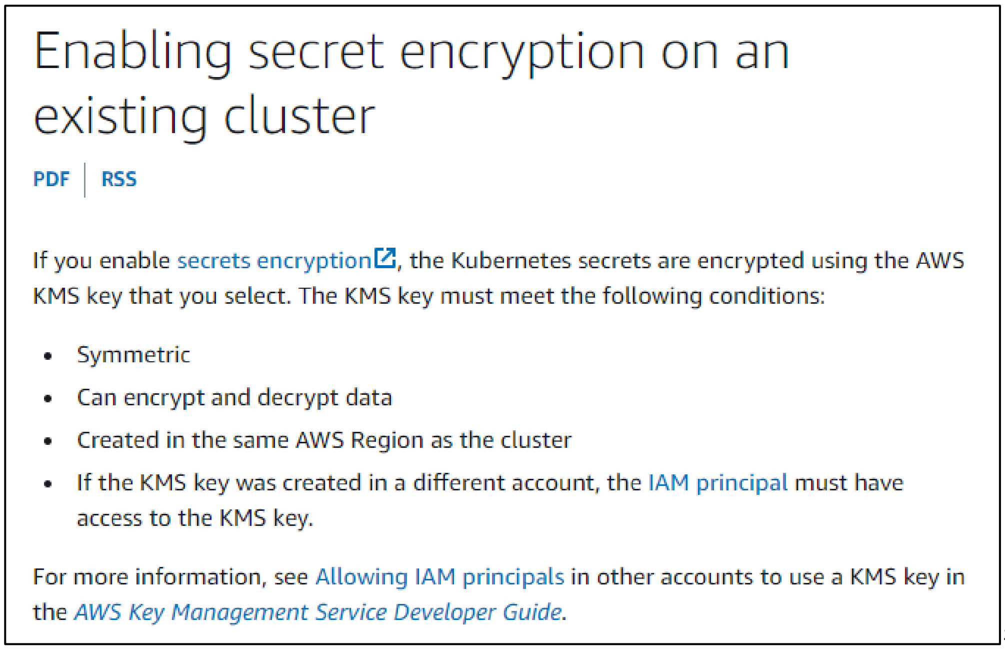


[51]

[51] https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/



[52]



Enabling secret encryption on an existing cluster

PDF | RSS

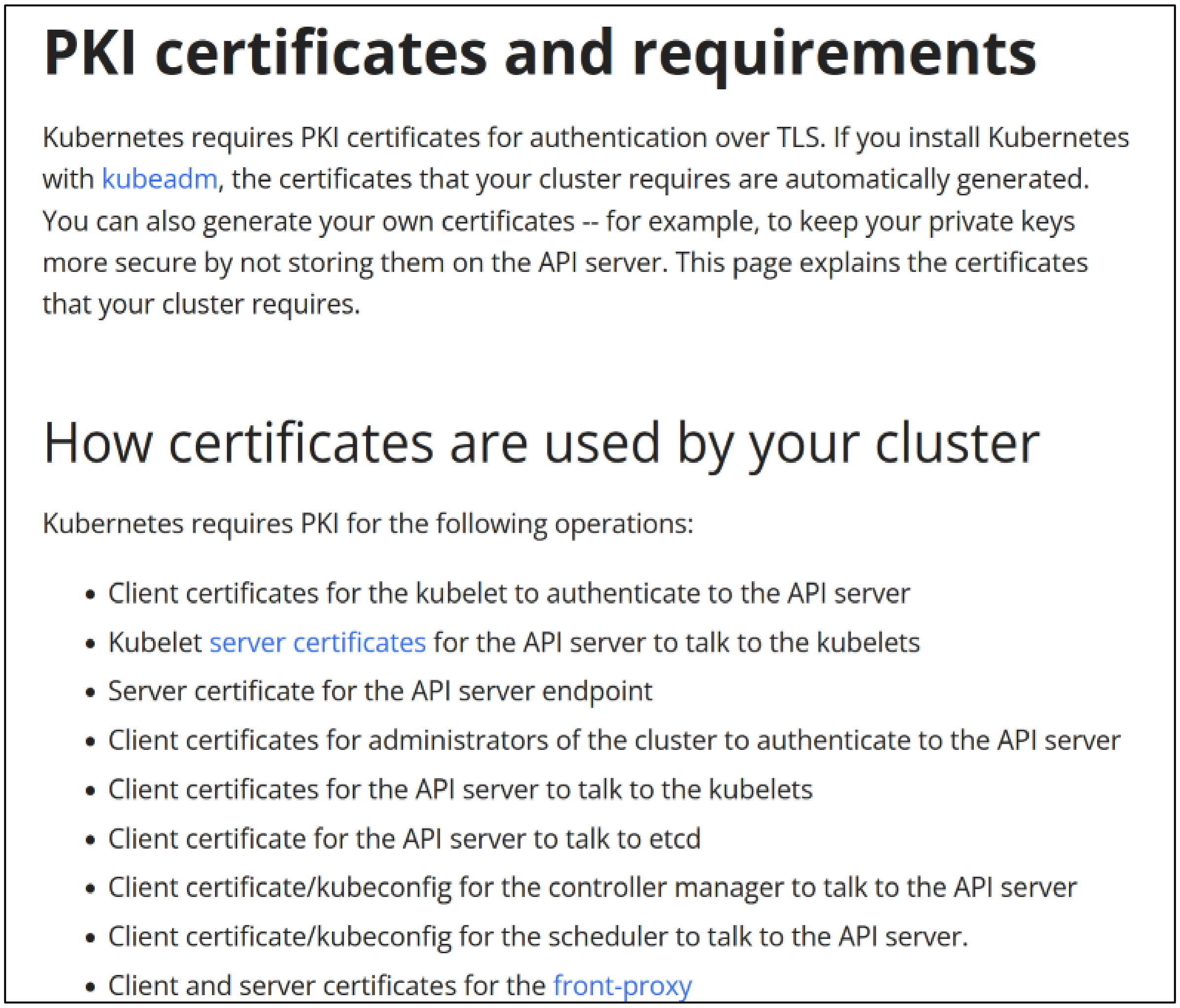
If you enable secrets encryption, the Kubernetes secrets are encrypted using the AWS KMS key that you select. The KMS key must meet the following conditions:

- Symmetric
- Can encrypt and decrypt data
- Created in the same AWS Region as the cluster
- If the KMS key was created in a different account, the IAM principal must have access to the KMS key.

For more information, see Allowing IAM principals in other accounts to use a KMS key in the *AWS Key Management Service Developer Guide*.

[53]

[52] https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/set-up-end-to-end-encryption-for-applications-on-amazon-eks-using-cert-manager-and-let-s-encrypt.html
[53] https://docs.aws.amazon.com/eks/latest/userguide/enable-kms.html



PKI certificates and requirements

Kubernetes requires PKI certificates for authentication over TLS. If you install Kubernetes with kubeadm, the certificates that your cluster requires are automatically generated. You can also generate your own certificates -- for example, to keep your private keys more secure by not storing them on the API server. This page explains the certificates that your cluster requires.

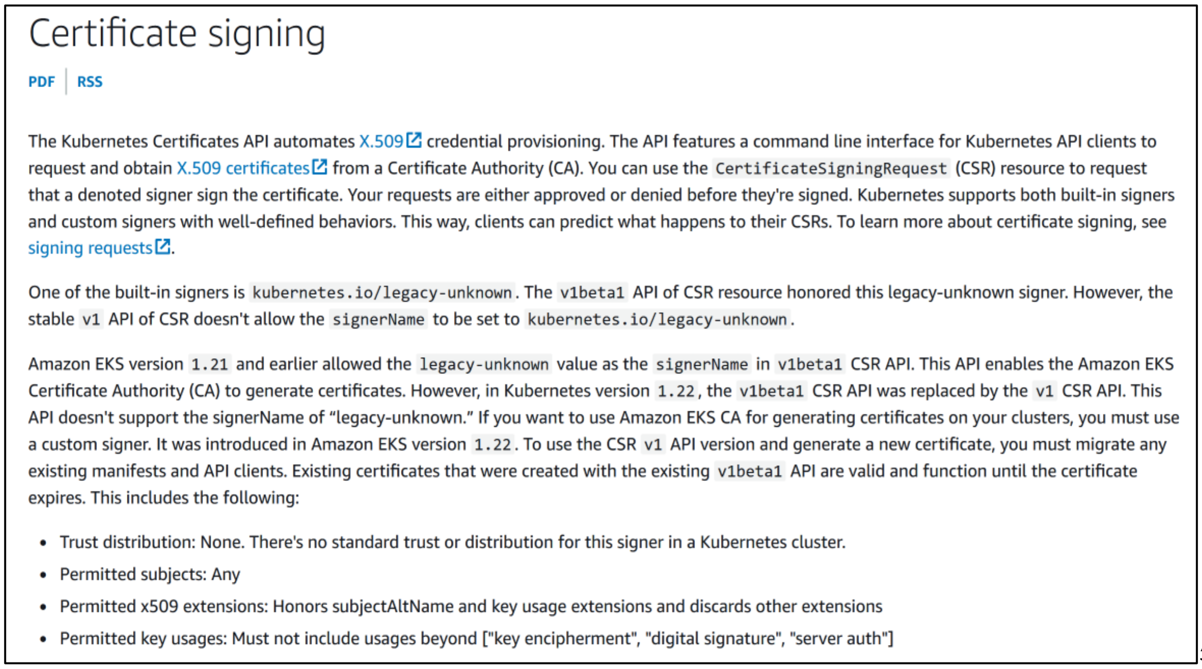
How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Kubelet server certificates for the API server to talk to the kubelets
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

[54]

[54] https://kubernetes.io/docs/setup/best-practices/certificates/



Certificate signing

PDF | RSS

The Kubernetes Certificates API automates X.509 credential provisioning. The API features a command line interface for Kubernetes API clients to request and obtain X.509 certificates from a Certificate Authority (CA). You can use the `CertificateSigningRequest` (CSR) resource to request that a denoted signer sign the certificate. Your requests are either approved or denied before they're signed. Kubernetes supports both built-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see signing requests.

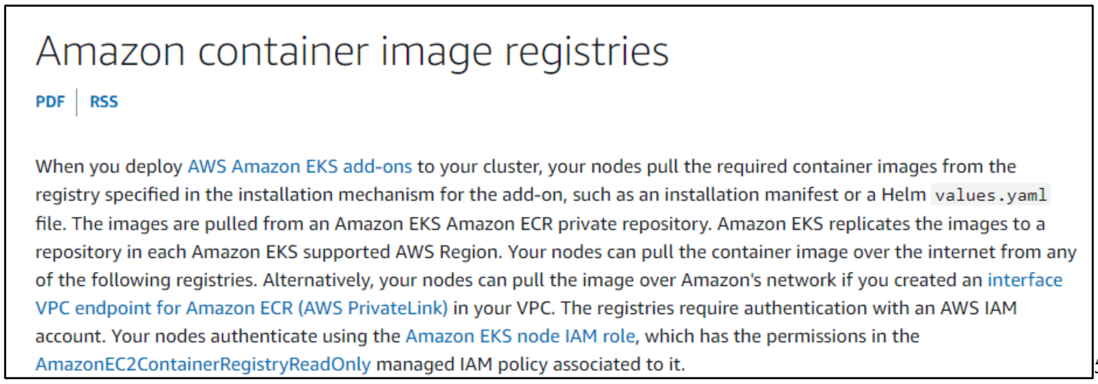
One of the built-in signers is `kubernetes.io/legacy-unknown`. The `v1beta1` API of CSR resource honored this legacy-unknown signer. However, the stable `v1` API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

Amazon EKS version `1.21` and earlier allowed the `legacy-unknown` value as the `signerName` in `v1beta1` CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version `1.22`, the `v1beta1` CSR API was replaced by the `v1` CSR API. This API doesn't support the signerName of "legacy-unknown." If you want to use Amazon EKS CA for generating certificates on your clusters, you must use a custom signer. It was introduced in Amazon EKS version `1.22`. To use the CSR `v1` API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the existing `v1beta1` API are valid and function until the certificate expires. This includes the following:

- Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
- Permitted subjects: Any
- Permitted x509 extensions: Honors subjectAltName and key usage extensions and discards other extensions
- Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]

[55]

63. Amazon EKS Web Services comprises an application repository (*e.g.*, a container) storing distributed applications which have been tested for installation in the programmable network device and programmable cloud device.



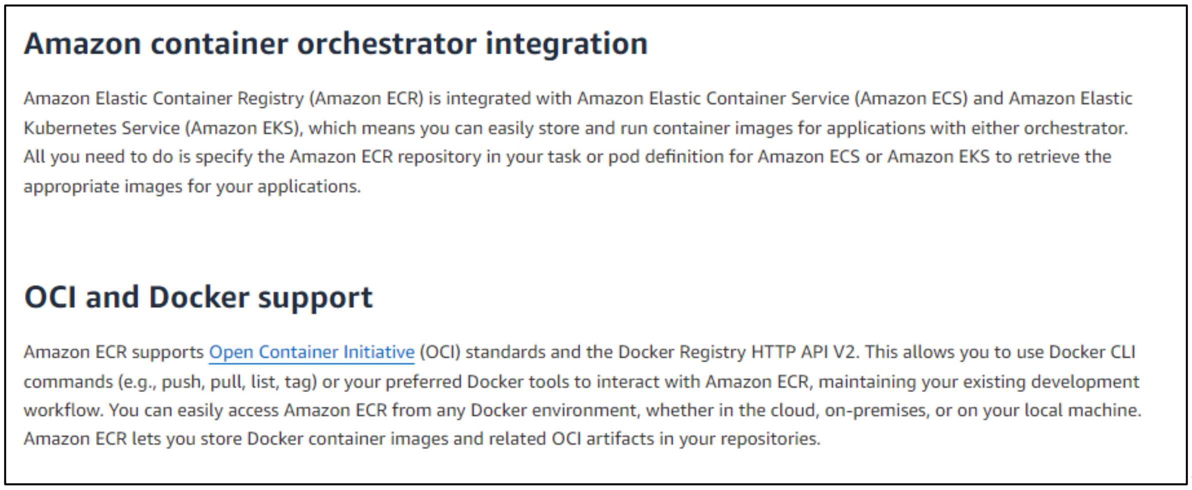
Amazon container image registries

PDF | RSS

When you deploy AWS Amazon EKS add-ons to your cluster, your nodes pull the required container images from the registry specified in the installation mechanism for the add-on, such as an installation manifest or a Helm `values.yaml` file. The images are pulled from an Amazon EKS Amazon ECR private repository. Amazon EKS replicates the images to a repository in each Amazon EKS supported AWS Region. Your nodes can pull the container image over the internet from any of the following registries. Alternatively, your nodes can pull the image over Amazon's network if you created an interface VPC endpoint for Amazon ECR (AWS PrivateLink) in your VPC. The registries require authentication with an AWS IAM account. Your nodes authenticate using the Amazon EKS node IAM role, which has the permissions in the AmazonEC2ContainerRegistryReadOnly managed IAM policy associated to it.

[56]

[55] https://docs.aws.amazon.com/eks/latest/userguide/cert-signing.html
[56] https://docs.aws.amazon.com/eks/latest/userguide/add-ons-images.html



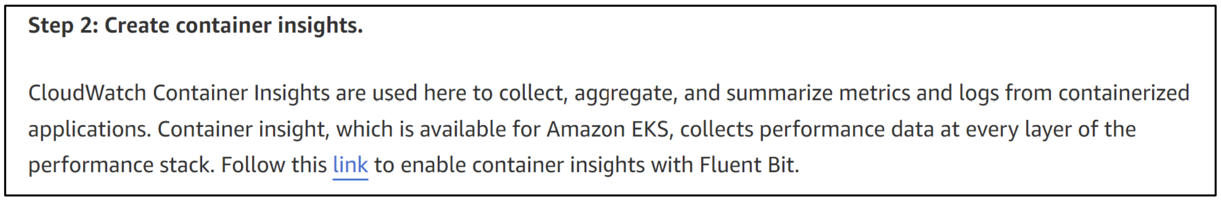
Amazon container orchestrator integration

Amazon Elastic Container Registry (Amazon ECR) is integrated with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS), which means you can easily store and run container images for applications with either orchestrator. All you need to do is specify the Amazon ECR repository in your task or pod definition for Amazon ECS or Amazon EKS to retrieve the appropriate images for your applications.

OCI and Docker support

Amazon ECR supports Open Container Initiative (OCI) standards and the Docker Registry HTTP API V2. This allows you to use Docker CLI commands (e.g., push, pull, list, tag) or your preferred Docker tools to interact with Amazon ECR, maintaining your existing development workflow. You can easily access Amazon ECR from any Docker environment, whether in the cloud, on-premises, or on your local machine. Amazon ECR lets you store Docker container images and related OCI artifacts in your repositories.

[57]



Step 2: Create container insights.

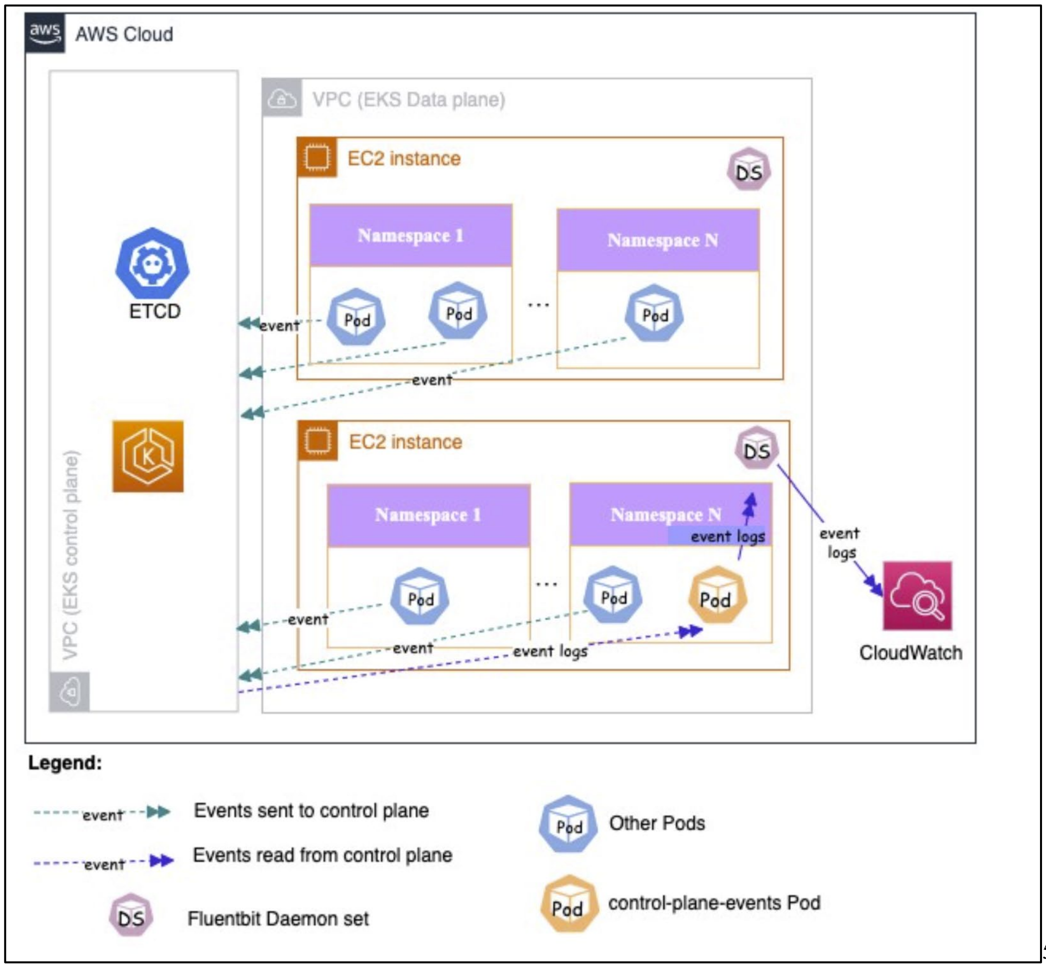
CloudWatch Container Insights are used here to collect, aggregate, and summarize metrics and logs from containerized applications. Container insight, which is available for Amazon EKS, collects performance data at every layer of the performance stack. Follow this link to enable container insights with Fluent Bit.

