fink Documentation

Release 1.0.2

fink

Dec 04, 2017

Contents

CHAPTER 1

Introduction

This userguide aims to make it easy for you to get started using the fink tools in your projects. fink helps you to code your infrastructure on AWS and put it under version control as infrastructure-as-code together with the implementation of your service. In this way you can fully automate your infrastructure and your service deployments.

fink provides tools for traditional compute services and also managed serverless computing services. fink was implemented internally at finklabs.

fink and userguide are released under MIT License.

The fink userguide starts with this introduction, then provides an overview on fink like a guided tour on how fink is structured and what it offers to you. The following parts each covers one fink tool. The remaining parts go into more technical topics like developing fink or using it as library to build your own tools based on fink.

This user guide assumes that you know the AWS services you want to automate so we do not cover AWS services in great detail and instead point to relevant documentation. But even if you are starting out on AWS, fink will help you to quickly leave the AWS webconsole behind and to move towards infrastructure-as-code.

1.1 Related documents

This section aims to provide to you a list of related documents that will be useful to gain a detailed understanding about what the fink tool suite does. With this background you will be able to tap into the full potential of the fink tools.

1.2 Problem reporting instructions

Please use Github issues to report fink issues: fink issues. To check on the progress check out the fink project board: fink project board

CHAPTER 2

Getting Started Guide

Welcome to the getting started guide of `fink`. In this guide we will cover what `fink` is and how to use it to create beautiful infrastructure as code (IaC):

`fink` tools to manage AWS infrastructure using CloudFormation and CodeDeploy:

- using AWS CloudFormation with `fink.cloud`
- deploy your application on AWS EC2 with AWS CodeDeploy scripts and `fink.code`

`fink` contains two more tools to manage serverless infrastructure on AWS:

- deploy and configure AWS Lambda with `fink.lambda`
- deploy API Gateway and manage API keys with `fink.api`

All of these things you can do for different Environments (dev, stage, prod)

2.1 Infrastructure as code

Infrastructure as Code (IaC) is a type of IT infrastructure that teams can automatically provision through code, rather than using a manual GUI-centered process. Through defining your infrastructure as code you can apply similar tools and workflows like when working with your application code like:

- version control
- test
- review
- share
- reuse (like creating test envs)
- audit

2.2 Installation

2.2.1 Install Python

First of all you need to have Python installed. Python should be 2.7 or higher

On MacOS try to use preinstalled Python

2.2.2 Install pip and virtualenv

virtualenv is a tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

MacOS

```
$ sudo pip install virtualenv --upgrade
```

2.2.3 Install fink and fink plugins

Install fink

First of all you need to create virtualenv and activate it. We recommend create virtualenv in the same directory as a project, and add it to `.gitignore`. It's pretty easy.

```
$ cd <project-folder>
$ virtualenv venv
$ source venv/bin/activate
$ pip install pip --upgrade # we always should have latest pip version in our␣
↪virtualenv
```

fink needs some `fink-glugins` so you should install these together. `fink-glugins` are powerful tool to add features to `fink` without having to directly modify the `fink` core. The easiest way is to put the dependencies into a *requirements_fink.txt* file:

```
fink
fink.config-reader
fink.lookups
fink.bundler
fink.slack-integration
fink.cloud
fink.code
fink.lambda
fink.api
```

You can find more information about plugins in docs then

```
$ pip install -U -r requirements_fink.txt
```

To check that everything is good and `fink` installed just do:

```
$ fink version
fink version 0.1.418
fink commands:
 * fink.cloud 1.0.0
 * fink.code 1.0.0
 * fink.lambda 1.0.0
 * fink.api 1.0.0
fink plugins:
 * fink.config-reader version 1.0.0
 * fink.bundler version 1.0.0
 * fink.slack-integration version 1.0.0
 * fink.lookups version 1.0.0
```

2.2.4 Setting the ENV environment variable

For almost all fink commands you need to set the `ENV` environment variable. `ENV` is required to recognize which config file to use (`fink_<env>.json`). Usually you have a different config file for each environment (dev, stage, prod). You need to set `ENV` before running any `fink` command.

The following command will use the `fink_dev.json` config file

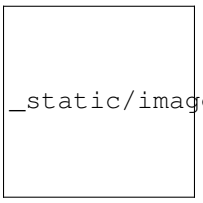
```
$ export PYTHONIOENCODING=UTF-8
$ ENV=dev cloud list
...
```

Alternatively you can set the `ENV` variable for the duration of your terminal session. Set it like this:

```
$ export ENV=dev
$ cloud list
...
```

2.2.5 preparing your AWS account for deployments using fink tools

When you store your infrastructure as code in a version control system you do not want to keep the secret credentials with them for obvious reasons. Also you want to have the flexibility to deploy the same stack to different accounts without changing the code. In order to achieve this we store the credentials in AWS SSM parameter store:



_static/images/docs/_static/images/AWS_parameter_store_parameters.png

Another preparation step you need to take is a basestack (look for fink.base-sample-stack). This stack contains some essential infrastructure elements you need so your infrastructure deployments will work like roles, policies, an S3 artifact bucket etc. The basestack exposes some values using `Outputs` which can be looked up during deployments. So you deploy the basestack once per account and all other stack use this infrastructure.

2.3 fink.cloud

fink.cloud is a tool which help you to manage and deploy your infrastructure. AWS Cloudformation uses helps you configure and manage your AWS infrastructure as code. In `cloud` we have our cloudformation templates generated by troposphere. With cloud you can easily create (and configure) infrastructure for different environments (and AWS accounts) like for example (dev, stage, prod).

2.3.1 Create your first stack with cloud

First of all you need to create two files:

- *cloudformation.py* - here you will describe your infrastructure using `troposphere`
- *fink_(dev|stage|prod)* - settings for your ENV in `json` format, needs to include all parameters for the cloudformation template + stack name. You should have separate config for each ENV.

Let's create a simple `fink_dev.json` *(please change all values according your AWS account)*:

```
{
  "cloud": {
    "stack": {
      "StackName": "fink.superc-sample-stack"
    },
    "parameters": {
      "VPCId": "lookup:stack:<stack-name>:DefaultVPCId",
      "ScaleMinCapacity": "1",
      "ScaleMaxCapacity": "1",
      "InstanceType": "t2.micro",
      "DefaultInstancePolicyARN": "lookup:stack:<stack-name>:DefaultInstancePolicyARN
↪",
      "AMI": "lookup:parameter:base_ami"
    }
  }
}
```

The values for `VPCId` and `DefaultInstancePolicyARN` are filled by by the `fink-lookups` which then will be used in the template. The `fink.lookups` plugin will search the outputs in the CloudFormation stack (as mentioned in the config).

Instead of `<stack-name>` you should provide your stack name or use hardcoded value(not recommended). It's time to create our first Infrastructure as Code. Let's do this. Here is a simple cloudformation.py script. Use it as a template for creating your infrastructure.

2.3.2 Deploy a stack to AWS

Before running a deployment we need to set some necessary ENV variables. **Remember**: You need this ENV variables exported each time before running any `fink` command.

```
$ export ENV=dev
$ export AWS_DEFAULT_PROFILE=fink # Default profile.
$ export AWS_DEFAULT_REGION=eu-west-1
$ export PYTHONIOENCODING=UTF-8
```

Run your first infrastructure deployment. It's really easy:

```
$ cloud deploy
```

More information about `fink.cloud` can be found in docs

2.4 fink.lambda

fink.lambda will help you to deploy, manage and control AWS Lambda functions. Runtimes supported by `lambda` are: **nodejs4.3, nodejs6.10, python2.7, python3.6**

2.4.1 Deploy a simple AWS Lambda function

Create a `fink_(dev|stage|prod).json` file *(please change all values according your AWS account)*:

```
"lambda": {
  "bundling": {
    "folders": [
        {
            "source": "./node_modules",
            "target": "./node_modules"
        }
    ],
    "zip": "bundle.zip"
  },
  "lambda": {
    "name" = "jenkins-fink-lifecycle-for-lambda",
    "description" = "lambda test for lambda",
    "role" = "lookup:stack:<stack-name>:LambdaArnForDeploy",
    "handlerFunction" = "handler.handle",
    "handlerFile" = "handler.py",
    "timeout" = "300",
    "memorySize" = "256",
    "vpc": {
            "subnetIds": [
                "lookup:stack:<stack-name>:LambdaSubnetIda",
                "lookup:stack:<stack-name>:LambdaSubnetIdb",
                "lookup:stack:<stack-name>:LambdaSubnetIdc"
            ],
        }
  }
}
```

then do:

```
$ lambda deploy
```

More information about `fink.lambda` can be found in docs

2.5 fink.code

fink.code will help you to deploy your application using AWS CodeDeploy. code will create an artifact bundle file and upload it to s3 which contains all files that you have in your `codedeploy` folder. Create the `fink_(dev|stage|prod).json` file *(please change all values according your AWS account)*:

```
"code": {
  "codedeploy": {
    "applicationName": "lookup:stack:fink-sample-stack:applicationName",
    "deploymentGroupName": "lookup:stack:fink-sample-stack:DeploymentGroup",
    "deploymentConfigName": "lookup:stack:fink-sample-stack:DeploymentConfig",
    "artifactsBucket": "lookup:stack:<stack-name>:s3DeploymentBucket"
  }
}
```

then do:

```
$ code deploy
```

More information about `fink.code` can be found in docs

2.6 fink.api

fink.api is a tool that will help you to deploy and manage your API with AWS API Gateway. All you need is to put your `swagger.yml` into the same folder as a `fink_(dev|stage|prod).json` file. Also, add some new configs into it *(please change all values according your AWS account)*:

```
"api": {
    "api": {
        "apiKey": "xxxxxxxxxxxxxx",
        "description": "Gcdt sample API based on dp api-mock",
        "name": "jenkins-fink-sample-api-dev",
        "targetStage": "mock"
    }
}
```

More information about `fink.api` can be found in docs

CHAPTER 3

Overview

Outline how fink works at a high level

- Identify the key functions
- Inputs and Outputs - identify inputs what you, the reader, need to enter
- and outputs, what you expect in response for example reports
- provide further details
- assumptions on user experience, for examples, experience or proficiency in related areas or previous training

CHAPTER 4

Installing fink

This chapter covers the fink installation. fink's behaviour can be customized using plugins. The fink plugin mechanism relies on standard Python package mechanisms. In order to get a good experience and get the most out of fink you need to know a few things about Python packaging.

This chapter aims to provide you with all the information you need to know on this topic.

4.1 Related documents

4.2 What you need to know about python package management

There is now a lot of packaging infrastructure in the Python community, a lot of technology, and a lot of experience. We will try cover some basic things and give you best practice what to use for Python package management.

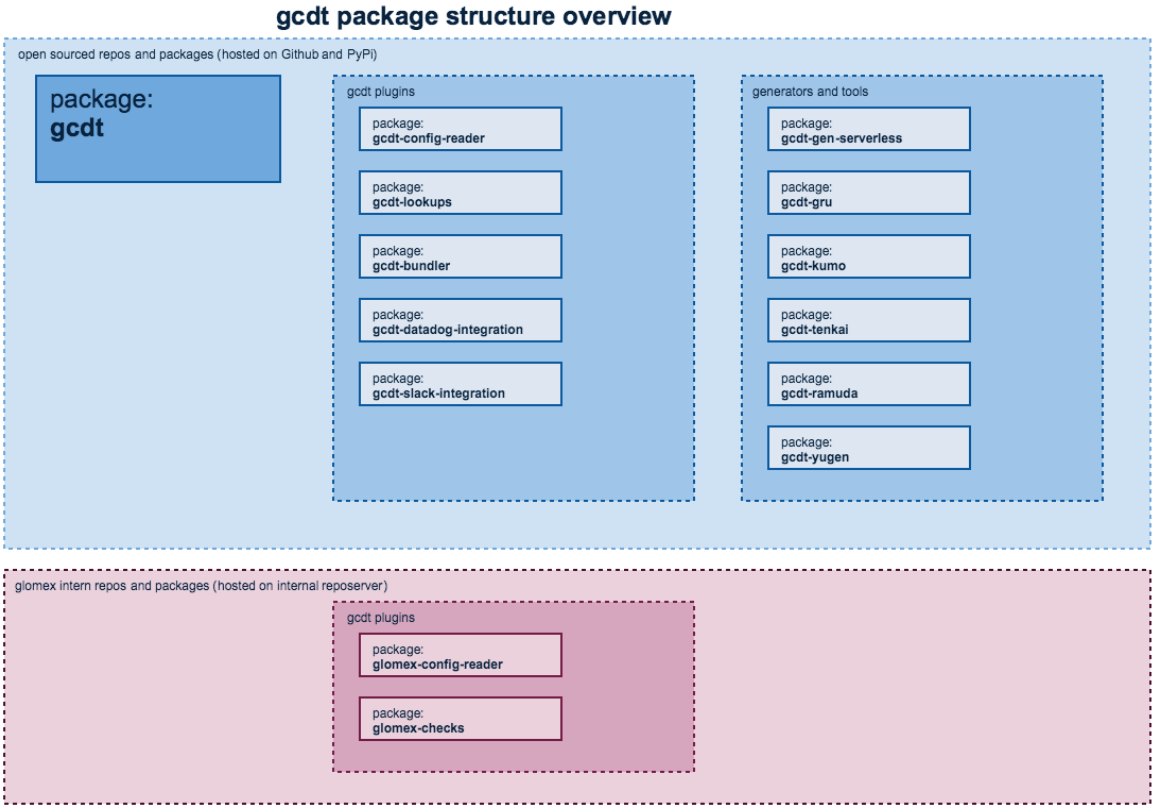
1. **Always use `virtualenv`.** A virtualenv is effectively an overlay on top of your system Python install. Creating a virtualenv can be thought of as copying your system Python environment into a local location. When you modify virtualenvs, you are modifying an isolated container. Modifying virtualenvs has no impact on your system Python.
2. **Use `pip` for installing packages.** Python packaging has historically been a mess. There are a handful of tools and APIs for installing Python packages. As a casual Python user, you only need to know of one of them: pip. If someone says install a package, you should be thinking create a `virtualenv`, activate a `virtualenv`, `pip install <package>`. You should never run `pip install` outside of a `virtualenv`. (The exception is to install `virtualenv` and pip itself, which you almost certainly want in your system/global Python.)
3. **Use `requirements` file for installing all project dependencies**. Always strictly specify the package version. Bad one: `somepackage=>2.0.3`. Good one: `somepackage==2.0.3`

Here is some useful links if you want dive deeper into Python package management.

