Building A Secure Containerized Nginx Web Server With EKS Container Orchestration Platform

ABSTRACT

This implementation guide explores AWS EKS container orchestration platform, configured with Cert Manager and AWS Private CA for workload certificate issuance, dynamic storage provisioning through EBS CSI driver and load balancing with AWS Load balancer Controller. The resulting build is a Secure Containerized Nginx Web Server. The IAC tool employed for this implementation is Terraform.

SitadConsulting

[Course title]

Table of Contents

Secure containerized NGINX Web Server Implementation Guide

1.0 Introduction

Today, we are making available a series of Terraform modules, particularly, for AWS and Kubernetes providers. These modules are based around the concept we have coined Terraform "micro-module". With this approach, we set the basic unit of a module to be a resource object.

Thanks to the innovative work by the folks at Hashi, we have leveraged both locals and for_each as core part of this Terraform micro-module idea. This we believe allows for a more expressive way of defining infrastructure solutions built from composable micro-modules. Additionally, these two constructs allowed us to decouple the code base from the configuration required to build out the infrastructure, effectively lowering the timescale for infrastructure provisioning.

2.0 Usage Example and Component Versions

We provided a usage example of this idea in which we had deployed a secure containerized NGINX Web Server instance, before we proceed further, we would like to provide for reference the versions of the components we have employed in executing this solution (just so you can avoid surprises!)

aws-load-balancer-controller	= v2.5.2
aws-ebs-csi-driver	= v1.20.0
cert-manager	= v0.1.0
secrets-store-csi-driver	= v1.3.4
secrets-store-csi-driver-provider-aws	= v0.3.3
aws-privateca-issuer	= v1.2.5
aws provider	= v5.3.0
Kubernetes provider	= v2.21.0
helm provider	= v2.9.0

3.0 Clone Repository and Initialize Modules and Providers

To begin, please clone the repository to your workspace. Execute the command shown below:

cd /home/dev01 (this workspace may be different in your own case)
git clone ***https://github.com/sitadconsulting/aws_infra.git***
cd aws_infra/projects/proj12 (review the files in this directory and update the “dev.auto.tfvars” file to reflect values from your own environment).

At the minimum, please update these variables

caller_user_id = <A user with Administrator privilege in your AWS account>
public_access_cidrs = < A list of IP address from which you can access the EKS cluster>
node_group_public_key = < Public key to access cluster worker nodes>

After editing the “dev.auto.tfvars” file, proceed to initialize the modules and providers for executing the first stage of the Plan and Apply Cycle Stages as shown below:

proj12]$ ***terraform init***

Initializing the backend...
Initializing modules...
- *acmpca_ca in ../../modules/acmpca_certificate_authority*
- *acmpca_ca_certificate in ../../modules/acmpca_certificate_authority_certificate*
- *acmpca_certificate in ../../modules/acmpca_certificate*
- *acmpca_permission in ../../modules/acmpca_permission*
- *autoscaling_group in ../../modules/autoscaling_group*
- *awsgatewayapi_iam_policy_attachment in ../../modules/iam_role_policy_attachment*
- *awsgatewayapi_iam_role in ../../modules/iam_role*
- *awspca_iam_policy_attachment in ../../modules/iam_role_policy_attachment*
- *awspca_iam_role in ../../modules/iam_role*
- *ec2_container_policy_attachment in ../../modules/iam_role_policy_attachment*
- *eip in ../../modules/eip*
- *eks_cluster in ../../modules/eks_cluster*
- *eks_cluster_iam_policy_attachment in ../../modules/iam_role_policy_attachment*
- *eks_cluster_iam_role in ../../modules/iam_role*
- *eks_cni_policy_attachment in ../../modules/iam_role_policy_attachment*
- *eks_ebs_csi_driver_iam_policy_attachment in ../../modules/iam_role_policy_attachment*
- *eks_ebs_csi_driver_iam_role in ../../modules/iam_role*
- *eks_lb_controller_iam_policy in ../../modules/iam_policy*
- *eks_lb_controller_iam_policy_attachment in ../../modules/iam_role_policy_attachment*
- *eks_lb_controller_iam_role in ../../modules/iam_role*
- *eks_node_group_iam_role in ../../modules/iam_role*
- *iam_access_key in ../../modules/iam_access_key*
- *iam_instance_profile in ../../modules/iam_instance_profile*
- *iam_openid_connect_provider in ../../modules/iam_openid_connect_provider*
- *iam_user in ../../modules/iam_user*
- *iam_user_login_profile in ../../modules/iam_user_login_profile*
- *iam_user_policy in ../../modules/iam_policy*
- *iam_user_policy_attachment in ../../modules/iam_user_policy_attachment*
- *internet_gateway in ../../modules/internet_gateway*
- *internet_gateway_route in ../../modules/route*
- *key_pair in ../../modules/key_pair*
- *launch_template in ../../modules/launch_template*
- *local_file in ../../modules/local_file*
- *nat_gateway in ../../modules/nat_gateway*
- *nat_gateway_route in ../../modules/route*
- *nginx_db_secret_iam_policy_attachment in ../../modules/iam_role_policy_attachment*
- *nginx_db_secret_iam_role in ../../modules/iam_role*
- *private_route_table in ../../modules/route_table*
- *private_route_table_association in ../../modules/route_table_association*

