Enterprise Security Configuration and Compliance Architectural Overview

Architectural Overview 2

System Stack 2

System Components 3

ESC2 End User Application 3

Web Server/API 3

ESC2 Database 3

Workflow Service 4

Security Dashboard 4

Syslog based Security Alerts – Future 4

Log Data – Future 4

Active Threat Alerts – Future 4

Design Pattern Based Architecture 4

Design Details 5

End User / API Communication Design 5

References 6

Resources 6

# Architectural Overview

The Enterprise Security Compliance and Configuration System (ESC2) is designed to be an open source system that is easy to implement by small and mid-sized companies.

The system components will be packaged using Docker into a single container. This design will simplify the deployment and configuration. Companies will be able to configure the architecture to turn on or off various services.

With simplicity of deployment in mind, the system must be able to provide both operational and analytic functionality in a single container and likely a single database. This may cause performance issues as the database is forced to provide transactional and analytic processing. While this may be an issue in the long run for companies that grow, the ESC2 product (as stated above) is focused on small and mid-sized companies. The reasoning that this hybrid operational warehouse approach will function acceptably, is that there are only a small number of technologists working with the system at the companies that adopt it.

To offset some of the performance risk, the system will be designed to host the database inside the docker image or externally. This should allow companies to boost the capacity of the database if required.

The architecture will be designed to function at minimal cost in either Amazon AWS or Microsoft’s Azure cloud infrastructures. It will also be possible to host the platform on premise using a Windows or Linux server.

The ESC2 system will be offered as open source software under the MIT License. All third party tools and libraries will also need to be either free or open source with compatible licenses. All source code will be accessible on GitHub at <https://github.com/brian-nelson/ESC2>. Defects or suggestions should be submitted as issues through GitHub.

# System Stack

**Database**:

The ESC2 system will be configurable to support Sql Server or PostgreSQL database. The docker image will need to support either choice. System SQL must be able to run on either database.

**Server Code**:

The server software will be written in .Net Core 5+ using the C# programming language.

**Server Side Scripting**:

Server side scripting will be done in Python 3.6+.

**Client Code**:

Client side web software will be written in JavaScript. User interface elements will be styled using Bootstrap. The client application will be based on React.

## System Components

The following diagram identifies the primary components of the web-enabled architecture that will be developed. The ellipses in green indicate future opportunities for expansion of the ESC2 system. The expanded feature sets will allow ESC2 to provide a complete view of security preparation, as well as active and emerging threats.

Diagram

Description automatically generated

### ESC2 End User Application

The end user application will be developed as a React based single page application. The user interface components will be coded in JavaScript and will be styled using Bootstrap libraries to allow the app to render on phones, tablets, laptops, and desktops. The application will allow users to manage the Information Technology (IT) Assets and apply the Implementation Guides (STIG or other) to them. Once implemented the application will allow the implementations to be audited and recommend further work needed to completely secure the IT assets.

### Web Server/API

The web server will be constructed using the .Net Core framework using the C# programming language and will implement the cross-platform Kestrel web server offered by Microsoft. The web server will host both the static JavaScript files for the React app, as well as a REST based application programmer interface (API).

### ESC2 Database

The ESC2 Database will initially be based on Microsoft SQL Server. In the future, PostgreSQL will be an additional option. The database is fully described in the ESC2 Data Dictionary document which is available in the GitHub repo. The database can exist inside the container, or it can be hosted on another machine. It will likely scale better if externally hosted.

### Workflow Service

The Workflow Service will allow the system to conduct automated tasks. These tasks may aggregate data into Fact tables. It will also be responsible for batch imports from external sources. The Workflow Service will be coded in .Net Core framework with libraries that support data access to the ESC2 database. The service will also allow Python Scripts to be executed. The ability to allow script execution will enable companies to extend the ESC2 system as required.

### Security Dashboard

It will be possible to connect security dashboard software to the database. A reference implementation will be provided using Grafana to allow companies to share security metrics beyond the users of the ESC2 system. Using Grafana or another 3rd party dashboard is optional as the user interface will have built in dashboard capabilities.

