***The goal of this assignment is to fully investigate and implement an enterprise style CI/CD deployment based on the microservice architectural style, which contains at least two services and a data layer.***

* Implementation
  + Architecture
    - Microservice Style
      * DEF, CITATION
      * DESIGN

Diagram

Description automatically generated

Architecture :

* Application divided into 4 microservices,
  + Data Query Service (in charge of querying, filtering records)
  + Info Service (evaluates category and other info) (can be a serverless application)
  + Recoding Service (accepts recording and persists data)
  + UI Service (provides UI and accept customer reading, displaying category and historical reading)
* Division is based on separation of concern, independent scalability, avoid single point of failure.
  + Data query will not cause application to slowdown recording as recording continuous on separate channel.
  + Independent recoding provides options to sell this product as to accept BP reading directly from devices.
* Communicates using REST independent channel.
  + Promotes loose coupling, independent scaling.
  + Recoding service can accept from message queue, promoting de-coupling
    - Recording service can now accept large scale data from source live devices.
* Development
  + Applications developed in NodeJS for both UI and Backend
    - Reduced Technology Stack
    - NodeJS provide Ractive Approach which is designed for better handling large volume calls with less constraint on thread and latency
  + Testing tool use Cypress for Module, UI, Integration testing and End to End
  + Distributed as Docker Image, and deployed on Kubernates cluster
    - Scale as required
    - Exposes application and infrastructure in cloud (like ports and credentials)
* Packaging
  + Delivered as docker image for unit and integration testing
  + Packaged as Helm Chart for solution delivery
    - Each microservice delivered as Helm chart (contains kubernets deployment file and docker image reference)
  + Helm chart is published in repository.
* DevOps Prospective
  + Improves testing, each Microservices tested independently.
  + Separate Deployment, upgrade, own life cycle
  + Separate story line for all microservices
    - Able to focus and fix the issues independently.
    - Ability to test independently
  + Achieve CI/CD for each microservices and deploy as Solution.

## Release Manager Plan

* Build Integrate and Deployment pipeline
  + Build Pipeline
    - Build pipeline builds docker image and packages Helm chart
    - Performs unit , integration testing
      * Runs docker Image.
      * Runs Cypress integration testing.
    - Push the docker image and helm chart to Container Repo
  + Integration Testing Pipeline
    - Helm Chart to deploy on Dev after build
    - Run end to end testing
    - Promote Product for QA Testing
      * Promotion will mean deployment on QA Cluster
      * DEV and QA should have version parity.
      * QA cycle should be in same sprint.
        + Closely couples QA and Dev
    - Promotes Products for Release
      * Tag the image as Release candidate.
    - Run the Release pipeline on Green (passive)
      * Update the version do basic verification test.
    - Divert traffic to Green
    - Run release pipeline on Blue
      * Run verification Test
    - Divert traffic back to Blue
  + Pipelines

|  |  |  |
| --- | --- | --- |
| Pipeline | Purpose | Env |
| Build Release Recording service | Docker, Helm, UnitTest | DEV |
| Build Release Query service | Docker, Helm, UnitTest | DEV |
| Build Release UI service | Docker, Helm, UnitTest | DEV |
| E2E Test for QA | Cypress Test E2E | QA |
| Run BLUE Deploy soln. | DEPLOY on Blue | PROD |
| Run GREEN Deploy soln. | DEPLOY on Green | PROD |
| MongoDB on DEV, QA | Deploy on Dev and QA | DEV |
| MongoDB on BLUE | Deploy on Blue | PROD |
| MongoDB on GREEN | Deploy on Green | GREEN |

* Performance
* Ease of configuration / installation
* Cost
* Licensing
* Monitoring and logging
* Scaling (in and out)
* Rollback plan
* Backup and restore strategy
* Security
* Support
* Vulnerability checks on images

JUSTIFICATION

* Pipeline for each Microservice and Solution
  + Each microservice can be delivered separately, patched upgraded separately
  + Deployment will be done using packaged Helm Chart
    - Solution packages all microservices Helm Chart using dependency
  + Each pipeline has defined purpose and not shared, reduce and admin entry during deployment
  + Release pipeline is different from Dev Deployment
    - Release pipelined restricted by strong governance
    - Release is created just before deployment
    - Each release is just 3 staged
      * Check the Artifact to be deployed
      * Trigger Deployment (manual)
      * Run Basic E2E Test (automatic)
      * If failed – Call Rollback, ignore if successful
* Advantages
  + Segregated pipelined, reduces SRE deployment error due to any entry error
  + Follows deployment pipeline encouraging CI/CD till Deployment
  + Follows Blue/Green deployment providing the zero downtime during deployment
* All vulnerability scanning is done as part of Build itself
  + Example Docker <https://docs.docker.com/engine/scan/>
  + <https://docs.docker.com/scout/>

- job: TrivyScanContainerImage

displayName: Scan container image by Trivy

steps:

- task: Docker@2

displayName: 'Build an image'

inputs:

command: build

repository: $(image\_name)

tags: $(image\_tag)

dockerFile: '\*\*/Dockerfile'

- script: |

mkdir report

trivy image -s HIGH,CRITICAL $(image\_name):$(image\_tag) | tee ./report/trivy-image-scan-report.txt

displayName: "Image scan by Trivy"

continueOnError: true

Security , secrets

* All security for POD and DATABASE connection is in CRDs which will be deployed buy cluster admin
* Each env cluster will have their CRD.
* These CDRs will be read during deployment time and injected to the Container exposed as Env Variables
* Data in CER is encrypted

Scaling In and Out

* Each Microservice will be scaled in and out based on load
* Triggering factor will be CPU
* Kubernetes orchestrates the scaling of application IN-OUT

Monitoring & Logging

* Log everything on Console
* Support Dynamic Logging using <https://github.com/yannvr/Winston-dynamic-loglevel>
* Use Azure DataDog
  + <https://docs.datadoghq.com/integrations/azure_container_service/>
    - *Azure Kubernetes Service allows you to quickly deploy a production-ready Kubernetes cluster.*
    - *Use the Datadog Azure integration to collect metrics from Azure Kubernetes Service.*

High Availability

* Each service maintains at-least 2 containers , Kubernetes manages the routing and maintaining of SLA
* Blue/Green Deployment guarantees zero downtime, and reduction of error post deployment.
* <https://stackoverflow.com/questions/58739513/google-kubernetes-engine-how-to-define-one-ingress-for-multiple-namespaces>

Define NGNIX with 2 replica, Ngnix will read config from ConfigMap., where proxy-pass will be defined as

location /segmentation {

proxy\_pass http://myservice.mynamespace.svc.cluster.local:80;

}