Fiftycuatro's Cloud Services Documentation

Release 0.0.1

Phillip Gomez

Mar 14, 2018

Contents

1 Overview **1**

2 User Accounts **3**

3 Terraform Configuration **5**

4 Site Accounts **7**

5 How-To **9**

5.1 Configuring Root AWS Account 9
5.2 Developer Workstation Setup 10
5.3 AWS Vault Setup 10
5.4 Member Account Creation 11
5.5 Adding Developer Accounts 11
5.6 References 11

CHAPTER 1

Overview

Managing cloud infrastructure can be overwhelming. This site attempt's to document how Fiftycuatro's cloud infrastructure is setup and managed. Currently mostly AWS services are used. There is no specific reason for using AWS other than I had a some experience with a few of the services over the years. There are a lot of AWS services with tons of options and it can quickly become unmanageable. To keep my head straight on what exactly is configured I am using Terraform. *"[Terraform] is an open source tool that codifes APIs into declaritive configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.*[1] This guide will provide in-depth descriptions on how the Fiftycuatro AWS accounts are setup and configured.

[1] Terraform home page. https://terraform.com

CHAPTER 2

User Accounts

Let's start out with describing how the user accounts are organized. It is considered an AWS best practice to create a single *Bastion* account which manages all IAM users, roles, groups, etc… and then create member accounts for each service environment.[2][3][5] The bastion IAM accounts will be setup to be able to assume various permissions in the member accounts. Additionally consolidated billing will be enabled so that all member accounts will be created under the bastion account allowing for a single place to view and pay bills.

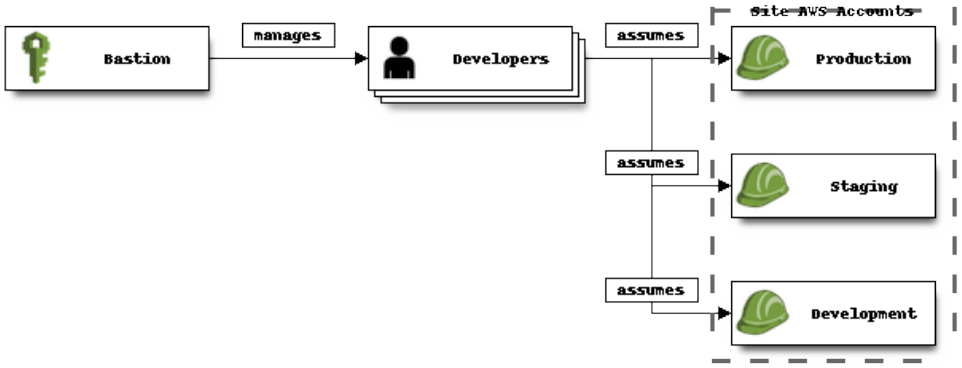

Fig. 1: AWS Account Hierarchy

The benefits to this extra level of user management is isolation between environments. Making any changes to staging (include breaking ones) will have no effect on production. This may not sound like an issue if you structure your service configurations properly and are careful when updating but there are sometime account limits that AWS imposes. For example, each account is limited to 1000 concurrently executing Lambda functions.[4] If you were to try and load test

[2] You need more than one AWS account: AWS bastions and assumed-role. https://engineering.coinbase.com/you-need-more-than-one-aws-account-aws-bastions-and-assume-role-23946c6dfde3

[3] AWS Multiple Account Security Strategy. https://aws.amazon.com/answers/account-management/aws-multi-account-security-strategy/

[5] Your single AWS account is a serious risk. https://cloudonaut.io/your-single-aws-account-is-a-serious-risk/

[4] Administering Lambda-based Applications, Managing Concurrency. https://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html

your staging environment this could starve resources from your production environment when running both under a single account.

CHAPTER 3

Terraform Configuration

Terraform configuration is split into two groups. The first is the bastion configuration which will contain all configuration for the bastion accounts and policies. The second group of accounts will be run by individual developers to run site specific configuration. Usually the site specific configuration will be organized around a single domain and include all the configuration necessary for management of cloud services. Terraform remote state will be stored in an S3 bucket that is setup in the bastion account.

