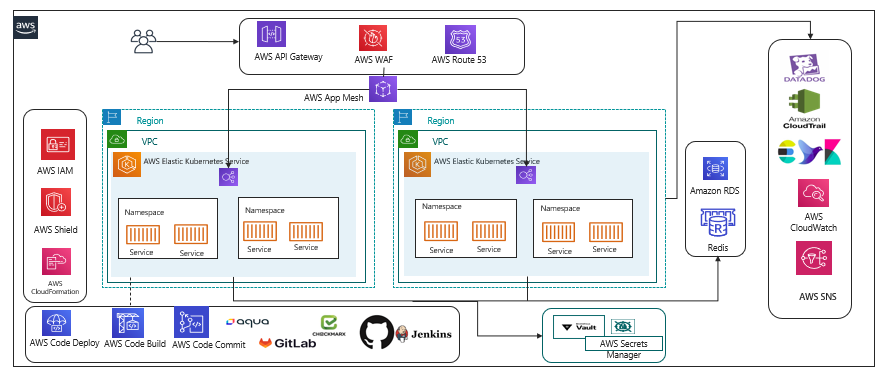
1. **DEVOPS STRATEGY**
2. **DEPLOYMENT ARCHITECTURE FOR EKS**

The proposed deployment architecture is integrated with the Kubernetes platform deployed across the region.  Continuous Integration and Continuous Deployment are configured either with the proposed tools or with the existing tools. The services are isolated and deployed in the specific namespace in the AWS account.

The below diagram and the tables give the details about the tool’s integration.



|  |  |  |  |
| --- | --- | --- | --- |
| **Existing Tools** | **Recommended Tools** | **Description** | **Recommendation** |
| Elastic Kubernetes Service |  | EKS-managed cluster is distributed across the two regions with High Availability  Autoscaling will be configured as part of the replica. |  |
| Elastic Container Registry |  | Base images are created for the containers to be deployed on EKS. We need centralized storage for configuring and uploading the non-prod and prod environments. |  |
| Terraform |  | Terraform for the IaC |  |
| AWS CloudFormation & Stackset Template | AWS CloudFormation StackSet | Templates configured for the provision of the AWS resources across multiple regions with a single cloud formation template. | AWS CloudFormation StackSet helps with a single cloud formation template deployed across the regions |
| Helm |  | Helm a package management tool will be configured for the release process and with the version control enabled.  Helm is used as a release management tool integrated with the Kubernetes |  |
| AWS App Mesh /Istio | AWS App Mesh | App Mesh uses the open-source Envoy proxy, making it compatible with a wide range of AWS partners and open-source tools.  App Mesh removes the need to update application code to change how monitoring data is collected or traffic is routed between services. App Mesh configures each service to export monitoring data and implements consistent communications control logic across your application. | AWS App Mesh configures each service to export monitoring data and implements consistent communications control logic across your application.  This makes it easy to quickly re-route network traffic when there are failures or when code changes need to be deployed. |
|  | Aqua | Aqua will be integrated as part of the platform to validate the prior image being uploaded onto the repository.  Security checks on the containers, images, code quality, and vulnerability checks will be integrated as part of the pipeline. | Aqua |
| Application Monitoring (EFK)  Cloud Watch | Datadog/CloudWatch | This is centralized and configured in the regions for extracting and maintaining the logs.  Additional Disk Volumes will be mounted as part of it so that the logs are always dumped to the volumes and extended without the changes affecting the system. | Datadog /CloudWatch |
| Infra Monitoring  CloudWatch | Datadog | Infra monitoring | Datadog: With the Instances monitoring, Application monitoring, Log Monitoring and all are available in a single dashboard.  It’s also reducing looking at multiple dashboards for monitoring purposes. |
| Datadog metrics monitoring |  | Datadog provides APM tools for monitoring the traces of the applications.  Datadog is tightly integrated with the Kubernetes pod for the monitoring of the internal execution along with the backend connection and the traces of the database execution.  Performance and the time taken for the execution of the queries are being traced and visualized at the centralized dashboard for monitoring. |  |
| Cloud Trail |  | Enables the auditing and monitoring of API calls made with the AWS Accounts. |  |
| HashiCorp Vault Management | AWS Secret Manager | Secret management Integrates the secrets and the cert keys fetched from the Vault and integrated into the services and pods during the deployment. | AWS Secret Manager: AWS Secret Manager is a managed service and pay-as-you-go.  AWS Secret manager allows key rotation which is manual in HashiCorp.  HashiCorp is available as an open source and it does lay for the additional cost incurred in terms of scaling across  Overhead cost for maintaining the HashiCorp Vault. |
| **Security Protections**  AWS Shield  AWS GuardDuty  AWS WAF |  | AWS Shield for protection across the DDos attack.  Threat detection services that continuously monitor our AWS accounts  Web Application Firewall to protect from common exploit threats. |  |
| AWS Elastic Cache for Redis |  | Create cross-Region read replica clusters for ElastiCache for Redis to enable low-latency reads.  Disaster recovery across AWS Regions. |  |
|  | GitLab |  | GitLab has the end-to-end feature integrated right from the code commit, code deployment, code review, branching strategy applied, Review and approval of the code, promotion of the code, Pipeline oriented execution, and Repository container inbuilt to store the images.  This makes it much easier and more integrated and helps with review and approval process as part of the pipeline |
| AWS Managed to build tools like codecommit, codebuild, codepipeline, codedeployment |  |  | AWS managed build tools for the Continuous Integration and Continuous Deployment in case GitLab is not opted for. |

