

# Drone Shuttles Ltd. Web Application Architecture with AWS

DEPLOYMENT GUIDE  
AMIT BHARDWAJ

Table of Contents

1. *PURPOSE*.....2

2. *OVERVIEW* .....2

3. *REQUIREMENT* .....2

4. *ARCHITECTURE DESIGN*.....3

5. *PRE DEPLOYMENT* .....8

6. *DEPLOYMENT* .....9

7. *ROLLBACK* .....14

*ANNEXURE - I*.....16

## 1. PURPOSE

The purpose of this document is to provide step by step process to deploy the Drone Shuttle Ltd. Web Application architecture on AWS Cloud.

## 2. OVERVIEW

This document focuses to deploy the Web Application architecture as per the project requirements from all prospects like availability, scalability, security and cost optimization.

### **Information about Customer:**

“ The customer Drone Shuttles Ltd. is currently running their website on an outdated platform hosted in their own datacenter.

They are about to launch a new product that will revolutionize the market and want to increase their social media presence with a blogging platform.

During their ongoing modernization process, they decided they want to use the Ghost Blog platform for their marketing efforts. “

## 3. REQUIREMENT

There are following requirement from Drone Shuttles Ltd for their Web Application:

- 3.1. Solution should be able to adapt to traffic spikes, as expected that during the new product launch or marketing campaigns there could be increases of up to 4 times the typical load.
- 3.2. It is crucial that the platform remains online even in case of a significant geographical failure. The customer is also interested in disaster recovery capabilities in case of a region failure.
- 3.3. The customer plans to have 5 DevOps teams working on the project. The teams want to be able to release new versions of the application multiple times per day, without requiring any downtime.
- 3.4. The customer wants to have multiple separated environments to support their development efforts.
- 3.5. The website will be exposed to the internet, thus the security team also needs to have visibility into the platform and its operations.
- 3.6. The customer has also asked for the ability to delete all posts at once using a serverless function.

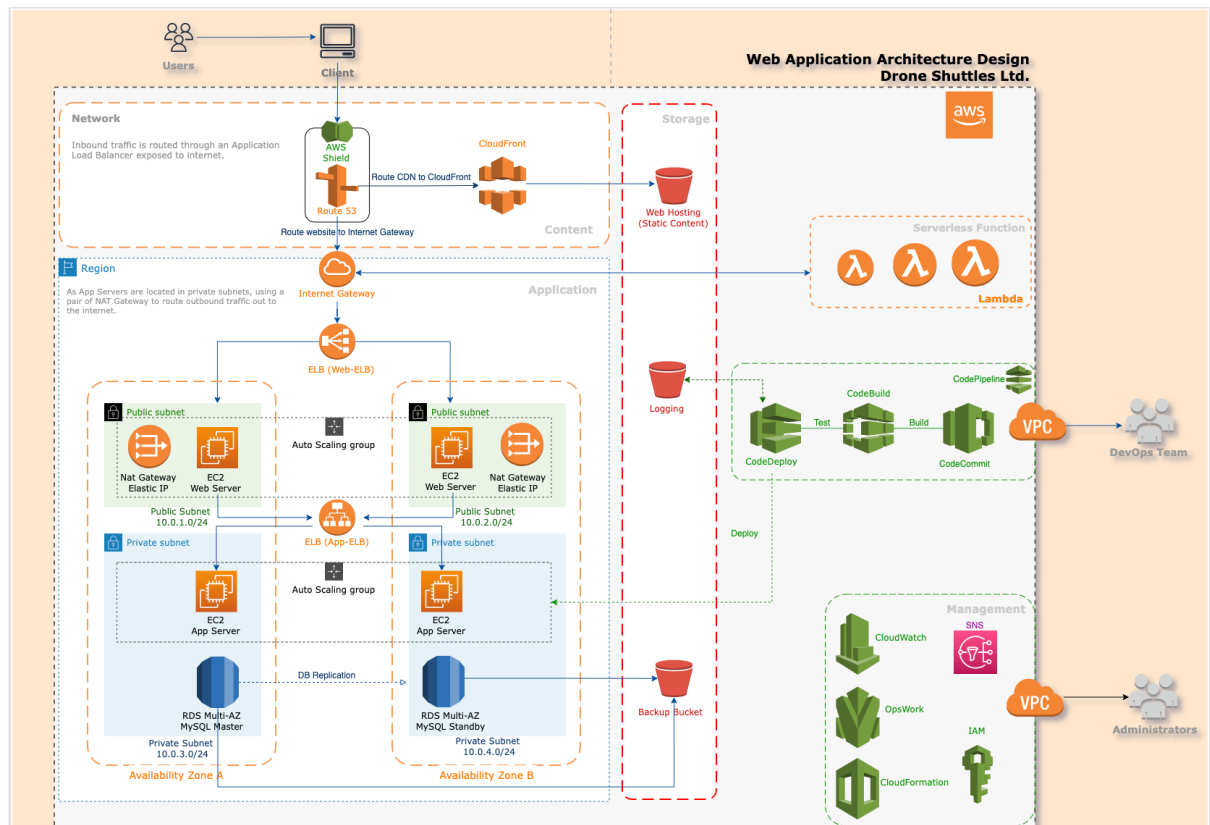
## 4. ARCHITECTURE DESIGN

The architecture is designed to deploy the infrastructure on AWS Cloud as per company requirement, so using multiple AWS services which will cover further in this section.

