AWS Storage Blog

Deploying Jenkins on Amazon EKS with Amazon EFS

by Luke Wells | on 24 JUL 2020 | in Advanced (300), Amazon Elastic File System (EFS), Amazon Elastic Kubernetes Service, AWS CLI, Storage, Technical How-to | Permalink | Comments | Share

UPDATE (5/17/2021):

It looks like the Jenkins Helm repos have moved and the configurable parameters have been updated as well! You will need to modify the commands listed in the blog accordingly. Check out the new Jenkins helm repos and configurable parameters at the following links:

- https://github.com/jenkinsci/helm-charts/blob/main/charts/jenkins/README.md
- https://github.com/jenkinsci/helm-charts/blob/main/charts/jenkins/VALUES_SUMMARY.md

Also, make sure your EKS cluster control plane is up to date using the following command:

```
eksctl upgrade cluster --name=myekscluster --approve
eksctl upgrade nodegroup --cluster=myekscluster --name=mynodegroup
```

Many applications that companies try to containerize require persistent or shared storage, such as DevOps tools, content management systems (CMS), and data science notebooks. With the introduction of the Amazon EFS Container Storage Interface (CSI) driver, now Generally Available, developers can use Amazon EFS to provide elastic, durable, and persistent shared storage to containerized applications running in Amazon EKS. In this blog post, I show you how to easily deploy Jenkins on Amazon EKS with Amazon EFS.

