Tampereen ammattikorkeakoulu

Using AWS Secrets Manager with Kubernetes

Karl-Juhan Jurvanen

BACHELOR'S / THESIS
December 2021

Degree Programme in ICT Engineering
Software Engineering

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

JURVANEN, KARL-JUHAN:
AWS Secrets Manager -palvelun käyttö Kubernetes-ympäristössä

Opinnäytetyö 28 sivua, joista liitteitä 6 sivua
Joulukuu 2021

Nykyaikaisissa ohjelmistoissa on useimmiten joitakin salassa pidettäviä tietoja, kuten salausavaimia ja käyttäjien tunnistetietoja. Näiden salaisuuksien turvallinen säilytys ja käsittely ovat tärkeä osa ohjelmiston tietoturvaa. Markkinoilla on useita kaupallisia ja ilmaisia tuotteita helpottamaan salaisuuksien käsittelyä. Opinnäytetyön tavoitteena oli tutustua Amazon Web Servicen tarjoamiin palveluihin, joita voidaan käyttää salaisuuksien säilyttämiseen. Erityisesti työssä keskityttiin AWS Secrets Manager -palveluun ja sen käyttämiseen.

Opinnäytetyössä tutkittiin AWS Secrets Manager -palvelun soveltuvuutta olemassa olevaan ohjelmistoprojektiin, joka on asennettu Kubernetes-klusteriin Amazonin Kubernetes -alustalla. Opinnäytetyössä tehtiin testitoteutus Secrets Managerin integroinnista Kubernetes-alustaan. Testitoteutus sisältää tarvittavat AWS-resurssit ja Kubernetes-konfiguraatiot, joilla integraatio saatiin tehtyä. Resurssien luonnin automatisointiin käytettiin AWS CloudFormationia.

Työn tutkimusvaiheessa selvisi, että Amazon ei tarjoa työkaluja Secrets Managerin käyttöön Kubernetes-klusterista. Projektissa päätettiin käyttää avoimen lähdekoodin Kubernetes External Secrets -projektia, jolla integraatio onnistui suoraviivaisesti. External Secrets on tarkoitettu Kubernetes-ympäristöön, mutta sitä voidaan käyttää monien salaisuuksien hallintaohjelmien kanssa.

Työssä tehty testitoteutus onnistui hyvin ja sen perusteella Secrets Manager todettiin soveltuvaksi toimeksiantajan vaatimuksiin. Se päätettiin ottaa käyttöön varsinaiseen projektiin.

Avainsanat: AWS, Secrets Manager, Kubernetes, IAM, CloudFormation

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

JURVANEN, KARL-JUHAN:
Using AWS Secrets Manager with Kubernetes

Bachelor's thesis 28 pages, appendices 6 pages
December 2021

Many modern software projects contain some information that needs to be kept private. These can be credentials to databases, cryptographic keys, or user credentials. Keeping these secrets secure is an important part of the project's security. This includes storing the secrets, access management and possibility to change the secrets if they are compromised. There are multiple commercial and open-source products meant for managing secrets.

The goal of this bachelor's thesis was to evaluate available secret management tools available on Amazon Web Services. The thesis focused on using AWS Secrets Manager as part of an existing software project running on Kubernetes. A proof-of-concept integration of AWS Secrets Manager and the Kubernetes application was created. The thesis explored creating required AWS resources and Kubernetes configuration to use Secrets Manager. AWS CloudFormation was used to automate resource creation.

Amazon does not offer a native tool for accessing Secrets Manager from a Kubernetes application, so part of the research was to find a suitable alternative. The Kubernetes External Secrets open-source project was selected. External Secrets is meant for Kubernetes, but it can be used with different secret management services. The implementation with Secrets Manager and Kubernetes External Secrets was found to work well. The commissioner decided to use it in the customer project.