### Architecture Overview :

To view the architecture open file in the directory :

[DroneShuttle\_Project/architecture/DroneShuttles\_ArchitectureDesign.drawio.html]



To implement this solution by deploying the following AWS services :

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>- AWS IAM</li> <li>- AWS VPC</li> <li>- AWS EC2</li> <li>- AWS ELB</li> <li>- AWS AutoScaling</li> <li>- AWS CloudFront</li> <li>- AWS RDS</li> <li>- AWS S3</li> <li>- AWS Route53</li> <li>- AWS Security Group &amp; NACL</li> </ul> | <ul style="list-style-type: none"> <li>- AWS Cloudwatch</li> <li>- AWS SNS</li> <li>- AWS CodeDeploy</li> <li>- AWS CloudBuild</li> <li>- AWS CloudCommit</li> <li>- AWS Code Pipeline</li> <li>- AWS Lambda</li> </ul> |
|--|---|

These services are included as per the customer's requirement in regards to :

- a. **Unexpected Traffic spikes**, will pre-warm are Web/App Servers with specific AWS provided AMI's and take the leverage of AutoScaling services feature - "Launch Configuration" and AutoScaling Groups with conditions as :
- Scale-up capacity by 1 instance if CPU > 80% for 5 minutes.
  - Scale-down capacity by 1 instance if CPU < 25% for 5 minutes.

Further in future, scaling up/down capacity can be changed as per the requirement.

- b. **For DevOps team**, they can leverage the AWS Services like CodeDeploy, CodeBuild, CodeCommit and CodePipeline.

Also CloudFormation and OpsWork Services can be used to re-deploy or modify the infrastructure and services.

- c. **For multiple separated environments**, Deploying the infrastructure using code through AWS Cloud Formation service so that the code can be reused to deploy the multiple environment like Dev, Staging and Prod.
- d. **To secure the environment**, as website is exposed to internet thus for Security team:
- Inbound traffic is routed through an Application Load Balancer exposed to internet.
  - App Servers are located in private subnets, using a pair of NAT Gateway to route outbound traffic out to the internet.
- e. **To delete all posts at once using a serverless function**, here can leverage the AWS Lambda feature.

"AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. You can trigger Lambda from over 200 AWS services and software as a service (SaaS) applications, and only pay for what you use."

- f. **For Disaster Recovery Solution**, objective for DR should be bringing the workload back up or avoiding downtime altogether.

DR solutions should be created based on the following objectives:

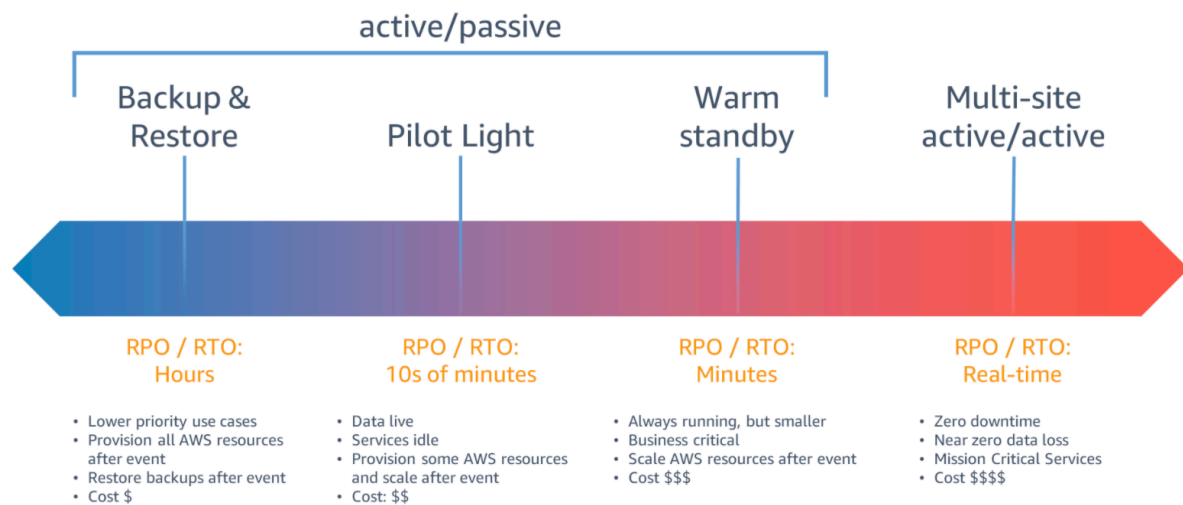
How much downtime you can afford or quickly must you want to recover?

- Recovery time objective (RTO): The maximum acceptable delay between the interruption of service and restoration of service. This determines an acceptable length of time for service downtime.

How much data you can afford to recreate or lose?

- Recovery point objective (RPO): The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data.

For DR strategies, AWS offers resources and services to build a DR strategy that meets your business needs.