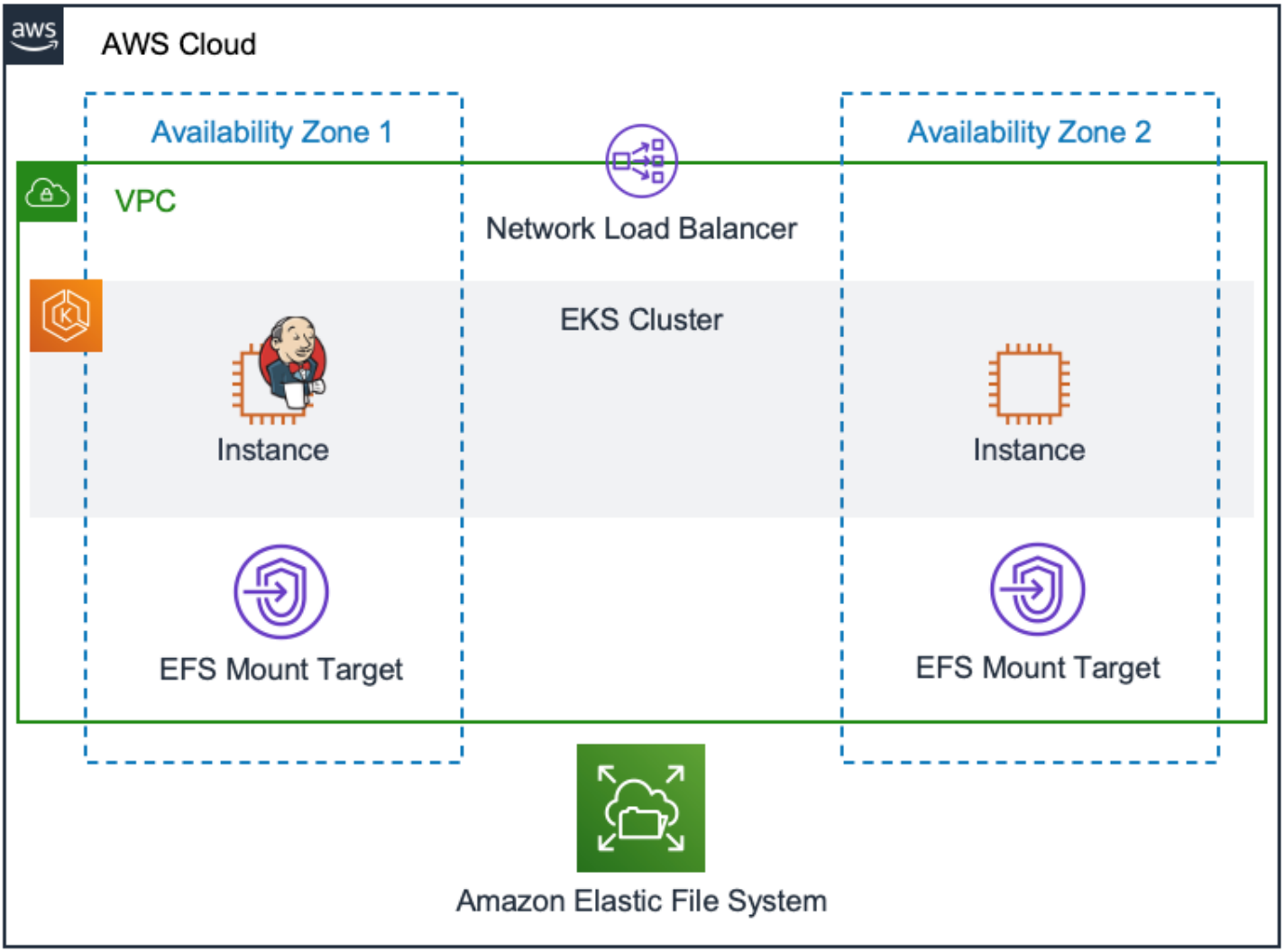
Overview of solution

Amazon EKS is a fully managed service that allows you to run upstream Kubernetes on AWS with a highly available control plane across multiple AWS Availability Zones. Amazon EFS provides a simple, scalable, fully managed elastic shared file system. With Amazon EFS, you can deploy shared file systems without the need to provision and manage capacity to accommodate growth. Kubernetes applications can use EFS file systems to share data between pods within and across AWS Availability Zones. EFS can also help Kubernetes applications be highly available because all data written to EFS is written to multiple AWS Availability Zones.

The Amazon EFS CSI driver makes it easy to configure elastic file storage for both Amazon EKS and self-managed Kubernetes clusters running on AWS using standard Kubernetes interfaces. If a Kubernetes pod is terminated and relaunched, the CSI driver reconnects the EFS file system, even if the pod is relaunched in a different AWS Availability Zone.

In this solution, I explore the benefits of using Amazon EFS as the shared persistent file storage layer for a containerized Jenkins application deployed on Amazon EKS. By default, Jenkins stores application configuration,

job configurations, change logs, and past build records in the `$JENKINS_HOME` directory. Configuring `$JENKINS_HOME` to point to a shared Amazon EFS file system enables us to quickly create a new Jenkins pod in another AWS Availability Zone in the event of node failure or pod termination. Additionally, having a shared file system enables us to further architect our Jenkins deployment for scale.



Tutorial

In this post, I guide you through creating a Kubernetes cluster on Amazon EKS, and an Amazon EFS file system with mount targets in two AWS Availability Zones. I then walk you through deploying the Amazon EFS CSI driver to provide a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon EFS file systems.

Once the infrastructure is set up, I walk you through deploying Jenkins to the Amazon EKS cluster, creating some user-specific configuration, and simulating a node failure in our EKS cluster. Lastly, I show you how check to make sure that your configuration state is persisted to Amazon EFS after Kubernetes launches a new Jenkins pod.

1. Create an Amazon EKS cluster
2. Create an Amazon EFS file system

3. Deploy the Amazon EFS CSI Driver to your Amazon EKS cluster
4. Deploy Jenkins to Amazon EKS
5. Simulate a Kubernetes node failure
6. Check for persisted state thanks to Amazon EFS!

Prerequisites

- An AWS account
- Installed the AWS CLI version 1.16.156 or later
- Installed the aws-iam-authenticator
- Installed the Kubernetes command line utility kubectl
- Installed eksctl (a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS)
- Installed Helm package manager for Kubernetes
- Basic understanding of Kubernetes and Amazon EFS

Create an Amazon EKS cluster

In this section, you use eksctl to create an Amazon EKS cluster.

1. Create an Amazon EKS cluster

Once you have the eksctl command line utility installed, create a cluster using the following command:

```
eksctl create cluster --name myekscluster --region us-east-1 --zones
us-east-1a,us-east-1b --managed --nodegroup-name mynodegroup
```

This command creates an Amazon EKS cluster in the us-east-1 Region with one EKS-managed node group containing two m5.large nodes in us-east-1a and us-east-1b. Cluster provisioning usually takes between 10 and 15 minutes.

2. Test that your kubectl configuration is correct

When the cluster is ready, test that the kubectl configuration is correct using the following command:

```
kubectl get svc
```

If you receive any authorization or resource type errors, see the unauthorized or access denied (kubectl) section in the troubleshooting documentation.

