The Smart Columbus Operating System
Operations and Maintenance Plan

for the Smart Columbus Demonstration Program

FINAL REPORT | January 10, 2020
Notice
This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof. The U.S. Government is not endorsing any manufacturers, products, or services cited herein and any trade name that may appear in the work has been included only because it is essential to the contents of the work.

Acknowledgement of Support
This material is based upon work supported by the U.S. Department of Transportation under Agreement No. DTFH6116H00013.

Disclaimer
Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the Author(s) and do not necessarily reflect the view of the U.S. Department of Transportation.
Acknowledgements

The Smart Columbus Program would like to thank project leads and development team of the Smart Columbus Operating System for their assistance in drafting and reviewing this Smart Columbus Operations and Management document.
Abstract

This Smart Columbus Operations and Maintenance Plan provides operational information for the Smart Columbus Operating System. The Operating System is the central data store for all relevant program data within the Smart Columbus demonstration and provides users data through an open data portal web interface as well as Application Programming Interfaces (APIs). Smart Columbus will use this plan as a guide to perform the necessary operations for optimum program functionality in addition to properly securing, backing up, maintaining and sharing the data.
Table of Contents

Executive Summary......................................................................................................................................................... 1

Chapter 1. Introduction.................................................................................................................................................. 3
  1.1. Scope and Purpose ............................................................................................................................................. 3
  1.2. Supporting Materials and References ........................................................................................................... 3
      1.2.1. Account Access ........................................................................................................................................ 3
      1.2.2. Related Document Links ......................................................................................................................... 3
      1.2.3. Terminology Support ................................................................................................................................ 3
      1.2.4. Technical References ............................................................................................................................... 3
  1.3. System Description ............................................................................................................................................ 4
      1.3.1. Operating System Website ...................................................................................................................... 4
      1.3.2. Technical Architecture ........................................................................................................................... 5
      1.3.3. Open Source ............................................................................................................................................ 7
      1.3.4. Operations and Maintenance Overview ................................................................................................ 8
  1.4. System Users ...................................................................................................................................................... 8
  1.5. Stakeholders ...................................................................................................................................................... 9

Chapter 2. Materials and Resources .......................................................................................................................... 11
  2.1. Resources ......................................................................................................................................................... 11
      2.1.1. Training .................................................................................................................................................... 13
      2.1.2. Software Delivery Skills ........................................................................................................................ 14
  2.2. Materials .......................................................................................................................................................... 16
      2.2.1. Software as a Service .............................................................................................................................. 16
      2.2.2. Infrastructure as a Service ....................................................................................................................... 16
      2.2.3. Other Tools ............................................................................................................................................. 17
      2.2.4. Material Costs .......................................................................................................................................... 18

Chapter 3. Operations ................................................................................................................................................ 19
  3.1. Operations Overview ....................................................................................................................................... 19
      3.1.1. Goals and Expectations .......................................................................................................................... 19
      3.1.2. Hours of Operation .................................................................................................................................. 19
      3.1.3. Stakeholder Communications Channels ............................................................................................... 19
      3.1.4. User Communications Channels ......................................................................................................... 20
      3.1.5. Operational Activities ........................................................................................................................... 20
  3.2. InfrAstructure ManagEment ............................................................................................................................ 21
      3.2.1. System Administration .......................................................................................................................... 21
      3.2.2. Configuration Management .................................................................................................................. 24
Table of Contents

3.3. **Software Development**...........................................................................................................25
  3.3.1. Continuous Delivery .............................................................................................................25
  3.3.2. Open Source Code Management ..........................................................................................30
  3.3.3. Third Party Software ............................................................................................................30

3.4. **Security Measures** ..................................................................................................................32
  3.4.1. Security .................................................................................................................................32
  3.4.2. User Management .................................................................................................................34

3.5. **Data and Code Use** ................................................................................................................36
  3.5.1. Privacy Management ..........................................................................................................36
  3.5.2. Open Source Community Management .................................................................................37

Chapter 4. **Maintenance** .................................................................................................................39
  4.1. **Maintenance Overview** ......................................................................................................39
  4.2. **Monitoring** ............................................................................................................................39
    4.2.1. Security Monitoring ............................................................................................................39
  4.3. **Preventative Maintenance** ................................................................................................42
    4.3.1. Preventive Maintenance Tasks ..........................................................................................43
    4.3.2. Server Administration .......................................................................................................44
    4.3.3. Website Data Backup ........................................................................................................45
  4.4. **Content Management** ..........................................................................................................47
    4.4.1. Content Updates ..................................................................................................................47
  4.5. **Project Monitoring** ..............................................................................................................48

Chapter 5. **Ownership Transition Activities** ..................................................................................49

Chapter 6. **Closing** ..........................................................................................................................51

Appendix A. **Account Access Information** ....................................................................................53

Appendix B. **Related Document Links** ..........................................................................................57

Appendix C. **Acronyms and Definitions** .........................................................................................59

Appendix D. **Glossary** .....................................................................................................................61

Appendix E. **References** .................................................................................................................65

Appendix F. **AWS Monthly Billing Sample** ................................................................................67
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Ingestion Services</td>
<td>6</td>
</tr>
<tr>
<td>Table 2</td>
<td>Access Services</td>
<td>6</td>
</tr>
<tr>
<td>Table 3</td>
<td>Data Curation Services</td>
<td>6</td>
</tr>
<tr>
<td>Table 4</td>
<td>Roles and Responsibilities</td>
<td>11</td>
</tr>
<tr>
<td>Table 5</td>
<td>Monthly Reoccurring and Support Costs</td>
<td>18</td>
</tr>
<tr>
<td>Table 6</td>
<td>Communications Cadence</td>
<td>19</td>
</tr>
<tr>
<td>Table 7</td>
<td>Preventive Maintenance Activities List</td>
<td>43</td>
</tr>
<tr>
<td>Table 8</td>
<td>Update Activities</td>
<td>47</td>
</tr>
<tr>
<td>Table 9</td>
<td>Accounts Directly Owned by Program Manager</td>
<td>53</td>
</tr>
<tr>
<td>Table 10</td>
<td>Accounts Managed Through AWS (Secrets Manager)</td>
<td>53</td>
</tr>
<tr>
<td>Table 11</td>
<td>Accounts Managed through LDAP</td>
<td>55</td>
</tr>
<tr>
<td>Table 12</td>
<td>Accounts Managed Through <a href="mailto:developers.smartcolumbusos@gmail.com">developers.smartcolumbusos@gmail.com</a></td>
<td>55</td>
</tr>
<tr>
<td>Table 13</td>
<td>Personal Accounts that are Created and Granted Access</td>
<td>56</td>
</tr>
<tr>
<td>Table 14</td>
<td>Related Documents</td>
<td>57</td>
</tr>
<tr>
<td>Table 15</td>
<td>Acronym List</td>
<td>59</td>
</tr>
<tr>
<td>Table 16</td>
<td>Glossary</td>
<td>61</td>
</tr>
<tr>
<td>Table 17</td>
<td>AWS Monthly Billing Sample</td>
<td>67</td>
</tr>
</tbody>
</table>

List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Smart Columbus Operating System Home Page</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Operating System Architecture Diagram</td>
<td>7</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Operations and Maintenance Activities Overview</td>
<td>8</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Smart Columbus Operating System Operational Activities</td>
<td>20</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Joomla Page Builder Displaying Pages on the Operating System Website</td>
<td>22</td>
</tr>
<tr>
<td>Figure 6</td>
<td>FreeIPA Active User Section</td>
<td>23</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Jenkins Homepage</td>
<td>23</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Continuous Delivery Overview</td>
<td>25</td>
</tr>
<tr>
<td>Figure 9</td>
<td>LDAP Browser Example Access</td>
<td>35</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Data Curation Process</td>
<td>36</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Smart Columbus Operating System Maintenance Functions</td>
<td>39</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Kafka Performance Metrics</td>
<td>41</td>
</tr>
<tr>
<td>Figure 13</td>
<td>API Performance and Load Metrics</td>
<td>41</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Kubernetes Cluster Metrics</td>
<td>42</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Amazon Web Services Dashboard</td>
<td>45</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Akeeba Backup Interface</td>
<td>46</td>
</tr>
<tr>
<td>Figure 17</td>
<td>AWS RDS Interface</td>
<td>47</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Transition Flow</td>
<td>49</td>
</tr>
</tbody>
</table>
Executive Summary

As the winner of the U.S. Department of Transportation’s (USDOT) first-ever Smart City Challenge, Columbus was awarded $40 million in grant funding and the designation as America’s Smart City. Columbus is using this opportunity to “become smart,” which starts with realizing the power of data through the Smart Columbus Operating System.

The Operating System is designed to serve as the technological backbone of all current and future smart city projects. Additionally, this technology helps measure the performance and progress of the USDOT grant that was awarded to Columbus. Entrepreneurs and developers throughout Columbus and the open source community will be inspired to create Columbus' mobility services of the future, through the data that will live in the Operating System, while fueling technology deployments within the USDOT grant scope. These deployments include:

- **Enabling Technologies**
  - Connected Vehicle Environment - seeks to enhance safety and mobility throughout the city's transportation system utilizing Connected Vehicle technologies and applications, with an emphasis on congested and high-crash intersections and corridors.

- **Enhanced Human Services**
  - Multimodal Trip Planning Application with Common Payment System - makes multimodal options easily accessible to all by providing a robust set of transit and alternative transportation options including routes, schedules and dispatching possibilities, while facilitating payment for multiple transportation systems in one common platform.
  - Mobility Assistance for People with Cognitive Disabilities - aids cognitively disabled travelers to use the Central Ohio Transit Authority’s fixed-route bus system independently through the use of a turn-by-turn navigation mobile application.
  - Prenatal Trip Assistance - to help reduce Columbus’ high infant mortality rate by helping expectant mothers get to prenatal care.
  - Smart Mobility Hubs - creates a centralized location with access to different transportation options to help residents get where they need to go.
  - Event Parking Management - creates ability for drivers to be able to view Downtown and Short North parking garages and surface lots availability, reserve and pay for spaces using a smartphone or web app, reducing congestion caused by searching for available parking during peak times.

- **Emerging Technologies**
  - Connected Electric Autonomous Vehicles - uses automated shuttles to alleviate first mile/last mile challenges in the Linden Neighborhood, connecting residents to the fixed-route bus system and important community facilities.
Chapter 1. Introduction

1.1. SCOPE AND PURPOSE

This document is the Smart Columbus Operating System Operations and Maintenance (O&M) Plan. It describes the skills, technologies and processes that are used by the current team to support and maintain the Operating System, in sufficient detail to support the transfer of its ownership after the USDOT grant period of the Smart Columbus Project. It provides a comprehensive overview of the Operating System environment, the elements that make it work, and processes and procedures for maintaining optimum functionality. It describes the materials and resources, various aspects of operation activities, and maintenance tasks and routines taking place inside the Operating System. It also explains how operational and maintenance activities relate to the Data Management Plan and Data Privacy Plan.

1.2. SUPPORTING MATERIALS AND REFERENCES

1.2.1. Account Access

All of the services discussed throughout the plan have accounts that are needed to access and use the Operating System. To simplify, all of these are listed together in Appendix A: Account Access Information. This appendix includes the account name for the service, description, credential type, credential location and current owner. This can be used to assist in transitioning operational control through adding or removing team members.

1.2.2. Related Document Links

There are many documents, ranging from guidance on USDOT requirements to how to ensure secure data management, that support the Smart Columbus program. The primary documents that related specifically to the operations and maintenance of the Operating System are found in Appendix B: Related Document Links. Included in the list are the descriptions of the documents as well as the location of the latest version.

1.2.3. Terminology Support

Throughout the O&M plan there are many terms and acronyms used to describe the activities needed for operations and maintenance of the Operating System. The acronyms that are used with the Operating System are listed in Appendix C: Acronyms and Definitions. A collection of project specific terms to help provide context and understanding are listed in Appendix D: Glossary.

1.2.4. Technical References

Continuous learning is a critical part of supporting the Operating System. As a standard practice the Operating System team gathers, and shares learning resources that they found useful in the software delivery process for the Operating System. This resource list is included in Appendix E: References.
Chapter 1. Introduction

1.3. SYSTEM DESCRIPTION

The USDOT projects as well as the Operating System are being designed and built to collect data from a variety of inputs such as public, nonprofit, education-based, and private sector contributors. These inputs may come from other systems, devices, and people. The Operating System is a platform designed for big data, analytics, and complex data exchange. It will capture the data and provide a means for access to aggregate and fuse for analytics and visualization.

Datasets accessible through the Operating System include the Smart Columbus demonstration projects, traditional transportation data, and data from other community partners, such as food pantries and medical services. The Operating System will be scalable and will demonstrate the potential for serving city and private sector needs beyond the life of the Smart City Challenge period of performance.

The Operating System consists of three parts: the user facing Operating System website, the technical architecture, and the open source code/community. They are all cloud based and are not linked to one another, joined only by their role in the process of open source contributions.

1.3.1. Operating System Website

The Operating System website makes data accessible to people throughout the Columbus community and beyond. It supports users’ ability to discover, access and download data. For new visitors, the Operating System website offers stories on how data is currently being used in real life scenarios as well as ways to navigate the datasets available to the public. It also provides links to contribute data and a “Contact Us” form to provide feedback or suggestions.

The site has been live since December 2017 and was opened to the general public in March 2018. As the website evolves, it will become even more user-centric to close the gap between analysts and the general public to solve transportation issues.