As shown in the above figure, [four strategies for DR that are highlighted in the DR whitepaper](#). From left to right, the graphic shows how DR strategies incur differing RTO and RPO.

To select the best strategy, must analyse benefits and risks with the business owner of a workload, as informed by engineering/IT. Determine what RTO and RPO are needed for the workload, and what investment in money, time, and effort you are willing to make.

Now to deploy the infrastructure as per above architecture design, created CloudFormation [1] templates as a Infrastructure-as-code (IaC) to reuse this solution for creating multiple environments (dev/staging/prod).

A template can be used repeatedly to create identical copies of the same stack (or to use as a foundation to start a new stack). Templates are simple YAML formatted text files that can be placed under your normal source control mechanisms, stored in private or public locations such as Amazon S3. With CloudFormation, you can see exactly which AWS resources make up a stack. You retain full control and have the ability to modify any of the AWS resources created as part of a stack.

CloudFormation not only handles the initial deployment of your infrastructure and environments, but it can also manage the whole lifecycle, including future updates. During updates, you have fine-grained control and visibility over how changes are applied, using functionality such as change sets[2], rolling update policies[3] and stack policies[4].

The repository consists of a set of nested templates that deploy the following:

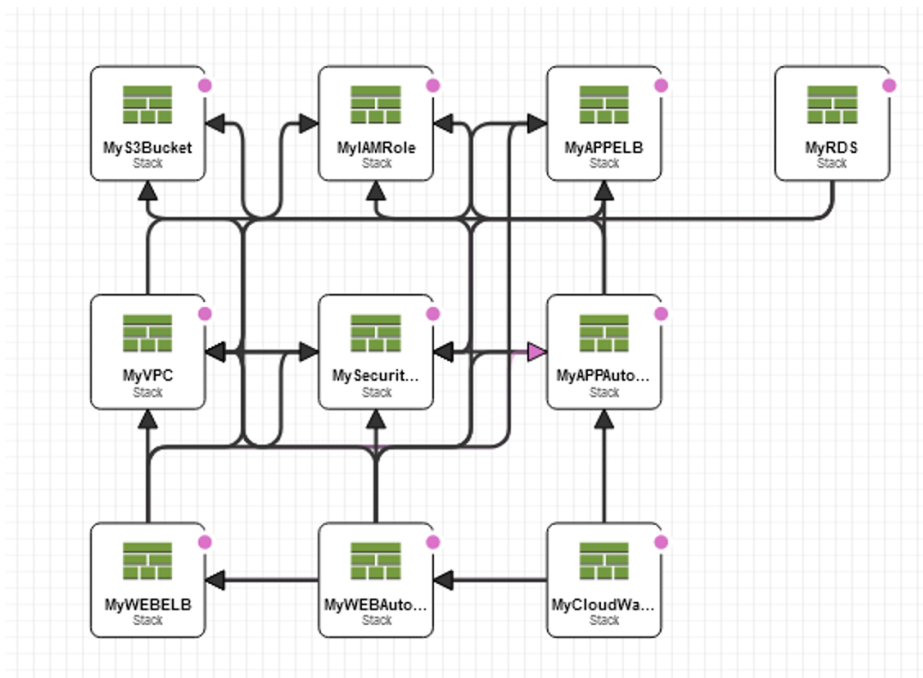
- A tiered VPC[5] with public and private subnets, spanning an AWS region.
- A highly available EC2 instance deployed across two Availability Zones[6] in an Auto Scaling[7] group.
- Two NAT gateways[8] to handle outbound traffic.
- An Load Balancer [9] to the public subnets to handle inbound traffic.
- RDS instance[10] with Multi-AZ to store application data.

The templates below are included in “DroneShuttle\_Project” repository and reference architecture:

### 1. master.yaml

- [master.yaml]
- This is the master template - deploy it to CloudFormation and it includes all of the nested templates automatically.

**View in Designer :**



### 2. webapp-iam.yaml

- [infrastructure/webapp-iam.yaml]
- This template deploys will create policy to allow EC2 instance full access to S3 & CloudWatch, and VPC Logs to CloudWatch.

### 3. webapp-s3bucket.yaml

- [infrastructure/webapp-s3bucket.yaml]
- This template deploys Backup Data Bucket with security data at rest and archive objects greater than 60 days, ELB logging, and Webhosting static content.

### 4. webapp-vpc.yaml

- [infrastructure/webapp-vpc.yaml]
- This template deploys a VPC with a pair of public and private subnets spread across two Availability Zones.
- It deploys an Internet gateway with a default route on the public subnets.
- It deploys 2 NAT gateways and default routes for them in the private subnets.

### 5. webapp-securitygroup.yaml

- [infrastructure/webapp-securitygroup.yaml]
- This template contains the security groups and Network ACLs required by the entire stack.

### 6. webapp-rds.yaml

- [infrastructure/webapp-rds.yaml]

- This template deploys a (Mysql) Relational Database Service with Multi-AZ. ~(20-30min to complete deploy).

**7. webapp-elb-appserver.yaml**

- [infrastructure/webapp-elb-appserver.yaml]
- This template deploys an Application Load Balancer that exposes our PHP APP services.