Output:

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   6m20s
```

Create an Amazon EFS file system

In this section, you create a security group, Amazon EFS file system, two mount targets, and an EFS Access Point. The access point allows the Amazon EKS cluster to use the EFS file system.

1. Capture your VPC ID

First, you must find the VPC ID that was created in the previous step using the following command:

```
aws ec2 describe-vpcs
```

2. Create a security group for your Amazon EFS mount target

Next, create a security group for the Amazon EFS mount target, which requires your VPC ID.

```
aws ec2 create-security-group \
--region us-east-1 \
--group-name efs-mount-sg \
--description "Amazon EFS for EKS, SG for mount target" \
--vpc-id identifier for our VPC (i.e. vpc-1234567ab12a345cd)
```

3. Add rules to the security group to authorize inbound/outbound access

Authorize inbound access to the security group for the Amazon EFS mount target (efs-mount-sg) to allow inbound traffic to the NFS port (2049) from the VPC CIDR block using the following command:

```
aws ec2 authorize-security-group-ingress \
--group-id identifier for the security group created for our Amazon
EFS mount targets (i.e. sg-1234567ab12a345cd) \
--region us-east-1 \
--protocol tcp \
--port 2049 \
--cidr 192.168.0.0/16
```

4. Create an Amazon EFS file system

Next, create an encrypted Amazon EFS file system using the following command and record the file system ID:

```
aws efs create-file-system \
--creation-token creation-token \
--performance-mode generalPurpose \
--throughput-mode bursting \
--region us-east-1 \
--tags Key=Name,Value=MyEFSFileSystem \
--encrypted
```

5. Capture your VPC subnet IDs

To create mount targets, you must have subnet IDs for my node group. Use the following command to find and record the subnets IDs:

```
aws ec2 describe-instances --filters Name=vpc-id,Values= identifier
for our VPC (i.e. vpc-1234567ab12a345cd) --query
'Reservations[*].Instances[].SubnetId'
```

6. Create two Amazon EFS mount targets

Now that you have the security group, file system ID, and subnets, you can create mount targets in each of the Availability Zones using the following command:

```
aws efs create-mount-target \
--file-system-id identifier for our file system (i.e. fs-123b45fa) \
--subnet-id identifier for our node group subnets (i.e. subnet-
1234567ab12a345cd) \
--security-group identifier for the security group created for our
Amazon EFS mount targets (i.e. sg-1234567ab12a345cd) \
--region us-east-1
```

Be sure to create a mount target in each of the two Availability Zones!

7. Create an Amazon EFS access point

Now that you have your file system, let's create an Amazon EFS Access Point. Amazon EFS access points are application-specific entry points into an EFS file system that make it easier to manage application access to shared datasets or, in our case, configuration. Regardless of how a container is built, access points can enforce a user identity, including the user's POSIX groups, for all file system requests that are made through them. For our purposes, let's create a Jenkins-specific EFS access point and choose to enforce user ID and a group ID of 1000

using the following command:

```
aws efs create-access-point --file-system-id identifier for our file
system (i.e. fs-123b45fa) \
--posix-user Uid=1000,Gid=1000 \
--root-directory
"Path=/jenkins,CreationInfo={OwnerUid=1000,OwnerGid=1000,Permissions=777}"
```

Record the access point ID (that is, fsap-0123456abc987634a) for future reference.

Deploy the Amazon EFS CSI driver to your Amazon EKS cluster

In this step, deploy the Amazon EFS CSI driver to the Amazon EKS cluster and create a persistent volume claim (PVC).

1. Deploy the Amazon EFS CSI driver to your Amazon EKS cluster

To deploy the Amazon EFS CSI driver, run the following command:

```
kubectl apply -k "github.com/kubernetes-sigs/aws-efs-csi-
driver/deploy/kubernetes/overlays/stable/?ref=master"
```

Output:

```
daemonset.apps/efs-csi-node created
csidriver.storage.k8s.io/efs.csi.aws.com created
```

2. Create efs-sc storage class YAML file

With the Amazon EFS CSI driver installed, you can create a storage class that enables you to provision persistent volumes to allow pods to use the Amazon EFS file system. To do this, copy the following configuration and save it to a file called storageclass.yaml.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
```

3. Create the efs-pv persistent volume YAML file

Let's go ahead and create a persistent volume and a persistent volume claim for our Jenkins app. First, copy the following configuration and save it to a file called persistentvolume.yaml. Make sure you modify the volumeHandle parameter to be your file system ID and access point ID:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: identifier for our file system::identifier for our
access point (i.e. fs-123b45fa::fsap-12345678910ab12cd34)
```

4. Create the efs-claim persistent volume claim YAML file

Next, copy the following configuration and save it to a file called persistentvolumeclaim.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
```

Note: Because Amazon EFS is an elastic file system, it does not enforce any file system capacity limits. The actual storage capacity value in persistent volumes and persistent volume claims is not used when creating the file system. However, since storage capacity is a required field in Kubernetes, you must specify a valid value, such as, 5Gi in this example. This value does not limit the size of your Amazon EFS file system.

