DG DIGIT
Unit D.2

D04.01 – Release of eGovERA v2.0.0 including foundation and upscale solutions for business users in a DIGIT externally accessible platform –

Documentation about the eGovERA porting in AWS

Specific Contract n° 640 under Framework Contract n° DI/07624-00 - ABCIV Lot 3: ISA² - 2016.32 European Interoperability Architecture

Date: *13/04/2022*
Doc. Version: 1.0

This template is based on PM² v2.5

DOCUMENT CONTROL INFORMATION

Settings	Value
Document Title:	Documentation* about the eGovERA porting in AWS *Part of the deliverable →D04.01 – Release of eGovERA v2.0.0 including foundation and upscale solutions for business users in a DIGIT externally accessible platform*
Project Title:	Specific Contract n° 640 under Framework Contract n° DI/07624-00 - ABCIV Lot 3: ISA² - 2016.32 European Interoperability Architecture
Document Author:	Valerio MEZZAPESA
EC Project Officer:	Raul Mario ABRIL JIMENEZ
External Contractor Project Manager:	Valerio MEZZAPESA
Doc. Version:	1.00
Sensitivity:	Internal
Date:	13/04/2022

Disclaimer

REVISION HISTORY

The following table shows the development of this document.

Version	Date	Description	Created by	Reviewed by
1.0	13.04.2021	First Deloitte Cloud Migration and Managed Service Version: Architectural Components Presentation and description.	Costantino De Petrillo Mattia Galli Jawad Lahouiny	Simone FRANCIOSI Valerio MEZZAPESA

Table of content

1 INTRODUCTION

1.1 Purpose of this document

The scope of this document is to describe the architecture of the application deployment in the AWS Cloud. The **target audience** is the European Commission IT administrators. The objective of this architecture is to create a host environment for eGovERA application and allow continuous development of its source code through an Amazon Code Pipeline

1.2 Release date

This version of the eGovERA Portal has been released on *13 April 2022*.

1.3 List of acronyms used in this document

The following table summarises the terms and acronyms mentioned in the document text for ease of reference.

Acronym	Definition
DT	Data Transfer
DNS	Domain Name Server
S3	Simple Storage Service
SNS	Simple Notification Service
IAM	Identity and Access Management
MFA	Multi Factor Authentication

2 THE EGOVERA PORTAL

2.1 AWS Architecture overview

The environment is in AWS Region eu-west-1 (Ireland). Administrators of the host environment are the Infrastructure as a Code Developers who can update the Cloud Formation stack templates of the AWS Environment via uploading the new templates in an AWS S3 bucket.

The Cloud Formation stack template is created with the use of three (3) files, *Master.yaml*, *Environment.yaml, Development.yaml*. The relation of these files is father-son with *Master.yaml* being the father file.

Application administrators can initially upload and then maintain the application's source code through the Application Code pipeline using the AWS service Code Commit (Git-based). Before the source code is uploaded by AWS Code Deploy to update eGovERA application, an email notification is sent (via an Amazon SNS topic) to the administrator using Code Pipeline's Manual Approval stage.

The Application Code Pipeline saves the updated code in the specified AWS Code Commit repository. After the manual approval by the administrator, AWS Code Deploy uses the CloudFormation stack combined with the AWS Elastic Beanstalk service to create the eGovERA application's environment with the last updated source code.