[58]

64. Amazon EKS Web Services comprise an application management portal (*e.g.*, the control plane, including associated dashboard) coupled to the programmable network device, programmable cloud device, and the application repository, wherein the application management portal manages usage of the distributed applications on the programmable network device and programmable cloud device.

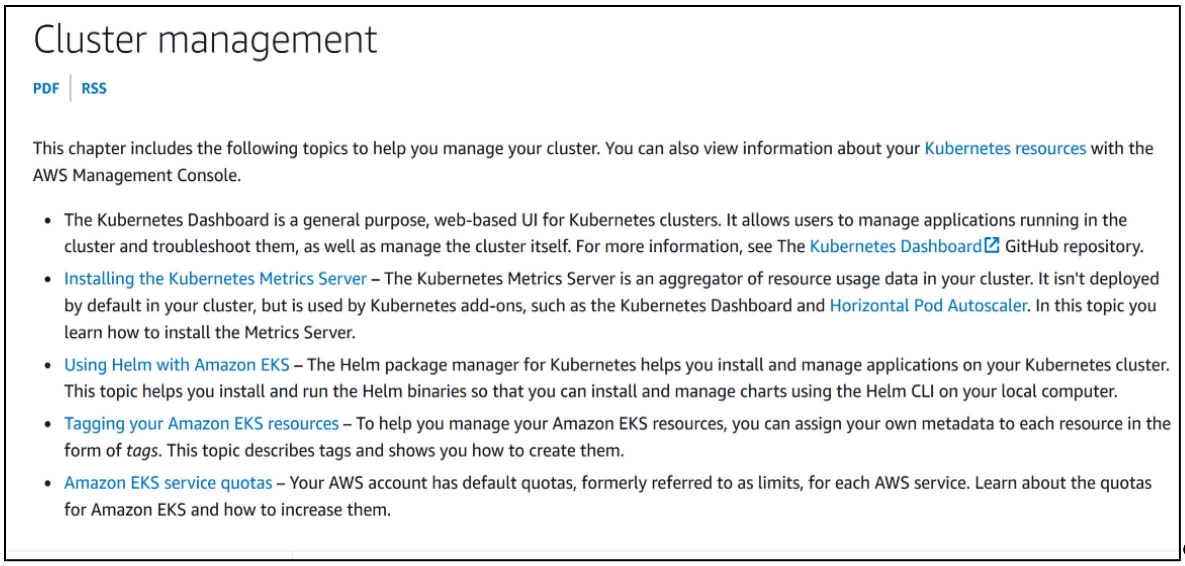
[57] https://aws.amazon.com/ecr/

[58] https://aws.amazon.com/blogs/containers/managing-kubernetes-control-plane-events-in-amazon-eks/



[59]

59 https://aws.amazon.com/blogs/containers/managing-kubernetes-control-plane-events-in-amazon-eks/



Cluster management

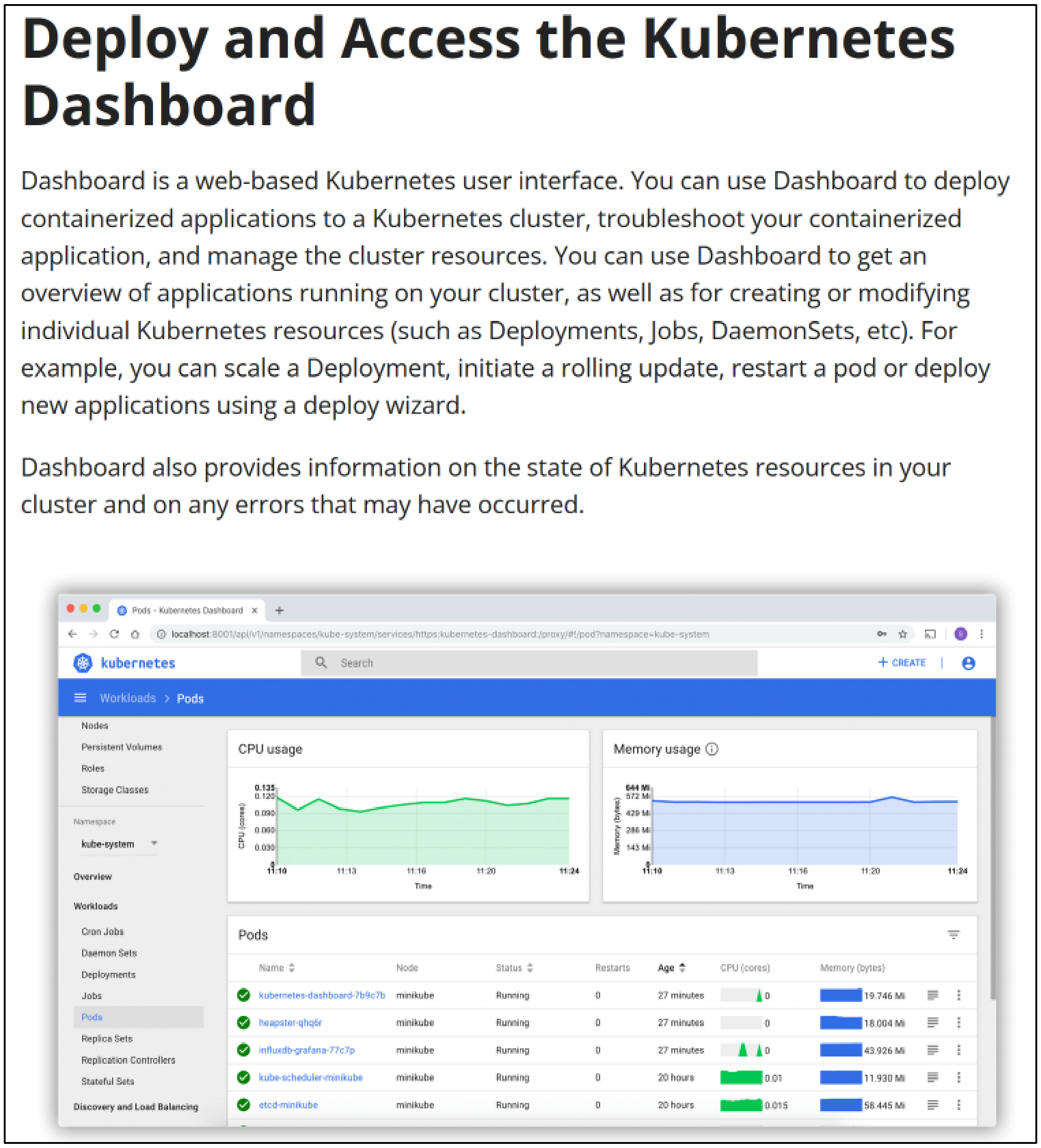
PDF | RSS

This chapter includes the following topics to help you manage your cluster. You can also view information about your Kubernetes resources with the AWS Management Console.

- The Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself. For more information, see The Kubernetes Dashboard GitHub repository.
- Installing the Kubernetes Metrics Server – The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster. It isn't deployed by default in your cluster, but is used by Kubernetes add-ons, such as the Kubernetes Dashboard and Horizontal Pod Autoscaler. In this topic you learn how to install the Metrics Server.
- Using Helm with Amazon EKS – The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local computer.
- Tagging your Amazon EKS resources – To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.
- Amazon EKS service quotas – Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Learn about the quotas for Amazon EKS and how to increase them.

[60]

[60] https://docs.aws.amazon.com/eks/latest/userguide/eks-managing.html



Deploy and Access the Kubernetes Dashboard

Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use Dashboard to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). For example, you can scale a Deployment, initiate a rolling update, restart a pod or deploy new applications using a deploy wizard.

Dashboard also provides information on the state of Kubernetes resources in your cluster and on any errors that may have occurred.

Pods

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
kubernetes-dashboard-7b9c7b	minikube	Running	0	27 minutes	0	19.746 Mi
heapster-qhq6r	minikube	Running	0	27 minutes	0	18.004 Mi
influxdb-grafana-77c7p	minikube	Running	0	27 minutes	0	43.926 Mi
kube-scheduler-minikube	minikube	Running	0	20 hours	0.01	11.930 Mi
etcd-minikube	minikube	Running	0	20 hours	0.015	58.445 Mi

[61]

65. Defendants have and continue to indirectly infringe one or more claims of the ’095 Patent by knowingly and intentionally inducing others, including Amazon customers and end-users, to directly infringe, either literally or under the doctrine of equivalents, by making, using, offering to sell, selling, and/or importing into the United States products that include the infringing

[61] https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

technology.

66. Defendants, with knowledge[62] that these products, or the use thereof, infringe the ’095 Patent at least as of the date of this Complaint, knowingly and intentionally induced, and continue to knowingly and intentionally induce direct infringement of the ’095 Patent by providing these products to end-users for use in an infringing manner. Alternatively, on information and belief, Defendants have adopted a policy of not reviewing the patents of others, including specifically those related to Defendants’ specific industry, thereby remaining willfully blind to the Patent-in-Suit at least as early as the issuance of the Patents-in-Suit.

67. Defendants have induced infringement by others, including end-users, with the intent to cause infringing acts by others or, in the alternative, with the belief that there was a high probability that others, including end-users, infringe the ’095 Patent, but while remaining willfully blind to the infringement. Defendants have and continue to induce infringement by its customers and end-users by supplying them with instructions on how to operate the infringing technology in an infringing manner, while also making publicly available information on the infringing technology via Defendants’ website, product literature and packaging, and other publications.

68. Plaintiff has suffered damages as a result of Defendants’ direct and indirect infringement of the ’095 Patent in an amount to be proven at trial.

69. Plaintiff has suffered, and will continue to suffer, irreparable harm as a result of Defendants’ infringement of the ’095 Patent, for which there is no adequate remedy at law, unless Defendants’ infringement is enjoined by this Court.

[62] The ’095 Patent Family was cited as relevant prior art against Amazon Technologies, Inc.’s U.S. Patent Application No. 16/129,632, which was filed on September 12, 2018, and issued as U.S. Patent No. 11,108,687 on August 31, 2021.

COUNT III
(Infringement of the ’871 Patent)

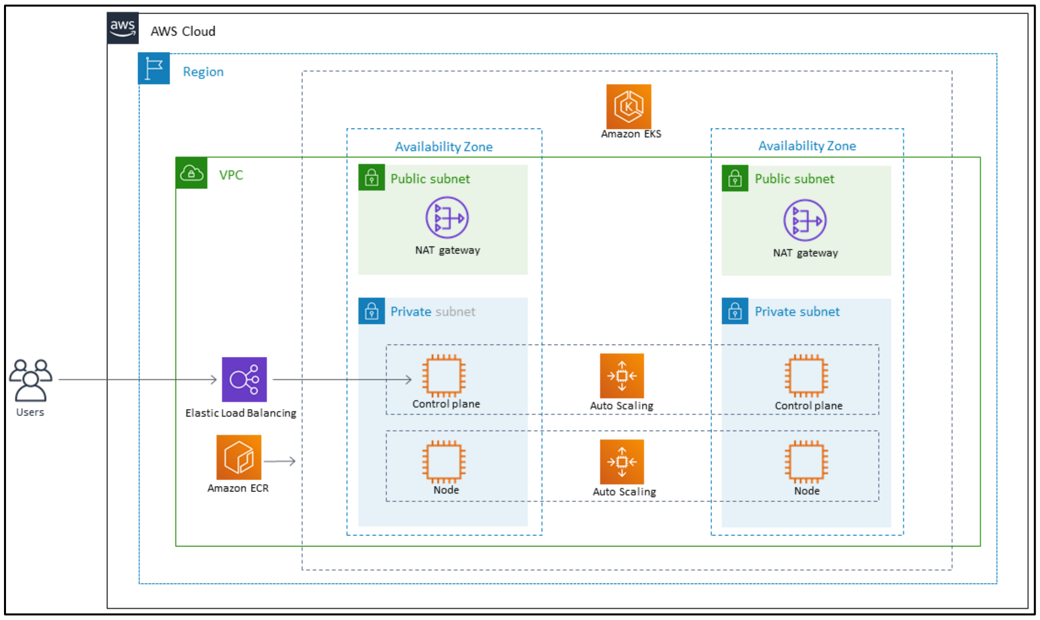
70. Paragraphs 1 through 29 are incorporated by reference as if fully set forth herein.

71. Plaintiff has not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the ’871Patent.

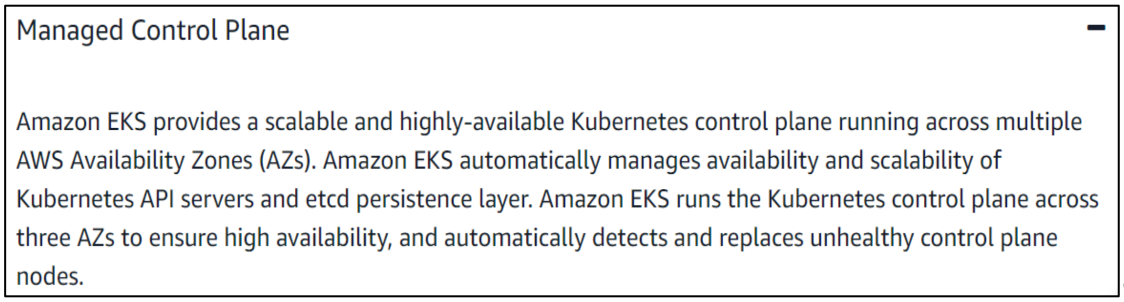
72. Defendants have and continue to directly infringe the ’871 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, offering to sell, selling, and/or importing into the United States products that satisfy each and every limitation of one or more claims of the ’871 Patent. Such infringing products include Amazon EKS Web Services.

73. For example, Defendants have and continue to directly infringe at least claim 1 of the ’871 Patent by making, using, offering to sell, selling, and/or importing into the United States products that include and/or support Amazon EKS Web Services.

74. Amazon EKS Web Services comprise a system for processing data packets in a network.



75. Amazon EKS Web Services comprise a programmable network device hosting a plurality of first network applications.

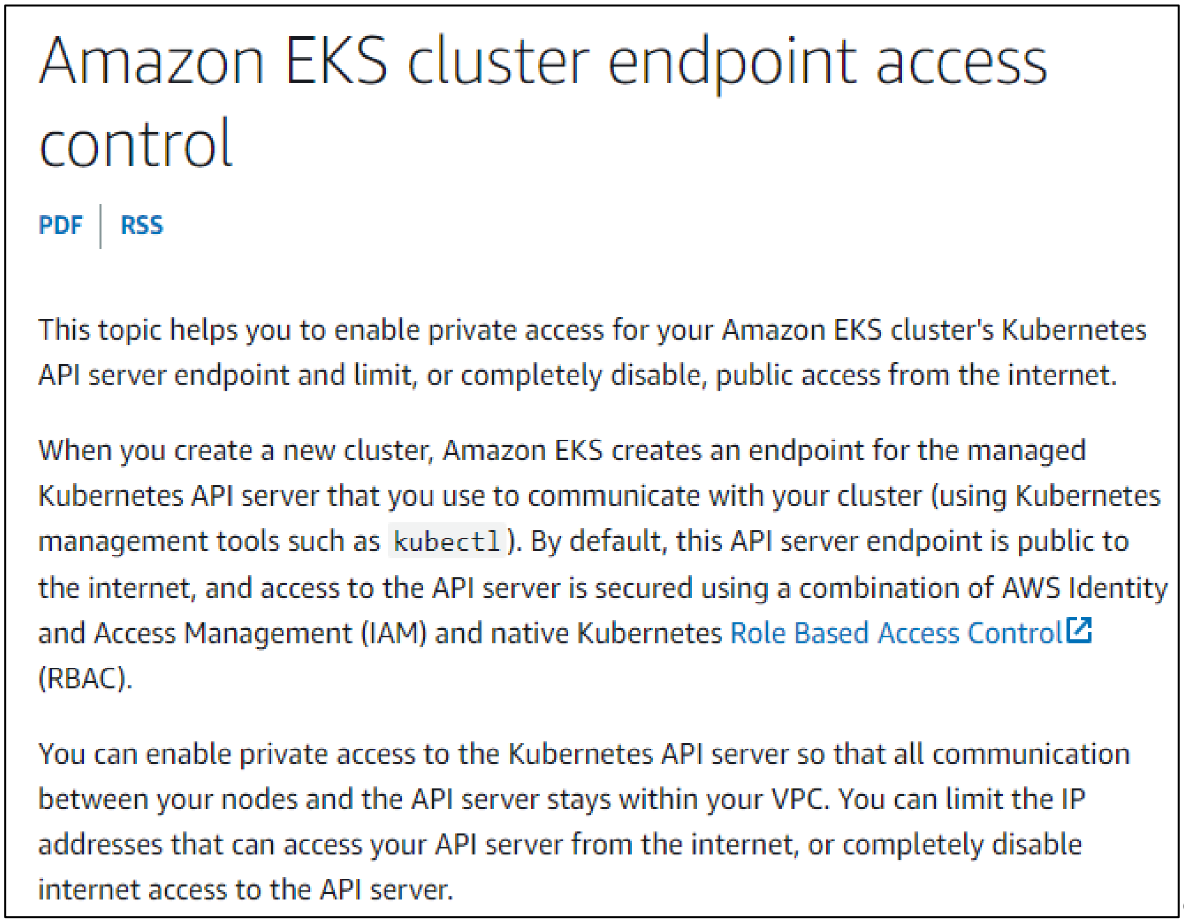

Managed Control Plane

Amazon EKS provides a scalable and highly-available Kubernetes control plane running across multiple AWS Availability Zones (AZs). Amazon EKS automatically manages availability and scalability of Kubernetes API servers and etcd persistence layer. Amazon EKS runs the Kubernetes control plane across three AZs to ensure high availability, and automatically detects and replaces unhealthy control plane nodes.