The Operating System’s Web presence is powered by Joomla (https://www.joomla.org/), an open source content management system (CMS). Release-ready open source code is prepared and transferred from the staging server to a production server, where they are processed before being uploaded to the Operating System website for public use. The production Operating System site can be found at https://www.smartcolumbusos.com/.
1.3.2. Technical Architecture

The technical architecture is designed to establish a regional platform that is sustainable, easily used to support the data ingestion needs of the USDOT grant projects, facilitate agency efficiency for city governments and be used by others. The Operating System architecture was created using the following guidelines:

- Highly Scalable – Ability to scale up to petabytes of data under management
- Cloud Agnostic – Use the cloud service provider of choice
- Plug-and-Play – Use only the microservices needed
- Near Real-Time – Stream data to, and consume streaming data from, the platform

The Operating System platform consists of three components: ingestion, access, and data curation. Each component is a collection of custom microservices built to a specific task. Note, the implementation of the Operating System for the City of Columbus utilized the AWS cloud service. The Operating System is designed and built to be easily deployable using other cloud services as well.

1.3.2.1. INGESTION

Within the ingestion component, there are microservices to gather, normalize, persist, and profile data.

Figure 1: Smart Columbus Operating System Home Page

Source: City of Columbus
Table 1: Ingestion Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gather</td>
<td>Retrieve/receive data to process into the platform</td>
</tr>
<tr>
<td>Normalize</td>
<td>Transform data to adhere to internal platform standards/schema</td>
</tr>
<tr>
<td>Persist</td>
<td>Upload data to storage for later analysis</td>
</tr>
<tr>
<td>Profile</td>
<td>Tracks ingestion performance statistics</td>
</tr>
<tr>
<td>Convert</td>
<td>Converts file type by creating and alternative version to ingest</td>
</tr>
</tbody>
</table>

Source: City of Columbus

1.3.2.2. **ACCESS**

The access component includes a web socket application programming interface (API), a representational state transfer API (REST API), and a user interface (UI) frontend. Documentation about using the API is included in the Operating System web site, available on each page describing one of the datasets.

Table 2: Access Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket API</td>
<td>Push data to subscribing users</td>
</tr>
<tr>
<td>REST API</td>
<td>Allow users to access data over Hyper Text Transfer Protocol Secure (HTTPS)</td>
</tr>
<tr>
<td>UI</td>
<td>Allow users to search and browse datasets</td>
</tr>
</tbody>
</table>

Source: City of Columbus

1.3.2.3. **DATA CURATION**

The administration component is an API for dataset and organization management.

Table 3: Data Curation Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage</td>
<td>Allow data curators to define datasets with metadata</td>
</tr>
</tbody>
</table>

Source: City of Columbus

Components (and the microservices within those components) are completely decoupled allowing the Operating System enhancements to be easier due to modular architecture. The microservices subscribe to Kafka topics, publishing and reacting to system events instead of communicating directly with each other. The event stream is the central hub used for all the components of the platform to communicate without having to be directly coupled to each other. Everything subscribes to the event stream message queue and when it takes action in the system, either by user input or external stimulus, the event is broadcasted to the other components that are programmed to listen for and respond to. Data persists in
object storage (e.g. S3) via PrestoDB, a distributed structured query language (SQL) engine that allows performant data storage and query, providing scalable storage.

Figure 2 provides a high-level view of the major components of the Operating System. For more detailed information about the software architecture and implementation, refer to the documentation included with the software repositories on GitHub, which are continually updated as enhancements are added to the system. Additional documentation of the system architecture is included in the Smart Columbus Operating System’s System Architecture and Standards Plan.

Figure 2: Operating System Architecture Diagram

Source: City of Columbus

1.3.3. Open Source

The Operating System is an open source software project which is freely available for external adopters and contributors. Open source initiatives embrace and celebrate principles of open exchange, collaborative participation, rapid prototyping, transparency, meritocracy, and community-oriented development.

Smart Columbus aims to:

- Make source code of the Operating System available to the public
- Provides a seamless contribution experience
- Make the platform easily deployable for all cities

The Operating System source code is made freely available on GitHub, an industry-standard platform for hosting open sourced code. It can be accessed at https://GitHub.com/smartcitiesdata. In order to maintain the resource of continuous feedback and contribution to a shared, open codebase, it is important to foster a healthy community around the code and an open source code management process. These processes allow the community of developers to extend the Operating System codebase as well as deploy and manage the running instance of the platform. The detailed open source activities are outlined in Sections 3.3.2 and 3.5.2.
### 1.3.4. Operations and Maintenance Overview

Proper operations and maintenance of the Operating System are needed to support the ability to ingest data, secure user access and system availability. The operations functions are defined by the activities needed to operate and enhance the Operating System. The maintenance functions are defined by the activities needed to ensure proactive monitoring, back up of systems, and processes established to keep content current. There are many activities that could be considered maintenance included in operations like patch updates. Where these have been integrated in the operations activities in how the software is delivered, they are also included in the operations section of this plan.

The functions needed to support the Operating System operations include infrastructure management, systems access, software deploy/enhancement, data and code use, and system enhancement. Chapter 3 provides the detail of operations activities. The maintenance functions are Monitoring, Preventive Maintenance updates and Content Management. The maintenance activities details can be found in Chapter 4.

![Figure 3: Operations and Maintenance Activities Overview](image)

_Source: City of Columbus_

### 1.4. SYSTEM USERS

There are differing types of users accessing the Operating System and their roles and access privileges vary. The main user roles interacting with the Operating System are:

- **Visitor / Data User** – Anyone who can access the Operating System website.
  - Can read public information including articles and view descriptions of projects
  - Can freely browse and interact with public datasets on the Operating System
  - Can access public data sets for analysis or application development
  - Can contact the Operating System via contact form
  - Can request to provide public data to be ingested into the Operating System

- **Restricted Data User** – A visitor who can access restricted data sets via identification access management
Can access specific restricted data sets
Can request to provide restricted data to be ingested into the Operating System

GitHub Repository User – A software developer who accesses the Operating System code on the GitHub repository.
Can read public information as a visiting software developer
Can download open source application source code
Can participate in the software development discussion forums
Can submit online requests, including request to upload open source code or request to collaborate with other developers

1.5. STAKEHOLDERS

A summary of the stakeholders’ roles and responsibilities is captured below:

USDOT – overall grant administrator and visionary of the power of data in relation to transportation and mobility needs. Help to ensure that all teams involved in the building of the Operating System are good stewards of the grant funds through alignment with the original mission of the grant.

City of Columbus – winner of the USDOT grant and day-to-day administrator of grant funds for all projects involved in the overall portfolio. Serve as adopters and stewards of the concept of a Smart City and leveraging data to make informed decisions.

Columbus Community – end users of the projects in the USDOT portfolio and advocates of the Smart City initiative. The people that are ultimately being served by all of the work being funded by the USDOT grant, including users of the Operating System and providers of data.

Technical Working Groups – working groups serve as technical resource advisors and provide input to the Smart Columbus project team which includes the city and consultant team as they develop concepts of operations, consider use cases, data needs and deployment of the projects.

Independent Evaluators (IEs) – serve as the evaluators for the Smart Columbus program including the Operating System. IEs analyze the baseline and post-implementation data from the projects to measure their performance.

USDOT Grant Project Teams – the teams of the USDOT portfolio projects are responsible for executing the projects aligned to the grant goals.

Smart Columbus Operating System Team – builds, operates and maintains the Operating System for the period of the grant.
Chapter 2. Materials and Resources

This section identifies the equipment/materials and personnel/resources used in system operations and maintenance.

2.1. RESOURCES

This section includes personnel aligned to their role and responsibilities needed for Operating System operation and maintenance.

Table 4: Roles and Responsibilities

<table>
<thead>
<tr>
<th>Personnel</th>
<th>Roles &amp; Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Manager</td>
<td>Oversees the operations, including scope, schedule and budgets, as well as evaluation requirements of the program, delegating to consultants, city employees or program partners as needed. Post grant this role would transform to a CEO or General Manager providing high level oversight.</td>
</tr>
<tr>
<td>Product Owner</td>
<td>Oversees roadmap and drives the prioritization of work that is aligned to the goals and roadmap for the Operating System through balancing user feedback, risk, effort, and potential value to the overall success of the Operating System.</td>
</tr>
</tbody>
</table>
| Software Developers | • Works within a culture and practices where the entire engineering or product team shares responsibility for the site reliability, operations, maintenance, security, and deployments of new features and bug fixes.  
  • Provides production support to existing applications and infrastructure.  
  • Implements minor enhancements using continuous integration development practices (for example, additional validation and translation rules)  
  • Implements bug fixes  
  • Maintains the system test suite using Test-Driven Development practices. (Note, on some teams, there may be a distinct role for software testers. This responsibility has been performed as part of the software development role on the current team.)  
  • Maintains the system development and deployment pipeline |
| System Administrator| • Role supported by Software Developer in a culture where the entire engineering or product team shares responsibility for the site reliability, operations, maintenance, security, and deployments of new features and bug fixes  
  • Administers user account additions and removals  
  • Configures and keeps up to date with patches for production and non-production servers, externally sourced tools, and externally sourced code libraries  
  • Engages with the open source community to plan for and prioritize feature requests  
  • Reviews contributions from the open source community to ensure code quality, test coverage and correct functionality |
### Personnel | Roles & Responsibilities
--- | ---
**Technical Lead** | • Acts as a member of the development team, performing all of the Software Developer and/or System Administrator roles, plus:
  o Provides leadership for technical architecture decisions
  o Provided input on requirements, practices, policies and communicating them to the development team.

**Scrum Master** | • A business-driven coordinator who works closely with product owners and stakeholders to ensure agile development processes are properly followed and prioritized work is completed.
  • Maintains project governance items including value stories, budgets, and people allocation.

**Data Curator** | • Serves as a primary point person for data providers, reviewing data to ensure data aligns with all policy requirements, working with the development team to ingest data and ensure freshness.
  • Data provider relationship management
  • Work with providers to review data that may be ingested, to determine data quality, value, and ethics align with the Smart Columbus mission
  • Prepare datasets (metadata and data) for ingestion
  • Perform ingestion using the data curation interface
  • Owns de-identification process and ensures all data ingested does not include Personally Identifiable Information (PII)
  • Periodic reviews of existing datasets

**Chief Security Officer** | • Duties may be shared by one person serving as Security and Privacy Officer if multiple people are not assigned individual roles
  • Chief Security Officer is a program level role but dedicates time to the Operating System project to lead security
  • Lead security strategy to support Operating System availability, integrity, and confidentiality
  • Consult on monitoring and responses to security issues
  • Ensures alignment and evolution of Data Management Plan
  • Provides security training

**Chief Privacy Officer** | • Duties may be shared by one person serving as Security and Privacy Officer if multiple people are not assigned individual roles
  • Chief Privacy Officer is a program level role but dedicates time to the Operating System project to lead privacy
  • Ensure the privacy of personal information
  • Consult on data management
  • Ensures alignment and evolution of Data Privacy Plan and Policies
  • Provides privacy training
### Personnel Roles & Responsibilities

<table>
<thead>
<tr>
<th>Personnel</th>
<th>Roles &amp; Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI/UX Designer</td>
<td>• Content Management&lt;br&gt;• Maintains access to the website backend for making changes to a section or the whole website. Able to prepare changes and publish it on the Operating System website. Can manage website resources, including user accounts.&lt;br&gt;• User Experience&lt;br&gt;• Design and implement changes based on evolving user needs&lt;br&gt;• Involved in user facing issues from ideation to implementation&lt;br&gt;• Ensures submissions to Operating System website are properly addressed&lt;br&gt;• Ensures form and function are maintained and improved through the product lifecycle</td>
</tr>
</tbody>
</table>

*Source: City of Columbus*

2.1.1. **Training**

A team of individuals with proper skills are required to ensure all functions required to support the Operating System operations. This document is not intended to teach the required skills necessary for operating the Operating System, instead it is to identify the needed skills and their relevance to the Operating System. Proficiency in the knowledge areas described below are required for operating the Operating System production environment efficiently.

Most of the skills described below can be acquired through either hiring someone who possesses the knowledge and skills already or has experience at an intermediate proficiency level and can learn the details of the operational or maintenance tasks over time.

2.1.1.1. **INFRASTRUCTURE AND OPERATIONS SKILLS**

- **Content Management System (CMS) Administration** – Possesses operational knowledge and skills for keeping a website working properly. At a minimum, they have the ability to upgrade CMS software, update extensions, perform the user management process and restore the site from the existing automated backups.

- **Open Source Administration** – Familiarity with maintaining projects using source control systems such as GitHub. They must have the ability to create and maintain source code repositories as well as managing users and their access levels. More importantly, they are able to package ready to release code so that it can be deployed to production environments. They also need to have the ability to manage discussion forum and remove/address inappropriate postings.

- **System Administration** – Familiarity with the Linux operating system and has the ability to identify and execute necessary commands to perform operations and maintenance duties. This skill is not used on a daily basis but is required in critical operations such as writing server provisioning scripts and applying emergency security patches. They also need to possess working knowledge of system security with the ability to analyze and identify network issues on system infrastructure.
Chapter 2. Materials and Resources

- Amazon Web Services (AWS) – Since the Operating System operations are hosted on Amazon servers, an individual with working knowledge of AWS is needed for procuring and configuring the server resources. This skill involves various aspects of operating a cloud-based computing environment for enabling and disabling resources to scale up or down as the Operating System demands. (Note, if the Operating System were moved to a different cloud service in the future, expertise in that cloud technology would be needed instead. That possible change is outside the scope of this document.)

- Terraform – As the Operating System AWS infrastructure management is automated and codified using Terraform, an individual should be familiar with utilizing Terraform to provision, decommission and document system infrastructure.

- Kubernetes – A large portion of Operating System exists inside of the Kubernetes ecosystem to provide orchestration, fault tolerance and high availability. Individuals working to administer the Operating System must be able to navigate Kubernetes deployments to monitor its health, deploy updates, and troubleshoot issues in production environments.

- Continuous Integration (CI) – All changes to running production software is built and deployed using a combination of Jenkins and Travis CI. System Administrators must ensure they are properly up to date and running all automated jobs.

