Explain what this is with a disclaimer…

Contains:

* Sections from a multi-role, support document delivered to a client 2018. Fully sanitized of client detail, some updates, and some generalization.
* Sections from a document providing guidance for creating CloudFormation templates. Original document delivered to same client in 2018.
* Some generic, non client specific insights / lessons learned, during creation of the 2018 POC, delivered to the same client in 2018.

**Automated Application Deployment (AAD) Web Application**

Table of Contents

**Purpose**

**Overview of AAD Web Application**

**End User**

***Overview***

***Different Resource Types***

***Accessing the Database Servers***

***Preliminary Set up with the System Administrator***

***Operations***

Signing in for the first time

Issues with database servers or jump servers

Issues with web servers

Changes to a deployed application

Accessing Resources

**System Administrator**

***Setup High Level***

***Setup Details***

Application Name

Provision and Configure the S3 Bucket to Hold AAD Resources

Set up Security Elements

Cognito User Pool

Cognito Identity Pool

Key Pair

IAM Roles

Provision the Shared Infrastructure

Install the AAD Web Application Files

Add Users to the AAD Web Application

***Shutting Down the AAD Web Application***

***Operations***

Provision a New AAD User

Password Reset

Modify a Deployed Application

***Other Information***

Multiple, Concurrent AAD Web Application in a Single Region

Multiple Current User Access of a Single AAD Web Application

Security Overview

Networking

Asset Tagging

EC2 Servers

Automated Redeployment of Web Servers

Details of interest to a Solution Architect

**Purpose**

This document covers the design, set-up, administration, and use of the Automated Application Deployment web application.

**Overview of AAD Web Application**

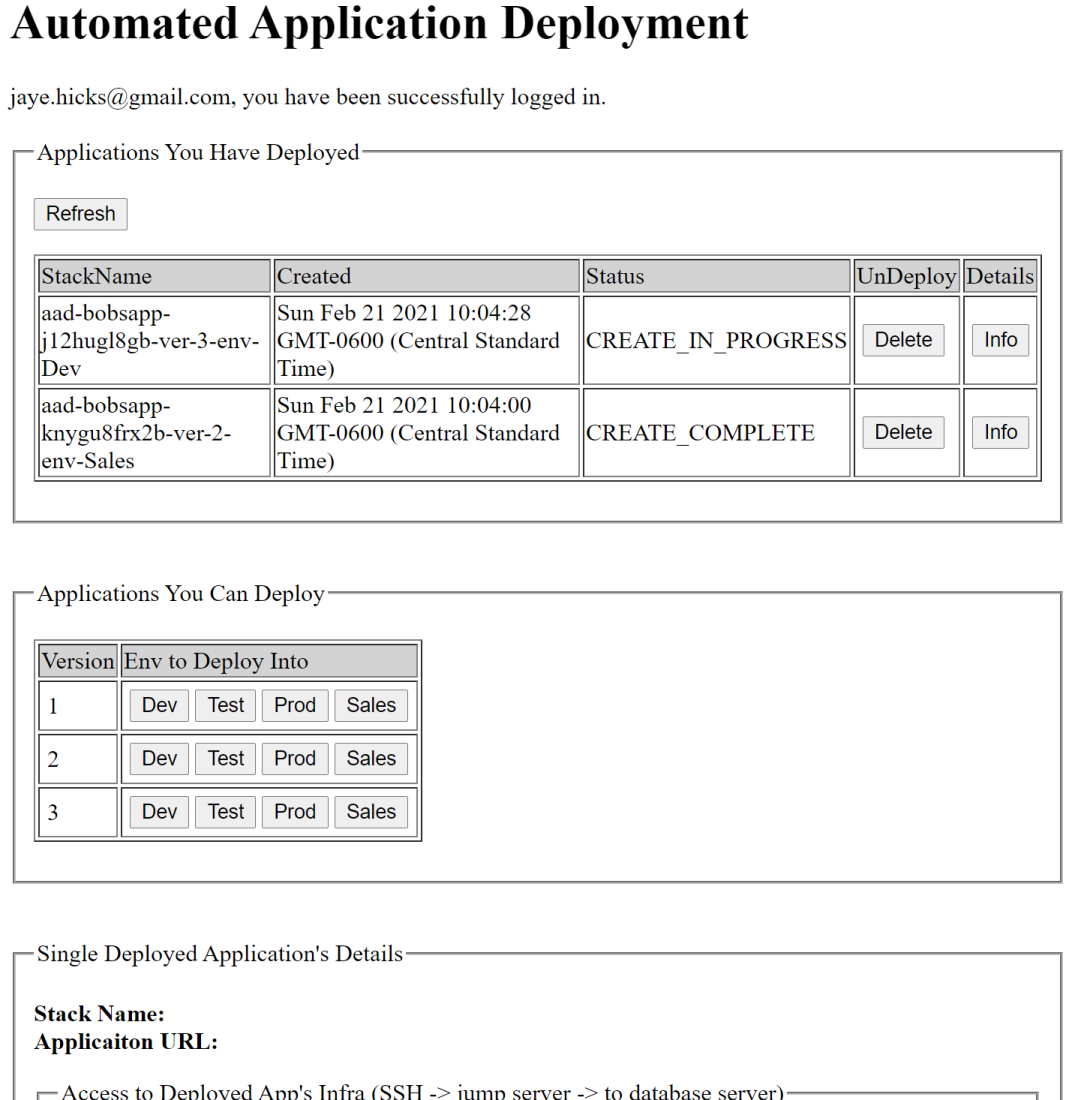
The Automated Application Deployment (AAD) web application automates the deployment of traditional web applications (e.g., Linux, Apache, MySQL, PHP) within a single AWS VPC. With proper set up and configuration, a fully functional web application can be deployed with nothing more than a single button push. Multiple deployments of an application, of the same or different version, can coexist in a single VPC. As long a free CIDR blocks remain within the VPC’s IP address range and EC2 limits are not exceeded, additional application stacks can be launched into the shared VPC. All individual application deployments are isolated from one another but share certain resources (e.g., VPC, Internet Gateway, NAT Gateway, Jump Servers) in order to facilitate both deployment efficiency and cost containment.