- private_subnet in ../../modules/subnet
- public_route_table in ../../modules/route_table
- public_route_table_association in ../../modules/route_table_association
- public_subnet in ../../modules/subnet
- secretsmanager_secret in ../../modules/secretsmanager_secret
- secretsmanager_secret_version in ../../modules/secretsmanager_secret_version
- security_group in ../../modules/security_group
- vpc in ../../modules/vpc
- vpc_dhcp_options in ../../modules/vpc_dhcp_options
- vpc_dhcp_options_assoc in ../../modules/vpc_dhcp_options_assoc
- vpc_security_group_rule in ../../modules/vpc_security_group_rule
- worker_node_policy_attachment in ../../modules/iam_role_policy_attachment

Initializing provider plugins...
- Finding hashicorp/kubernetes versions matching "2.21.0"...
- Finding hashicorp/tls versions matching "4.0.4"...
- Finding latest version of hashicorp/template...
- Finding hashicorp/helm versions matching "2.9.0"...
- Finding hashicorp/time versions matching "0.9.1"...
- Finding hashicorp/aws versions matching "5.3.0"...
- Finding hashicorp/local versions matching "~> 2.3.0"...
- Installing hashicorp/time v0.9.1...
- Installed hashicorp/time v0.9.1 (signed by HashiCorp)
- Installing hashicorp/aws v5.3.0...
- Installed hashicorp/aws v5.3.0 (signed by HashiCorp)
- Installing hashicorp/local v2.3.0...
- Installed hashicorp/local v2.3.0 (signed by HashiCorp)
- Installing hashicorp/kubernetes v2.21.0...
- Installed hashicorp/kubernetes v2.21.0 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.4...
- Installed hashicorp/tls v4.0.4 (signed by HashiCorp)
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)
- Installing hashicorp/helm v2.9.0...
- Installed hashicorp/helm v2.9.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

4.0 Review Main.tf File and Get Familiar with The Terraform Plan and Apply Cycle Stages

In the main.tf file of the root module, we have indicated with comment headers, the required Terraform "Plan and Apply" cycle stages for executing the deployment (This could be handled neatly with a CICD pipeline -out of scope)

There are a total of Six commented stages beginning with stage 2, please only uncomment the code when you need to execute a particular stage. After executing stage 2, you should have a fully functioning EKS Cluster with worker nodes joined to the cluster, ready to start layering on your Kubernetes workloads. (***Please Note: After, uncommenting the lines at each stage, remember to execute a "terraform init" in other to initialize the new modules you are introducing***.)

After executing stage 2 apply, it is important to manually install aws-load-balancer-controller CRDs (until the HELM resource is able to handle the CRDS installs automatically.) in order to ensure successful installation of the remaining aws-load-balancer-controller components. Please execute the command below:

kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller/crds?ref=master"

If at any point between stage 2 and stage 6, you notice a failed apply, please re-execute the apply again, as there could be timing issues with many moving parts!

