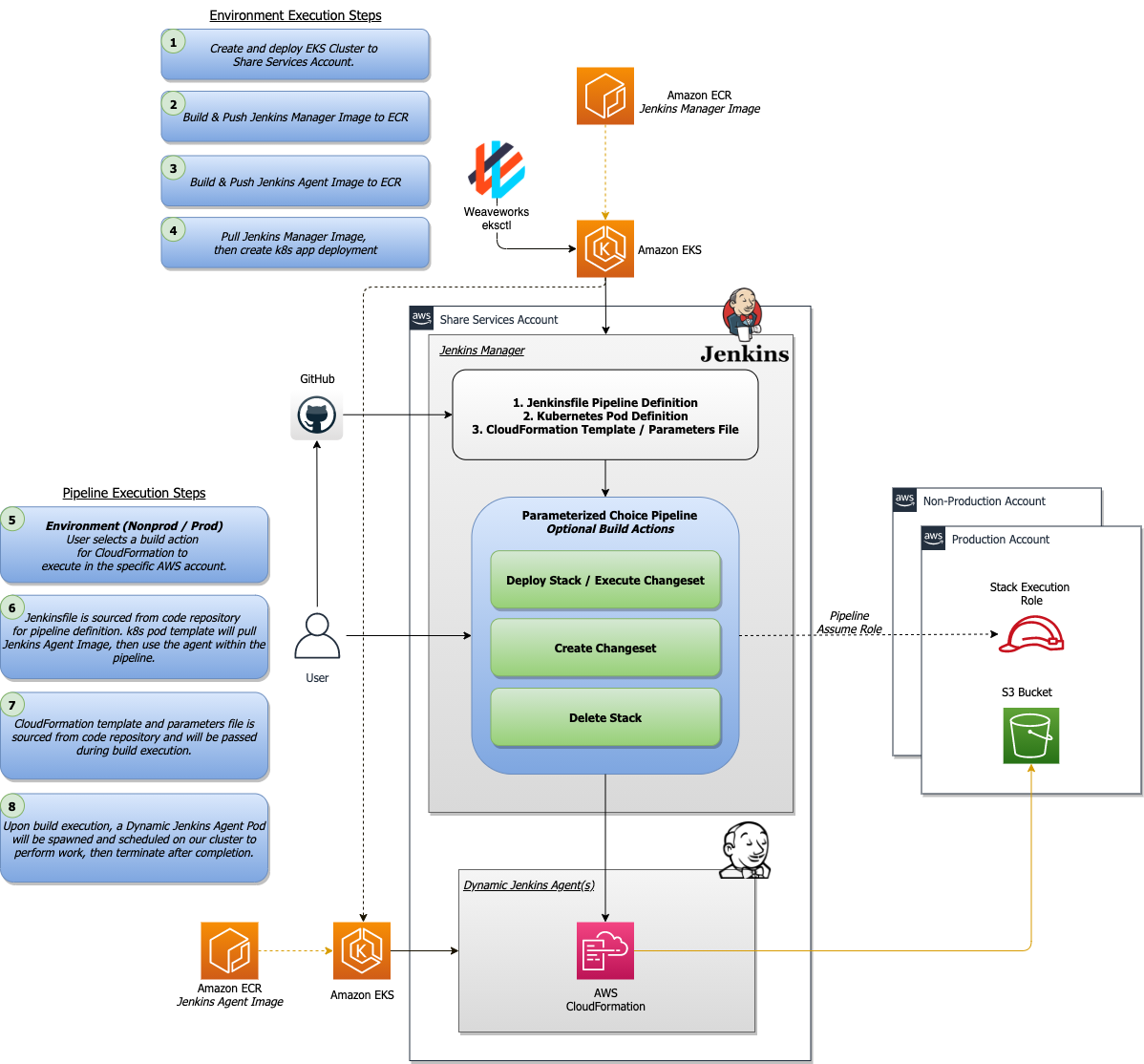
# **Orchestrate Jenkins Workloads using Dynamic Pod Autoscaling with Amazon EKS**

In this blog post, we’ll demonstrate how to leverage [Jenkins](https://www.jenkins.io/) with [Amazon Elastic Kubernetes Service (EKS)](https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html) by running a Jenkins Manager within an EKS pod. By doing so, we can run Jenkins workloads by allowing Amazon EKS to spawn dynamic Jenkins Agent(s) to perform application and infrastructure deployment.

In the effort to setup our Amazon EKS cluster with Jenkins, we’ll use the [eksctl](https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html) simple CLI tool for creating clusters on EKS. Then, we’ll build both the Jenkins Manager and Jenkins Agent image. Afterwards, we’ll run a container deployment on our cluster to access the Jenkins application and use the dynamic Jenkins Agent pods to run pipelines & jobs.

## Solution Overview

The following architecture illustrates the execution steps.

 *Figure 1. Solution Overview Diagram*

Disclaimer(s): *(Note: This Jenkins application is not configured with a persistent volume storage, therefore you will need to establish and configure this template to fit that requirement).*

To accomplish this deployment workflow we’re going to do the following:

* Centralized Shared Services account
  1. Deploy the Amazon EKS Cluster.
  2. Create Amazon ECR for Jenkins Manager and Jenkins Agent.
  3. Deploy the kubernetes manifest file for the Jenkins Manager.
* Target Account(s) - Establish a set of [AWS Identity and Access Management (IAM)](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html) roles with permissions for cross-across access from the Share Services account into the Target account(s).
* Jenkins Plugins - Install and configure the [Kubernetes Plugin](https://plugins.jenkins.io/kubernetes/) and [CloudBees AWS Credentials Plugin](https://plugins.jenkins.io/aws-credentials/) from Manage Plugins *(You will not have to manually install this since it will be packaged and installed as part of the Jenkins image build).*
* Jenkins Pipeline Example - Fetch code from repository by using CloudFormation to deploy an S3 Bucket to the Target account with Jenkins parameterized pipeline.

## Account Prerequisite(s)

The following below is the minimum requirement in order to ensure this solution will work.

* Shared Services Account: This is where the Amazon EKS Cluster will reside.
* Target Account: This is destination of the CI/CD pipeline deployments.

## Build Requirements

* [AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html)
* [Docker CLI](https://docs.docker.com/get-docker/) - *Note: The docker engine must be running to build images.*
* [aws-iam-authenticator](https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html)
* [kubectl](https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html)
* [eksctl](https://github.com/weaveworks/eksctl)

## Build Requirements Setup

Verify if the AWS CLI was installed by executing this command in your terminal aws --version. If you have an existing account, and your machine is configured for programmatic access, then you can proceed to the next step. In this walkthrough, we’ll be using Administrator privileges for our deployment. Additionally, Amazon ECR requires that users must have permission to make calls to the ecr:GetAuthorizationToken. These privileges are for demonstration purposes only and not recommended for production. Before proceeding, please ensure that you’ve properly install all the build requirements necessary.

* Configure your AWS credentials with the associated region. The following steps show how you can do this.

# Export access keys  
~ export AWS\_ACCESS\_KEY\_ID=AKIAXXXXXXXXXXXXXXXX  
~ export AWS\_SECRET\_ACCESS\_KEY=XXXXXXXXXXXXXXXXXXXXXXXX  
~ export AWS\_SESSION\_TOKEN=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
~ export AWS\_DEFAULT\_REGION=<REGION-NAME>  
  
### OR ###  
  
# Export AWS credentials profile  
~ export AWS\_PROFILE=<PROFILE-NAME>

* Verify if the build requirements were installed correctly by executing the following commands below.

# Check docker version  
~ docker --version  
  
# Check kubectl version  
~ kubectl version —client  
  
# Check eksctl version  
~ eksctl version  
  
# Check aws-iam-authenticator version  
~ aws-iam-authenticator version

## Create EKS Cluster

* Clone the repository and execute the following commands below. Make sure your AWS configuration is pointing to the correct region you want EKS deployed. If you do not specify the region, the AWS Profile you setup earlier will be used as the default for where the cluster will reside.