The thesis is aimed at specialists with basic understanding of Amazon Web Services and Kubernetes. It is not meant to be used as a guide or walkthrough of using Secrets Manager, but as a proof-of-concept implementation.

Key words: AWS, Secrets Manager, Kubernetes, IAM, CloudFormation

CONTENTS

ABBREVIATIONS AND TERMS

Kubernetes	Container orchestration tool for scalable systems
AWS	Amazon Web Services
ARN	Amazon Resource Name Unique identifier for resources in AWS
IAM	AWS Identity and access management Provides access control to AWS resources
EKS	AWS Elastic Kubernetes Service Managed Kubernetes service provided by Amazon
YAML	Markup language used for providing configuration for systems like Kubernetes
API	Application Programming Interface Interface to access an application programmatically
AWS CLI	Amazon Command-line interface Tool used to access AWS APIs via command line interface
OIDC	OpenID Connect Authentication tool using OAuth 2.0 authorization framework
RBAC	Role-based access control Method of regulating user access to Kubernetes resources based on roles

1 INTRODUCTION

Secret management is an important aspect of modern software development. Most applications have sensitive data such as database credentials, API keys or cryptographic keys. Managing these secrets involves storing them securely, restricting users and applications that can access them and rotating them if they are compromised.

There are multiple commercial and open-source products available to make secret management easier to do securely. This Bachelor's thesis focuses on using the Amazon Secrets Manager service, which offers secure storage and access to different types of secrets in the cloud. The goal of the thesis project is to test Secrets Manager and evaluate the service in combination with an application running on Amazon Elastic Kubernetes Service (EKS).

This report will introduce technologies used and outline a proof-of-concept project to test integration between Secrets Manager and the application using secrets. AWS does not provide a native integration between Secrets Manager and EKS, which prompted the need for this thesis.

This thesis is made as part of research for a larger software project for LAAVAT OY. The company offers security solutions for customers producing Internet of Things (IoT) devices. IoT devices, or devices connected to internet are becoming more and more common for many industries. Reliability and security of the devices is vital for the customers. (LAAVAT, n.d.)

> Our goal is to make it easy for our customers to build secure IoT devices without a need to invest considerably in embedded security and cryptography expertise. We help our customers to achieve accelerated time to market with reduced implementation costs and, at the same time, compliance with various IoT security standards and regulations. (LAAVAT, n.d.)

The LAAVAT signing service is a cloud-based solution offering secure execution of cryptographic operations for IoT devices. The product is built using microservice architecture based on Kubernetes. The primary platform used is Amazon Web Services (AWS), so the thesis also focuses on AWS technologies. The thesis will not describe the signing service, instead a separate proof-of-concept project is used as an example.

2 TECHNOLOGIES

2.1 Kubernetes

Kubernetes is an open-source software platform to run containerized applications. It allows orchestrating large numbers of containers spread over multiple worker machines. (Kubernetes documentation n.d.)

Kubernetes was originally developed by Google and first released in 2014. The ownership of Kubernetes project was later moved to the Cloud Native Computing Foundation. (Kubernetes documentation n.d.)

The name Kubernetes originates from Greek language, meaning helmsman or pilot. Nautical themes repeat in the Kubernetes ecosystem in naming of projects and tools. The word Kubernetes is often abbreviated to k8s. (Kubernetes documentation n.d.)

The primary units of Kubernetes cluster are pods and nodes. Nodes are the worker machines that to the cluster. Each cluster has at least one worker node. The cluster also has several control nodes that form the clusters control plane. The control plane handles cluster operations like networking and pod scheduling. When using a managed Kubernetes service like Amazon's Elastic Kubernetes Service (EKS), the cloud provider manages the control plane.

Kubernetes is a complex system with many components that are outside the scope of this thesis. The following chapters discuss the important parts of Kubernetes.

