Deployment within a Landing Zone with AWS Deployment Framework (ADF)

6 January 2023 - 8 min. read

AWS Deployment Framework (ADF) | Cloud adoption | governance and compliance | Landing Zone | multi-account strategy

Over the years, the cloud world has moved towards an increasingly *Infrastructure-as-Code* oriented approach bringing enormous advantages such as replicability and standardization.

Infrastructure-as-Code is notoriously associated with the deployment of infrastructures capable of hosting web applications or, more generally, business workloads.

This context changes when we face scenarios in which we have to configure our Cloud environment to host these infrastructures. Proper preparation requires the deployment of a **structured Landing Zone**.

How can we use the *Infrastructure-as-Code* paradigm when concerning Landing Zones?

We already discussed the concept of a Landing Zone earlier in these articles:

- That's why you need a Landing Zone (even if you don't know it yet!) - Proud2beCloud Blog
- Landing Zone on AWS: design strategies and best practices - Proud2beCloud Blog
- Examples of Landing Zone implementations - Proud2beCloud Blog

In today's article, we will examine how we can manage a Landing Zone using AWS Deployment Framework (ADF), a tool developed by the AWS Proserv team. This framework aims to simplify and automate the deployment of services and infrastructures within a Landing Zone.

Why ADF?

Let us draw on our experience to think about the pros and cons.

We can consider ADF as an alternative to AWS CloudFormation StackSet. The purpose of both solutions is to deploy CloudFormation templates to manage multi-account and multi-region scenarios.

So, **why should we choose ADF**?

To answer this question, we are retracing our experience to highlight the pros and cons that we hope will be useful to you.

Pros

- With ADF, we can create Accounts and Organizational Units programmatically;
- ADF provides more flexibility than StackSet;
- The ADF installation involves creating a CI/CD pipeline that allows us to automate the generation of further deployment pipelines after a simple git push;
- It is perfect for maintaining and creating automation for medium and large companies (enterprises).

Cons

- Keeping this framework as flexible as possible leads us to create ad hoc templates that act as plug-ins; you can find sample templates that do not cover all needs;
- Steep learning curve;
- Overkilling for startups and small businesses.

How to install ADF

Prerequisites

The prerequisites for installing ADF within our Organization are administrative access to the master account or access to a stand-alone account; in the latter case, ADF will create the Organization starting from it.

The other prerequisite, critical to make the account creation work, is always to have a non-default CloudTrail trail within the master account since ADF will use it to inspect the events.

Setup

To install ADF, follow the guide inside the ADF repository.

Due to the constant updating of the source code versions within the Serverless Application Repository, one may come across unstable releases. For example, account bootstrapping may fail due to missing dependencies.

To solve this problem, we followed an alternative procedure using the GitHub repository directly, and the following steps:

```
git clone https://github.com/awslabs/aws-deployment-framework.git
git checkout tags/v3.1.2
git cherry-pick -X theirs 0f971387e9a756a62719cc3be63a41fb8f370912
git cherry-pick –continue
```

How does it work

Components

ADF mainly exploits two accounts with specific tasks to manage deployments in the Organization:

- Master Accounts
- Deployment Accounts

Master account

The Master Account is the only account enabled to contact the AWS Organization APIs. Therefore it is the only one that can create accounts and organizational units. The process of creating an account starts from the repository that we find within the Master account. Browsing the AWS CodeCommit service in the us-east-1 region, we will find the aws-deployment-framework-bootstrap repository. At this point, we will have to write the code necessary to create the multi-account scenario.

A pipeline of the AWS CodePipeline service is invoked for deployment, which calls the Organization APIs to create the accounts previously defined in the repository. The pipeline steps provide the creation of the account and its move from the root organization to the Organizational unit indicated in the repository.

Using the AWS CloudWatch event service, ADF intercepts the event of type MoveAccount and invokes an AWS Step Function that creates or updates the basic CloudFormation stacks of the newly created account and those we will create. Finally, another Step Function is invoked within the Deployment account. We will analyze it in the next paragraph.

Deployment Account

ADF creates the deployment account during setup. It is the one who takes care of the deployment of all the services and resources within our Organization. For the deployment account to manage child accounts, it needs IAM roles that are federated with it, allowing the assume role action. When ADF creates a new account, it will invoke an AWS Step Function to enable cross-account logins.

It updates/creates the roles used by ADF to access that account. Being a deployment framework, you should have already understood that this account is central since, from this point, we will be able to create countless pipelines in just a few clicks; these pipelines are created by ADF using an AWS CodeCommit repository and an AWS CodePipeline.

