**Level 4**

**BCS DevOps Engineer**

**Assessment Method 1**

**Project and Practical Assessment**

**Work based Project Signoff**

**Version 1**

**August 2021**

## Overview

This mapping document is to help facilitate the timely sign off for the intended Work based project for Assessment Method 1.

The apprentice should complete the following project mapping to clearly explain how the proposed work based project will meet all the KSB’s for Assessment Method 1.

The EPAO will review the mapping document and sign it off. In the case that more detail is needed the EPAO will provide feedback to request further information.

|  |
| --- |
| **Information outlineTop tip:**   * Please take the time to fully read and understand the assessment plan for this Assessment Method. |

We strongly recommend that the mapping document is completed and used. Failure to do so will likely cause delays in the EPAO being able to sign off the proposed work-based project.

## Assessment method 1: Project and Practical Assessment

## Apprentice Details

|  |  |
| --- | --- |
| **Name** | Arbnor Aljilji |
| **ULN** | 4415055256 |
| **Training Provider** | Makers |
| **Employer** | T. Rowe Price |

**Project Brief**

|  |
| --- |
| Symphony Simplifier is a front-end application that will not only enhance the communication between teams but backlog the communication for auditing purposes. It is a way of simplifying the communication between teams as messages are not to be seen via email or being missed on Symphony. As the messages are to be logged for auditing reasons, the client would want to keep a copy of the messages sent out as evidence for their monthly status report, and quarterly review.  I plan to set up meetings with the client over Zoom, and in-person to ask questions regarding their needs for the project and come up with an Architectural Design Record to which help me write understandable User Stories. These User Stories will help me design the MVP of the app within the time set. The source code of the app will sit in Unity. Unity is an inhouse platform that automates all stages of the SDLC and will deploy immutable infrastructure to AWS using Unity Deploy. Within Unity Deploy, I can create my own resource, and deploy/destroy new versions of my release. Within the app, I will be using Prometheus and Grafana to monitor the app to ensure it is providing the service required. To ensure there is minimal vulnerability in the app, I will only install libraries/modules using artifactory which stores our internal checked dependencies. SonarQube will scan the development for any code smells and will report to TEA which is an inhouse app sitting under Unity, that collects test coverage. To understand the attacks that could occur to the app, I will highlight prior to the design of the app what security threats could exist and how to tackle them using our inhouse apps that already provide security vulnerability. Drawing a diagram of what likelihood v impact will highlight potential risks. To employ a systematic approach to the problems, I have resource such as Splunk and Grafana to monitor what the current issue is before trying to debug. TDD is the principle I will try and follow throughout the app, however, in some cases I have worked within teams that code before test, and then refactor. Gitlab-ci-.yml would do majority of the release automation and orchestration for CI/CD. As this app will be a front-facing app, I will be deciding between using Angular or React for fronend, and NodeJS + VueJS will be the framework for the application in the backend. Due to Terraform being wrapped around Unity, there will be certain limitation to what I can do as it’s already existing. However, I will still create my own Topology and Strategy which uses Terraform to create the resource, and to be able to deploy/destroy at my command. |

**Write a short and clear explanation in the Project Mapping column how each KSB below will be met through the proposed project. The EPAO will then review.**