2.1.1 Kubernetes pods

The primary unit in a Kubernetes cluster is a pod. The pod encapsulates one or more containers and their configuration. When a pod is created, the control plane assigns it to an available node in the cluster. Pods are typically deployed

as part of a Deployment, which takes care of the pod lifecycle. With deployments the pod can be replicated for scalability and failure tolerance. If a pod in deployment fails, Kubernetes will replace it with a new one.

2.1.2 Kubernetes secrets

Pods sometimes need some confidential information to work. This can for example be credentials to a database or a cryptographic private key. A secret is object in Kubernetes to store these sensitive items separate from the pod. The secret data can be passed to a pod as container environment variables or as a file mounted to the container. (Kubernetes documentation n.d.)

Secrets in Kubernetes are stored in the cluster's underlying data storage. By default, Secrets are stored unencrypted, so it is up to the cluster operator to make sure secrets are encrypted at rest. Amazon EKS offers possibility to add additional layer of encryption for secrets using customer managed encryption keys. (Kubernetes documentation n.d.)

2.1.3 Kubernetes role-based authorization

Role-based access control (RBAC) is a way to restrict users' or applications' access to Kubernetes resources based on the role of the user. For example, it can be used to deny users access to read secrets in a specific namespace.

Using RBAC requires Role and RoleBinding resources. The role specifies resources and actions permitted, and role binding is used to attach the role to a user or application service account. Role and role binding can be tied to a specific namespace or made cluster-wide by using ClusterRole and ClusterRoleBinding instead.

Appendix 3 is an example Kubernetes manifest that creates a role and role binding for our sample application.

2.1.4 Helm releases

Helm is a package manager for Kubernetes. The Helm project is part of the Cloud Native Computing Foundation. Helm has become a popular way to manage and deliver software aimed for Kubernetes.

A program intended to be installed with helm is packaged into a helm chart. The chart contains any Kubernetes objects created in the installation like deployments, pods, networking configuration, etc. It also contains configuration parameters that can be changed when installing or updating.
In this thesis project helm is mainly used to install required external programs that suggest using helm for installation.

2.2 Amazon Web Services

Amazon Web Services (AWS) is a leading Cloud computing platform. Amazon advertises that AWS Cloud runs in 25 different geographical regions around the world. AWS offers hundreds of different services starting with computing and related services such as storage, monitoring, networking, and access control. AWS also offers many other more exotic services involving for example A.I. and Machine Learning that are outside the scope of this thesis. (Amazon AWS overview)

Relevant AWS resources that are referenced in the thesis are Elastic Kubernetes Service (EKS), CloudFormation, Secrets Manager and Identity and Access Management (IAM). LAAVAT product utilizes other services in the background such as databases and storage through S3, but these are not the focus for this thesis.

2.2.1 CloudFormation

CloudFormation is a tool to automate the creation of AWS resources. When first testing a service, it is typical to use the Amazon console to create resources manually. However, when using the service as part of a product, automation is important. CloudFormation is a tool that allows deploying different resources with specific configuration.

When using CloudFormation, the user creates a template file that contains AWS resources to be created by CloudFormation. This file can be in JSON or YAML format. YAML is used to demonstrate here. The template can also contain things like parameters, conditions, and output values to help automate resource creation.

The template can be deployed with possible parameters. When deployed, CloudFormation creates a stack that contains resources specified in the template.

2.2.2 AWS Secrets Manager

Secrets Manager is a managed service for protecting secrets used by applications. It stores the secrets in an encrypted state, and customer can use their own cryptographic keys for extra security. Secrets Manager provides programmatic way to create, access and update the secrets through the Amazon API.

2.2.3 AWS Identity and Access manager (IAM)

IAM is the main way to grant users and programs access to resources on AWS. Typically, access is granted by creating a role and attaching a policy to the role. A user who can assume this role is granted access to permissions in the policy.

For a user accessing AWS resources, username and password are often used. Machine credentials are needed to grant machine resource like a kubernetes pod

access to AWS. These can be simple access key and secret or more complex dynamic credentials.