4.3 fink package structure

The following diagram gives an overview on the fink packages. Please note how we grouped the fink packages in the following categories:

- fink - the fink core (livecycle mechanism, fink tools)
- fink plugins - packages to customize how you use fink
- fink generators and tools - scaffolding and tools to make your work even more efficient



At finklabs we have very few (currently one) fink packages we do not want to open-source. The finklabs-config-reader has very opinionated defaults on how we use fink on our AWS infrastructure that is very specific and optimized for our media usecase.

4.4 Maintaining dependencies for your project

It is a very common practice not to install Python packages by hand. Instead dependencies and version are managed in a documented and repeatable way. Basically you add the names and versions of your packages to a text file. Most projects also group their dependencies into `direct` dependencies of the service or application and packages they need to develop, build, test and document.

The grouping is not enforced by packaging but to have a std. within an organization is beneficial especially if your want to reuse CI/CD tools.

A little opinionated but pretty common:

- `requirements.txt` tools and packages your service directly depends on

- `requirements_def.txt` tools and packages you need to develop and test your service
- `requirements_fink.txt` fink and fink plugins you use to deploy your service to AWS

The easiest way to install fink is via pip and virtualenv.

4.5 Defining which fink plugins to use

fink needs at least some fink-glugins so you should want to install these together. Add `fink` and the plugins you use to *requirements_fink.txt*

```
fink
fink.say-hello
fink.config-reader
fink.lookups
fink.bundler
fink.slack-integration
fink.gen-serverless
fink.cloud
fink.code
fink.lambda
fink.api
```

This is also a best practice to use the same `requirements_fink.txt` file on your build server, too.

4.6 Setup virtualenv

Using virtualenvs for Python is considered best practice. This is what you need to do:

- create a virtualenv ('$ virtualenv venv')
- install the packages you want to use (see above)
- a virtualenv works basically like every other technical device, you need to switch it on before you can use it ('$ source ./venv/bin/activate')

Prepare the venv:

```
$ virtualenv venv
```

Activate the venv for use:

```
$ source ./venv/bin/activate
```

4.7 Installing all dev dependencies in one go

Install the dependencies into venv:

```
$ pip install -U -r requirements_fink.txt
```

Now you can start using fink:

```
$ fink version
```

BTW, `fink version` shows you all the versions of fink and installed plugins. So you can use this to quickly check which plugins are installed.

4.8 Deactivate a virtualenv

I do not throw away my lawn mower once I am done but with my terminals I do that. But you can deactivate a virtualenv:

```
$ deactivate
```

4.9 Updating fink

You should frequently update your fink installation to get access to new features and bugfixes. When updating your fink installation, please update fink and all the plugins. Just updating fink or a single plugin could easily break your fink installation.

```
$ pip install -U -r requirements_fink.txt
```

This will update `fink` and all `fink plugins` specified in *requirements_fink.txt*

4.9.1 General remarks on "breaking changes" and deprecated features

We have two conflicting goals when maintaining fink:

- we want to introduce new features and replace older ones with newer implementations
- we want to be compatible with existing infrastructure and configurations

Besides that this is a pretty standard situation for a deployment tool and many other software projects. Nevertheless we want to make it explicit and consequently document here how we handle this.

We make sure that fink does work with our existing configurations. Frequent releases are ok but our squads expect downward compatibility for new patch versions. We could accept minor releases with breaking changes but expect an "update-documentation" which documents all steps that are necessary to upgrade.

We recently discussed and confirmed this in the fink grooming on 05.07.17 with representatives of the CO and VE squads.

In case we need to deprecate something we announce that in the fink changelog respectively in the changelog of fink plugins and tools.

The following sections go into `dependency specification` and provide information for how to upgrade from one version to the next.

4.9.2 Dependency specification (pinning versions)

Like we said above we guarantee compatibility with your installation procedures and configurations for patch versions. If you need to enforce compatibility it is recommended to best pin fink packages to minor versions in `requirements_fink.txt`.

This is just a sample, you need to pin to the actual versions you want to use:

```
fink==1.0.*
fink.config-reader==1.0.*
fink.lookups==1.0.*
fink.bundler==1.0.*
fink.slack-integration==1.0.*
fink.cloud==1.0.*
fink.code==1.0.*
fink.lambda==1.0.*
fink.api==1.0.*
```

Detailed information on Version Identification and Dependency Specification.

4.9.3 Updating from gcdt to fink 1.0.0

changed tool names

To better match AWS service names we changed the tool names in fink 1.0.0. The old Japanese names used by gcdt have been nice but proved difficult to remember. To ease migration the gcdt commands still can be used for another say 3-4 month but after that we use the new command names only.

fink command	gcdt command	AWS service name
cloud	kumo	CloudFormation
code	tenkai	CodeDeploy
lambda	ramuda	AWS Lambda
api	yugen	API Gateway

removed long deprecated hook mechanism

Removed long deprecated hook support from cloud, code and lambda. No more 'pre_bundle', 'pre_hook', 'pre_create_hook', 'pre_update_hook', 'post_create_hook', 'post_update_hook', 'post_hook' any more.

Please use the fink lifecycle hook mechanism instead:

cloud lifecycle hooks using hooks in fink list of available hooks to use

you explicitly need to install all used fink tools

You need to install fink-tools (fink-cloud, fink-code, fink-lambda, fink-api). Please add the tools to your `requirements_fink.txt` like described in the installation section above.

remove the deprecated ami lookup

Newer finklabs `base_ami` uses a different naming scheme. Consequently the ami lookup implemented in fink provides you with old base_ami ids.

moved cloudformation helpers to fink.cloud

If you use `iam` and `route53` helpers you need to change imports for these submodules from `gcdt` to `fink.cloud`.

we introduce config validation for all fink tools and plugins

The new config validation looks for required properties and does format checks in some cases. fink now also validates datatypes. We found some errors where `string type` was uses in existing configurations instead of the correct `integer type` and you need to fix this before your configuration can pass as valid:

A sample for a valid use of integer values:

```
"lambda": {
  ...
  "memorySize": 128,
  "timeout": 15,
}
```

CHAPTER 5

fink tool

5.1 Related documents

5.1.1 Setting the ENV variable

You you need to set an environment variable "ENV" which indicates the account/staging area you want to work with. This parameter tells the tools which config file to use. For example if you want to set the environment variable ENV to 'DEV' you can do that as follows:

```
export ENV=DEV
```

5.2 Usage

To see available commands, call fink without any arguments:

```
$ fink
Usage:
        fink config
        fink version
```

5.3 Commands

5.3.1 config

This command is intended to provide tooling support for maintaining configuration.

The `config` command does the following:

- read configuration defaults

- read the config from file ('fink_<env>.json')
- run the lookups
- format and output the config to the console

5.3.2 version

If you need help please ask on the fink slack channel or open a ticket. For this it is always great if you are able to provide information about the fink version you are using. A convenient way to find out the version of your fink install provides the following command:

```
$ fink version
WARNING: Please consider an update to fink version: 0.1.433
fink version 0.1.432
fink plugins:
 * fink.config-reader version 0.0.11
 * fink.bundler version 0.0.27
 * fink.slack-integration version 0.0.11
 * fink.lookups version 0.0.12
```

`fink version` also provides you with an easy way to check whether a new release of fink is available.

CHAPTER 6

cloud command

`cloud` is finks CloudFormation deploy tool.

6.1 Related documents

6.2 Usage

To see available commands, call cloud without any arguments:

```
Usage:
        cloud deploy [--override-stack-policy] [-v]
        cloud list [-v]
        cloud delete -f [-v]
        cloud generate [-v]
        cloud preview [-v]
        cloud dot [-v]
        cloud stop [-v]
        cloud start [-v]
        cloud version

-h --help           show this
-v --verbose        show debug messages
```

6.3 Commands

6.3.1 deploy

will create or update a CloudFormation stack

to be able to update a stack that is protected by a stack policy you need to supply "–override-stack-policy"

6.3.2 list

will list all available CloudFormation stacks

6.3.3 delete

will delete a CloudFormation stack

6.3.4 generate

will generate the CloudFormation template for the given stack and write it to your current working directory.

6.3.5 preview

will create a CloudFormation ChangeSet with your current changes to the template

6.3.6 dot

Visualize the cloudformation template of your stack using `cloud dot`.

Installation of the dot binary is required on your Mac to convert the graph into svg (http://www.graphviz.org/Download_macos.php).

```
$ brew install graphviz
```

6.3.7 stop

```
"cloud stop" is a brand new feature we start rolling out to finklabs AWS accounts.
We would like your feedback. Please talk to us if you require any improvements
(additional resources etc.).
```

Use `cloud stop` to stop resources contained in your cloudformation stack using `cloud stop`.

`cloud stop` currently comprises of the following features:

- resize autoscaling group to minSize=0, maxSize=0
- stop Ec2 instances
- stop RDS

Add this optional configuration to the `cloud` section of the config file to exclude your stack resources from start / stop.

```
...
"deployment": {
  "DisableStop": true
}
```

6.3.8 start

```
"cloud start" is a brand new feature we start rolling out to finklabs AWS accounts.
We would like your feedback. Please talk to us if you require any improvements.
```

Start resources contained in your cloudformation stack using `cloud start`.

`cloud start` currently comprises of the following features:

- start RDS
- start EC2 instances
- restore autoscaling group minSize, maxSize to original values

6.3.9 version

will print the version of fink you are using

6.4 Folder Layout

The folder layout looks like this:

`cloudformation.py` -> creates troposphere template, needs a method like this:

```
def generate_template():
    return t.to_json()
```

`fink_dev.json` -> settings for dev in json format, needs to include all parameters for the cloudformation template + stack name

Further settings files, depending on your environments in the format of `fink_<ENV>.json`

6.4.1 Config file example

```
"stack": {
    "StackName": "sample-stack"
}
```

You like examples better than documentation? Check out our sample-stack at https://github.com/finklabs/fink-sample-stack/tree/master/infrastructure

6.4.2 Configuring RoleARN for a cloudformation stack

There is a new Feature in CloudFormation which lets a User specify a Role which shall be used to execute the Stack. Docs can be found at http://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/API_CreateStack.html This can be used to limit access of users drastically and only give CloudFormation the permission to do all the heavy lifting.

```
"stack": {
    "RoleARN": "arn:aws:iam::<AccountID>:role/<CloudFormationRoleName>"
}
```

Make sure the role may be assumed by CloudFormation. See also: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-iam-servicerole.html

6.4.3 Configuring NotificationARNs for a cloudformation stack

Amazon Simple Notification Service topic Amazon Resource Names (ARNs) that AWS CloudFormation associates with the stack.

```
{
  "cloud": {
    "stack": {
      "StackName": "infra-dev-cloud-sample-stack",
      "NotificationARNs": [
        "arn:aws:sns:eu-west-1:123456789012:mytopic1",
        "arn:aws:sns:eu-west-1:123456789012:mytopic2"
      ]
    },
    ...
```

Specify an empty list to remove all notification topics.

6.4.4 Setting the ENV variable

You you need to set an environment variable "ENV" which indicates the account/staging area you want to work with. This parameter tells the tools which config file to use. For example if you want to set the environment variable ENV to 'DEV' you can do that as follows:

```
export ENV=DEV
```

This can also be exploited to have different configuration for different regions which is not yet directly supported.

```
export ENV=DEV_eu-west-1
```

Will load the config file named `fink_dev_eu-west-1.json`

6.5 Howto

1. create and fill `cloudformation.py` with the contents of your stack
2. create and fill `settings_<env>.conf` with valid parameters for your CloudFormation template
3. call `cloud deploy` to deploy your stack to AWS

6.6 Kumo lifecycle hooks

Kumo lifecycle hooks work exactly like fink lifecycle hooks but have a specialized integration for cloud templates.

```
def my_hook(params):
    context, config = params
    ...

def register():
```

```
    """Please be very specific about when your hook needs to run and why.
    E.g. run the sample stuff after at the very beginning of the lifecycle
    """
    fink_signals.initialized.connect(my_hook)


def deregister():
    fink_signals.initialized.disconnect(my_hook)
```

One cloud speciality for the `command_finalized` hook is that you can access the context attribute 'stack_output'. to access and use outputs of your stack within the hook implementation.

6.7 DEPRECATED Kumo legacy hooks

The hooks in this section are deprecated please use fink lifecycle hooks (see above).

Please note the legacy hooks will be removed with the next minor release (v 0.2.0).

cloud offers numerous hook functions that get called during the lifecycle of a cloud deploy run:

- pre_hook()
  - gets called before everything else - even config reading. Useful for e.g. creating secrets in credstash if they don't exist
- pre_create_hook()
  - gets called before a stack is created
- pre_update_hook()
  - gets called before a stack is updated
- post_create_hook()
  - gets called after a stack is created
- post_update_hook()
  - gets called after a stack is updated
- post_hook()
  - gets called after a stack is either updated or created

You can basically call any custom code you want. Just implement the function in `cloudformation.py`

Multiple ways of using parameters in your hook functions:

- no arguments (as previous to version 0.0.73.dev0.dev0)
- use kwargs dict and just access the arguments you need e.g. "def pre_hook(**kwargs):"
- use all positional arguments e.g. "def pre_hook(awsclient, config, parameters, stack_outputs, stack_state):"
- use all arguments as keyword arguments or mix.
- with version 0.0.77 we decided to move away from using boto_sessions towards awsclient (more flexible and low-level).

