beedo documentation

Release 0.9.12

beedo

Mar 29, 2017

Contents

1 Why another CI? 1
- 1.1 cheaper than saas or dedicated ci server . . . 1
- 1.2 development environment evolved . . . 1
- 1.3 declarative vs. build scripts . . . 1
- 1.4 pipeline-as-code makes this more obvious . . . 1
- 1.5 simple . . . 1
- 1.6 design decisions . . . 1

2 Serverless continuous integration 3
- 2.1 Contents . . . 3
- 2.2 What is it? . . . 4
- 2.3 Supported languages . . . 4
- 2.4 Prerequisites . . . 4
- 2.5 Current Limitations (due to the Lambda environment itself) . . . 4
- 2.6 Installation . . . 5
- 2.7 Configuration . . . 9
- 2.8 Updating . . . 14
- 2.9 Security . . . 14
- 2.10 Language Recipes . . . 15
- 2.11 Extending with ECS . . . 17
- 2.12 Questions . . . 17
- 2.13 License . . . 19

3 Build on docker 21

4 Logging 23
- 4.1 testing the logging config . . . 23
- 4.2 fileConfig sample: . . . 24
- 4.3 dictConfig . . . 24

5 Release history 27
- 5.1 0.9.11 (2017-02-11) . . . 27
- 5.2 0.9.10 (2016-11-23) . . . 27
- 5.3 0.9.8 (2016-10-02) . . . 27
- 5.4 TODO . . . 27

6 Development 29

6.1 locations for beedo-ci artefacts: . . . 29
6.2 url used to launch stack . . . 29
6.3 run virtualenv . . . 29
6.4 alternative install virtualenv during bundle and use that! . . . 30
6.5 working / troubleshooting on a amazon ami . . . 30
6.6 amazonlinux in docker container . . . 30
6.7 release beedo-ci . . . 31

CHAPTER 1

Why another CI?

cheaper than saas or dedicated ci server

development environment evolved

- AWS services provide a lot of infrastructure that in the past was part of the CI server
- Github provides much of the GUI part that was covered by CI server in the past

declarative vs. build scripts

I used many build systems over time like make, ant, etc. and I always found that the declarative way is making it more complicated for me. I am a programmer and I am perfectly capable to write a little build script in Python or bash.

pipeline-as-code makes this more obvious

need the build script as script, must be able to run locally (outside of the ci server).

simple

design decisions

- Use Github and AWS
- python or bash build scripts (probably should be configurable and support npm, grunt and friends as well)
- build script handles artifact creation and HTML reports

- build config file configures whats is happening with the artifact and reports + notifications
- all build information is exchanged via environment variables

CHAPTER 2

Serverless continuous integration

Automate your testing and deployments with:

- 100 concurrent builds out of the box (can request more)
- No maintenance of web servers, build servers or databases
- Zero cost when not in use (ie, 100% utilization)
- Easy to integrate with the rest of your AWS resources

Contents

- *Overview*
- *Installation*
- *Configuration*
- *Updating*
- *Security*
- *Language Recipes*
- *Extending with ECS*
- *Questions*

What is it?

Beedo CI is a package you can upload to AWS Lambda that gets triggered when you push new code or open pull requests on GitHub and runs your tests (in the Lambda environment itself) – in the same vein as Jenkins, Travis or CircleCI.

It integrates with Slack, and updates your Pull Request and other commit statuses on GitHub to let you know if you can merge safely.

It can be easily launched and kept up-to-date as a CloudFormation Stack, or you can manually create the different resources yourself.

(Support for running under Google Cloud Functions may be added in the near future, depending on the API they settle on)

Supported languages

- Node.js (multiple versions via nave)
- Python 2.7
- Java (OpenJDK *1.8 and 1.7*)
- Go (*any version*)
- Ruby (*2.3.1, 2.2.5, 2.1.10, 2.0.0-p648*)
- PHP (*7.0.10, 5.6.25*)
- Native compilation with a *pre-built gcc 4.8.5*
- Rust (*1.11.0, 1.10.0*, but any version should work)
- Check the *Recipes* list below for the status of other languages/tools

Prerequisites

- An Amazon AWS account
- A GitHub OAuth token (*see below*)
- (optional) A Slack API token (*see below*)

Current Limitations (due to the Lambda environment itself)

- No root access
- 5 min max build time
- Bring-your-own-binaries – Lambda has a limited selection of installed software
- 1.5GB max memory
- Linux only

You can get around many of these limitations by *configuring Beedo CI to send tasks to an ECS cluster* where you can run your builds in Docker.

Installation

The easiest way to install Beedo CI is to [spin up a CloudFormation stack using beedo.template – this is just a collection of related AWS resources, including the main Beedo CI Lambda function and DynamoDB tables, that you can update or remove together – it should take around 3-4 minutes to spin up.

You can run multiple stacks with different names side-by-side too (eg, `beedo-private` and `beedo-public`).

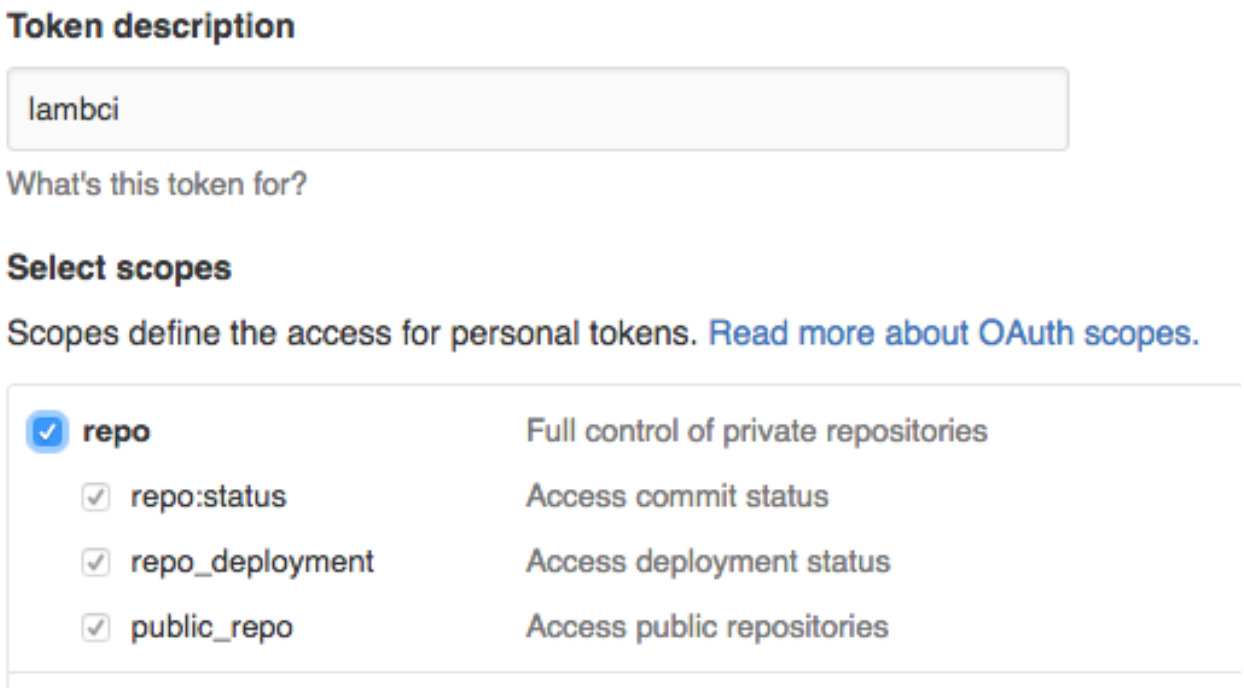
As part of the stack setup, you can supply your GitHub and Slack API tokens, as well as a list of repositories you want to trigger Beedo CI, but you don't have to – you can add these later, either by *updating the CloudFormation stack*, or using the AWS DynamoDB console or beedo command line. If you'd prefer to do that, you can skip straight to Step 3.