Static credentials can be used for granting access to the kubernetes pod. This would mean creating an IAM user for the application and generating an access key. Then the key id and secret can be passed to the pod using a kubernetes secret.

Access can also be granted using machine credentials of the kubernetes node when running on EKS. AWS automatically creates a machine user for virtual machines running on AWS, this also applies to kubernetes nodes. A policy can be added to the machine role. A limitation with this approach is that granting access to the node means that every pod running on the node gets the same access. This can be a problem when trying to secure the environment.

Third way to grant access to pods running on EKS is to use IAM roles for service accounts. This allows granting permissions only to the applications that need them.

2.2.4 IAM roles for service accounts

IAM roles for service accounts is a way to tie a kubernetes service account to AWS IAM role. This allows granting access to AWS resources to specific pods that have a specific service account. On EKS configuring pod IAM access is straightforward, because EKS automatically creates an IAM identity provider for pods. (Amazon, n.d.)

AWS allows using OpenID connect (OIDC) to grant access to federated identities. IAM roles for service accounts uses OIDC to allow authenticating pods to AWS APIs. AWS EKS hosts an OIDC discovery endpoint for clusters that can be added to AWS IAM as a provider. (Amazon n.d.)

2.3 Kubernetes External Secrets

Kubernetes External Secrets is an open-source project that allows using external secret managers from a kubernetes cluster. It supports many of commonly used secret manager products, such as AWS secrets manager, Azure Key Vault and Hashicorp Vault. Here we will focus on AWS Secrets Manager. (Kubernetes External Secrets n.d.)

Kubernetes External Secrets project was started as an open-source project by GoDaddy. It was later combined with other similar projects and moved to the external-secrets Github organization. After this it has become a popular way to access external secret management backends from Kubernetes. (Kubernetes External Secrets n.d.)

Installing the External Secrets Helm chart creates a deployment that runs the External Secrets controller, and it also extends the kubernetes API with a new ExternalSecret resource. Secrets stored in AWS Secrets Manager can be used by creating an ExternalSecret resource with Secrets Manager as the backend. The controller uses this new ExternalSecret to create a kubernetes secret with data from Secrets Manager. Figure 1 shows steps to fetch secret data from AWS Secrets Manager. External Secrets can also be setup to poll Secrets Manager for changes. This way the kubernetes secret can be synced with any changes made in AWS end.

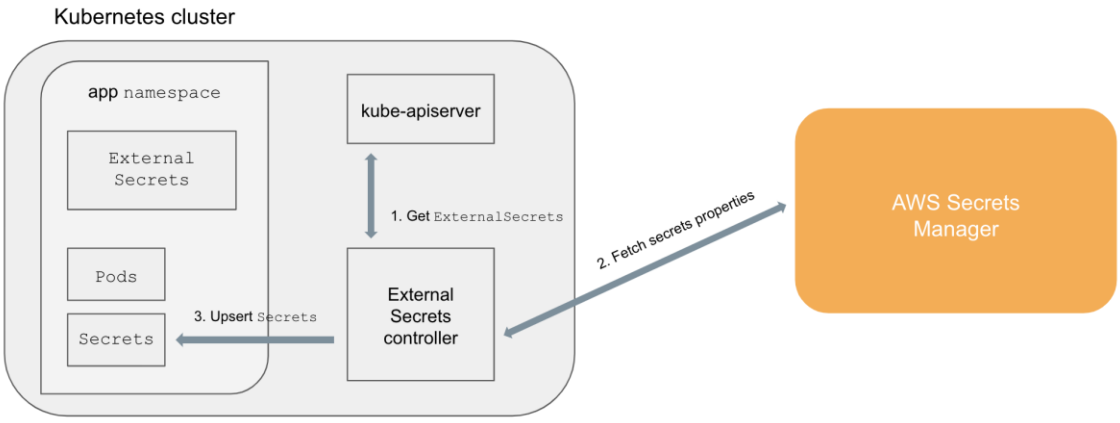

FIGURE 1. External Secrets creation sequence (Kubernetes External Secrets n.d.)