[63]

76. For example, the programmable network device can be the Endpoint for the managed Kubernetes API server.

[63] https://aws.amazon.com/eks/features/#:~:text=Amazon%20EKS%20provides%20a%20scalable,servers%20and%20etcd%20persistence%20layer



Amazon EKS cluster endpoint access control

PDF | RSS

This topic helps you to enable private access for your Amazon EKS cluster's Kubernetes API server endpoint and limit, or completely disable, public access from the internet.

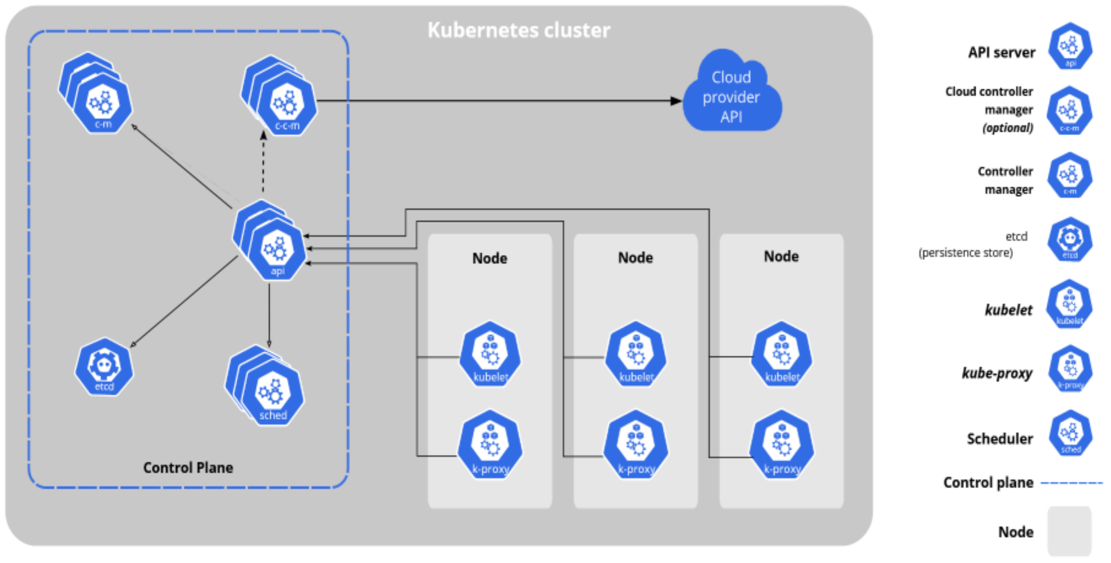
When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). By default, this API server endpoint is public to the internet, and access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC).

You can enable private access to the Kubernetes API server so that all communication between your nodes and the API server stays within your VPC. You can limit the IP addresses that can access your API server from the internet, or completely disable internet access to the API server.[64]

77. By way of further example, the programmable network device can be a kube-proxy and/or kubelet of a node within the first availability zone.

[64] https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

This document outlines the various components you need to have for a complete and working Kubernetes cluster.

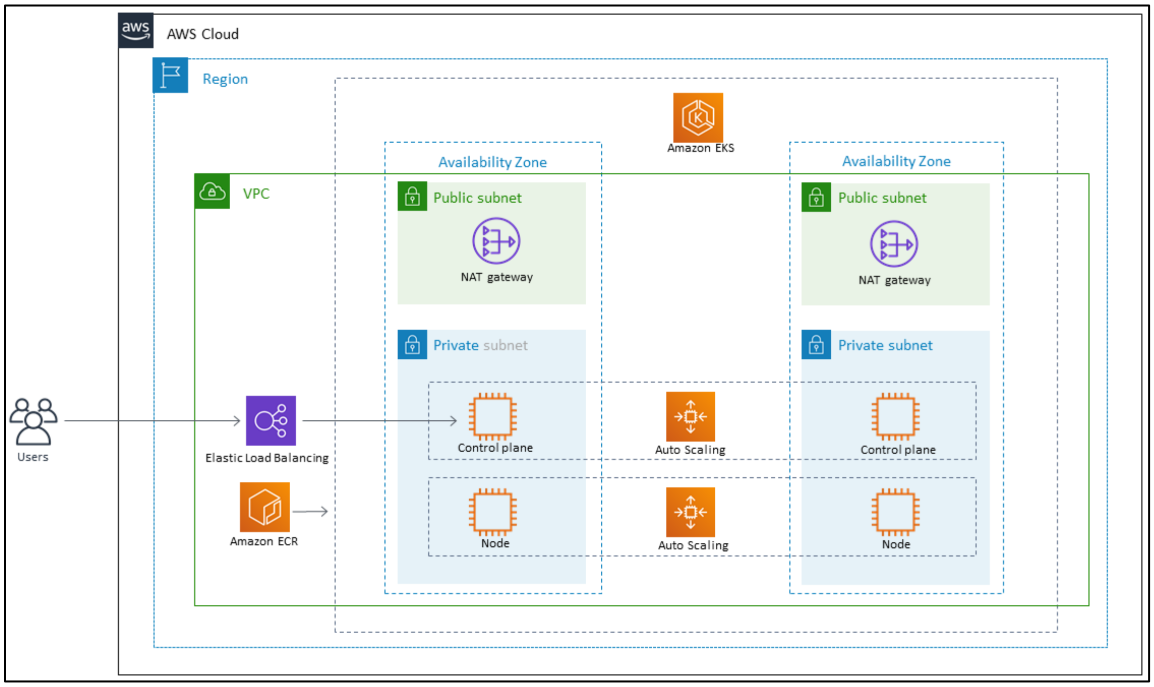


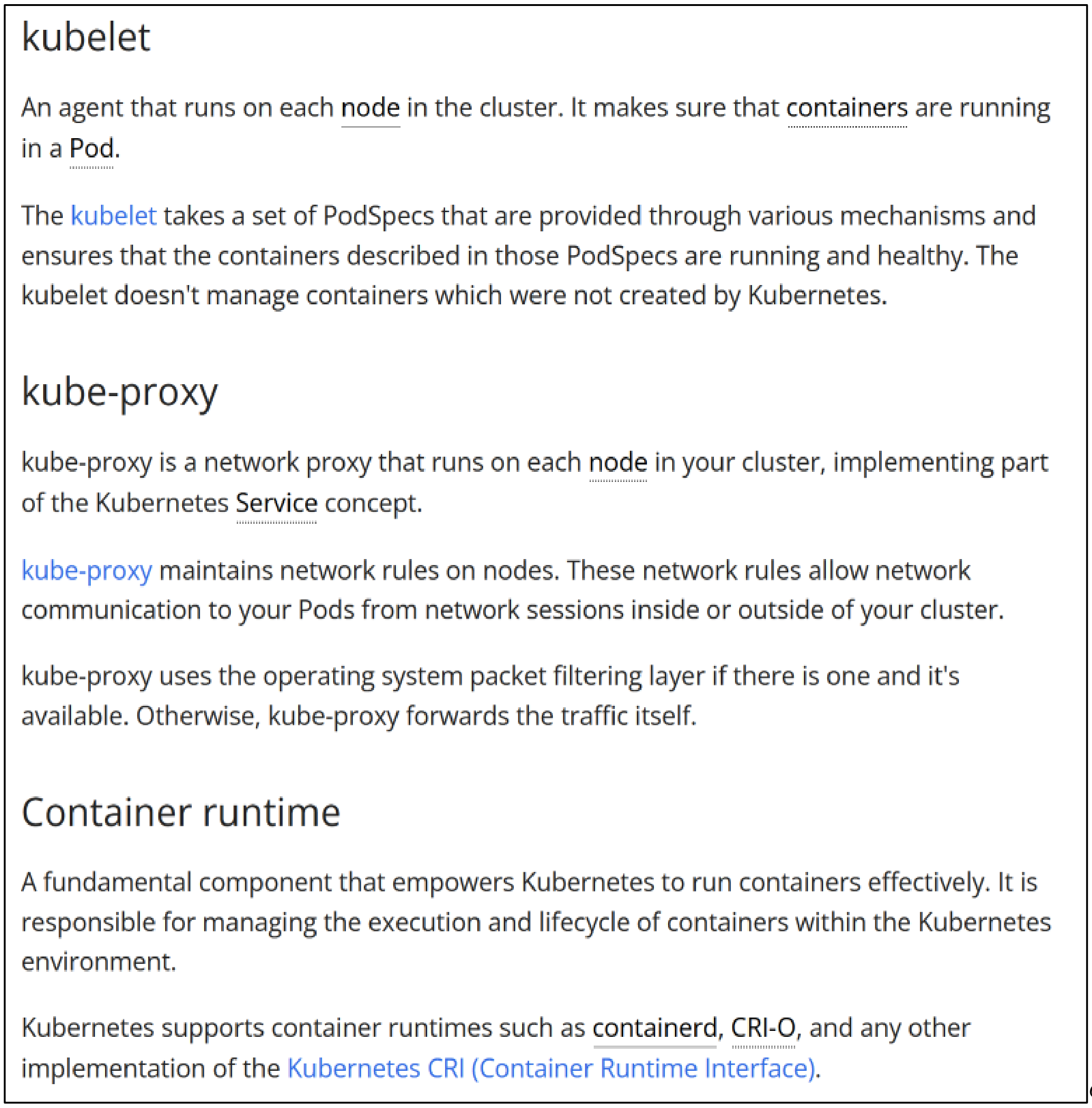
The components of a Kubernetes cluster

[65]

78. Amazon EKS Web Services comprise a programmable cloud device adapted to host a plurality of cloud applications, wherein at least one of the plurality of first network applications in the programmable network device and at least one of the plurality of second network applications in the programmable cloud device are in secure communication with each other through a virtual fabric to form a distributed application. For example, the cloud device (*e.g.*, a node of the second EKS Availability Zone) can be the entire second EKS Availability Zone itself, and/or a Kubernetes container runtime of the node associated with the aforementioned kubelet and/or kube-proxy.

[65] https://kubernetes.io/docs/concepts/overview/components/


AWS Cloud
Region
Amazon EKS
Availability Zone
Availability Zone
VPC
Public subnet
NAT gateway
Public subnet
NAT gateway
Private subnet
Private subnet
Users
Elastic Load Balancing
Control plane
Auto Scaling
Control plane
Amazon ECR
Node
Auto Scaling
Node



kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

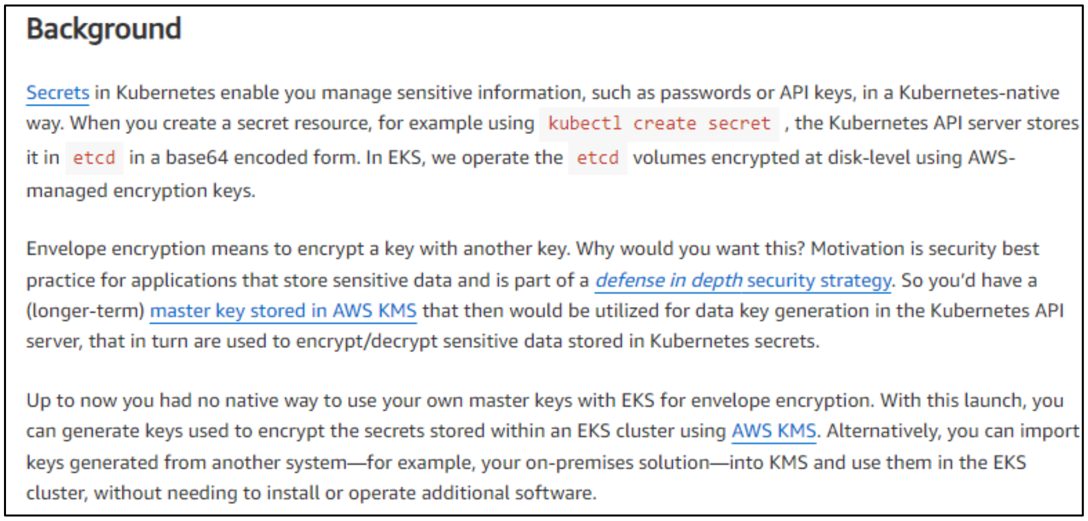
Container runtime

A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).[66]

79. By way of further example, the network applications in both the programmable network device and the programmable cloud device can be in secure communication with each other, forming a distributed application. The secure distributed application is evidenced by Amazon EKS's use of Secrets and end-to-end encryption.

[66] https://kubernetes.io/docs/concepts/architecture/

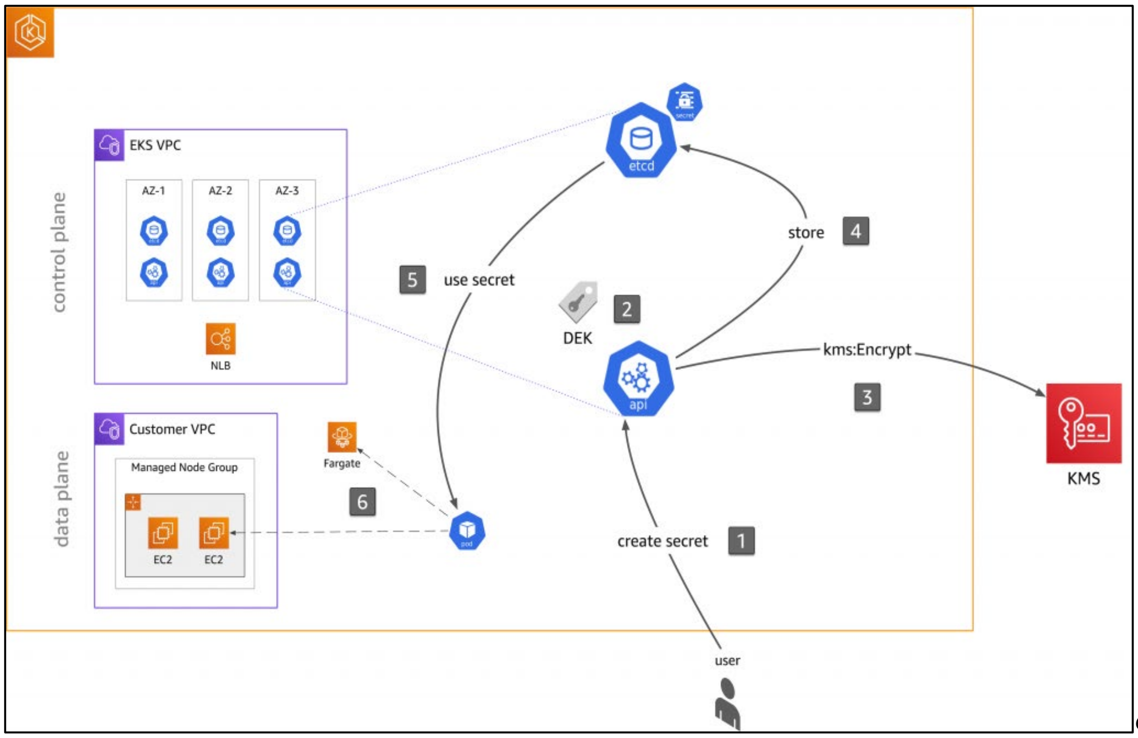


Background

Secrets in Kubernetes enable you manage sensitive information, such as passwords or API keys, in a Kubernetes-native way. When you create a secret resource, for example using `kubectl create secret` , the Kubernetes API server stores it in `etcd` in a base64 encoded form. In EKS, we operate the `etcd` volumes encrypted at disk-level using AWS-managed encryption keys.

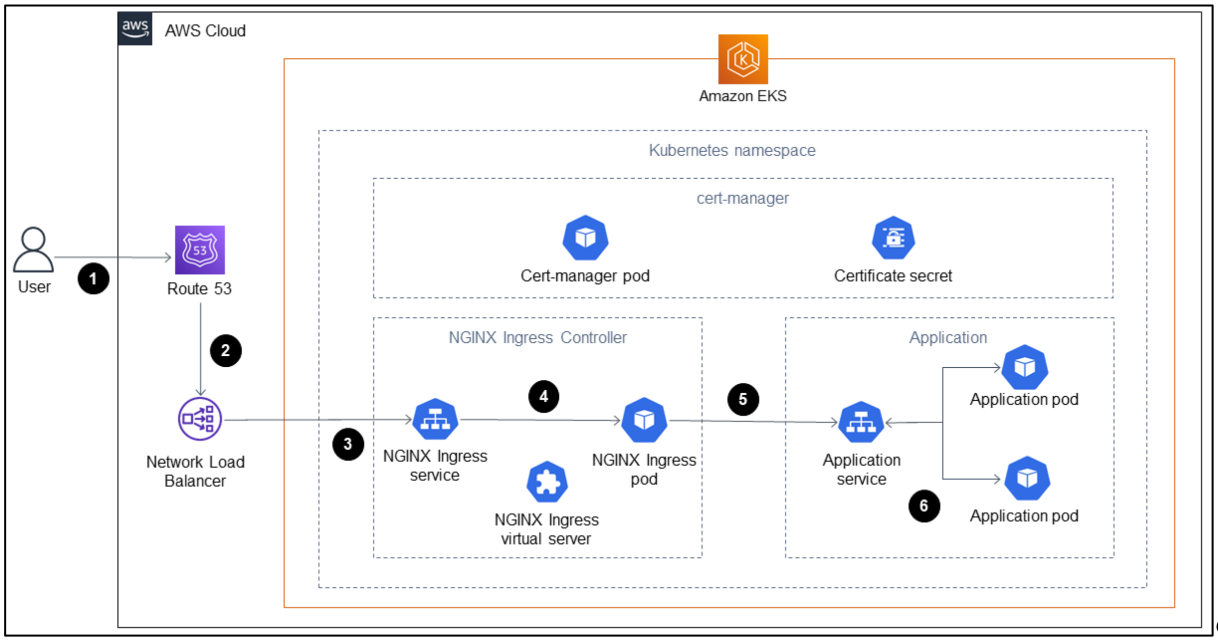
Envelope encryption means to encrypt a key with another key. Why would you want this? Motivation is security best practice for applications that store sensitive data and is part of a *defense in depth* security strategy. So you'd have a (longer-term) master key stored in AWS KMS that then would be utilized for data key generation in the Kubernetes API server, that in turn are used to encrypt/decrypt sensitive data stored in Kubernetes secrets.