1. Create a GitHub token

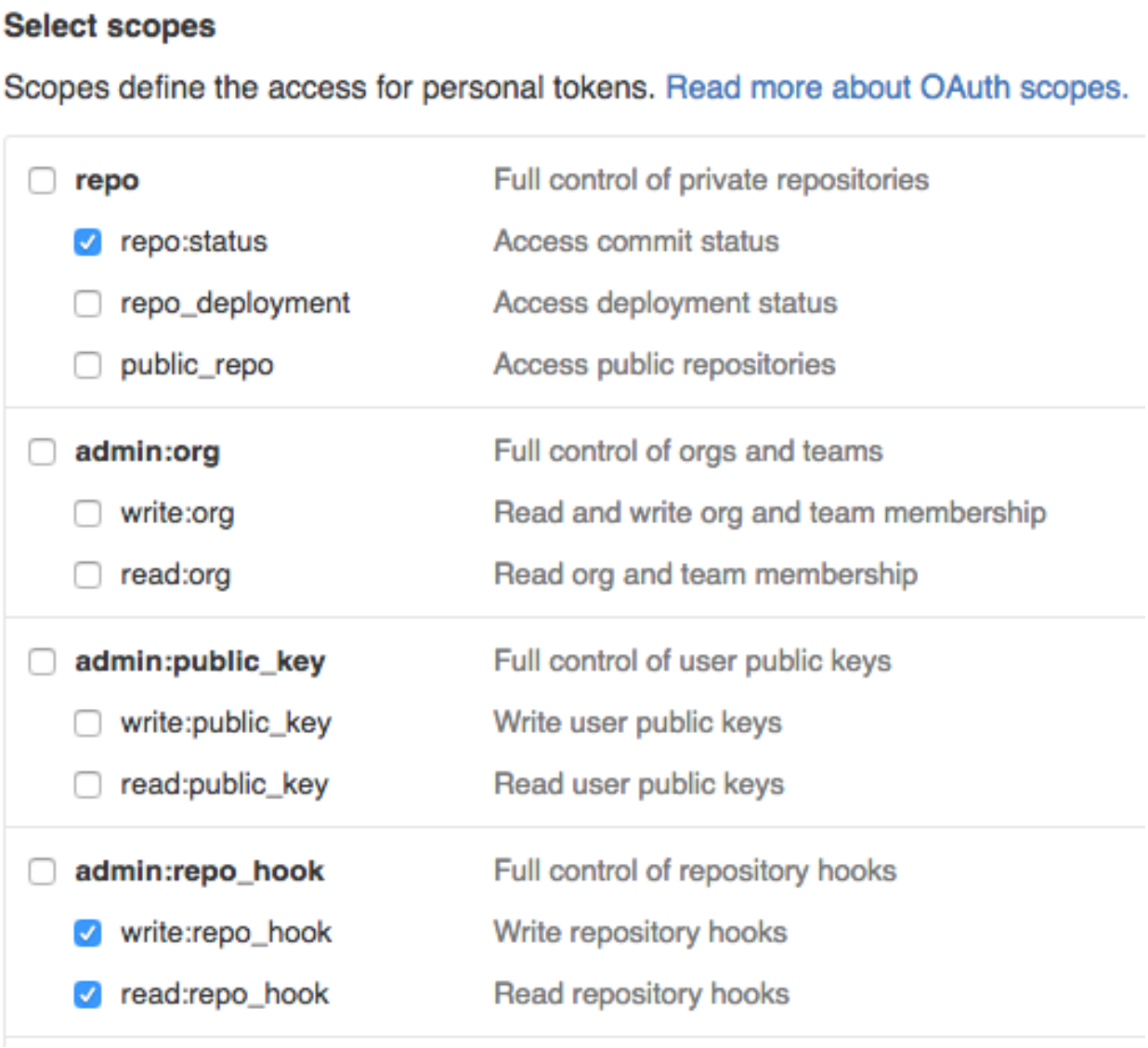
You can create a token in the Personal access tokens section of your GitHub settings. If you're setting up Beedo CI for an organization, it might be a good idea to create a separate GitHub user dedicated to running automated builds (GitHub calls these "machine users") – that way you have more control over which repositories this user has access to.

Click the Generate new token button and then select the appropriate access levels.

Beedo CI only needs read access to your code, but unfortunately GitHub has rather crude access mechanisms and doesn't have a readonly scope for private repositories – the only options is to choose `repo` ("Full control"). Other CI systems have the same frustrations.



If you're only using Beedo CI for public repositories, then you just need access to commit statuses and repository hooks (even the latter you can do away with if you're adding/removing the hooks manually):