3 PROJECT

The goal of the project is to create a proof of concept for using AWS Secrets Manager to store secrets used in Kubernetes cluster. This chapter will go through creating a cluster on AWS Elastic Kubernetes Service, creating a secret in Secrets Manager and steps needed to use the secret from the cluster. IAM role and policy is used to grant the application access to secret.

3.1 Creating Kubernetes cluster on AWS EKS

AWS EKS clusters can be created and managed in multiple ways, either using Amazon web console, CloudFormation, AWS CLI or eksctl, a purpose-built command-line interface for EKS. For this project CloudFormation was chosen to manage all resources on AWS.

A sample CloudFormation template for deploying a Kubernetes cluster on EKS can be found in **Appendix 1**. Using the template assumes that a VPC and subnets for the cluster are created beforehand. The template can be used via AWS CLI with following command:

```
aws cloudformation deploy
    --template-file ./cluster.yml
    --stack-name <STACK_NAME>
    --parameter-overrides   VpcId=<VPC_ID>
                            SubnetsPrivate=<PRIVATE_SUBNETS>
                            SubnetsPublic=<PUBLIC_SUBNETS>
    --capabilities CAPABILITY_IAM
```

When the stack is created, Cluster OpenID Connect issuer URL is stored in the stack outputs. This is needed when pods in the cluster need to be given access to Amazon resources. The issuer URL looks like example https://oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E. This value is used in next step.

3.1.1 Cluster IAM provider

After Kubernetes cluster is created on EKS, it needs to be configured as a trusted identity provider for the AWS account. This makes it possible for workloads in the cluster to access resources in the AWS account. In this project we configure Kubernetes pods to access secrets in AWS Secrets Manager, but the same method can be used for other resources.

AWS IAM allows different types of external identity providers to be added. EKS uses an OpenID Connect provider. The provider can be added using AWS console or CLI. The cluster provider cannot be created via CloudFormation. Here we use the command-line interface.

```
aws iam create-open-id-connect-provider
    --url  https://oidc.eks.us-west-
    2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
    --thumbprint-list 9e99a48a9960b14926bb7f3b02e22da2b0ab7280
    --client-id-list "sts.amazonaws.com"
```

The command takes three parameters: “--url” the provider URL we got when creating the cluster, this is unique for our cluster. --client-id-list List of client IDs. sts.amazonaws.com is expected client value for this use case. “--thumbprint-list” list of server certificate thumbprints for the OpenID Connect (OIDC) identity provider's server certificates. The server certificate thumbprint used is the hex-encoded SHA-1 hash value of the X.509 certificate used by the domain where the OpenID Connect provider makes its keys available. It is always a 40-character string. (Amazon, n.d.) For OIDC provider the certificate thumbprint here needs to be the Amazon root certificate authority, whose SHA-1 thumbprint is 9e99a48a9960b14926bb7f3b02e22da2b0ab7280. This is not unique for each cluster.

After this step the workloads inside our cluster can authenticate to access AWS resources through IAM. Right now, we have not created any roles for our workloads, so they do not have any permissions. The next steps are to create a sample secret to AWS Secrets manager and a role that has permissions to retrieve the secret value. Then we can create an External Secret to the cluster that will access the Secrets Manager secret and sync it inside the cluster.