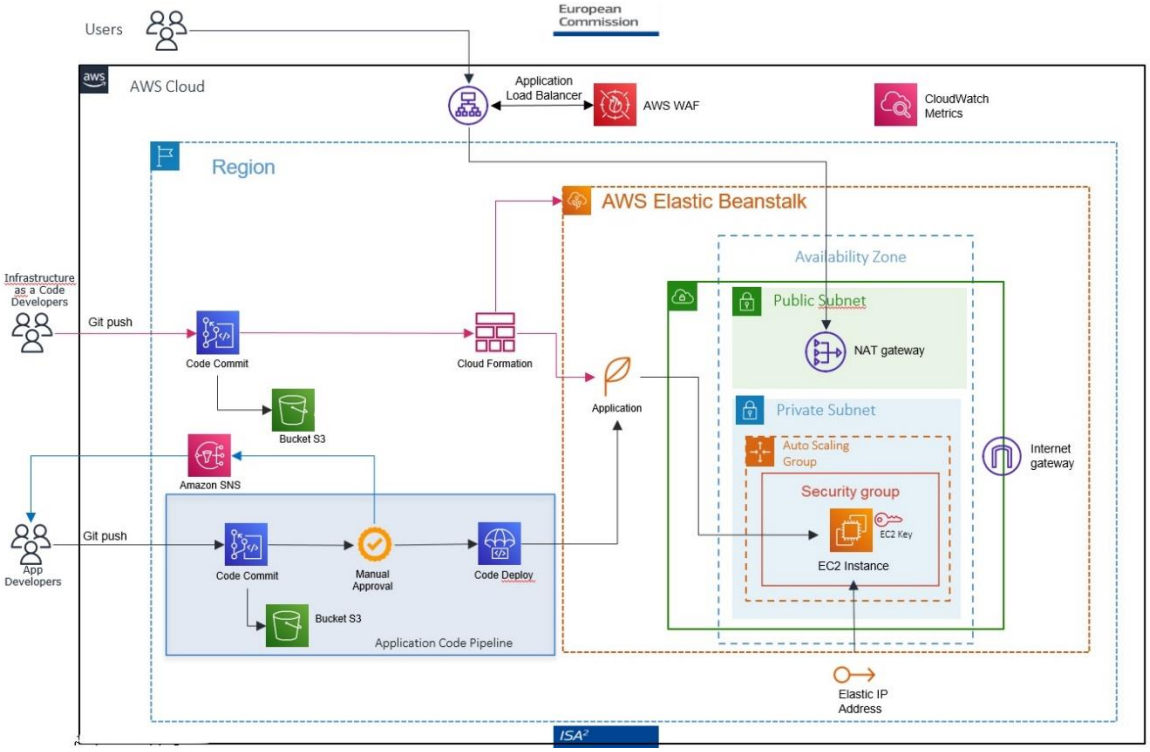

Figure 1 Infrastructure overview

2.2 AWS Detailed architecture overview

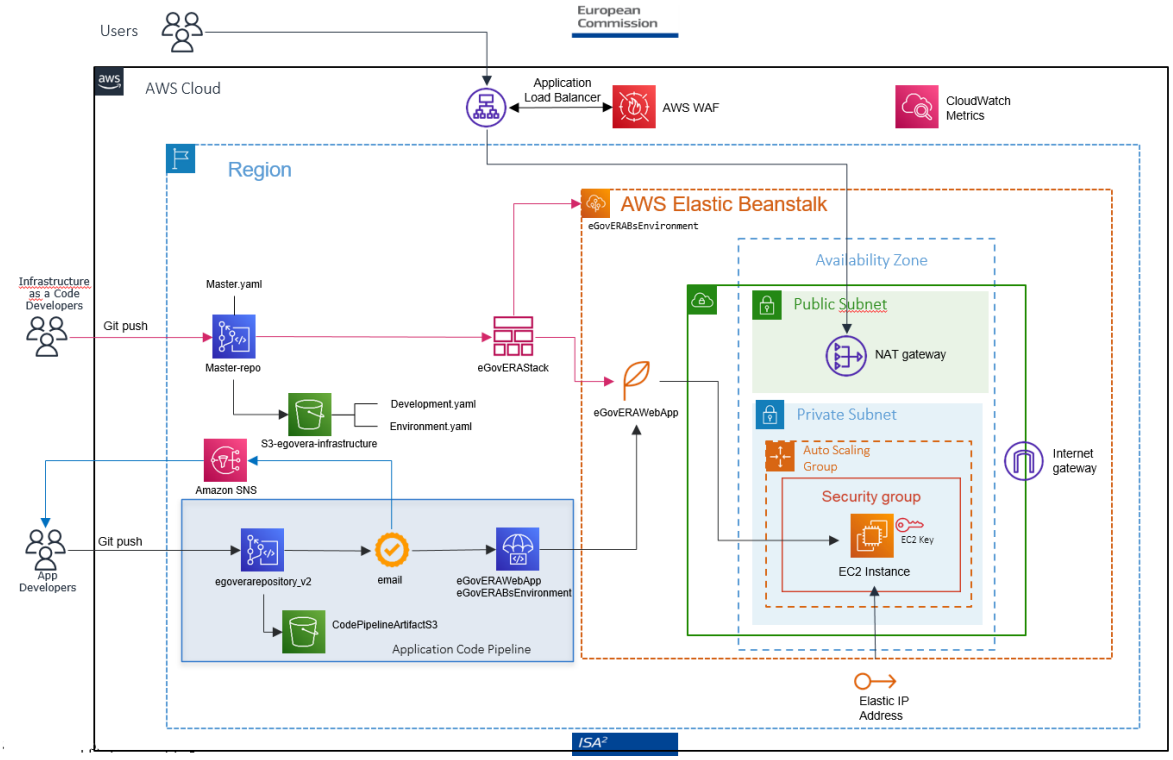

Figure 2 Detailed architecture overview

Master.yaml File

Master.yaml is used as a father file of *Environment.yaml* and *Development.yaml* templates. It is used by AWS Cloud Formation to create the initial Cloud Formation stacks of the infrastructure as well as the developers' pipeline for the source code. *Master.yaml* file also implements the dependency of the created stacks. Development stack depends on the Environment stack which is created first.

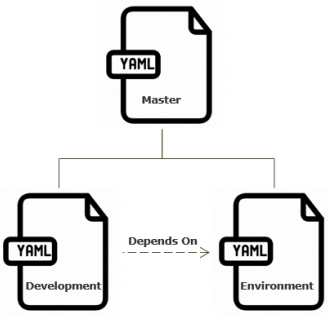

Figure 3 Templates of the architecture

Environment.yaml File

Environment.yaml this is the first nested stack created by Cloud Formation, implements the AWS Cloud Infrastructure. Consists of the AWS Elastic Beanstalk environment which will host the application.

- **Parameters**:
    - WebApplicationName
    - BeanstalkEnvironmentName
    - InstanceType
    - Email
- **Resources**:
    - VPC
    - Subnet
    - InternetGateway
    - RouteTable
    - NatGateway
    - Application
    - Environment
    - EnvironmentConfiguration
        - InstanceTypes
        - VPC
        - Subnets
        - ElasticLoadBalancer Subnets
        - Notification Endpoint
        - ServiceRole
        - LoadBalancerType
        - Autoscaling: DisableIMDSv1
        - Autoscaling: IamInstanceProfile
        - ManagedActionsEnabled
        - PreferredStartTime
        - UpdateLevel

Development.yaml File

Development.yaml is the second nested stack created by Cloud Formation, implements the AWS Code Pipeline. The pipeline is composed of 3 steps: commit, manual approval, deploy. After this step, the source code will be deployed into the running environment.

- **Parameters**:
    - BranchName
    - eGovRepositoryName
    - eGovRepositoryDescription
    - ApplicationName
    - EnvironmentName

- ManualApproval
- SNSTopic

- **Resources**
  - CodePipelineArtifactS3
  - CodePipelineArtifactS3Policy
  - AmazonCloudWatchEventRule
  - AppPipeline
    - Stages
      - Source: SourceCodeCommit
      - Approval: ManualApproval
      - Deploy: ElasticBeanstalkApp
  - ArtifactStore

2.3 AWS Networking

To connect to eGovERA application traffic will first be received by Amazon Route 53 which will send it through the Application Load Balancer. After traffic is allowed by AWS WAF goes to the VPC and the Application Load Balancer that is targeting the EC2 Instance allows the users to connect to the application.