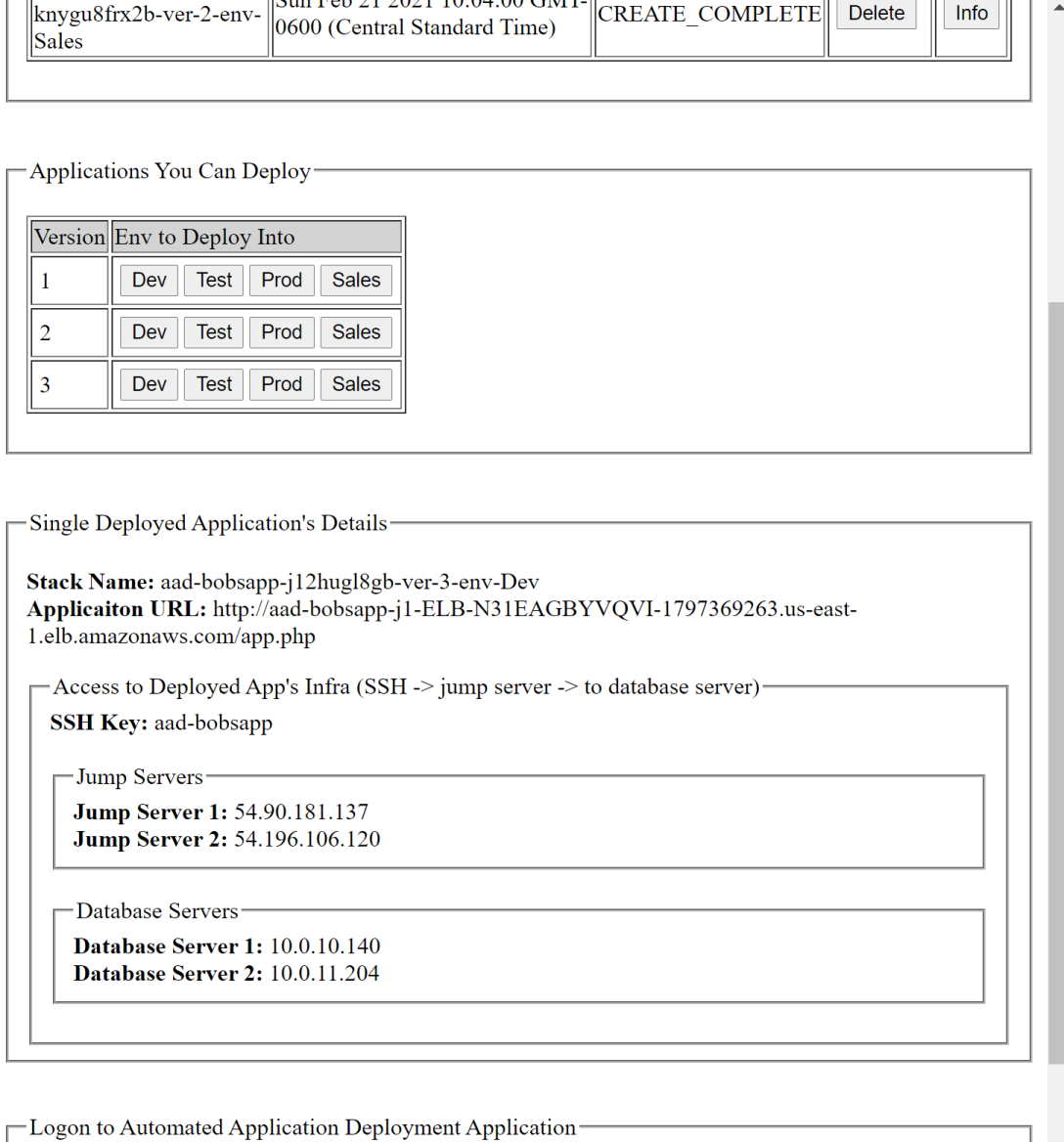
**End User**

***Overview***

The Automated Application Deployment (AAD) web application, hosted in AWS, provides for the automated deployment of traditional web applications into AWS. The screen layout includes four major sections (see figure below):

* Deployed Applications
  + Provides a list of
    - All deployed applications currently hosted in the shared AWS VPC
    - All applications in the process of being deployed into the shared AWS VPC
    - All deployed applications currently hosted in the AWS VPC and in the process of being deleted (i.e., undeployed)
  + Each line item in the list of deployed applications represents a unique application instantiation and provides information specific to that application. Three buttons are provided in this section.
    - Two buttons are provided at the end of each line item. The “Delete” button will undeploy the specific application and the “Info” button will update the “Details for a Deployed Application” section of the AAD web application interface with information specific to the application.
    - A “Refresh” button is located at the top of the section that allows you to update the list of applications. The status of individual applications will change over time based on user action.
* Application Inventory
  + Contains a list of the applications that can be deployed. A given application can be deployed into a specific environment which would determine infrastructure detail (e.g., deployment of an application in “dev” could be on a smaller EC2 instance than an application in “prod”). In addition, multiple versions of a single application can exist in the inventory and would be differentiated by the name of the application.
  + The AAD web application obtains the application inventory list by reading the contents of an S3 bucket that is manually configured (not to be confused with the S3 bucket that is manually configured to host the AAD web application).
  + The team responsible for the application(s) launched by the AAD web application will need to work closely with the system administrator responsible for the AAD web application during the initial configuration of the AAD web application to establish this list of applications that can be deployed by the AAD web application.
* Details for a Deployed Application
  + Selecting the “Info” button at the end of a line item for a specific application in the “Deployed Applications” section of AAD web application interface will display details for that specific application in this section of the interface.
  + The application details displayed include:
    - Stack name – name autogenerated by the AAD web application when the application is deployed. It follows the naming convention ‘aad-’ + <application name> + <unique random characters>
    - Application URL – URL enabling access to a specific deployed application (i.e., a URL that retrieves the index.html page on the web server of the deployed application).
    - SSH Key – key pair required to remotely access AWS virtual machines supporting the deployed application. It is possible to remotely access a jump server which will in turn enable access to database servers.
    - Jump Server 1 / Jump Server 2 –public IP addresses that can be remotely accessed in order to then remotely access the database servers.
    - Database Server 1 / Database Server 2 – private IP addresses that can be used by a remote login session to a Jump Server in order to access a Database Server.
    - To reduce complexity, a single key pair is specified and used to access the jump servers and all of the deployed application’s database servers. The CloudFormation templates supporting the AAD web application as well as the portal html file and JavaScript file could be extended to allow for the designation of a separate key pair for each and every server involved in the AAD web application and the applications that it deploys.
* Logon to AADUserPool Application
  + User login is required in order to access the AAD web application. This is not to be confused with user logon to gain access to the application(s) deployed by the AAD web application.
  + User Ids and Passwords are established by the system administrator supporting the AAD web application. Valid email addresses are used for User Ids. An introductory email will be sent to newly established users and this email will provide the URL for the AAD web application along with the temporary password required for the initial log on. At the initial log on new users are required to supply a permanent password.
  + Provisioning new users, removing existing users, and password resets are manually performed by the system administrator responsible for the AAD web application.





***Different Resource Types***

There are three different type of AWS resources involved in the launch and normal operation of the AAD web application. See the diagram below.

* AAD web application resources include the resources dedicated to the launch and operation of the AAD web application itself.
  + One S3 bucket contains the AAD web application. It is configured for static web site hosting and provides the portal used by end-users to launch applications.
  + One S3 bucket contains two different type of resources
    - Resources required to bootstrap-create the AWS infrastructure that the AAD web application will launch applications into (e.g., VPC, IGW, NAT Gateway, DynamoDB table, etc.) These resources are CloudFormation templates.
    - Resources required to launch and test applications. This includes deployment packages and files containing test patterns used in automated testing.
* VPC-based, leveraged resources that are leveraged across all application deployments (e.g., single VPC, NAT Gateways, Jump Servers)
* VPC-based, transient resources dedicated to a single application deployment (e.g., subnet). Some of these resources are under control of automated processes (i.e., auto scaling group).

***Accessing the Database Servers***

Database Servers can be remotely accessed to facilitate final configuration or normal maintenance. To remotely access a Database Server. In order to execute the steps below you will need to elevate yourself to super user (e.g., “sudo su”).

* Remotely access a Jump Server
* Place the private key of the SSH key pair on the Jump Server
  + Create a file that contains the private key
  + Change the permissions of the file containing the private key (e.g., “chmod 0600 key.pem”)
* Remotely access a Database Server (e.g., “ssh ec2-user@10.0.5.225 -i key.pem”)

***Preliminary Set up with the System Admins***

A member of the application team will need to work closely with the system administrator supporting the AAD web application to establish an application name, load the application deployment packages, and secure the email addresses of users who will deploy applications using the AAD web application.

***Operations***

*Signing on for the first time* - access the AAD web application from the URL provided in the introductory email. Your user id is your email address and a temporary password is supplied in the email. You will need to supply a permanent password (at least 6 characters in length) at initial log in. If you encounter any irregularities at your first log on, multiple back-to-back log ons can often clear up these issues.

*Issues with database servers or jump servers* – database servers and jump servers can remotely accessed in order finalize configuration or resolve issues. In the case of database servers, the resolution of some issues may require use of the AAD web application to undeploy and then redeploy the entire application.

*Issues with web servers* – web/application server issues can be cleared up by requesting the system administrator to manually terminate the instance. The AutoScaling Group controlling the web servers will launch a replacement instance.

*Need a new user or remove existing user* – the system administrator will manage the users allowed to access the AAD web application.

