

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/17/2025 | Meghan Walsh | Analyzed the constraints and design requirements for the development of a web-based game for The Gaming Room, based on their current Android mobile application, Draw It or Lose It. |
| 1.1 | 3/24/2025 | Meghan Walsh | Evaluated the characteristics, advantages, and weaknesses of each operating system to present to The Gaming Room. |
| 1.2 | 4/7/2025 | Meghan Walsh | Analyzed the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room company has a successful mobile application available only on the Android store, Draw It or Lose It, and is seeking to develop a web-based game that serves multiple platforms based on the Android application version. The staff at The Gaming Room does not know how to set up the environment and therefore, to facilitate the development of the web-based version of the gaming app, they will need our help in streamlining the development. Our current aim is to address their specific software requirements while adapting the mobile Android application to function as web-based game that serves multiple platforms:

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time.

## Requirements

1. Adapt the mobile Android application, Draw It or Lose It, to function as web-based game that serves multiple platforms.

2. Each game will have the ability to have one or more teams involved.

3. Each team will have multiple players assigned to it.

4. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.

5. Only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

1. A significant design constraint is regarding the operating platform requirements of the web-based application. Where the mobile application Draw It or Lose It is adapted to the operating platforms of Android phones, our application must be adapted to work with the operating platforms of laptops and desktops. This will require a significant change when compared to the software needs of their current mobile application, as the operating systems of laptops and desktops differ significantly to that of mobile phones. Additionally, the application must serve multiple platforms. There are significant differences between the operating platforms of various laptop and desktop machines, so we must design and test the application to ensure the application works seamlessly across a plethora of operating systems.

2. The application must allow for one or more teams involved in each game. Our application must be designed with this in mind, so that is can easily adapt to handle a range of one-to-many teams.

3. The application must allow for multiple players assigned to it. Therefore, our application must be adaptable to the number of players involved and designed with this in mind.

4. The application must require players to input *only* unique game and team names. Thus, it must handle situations where users input an already used name, and allow users to check whether a name is in use when choosing a team name.

5. The application must allow for only one instance of the game can exist in memory at any given time. This should be done via creating unique identifiers for each instance of a game, team, or player.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

The ProgramDriver Class is the main entry point of the application. As demonstrated by the arrow with the <<uses>> sign, the ProgramDriver Class uses the SingletonTester Class. Therefore, the ProgramDriver class uses, and depends upon, the SingletonTester class. Thus, this relationship demonstrates the OOP principle of Dependency, which occurs when one class relies on another for functionality.

The Entity class is a parent class for the classes: Game, Team, and Player, providing shared fields and methods across the child classes. The Entity class demonstrates the OOP principle of Encapsulation, as it stores fields like ID, as well as methods like toString(), in a single unit (the class) and manages it’s fields’ and methods’ access by making them public (+) or private (-). GameService, Game, Team, and Player, also demonstrate Encapsulation through their own bundling of fields and methods. The relationship between parent class Entity and the child classes Game, Team, and Player, as shown by the open arrows pointing from each child class to Entity, demonstrates the OOP principle of Inheritance. Each child class inherits Entity, and thus gets access to the public methods or fields encapsulated by the Entity class, like getName() and getId(). Given the diagram, it is likely that Entity demonstrates the Single Responsibility Principle (SRP), meaning it likely is solely responsible for one action, managing common attributes, and does not handle any other logic of the code.

The classes Game and Team have a one-to-many relationship, where a Game contains many teams. Similarly, the classes Team and Players have a one-to-many relationship, where a Team contains many players. These relationships demonstrate the OOP principle Composition, which describes a relationship between objects where one object owns or contains other objects, as Games contain multiple teams and Teams contain multiple players. These OOP principles and relationships allow for the us to fulfill the software requirement that one or more teams be able involved in one game (one to many teams are contained in one game) and the software requirement that one or more players can be on one team (one to many players are contained in one team).

