# **AWS-CLI:**

The AWS Command Line Interface (AWS CLI) is an open-source tool that enables you to interact with AWS services using commands in your command-line shell. With minimal configuration, the AWS CLI enables you to start running commands that implement functionality equivalent to that provided by the browser-based AWS Management Console from the command prompt in your terminal program.

The installation of AWS-CLI can be done by executing the following commands:

1. Use the curl command – The -o option specifies the file name that the downloaded package is written to. The options on the following example command write the downloaded file to the current directory with the local name awscliv2.zip.

$ **curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"**

1. Unzip the installer. If your Linux distribution doesn't have a built-in unzip command, use an equivalent to unzip it. The following example command unzips the package and creates a directory named aws under the current directory.

$ **unzip awscliv2.zip**

1. Run the install program. The installation command uses a file named install in the newly unzipped aws directory. By default, the files are all installed to /usr/local/aws-cli, and a symbolic link is created in /usr/local/bin. The command includes sudo to grant write permissions to those directories.

$ **sudo ./aws/install**

1. Move the downloaded file to the path, The default value is /usr/local/bin.

$ **./aws/install -i */usr/local/aws-cli* -b */usr/local/bin***

1. Now, Check the version of the aws-cli by running the following command. And the output is given below.

$ **aws –version**

aws-cli/2.4.5 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86\_64 botocore/2.4.5

# **Terraform:**

* The terraform can be installed in your local machine by using the following Command:

$ Wget <https://releases.hashicorp.com/terraform/1.2.7/terraform_1.2.7_linux_amd64.zip>

* Move the downloaded terraform binary file to the /usr/local/bin

$ sudo mv terraform /usr/local/bin

## Check the terraform Version by using the command and output shown below:

## $ terraform -version

## Terraform v1.2.6

## on linux\_amd64

## IAM User for Terraform:

We need to create an IAM User for Terraform with the following permissions:

* IAM User with Administrative Access
* Amazon EC2 Full Access
* Amazon S3 Full Access
* Amazon EKS Full Access
* Amazon DynamoDB Full Access
* Amazon VPC Full Access

The above-mentioned permissions need to be attached to the IAM User created for Provisioning AWS Resources through Terraform and AWS-CLI.

By using the following commands we can create IAM Users, Policy and Roles.

**To create an IAM user (AWS CLI)**:

1. To Create User

$ aws iam create-user

1. To create Role

$ aws iam create-role --role-name EXAMPLE\_ROLE

1. To attach Policy to Role

$ aws iam attach-role-policy --policy-arn arn:aws:iam::ACCOUNT\_ID:policy/EXAMPLE\_POLICY --role-name AmazonEKS\_EBS\_CSI\_DriverRole

# **Backup and Restore EKS Kubernetes Using Velero:**

**Velero Installation and setup on Amazon EKS:** Velero is an open-source tool that helps automate the backup and restore of Kubernetes clusters, including any application and its data.

**Velero enables the following use cases:**

* Disaster recovery — backup of the cluster and restore in case of a disaster.
* Application migration — migrate an application along with its data from one cluster to another.
* Application cloning — replicating production environments for testing and debugging.

**In this module, you will learn the below things:**

1.how to backup and restore an EKS cluster using Velero.

2. how to backup and restore data on 2 different EKS Cluster.

3. how to backup and restore the data at a particular time(Scheduled backup and restore)

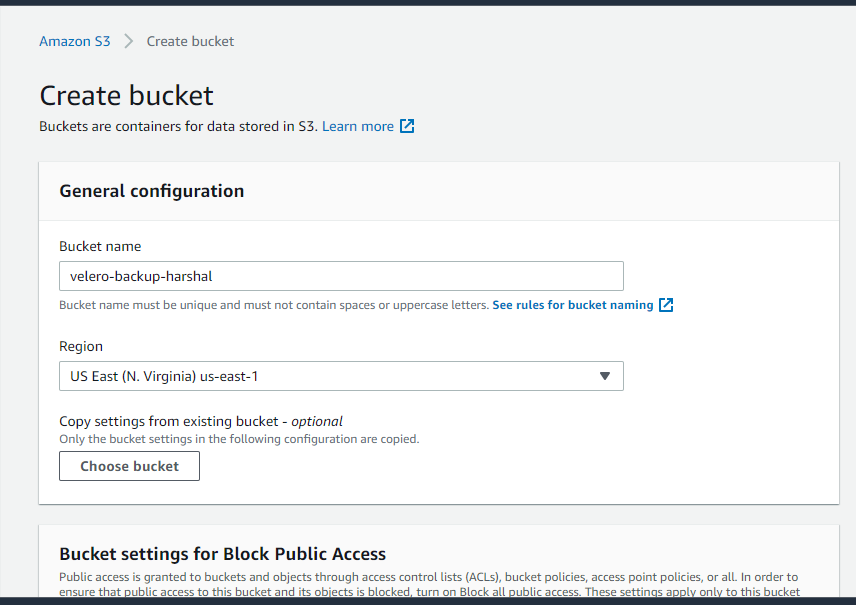
**Let’s start:**

**Prerequisites**

1. AWS CLI needs to be configured in the machine where you execute Velero commands.
2. Kubectl needs to be configured with the EKS cluster where you need to take the backup.