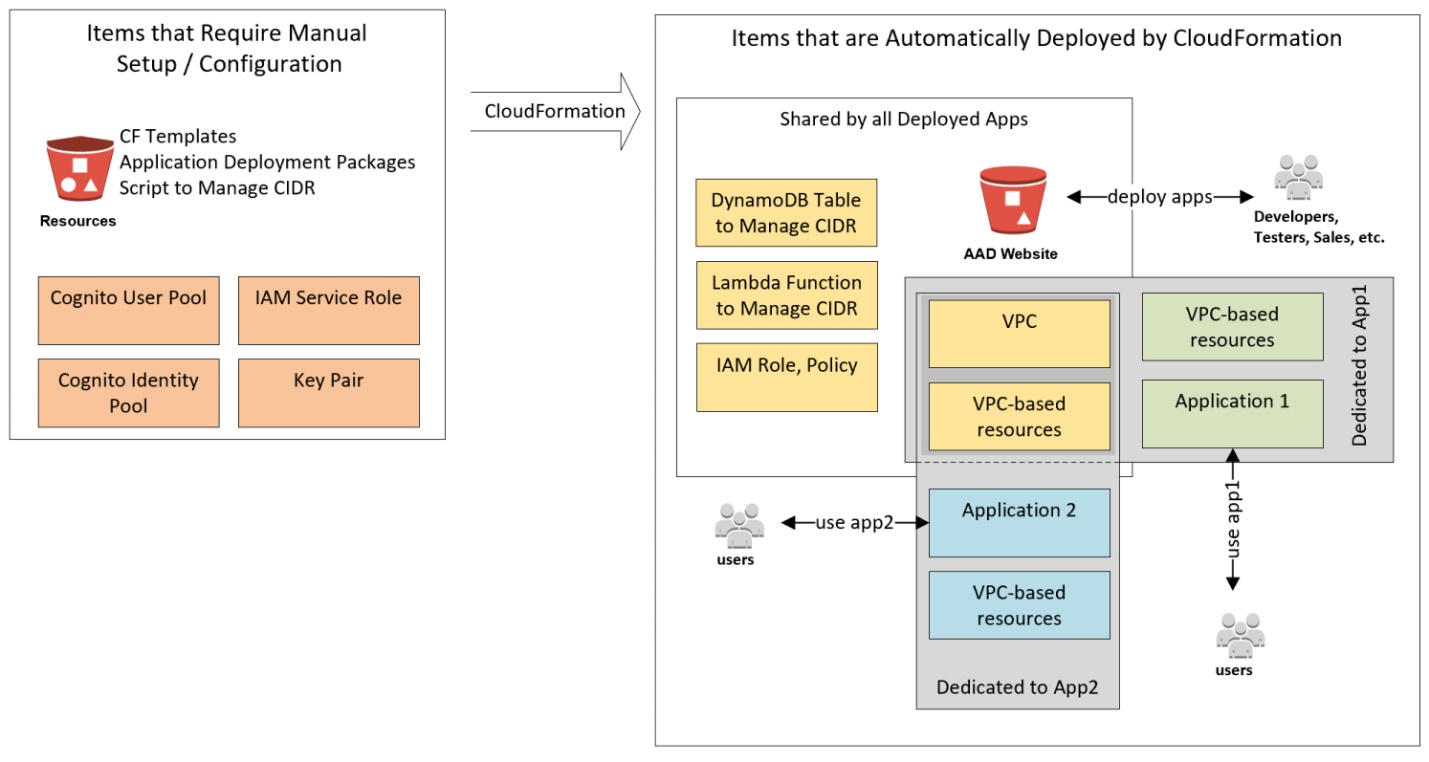
*Changes to a deployed application* – you can change a deployed application by requesting that your system administrator manually access the corresponding stack, via CloudFormation, and make updates. Making changes to a deployed application will result in the replacement of the web/application servers and other resources. Carefully weigh the level of disruption that this will introduce to the end users of the deployed application. You can change the following parameters that are used by the CloudFormation template to deploy your application.

* Application Version
* Environment Selection
* Minimum number of application servers
* Maximum number of application servers
* Desired number of application server

If you update a deployed application

* The URL for your application will remain the same
* In all cases, the following resources will be replaced
  + Launch Configuration (i.e., for App/Web servers)
  + App/Web servers
* In most cases, the following resources will be replaced
  + Database Servers
  + Private Security Groups
  + Public Security Groups

*Accessing AWS Resources that Support Your Deployed Application –* with the exception of what has been previously mentioned (e.g., accessing Jump Servers and Database Servers), even if your personal IAM credential allow you to modify the AWS resources provisioned by the AAD web application, do not do so. The AWS resources provisioned by the AAD web application should only be modified through the AAD web application interface or via requests to the system administrator.



**System Administrator**

***Setup High Level***

You will need to create several infrastructure components to support deployment of the Automated Application Deployment web application. S3 buckets will need to be provisioned and configured as well as components from the Cognito service and the IAM service. After setting up the infrastructure you will need to work with a representative of the application team in order to place the application deployment packages into an S3 bucket as well as procure end user email addresses.

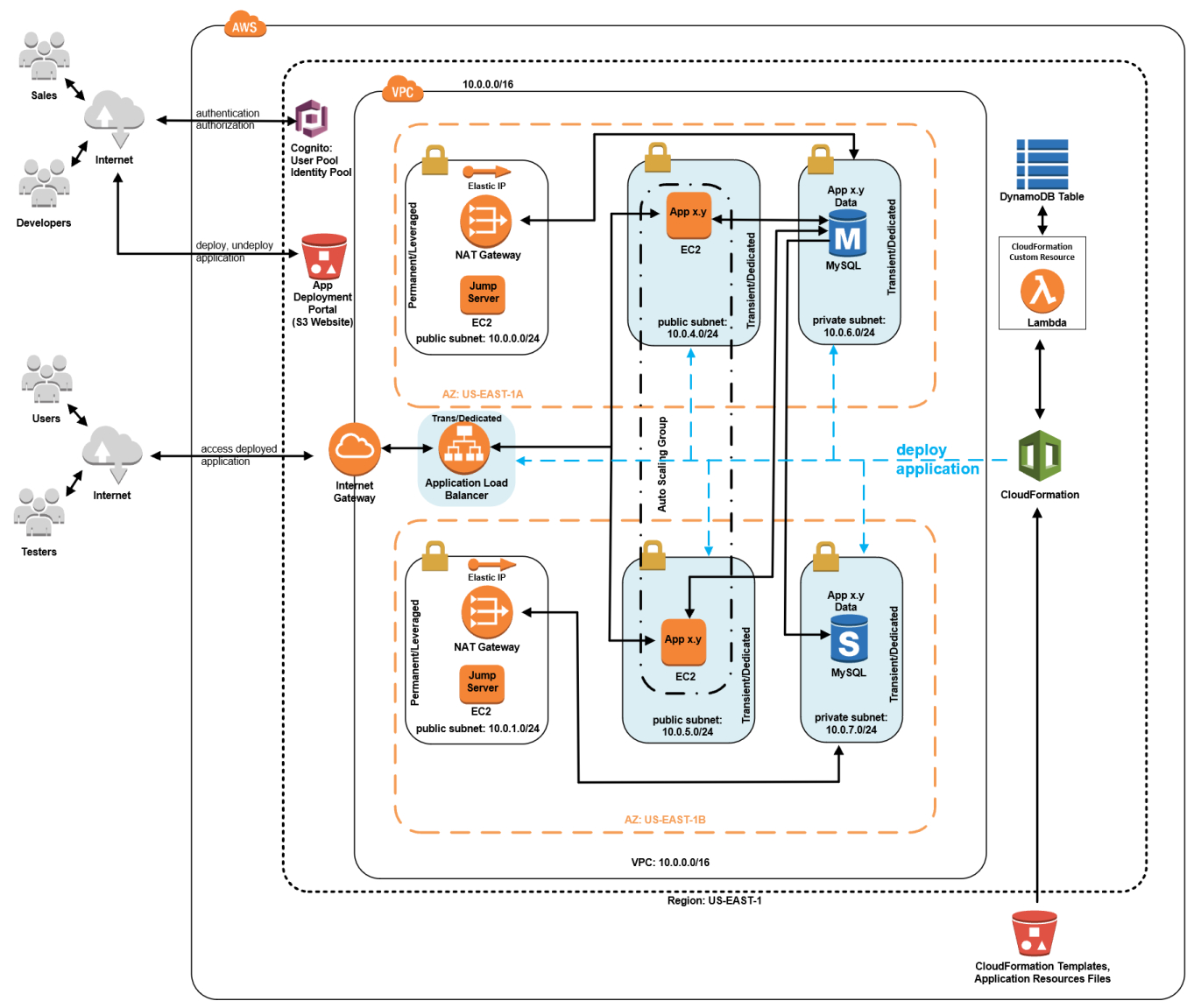
NOTE: it will be more efficient to ensure that you have AWS IAM AdministratorAccess level access before attempting to execute the steps listed below.

The basic steps include:

1. Create and configure an S3 bucket to provide resources to the AAD web application
2. Load the S3 AAD web application resource bucket with CloudFormation templates, application deployment packages, automated CIDR management script, etc.
3. Create the necessary security elements to support the AAD web application: Cognito User Pool, Cognito Identity Pool, Key Pair, IAM service role. You will also need to modify a preexisting IAM role that was generated by creation of the Cognito User Pool.
4. Create, via a master CloudFormation template in the resource bucket, the static, shared infrastructure that will be leveraged by all deployed applications.
5. The previous step generates another S3 bucket that is configured for static web site hosting and will be used to house the AAD web application. You need to load this bucket with the html file and JavaScript file pair that make up the AAD web application.
6. Modify the welcome email message for new users in the Cognito User Pool so that it includes the URL to the hosted AAD web application. In our base example this URL is: http://aadportal-bobsapp.s3-website-us-east-1.amazonaws.com/portal.html
7. Using the email addresses that you have been supplied for AAD web application end users, create new users in the Cognito User pool. Creating a new user sends that user an introductory email and enables them to being using the AAD web application.

***Set Up Details***

Logical infrastructure diagram.