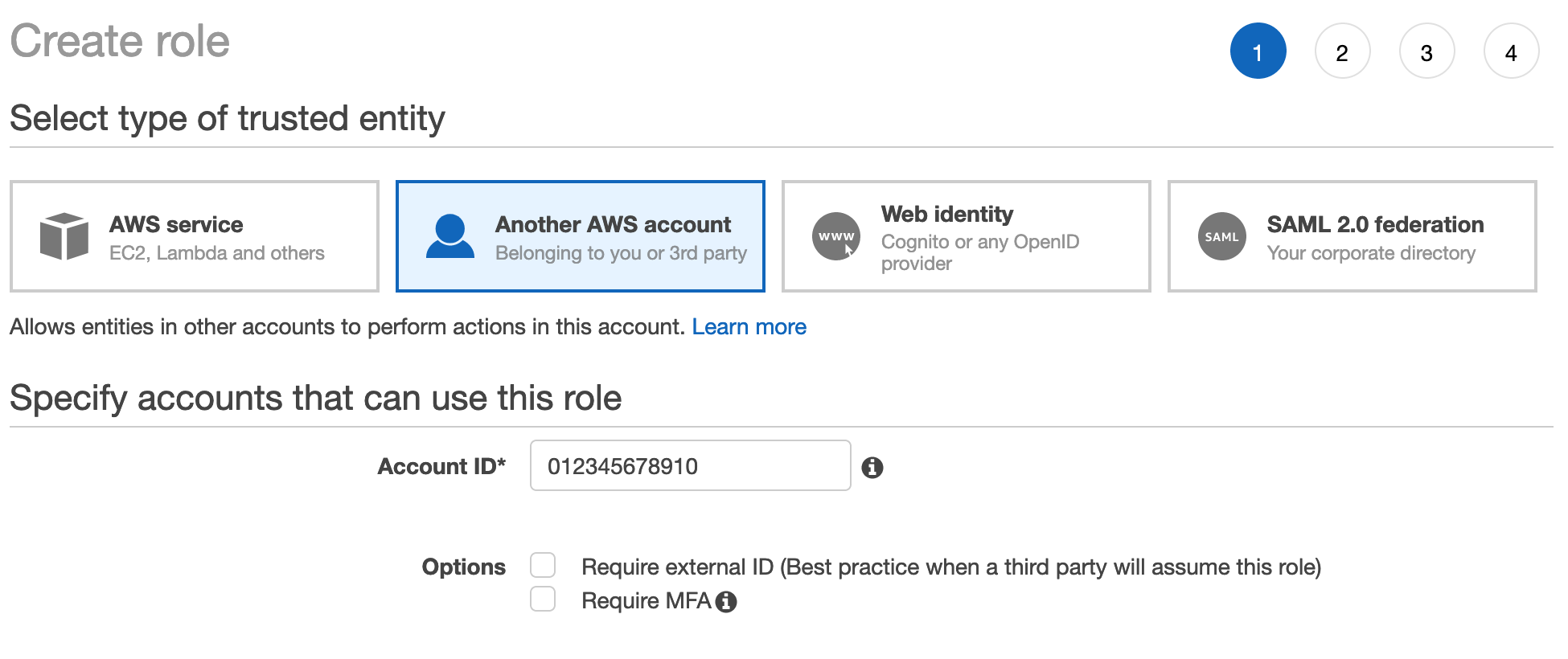
# Clone the git repository  
~ git clone https://gitlab.aws.dev/toussaiv/jenkins-cloudformation-example

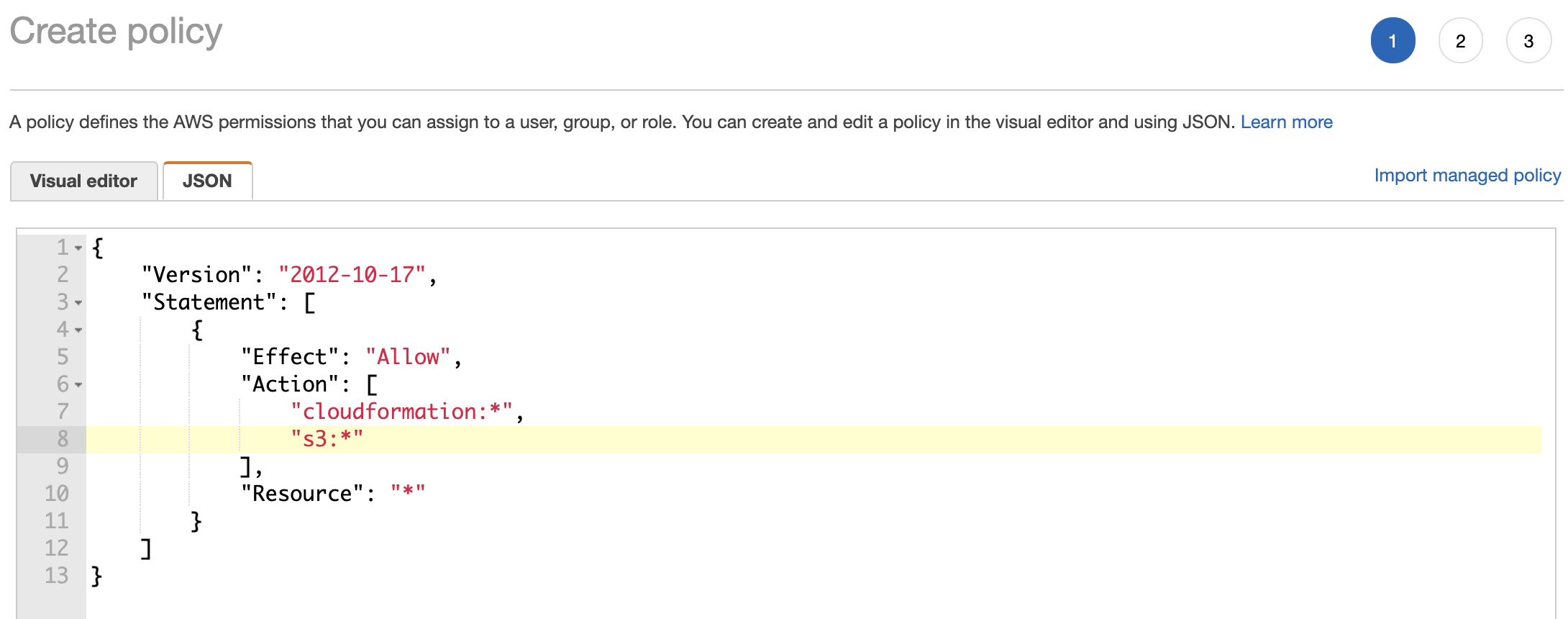
* The following parameters is an example which will vary based on your preference. If you choose to deploy with a different name, region, zone, or node capacity please modify accordingly.

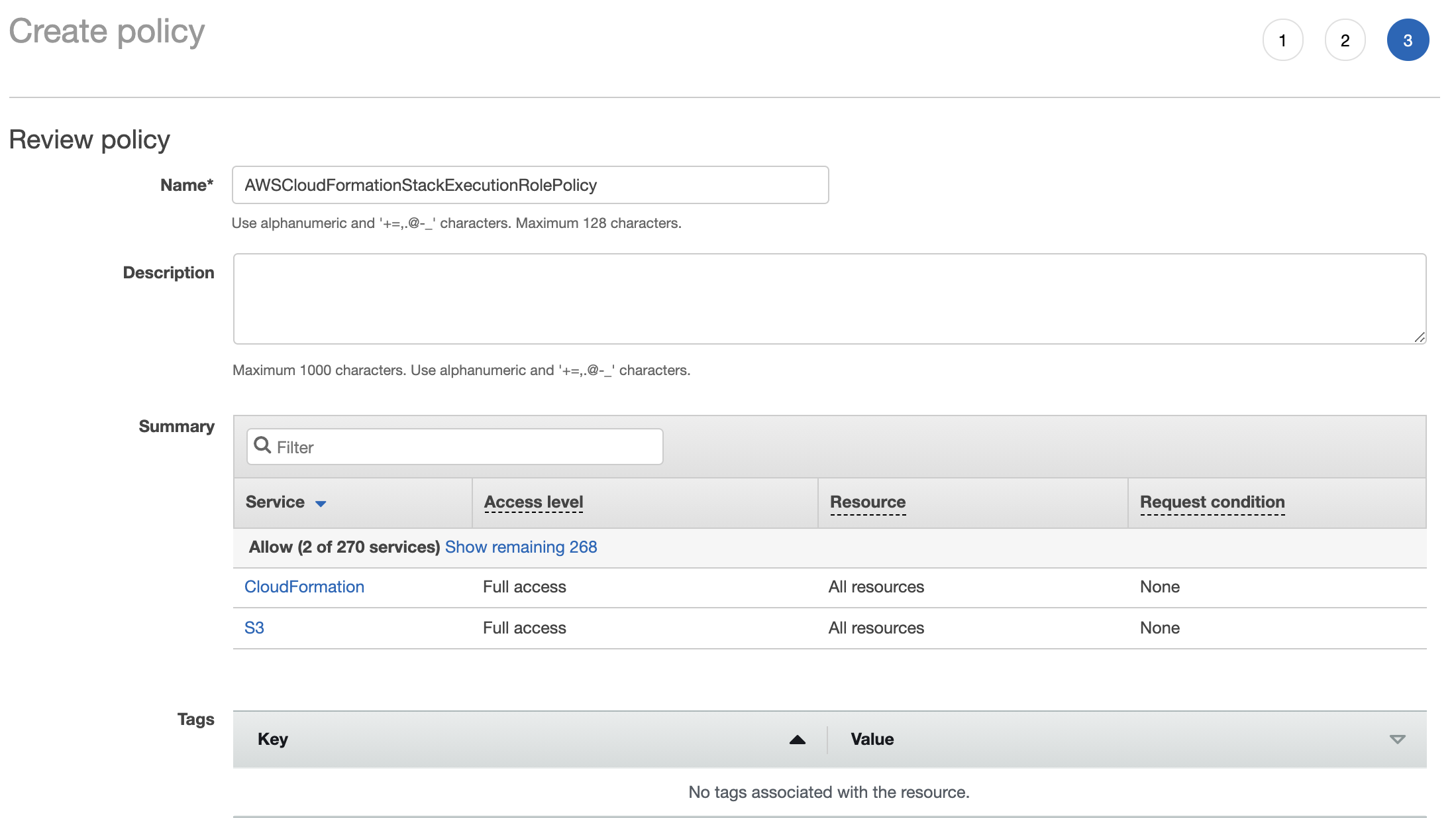
# Create the EKS Cluster  
~ eksctl create cluster \  
> --name <CLUSTER-NAME> \  
> --region <REGION> \  
> --with-oidc \  
> --zones "<AVAILABILITY-ZONE-1>,<AVAILABILITY-ZONE-2>" \  
> --nodegroup-name <NODEGROUP-NAME> \  
> --nodes-min 2 \  
> --nodes-max 4 \  
> --enable-ssm \  
> --managed \  
> --asg-access  
  