2.1.2. Software Delivery Skills

- Software Development – A software developer should be proficient in the following programming languages for the Web:
  - HTML, CSS, and JavaScript (React)
  - Elixir and functional paradigms
  - Infrastructure as code using Terraform, AWS, and Kubernetes
  - Agile software development principles and practices

- Delivery Leadership – Ability to balance team needs while being aligned to driving outcomes. Ability to track and report delivery forecasting and burn up metrics. Servant leadership skills such as building team and individual health by encouraging, enabling, and energizing people.

- Data Curation – Data analytic and management skills including the following
  - Ability to curate data from various sources and formats
  - Ability to build relationships with data providers
  - Ability to review data for PII, data quality, and value

- Security and Privacy – Current knowledge on software, data privacy, security practices, tools, and processes
  - Ability to take a risk-based approach considering actual threats and organizational drivers
  - Security automation tooling for static and dynamic code analysis as well as automating security throughout the software development lifecycle (SDLC)
  - Understanding of Open Web Application Security Project (OWASP) top 10 and secure coding practices
  - Experience with security awareness and educating peers on security and privacy best practices
- Security Operations experience including incident response, security monitoring with relevant tooling, and analysis
- Experience working with sensitive data such as PII and working with regulations such as HIPAA, GDPR, or other related regulations

- User Experience & Content Management – Well versed in designing and implementing changes which are user facing.
  - Maintains access to the website backend for making changes to a section or the whole website.
  - Able to prepare changes and publish it on the Operating System website.
  - Can manage website resources, including user accounts.
  - Ability to publish various forms of media on the website.
  - Versed in basic front end development, copywriting, and design.
2.2. MATERIALS

This section specifies the type of materials and costs broken out by software as a service, infrastructure as a service and other tools (hardware and software). Each subsection describes how the materials are used. The last subsection lists out all the costs associated with operating and maintaining the Operating System.

2.2.1. Software as a Service

Software functionality/tools not managed or maintained by the Operating System but used the development lifecycle.

- GitHub – All source code for the Operating System is managed through GitHub. GitHub also provides features and capabilities for managing a collaborative code sharing environment. (Web site for more information: www.github.com)
- Dockerhub – Stores pre-built software packages of Operating System applications. (hub.docker.com)
- Travis – Validates and builds the open sourced applications of Operating System. (travis-ci.com)
- Slack – Provides inter-team communication tool. (www.slack.com)
- Gitter.im – Extra-team communication tool (team to open source community) (gitter.im)
- ZenHub – Issue tracking and organization for project agile planning (www.zenhub.com)
- Hex.pm – Stores pre-built software libraries developed by Operating System team (Elixir) (hex.pm)
- Npmjs.com – Stores pre-built software libraries developed by Operating System team (Nodejs/javascript) (www.npmjs.com)

2.2.2. Infrastructure as a Service

Infrastructure as a Service is consuming compute and storage resources on-demand and paying by use rather than assuming the capital expenditure and ongoing maintenance of physical hardware.

2.2.2.1. AMAZON WEB SERVICE

AWS is a cloud-based provider of compute resources. (Web site for more information: aws.amazon.com) Since these services are managed independently, only a high-level description is provided in this section. Directly managed materials and resources are covered in greater detail in Chapter 3.

The following AWS services are used in providing the Operating System platform:

- Elastic Compute Cloud (EC2) service – Provides virtual servers instantiated from Amazon Machine Images provided by AWS. These are simple virtual private services which can be configured to host nearly any necessary application. Outside of the EC2 servers which are provisioned according to EKS configuration (detailed below), one-off EC2 servers are used for stand-alone services.
- Elastic Kubernetes Service (EKS) – Provides managed Kubernetes in AWS. The service is backed primarily with EC2 servers (detailed above), with an additional layer of a control plane providing orchestration of Operating System services onto EC2 servers.
- Simple Storage Service (S3) – Provides massively scalable storage for the Operating System for storage of ingested datasets, backups and various assets needed to help the Operating System operate efficiently.
- Relational Database Service (RDS) – Provides managed SQL databases used by the Operating System as backend storage for necessary services and applications such as Joomla.
• Secrets Manager – Provides secure storage of passwords, tokens, and other sensitive access credentials

• Identity and Access Management (IAM) – Provides identity and access management for authorizing use of AWS resources by human and automated service account users

• Redis for Elasticache – Provides persistent storage of platform system state in a robust, fast-access caching service

• Route53 – Provides Domain Name System resolution services, translating human-readable domain names (www.example.com) to network addresses (192.168.0.1).

• Elastic Container Service (ECS) – Runs Jenkins CI Server which automates deployments to Operating System development and production environments

• Elastic Container Registry (ECR) – ECS provides containerized service management with less complexity/functionality than EKS (see above). ECR provides private hosting of container images.

• Virtual Private Cloud (VPC), network routes, security groups – Provides isolated private network environments with access controls for ingress and egress of traffic.

2.2.3. Other Tools

2.2.3.1. SOFTWARE TOOLS

This section discusses the operating equipment, software tools, and other tangible computing hardware used for operating the Operating System.

Software programs and services used by the Operating System team. As appropriate, these tools may be executed locally on the developers’ workstations, or executed on shared servers accessed over the computer network.

• Terraform – Defines and deploys cloud infrastructure for the operating system as code which can be tracked in source control. (Local) (Web site for more information: www.terraform.io)

• Heptio – Authenticates and authorizes changes to AWS environments. (Local) (www.heptio.com)

• Kubectli / Helm – Issues commands to, alter, and maintain Kubernetes services. (Local) (kubernetes.io/docs/reference/kubectl and helm.sh)

• Tiller – Executes actions to alter and maintain Kubernetes services in response to Helm (Server) (helm.sh)

• Jenkins – Automates deployments to Operating System development and production environments (Server) (jenkins.io)

• OpenVPN – Secure and restricted access to platform network resources and tools (Server) (www.openvpn.net)

• OWASP Zed Attack Proxy – provides penetration testing (Server) (www.zaproxy.org)

• Kibana – provides a web interface to search system logs (www.elastic.co/products/kibana)
  ○ Fluentd – collects and aggregates those logs for Kibana to search and visualize (www.fluentd.org)

• Joomla – CMS for web presence (Server) (www.joomla.org)
  ○ Akeeba – Automates back up services for Joomla (Server) (www.akeebabackup.com)
2.2.3.2. HARDWARE

- A new development/support team would need laptops or desktops in order to do their work. The existing development/support team predominately used macOS, but any operating system should suffice.

2.2.4. Material Costs

Table 5: Monthly Reoccurring and Support Costs, summarizes the monthly costs for the all materials. All costs have been normalized to the monthly level. If a software, service, or other resource is not listed below then there are no costs involved with using or accessing.

<table>
<thead>
<tr>
<th>Material/Software</th>
<th>Description</th>
<th>Monthly Costs</th>
</tr>
</thead>
</table>
| Amazon Web Services (hosting) (EC2, SES, S3) | • Production server are always-on  
• Development servers are enabled as needed and can be disabled after use  
• S3 storage for system backup  
• Costs vary on usage and a sample breakdown of cost can be found in Appendix F | ~$12,000/month (includes all environments) |
| GitHub Repositories        | • 12-private repository package(s)                                           | $88/month                          |
| Akeeba                     | • Akeeba Backup PRO  
• Akeeba Release System                                                   | $4.68/month                        |
| ZenHub                     | • 12 users  
• Add-on for GitHub that supports issue management and tracking             | $5.00/month                        |
| OpenVPN                    | • 12 users  
• Provides secure access to the development/testing environment             | $15/month                          |
| Logman                     | • User & site analytics                                                     | $8.25/month                        |
| SP Page Builder            | • Page builder for Joomla                                                   | $3.92/month                        |

Source: City of Columbus
Chapter 3. Operations

The following section includes a detailed discussion of operational activities for the Smart Columbus Operating System. It provides insights into day-to-day events and describes high level processes governing system operations.

3.1. OPERATIONS OVERVIEW

3.1.1. Goals and Expectations

The Operating System provides open data sharing and collaboration for users from around the world. The primary objectives of the Operating System are as follows:

- Providing a secure and collaborative environment for sharing the Operating System open source code
- Providing a secure and collaborative environment for sharing the Operating System open data
- Keeping the Operating System operational and providing optimal data services to its users
- Be prepared for addressing and resolving emergency situations promptly

3.1.2. Hours of Operation

The Operating System is expected to be operational continuously except for scheduled downtimes. Currently, the Operating System technical support is available only during normal working hours. The Operating System team is available from 9 am ET to 5 pm ET, Monday through Friday. Any website issues that occur during off hours are addressed on the next business day.

3.1.3. Stakeholder Communications Channels

Stakeholder awareness, feedback and involvement is critical to operating and maintaining the Operating System. Table 6: Communications Cadence provides a list of typical cadences to ensure ongoing communications.

Table 6: Communications Cadence

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Stakeholders</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backlog Refinement</td>
<td>Prioritize work to be done by the team in the near future</td>
<td>Operating System Team Leads</td>
<td>Weekly</td>
</tr>
<tr>
<td>Team Leads Meeting</td>
<td>Review of work done and in progress (at a higher level than daily standups), and address any blockers</td>
<td>Operating System Team Leads</td>
<td>Weekly</td>
</tr>
<tr>
<td>Demo</td>
<td>Demonstration of work completed and gathering Stakeholder feedback</td>
<td>All</td>
<td>Monthly</td>
</tr>
</tbody>
</table>
### 3.1.4. User Communications Channels

Users have multiple channels to communicate with the Operating System team and each other. The following provides a list of the primary channels:

- **Website** – Visitors and users can complete a “Contact Us” form on the website to submit feedback, make an inquiry, or to leave a comment. All submissions are routed to the Operating System team. Suggestions for enhancements to existing features and for new features are reviewed by the Product Owner, and may be added to the prioritized product backlog for development by the team.

- **Survey** – User feedback is collected through a survey found on the website.

- **Discussion forum** – User feedback or questions from outside developers can be posted on the Gitter public forum, which is monitored by the Operating System team.

- **GitHub** – Outside developers can participate in the discussion forums and submit online requests, including request to upload open source code or request to collaborate with other developers.

### 3.1.5. Operational Activities

The operations functions are defined by the activities needed to operate and enhance the Operating System. These functions include Infrastructure Management, Systems Access, Software Deploy/Enhancement, Data and Code Use, and System Enhancement.
3.2. INFRASTRUCTURE MANAGEMENT

The Smart Columbus Operating System project has a functioning infrastructure in existence for providing the software development and production (user-facing) environments. All interactions with the infrastructure are for ongoing maintenance purposes only.

3.2.1. System Administration

System administration is comprised of administration for AWS, Joomla, Lightweight Directory Access Protocol (LDAP) and Jenkins.

3.2.1.1. AMAZON WEB SERVICE (AWS) ADMINISTRATION

The Operating System is hosted inside of the AWS cloud, with all services and servers running inside of one or more amazon resources. The Operating System uses the following AWS features:

- Elastic Kubernetes Service (EKS)
- Relational Database Service (RDS)
- Route53
- Elastic Container Services + Elastic Container Registry (ECS + ECR)
- ElastiCache for Redis
- VPC, Security Groups, and Subnets
- Identity and Access Management (IAM)
- Amazon Simple Storage Service (S3)

3.2.1.2. AWS COMMON MAINTENANCE ACTIVITIES

Administration activities for maintaining AWS are scoped to adding and revoking privileges via AWS IAM. This occurs only when the composition of the team changes. Creation and removal of resources in AWS should be performed using Terraform and is considered out of scope of typical administration and would be a development task.

3.2.1.3. JOOMLA ADMINISTRATION

Joomla is a CMS for hosting the Operating System website. The Joomla CMS allows the Operating System website to be built upon its software platform, which comes with key portal functions such as:

- User management
- Website templates
- Page building wizards
- Basic navigation
- Menus
- Color theme
- Screen layout options
- User interaction
3.2.1.4. JOOMLA COMMON MAINTENANCE ACTIVITIES

The Operating System goes through software upgrades for updating Joomla CMS software itself and also for the Joomla extensions. This task involves installing software patches and code changes to Joomla via a back-end administrator panel. Joomla updates are released irregularly by the Joomla community and only security critical patches are applied to the Operating System in an expedited manner. Non-security Joomla updates and versions are selectively chosen when required for increased functionality and when updates are least disruptive to the Operating System team development cycle. The team is notified of available updates via the Slack Rich Site Summary (RSS) feed (see Section 4.2 Monitoring) and historically has occurred several times each month. Joomla is typically installed on a development server first for visual and functional verification before being released to the production environment, unless the change is very minor. Effort varies based on the software being changed.

3.2.1.5. LDAP ADMINISTRATION

IAM for non-AWS resources such as Continuous Integration systems, Virtual Private Network (VPN) access, etc. is managed through FreeIPA LDAP tool. The Operating System uses FreeIPA as its LDAP backend. Administrative activities mostly consist of adding and removing users from the system and groups. This is covered further in Section 3.4.2 below.
3.2.1.6. JENKINS ADMINISTRATION

Administration of Jenkins is typically minimal and only includes patch management which is often applied as a development activity. Administrators are responsible for updating, installing, and removing Jenkins plugins. More details about Jenkins is covered in Section 3.3 below.

3.2.1.7. JENKINS COMMON MAINTENANCE ACTIVITIES

Maintenance activities for administering Jenkins is only involved in keeping Jenkins and its plugins up to date. This is often a developer task as Jenkins configuration is codified, but when stop gaps become necessary this is added to normal system administration. Additionally, Jenkins jobs may need to be executed to change or restore system state for various applications. These jobs are typically codified and maintenance is included in development. This is covered more in Section 3.2.2.3 below.
3.2.2. Configuration Management

All Operating System microservices are configured via configuration management code stored in source control following the concept of Infrastructure-as-Code. All configuration changes are thereby committed to a GitHub source code repository. All changes are audited by the team as these changes move through the CI pipeline and are approved by the team before they are deployed to production. This code is stored in the “common” repository of the SmartColumbusOS GitHub organization and deployed via the Terraform tool.