**8. webapp-autoscaling-appserver.yaml**

- [infrastructure/webapp-autoscaling-appserver.yaml]
- This template deploys an EC2 to the private subnets using an Auto Scaling group.

**9. webapp-elb-webserver.yaml**

- [infrastructure/webapp-elb-webserver.yaml]
- This template deploys a Webserver Load Balancer.

**10. webapp-autoscaling-webserver.yaml**

- [infrastructure/webapp-autoscaling-webserver.yaml]
- This template deploys an EC2 to the private subnets using an Auto Scaling group.

**11. webapp-cdn.yaml**

- [infrastructure/webapp-cdn.yaml]
- Run this template separately. CDN Deployment is time consuming ~(30-40min to complete deploy).

**12. webapp-cloudwatch.yaml**

- [infrastructure/webapp-cloudwatch.yaml]
- This template deploys Cloudwatch Service, CPU Utilization Alarm.

**13. webapp-route53.yaml**

- [infrastructure/webapp-route53.yaml]
- This template deploys Route 53 record set to update ELB Alias and CNAME CDN.



## 5. PRE DEPLOYMENT

### Assumptions:

1. You can launch this CloudFormation stack in the following Region in your account:
  - US East (N. Virginia) (Default in our case)
  - US East (Ohio)
  - US West (N. California)p
  - US West (Oregon)
2. You can launch this infrastructure in any other regions as well, but you just have to make an entry in [master.yaml] file in the “RegionMap” section with region name and AMI details, for more details please refer to the [master.yaml] file.
3. Creating and deploying the Cloud Formation stack with Admin permission of the your AWS Account.
4. AWS CLI is installed on your workstation/machine.
5. This set of templates deploys the following network design:  
Note : Change the VPC or subnet IP ranges as per your requirement.

Item	CIDR Range	Usable IPs	Description
VPC	10.0.0.0/16	65,536	The whole range used for the VPC and all subnets
Public Subnet 1	10.0.1.0/24	251	The public subnet in the first Availability Zone
Public Subnet 2	10.0.2.0/24	251	The public subnet in the second Availability Zone
Private Subnet 1	10.0.3.0/24	251	The private subnet in the first Availability Zone
Private Subnet 2	10.0.4.0/24	251	The private subnet in the second Availability Zone

### Prerequisites:

Before deploying of the infrastructure for Drone Shuttle architecture :

1. In AWS account must have one VPC available to be created in the selected region (our case us-east-1)
2. You have to update follow parameter value in master.yaml file:
  - Update your own office/home public IP address CIDR range.
  - Update your own S3 bucket URL of infrastructure code.
  - Update your AWS EC2 key pair.
  - Update your desired hosted zone.
  - Update with your required instance type
  - Upload all files in the “infrastructure” directory on your CloudFormation S3 bucket in the same region.
3. Optional : If needs to test/deploy route 53

- Installed Domain in Route 53.
- Installed Certificate (in your selected region & also in us-east-1) and update the CertARN in the master.yaml file.

## 6. DEPLOYMENT

In this section, will explain about the step by step process to deploy the infrastructure as per architecture design.

The process of deploying the infrastructure have following major steps as below :

### **Step 1.** Deploy the overall infrastructure (~ 25 - 35 minutes)

Run the master.yaml file from your workstation.

*Note :*

*stack-name; <env> value should be used as – dev / staging / prod  
template-body; path of the master file on the local machine*

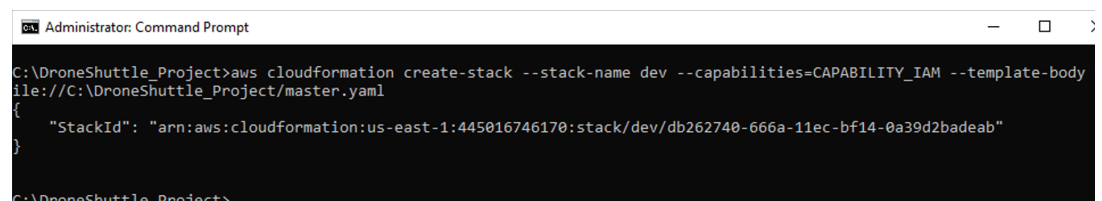
#### ➤ **To deploy the infrastructure :**

```
aws cloudformation create-stack \  
--stack-name <env> \  
--capabilities=CAPABILITY_IAM \  
--template-body  
file:///path_to_template//DroneShuttles_Project//master.yaml
```