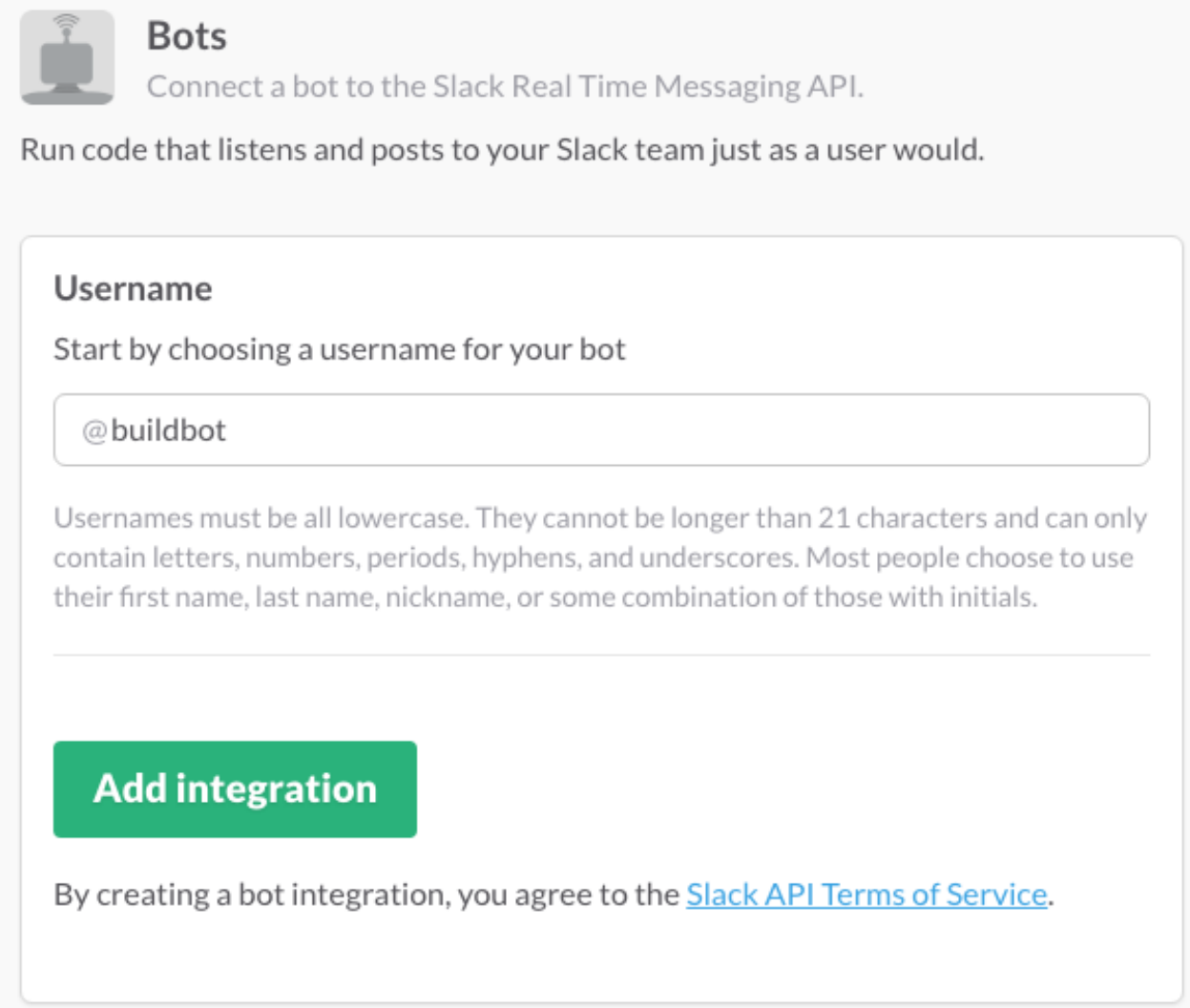


Then click the "Generate token" button and GitHub will generate a 40 character hex API token.

2. Create a Slack token (optional)

You can obtain a Slack API token by creating a bot user (or you can use the token from an existing bot user if you have one) – this direct link should take you there, but you can navigate from the App Directory via `Browse Apps > Custom Integrations > Bots`.

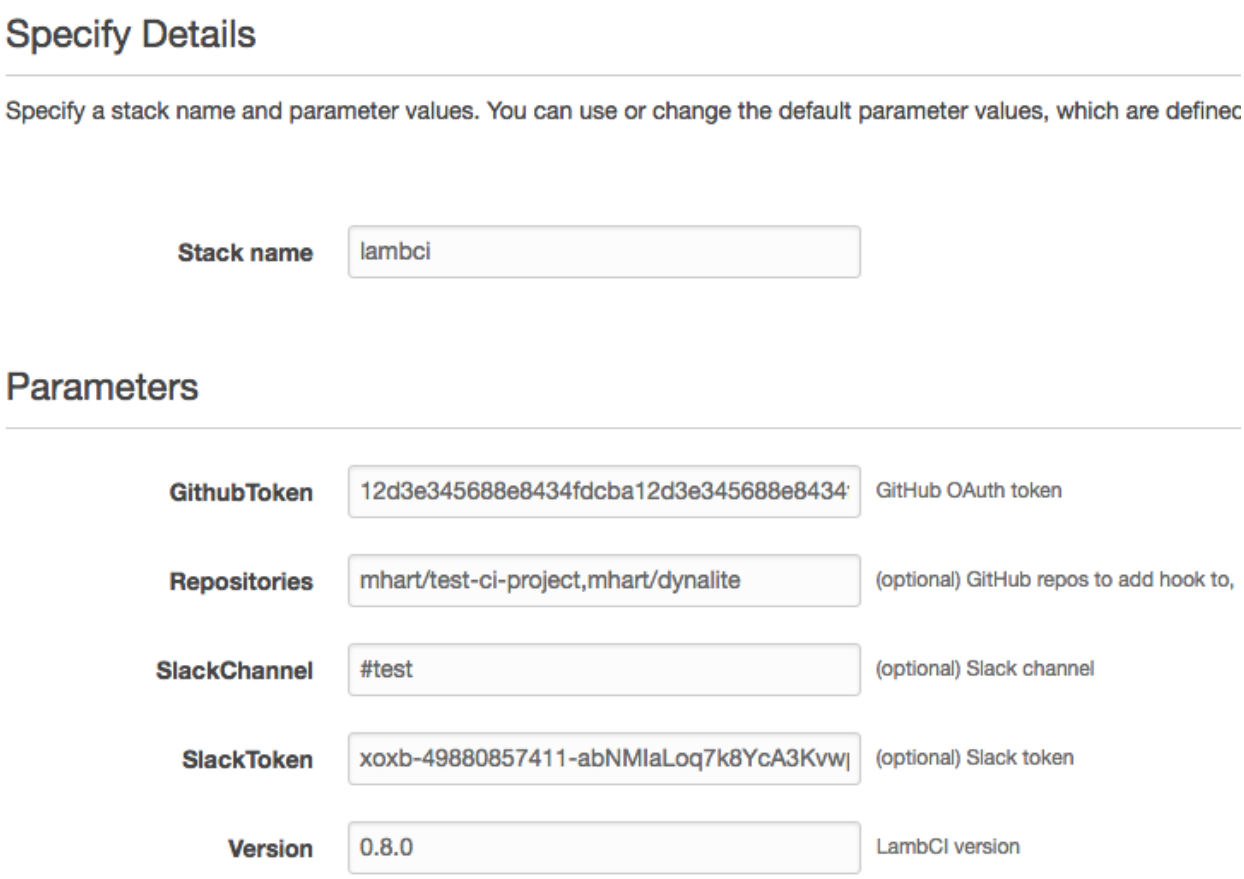
Pick any name, and when you click "Add integration" Slack will generate an API token that looks something like `xoxb-<numbers>-<letters>`