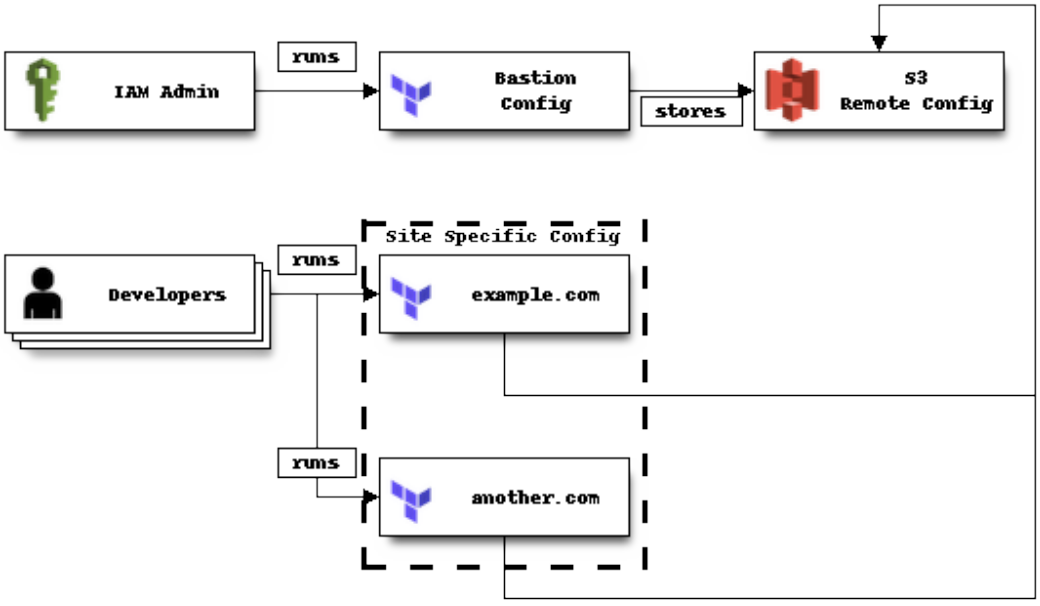

Fig. 1: AWS Terraform Configuration

CHAPTER 4

Site Accounts

The site accounts are created using the AWS Organization's console page. Each account requires an email address but there should be no need to log into this account directly and instead will be logged in using a developer account that has permissions to assume the admin role in that account.

CHAPTER 5

How-To

5.1 Configuring Root AWS Account

Bootstrapping of a main root account should be a one time event. Terraform is not used here as there are only a few steps required and it is a recommended best practice to not create any access and secret keys for the main root account. Essentially we are going to create a single admin user account that will have full IAM access. This account will be the main account used to setup the bastion users. This account only needs to create new users, groups, roles, and policies as the service specific configuration will be done under the individual development accounts that this `admin` account creates.

The following steps will bootstrap the root AWS account and setup an admin user that can be used to setup other users.

> **Warning:** These steps require main root account access. It is recommended best practice to setup multi-factor authentication (MFA) on this main root account as if it gets compromised all users of your organization will be compromised. Additionally MFA can be setup on each IAM account as well.

1. Log into Root AWS account.
2. Enable consolidated billing using the AWS console.
3. Create an IAM user. I named this user `admin`.
4. Create a group for IAM admins. I named this group `IAMAdmins`
5. Attach `IAMFullAccess` policy to the `IAMAdmins` group
6. Add `admin` user to the `IAMAdmins` group.
7. Attach the `AmazonS3FullAccess` policy directly `admin` user.
8. Create the following policy and name it `STSGetFederationToken`. This policy will allow the ability to create temporary login urls.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sts:GetFederationToken",
      "Resource": "*"
   }]
}
```

9. Attach `STSGetFederationToken` policy to `admin` user.
10. Optionally enable MFA for `admin`.
11. Create a S3 bucket to store terraform remote state. I named this bucket `terraform-remote-state.fiftycuatro.com`.

5.2 Developer Workstation Setup

The following steps will list the following tools used on a developer workstation/laptop:

5.2.1 OSX

1. Install homebrew.

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
↪install/master/install)"
```

2. Install aws-vault.

```
$ brew cask install aws-vault
```

3. Install Terraform.

```
$ brew install terraform
```

5.2.2 Windows

Should work relatively the same as OSX without the use of homebrew.

5.3 AWS Vault Setup

5.3.1 Adding Credentials

1. Create access token for IAM account using AWS web console and save for later step.
2. Add profile to `~/.aws/config`.

```
[profile iam-admin]
region = us-east-1
```

3. Optional: If MFA is enabled for IAM account add the following entry replacing `[account_id]` and `[account_name]` respectively.

```
[profile iam-admin]
region = us-east-1
mfa_serial = arn:aws:iam::[account_id]:mfa/[account_name]
```

4. Add `access_key` and `secret_key`

```
$ aws-vault add iam-admin
```

5.3.2 Logging into AWS Console

1. `aws-vault` can be used to create temporary credentials and login urls for console access for any profile that allows `sts:GetFederationToken`.

```
$ aws login iam-profile
```

5.4 Member Account Creation

A member account should be created when attempting to isolate environments. Usually this is used to separate sites along with production, staging, developments within a single site.

1. Log into

5.5 Adding Developer Accounts

Developer accounts

5.6 References