Up to now you had no native way to use your own master keys with EKS for envelope encryption. With this launch, you can generate keys used to encrypt the secrets stored within an EKS cluster using AWS KMS. Alternatively, you can import keys generated from another system—for example, your on-premises solution—into KMS and use them in the EKS cluster, without needing to install or operate additional software.

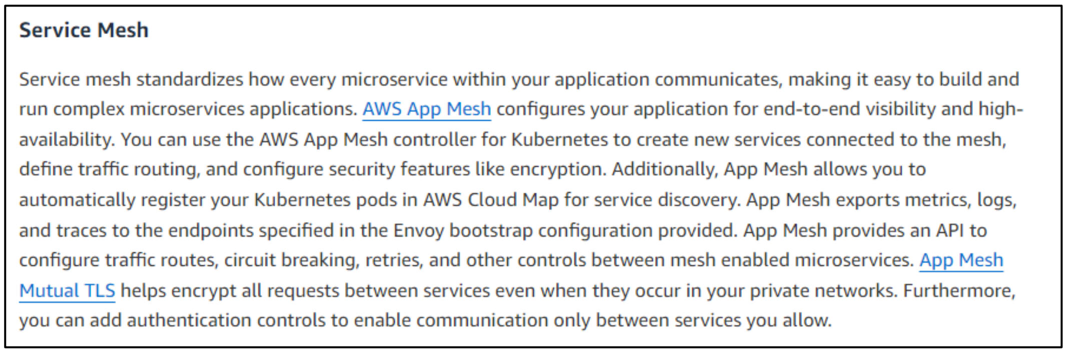


[67]

67 https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/



[68]

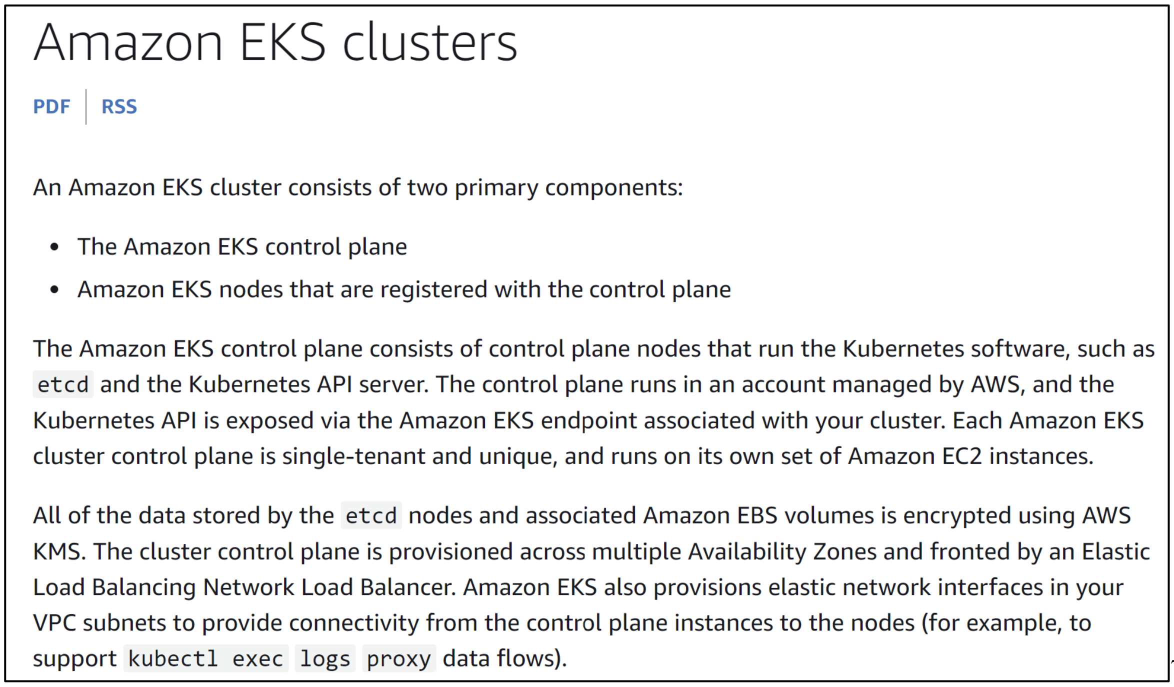


Service Mesh

Service mesh standardizes how every microservice within your application communicates, making it easy to build and run complex microservices applications. AWS App Mesh configures your application for end-to-end visibility and high-availability. You can use the AWS App Mesh controller for Kubernetes to create new services connected to the mesh, define traffic routing, and configure security features like encryption. Additionally, App Mesh allows you to automatically register your Kubernetes pods in AWS Cloud Map for service discovery. App Mesh exports metrics, logs, and traces to the endpoints specified in the Envoy bootstrap configuration provided. App Mesh provides an API to configure traffic routes, circuit breaking, retries, and other controls between mesh enabled microservices. App Mesh Mutual TLS helps encrypt all requests between services even when they occur in your private networks. Furthermore, you can add authentication controls to enable communication only between services you allow.

[69]

[68] https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/set-up-end-to-end-encryption-for-applications-on-amazon-eks-using-cert-manager-and-let-s-encrypt.html.
[69] https://aws.amazon.com/eks/features/



Amazon EKS clusters

PDF | RSS

An Amazon EKS cluster consists of two primary components:

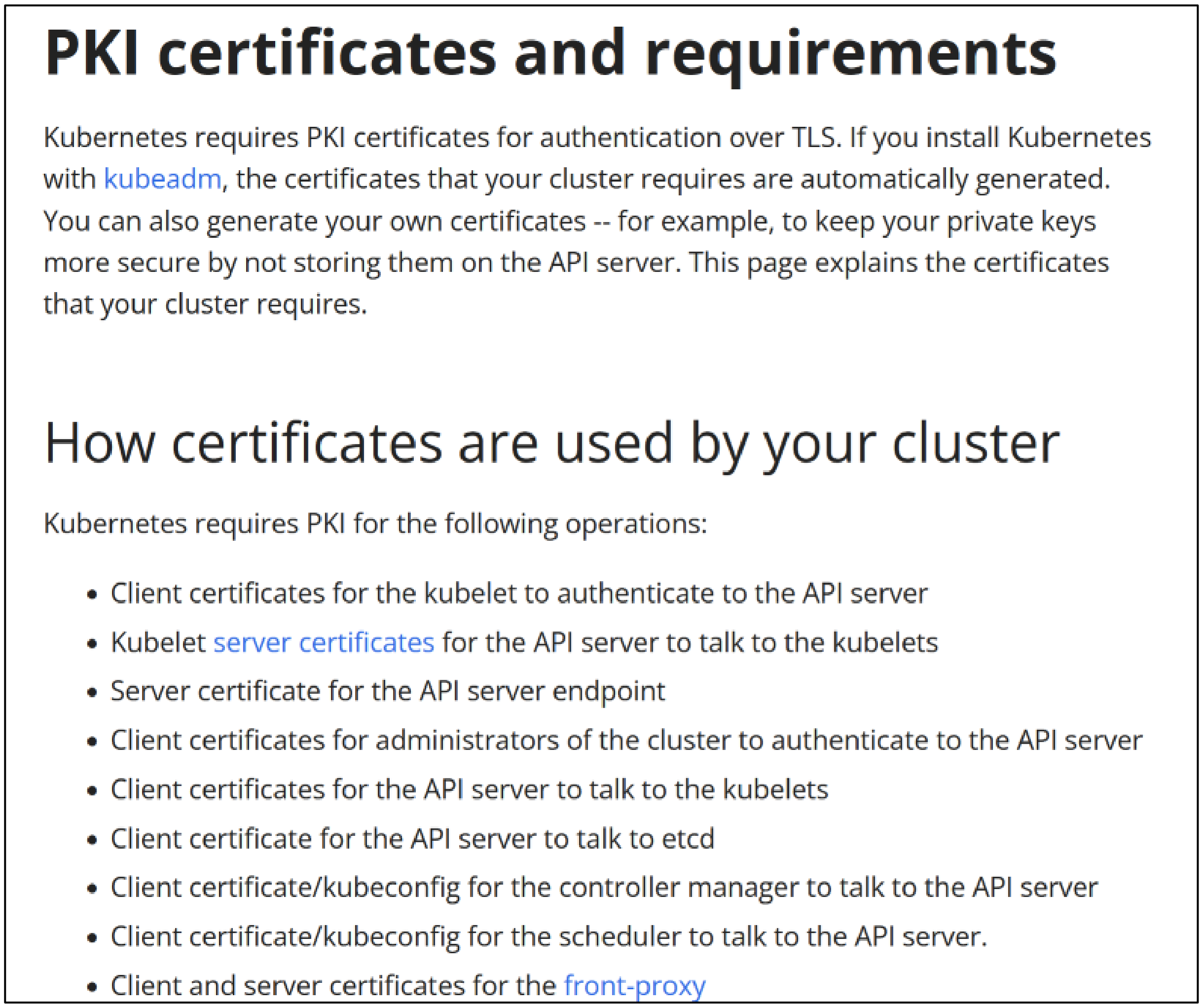
- The Amazon EKS control plane
- Amazon EKS nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as `etcd` and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).[70]

80. For example, Kubernetes implements PKI between the aforementioned programmable network device and cloud device, implementing a virtual fabric to form a distributed application running between server components.

[70] https://docs.aws.amazon.com/eks/latest/userguide/clusters.html



PKI certificates and requirements

Kubernetes requires PKI certificates for authentication over TLS. If you install Kubernetes with kubeadm, the certificates that your cluster requires are automatically generated. You can also generate your own certificates -- for example, to keep your private keys more secure by not storing them on the API server. This page explains the certificates that your cluster requires.

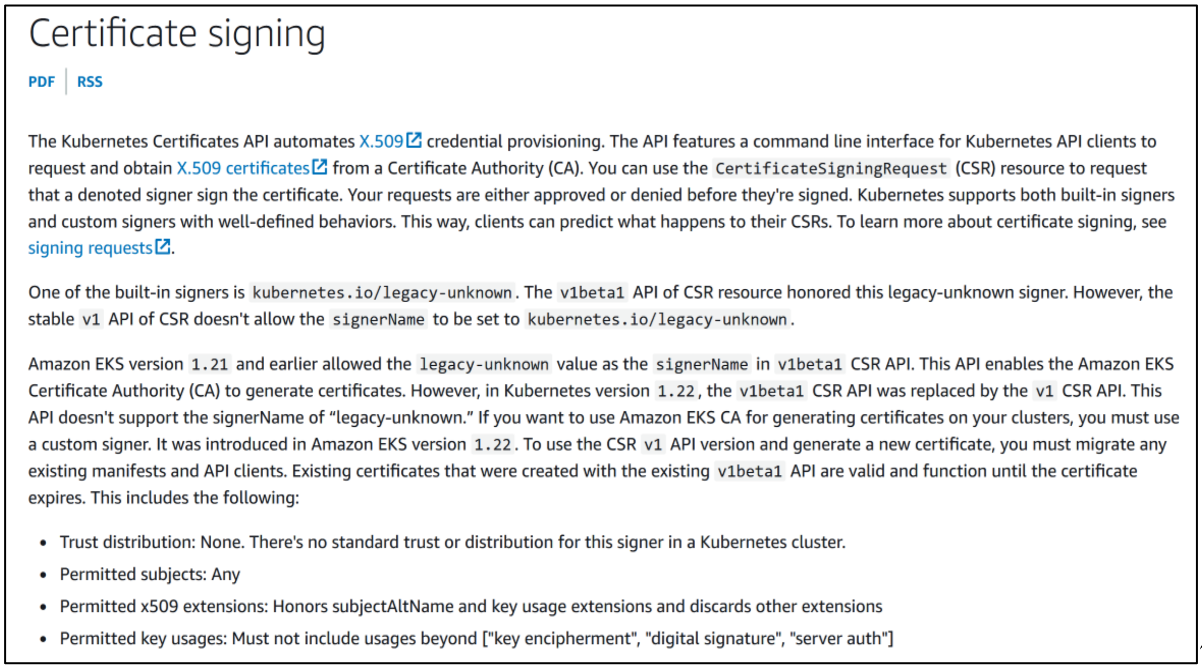
How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Kubelet server certificates for the API server to talk to the kubelets
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

[71]

[71] https://kubernetes.io/docs/setup/best-practices/certificates/


Certificate signing

PDF | RSS

The Kubernetes Certificates API automates X.509 credential provisioning. The API features a command line interface for Kubernetes API clients to request and obtain X.509 certificates from a Certificate Authority (CA). You can use the `CertificateSigningRequest` (CSR) resource to request that a denoted signer sign the certificate. Your requests are either approved or denied before they're signed. Kubernetes supports both built-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see signing requests.

One of the built-in signers is `kubernetes.io/legacy-unknown`. The `v1beta1` API of CSR resource honored this legacy-unknown signer. However, the stable `v1` API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

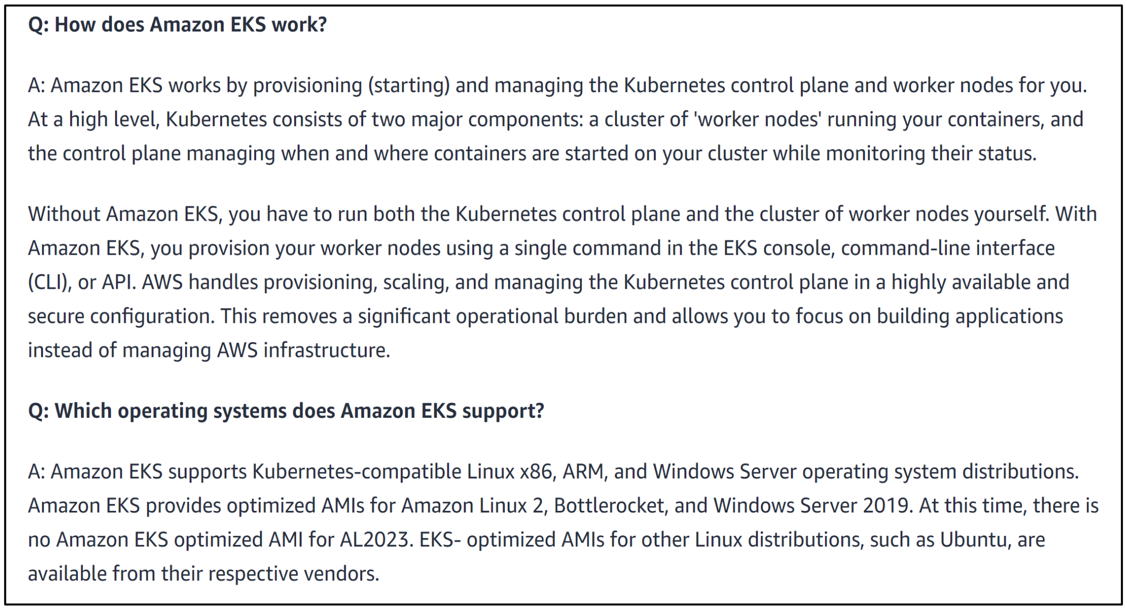
Amazon EKS version `1.21` and earlier allowed the `legacy-unknown` value as the `signerName` in `v1beta1` CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version `1.22`, the `v1beta1` CSR API was replaced by the `v1` CSR API. This API doesn't support the signerName of "legacy-unknown." If you want to use Amazon EKS CA for generating certificates on your clusters, you must use a custom signer. It was introduced in Amazon EKS version `1.22`. To use the CSR `v1` API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the existing `v1beta1` API are valid and function until the certificate expires. This includes the following:

- Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
- Permitted subjects: Any
- Permitted x509 extensions: Honors subjectAltName and key usage extensions and discards other extensions
- Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]

[72]

81. Amazon EKS Web Services comprise a programmable network device and a programmable cloud device, wherein the programmable network device and programmable cloud device are each powered by a sandboxing operating system which facilitates deployment of the plurality of first and second network applications. For example, the Network Device and Cloud Device are each powered by a sandboxing operating system (*e.g.*, a server-based operating system running virtual machines partitioned from each other), which facilitate deployment (*e.g.*, deploying an application through Kubernetes hosting models, including during initial deployment and updates).

[72] https://docs.aws.amazon.com/eks/latest/userguide/cert-signing.html



Q: How does Amazon EKS work?

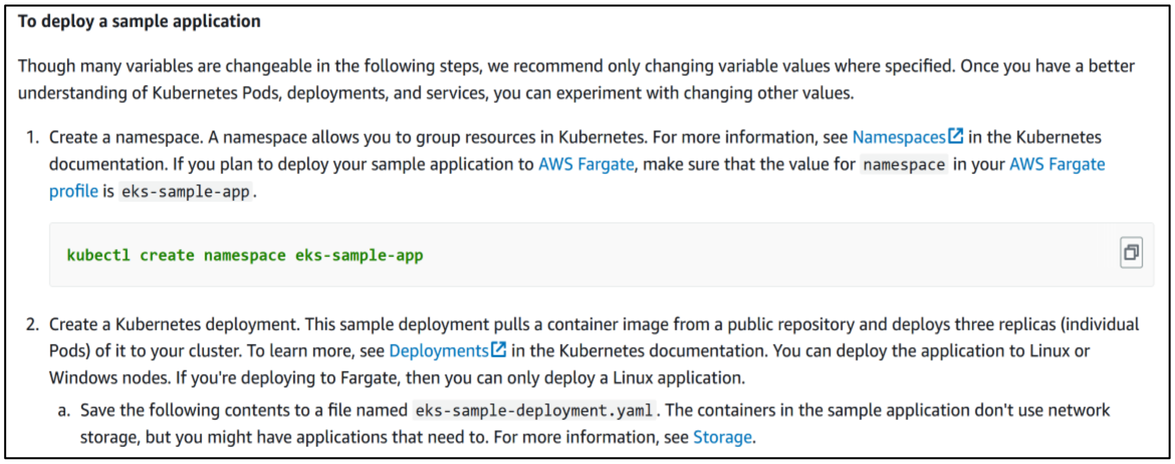
A: Amazon EKS works by provisioning (starting) and managing the Kubernetes control plane and worker nodes for you. At a high level, Kubernetes consists of two major components: a cluster of 'worker nodes' running your containers, and the control plane managing when and where containers are started on your cluster while monitoring their status.