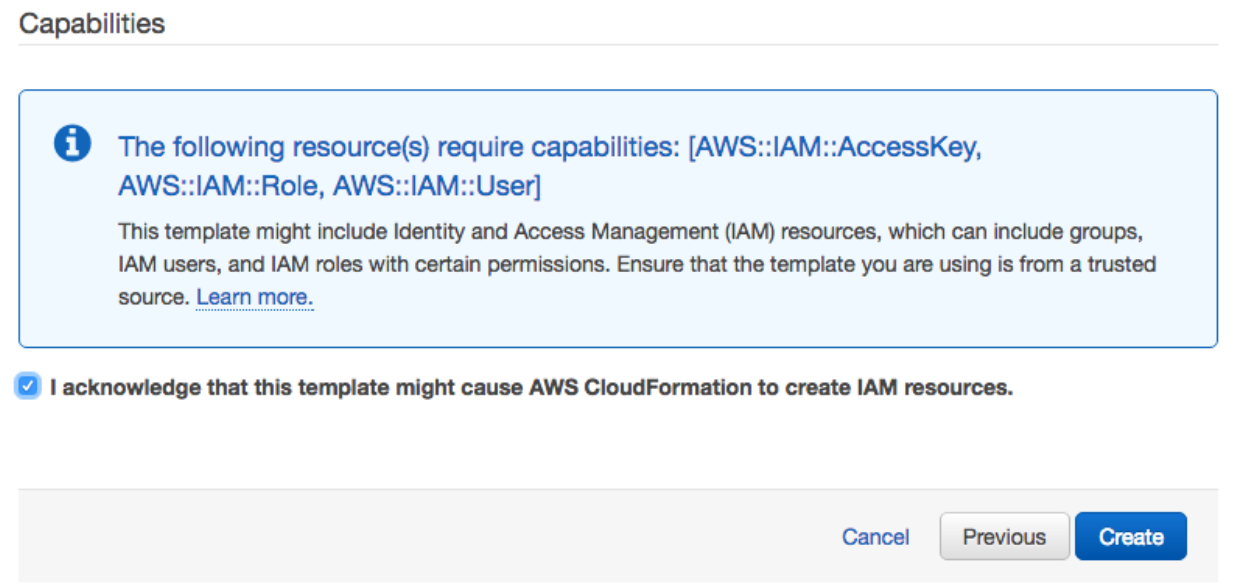
3. Launch the Beedo CI CloudFormation stack

You can either [use this direct link] or navigate in your AWS Console to `Services > CloudFormation`, choose "Create Stack" and upload `beedo.template` from the root of this repository, or use the S3 link:

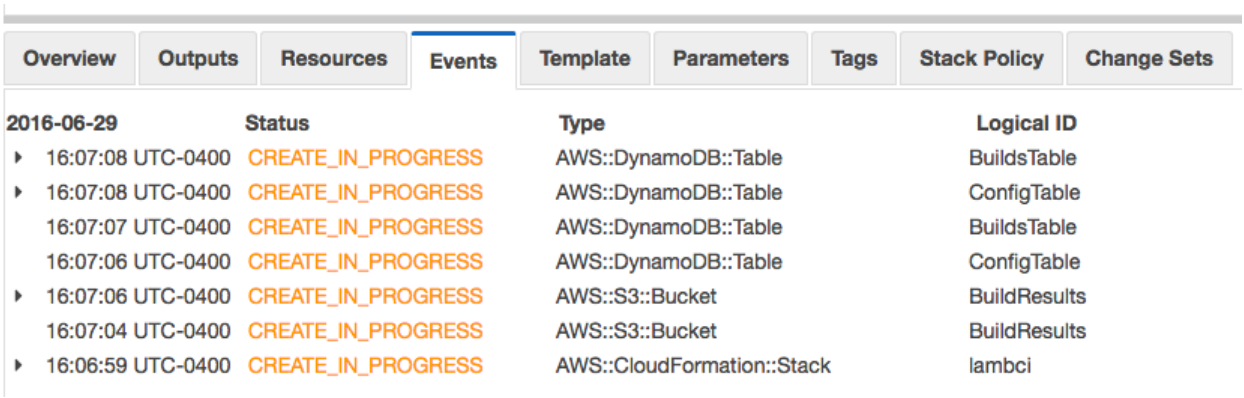
Then click Next where you can enter a stack name (`beedo` is a good default), API tokens, Slack channel and a comma-separated list of any repositories you want to add hooks to:



Click Next, and then Next again on the Options step (leaving the default options selected), to get to the final Review step:



Check the acknowledgment checkbox and click Create to start the resource creation process:



Overview | Outputs | Resources | Events | Template | Parameters | Tags | Stack Policy | Change Sets

2016-06-29	Status	Type	Logical ID
16:07:08 UTC-0400	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	BuildsTable
16:07:08 UTC-0400	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	ConfigTable
16:07:07 UTC-0400	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	BuildsTable
16:07:06 UTC-0400	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	ConfigTable
16:07:06 UTC-0400	CREATE_IN_PROGRESS	AWS::S3::Bucket	BuildResults
16:07:04 UTC-0400	CREATE_IN_PROGRESS	AWS::S3::Bucket	BuildResults
16:06:59 UTC-0400	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	lambci

Once your stack is created (should be done in a few minutes) you're ready to start building!

By default Beedo CI only responds to pushes on the master branch and pull requests (*you can configure this*), so try either of those – if nothing happens, then check `Services > CloudWatch > Logs` in the AWS Console and see the *Questions* section below.

You can check that the hooks have been installed in a repository correctly by going to `Settings > Webhooks and services` on the GitHub repository page (ie, `https://github.com/<user>/<repo>/settings/hooks`). There should be a Service listed as `Amazon SNS` – if you click the edit (pencil) button then you can choose to "Test Service" (it should send a push event).

Configuration

Many configuration values can be specified in a `.beedo.js`, `.beedo.json` or `package.json` file in the root of your repository – and all values can be set in the DynamoDB configuration table (named `<stack>-config`, eg, `beedo-config`)

For example, the default command that Beedo CI will try to run is `npm install && npm test`, but let's say you have a python project – you could put the following in `.beedo.json` in your repository root:

```
{
  "cmd": "pip install --user tox && tox"
}
```

(Beedo CI bundles `pip` and adds `$HOME/.local/bin` to `PATH`)

If you have a more complicated build setup, then you could specify `make` or create a bash script in your repository root:

```
{
  "cmd": "./beedo-test.sh"
}
```

Overriding default properties

Beedo CI resolves configuration by overriding properties in a cascading manner in the following order:

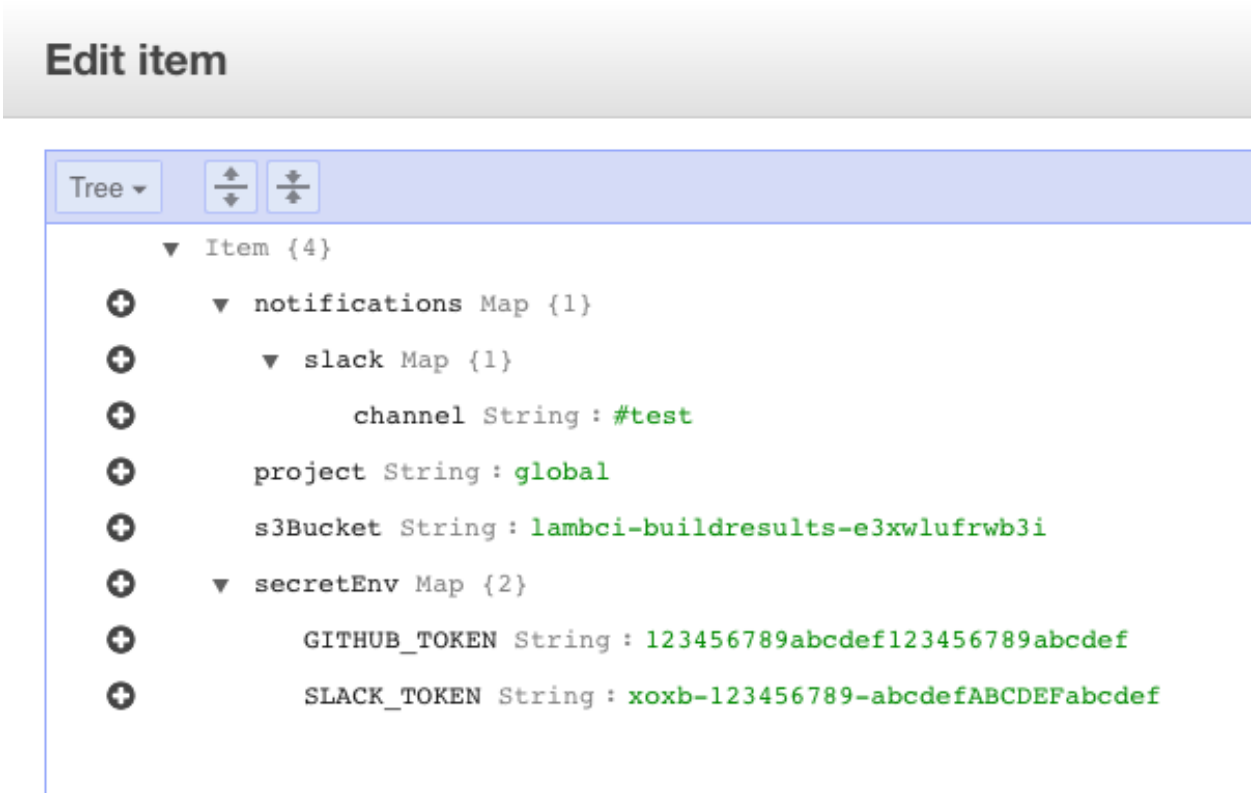
1. Default config (*see below*)
2. `global` project key in `beedo-config` DynamoDB table
3. `gh/<user>/<repo>` project key in `beedo-config` DynamoDB table

4. `beedo` property in `package.json` file in repository root
5. `.beedo.js` or `.beedo.json` file in repository root

You can use the command line to edit the DynamoDB config values:

```
beedo config secretEnv.GITHUB_TOKEN abcdef01234
beedo config --project gh/mhart/kinesalite secretEnv.SLACK_TOKEN abcdef01234
```

Or the AWS console:



So if you wanted to use a different Slack token and channel for a particular project, you could create an item in the config table with the project key `gh/<user>/<repo>` that looks similar to the global config above, but with different values:

```
{
  project: 'gh/mhart/kinesalite',
  secretEnv: {
    SLACK_TOKEN: 'xoxb-1234243432-vnjcnioeiurn'
  },
  notifications: {
    slack: {
      channel: '#someotherchannel'
    }
  }
}
```

Using the command line:

```
beedo config --project gh/mhart/kinesalite secretEnv.SLACK_TOKEN xoxb-1234243432-
↪vnjcnioeiurn
beedo config --project gh/mhart/kinesalite notifications.slack.channel '
↪#someotherchannel'
```

Config file overrides

Here's an example `package.json` overriding the `cmd` property:

```
{
  "name": "some-project",
  "scripts": {
    "beedo-build": "eslint . && mocha"
  },
  "beedo": {
    "cmd": "npm install && npm run beedo-build"
  }
}
```

And the same example using `.beedo.js`:

```
module.exports = {
  cmd: 'npm install && npm run beedo-build'
}
```

The ability to override config properties using repository files depends on the `allowConfigOverrides` property (*see the default config below*).

Branch and pull request properties

Depending on whether Beedo CI is building a branch from a push or a pull request, config properties can also be specified to override in these cases.

For example, to determine whether a build should even take place, Beedo CI looks at the top-level `build` property of the configuration. By default this is actually `false`, but if the branch is `master`, then Beedo CI checks for a `branches.master` property and if it's set, uses that instead:

```
{
  build: false,
  branches: {
    master: true
  }
}
```

If a branch just has a `true` value, this is the equivalent of `{build: true}`, so you can override other properties too – ie, the above snippet is just shorthand for:

```
{
  build: false,
  branches: {
    master: {
      build: true
    }
  }
}
```

So if you wanted Slack notifications to go to a different channel to the default for the `develop` branch, you could specify:

```
{
  branches: {
    master: true,
    develop: {
      build: true,
      notifications: {
        slack: {
          channel: '#dev'
        }
      }
    }
  }
}
```

You can also use regular expression syntax to specify config for branches that match, or don't match (if there is a leading `!`). Exact branch names are checked first, then the first matching regex (or negative regex) will be used:

```
// 1. Don't build gh-pages branch
// 2. Don't build branches starting with 'dev'
// 3. Build any branch that doesn't start with 'test-'
{
  build: false,
  branches: {
    '/^dev/': false,
    '!/^test-/': true,
    'gh-pages': false,
  }
}
```

Default configuration

This configuration is hardcoded in `utils/config.py` and overridden by any config from the DB (and config files)

```
{
  cmd: 'npm install && npm test',
  env: { // env values exposed to build commands
  },
  secretEnv: { // secret env values, exposure depends on inheritSecrets config below
    GITHUB_TOKEN: '',
    SLACK_TOKEN: '',
  },
  s3Bucket: '', // bucket to store build artifacts
  notifications: {
    slack: {
      channel: '#general',
      username: 'Beedo CI',
      iconUrl: 'https://beedo-ci.s3.amazonaws.com_static/images/logo-48x48.png',
      asUser: false,
    },
  },
  build: false, // Build nothing by default except master and PRs
  branches: {
    master: true,
  },
  pullRequests: {
    fromSelfPublicRepo: true, // Pull requests from same (private) repo will build
```

```
    fromSelfPrivateRepo: true, // Pull requests from same (public) repo will build
    fromForkPublicRepo: { // Restrictions for pull requests from forks on public repos
      build: true,
      inheritSecrets: false, // Don't expose secretEnv values in the build command␣
↪environment
      allowConfigOverrides: ['cmd', 'env'], // Only allow file config to override cmd␣
↪and env properties
    },
    fromForkPrivateRepo: false, // Pull requests from forked private repos won't run␣
↪at all
  },
  s3PublicSecretNames: true, // Use obscured names for build HTML files and make them␣
↪public
  inheritSecrets: true, // Expose secretEnv values in the build command environment␣
↪by default
  allowConfigOverrides: true, // Allow files to override config values
  clearTmp: true, // Delete /tmp each time for safety
  git: {
    depth: 5, // --depth parameter for git clone
  },
}
```