6.8 Using fink functionality in your cloudformation templates

Historically `cloudformation.py` templates imported functionality from fink and finklabs_utils packages. With version 0.0.77 we consolidated and copied `get_env` over to fink.utils.

Made functionality available in fink (sounds awful but it was there already anyway) :

- fink.utils: get_env now available

Continued no changes:

- fink.iam: IAMRoleAndPolicies

The following functionality requires `awsclient` to lookup information from AWS. The `awsclient` is available in the cloudformation template only within the scope of a hook (see above). Consequently you need to execute your calls within the scope of a hook:

- fink.servicediscovery: get_outputs_for_stack
- fink.route53: create_record
- fink.cloud_util: ensure_ebs_volume_tags_autoscaling_group

6.9 Accessing context and config in cloudformation

In the last few month we learned about a few usecases where it is desired to have access to config and context within your template. We had some workarounds using hooks but now there is a proper implementation for this feature.

In order to access context and config in your `cloudformation.py` you need to add both `context` and `config` as arguments to the 'generate_template? function of your template:

```
def generate_template(context, config):
    template = troposphere.Template()
    ph.initialize(template, 'miaImportProcessor')
    assemble_particles(template, context, config)
    return template.to_json()
```

In case you do not want to use this information in your template you don't have to use it (like before).

```
def generate_template():
    ...
```

6.10 Stack Policies

cloud does offer support for stack policies. It has a default stack policy that will get applied to each stack:

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:Modify",
      "Principal": "*",
      "Resource" : "*"
    },
    {
      "Effect" : "Deny",
```

```
      "Action" : ["Update:Replace", "Update:Delete"],
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

This allows an update operation to modify each resource but disables replacement or deletion. If you supply "–override-stack-policy" to cloud then it will use another default policy that gets applied during updates and allows every operation on every resource:

```
{
  "Statement" : [
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

If you want to lock down your stack even more you can implement two functions in your cloudformation.py file:

- get_stack_policy()
- – the actual stack policy for your stack
- get_stack_policy_during_update()
- – the policy that gets applied during updates

These should return a valid stack policy document which is then preferred over the default value.

6.11 Signal handling

cloud receives a SIGINT or SIGTERM signal during a stack update `cancel_update_stack` is called for the stack.

CHAPTER 7

cloud config 1.0.3

7.1 Description

Documentation of the config file format for *cloud* (a fink tool). Note: if you want to add to the documentation please edit the openapi_cloud.yaml file

7.2 Data Structures

7.2.1 cloud - structure

The cloud config is organized into the following structure:

Name	Required	Type	Format	Properties	Description
defaults	Yes	*defaults*			finetune fink tool lifecycle (override at own risk)
deployment	No	*deployment*			details regarding the deployment phase of the stack
parameters	No	*parameters*			parameters used in the cloudformation template
stack	Yes	*stack*			configure stack details

7.2.2 stack - structure

Use the *stack* section to configure stack details.

Name	Required	Type	Format	Properties	Description
NotificationARNs	No	array of *arn*			configure SNS recipients for stack events
RoleARN	No	string			role to use for cloudformation deployment
StackName	Yes	string			name of your cloudformation stack
TemplateBody	No	string			
artifactBucket	No	*arn*			s3 bucket use to upload the cloudformation template to AWS

7.2.3 parameters - structure

AWS cloudformation parameters. Parameters in the config must match the parameters used in the template!

7.2.4 deployment - structure

Stack properties specific to the deployment phase.

Name	Required	Type	Format	Properties	Description
DisableStop	No	boolean			disable the cloud stop & start mechanism for this stack.

7.2.5 defaults - structure

Default properties to finetune fink tool lifecycle (override at own risk).

Name	Required	Type	Format	Properties	Description
non_config_commands	Yes	array of string		{'default': ['stop', 'start', 'list']}	cloud commands that do not require a config file.
validate	Yes	boolean		{'default': True}	use this if you need to switch off config validation.

7.2.6 arn - structure

Amazon Resource Name

Name	Required	Type	Format	Properties	Description
arn	No	string			Amazon Resource Name

CHAPTER 8

cloud particles

At finklabs we create our infrastructure via AWS cloudformation and in order to do that efficiently we wanted to use reusable building blocks. For that to achieve we where evaluating different solutions available to us via open source.

So this is basically conceptual paper-ware (structure and a few helpers). You still need to write the particles.

Please be aware not to let cloud particles stop you from anything. In case you do not have particles or you do not want to write any you can still build beautiful infrastructure from the raw services provided by AWS.

cloud particles are perfectly optional. There is no tight coupling! You can totally bring your own building-block-mechanism and still use cloud for deployment. You do not even have to use troposphere - as long as your mechanism can export a valid json cloudformation template we are fine. Actually we encourage you to do so. Please share with us what you come up with.

8.1 Related documents

8.2 Goals

- codify best practices for infrastructure
- use `cloudformation.py` to assemble a stack from particles
- complexity is handled in particle

8.3 Detailed requirements

- particle has default parameters that can be overridden
- particle provides default permission that can be overridden
- we distribute particles as python packages (later move away from github subprojects)
- we want to distribute generic std. particles company wide (e.g. finklabs-particles)

- we want to distribute squad specific particles (e.g. mes-particles)

8.4 Status on cloud particles implementation

- cloud particle implementation is based on MES `template_generator.py`
- answered "what is the minimum information we need to provide to use a particle?"
- restore troposphere character (talked about context => template is the context)
- added SERVICE_NAME and DEFAULT_TAGS to template
- I liked the "template_generator" implementation but class structure gets in the way when creating stacks from multiple particle sources
- move cloudformation parameters and outputs into particle
- move permissions profile to particle

TODOs

- create particle profiles using awacs to further shrink the particles
- look into naming conventions / tooling for autogenerated resource names here it is important that in case we generate random names we can regenerate the same names during testing (fink placebo tools)
- share particles via package (need Github repo, Jenkins build, ...)

8.5 Usage

To build better infrastructure at finklabs we want to assemble infrastructure from reusable particles.

The `fink.cloud_particle_helper` module contains the functionality (initialize, get_particle_permissions, and Particle) to integrate the cloud particles into troposphere

8.6 Sample particles

8.6.1 instance

will create or update a CloudFormation stack

8.7 Quickstart example using particles

With cloud particles you can import particles from multiple sources:

```
from fink_cloud import cloud_particle_helper as ph
import eventbus_particle as eb
import reusable_particles as rp
```

We use `cloudformation.py` to assemble a stack from particles:

```
def assemble_particles(template):
    ################ parameters ##############################################
    param_sns_alerts_topic = template.add_parameter(troposphere.Parameter(
        'SNSAlertsTopic',
        Description='Name for topic that receive notifications for validation.',
        Type='String'
    ))

    ################ particles ###############################################
    particles = []  # list of Particle()
    sg_frontend_web = Ref('%sFrontendWeb' % template.SERVICE_NAME)

    ################ s3 bucket ###############################################
    particles.append(rp.create_s3_bucket(template))
    ...
```

Under the hood we use `troposphere` to code cloudformation templates. The troposphere template instance is used as a common `context` to exchange information between cloud particles. With cloud each `cloudformation.py` needs to implement a generate_template function.

```
def generate_template():
    template = troposphere.Template()
    ph.initialize(template, 'miaImportProcessor')
    assemble_particles(template)
    return template.to_json()
```

8.8 Developing your own particles

We just started with cloud particles and plan to provide more help on particle development in the future.

CHAPTER 9

code command

`code` is finks CodeDeploy tool.

9.1 Related documents

9.2 Usage

To see available commands, call this:

```
Usage:
        code bundle [-v]
        code deploy [-v]
        code version

-h --help           show this
-v --verbose        show debug messages
```

9.2.1 deploy

bundles your code then uploads it to S3 as a new revision and triggers a new deployment

9.2.2 version

will print the version of fink you are using

9.3 Folder Layout

codedeploy -> folder containing your deployment bundle

codedeploy_env.conf -> settings for your code

```
"codedeploy": {
    "applicationName": "mep-dev-cms-stack2-mediaExchangeCms-F5PZ6BM2TI8",
    "deploymentGroupName": "mep-dev-cms-stack2-mediaExchangeCmsDg-1S2MHZ0NEB5MN",
    "deploymentConfigName": "CodeDeployDefaultemplate.AllAtOnce01",
    "artifactsBucket": "7finity-portal-dev-deployment"
}
```

9.4 code configuration

9.4.1 add stack_output.yml to your code bundle

If you need a convenient way of using the stack output during codedeploy on your instance then you can use this feature.

`code` adds a `stack_output.yml` to the bundle artifact if you add the following configuration:

```
{
    'stack_output': 'lookup:stack:<your_stack_name>'
    ...
}
```

9.4.2 Adding a settings.json file

code supports a `settings` section. If it is used a `settings.json` file is added to the zip bundle containing the values. You can specify the settings within the `code` section.

```
    ...
    "settings": {
        "MYVALUE": "FOO"
    }
```

You can use lookups like for the rest of the configuration. Note that the values are looked up BEFORE the the instance is deployed via codedeploy. If values change during the instance lifecycle it does not recognise the changes. For values that must be updated you should lookup the values in your code using for example credstash.

```
    ...
    "settings": {
        "accountId": "lookup:stack:infra-dev:AWSAccountId"
    }
```

9.4.3 Configure log group

In case `code deploy` fails we attempt to provide the log output from the ec2 instance to ease your troubleshooting. The default log group is '/var/log/messages'. In case your ec2 instances are configured to log into another log group you can provide the necessary log group configuration to code like this:

```
"code": {
    ...
    "deployment": {
        "LogGroup": "/my/loggroup"
```

```
    }
}
```

Note: as a convention each ec2 instances has its own log stream with using the instanceId as name of the stream.

9.4.4 Setting the ENV variable

You you need to set an environment variable "ENV" which indicates the account/staging area you want to work with. This parameter tells the tools which config file to use. For example if you want to set the environment variable ENV to 'DEV' you can do that as follows:

```
export ENV=DEV
```

9.5 Signal handling

code receives a SIGINT or SIGTERM signal during a deployment `stop_deployment` is called for the running deployment with `autoRollbackEnabled`.

CHAPTER 10

fink.code configuration 1.0.1

10.1 Description

Documentation of the config file format for *code* (a fink tool). Note: if you want to add to the documentation please edit the openapi_code.yaml file

10.2 Data Structures

10.2.1 code - structure

The code config is organized into the following structure:

Name	Required	Type	Format	Properties	Description
bundling	No	*bundling*			configure details regarding bundle artifact
codedeploy	Yes	*codedeploy*			configure codedeploy details
defaults	Yes	*defaults*			finetune fink tool lifecycle (override at own risk)

10.2.2 codedeploy - structure

The *codedeploy* section.

Name	Required	Type	Format	Properties	Description
applicationName	Yes	string			
artifactsBucket	No	string			s3 bucket name used to upload the artifact bundle
deploymentConfigName	Yes	string			
deploymentGroupName	Yes	string			

10.2.3 bundling - structure

Configuration specific to the bundling phase.

Name	Required	Type	Format	Properties	Description
folders	Yes	array of *foldersItem*			an array of folder items
zip	No	string		{'default': 'bundle.zip'}	filename for the artifact bundle (e.g. bundle.zip)

10.2.4 defaults - structure

Default properties to finetune fink tool lifecycle (override at own risk).

Name	Required	Type	Format	Properties	Description
log_group	Yes	string		{'default': '/var/log/messages'}	finklabs specific configuration from baseami.
settings_file	Yes	string		{'default': 'settings.json'}	validate the tool configuration against openapi spec.
stack_output_file	Yes	string		{'default': 'stack_output.yml'}	validate the tool configuration against openapi spec.
validate	Yes	boolean		{'default': True}	use this if you need to switch off config validation.

10.2.5 foldersItem - structure

Name	Required	Type	Format	Properties	Description
source	Yes	string			
target	No	string			

CHAPTER 11

lambda command

`lambda` is finks AWS Lambda deployment tool.

11.1 Related documents

11.2 Usage

To see available commands, call this:

```
Usage:
        lambda clean
        lambda bundle [--keep] [-v]
        lambda deploy [--keep] [-v]
        lambda list
        lambda metrics <lambda>
        lambda info
        lambda wire [-v]
        lambda unwire [-v]
        lambda delete [-v] -f <lambda> [--delete-logs]
        lambda rollback [-v] <lambda> [<version>]
        lambda ping [-v] <lambda> [<version>]
        lambda invoke [-v] <lambda> [<version>] [--invocation-type=<type>] --payload=
↪<payload> [--outfile=<file>]
        lambda logs <lambda> [--start=<start>] [--end=<end>] [--tail]
        lambda version

Options:
-h --help               show this
-v --verbose            show debug messages
--keep                  keep (reuse) installed packages
--payload=payload       '{"foo": "bar"}' or file://input.txt
--invocation-type=type  Event, RequestResponse or DryRun
--outfile=file          write the response to file
```

```
--delete-logs           delete the log group and contained logs
--start=start           log start UTC '2017-06-28 14:23' or '1h', '3d', '5w', ...
--end=end               log end UTC '2017-06-28 14:25' or '2h', '4d', '6w', ...
--tail                  continuously output logs (can't use '--end'), stop 'Ctrl-C'
```

11.2.1 clean

removes local bundle files.

11.2.2 bundle

zips all the files belonging to your lambda according to your config and requirements.txt and puts it in your current working directory as `bundle.zip`. Useful for debugging as you can still provide different environments.

11.2.3 deploy

Deploy an AWS Lambda function to AWS. If the lambda function is non-existent it will create a new one.

For an existing lambda function lambda checks whether the hashcode of the bundle has changed and updates the lambda function accordingly. This feature was added to lambda so we are able to compare the hashcodes locally and save time for bundle uploads to AWS.

This only works if subsequent deployments are executed from the same virtualenv (and same machine). The current implementation of the fink-bundler starts every deployment with a fresh virtualenv. If you want the hashcode comparison you need to provide the `--keep` option. With the '–keep' option the virtualenv is preserved. Otherwise the hashcodes of the lambda code bundles will be different and the code will be deployed.

If you can not reuse ('–keep') the virtualenv for example in case you deploy from different machines you need to use `git` to check for code changes and skip deployments accordingly.

In any case configuration will be updated and an alias called "ACTIVE" will be set to this version.

11.2.4 list

lists all existing lambda functions including additional information like config and active version:

```
dp-dev-store-redshift-create-cdn-tables
    Memory: 128
    Timeout: 180
    Role: arn:aws:iam::644239850139:role/lambda/dp-dev-store-redshift-cdn-
↪LambdaCdnRedshiftTableCr-G7ME657RXFDB
    Current Version: $LATEST
    Last Modified: 2016-04-26T18:03:44.705+0000
    CodeSha256: KY0Xk+g/Gt69V0siRhgaG7zWbg234dmb2hoz0NHIa3A=
```

11.2.5 metrics

displays metric for a given lambda:

```
dp-dev-ingest-lambda-cdnnorm
    Duration 488872443
    Errors 642
    Invocations 5202
    Throttles 13
```

11.2.6 wire

"wires" the AWS Lambda function to an event source.

11.2.7 unwire

delets the event configuration for the lambda function

11.2.8 delete

deletes a lambda function

If you use the `--delete-logs` the cloudwatch log group associated to the AWS Lambda function is deleted including log entries, too. This helps to save cost for items used in testing.

11.2.9 rollback

sets the active version to ACTIVE -1 or to a given version

11.2.10 invoke

In this section, you invoke your Lambda function manually using the `lambda invoke` command.