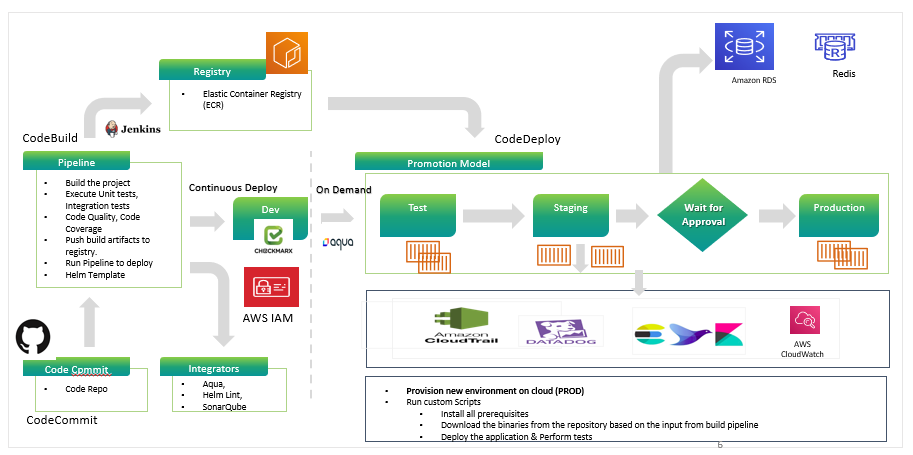
1. **AWS GOVERNANCE**

AWS governance helps to support the organization’s workflow and to apply policies that make the DevOps execution much more streamlined and aligned with the team for faster execution.

The following are the few policies and roles that will be utilized as part of AWS Cloud for the current execution.