5. Deploy the efs-sc storage class, efs-pv persistent volume, and efs-claim persistent volume claim

It's worth noting that in an enterprise Kubernetes deployment, these resources may be provisioned by different users, or roles, within your organization. A Kubernetes administrator would likely provision storage classes and persistent volumes to make storage available to users. Users, or developers, would then consume that storage by provisioning a persistent volume claim. That said, go ahead and deploy these resources with the following command:

```
kubectl apply -f
storageclass.yaml,persistentvolume.yaml,persistentvolumeclaim.yaml
```

6. Check to make sure the Kubernetes resources were created

Check to make sure that the storage class, persistent volume, and persistent volume claims were created using the following command:

```
kubectl get sc,pv,pvc
```

Output:

```
NAME                                          PROVISIONER
AGE
storageclass.storage.k8s.io/efs-sc            efs.csi.aws.com
45s
storageclass.storage.k8s.io/gp2 (default)     kubernetes.io/aws-ebs
45m

NAME                        CAPACITY   ACCESS
MODES   RECLAIM POLICY
STATUS   CLAIM               STORAGECLASS   REASON   AGE
persistentvolume/efs-pv   5Gi        RWX            Retain
Bound    default/efs-claim   efs-sc                  45s

NAME                                STATUS   VOLUME   CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
persistentvolumeclaim/efs-claim   Bound    efs-pv   5Gi        RWX
efs-sc         45s
```

Deploy Jenkins to Amazon EKS

In this section, you deploy Jenkins to your Amazon EKS cluster using Helm, a package manager for Kubernetes

that helps you install and manage applications on your Kubernetes cluster.

1. Add the Helm stable chart repository

First, add the official Helm stable chart repository using the following command:

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```

2. Install Jenkins on your EKS cluster

Go ahead and install Jenkins using the following command, being sure to reference the existing PVC:

```
helm install jenkins stable/jenkins --set
rbac.create=true,master.servicePort=80,master.serviceType=LoadBalancer
,persistence.existingClaim=efs-claim
```

3. Capture the ingress loadbalancer name

It usually takes 2–3 minutes for the pod to provision and the node to register as a healthy target behind the load balancer. Now is a great time to refill the beverage of your choice! Once you've refueled, get the ingress load balancer name using the following command:

```
printf $(kubectl get service jenkins -o
jsonpath="{.status.loadBalancer.ingress[].hostname}");echo
```

4. Collect the admin password and connect to your Jenkins cluster!

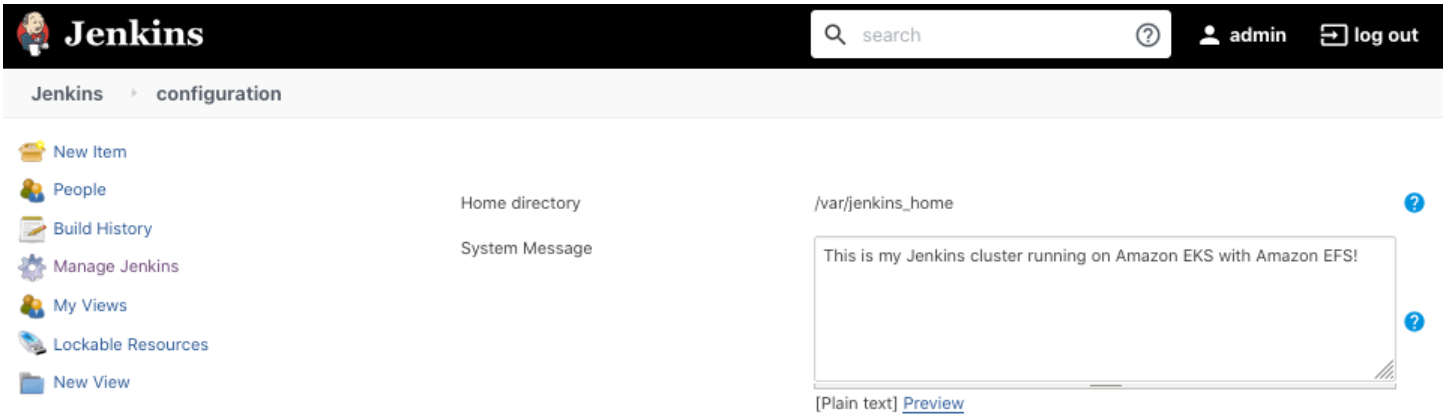
Open up a web browser, and connect to your fresh Jenkins cluster deployed on Amazon EKS and backed by Amazon EFS! To get the password for the admin user, run the following command:

```
printf $(kubectl get secret jenkins -o jsonpath="{.data.jenkins-admin-
password}" | base64 --decode);echo
```