Service	Components	Description
Amazon Route 53	1 Hosted Zone	Cloud DNS web server service
Amazon VPC	1 Availability Zone (AZ)	Virtual Private Cloud
Elastic Load Balancer	Application Load Balancer	Distributes incoming application traffic
Elastic IP	Static IPv4 address	EIP is attached with an EC2 after its creation
ENI		Elastic Network Interfaces

Amazon Route 53: 1 Hosted Zone (cloud DNS web service)
Load Balancer: 1 Application Load Balancer
Amazon Virtual Private Cloud: 1 Internet Gateway, 1 Public Subnet, 1 Private Subnet
Public Subnet: 1 NAT Gateway
Private Subnet: 1 Security Group
Security Group: associated with the EC2 instances

AWS VPC Components	
Availability Zone	eu-west-1a
Internet Gateway	Connects eGovERA application environment to the Internet

Availability Zone Components	
Private Subnet	NAT Gateway
Public Subnet	Security Group associated with Ec2 instances

EIP Address Components	
Name	eGovERABsEnvironment
Allocated IPv4 address	(autogenerated)
Type	(autogenerated)
Allocation ID	(autogenerated)
Reverse DNS record	(autogenerated)
Associated instance ID	(autogenerated)
Network interface owner account	(autogenerated)

Network Interfaces	
Name	eGovERA-ENI
Name	eGovERA-ENI
Network interface ID	(autogenerated)
Subnet ID	(autogenerated)
VPC ID	(autogenerated)
Availability Zone	
Security groups	(autogenerated)
Interface Type	(autogenerated)
Description	-
Instance ID	(autogenerated)
Status	in use
Public IPv4 address	(autogenerated)
Primary private IPv4 address	(autogenerated)
Owner	931566614796

2.4 AWS Storage

The following storages are used to download the output of the eGovERA application and to store the source code of the pipeline.

AWS Data Transfer	AWS S3 Bucket
DT inbound: Internet 10 GB per month	DT inbound 5 GB per month
DT outbound: Internet 10 GB per month	DT outbound 5 GB per month
DT Intra-Region: 10 GB per month	-

2.5 AWS Compute

This Environment is a PaaS (Platform as a Service) that uses Elastic Beanstalk to deploy instances.

The services implemented for Production/Non-Production environment are:

AWS Code Pipeline: 4 active pipelines used per account per month

Amazon EC2 Auto Scaling: monitor the health of running instance

Name: awseb-e-faeaqqmqtc-stack-AWSEBAutoScalingGroup-1560E2J4XYJ2W

Launch Template: AWSEBEC2LaunchTemplate_wZMm5ADWuYEF | Version 1

Instances	Status	Desired capacity	Min	Max	Availbility Zone
1	-	1	1	1	eu-west-1a

Amazon EC2: Instance Type: m5.xlarge for Prod and t4g.large for Non-Prod

Storage Amount: 30 GB

Operating system: Linux

Pricing strategy: On-Demand Instances

Production Instance type:

Instance Size	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)***	EBS Bandwidth (Mbps)
m5.xlarge	4	16	EBS-Only	Up to 10	Up to 4,750

M5 instances are the latest generation of General-Purpose Instances powered by Intel Xeon® Platinum 8175M processors. This family provides a balance of compute, memory, and network resources, and is a good choice for many applications.

Non-Production Instance type:

Instance Size	vCPU	Memory (GiB)	Baseline Performance / vCPU	CPU Credits Earned / Hr	Network Burst Bandwidth (Gbps)***
t4g.large	2	8	30%	36	Up to 5

Amazon EC2 T4g instances are powered by Arm-based AWS Graviton2 processors and deliver up to 40% better price performance over T3 instances for a broad set of burstable general-purpose workloads.

T4g instances accumulate CPU credits when a workload is operating below baseline threshold. Each earned CPU credit provides the T4g instance the opportunity to burst with the performance of a full CPU core for one minute when needed. T4g instances can burst at any time for as long as required in Unlimited mode.

2.6 AWS Identity and access management (IAM)

AWS Identity and Access Management (IAM) provides fine-grained access control across all AWS services.

Username	Groups	MFA	Password age	Active Key age
matgalli@deloitte.it	Admins	Virtual	4 March 2022	4 March 2022
goraiopoulos@deloitte.gr	Admins	Virtual	4 March 2022	4 March 2022
cdepetrillo@deloitte.it	Admins	Virtual	4 March 2022	
Abrilra	Admins	None	28 February 2022	
staramas@deloitte.gr	Developers	Virtual	31 March 2022	

Group name	Users	Permissions	Creation time
Admins	4	AdministratorAccess	25 February 2022
Developers	1	AWSCodeCommitFullAccess, CodePipelineFullAccess,IAMFullAccess, AWSCodePipelineApproverAccess, AWSCodePipelineFullAccess	31 March 2022

Role name	Trusted entities
AmazonCloudWatchEventRole	AWS Service: events
CloudFormationRole	AWS Service: Cloud Formation, Code Deploy
CodePipelineServiceRole	AWS Service: codepipeline
AWSServiceRoleForAutoScaling	AWS Service: autoscaling

2.7 Security AWS WAF

AWS WAF is a web application firewall that lets you monitor the HTTP(S) requests that are forwarded to an Amazon Application Load Balancer.

AWS WAF also lets you control access to your content. Based on conditions that you specify, such as the IP addresses that requests originate from or the values of query strings, the service associated with your protected resource responds to requests either with the requested content or with an HTTP 403 status code (Forbidden). You can also configure CloudFront to return a custom error page when a request is blocked.

The Web Application Firewall (WAF) consists of a Web Access Control List named ACLeGovERA that contains the following rules:

Vendor	Rule Name	Priority	WCU Capacity
AWS	Core rule set	0	700
AWS	Know bad inputs	1	200
AWS	Amazon IP reputation list	2	25
AWS	Anonymous IP list	3	50
AWS	Admin Protection	4	100

3 INFRASTRUCTURE AS A CODE

3.1 AWS Cloud formation templates

The Pipeline to create the Cloud Formation Stacks consists of two (2) Stages, the first one is the Source stage and the second the Deploy stage.

