[all the] CodeBuild [you can stomach] in 45 minutes

Why?

- CodeBuild felt a bit mysterious and odd to me upon first use
- Not all of the information is easy to find
- It helps to see a few working examples

Who am I?

[Ken Bio]

About Ken Rimple (@techcast)

- Director of Training and Mentoring, Chariot Solutions
- Present about emerging / emergent tech topics
- Mentor clients in AWS, Docker, Angular, React, Spring, other techs
- Host TechChat Tuesdays on YouTube, Chariot TechCast podcast

So, what is AWS CodeBuild?

A cloud-based application builder

- A build runner similar to Jenkins and CircleCI
- Works within AWS and uses the AWS ecosystem
- Can pull code from CodeCommit repositories or GitHub
- Can be used as part of a bigger build pipeline (AWS Code Pipeline)
- CodeBuild terminology
- Running CodeBuild Build Specs Locally
- Deploying CodeBuild Projects with CloudFormation
- Easy things are actually harder things
- Running CodeBuild in the cloud

Terminology

- **`BuildSpec`** - a YAML file that defines the build activities and settings
- **`CodeBuild Project`** - ties the BuildSpec into the AWS environment
- **`Report`** - testing results, analysis, made available after the build
- **`Artifact`** - a built object or objects that can be used by down-stage build tools like Code Pipeline
- **`Build Run`** - An execution of a CodeBuild project with associated logs, artifacts and reports

Projects inject external parameters and define access to networks and resources such as ECS, CloudWatch, S3, and others.

Why use CodeBuild over Jenkins, CircleCI, etc?

- You need access to internal AWS resources
- You need to run tests within a VPC
- You need to deploy resources or manipulate AWS via the AWS CLI
- You want to use CodeCommit, Code PipeLine AWS features for an end-to-end CI suite

Don’t need access to AWS resources or the CLI during a build cycle? CircleCI or Jenkins could be a better selection…

 Copyright 2020, Chariot Solutions

Elements of a CodeBuild Project

CodeBuild Mechanics - The BuildSpec

- CodeBuild Executes a BuildSpec file
- The BuildSpec file can *either*
  - live within the project (easiest to test), *or*
  - within an S3 bucket
- Default filename is **`/buildspec.yml`**

A Simple BuildSpec (YAML)

```
version: 0.2                                    ①

phases:                                         ②
  build:
    commands:
      - ./run-tests.sh                          ③
      - ./run-build.sh
```

① Version **`0.2`** is the latest version and is recommended

② Phases include **`install`**, **`pre_build`**, **`build`**, and **`post_build`**

③ Each command is executed in sequence, and any command returning a non-zero return code causes the build to fail

CodeBuild Mechanics - The Project Source

- A *GIT-based* project is pulled by the CodeBuild engine
  - by tag, commit, or branch
  - It is used as the root of the build script

GitLab, GitHub, CodeCommit are typical SCMs for CodeBuild

CodeBuild Mechanics - The Build Lifecycle

- All builds go through several phases
  - Install

    - Pre-build
    - Build
    - Post-build

CodeBuild Mechanics - The Build Artifact(s) and Reports

- The build, once successful, can be published as an *artifact*
- Reports can be generated after a successful *or* failed build
    - Reports can be in several formats, with JUNITXML being the most common

CodeBuild Mechanics - Notifications

- CodeBuild can use SES to notify about build lifecycle events
- CodeBuild can also report build success/failure to your SCM
- These topics are not covered here

 Copyright 2020, Chariot Solutions

Running a BuildSpec

Run BuildSpecs locally using a Docker image

- One time setup:
  - Download the CodeBuild script and selected CodeBuild Image to your local machine
  - Uses **`codebuild_build.sh`** from https://tinyurl.com/yxosr3uo
  - Follow instructions for building the AWS build image from https://tinyurl.com/y5qz7txs

This lets you diagnose CodeBuild problems locally!

Scripting a codebuild run

```
/bin/bash codebuild_build.sh \
  -i aws/codebuild/amazon-linux-2-v-3:latest \            ①
  -a /tmp/cb \                                            ②
  -s `pwd` -c -m                                        ③ ④ ⑤
```

① -i Docker image you built / tagged (AL 2, v3 is suggested)

② -a Location for artifact downloads

③ -s Source files location (current dir here)

④ -c Use local credentials in **`~/.aws/credentials`**

⑤ -m Mount the source directory directly in Docker

Assumes a **`buildspec.yml`** in the root directory of the project

Demo: Running CodeBuild on local machine

Defining a Build Project

- Two ways
  - The AWS Console (one-off build projects)
  - CloudFormation / CDK / etc. (configuration-as-code)

Always use CloudFormation to configure projects so you can tear down and set up again easily

Run BuildSpecs in the CodeBuild console

- Install the BuildSpec as as part of a CodeBuild project
  - Configure the source repository
  - Configure the IAM Role
  - Configure the BuildSpec from S3 or a project file/directory
  - Configure other aspects (report/artifact locations, alerts, triggers, etc)

You can configure CodeBuild as a CloudFormation stack. Examples on my Git repo.