|  |  |
| --- | --- |
| **Area** | **Description** |
| Authentication | An authentication mechanism will be enabled for the validation of the user identity validation across the domain users configured. |
| Authorization | Access rights for the individuals are configured for controlling access across the resources /services.  AWS Standard practices will be performed so that the project and the BU across the region are handled only by the respective projects. |
| Secret Management | The isolation of the access will be done through the Identity & Access management. |
| Identity Management and access control | Centrally managing users, security credentials such as access keys, and permissions control which resources users can access. Using IAM  **Users** can access the assigned resources through the permission configured through the policies.  **Groups**:   * A group is configured with common group policies. * Policy applied is immediately traversed across all users in the group. * The groups are customized for each project specific.   **Roles**   * Configured for the complete isolation of the independent projects. * Roles are not configured with any users however they can be intake or consumed by the authorized users. * Different roles will be configured for the project-specific execution.   **Policies:**  Policies are one of the important key factors to maintain the roles and permission across the cloud.  AWS policies in specific the following will be configured for the platform:   * Access Control * Identity * Resource-level permission * Service control policies * Access control List * Session Policies   Privilege Admin access will be configured so that the template layout, formation of the Cloud Formation Template of the services across the platform, validation of the user account and unique Id creation for the project, and settings up the policies applied across the board. |
| Backup Plan | A regular backup process will be configured for the continuous delta data taken regularly for the Disaster Recovery. |
| Tagging | Enables the group of resources identified with common names.  We can have the report generated and the custom commands executed as part of the report generation. |
| Governance | The policy includes defining risks, Alignment of the internal policies, Cloud formation template strategy customized, TLS version updated, capability embedded in all resources, and Adherence to governance requirements. |
| Audit | Enable the independent or internal assessment of the cloud resources that are part of the accounts. |
| Change Management | Approved changes being traversed across the production environment |
| Arch Council | To approve the specific tools for the evaluation and implementation |
| Cost Control | Asset Inventory management to optimize the cost. |

1. **DEVOPS CI/CD PIPELINE**



The following are the steps execution as part of the pipeline the CICD pipeline is configured with all the requirement stages for the build execution:

* CICD Pipeline is configured with multi-stage build process for the pipeline build execution.
* Continuous Integration of the build process starts and triggers the staged build process.
* The triggered build process on compilation is successful proceed with the next gated check with the code quality.
* During the process the image is validated for all the security integration with the scanning using the tool.
* The image of having the vulnerability criteria falls into the criteria like high, medium and low grouping.
* The rules being set with the expectation for the code to pass in and to upload on to the Registry.
* On successful compilation the build is uploaded onto the Container Registry.
* The artifact is then called using the Continuous Integration process for the artifact deployment onto the K8 cluster.

**Release process**

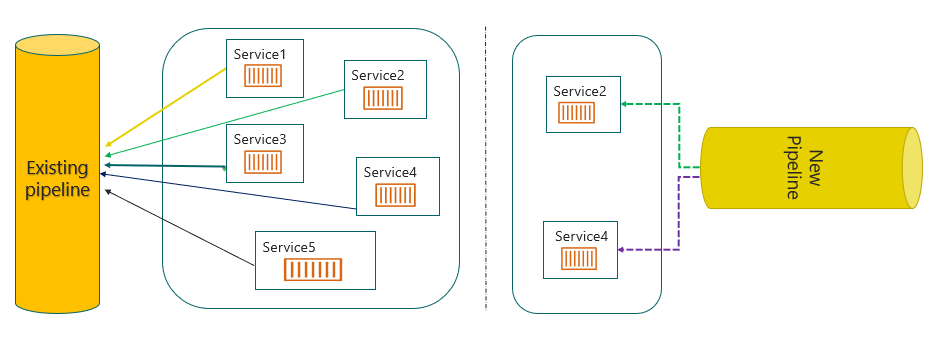
* The releases are done using the templates configured with the chart for the specific repos and the same can be reused across the board. The releases are version controlled and the roll back can be done at any point of time.
* All the artifacts are stored on the Container Repo and tagged with the defined labels.
* The promotion model is being set so that the same artifact can be promoted onto the next environment using the pipeline.
* Non prod environment setup done for doing their testing without disturbing any existing environment.
* Performance test scripts are integrated to run on the code deployed and validated.
* Alerts are being configured at all the levels in case of failure to trigger so that they’re completely monitored with the tools.

**Application and Infrastructure Monitoring**

* The application infrastructure monitoring is being done using the Datadog & CloudWatch integrated as part of the platform.
* The Datadog tool will be tightly integrated & the agent configured for the entire monitoring.
* The agent installed on the monitoring will be retrieving all the data and the logs related to the application and infrastructure.
* The APM (Application Performance Monitoring) will be integrated with the platform for the monitoring and all the services, and the pods will be integrated for the traces capture of the platform logs.