SNS Notifications (for email, SMS, etc)

By default, the CloudFormation template doesn't create an SNS topic to publish build statuses (ie, success, failure) to – but if you want to receive build notifications via email or SMS, or some other custom SNS subscriber, you can specify an SNS topic and Beedo CI will push notifications to it:

```
notifications: {
  sns: {
    topicArn: 'arn:aws:sns:us-east-1:1234:beedo-StatusTopic-1WF8BT36'
  }
}
```

The Lambda function needs to have permissions to publish to this topic, which you can either add manually, or by modifying the CloudFormation `beedo.template` and updating your stack.

Add a top-level SNS topic resource:

```
"StatusTopic" : {
  "Type": "AWS::SNS::Topic",
  "Properties": {
    "DisplayName": "Beedo CI"
  }
}
```

And then add the following to the `LambdaExecution.Properties.Policies` array to give the Lambda function the correct permissions:

```
{
  "PolicyName": "PublishSNS",
  "PolicyDocument": {
    "Statement": {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
```

```
      ],
      "Resource": {"Ref": "StatusTopic"}
    }
  }
}
```

Build status badges

Each branch has a build status image showing whether the last build was successful or not. For example, here is Beedo CI's latest `master` status (yes, Beedo CI dogfoods!):

You can see the URLs for the branch log and badge image near the start of the output of your build logs (so you'll need to run at least one build on your branch to get these):

```
Branch log: https://<bucket>/<project>/branches/master/<somehash>.html
Branch status img: https://<bucket>/<project>/branches/master/<somehash>.svg
```

Updating

You can update your CloudFormation stack at any time to change, add or remove the parameters – or even upgrade to a new version of Beedo CI.

In the AWS Console, go to `Services > CloudFormation`, select your Beedo CI stack in the list and then choose `Actions > Update Stack`. You can keep the same template selected (unless you're updating Beedo CI and the template has different resources), and then when you click Next you can modify parameters like your GitHub token, repositories, Slack channel, Beedo CI version, etc.

Beedo CI will do its best to update these parameters correctly, but if it fails or you run into trouble, just try setting them all to blank, updating, and then update again with the values you want.

Security

The default configuration passes secret environment variables to build commands, except when building forked repositories. This allows you to use your AWS credentials and Git/Slack tokens in your build commands to communicate with the rest of your stack. Set `inheritSecrets` to false to prevent this.

HTML build logs are generated with random filenames, but are accessible to anyone who has the link. Set `s3PublicSecretNames` to false to make build logs completely private (you'll need to use the AWS console to access them), or you can remove `s3Bucket` entirely – you can still see the build logs in the Lambda function output in CloudWatch Logs.

By default, the `/tmp` directory is removed each time – this is to prevent secrets from being leaked if your Beedo CI stack is building both private and public repositories. However, if you're only building private (trusted) repositories, then you can set the `clearTmp` config to false, and potentially cache files (eg, in `$HOME`) for use across builds (this is not guaranteed – it depends on whether the Lambda environment is kept "warm").

If you discover any security issues with Beedo CI please email security@beedo.org.

Language Recipes

Beedo CI doesn't currently have any language-specific settings. The default command is `npm install && npm test` which will use the default Lambda version of Node.js (4.3.x) and npm (2.x).

The way to build with different Node.js versions, or other languages entirely, is just to override the `cmd` config property (specifying a `test` property in a `package.json` file would work too).

Beedo CI comes with a collection of helper scripts to setup your environment for languages not supported out of the box on AWS Lambda – that is, every language except Node.js and Python 2.7

Node.js

Beedo CI comes with nave installed and available on the `PATH`, so if you wanted to run your npm install and tests using the latest Node.js v6.x and npm v3.x, you could do specify:

```
{
  "cmd": "nave use 6 bash -c 'npm install && npm test'"
}
```

If you're happy using the built-in npm to install, you could simplify this a little:

```
{
  "cmd": "npm install && nave use 6 npm test"
}
```

There's currently no way to run multiple builds in parallel but you could have processes run in parallel using a tool like npm-run-all – the logs will be a little messy though!

Here's an example package.json for running your tests in Node.js v4, v5 and v6 simultaneously:

```
{
  "beedo": {
    "cmd": "npm install && npm run ci"
  },
  "scripts": {
    "ci": "run-p ci:*",
    "ci:node4": "nave use 4 npm test",
    "ci:node5": "nave use 5 npm test",
    "ci:node6": "nave use 6 npm test"
  },
  "devDependencies": {
    "npm-run-all": "*"
  }
}
```

Python 2.7

Beedo CI comes with pip installed and available on the `PATH`, and Lambda has Python 2.7 already installed. `$HOME/.local/bin` is also added to `PATH`, so local pip installs should work:

```
{
  "cmd": "pip install --user tox && tox"
}
```

Java

The Java SDK is not installed on AWS Lambda, so needs to be downloaded as part of your build – but the JRE *does* exist on Lambda, so the overall impact is small.

Beedo CI includes a script you can source before running your build commands that will install and setup the SDK correctly, as well as Maven (v3.3.9). Call it with the OpenJDK version you want (`1.7` or `1.8`) – omitting it defaults to `1.8`:

```
{
  "cmd": ". ~/init/java 1.7 && mvn install -B -V && mvn test"
}
```

You can see an example of this working

Go

Go is not installed on AWS Lambda, so needs to be downloaded as part of your build, but Go is quite small and well suited to running anywhere.

Beedo CI includes a script you can source before running your build commands that will install Go and set your `GOROOT` and `GOPATH` with the correct directory structure. Call it with the Go version you want (any of the versions on the Go site) – omitting it defaults to `1.7`:

```
{
  "cmd": ". ~/init/go 1.6.3 && make test"
}
```

You can see examples of this working

Ruby

Ruby is not installed on AWS Lambda, so needs to be downloaded as part of your build.

Beedo CI includes a script you can source before running your build commands that will install Ruby, rbenv, gem and bundler. Call it with the Ruby version you want (currently: `2.3.1`, `2.2.5`, `2.1.10` and `2.0.0-p648`) – omitting it defaults to `2.3.1`:

```
{
  "cmd": ". ~/init/ruby 2.2.5 && bundle install && bundle exec rake"
}
```

You can see an example of this working

PHP

PHP is not installed on AWS Lambda, so needs to be downloaded as part of your build.