# Verify that EKS nodes are up running  
~ kubectl get nodes

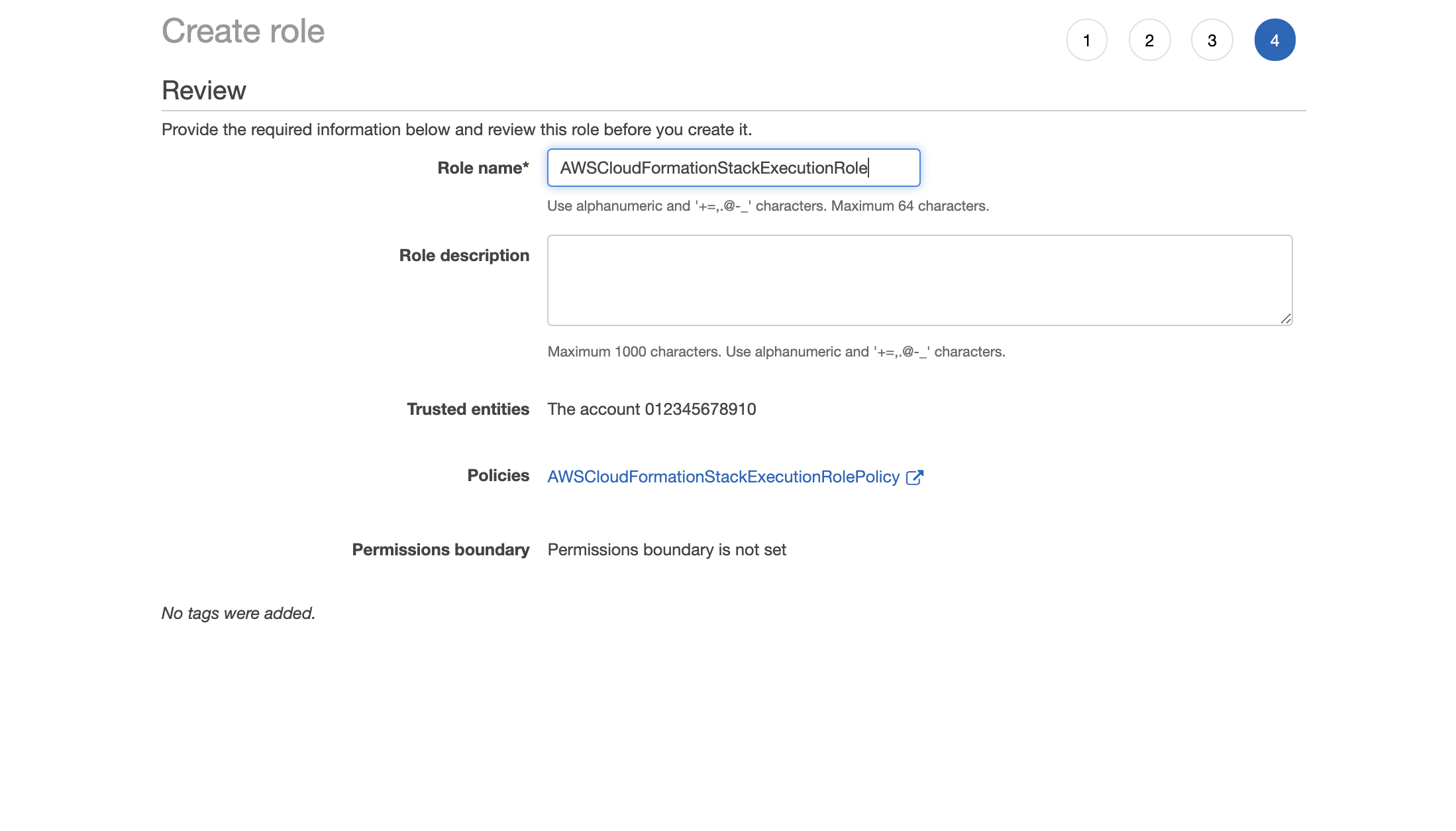
## Create Cross-Account IAM Roles

* Create an IAM role that has a common name in each target account. The role name we’ve created for use is called AWSCloudFormationStackExecutionRole. The role must have permissions to perform CloudFormation actions and any actions pertaining to the resources that will be created. In our case, we will be creating and S3 Bucket using CloudFormation.
* This IAM role must also have an established trust relationship to the Shared Services account. In this case, the Jenkins Agent will be granted the ability to assume the role of the particular target account from the Shared Services account. See the images below.

 *Figure 2. Create and IAM Role and Enter the Account ID of the Shared Services account*

 *Figure 3. Create IAM Policy with CloudFormation and S3 actions*

 *Figure 4. Review IAM Policy*

 *Figure 5. Attach IAM Policy to IAM Role and complete IAM role creation*

## Create AWS ECR Repository

* This command will create an AWS ECR Repository and will reference the ecr-permission-policy.json that will allow permissions to push and pull images. You must update the ecr-permission-policy.json with the AWS Account ID before executing the script.

{  
 "Version": "2008-10-17",  
 "Statement": [  
 {  
 "Sid": "PermissionPolicy",  
 "Effect": "Allow",  
 "Principal": {  
 "AWS": "arn:aws:iam::<AWS-ACCOUNT-ID>:root"  
 },  
 "Action": [  
 "ecr:BatchCheckLayerAvailability",  
 "ecr:BatchGetImage",  
 "ecr:CompleteLayerUpload",  
 "ecr:GetDownloadUrlForLayer",  
 "ecr:InitiateLayerUpload",  
 "ecr:PutImage",  
 "ecr:UploadLayerPart"  
 ]  
 }  
 ]  
}

# Create an AWS ECR Repository with repository permissions  
~ ./create-ecr-repo.sh "<REPOSITORY-NAME>" "<REGION-NAME>"

## Build Docker Images

* This command is used to build the custom Jenkins images for the Jenkins Manager and the Jenkins Agent. You must navigate to the docker/ directory, then execute the command according to the required parameters with the AWS account ID, repository name, region, and the build folder name “jenkins manager” or “jenkins-agent” that resides in the current docker directory. The custom docker images will contain a set of starter package installs and the AWS CLI.

