**Pulumi Reference Code:**

package main

*import (*

*"github.com/pulumi/pulumi-aws/sdk/v4/go/aws/s3"*

*"github.com/pulumi/pulumi/sdk/v4/go/pulumi"*

*)*

*func main() {*

*pulumi.Run(func(ctx \*pulumi.Context) error {*

*// Create an S3 bucket*

*bucket, err := s3.NewBucket(ctx, "myBucket", nil)*

*if err != nil {*

*return err*

*}*

*// Export the bucket name*

*ctx.Export("bucketName", bucket.ID())*

*return nil*

*})*

*}*

**Kubernetes Reference Code:**

package main

import (

"context"

"fmt"

"os"

"k8s.io/client-go/kubernetes"

"k8s.io/client-go/tools/clientcmd"

"k8s.io/client-go/util/homedir"

"k8s.io/client-go/util/rest"

)

func main() {

home := homedir.HomeDir()

kubeconfig := filepath.Join(home, ".kube", "config")

// use the current context in kubeconfig

config, err := clientcmd.BuildConfigFromFlags("", kubeconfig)

if err != nil {

panic(err.Error())

}

// create the clientset

clientset, err := kubernetes.NewForConfig(config)

if err != nil {

panic(err.Error())

}

// Define a simple Deployment

deployment := &appsv1.Deployment{

ObjectMeta: metav1.ObjectMeta{

Name: "example-deployment",

},

Spec: appsv1.DeploymentSpec{

Replicas: int32Ptr(3),

Selector: &metav1.LabelSelector{

MatchLabels: map[string]string{

"app": "example-app",

},

},

Template: corev1.PodTemplateSpec{

ObjectMeta: metav1.ObjectMeta{

Labels: map[string]string{

"app": "example-app",

},

},

Spec: corev1.PodSpec{

Containers: []corev1.Container{

{

Name: "example-container",

Image: "nginx:latest",

},

},

},

},

},

}

// Create the Deployment

result, err := clientset.AppsV1().Deployments("default").Create(context.TODO(), deployment, metav1.CreateOptions{})

if err != nil {

panic(err.Error())

}

fmt.Printf("Created Deployment %q.\n", result.GetObjectMeta().GetName())

}

func int32Ptr(i int32) \*int32 { return &i }

**Automation Scripts with Golang:**

*package main*

*import (*

*"fmt"*

*"io/ioutil"*

*"time"*

*)*

*const (*

*sourceFilePath = "input.txt"*

*destinationFilePath = "backup.txt"*

*interval = 5 \* time.Minute*

*)*

*func readFileContent(filePath string) ([]byte, error) {*

*content, err := ioutil.ReadFile(filePath)*

*if err != nil {*

*return nil, err*

*}*

*return content, nil*

*}*

*func writeFileContent(filePath string, content []byte) error {*

*err := ioutil.WriteFile(filePath, content, 0644)*

*if err != nil {*

*return err*

*}*

*return nil*

*}*

*func main() {*

*for {*

*sourceContent, err := readFileContent(sourceFilePath)*

*if err != nil {*

*fmt.Println("Error reading source file:", err)*

*time.Sleep(interval)*

*continue*

*}*

*destinationContent, err := readFileContent(destinationFilePath)*

*if err != nil {*

*fmt.Println("Error reading destination file:", err)*

*time.Sleep(interval)*

*continue*

*}*

*if string(sourceContent) != string(destinationContent) {*

*err := writeFileContent(destinationFilePath, sourceContent)*

*if err != nil {*

*fmt.Println("Error writing to destination file:", err)*

*} else {*

*fmt.Println("File content updated successfully.")*

*}*

*} else {*

*fmt.Println("No changes detected.")*

*}*

*time.Sleep(interval)*

*}*

*}*

**AWS Reference code with Golang:**

package main

import (

"context"

"fmt"

"os"

"os/exec"

"path/filepath"

"strings"

"time"

"github.com/aws/aws-sdk-go-v2/aws"

"github.com/aws/aws-sdk-go-v2/config"

"github.com/aws/aws-sdk-go-v2/service/elasticbeanstalk"

)

const (

applicationName = "YourApplicationName"

environmentName = "YourEnvironmentName"

zipFileName = "app.zip"

)

func main() {

// Build the Golang application

if err := runCommand("go", "build", "-o", "myapp"); err != nil {

fmt.Println("Error building Golang application:", err)

os.Exit(1)

}

// Create a ZIP file with the application code

if err := runCommand("zip", "-r", zipFileName, "myapp"); err != nil {

fmt.Println("Error creating ZIP file:", err)

os.Exit(1)

}

// Deploy to Elastic Beanstalk

if err := deployToElasticBeanstalk(); err != nil {

fmt.Println("Error deploying to Elastic Beanstalk:", err)

os.Exit(1)

}

fmt.Println("Deployment successful!")

}

func deployToElasticBeanstalk() error {

cfg, err := config.LoadDefaultConfig(context.TODO())

if err != nil {

return err

}

client := elasticbeanstalk.NewFromConfig(cfg)

// Create a new Elastic Beanstalk application version

createAppVersionOutput, err := client.CreateApplicationVersion(context.TODO(), &elasticbeanstalk.CreateApplicationVersionInput{

ApplicationName: aws.String(applicationName),

VersionLabel: aws.String(time.Now().Format("20060102150405")),

SourceBundle: &elasticbeanstalk.S3Location{

S3Bucket: aws.String("your-s3-bucket-name"),

S3Key: aws.String(zipFileName),

},

})

if err != nil {

return err

}

fmt.Printf("Application version created: %s\n", \*createAppVersionOutput.ApplicationVersion.VersionLabel)

// Update the Elastic Beanstalk environment to use the new version

\_, err = client.UpdateEnvironment(context.TODO(), &elasticbeanstalk.UpdateEnvironmentInput{

ApplicationName: aws.String(applicationName),

EnvironmentName: aws.String(environmentName),

VersionLabel: createAppVersionOutput.ApplicationVersion.VersionLabel,

})

if err != nil {

return err

}

fmt.Printf("Environment updated: %s\n", environmentName)

return nil

}

func runCommand(command string, args ...string) error {

cmd := exec.Command(command, args...)

cmd.Stdout = os.Stdout

cmd.Stderr = os.Stderr

return cmd.Run()

}

func cleanUp() {

// Clean up temporary files if needed

if err := os.Remove("myapp"); err != nil {

fmt.Println("Error cleaning up:", err)

}

if err := os.Remove(zipFileName); err != nil {

fmt.Println("Error cleaning up:", err)

}

}

func init() {

// Ensure clean-up is performed even if the script exits with an error

cleanUp()

}

func cleanupOnExit() {

if r := recover(); r != nil {

// Recovered from panic, perform clean-up

cleanUp()

}

}

func main() {

defer cleanupOnExit()

automation

// Your main function code here

}