Without Amazon EKS, you have to run both the Kubernetes control plane and the cluster of worker nodes yourself. With Amazon EKS, you provision your worker nodes using a single command in the EKS console, command-line interface (CLI), or API. AWS handles provisioning, scaling, and managing the Kubernetes control plane in a highly available and secure configuration. This removes a significant operational burden and allows you to focus on building applications instead of managing AWS infrastructure.

Q: Which operating systems does Amazon EKS support?

A: Amazon EKS supports Kubernetes-compatible Linux x86, ARM, and Windows Server operating system distributions. Amazon EKS provides optimized AMIs for Amazon Linux 2, Bottlerocket, and Windows Server 2019. At this time, there is no Amazon EKS optimized AMI for AL2023. EKS- optimized AMIs for other Linux distributions, such as Ubuntu, are available from their respective vendors.

[73]



To deploy a sample application

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes Pods, deployments, and services, you can experiment with changing other values.

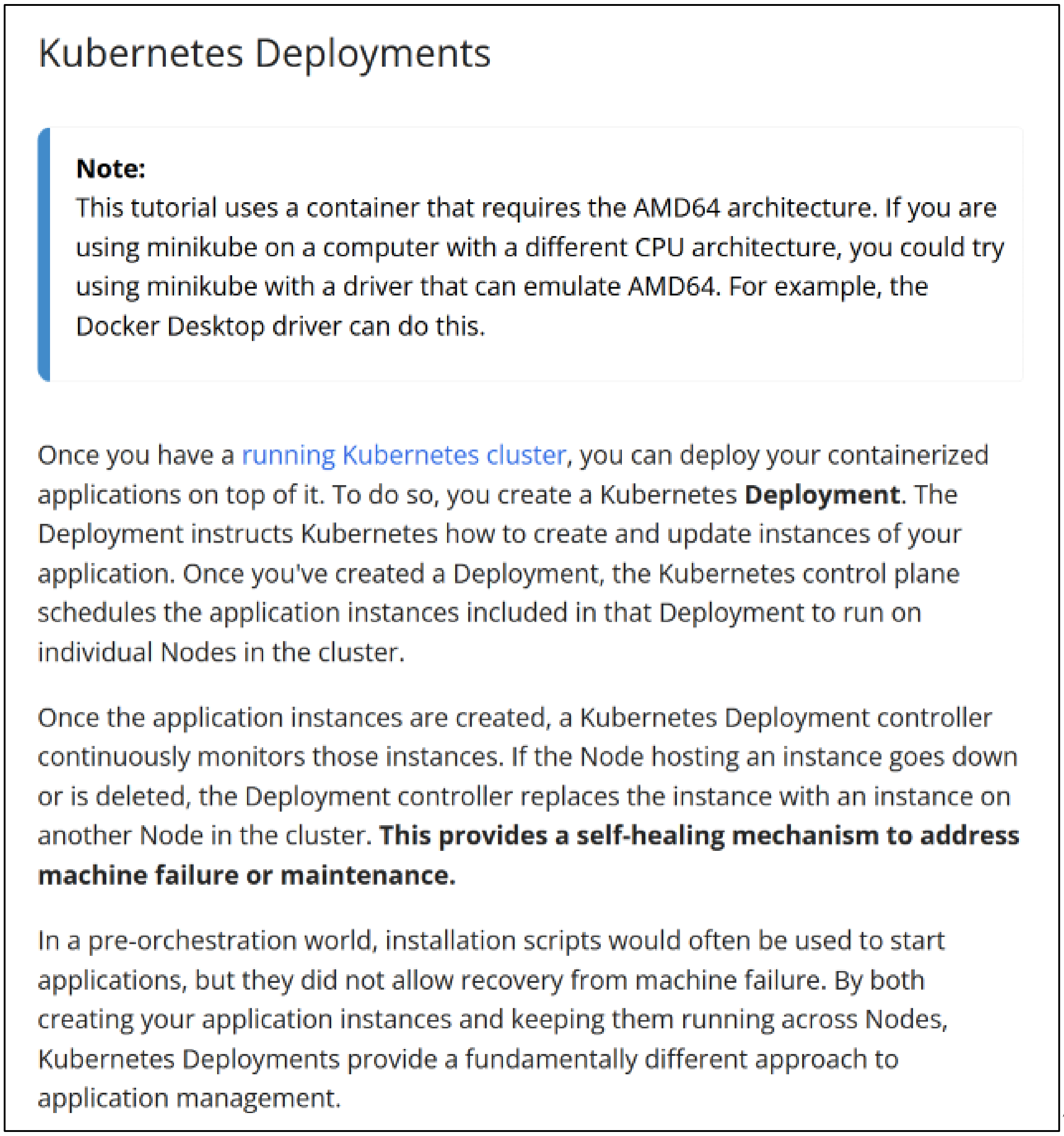
1. Create a namespace. A namespace allows you to group resources in Kubernetes. For more information, see Namespaces in the Kubernetes documentation. If you plan to deploy your sample application to AWS Fargate, make sure that the value for `namespace` in your AWS Fargate profile is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

2. Create a Kubernetes deployment. This sample deployment pulls a container image from a public repository and deploys three replicas (individual Pods) of it to your cluster. To learn more, see Deployments in the Kubernetes documentation. You can deploy the application to Linux or Windows nodes. If you're deploying to Fargate, then you can only deploy a Linux application.
   a. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see Storage.

[74]

[73] https://aws.amazon.com/eks/faqs/
[74] https://docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html


Kubernetes Deployments

Note:
This tutorial uses a container that requires the AMD64 architecture. If you are using minikube on a computer with a different CPU architecture, you could try using minikube with a driver that can emulate AMD64. For example, the Docker Desktop driver can do this.

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment**. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

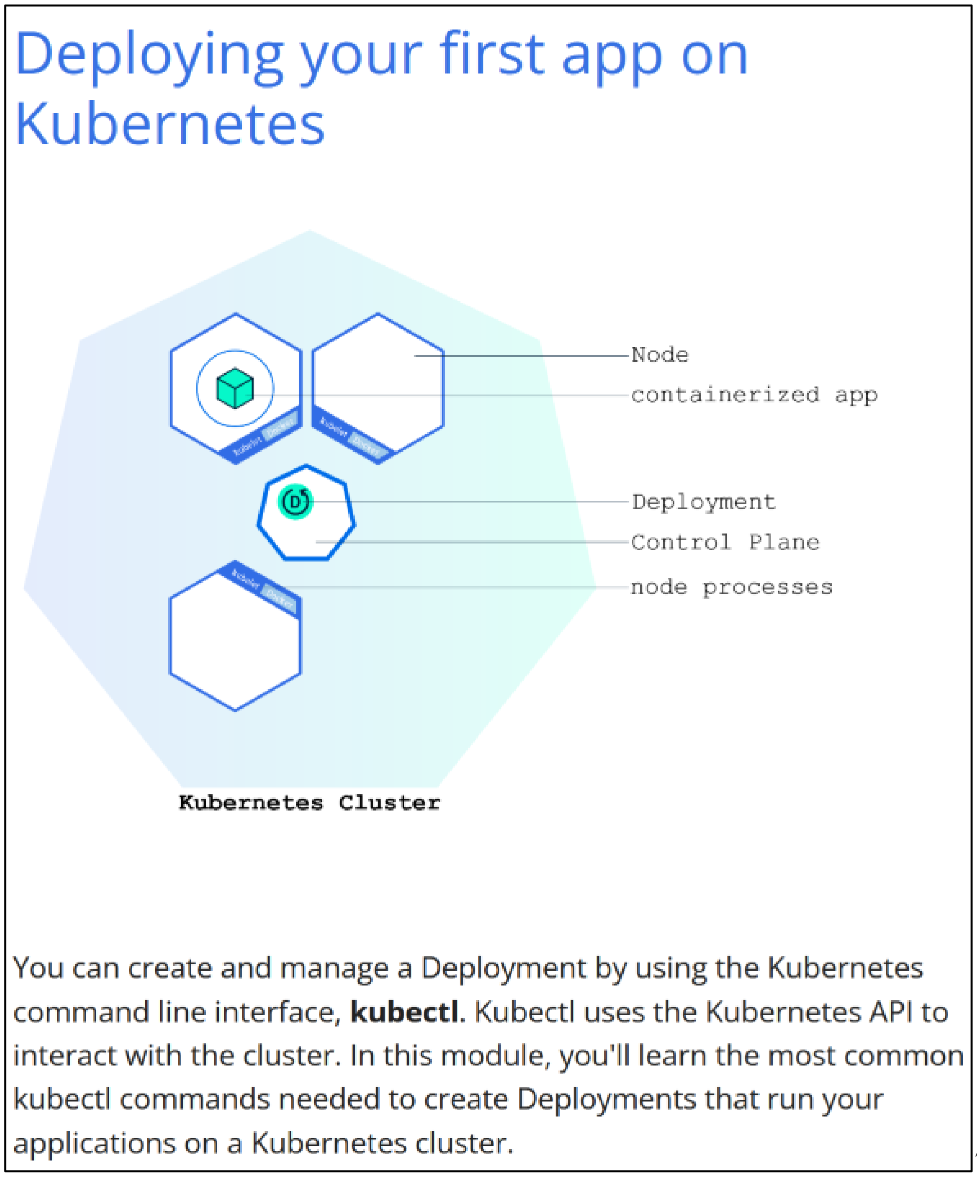
Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**

In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to application management.

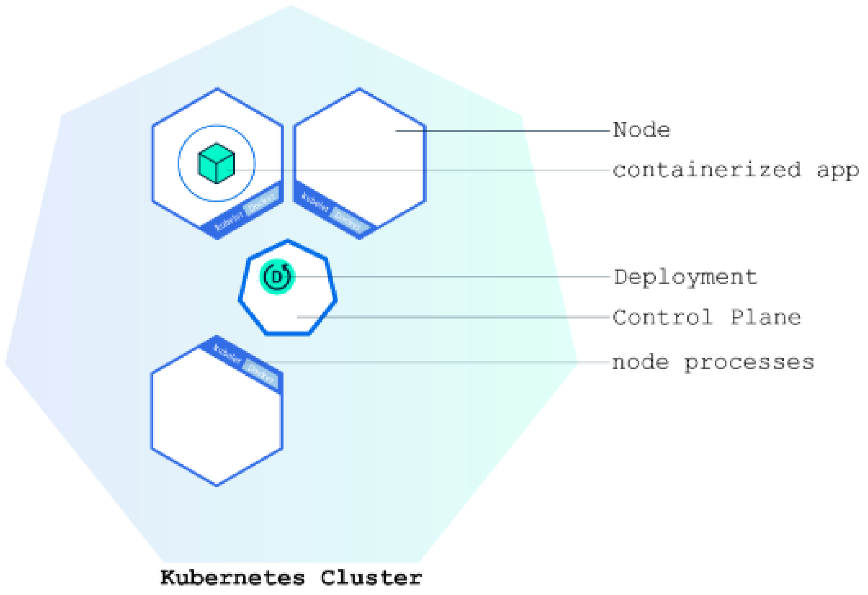
[75]

82. For example, the network device and programmable cloud device, as discussed above, can each facilitate deployment of the plurality of network applications (*e.g.*, across nodes, node processes, and control planes).

[75] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

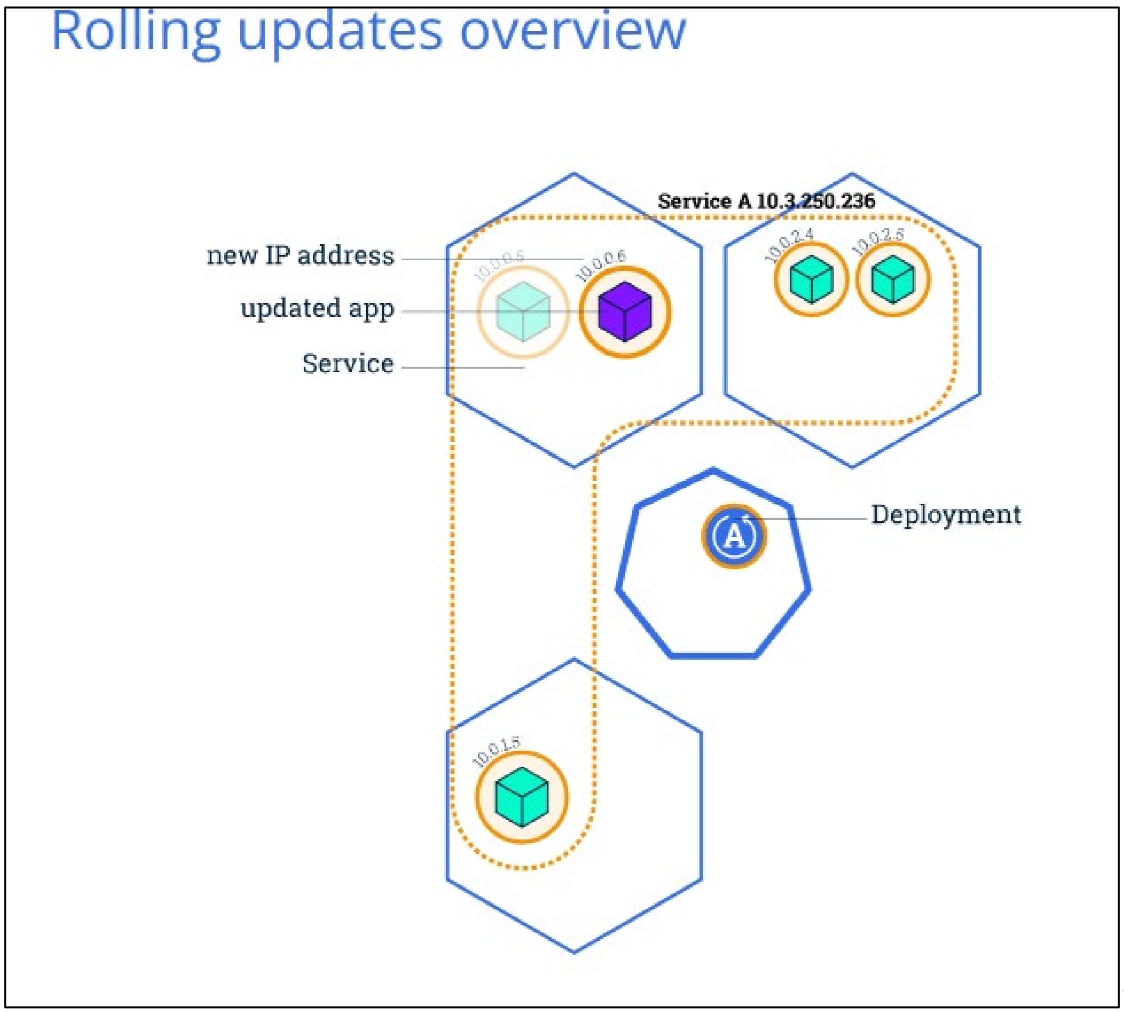


Deploying your first app on Kubernetes



You can create and manage a Deployment by using the Kubernetes command line interface, **kubectl**. Kubectl uses the Kubernetes API to interact with the cluster. In this module, you'll learn the most common kubectl commands needed to create Deployments that run your applications on a Kubernetes cluster.[76]

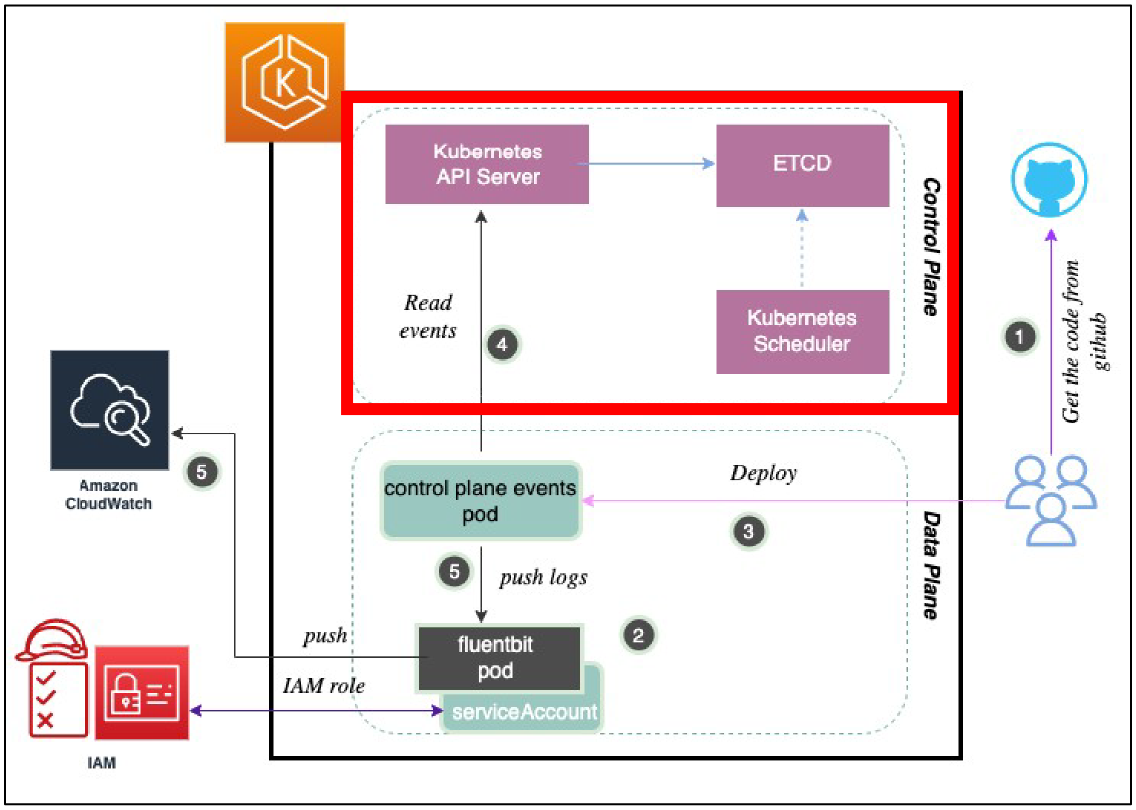
[76] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