```
$ lambda invoke my_hello_world \
--invocation-type RequestResponse \
--payload '{"key1":"value1", "key2":"value2", "key3":"value3"}'
```

If you want you can save the payload to a file (for example, input.txt) and provide the file name as a parameter:

```
$ lambda invoke my_hello_world \
--invocation-type RequestResponse \
--payload file://input.txt
```

The preceding invoke command specifies RequestResponse as the invocation type, which returns a response immediately in response to the function execution. Alternatively, you can specify Event as the invocation type to invoke the function asynchronously.

11.2.11 logs

The `lambda logs` command provides you with convenient access to log events emitted by your AWS Lambda function.

The command offers '–start' and '–end' options where you can filter the log events to your specification. You can use human readable dates like '2017-07-24 14:00:00' or you can specify dates in the past relative to `now` using '1m', '2h', '3d', '5w', etc.

```
$ lambda logs ops-dev-captain-crunch-slack-notifier --start=1d
MoinMoin fin0007m!
2017-07-07
07:00:32  START RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8 Version: $LATEST
07:00:36  END RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8
07:00:36  REPORT RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8        Duration:␣
↪4221.50 ms    Billed Duration: 4300 ms        Memory Size: 128 MB     Max Memory␣
↪Used: 43 MB
Bye fin0007m. Talk to you soon!
```

The '–start' option has a default of '1d'. This means if you run `lambda logs <your-function-name>` you get the log output of your function for the last 24 hours.

```
$ lambda logs ops-dev-captain-crunch-slack-notifier
MoinMoin fin0007m!
2017-07-07
07:00:32  START RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8 Version: $LATEST
07:00:36  END RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8
07:00:36  REPORT RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8        Duration:␣
↪4221.50 ms    Billed Duration: 4300 ms        Memory Size: 128 MB     Max Memory␣
↪Used: 43 MB
Bye fin0007m. Talk to you soon!
```

You can use `lambda logs` to **tail** the log output of your lambda function. The default start date in tail mode is 5 minutes before. You can specify any past start date in tail mode but you can not specify an '–end' option in tail mode. To exit the `lambda logs` tail mode use `Ctrl-C`.

```
$ lambda logs ops-dev-captain-crunch-slack-notifier --start=1d --tail
MoinMoin fin0007m!
Use 'Ctrl-C' to exit tail mode
2017-07-07
07:00:32  START RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8 Version: $LATEST
07:00:36  END RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8
07:00:36  REPORT RequestId: f75cd7de-62e1-11e7-937d-ef5726c6f5c8        Duration:␣
↪4221.50 ms    Billed Duration: 4300 ms        Memory Size: 128 MB     Max Memory␣
↪Used: 43 MB
^CReceived SIGINT signal - exiting command 'lambda logs'
```

11.2.12 version

will print the version of fink you are using

11.3 Folder Layout

11.4 Sample config file

sample fink_dev.json file:

```
{
  "lambda": {
    "lambda": {
      "name": "dp-dev-store-redshift-load",
      "description": "Lambda function which loads normalized files into redshift",
      "role": "arn:aws:iam::644239850139:role/lambda/dp-dev-store-redshift-cdn-lo-
↪LambdaCdnRedshiftLoad-DD2S84CZFGT4",
      "handlerFunction": "handler.lambda_handler",
      "handlerFile": "handler.py",
      "timeout": "180",
      "memorySize": "128",
      "events": [
        {
          "event_source": {
            "arn":  "arn:aws:s3:::my-bucket",
            "events": ["s3:ObjectCreated:*"]
          }
        },
        {
          "event_source": {
            "name": "send_reminder_to_slack",
            "schedule": "rate(1 minute)"
          }
        }
      ],
      "vpc": {
        "subnetIds": [
          "subnet-87685dde",
          "subnet-9f39ccfb",
          "subnet-166d7061"
        ],
        "securityGroups": [
          "sg-ae6850ca"
        ]
      }
    },
    "bundling": {
      "zip": "bundle.zip",
      "preBundle": [
        "../bin/first_script.sh",
        "../bin/second_script.sh"
      ],
      "folders": [
        {
          "source": "../redshiftcdnloader",
          "target": "./redshiftcdnloader"
        },
        {
          "source": "psycopg2-linux",
          "target": "psycopg2"
        }
      ]
    },
    "deployment": {
      "region": "eu-west-1",
      "artifactBucket": "7finity-$PROJECT-deployment"
    }
  }
```

```
}
```

11.5 lambda configuration as part of the fink_<env>.json file

11.5.1 log retention

Possible values for the log retention in days are: 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, and 3653.

```
{
    "lambda": {
        ...
        "logs": {
            "retentionInDays": 90
        }
}
```

11.5.2 S3 upload

lambda can upload your lambda functions to S3 instead of inline through the API. To enable this feature add this to the "lambda" section of your `fink_<env>.json` config file:

```
"deployment": {
    "region": "eu-west-1",
    "artifactBucket": "7finity-$PROJECT-deployment"
}
```

You can get the name of the bucket from Ops and it should be part of the stack outputs of the base stack in your account (s3DeploymentBucket).

11.5.3 runtime support

fink supports the `nodejs4.3`, `nodejs6.10`, `python2.7`, `python3.6` runtimes.

Add the runtime config to the `lambda` section of your fink configuration.

```
    "runtime": "nodejs4.3"
```

At this point the following features are implemented:

- install dependencies before bundling (dependencies are defined in package.json)
- bundling (bundle the lambda function code and dependencies)
- deployment (the nodejs4.3 lambda function is setup with the nodejs4.3 runtime)
- configuration (bundles `settings_<env>.conf` file for your environments)
- nodejs support is tested by our automated fink testsuite
- if no runtime is defined fink uses the default runtime `python2.7`

Note: for this to work you need to **have npm installed** on the machine you want to run the lambda bundling!

11.5.4 AWS Lambda environment variables

Ramuda supports AWS Lambda environment variables. You can specify them within the `lambda` section.

```
    ...
    "environment": {
        "MYVALUE": "FOO"
    }
```

More information you can find in AWS docs.

11.5.5 Adding a settings.json file

Ramuda supports a settings section. If used a `settings.json` file is added to the zip bundle. You can specify the settings within the `lambda` section.

```
    ...
    "settings": {
        "MYVALUE": "FOO"
    }
```

You can use lookups like for the rest of the configuration. Note that the values are looked up BEFORE the AWS Lambda function is deployed. If values change during the AWS Lambda function lifecycle it does not recognise the changes. For values that must be updated you should lookup the values in your code using for example credstash.

```
    ...
    "settings": {
        "accountId": "lookup:stack:infra-dev:AWSAccountId"
    }
```

11.5.6 Adding event configuration

fink can be used to easily schedule functions to occur on regular intervals. Just list your expressions to schedule them using cron or rate syntax in your fink_<env>.json config file like this:

```
...
"events": [{
    "event_source": {
        "name": "send_reminder_to_slack",
        "schedule": "rate(1 minute)"
    }
}]
```

The schedule expression defines when to execute your lambda function in cron or rate format.

Supported event types.

Similarly, you can have your functions execute in response to events that happen in the AWS ecosystem, such as S3 uploads, Kinesis streams, and SNS messages, etc..

In your fink_<env>.json config file, define your event sources. The following sample config will execute your AWS Lambda function in response to new objects in your my-bucket S3 bucket. Note that your function must accept event and context parameters.

```
...
"events": [{
    "event_source": {
        "arn":  "arn:aws:s3:::my-bucket",
        "events": ["s3:ObjectCreated:*"],
        "suffix": ".jpg"
    }
}],
```

Similarly, for a Simple Notification Service (SNS) event:

```
...
"events": [{
    "event_source": {
        "arn":  "arn:aws:sns:::your-event-topic-arn",
        "events": ["sns:Publish"]
    }
}]
```

Kinesis is slightly different as it is not event-based but pulling from a stream:

```
...
"events": [{
    "event_source": {
        "arn": arn:aws:kinesis:eu-west-1:1234554:stream/your_stream"
        "starting_position": "TRIM_HORIZON",
        "batch_size": 50,
        "enabled": true
    }
}]
```

Lambda@Edge needs a CloudFront event trigger.

```
...
"events": [{
    "event_source": {
        "arn": "arn:aws:cloudfront::420189626185:distribution/E1V934UN4EJGJA",
        "cache_behavior": "*",
        "cloudfront_event": "origin-request"
    }
}]
```

11.6 Deploying AWS Lambda@Edge

AWS Lambda@Edge is relatively new so we have to deal with some (hopefully temporary) limitations (like it is only available in Virginia):

- AWS Lambda@Edge functions must be deployed to 'us-east-1' region
- can not use artifact bucket for upload
- max memory use 128 MB
- max timeout 3 seconds for 'origin request' and 'origin response' events; for 'viewer request' and 'viewer response' events timeout is 1 second

- lambda@edge does not implement ALIAS or $LATEST so we use the version-nbr of the last published version of the lambda function for wiring
- unwire removes the lambda trigger for the configured CloudFront distribution (regardless if the lambda function is the same as the configured one or not)
- 'lambda wire' and 'lambda unwire' finish after initiation of the replication to CloudFront. This means CloudFront distribution is in 'Pending' state when fink.lambda exits.
- you cannot delete lambda functions that have been replicated to CloudFront edge locations

11.7 Setting the ENV variable

You you need to set an environment variable "ENV" which indicates the account/staging area you want to work with. This parameter tells the tools which config file to use. For example if you want to set the environment variable ENV to 'DEV' you can do that as follows:

```
export ENV=DEV
```

11.8 Environment specific configuration for your lambda functions

Please put the environment specific configuration for your lambda function into a `fink_<env>.json` file. For most teams a useful convention would be to maintain at least 'dev', 'qa', and 'prod' envs.

11.9 Defining dependencies for your NodeJs lambda function

A sample `package.json` file to that defines a dependency to the `1337` npm module:

```
{
  "name": "my-sample-lambda",
  "version": "0.0.1",
  "description": "A very simple lambda function",
  "main": "index.js",
  "dependencies": {
    "1337": "^1.0.0"
  }
}
```

11.10 Sample NodeJs lambda function

From using lambda extensively we find it a good practise to implement the `ping` feature. With the ping `ramdua` automatically checks if your code is running fine on AWS.

Please consider to implement a `ping` in your own lambda functions:

```
var l33t = require('1337')


exports.handler = function(event, context, callback) {
    console.log( "event", event );
```

```
    if (typeof(event.lambda_action) !== "undefined" && event.lambda_action == "ping")
↪{
        console.log("respond to ping event");
        callback(null, "alive");
    } else {
        console.log(l33t('finklabs rocks!'));  // 910m3x r0ck5!
        callback();  // success
    }
};
```

CHAPTER 12

api command

`api` is finks API Gateway deployment tool.

12.1 Related documents

12.2 Usage

To see available commands, call this:

```
Usage:
        api deploy [-v]
        api delete -f [-v]
        api export [-v]
        api list [-v]
        api apikey-create <keyname> [-v]
        api apikey-list [-v]
        api apikey-delete [-v]
        api custom-domain-create [-v]
        api version

-h --help           show this
-v --verbose        show debug messages
```

12.2.1 deploy

creates/updates an API from a given swagger file

12.2.2 export

exports the API definition to a swagger file

12.2.3 list

lists all existing APIs

12.2.4 apikey-create

creates an API key

12.2.5 apikey-list

lists all existing API keys

12.2.6 apikey-delete

deletes an API key

12.2.7 version

will print the version of fink you are using

12.3 Folder Layout

`swagger.yaml` -> API definition in swagger with API Gateway extensions

```
{
    "api": {
        "api": {
            "name": "dp-dev-serve-api-2",
            "description": "description",
            "targetStage": "dev",
            "apiKey": "xxx",
            "cacheClusterEnabled": true
            "cacheClusterSize": "0.5"
            "methodSettings": {
                "/path/to/resource/GET": {
                    "cachingEnabled": false
                }
            }
        }
    },
    "lambda": {
        "lambda": {

            "entries": [
              {
                "name": "dp-dev-serve-api-query",
                "alias": "ACTIVE"
              },
              {
                "name": "dp-dev-serve-api-query-elasticsearch",
                "alias": "ACTIVE"
```

```
                }
            ]
        ...
        }
    }
}
```

Set the config attribute `cacheClusterEnabled` to `true` in your fink_<env>.json config file to enable a cache cluster for the specified stage resource.

Set the config attribute `cacheClusterSize` to '0.5'|'1.6'|'6.1'|'13.5'|'28.4'|'58.2'|'118'|'237' in your fink_<env>.json config file to configure the size for an enabled cache cluster. Default setting is '0.5'.

The config attribute `methodSettings` allows you to define settings related to a setting_key. A `setting_key` is defined as <resource_path>/<http_method>. So it is important that your setting_key contains the http_method (GET, PUT, OPTIONS, etc.), too. You can specify method setting properties as defined in the AWS docs: https://botocore.readthedocs.io/en/latest/reference/services/apigateway.html#APIGateway.Client.update_stage like for example 'cachingEnabled', 'loggingLevel', etc.

12.3.1 Create custom domain

Currently the certificates need to be deployed in `us-east-1` and used in the `certificateArn` in the `customDomain` section. If you use ACM lookup (fink-lookups) to lookup your certificate arn for api it uses `us-east-1` already.