# Build a docker image and push to AWS ECR Repository  
~ ./build-image.sh "<AWS-ACCOUNT-ID>" "<REPOSITORY-NAME>" "<REGION-NAME>" "<BUILD-FOLDER-NAME>"

* After you’ve built both images, navigate to the “k8s” directory and execute the manifest.yaml template to setup the Jenkins application. *(Note: This Jenkins application is not configured with a persistent volume storage, therefore you will need to establish and configure this template to fit that requirement).*

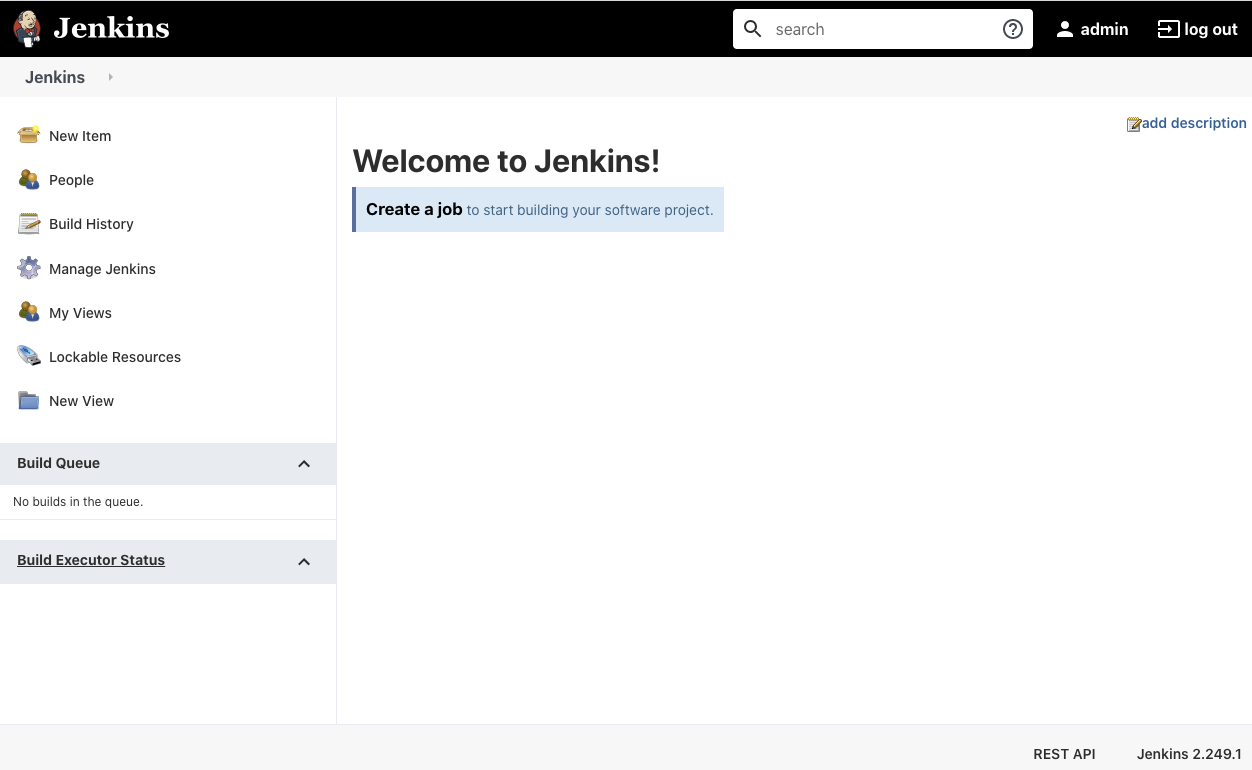
# Update kubeconfig and set the context of the cluster  
~ aws eks update-kubeconfig <CLUSTER-NAME> --region <REGION-NAME>  
  
# Apply the kubernetes manifest to deploy the Jenkins manager application  
~ kubectl apply -f manifest.yaml

## Jenkins

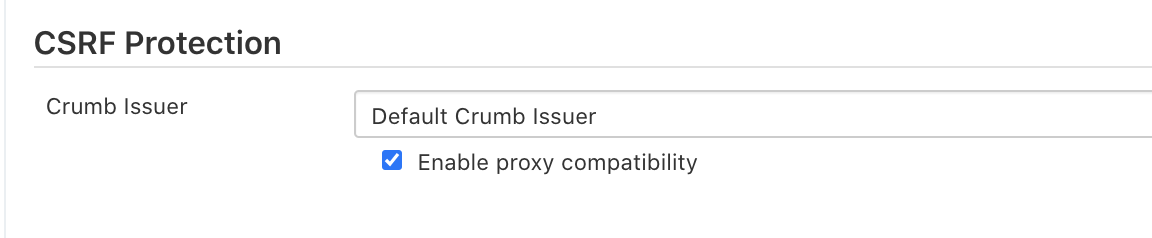
* Run the following command to make sure your EKS pods are ready and running. We have 1 pod, hence the 2/2 output below. Fetch and navigate to the Load Balancer URL. The next step is to get the password to login to Jenkins as the **admin** user. Run the following command below to get the auto generated initial Jenkins password. Please update your password after logging in.

# Fetch the Application URL or navigate to the AWS Console for the Load Balancer  
~ kubectl get svc -n jenkins  
  
# Verify that jenkins deployment/pods are up running  
~ kubectl get pods -n jenkins  
  
# Replace with jenkins manager pod name and fetch Jenkins login password  
~ kubectl exec -it pod/<JENKINS-MANAGER-POD-NAME> -n jenkins -- cat /var/jenkins\_home/secrets/initialAdminPassword

* The [Kubernetes Plugin](https://plugins.jenkins.io/kubernetes/) and [CloudBees AWS Credentials Plugin](https://plugins.jenkins.io/aws-credentials/) should be installed as part of the Jenkins image build from the Managed Plugins.

 *Figure 6. Jenkins Login Page*

* Navigate: Manage Jenkins → Configure Global Security
* Set the Crumb Issuer to remove the error pages to prevent Cross Site Request Forgery exploits.

 *Figure 7. Configure Global Security*

## Configure Jenkins Kubernetes Cloud

* Navigate: Manage Jenkins → Manage Nodes and Clouds → Configure Clouds
* Click: Add a new cloud → select Kubernetes from the drop menus

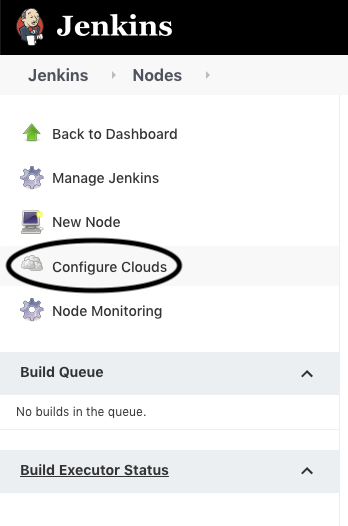
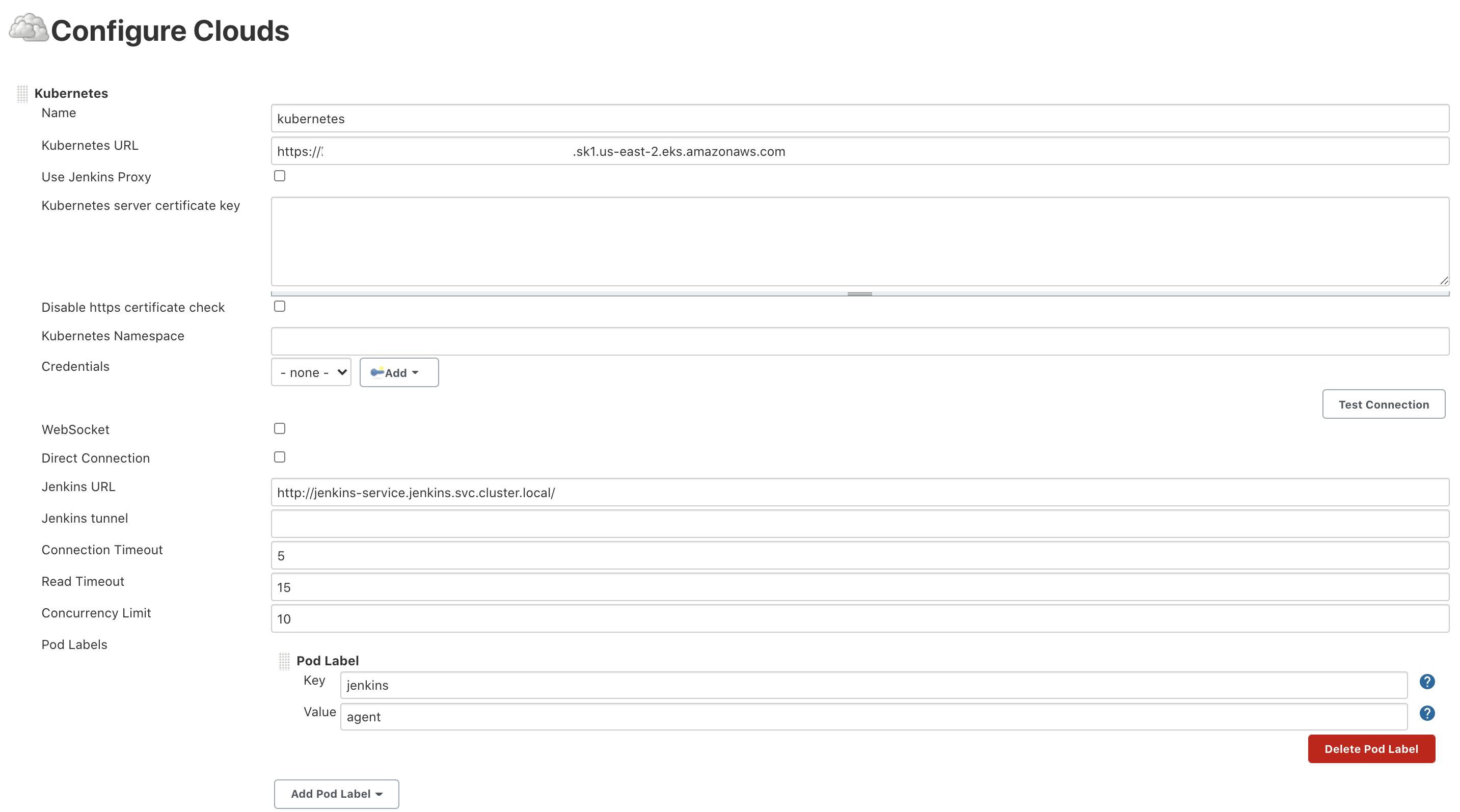