The following steps should be taken, in the order presented, to create an instantiation of the AAD web application.

*1) Application Name*

When you are engaged by an application team representative to set up the Automated Application Deployment web application to support automated launching of applications, your first task is to capture a short, concise name (e.g., application name or representative name for the collection of applications to be deployed). This name should be 8 – 12 characters in length, meaningful, and suitable to be embedded in the name of CloudFormation stack names, an SSH key pair, the names of S3 bucket names, etc. Take note of naming restrictions across the different services, particularly S3 as is probably the most restrictive. The string “bobsapp” is the application name that will be used in the base example found throughout this document.

NOTE: it is highly unlikely, but possible, that application name that you select combined with the prefixes of “aadportal-“ and “aadresources-“ will combine to form an S3 bucket name of a preexisting S3 bucket. You will detect whether or not this is an issue for the resources bucket when you manually create it. The portal bucket is automatically generated by CloudFormation and you should investigate to determine if the name that will be used for it is available. You will have to select a suitable application name as the CloudFormation templates and the single page web application are designed to source files from S3 buckets that follow a specific naming standard.

*2) Provision and Configure the S3 Bucket to Hold AAD Resources*

Create an S3 bucket to contain resources for the AAD web application. After creation and configuration, you will need to load it with all of the necessary resources. Here are the steps required to configure and load the bucket.

1. Use the following pattern to name this bucket is “aadresources-” + <application name>. In order for the AAD web application to function properly, it is critical that the name for this bucket follow this naming standard exactly. Also, it is strongly suggested that you ensure there are no leading, trailing, or embedded hyphens in the application name. When the string “addresources-“ is concatenated with the application name there should only be one hyphen that occurs just before the application name and this hyphen is already supplied for you in the string “addresources-“.
2. Comply with organizational tagging policies (i.e., enable accounting, security, project affiliation of AWS resources, etc.)
3. Configure the S3 bucket
   1. Set up a bucket policy that grants “S3:GetObject” access privilege to the general public. For a production implementation of the AAD web application you would want to refine this access level from the general public to an IAM user or a group of IAM users that have AdministratorAccess level permission for the AWS account. NOTE: in the example JSON below the ARN value supplied after the “Resource” name will vary from S3 bucket to S3 bucket. You will need to supply the ARN of the S3 resource bucket that you are configuring and attach “/\*” to the end of the ARN. The values for “Id”, “Version”, and “Sid” are arbitrary but you may choose to supply values meaningful to your organization.

{

"Id": "Policy1234567890123",

"Version": "2012-10-17",

"Statement":

[

{

"Sid": "Stmt1234567890123",

"Action":

[

"s3:GetObject"

],

"Effect": "Allow",

"Resource": "arn:aws:s3:::aadresources-bobsapp/\*",

"Principal": "\*"

}

]

}

1. Ensure that public read access exists on the entire bucket. This should come as a result of setting up the bucket policy in the step above. You can visually confirm public read access by accessing the AWS Console listing of S3 buckets. In the “Access” column for this bucket you should see an icon with “public” showing up in white letters against an orange background.
2. Create two folders, at the root level, inside of the bucket. Name one folder “web-app-res” and name the other folder “deploy-app-res”.
   1. These folders must be created and named exactly as specified in this step to enable the AAD web application to function properly. JavaScript in the AAD web application relies on these specific folder names.
   2. Ensure these folders are accessible to the general public. At the AWS Console when these two folders are listed as the contents of the S3 resources bucket, highlight the box at the top of the name column in order to select both folders and then click on the “Actions” button and then select the “Make public” option in order to ensure the folders and the contents of the folders are accessible to the general public.
3. Load the two folders as follows:
   1. Into the “web-app-res” folder load the CloudFormation templates and the IP management Python script (i.e., autosubnet.py zipped into cidr-assignment.zip). Include these files:
      1. “app-infrastructure.yaml”
      2. “cidr-assignment.zip”
      3. “cidr-assignment.yaml”
      4. “master-cf-process.yaml”
      5. “portal-website.yaml”
      6. “shared-infrastructure.yaml”
      7. NOTE: even though the bucket policy you created in an earlier step grants public access to the bucket, you will have to enable public access on each individual file that you place into the bucket. An easy way of doing so is to load all of the files and then at the AWS Console screen where you list the contents of the “web-app-res” folder check the box at the top of the name column to select all files in the folder and then select the “Actions” button to then selection the “make public” option on the pop up menu.
   2. Into the “deploy-app-res” folder load the application deployment packages.
      1. Work with a representative of the application team in order to achieve this step.
      2. The base AAD web application employs a simple application deployment packaging scheme. You are welcome to introduce a different deployment scheme. If you decide to introduce a different deployment scheme, the JavaScript in the file “portal.html” will require extensive modifications as well as the initial configuration of the EC2 web application servers in the AutoScaling Group Launch Configuration.
      3. The default application deployment scheme explained: each distinct version of the application is comprised of two zip files. One zip file provides the application itself and the other zip file supports automated testing to ensure that the application was properly deployed. The application file is name “App” + <version number> + “.zip” (e.g., App1.zip) and the test file is named “Test” + <version number > + “.zip” (e.g., Test1.zip). The deployment of a specific version of the application requires both the application file and the test file. The two files names for a single version of the application must have the same version suffix (e.g., App1 and Test1) and it must be unique across all application versions.
      4. Place all of the application deployment files (e.g., “App1.zip”) into the “deploy-app-res” folder. A flat structure – no subfolders.
      5. Place all of the application installation verification files (e.g., “Test1.zip”) into the same “deploy-app-res” folder. A flat structure – no subfolders.
      6. NOTE: even though the bucket policy you created earlier grants public access to the bucket, you will have to enable public access on each individual file that you place into the bucket. An easy way of doing so is to load all of the files and then at the AWS Console screen where you list the contents of the “deploy-app-res” folder check the box at the top of the “name” column to select all files in the folder and then select the “actions” button to then selection the “make public” option on the pop up menu.
4. Take note of the URL for the S3 object “master-cf-process.yaml” as you will need to supply it as a parameter when you use CloudFormation, later on in the installation process, to create the stack responsible for bootstrap-launching the AAD web application.

*3) Set up Security Elements*

The AAD web application portal requires a Cognito User Pool, a Cognito Identity Pool, a Key Pair, modification of an existing IAM Role, creation of a new IAM role, and the collection of the email addresses for end users (will be used as their user id).

***Cognito User Pool:***

Create a Cognito User Pool named “AADUserPool.” This pool will manage user logons to the AAD web application. Use the following values to configure the User Pool. Note that some of the values listed below are autogenerated for you when you set the pool up.

General Settings:

*How do you want your end users to sign in?*

Usernameradio button not selected; instead “email…” radio button selected

Email… “email address or phone number” radio button is selected

Req attrs none

Custom attrs none

Min pw len 6

Pw policy none

Usr sign ups only admins can create users

MFA off

Verifications none

Adv Sec no

Tags none

App clients AADAppClient (see details below)

Triggers none

Attributes

Users log in using: email address or phone number….

Allow email addresses

Required attributes none

Policies

Min length pw: Min of 6 chars (no other requirements)

Who can create Only allow administrators to create users

Days to expire 90

MFA&Verify

MFA Off

Req Verification None

Since neither phone or email verified, no auto recovery of passwords by users

Role None

Advanced Security: None

Custom messages: Add URL to Portal (see details below) and tip them off about password reset

Tags

None

Devices

None

App Client

App client name: AADAppClient

App client id: 12345678901234567890123456 (auto generated by Cognito)

App client secret: none

Refresh token 30 days

ADMIN\_NO\_SRP\_AUTH Off

CUSTOM\_AUTH\_FLOW\_ONLY Off

USER\_PASSWORD\_AUTH On

Triggers

None

Analytics

None

App client settings

Enabled identity providers: None

Callback URLs: None

Sign out URLs: None

OAuth 2.0

Allowed OAuth Flows: None

Allowed OAuth Scopes: None

Domain prefix None

Full Domain: None

UI customizations: None (noticed some kind of error message on this page)

Resources servers: None

Federation

Federated Id providers: None

Attribute mapping: None

NOTE:

Above in the “Custom messages” section you specify text that all newly provisioned users will receive (i.e., via an email). It is very convenient and efficient to supply the user with their user id, their temporary password, and the URL to the Automated Application Deployment web application portal. However, the URL is only available after you successfully create a stack based on the CloudFormation master stack template. In our base example the master stack template is located in an S3 bucket:

https://aadresources-bobsapp.s3.amazonaws.com/web-app-res/master-cf-process.yaml

Executing this stack will provision all of the infrastructure that will be shared by all of the applications that are launched by the Automated Application Deployment web application. An example of the URL for the web application portal, in our base example, is:

<https://aadportal-bobsapp.s3.amazonaws.com/portal.html>

Since you are probably creating the Cognito User Pool before you use CloudFormation to launch the shared infrastructure stack (via the master-cf-process.yaml template) you will need to come back to Cognito and edit the Custom message for your Corgnito User Pool inserting the URL.