Finally, GameService is a class responsible for creating and managing new GameService Objects. It is a Singleton class, meaning it ensures there is only one instance of itself is used all the other classes in the application. The usage of the singleton design pattern allows us to meet the software requirement that the application must allow for only one instance of the game to exist in memory at any given time. This singleton design is done by making the constructor private (-GameService()) so that other classes cannot create a new instance using the constructor, creating the single private static instance of itself (-service: GameService), and making a public, static method (+getInstance(): GameService) that other classes can use to access the single private static instance of itself. Using the public methods encapsulated in GameService will allow us to meet the software requirement that the application must require players to input *only* unique game and team names and allow users to check whether a name is in use when choosing a team name. Using the GameService methods +addGame(name:String), +getGame(id:long), +getGame(name:String), +getGameCount(), +getNextPlayerId(), +getNextTeamId(), we will be able to access current games to see if a specific game name and/or ID already exists, add new Game names and/or IDs, and iterate through each game, using the methods to check if a specific game name or id already exists. Additionally, the OOP principle of polymorphism allows GameService to select which method to use ( +getGame(id:long) or +getGame(name:String)) based on the inputted parameter’s datatype, which will help us ensure that the correct game is retrieved based on either its unique ID or name.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS computers are known for their user-friendly user interface, excellent integration with other Apple devices, and high-resolution Retina displays. This allows for a seamless, visually appealing, and easily accessible user experience. MacOS also has built-in development tools like Python and Docker. However, when compared to Linux or Windows, given the proprietary nature of the software, they lack the customization when it comes to hosting complex web apps. Thus, if any of the Gaming Room’s server require specific and personalized configurations, it may not be adaptable to MacOS. They are also generally more expensive than other machines. MacOS is not traditionally favored for hosting complex web applications but it may still be functional as a server-side environment for development and testing | Linux computers are known for being open-source and customizable. Where MacOS’s software is proprietary and users have limited control over the operating system, Linux allows full control over the operating system, which enables users to modify and optimize it for a web-based software application. It’s open-source nature also makes it cost-effective, as Linux is completely free, with no licensing fees. However, it does not have support for proprietary applications like Adobe products and Microsoft Office. Additionally, Linux doesn’t have centralized customer support team, so the application developers may have to trouble-shoot Linux-based issues themselves. As a result, Linux is widely used in server environments because it is seen as stable, secure, and scalable. Additionally, many developers have in-depth experience with Linux machines. All of the above make Linux a strong choice for hosting The Gaming Room’s server-side of the application. | Whereas Linux relies predominantly on the command-line, Windows Server has a well-developed GUI, making it more user-friendly for managing server-side applications. It is also compatible with both open-source and proprietary software, allowing a middle-ground between the proprietary software of MacOS and the open-source software of Linux. However, Windows is known for its security concerns, as it has been more vulnerable to malware and attacks. As a result, users often complain of frequent patching and security updates. As a result, if the server-side of the web-based application is hosted on Windows, it will need to include robust security features designed specifically for windows-hosted servers, as the server-side typically stores sensitive user information like passwords and usernames. | The main advantage of mobile devices is their portability, and thus are useful for small-scale applications and testing. They also require less power consumption and are more energy-efficient. However, mobile devices have limited performance and hardware constraints, as they are not designed to handle high traffic or complex databases, which is a significant drawback for hosting the server-side of the web application. They have weaker processors, limited RAM, and slower storage. Android and iOS devices are not designed for server hosting. Mobile devices are not designed for server hosting, and as a result, they lack the infrastructure, security, and stability that is required to develop and maintain a web-based application’s server-side like The Gaming Room’s game server. |
| **Client Side** | MacOS machines are significantly more expensive than either Linux or Windows machines. Additionally, macOS software is proprietary. As a result, developing and testing client-side code is expensive, as you will need to purchase the necessary software or machines to use. Additionally, developing across different types of MacOS machines requires that one acquires different types of Mac devices (e.g., MacBook, iMac, Mac Mini, M1/M2 chip models) to test your client software across varying MacOS hardware and software. This testing period is not only costly, but time intensive, especially since developers must test across Intel and Apple Silicon chips. Developers will need expertise in macOS-specific languages like Swift, software like XCode, and macOS specific hardware. However, macOS still boasts a relatively consistent environment compared to Linux, which allows for a more consistent performance and straightforward UI testing in macOS. Additionally, macOS machines are very popular and many users will already feel comfortable with the client-side UX/UI design of mac products. | Given the open-source nature of Linux, the cost of working with Linux machines is significantly lower than that of macOS machines. However, whereas MacOS machines are more consistent in their hardware, Linux can run on a wide range of hardware and thus developers may still need to test the client-side software across multiple distributions and a plethora of hardware. Distributions all have different package management systems, desktop environments, and configurations that will add more time to the testing phase of developing the client-side. Most developers are used to using Linux and its open-source software, so developers may feel more knowledgeable developing in Linux. However, User Interface development will need to work seamlessly across the several desktop environments (GNOME, KDE, etc.) that Linux machines may be running. Thus, there exists a need for developers with expertise in these areas and more time to test on all desktop environments. Additionally, Linux has a much smaller user base compared to MacOS and Windows, which is a large consideration one must take into account before developing client-side software in Linux, a Linux-hosted client-side may feel unfamiliar or comfortable to users. | Whereas Linux is cheaper to develop software with, developing for Windows often requires paid licenses. Additionally, development environments like Visual Studio and third-party libraries or frameworks may require subscriptions or payments. However, Window’s primary costs are consistently cheaper than MacOS’s primary costs, offering a cost-wise middle ground between mac and Linux. Another benefit of Windows is that it is popular. Whereas users may not feel as comfortable or familiar with a Linux-hosted client-side, Windows is well-known and often preferred by the large base of users that already have windows machines. Unlike MacOS, which has a more consistent hardware configuration, Windows has a variety of hardware configurations, drivers, and system setups which require more developers with expertise in these software and hardware components and more time to test the client-side across all of them. When compared to macOS, Windows also allows for more UI/UX configuration, which is an important benefit when designing the client-side of an application. | Mobile Devices run mobile versions of a plethora of platforms (iOS, Android, Web, etc.), thus each platform requires developers with expertise on that specific platform. Additionally, developers must purchase resources for each platform to develop and test on, making it more costly. The specific expertise of mobile-device web developers will also be needed. However, mobile devices prioritize user interface design over things like higher performance. The UI/UX-focused nature of mobile phones allow for responsive design, touch interfaces, etc., which can make the client-side user experience easier, simpler, and more enjoyable for the user. |
| **Development Tools** | Beyond the languages, tools, and IDEs used across-platforms, which include JavaScript, Python, Visual Studio Code, etc., languages and tools optimized for development in MacOS include:  Swift, the primary programming language designed specifically for MacOS machines.  XCode, Apple’s official IDE, which is optimized for performance on MacOS machines.  Homebrew, the preferred package manager of MacOS, which will aid in the installation of necessary programs like Python we may need during game development.  XQuartz, the open-source version of Window’s X11 for macOS. Given that the Gaming Room wants us to develop multi-platform application, if we decide to develop a Linux-hosted client-side, XQuartz provides the necessary infrastructure to run the graphical client-side elements on macOS since many Linux applications, especially those that use graphical interfaces, depend on X11. | As Linux is widely accepted to be the best operating platform for software development when compared to MacOS or Windows, there are a plethora of programming languages and tools at our disposal.  C++ is highly optimized for performance on Linux systems. The low-level nature of C++ is beneficial for building high-performance, resource-intensive games.  **Node.js** is very efficient on Linux systems because it is non-blocking and has event-driven architecture. Thus, it is great for the specific development needs of game applications.  Most programming languages used by developers, like Python and Java, work very well on Linux machines.  IDEs compatible with Linux machines include Eclipse and JetBrains IntelliJ IDEA / PyCharm, among a host of other IDEs available in free and paid versions.  Vim is another tool that is very popular among Linux developers. It is a highly customizable and efficient text editor. As a result, many developers on the game app project will already know and be comfortable using Vim. | C# is one of the most popular languages for game development. It was developed by Microsoft and runs on the .NET Framework, thus is highly optimized to Windows machines.  Visual Studio is a Windows-optimized IDE that is great for C# and .NET development, among other languages.  PowerShell is the command-line shell scripting language built on the .NET framework. It is built for task automation and configuration management, and thus is very important for Windows-based automation and deploying scripts.  Windows Containers provide a lightweight, isolated environment that could make our game application development easier to develop, deploy, and manage. It allows for containerized game deployment in Windows without relying on Linux. | Development tools on mobile devices must allow for **cross-platform compatibility**, **performance optimization**,and **UI responsiveness**, as it must support both Android and iOS platforms.  As such we should consider:  Swift, the primary programming language designed specifically for MacOS machines, for iOS mobile devices.  XCode, Apple’s official IDE, which is optimized for performance on MacOS machines, including iOS mobile devices.  Kotlin, the official language for Android development, can be used for Android mobile devices.  Java, which is widely supported and used in many Android applications, is great for Android applications and is cross-platform.  In addition to Java, using cross platform languages like Python will allow for the required cross platform development. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**

