Second Task

This has full description of components of framework, technologies, CI/CD setup and Flowchart

**Introduction to Continuous Testing**

Continuous Testing’s primary goal is assessing business risk coverage.Continuous Testing provides instant insight on whether a release candidate is too risky to proceed through the delivery pipeline. Continuous Testing expects testing to be embedded within the development process, not tacked on at the end. Continuous Testing’s tests must be broad enough to detect when an application change inadvertently impacts functionality that users have come to rely on. Continuous Testing involves continuously reviewing and optimizing the test suite to eliminate redundancy and maximize business risk coverage.Integrate functional testing into CI/CD so it’s a seamless part of the delivery pipeline.

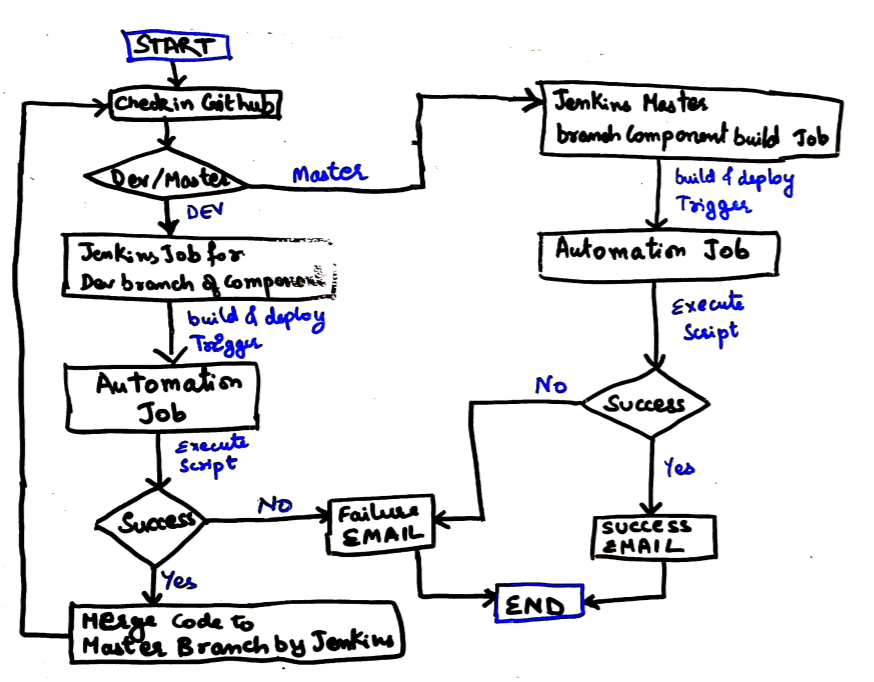
**Architecture and technologies used**

* Frontend - static website using AngularJS framework. Stored in GitHub repository together with API. Has some set of Unit Tests and UI tests.
* API - ASP.NET Web API application with Application API endpoints. Stored in GitHub repository together with Frontend. With some set of Unit Tests and API tests.
* Push - ASP.NET Core microservice used like a WebSocket server for keeping connections with clients and forwarding push messages to them. Stored in separate GitHub repository with some set of Unit Tests.
* Printing - WCF service used for generating pdf files for printing. Stored in separate GitHub repository with some set of Unit Tests.
* Shared Code - Class Library project for keeping shared business logic code. It is referenced to API using Nuget packages. Stored in separate GitHub repository with some set of Unit Tests. With every commit it is automatically compiled and deployed to Nuget server with next version.

**Continuous Integration and Testing**

We can implement this by Jenkins Setup and trigger Automation framework with Jenkins Job post components build and deploy and based on the result of automation trigger merge of the branches.

**FlowChart**

****

**Prequisites:**

1. *AWS account* needs to be created with access to “On Demand instance creation”
2. *Create static IP’s* for all the instances in AWS ,this will be used AWS cli code
   1. Instance 1 : Frontend and API
   2. Instance 2 : Push
   3. Instance 3 : Printing service and master redis instance
   4. Instance 4 : Redis instance as slave, to handle failure cases
   5. Note: from the given architecture design it seems like the shared code will be hosted and taken care automatically by NuGet(i am not aware of NuGet, so i am assuming that this will be already taken care)
3. Setup jenkins with a static url and ip
4. AWS cli code for *build and Deployment of each component* of the the whole architecture needs to be written as separate job for *Dev and for Master* branch in Jenkins using the static IP’s of AWS..
   1. **Make sure to configure execute the build with Unit tests**
5. Configure a Job in jenkins to build and execute Automation framework with parameters to be passed:
   1. Which environment to execute on
   2. What scripts to execute
   3. Any other parameters as needed by the automation framework.

