**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** | Michael Plasom-Scott |
| **ULN** | 7602378562 |
| **Training Provider** | Makers |
| **Employer** | Department of Work and Pensions |

**Project Brief**

|  |
| --- |
| The Customer Notifications Management Platform was a platform originally designed to support Virtual Machine Environment Replacement project systems, and ultimately the goal was to receive a simple input and output a print file. This process happened overnight. Over time more apps have been onboarded into this CNMP space, including Notifications SMS, and other (non-VMER) applications. These applications are often incredibly complex, time-sensitive, and required to process data 24/7. CNMP traffic is consistently growing, as are the print files processed, which means more complex templates required, and as a result of the new applications a lot more resources are being required. On top of these complexities, there are multiple areas where the platform can be improved upon. Notably, the monitoring solution doesn’t offer a comprehensive view even if used correctly. This means that the platform is not correctly identifying some error messages from Exstream, which cause Exstream to shut down with no alert generated. There are also other issues related to the alert stack resulting in alerts not being processed properly.  The solving of these issues becomes a project – Notifications Platform Enhancement. The aim of NPE is to ensure we end up with a platform that is reliable, has a high fault tolerance, and better detection of alerts with more rigorous testing and ultimately less failures. A major part of the project is the ability to utilise a blue-green deployment model that is achievable with Exstream applications. Blue-green deployment models employ the use of two distinct identical production environments, one that is running the current application version and one that is available to upgrade or test new releases on. The traffic then gets slowly re-routed via a load balancer from the blue env. to the green env., and then the updates are then made to the environment that is now behind. This decreases downtime, enables extremely quick rollbacks if something is wrong in the new environment, and allows more thorough tests to run on a production environment.  The specific part of the project I aim to take the lead on is the Secure File Transfer (SFT) inbound handling and routing component. The low-level architecture for this is still in draft, but the overview is the reception of a file to the SFT Inbox EC2 instance 🡪 the file is stored in AWS’ Elastic File System 🡪 the EC2 instance retrieves the file and processes it 🡪 it is stored in a ReadyToSend EFS outbox 🡪 it is sent from the EC2 instance to whichever production server is live at that time, either blue or green. This project will require experimentation, testing, and creating a lot of infrastructure using Terraform. It will also require me to be able to create the necessary Packer images to deploy the immutable infrastructure, and create Configuration as Code using Ansible. I will also have to consider the integration with existing architecture, such as the database that will be used to store archived files, and the use of failover tools to lessen the impact of any failures. |