Linux is perfect for transforming Draw It or Lose It into distributed, multi-platform game. Linux is the best platform for web and API hosting, security, flexibility, and scalability at the lowest cost. Linux is open-source and customizable. Given Linux allows full control over the operating system, developers can modify and optimize it for a web-based software application. It is the most cost-effective option, as Linux is completely free, allowing us to direct funds to other needs, such as storage or memory management options. Additionally, Linux is widely used in server environments because it is seen as stable, secure, and scalable. Many developers already have in-depth experience with Linux machines and therefore there will not be a strong learning curve for developers or need for developers with a more niche expertise area. As Linux is widely accepted to be the best operating platform for software development when compared to MacOS or Windows, there are a plethora of programming languages and tools at our disposal. This includes C++, which is highly optimized for performance on Linux systems. The low-level nature of C++ is beneficial for building high-performance, resource-intensive games. Additionally, Node.js is very efficient on Linux systems because it is non-blocking and has event-driven architecture. Thus, it is great for the specific development needs of game applications. Most programming languages used by developers, like Python and Java, work very well on Linux machines. IDEs compatible with Linux machines include Eclipse and JetBrains IntelliJ IDEA / PyCharm, among a host of other IDEs available in free and paid versions. It also has easy integration with Android tools through Android Studio and the Android SDK. Linux also powers most of the world’s web servers and cloud environments like AWS, GCP, and Azure.