1. **STRANGLER DEVOPS PIPELINE**

Proposed Strangler DevOps pipeline for the existing as well as the new pipeline.



The strangler proposed DevOps pipeline works as follows:

* The monolithic and the new micro services configured will be running in parallel in the pipeline.
* The new micro service will be built independently from the existing Monolithic service.
* The new micro service as well as the existing monolithic service pipeline will have the services running in both the platform.
* The monolithic service that is split and built in new micro service is being tested
* The new micro services developed will be validated from the new cluster.
* The new micro services are validated for the parallel execution, working as expected then the service will be removed from the old pipeline.
* The  cycle continues further for the other microservice split onto the new pipeline.
* The existing pipeline needed changes will be coordinated with the team so that there is no impact.

1. **BRANCHING STRATEGY**

We are proposing the two strategies for the branching to be followed and the most common strategies followed:

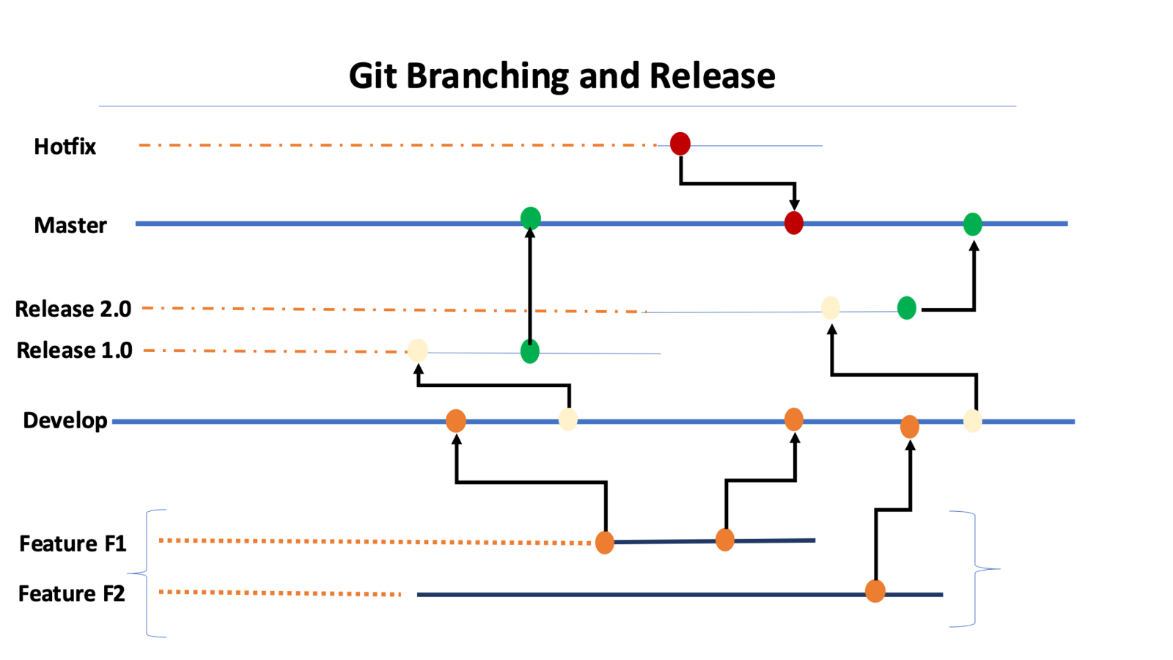
* GitLab Flow
* Trunk-Based Development

**Option 1:**

**GitLab Flow:**

The branching strategy consist of the following branches:

* Main Branches
* Master
* Develop
* Support Branches
* Release Candidate branch
* Hotfix branch