4.1 Stage Six, Plan and Apply Cycle – Review and Fix Deployment Issues if Any.

At stage 6, you should have a running Nginx container which you can shell into. However, if there is a failure with starting the container at this stage, then check the logs for the "ebs-csi-controller" - use the command below:

kubectl logs deployment/ebs-csi-controller -n kube-system -c csi-provisioner

for the storage controller to be functional, an example log should have the output as shown below:

1 feature_gate.go:241] Setting GA feature gate Topology=true. It will be removed in a future release.
1 feature_gate.go:249] feature gates: &{map[Topology:true]}
1 csi-provisioner.go:154] Version: v3.5.0
1 csi-provisioner.go:177] Building kube configs for running in cluster...
1 common.go:111] Probing CSI driver for readiness
1 csi-provisioner.go:230] Detected CSI driver ebs.csi.aws.com
1 csi-provisioner.go:240] Supports migration from in-tree plugin: kubernetes.io/aws-ebs
1 common.go:111] Probing CSI driver for readiness
1 csi-provisioner.go:299] CSI driver supports PUBLISH_UNPUBLISH_VOLUME, watching VolumeAttachments
1 controller.go:732] Using saving PVs to API server in background
1 leaderelection.go:245] attempting to acquire leader lease kube-system/ebs-csi-aws-com...
1 leaderelection.go:255] successfully acquired lease kube-system/ebs-csi-aws-com

1 leader_election.go:178] became leader, starting
1 controller.go:811] Starting provisioner controller
ebs.csi.aws.com_ebs-csi-controller-689b48f644-7xlbt_da3ab4a2-b995-4f57-9df2-7c90c01c0409!
1 volume_store.go:97] Starting save volume queue
1 controller.go:860] Started provisioner controller
ebs.csi.aws.com_ebs-csi-controller-689b48f644-7xlbt_da3ab4a2-b995-4f57-9df2-7c90c01c0409!

If you do not see the highlighted lines in the example output above, then you need to restart the storage controller with the command below

kubectl delete pods -n kube-system -l=app=ebs-csi-controller.

Again, check the logs and ensure the *highlighted lines* above are now showing in your own deployment (Please be patient!)

If all checks out, then your deployment should be successful for this stage Six, and you should be able to exec into the Nginx container to see all your mounted volumes.

4.2 Accessing Container After Successful Deployment

Execute the example command below to shell into the container:

kubectl -n eks-sample-app exec -it eks-sample-linux-deployment-979d6659-dhlrl -- /bin/bash

The output below display the running instance of the Nginx container - "eks-sample-linux-deployment-979d6659-dhlrl" it is a part of the "eks-sample-app" namespace resource.

[dev01@houssitdvm02 proj12]$ ***kubectl get all -n eks-sample-app***

NAME	READY	STATUS	RESTARTS	AGE
pod/aws-privateca-issuer-798ff7fb46-hdt2l	1/1	Running	0	7h45m
pod/eks-sample-linux-deployment-979d6659-dhlrl	1/1	Running	0	5h47m
pod/eks-sample-linux-deployment-979d6659-vqllw	1/1	Running	0	5h47m
pod/secrets-store-csi-driver-provider-aws-db99j	1/1	Running	0	7h45m
pod/secrets-store-csi-driver-provider-aws-tfzrk	1/1	Running	0	7h45m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/aws-privateca-issuer	ClusterIP	172.20.241.244	<none>	8080/TCP	7h45m
service/eks-sample-linux-service	NodePort	172.20.227.11	<none>	80:30705/TCP	6h34m

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
daemonset.apps/secrets-store-csi-driver-provider-aws	2	2	2	2	2	kubernetes.io/os=linux	7h45m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/aws-privateca-issuer	1/1	1	1	7h45m
deployment.apps/eks-sample-linux-deployment	2/2	2	2	5h47m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/aws-privateca-issuer-798ff7fb46	1	1	1	7h45m
replicaset.apps/eks-sample-linux-deployment-979d6659	2	2	2	5h47m

4.3 Stage 7 – Deployment of Ingress Resource

At Stage 7, you are about to deploy the ingress resource, this requires some manual intervention. In the main.tf file please comment out this attribute of the ingress module as show below:

#ingress_spec_tls = each.value.tls

Also, in the kubernetes.tf file, please comment out the lines show below:

```
#"alb.ingress.kubernetes.io/listen-ports" = "[{\"HTTPS\" : 443}, {\"HTTPS\" : 80}]"
#host = "k8s-ekssampl-ekssampl-6a90e9bd72-223802714.us-east-1.elb.amazonaws.com"
#tls       = [
#    {
#        hosts              =    [
#            "k8s-ekssampl-ekssampl-6a90e9bd72-223802714.us-east-1.elb.amazonaws.com"
#        ]
#        secret_name = "sitadconsulting-com-tls-cert"
#    },
#]
```

Why do we need to do this? Simply because, without a registered domain name you will need the ingress resource to generate a dns hostname that can be used as a placeholder for the commented-out sections before you we can secure it

Once the ingress resource is generated you can replace the DNS name in the commented sections above, but do not remove the comments yet until you execute stage 8.