**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. |  | * Use of feature branches * Tags (snapshot to release) * Merge requests * Multiple people contributing to one repo under various features |  |
| 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. |  | * SAST using tfsec to test for vulnerabilities in Terraform etc. * Use of dependency scanning in pipeline * Awareness of penetration testing on S3 database instance |  |
| K7 General purpose programming and infrastructure-as-code |  | * Terraform for overall infrastructure deployment * Packer for writing, building, and provisioning of images * Ansible for writing configuration as code * Python for switchover component |  |
| 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. |  | * Unit testing of Packer built image to ensure each line of code functions correctly. Ansible is in theory declarative and “self-testing” but use verification steps to test where this may not be guaranteed. * Terraform Plan as “unit testing” to ensure infrastructure is correct. * Fully automated testing of the overall component. Using tools like ServerSpec to ensure features are working. * Utilise ServerSpec abilities as a “unit test” Tests should always pass. Tests to be written before development. * Robot Framework to “integration test” the functioning of the SFT components. |  |
| 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. |  | * Static code analysis built into CI/CD pipelines * Static Application Security Testing built into CI/CD pipelines using tfsec |  |
| S11 Employ a systematic approach to solving problems, using logic and hypotheses / experimentation to identify the source of issues. |  | * Identify issue – document it thoroughly * Raise issue with documentation in a Jira ticket * Research and complete ticket and/or reach out for knowledge * Document solution in ticket and Confluence * Experimentation is needed for best solution on active/passive load balancing model |  |
| S14 Write tests and follow Test Driven Development discipline in various different contexts. |  | * Write tests before deployment with requirements for pass and failure based on needs * Tests should always pass. * Utilise ServerSpec abilities as a “unit test”. * Robot Framework to “integration test” the functioning of the SFT components. |  |
| S17 Code in a general purpose programming language. |  | * Write Python (or similar) for Lambda module to automate switchover of SFT |  |
| S18 Specify cloud infrastructure in an infrastructure-as-code domain-specific language. |  | * Infrastructure as code will be written using Terraform * I will use Packer to create machine images * I will use Ansible to provide the Configuration as code |  |
| S20: Writing code in such a way that makes merging easier and facilitates branching by abstraction - i.e. feature toggling. |  | * Semantic versioning for clear compatibility * Feature based commits and regression options * Feature flags to be used where appropriate |  |
| S22 Incremental refactoring by applying small behaviour-preserving code changes to evolve the architecture. |  | * Small commits, minor releases. Always backwards compatible if possible. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | * Reduce time to deliver changes, and patches * Improve automated failover time through clear planning * Reduce impact of failed changes – SFT inbox is built to support blue/green architecture on Comms Server |  |
| 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. |  | * Improve reliability * Main “user” is support teams – offer clear, easy to read logging using Splunk/Cloudwatch Logs * Component will be integrated to new “tracing” functionality which is a feature requested by wider business |  |
| K21 Architecture principles, common patterns and common strategies for translating user needs into both cloud infrastructure and application code. |  | * Attend design authority and support/present designs for this component * Understand the details of how this creates high availability and drives forward ability to deliver change more safely/quickly * Standard DWP practices and patterns to be complied with where possible |  |
| S3 Translate user needs into deliverable tasks, writing clear, concise and unambiguous user stories that the whole team can understand. |  | * Involvement in requirement gathering phase * Assist in analysis of designs and requirements and conversion to tasks. * Involvement in translating tasks onto tools like Jira boards/Confluence documentation * Involvement in sprint planning sessions |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | * Use of GitLab CI/CD * Merge small features/items regularly – fail fast, test early, fix early. * Fully automated throughout. No need to manually touch infra or manually run tests (except triggering in local dev) * High value role of DevOps Engineer to build tools, not to “operate” |  |
| K15 The principles and application of Continuous Integration, Continuous Delivery and Continuous Deployment, including the differences between them. |  | * Continuous integration with use of automated tests and frequent merges * Continuous delivery with use of automated releases * Continuous deployment use of automation in deployment phase |  |
| 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. |  | * Fully automated throughout. No need to manually touch infra or manually run tests (except triggering in local dev) * Particular use of Continuous Integration for Packer builds, and Continuous Delivery for Terraform. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | * Immutable infrastructure being image with all that is needed to run application * Components are therefore replaced with newer versions not upgraded * No modifications after deployment |  |
| S5 Deploy immutable infrastructure. |  | * Project as a whole deals with deployment of immutable architecture in various ways * Packer/image based pipeline * Automatic regular builds, automatic testing, zero risk deployments/patches |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | * Monitoring failures in system using Dynatrace/Splunk – using knowledge of failures and alert thresholds to find potential flaws * Configuring thresholds to get prompt updates on status * Use of data visualization to present and make data accessible to other teams |  |
| S6 Install, manage and troubleshoot monitoring tools. |  | * Tools like Dynatrace and Splunk must be installed, managed, and troubleshot * Integration with Splunk, monitoring software * Integration with pre-existing Notifications DB |  |
| S19 Interpret logs and metrics data within the appropriate context to identify issues and make informed decisions. |  | * Assist and advise in Splunk and Dynatrace metric config * Dynatrace, auto-thresholds and manual thresholds * Splunk alarms/dashboards for certain information/data |  |
| 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. |  | * Ongoing commitment to the product, forming part of the “team”. Not a siloed team that builds and runs. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | * Use of EFS for persistence of data on failure * Use of S3 for archiving * Ensure SFT Agent is compatible with Notifications Database that already exists |  |
| S7 Navigate and troubleshoot stateful distributed systems, in order to locate issues across the end-to-end service. |  | * Use of centralised logging to enable easy navigation and troubleshooting across service * Notifications database integration |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | * Use of shell scripts, Python, Lambda for provisioning and automation of tasks such as failover. * Terraform and declarative, idempotent languages where at all possible in scripting * Terraform uses AWS APIs |  |
| K17 What an API is, how to find them and interpret the accompanying documentation. |  | * Understanding of API as application program interface, and its role in communication between two programmes * Use of DWP Internal API exchange and documentation * External APIs such as AWS API documented online on vendor website. * Utilize AWS API and documentation provided by AWS |  |
| S12 Automate tasks where it introduces improvements to the efficiency of business processes and reduces waste, considering the effort and cost of automation. |  | * Creation and use of automated testing * Creation and use of automated deployments |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **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). |  | * Research and understand how best to secure data, bring research to team * Sensitive data using encryption at rest in amazon Encrypted file system * Configuration of TLS/HTTPS for data in transit between EC2 and EFS * Knowledge of how data encryption is handled within existing database * Use of security groups in order to secure data, knowledge of who has access |  |
| S10 Assess identified and potential security threats and take appropriate action based on likelihood v impact. |  | * Work with security architect to identify potential security threats * E.g, insecure storage of private keys used for EFS encryption, Malware attacks, vulnerabilities * Mitigate threats using scanning/vulnerability monitoring tools, anti-virus software * Knowledge of use of well-configured security groups in the case of a successful attack would prevent attack from spreading |  |