**Setting up CI/CD**

1. Configure webhooks in Github for the repository which should trigger the CI/CD pipeline, If you want for all repositories then set this for all
   1. Go to your project repository in GitHub.
   2. Go to "settings" in the right corner.
   3. Click on "webhooks."
   4. Click "Add webhooks."
   5. Write the Payload URL as <”url of your jenkins webhook”>
      1. **https://USERNAME:PASSWORD@jenkinsurl/github-webhook/**
   6. Give a secret and then select the radio button of “Just the push event”
2. Configure corresponding webhook in Jenkins
   1. Under ‘Manage Jenkins’ -> ‘Manage Plugins’, select and install both Github and Git plugins. Restart to finish the installation
   2. Go to Manage Jenkins -> Configure System
   3. choose ‘Manually manage hook URLs” under the ‘Github Web Hook’ section.
   4. In the job add the https:// url for your repository in the ‘GitHub project’ textfield under the general settings.
   5. enable Git under ‘Source Code Management’. Use the SSH style syntax for the URL repository: git@github.com:user/repo.git (this is required as it’s a private repo), and specify a branch if as dev/ master as per the job.
   6. Under ‘Build Triggers’, tick ‘Build when a change is pushed to Github’.
   7. ‘Security Realm’ section of ‘Manage Jenkins’ -> ‘Configure System’. Depending on your setup, these steps could differ, but in essence you need to create a new user for Github
   8. Click on "Save."
3. Create a Job to trigger merge of the dev branch of the passed component with the master branch
4. For each job of deployment dev and master branch configure in Build section “Trigger/calls builds on other Project” and configure the “automation framework” job details and pass parameter as follows:
   1. pass the environment details
   2. the script to be executed, for dev just the component test script and for Master jobs pass all test scripts
   3. any other details as needed by the automation job as parameters
5. In the Post build section of deployment jobs of dev add a section to wait for automation job result such that if
   1. Automation job is successful then trigger the Merge Job and send a mail with the status
   2. Automation job is unsuccessful then trigger a mail with automation status
6. In Post build section of master branch deployment job setup to trigger email with the status of Automation

**NOTE: This setup will ensure that whenever a code is checked in in Github repository for the specified branch of the hook the code will be deployed and corresponding automation script will be executed**

**CI/CD Pipeline**

We can also configure a pipeline which triggers to setup all the components one after another and trigger the automation framework in the end to execute all the test scripts. For this the only change would be :

1. go to the Jenkins and create a pipeline. we have to provide the order of deployment jobs as below:
   1. MSSQL
   2. Redis setup
   3. Shared code
   4. Printing
   5. Push
   6. API and frontend
   7. Automation Framework.
2. In the pipeline, select "GitHub hook trigger for GITScm polling." for the repository and the branch which should trigger the whole pipeline.

**Automation framework**

The framework that needs to be setup for the testing this architecture has to have multiple components:

1. Front End AngularJS Testing : Protractor with Cucumber:
   1. It is an end-to-end testing framework built specifically for AngularJS. It allows you to create tests that interact with a browser like a real user would. One of the greatest features of Protractor is its ability to “be smart” about waiting for a page to load, limiting the amount of waits and sleeps you use in your suite. Protractor supports Jasmine. Jasmine allows you to write your specs based on the behavior of the application. This is great for unit tests.
   2. Cucumber is another BDD framework that focuses more on features or stories. It mimics the format of user stories and utilizes Gherkin. Cucumber provides your team with living documentation, built right into your tests, so it is a great option for incorporating with your Protractor tests. It also allows you to better organize suites of tests together with tags and hooks.
2. Web API: RestSharp
   1. It is one of the several ways to create a web service or web request in .NET. It comes in particularly handy for Windows phone applications, where REST or SOAP are often used to communicate with external data. It supports both synchronous and asynchronous requests, making it a perfect fit for Windows applications
3. XUnit for socket testing or SocketTest tool
   1. We can achieve socket testing using Xunit and creating a Socket proxy/interface, mock, Socket concrete class and Socket wrapper
   2. The tool Socket Test is a mocking tool used to test the integration points where sockets are present
4. Watin or Separate wrapper to trigger printing services: SoapUI/SOA Cleaner/ WCFStorm
   1. We will have to write our own Soap serializer or deserializer using Watin framework and implement a lot more classes to handle the Soap requests
   2. Another option is to use SoapUI/ SOA cleaner/ WCFStorm to test out the printing Service implemented in WCF
5. MSSQL:
   1. We can write a DBUtil with basic System.Data.SqlClient
   2. Using this we can establish connection with MSSQL and then write our custom queries and get responses to be used during testing

Unit Tests by developers NUnit/ MSTest :

Basic unit tests need to be covered by developers for each module. Such that it has at least 85% coverage. This is Developers responsibility