Refer to *“6) Add Users to the AAD Web Application”* section of this document for more details on setting up the “Messages customizations” portion of the Cognito User Pool.

***Cognito Identity Pool:***

Create a Cognito Identity Pool to enable users authenticated in the Cognito User Pool to be able to obtain temporary security credentials via IAM Security Token Services. Use the configuration values below to set up the Identity Pool. Confusing (and not necessary), in our base example the Cognito User Pool and the Cognito Identity Pool were given the exact same name.

Identity Pool Name AADUserPool

Identity Pool IDus-east-1:12345678-1234-1234-1234-123456789012 (auto generated)

Unauthenticated roleCognito\_AADUserPoolUnauth\_Role (auto generated**)**

Authenticated roleCognito\_AADUserPoolAuth\_Role (auto generated)

Unauthenticated Id not able to access

Authentication providers

Cognito Tab

User Pool IDus-east-1\_abcdefgh (the Cognito User Pool previously set up)

App client id12345678901234567890123456(from Cognito User Pool)

Push synchronization left default values in place

Cognito Streams left default values in place

Cognito Events left default values in place

***Key Pair:***

In the EC2 section of the AWS Console create a Key Pair in the region in which you will deploy the AAD web application. Ideally, you will use the application name provided earlier in the process by the application team representative for the Key Pair name. In our base example the key pair name is “aad-bobsapp.”

***IAM Roles:***

You will make modifications to a preexisting IAM role, that was auto created with the User Pool, and create a new IAM role.

Create a service role, named AADUserPool, to enable CloudFormation to create/modify/delete AWS resources in the account. Go to the IAM section of the AWS Console and create a service policy for the CloudFormation service and give the role the AWS managed policy named “AdministratorAccess” which will enable CloudFormation to create/modify/delete AWS resources in the account. Note the ARN of this newly created role as it will be required in the step below.

Successful creation of the Cognito User Pool will result in the automatic generation of two roles; one for authenticated users and one for unauthenticated users. Ensure that you leave both of these roles in place; both are required for proper execution of Cognito Identity Pools. You will need to add permissions to the role for authenticated users. The role should be named something like “Cognito\_AADUserPoolAuth\_Role” The permissions granted to the role can be contained in the single in-line policy. Ensure that you do not delete the first statement that is auto generated in the policy (i.e., it involves mobile analytics, cognito sync, and cognito identity). You will be adding two new statements. In the second statement that you add, you will need to include the ARN of the IAM service role that you created earlier. The final permissions for this role should read something like this example:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"mobileanalytics:PutEvents",

"cognito-sync:\*",

"cognito-identity:\*"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"cloudformation:\*"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": "iam:PassRole",

"Resource": "arn:aws:iam::12345678901:role/AADUserPool"

}

]

}

NOTE:

Currently, S3 static hosting does not support SSL certificates (i.e., an https://... URL for your web site hosted in S3). Most modern browsers will display warning messages when it loads a page and detects the collection of user id and password information. To get around this warning you can place the S3 bucket behind a CloudFront distribution that provides support for SSL certificates (i.e., you can get an https://... ULR for the AAD web application).

*Brad, just give me a call and I’ll walk you through this. It won’t take much effort or a very long time.*

*4) Provision the Shared Infrastructure*

The creation and configuration of AWS resources in previous steps has been done manually. These resources can be safely left in place for long periods of time without any major cost concerns in case you want to take a particular deployment of the AAD web application off line for a long period of time but want to retain the ability to quickly bring it back up at some future date.

The AWS resources created and configured in this step will be done so automatically via CloudFormation. These resources include a VPC, route tables, Internet Gateways, Security Groups, Subnets, Jump Servers, NAT Gateways, etc. They should not be left in place for any period of time without completely understanding the cost implications. While some of these elements do not incur costs (e.g., VPC, Security Groups, Route Tables) others do; primarily the Jump Servers and the NAT Gateways (surprisingly NAT Gateways incur EC2 charges).

To enact the following steps use an IAM user account that has AdministratorAccess access.

In the CloudFormation section of the AWS console create a new CloudFormation stack by Choosing a template by specifying an S3 template URL. In our base example the URL would be:

https://aadresources-bobsapp.s3.amazonaws.com/web-app-res/master-cf-process.yaml

For the “Stack name” enter “AAD-“ concatenated with the application name. In this base example the name would be ‘AAD-bobsapp.’ **It is critical that you use upper case ‘AAD’ and not lower case ‘aad.’** In the AAD web application’s JavaScript, upper case ‘AAD’ is used to prefix the CloudFormation stacks used to provision common infrastructure that will be shared by all of the LAMP stacks launched by the AAD web application. Lower case ‘aad’ is used to prefix the CloudFormation stacks used to provision infrastructure that supports a single LAMP application installation. Using the AAP web application, users can not only deploy applications, they can also delete the applications that they have deployed. To avoid an AAD web application end user from accidentally deleting a common / shared infrastructure, the ‘AAD’ CloudFormation stacks are filtered out by the JavaScript when presenting the list of active CloudFormation stacks that can be operated upon.

For the “S3 bucket name”, supply the name of the S3 bucket that you set up to hold AAD web applications resources (i.e., CloudFormation templates, application deployment packages, etc.). Only supply the name of the S3 bucket; do not enter a URL to the bucket or an ARN for the bucket or anything else.

For the “Key Pair” name supply the name of the Key Pair that you set up in a previous step. Do not include the “.pem” extension. Only enter the name of the Key Pair.

For the “First octect” and “Second octet” fields, go with the default values if at all possible. You do not have complete control over the CIDR block that will be used for the shared VPC. It will be a /16 CIDR block so you only have control over the first two octects of the four octect CIDR specification of the IP range that the shared VPC will use.

All AWS resources should be tagged. The CloudFormation templates provided with the base implementation of the AAD web application, automatically apply tags to the resources that they provision. The tag used is “CreatedBy : <CF template name>”

After entering the input values on the first screen and hitting the “next” button you will advance to the next page on which you will enter no values (i.e., accept defaults) and select the “next” button. This will advance you to the final review page. Skipping down to the bottom of the page you will need to check any boxes that appear in the “Capabilities” section. See the “Note” section below for an explanation. After checking all the boxes to “acknowledge” and “accept” what the CloudFormation might do, select the “create” button. **If you fail to check all boxes in the Capabilities section, this CloudFormation template will ultimately fail.**

**Summary of the values you should enter (per our base example):**

Stack name: AAD-bobsapp

S3 Bucket name: aadresources-bobsapp

KeyPair name: aad-bobsapp

First octect: 10

Second octect: 0

In the CloudFormation section of the AWS Console, you will know that the shared infrastructure has been accurately and completely provisioned once you see that four stacks have reached “CREATE-COMPLETE” status. In our base example the names of the four stacks to verify this would be:

***Stack Name*** ***High Level Description of Purpose***

AAD-bobsapp coordinates the creation of nested stacks

AAD-bobsapp-SharedInfra-abc… creates the share infrastructure

AAD-bobsapp-CIDRAssign-def…. manages the assignment of CIDR blocks

AAD-bobsapp-WebAppPortal-ghi… creates the S3 bucket configured for static web hosting

Once again, double check to make sure that these stacks all begin with “AAD” (i.e., upper case) and not “aad” (i.e., lower case). If these stacks are prefixed with lower case “aad” they will show up in the AAD web application interface and would then be subject to accidental deletion by the end users that are deploying and undeploying (i.e., deleting) CloudFormation stacks that represent applications deployed by the AAD web application.

**Note**: CloudFormation will evaluate any template that is sent to it looking for the provisioning of IAM resources. This is an important security feature because were an IAM user or an IAM role created by a user with AdministratorAccess access over the AWS account, the user/role they create could do virtually anything across the account. Therefore, if CloudFormation finds one or more IAM resources being provisioned in a template it will require that someone acknowledge the risk and take responsibility for any consequence that results from provisioning the IAM resource(s). In the CloudFormation portion of the AWS Console this acknowledgement takes the form of checking a box(s) that is labeled something like “I acknowledge this template may provision IAM resources.” If you do not check this box on the last screen (i.e., review screen) before you select the “create” button then CloudFormation will fail to provision the SharedInfra stack because the nested the CIDRAssign stack will fail when it gets around to attempting to provision the IAMRole resource or the IAMPolicy resource.