Demo: Running a CodeBuild project from the AWS Console

CloudFormation configuration example

 Copyright 2020, Chariot Solutions

Some BuildSpec Tips

Any non-zero command result is going to fail your build!

- CodeBuild expects your statements to execute with a zero return code!
- Any command (even shell commands) that return a non-zero fail the build
- This can be *hard* to debug
  - Check **`$?`** with an echo to see the return of the last command

Even the lowly `ls` command can fail you

```
version: 0.2

phases:
  build:
    commands:
      - ls dirthatdoesntexist
```

The return value of **`ls`** when not finding a dir is > 0

Organizing your builds

- Modularize your builds into *separate BuildSpecs* by Git *project* or *module*
- Create *shell scripts* to run build guts to avoid *huge* BuildSpecs
  - Return **`0`** or *meaningful error values* and output from the shell scripts
  - You can also use these for *local* developers
- Orchestrate with other tools like *CodePipeline*

Smaller, more modular builds are easier to orchestrate

Configuring your language/runtime

- Use the **`install`** phase, **`runtime-versions`** section of the BuildSpec
  - Makes sure a language is available for use in the selected container
  - Installs the necessary version of the language

- Choose from supported languages

See https://tinyurl.com/y3ufegga for details…

Setting Java, NodeJS versions in INSTALL phase

```
phases:
  install:
    runtime-versions:
      java: corretto11                    ①
      nodejs: 12
  build:
    commands:
      - npm install                       ②
      - gradle test                       ③
```

① Corretto is the available JVM from Amazon. It does **not** support Java 14

② NodeJS includes **`npm`** and related tools

③ Java includes Maven and Gradle out of the box

Pre and post build steps

- Add a **`pre_build`** section to run before the build
  - Triggering AWS CLI commands to create or reset resources
  - Downloading external images or support files
- Add a **`post_build`** section to run after the build completes
  - Generate documentation from source code
  - Collect items to deliver as an archive with **`tar`**, **`gz`**
- Add a **`finally`** section to run regardless of pass/fail (for test reports)

Example `finally`: creating reports for Cypress

 Copyright 2020, Chariot Solutions

```
build:
  commands:
    - cd cypress-tests
    - npm install
    - NO_COLOR=1 cypress --chromium                  ①
  finally:
    - npm run merge-report; npm run generate-report

artifacts:
  files:
    - '**/*'
  base-directory:
    - 'cypress-tests/cypress'
```

① Important - Cypress outputs ANSI escape codes that make a mess of logs

Install software in the Build Image

- You can install utilities with **`wget`**, **`tar`**, **`gunzip`** etc...
- No permissions required if you are running it in userspace
- You can run as **`root`** if needed to install packages via the package manager of the OS

This slows down build times. It may be better to create and use your own custom build image

Installing Software Example: pandoc

```
...
phases:
  pre_build:
    commands:
      - yum update -y
      - yum install -y pandoc
  build:
    commands:
      - pandoc src/content/simpledoc.markdown -o src/content/simpledoc.html
artifacts:
  files:
    - src/content/simpledoc.html
```

- To do this you must run as "root" in the container

Don’t pick huge utilities, customize the image instead

Running Docker in CodeBuild

- Run builds with **`docker build`** and publish to AWS ECR

```
phases:
  install:
    runtime-versions:
      docker: 19
  pre-build:
    commands:
      - $(aws ecr get-login --no-include-email --region us-east-1)
  build:
    commands:
      - docker build -t foo/bar build .
      - docker tag ...
      - docker push ...
```

You must configure the job to run as **`root`**

 Copyright 2020, Chariot Solutions

Wrap-up

- CodeBuild is a huge tool, with lots of settings
  - It is AWS-native
  - It can access your applications and run behind a VPC
  - It can run customized containers
- You can test the Build Specs locally
- You can install CodeBuild projects via CloudFormation
- You can use CodeBuild projects as part of a full build pipeline with CodePipeline or do the uploads yourself in CodeBuild

Questions?

 Copyright 2020, Chariot Solutions