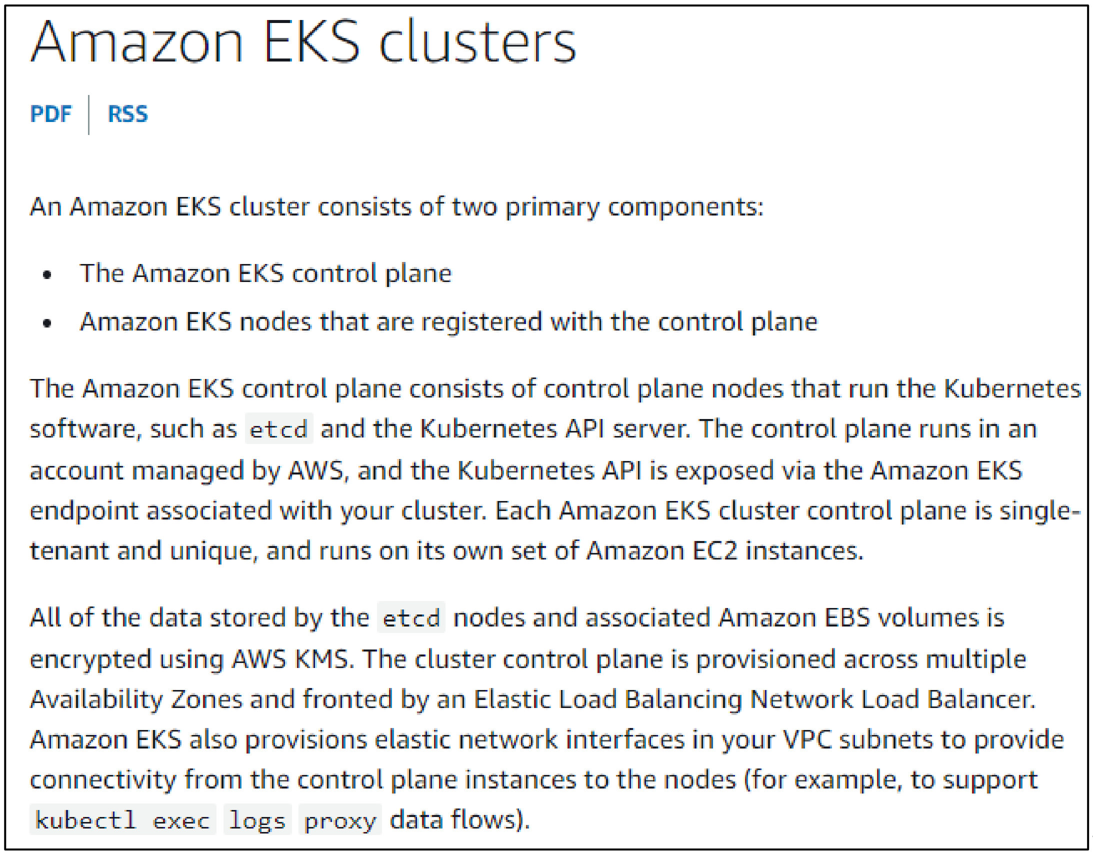


[77]

83. Amazon EKS Web Services comprise an application management portal coupled to the programmable network device and programmable cloud device and capable of managing upgrades of the first and second network applications with substantially no interruption to operation of the programmable network device and programmable cloud device. For example, the application management portal includes the Control Plane of a managed node group, which includes the dashboard, API server, controllers, and schedulers associated with that Control Plane.

[77] https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/


Kubernetes API Server
ETCD
Kubernetes Scheduler
Control Plane
Read events
4
1
Get the code from github
Amazon CloudWatch
5
control plane events pod
Deploy
3
Data Plane
5
push logs
push
fluentbit pod
2
IAM role
serviceAccount
IAM



Amazon EKS clusters

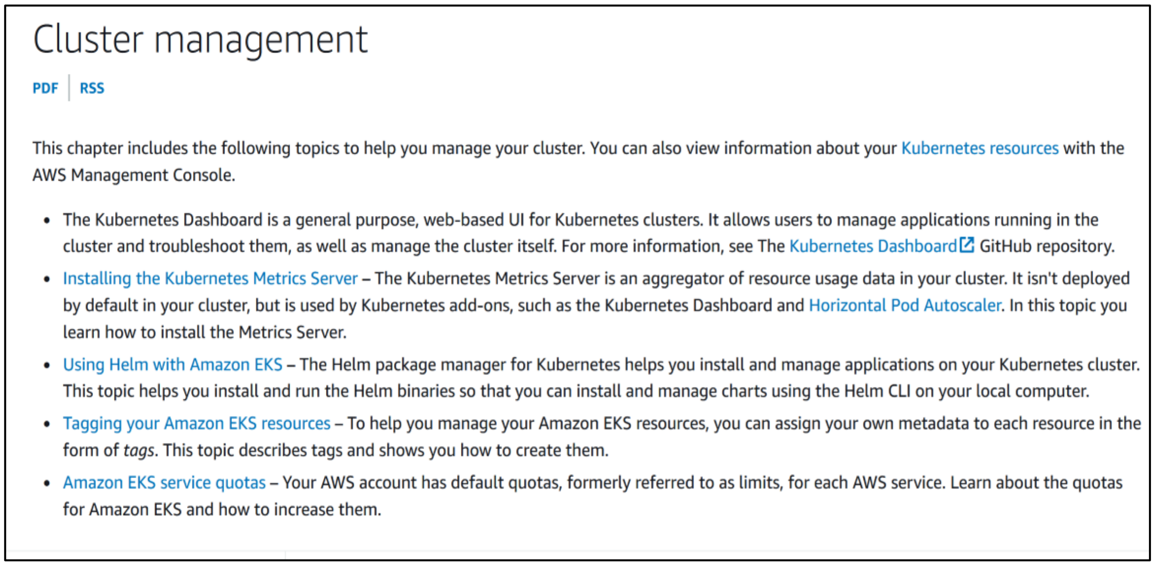
PDF | RSS

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as `etcd` and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted using AWS KMS. The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).[78]



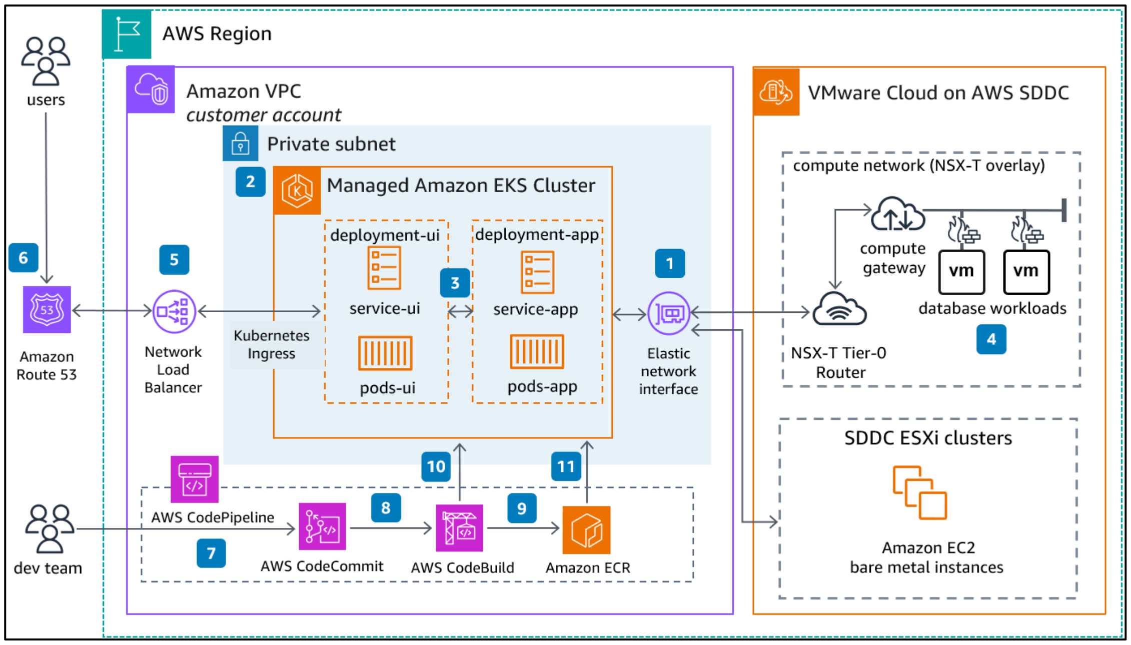
Cluster management

PDF | RSS

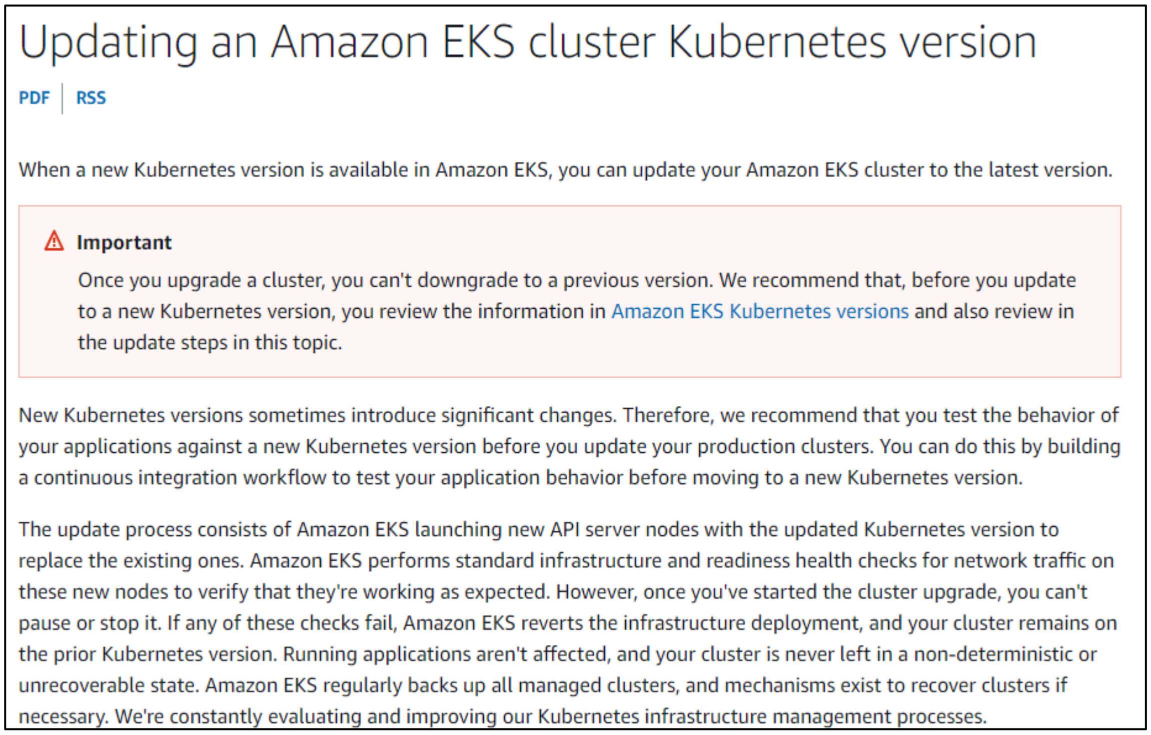
This chapter includes the following topics to help you manage your cluster. You can also view information about your Kubernetes resources with the AWS Management Console.

- The Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself. For more information, see The Kubernetes Dashboard GitHub repository.
- Installing the Kubernetes Metrics Server – The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster. It isn't deployed by default in your cluster, but is used by Kubernetes add-ons, such as the Kubernetes Dashboard and Horizontal Pod Autoscaler. In this topic you learn how to install the Metrics Server.
- Using Helm with Amazon EKS – The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local computer.
- Tagging your Amazon EKS resources – To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.
- Amazon EKS service quotas – Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Learn about the quotas for Amazon EKS and how to increase them.[79]

[78] https://docs.aws.amazon.com/eks/latest/userguide/clusters.html
[79] https://docs.aws.amazon.com/eks/latest/userguide/eks-managing.html



[80]



Updating an Amazon EKS cluster Kubernetes version

PDF | RSS

When a new Kubernetes version is available in Amazon EKS, you can update your Amazon EKS cluster to the latest version.

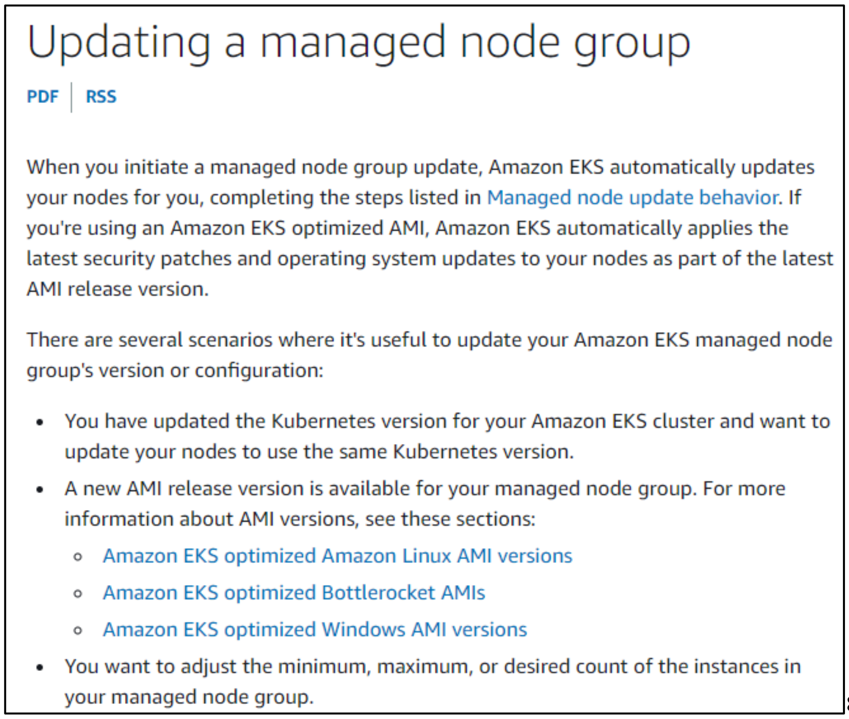
Important

Once you upgrade a cluster, you can't downgrade to a previous version. We recommend that, before you update to a new Kubernetes version, you review the information in Amazon EKS Kubernetes versions and also review in the update steps in this topic.

New Kubernetes versions sometimes introduce significant changes. Therefore, we recommend that you test the behavior of your applications against a new Kubernetes version before you update your production clusters. You can do this by building a continuous integration workflow to test your application behavior before moving to a new Kubernetes version.

The update process consists of Amazon EKS launching new API server nodes with the updated Kubernetes version to replace the existing ones. Amazon EKS performs standard infrastructure and readiness health checks for network traffic on these new nodes to verify that they're working as expected. However, once you've started the cluster upgrade, you can't pause or stop it. If any of these checks fail, Amazon EKS reverts the infrastructure deployment, and your cluster remains on the prior Kubernetes version. Running applications aren't affected, and your cluster is never left in a non-deterministic or unrecoverable state. Amazon EKS regularly backs up all managed clusters, and mechanisms exist to recover clusters if necessary. We're constantly evaluating and improving our Kubernetes infrastructure management processes. [81]

[80] https://docs.aws.amazon.com/architecture-diagrams/latest/modernize-applications-with-microservices-using-amazon-eks/modernize-applications-with-microservices-using-amazon-eks.html
[81] https://docs.aws.amazon.com/eks/latest/userguide/update-cluster.html



Updating a managed node group

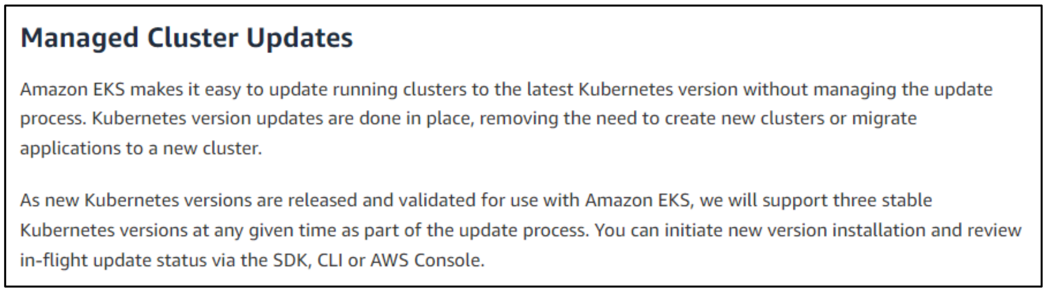
PDF | RSS

When you initiate a managed node group update, Amazon EKS automatically updates your nodes for you, completing the steps listed in Managed node update behavior. If you're using an Amazon EKS optimized AMI, Amazon EKS automatically applies the latest security patches and operating system updates to your nodes as part of the latest AMI release version.

There are several scenarios where it's useful to update your Amazon EKS managed node group's version or configuration:

- You have updated the Kubernetes version for your Amazon EKS cluster and want to update your nodes to use the same Kubernetes version.
- A new AMI release version is available for your managed node group. For more information about AMI versions, see these sections:
  - Amazon EKS optimized Amazon Linux AMI versions
  - Amazon EKS optimized Bottlerocket AMIs
  - Amazon EKS optimized Windows AMI versions
- You want to adjust the minimum, maximum, or desired count of the instances in your managed node group.

[82]



Managed Cluster Updates

Amazon EKS makes it easy to update running clusters to the latest Kubernetes version without managing the update process. Kubernetes version updates are done in place, removing the need to create new clusters or migrate applications to a new cluster.

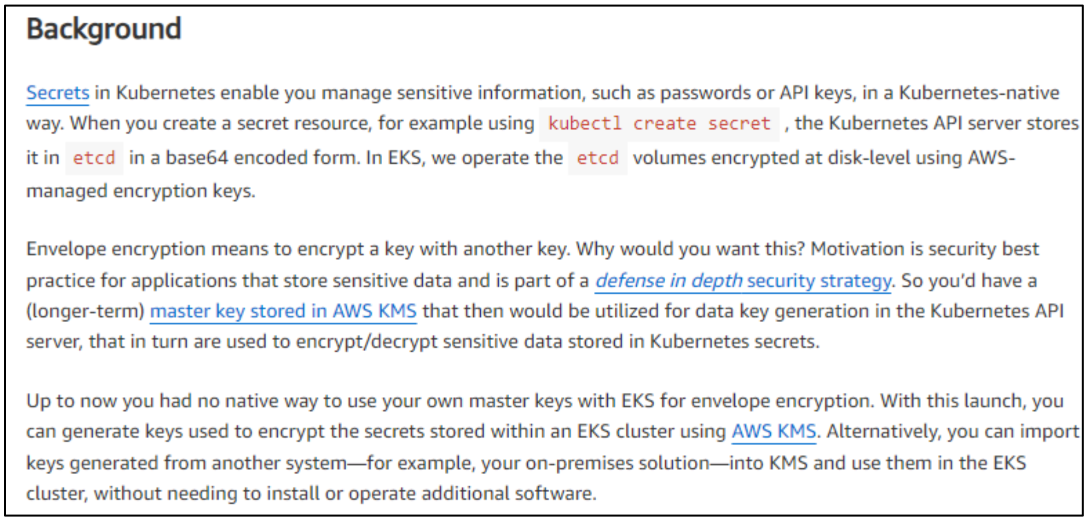
As new Kubernetes versions are released and validated for use with Amazon EKS, we will support three stable Kubernetes versions at any given time as part of the update process. You can initiate new version installation and review in-flight update status via the SDK, CLI or AWS Console.