Image: img/jenkins-02.png

*Figure 8. Jenkins Configure Nodes & Clouds*

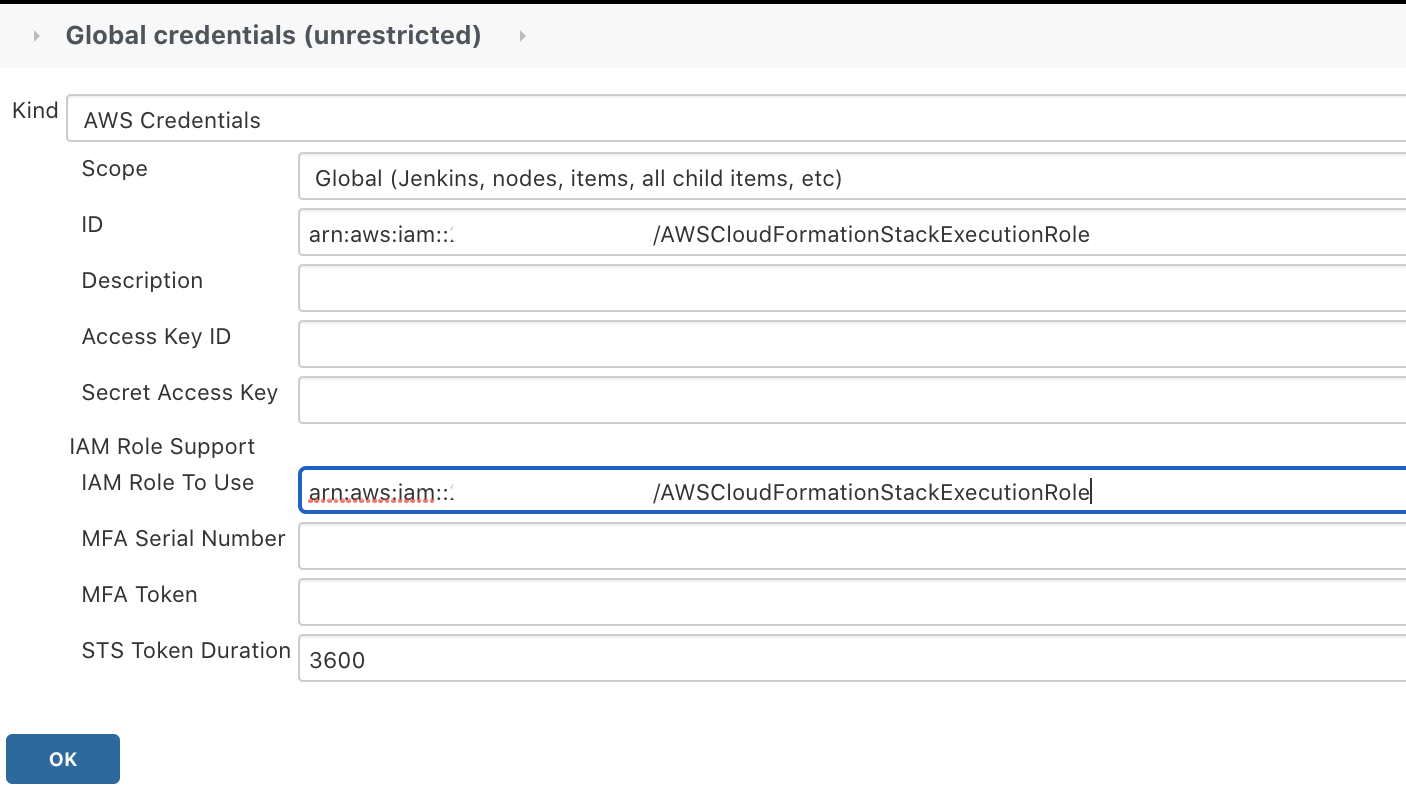
*Note: Before proceeding, please ensure that you have access to your Amazon EKS cluster information, whether it is through Console or CLI.*

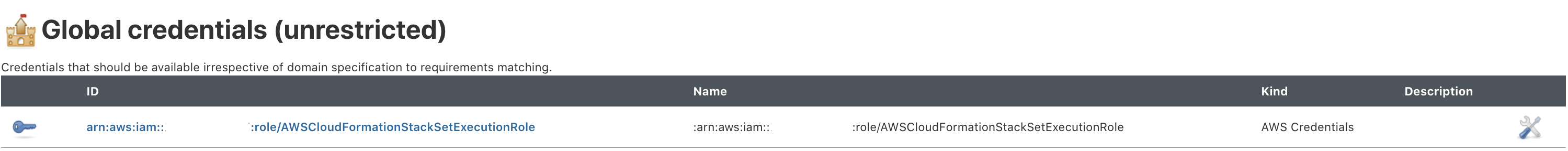
* Enter a Name in the field of the Kubernetes Cloud configuration.
* Enter the Kubernetes URL which can be found via AWS Console by navigating to the Amazon EKS service and locating the API server endpoint of the cluster or run the command kubectl cluster-info.
* Enter the namespace that will be used in the Kubernetes Namespace field. This will determine where the dynamic kubernetes pods will spawn. In our case, the name of the namespace is jenkins.
* During the initial setup of Jenkins Manager on kubernetes, there is an environment variable JENKINS\_URL which automatically uses the Load Balancer URL to resolve requests. However, we will resolve our requests locally to the cluster IP address.
  + The format is done as the following: [https://<service-name>.<namespace>.svc.cluster.local](https://(service-name).(namespace).svc.cluster.local/)

 *Figure 9. Configure Kubernetes Cloud*

## Set AWS Credentials

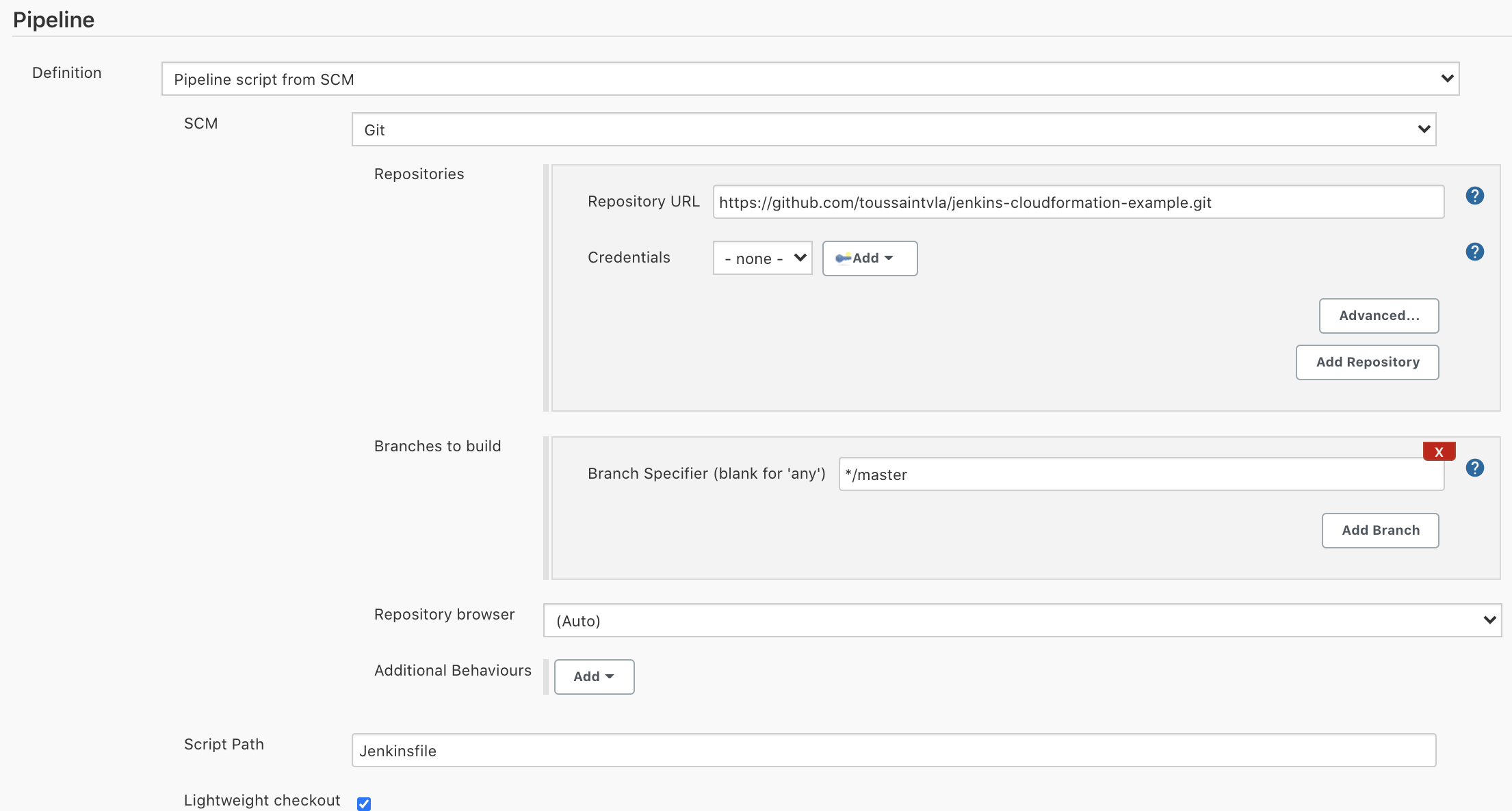
* You must have the AWS Credentials Binding Plugin installed for before this step. Enter the unique ID name as shown in the example below.
* Enter the IAM Role ARN you created earlier for both the ID and IAM Role to use in the field as shown below.