Beedo CI includes a script you can source before running your build commands that will install PHP, phpenv and composer. Call it with the PHP version you want (currently: `7.0.10` and `5.6.25`) – omitting it defaults to `7.0.10`:

```
{
  "cmd": ". ~/init/php 5.6.25 && composer install -n --prefer-dist && vendor/bin/
↪phpunit"
}
```

These versions are compiled using php-build with the default config options and overrides of `--disable-cgi` and `--disable-fpm`.

You can see an example of this working

Native (gcc) compilation

AWS Lambda also has no native compiler, so you need to download one as part of your build process. We have a precompiled gcc 4.8.5 that works in the Lambda environment with a full set of linux headers:

```
{
  "cmd": ". ~/init/gcc && npm install && npm test"
}
```

Keep in mind that native compilation is finicky at best, especially when installed in a non-default location, so it may not work out-of-the-box for complicated libraries that depend on other headers/libraries.

You can see examples of this working

Rust

Rust is not installed on AWS Lambda, so needs to be downloaded as part of your build.

Beedo CI includes a script you can source before running your build commands that will install Rust, cargo and gcc. Call it with the Rust version you want (currently: `1.11.0` and `1.10.0`) – omitting it defaults to `1.11.0`:

```
{
  "cmd": ". ~/init/rust 1.10.0 && cargo build && cargo test"
}
```

You can see an example of this working

Extending with ECS

Beedo CI can run tasks on an ECS cluster, which means you can perform all of your build tasks in a Docker container and not be subject to the same restrictions you have in the Lambda environment.

This needs to be documented further – for now you'll have to go off the source and check out the beedo/ecs repo.

Questions

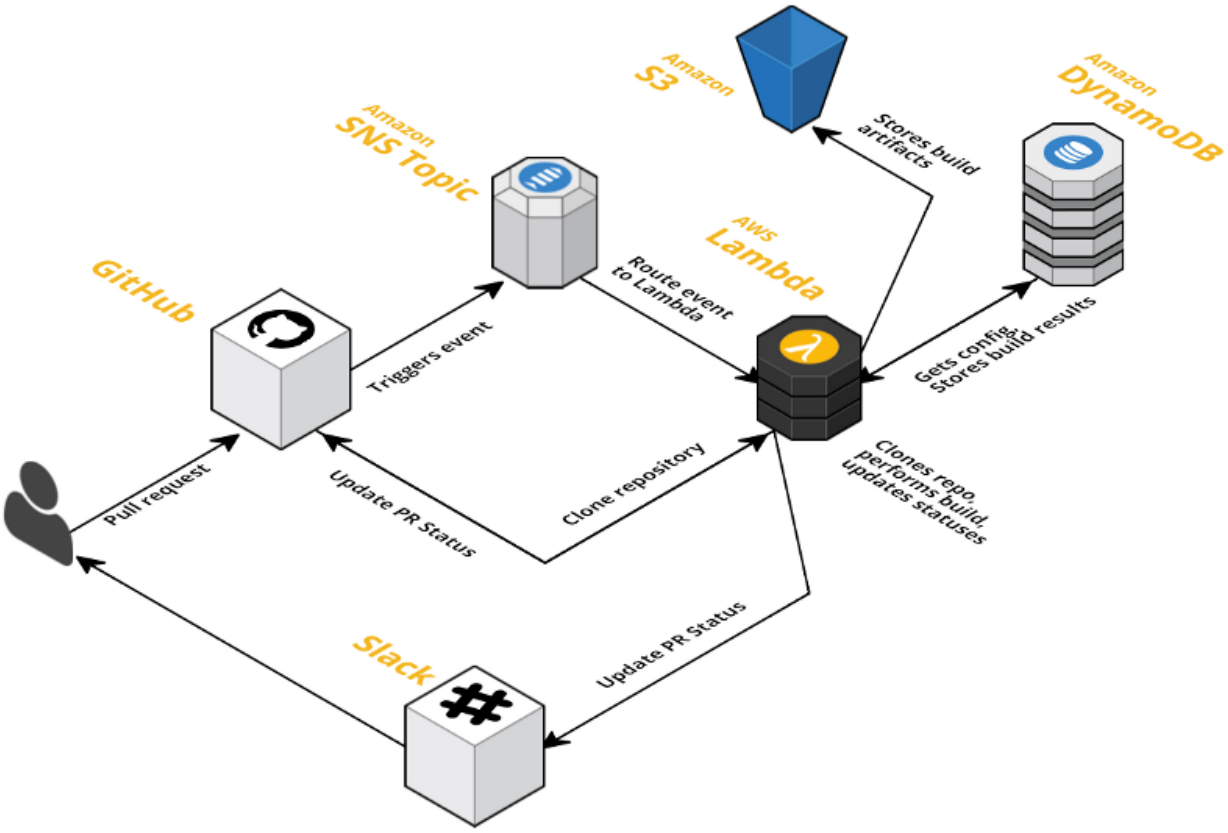
What does the Lambda function do?

1. Receives notification from GitHub (via SNS)
2. Looks up config in DynamoDB

3. Clones git repo using a bundled git binary
4. Looks up config files in repo
5. Runs install and build cmds on Lambda (or starts ECS task)
6. Updates Slack and GitHub statuses along the way (optionally SNS for email, etc)
7. Uploads build logs/statuses to S3

How do all the pieces fit together?

Something like this:



Why isn't my build triggering on large pushes?

Most GitHub events are relatively small – except in the case of branch pushes that involve hundreds of files (pull request events are not affected). GitHub keeps events it sends under the SNS limit of 256kb by splitting up larger events, but because Lambda events are currently limited to 128kb (which will hopefully be fixed soon!), SNS will fail to deliver them to the Lambda function (and you'll receive an error in your CloudWatch SNS failure logs).

If this happens, and Beedo CI isn't triggered by a push, then you can just create a dummy commit and push that, which will result in a much smaller event:

```
git commit --allow-empty -m 'Trigger Beedo CI'
git push
```

License

MIT

CHAPTER 3

Build on docker

Steps necessary to setup an ECS cluster to run your Beedo CI builds

Running builds within docker containers is an optional feature of Beedo CI. This means the feature is not active by default.

To activate docker builds you need to setup a ECS cluster in your account. Don't worry it is just a few clicks.

For now we assume you run a micro instance in your ECS cluster. Of course you can scale up to many big irons. Easy to do but not within the scope of this howto.

1. Login to the AWS console of your account and create a ESC Cluster named "BeedoCluster". Just give the name - thats all.
2. Launch an EC2 instance into your BeedoCluster. TODO
3. Add the BeedoCluster Arn as parameter to your existing "Beedo CI Stack" and run an update on the stack.