# Project Checklist Table

This is to cross check the project work completed by the apprentice meets the KSBs required by this assessment method.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Quality**   * Writes code, both general purpose and infrastructure-as-code (including cloud infrastructure) that is correctly versioned and easy to merge, while adhering to the principles of distributed Source Control. * Demonstrates an iterative approach to evolving code consistent with cloud security best practice, evidenced by a lack of vulnerabilities and that all dependent components are present at run time. * Writes code around unit tests, including the appropriate use of test doubles and mocking strategies. * Explains troubleshooting methods used to identify and resolve issues and gives an example of identifying and remediating an issue that compromised code quality. | | | |
|  | | **Project Mapping** | **EPAO Feedback** |
| K2 The principles of distributed Source Control, including how to exploit the features of the tool, such as branching. |  | Using Unity to initialize the project creation, which hosts the code in gitlab. Working on different branches for different features. Source control gives traceability, history, easier to maintain code |  |
| K5 A range of modern security tools and techniques – e.g. threat modelling, vulnerability scanning and dependency checking, with a general awareness of penetration testing - in order to deal with threats and attack vectors within code and across the cyber domain. |  | Using artifactory to pull libraries from inhouse and not from google, using Renovate to scan dependencies, using TEA inhouse to scan code for code smells, setting meetings with Cyber security team to speak about the application as it’s front end app |  |
| K7 General purpose programming and infrastructure-as-code |  | Coding in JavaScript, Terraform will be wrapped around Unity so won’t be needing to do infra-as-code |  |
| K14 Test Driven Development and the Test Pyramid. How the practice is underpinned by unit testing, the importance of automation, appropriate use of test doubles and mocking strategies, reducing a reliance on end-to-end testing. |  | Automate my unit tests in the gitlab ci/cd pipeline, using mocks to do api testing, understand the difference in the testing pyramid (unit, integration, system, and acceptance) |  |
| S9 Application of a range of cloud security tools and techniques - e.g. threat modelling, vulnerability scanning, dependency checking, reducing attack surface area - incorporating these tools and techniques into the automated pipeline wherever possible. |  | Using artifactory to download libraries rather than directly from the internet, scanning dependencies within my app, ensuring code coverage |  |
| S11 Employ a systematic approach to solving problems, using logic and hypotheses / experimentation to identify the source of issues. |  | Looking at logs within Splunk, aws logs, or the application error to understand the problem occurring. |  |
| S14 Write tests and follow Test Driven Development discipline in various different contexts. |  | Write Unit tests before writing code. Red, Green, Red method when appropriate |  |
| S17 Code in a general purpose programming language. |  | JavaScript/NodeJS |  |
| S18 Specify cloud infrastructure in an infrastructure-as-code domain-specific language. |  | Terraform (wrapped within Unity) |  |
| S20: Writing code in such a way that makes merging easier and facilitates branching by abstraction - i.e. feature toggling. |  | Pushing small code changes and annotating with the write merge description for easing understanding, feature toggling to modify behaviour without needing to change code |  |
| S22 Incremental refactoring by applying small behaviour-preserving code changes to evolve the architecture. |  | Pushing small code changes that will help me evolve the architecture and refactor, making it easier for other devs to read it |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Meeting User Needs**   * Writes user stories that are understandable to a wide range of stakeholders, stand up to scrutiny and lend themselves to a solution based on common architectural patterns - i.e. reducing the number of moving/redundant parts; passes all acceptance tests. * The piece of code meets the ‘must have’ identified functional/non-functional user needs encapsulated in the acceptance criteria for the task. * Creates a quality product in terms of Mean Time To Recovery (MTTR) - i.e. reduced time to fix bugs. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K4 The business value of DevOps in terms of Time, Cost, Quality, with an emphasis on building in internal Quality throughout the lifetime of the product. |  | Monitoring the application to ensure minimal downtime, speaking with stakeholders regarding the app, bringing Devs together with IT Ops |  |
| K10 How the user experience sits at the heart of modern development practices in terms of strategies to understand diverse user needs, accessibility and how to drive adoption. |  | Having regular meetings with the client when necessary |  |
| K21 Architecture principles, common patterns and common strategies for translating user needs into both cloud infrastructure and application code. |  | Discussion with the client to understand their application need to be able to understand what resources I would need for cloud infra, and my app framework |  |
| S3 Translate user needs into deliverable tasks, writing clear, concise and unambiguous user stories that the whole team can understand. |  | Using Jira to track my user stories to create my MVP and build further upon the client requests |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **The CI-CD Pipeline**   * Builds a fully functioning, automated CI-CD pipeline with all tests passing. * Evidences a code commit progressing seamlessly from a build artefact to the end user. * Explains the pipeline capability, including the benefits of frequent merging of code, in terms of Continuous Integration/Delivery/Deployment. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K1 Continuous Integration - the benefits of frequent merging of code, the creation of build artefacts and ensuring all tests pass, with automation throughout - including common tooling. |  | Using gitlab-ci.yml to create the stages of CI/CD and automating my tests, looking back at (traceability) from previous versions |  |