Look for the acknowledgment check box(s) inside of the “Capabilities” box just above the “create” button. Because the web servers that are part of a deployed application are under control of an AutoScaling Group you will also need to check a box that acknowledges the automated provisioning (i.e., potential expansion) of EC2 instances in order for the AAP web application to be able to operate properly.

*5) Install the AAD Web Application, Single Page Web Application Files*

The previous step created an S3 bucket that is configured for static website hosting. This bucket is empty and requires you to place two files into it. The portal.html file will require configuration but the amazon-cognito-identity-min.js file should be moved over “as is” and will require no modification.

Don’t worry about the file portal.html not being named “index.html” as the CloudFormation template that configured the S3 bucket for static web hosting has taken care of mapping incoming http requests to portal.html. Since the file requires modification you might choose to modify it, see step below, before uploading it to the S3 bucket configured for static web hosting.

Ensure that both files are set to allow public read access. When you upload a file(s) to S3, via the AWS Console, the second step in the multi-screen process allows you to set permissions on the file(s). Ensure that you select “Grant public read access to this object(s)” in the “Manage public permissions” portion of the screen presented to you on the second step of the upload process.

Configure the Portal.html File

* Set the following global variables. Towards the top of the file locate a section inside of a <script> </script> tag pair with a block comment line that contains the string “Configuration Settings”. In this section make the following changes
  + Set the global variable REGION to the region you are working in. In our base example this line would read “var REGION = ‘us-east-1’;”
  + Set the global variable USER\_POOL\_ID to the id that Cognito auto generated for you when you created the User Pool in our base example this line would read “ var USER\_POOL\_ID = 'us-east-1\_123456789';”
  + Set the global variable USER\_POOL\_APP\_CLIENT\_ID to the id that Cognito generated for you when you created the App Client while creating the User Pool In our base example this line would read “ var USER\_POOL\_APP\_CLIENT\_ID = '123456789012345678901234567';”
  + Set the global variable IDENTITY\_POOL\_ID to the Id that Cognito generated for you when you created the Identity Pool. In our base example this line would read “var IDENTITY\_POOL\_ID = 'us-east-1:12345678-1234-1234-1234-123456789012';”
  + Set the global variable APP\_TO\_DEPLOY\_NAME to the application name you collected from the representative of the application team at the beginning of this entire process. In our base example this line would read “var APP\_TO\_DEPLOY\_NAME = 'bobsapp';”
* Make in-line adjustments to the source code of functions.
  + In the function logInToApp() set the key to its proper value by inserting two values. Set the Region you are working in and set the User Pool Id
  + In the function provisionStack() set the ARN of the IAM service role that you created in a previous step that enables CloudFormation to add/modify/delete AWS resources
  + In the function provisionStack() set the location of the CloudFormation template that deploys applications. This is the URL of the S3 bucket that we configured in a previous step to provide resources for the AAD web application.

Once you have accurately and successfully configured the portal.html file move it over, along with the amazon-cognito-identity-min.js file to the S3 bucket that was configured to provide static website hosting of the AAD web application. In our base example the URL to the AAD Web Application is: http://aadportal-bobsapp.s3-website-us-east-1.amazonaws.com

*6) Add Users to the AAD Web Application*

The final step to setting up the AAD web application is to enable user access to the AAD Web application.

Access the Cognito portion of the AWS Console and access the User Pool that you previously created. Under General Settings select the “Message customizations” section. Make the following changes

* Change the email subject to read something like “AAD Portal – your temporary credentials”
* Change the email message to include
  + The URL of the AAD web application. You can construct this URL yourself based on the URL of the S3 bucket configured to serve as a static web site for the AAD web application or you can retrieve this from the CloudFormation portion of the AWS Console by accessing the Outputs section of the AAD-WebAppPortal-ghi… stack. In our base example this URL would be: http://aadportal-bobsapp.s3-website-us-east-1.amazonaws.com/portal.html
  + The user’s temporary password is… Choose something simple such as “password” or the company name or the users own email address. A unique, random password for every new user is probably more trouble than it is worth for this low-volume, one-off, manual process.
  + Additionally, you may elect to provide additional information. For example, new users will be forced to change their password to a permanent one at first log. You might warn them of this. Additionally, irregularities (especially during the first log on or after session cookies have expired) can be cleared up with multiple, back-to-back log on attempts. When in doubt reload the page and try again.

Now cycle through the list of email addresses that you collected for all potential AAD web application end users. Create a new user inside the Cognito User Pool for each email address.

* Access the Cognito portion of the AWS Console and access the User Pool that you previously created. In the “Users and Groups” section select the “Users” tab. Select the “Create user” button
* On the pop up screen
  + Type in the new users email address in the field titled “Username (Required)”
  + Ensure that “Send an invitation to this new user?” is checked and that the “email” box directly below this line is also checked
  + Type in the new user’s temporary password
  + Ensure the box “Mark phone number as verified” is not checked
  + Type in the new users email address in the field titled “Email” and ensure the box immediately below this line titled “Mark email as verified?” is checked
  + Click on the “Create user” button. Normally, the new user will receive an invitation email within a minute or two.

***Shutting Down the AAD Web Application***

The preferred and most efficient process to uninstall the AAD web application is:

* Use the AAD web application interface (in our base example - http://aadportal-bobsapp.s3-website-us-east-1.amazonaws.com/portal.html) to undeploy all deployed applications. That involves selecting the Delete button for each deployed application listed in the “Deployed Applications” section of the AAD web application user interface.
* Manually delete the “portal.html” file and the “amazon-cognito-identity-min.js” file contained in the S3 bucket configured for static web hosting. In our base example this bucket is named “aadportal-bobsapp”
* From the CloudFormation section of the AWS Console, select the master stack (in our base example this stack is named “AAD-bobsapp “) and delete the stack.
* Once you see all four stacks disappear from the CloudFormation section of the Console you know that you have successfully removed all of the AAD web application resources that incur charges of any significance. The four stacks that you want to ensure are deleted, in our base example, would be named:
  + AAD-bobsapp
  + AAD-bobsapp-SharedInfra-<random string>
  + AAD-bobsapp-CIDRAssign-<random string>
  + AAD-bobsapp-WebAppPortal-<random string>
* At this point you may elect to leave the remaining AWS resources, that you manually provisioned and configure, up in the event that you might want to reconstitute the AAD web application at a future date. Doing so will not introduce monthly charges as all of the AWS resources that generate monthly charges of any significance have been deleted / removed by this point. However, if you want to completely remove the remaining AWS resources and configuration settings:
  + Delete the S3 resource bucket that you set up for the AAD web application. In our base example this bucket is named “aadresources-bobsapp”. The easiest way to delete a bucket is to first delete all of its contents and to then delete the bucket itself.
  + Delete the security elements that you put in place to support the AAD web application
    - Key Pair – in our base example this was named “aad-bobsapp”
    - Cognito User Pool – in our base example this was named “AADUserPool”
    - Cognito Identity Pool – in our base example this was named “AADUserPool”
    - IAM Roles
      * Service role
        + in our base example this role was named “AADUserPool”
        + it was set up to give the CloudFormation service AdministratorAccess privileges
      * User Pool Authenticated users
        + In our base example this role was named “Cognito\_AADUserPoolAuth\_Role”
        + It is an “in-line” policy and will be deleted when you delete the Cognito User Pool
      * User Pool Unauthenticated users
        + In our base example this role was named “Cognito\_AADUserPoolUnauth\_Role”
        + It is an “in-line” policy and will be delete automatically when you delete the Cognito User Pool
* If something goes wrong either during normal operations or uninstalling the AAD web application, you can search for resources with the “AAD - <creating stack>” tag in order to clean up manually. The value of the tag “CreatedBy” would be “Master CF Stack”, “Shared Infrastructure Stack”, “CIDR Assignment Stack”, or “Portal Website Stack.” Not all of the resources that you can create in a CloudFormation template support tagging (e.g., AWS::EC2::EIP) so any such resource will not be tagged with the “AAD - <creating stack” name tag that enables you to locate them easily.

***Operations***

If you elect to use the delivered Cognito User Pool functionality for managing and authenticating users:

***Provision a New User***

You can create a new AAD web application user with the Cognito User Pool section of the AWS Console.

NOTE:

At the user’s first log on they will be forced to change their password. After entering the new password in the dialogue box, the best course of action would be for the user to reload the AAD web application portal.html page into their browser and log in at the bottom of the page using the new password. If the page is not reloaded, the user will most likely not be able to successfully log into the application.

***Login Irregularities***

Given a login issue of any sort, always reload the page (use the browsers refresh/reload function) and try the log on again.

