PIPELINES DI CD/CI CROSS-ACCOUNT CON AWS CODEPIPELINE

Amazon CloudWatch | AWS CodePipeline | CI/CD | Continuous Deployment | Continuous Integration

beSharp | 26 Luglio 2019

La suddivisione dei progetti software Cloud su più account AWS è una best practice Amazon consolidata ed estremamente importante per assicurare la segmentazione delle competenze delle varie entità (internal stakeholder, subcontractors etc.) che stanno partecipando al progetto. In questo contesto risulta spesso consigliabile isolare in un account dedicato i vari repository contenenti il codice del progetto (e.g. frontend, backend diviso in microservizi, app mobile) e hostati su AWS CodeCommit. Questo permette di dare permessi granulari ai vari sviluppatori, per lavorare solo sulla parte del codice di loro effettiva competenza.

Tuttavia, per velocizzare le operazioni di deploy del codice nei vari ambienti (e.g sviluppo, staging, produzione) e renderle resilienti a possibili errori umani, risulta essenziale creare delle pipeline di CD/CI che si occupino di effettuare il deploy del codice sui relativi ambienti in account AWS differenti.

Per fare questo è possibile utilizzare CodePipeline, il servizio managed di pipelines offerto da AWS. Tramite CodePipeline è infatti possibile effettuare il source del codice da un repository Git da CodeCommit, eseguire i test automatici tramite un container Docker creato per questo scopo (utilizzando il servizio CodeBuild) ed infine procedere al deploy del codice. CodePipeline supporta il deploy “classico” di applicativi su macchine EC2 tramite AWS CodeDeploy, il deploy di servizi serverless (Lambda Functions) tramite AWS CloudFormation e il deploy di container su Amazon ECS, sia in modalità serverless con Fargate (container as-a-service) che nella modalità tradizionale con i container hostati su istanze EC2.

La creazione di una Pipeline cross-account, tuttavia, può essere effettuata esclusivamente utilizzando la AWS CLI, in quanto – ad oggi – non è possibile eseguire questa operazione dalla web console. Inoltre, la creazione di questo tipo di pipeline richiede una serie di passaggi preparatori

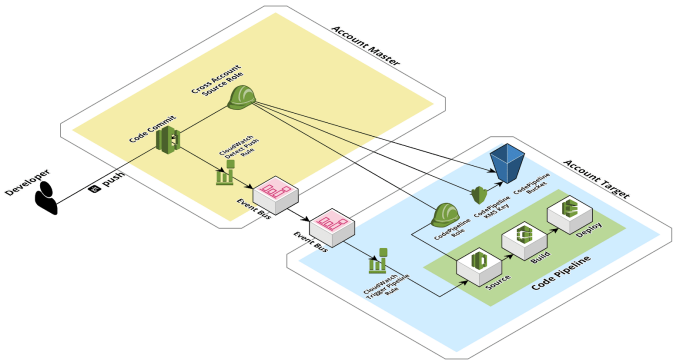
aggiuntivi, che a breve andremo a descrivere, rispetto alla creazione di una pipeline su singolo account.

Nella configurazione qui proposta, le pipeline vengono create direttamente nell'account su cui verrà eseguito il deploy: per esempio, la pipeline che si occupa di eseguire la build e il deploy del ramo development si trova sull'account dove è presente l'infrastruttura AWS relativa all'ambiente di development. Gli sviluppatori vengono informati dell'esito dei vari step della pipeline tramite un sistema di notifiche automatico tramite Cloudwatch events ed SNS. Tuttavia è anche possibile, se necessario, creare le pipeline nell'account dove è presente il repository ed eseguire solo lo step di deploy nell'account contenente l'infrastruttura. Alternativamente, è possibile utilizzare un account dedicato alle sole pipelines, che eseguono il source dall'account dei repositories e, una volta completata la build, procedono al deploy del codice negli account dei vari ambienti.