3.2 Creating Secrets on AWS

Amazon Secrets Manager allows creating secrets via the console, CloudFormation or the command line interface. Here CloudFormation is used as an example. Below sample CloudFormation template creates a secret with fields username and password. Using GenerateSecretString-property allows Secrets Manager to generate a random password value for us using the restrictions set with PasswordLength and ExcludeCharacters fields. The ExcludeCharacters field is useful if the target application does not allow certain characters in the generated password.

```
AWSTemplateFormatVersion: "2010-09-09"

Resources:
  RDSInstanceApplicationSecret:
    Type: AWS::SecretsManager::Secret
    Properties:
      Description: 'Application user credentials to RDS database'
      Name: 'TestAppSecret'
      GenerateSecretString:
        SecretStringTemplate: '{"username": "admin"}'
        GenerateStringKey: 'password'
        PasswordLength: 16
        ExcludeCharacters: '"@''/\^$'
```

This creates a secret with a value like below with a randomly generated password. This secret can now be used by the application.

```
{
    "password": "7NL|K3d(9sfxlnJV",
    "username": "admin"
}
```

Secrets Manager allows more specific configuration of the secret if necessary. This includes adding custom encryption keys and replicating the secret to multiple regions.

3.3 IAM Role to access secrets

To access the secret from our cluster using a pod identity we must create an IAM Role associated with the pod. This can again be done though console, CLI or CloudFormation. **Appendix 2** shows a CloudFormation template that creates IAM Role and Policy for a pod in the cluster to access the secret.

The role is tied to the OIDC provider and additionally uses condition to only allow pods from a specific cluster namespace and service account. This can be used to ensure that only specific pods in the cluster can access the AWS resources. The policy tied to the role specifies what the role is allowed to do. In this case the policy allows read-only access to the listed secrets in AWS Secrets Manager.

3.4 Installing External Secrets controller

Before we can create an external secret, we must deploy the Kubernetes External Secrets application to the cluster. The application is provided with a Helm chart, so we use it for installation. The helm chart can be installed with the following commands.

```
helm repo add external-secrets https://external-
secrets.github.io/kubernetes-external-secrets/
helm install <RELEASE_NAME> external-secrets/kubernetes-external-secrets
```

The first command adds the external-secrets helm repository from the internet and second command installs it. The chart provides additional configuration, but it is not necessary for this proof-of-concept installation. (Kubernetes External Secrets n.d.)

After running the commands, the installation can be verified with following command. This should show external-secrets pod running.

```
kubectl get pods
```

3.5 Creating External Secret

The external secrets controller can now be used to create external secret – type resources to the cluster. Below is a YAML template for an external secret from AWS secrets manager.

```yaml
apiVersion: "kubernetes-client.io/v1"
kind: ExternalSecret
metadata:
  name: db-secret
  namespace: app
spec:
  backendType: secretsManager
  roleArn: arn:aws:iam::123456789012:role/appExternalSecrets
  region: eu-west-1
  data:
    - key: TestAppSecret
      name: password
    - key: TestAppSecret
      name: username
```

The resource can be created with kubectl command:

```
kubectl apply -f external-secret.yaml
```

In the template the resource type is ExternalSecret, a custom type created by the Kubernetes External Secrets helm deployment. Metadata fields specify the resource name and destination namespace. Field roleArn specifies the AWS IAM role to use when attempting to read the secret from AWS Secrets Manager. This must match the role ARN created in chapter 4.3. The data array values specify which secrets to access from AWS. The keys must match the secret name created in chapter 4.2 and the name specifies which field to read from the secret. Since we want both username and password, we have two fields.

When the ExternalSecret is created, the Kubernetes External Secrets controller attempts to read the secret value from the backend. In the AWS Secrets

Manager case, it uses the AWS IAM role specified by roleArn to access Secrets Manager.