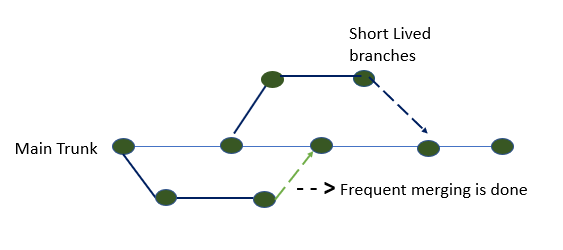
**Proposed Flow:**

* Standard conventions are followed for the branching strategy so that the product is uniquely identified.
* The feature branches are short lived branches which are being developed and merged into the development branches.
* The development branches are then merged on to the supporting branches called as the Release branches.
* Tags are being created from the branches to do the releases.
* The release are then tagged based on the short outcomes of the releases.
* Finally approved changes are being merged back onto the Master branch and the deployment happens from this branch.
* The master branch is then again branched out for the new changes in development/feature to worked upon by developers.

The hotfix branches are created in case of any critical issues to be fixed on the production release being done from the master branch

**Option 2:**

**Trunk Based Development**



Recently the changes are with the strategy aligned to get the development with the short-lived branches so that we have faster releases. Trunk based development is a version control management practice where developers check-in their code to trunk or master branch. The developers finish the new work on the feature, and they merge the changes on to the new code trunk. There are challenges seen during this time with the conflict arising, Conflicts are increasingly complex as development teams grow and the code base scales. The trunk-based development model reduces these conflicts.

**Advantages**

* It allows continuous code integrati–n - In the trunk-based development model, there is a repository with a steady stream of commits flowing into the master branch.
* It ensures continuous code revi–w - Small commits of trunk-based development make code review a more efficient process. With small branches, developers can quickly see and review small changes.
* It enables consecutive production code releas–s - Developers should make frequent, daily merges to the master branch.
* It’s ready to deploy to production at any time.
* This gives the team agility to frequently deploy to production. It also ensures that the further goals of production releases to be made much more frequent than the current process.
* Quicker to deploy and resolve the challenges rather than waiting for the long cycle completion.
* More collaboration during the execution and work as a one team with fully owned responsibilities.

**Recommendati–n - Option 2** is recommended since most of the organizations are aligned with this strategy process due to frequent merges to the master branch.

This can be reconsidered during the discovery process based on further engagement.

1. **RELEASE MANAGEMENT**

**Release Process**

* The release process will be created with the managers & cab approval obtained prior the releases.
* Release processes are defined with the CI/CD Integrated
* Deployment will be done through the build tool.
* Frequent branches and the tagging process will be created for the deployment.
* Release will be tagged and stored as a Tag in the Repo.
* Hotfix branches will be created during the process of any critical fixes that need to be part of the deployment.
* Communicate to the Stake holders on the release complete
* The Helm chart will be used for the version control of the Releases.
* The Release will be deployed using the pipeline with the specific branching Tag created for the deployment.
* The Releases will be version controlled and also deployed from the specific pipeline.
* The promotion of artifacts will be done using pipeline.

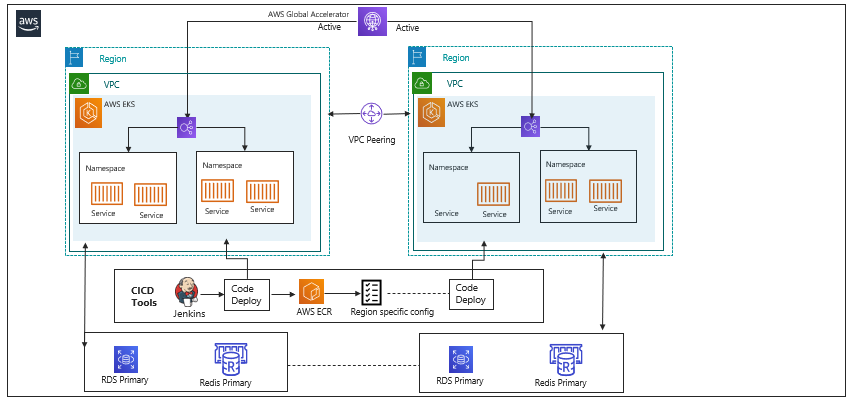
**DevOps Cloud Security for Application**

* Security Virtual Private cloud will be configured.
* Security Rules for the private and public cloud configured.
* Access Identity and Management (IAM) Access Analyzer
* Key Management Service Integrated for encryption.
* S3 configuration and monitoring
* Security assessments on the firewall rules & configuration
* Access to the corresponding repos

1. **DISASTER RECOVERY**

**Please refer to the section** [**Deployment Architecture for EKS**](bookmark://_Deployment_Architecture_for)  for the deployment.