3.2 Source stage

Configuration of this stage defines the Action provider (AWS Code Commit), branch and repository names. In this stage also is defined how the events are captured to be executed, for this Amazon Cloud Watch Events service is used. The Master Pipeline to create the Cloud Formation stacks can be changed/updated only manually ("Release change" button).

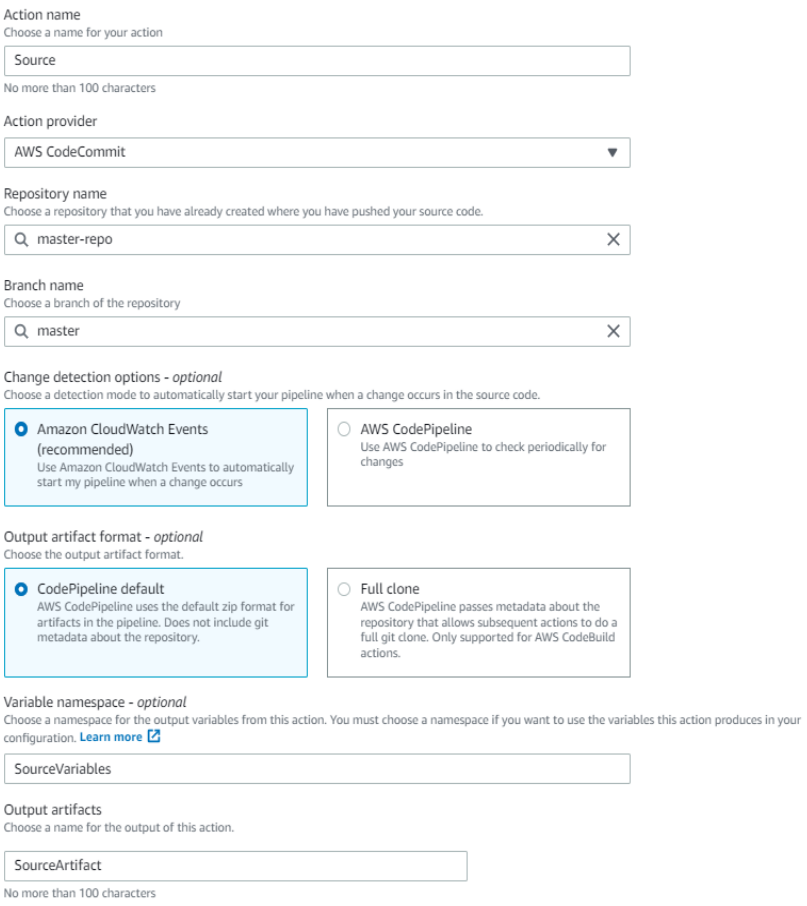

Figure 4 Source stage

3.3 Deploy Stage

In this stage the Environment (eGovERABsEnvironment) is first created due to the declared dependency ("DependsOn: NestedStackB"). After creation of the Environment, the creation of AWS Code Pipeline for source code begins.

In Deploy Stage the AWS Elastic Beanstalk Autoscaling Group, including the EC2 instance, is first created along with the Security Group as defined in the Elastic Beanstalk configuration template (*Environment.yaml*). After that the Development pipeline is created to import eGovERA app source code to eGovERABsEnvironment.