 *Figure 10. AWS Credentials Binding*

 *Figure 11. Managed Credentials*

## Create a pipeline

* Navigate to the Jenkins main menu and select new item
* Create a Pipeline

 *Figure 12. Create a Pipeline*

## Examine and modify code repository files

* Setup a Kubernetes YAML template. In this example, we will be using the k8sPodTemplate.yaml file stored in the k8s/ folder.
* The custom Jenkins Agent image we built earlier uses the Jenkins inbound-agent as the base image with the AWS CLI installed. Specify the container image in the file that will source the image with the associated AWS account and region.
* You can keep everything else as default, but depending on you specifications you can choose to modify the amount of resources that must be allocated.

# Kubernetes YAML file  
apiVersion: v1  
kind: Pod  
metadata:  
 labels:  
 some-label: jenkins-pipeline  
spec:  
 serviceAccountName: jenkins-agent # Enter the service account name being used  
 securityContext:  
 runAsUser: 0  
 fsGroup: 0  
 runAsNonRoot: false  
 containers:  
 - name: jenkins-agent  
 image: <REPLACE WITH YOUR AWS ACCOUNT>.dkr.ecr.<REPLACE WITH YOU AWS REGION>.amazonaws.com/test-jenkins-agent:latest # Enter the jenkins inbound agent image.  
 command:  
 - cat  
 tty: true  
 resources:  
 requests:  
 cpu: 100m  
 memory: 256Mi  
 limits:  
 cpu: "1"  
 memory: 1024Mi

* The deploy-stack.sh accepts four different parameters. The first parameter set is the stack name. The second parameter is the name of the parameters file name which resides in the parameters/ folder. The third parameter is the name of the template which reside in the cloudformation/ folder. The fourth parameter is the boolean condition to decide whether to execute the deployment right away or create a changeset. The fifth parameter is the region of the target account where the stack should be deployed. I will show an example of how this is used later.

#!/bin/bash  
  
# Please ensure that you have the correct AWS credentials configured.  
# Enter the name of the stack, the parameters file name, the template name, then changeset condition, and finally the region name.  
  
if [ $# -ne 5 ]; then  
 echo "Enter stack name, parameters file name, template file name to create, set changeset value (true or false), and enter region name. "  
 exit 0  
else  
 STACK\_NAME=$1  
 PARAMETERS\_FILE\_NAME=$2  
 TEMPLATE\_NAME=$3  
 CHANGESET\_MODE=$4  
 REGION=$5  
fi  
  
if [ ! -f "cloudformation/$TEMPLATE\_NAME.yaml" ]; then  
 echo "CloudFormation template $TEMPLATE\_NAME.yaml does not exist"  
 exit 0  
fi  
  
if [ ! -f "parameters/$PARAMETERS\_FILE\_NAME.properties" ]; then  
 echo "CloudFormation parameters $PARAMETERS\_FILE\_NAME.properties does not exist"  
 exit 0  
fi  
  
if [[ $CHANGESET\_MODE == true ]]; then  
 aws cloudformation deploy \  
 --stack-name $STACK\_NAME \  
 --template-file cloudformation/$TEMPLATE\_NAME.yaml \  
 --parameter-overrides file://parameters/$PARAMETERS\_FILE\_NAME.properties \  
 --capabilities CAPABILITY\_NAMED\_IAM \  
 --region $REGION  
else  
 aws cloudformation deploy \  
 --stack-name $STACK\_NAME \  
 --template-file cloudformation/$TEMPLATE\_NAME.yaml \  
 --parameter-overrides file://parameters/$PARAMETERS\_FILE\_NAME.properties \  
 --capabilities CAPABILITY\_NAMED\_IAM \  
 --region $REGION \  
 --no-execute-changeset  
fi

* The delete-stack.sh accepts the name and region of the stack that was created to delete the stack. I will show an example of how this is used later.

#!/bin/bash  
  
# Please ensure that you have the correct AWS credentials configured.  
# Enter the name of the stack you want to delete, then enter the name of the region.  
  
if [ $# -ne 2 ]; then  
 echo "Enter stack name to delete & region name."  
 exit 0  
else  
 STACK\_NAME=$1  
 REGION=$2  
fi  
  
aws cloudformation delete-stack \  
--stack-name $STACK\_NAME \  
--region $REGION  
  
aws cloudformation wait stack-delete-complete \  
--stack-name $STACK\_NAME \  
--region $REGION

* In this Jenkinsfile pipeline, the k8sPodTemplate.yaml is used to specify the kubernetes pod details and the inbound-agent that will be used to run the pipeline.
* This pipeline stage will consist of the several stages for stack deployment, create changeset, execute changeset, and stack deletion. The deploy-stack.sh will execute when selected via parameters, and likewise the delete-stack.sh will be executed when selected by the parameters.
* If you observe closely, there are several variables used within the pipeline stage actions below
  + CHANGESET\_MODE = True, this will proceed to deploy a stack or execute changeset.
  + CHANGESET\_MODE = False, this will only create a changeset without executing the changes.
  + STACK\_NAME = example-stack, In this example the name of the stack is called *example-stack*.
  + PARAMETERS\_FILE\_NAME = example-stack-parameters, this will pass the parameter values into the stack, <parameter-file-name>.properties. In our case we are using example-stack-parameters.properties under the parameters/ folder.
  + TEMPLATE\_NAME = S3-Bucket, The name of this variable is equivalent to the format <template-name>.yaml. In this example the template name is called S3-Bucket, under the cloudformation/ folder.
  + CFN\_CREDENTIALS\_ID = arn:aws:iam::role/AWSCloudFormationStackExecutionRole, This is the Unique ID that references the IAM Role ARN which is the account we will assume role using the [AmazonWebServicesCredentialsBinding](https://www.jenkins.io/doc/pipeline/steps/credentials-binding/) to perform our deployment based on the selected choice parameterized pipeline.
  + REGION = us-east-1, Enter the region you’re using for the target account.
  + TOGGLE = true, If you do not set the toggle flag to true before executing the build action, it will automatically abort the pipeline for any action. This is to prevent accidental build execution changes from taking place without confirmation.
* The commands below will utilize the deploy-stack.sh and delete-stack.sh files we showed earlier.