4.4 Stage 8 – Generation of Certificate and Accessing Containerized NGINX Web Server Instance

Stage 8 is when you generate the certificate (issuing, storing and management of the certificate is handled by AWSPCA and Cert-Manager respectively) required for enabling TLS for the ingress resource. there is a section of the certificate configuration that requires the DNS name to be specified, in our case, since we do not have a registered domainname we use the DNS name generated by the ingress resource earlier. Once the certificate is generated you can the uncomment the sections for Stage 7 and execute terraform apply in order to secure the ingress resource.

The certificate generated is stored as a Kubernetes secret object under the secret name attribute specified in both the certificate and ingress resource configuration. For our example the secret name is "sitadconsulting-com-tls-cert".

The next step is to extract the certificate and its key from the Kubernetes secret object (sitadconsulting-com-tls-cert) and imported it into ACM. This we have done manually, but we believe it could be automated. The commands below with extract the certificate and the private key to be imported in ACM

kubectl get secret sitadconsulting-com-tls-cert -n eks-sample-app -o 'go-template={{index .data "tls.crt"}}' | base64 --decode

kubectl get secret sitadconsulting-com-tls-cert -n eks-sample-app -o 'go-template={{index .data "tls.key"}}' | base64 --decode

Once the output of the commands above has been imported in ACM, the image below shows the state of the imported certificate:

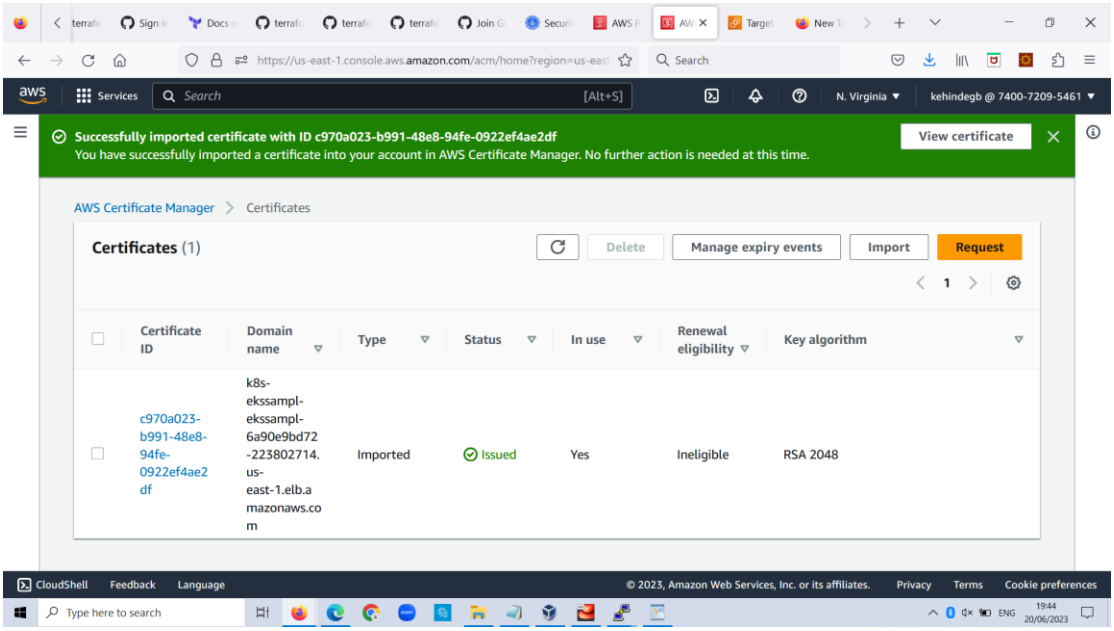

Figure 4.4.1 State of imported Certificate

Once the certificate import has propagated (allow for about 5 - 10 mins) to the loadbalancer, you should see the details shown in the next two images:

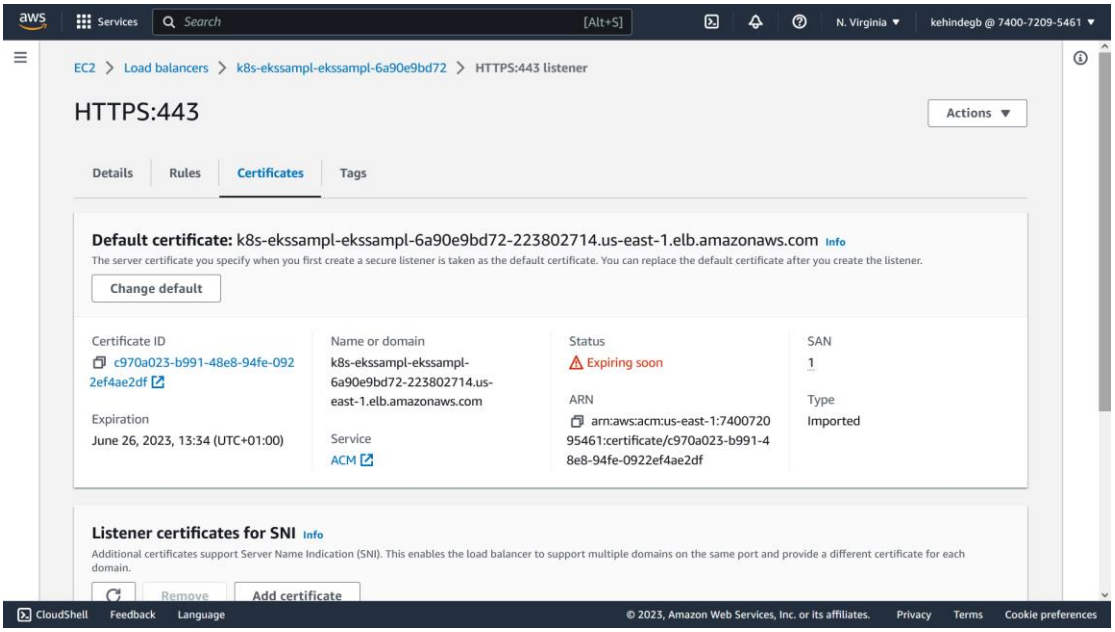

Figure 4.4.2 Certificate details viewed from the Loadbalancer console

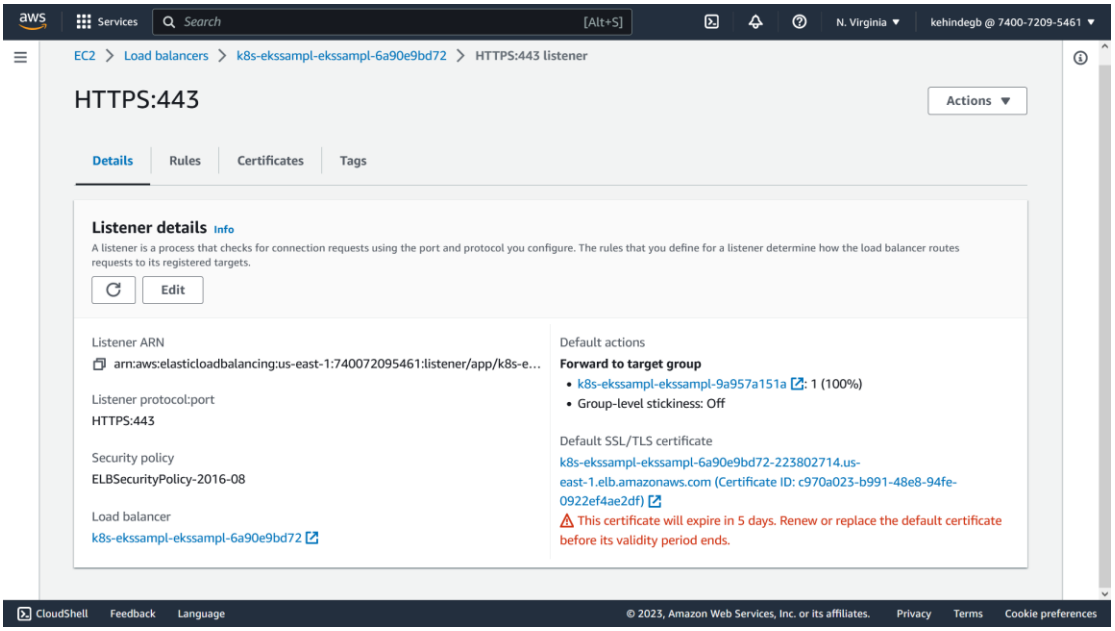

Figure 4.4.3 Listener details viewed from Loadbalancer console

The final step is to enable access to the secure containerized Nginx instance from a web browser. This we achieved by exporting the CA root certificate of the AWS Private CA instance and importing it into the certificate truststore for the web browser we tested with(Firefox). Once this is done, we pointed the browser to the DNS name of the ingress resource in our case. The resulting outcome is depicted in figure 4.4.4 below:

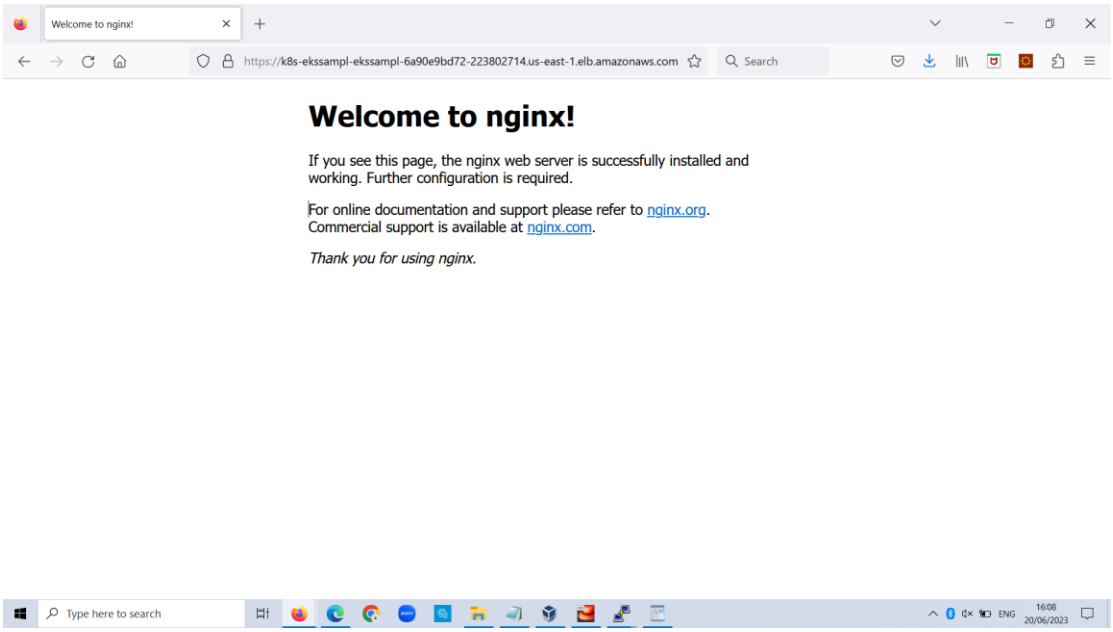

Figure 4.4.4 Secure access to containerized Nginx instance.

5.0 Appendix

List of Deployed Kubernetes Objects

And finally, we provide below the list of deployed Kubernetes object across all namespaces.
List of Kubernetes resources deployed across all namespaces:

[dev01@houssitdvm02 ~]$ ***kubectl get all -A***

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
cert-manager	pod/cert-manager-869f7b446-jn7km	1/1	Running	0	4h58m
cert-manager	pod/cert-manager-cainjector-869d958c8-pzrvq	1/1	Running	0	4h58m
cert-manager	pod/cert-manager-webhook-67cf5854d-fdmj5	1/1	Running	0	4h58m
eks-sample-app	pod/aws-privateca-issuer-798ff7fb46-hdt2l	1/1	Running	0	4h58m
eks-sample-app	pod/eks-sample-linux-deployment-979d6659-dhlrl	1/1	Running	0	3h1m
eks-sample-app	pod/eks-sample-linux-deployment-979d6659-vqllw	1/1	Running	0	3h1m
eks-sample-app	pod/secrets-store-csi-driver-provider-aws-db99j	1/1	Running	0	4h58m
eks-sample-app	pod/secrets-store-csi-driver-provider-aws-tfzrk	1/1	Running	0	4h58m
kube-system	pod/aws-load-balancer-controller-7f4c796ddf-b9d2g	1/1	Running	0	104m
kube-system	pod/aws-load-balancer-controller-7f4c796ddf-v98qq	1/1	Running	0	104m
kube-system	pod/aws-node-mgv7b	1/1	Running	0	5h4m
kube-system	pod/aws-node-rp9mv	1/1	Running	0	5h4m
kube-system	pod/coredns-79df7fff65-n26kz	1/1	Running	0	5h21m
kube-system	pod/coredns-79df7fff65-ttbfd	1/1	Running	0	5h21m