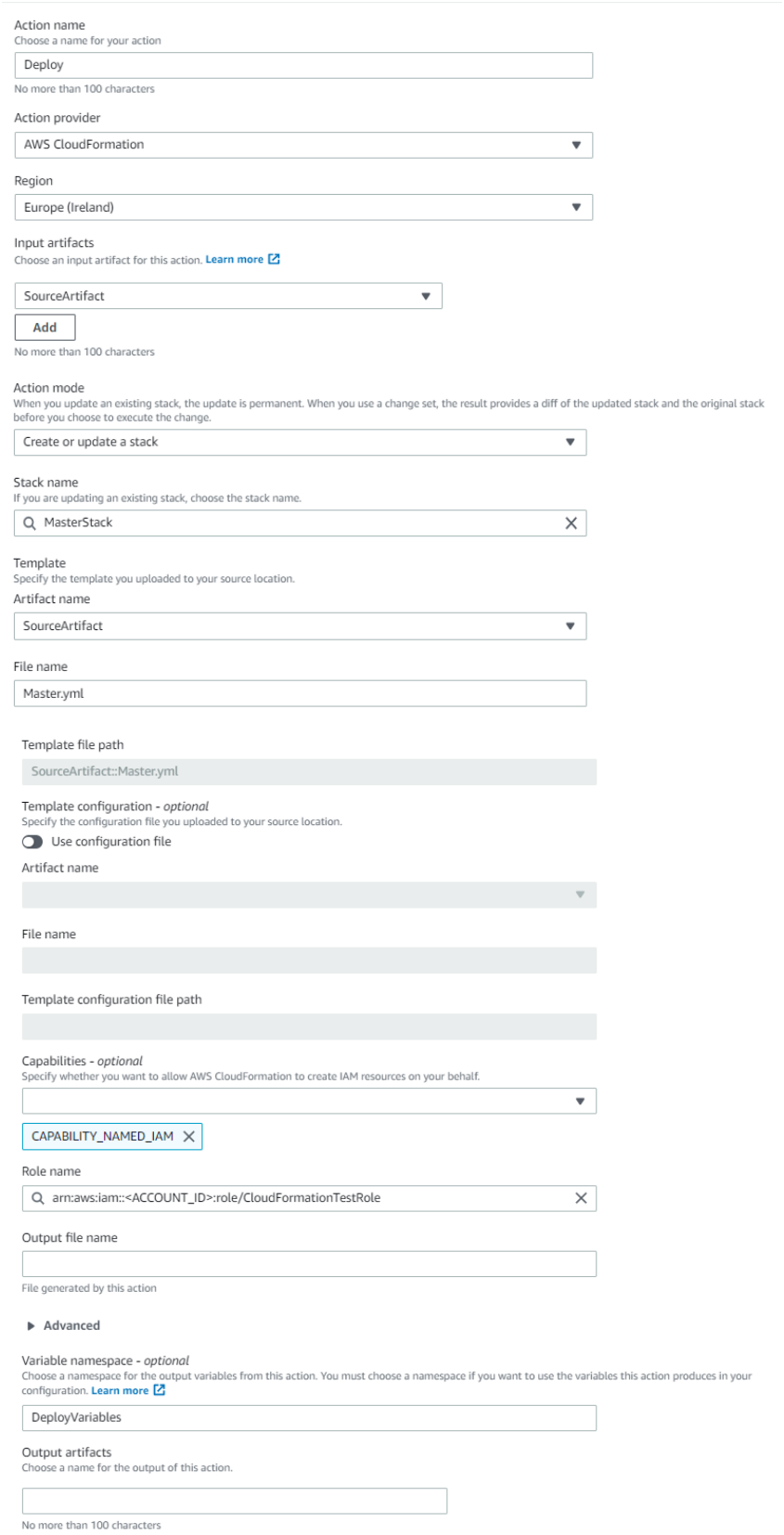

Figure 5 Deploy stage

3.4 Usage of development pipeline for source code updates

The codecommit-events-pipeline for Development consists of three (3) Stages, first Source Code Commit stage then a Manual Approval Stage and third the Deploy Stage.

Source code commit stage

The first stage is configured to use Amazon Cloud Watch Events to capture commit actions that occur in the AWS Code Commit Repository (egoverarepository_v2) where the source code of eGovERA app resides. During the initial start of the app the repository will be empty. After the first commit of the code ("git push") inside egoverarepository_v2, this stage is triggered, and the pipeline procedure starts.

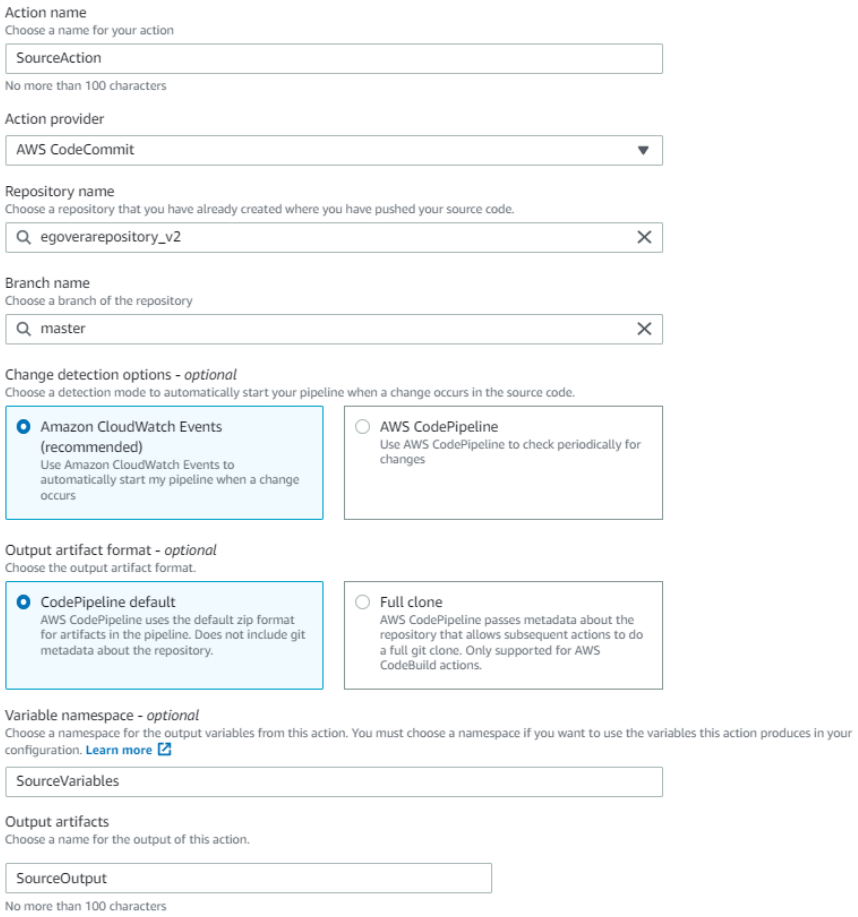

Figure 6 Source code commit stage

3.5 Manual approval stage

In the second stage of Application Code Pipeline an email is sent to the designated account. For this procedure to complete and the updated code to be deployed in Beanstalk Environment, an approval is needed in the source code pipeline stage ("ManualApproval"). The approval can be given through clicking the URL sent in the email. This redirects to AWS Code Pipeline console inside Development Pipeline so the administrator can approve the commit there.

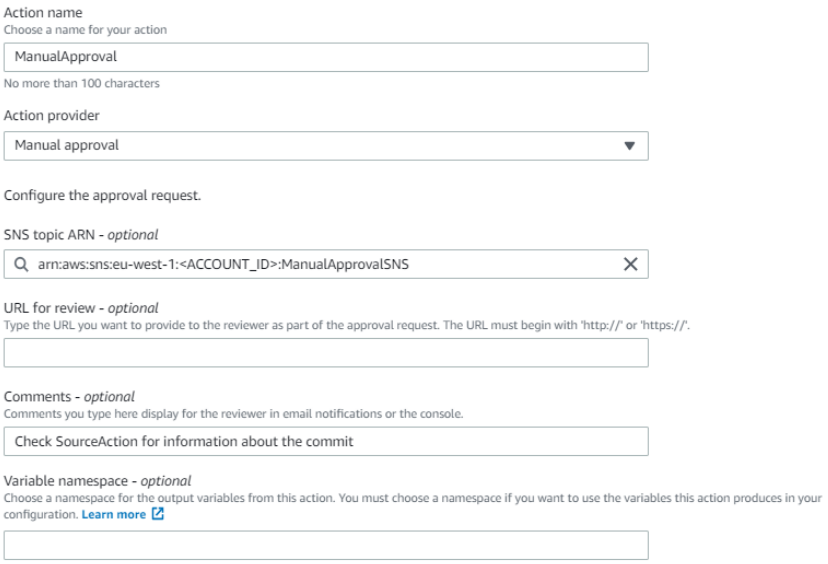

Figure 7 Manual approval stage

3.6 Deploy source stage

In this last stage the committed source code updates the code running in Elastic Beanstalk Environment with the updated one. This is achieved by providing the template with the application name (eGovERAWebApp), that is created by AWS Elastic Beanstalk, and the environment of the specified application (eGovERABsEnvironment).