```
"customDomain": {
  "basePath": "",
  "certificateName": "wildcard.finklabs.com-2017-3-2",
  "certificateArn": "lookup:acm:*.infra.finklabs.cloud",
  "domainName": "unittest-fink-sample-api-dev-eu-west-1.dev.mes.finklabs.cloud",
  "hostedDomainZoneId": "lookup:stack:infra-dev:internalDomainHostedZoneID",
  "route53Record": "unittest-fink-sample-api-dev-eu-west-1.dev.infra.finklabs.cloud"
}
```

12.3.2 Setting the ENV variable

You you need to set an environment variable "ENV" which indicates the account/staging area you want to work with. This parameter tells the tools which config file to use. For example if you want to set the environment variable ENV to 'DEV' you can do that as follows:

```
export ENV=DEV
```

CHAPTER 13

api tool config 1.0.1

13.1 Description

Documentation of the config file format for *api* (a fink tool). Note: if you want to add to the documentation please edit the openapi_api_gateway.yaml file!

13.2 Data Structures

13.2.1 api - structure

The api tool config is organized into the following structure:

Name	Required	Type	Format	Properties	Description
api	Yes	*api_def*			
customDomain	No	*customDomain*			
lambda	No	*lambda*			

13.2.2 api_def - structure

Use the *api* section to configure API Gateway details.

Name	Required	Type	Format	Properties	Description
apiKey	Yes	string			
cacheClusterEnabled	No	boolean			Enables a cache cluster for the Stage resource specified in the input
cacheClusterSize	No	string			Specifies the cache cluster size for the Stage resource specified in the input, if a cache cluster is enabled (defaults to '0.5')
description	Yes	string			The description for the Deployment resource to create
methodSettings	No	*methodSettings*			A map that defines the method settings for a Stage resource
name	Yes	string			The name of the Stage resource for the Deployment resource to create
targetStage	Yes	string			The name of the Stage resource for the Deployment resource to create

Methodsettings schema:

A map that defines the method settings for a Stage resource

13.2.3 customDomain - structure

Configuration necessary to setup a custom domain for the api.

Name	Required	Type	Format	Properties	Description
basePath	Yes	string			
certificateArn	Yes	*arn*			The reference to an AWS-managed certificate. AWS Certificate Manager is the only supported source
certificateName	Yes	string			The user-friendly name of the certificate
domainName	Yes	string			The name of the DomainName resource
hostedDomainZoneId	Yes	string			
route53Record	Yes	string			

13.2.4 lambda - structure

Name	Required	Type	Format	Properties	Description
entries	No	array of *entries*			

13.2.5 entries - structure

Name	Required	Type	Format	Properties	Description
alias	Yes	string			
name	Yes	string			
swaggerRef	No	string			

13.2.6 arn - structure

Amazon Resource Name

Name	Required	Type	Format	Properties	Description
arn	No	string			Amazon Resource Name

CHAPTER 14

Plugins for fink

14.1 Introduction

Plugins are a way to add features to fink without having to directly modify the fink core.

This fink plugin userguide aims to make it easy for you to get started using plugins in your projects. fink plugins help you customize the fink tools towards your specific project needs.

fink plugins are available for many different areas from reading your specific configuration format, to looking up credentials from your secret-store. fink plugins were implemented internally at finklabs.

finklabs – The Global Media Exchange – is a provider of a global, open marketplace for premium video content as well as a technical service provider for the entire value chain of online video marketing.

fink plugins and userguide are released under MIT License.

The fink plugins userguide starts with this introduction, then provides an overview on how to use and configure fink plugins in general. The following parts each cover one fink plugin.

This user guide assumes that you know fink and the AWS services you want to automate so we do not cover AWS services in great detail and instead point to relevant documentation. But even if you are starting out on AWS, fink will help you to quickly leave the AWS webconsole behind and to move towards infrastructure-as-code.

14.1.1 Related documents

This section aims to provide to you a list of related documents that will be useful to gain a detailed understanding about what the fink tool suite does. With this background you will be able to tap into the full potential of the fink tools.

14.2 Overview

This sections provides and overview on the fink plugin system. It covers everything necessary to understand how it works at a high level.

14.2.1 Plugin system key functions

Main functionality of the fink plugin system is to provide means so fink can be customized and extended without changing the core functionality. This is necessary for example in case not everybody uses slack or datadog. In this situation one can just not use the plugin or use a plugin which supports an alternate service.

Plugins are added to fink via python packages. The following section covers how to do that.

The fink plugin mechanism also encapsulates the plugin code in a way that it is separated from fink core. This enables us to change and test a plugin as a component independent from fink core. More details about the `plugin mechanism` are covered in the next chapter.

14.2.2 Plugin installation

Plugins are maintained as standard python packages. Just install plugins you want via `pip install <plugin_name>`. Same idea applies to removing plugins from a project setup. Using `pip uninstall <plugin_name>` removes the plugin.

A more sophisticated way of doing that which goes well with CI/CD is to simply add your fink plugins to your projects `requirements_fink.txt` file. Especially if you need more tools and plugins this makes setting up your CI environment easy and reproducible. `pip install -r requirements.txt -r requirments_dev.txt` installs all the packages you need for your service and for developing it.

14.2.3 Plugin configuration

Configuration for a plugin are specific for that plugin so please consult the plugins documentation for specific configuration options. General mechanism is that you add the configuration for the plugin to your `fink_<env>.json` file. Add a section with the plugin name like in the following snippet:

```
...
'plugins': {
    ...
    'fink.slack_integration': {
        'slack_webhook': 'lookup:secret:slack.webhook:CONTINUE_IF_NOT_FOUND'
    },
    ...
}
```

14.2.4 Plugin descriptions

The following table lists the plugins and gives a brief overview what each plugin is used for.

Plugin | Description —— | ——— fink.config_reader | read configuration files in json, python, or yaml format fink.lookup | lookup information related to your AWS account fink.bundler | create code bundles for code and lambda. fink.say_hello | simple plugin to demonstrate how plugins work / are developed fink.slack_integration | send deployment status information to slack

Please refer to detailed plugin's documentation later in this document folder for detailed information about that plugin.

Later we are going to put plugins in separate repositories so they can have independent owners and development / release cycles. With that we move the detailed plugin documentation to the plugin README and documentation.

14.3 fink plugin mechanism

The previous chapter gave an overview on the fink plugin system so please make sure that you have read through that one. This section goes into more detail in how the fink plugin mechanism works. So you can customize plugins or even write new ones.

fink plugins are standard python packages which are installed separately. How to do this is covered in the previous chapter plugin overview. To understand how the plugin mechanism works one must know about the `fink lifecycle` which is covered in the next section.

If a fink command is entered on the command line things are processed in the following order:

- Python interpreter loads the fink package
- CLI options and arguments are parsed
- we check if any relevant plugins are installed (details in `plugin entry_points`)
- relevant plugins are loaded and check if they comply to fink plugin structure.
- register() function of each plugin is called.
- then the fink lifecycle is executed
- each lifecycle step fires an event which we call `signal`

14.3.1 Anatomy of a plugin

Each fink plugin must implement `register()` and `deregister()` functions to be a valid fink-plugin that can be used. Note how the `register()` function connects the plugin function `say_hello` with the `initialized` lifecycle step. `deregister()` just disconnects the plugin functionality from the fink lifecycle.

```
def say_hello(context):
    """
    :param context: The boto_session, etc.. say_hello plugin needs the 'user'
    """
    print('MoinMoin %s!' % context.get('user', 'to you'))

...

def register():
    """Please be very specific about when your plugin needs to run and why.
    E.g. run the sample stuff after at the very beginning of the lifecycle
    """
    fink_signals.initialized.connect(say_hello)
    fink_signals.finalized.connect(say_bye)


def deregister():
    fink_signals.initialized.disconnect(say_hello)
    fink_signals.finalized.disconnect(say_bye)
```

Handing in information to your plugin functions. Look into fink.fink_signals for details.

```
def my_plug_function(params):
    """
    :param params: context, config (context - the env, user, _awsclient, etc..
                   config - The stack details, etc..)
    """
```

```
    context, config = params
    # implementation here
    ...
```

All fink lifecycle steps provide the `(context, config)` tuple besides `initialized` and `finalized`. These two only provide the context.

The same lifecycle & signals mechanism applies to fink hooks. So if you ever wondered how fink hooks are work - now you know.

14.3.2 Overview of the fink lifecycle

The fink lifecycle is the essential piece of the fink tool core. It is like the clockwork of a watch. The fink lifecycle makes sure that everything is executed in the right order and everything works together like commands, hooks, plugins, etc.

The fink lifecycle is generic. This means the fink lifecycle is the same for each and every fink tool. But it is possible that a tool does not need a certain lifecycle step to it just skips it. For example there is no bundling for cloud(, yet?).

The coarse grained fink lifecycle looks like that:

If during processing of a lifecycle an error occurs then the processing stops.

14.3.3 List of fink signals

The list of fink signals you can use in plugins or hooks:

- initialized - after reading arguments and context
- config_read_init
- config_read_finalized
- check_credentials_init
- check_credentials_finalized
- lookup_init
- lookup_finalized
- config_validation_init
- config_validation_finalized
- bundle_pre - we need this signal to implement the prebundle-hook
- bundle_init
- bundle_finalized
- command_init
- command_finalized
- error

- finalized

The order of this list also represents the order of the lifecycle steps with in the fink lifecycle.

14.3.4 Developing plugins

If you want to develop a plugin to integrate some service or to optimize the configuration for your environment we recommend that you "fork" the say_hello plugin so you have the right structure and start from there.

If you need help developing your plugin or want to discuss your plans and get some feedback please don't be shy. The SRE squad is here to help.

14.3.5 Testing a plugin

Testing a fink plugin should be easy since its code is decoupled from fink core. It is a good practice to put the tests into the `tests` folder. Also please prefix all your test files with `test_` in this way pytest can pick them up. Please make sure that your plugin test coverage is on the save side of 80%.

14.4 Overview on configuration

Configuration and configuration file formats are very dear to us and we already know to you, too. For example a team likes to use multiple tool specific hocon files so this functionality is provided by a team specific `team.config_reader`. Other teams like to have all the configuration for a service environment in one single config file (fink.config_reader).

Regarding configuration file formats many things are possible and it should be very simple to implement your specific requirements as a plugin so you could use XML or INI format. Talk to us and do not shy away from good old json config.

14.4.1 Structure of the configuration (internal representation)

The datastructure used by fink internally to represent the configuration is basically json compatible.

We have configuration on the following levels:

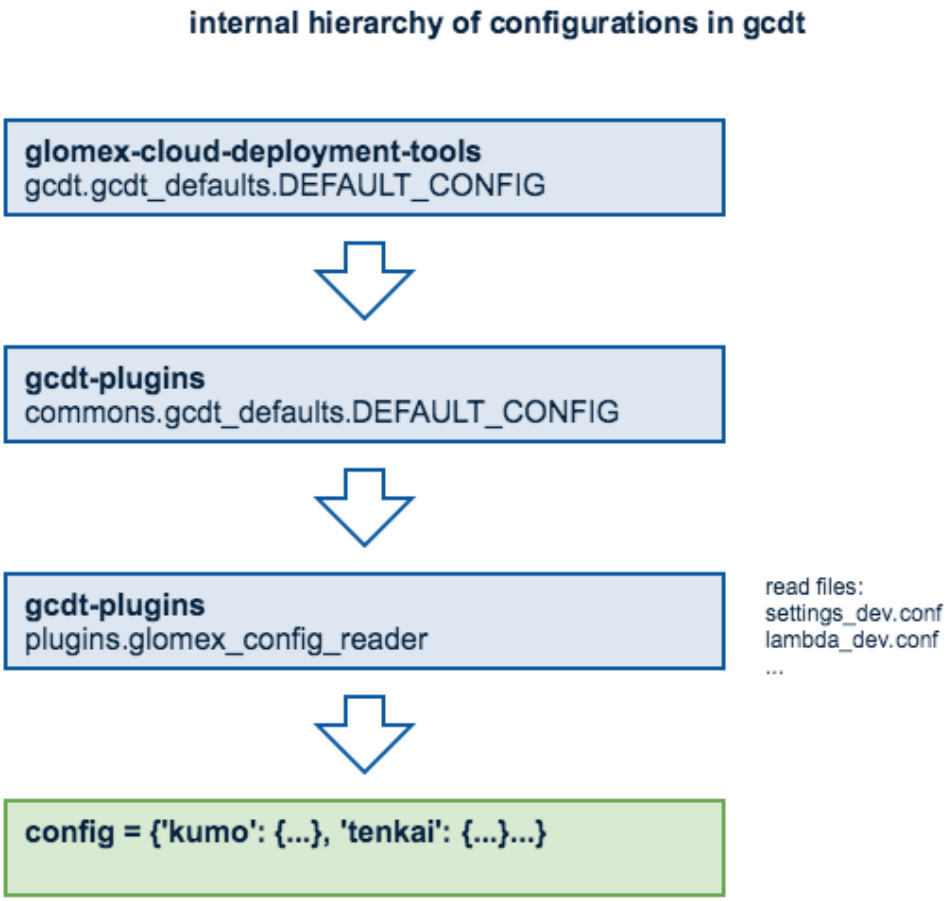
- top-level
- tool-level (tools are cloud, code, lambda, api)
- plugin specific configuration

```
{
    'cloud': {
        ...
    },
    'code': {
        ...
    },
    'lambda': {
        ...
    },
    'api': {
        ...
    },
```

```
    'plugins' {
        'plugin_a': {
            ...
        },
        ...
    }
}
```

14.4.2 Multiple levels of fink configuration

Configuration is assembled in multiple levels:



The multiple levels of configurations represent different stages in the lifecycle process. This allows to have a very generic "catch-all" configuration but to override this configuration in specific cases when we have more specific information. Like when using a plugin. For example the fink.config_reader looks for `.json` config files.

14.4.3 Context

The fink `context` is the "internal" datastructure fink is using to process the CLI command that need to be executed. So for instance each plugin or hook can find out about the `tool`, `command`, or `env` it is currently processing. With

the context we follow the convention to prefix private attributes with an '_' like with the `_awsclient` that plugins use to access AWS services but `_awsclient` does not show up in the slack notification.