kube-system	pod/ebs-csi-controller-689b48f644-7xlbt	5/5	Running	0	3h42m
kube-system	pod/ebs-csi-controller-689b48f644-9plkt	5/5	Running	0	3h42m
kube-system	pod/ebs-csi-node-jw6bm	3/3	Running	0	4h58m
kube-system	pod/ebs-csi-node-n7hdd	3/3	Running	0	4h58m
kube-system	pod/kube-proxy-cmq7v	1/1	Running	0	5h4m
kube-system	pod/kube-proxy-xmttp	1/1	Running	0	5h4m
kube-system	pod/secrets-store-csi-driver-mh8vs	3/3	Running	0	4h58m
kube-system	pod/secrets-store-csi-driver-w5l4n	3/3	Running	0	4h58m

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
cert-manager	service/cert-manager	ClusterIP	172.20.147.212	<none>	9402/TCP	4h58m
cert-manager	service/cert-manager-webhook	ClusterIP	172.20.116.3	<none>	443/TCP	4h58m
default	service/kubernetes	ClusterIP	172.20.0.1	<none>	443/TCP	5h21m
eks-sample-app	service/aws-privateca-issuer	ClusterIP	172.20.241.244	<none>	8080/TCP	4h58m
eks-sample-app	service/eks-sample-linux-service	NodePort	172.20.227.11	<none>	80:30705/TCP	3h48m
kube-system	service/aws-load-balancer-webhook-service	ClusterIP	172.20.215.251	<none>	443/TCP	4h12m
kube-system	service/kube-dns	ClusterIP	172.20.0.10	<none>	53/UDP,53/TCP	5h21m

NAMESPACE	NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
eks-sample-app	daemonset.apps/secrets-store-csi-driver-provider-aws	2	2	2	2	2	kubernetes.io/os=linux	4h58m
kube-system	daemonset.apps/aws-node	2	2	2	2	2	<none>	5h21m
kube-system	daemonset.apps/ebs-csi-node	2	2	2	2	2	kubernetes.io/os=linux	4h58m
kube-system	daemonset.apps/kube-proxy	2	2	2	2	2	<none>	5h21m
kube-system	daemonset.apps/secrets-store-csi-driver	2	2	2	2	2	kubernetes.io/os=linux	4h58m

NAMESPACE	NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cert-manager	deployment.apps/cert-manager	1/1	1	1	4h58m
cert-manager	deployment.apps/cert-manager-cainjector	1/1	1	1	4h58m
cert-manager	deployment.apps/cert-manager-webhook	1/1	1	1	4h58m
eks-sample-app	deployment.apps/aws-privateca-issuer	1/1	1	1	4h58m
eks-sample-app	deployment.apps/eks-sample-linux-deployment	2/2	2	2	3h1m
kube-system	deployment.apps/aws-load-balancer-controller	2/2	2	2	4h12m
kube-system	deployment.apps/coredns	2/2	2	2	5h21m
kube-system	deployment.apps/ebs-csi-controller	2/2	2	2	4h58m

NAMESPACE	NAME	DESIRED	CURRENT	READY	AGE
cert-manager	replicaset.apps/cert-manager-869f7b446	1	1	1	4h58m
cert-manager	replicaset.apps/cert-manager-cainjector-869d958c8	1	1	1	4h58m
cert-manager	replicaset.apps/cert-manager-webhook-67cf5854d	1	1	1	4h58m
eks-sample-app	replicaset.apps/aws-privateca-issuer-798ff7fb46	1	1	1	4h58m
eks-sample-app	replicaset.apps/eks-sample-linux-deployment-979d6659	2	2	2	3h1m
kube-system	replicaset.apps/aws-load-balancer-controller-7f4c796ddf	2	2	2	4h12m
kube-system	replicaset.apps/coredns-79df7fff65	2	2	2	5h21m
kube-system	replicaset.apps/ebs-csi-controller-689b48f644	2	2	2	4h58m