Automatic Deployments: CodeBuild

Getting GitHub credentials, creating an S3 bucket for build artifacts, and telling CodeBuild to pull the changes from GitHub will be demonstrated in this lesson.

We'll cover the following

Objective

- Automatically update our application when a change gets pushed to GitHub.

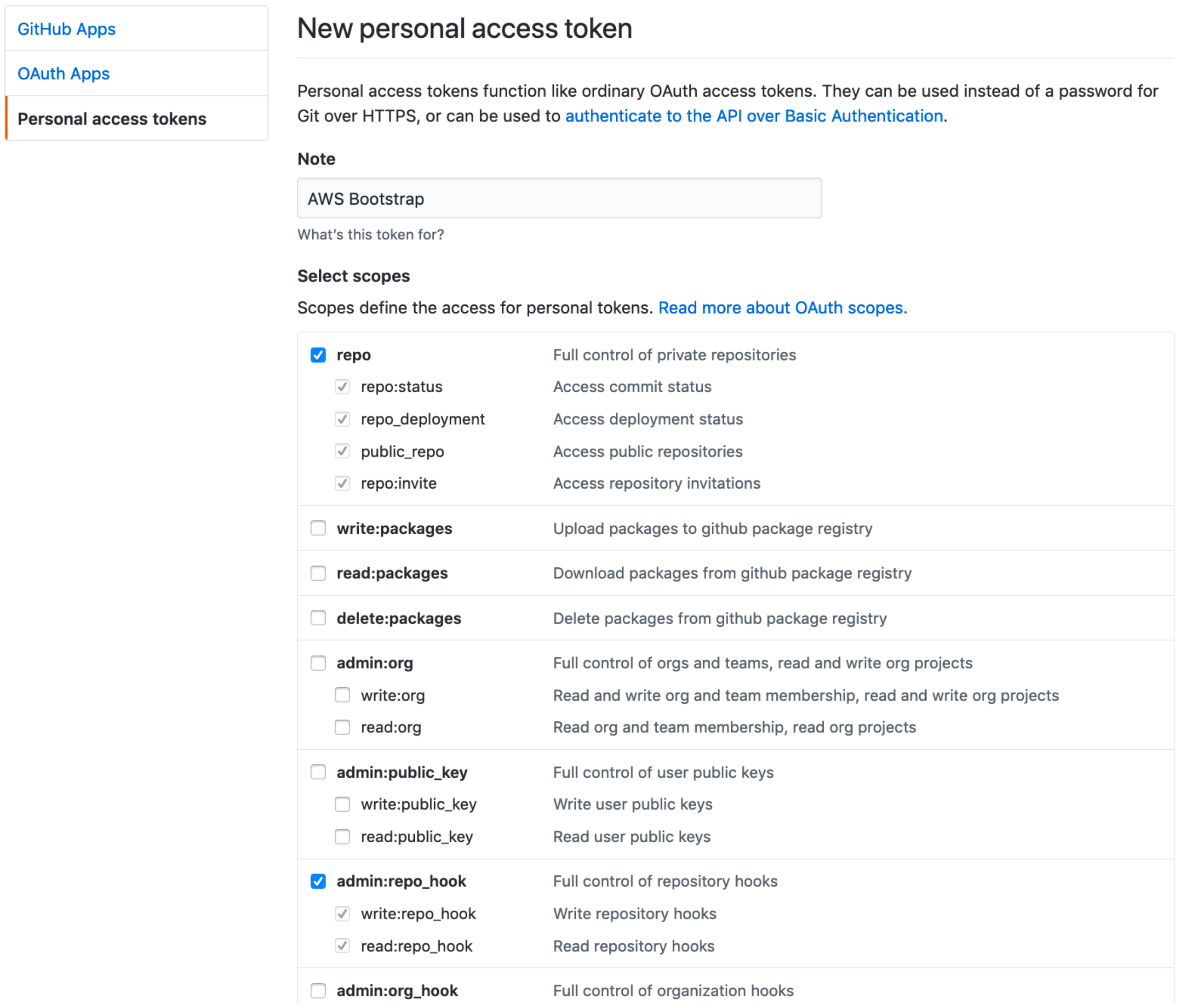
Steps

- Get GitHub credentials.
- Creating S3 bucket for build artifacts.
- CodeBuild to pull changes from GitHub.

In this section, we're going to use CodeBuild, CodeDeploy, and CodePipeline so that our application gets updated automatically as soon as we push a change to GitHub.

GitHub access token

We will need a GitHub access token to let CodeBuild pull changes from GitHub. To generate an access token, go to https://github.com/settings/tokens/new and click *Generate new token*. Give it *repo* and *admin:repo_hook* permissions, and click *Generate token*.



GitHub Apps
OAuth Apps
Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note
AWS Bootstrap
What's this token for?

Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ repo	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
☐ write:packages	Upload packages to github package registry
☐ read:packages	Download packages from github package registry
☐ delete:packages	Delete packages from github package registry
☐ admin:org	Full control of orgs and teams, read and write org projects
☐ write:org	Read and write org and team membership, read and write org projects
☐ read:org	Read org and team membership, read org projects
☐ admin:public_key	Full control of user public keys
☐ write:public_key	Write user public keys
☐ read:public_key	Read user public keys
✓ admin:repo_hook	Full control of repository hooks
✓ write:repo_hook	Write repository hooks
✓ read:repo_hook	Read repository hooks
☐ admin:org_hook	Full control of organization hooks

GitHub Access Token Generation

Tokens and passwords are sensitive information and should not be checked into source repositories. There are sophisticated ways to store them, but for now we'll put our new token in a local file that we can later read into an environment variable.

```
mkdir -p ~/.github
echo "aws-bootstrap" > ~/.github/aws-bootstrap-repo
echo "<username>" > ~/.github/aws-bootstrap-owner
echo "<token>" > ~/.github/aws-bootstrap-access-token
```

terminal

Line #3: Replace `<username>` with your GitHub username.

Line #4: Replace `<token>` with your GitHub access token.

S3 bucket for build artifacts

CodePipeline requires an S3 bucket to store artifacts built by CodeBuild. We chose to create this bucket outside of our main CloudFormation template because CloudFormation is unable to delete S3 buckets unless they're empty. This limitation becomes very inconvenient during development, because you would have to delete the S3 bucket manually every time you tear down your CloudFormation stack. Therefore, we like to put resources such as these in a separate CloudFormation template called `setup.yml`.

```
AWSTemplateFormatVersion: 2010-09-09

Parameters:
  CodePipelineBucket:
    Type: String
    Description: 'The S3 bucket for CodePipeline artifacts.'

Resources:
  CodePipelineS3Bucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    Properties:
      BucketName: !Ref CodePipelineBucket
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256
```

setup.yml

Now let's edit our `deploy-infra.sh` script to define the S3 bucket name for our CodePipeline.

```
AWS_ACCOUNT_ID=`aws sts get-caller-identity --profile awsbootstrap \
  --query "Account" --output text`
CODEPIPELINE_BUCKET="$STACK_NAME-$REGION-codepipeline-$AWS_ACCOUNT_ID"
```

terminal

Line #1: This is a way to programmatically get the AWS account ID from the AWS CLI.

Line #3: S3 bucket names must be globally unique across all AWS customers. Adding our account ID to the bucket name helps prevent name conflicts.

Then we need to deploy `setup.yml` from our `deploy-infra.sh` script, just before we deploy main.yml.

```
# Deploys static resources
echo -e "\n\n=========== Deploying setup.yml ==========="
aws cloudformation deploy \
  --region $REGION \
  --profile $CLI_PROFILE \
  --stack-name $STACK_NAME-setup \
  --template-file setup.yml \
  --no-fail-on-empty-changeset \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides \
    CodePipelineBucket=$CODEPIPELINE_BUCKET
```

deploy-infra.sh

Start and stop scripts

Next, we need to create a couple of simple scripts to tell CodeDeploy how to start and stop our application.

```
#!/bin/bash -xe
source /home/ec2-user/.bash_profile
cd /home/ec2-user/app/release
npm run start
```

start-service.sh

Line #2: Makes sure any user-specific software that we've installed (e.g., `npm` via `nvm`) is available.

Line #3: Changes into the working directory in which our application expects to be run.

Line #4: Runs the start script we put in `package.json`.

```
#!/bin/bash -xe
source /home/ec2-user/.bash_profile
[ -d "/home/ec2-user/app/release" ] && \
cd /home/ec2-user/app/release && \
npm stop
```

stop-service.sh

The build specification

Next, we need to tell CodeBuild how to build our application. To do this, CodeBuild has a specification, which we use in a file named `buildspec.yml`.

```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 10
  pre_build:
    commands:
      # run 'npm install' using versions in package-lock.json
      - npm ci
  build:
    commands:
      - npm run build
artifacts:
  files:
    - start-service.sh
    - stop-service.sh
    - server.js
    - package.json
    - appspec.yml
    - 'node_modules/**/*'
```

buildspec.yml

The deployment specification

Now, we need to tell CodeDeploy what to do with the build artifacts created by CodeBuild. To do this, CodeDeploy also has a specification, which we use in a file named `appspec.yml`.

```yaml
version: 0.0
os: linux
files:
  # unzip the build artifact in ~/app
  - source: /
    destination: /home/ec2-user/app/release
permissions:
  # change permissions from root to ec2-user
  - object: /home/ec2-user/app/release
    pattern: "**"
    owner: ec2-user
    group: ec2-user
hooks:
  ApplicationStart:
    # start the application
    - location: start-service.sh
      timeout: 300
      runas: ec2-user
  ApplicationStop:
    # stop the application
    - location: stop-service.sh
      timeout: 300
      runas: ec2-user
```

appspec.yml

At this point, let’s commit what we have so far to GitHub.

```
git add appspec.yml buildspec.yml start-service.sh stop-service.sh deploy-infra.sh setup.yml
git commit -m "Add codebuild / codedeploy spec"
git push
```

terminal

In the next lesson, we are going to install CodeDeploy agent on our EC2 instance.