Welcome to Jenkins!

5. Personalize your Jenkins cluster

Once you are logged in to your Jenkins cluster, personalize your configuration so you have some unique configuration persisted to your Amazon EFS file system for future reference. You can do this by clicking **Manage Jenkins**, then **Configure System,** and then adding a **System Message**. If you want to explore more Jenkins automation, check out the DevOps blog on setting up a CI/CD pipeline by integrating Jenkins with AWS CodeBuild and AWS CodeDeploy. Once you're done with your initial configuration, scroll down and choose **Save**:



Simulate a Kubernetes node failure

1. Capture the Jenkins pod name

Before simulating a node failure, check to ensure that your **System Message** is safely stored on an Amazon EFS file system. To do this, get the Jenkins pod name by running the following command:

```
kubectl get pods
```

Output:

```
NAME                       READY   STATUS    RESTARTS   AGE
jenkins-8456fcdcf8-q85jw   2/2     Running   0          1h45m
```

2. Open a shell to your Jenkins container

Get a shell to the running container by utilizing the following command:

```
kubectl exec -it Jenkins pod name (i.e. jenkins-8456fcdcf8-q85jw) --
/bin/sh
```

3. Confirm your Amazon EFS file system is mounted

In the command prompt, check to make sure that your Amazon EFS file system is mounted with the proper mount options at the $JENKINS_HOME directory using the following command:

```
mount | grep $JENKINS_HOME
```

Output:

```
127.0.0.1:/ on /var/jenkins_home type nfs4
(rw,relatime,vers=4.1,rsize=1048576,wsize=1048576,namlen=255,hard,nores
vport,proto=tcp,port=20261,timeo=600,retrans=2,sec=sys,clientaddr=127
.0.0.1,local_lock=none,addr=127.0.0.1)
```

4. Confirm your Jenkins configuration is stored in your Amazon EFS file system

Finally, check to make sure that your **System Message** is properly updated in the config.xml located in the $JENKINS_HOME directory, which you just confirmed is mounted as an Amazon EFS file system, using the following command:

```
grep systemMessage $JENKINS_HOME/config.xml
```

Output:

```
<systemMessage>This is my Jenkins cluster running on Amazon EKS with
Amazon EFS!</systemMessage>
```

5. Collect the node that is running your Jenkins pod

Exit out of the shell to the running container. Next, find the node that is running your Jenkins pod using the following command:

```
kubectl get pod -o=custom-
```

```
columns=NODE:.spec.nodeName,NAME:.metadata.name
```

Output:

```
NODE                              NAME
ip-192-168-3-131.ec2.internal     jenkins-8456fcdcf8-q85jw
```

6. Simulate a node failure

Simulate a node failure by removing the node from service and evicting all of your pods from the node. You need the –ignore-daemonsets flag to ensure that DaemonSet-managed pods are also evicted. You also need the –– delete-local-data flag to be sure that you don't persist any data. Use the following command:

```
kubectl drain node name (i.e. ip-192-168-3-131.ec2.internal) --delete-
local-data --ignore-daemonsets
```

7. Check to make sure a new pod is started

Check to make sure that the Kubernetes deployment has started a new pod, in a different Availability Zone and on the other node in your EKS cluster, using the following command:

```
kubectl get pod -o=custom-
columns=NODE:.spec.nodeName,NAME:.metadata.name
```

Output:

```
NODE                              NAME
ip-192-168-49-235.ec2.internal    jenkins-8456fcdcf8-xq6w6
```

Check for persisted state thanks to Amazon EFS!