Thats all. Once you have set the BeedoCluster parameter the builds run in the docker container.

CHAPTER 4

Logging

This document contains details on how to implement logging in Beedo CI.

testing the logging config

https://pythonhosted.org/testfixtures/logging.html

logging config

http://docs.python-guide.org/en/latest/writing/logging/

in main or similar:

```
import logging
from logging.config import dictConfig

logging_config = dict(
    version = 1,
    formatters = {
        'f': {'format':
              '%(asctime)s %(name)-12s %(levelname)-8s %(message)s'}
        },
    handlers = {
        'h': {'class': 'logging.StreamHandler',
              'formatter': 'f',
              'level': logging.DEBUG}
        },
    root = {
        'handlers': ['h'],
        'level': logging.DEBUG,
        },
)
```

```
dictConfig(logging_config)

logger = logging.getLogger()
logger.debug('often makes a very good meal of %s', 'visiting tourists')
```

now use the config in the rest of the app use

info https://www.internalpointers.com/post/logging-python-sub-modules-and-configuration-files

```
logger = logging.getLogger()
```

fileConfig sample:

```
[loggers]
keys=root

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=simpleFormatter
args=(sys.stdout,)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
datefmt=
```

dictConfig

```
logging.config.dictConfig({
    'version': 1,
    'disable_existing_loggers': False,  # this fixes the problem
    'formatters': {
        'standard': {
            'format': '%(asctime)s [%(levelname)s] %(name)s: %(message)s'
        },
    },
    'handlers': {
        'default': {
            'level':'INFO',
            'class':'logging.StreamHandler',
        },
```

```
    },
    'loggers': {
        '': {
            'handlers': ['default'],
            'level': 'INFO',
            'propagate': True
        }
    }
})
```

CHAPTER 5

Release history

0.9.11 (2017-02-11)

- made beedo github repo public

0.9.10 (2016-11-23)

- improved logging
- minor bugfixes

0.9.8 (2016-10-02)

- add git binary to lambda + functionality
- find and run .beedo.py
- build report page
- github integration
- github webhook to trigger build
- build status badge

TODO

- github PR builder set status
- use/dogfood aws-deploy tools (template, CF, lambda)

- port nodejs parts to Python
- use API gateway
- consolidate configuration + config file
- add shell build file feature
- better sample / demo repo (supercars?)
- (output in browser via webhooks)
- pseudo terminal (pty, terminado, ...) so clint.textui -> colored works
- webhook to trigger dependent builds
- beedo build status api / + silent api
- beedo firmware
- writeup usecase
- logo-48x48.png

CHAPTER 6

Development

This is currently a collection of recipes not a full dev guide, sorry.

locations for beedo-ci artefacts:

https://beedo-ci.s3.amazonaws.com/templates/beedo.template https://beedo-ci.s3.amazonaws.com/fn/beedo-ci-0.9.8.zip

url used to launch stack

https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/new?stackName=beedo-ci&templateURL=https://beedo-ci.s3.amazonaws.com/templates/beedo.template

note: please do not forget to change the region if you prefer to launch the stack in a different region.

run virtualenv

http://willyg302.github.io/blog/posts/2015-03-29-python-on-aws-lambda/

Nope, what we've got to do is make it so that Bob goes from zero to hero with minimal effort. To do so, we're going to need to write a Makefile:

```
VERSION = 12.0.5
VIRTUALENV = env
PYTHON = $(which python)

all:
    wget https://pypi.python.org/packages/source/v/virtualenv/virtualenv-$(VERSION).
↪tar.gz
    tar xzf virtualenv-$(VERSION).tar.gz
```

```
    $(PYTHON) virtualenv-$(VERSION)/virtualenv.py $(VIRTUALENV)
    rm -rf virtualenv-$(VERSION)
    rm virtualenv-$(VERSION).tar.gz
    $(VIRTUALENV)/bin/pip install beautifulsoup4
This vomit of make-y goodness bootstraps virtualenv locally without using sudo. The␣
↪only requirement is that the user has Python installed, which is a pretty␣
↪reasonable assumption to make.
```

alternative install virtualenv during bundle and use that!

```
import virtualenv
virtualenv.create_environment('./venv', site_packages=True)
```

working / troubleshooting on a amazon ami

http://www.perrygeo.com/running-python-with-compiled-code-on-aws-lambda.html

ec2_finklabs:

```
$ aws ec2 run-instances --image-id ami-0044b96f --count 1 --instance-type t2.micro␣
↪ --key-name ec2_finklabs --security-groups sg-4ef92027 --region eu-central-1 --
↪profile superuser-bee-do

$ ssh -i ./.aws/ec2_beedo.pem ec2-user@52.57.225.212

$ scp -i ./.aws/ec2_beedo.pem ~/devel/bee-do/beedo/lambda.zip ec2-user@52.57.225.212:~
↪/
```

b = {'config': {'virtualenv': {'name': 'mvenv', 'sitePackages': True}}, 'cloneDir': '.'}

from beedo.actions import build build._virtualenv(b)

amazonlinux in docker container

```
$ docker run -it 137112412989.dkr.ecr.us-west-2.amazonaws.com/amazonlinux:latest /bin/
↪bash
```

docu: http://docs.aws.amazon.com/AmazonECR/latest/userguide/amazon_linux_container_image.html

docker

```
$ docker build -t beedo .
```

and run from console:

```
$ docker run --rm -it beedo
```

using a container to pip install a package

```
$ docker run --rm -it -v beedo

$ virtualenv venv

$ . ./venv/bin/activate

$ pip install xattr

$ cp venv/lib/python27/site-packages/xattr .
```

ECS

prerequisites

- create ECS cluster manually (two clicks)
- create ecsInstanceRole (http://docs.aws.amazon.com/AmazonECS/latest/developerguide/instance_IAM_role.html)
- launch an instance use tool script
- upload container to ECS repo

build and upload a container:

1. Retrieve the docker login command that you can use to authenticate your Docker client to your registry: $ aws ecr get-login –region eu-central-1
2. Run the docker login command that was returned in the previous step.
3. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:

```
$ docker build -t beedocore .
```

1. After the build completes, tag your image so you can push the image to this repository:

```
$ docker tag beedocore:latest 580762641671.dkr.ecr.eu-central-1.amazonaws.com/
↪beedocore:latest
```

1. Run the following command to push this image to your newly created AWS repository:

```
$ docker push 580762641671.dkr.ecr.eu-central-1.amazonaws.com/beedocore:latest
```

release beedo-ci

```
$ bumpversion patch
```

Upload release to PiPy:

```
$ python setup.py sdist upload
```

Install develop (see above):

```
$ pip install -e .
```

Add the version change to the commit:

```
$ git commit -a --amend --no-edit
```

if cache bothers you:

```
$ pip install --no-cache-dir botodeploy==0.0.2
```