Systems are configured via two forms of configuration management code:

- Hashicorp Terraform’s Hashicorp Configuration Language (HCL) for cloud provider infrastructure
- Yet Another Markup Language (YAML) which is formatted Kubernetes manifest files produced by the Helm templating tool

3.2.2.1. CLOUD INFRASTRUCTURE

All cloud infrastructure resources are defined in HCL, a json-like configuration domain specific language. The following is a list of the cloud infrastructure resource:

- Kubernetes managed control plane (EKS)
- Kubernetes virtual server nodes (EC2)
- Managed PostgreSQL database (RDS)
- Managed Redis caching service (Redis for ElastiCache)
- S3 object storage buckets
- IAM user accounts and access policies
- Secrets (password or token values) (SecretsManager)
- Elastic Container Service (currently hosting the Jenkins service) (ECS)
- Network abstractions (network routes, VPC containers, security groups, and network access translation (NAT) gateways
- Ancillary virtual machine instances (such as those running the FreeIPA service) (EC2)

Configuration values that vary from one environment to the next such as names, or sizing values that determine instance sizes, counts, or resource allocations are abstracted from the configuration code into environment-specific variables in the “variables/{environment}.tfvars” files of the “common” repository.

3.2.2.2. MICROSERVICES

Microservices are small, single-purpose software applications that are loosely coupled to one another and combine to provide broader functionality. The Operating System components primarily run in Docker containers managed by Kubernetes. Their deployment configuration is defined by the merger of a manifest of declarative deployment instructions and generic values with an override file of values in a “deploy” repository.

The generic microservice charts are defined via YAML template files in the “charts” repository of the smartcitiesdata GitHub organization. These templates are written in YAML with variable placeholders written in the Go programming language templating standard (https://golang.org/pkg/text/template/). Default values for quantities such as container counts or allocations of memory and central processing unit (CPU) time, as well as names, are contained in the “values.yaml” file of each chart. Because the general platform the Operating System is built on is intended to be used by additional cities other than Columbus, Ohio, default values are provided for any user that allows deploying the system in a
minimally functional way (low resource/dollar consuming, not performance optimized) that can be customized by each deploying city to meet their specific requirements. Values specific to the Operating System used to override default values in a chart are contained in the “{application-name}.yaml” file of the SmartColumbusOS/{application-name}-deploy repository for each application. These override values are defined in the same structure as the default values for the application’s chart but with different specifics pertaining to the Operating System. When deployed via the Helm tool, the override values of the deploy repositories replace the default values when templates are generated and before they are sent to the Tiller server in Kubernetes to apply the changes to the running application.

3.2.2.3. JENKINS DEPLOYMENT

All of the Operating System infrastructure and applications are deployed via Jenkins. Jenkins jobs are defined in a Jenkins file within the application’s “deploy” repository or the “common” repository in the SmartColumbusOS GitHub organization. Jenkins files are a declarative list of instructions written in the Groovy programming language and orchestrate the execution of Terraform and Helm command line tools. HCL infrastructure code and YAML application templates deploy the Operating System components. Occasionally Jenkins files define commands to query the cloud provider for information needed to supply deploy-time variables to application templates such as network identifiers.

3.3. SOFTWARE DEVELOPMENT

3.3.1. Continuous Delivery

The Operating System team practices continuous delivery (CD) through the CI of new features and bug fixes into the code base, frequently deploying new versions and patches to production multiple times a week or even in a single day. Small increments of work are continually discovered, developed and deployed to production, following this sequence:

- Feature Discovery – user needs are gathered and evaluated to determine features that would create the most business value
- Feature Development – where features are built and tested through CI
- Feature Deployment – deploying and demoing new features and patches to production as they become available

Figure 8: Continuous Delivery Overview

Source: City of Columbus

3.3.1.1. FEATURE DISCOVERY

If it is generally understood “what” is being done when a project begins, then the discovery phase is the beginning of understanding the “why” and “how.” Discovery is a cross functional design process translating business requirements into tangible, usable software. A larger discovery may be held at the very beginning of a project to develop a roadmap and flesh out epics (group of stories that together create
value) and stories (small chunks of work aligned to business value). Iterative discovery sessions can be conducted each time to add or refine a feature.

- Discovery activities may include group ideation, sketching, wireframing, paper prototype testing, voting, prioritization, and story sizing
- Length and depth of discovery depends on size of feature (smaller features may be a few days, larger features may be a few weeks)
- May include development or use of personas (represents a specific user and their needs)
- Stakeholders and end users are interviewed and consulted during discovery to determine the rationale (problem statement) and importance (in order of priority) of the feature, as well as how to best bring it into reality
- Working within Agile, just enough high-fidelity design should occur in order to allow the developers to quickly size and start work
- Iterative testing and feature refinement occurs via fast feedback from front-line users in the production environment

### 3.3.1.2. FEATURE DEVELOPMENT

The Operating System is continually enhanced with new features and capabilities. Each feature requires time and resources to design, develop, test and deploy to production. In this section, the workflow of the Operating System feature development is described.

#### 3.3.1.2.1 Defining the Story

Small, story-driven feature branches are created for every piece of work. The code is thoroughly tested via a suite of test code that is automatically run against each attempt to merge changes. The frequency and small size of changes prevents feature code from drifting too far from the master branch in any given change. Broad test coverage and automation ensure bugs are more difficult to merge into the master branch, allowing for a higher degree of confidence in making changes directly to the master and releasing them to production early and often. Steps for creating stories is as follows:

- Describe a concept for the feature.
- Write a user story that describes the value of the work to the user, the criteria for when the work is considered done, and any technical notes that help the developers.
- Estimate the relevant amount of effort for this specific work and prioritize when it should be done in comparison to other cards.

#### 3.3.1.2.2 New Repositories

When a new component to the Operating System is created, the following is created in a GitHub repository under the project organization account:

- README.md file with a description of the code and outline for building and testing it
- A Travis CI configuration YAML file with instructions for the service to build, test, and push any produced artifacts to the appropriate location
- A license file supplying the official open source license (Apache 2.0)
- If the project is an application, a DockerHub repository for the artifacts
• If the project is an application, a GitHub "+-deploy" repository containing the production deployment configuration code for the running instance
• If the project is a library, a Hex.pm repository for the artifacts

3.3.1.2.3 Configuration

All feature components include a ".travis.yaml" file in the project Git repository's root directory to allow configuration of the necessary build, test, and package deployment steps Travis performs. Any GitHub user account that has administrative permissions granted to the project GitHub organization and has granted Travis the ability to read "organizations" from his/her GitHub profile can configure the repository's Travis settings from the https://travis-ci.org/ site by navigating to the "organization" link from his/her profile and selecting the "settings" button for the specific project repository.

3.3.1.2.4 Testing

The Operating System team follows the Test-Driven Development (TDD) philosophy. This means tests are written before any code is written. Tests are written to assert the expected behavior of the system which drives development of the issue until all behaviors have been implemented and tested. This allows higher development velocity with lower defects and prevents regressions in the system. The test initially fails and continues to fail until the proper software logic is implemented and makes the test pass. This creates testable code and over time ensures a well-tested code base with lower defect rates and higher velocity. Developers can have confidence that their changes donot break other parts of the system and that they are writing only the amount of code needed for the feature they are working on. Test are created and stored in the same GitHub repository as the code. Automated tests consist of both Unit Tests (that test a function or small piece of code) and Integration Tests (that test overall app behavior). All applications have some degree of test coverage spanning both unit and integration tests.

3.3.1.2.5 Writing/Updating the Code

Changes to the source code undergo the following process in a repeating cycle. All code changes follow this process regardless of whether the source of the change is an external contributor or Operating System team member:
1. A branch of the code is created from the primary (master) branch; code is changed or added, including updating relevant code documentation and tests
2. Once changes to code are desired to be merged back to the primary branch, the submitter opens a Pull Request (PR) which begins the process to submit code for review and ultimate integration into the master.
3. Submitter inputs initials in the comments of the PR to signify acceptance of the Contributor License Agreement (CLA).
4. The CI system runs all tests against the changed code automatically to validate quality. Additional non-functional checks such as code coverage, consistent formatting, and language conventions are also applied during this process.
5. At least one Operating System team member reviews the PR and provides feedback, both positive and negative, to ensure the submission contributes positively to the growth of the codebase and adheres to project and language standards. The Operating System team member formally approves the submission via the GitHub interface when a change is accepted and ready to merge into the primary branch.
   a. The Operating System team representative should communicate to an external contributor appreciation for the contribution and at least a general idea of expected timeline for the submission to be included in the next published release.
6. The merged code then receives a Git semantic version tag that is simultaneously applied to its corresponding build artifact for uniquely identifying the specific version of the code.

7. The merged code is then deployed into a production-like staging environment and various tests are run against the changed system to ensure overall quality and correctness have been maintained.

8. If all tests pass and no unexpected behaviors or side effects are detected, the changes are reviewed by the Operating System team leadership and a Change Advisory Board (CAB) is held to determine impact to end users of deploying the new changes to the running production instance.

9. When changes are judged ready for final deployment, a Git tag is applied to the repository storing deployment configuration for the running production instance of the platform and a Jenkins job is run to update the current components with any changed ones.

10. In the event of unforeseen problems, the previous version of the code is re-deployed to replace the unsatisfactory version by running the previous versioned Jenkins job and if the code cannot be fixed immediately by an additional change the Git merge is reverted and the new code is removed from master branch altogether.

3.3.1.3. FEATURE DEPLOYMENT

Platform components providing their specific functionality as part of the Operating System (as distinct from the general platform open source code) are managed via “infrastructure as code” in privately available GitHub repositories. Deployment of the components is orchestrated by a privately hosted Jenkins CI server controlled by the Operating System team within the umbrella of the project’s AWS infrastructure. The Smart Columbus instance of Jenkins is run from within a Docker container on the AWS Elastic Container Service (ECS) with the ability to schedule worker processes, also within containers, on the ECS service. The Jenkins server has a GitHub service account created by the Operating System team that has permissions to access the private GitHub organization for which it performs CI. Credentials for accessing the account are stored within the Jenkins internal credential management store and the server itself is connected to the project LDAP service to grant management capability via a centrally managed identity service.

3.3.1.3.1 Configuration

Project infrastructure is maintained by Terraform configuration code while applications are maintained via Helm YAML files. When changes are needed to infrastructure resources or applications are deployed, the values are changed or added to the files in the private repository.

3.3.1.3.2 Infrastructure

Platform cloud infrastructure is automatically deployed and triggered by GitHub events within the infrastructure repository. When a PR is opened against the infrastructure code repository, a webhook configured in the repository causes the Jenkins server to check-out the code and apply it against the development environment. Assuming expected functionality is validated and the code change is approved and merged into the master branch of the repository, a second automated webhook triggers the same code to be deployed to the staging environment. When functionality has been further vetted and approved, a new GitHub semantic version tag is applied to the repository to signify a release of the code. This triggers the creation of a new job within the Jenkins service which is manually triggered when the Operating System team is prepared to deploy the changes to the production environment. Any rollbacks found necessary can then be performed by manually triggering the release job within Jenkins for the last known-good release version of the code.
The Operating System cloud infrastructure is managed through the same code process as above with the exception of the way in which changes are validated and deployed from one environment to the next due to the difficulty of adequately testing infrastructure changes outside of a live and for-pay provider cloud.

- Changes are made to the Terraform infrastructure code and is needed to validate desired functionality within the Sandbox environment
- Once changes are ready for final validation, a PR is opened and the new code is deployed to the shared development environment
- If no errors or unexpected side effects are observed, the code is merged to the master branch and deployed to the staging environment for final CAB review with Operating System team leadership before being deployed to production
- Following the CAB review, the master branch is tagged with a new Git release tag and a Jenkins job generated from the version tag is run to deploy the changes to production

### 3.3.1.3.3 Features

All platform applications are deployed as containerized services to a Kubernetes cluster in AWS. Helm is used to encode and configure the services. Applications are defined by a series of YAML template files maintained from the project's open source "charts" repository. As the Helm tool allows for runtime override of all configurable values, each application has configurable values managed from a separate repository within the project's closed-source organization. These values are limited to a single YAML file, defining each value to override a default according to Helm best practices.

When a new release of an application is ready to be deployed, or changes to configuration values are needed (such as compute resource limits, number of replicas, etc.), the value is updated in the private "*-deploy" repository by an Operating System team member and a PR is opened for review by another member of the Operating System team.

Changes to the master branch of open source application repositories automatically generate a new version of the application container with the "development" tag, so if no changes are needed to an application configuration, the latest code can always be deployed to the "dev" environment by selecting the "DEV_DEPLOYMENT" checkbox from the application’s deploy repository master build job in Jenkins and clicking the "build" button.

When an application release has been generated and is ready to be deployed beyond the "dev" environment, the deploy repository for the application is updated to reflect the new release tag version. A PR is opened and reviewed by the Operating System team and once approved and merged, a trigger configured in GitHub responds to changes to the code for that repository which causes Jenkins to automatically checkout the new code and deploy it to the staging environment.

Once a new version has been validated in the staging environment and the Operating System team is prepared to deploy to production, a new semantic version tag is created on the project’s closed-source deploy repository which automatically creates a job in Jenkins with that tag as the name. This job can be run at any time to deploy the changes to the production environment and should a rollback be necessary, the same process can be performed for the last known-good semantically versioned release build. When deploying to production the following is expected:

- Most deployments should not cause downtime and are done at developer convenience (once reviewed and assessed for risk)
- Downtime deployments are scheduled for minimum customer impact
- Features are not considered complete until they are demonstrated as working in production

### 3.3.1.4 RELEASE ARTIFACTS

The project produces the following build artifacts which are used to deploy the running instance of the Operating System as well as allow other adopters to deploy versions of the platform.
• Hex packages – The format of the compiled Elixir libraries, stored in, and pulled from the managed third party Hex.pm service by notations in an Elixir project mix.exs file.