```
{
  'version': '0.1.426',
  'command': 'preview',
  '_awsclient': <fink.fink_awsclient.AWSClient object at 0x10291a490>,
  'env': 'dev',
  '_arguments': {
    '--override-stack-policy': False,
    '-f': False,
    'delete': False,
    'deploy': False,
    'dot': False,
    'generate': False,
    'list': False,
    'preview': True,
    'version': False
  },
  'tool': 'cloud',
  'plugins': [
    'fink.config-reader',
    'fink.bundler',
    'fink.say-hello',
    'fink.slack-integration',
    'fink.doctor',
    'fink.gru',
    'fink.lookups'
  ],
  'user': 'markfink'
}
```

14.5 fink.config-reader plugin

Read config from files in json, python or yaml format. There is a section called `overview on configuration` above. Please make sure you have that one covered.

14.5.1 Related documents

14.5.2 json configuration files

The fink.config_reader plugin allows us to have configurations in json format.

The configuration files are environment specific. This means the config file looks like fink_<env>.json' where stands for the environment you use (some thing like dev, stage, prod, etc.).

14.5.3 yaml configuration files

The fink.config_reader plugin allows us to have configurations in `yaml` format.

The configuration files are environment specific. This means the config file looks like fink_<env>.yaml' where stands for the environment you use (some thing like dev, stage, prod, etc.).

14.5.4 python configuration files

The fink.config_reader plugin allows us to have configurations in python format (with a .py extension).

The configuration files are environment specific. This means the config file looks like fink_<env>.py‘ where stands for the environment you use (some thing like dev, stage, prod, etc.).

The python configuration files have a few specifics which are documented here.

The python configuration only work if they follow the convention to implement a `generate_config()` function. The `generate_config()` needs to return a dictionary with the configuration. Please follow the configuration structure described below.

```
def generate_config():
    return CFG
```

You can also use hooks in the python config files. Just implement the `register`, `deregister` like they are described in the plugin section to make that work.

14.5.5 finkignore patterns

fink supports multiple ways of ignoring files from bundling. The file format of pattern files is the same as gitignore files. This means you can use wildcards to exclude files from bundling like for example `*.pyc`. This is currently relevant for the fink tools `code` and `lambda`.

The following files are supported:

- .lambdaignore - in the user home directory
- .finkignore - in the current directory
- .npmignore - in the current directory

Alternatively you can provide the ignore pattern with your `fink_<env>.json` config file:

```
{
    "finkignore": ["*.pyc", "trash"]
    ...
}
```

On a more technical note: all ignore patterns you provide are consolidated by the config reader and provided to plugins and tools as `config['finkignore']`.

14.5.6 reference to base config file

The fink.config_reader plugin supports a `baseconfig` property which gives a basepath. This works with all supported config file formats like json, yaml and .py config files.

```
{
  "baseconfig": "baseconfig",
  "lambda": {
    "lambda": {
      "runtime": "nodejs4.3",
  ...
```

With that you can for example implement the following config structure with your config files:

```
baseconfig.json
-> fink_dev.json
-> fink_stage.json
```

In this sample `baseconfig.json` contains all common config values for all environments. `fink_dev.json` contains only values specific for the development environment (same with `fink_stage.json`). Config values can be overridden by the dependent `fink_<env>.json` file, too.

14.6 fink.lookups plugin

The lookups functionality was previously part of the hocon config reader. The lookup functionality was refactored into this `fink-lookups` plugin and with the refactoring we also pinned the functionality it into a dedicated lifecycle step.

14.6.1 Related documents

14.6.2 lookup stack output

The `stack` lookup is used to substitute configuration where the value is an output from another cloudformation stack.

format: `lookup:stack:<stackname>:<output>` sample: `lookup:secret:slack.token`

14.6.3 lookup acm certificate

format: `lookup:acm:<name_1>:...:<name_n>:` sample: `lookup:acm:foo.mes.finklabs.cloud:supercars.infra.finklabs.cloud:*.dev.infra.finklabs.cloud`

'acm' lookup uses the AWS ACM (Certificate Manager) functionality. It is configured as default lookup.

Features of the `acm lookup`:

- pass a list of hostnames that should be secured.
- check all certificates in ACM if the configured CN (DomainName) or SANs (SubjectAlternativeNames) (including wildcards) if they match for the given list of hostnames
- the chosen certificates STATUS must be **ISSUED**
- if there are multiple matches, use the one with the most distant expiry date
- return the ARN of the certificate
- wildcards for hosted zone are expressed with "*."
- 'ERROR' in case a certificate matching the specified list of names can not be found

Note: if you use ACM lookup in api / API Gateway you need to deploy the certificates to the `us-east-1` region.

14.6.4 lookup secret

The `secret` lookup is used to substitute configuration where the value is a password, or other sensitive information that you can not commit to a sourcecode repository. The keys are stored in credstash (DynamoDB + KMS).

format: `lookup:secret:<name>.<subname>`

lookup the 'slack.webhook' entry from credstash sample: `lookup:secret:slack.webhook`

lookup the 'slack.webhook' entry from credstash sample: `lookup:secret:slack.webhook:CONTINUE_IF_NOT_FOUND`

note in the second example that the `slack.webhook` lookup does not fail it the accounts credstash does not have the `slack.webhook` entry.

more info on storing keys in AWS using credstash

14.6.5 lookup parameter

The `parameter` lookup is used to substitute configuration where the value is a password, or other sensitive information that you can not commit to a sourcecode repository. The keys are stored in AWS Simple Systems Manager (SSM) parameter store. If you want to replace credstash (see above) you can use type 'SecureString' to store your parameters encrypted. Like with credstash your encryption key is stored in KMS.

format: `lookup:parameter:<name>.<subname>`

lookup the 'slack.webhook' entry from SSM parameter store sample: `lookup:parameter:slack.webhook`

lookup the 'slack.webhook' entry from SSM parameter store sample: `lookup:parameter:slack.webhook:CONTINUE_IF_NOT_FOUND`

note in the second example the `slack.webhook` lookup does not fail it the accounts SSM parameter store does not have the `slack.webhook` entry.

more info on SSM parameter store

14.7 fink.lookups plugin config 1.0.4

14.7.1 Description

Documentation of the config file format for *fink.lookups* (a fink plugin). Note: if you want to add to the documentation please edit the openapi_lookups.yaml file

14.7.2 Data Structures

fink.lookups - structure

The plugin config is organized into the following structure:

Name	Required	Type	Format	Properties	Description
defaults	Yes	*defaults*			finetune fink tool lifecycle (override at own risk)

defaults - structure

Default properties to fine tune fink tool lifecycle (override at own risk).

Name	Required	Type	Format	Properties	Description
lookups	Yes	array of *lookup_type*		{'default': ['parameter', 'secret', 'stack', 'acm']}	define which lookups can be used for this deployment
validate	Yes	boolean		{'default': True}	use this if you need to switch off config validation.

lookup_type - structure

Name	Required	Type	Format	Properties	Description
lookup_type	No	string			

lookup_format - structure

Name	Required	Type	Format	Properties	Description
lookup_format	No	string			

14.8 fink.bundler plugin

Create code bundles for code and lambda.

14.8.1 Related documents

14.8.2 Sample config

bundling sample for cloud:

```
...
"bundling": {
    "folders": [
        { "source" = "./codedeploy", target = "." },
        { "source" = "../app", target = "app" },
        { "source" = "../images", target = "images" },
        { "source" = "../supercars", target = "supercars" }
    ]
}
```

Bundling sample for lambda:

```
...
"bundling": {
  "zip": "bundle.zip",
  "folders": [
    { "source": "./vendored", "target": "."},
    { "source": "./impl", "target": "impl"}
  ]
},
```

14.9 fink.say-hello plugin

This is probably the friendliest plugin ever. It simply greets the user.

Purpose of this plugin is to demonstrate how the fink plugin mechanism works to developers. You can use it as blueprint to jump-start developing your own plugin.

```
# -*- coding: utf-8 -*-
"""A fink plugin which demonstrates how to implement hello world as plugin."""
from __future__ import unicode_literals, print_function

from fink import fink_signals


def say_hello(context):
    """say hi.
    :param context: The boto_session, etc.. say_hello plugin needs the 'user'
    """
    print('MoinMoin %s!' % context.get('user', 'to you'))


def say_bye(context):
    """say bye.
    :param context: The boto_session, etc.. say_hello plugin needs the 'user'
    """
    print('Bye %s. Talk to you soon!' % context.get('user', 'then'))


def register():
    """Please be very specific about when your plugin needs to run and why.
    E.g. run the sample stuff after at the very beginning of the lifecycle
    """
    fink_signals.initialized.connect(say_hello)
    fink_signals.finalized.connect(say_bye)


def deregister():
    fink_signals.initialized.disconnect(say_hello)
    fink_signals.finalized.disconnect(say_bye)
```

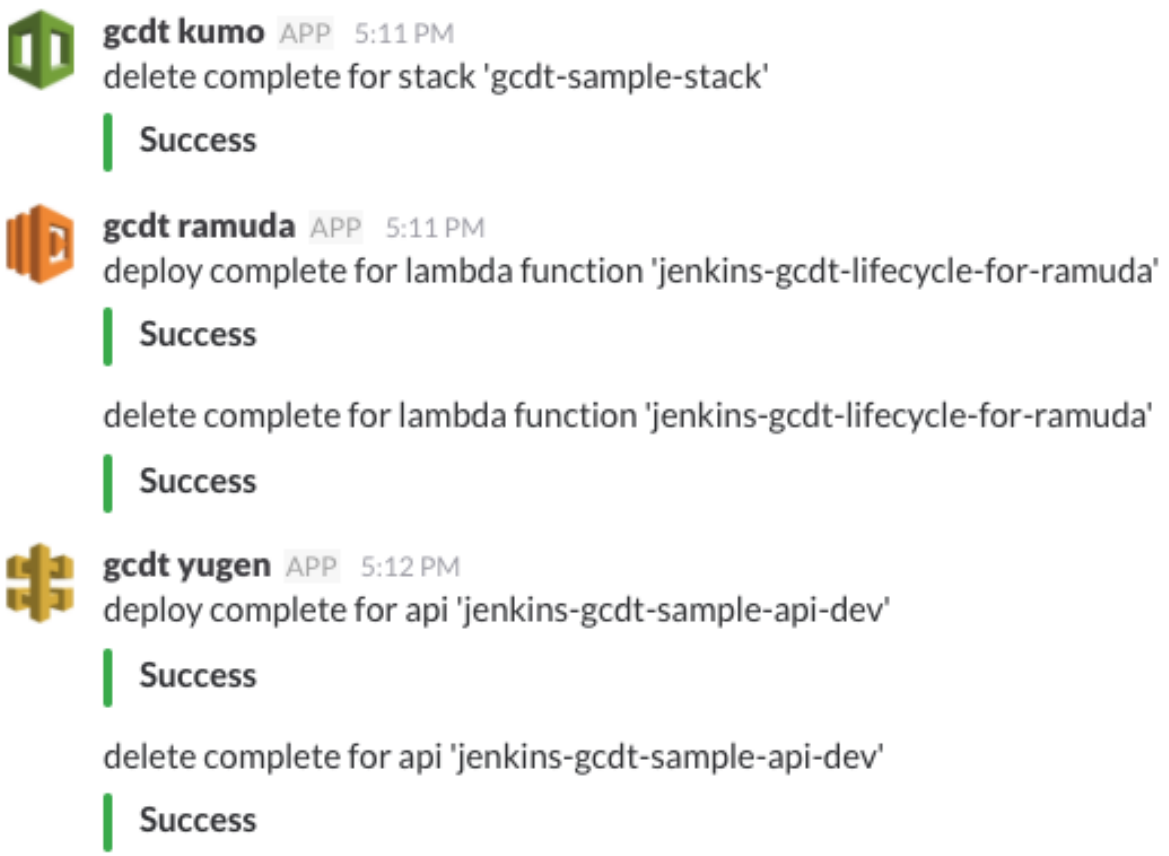
14.10 fink.slack-integration plugin

Announce the status of your deployments on slack.

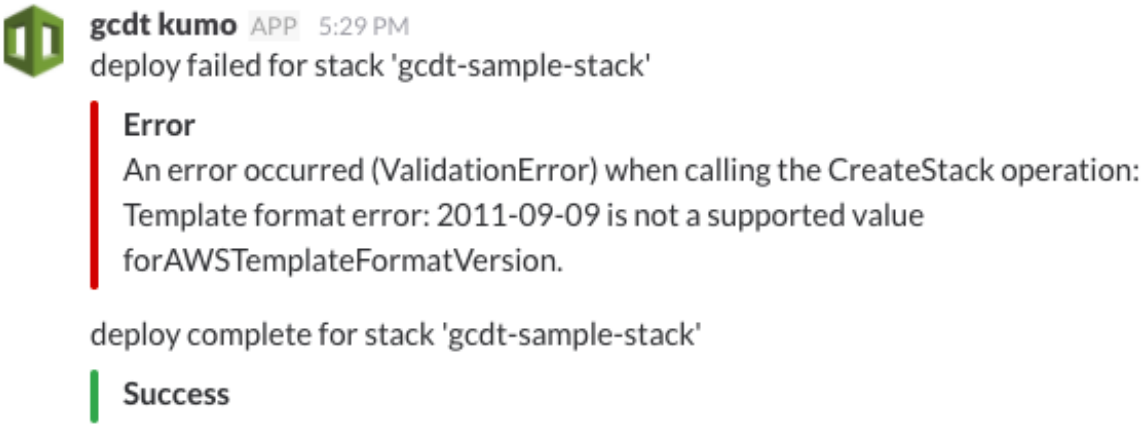
14.10.1 Related documents

14.10.2 slack integration plugin functionality

Announce deployments on the slack channel for your squad:



In case a deployments fails you get a notification, too:



14.10.3 Setup

To setup the slack integration for your account you need two things:

- a slack webhook
- you need to add the slack webhook to SSM parameter store so the `lookup:parameter:slack.webhook` works

14.10.4 Configuration

`slack.webhook` is provided via parameter lookup:

```
...
"plugins": {
    "fink.slack_integration": {
        "channel": "<my_teams_slack_channel>"
    },
    ...
}
```

Note the `slack.webhook` configuration is provided via default configuration. You do not need to change that as long as you are happy with the default config:

```
"slack_webhook": "lookup:parameter:slack.webhook:CONTINUE_IF_NOT_FOUND"
```

14.11 fink-slack-integration plugin config 1.0.2

14.11.1 Description

Documentation of the config file format for *fink.slack-integration* (a fink plugin). Note: if you want to add to the documentation please edit the openapi_slack_integration.yaml file

14.11.2 Data Structures

fink.slack_integration - structure

The plugin config is organized into the following structure:

Name	Required	Type	Format	Properties	Description
defaults	Yes	*defaults*			finetune fink tool lifecycle (override at own risk)

defaults - structure

Default properties to finetune fink tool lifecycle (override at own risk).

Name	Required	Type	Format	Properties	Description
slack_webhook	Yes	string		{'default': 'lookup:parameter:slack.webhook:CONTINUE_IF_NOT_FOUND'}	webhook for posting to slack
validate	Yes	boolean		{'default': True}	use this if you need to switch off config validation.

CHAPTER 15

Frequently Asked Questions (faq)

15.1 Homebrew Python

If you installed Python via Homebrew on OS X and get this error:

```
must supply either home or prefix/exec-prefix -- not both
```

You can find a solution on here

15.2 Python package errors

Please ensure that you have the latest version of `pip`, `setuptools` and `virtualenv`

If you have error like this:

```
pip._vendor.pkg_resources.DistributionNotFound:
```

or

```
pkg_resources.DistributionNotFound: regex==2017.6.07
```

you should update your `pip` and `virtualenv` packages

```
$ pip install -U pip
$ pip install -U virtualenv
```

15.3 Bundling error

This error is usually caused by not having installed the `fink-bundler` plugin:

```
(.python) root@:/app# AWS_PROFILE=superuser-dev ENV=qa lambda deploy
ERROR: u'_zipfile'
ERROR: u'_zipfile'
Traceback (most recent call last):
  File "/root/.python/bin/lambda", line 11, in <module>
    sys.exit(main())
  File "/root/.python/local/lib/python2.7/site-packages/fink/lambda_main.py", line␣
↪255, in main
    dispatch_only=['version', 'clean']))
  File "/root/.python/local/lib/python2.7/site-packages/fink/fink_lifecycle.py", line␣
↪195, in main
    return lifecycle(awsclient, env, tool, command, arguments)
  File "/root/.python/local/lib/python2.7/site-packages/fink/fink_lifecycle.py", line␣
↪142, in lifecycle
    raise(e)
KeyError: u'_zipfile'
```

You need to add `fink-bundler` into *requirements_fink.txt* and do:

```
$ pip install -U -r requirements_fink.txt
```

15.4 Missing configuration error

After updating `fink` to the latest version you get the following error:

```
Configuration missing for 'cloud'
```

This error appears if you used `hocon` based configs without having installed the `finklabs-config-reader` plugin. You can install it or use conf2json util (only for finklabs users) to transform your `hocon` configs into `json` one.

15.5 Environment variable error

If you run any `fink` commands (cloud, code, lambda etc) and get the following error:

```
ERROR: 'ENV' environment variable not set!
```

Environment variable "ENV" indicated the account/staging area you want to work with. This parameter tells the tools which config file to use. Please be sure that you provide the correct environment variables (ENV=PROD/DEV/etc.)

```
$ export ENV=DEV
```

15.6 Using hooks in fink

We implemented hooks in fink similar to the plugin mechanism.

You can use hooks in fink in the following places:

- use hooks in a `cloudformation.py` template
- use hooks in a `fink_<env>.py` config file

- use hooks in a `hookfile.py`. Please specify the location of the `hookfile` in your config file.

For details on fink_lifecycle and fink_signals please take a look into the fink plugins section of this documentation.

CHAPTER 16

Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

16.1 [0.1.436] - 2017-08-17

16.1.1 Fixed

- api: default value for cache size property (#328)

16.2 [0.1.435] - 2017-08-17

16.2.1 Added

- api: add cache size property (#328)

16.3 [0.1.433] - 2017-08-14

16.3.1 Added

- cloud: stop / start cloudformation stack (#342)

16.4 [0.1.432] - 2017-08-10

16.4.1 Fixed

- fink: removed explicit requests dependency (#359)

16.5 [0.1.431] - 2017-08-10

16.5.1 Added

- fink-lookups: added acm lookup documentation, do acm lookup by default (#359)

16.5.2 Deprecated

- fink-lookups: ami was already deprecated but made it more explicit + docs

16.6 [0.1.430] - 2017-08-08

16.6.1 Added

- api: implement cache settings on method level (#328)

16.7 [0.1.429] - 2017-08-07

16.7.1 Added

- cloud: fixed compatibility issue with templates in update case (#357)

16.8 [0.1.428] - 2017-08-03

16.8.1 Added

- cloud: use 'context' and 'config' in cloudformation templates (#353)
- cloud: add 'stack_output' attribute to context (#353)
- added section on version pinning to installation guide

16.9 [0.1.427] - 2017-08-01

16.9.1 Added

- api: cacheClusterEnabled setting (#328)

16.10 [0.1.426] - 2017-08-01

16.10.1 Added

- cloud: cloud_particle_helper plus docs (#244)

16.11 [0.1.425] - 2017-08-01

16.11.1 Fixed

- cloud, code: report correct deployment status on slack (#348)

16.12 [0.1.424] - 2017-08-01

16.12.1 Added

- lambda: mechanism for flexible event wiring (#264)

16.12.2 Deprecated

- lambda: 'events' config dictionary like 's3Sources' and 'timeSchedules' (#264)

16.13 [0.1.423] - 2017-07-21

16.13.1 Added

- code: improve output in case of errors (#316)

16.14 [0.1.422] - 2017-07-18

16.14.1 Fixed

- cloud: fix warning message for deprecated format (#337)

16.15 [0.1.421] - 2017-07-18

16.15.1 Fixed

- dependencies for logcapture mechanism (#285)

16.16 [0.1.420] - 2017-07-18

16.16.1 Added

- cloud: add SNS notifications (#185)
- logcapture mechanism for tests (#285)

16.16.2 Deprecated

- cloud: “cloudformation” config section, use “parameters” & “stack” instead (#337)

16.17 [0.1.419] - 2017-07-17

16.17.1 Added

- cloud: add status message for empty changeset (#126)

16.18 [0.1.418] - 2017-07-17

16.18.1 Added

- lambda & code settings files - json format (#295)

16.19 [0.1.417] - 2017-07-13

16.19.1 Added

- getting started guide (#312)

16.20 [0.1.415] - 2017-07-12

16.20.1 Fixed

- cloud update when using the artifactBucket setting (#332)

16.21 [0.1.413] - 2017-07-07

16.21.1 Added

- lambda logs command (#247)

16.22 [0.1.412] - 2017-07-05

16.22.1 Added

- fix some docu issues for lambda

16.23 [0.1.411] - 2017-07-04

16.23.1 Added

- configure cloudwatch logs for lambda (#191)

16.24 [0.1.410] - 2017-07-03

16.24.1 Fixed

- fix 'file://' prefix for lambda invoke payload (#246)

16.25 [0.1.409] - 2017-07-03

16.25.1 Added

- use roleARN for cloud delete, too (#162)

16.26 [0.1.408] - 2017-06-30

16.26.1 Added

- cloud preview for new stack (#73)

16.27 [0.1.407] - 2017-06-30

16.27.1 Fixed

- do not fail check_fink_update when PyPi is down (#313)

16.28 [0.1.406] - 2017-06-30

16.28.1 Added

- support for AWS Lambda ENV variables (#262)

16.29 [0.1.405] - 2017-06-30

16.29.1 Fixed

- minor documentation changes for cloud. Better description of usage of a role for CloudFormation (#162)

16.30 [0.1.404] - 2017-06-29

16.30.1 Added

- handle SIGTERM and SIGINT signals and stop running deployments accordingly (#40)

16.31 [0.1.403] - 2017-06-22

16.31.1 Added

- define GracefulExit exception (#40)

16.32 [0.1.402] - 2017-06-21

16.32.1 Fixed

- lambda redeploy version without changes issue (#145)

16.33 [0.1.401] - 2017-06-20

16.33.1 Fixed

- cloud artifactBucket handling (#292)

16.34 [0.1.400] - 2017-06-20

16.34.1 Fixed

- datetime handling (#226)

16.35 [0.1.399] - 2017-06-19

16.35.1 Fixed

- code clean up /tmp files (#60)

16.36 [0.1.398] - 2017-06-16

16.36.1 Added

- cloud use special role for cloudformation deployments (#162)

16.37 [0.1.397] - 2017-06-16

16.37.1 Fixed

- cloud parameter diffing without params fails (#64)
- cloud parameter diffing flaws (#184)

16.38 [0.1.396] - 2017-06-12

16.38.1 Added

- lambda invoke command (#246)

16.39 [0.1.394] - 2017-06-09

16.39.1 Added

- add stack_output.yml to code bundle artifact (#266)

16.40 [0.1.393] - 2017-06-09

16.40.1 Added

- lambda works with python3.6

16.41 [0.1.392] - 2017-06-07

16.41.1 Added

- added some documentation related to fink-bundler and AWS Lambda runtimes

16.42 [0.1.391] - 2017-06-02

16.42.1 Added

- support for nodejs6.10 + python3.6 runtimes
- specify folders (source & target) for code bundles
- lambda '–keep' option to speed up your dev cycles

16.43 [0.1.390] - 2017-05-31

16.43.1 Added

- prepare for nodejs6.10 runtime (PR 293)

16.44 [0.1.8] - 2017-04-27

16.44.1 Added

- improve exception handling (#285) plus proper error message for missing config

16.45 [0.1.7] - 2017-04-27

16.45.1 Added

- check AWS Lambda runtime in lambda config (#254)

16.46 [0.1.6] - 2017-04-26

16.46.1 Added

- getLogger helper to be used by fink plugins (#213)

16.47 [0.1.5] - 2017-04-26

16.47.1 Added

- fink plugin version info in version cmd and datadog (#250)
- use fink plugins in E2E test lifecycle (#250)

16.48 [0.1.4] - 2017-04-07

16.48.1 Added

- fink package publicly available on PyPi (#250)

16.49 [0.1.0] - 2017-04-05

16.49.1 Added

- open source on Github (#255)
- moved build jobs to new infra jenkins (#255)

16.49.2 Changed

- it is now mandatory for fink users to maintain plugin dependencies

16.50 [0.0.84] - 2017-03-30

16.50.1 Added

- support for hooks in cloudformation templates and hookfiles (#218)

16.51 [0.0.83] - 2017-03-29

16.51.1 Added

- updated fink plugin dependency

16.52 [0.0.82] - 2017-03-29

16.52.1 Added

- added scaffolding mechanism
- use MIT LICENSE (#253)

16.52.2 Fixed

- tamed greedy lookup (#258)
- cloud fixed deprecated pre_hook (#259)

16.53 [0.0.81] - 2017-03-24

16.53.1 Added

- included plugin documentation
- fink.config_reader for json config files (#218)

16.53.2 Fixed

- lambda bundle includes settings_<env>.conf file (#249)
- minor improvements from code review sessions (#230)
- missing environment is not properly handled (#248)

16.54 [0.0.80] - 2017-03-09

16.54.1 Fixed

- fixed lambda vpc config handling (#225)

16.55 [0.0.79] - 2017-03-08

16.55.1 Fixed

- cloud docu bug (#183)
- servicediscovery timestamp localization issue (#217)
- lambda bundling issue (#225)

16.56 [0.0.78] - 2017-03-06

16.56.1 Added

- moved hocon config_reader to plugin (#150)
- split fink.lookups plugin from config_reader (#150)
- improved slack plugin (webhooks, consolidated msgs) (#219)
- extracted bundle step into bundler plugin (#150)

16.57 [0.0.77] - 2017-02-20

16.57.1 Added

- moved to std python logging + activate DEBUG logs via `-v` (#175)
- std. fink lifecycle (#152)
- removed finklabs_utils as installation dependency (#152)
- cloudformation utilities need awsclient (see cloud documentation) (#152)
- plugin mechanism (#152)
- moved datadog and slack reporting functionality to fink plugins (#152)
- cmd dispatcher + testable main modules + tests (#152)
- migrated boto_session to awsclient (#152)

16.58 [0.0.76] - 2017-01-30

16.58.1 Added

- lambda replaced git short hash in target filename with sha256 (#169)
- requirements.txt and settings_<env>.conf optional for lambda (#114)
- made boto_session a parameter in credstash (#177)

16.59 [0.0.75] - 2017-01-24

16.59.1 Added

- added fink installer (#201)
- fink outdated version warning (#155)
- moved docs from README to sphinx / readthedocs (PR194)
- pythonic dependency management (without pip-compile) (#178)
- removed finklabs-utils dependency (#178)

16.59.2 Changed

- moved CHANGELOG.md to docs folder

16.60 [0.0.73] - 2017-01-09

16.60.1 Fixed

- (#194)

16.61 [0.0.64] - 2016-11-11

16.61.1 Fixed

- wrong boto client used when getting lambda arn

16.62 [0.0.63] - 2016-11-08

16.62.1 Fixed

- pre-hook fires before config is read (#165)

16.63 [0.0.62] - 2016-11-07

16.63.1 Added

- lambda pre-bundle hooks

16.63.2 Fixed

- compress bundle.zip in lambda bundle/deploy

16.64 [0.0.61] - 2016-11-07

16.64.1 Fixed

- moved build system to infra account (#160)

16.65 [0.0.60] - 2016-10-07

16.65.1 Added

- cloud now has the visualize cmd. Req. dot installation (#136).
- code now has the slack notifications (#79).- FIX moved tests to pytest to improve cleanup after tests (#119).
- cloud now has parametrized hooks (#34).

16.65.2 Fixed

- moved tests to pytest to improve cleanup after tests (#119).
- lambda rollback to previous version.
- cloud Parameter diffing does not work for aws coma-seperated inputs (#77).

- lambda fail deployment on failing ping (#113).
- moved tests to pytest to improve cleanup after tests (#119).
- speedup tests by use of mocked service calls to AWS services (#151).

16.66 [0.0.57] - 2016-09-23

16.66.1 Added

- code now supports execution of bash scripts before bundling, can be used to bundle packages at runtime.

16.66.2 Fixed

- code now returns proper exit codes when deployment fails.

16.67 [0.0.55] - 2016-09-16

16.67.1 Added

- cloud utils EBS tagging functionality (intended for post hooks)
- cloud now supports host zones as a parameter for creating route53 records

16.68 [0.0.51] - 2016-09-05

16.68.1 Fixed

- cloud parameter diff now checks if stack has parameters beforehand

16.69 [0.0.45] - 2016-09-01

16.69.1 Added

- lambda autowire functionality
- fink sends metrics and events to datadog

16.69.2 Fixed

- api will add invoke lambda permission for new paths in existing APIs

16.70 [0.0.35] - 2016-08-29

16.70.1 Added

- consolidated slack configuration to .fink files
- configuration for slack_channel

16.71 [0.0.34] - 2016-08-tbd

16.71.1 Fixed

- refactored api structure to api_main and api_core
- improved api testability and test coverage
- further improved lambda test coverage

16.72 [0.0.33] - 2016-08-18

16.72.1 Added

- fink pull request builder

16.72.2 Fixed

- refactored code structure to code_main and code_core
- improved code testability and test coverage
- refactored lambda structure to lambda_main and lambda_core
- improved lambda testability and test coverage

16.73 [0.0.30] - 2016-08-02

16.73.1 Added

- refactored cloud structure to cloud_main and cloud_core
- improved cloud testability and test coverage
- Rate limiting when preview with empty changeset (#48)

16.73.2 Removed

- cloud validate
- cloud scaffold

16.74 [0.0.29] - 2016-07-21

16.74.1 Added

- bump finklabs-utils to 0.0.11

16.74.2 Fixed

- create_stack was broken

16.75 [0.0.26] - 2016-07-19

16.75.1 Added

- cloud now supports stack policies, see README for details
- cloud now displays changes in CloudFormation template parameters

16.75.2 Fixed

- prettify output
- removed debug output
- removed some unnecessary import validations
- cloud will now exit when importing a cloudformation.py not from your current working directory

CHAPTER 17

Development of fink

17.1 Contributing

If you find any bugs or if you need new features please feel free to issue a pull request with your changes.

17.2 Issues and Feature Requests

Please open a GitHub issue for any bug reports and feature requests.

17.3 Common for all Tools

- All tools imply that your working directory is the directory that contains the artifact you want to work with.
- Furthermore you are responsible for supplying a valid set of AWS credentials. A good tool is aws-mfa
- You you need to set an environment variable "ENV" which indicates the account/staging area you want to work with. This parameter tells the tools which config file to use. Basically something like fink_$(ENV).json is evaluated in the configuration component.

17.4 Installing the development version locally

To install your local development version (after checkout):

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

use pip to install the dev requirements:

```
$ pip install -r requirements_dev.txt
```

17.5 Running Unit-Tests

Use the pytest test-runner to run the fink unit tests. A few tests (with '_aws' in the file name) need AWS. Please turn on your VPN and set the `AWS_DEFAULT_PROFILE`, `ENV`, and `ACCOUNT` environment variables. Details here: https://confluence.finklabs.com/display/OPSSHARED/Deployment+on+AWS.

You need to install the development version of this package so you can run the tests:

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

```
$ export AWS_DEFAULT_PROFILE=superuser-dp-dev
$ export ENV=DEV
$ export ACCOUNT=dp # => or your team account
```

Note: You need to enter an MFA code to run the tests.

```
$ python -m pytest tests/test_cloud*
```

Please make sure that you do not lower the fink test coverage. You can use the following command to make sure:

```
$ python -m pytest --cov fink tests/test_lambda*
```

This requires the `coverage` package (included in the `requirements_dev.txt` file):

```
$ pip install -r requirements_dev.txt
```

17.6 Mock calls to AWS services

For testing fink together with botocore and AWS services we use placebo_awsclient (a tool based on the boto maintainers placebo project). The way placebo_awsclient works is that it is attached to the botocore session and used to record and later playback the communication with AWS services.

The recorded json files for fink tests are stored in 'tests/resources/placebo_awsclient'.

fink testing using placebo playback is transparent (if you know how to run fink tests nothing changes for you).

To record a test using placebo (**first remove old recordings if any**):

```
$ rm -rf tests/resources/placebo_awsclient/tests.test_code_aws.test_code_exit_codes/
$ export PLACEBO_MODE=record
$ python -m pytest -vv --cov-report term-missing --cov fink tests/test_code_aws.
↪py::test_code_exit_codes
```

To switch off placebo record mode and use playback mode:

```
$ export PLACEBO_MODE=playback
```

To run the tests against AWS services (without recording) use `normal` mode:

```
$ export PLACEBO_MODE=normal
```

Please note:

- prerequisite for placebo to work is that all fink tools support that the awsclient is handed in as parameter (by the test or main). If a module creates its own botocore session it breaks fink testability.

- in order to avoid merging placebo json files please **never record all tests (it would take to long anyway)**. only record aws tests which are impacted by your change.
- fink testing using placebo works well together with aws-mfa.
- if you record the tests twice the json files probably get messed up. Please do not do this.
- Please commit the placebo files in a separate commit. This makes reviewing of pull requests easier.

17.7 documenting fink

For fink we need documentation and we publish it on Readthedocs. Consequently the tooling is already set like sphinx, latex, ... We would like to use markdown instead of restructured text so we choose recommonmark.

Detailed information on using markdown and sphinx

17.7.1 Installation of docu tools

```
$ pip install -r requirements_docs.txt
```

If you need to create the pdf docu install pdflatex (... you also need texlive!).

```
$ brew cask install mactex
```

17.7.2 build docu

In order to build the html and pdf version of the documentation

```
$ make html
$ make latexpdf
```

17.7.3 Release docu to Readthedocs

To release the documentation to Readthedocs most of the time there are no additional steps necessary. Just connect your rtfd account to your github repo.

17.7.4 Initialize api docu

We used the sphinx-apidoc tool to create the skeleton (80_fink_api.rst) for fink' api documentation.

```
$ sphinx-apidoc -F -o apidocs fink
```

17.8 Implementation details

This section is used to document fink implementation for fink developers and maintainers.

17.8.1 configuration

note: fink design has a section on openapi, too

configuration in fink is implemented in a few places:

- openapi functionality in fink (in 'fink/fink_openapi.py')
- openapi based validation for each tool (in 'fink_<tool>/openapi_<tool>.yaml' and 'fink_<tool>/plugin.py')
- the functionality to create docs from openapi specs

actual tool config via config-reader	actual command is non-config-command	need to run incept_defaults	need to run validate_config
yes	yes	yes	yes
yes	no	yes	yes
no	yes	yes *)	no
no	no	no	no

*) The above table shows that there is a case where we run a `non-config-command` and do not have configuration from the config reader. In this case we still need the defaults but the defaults alone might not be a valid configuration. To make this case work the `incept-defaults` functionality needs to disable the config validation for the impacted configuration part.

17.9 fink design

17.9.1 Design Goals

- support development teams with tools and templates
- ease, simplify, and master infrastructure-as-code

17.9.2 Design Principles

- write testable code
- tests need to run on all accounts (not just dp account)
- make sure additions and changes have powerful tests
- use pylint to increase your coding style
- we adhere to Semantic Versioning.

17.9.3 Design Decisions

In this section we document important design decisions we made over time while maintaining fink.

Use botocore over boto3

With botocore and boto3 AWS provides two different programmatic interfaces to automate interaction with AWS services.

One of the most noticeable differences between botocore and boto3 is that the client objects:

1. require parameters to be provided as `**kwargs`

2. require the arguments typically be provided as `CamelCased` values.

For example::

```
ddb = session.create_client('dynamodb')
ddb.describe_table(TableName='mytable')
```

In boto3, the equivalent code would be::

```
layer1.describe_table(table_name='mytable')
```

There are several reasons why this was changed in botocore.

The first reason was because we wanted to have the same casing for inputs as well as outputs. In both boto3 and botocore, the response for the `describe_table` calls is::

```
{'Table': {'CreationDateTime': 1393007077.387,
           'ItemCount': 0,
           'KeySchema': {'HashKeyElement': {'AttributeName': 'foo',
                                            'AttributeType': 'S'}},
           'ProvisionedThroughput': {'ReadCapacityUnits': 5,
                                     'WriteCapacityUnits': 5},
           'TableName': 'testtable',
           'TableStatus': 'ACTIVE'}}
```

Notice that the response is `CamelCased`. This makes it more difficult to round trip results. In many cases you want to get the result of a `describe*` call and use that value as input through a corresponding `update*` call. If the input arguments require `snake_casing` but the response data is `CamelCased` then you will need to manually convert all the response elements back to `snake_case` in order to properly round trip.

This makes the case for having consistent casing for both input and output. Why not use `snake_casing` for input as well as output?

We choose to use `CamelCasing` because this is the casing used by AWS services. As a result, we don't have to do any translation from `CamelCasing` to `snake_casing`. We can use the response values exactly as they are returned from AWS services.

This also means that if you are reading the AWS API documentation for services, the names and casing referenced there will match what you would provide to botocore. For example, here's the corresponding API documentation for `dynamodb.describe_table <http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_DescribeTable.html>__`.

17.9.4 Use pytest over nose

For many years py.test and nose coexisted as Python unit test frameworks in addition to std. Python unittest. Nose was developed by Mozilla and was popular for quite some time. In 2015 Mozilla switched from nose to pytest.

http://mathieu.agopian.info/presentations/2015_06_djangocon_europe/

There are many arguments in favour of pytest. For us the most important is pytest fixtures which provides us with a reliable and reusable mechanism to prepare and cleanup resources used during testing.

17.9.5 Capturing of log output during test

In our tests we use a `logcapture` fixture from fink_testtools.helpers to capture log output. The fixture use the textfixtures package under the hood.

use it like this:

```
logcapture.check(
    ('root', 'INFO', 'a message'),
    ('root', 'ERROR', 'another error'),
)
```

or:

```
records = list(logcapture.actual())
assert records[0][2] == 'a message'
```

details here: http://testfixtures.readthedocs.io/en/latest/logging.html

17.9.6 Use Sphinx, Readthedocs, and Markdown for documentation

Many, many documentation tools populate this space since it is so easy to come up with something. However for Open Source projects Readthedocs is the dominant platform to host the documentation.

The Sphinx is the Python std. docu tool. In combination with markdown tools set is a very convenient way to create Readthedocs conform documentation.

17.9.7 Keep a changelog

We where already keeping a changelog which "almost" followed the guide. In June 2017 we decided to make the format more explicit.

The fink changelog format is defined here: http://keepachangelog.com/en/1.0.0/

17.9.8 Use docopt to build the command line interface

There is a never-ending discussion going about pros and cons of CLI tools for Python. Some of these tools are contained in the Python std. library, some are independent open source library additions. At the moment the most popular tools are Optparse, Argparse, Click, and Docopt

https://www.youtube.com/watch?v=pXhcPJK5cMc

We decided to use docopt for out command line interface because it is simple and very flexible. In addition we developed a `dispatch mechanism` to ease the docopt usage and to make the fink CLI commands testable.

17.9.9 Using Maya: Datetimes for Humans

We had some issues in the past using datetimes correctly across different timezones and locales. finklabs SRE team took an initiative to improve the situation and we. We looked into pytz and maya. Maya had a convincing offering and is maintained by Kenneth Reitz so we decided to use it for datetime handling within fink.

17.9.10 Plugin mechanism

fink uses entry points similar to pluggy to find installed plugins

For communication with the plugin we use Blinker signals. This helps us to decouple the fink code base from plugin code and vice-versa. Blinker is of cause only one way to do that. Blinker is fast, simple, well documented, etc. so there are some popular frameworks using it (Flask, Pelican, ...).

17.9.11 Config handling using openapi

A wide area of fink functionality is related to configuration:

- default values for tools and plugins
- validation of configuration
- scaffolding of minimal (required properties) configuration
- sample of complete configuration
- documentation of configuration

To cover all this fink configuration usecases we decided to use the `openapi` specifications. Using openapi alows us to build on existing workflows and tooling.