Inside the repository, we will find a deployment_map folder with the definitions of the pipelines configured and implemented in the Deployment account.
Once the commits containing the deployment_maps have been pushed, a pipeline will start, creating other pipelines that will mirror the deployment maps in the repository: basically a pipeline of pipelines.
Usage examples

Account Creation

As we said earlier, to create an account via ADF, we will need to work on the master account of the Organization, as it is the only one allowed to call the Organization's APIs. Once on the master account of the Organization, in the us-east-1 region and on the CodeCommit service, we will need to clone the code repository called "aws-deployment-framework-bootstrap" and insert inside the adf-accounts folder a file similar to the following:

```
accounts:
  - account_full_name: deployment
    organizational_unit_path: /deployment
    email: adf@proud2becloud.com
    alias: deployment
```

```
    tags:
      - created_by: adf
```

Once this template is pushed, the pipeline will pull the code, run the tests, and call the Organization APIs in order to:

- Create the account with name and alias *deployment*
- Put it inside the *Deployment* OU
- And pass it *adf@proud2becloud.com* as notification mail

As soon as this procedure is complete, the Step Function will start creating the basic CloudFormation stacks and invoke the Step Function within the deployment account to enable cross-account access from child accounts.

This mechanism makes it possible to ensure that as few people/services as possible have access to the master account, according to the least privilege and zero-trust principles.

Once this step function is completed, it will launch all the pipelines related to the organizational unit deployment.

In this way, ADF will automate not only the creation of an account, but also the configuration of the services within it.

Creating a pipeline

Let's say we need a pipeline that creates a VPC in all accounts within the organization unit deployment. To do so, we will have to go to the deployment account on the aws-deployment-framework-pipeline repository, enter the deployment_maps subfolder, and create a file similar to the following:

```
pipelines:
  - name: vpc
    default_providers:
      source:
        provider: codecommit
        properties:
          account_id: 111111111111 # account con repository di codice
      build:
        provider: codebuild
```

```
        properties:
          environment_variables:
            CONTAINS_TRANSFORM: True
      deploy:
        provider: cloudformation
        properties:
          action: replace_on_failure
    params:
      notification_endpoint: adf+deployment@proud2becloud.com
    targets:
      - path: /deploymnet
        regions: eu-west-1
```

This is the definition of an ADF pipeline where you'll only need to specify which account will find the repository (source), which type of deployment to use (deploy), and which account to deploy (target). When the push of the configuration file occurs, the ADF pipeline will start. It will create the pipeline on the deployment account starting from this configuration file. If the VPC repository is not in the account identified by its id (e.g. 111111111111), then the pipeline will create it.
Once done, it will push into the newly created repository with the code, i.e., the CloudFormation template and parameters. The CloudFormation template is standard, while the parameters can be written using a YAML language (unlike the regular use of CloudFormation).

```
Parameters:
  VpcCidr: 10.0.0.0/16
  DeployNat: true
```

One crucial thing about parameters is the filename since ADF uses the account name and the region we want to deploy to, to compose the parameters filename.

Let's say one of the accounts inside the organization unit is called adf-dev, and we want to deploy in Ireland; the parameter file should be called adf-dev_eu-west-1.yml. Once the deployment is complete, every account in the /deployment organization unit will have the VPC stack, even after the first deployment.

To Conclude

The AWS Deployment Framework is a pretty new tool that can revolutionize the way resources, services, and infrastructures are managed and deployed within Landing Zones on AWS.

We have tested it far and wide in recent months in different scenarios with different complexities: from infrastructures with only a few accounts, to Landing Zones with hundreds of accounts distributed across multiple regions. We came up with numerous thoughts about the pros and cons we need to keep in mind when choosing the perfect tool according to our use case.

It is important to remember that there is no absolute best solution: it is always necessary to carry out a thorough analysis of each specific case. We hope that our point of view will make it easier for you!

Have you already used ADF within your Landing Zone? Tell us about your experience in the comments!
See you in 14 days on Proud2beCloud for a new article

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Manuel Petrungaro

Solution Architect @ beSharp. I deal with the creation and engineering of complex infrastructures in the Cloud.Chess enthusiast and member of the "Computer Knowers" club

---

Simone Merlini

CEO and co-founder of beSharp, Cloud Ninja and early adopter of any type of * aaS solution. I divide myself between the PC keyboard and the one with black and white keys; I specialize in deploying gargantuan dinners and testing vintage bottles.