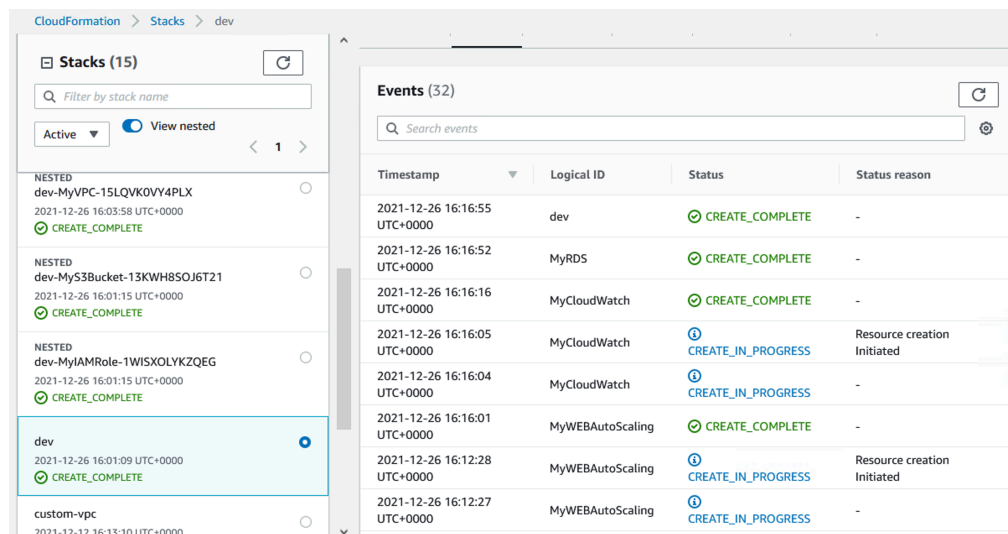
#### **Example :**

```
aws cloudformation create-stack --stack-name dev --  
capabilities=CAPABILITY_IAM --template-body  
file:///C:/DroneShuttle_Project/master.yaml
```

**Result :** After running the command, stack output looks like below :



```
Administrator: Command Prompt  
C:\DroneShuttle_Project>aws cloudformation create-stack --stack-name dev --capabilities=CAPABILITY_IAM --template-body  
file:///C:/DroneShuttle_Project/master.yaml  
{  
  "StackId": "arn:aws:cloudformation:us-east-1:445016746170:stack/dev/db262740-666a-11ec-bf14-0a39d2badeab"  
}
```



➤ **To update the infrastructure:**

```
aws cloudformation update-stack \
  --stack-name <env> \
  --capabilities=CAPABILITY_IAM \
  --template-body
  file:///path_to_template//DroneShuttles_Project//master.yaml
```

**Step 2.** To create CloudFront (~ 35 - 45 minutes)

If you don't want to create Cloudfront, then you can avoid Step2.

*Note :*

*stack-name; <envCDN> value should be used as – devCDN / stagingCDN / prodCDN*

➤ **To create CloudFront :**

```
aws cloudformation create-stack \
  --stack-name <envCDN> \
  --capabilities=CAPABILITY_IAM \
  --template-body
  file:///path_to_template//DroneShuttles_Project//infrastructure//webapp-cdn.yaml
```

➤ **To update CloudFront :**

```
aws cloudformation update-stack \
  --stack-name <envCDN> \
  --capabilities=CAPABILITY_IAM \
  --template-body
  file:///path_to_template//DroneShuttles_Project//infrastructure//webapp-cdn.yaml
```

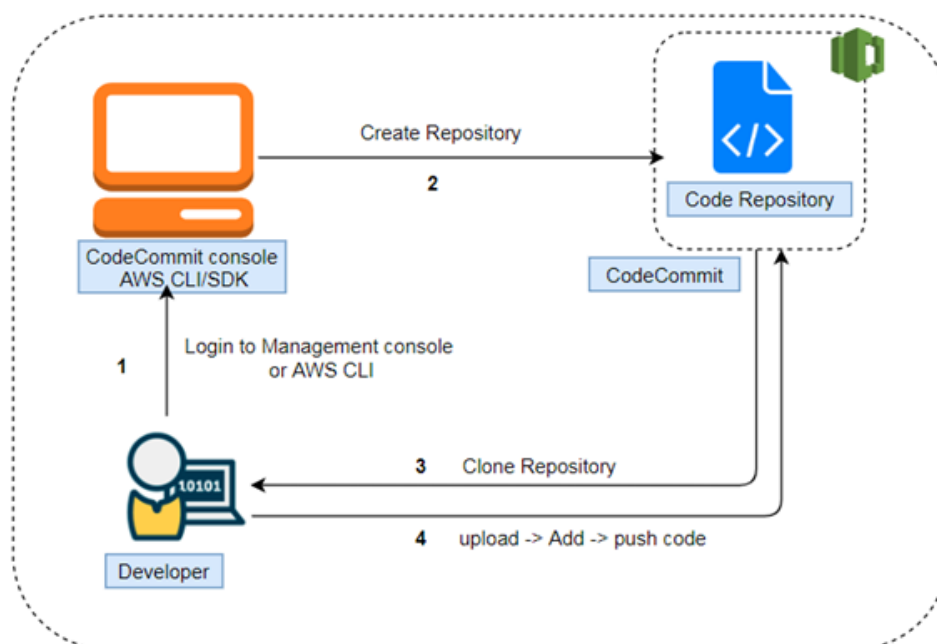
**Step 3.** To Setup and Build pipeline on AWS for DevOps Team  
In this step, will build a four step CI/CD pipeline using AWS native tools.