Prima di procedere con la descrizione dei vari step necessari per la creazione di una Pipeline cross-account è necessario definire un po' di nomenclatura:

- **Account Master: è l'account su cui è presente il repository che verrà utilizzato come sorgente del codice.**
- **Account Target**: è l'account dove è presente l'infrastruttura dell'ambiente sul quale verrà eseguito il deploy. La pipeline verrà creata su questo account.
- **CodePipeline Bucket**: Il bucket S3 al quale si appoggia la pipeline per salvare i bundle del codice che vengono utilizzati dai vari step.
- **CodePipeline Role**: Il Ruolo AWS che viene utilizzato dalla pipeline in fase di esecuzione. Questo ruolo deve essere creato nell'Account Target.
- **Cross Account Source Role**: il ruolo che viene assunto dalla pipeline per eseguire lo step di source del codice. Questo ruolo deve essere creato nell'Account Master.
- **CodePipeline KMS Key**: chiave KMS (AWS Key Management Service) che viene utilizzata per criptare i file creati dalla pipeline nel CodePipeline Bucket. Va creata prima della creazione della pipeline nell'account Target ed è necessario configurarla in modo che sia utilizzabile anche dall'Account Master.
- **CloudWatch Events Role**: Ruolo che userà CloudWatch per avviare la pipeline a seguito di un commit.
- **Event Buses**: servizio di AWS che consente di trasferire eventi di CloudWatch da un'account ad un altro.

I componenti sopra descritti e le loro relazioni reciproche sono rappresentati schematicamente in figura.



Procediamo ora con la descrizione passo a passo della procedura utilizzata: per prima cosa è necessario procedere alla creazione della CodePipeline KMS Key. Nella schermata riguardante i permessi è necessario inserire l'opzione per concedere l'utilizzo della chiave anche all'account Master.

A questo punto bisogna creare il CodePipeline Bucket con una bucket policy di questo tipo:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "<CodePipeline Bucket ARN>/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "<CodePipeline Bucket ARN>/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        },
        {
            "Sid": "crossaccount",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::<Account Master ID>:role/<Cross Account Source Ro
le>"
            },
            "Action": "s3:*",
            "Resource": [
                "<CodePipeline Bucket ARN>",
                "<CodePipeline Bucket ARN>/*",
```

```
                "<CodePipeline Bucket ARN>/**"
            ]
        }
    ]
}
```

Questo tipo di policy consente l'accesso al bucket al ruolo Cross Account Source Role e impedisce l'upload di file non criptati nel bucket S3. Al posto di <Cross Account Source Role> nella policy va inserito il nome che si desidera usare per il Cross Account Source Role.

Dopo aver creato la chiave ed il bucket è il momento di creare i due ruoli CodePipeline Role e Cross Account Source Role rispettivamente nell' Account Target e nell'Account Master. Ognuno dei 2 ruoli avrà tre policy:

- **Cross Account Source Role:**

  - Permesso di lettura da codecommit:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:BatchGet*",
                "codecommit:Get*",
                "codecommit:List*",
                "codecommit:Describe*",
                "codecommit:Post*",
                "codecommit:Merge*",
                "codecommit:Test*",
                "codecommit:Update*",
                "codecommit:UploadArchive",
                "codecommit:GitPull"
            ],
            "Resource": [
                "arn:aws:codecommit:eu-west-1:<Account Master ID>:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "codecommit:ListRepositories",
            "Resource": "*"
        }
    ]
}
```

- Permesso di usare la CodePipeline KMS Key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:ReEncrypt*",
                "kms:Decrypt"
            ],
            "Resource": [
                "<CodePipeline KMS Key ARN>"
            ]
        }
    ]
}
```

- Permesso di accesso al CodePipeline Bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example",
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "<CodePipeline Bucket ARN>/*",
                "<CodePipeline Bucket ARN>/**"
            ]
        }
    ]
}
```

- **CodePipeline Role:**

- Permesso di fare assume role sul ruolo Cross Account Source Role nell'account master:

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "<Cross Account Source Role ARN>"
    }
}
```

- Codepipeline Service Policy:

```
{
    "Statement": [
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetBucketVersioning"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline*",
                "arn:aws:s3:::elasticbeanstalk*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "codecommit:CancelUploadArchive",
                "codecommit:GetBranch",
                "codecommit:GetCommit",
                "codecommit:GetUploadArchiveStatus",
                "codecommit:UploadArchive"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetApplicationRevision",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
```

```
            "Action": [
                "elasticbeanstalk:*",
                "ec2:*",
                "elasticloadbalancing:*",
                "autoscaling:*",
                "cloudwatch:*",
                "s3:*",
                "sns:*",
                "cloudformation:*",
                "rds:*",
                "sqs:*",
                "ecs:*",
                "iam:PassRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "opsworks:CreateDeployment",
                "opsworks:DescribeApps",
                "opsworks:DescribeCommands",
                "opsworks:DescribeDeployments",
                "opsworks:DescribeInstances",
                "opsworks:DescribeStacks",
                "opsworks:UpdateApp",
                "opsworks:UpdateStack"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStacks",
                "cloudformation:UpdateStack",
                "cloudformation:CreateChangeSet",
                "cloudformation:DeleteChangeSet",
                "cloudformation:DescribeChangeSet",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:SetStackPolicy",
                "cloudformation:ValidateTemplate",
                "iam:PassRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
```