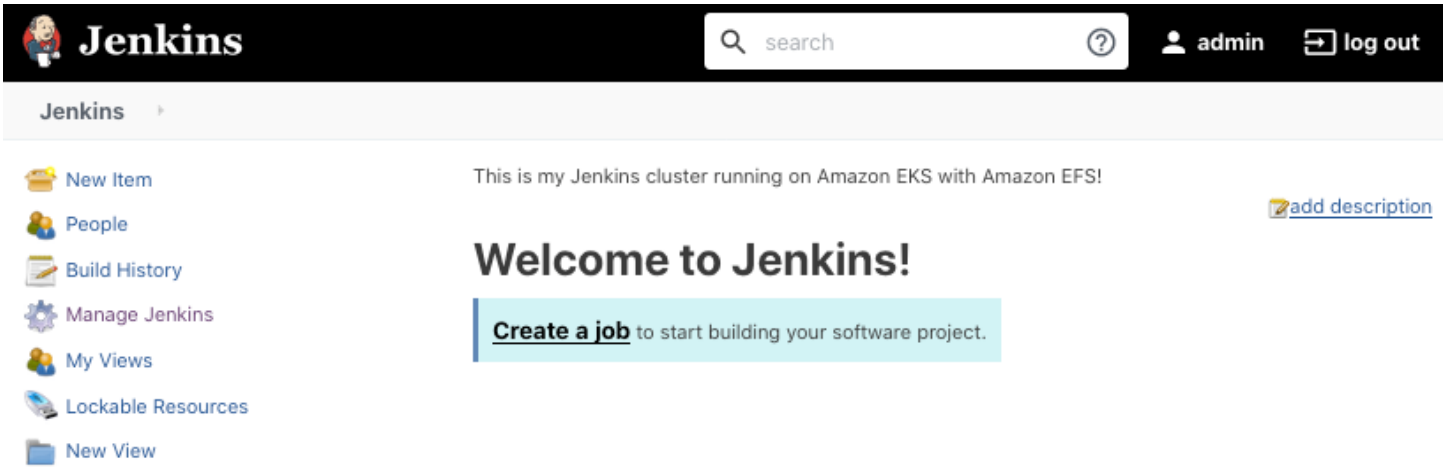
Now that the new Jenkins pod is up and running on a different node and in a different Availability Zone, check to make sure that the configuration modifications you made in the previous step persisted.

1. Log back into your Jenkins cluster

Do so by logging back into the Jenkins cluster. Refer to the `kubectl get service` and `kubectl get secret` commands to fetch the load balancer URL and password for the admin user:

2. Confirm your personalization and Jenkins configuration persists!

Depending on how much personalization you did, you should, at a minimum, see the **System Message** that we configured in the earlier step!



Cleaning up

To avoid incurring future charges, delete the resources you created.

1. Delete your EKS cluster using the following command:

```
eksctl delete cluster --name myekscluster
```

2. Delete both your EFS mount targets using the following command for each of the two mount targets:

```
aws efs delete-mount-target --mount-target-id identifier for our file
system mount targets (i.e. fsmt-123b45fa)
```

3. Delete your EFS file system using the following command:

```
aws efs delete-file-system --file-system-id identifier for our file system
(i.e. fs-123b45fa)
```

Conclusion

In this post I showed you how to easily deploy Jenkins on Amazon EKS with Amazon EFS providing elastic, durable, and persistent shared storage. Amazon EFS and the Amazon EFS CSI driver enable customers to quickly deploy highly available applications on Kubernetes that require elastic, durable, and persistent file storage. Amazon EFS removes the undifferentiated heavy lifting required to deploy and manage shared file systems for Kubernetes applications. Creating containerized applications that can reconnect to a shared file system if a pod is terminated and relaunched in a different AWS Availability Zone is a powerful capability that enables you to build

highly available applications without having to worry about deploying and managing the underlying infrastructure. Such capabilities enable many Kubernetes workloads, including developer tools, like Jenkins, Jira, and GitLab.

Are you running Kubernetes applications that require elastic, durable, and persistent shared storage? If so, I'd encourage you to explore the user guide for deploying Amazon EFS CSI driver and getting started with Amazon EFS! Thanks for reading this blog post, if you have any comments or questions, don't hesitate to leave them in the comments section.

TAGS: Amazon Elastic File System (Amazon EFS), Amazon Elastic Kubernetes Service, AWS Cloud Storage, AWS Command Line Interface (AWS CLI)

Comments