Disaster Recovery diagram projected here is for two regions with active-active configuration.



The above diagram gives the solution running across the regions with the active mode of configuration done.  The proposed model will ensure that all the required services are up and running.

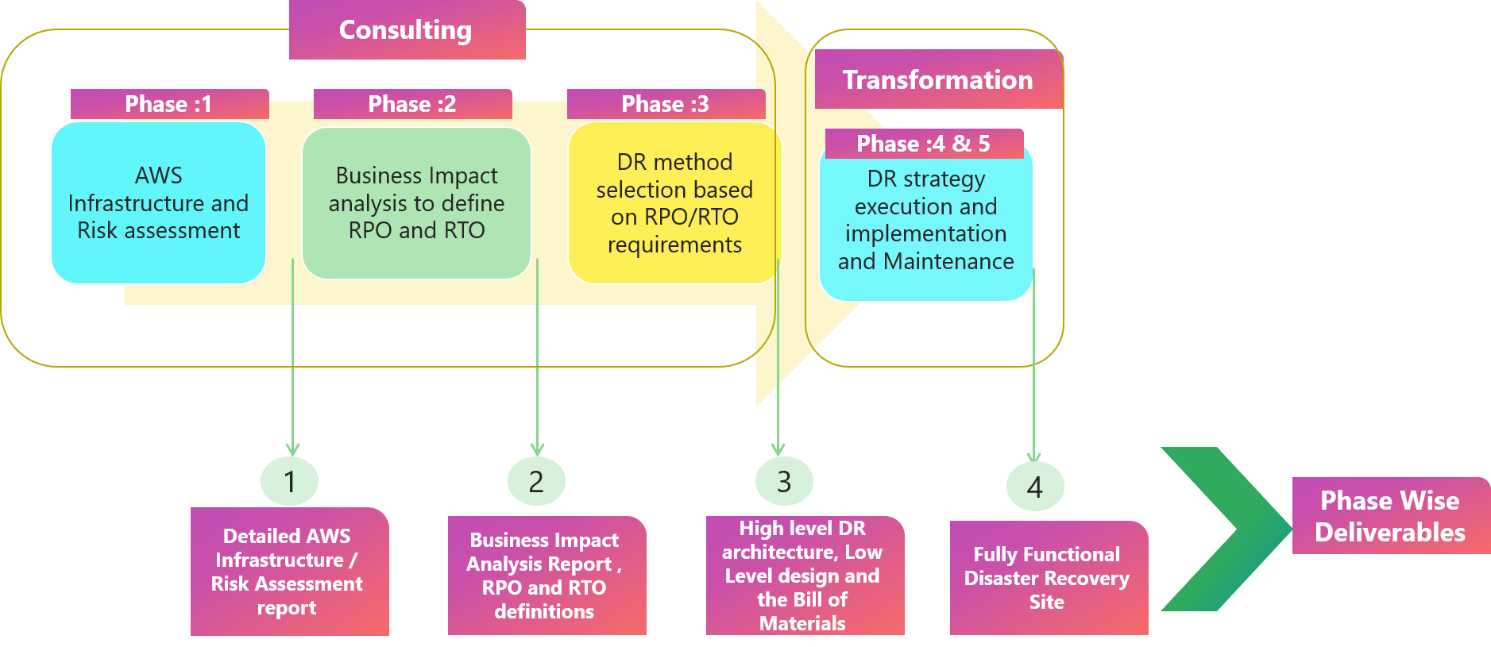
The backup and the pilot mode of execution will be integrated and executed during the process of execution.