Linux is the best operating platform for usage in cloud environments and works with many cloud infrastructure services. AWS (Amazon Web Services), a cloud computing platform provided by Amazon, has traditionally been viewed as the most cost-effective option in cloud computing and remains the leader in the cloud computing market. AWS has a great advantage over the competition as they command a large scale of businesses and can offer lower prices than the competition.

As a result, I recommend running Linux on AWS cloud servers, as this will provide the customizability, security, and performance of Linux with the storage and memory requirements of AWS’s cloud instances, which boast impressive scalability, availability, and integrated services. AWS provides several Linux-based operating systems for use on its cloud infrastructure that the Draw It or Lose It game can use.

1. **Operating Systems Architectures**

One of the most important features of Linux is that it allows full control over the operating system. This makes Linux architecture appealing for development, particularly for projects like Draw It or Lose It, because it allows for developers to customize the system to meet their specific needs. Additionally, Linux supports a broad array of hardware widely-used architectures, including x86 (32-bit), x86\_64 (64-bit), ARM and ARM64 (AArch64), making it highly versatile. Linux also follows a standard structure with directories like /bin, /etc, /home, /usr, and /var, which organize system and user files logically, in a way that is already familiar to developers working on Draw It or Lose It. Additionally, because we are transforming the game into a web application, we need our chosen operating system to have strong networking capabilities. Linux is a great choice because it includes a configurable and powerful networking stack that supports IPv4, IPv6, firewalling, routing, VPNs, and bridging. As with other aspects of the platform, Linux’s networking capabilities allow more control and configuration over how data is transferred across devices and networks, which allows for developers to have a networking environment for that is optimized for the game.

These features are enhanced in a cloud environment, as aspects like storage and memory are even more scalable, available, and manageable, and the Linux instances can be easily replicated, resized, or backed up.