[83]

84. Amazon EKS Web Services comprise an application management portal coupled to the programmable network device and programmable cloud device, wherein the application management portal verifies authenticity of the upgrades to the plurality of first network applications and the plurality of second network applications and wherein the verification is based on unique security keys associated with each of the plurality of first and second network applications. For example, the application management portal can verify the authenticity to the

[82] https://docs.aws.amazon.com/eks/latest/userguide/update-managed-node-group.html
[83] https://aws.amazon.com/eks/features/

upgrades to both sets of network application based on unique security keys.

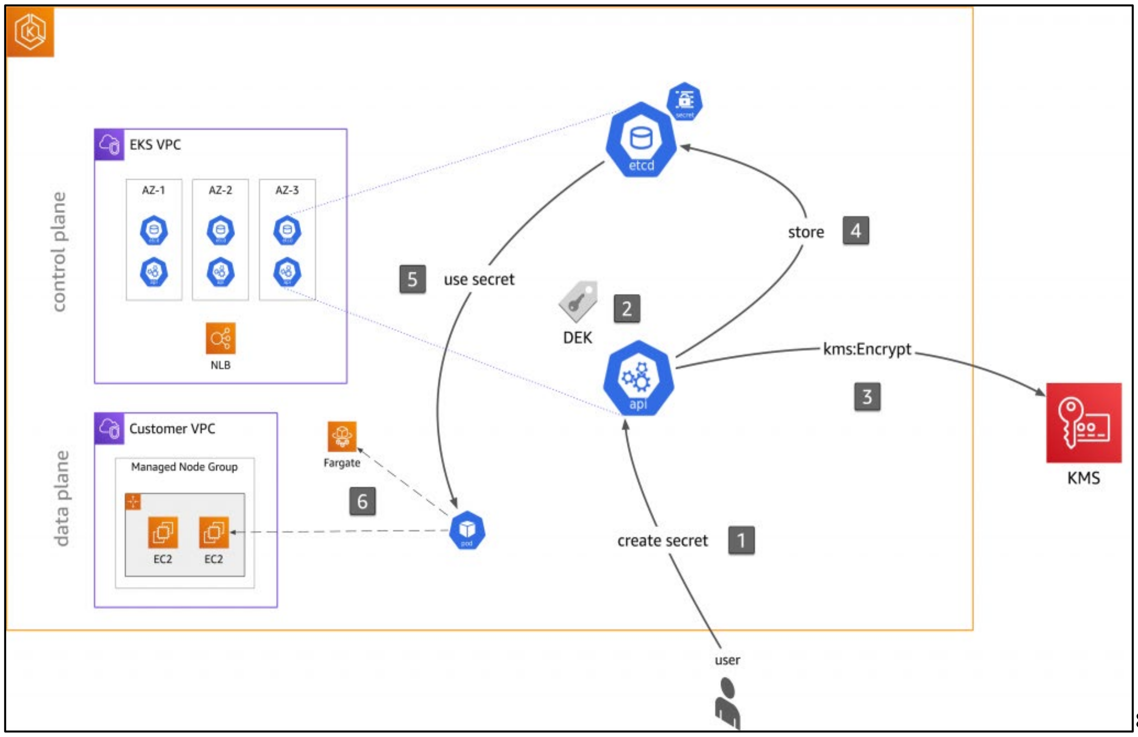


Background

Secrets in Kubernetes enable you manage sensitive information, such as passwords or API keys, in a Kubernetes-native way. When you create a secret resource, for example using `kubectl create secret` , the Kubernetes API server stores it in `etcd` in a base64 encoded form. In EKS, we operate the `etcd` volumes encrypted at disk-level using AWS-managed encryption keys.

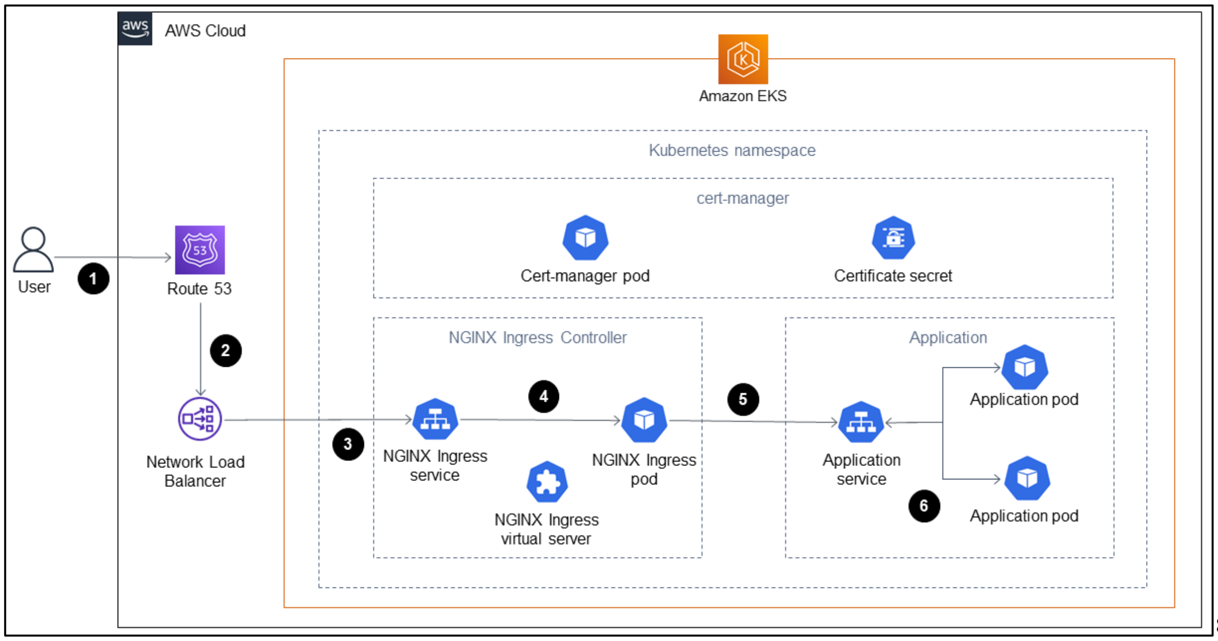
Envelope encryption means to encrypt a key with another key. Why would you want this? Motivation is security best practice for applications that store sensitive data and is part of a *defense in depth* security strategy. So you'd have a (longer-term) master key stored in AWS KMS that then would be utilized for data key generation in the Kubernetes API server, that in turn are used to encrypt/decrypt sensitive data stored in Kubernetes secrets.

Up to now you had no native way to use your own master keys with EKS for envelope encryption. With this launch, you can generate keys used to encrypt the secrets stored within an EKS cluster using AWS KMS. Alternatively, you can import keys generated from another system—for example, your on-premises solution—into KMS and use them in the EKS cluster, without needing to install or operate additional software.

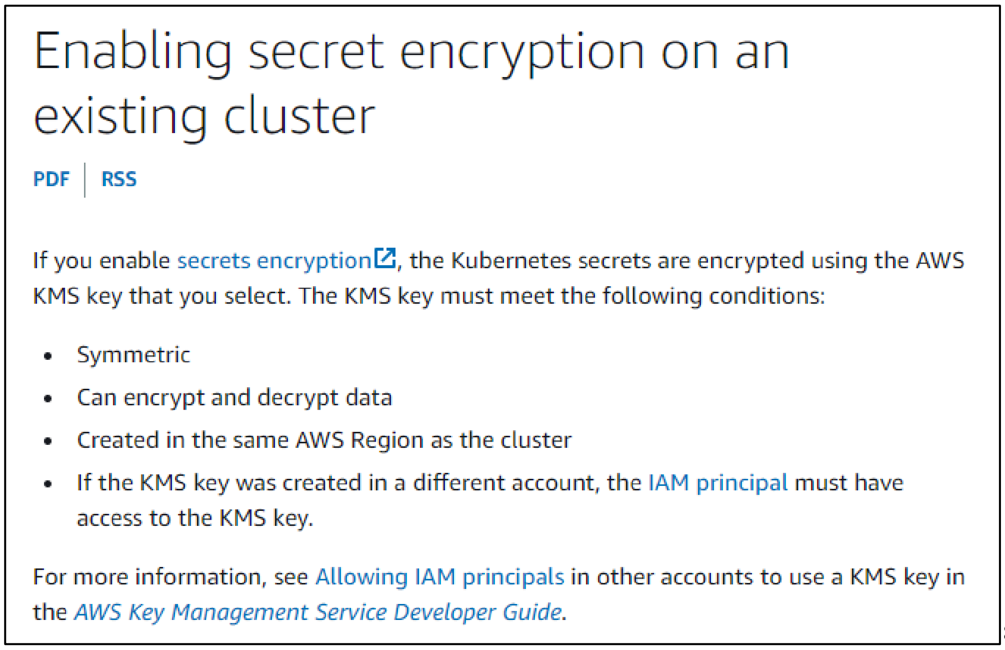


84

84 https://aws.amazon.com/blogs/containers/using-eks-encryption-provider-support-for-defense-in-depth/



[85]



Enabling secret encryption on an existing cluster

PDF | RSS

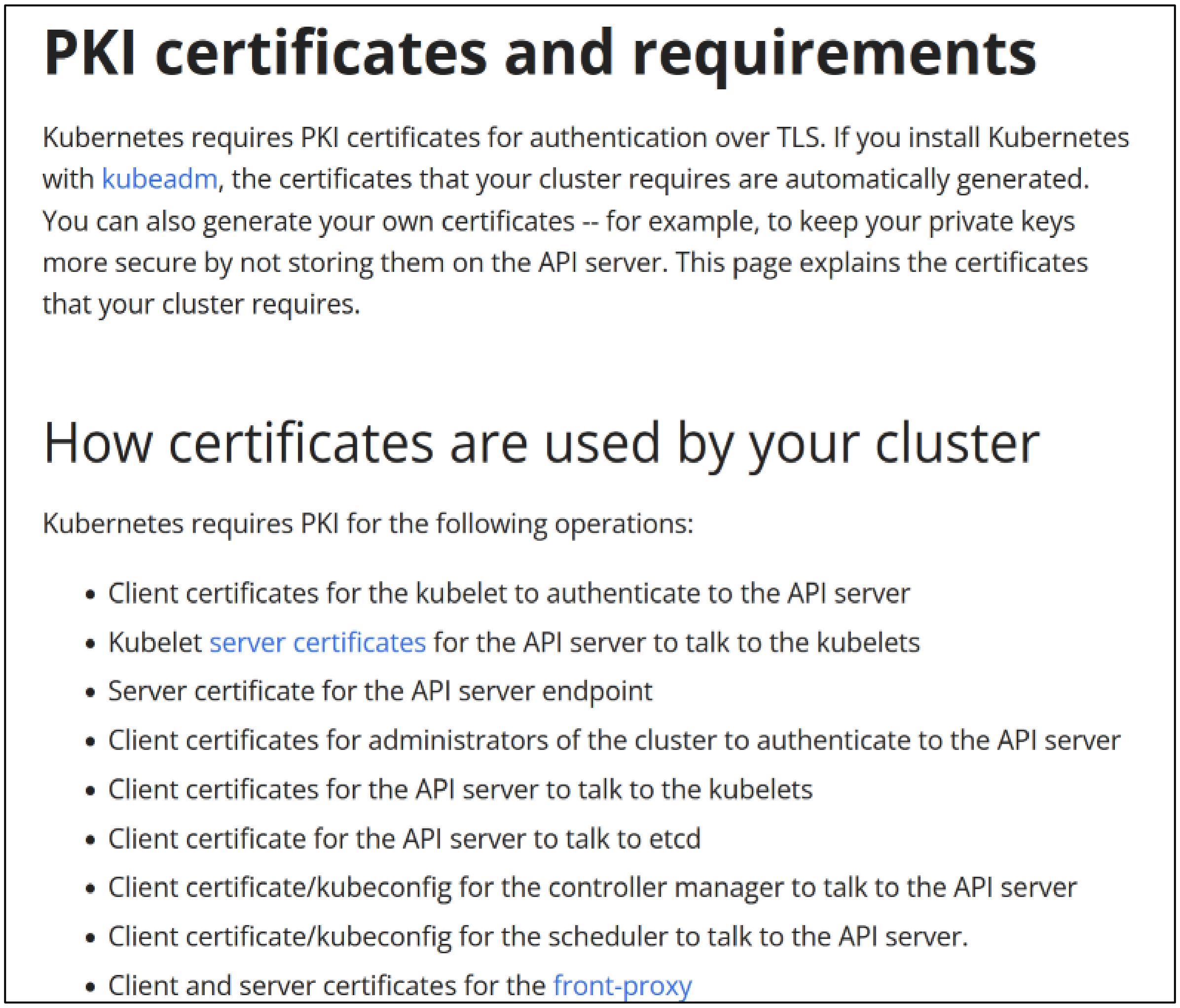
If you enable secrets encryption, the Kubernetes secrets are encrypted using the AWS KMS key that you select. The KMS key must meet the following conditions:

- Symmetric
- Can encrypt and decrypt data
- Created in the same AWS Region as the cluster
- If the KMS key was created in a different account, the IAM principal must have access to the KMS key.

For more information, see Allowing IAM principals in other accounts to use a KMS key in the *AWS Key Management Service Developer Guide*.

[86]

[85] https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/set-up-end-to-end-encryption-for-applications-on-amazon-eks-using-cert-manager-and-let-s-encrypt.html
[86] https://docs.aws.amazon.com/eks/latest/userguide/enable-kms.html



PKI certificates and requirements

Kubernetes requires PKI certificates for authentication over TLS. If you install Kubernetes with kubeadm, the certificates that your cluster requires are automatically generated. You can also generate your own certificates -- for example, to keep your private keys more secure by not storing them on the API server. This page explains the certificates that your cluster requires.

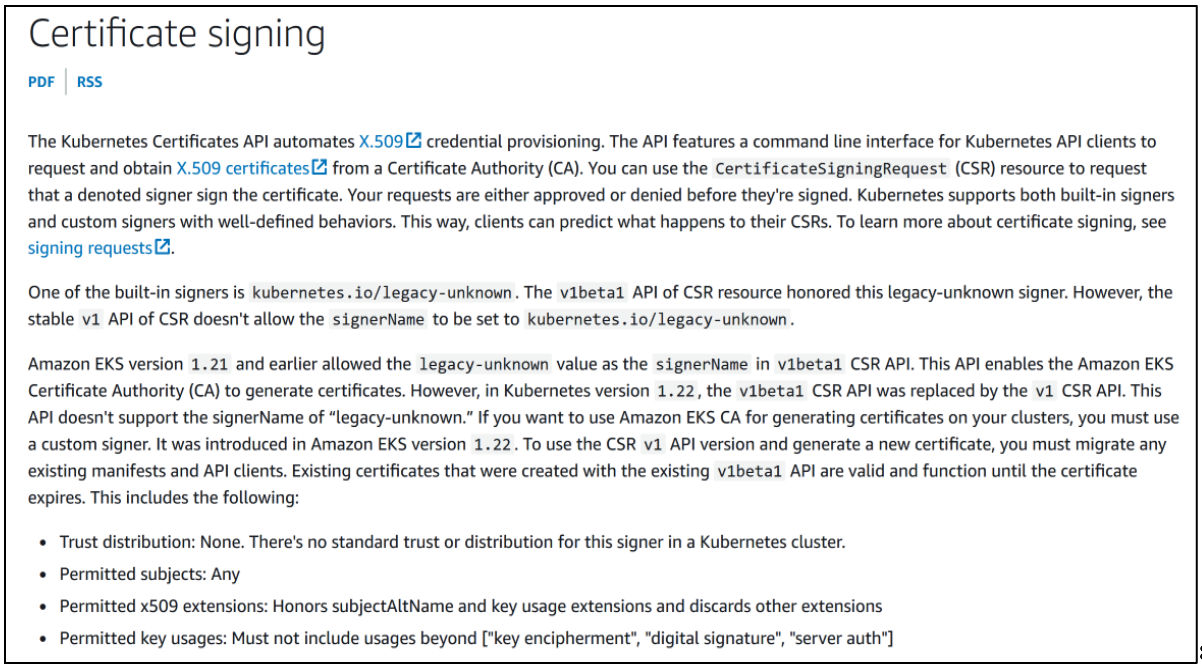
How certificates are used by your cluster

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Kubelet server certificates for the API server to talk to the kubelets
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the front-proxy

[87]

[87] https://kubernetes.io/docs/setup/best-practices/certificates/



Certificate signing

PDF | RSS

The Kubernetes Certificates API automates X.509 credential provisioning. The API features a command line interface for Kubernetes API clients to request and obtain X.509 certificates from a Certificate Authority (CA). You can use the `CertificateSigningRequest` (CSR) resource to request that a denoted signer sign the certificate. Your requests are either approved or denied before they're signed. Kubernetes supports both built-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see signing requests.

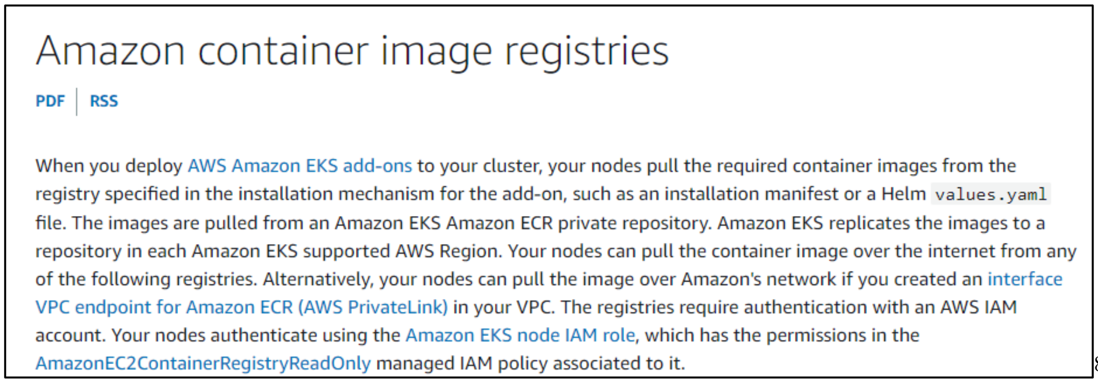
One of the built-in signers is `kubernetes.io/legacy-unknown`. The `v1beta1` API of CSR resource honored this legacy-unknown signer. However, the stable `v1` API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

Amazon EKS version `1.21` and earlier allowed the `legacy-unknown` value as the `signerName` in `v1beta1` CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version `1.22`, the `v1beta1` CSR API was replaced by the `v1` CSR API. This API doesn't support the signerName of "legacy-unknown." If you want to use Amazon EKS CA for generating certificates on your clusters, you must use a custom signer. It was introduced in Amazon EKS version `1.22`. To use the CSR `v1` API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the existing `v1beta1` API are valid and function until the certificate expires. This includes the following:

- Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
- Permitted subjects: Any
- Permitted x509 extensions: Honors subjectAltName and key usage extensions and discards other extensions
- Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]

[88]

85. Amazon EKS Web Services comprise an application repository (*e.g.*, a container) coupled to the application management portal and capable of storing distributed applications for installation in the programmable network device and programmable cloud device.