The following are the flow will be enabled for the execution:

* Active- Active mode enabled for both the regions so that the recovery will be available always on with minimal or zero downtime.
* Real time execution is made available.
* Reduce the recovery time to the nearest zero down time.
* Backup frequency needs to be more due to the dependency on the recovery of current execution.
* The estimated backup will be depending on the RTO/RPO

The only disadvantage is that the cost of the AWS services is too high.

1. **PROPOSED APPROACH FOR DISASTER RECOVERY**



In this architecture, we will build the realize and the dependencies applications in AWS:

* AWS will be connected through AWS Direct Connect/VPN for secure access.
* Application data will be cloned onto the target regions and continuous replication will be enabled.
* In this scenario, there are two parallel environments running for the platform.
* End user traffic will be routed using AWS managed services.
* Further, the responses sent from both the regions would be logged for analysis and validation.
* The platform will be tested in the AWS environment and will be cutover during a change window as per the scheduled.
* The DR process is being configured with the active – active mode where the clusters are being configured with the High availability.

**Continuous Integration and Continuous Deployment**

* Configuration code is configured in muti-region.
* Deployment of the infrastructure changes and configuration/code changes to be deployed at the same time with Primary Region.
* Analysis on the definition of RTO & RPO.
* Implement the DR method based on the previous results obtained.
* Code stored in the repo will be trigged in the pipeline for the active application deployment .

**Node Cluster Backup**

* EKS cluster nodes are configured in the multi-region for the DR process.
* The node cluster will be configured in two regions for the distribution.
* Distributed pods deployment in both regions across the cluster
* Actively monitor the active regions that are up and running.
* A template will be created for the deployment execution during the DR.

**Databases & Datastores**

* RDS distribution across regions will be configured during the initial setup.
* Database backups are configured regularly based on the discussion.
* The backup will be created using AWS-managed resources.
* DevOps will facilitate a Backup process for the regions.

**Other Options**

**Warm standby setup**

The warm standby approach is a backup strategy that involves maintaining a secondary system that is synchronized with the primary system, ready to take over in case of failure.

* The main active region, which handles all the live traffic and user requests.
* Standby region is kept in sync with the active region. can be achieved by replicating the data.
* The data from the active region is regularly replicated in the Standby region.
* Monitoring and health checks for both the active and standby region.
* In case of a failure in the active region, the failover process is initiated.
* Once the active region is back online, it may require recovery and rebuilding to restore it to its original state.

**Active - Passive setup**

where the passive will be available in the warm setup so that we can switch over the Egress during the fail-over.

* Here only one region will be active at any point in time.
* The process remains the same as that of the above with minimal changes.
* There is a minimal change made for the execution to make it Live for DR.
* Cutover will be performed with the analysis of the region’s failover.

1. **TECHNOLOGY STACK – INDICATIVE**

|  |  |
| --- | --- |
| **Feature** | **Technology** |
| Programming language and Framework | Typescript, JavaScript, React, Java 11 |
| Cloud | AWS, CloudFront, EKS, ECS, Redis, IAM, RDS, MYSQL |
| Authentication | Existing User Authentication module, HashiCorp Vault |
| Networking | AWS Global Accelerator (for Disaster Recovery), AWS WAF |
| API Gateway | AWS API Gateway |
| Microservices​ | NestJS for I/O bound functions and Java Springboot for CPU bound operations  Docker, Kubernetes |
| Database/Storage​ | MySQL, MongoDB, Elasticsearch |
| In memory Cache | Redis, Memcache, ElastiCache |
| CI/CD pipeline | Jenkins/GitHub/GitLab/AWS CodeCommit Pipeline, Docker, EKS, ECS, Kibana, EFK, Datadog, Prometheus, Terraform |
| QA Tools | Rest Assured (API Automation), Test Execution: SauceLab (Only for Mobile), |
| Performance Testing | JMeter |
| Web Server | NGinX |
| Messaging | RabbitMQ, SQS |
| Feature Toggle | Six Pack |