Use below services to build CI/CD pipeline :

- **Source Control repository** : AWS CodeCommit
- **Build** : AWS CodeBuild
- **Deployment** : AWS CodeDeploy
- **CI/CD Pipeline** : AWS CodePipeline
- **Notification** : AWS SNS

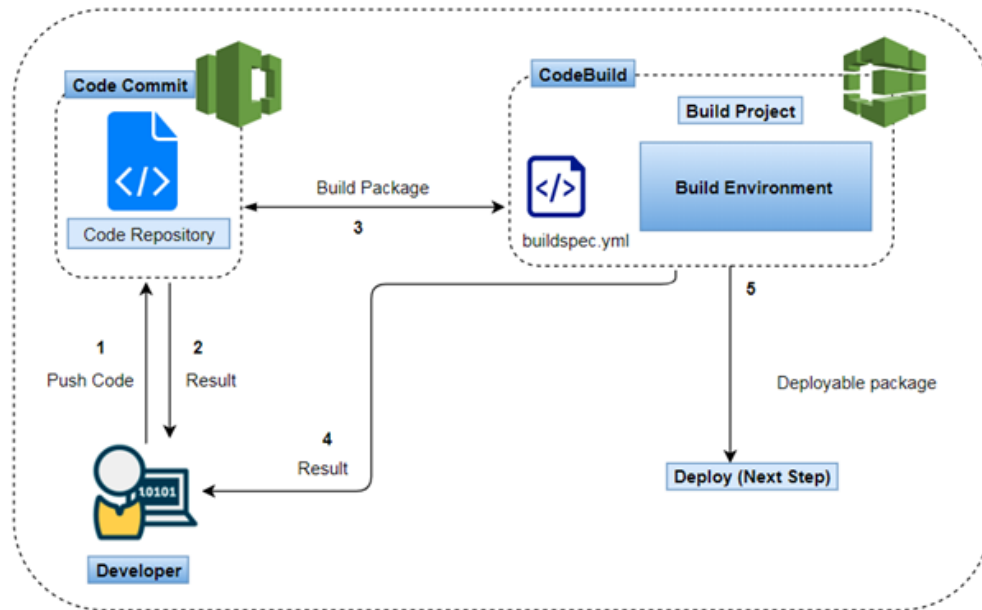
➤ **Source Control repository – AWS CodeCommit**

To Setup ; <https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up.html>



➤ **Building Package – AWS CodeBuild**

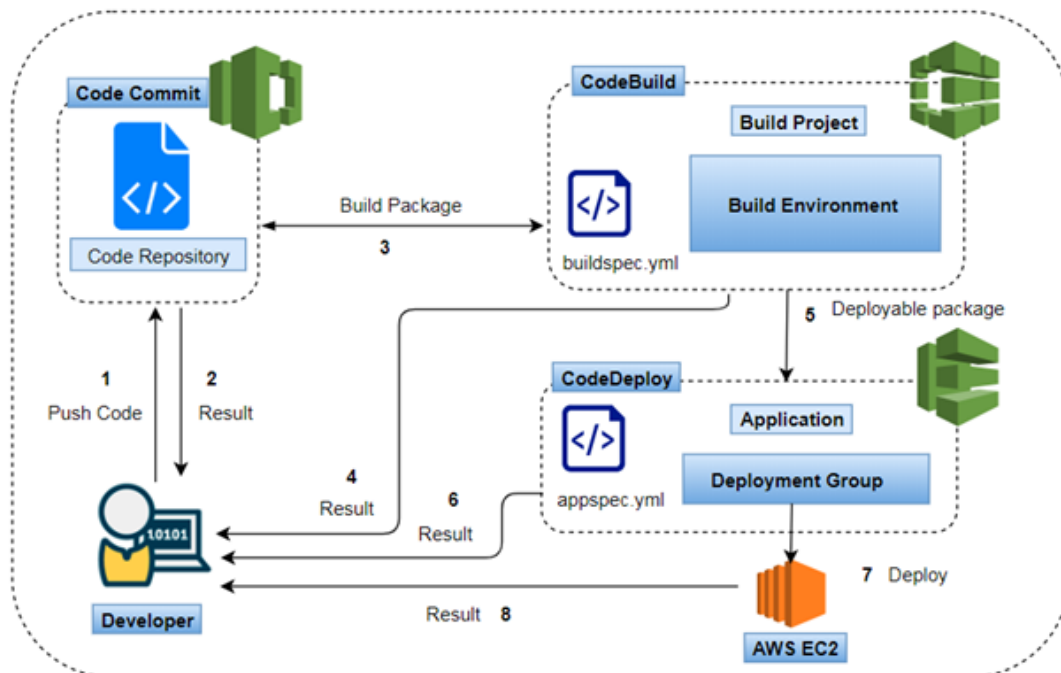
To Plan a build in AWS CodeBuild;  
<https://docs.aws.amazon.com/codebuild/latest/userguide/planning.html>