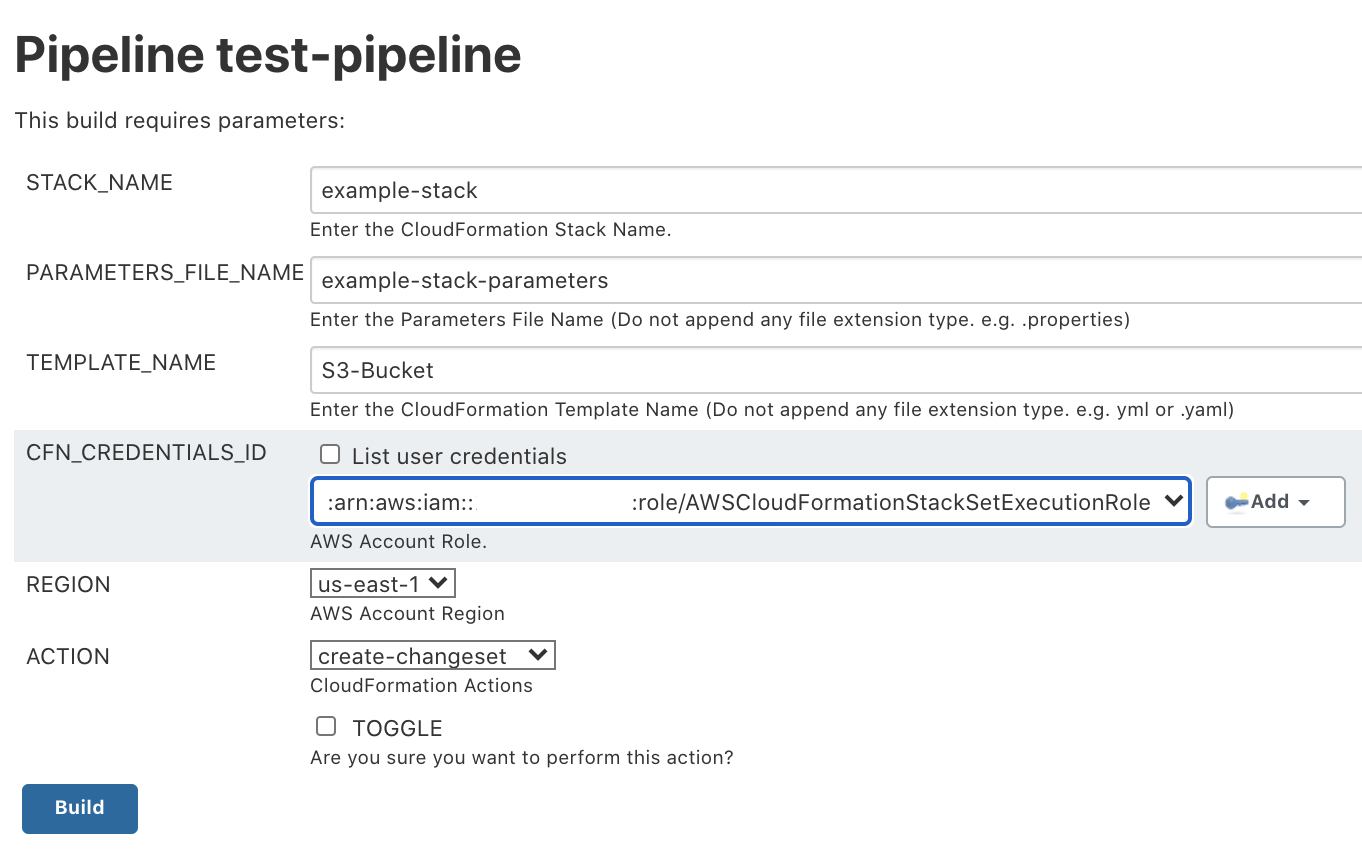
# Deploy a Stack or Execute a Changeset  
~ cloudformation/deploy-stack.sh ${STACK\_NAME} ${PARAMETERS\_FILE\_NAME} ${TEMPLATE\_NAME} ${CHANGESET\_MODE} ${REGION}  
  
# Delete a CloudFormation Stack  
~ cloudformation/delete-stack.sh ${STACK\_NAME} ${REGION}

* The steps in the pipeline will execute based on the parameter choices you select.

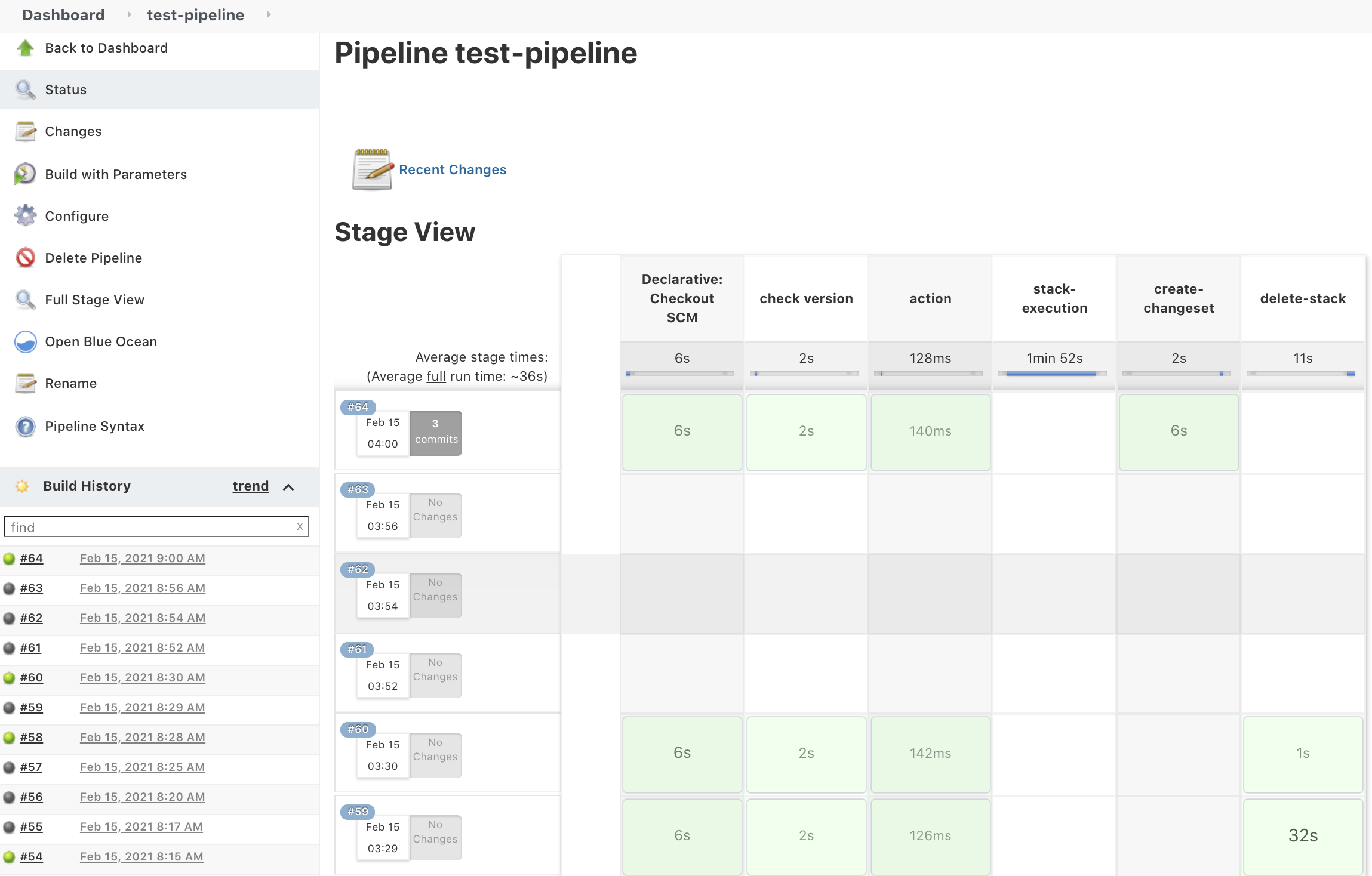
pipeline {  
 agent {  
 kubernetes {  
 // This is a YAML representation of the Pod, to allow setting any values not supported as fields.  
 yamlFile 'k8s/k8sPodTemplate.yaml' // Declarative agents can be defined from YAML.  
 }  
 }  
  
 parameters {  
 string(name: 'STACK\_NAME', defaultValue: 'example-stack', description: 'Enter the CloudFormation Stack Name.')  
 string(name: 'PARAMETERS\_FILE\_NAME', defaultValue: 'example-stack-parameters', description: 'Enter the Parameters File Name (Do not append any file extension type. e.g. .properties)')  
 string(name: 'TEMPLATE\_NAME', defaultValue: 'S3-Bucket', description: 'Enter the CloudFormation Template Name (Do not append any file extension type. e.g. yml or .yaml)')  
 credentials(name: 'CFN\_CREDENTIALS\_ID', defaultValue: '', description: 'AWS Account Role.', required: true)  
 choice(  
 name: 'REGION',  
 choices: [  
 'us-east-1',  
 'us-east-2'  
 ],  
 description: 'AWS Account Region'  
 )  
 choice(  
 name: 'ACTION',  
 choices: ['create-changeset', 'execute-changeset', 'deploy-stack', 'delete-stack'],  
 description: 'CloudFormation Actions'  
 )  
 booleanParam(name: 'TOGGLE', defaultValue: false, description: 'Are you sure you want to perform this action?')  
 }  
  