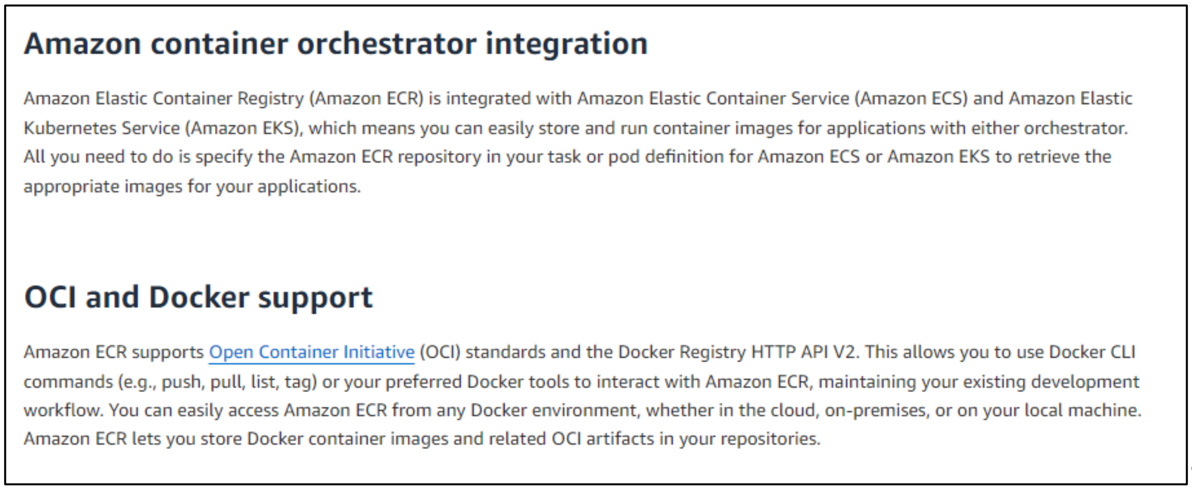
Amazon container image registries

PDF | RSS

When you deploy AWS Amazon EKS add-ons to your cluster, your nodes pull the required container images from the registry specified in the installation mechanism for the add-on, such as an installation manifest or a Helm `values.yaml` file. The images are pulled from an Amazon EKS Amazon ECR private repository. Amazon EKS replicates the images to a repository in each Amazon EKS supported AWS Region. Your nodes can pull the container image over the internet from any of the following registries. Alternatively, your nodes can pull the image over Amazon's network if you created an interface VPC endpoint for Amazon ECR (AWS PrivateLink) in your VPC. The registries require authentication with an AWS IAM account. Your nodes authenticate using the Amazon EKS node IAM role, which has the permissions in the AmazonEC2ContainerRegistryReadOnly managed IAM policy associated to it.

[89]

[88] https://docs.aws.amazon.com/eks/latest/userguide/cert-signing.html
[89] https://docs.aws.amazon.com/eks/latest/userguide/add-ons-images.html



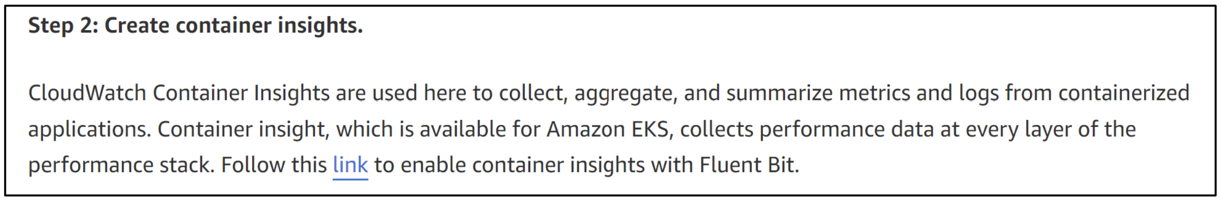
Amazon container orchestrator integration

Amazon Elastic Container Registry (Amazon ECR) is integrated with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS), which means you can easily store and run container images for applications with either orchestrator. All you need to do is specify the Amazon ECR repository in your task or pod definition for Amazon ECS or Amazon EKS to retrieve the appropriate images for your applications.

OCI and Docker support

Amazon ECR supports Open Container Initiative (OCI) standards and the Docker Registry HTTP API V2. This allows you to use Docker CLI commands (e.g., push, pull, list, tag) or your preferred Docker tools to interact with Amazon ECR, maintaining your existing development workflow. You can easily access Amazon ECR from any Docker environment, whether in the cloud, on-premises, or on your local machine. Amazon ECR lets you store Docker container images and related OCI artifacts in your repositories.

[90]



Step 2: Create container insights.

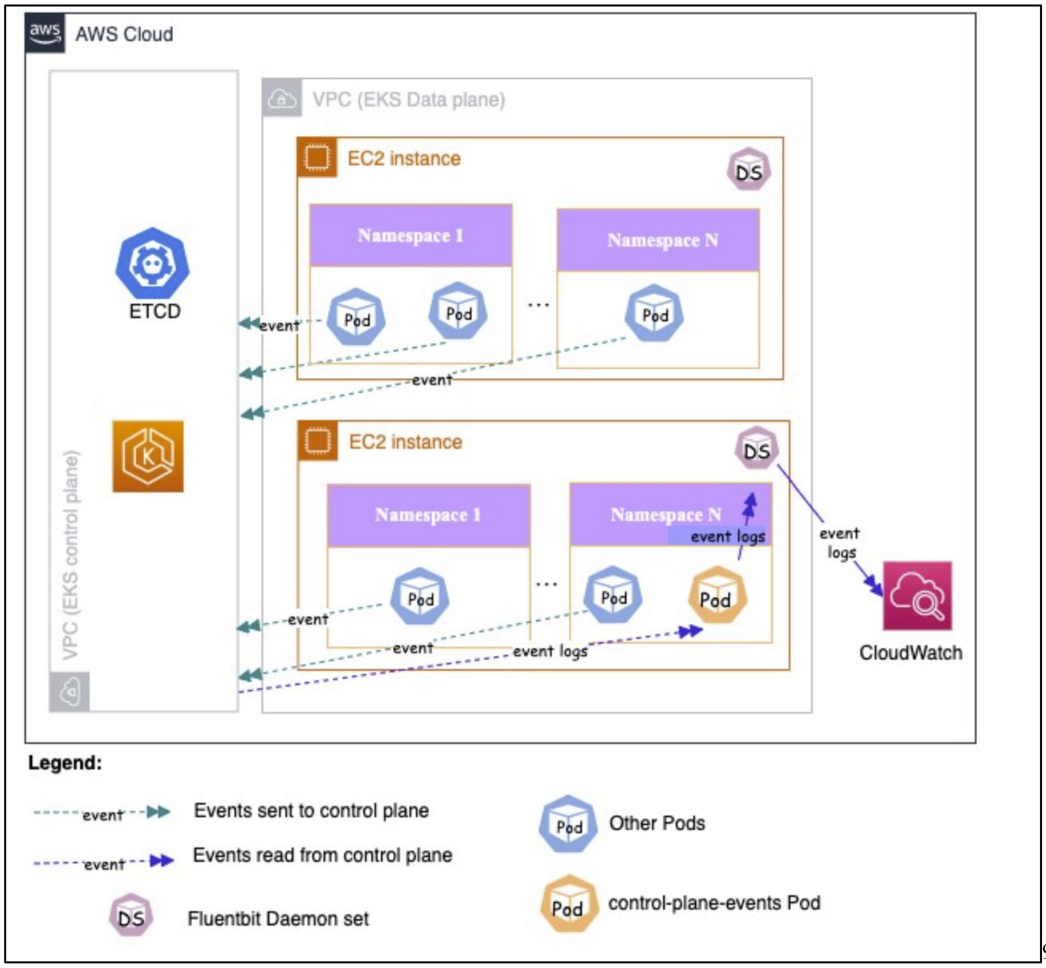
CloudWatch Container Insights are used here to collect, aggregate, and summarize metrics and logs from containerized applications. Container insight, which is available for Amazon EKS, collects performance data at every layer of the performance stack. Follow this link to enable container insights with Fluent Bit.

[91]

86. Amazon EKS Web Services comprise an application repository coupled to the application management portal, wherein the application management portal (*e.g.*, the control plane, including its associated dashboard) further manages usage of the distributed applications on the programmable network device and programmable cloud device.

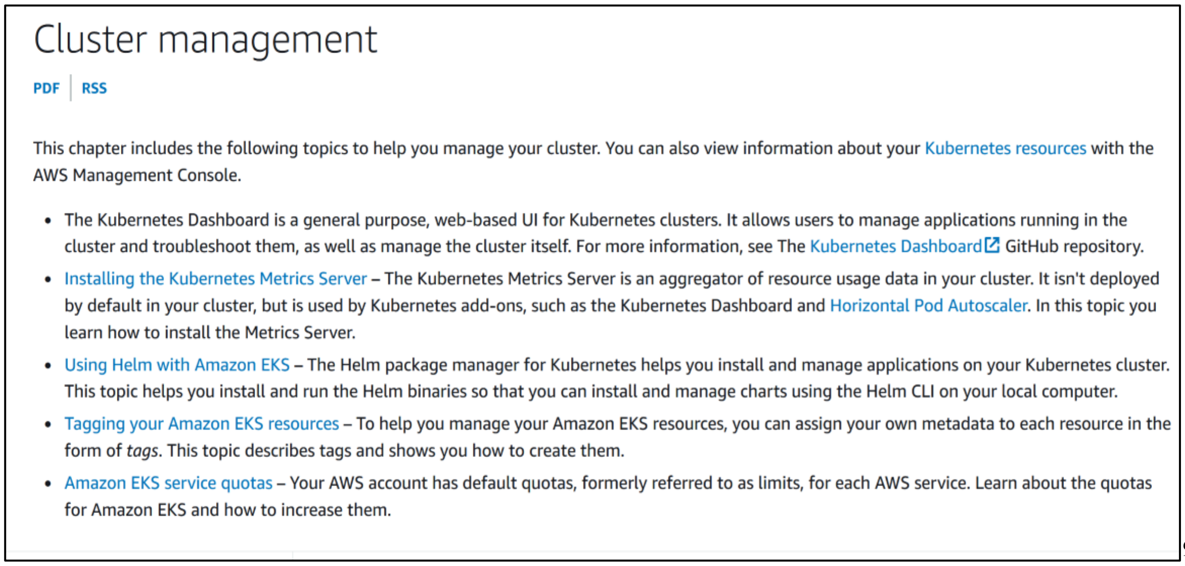
[90] https://aws.amazon.com/ecr/

[91] https://aws.amazon.com/blogs/containers/managing-kubernetes-control-plane-events-in-amazon-eks/



[92]

92 https://aws.amazon.com/blogs/containers/managing-kubernetes-control-plane-events-in-amazon-eks/



Cluster management

PDF | RSS

This chapter includes the following topics to help you manage your cluster. You can also view information about your Kubernetes resources with the AWS Management Console.

- The Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself. For more information, see The Kubernetes Dashboard GitHub repository.
- Installing the Kubernetes Metrics Server – The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster. It isn't deployed by default in your cluster, but is used by Kubernetes add-ons, such as the Kubernetes Dashboard and Horizontal Pod Autoscaler. In this topic you learn how to install the Metrics Server.
- Using Helm with Amazon EKS – The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local computer.
- Tagging your Amazon EKS resources – To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.
- Amazon EKS service quotas – Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Learn about the quotas for Amazon EKS and how to increase them.

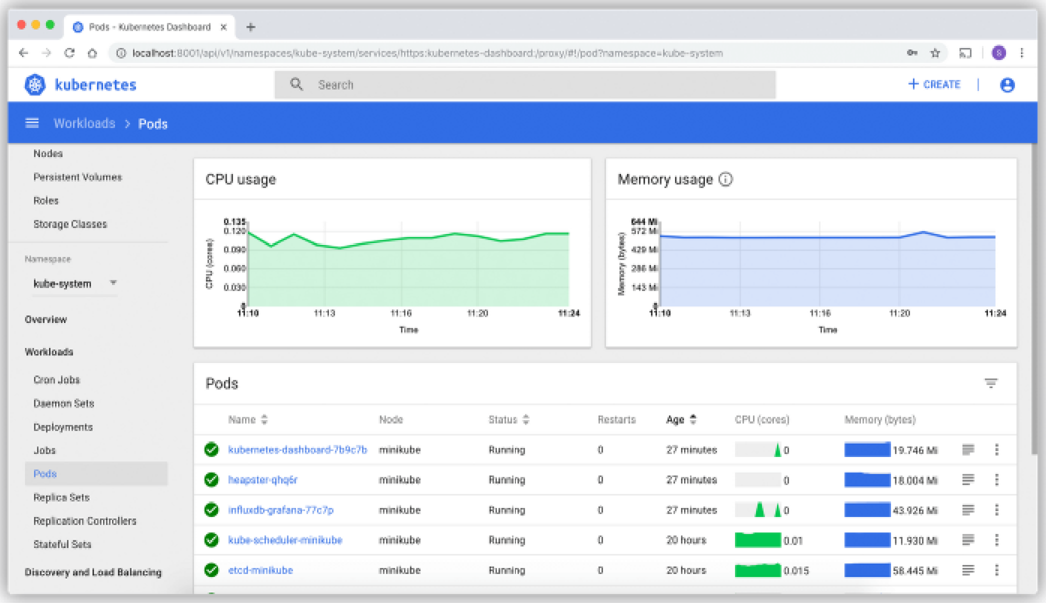
[93]

[93] https://docs.aws.amazon.com/eks/latest/userguide/eks-managing.html

Deploy and Access the Kubernetes Dashboard

Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use Dashboard to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). For example, you can scale a Deployment, initiate a rolling update, restart a pod or deploy new applications using a deploy wizard.

Dashboard also provides information on the state of Kubernetes resources in your cluster and on any errors that may have occurred.



Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
kubernetes-dashboard-7b9c7b	minikube	Running	0	27 minutes	0	19.746 Mi
heapster-qhq6r	minikube	Running	0	27 minutes	0	18.004 Mi
influxdb-grafana-77c7p	minikube	Running	0	27 minutes	0	43.926 Mi
kube-scheduler-minikube	minikube	Running	0	20 hours	0.01	11.930 Mi
etcd-minikube	minikube	Running	0	20 hours	0.015	58.445 Mi

[94]

87. Defendants have and continue to indirectly infringe one or more claims of the '871 Patent by knowingly and intentionally inducing others, including Amazon customers and end-users, to directly infringe, either literally or under the doctrine of equivalents, by making, using, offering to sell, selling, and/or importing into the United States products that include the infringing

[94] https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

technology.

88. Defendants, with knowledge[95] that these products, or the use thereof, infringe the ’871 Patent at least as of the date this Complaint, knowingly and intentionally induced, and continue to knowingly and intentionally induce, direct infringement of the ’871 Patent by providing these products to end-users for use in an infringing manner. Alternatively, on information and belief, Defendants have adopted a policy of not reviewing the patents of others, including specifically those related to Defendants’ specific industry, thereby remaining willfully blind to the Patent-in-Suit at least as early as the issuance of the Patents-in-Suit.

89. Defendants have induced infringement by others, including end-users, with the intent to cause infringing acts by others or, in the alternative, with the belief that there was a high probability that others, including end-users, infringe the ’871 Patent, but while remaining willfully blind to the infringement. Defendants have and continue to induce infringement by its customers and end-users by supplying them with instructions on how to operate the infringing technology in an infringing manner, while also making publicly available information on the infringing technology via Defendants’ website, product literature and packaging, and other publications.

90. Plaintiff has suffered damages as a result of Defendants’ direct and indirect infringement of the ’871 Patent in an amount to be proven at trial.

91. Plaintiff has suffered, and will continue to suffer, irreparable harm as a result of Defendants’ infringement of the ’871 Patent, for which there is no adequate remedy at law, unless Defendants’ infringement is enjoined by this Court.

[95] The ’871 Patent Family was cited as relevant prior art against Amazon Technologies, Inc.’s U.S. Patent Application No. 16/129,632, which was filed on September 12, 2018, and issued as U.S. Patent No. 11,108,687 on August 31, 2021.

DEMAND FOR JURY TRIAL

Plaintiff hereby demands a jury for all issues so triable.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff prays for relief against Defendants as follows:

a. Entry of judgment declaring that Defendants have directly and/or indirectly infringed one or more claims of each of the Patents-in-Suit;

b. An order pursuant to 35 U.S.C. § 283 permanently enjoining Defendants, their officers, agents, servants, employees, attorneys, and those persons in active concert or participation with them, from further acts of infringement of the Patents-in-Suit;

c. An order awarding damages sufficient to compensate Plaintiff for Defendants' infringement of the Patents-in-Suit, but in no event less than a reasonable royalty, together with interest and costs;

d. Entry of judgment declaring that this case is exceptional and awarding Edge its costs and reasonable attorney fees under 35 U.S.C. § 285; and

e. Such other and further relief as the Court deems just and proper.

Dated: November 1, 2024

Respectfully submitted,

/s/ Vincent J. Rubino, III
Alfred R. Fabricant
NY Bar No. 2219392
Email: ffabricant@fabricantllp.com
Peter Lambrianakos
NY Bar No. 2894392
Email: plambrianakos@fabricantllp.com
Vincent J. Rubino, III
NY Bar No. 4557435
Email: vrubino@fabricantllp.com
FABRICANT LLP
411 Theodore Fremd Avenue
Suite 206 South
Rye, New York 10580

Telephone: (212) 257-5797
Facsimile: (212) 257-5796

ATTORNEYS FOR PLAINTIFF
EDGE NETWORKING SYSTEMS LLC