• Docker containers – The format of platform microservice applications, stored in, and pulled from the managed third party DockerHub services by command line instructions or references in Kubernetes manifest or template files.

• Helm charts – A collection of Kubernetes manifest files detailing the form and configuration of the running system intended for deployment as declarative instructions. Bundled via shell script and published as compressed, zipped archives stored in the template file repo within the “docs” subdirectory and accessible to the Helm tool over https.

• Configuration repos – Runtime configuration files with specific production overrides to default values, including version numbers of applications intended for production deployment, as well as Jenkins CI files for performing deployment operations and any Terraform infrastructure-as-code instructions for producing cloud provider infrastructure resources as needed.

3.3.2. Open Source Code Management

The open source code management process allows the community of developers to extend the Operating System codebase as well as deploy and manage their own running instance of the platform.

3.3.2.1. CONTINUOUS INTEGRATION (CI)

The components of the Operating System are open source and as such, development of platform features are available to any developer from the general public willing to contribute. To enable proper feedback on submitted feature enhancements or bug fixes, the platform Operating System team uses Travis CI as the CI service for building and testing all applications because it is freely available for open source projects and is publicly available without placing the burden of maintenance or security of the CI platform on the Operating System team.

3.3.2.2. RELEASE ARTIFACTS

All release artifacts listed above in Section 3.3.1.4 except for the Columbus implementation runtime configuration/deployment code is available for others to build and test their own version of the platform.

3.3.3. Third Party Software

For the purposes of this document, third party software is defined as software or software services that are entirely managed and maintained by external providers and/or maintenance of the software code is a for-pay service by an external provider. The following are a list of third party software tools and products that are used by or interface with the Operating System in some way and the extent of their integration.

3.3.3.1. GITHUB

GitHub, a web-based Git repository hosting service, was chosen to host the open source applications on public repositories. GitHub provides Operating System contributors a web-based interface and allows them access to source code with collaborative software development features such as Wiki, bug tracking, code review, etc. GitHub provides a flexible software workspace management infrastructure, ideal for open source software release staging, and an area for vetting source code before releasing them on the Operating System. Source Code management for all project microservices and configuration/deployment code used to deploy and manage the platform:
- Provides private access to organization member accounts of proprietary, Columbus-specific code and configuration (the SmartColumbusOS organization repositories).
- Provides public access to view and fork the code in public repos (the smartcitiesdata organization repositories).
- Allows controlled dialogue and exchange of ideas between maintaining Operating System team and community.
- Maintaining Operating System team approves or disapproves submissions of code from external contributors to ensure standards and code quality.
- Tracks issues (feature requests or bugs) reported against repositories.
- Provides wiki information for persistent Operating System team FAQs and collected wisdom.

3.3.3.2. ZENHUB
Aggregates GitHub issues (feature requests and bugfixes) across all of the open and closed source Git repositories allowing for single unified view and prioritization.
- Free for open source repositories, paid for closed source repositories (see Chapter 2. Materials and Resources section for cost details)
- Uses existing GitHub accounts for authentication and authorization

3.3.3.3. GOOGLE ANALYTICS
Provides tracking of web interaction with the Operating System via tracking tags or cookies embedded in the Operating System user experience.

3.3.3.4. DOCKERHUB
Provides artifact hosting of published microservice containers for deployment.
- Requires dedicated account created with Dockerhub
- Organization provides unified management of published artifacts delegated to member accounts
- Free for open source projects

3.3.3.5. TRAVIS CI
Provides build, test, and artifact publishing of project source code.
- Free for open source projects
- Currently not being utilized for any closed source projects or deployment to running environments

3.3.3.6. JOOMLA
Provides a uniform web interface for content updates and centrally styling a web service, targeted at non-developers for non-functional updates of site content.
- Theme software is customized and developed by Dark Rhino (formerly Proteon Software)
- Several integrations or plugins to the open source Joomla platform have been developed and can be updated by Dark Rhino as a for-pay service
3.3.3.7. HEX.PM

Provides artifact hosting of published source code libraries.
- Free for open source projects
- Requires dedicated accounts attached to central organization that delegates management privileges

3.3.3.8. SLACK

Provides internal Operating System team, synchronous chat communication.
- Free plan with 10,000 message rolling limits; not intended to be a long-term reference to information
- Provides third-party integration with other services for event notifications such as GitHub project builds or monitoring service warnings

3.3.3.9. GITTER

Provides open source community group chat space.
- Free and open
- Uses existing GitHub accounts for identity management

3.4. SECURITY MEASURES

3.4.1. Security

The Data Management and Privacy Plans (links to plans can be found in Appendix B: Related Document Links) outline the security controls for the Operating System. The Operating System has implemented patch management and SDLC processes for ensuring secure systems. These processes allow many security concerns to be handled via automation and code, with processes for manual interventions.

3.4.1.1. JOOMLA

Since Joomla is one of the main user access points of the Operating System, it is important that this remains up to date and secured. The below measures are key activities to ensure that Joomla is able to maintain its secure state.

3.4.1.1.1 Joomla Patch Updates

Joomla software updates are designed to counteract potential security risks affecting the Operating System operations. Typically, Joomla displays an alert message when an installed Joomla software component or extension has an update. The system administrator simply follows the instructions provided to upgrade the software to the latest revision.

3.4.1.1.2 reCAPTCHA

The reCAPTCHA is a software component of Joomla system designed to discern whether the website user is human or a “bot,” an automated, programmatically driven attempt to register in the system. reCAPTCHA prevents the machine-generated bot from submitting requests to the Operating System which could be malicious.
3.4.1.3 Multi Factor Authentication

Access to Joomla administration is controlled by multi-factor of authentication. For a user to access the administration interface they must have both their password and a One Time Password (OTP) generated by Google Authenticator. This ensures that a compromised password does not alone allow an unauthorized user to access the system as all authentication requires both the password and OTP which can only be generated by a device the authorized user owns.

3.4.1.2 AWS

AWS is the hosting provider for all of the Operating System applications and as such must be properly secured. The following provisions are taken to ensure security on AWS resources

3.4.1.2.1 Multi-Factor Authentication

Like Joomla, all AWS access must be acquired via multi-factors of authentication. This ensures that AWS access is not granted to those without proper authorization to access the environment.

3.4.1.2.2 Automatic Updates

Where possible the Operating System services are configured to apply automatic updates overnight to ensure the most up to date services are running. This ensures that security patches are properly applied to running services.

3.4.1.2.3 Manual Updates

Some pieces of infrastructure cannot be updated automatically. One example of this is EKS which must have its updates initiated manually. The manual updates are triggered through Slack security bulletins that notify the Operating System team that an update is needed.

3.4.1.2.4 Security Bulletins

The Operating System team members are subscribed (via Slack) to AWS security bulletins so that issues which are not automatically patched are announced to the team so that their resolution can be prioritized.

3.4.1.2.5 Code Review

The first line of defense against security issues arising in application code is code review. All code must be reviewed by another team member who did not complete the work to allow this work to be integrated into release candidates for production. Code review allows for developers to look for vulnerabilities which are introduced by changes to the codebase before it is ever merged with the existing codebase.

3.4.1.2.6 Static Analysis

Software which is customer facing has automated static analysis. Static Analysis provides a way to analyze code before it is deployed to a production server for known issues which can cause security or performance problems. The Operating System uses the Sobelow Elixir library to perform this analysis and this is packaged directly with the pieces of the Operating System which must be statically analyzed.

3.4.1.2.7 Dynamic Analysis

Dynamic Analysis is defined as scanning running applications for vulnerabilities. The Operating System uses OWASP Zed Attack Proxy to scan running applications which are user facing to identify any
vulnerabilities which are not discovered by static analysis. Issues which are discovered during this analysis are prioritized and fixed through the standard SDLC.

### 3.4.1.2.8 Library Updates

The Operating System relies on many third-party libraries listed in Section 2.2.1 which must be kept up to date to ensure minimal defects and security vulnerabilities exist in the Operating System applications. The Operating System applications automatically scan their dependencies as part of the CI process to ensure they are up to date and provide developers with feedback to update out of date dependencies.

### 3.4.2. User Management

The Operating System has two system classifications that are used to control access to the data and the system for administration. This section describes the actions performed for each classification. The system classifications:

- **System Administrators** – People who support the infrastructure and code that operates the Operating System.
- **End Users** – People that interact with the Operating System via a website or Application Programming Interface (API).

#### 3.4.2.1 SYSTEM ADMINISTRATORS

People who support the infrastructure and code that operates the Operating System are classified as System Administrators. System Administrators are responsible for ensuring the correct people have access and people who no longer need access are removed. This is reviewed on a quarterly basis.

##### 3.4.2.1.1 Adding Users

Users need to be added as an IAM user in the Application Lifecycle Management (ALM) AWS Account. The following is used when adding users:

- A separate account is created in the production AWS account for users that need to edit the production system.
- Users must enable Multi-Factor Authentication (MFA).
- IAM provides a reminder that prompts the user when their credentials have aged out.
- Access Keys and Secret Keys are rotated every 90 days (IAM provides a reminder that prompts the users when their credentials have aged out).
- Users are added to LDAP ([https://iam-master.alm.internal.smartcolumbusos.com](https://iam-master.alm.internal.smartcolumbusos.com))
- Users are added to the following groups in LDAP
  - vpnusers
  - scosdevs
  - ipausers
  - admin
  - user
3.4.2.1.2 Removing Users

When someone leaves the team or no longer need administrative access to the system, the user is removed from all of the above listed accounts.

3.4.2.2. RESTRICTED DATA USER ACCESS

Users who require access to restricted data sets need to be granted access by the System Administrator. The System Administrator is also responsible for removing access if no longer needed. Users needing this level of access are identified as part of the dataset curation process.

3.4.2.2.1 Adding Users

- Users are added to LDAP (https://iam-master.alm.internal.smartcolumbusos.com)
  - Users are added to the external_users group in LDAP
  - Users need to be added as a member of their associated organization in order to be able to access restricted datasets.

![LDAP Browser Example Access](image)

*Source: City of Columbus*

3.4.2.2.2 Removing Users

When someone no longer needs access to the restricted data, there are two typical scenarios based on access need.

- Retain access to Organization but remove access from dataset – remove user from organization’s members in LDAP.
- Remove all access to data and system – remove user from LDAP to remove from the system.
3.5. DATA AND CODE USE

3.5.1. Privacy Management

The Operating System is aligned to the guidance of both the Data Management Plan (DMP) and the Data Privacy Plan (DPP). The DPP and DMP set out high-level privacy protections and oversight governing the Operating System on how to protect the privacy of users and participants, and guard against potential breaches. The data curation, ingestion, and management processes are designed as critical controls for privacy management to ensure proper handling of data before it is added to the Operating System. The Operating System is built to provide access to public data sets and PII is not ingested during the curation process. All data is de-identified per the De-Identification Policy prior to data ingestion. Links to these documents are included in Appendix B.

3.5.1.1. DATA CURATION, INGESTION, AND MANAGEMENT

Data curation process is designed to ensure that the data is verified for technical, ethical, and privacy issues. Curation process is completed by the Data Curator who has standard team member access, see Section 3.4.2 above for more details. Data consumer and data provider mutually agree to a data sharing agreement (also known as a Service Level Agreement (SLA)). This SLA outlines the responsibilities of both parties. Following the SLA, the Data curator collects metadata (technical, operational, and business) and a data sample from the data provider. Both resources are used to assess the consumability, usability, and privacy of the data. During this assessment, benefits of ingesting the data should always outweigh the risks. Once the data curation is complete, data is sent for ingestion.

Data ingestion includes setting up data schema, metadata collected during the curation process and ingestion of the data. During ingestion, quality control includes validating records against the defined schema to flag unusual entries. Should outliers, missing, or otherwise anomalous entries be found, the data providers are contacted for verification. Resolution of the flagged data may range from entry of a null value to rejection of the dataset, depending on the established SLA. Once validated, the data is moved to the production website where the data is made available to users of the Operating System.

Data management includes updating metadata fields, ingesting an updated version of a dataset, and/or deleting a dataset. Updates to data and metadata are performed based on the inputs from the data provider. Deletion of datasets would be determined based on the relevance, usefulness, and quality of the data. In some instances, lack of alignment with SLAs might lead to removal of dataset(s) as well. Any updates to the data including metadata changes, dataset version changes, and data removal are handled in Data Management.

Figure 10: Data Curation Process

Source: City of Columbus
3.5.2. Open Source Community Management

As an open source software project, the Operating System is freely available for external adopters and contributors. In order to maintain the resource of continuous feedback and contribution to a shared, open codebase, the following practices are important to foster a healthy community around the code.

3.5.2.1. ISSUES

All desired features and bugs should be created in GitHub as issues attached to a specific repository based on relevance to a given part of the codebase, or to the general project repository if they are too broad to apply to a specific component. Tracking issues directly in GitHub and prioritizing from the integrated ZenHub application allows the larger community to follow the progress and prioritization of the Operating System team and coordinate their own efforts accordingly.

3.5.2.2. DOCUMENTATION

Documentation is a critical tool in open source for attracting and engaging adopters and contributors to the platform by making it easy to understand how a project works and how to interact with it. Documentation is as actively maintained as the source code itself in the following places:

- Code comments – denoted by both simple “code comment” characters at the beginning of comment lines within the code as well as conventions specific to the Elixir language of documentation attributes (@moduledoc, @doc) followed by comment blocks featuring explanations and code samples.
- READMEs – within the root of every Git project and describing the purpose of the repository and either instructions for setup, testing, and use of the code or directions to more extensive documentation providing the same. Written in markdown.
- Wiki – attached to the primary project repository to centralize information and reduce duplication of documentation. Written in markdown and linking accordingly to the respective repositories, tutorials, additional resources, or embedding diagrams and visual aids as needed. The wiki is the primary location for providing instructions to prospective new contributors on how to setup a development environment to support the project, where to locate issues, and how to contact the Operating System team for further clarification of questions.