### ➤ Deployment – AWS CodeDeploy

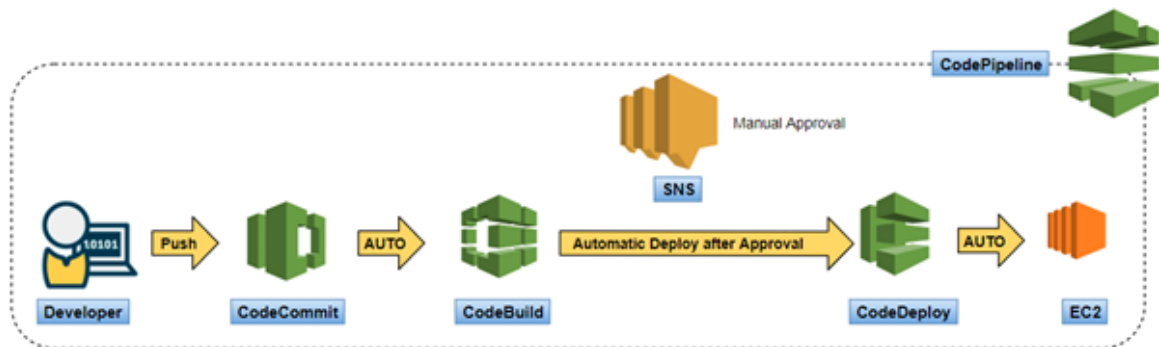
To deploy, CodeDeploy tutorials ;

<https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials.html>



➤ **To Create CI/CD pipeline – AWS CodePipeline**

To create CI/CD pipeline; <https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/>



## 7. ROLLBACK

To rollback the Infrastructure and CloudFront project from CloudFormation, run the below commands :

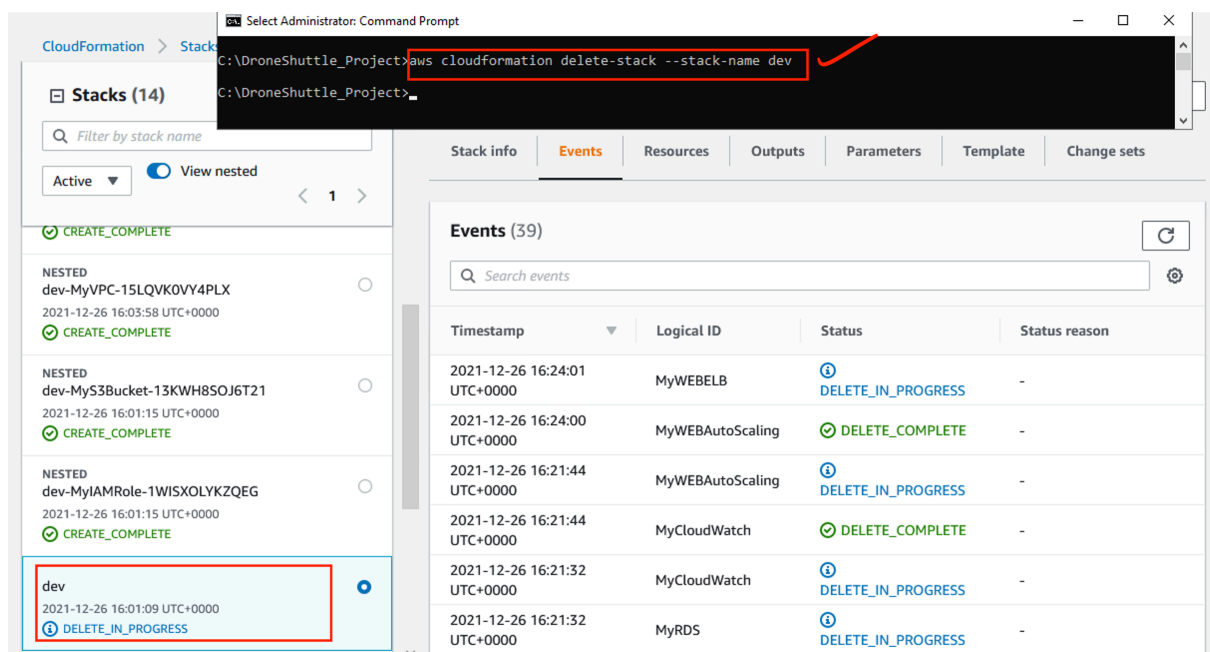
### Step 1 : To delete the Infrastructure Project

```
aws cloudformation delete-stack --stack-name <env>
```