| K15 The principles and application of Continuous Integration, Continuous Delivery and Continuous Deployment, including the differences between them. |  | Using gitlab for CI, Unity for CD |  |
| S15 Release automation and orchestration as part of a Continuous Integration workflow and Continuous Delivery pipeline, automating the delivery of code from source control to the end users. |  | Releases into production requires approval before hand, however, automating from CI/CD until releases happen within Unity |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Refreshing and Patching**   * Deploys immutable infrastructure that enables the regular recycling of servers and refreshing of associated software based on manual processes. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K8 Immutable infrastructure and how it enables continuous refreshing of software, namely the updating of the operating system, container and security patching. |  | Destroying the current deployment of the app and redeploying a new version whenever a new prod change is made |  |
| S5 Deploy immutable infrastructure. |  | Infra cannot be changed in prod without need release approval, code changes require approval from other colleagues, Deployment in AWS cannot change without destroying and redeploying the cluster – Using Unity |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Operability**   * Installs and manages monitoring and alerting tools that provide coverage of the infrastructure and applications, including RAM and CPU utilisation, application error rates and availability (health check). * Configures appropriate alerting thresholds and visualisations. Interprets these in terms of failure scenarios and remedial/follow up actions taken to deliver continuous improvement. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K11 Monitoring and alerting technologies and an awareness of the insights that can be derived from the infrastructure and applications - collecting logs and metrics, configuring alerting thresholds, firing alerts and visualising data. |  | Setting metrics using Grafana and Prometheus for observability, using splunk for collecting logs |  |
| S6 Install, manage and troubleshoot monitoring tools. |  | Using renovate to monitor dependencies vulnerability and create MRs, using Grafana and Prometheus for observability |  |
| S19 Interpret logs and metrics data within the appropriate context to identify issues and make informed decisions. |  | Using Splunk for logging, such as when Vault does a read access to the / key, understanding the error within the IDE itself, reading AWS logs when the app is live in prod (if error causes) |  |
| B3 Displays a commitment to the mantra 'You build it, you run it', taking ownership of deployed code and being accountable for its continual improvement, learning from experience and taking collective responsibility when things fail. |  | Using udemy, acloudguru and other resources to keep learning, I’ll be taking ownership of the application, setting up the meetings myself with client and meetings with other colleagues for assistance, monitoring the application myself once it’s in prod |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Persistence**   * Employs and operates an appropriate data persistence technology, such as database, configuration/infrastructure state management to meet non-functional and functional needs. * Explains troubleshooting steps taken to locate issues across the end-to-end service. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K12 The persistence/data layer, including which database/storage technologies are appropriate to each platform type and application when considering non-functional and functional needs; e.g. monolith, microservice, read heavy, write heavy, recovery plans. |  | * Microservice architecture, * Primary database model will be DBMS * Using Postgresql object-relational database * Multi-AZ deployment (includes automatic failover) |  |
| S7 Navigate and troubleshoot stateful distributed systems, in order to locate issues across the end-to-end service. |  | Understand what the current issue is by reading logs from different services, the IDE, Splunk, AWS logs and drawing diagrams of what the potential cause could be |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Automation**   * Introduces process efficiencies by automating the setting up/deploying of the project (infrastructure and applications) from scratch, both locally, including all tests, and to a hosted environment. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K13 Automation techniques, such as scripting and use of APIs. |  | Using Unity to deploy the infra to AWS in dev/prod envs, automating the tests using gitlab-ci |  |
| K17 What an API is, how to find them and interpret the accompanying documentation. |  | API – software component that allows two apps to talk to each other. Googling the api, or on the website host would have more info about the APIs on how to use |  |
| S12 Automate tasks where it introduces improvements to the efficiency of business processes and reduces waste, considering the effort and cost of automation. |  | Storing data logs of messages sent into a file/lookup rather than manually looking through every chat to see what message got sent on each date |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Security**   * Builds in security so that all data in transit is encrypted and secure. * Explains the types of threats and the rationale behind the decision to either encrypt data at rest or not. | | | |
|  |  | **Project Mapping** | **EPAO Feedback** |
| K16 How best to secure data; e.g. encryption in transit, encryption at rest and access control lists (ACL). |  | Using inhouse tools such as vault to securely store data such as usernames and passwords |  |
| S10 Assess identified and potential security threats and take appropriate action based on likelihood v impact. |  | Design likelihood v impact diagrams in the planning stage, set up access keys within Vault from early stages of production |  |