3.5.2.3. DIAGRAMS

Accurate and current architecture diagrams are included in the central project (https://github.com/smartcitiesdata/smartcitiesdata) README and outlined as needed in more detail within the wiki. Diagrams afford the community easier understanding of the distributed architecture and where to engage with the code by function or associated technology.

3.5.2.4. PULL REQUEST (PR) FEEDBACK

Contributors require feedback on their PR to engage actively with an open source project and feel their contribution is appreciated. As PRs are opened by external contributors, Operating System team should:

- Thank the contributor for their submission
- Ask about the nature of the PR if it is not immediately provided with the contribution
- The result of building and testing the PR should be automatically reflected in the GitHub UI for the contributor to reference if there are problems they need to fix.
- If the PR cannot be accepted, provide a thoughtful explanation as to the reasoning for the rejection and recommended steps to remediate
• Outline to the contributor roughly when the accepted PR is released and deployed

3.5.2.5. RELEASE NOTES

When release artifacts are generated, these are outlined in the accompanying text field in GitHub as a brief description and bulleted list to tell the community what features and potential side effects they are accepting by deploying a specific release of the software.

3.5.2.6. ROADMAP

The roadmap is an enumeration of desired features (functional and non-functional) and their relative priority order for implementation by the community. It exists as a document within the project wiki on GitHub for the community to reference and influence based on the changing needs within the community.

3.5.2.7. STANDUP

The community “standup” update call is an event each month where the Operating System team outlines the progress of the project in an open video conference call with members of the community and allows for community members to call out their own work and dialogue about the future of the project. This is currently a virtual meeting that is scheduled each month and emailed to the current “mailing list” for the community but is planned to be migrated to a link hosted and publicized from the project wiki or website to be inclusive of a wider audience.

3.5.2.8. MODERATING

In the event of belligerent or abusive exchange between members of the community, it is the responsibility of the Operating System team to:

• Remind offenders of the community standards and expectations for behavior, directing them to the Code of Conduct documented within the central wiki

• Block repeat offenders within the GitHub interface and report to GitHub for further remediation as needed

3.5.2.9. CONTRIBUTOR LICENSE AGREEMENT (CLA)

The Operating System team is responsible for managing a process by which external contributors agree to a CLA when submitting to the project that absolves the project of any financial or legal responsibility to the contributor in exchange for their submission. This can be done automatically via bot integration with GitHub that enforces an e-signature process on every open PR via the GitHub comments feature of the PR dialogue or by deploying and managing a long-running web service for collecting and storing CLA approvals from contributors on a one-time basis provided the language of the CLA clearly expresses the scope of the agreement to any and all future contributions by the submitting user within the agreed upon timeframe.
Chapter 4. Maintenance

4.1. MAINTENANCE OVERVIEW
Ongoing maintenance of the Operating System is needed to ensure proactive monitoring, back up of systems, and processes established to keep content current. The maintenance functions are Monitoring, Preventive, and Content Management.

![Maintenance Diagram]

Figure 11: Smart Columbus Operating System Maintenance Functions
Source: City of Columbus

4.2. MONITORING

4.2.1. Security Monitoring

4.2.1.1. LOGGING
The Operating System applications produce logs which create an audit trail of events that happen in the applications. These logs are aggregated by Fluentd, indexed and stored within ElasticSearch, and made searchable by Kibana. When there is evidence or alerting that an expected event has occurred in the system, a developer or administrator uses these tools to trace this event and plan remediation.
4.2.1.2. **AWS CLOUDWATCH**

The Operating System utilizes AWS CloudWatch to monitor running applications in the AWS cloud. This allows for near real-time alerting of unexpected events in the system. These events can be configured to send emails or post to a Slack instance allowing developers and stakeholders to be notified as soon as possible.

4.2.1.3. **AWS AUDIT TRAIL**

AWS provides an audit trail for many activities which happen in the system. When a resource is modified or changed, developers and administrators can check this audit trail to understand what has been modified.

4.2.1.4. **SECURITY BULLETIN SUBSCRIPTION**

Many software and service vendors like AWS provide an RSS or similar feed to subscribe to for bulletins on updates or security patches. The Operating System subscribes to these feeds which are monitored by developers to allow them to create and prioritize work items. Subscribing to these bulletins reduces the cost and onus on keeping up to date with the many different components underlying the Operating System.

4.2.1.5. **DYNAMIC APPLICATION SCANNING**

As covered in Section 3.4 above, dynamic application scans using OWASP Zed Attack Proxy (www.zaproxy.org) are performed nightly against public portions of the Operating System. This allows new vulnerabilities which arise from server or application changes to be detected as soon as possible. Additionally, these scans provide an audit log of system security over time.

4.2.1.6. **PERFORMANCE MONITORING**

Metrics are recorded in real time for the internal data pipeline, the user facing applications, as well as the infrastructure. These metrics are configurable and viewable using the internal-facing Grafana dashboards.

4.2.1.7. **KAFKA PERFORMANCE**

The Kafka dashboards provide insight into the performance of the data pipeline (Apache Kafka) using metrics such as messages per second and lag per consumer group.
4.2.1.8. USER-FACING APPLICATIONS

The user-facing dashboards provide insight into the rates at which the external API endpoints are called, as well as the average call durations.

Figure 12: Kafka Performance Metrics

Source: City of Columbus

Figure 13: API Performance and Load Metrics

Source: City of Columbus
4.2.1.9. INFRASTRUCTURE

The infrastructure dashboards show the internal performance of the infrastructure within Kubernetes. This includes a by-node-breakdown of memory usage, disk space, idle CPU, disk writes, and network activity. The values reported by the dashboard indicate the relative health of the system and infer resource consumption for long-term planning of operations like increasing or decreasing system resources.

![Kubernetes Cluster Metrics]

Figure 14: Kubernetes Cluster Metrics
Source: City of Columbus

4.3. PREVENTATIVE MAINTENANCE

The Smart Columbus Operating System’s operations are 24 hours, 7 days a week. Maintenance activities are necessary to keep the services current with new contents, e.g., new applications, datasets, data stories, and related project updates. These maintenance activities occur Monday through Friday from 9 am to 5 pm Eastern Time.

This section outlines various maintenance tasks, including preventive and corrective maintenance activities and other adjustments as needed.
4.3.1. Preventive Maintenance Tasks

On a regular basis, the Operating System team perform preventive maintenance tasks in order to keep the system operating optimally. The activities in Table 7 take place on determined frequency or by triggering events.

Table 7: Preventive Maintenance Activities List

<table>
<thead>
<tr>
<th>ACTIVITIES</th>
<th>DESCRIPTION/COMMON TASKS</th>
<th>FREQUENCY</th>
<th>EFFORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>User management</td>
<td>Checking user registration notification for spammers and removing unrelated users</td>
<td>Daily</td>
<td>0.25 hrs.</td>
</tr>
<tr>
<td>Website backup</td>
<td>Backup is automated, however, the system administrator needs to check the backup status to ensure a complete backup is stored properly. Checking performance statistics.</td>
<td>Weekly</td>
<td>0.25 hrs.</td>
</tr>
<tr>
<td>Website sweep</td>
<td>Review various website areas to ensure the information is presented properly to visitors and registered users. Additionally, triggered by request from program managers</td>
<td>Daily / On demand</td>
<td>0.25 hrs.</td>
</tr>
<tr>
<td>Security checking</td>
<td>Checking file/folder permission and reviewing log files for indication of errors or tampering. Review IAM policies to ensure least-privileged access. Additionally, triggered when team composition changes</td>
<td>Daily / On demand</td>
<td>0.25 hrs.</td>
</tr>
<tr>
<td>Respond to “Contact Us” inquiries</td>
<td>Triggered by any inquiries related to services and procedures in the scope of this document and deployed services.</td>
<td>On demand</td>
<td>0.25 hrs.</td>
</tr>
<tr>
<td>AWS Server / Services monitoring</td>
<td>Checking server performance, security conditions, resource levels, etc. Review CloudWatch logs, AWS Audit results, review AWS Config, Guard Duty, and other AWS services that provide situational awareness of the access and management of operational services and deployed infrastructure.</td>
<td>Daily</td>
<td>0.5 hrs.</td>
</tr>
<tr>
<td>Download log saving</td>
<td>Copying the open source download log to offline storage.</td>
<td>Monthly</td>
<td>0.25 hrs.</td>
</tr>
<tr>
<td>S3 backup buckets cleaning</td>
<td>Checking for the status of backups. Removing old backup files. Recommendations on cost efficient storage solutions based on required availability, durability, and access requirements in accordance with the data lifecycle policies.</td>
<td>Quarterly</td>
<td>1 hr.</td>
</tr>
</tbody>
</table>
### Chapter 4. Maintenance

#### 4.3.2. Server Administration

All Operating System servers are virtualized within the AWS cloud portfolio of services. This means maintenance and management of servers is minimal and can be done either via the AWS web console or the AWS-provided command line utility. Individual virtual servers provided by Amazon’s EC2 product (including worker nodes for Kubernetes) must have their pre-built machine image updated periodically to incorporate the latest patches and updates to their Linux operating systems and core packages. Managed services such as RDS databases, ElastiCache services, and the managed Kubernetes controller must periodically have their versions updated. These changes should be tracked and deployed via the project's Terraform infrastructure code but can be done via the Amazon web console in the event an emergency hotfix is needed.

The majority of server administration is performed via EC2. S3 is configured with storage buckets and is only administrated as necessary. Likewise, RDS and EKS are managed services and do not require maintenance frequently. Listed are EC2 activities performed only as documented below:

- Cloning the production server image to create a development server instance when a development service is desired for testing.
- Managing data volumes (e.g. increase and decrease storage capacity) when monitoring indicates storage volumes are underutilized or nearing capacity.
- Creating snapshots of the server for quick replication (preventive backups).
- Configuring Network Security groups to control network access (when new network services are deployed).

Development servers are typically stopped to conserve costs as AWS services are charged by usage by the hour. Development is done on local machines. The “Development server” in this case is used for testing changes made to the site. This allows the conservation of costs by only using the development server when testing changes and turning it off when testing is complete. The rate to operate the Operating System is the standard cost associated with cloud instances and AWS schedule for those instances and managed services.

Production servers are enabled and continuously operational while the Operating System website is active. Figure 15: Amazon Web Services Dashboard displays the EC2 dashboard where common resources can be manipulated.

---

**ACTIVITIES** | **DESCRIPTION/COMMON TASKS** | **FREQUENCY** | **EFFORT**
---|---|---|---
AWS server image maintenance | Creating new server instances of new images. Making a clone of the AWS server image for potential quick recovery. Installing Joomla, Operating System customization, database, etc. Testing to ensure operational integrity. | Monthly | 3 hrs.

Add new dataset | The amount of effort and time associated with adding a new dataset to the system can vary greatly. If the dataset being ingested is similar to other datasets the effort and time should be minimal. As the Operating System has grown we have encountered new use cases (mostly based on how the data is access from the original source and/or the structure of the data) that have required code changes to be able to support new features. | On Demand | 4 hrs.

*Source: City of Columbus*
4.3.2.1. IDENTIFICATION AND ACCESS MANAGEMENT FOR SERVER ADMINISTRATION

Amazon Web Services is used for infrastructure for the Operating System; however, these guidelines can be followed for any cloud vendor. Server Administrators and developers are created and managed through AWS IAM service and accounts are managed through an organizational unit (OU) management scheme. Users are created and maintained in the Application Lifecycle Management (ALM) account and then roles are assumed to access the other AWS sub-accounts. The production sub-account only allows users to assume read-only roles from their ALM account. Users must be created and given access to manage production from the production account.

For more context on AWS IAM and OU visit:

- IAM: [https://aws.amazon.com/iam/](https://aws.amazon.com/iam/)
- OU: [https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_ous.html](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_ous.html)

4.3.3. Website Data Backup

The Operating System website contents are backed up on a daily basis to AWS S3, a remote cloud-based storage service on the AWS cloud. In case of emergency, e.g., the physical server is completely destroyed by a natural disaster, the website can be recreated and restored to its last daily backup point.
Figure 16 shows the graphic user interface of the Akeeba Backup tool. Retention for S3, if not stated, is indefinite or defined by the Data Management Plan.

4.3.3.1. WEBSITE BACKUP

All Joomla data along with configuration are backed up every night at 11 p.m. Eastern Time, controlled by a Linux scheduler **crontab**. A Joomla extension called Akeeba Backup PRO, as shown in Figure 16 is used for the backup automation. All webpages and the database are backed up to S3 storage. The entire Operating System website can be restored to another server within one hour using the Akeeba restoring procedure.

![Figure 16: Akeeba Backup Interface](source: City of Columbus)

4.3.3.2. OPEN SOURCE APPLICATIONS BACKUP

Open source application packages (zip files) are stored in a different data volume on the AWS cloud to segregate them from the web server to avoid spreading any corrupted files. Open source code does not change often and any incremental changes get typically backed up daily to remote online storage. Restoration of the open source data files are performed manually by copying the files from remote storage to the Operating System via a script.

4.3.3.3. OPEN SOURCE APPLICATIONS DATA BACKUP

Data that is generated within the applications running on the Operating System is stored in private S3 buckets via Presto/Hive during processing by the system and can be restored from there. Data that is in a state of processing; i.e. a dataset that is being ingested into the data platform or operating system processing metrics, is not backed up. This is usually a small amount of data, and if the data source is reachable, then can be recovered by reprocessing the dataset. Data restoration from S3 back to
Chapter 4. Maintenance

Presto/Hive is a time-consuming process. If this is a planned recovery, additional time will be needed to restore the data.

4.3.3.4. RELATIONAL DATABASES

Relational Databases, namely, Postgres are maintained via AWS RDS. See Figure 17 for RDS interface in AWS. Databases can also be interacted with using the AWS command line interface. Databases are backed up daily (01:25-02:25 UTC (GMT)) automatically by AWS. Since databases are encrypted at rest, they are also backed up in an encrypted state. Database backups that are captured automatically are retained for 14 days. During some operations a manual backup of the database is taken as an additional preventative measure and is part of the rollback plan. These backups are retained indefinitely or as defined by the rollback plan for that operation. RDS is backed up to S3 and would be restored from there. In case of a disaster scenario, the Operating System website and its data can be restored and deployed to any Linux server within a few hours after notification.