 stages {  
  
 stage('check version') {  
 steps {  
 ansiColor('xterm') {  
 container("jenkins-agent") {  
 sh 'aws --version'  
 sh 'aws sts get-caller-identity'  
 }  
 }  
 }  
 }  
  
 stage('action') {  
 when {  
 expression { params.ACTION == 'create-changeset' || params.ACTION == 'execute-changeset' || params.ACTION == 'deploy-stack' || params.ACTION == 'delete-stack'}  
 }  
 steps {  
 ansiColor('xterm') {  
 script {  
 if (!params.TOGGLE) {  
 currentBuild.result = 'ABORTED' //If you do not set the toggle flag to true before executing the build action, it will automatically abort the pipeline for any action.  
 } else {  
 if (params.ACTION == 'create-changeset') {  
 env.CHANGESET\_MODE = false  
 } else {  
 env.CHANGESET\_MODE = true  
 }  
 }  
 }  
 }  
 }  
 }  
  
 stage('stack-execution') {  
 when {  
 expression { params.ACTION == 'deploy-stack' || params.ACTION == 'execute-changeset' }  
 }  
 steps {  
 ansiColor('xterm') {  
 container("jenkins-agent") {  
 withCredentials([[  
 $class: 'AmazonWebServicesCredentialsBinding',  
 credentialsId: "${CFN\_CREDENTIALS\_ID}",  
 accessKeyVariable: 'AWS\_ACCESS\_KEY\_ID',  
 secretKeyVariable: 'AWS\_SECRET\_ACCESS\_KEY']]) {  
 sh 'cloudformation/deploy-stack.sh ${STACK\_NAME} ${PARAMETERS\_FILE\_NAME} ${TEMPLATE\_NAME} ${CHANGESET\_MODE} ${REGION}'  
 }  
 }  
 }  
 }  
 }  
  
 stage('create-changeset') {  
 when {  
 expression { params.ACTION == 'create-changeset' }  
 }  
 steps {  
 ansiColor('xterm') {  
 container("jenkins-agent") {  
 withCredentials([[  
 $class: 'AmazonWebServicesCredentialsBinding',  
 credentialsId: "${CFN\_CREDENTIALS\_ID}",  
 accessKeyVariable: 'AWS\_ACCESS\_KEY\_ID',  
 secretKeyVariable: 'AWS\_SECRET\_ACCESS\_KEY']]) {  
 sh 'cloudformation/deploy-stack.sh ${STACK\_NAME} ${PARAMETERS\_FILE\_NAME} ${TEMPLATE\_NAME} ${CHANGESET\_MODE} ${REGION}'  
 }  
 }  
 }  
 }  
 }  
  
 stage('delete-stack') {  
 when {  
 expression { params.ACTION == 'delete-stack' }  
 }  
 steps {  
 ansiColor('xterm') {  
 container("jenkins-agent") {  
 withCredentials([[  
 $class: 'AmazonWebServicesCredentialsBinding',  
 credentialsId: "${CFN\_CREDENTIALS\_ID}",  
 accessKeyVariable: 'AWS\_ACCESS\_KEY\_ID',  
 secretKeyVariable: 'AWS\_SECRET\_ACCESS\_KEY']]) {  
 sh 'cloudformation/delete-stack.sh ${STACK\_NAME} ${REGION}'  
 }  
 }  
 }  
 }  
 }  
  
 }  
}

## Jenkins Pipeline: Execute a pipeline

* Click Build with Parameters then select a build action.

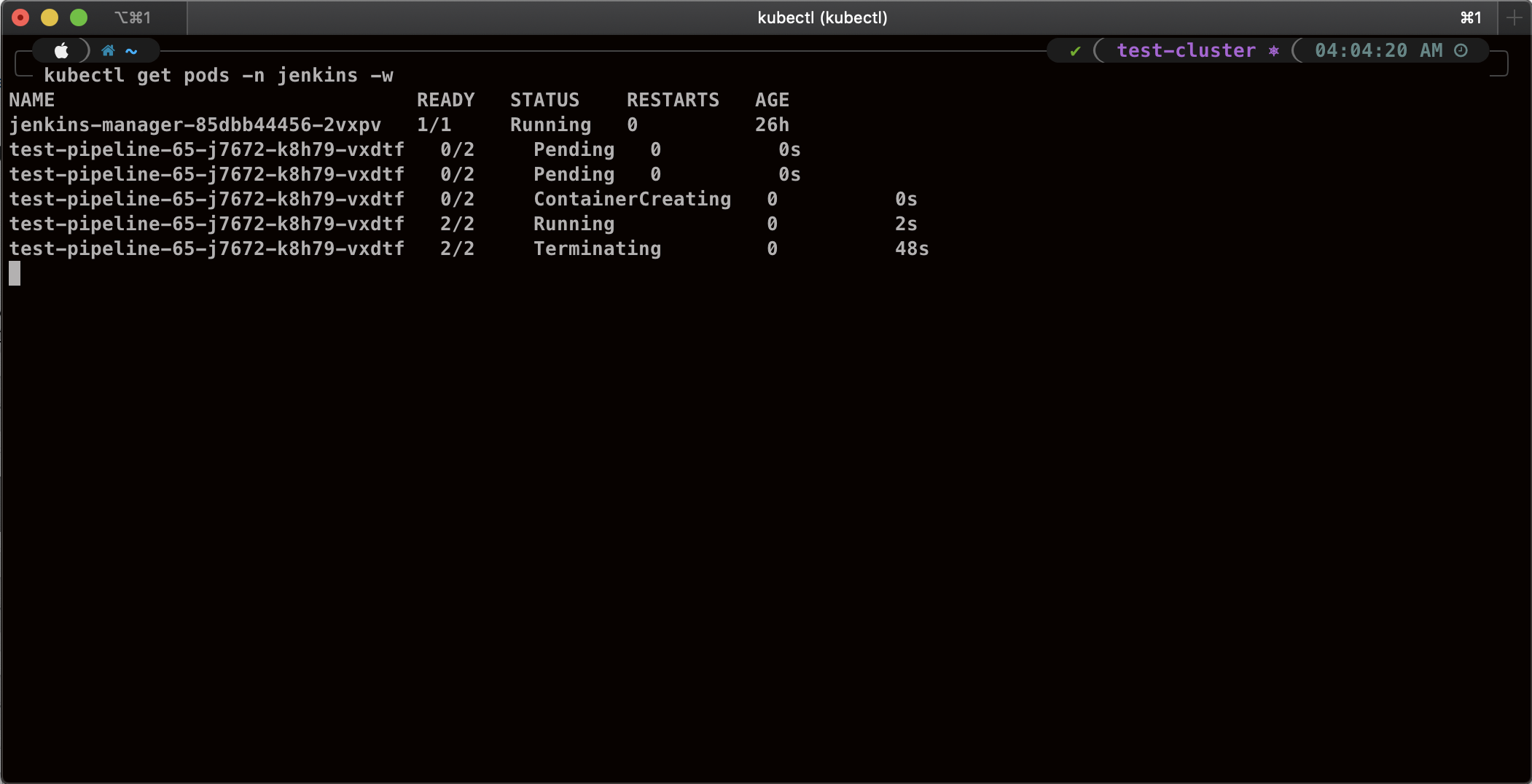
 *Figure 13. Build with Parameters*

* You can examine the pipeline stages a bit further for the choice you selected. You can view more details of the stages below and verify in your AWS account that the CloudFormation stack was executed.

 *Figure 14. Pipeline Stage View*

* The Final Step is to execute your pipeline and watch the pods spin up dynamically in your terminal. As you can see below the jenkins agent pod spawned then terminated after the work was completed. You can watch this task on your own by executing the following command:

# Watch the pods spawn in the "jenkins" namespace  
~ kubectl get pods -n jenkins -w

 *Figure 15. Watch Jenkins Agent Pods Spawn*

## Code Repository

* [Amazon EKS Jenkins Integration](https://gitlab.aws.dev/toussaiv/jenkins-cloudformation-example)

## References

* [Jenkins Kubernetes Plugin](https://plugins.jenkins.io/kubernetes/)
* [CloudBees AWS Credentials Plugin](https://plugins.jenkins.io/aws-credentials/)

## Conclusion

This post guided you through the process of building out Amazon EKS and integrating Jenkins to Orchestrate Workloads. We demonstrated how you can use this to deploy in multiple accounts with dynamic Jenkins agents and create alignment to your business with similar use cases. To learn more about Amazon EKS, head over to our [documentation](https://aws.amazon.com/eks/getting-started/) pages or explore our [console](https://console.aws.amazon.com/eks/home?region=us-east-1#).