***Password Reset***

Password resets were not exhaustively tested and it is unclear if modifications would be required to the JavaScript contained within the portal.html file. During initial exploration, I noticed that Cognito supplies a numeric code (vial email) and I’m not quite sure how that fits into a password reset process flow. One tried and tested method is to delete the user from the Cognito User Pool and add them back. In order to delete a user from a Cognito User Pool you must first “disable” the user. The “delete” button does not show up in the AWS Console for a Cognito User Pool user until that user is disabled.

***Modify a Deployed Application***

You can adjust/change the configuration of a deployed application by going into the CloudFormation section of the AWS console and making a change to the deployed stack that is based on the (i.e., “app-infrastructure.yaml”) template. In our base example the stack name is “aad-bobsapp” + <unique random string>

Changes to your deployed application will result in the replacement of the web/application servers and other resources; carefully weigh the level of disruption that this will introduce. You can change the following parameters that are used in the deployment of your application.

* Application Version
* Environment Selection
* Minimum number of application servers
* Maximum number of application servers
* Desired number of application server

If you update a deployed application

* The URL for your application will remain the same
* In all cases, the following resources will be replaced
  + Launch Configuration (i.e., for App/Web servers)
  + App/Web servers
* In most cases, the following resources will be replaced
  + Database Servers
  + Private Security Groups
  + Public Security Groups

***Other Information***

***AAD Web Application Users***

A single implementation of the AAD web application can support multiple users. It is possible for an AAD web application end user to delete a stack launched by a different AAD web application. Addressing this possibility is beyond the scope of the base implementation of the AAD web application.

***Key Pairs***

In the base implementation of the AAD web application, a single key pair is designated for accessing all EC2 instances (e.g., jump servers, database servers). The AAD web application could be extended to allow for multiple key pairs to be captured (e.g., one for shared infrastructure and a unique key pair for each individual application launched)

***Secured S3 Bucket***

In the base implementation of the AAD web application, a single S3 bucket is used to house the CloudFormation templates required to bootstrap the AAD web application, the templates required to deploy an application, the deployment packages for the applications, and the html and JavaScript that make up the single page web application portion of the overall AAP web application. The html and JavaScript files making up the single page web application will need to be moved to an S3 bucket that is provisioned by a CloudFormation template that bootstraps/launches the AAP web application. The other groups of files can be moved to their own dedicated S3 bucket and appropriate security levels can be put in place for each bucket.

***Multiple, Concurrent AAD Web Application in a Single Region***

In its current form you cannot deploy multiple versions of the AAD web application in the same region. This is primarily due to the way that the automated deployment of AWS resources was decomposed into multiple CloudFormation templates. With the decomposition scheme used, information about AWS resources is shared via the CloudFormation Output section using Export/Import. By way of example consider the shared VPC resources created in “shared-infrastructure.yaml.” These resources are leveraged by multiple templates and are accessed via CloudFormation’s Import functionality. The problem this introduces for bringing up multiple, concurrent AAD web applications instantiations in a single Region is that Export/Import values are global across all CloudFormation templates within a Region. In our example, the values of the shared VPC resources for a single AAD web application would need to remain separate from other AAD web application instances in the same region. Passing parameters to nested stacks is a possible solution although it would greatly increase the size and complexity of the parameter section for nested stacks, particularly the “app-infrastructure.yaml” template.

Other issues that you will have to resolve are ensuring unique names across Cognito User Pools, Cognito Identity Pools, IAM service roles, and key pairs. Reusing any of these across different instantiations of the AAD web application would most likely prove problematic.

***Multiple Current User Access of a Single AAD Web Application***

The CloudFormation custom resource (i.e., Lambda function) is designed as a single-threaded resource. If does not prevent assigning the same CIDR block to two or more application stacks in the event that two separate end users are running the AAD web application at the same time.

***Security Overview***

Users are created and maintained in a Cognito User Pool. The JavaScript in portal.hml pushes users that successfully log into the AAD web application into a Cognito Identity Pool. It is this Cognito Identity Pool that enables (via temporary IAM credentials [STS]) to interact with CloudFormation. The CloudFormation service is given the ability to interact with the AWS services to create/modify/delete the account’s resources via IAM role passing (explained in the next paragraph).

You need to create an IAM service role that provides the ability to create/update/delete a wide variety of AWS services. A role with the AWS managed policy “AdministratorAccess” works quite well. To pair permissions down to the bare minimum required you will need to discover the minimal set of permissions required across the minimum set of AWS services accessed by the AAD web application.

The role for authenticated identities (in our base example the role is named “Cognito\_AADUserPoolAuth\_Role” will have to have the permission to pass on the IAM service role that you have created (i.e., the one with AWS AdministratorAccess – unless you decide to pair this back) to the AWS CloudFormation service. This passing of an IAM role occurs in the JavaScript of the potal.html file (in the function provisionStack()).

***Networking***

When you launch the Automated Application Deployment Web Application, for the shared VPC, you specify the first two octets of the four octets of the VPC’s CIDR block. For example, if you wanted to use the CIDR block “10.11.0.0/16” you would specify “10” and “11.” The first four /24 CIDR blocks of VPCs CIDR block are reserved. Assuming you entered “10” and “11” at launch, 10.11.0.0/24, 10.11.1.0/24, 10.11.2.0/24, and 10.11.3.0/24 would be reserved. Two of these reserved blocks are used for the two shared public subnets that contain the NAT Gateways. The other two have been left for future use.

When applications are deployed using the Automated Application Deployment Web Application, four /24 CIDR blocks will be assigned to each application deployment. Two CIDR blocks will be public subnets and two CIDR blocks will be private subnets. When applications are undeployed CIDR blocks are automatically reclaimed.

***Asset Tagging***

AWS assets created by the AAD Web Application, that are capable of being tagged, are tagged with the “CreatedBy” tag with the value being set to “AAD – <creating stack>.” <creating stack> will be one of the following values: “Master CF Stack”, “Shared Infrastructure Stack”, “CIDR Assignment Stack”, or “Portal Website Stack.”

***EC2 Servers***

In the default state of the “app-infrastructure.yaml” each application deployment will result in the provisioning two web servers, one in each of the dedicated public subnets, and two database servers, one in each of the dedicated private subnets. The web servers are launched indirectly via an AutoScaling group which not only maintains a minimum number of web servers but also introduces the ability to introduce scaling (if you so choose) and the automated redeployment of web servers if one goes down or you decide to change the configuration of the web servers for a deployed application. See “Automated Redeployment of Web Servers” below for more detail. At application deployment time the database servers will require a small amount of final, manual configuration (e.g., master/slave, database user access, etc.). Database servers are launched once, at stack creation time, and are not under control of an AutoScaling or any form of high availability monitoring. The AAD web application can be enhanced to provide monitoring and automated control of the database servers.

***AMIs***

Over time it will be necessary to maintain the AMI maps in the CloudFormation templates to ensure that the references AMIs are not only supported but are the most appropriate.

***Automated Redeployment of Web Servers***

The EC2 instances launched via the AutoScaling group use three config keys. The “configure\_cfn” key sets up the EC2 instance to check back with the CloudFormation service every 5 minutes and take action if the configuration of the deployed EC2 instance is no longer consistent with the desired configuration as defined by the stack. In our base example, a stack would be named “aad-bobsapp”+ <random id> . This stack was launched using the “app-infrastructure.yaml” template and the stack represents a deployed application. If an EC2 instance in a deployed stack detects a difference between its configuration and current configuration defined in CloudFormation, CloudFormation will replace the EC2 instance because the AutoScaling group in the “app-infrastructure.yaml” template is defined with the “WillReplace” flag set to true. This allows application updates after an application has been deployed and for EC2 instances an update results in a replacement.