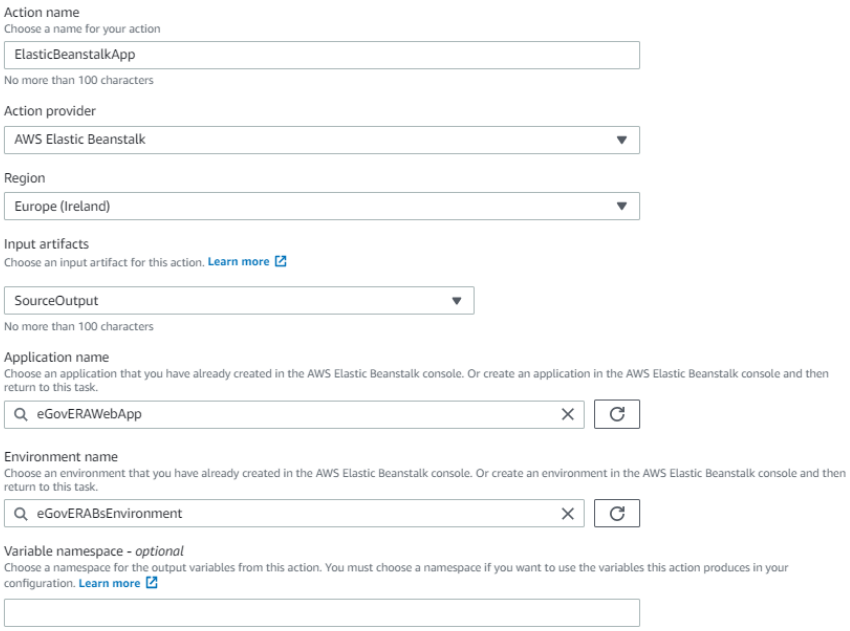

Figure 8 Deploy stage

4 CONFIGURATION ACTIVITIES

In this section is described the different roles and policies that are assumed to create the environment using Cloud Formation.

An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. These roles are used to delegate access to users, applications, or services that don't normally have access to your AWS resources.

An IAM role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when assuming a role, it provides you with temporary security credentials for your role session.

AWS service role

A role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions required for the service to access the AWS resources that it needs.

4.1 Creation by hand

- Create IAM Role:
    a. AWSServiceRoleForAmazonSSM:
    i. Policy attached to System Manager – Inventory and Maintenance Windows
    1. AmazonSSMServiceRolePolicy

    b. AWSServiceRoleForAutoScaling:
    i. Policy attached to EC2 Auto Scaling:
    1. AutoScalingServiceRolePolicy

    c. aws-ElasticBeanstalk-Ec2-Role:
    i. Policy attached to EC2:
    1. AWSElasticBeanstalkWebTier
    2. AWSElasticBeanstalkMulticontainerDocker
    3. AWSElasticBeanstalkWorkerTier

    d. aws-ElasticBeanstalk-Service-Role:
    i. Policy attached to ElasticBeanstalk:
    1. AWSElasticBeanstalkEnhancedHealth
    2. AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy

    e. CodePipelineServiceRole:
    i. Policy attached to Codepipeline: AWS-CodePipeline-Service (source code)

    f. AmazonCloudWatchEventRole:
    i. Policy attached to CloudWatch: aws-Pipeline-Execution (source code)

    g. CloudFormationRole:
    i. Policy attached to CloudFormation:
    1. AmazonS3FullAccess
    2. ElasticLoadBalancingFullAccess
    3. AmazonVPCFullAccess
    4. AWSWAFFullAccess
    5. AmazonSNSRole
    6. AWSCodePipelineFullAccess

7. CloudWatchEventsFullAccess
8. AdministratorAccess-AWSElasticBeanstalk
9. AWSCloudFormationFullAccess
10. IAM-PassRoleToCloudWatch (source code)

- Create an S3 bucket
- Upload Environment.yaml and Development.yaml on s3 bucket created
- Check the TempalteURL on Master.yml (change it with current object URL)
- Create 2 repository on Code Commit
    1. master-repo: To add Master.yml file, generate credentials, run command git
    2. egoverarepository_v2: To add the source code of egovera (unzipped)
- Create Git credential
- Clone repository to your pc
- Insert egovera source code by using Git commands
- git add -A / git commit -m "adding egovera source code" / git push
- Insert Master.yml by using Git commands.
- git add -A / git commit -m "adding Master.yml" / git push

4.2 Create Pipeline MasterPipeline

1. Check eu-west-1 Ireland
2. Create a Default VPC
3. Select Code Commit as source provider with the repository that contains Master.yaml, branch name master
4. Skip build stage
5. Select CloudFormation as deploy provider
6. Create or Update stack
7. Stack name: eGovERAStack
8. Artifact name: SourceArtifact
9. File name: Master.yml
10. Capabilities: CAPABILITY_NAMED_IAM
11. Role name: CloudFormationRole
12. The source code developer pipeline will create automatically with these stages: Code Commit, Manual approval, Code Deploy (remember to approve).
13. Only for first release check if the source code files are inside the Code Commit repository egoverarepository_v2.
14. Wait for the notifications (manual approval), approve.
15. Associate Load Balancer to WAF

4.3 Management and Maintenance

Inside the *Master.yaml* file you will find the creation of the nested stack that are connected via TemplateURL.