```
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

- Permesso di accesso al CodePipeline Bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example",
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "<CodePipeline Bucket ARN>/*",
                "<CodePipeline Bucket ARN>/**"
            ]
        }
    ]
}
```

A questo punto siamo pronti per creare la pipeline usando i comandi dell'AWS CLI. Per prima cosa creiamo un file JSON contenente i parametri per la pipeline:

```
{
    "pipeline": {
        "roleArn": "<CodePipeline Role ARN>",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "roleArn": "<Cross Account Source Role ARN>",
                        "actionTypeId": {
```

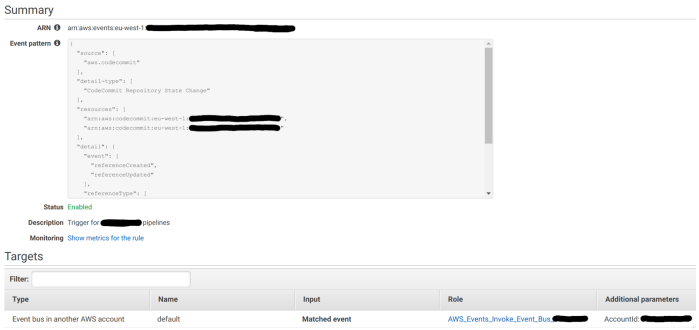
```
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeCommit"
                        },
                        "outputArtifacts": [
                            {
                                "name": "CodeBundle"
                            }
                        ],
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "BranchName": "BranchName",
                            "RepositoryName": "RepositoryName"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "CodeBundle"
                            }
                        ],
                        "name": "CodeBuild",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeBuild"
                        },
                        "outputArtifacts": [
                            {
                                "name": "BuildBundle"
                            }
                        ],
                        "configuration": {
                            "ProjectName": "ProjectName"
                        },
                        "runOrder": 1
                    }
                ]
            }
        ],
        "artifactStore": {
            "type": "S3",
            "location": "<CodePipeline Bucket Name>",
            "encryptionKey": {
                 "id": "<CodePipeline KMS Key ARN>",
                 "type": "KMS"
            }
        },
        "name": "Pipeline Name",
        "version": 1
    }
}
```

Una volta creato il file è possibile usarlo per generare la pipeline tramite la AWS CLI:

```
aws codepipeline create-pipeline --cli-input-json file://pipeline.json
```

Fatto questo la nostra pipeline è finalmente pronta! Manca solo un ultimo tocco: far sì che la pipeline venga triggerata in modo automatico ogni volta che uno sviluppatore esegue il push su CodeCommit.

Per implementare questo sistema di triggering automatico è necessario creare una CloudWatch event rule sull'Account Master, che viene attivata nel momento in cui si esegue un commit; la regola ha come Target l'Event Bus dell'Account Target. In figura è riportato un esempio di configurazione per questa Rule.



Sarà quindi necessario aggiungere un permesso all'event bus dell'account target per far sì che l'Account Master possa scrivere i messaggi nell'account Target. Per fare ciò, è sufficiente aggiungere la permission all'account ID del master dalla dashboard di CloudWatch – Event Buses nell'account target. Infine, sempre nell'Account Target, è necessario creare una seconda CloudWatch Rule, triggerata dallo stesso evento usato per la Rule dell'account master, che avvia la pipeline usando il ruolo CloudWatch Events Role.

Questo ruolo dovrà semplicemente avere una policy che gli consenta di avviare l'esecuzione delle pipeline:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "<Pipeline ARN>"
            ]
        }
    ]
}
```

Una volta aggiunto quest'ultimo componente, la pipeline verrà triggerata in modo automatico ogni volta che uno sviluppatore esegue un commit sul CodeCommit nell'account master.

In questo articolo vi abbiamo presentato un metodo innovativo per creare pipeline cross-account, che semplifica notevolmente la gestione dei deploy per progetti suddivisi su più account AWS.

Se siete interessati a questa soluzione, o avete domande e suggerimenti, non esitate a contattarci, saremo felici di aiutarvi!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it