**Solution for Elasticache cluster configuration**

This repository contains a solution designed to streamline the deployment process of multi region Elasticache cluster and Global Datastore using Python scripting and region-specific CloudFormation templates.

The solution comprises the following components:

**Elasticache cluster & Global Datastore:** Global Datastore in Amazon Elasticache for Redis provides fully managed, fast, reliable, and secure cross-region replication. With this, we can write to the Elasticache for Redis cluster in one region and have the data available to be read from two other cross-region replica clusters, thereby enabling low-latency reads and disaster recovery across regions.

**Python Script**: The deploy.py script is an integral part of the deployment process. It automates various tasks and orchestrates the deployment of resources.

**Region Specific CloudFormation Templates**: The repository includes CloudFormation templates tailored to specific regions. These templates facilitate the provisioning of resources in a targeted manner.

Pre-requisites:

Before deploying the solution, make sure to complete the following pre-requisites:

1. AWS Profile Setup

Ensure that the AWS profile is set up. We can also configure the AWS Access Key (AK) and Secret Key (SK) by running the command "aws configure" in the terminal.

2. CloudFormation Templates

The region-specific CloudFormation templates required for the deployment are uploaded to an Amazon S3 bucket.  
[https://ec-global-ds.s3.amazonaws.com/<region>-template.json](https://ec-global-ds.s3.amazonaws.com/%3cregion%3e-template.json)  
  
These templates are publicly available, allowing for streamlined access during the deployment process. These templates are present in the repository also for reference, in case of any issue in accessing S3.

3. Deploying the Solution

To deploy the solution, follow these steps:

i) Clone the Repository: Begin by cloning this repository to the local environment.

ii) Navigate to the Repository:  
 cd <repository\_folder>

iii) Execute Deployment Script python3 deploy.py

This will initiate the deployment process and utilize the region-specific CloudFormation templates stored in the S3 bucket. The deployment script will automate the necessary tasks to set up resources in the AWS environment.

The Python code incorporates the boto3 interface, which serves to initiate the creation of a stack. This process utilizes variable definitions within the script and pass them to cloud formation templates sourced from an S3 repository. The cloud formation template encompasses a series of steps for resource creation. The resources created are:

* **VPC**: Establishes a Virtual Private Cloud to isolate and manage network resources.
* **Subnets**: Creates subnetworks within the VPC to segment and organize resources.
* **Elasticache Security Group**: Sets up a security group to control inbound and outbound network traffic for ElastiCache.
* **Elasticache Subnetgroup**: Forms a subnet group to determine the placement of ElastiCache nodes.
* **Elasticache Parameter Group**: Configures a parameter group to fine-tune the behaviour of the ElastiCache instance.
* **Elasticache Replication Group**: Initiates the creation of a replication group for ElastiCache to enable redundancy and scaling.
* **Elasticache Global Replication Group**: Sets up a global replication group for ElastiCache to support cross-region data replication.
* **Elasticache cluster and Global data store**

Collectively, these steps within the cloud formation template ensure the systematic creation of essential resources for the Elasticache and Global data store deployment, thereby establishing a well-structured and functional environment.

**Requirement Mapping**

For all new clusters, the following requirements must be met.

1. Staging and Production clusters should be configured with identical settings while providing support for a difference in the number of nodes between environments – Achieved.
2. Clusters should be deployed to all available regions: [us-east-2, eu-west-1, ap-east-1] – Achieved.
3. Encryption must be enabled for all data at rest and in-transit – Achieved.
4. The following resource tags must be included:
   1. Owner [must be one of the following values]
      1. Data Services
      2. Platform API
      3. Infrastructure
      4. Web
   2. Service [arbitrary string]
   3. Name [arbitrary string]
   4. Classification [must be one of the following values]
      1. Internal
      2. Confidential
      3. Public – Achieved.
5. Additional tags can be supplied via code and will be appended to the list of tags required above – Achieved.
6. Clusters should be configured to run in clustered mode – Achieved.
7. “maxmemory-policy” redis parameter should be set to “allKeys-lru” – Achieved.

**Reusability:**

The Python script already includes the necessary environment variables. In the future, we can further enhance automation by making improvements to this script. These enhancements are intended to increase the reusability of the framework and allow for more efficient cluster upgrades and modifications. We can use a property file to store all the environment variables so that any updates in future will only require update in the property file, not the script.

**Potential Benefits:**

AWS CloudFormation offers distinct advantages for users deeply entrenched in AWS services. With native integration into the AWS ecosystem, CloudFormation provides seamless access to new AWS features and straightforward management of AWS-centric environments. Its unified approach to managing AWS resources and application code, declarative syntax, and lack of additional tool installation make it user-friendly, especially for newcomers. Moreover, CloudFormation's alignment with AWS security and compliance tools, direct creation from the AWS Management Console, and prebuilt resource types streamline deployment while adhering to AWS best practices.

**Potential Weakness:**

CloudFormation templates are designed specifically for AWS services. If in the future we want to set up the environment in different cloud platforms, we may need to transition the code to cloud-agnostic tools like Terraform or Ansible.

When dealing with deploying across multiple regions, a single CloudFormation template might not suffice. Region-specific templates could be necessary to manage objects that span different regions. This complexity increases when handling references to objects created on-the-fly across regions. Establishing dynamic VPC peering could be required. Tools such as Terraform and Ansible handle the challenge of managing dynamically created objects and their references more efficiently, making cross-region deployment smoother and more manageable.