If the controller has the correct access rights to read the secret from Secrets Manager is synced into a Kubernetes Secret in the namespace of the External Secret. The controller also watches for changes in the backend secret values. This means the secret can be updated externally and the controller updates it in the cluster. The External Secret can be checked with kubectl. The kubectl output shows that the secret was created successfully, and the value was synced eight seconds ago.

```
$ kubectl get externalsecret -n app
NAME       LAST SYNC   STATUS    AGE
db-secret  8s          SUCCESS   2d
```

3.6 Using the secret in an application

The application using the secret in the same way as if it was a traditional kubernetes secret object. The application does not know that the secret is from AWS Secrets Manager. Following sample YAML snippet shows a simple application that uses a secret.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-application
spec:
  containers:
  - name: test
    image: alpine
    volumeMounts:
    - name: db-secret
      mountPath: "/data"
      readOnly: true
  volumes:
  - name: secret
```

```
    secret:
      secretName: db-secret
```

A volume is added to the pod specification for the secret, which is mounted to data folder. The application can use the secret by reading the file created inside the volume. The Kubernetes External Secrets controller will update the volume if the secret is changed in AWS. This means that the secret can be rotated without needing to make manual updates inside the cluster. The application will automatically get the new secret value from Secrets Manager.

4 DISCUSSION

When starting the project, the goal was to see if the LAAVAT product could use AWS Secrets Manager without adding too much complexity. The upsides were clear, the product was already on the AWS platform and Secrets Manager provides secure storage for secrets with access control. However, the difficulty was that EKS did not have native tools for integrating with Secrets Manager.

Key technologies that made the project usable were Kubernetes External Secrets -project and IAM roles for service account by Amazon. Kubernetes External Secrets proved to be very flexible and easy to use integration for AWS Secrets Manager. A large upside was that secrets are mounted as native Kubernetes secrets. This meant that no changes were needed to the application itself. It also synchronizes the secrets between Secrets Manager and the cluster. This means that they can be updated in Secrets Manager, and they automatically update in the cluster.

The possibility of adding IAM roles to pod service accounts makes secrets accessible without static AWS credentials.

Moving forward, Kubernetes External Secrets has been included in LAAVAT Kubernetes clusters running on AWS. All runtime secrets needed by the application have been moved to AWS Secrets Manager, which makes managing them easier.

REFERENCES

LAAVAT. 2021. LAAVAT Whitepaper. Read on 01.12.2021.
https://www.laavat.com

Kubernetes documentation. N.d. Read on 24.10.2021.
https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Amazon AWS overview. Released on 05.08.2021. Read on 18.10.2021.
https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html

Amazon. IAM Roles for Service Accounts. User guide. N.d. Read on 02.08.2021.
https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html

Kubernetes External Secrets. N.d. Read on 10.11.2021
https://github.com/external-secrets/kubernetes-external-secrets

APPENDICES

Appendix 1. Cloudformation template to create EKS cluster

```
AWSTemplateFormatVersion: "2010-09-09"

Description: "Sample EKS cluster"

Parameters:

  VpcId:
    Type: String
  SubnetsPrivate:
    Type: CommaDelimitedList
  SubnetsPublic:
    Type: CommaDelimitedList

  ClusterName:
    Type: String
    Description: EKS Kubernetes cluster name.
    Default: Test cluster 01

Resources:

  ClusterServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action:
              - sts:AssumeRole
            Effect: Allow
            Principal:
              Service:
                - eks.amazonaws.com
        Version: "2012-10-17"
      ManagedPolicyArns:
        - Fn::Sub:
arn:${AWS::Partition}:iam::aws:policy/AmazonEKSClusterPolicy
        - Fn::Sub:
arn:${AWS::Partition}:iam::aws:policy/AmazonEKSVPCResourceController
      Tags:
        - Key: Name
          Value:
            Fn::Sub: ${AWS::StackName}/ServiceRole

  ControlPlaneSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
```

```
      GroupDescription: Communication between the control plane and
worker nodegroups
      Tags:
        - Key: Name
          Value:
            Fn::Sub: ${AWS::StackName}/ControlPlaneSecurityGroup
      SecurityGroupEgress:
        - IpProtocol: "-1" # all
          FromPort: -1 # all
          ToPort: -1 # all
          CidrIp: 0.0.0.0/0
          Description: "Allow outbound traffic"
      VpcId:
        Ref: VpcId