*Note :*

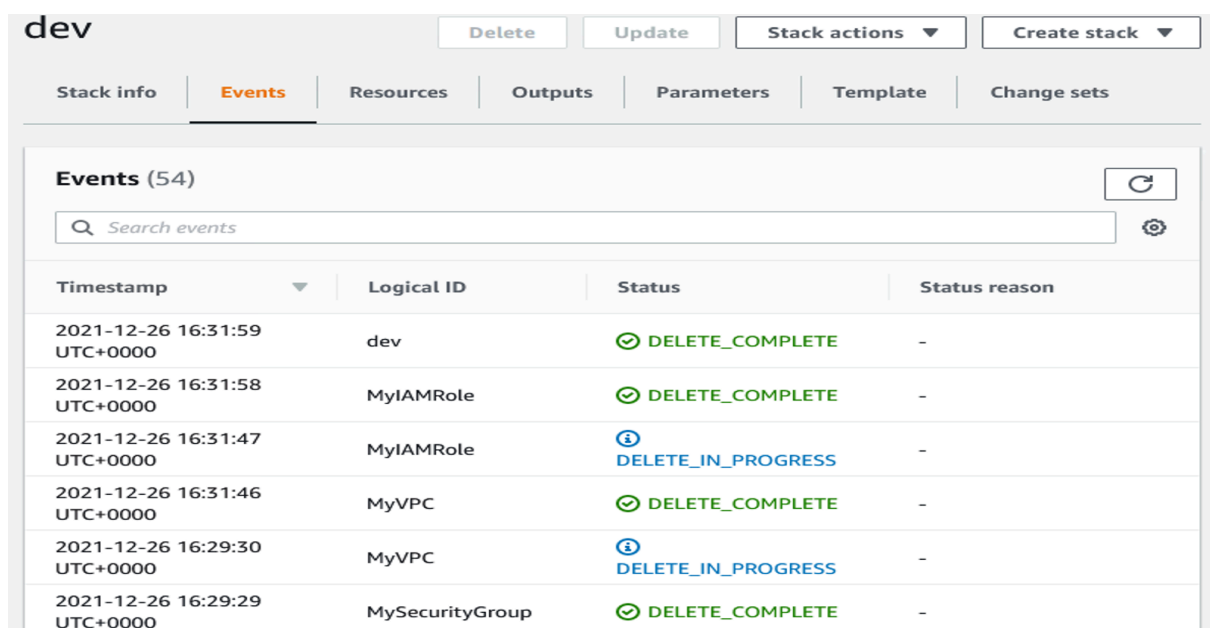
*stack-name; <env> value should be used as – dev / staging / prod  
template-body; path of the master file on the local machine*

**Result :** After running the delete stack command, output looks like :



The screenshot shows the AWS CloudFormation console. On the left, the 'Stacks (14)' list shows the 'dev' stack in the 'DELETE\_IN\_PROGRESS' state. On the right, the 'Events (39)' tab for the 'dev' stack shows a list of events. The command prompt window shows the command 'aws cloudformation delete-stack --stack-name dev' being executed. The console shows the 'dev' stack in the 'DELETE\_IN\_PROGRESS' state. The 'Events' tab for the 'dev' stack shows a list of events, including the successful deletion of the stack.

Timestamp	Logical ID	Status	Status reason
2021-12-26 16:24:01 UTC+0000	MyWEBELB	DELETE_IN_PROGRESS	-
2021-12-26 16:24:00 UTC+0000	MyWEBAutoScaling	DELETE_COMPLETE	-
2021-12-26 16:21:44 UTC+0000	MyWEBAutoScaling	DELETE_IN_PROGRESS	-
2021-12-26 16:21:44 UTC+0000	MyCloudWatch	DELETE_COMPLETE	-
2021-12-26 16:21:32 UTC+0000	MyCloudWatch	DELETE_IN_PROGRESS	-
2021-12-26 16:21:32 UTC+0000	MyRDS	DELETE_IN_PROGRESS	-



The screenshot shows the AWS CloudFormation console for the 'dev' stack. The 'Events (54)' tab shows a list of events. The command prompt window shows the command 'aws cloudformation delete-stack --stack-name dev' being executed. The console shows the 'dev' stack in the 'DELETE\_IN\_PROGRESS' state. The 'Events' tab for the 'dev' stack shows a list of events, including the successful deletion of the stack.

Timestamp	Logical ID	Status	Status reason
2021-12-26 16:31:59 UTC+0000	dev	DELETE_COMPLETE	-
2021-12-26 16:31:58 UTC+0000	MyIAMRole	DELETE_COMPLETE	-
2021-12-26 16:31:47 UTC+0000	MyIAMRole	DELETE_IN_PROGRESS	-
2021-12-26 16:31:46 UTC+0000	MyVPC	DELETE_COMPLETE	-
2021-12-26 16:29:30 UTC+0000	MyVPC	DELETE_IN_PROGRESS	-
2021-12-26 16:29:29 UTC+0000	MySecurityGroup	DELETE_COMPLETE	-

**Step 2 : To delete CloudFront**

```
aws cloudformation delete-stack --stack-name <envCDN>
```

*Note :*

*stack-name*; <envCDN> value should be used as – devCDN / stagingCDN / prodCDN

**Step 3 : Delete CI/CD Pipeline**



## ANNEXURE - I

### Reference:

- [1] AWS CloudFormation; <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-what-is-concepts.html>
- [2] change sets; <https://aws.amazon.com/blogs/aws/new-change-sets-for-aws-cloudformation/>
- [3] rolling update policies; <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-updatepolicy.html>
- [4] stack policies; <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/protect-stack-resources.html>
- [5] VPC; [http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC\\_Introduction.html](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html)
- [6] Availability Zones; <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>
- [7] Auto Scaling; <https://aws.amazon.com/autoscaling>
- [8] NAT gateways; <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-nat-gateway.html>
- [9] Load Balancer ; <https://aws.amazon.com/elasticloadbalancing/applicationloadbalancer/>
- [10] RDS Instance Multi-AZ; <https://aws.amazon.com/rds/features/multi-az/>