***Details of Interest to a Solutions Architect / System Administrator***

* The AAD web application is a single page web application (i.e., jQuery) that relies on four CloudFormation templates. Three are run a single time, as part of bringing up the AAD web application, and the fourth (i.e., app-infrastructure.yaml) is run each time an application is deployed by the AAD web application.
  + The “shared-infrastructure.yaml” template
    - Creates the base infrastructure utilized by all other templates
    - Do not manually create this infrastructure under the premise that it is only created once. Inevitably, you will enhance/modify the shared infrastructure and the ability to automatically deploy the updated shared infrastructure a second, third, fourth, etc. time is valuable.
    - The shared infrastructure includes
      * A VPC to contain all application deployments
      * Two Public Subnets
      * Internet Gateway
      * Two NAT Gateways (one for each public subnet)
      * Two EIPs to go with the NAT Gateways
      * Two Route Tables for Private Subnet traffic
      * Two Route Tables for Private Subnet traffic
      * Two Jump Servers
      * S3 Resource bucket contains
        + CloudFormation templates to create stacks to create infrastructure that will be shared

Shared Infrastructure Stack

CIDR Assignment Stack

Finds and reserves CIDR address blocks for all subnets that a deployed application will use

Releases all CIDR address blocks for all subnets that were used by an application being undeployed

This processing is supported by

A DynamoDB Table

A Lambda Function

Portal Website stack

Static web site that hosts the portal used to deploy applications, undeploy applications, and view a list of deployed applications

* + - * + A CloudFormation template, Application Infrastructure stack, which will create dedicated infrastructure for the sole use of an individual, deployed application
        + A Python script deployment package that will be used by a Lambda function to manage the CIDR blocks assigned to individual application deployments
        + The application deployment packages

An application can have multiple versions

Each application version will contain its own separate, independent deployment package

An application deployment package contains all of the files necessary to deploy the application and perform automated testing to ensure reliable deployment (i.e., CloudFormation will wait on the results of this automated testing)

* + The “portal-website.yaml” template
    - An S3 Bucket website is created to host the AAD web application
    - JavaScript packages that will be used in the html file
      * JQuery
      * AWS SDK (access to most AWS services)
      * AWS SDK for Cognito
    - Major JavaScript functionality provided
      * Deploy an application (i.e., create a CF Stack)
      * Undeploy an application (i.e., delete a CF Stack)
      * List all deployed applications (i.e., list all deployed CF Stacks)
      * List all applications available for deployment (i.e., contents of the S3 bucket containing application deployment packages)
      * Log in to the AAD web application
  + The “cidr-assignment.yaml” template
    - Finds and reserves CIDR address blocks for all subnets that a deployed application will use
    - Releases all CIDR address blocks for all subnets that were used by an application being undeployed
    - This processing is supported by
      * A DynamoDB Table
      * A Lambda Function
  + The “app-infrastructure.yaml” template
    - For application deployment
      * Obtain CIDR address blocks by calling the “CIDR-Assignment” Template
      * Create the VPC infrastructure and AWS resources necessary to support an application deployment
        + Subnets dedicated to the application (2 public and two private)
        + AutoScaling Group and Launch Configuration (this will launch EC2 instances)
        + ELB
        + Security Groups
        + Database servers
  + The HTML file
    - JavaScript packages
      * Publicly source packages; specify URL within HTML to include
        + JQuery
        + AWS SDK (access to most AWS services)
      * Local packages collocated with HTML file; specify file name within HTML to include
        + AWS SDK for Cognito (i.e., amazon-cognito-identity.min.js)
    - JavaScript Functions
      * Deploy an application (i.e., create a CF Stack)
      * Undeploy an application (i.e., delete a CF Stack)
      * List all deployed applications (i.e., provisioned CF Stacks)
      * List all applications available for deployment (i.e., contents of the S3 bucket containing application deployment packages)
      * Log in to the AAD web application

**CloudFormation Template Coding Guidance**

**Debugging Templates**

***AWS Console***

Error messages you get in the Events section can be misleading. For example, if your CF template attempts to retrieve a nonexistent file from an S3 bucket you won’t get a “object/file doesn’t exist” error message. Instead you will get a “…permission error…” message.

Error messages you can be very ambiguous. If you get a malformed template error (i.e., error message on the syntax of your CF template, before it gets processed) it will be very generic. If you don’t see anything wrong with it, go to the end of the line indicated by the error message and delete any spaces or tabs that might occur before the carriage return.

You can stream application bootstrapping logs from a CloudFormation stack (i.e., cloud-init.log, cfn-init.log, cfn-hup.log, and cfn-wire.log files) to CloudWatch Logs.

**Official AWS CF Template Best Practice**

***Planning and organizing***

* Organize Your Stacks By Lifecycle and Ownership
* Use Cross-Stack References to Export Shared Resources
* Use IAM to Control Access
* Reuse Templates to Replicate Stacks in Multiple Environments
* Verify Quotas for All Resource Types
* Use Nested Stacks to Reuse Common Template Patterns

***Creating templates***

* Do Not Embed Credentials in Your Templates
* Use AWS-Specific Parameter Types
* Use Parameter Constraints
* Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances
* Use the Latest Helper Scripts
* Validate Templates Before Using Them

***Managing stacks***

* Manage All Stack Resources Through AWS CloudFormation
* Create Change Sets Before Updating Your Stacks
* Use Stack Policies
* Use AWS CloudTrail to Log AWS CloudFormation Calls
* Use Code Reviews and Revision Controls to Manage Your Templates
* Update Your Amazon EC2 Linux Instances Regularly

**Naming**

***When You Generate a Name***

For names used within a template (e.g., logical resource ids, parameter names), you are limited to alphanumeric characters only; no special characters are allowed.

For string values (as opposed to names) you can judiciously introduce hyphens (i.e., “-“) to increase readability (i.e., when an acronym in all caps precedes a word beginning with a capital [e.g., VPC-CIDR-Block] is more readable than VPCCIRBlock). Once again, this is for **string** values and not for CloudFormation logical resource names such as logical names of resources, parameters, output values.

Employ mixed case in naming (e.g., AnExampleNameWithMixedCase)

*When An Acronyms Occurs in a Name*

* Capitalize all letters in the acronym (e.g., RDS)
* If you have two acronyms occurring side by side in your name you may choose to only capitalize the first letter of an acronym if this helps (e.g., CidrIPaddress might be more readable than CIDRIPaddress)

***Mapping Section, Specific Guidance***

* Follow all naming standards and guidance contained in this document for maps except in the following cases:
  + Maps should both index on and produce values that can be used, either inside a CloudFormation template or outside, directly without translation. In order to achieve this goal you can deviate from the naming standards and guidance contained in this document. Always strive to work in AWS generated/maintained values versus your own representation of those values.
  + Map output values: if a map was returning a region name that you need to feed into further CloudFormation processing you should use the value “us-east-1” for the North Virginia region instead of something like “north-virginia” which would have to be translated into “us-east-1” before serving as a parameter to an AWS API call.
  + Map input values: in the example map below, from an efficiency standpoint, it would make sense to invoke the map by calling the function “!Ref AWS::Region” to supply the input value to the map.

Mappings:

RegionMap:

us-east-1:

AmazonLinux : "ami-c481fad3"

us-east-2:

AmazonLinux : "ami-71ca9114"

us-west-1:

AmazonLinux : "ami-de347abe"

us-west-2:

AmazonLinux : "ami-b04e92d0"

***Outputs Section, Specific Guidance***

* It is best practice to break up complex templates into a master template and subordinate, nested templates. When you are following this design pattern
  + When creating an export name for a resource in the output section use this naming formula: <name of template the output section occurs in [or some portion of the name]> + “-“ + <meaningful name [following mixed case]>
  + For example, consider the snippet below from the Output section of a template named master-cf-process.yaml:

Outputs:

KeyPair:

Description: Name of KeyPair associated to an EC2 instance

Value: !Ref iKeyPair

Export:

Name: Master-KeyPair

**Reserved Words (do not use)**

When you create a name, do not create an inherently confusing one. Try to avoid using strings that have a special meaning within CloudFormation templates (e.g., Resources).

* For overall readability and to avoid confusion, do not use the exact string, or a variation of the string that only differs from the original string by the case of a letter(s).
* If you must use a reserved word, consider embedding it in a name by concatenating it with a prefix and / or suffix.

|  |  |
| --- | --- |
| **String/Name/Word/Value** | **Note** |
| *Common strings with special meaning:*  “String”, “Number”, “true”, “false”  “Type”, “Properties”, “Default”, “Value”, “Name”, | Special meaning assigned to these strings.   * Often the special meaning only applies when the string occurs in a certain context within a CF Template * Regardless, you should still avoid using a name that exactly matches a common string that has special meaning. |
| *Names of the 8 major CF sections:*  “Resources”, “Version”, “Description”, “Parameters”, “Outputs”, “Mappings”, “Metadata”, “Conditions” | Special meaning assigned to these strings.   * Often the special meaning only applies when the string occurs in a certain context within a CF Template * Regardless, you should still avoid using a name that exactly matches a common string that has special meaning. |

**Available Online Resources**

|  |  |
| --- | --- |
| **Document** | **URL** |
| CloudFormation User’s Guide | https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html |
| CF Resource Types | https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html |
| Template Snippets | https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/CHAP\_TemplateQuickRef.html |
| Sample Templates | https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sample-templates.html |
| CloudFormation Doco | https://aws.amazon.com/documentation/cloudformation/ |
| User Guide Template reference | https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-reference.html |

**Insights**

asdf