  EKSCluster:
    Type: 'AWS::EKS::Cluster'
    Properties:
      Name:
        Ref: ClusterName
      RoleArn:
        GetAtt: ClusterServiceRole.Arn

      ResourcesVpcConfig:
        SecurityGroupIds:
          - Ref: ControlPlaneSecurityGroup
        SubnetIds:
        # Join public and private subnets to one list
          Fn::Split:
            - ","
            - !Join
              - ","
              - [!Join [",",!Ref SubnetsPrivate], !Join [",",!Ref
SubnetsPublic]]

Outputs:
  CertificateAuthorityData:
    Value:
      Fn::GetAtt:
        - EKSCluster
        - CertificateAuthorityData
  ClusterOidcProvider:
    Value:
      Fn::Select:
        - "1"
        - Fn::Split:
            - "//"
            - Fn::GetAtt:
                - EKSCluster
                - OpenIdConnectIssuerUrl
  Endpoint:
```

```
    Value:
      Fn::GetAtt:
        - EKSCluster
        - Endpoint
    Export:
      Name:
        Fn::Sub: ${AWS::StackName}::Endpoint
```

Appendix 2. CloudFormation template for pod IAM Role and Policy

```
AWSTemplateFormatVersion: "2010-09-09"

Description: "Iam role and policy for Kubernetes service account"

Parameters:
  ClusterOidcProvider:
    Type: String
    Description: "Cluster OpenID Connect provider URL"

  NameSpace:
    Type: String
    Description: "Namespace of the service that will use this IAM Role"

  ServiceAccountName:
    Type: String
    Description: "Name of the service account that will use this IAM
Role"

  SecretArns:
    Type: CommaDelimitedList
    Description: "Comma separated list of secret ARNs that external
secrets service can read"

  SecretEncryptionKMSKeyARN:
    Type: String
    Description: "ARN for KMS key used to encrypt kubernetes secrets"

Resources:
  IamRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Fn::Sub: |
          {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Principal": {
                    "Federated": "arn:aws:iam::${ AWS::AccountId }:oidc-
provider/${ ClusterOidcProvider }"
                },
                "Action": "sts:AssumeRoleWithWebIdentity",
                "Condition": {
                    "StringEquals": {
                      "${ ClusterOidcProvider }:sub":
"system:serviceaccount:${ NameSpace }:${ ServiceAccountName }"
                    }
```

```
                }
              }
            ]
          }
      RoleName:
        Fn::Sub: "${AWS::StackName}-iam-service-account"
      Tags:
        - Key: Name
          Value:
            Fn::Sub: ${AWS::StackName}-iam-service-account

  IamPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - "secretsmanager:GetResourcePolicy"
              - "secretsmanager:GetSecretValue"
              - "secretsmanager:DescribeSecret"
              - "secretsmanager:ListSecretVersionIds"

            Effect: Allow
            Resource:
              Ref: SecretArns
          - Action:
              - kms:DescribeKey
              - kms:Encrypt
              - kms:Decrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey
              - kms:GenerateDataKeyWithoutPlaintext
            Effect: Allow
            Resource:
              - Ref: SecretEncryptionKMSKeyARN
              - Fn::Sub:
arn:${AWS::Partition}:kms:${AWS::Region}:${AWS::AccountId}:key/*
        Version: "2012-10-17"
      PolicyName:
        Fn::Sub: ${AWS::StackName}-Policy
      Roles:
        - Ref: IamRole

Outputs:
  ExternalSecretsRoleArn:
    Value:
      Fn::GetAtt:
        - IamRole
        - Arn
```

Appendix 3. Kubernetes RBAC example manifest YAML file

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: app
  name: secret-read-only
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
---

apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "sample-app" service account
# to read secrets in the "app" namespace.
kind: RoleBinding
metadata:
  name: read-secrets
  namespace: app
subjects:
- kind: ServiceAccount
  name: sample-app
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: secret-read-only
  apiGroup: rbac.authorization.k8s.io
```