# **CREATE S3 BUCKET AND IAM USER FOR VELERO**

1. Create the S3 bucket using AWS console. go to AWS -> S3-> create a bucket,



1. Create an IAM user. go to AWS -> IAM Console -> add user

Graphical user interface, text, application, email

Description automatically generated

1. Add the below permission to the user. replace ${BUCKET} with the S3 bucket name which we created for velero.

{  
"Version":"2012-10-17",  
"Statement":[  
{  
"Effect":"Allow",  
"Action":[  
"ec2:DescribeVolumes",  
"ec2:DescribeSnapshots",  
"ec2:CreateTags",  
"ec2:CreateVolume",  
"ec2:CreateSnapshot",  
"ec2:DeleteSnapshot"  
],  
"Resource":"\*"  
},  
{  
"Effect":"Allow",  
"Action":[  
"s3:GetObject",  
"s3:DeleteObject",  
"s3:PutObject",  
"s3:AbortMultipartUpload",  
"s3:ListMultipartUploadParts"  
],  
"Resource":[  
"arn:aws:s3:::${VELERO\_BUCKET}/\*"  
]  
},  
{  
"Effect":"Allow",  
"Action":[  
"s3:ListBucket"  
],  
"Resource":[  
"arn:aws:s3:::${VELERO\_BUCKET}"  
 ]  
}  
]  
}

1. Add user credentials to the server.

using below command:

$ aws configure

Text

Description automatically generated

# **INSTALL VELERO Client:**

Install Velero binary

wget <https://github.com/vmware-tanzu/velero/releases/download/v1.3.2/velero-v1.3.2-linux-amd64.tar.gz>

Extract the tarball:

tar -xvf velero-v1.3.2-linux-amd64.tar.gz -C /tmp

Move the extracted velero binary to /usr/local/bin:

sudo mv /tmp/velero-v1.3.2-linux-amd64/velero /usr/local/bin

Verify installation:

velero version

output:

Client: Version: v1.3.2 Git commit: 55a9914a3e4719fb1578529c45430a8c11c28145 <error getting server version: the server could not find the requested resource (post serverstatusrequests.velero.io)>

if you got the velero not found message then set a below path variable for velero.

e.x: export PATH=$PATH:/usr/local/bin

**Install Velero on EKS:**

velero install \  
--provider aws \  
--plugins velero/velero-plugin-for-aws:v1.0.1 \  
--bucket <bucketname>\  
--backup-location-config region=<region> \  
--snapshot-location-config region=<region> \  
--secret-file /root/.aws/credentials

replace your bucket name, region, and credentials path in the command.

E.X:

velero install \  
--provider aws \  
--plugins velero/velero-plugin-for-aws:v1.0.1 \  
--bucket velero—eks-backup \  
--backup-location-config region=us-east-1 \  
--snapshot-location-config region=us-east-1 \  
--secret-file /root/.aws/credentials

Inspect the resources created:

kubectl get all -n velero

Graphical user interface, text

Description automatically generated

**DEPLOY TEST APPLICATION:**

Create namespace and deploy the application

kubectl create namespace <namespacename>

kubectl create namespace harshal

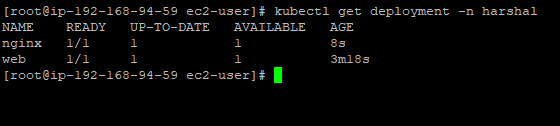
Deploy 2 sample applications in the harshal namespace.

kubectl create deployment web --image=gcr.io/google-samples/hello-app:1.0 -n harsshal

kubectl create deployment Nginx --image=nginx -n test

Verify deployment:

kubectl get deployments -n harshal



# **BACKUP AND RESTORE:**

Let’s back up the harshal namespace using velero

Backups:

velero backup create <backupname> --include-namespaces <namespacename>

velero backup create test1 --include-namespaces harshal

Check the Status of backup created:

velero backup describe <backupname>

Check-in S3 bucket:

backup is stored in the S3 bucket.

Graphical user interface, application, email

Description automatically generated

Let’s delete the ‘harshal’ namespace to simulate a disaster:

kubectl delete namespace harshal

Text

Description automatically generated

Restore harshal namespace:

Restore:

Run the velero restore command from the backup created. It may take a couple of minutes to restore the namespace.

velero restore create --from-backup <backupname>

velero restore create --from-backup test1

Verify if deployments, replica sets, services, and pods are restored.

In the below image you see, harshal namespace is newly created and restored using velero, and all old deployment, pods, and svc are also restored back.