NestedStackA is referred to the Development.yaml

NestedStackB is referred to the Environment.yaml

The URL is the only parameters that can be changed:
NestedStackA
TemplateURL: "https://stack-nidification.s3.eu-west-1.amazonaws.com/Development.yml"
NestedStackB:
TemplateURL: https://stack-nidification.s3.eu-west-1.amazonaws.com/Environment.yaml

Master.yaml template		
Nested Stacks	Refers to	TemplateURL
NestedStackA	Development.yaml	https://stack-nidification.s3.eu-west-1.amazonaws.com/Development.yml
NestedStackB	Environment.yaml	https://stack-nidification.s3.eu-west-1.amazonaws.com/Environment.yaml

Inside the *Development.yaml* file you will find all the information about the source code development pipeline.

The only parameters that can be changed are the following:

Development.yaml file components:

1. Parameters
    - 1.1. BranchName: CodeCommit branch name default: **master**
    - 1.2. eGovRepositoryName: CodeCommit repository name of source egovera default: **egoverarepository_v2**
    - 1.3. eGovRepositoryDescription: CodeCommit repository Description default: **Source code of egovera version 2**
    - 1.4. ApplicationName: CodeDeploy application name default: **eGovERAApp**
    - 1.5. EnvironmentName: CodeDeploy environment name default: **eGovERABsEnvironment**
    - 1.6. ManualApproval: Insert the comment of your Manual Approval default: **Check SourceAction for information about the commit**
    - 1.7. SNSTopic: Insert the arn of your SNS default: **arn:aws:sns:eu-west-1:<ACCOUNT_ID>:ManualApprovalSNS**

Development.yaml template		
Parameters	**Default**	**Description**
BranchName	master	Code Commit branch name
eGovRepositoryName	egoverarepository_v2	Code Commit Repository name of eGovERA source code
eGovRepositoryDescription	Source code of egovera version 2	Code Commit repository Description
ApplicationName	eGovERAApp	Code Deploy application name
EnvironmentName	eGovERABsEnvironment	Code Deploy environment name
ManualApproval	Check Source Action for information about the commit	Insert the comment of your Manual Approval
SNSTopic	arn:aws:sns:eu-west-1:931566614796:ManualApprovalSNS	Insert the arn of your SNS

All the information concerning the creation of the environment is contained in the *Environment.yaml* file.

The only parameters that can be changed are the following:

Environment.yaml file components:

1. Parameters:

1.1. WebApplicationName: describes the application name default: **eGovERAWebApp-DEV/PROD**

1.2. BeanstalkEnvironmentName: describes the name of the environment created default: **eGovERABsEnvironment-DEV/PROD**

1.3. InstanceType: describes the type of the instances default: **t4g.large / m5.xlarge**

1.4. Email: specify the email recipient for the Manual Approval sent by the AWS SNS topic default: **staramas@deloitte.gr**

Development.yaml template		
Parameters	**Default**	**Description**
WebApplicationName	eGovERATestApp	The application name
BeanstalkEnvironmentName	eGovERATestBsEnvironment	The name of the environment created
InstanceType	t4g.large	Type of the instances
Email	staramas@deloitte.gr	Email recipient for Manual Approval sent by the AWS SNS topic

5 BENEFITS OF USING INFRASTRUCTURE AS A CODE

Infrastructure as a Code is the process of provisioning and managing cloud resources by writing a template file that is both human readable and machine consumable. This approach standardizes the setup process, reduces chances of incompatibilities and boosts systems' overall performance. In AWS, the built-in choice for Infrastructure as a Code is the AWS Cloud Formation service.

Visibility:

These templates serve as a reference of what resources are on the account, and what their settings are. You do not have to navigate to the AWS console to check the parameters.

Stability:

If you accidentally change the wrong setting or delete the wrong resource in the web console you can break things. Infrastructure as code helps with management of the resources used already or to be added, especially when it is combined with version control.

Scalability:

With infrastructure as code, you can write it once and then reuse it when needed. One template can be used as the basis for multiple services, in multiple regions, making it easier to horizontally scale.

Security:

Infrastructure as a Code gives a unified template for how to deploy architectures. If you create one well secured architecture, you can reuse it multiple times with each deployment having the same security settings.

Transactional:

CloudFormation not only creates resources on your AWS account but also waits for them to stabilize while they start. It verifies that provisioning was successful, and if there is a failure it can roll the infrastructure back to a past known good state.

6 BENEFITS OF USING AWS CLOUD FORMATION

AWS CloudFormation is an AWS service that uses template files to automate the setup of AWS resources.

It can also be described as infrastructure automation or Infrastructure-as-Code (IaC) tool and a cloud automation solution because it can automate the setup and deployment of various Infrastructure-as-a-Service (IaaS) offerings on the AWS CloudFormation supports virtually every service that runs in AWS.

In general, if a service runs on AWS, it is a safe bet that you can use CloudFormation to automate its configuration and deployment.

Deployment speed:

When you create CloudFormation templates to manage how AWS resources are configured and deployed, you can deploy multiple instances of the same resources instantaneously using just one template. This approach leads to much faster deployment than you could achieve if you had to manually set up each deployment by running commands on the CLI (Command Line Interface) or pressing buttons in the AWS console.

Scaling up:

Even if you do not initially expect to deploy multiple instances of the same AWS resources, CloudFormation templates are useful because they ensure that you can scale your environment up quickly when the time comes. By keeping CloudFormation templates on hand you can add more virtual machine instances or storage space.

When demand decreases you can take some of your deployments offline while still retaining the ability to redeploy them quickly using CloudFormation when demand increases.

Service integration:

A single CloudFormation template can manage the deployment of individual services or resources and multiple resources. This management ability means you can use CloudFormation to integrate different AWS cloud services.

Managing multiple services through a single template makes it easy to integrate AWS services as you build out a complete cloud environment.

Consistency:

When you use CloudFormation templates to define and deploy AWS resources, you can apply precisely the same configuration repeatedly. In this way, CloudFormation ensures that your applications and services will be consistent and identical, no matter how many instances you create.

Security:

Along similar lines, although CloudFormation is not a security tool, it can improve the overall security of your AWS environment by reducing the risk of oversights or human errors that could turn into breaches.