1. **Storage Management**

The game application will have many high-definition image files and historical records of all players, teams, and games ever played, which must be factored into how much storage is needed. Given the game will handle multiple game instances at the same time, we will need to account for the high storage needs of many games, each with their own player data, teams, scores, etc., all running simultaneously, and thus pulling from and updating our data in storage. A computer’s storage space and capabilities, along with the level of abstraction enacted upon it by the operating system, is variable across the many machines a user could use to access our web application. Directing the web application’s storage requirements to a cloud computing platform, which provides companies computing services over the internet instead of using local servers or personal devices, will allow us to create a storage system and environment that fits our storage needs. Using Linux instances on AWS allows for a level of configuration and salability that cannot be found on any other platform. Cloud computing is a strong choice for storage management because storage attributes can change based on design requirements. A key benefit of cloud computing is the ability to consume what is needed when it is needed, which is great for storing the game’s assets like images, drawings, or user information, regardless of the current storage needs. Linux servers can easily connect to Amazon S3, for example, which offers virtually unlimited storage. Additionally, cloud storage options like AWS also typically have Pay-as-you-Go options so that we pay only for the resources we use, which significantly reduces the game’s costs.

1. **Memory Management**

Given the Draw It or Lose game requires pictures that will need to be rendered at a steady and rapid rate, developers will need to consider how memory is effectively managed in the software application to render the pictures in a minimally memory-intensive way. Additionally, running lists of players, teams, and games in current play must be constantly updated, and thus the memory requirements of this constantly updating database must be managed appropriately. Given the memory requirements of the game, offloading memory management to remote cloud platforms will allow for the memory requirements and hardware needed to optimize memory management. Linux is known for its robust and customizable memory management system, including fail safes like Out-Of-Memory Killer Protection, optimization tools like top and htop, and using virtual memory to abstract physical RAM. Linux cloud servers like AWS’s Linux cloud instances can be scaled up or down based on demand, providing a flexible amount of memory for the game applications. Whether we have a small team of players or a massive user base, we can allocate more computing resources, like CPU or RAM as is needed at any given time.

1. **Distributed Systems and Networks**

For Draw It or Lose It to be optimized as a web application, it will need to be transformed into a distributed system. At the core of a multi-platform web-based game is a central server that clients can connect to. Using a Linux instance of AWS allows us to create a centralized backend that maintains the application, handles client requests, maintains a database of game information, and manages storage and memory as it is needed. This allows for a more consistent and stable backend that is not reliant on the devices or network conditions of the players. To manage communication between the front-end web client and the backend, we will use RESTful APIs. REpresentational State Transfer (REST) is one of the most widely used approaches for building web-based APIs and is thus perfect for communication between the front-end and the back-end of our web application. It will allow for the client/server communication to be easy interactable, secure, and responsive. For connectivity and outages issues on the client side, we can use retry logic, timeout handling, and reconnect and resume protocols. For connectivity and outages on the server-side, cloud platforms like AWS are built with redundant networking, which includes: the usage of multiple data centers to make sure servers are always available, Virtual Private Cloud (VPC) networks to control network traffic, and load balancing to distribute traffic to the server to multiple Linux-based instances as needed.

1. **Security**

Given the fast-changing, rapidly evolving nature of computer systems, the application programmer, needs to implement protection mechanisms order to guard the application against an evolving threat landscape. Given this, I suggest that developers focus on preventative countermeasures for potential security attacks. Cyber threats are rapidly getting more advance and thus developers should implement up-to-date antivirus and anti-malware software. Our developers should implement the Principle of Least Privilege (PoLP), a fundamental security guideline that ensures programs, users, and systems are granted only the necessary permissions to perform their tasks, which minimizes damage from any security breaches. Additionally, role-based access control (RBAC) should be used to ensure each user has only the necessary permissions.  Just-in-Time (JIT) privileges, which grants users temporary, need-based access to privileged accounts and resources, can be used as well, as allowing privileged accounts for only a specific, limited timeframe reduces the risk of unauthorized access or data breaches. Linux boasts a secure and stable OS which supports many built-in and configurable security features including key-based authentication for secure server access using SSH, Discretionary Access Control (DAC), where every file and process has an owner, group, and set of permissions, and Mandatory Access Control (MAC) to restrict access to resources based on pre-determined rules. Linux also allows developers to configure firewall rules, so we can determine what traffic is allowed on our system. Linux based could servers like those by AWS can use VPC networks to control network traffic, IP address ranges, subnets, and route tables, allowing for greater control over the access and data stored on our applications backend. Network Access Control Lists can control traffic at the subnet level. Additionally, AWS offers encryption options for data stored in services like S3. Amazon also has their own threat detection services like GuardDuty and CloudWatch that we can utilize.