Figure 17: AWS RDS Interface
Source: City of Columbus

4.3.3.5. OTHER DATABASES

Redis is an in-memory cache database offered as part of AWS ElastiCache service. The Operating System uses it for fast storage of system state. It is backed up daily as part of the service offering.

4.4. CONTENT MANAGEMENT

4.4.1. Content Updates

Content Update activities are limited in scope to updates to Joomla Content. Table 8: Update Activities highlights activities, their frequencies and their efforts.

Table 8: Update Activities

<table>
<thead>
<tr>
<th>ACTIVITIES</th>
<th>DESCRIPTION/COMMON TASKS</th>
<th>FREQUENCY</th>
<th>EFFORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub repository</td>
<td>Content updates are archived in GitHub to provide content as code.</td>
<td>As needed</td>
<td>1 hr.</td>
</tr>
</tbody>
</table>
## 4.5. PROJECT MONITORING

The Smart Columbus Operating System’s performance will be measured based on the hypothesis, metrics, and data identified in the Performance Measurement Plan (PIMP). Key stakeholders in the development of PIMP include the Smart Columbus project team and Independent Evaluators (IEs) selected by the USDOT. For the Operating System, both quantitative and qualitative data will be analyzed to measure its performance. Quantitative data includes data requests, data downloads, unique vs returning users, etc. This data is collected using analytics tools like Google Analytics. Qualitative data includes user feedback on their ability to feature data on the Operating System, discover and use the data, and resourcefulness of the data available for the intended purpose, etc. This data is collected through user surveys posted on the Operating System’s site. Baseline and post-implementation data required for PIMP evaluation will be collected and stored in the Operating System. The Smart Columbus evaluation team will conduct the performance measurements at the end of the project period using the project data available on the Operating System. IEs will access the project data only through the Secure Data Commons platform. Secure Data Commons (SDC) is a Collaborative Transportation Research and Analytics Platform designed and built by the USDOT to enable transportation related data analytics. The SDC platform ingests the evaluation data for the Smart Columbus program from the Smart Columbus Operating System and other sources. Data will be transferred using the API created for individual datasets. Once data is available, IEs will be able to login to their accounts on the SDC, access the data, perform analysis on the SDC platform using the available integrated tools, and download the results. To download the evaluation data from the SDC for analysis using on premise tools, the IEs need to request access from the source data providers. Performance measurement results will be documented and submitted to the USDOT at the end of the period of performance of the Smart Columbus program.
Chapter 5. Ownership Transition Activities

This section is intended to define the basic setups that are anticipated based on the current state of the Operating System. The Program Manager is the ultimate owner of all Operating System accounts. The ownership is managed through a cascading approach. For example, the Program Manager owns the AWS account which contains the AWS Secrets Manager. The AWS Secrets Manager allows the System Administrators to manage accounts like the VPN Server Account and DockerHub Service Account.

A complete list of accounts and how they are managed are included in Appendix A. In general, transition activities for accounts begin with creating a new account, giving access, and ultimately removing access. The following outlines the overall technical activities expected to transition ownership of the Operating System. The transfer of knowledge is best to happen through pairing in with the existing team on the operations and maintenance over an estimated period of at least one month.

Figure 18: Transition Flow

*Source: City of Columbus*

- Create development environment
  - Setup a development environment (See Section 2.2.3.2 Hardware for suggested specifications)
  - Ensure AWS command line interface is configured
  - Create GitHub account and be added as an owner to SmartColumbusOS organization
  - Use [http://GitHub.com/smartcolumbusos/team-configs](http://GitHub.com/smartcolumbusos/team-configs) repo for further pairing station setup (requires GitHub access)
  - Install VPN client
- Create new user credentials
  - Create AWS accounts for people of future group
  - Create LDAP account and add to Administration Group
  - Setup Kubernetes configurations
  - Train on the locations of accounts, passwords, and secrets accessed in AWS
  - Grant access to VPN through LDAP
  - Create DockerHub account and associate to SmartColumbusOS
  - Create Joomla account and elevate their privileges
  - Send Slack invite to user-supplied email
Chapter 5. Ownership Transition Activities

- Transfer Ownership
  - Transfer Secure SHell (SSH) keys and revise keys in Jenkins
  - Transfer email distribution list

- Transfer Billing
  - OpenVPN
  - GitHub
  - ZenHub
  - AWS

- Remove Access
  - Delete old team user accounts in AWS
  - Delete old team user accounts in LDAP
  - Remove old team users from GitHub organization
  - Remove old team users from Slack account
  - Remove old team users from Dockerhub organization
  - Remove old team users from Joomla
  - Rotate SSH keys issued to existing server instance
Chapter 6. Closing

The O&M plan defines the foundation of skills, processes, and technologies for operating and maintaining the Operating System. As the Operating System evolves, the O&M plan will also be revised every 6 months to reflect any changes in technologies and processes. The information in this plan will serve as a foundation to define the transition process to any new maintainer of the Operating System.
Appendix A. Account Access Information

This section lists out the information needed to maintain or transfer all of the accounts associated with Smart Columbus Operating System. All Operating System accounts are owned by the Program Manager through a cascading approach where the Program Manager owns the account to the main systems that are used to manage the sub accounts.

Table 9: Accounts Directly Owned by Program Manager

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>&lt;team member username&gt;</td>
<td>Individually assigned by any other members with administrative credentials</td>
<td>Username and password</td>
<td>Managed from the AWS IAM web console by any team member with administrative access (AWS IAM &gt; ALM account &gt; Global region)</td>
</tr>
<tr>
<td>LDAP</td>
<td>&lt;team member username&gt;</td>
<td>Central identity management for internal resources by FreeIPA LDAP server</td>
<td>Username and password</td>
<td>Managed from the FreeIPA LDAP service by any team member with administrative access</td>
</tr>
<tr>
<td>Slack</td>
<td>&lt;team member username&gt;</td>
<td>Real-time project coordination and collaboration chat for team members</td>
<td>Username and password</td>
<td>Created by the individual on invitation by existing team member with administrative credentials</td>
</tr>
</tbody>
</table>

Source: City of Columbus

Table 10: Accounts Managed Through AWS (Secrets Manager)

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>jenkins_user</td>
<td>Service account for Jenkins server to deploy AWS resources</td>
<td>Access key and secret</td>
<td>Managed from AWS IAM web console by any team member with administrative access (AWS IAM &gt; ALM account &gt; Ohio region)</td>
</tr>
<tr>
<td>VPN server</td>
<td>openvpn_ad</td>
<td>Management of the secure VPN service</td>
<td>Password</td>
<td>AWS Secrets manager &gt; ALM account &gt; Ohio region</td>
</tr>
<tr>
<td>Dockerhub Service Account</td>
<td>scosaccount</td>
<td>Service account allowing Travis CI builds to push successful artifacts to the repository automatically</td>
<td>Username and password</td>
<td>AWS Secrets Manager service &gt; ALM account &gt; Ohio region &gt; dockerhub_scosaccount</td>
</tr>
</tbody>
</table>
## Appendix A. Account Access Information

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gmail</td>
<td><a href="mailto:developers.smartcolumbusos@gmail.com">developers.smartcolumbusos@gmail.com</a></td>
<td>Email account for assigning ownership of other services to the project rather than an individual</td>
<td>Username and password 2-factor registration token</td>
<td>Managed in AWS IAM web console by any team member with administrative access (AWS IAM &gt; ALM account &gt; Ohio region)</td>
</tr>
<tr>
<td>Hex</td>
<td>public-hex-pm-account</td>
<td>Programmatic access to the project Elixir package hosting service for pushing new build artifacts by the CI service</td>
<td>API-token</td>
<td>AWS Secrets Manager &gt; ALM account &gt; Ohio region &gt; public-hex-pm-account</td>
</tr>
<tr>
<td>AWS</td>
<td>{dev,staging, prod}-odo</td>
<td>Access to AWS resources (S3 files) for platform application component</td>
<td>Access key and secret</td>
<td>Created in each AWS environment subaccount by any team member with administrative access; stored in the vault secret storage tool for that environment. (AWS IAM &gt; environment subaccount &gt; Global region)</td>
</tr>
<tr>
<td>AWS</td>
<td>{dev,staging, prod}-reaper</td>
<td>Access to AWS resources (S3 files) for platform application component</td>
<td>Access key and secret</td>
<td>Created in each AWS environment subaccount by any team member with administrative access; stored in the vault secret storage tool for that environment. (AWS IAM &gt; environment subaccount &gt; Global region)</td>
</tr>
<tr>
<td>AWS</td>
<td>{dev,staging, prod}-discovery-api</td>
<td>Access to AWS resources (S3 files) for platform application component</td>
<td>Access key and secret</td>
<td>Created in each AWS environment subaccount by any team member with administrative access; stored in the vault secret storage tool for that environment. (AWS IAM &gt; environment subaccount &gt; Global region)</td>
</tr>
<tr>
<td>AWS</td>
<td>scos_{dev, staging, prod}_joomla_access</td>
<td>Access to AWS resources (S3) for environment Joomla instance to write backup file</td>
<td>Access key and secret</td>
<td>Created in each AWS environment subaccount by any team member with administrative access (AWS IAM &gt; environment subaccount &gt; Global region)</td>
</tr>
<tr>
<td>AWS RDS instance</td>
<td>{dev,staging, prod}-lime-db-password</td>
<td>Administrative access to the environment’s Lime Survey Postgres instance</td>
<td>Password</td>
<td>Created in each AWS environment via Terraform, stored in each environment Secrets Manager (AWS Secrets Manager &gt; environment subaccount &gt; Oregon region)</td>
</tr>
</tbody>
</table>
### Appendix A. Account Access Information

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lime Survey server</td>
<td>lime-web-password-{dev, staging, production}</td>
<td>Administrative user account to the environment’s Lime Survey server</td>
<td>Username and password</td>
<td>Created in each AWS environment via Terraform, stored in each environment Secrets Manager (AWS Secrets Manager &gt; environment subaccount &gt; Oregon region)</td>
</tr>
<tr>
<td></td>
<td>{dev, staging, prod}-metastore-rds-password</td>
<td>Administrative access to the environment’s Hive Metastore Postgres instance</td>
<td>Password</td>
<td>Created in each AWS environment via Terraform, stored in each environment Secrets Manager (AWS Secrets Manager &gt; environment subaccount &gt; Oregon region)</td>
</tr>
<tr>
<td>LDAP</td>
<td>binduser</td>
<td>Non-privileged read access to the LDAP directory for linking other services to LDAP authentication/authorization</td>
<td>Password</td>
<td>AWS Secrets Manager &gt; ALM account &gt; Ohio region</td>
</tr>
<tr>
<td>LDAP server</td>
<td>alm-freeipa-admin-password</td>
<td>Administrative access to the LDAP server</td>
<td>Password</td>
<td>AWS Secrets Manager &gt; ALM account &gt; Ohio region</td>
</tr>
</tbody>
</table>

Source: City of Columbus

### Table 11: Accounts Managed through LDAP

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
<td>&lt;team member username&gt;</td>
<td>Secure network access to the project AWS environments</td>
<td>Username and password</td>
<td>Automatically linked to the project LDAP service</td>
</tr>
<tr>
<td>Jenkins</td>
<td>&lt;team member username&gt;</td>
<td>CI and CD service for private project code and deployments</td>
<td>Username and password</td>
<td>Automatically linked to the project LDAP service</td>
</tr>
</tbody>
</table>

Source: City of Columbus

### Table 12: Accounts Managed Through developers.smartcolumbusos@gmail.com

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Analytics</td>
<td>SCOS</td>
<td>Project account for managing access to the Google Analytics service</td>
<td>Username and password</td>
<td>Individually granted by the administrator</td>
</tr>
</tbody>
</table>

Source: City of Columbus
## Appendix A. Account Access Information

### Table 13: Personal Accounts that are Created and Granted Access

<table>
<thead>
<tr>
<th>Service</th>
<th>Account Name</th>
<th>Description</th>
<th>Credential Type</th>
<th>Credential Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>scos-jenkins</td>
<td>Jenkins service account to access private repositories</td>
<td>Username and password</td>
<td>Managed from the project Jenkins’ internal credential store</td>
</tr>
<tr>
<td>Dockerhub</td>
<td>&lt;team member username&gt;</td>
<td>Access to Dockerhub container artifact hosting repository</td>
<td>Username and password</td>
<td>Created by team member; access granted by any team member with administrative access inviting user to the project organization account</td>
</tr>
<tr>
<td>Travis CI</td>
<td>&lt;team member username&gt;</td>
<td>Create and configure CI and CD jobs for all project public code repositories</td>
<td>Username and password</td>
<td>Automatically created for GitHub user accounts with membership in the project public GitHub organization; user must grant programmatic access to their profile “organizations” when logging into Travis for the first time.</td>
</tr>
<tr>
<td>Gitter</td>
<td>&lt;team member username&gt;</td>
<td>Community chat service for coordination across team and non-team contributors</td>
<td>Username and password</td>
<td>Automatically created when logging into the Gitter service with existing GitHub account.</td>
</tr>
<tr>
<td>ZenHub</td>
<td>&lt;team member username&gt;</td>
<td>Kanban organization of project story/bug/spike cards recorded in GitHub</td>
<td>Username and password</td>
<td>Automatically linked to GitHub account with membership in the project GitHub organizations when logging into the ZenHub service with existing GitHub account.</td>
</tr>
</tbody>
</table>

*Source: City of Columbus*
## Appendix B. Related Document Links

Table 14 contains list of project related documents and their respective links.