Easy updates:

In addition to deploying new resources, you can apply changes to existing resources with CloudFormation templates. This ability simplifies the process of, for example, adding more storage to a fleet of ec2 instances or changing access control rules.

Auditing and change management:

When you use CloudFormation to manage your infrastructure, you can track changes based on which templates you have applied and how they change over time. Change tracking in CloudFormation means that you will be able to determine how your AWS

services and resources have changed over time without looking through logs to reconstruct the timeline of updates.

7 FUTURE WORKS

7.1 EC2 Autoscaling group

The aim of this project is to have an elastic environment that allows us to keep up with future changes and needs. A possible upgrade could be the increase in the number of ec2 instances, which would lead to expand the amount of users who can simultaneously use the application.

Amazon EC2 Auto Scaling is one way to maximize the benefits of the AWS Cloud. The application gain the following benefits:

- Better fault tolerance. Amazon EC2 Auto Scaling can detect when an instance is unhealthy, terminate it, and launch an instance to replace it. You can also configure Amazon EC2 Auto Scaling to use multiple Availability Zones. If one Availability Zone becomes unavailable, Amazon EC2 Auto Scaling can launch instances in another one to compensate.
- Better availability. Amazon EC2 Auto Scaling helps ensure that your application always has the right amount of capacity to handle the current traffic demand.
- Better cost management. Amazon EC2 Auto Scaling can dynamically increase and decrease capacity as needed. Because you pay for the EC2 instances you use, you save money by launching instances when they are needed and terminating them when they aren't.

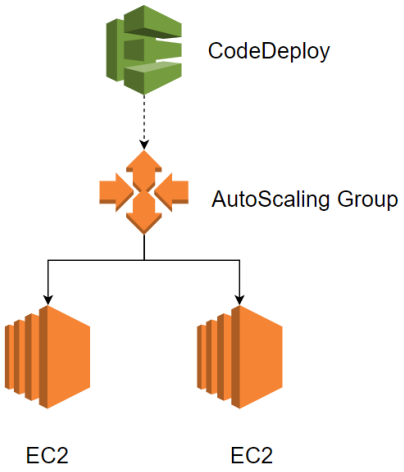

Figure 9 EC2 Auto Scaling Group

7.2 AWS CloudFront

It is also possible to improve the availability of resources by using the Cloud Front services which delivers your content through a worldwide network of data centers called edge locations. When a user requests content that you're serving with CloudFront, the request is routed to the edge location that provides the lowest latency, so that content is delivered with the best possible performance.

CloudFront speeds up the distribution of your content by routing each user request through the AWS backbone network to the edge location that can best serve your content. Typically, this is a CloudFront edge server that provides the fastest delivery to the viewer. Using the AWS network dramatically reduces the number of networks that your users' requests must pass through, which improves performance.

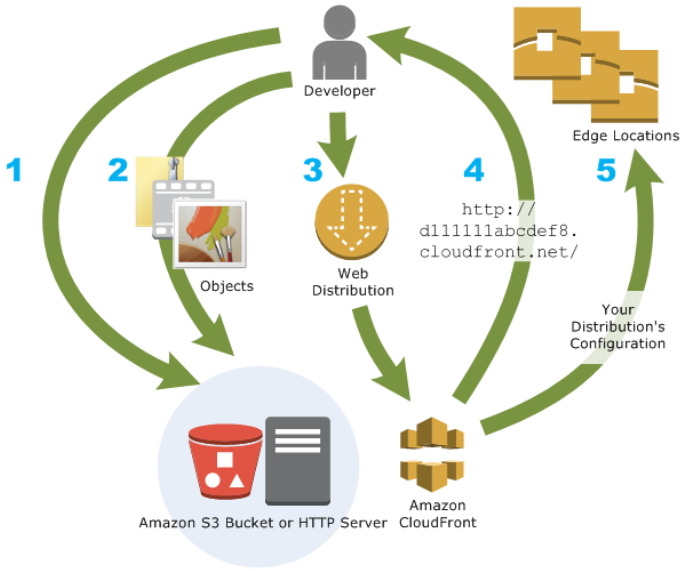

Figure 10 AWS CloudFront

8 POLICIES

8.1 AWS-CodePipeline-Service

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codecommit:CancelUploadArchive",
                "codecommit:GetBranch",
                "codecommit:GetCommit",
                "codecommit:GetUploadArchiveStatus",
                "codecommit:UploadArchive"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetApplicationRevision",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
```

```
54.       {
55.          "Action": [
56.             "iam:PassRole"
57.          ],
58.          "Resource": "*",
59.          "Effect": "Allow"
60.       },
61.       {
62.          "Action": [
63.             "elasticbeanstalk:*",
64.             "ec2:*",
65.             "elasticloadbalancing:*",
66.             "autoscaling:*",
67.             "cloudwatch:*",
68.             "s3:*",
69.             "sns:*",
70.             "cloudformation:*",
71.             "rds:*",
72.             "sqs:*",
73.             "ecs:*"
74.          ],
75.          "Resource": "*",
76.          "Effect": "Allow"
77.       }
78.    ]}
```

8.2 aws-Pipeline-Execution

```
1.  {
2.  "Version": "2012-10-17",
3.     "Statement": [
4.        {
5.           "Action": "codepipeline:StartPipelineExecution",
6.           "Resource": "arn:aws:codepipeline:eu-west-
    1:<ACCOUNT_ID>:codecommit-events-pipeline",
7.           "Effect": "Allow"
8.        }
9.     ]
10. }
```

8.3 IAM-PassRoleToCloudWatch

```
1.  {
2.  "Version": "2012-10-17",
3.  "Statement": [
4.  {
5.  "Effect": "Allow",
6.  "Action": [
7.  "iam:PassRole"
8.  ],
9.  "Resource": "
    arn:aws:iam::<ACCOUNT_ID>:role/AmazonCloudWatchEventRole-
    DEV/PROD"
10. }
11. ]
12. }
```