### Syslog based Security Alerts – Future

A near real-time feed from any security appliances, routers, switches will allow dashboards to respond to current threats. These will utilize Syslog based publishing of events from these devices to the ESC2 System. ESC2 will have a built in listener for Syslog events that will automatically write the entries to the ESC2 database.

### Log Data – Future

Additional server logs will be published into the ESC2 system in the future. These will come from web servers, and other software sources.

### Active Threat Alerts – Future

Integration with various government publications that are focused on sharing current and emerging threats. This feed will likely be from a governmental source. Further research needs to be completed to determine if this will be based on an RSS feed or if there is another technology available for integration.

# Design Pattern Based Architecture

The architecture for the ESC2 System is based on standardized design patterns. A design pattern is a named, and well-defined solution to a common programming problem. Using design patterns allow for organized and reusable code where each component is responsible for a specific well understood task. (Gamma, 1995)

For example, a Repository object can be defined as the object responsible for storing a file to defined location. There may be many software implementations of the Repository object that can be used to save the same file to different storage locations.

Possible File Repository Implementations:

* AwsS3FileRepository – Stores files to AWS S3
* AzureBlobStorageFileRepository – Stores files to Azure Blob Storage
* LocalSystemFileRepository – Stores files to the local hard drive

These three classes do one simple task, which is to store a file to a specific type of data storage. The internal implementation for each is different, but they will share a common interface definition that will allow the program to be reconfigured even after it is compiled to alter where files are stored.

Additional implementations can be defined as new storage locations are designed. The application can then be configured to use a specific type of the File Repository as desired.

Building applications with standardized building blocks simplifies construction and maintenance as developers are immediately able to grasp the intent of each block of code.

# Design Details

The following sections detail out specific components discussed in the above overview. Additional details will be included as the design proceeds.

## End User / API Communication Design

Diagram

Description automatically generated

Design Patterns in Use

* Data Access Object is really a renamed Repository. The distinction here is that it doesn’t directly retrieve data from a source but works through the API to gain data.
* Data Controller is an implementation of the Front Controller pattern. It translates a web request received of HTTP/S into instructions in code. (Fowler, 2003)
* The Repository are responsible for managing rows in the database and translating them to/from objects in the application. There is one repo class per table in the database. (Fowler, 2003)

Component Definition

* React User Interface components will request data using a single data Service object.
* The data requests that the Service receives are proxied to one or more Data Access Objects.
* The Data Access Objects are responsible for loading one type of data from the web hosted API.
* Axios is a library that simplifies the asynchronous data requests.
* The data requests are sent via HTTPS over the network (local or internet) to the web server.
* The Kestrel based Web server receives the data request in various Data Controller objects. Each Data Controller is responsible for one type of data.
* A Data Service receives the requests from the controllers and proxies them to the appropriate Data Repo.
* The Data Repo translates the objects to SQL Select, Insert, Update or Delete statements and executes them against the database.

This model of data communication is abstracted into Pattern based components for maximum reuse of code with minimal coding.

# References

*MIT License –* Wikipedia. Retrieved December 18, 2021, from https://en.wikipedia.org/wiki/MIT\_License.

*Managing information security risk - NIST*. (2011). Retrieved December 6, 2021, from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-39.pdf.

Fowler. (2003). *Patterns of enterprise application architecture (1st edition)*. Addison-Wesley.

Hohpe, & Woolf, B. (2004). *Enterprise integration patterns: designing, building, and deploying messaging solutions (1st edition)*. Addison-Wesley.

Metsker. (2004). *Design patterns in C (1st edition)*. Addison Wesley.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design patterns : elements of reusable object-oriented software*. Addison-Wesley.

# Resources

**Security**

* <https://us-cert.cisa.gov/ncas>
* <https://public.cyber.mil/stigs/downloads/>

**Technologies planned for usage in project**

* <https://reactjs.org/>
* <https://getbootstrap.com/>
* <https://axios-http.com/docs/intro>
* <https://www.docker.com/>
* <https://en.wikipedia.org/wiki/Representational_state_transfer>
* <https://en.wikipedia.org/wiki/RSS>
* <https://grafana.com/>