### Table 14: Related Documents

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Management Plan</td>
<td>The purpose of the Data Management Plan (DMP) is to document how the data within the Operating System will be added, made accessible, and/or stored within the Operating System.</td>
<td>Found in References - Systems Engineering and Design: <a href="https://smart.columbus.gov/projects/smart-columbus-operating-system">https://smart.columbus.gov/projects/smart-columbus-operating-system</a></td>
</tr>
<tr>
<td>Data Privacy Plan</td>
<td>This Data Privacy Plan (DPP) provides high-level guidance, principles, and policies to ensure the privacy of Smart Columbus Demonstration data subjects and project participants.</td>
<td>Found in References - Systems Engineering and Design: <a href="https://smart.columbus.gov/projects/smart-columbus-operating-system">https://smart.columbus.gov/projects/smart-columbus-operating-system</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Direct Document Link: <a href="https://d2rfd3nxvhnf29.cloudfront.net/2019-09/SCC-D-Data%20Privacy%20Plan-FINAL-20190906%5B1%5D_0.pdf">https://d2rfd3nxvhnf29.cloudfront.net/2019-09/SCC-D-Data%20Privacy%20Plan-FINAL-20190906%5B1%5D_0.pdf</a></td>
</tr>
<tr>
<td>De-Identification Policy</td>
<td>This policy documents the process to de-identify such information so that it cannot be used to identify individuals or that reasonable basis does not exist to believe that the information could be used to identify individuals.</td>
<td>Found in References - Systems Engineering and Design: <a href="https://smart.columbus.gov/projects/smart-columbus-operating-system">https://smart.columbus.gov/projects/smart-columbus-operating-system</a></td>
</tr>
<tr>
<td>Performance Measurement Plan (PfMP)</td>
<td>This Performance Measurement Plan describes the outcomes of Smart Columbus and how the objectives of each of projects relate to them.</td>
<td>Found in References - Program Plans: <a href="https://smart.columbus.gov/programs/smart-city-demonstration">https://smart.columbus.gov/programs/smart-city-demonstration</a></td>
</tr>
</tbody>
</table>
## Appendix B. Related Document Links

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
</table>
| System Architecture and Standards Plan (SASP) | This SASP documents the architecture for each of the projects associated with the Smart City demonstration program and associated standards (whether Intelligent Transportation Systems (ITS) or other related standards) that will be referenced and applied to each project. | Found in Task B Program Systems Engineering: Deliverables: [https://smartcolumbusprogram.sharepoint.com/TaskBDeliverables/Forms/ContentType.aspx](https://smartcolumbusprogram.sharepoint.com/TaskBDeliverables/Forms/ContentType.aspx)  

*Source: City of Columbus*
# Appendix C. Acronyms and Definitions

Table 15 contains list of project specific acronyms used throughout this document.

<table>
<thead>
<tr>
<th>Abbreviation/Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM</td>
<td>Application Lifecycle Management</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>CAB</td>
<td>Change Advisory Board</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CLA</td>
<td>Contributor License Agreement</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CVE</td>
<td>Connected Vehicle Environment</td>
</tr>
<tr>
<td>DMP</td>
<td>Data Management Plan for the Smart Columbus Demonstration Program</td>
</tr>
<tr>
<td>DMS</td>
<td>Data Management System</td>
</tr>
<tr>
<td>DPP</td>
<td>Data Privacy Plan for the Smart Columbus Demonstration Program</td>
</tr>
<tr>
<td>EC2</td>
<td>Elastic Compute Cloud</td>
</tr>
<tr>
<td>ECR</td>
<td>Elastic Container Registry</td>
</tr>
<tr>
<td>ECS</td>
<td>Elastic Container Services</td>
</tr>
<tr>
<td>EKS</td>
<td>Elastic Kubernetes Service</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HCL</td>
<td>Hashicorp Configuration Language</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity and Access Management</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MFA</td>
<td>Multi-Factor Authentication</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>O&amp;M</td>
<td>Operations and Maintenance</td>
</tr>
</tbody>
</table>
### Appendix C. Acronyms and Definitions

<table>
<thead>
<tr>
<th>Abbreviation/Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTP</td>
<td>One Time Password</td>
</tr>
<tr>
<td>OU</td>
<td>Organizational Unit</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Application Security Project</td>
</tr>
<tr>
<td>PII</td>
<td>Personally Identifiable Information</td>
</tr>
<tr>
<td>PR</td>
<td>Pull Request</td>
</tr>
<tr>
<td>RDS</td>
<td>Relational Database Service</td>
</tr>
<tr>
<td>RSS</td>
<td>Rich Site Summary</td>
</tr>
<tr>
<td>S3</td>
<td>Amazon Simple Storage Service</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SC</td>
<td>Smart Columbus</td>
</tr>
<tr>
<td>SDC</td>
<td>Secure Data Commons</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Lifecycle</td>
</tr>
<tr>
<td>SES</td>
<td>Simple email service</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SoS</td>
<td>System of Systems</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SR</td>
<td>Security (Requirements)</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid State Drive</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure SHell</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>TDD</td>
<td>Test Driven Development</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>USDOT</td>
<td>U.S. Department of Transportation</td>
</tr>
<tr>
<td>UX</td>
<td>User Experience</td>
</tr>
<tr>
<td>VPC</td>
<td>Virtual Private Cloud</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>YAML</td>
<td>Yet Another Markup Language</td>
</tr>
</tbody>
</table>

Source: City of Columbus
Table 16 contains project specific terms used throughout this document.

Table 16: Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>A software application.</td>
</tr>
<tr>
<td>Travelers (End Users)</td>
<td>Travelers are End Users who utilize the features of SMHs to begin, pass through or complete their trips.</td>
</tr>
<tr>
<td>Bot</td>
<td>An autonomous program on a network (especially the Internet) that can interact with computer systems or users, especially one designed to respond or behave in a pre-defined way.</td>
</tr>
<tr>
<td>Commercial Off-the-Shelf System (COTS)</td>
<td>Software or hardware product that are ready-made and available for sale to the public.</td>
</tr>
<tr>
<td>Cloud Based</td>
<td>Stored, managed, and processed on a network of remote servers hosted on the Internet, rather than on local servers or personal computers.</td>
</tr>
<tr>
<td>Data Management System (DMS)</td>
<td>A secure, Software as a Service (SaaS) web-based app that allows management of an entire parking meter network.</td>
</tr>
<tr>
<td>Data Privacy</td>
<td>The reasonable expectation that data of a sensitive nature will be kept confidential, sanitized and/or encrypted, and respectfully and responsibly maintained by all Users, managers and collectors of the data.</td>
</tr>
<tr>
<td>Data Retention</td>
<td>The continued storage of data for compliance or business reasons.</td>
</tr>
<tr>
<td>Data Security</td>
<td>The tools, policies, practices, and procedures used to protect data from being accessed, manipulated, destroyed, or leveraged by those with a malicious intent or without authorization, as well as the corrective actions taken when data breaches are suspected or have been identified.</td>
</tr>
<tr>
<td>Data Volume</td>
<td>The amount of data in a file or database.</td>
</tr>
<tr>
<td>Dependency</td>
<td>When one project, agency, or entity requires data or functionality provided by another project, agency, or entity to meet its objectives.</td>
</tr>
<tr>
<td>Dockless</td>
<td>A station-free concept developed for bicycles that allows riders to end their rides and park the bikes anywhere. This mechanism offers affordable transit, attempts to solve FMLM issues, services areas without transit, and offers to cities ridership data.</td>
</tr>
<tr>
<td>Enabling Technologies</td>
<td>An innovation that alone or paired with an existing solution produces a better End User solution at a rapid rate.</td>
</tr>
<tr>
<td>Enhanced Human Services</td>
<td>Meeting human needs through the application of technology that focuses on prevention as well as remediation of problems and maintains a commitment to improving the overall quality of life of users of the technology-based solutions.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Experience Columbus</td>
<td>An organization whose mission is to market and promote Columbus services, attractions, and facilities to visitors, meeting planners, convention delegates and residents.</td>
</tr>
<tr>
<td>Failure Operations</td>
<td>When a complete failure of the intersection occurs, primarily due to loss of power or other malfunctions.</td>
</tr>
<tr>
<td>Functional Paradigms</td>
<td>A style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.</td>
</tr>
<tr>
<td>Git Semantic Version</td>
<td>Version numbers that convey meaning about the underlying code and what has been modified from one version to the next.</td>
</tr>
<tr>
<td>Infrastructure as a Service</td>
<td>A standardized, highly automated offering in which computing resources owned by a service provider, complemented by storage and networking capabilities, are offered to customers on demand.</td>
</tr>
<tr>
<td>Interactive Voice Response (IVR)</td>
<td>IVR technology allows a computer to interact with humans. Through voice prompts, a User tactilely enters tones using number keys on a keypad. In telecommunications, IVR allows customers to interact with a company's host system via a telephone keypad, after which customers can inquire about services through the IVR dialogue.</td>
</tr>
<tr>
<td>Kafka topic</td>
<td>A distributed, persistent message log</td>
</tr>
<tr>
<td>Micro Services</td>
<td>Microservice architecture, or simply microservices, is a distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations.</td>
</tr>
<tr>
<td>Multimodal Transportation</td>
<td>Travel done via more than one mode of transportation.</td>
</tr>
<tr>
<td>Normal Operations</td>
<td>When a signalized intersection is cycling through its preplanned phases correctly, servicing all approaches including pedestrian phases.</td>
</tr>
<tr>
<td>Open Data</td>
<td>Information that is freely available for anyone to use and republish as they wish.</td>
</tr>
<tr>
<td>Open-Source Concepts</td>
<td>The notion of open collaboration and voluntary contribution for software development by writing and exchanging programming code.</td>
</tr>
<tr>
<td>Parking Facility</td>
<td>Land or a structure used for light-duty vehicle parking.</td>
</tr>
<tr>
<td>Procurement</td>
<td>The act of obtaining or acquiring goods, services or works, from a competitive bidding process.</td>
</tr>
<tr>
<td>Real-Time Data</td>
<td>Information that is delivered immediately after collection.</td>
</tr>
<tr>
<td>Software as a Service (SaaS)</td>
<td>An extension of the Application Service Provider (ASP) centralized computing model, the SaaS delivery model hosts and provides web-based interface access to a software app over a network. The service is managed centrally, in large data centers, most often in a cloud-computing environment.</td>
</tr>
<tr>
<td>Third Party</td>
<td>Organizations not affiliated with the Smart Columbus Program.</td>
</tr>
<tr>
<td>Transportation Network Companies (TNCs)</td>
<td>Private businesses, nonprofits and quasi-governmental agencies that offer one or more types of transportation for use in exchange for payment.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Terraform</td>
<td>An open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp Configuration Language (HCL)</td>
</tr>
<tr>
<td>Test Driven Development</td>
<td>A software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved so that the tests pass.</td>
</tr>
<tr>
<td>&quot;v&quot; Static Character</td>
<td>Static letter “v” represents the Requirement version.</td>
</tr>
<tr>
<td>Version Number</td>
<td>An integer incrementing by one, indicating the number of revisions made to the Requirement.</td>
</tr>
</tbody>
</table>

Source: City of Columbus
Appendix E. References

An evolving list of references is made available to the Operating System users at https://GitHub.com/smartcitiesdata/smartcitiesdata/wiki/Resources.

E.1 BOOKS

- The Journey to Enterprise Agility – Daryl Kulak, Hong Li
- Programming Elixir 1.6 - Dave Thomas
- Elixir in Action - Sasa Juric
- Learn Functional Programming with Elixir - Ulisses Almeida
- The Little Elixir and OTP Guidebook - Benjamin Tan Wei Hao
- Designing Elixir Systems with OTP - James Edward Gray II, Bruce Tate
- Programming Phoenix 1.4 - Chris McCord, Bruce Tate, Jose Valim
- Kubernetes in Action - Marko Luksa
- Kafka: The Definitive Guide - Todd Palino, Gwen Shapira, Neha Narkhede

E.2 TUTORIALS AND ONLINE COURSES

- Elixir - exercism
- Elixir - Elixir School
- Elixir for Programmers - Dave Thomas
- Kubernetes - kubernetes.io
- Introduction to Kubernetes - The Linux Foundation
- Kubernetes the Hard Way - Kelsey Hightower
- Kafka - Simplilearn
- Getting Started with Apache Kafka - Pluralsight
- Presto - Presto.io
- React - Getting Started
- Docker - Get Started
- Amazon S3 - Getting Started
- Object-based storage
Table 17 includes a sample breakdown of monthly costs for using AWS.

Table 17: AWS Monthly Billing Sample

<table>
<thead>
<tr>
<th>Service</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td>$275.00</td>
</tr>
<tr>
<td>Data Transfer</td>
<td>$1,000.00</td>
</tr>
<tr>
<td>RDS</td>
<td>$2,000.00</td>
</tr>
<tr>
<td>Secrets Manager</td>
<td>$25.00</td>
</tr>
<tr>
<td>Support</td>
<td>$1,300.00</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>$5.00</td>
</tr>
<tr>
<td>EC2</td>
<td>$6,500.00</td>
</tr>
<tr>
<td>ECR</td>
<td>$30.00</td>
</tr>
<tr>
<td>Config</td>
<td>$15.00</td>
</tr>
<tr>
<td>Route 53</td>
<td>$15.00</td>
</tr>
<tr>
<td>EKS</td>
<td>$800.00</td>
</tr>
<tr>
<td>Cost Explorer</td>
<td>$5.00</td>
</tr>
<tr>
<td>Cloud Watch</td>
<td>$100.00</td>
</tr>
<tr>
<td>EFS</td>
<td>$5.00</td>
</tr>
<tr>
<td>CloudTrail</td>
<td>$25.00</td>
</tr>
<tr>
<td>ElastiCache</td>
<td>$250.00</td>
</tr>
<tr>
<td>Key Mgmt</td>
<td>$75.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$12,425.00</strong></td>
</tr>
</tbody>
</table>

*Source: City of Columbus*