Luke Air Force Base
Mass Notification System

Created by Ryan Shah, Brandon Park, and Luis Plascencia

Table of Contents

Fictitious Press Release

This fictitious press release is part of a popular concept created by Amazon called the PRFAQ-- a fictitious press release and frequently asked questions. Cal Poly's Digital Transformation Hub (DxHub) taught us this strategy as we were struggling to get a feel for the problem at hand. At first we believed we had an understanding of the problem up until we were asked questions that covered completely different aspects of the problem.

In this fictitious press release, you will find our thought process and overall dissection of the problem at hand. The goal is to first start with the recipient of the alert message and progress up the ladder back to the Group Commander sending the message.

Fictitious Press Release

Luke Air Force Base Enables Text Message Alerts to Base Personnel with New Mass Notification System

- New cloud based alert system ensures that credentialed base personnel receive high priority text message alerts for on base emergencies like dangerous storms, active shooter situations, and notifications from their commanders.

Luke Air Force Base, AZ – January 1st, 2022

[Summary] Air Force Magazine | Technology - Luke Air Force Base's 56th Maintenance Group has launched PROJECT HERMES, a new text and email message based-notification system that enables commanders to push custom emergency and other high importance messages directly to military personnel under their command. Personnel register on a web portal providing their cell phone, name, rank, title, and email address; no CAC card needed. Signup is mandatory, enabling commanders to send targeted messages to their subordinates while also allowing base leadership to send base wide notifications. As such, the system is dually used for broadcasting emergency messages as well as facilitating messaging between leaders and members of specific groups within the base.

[Problem] Luke AFB has a legacy mass notification strategy that relies on loudspeaker, social media, messaging, email messages, and a limited ability to send group SMS text messages. These channels are either easy to ignore, easy to miss, or overkill for specific situations. Base command needs to be able to send base-wide notifications to all credentialed military personnel for events like accident and weather alerts, security alerts, and more; historically sending an average of 2 notifications per month. In addition to this, superiors need the ability to send targeted messages, via SMS text, messages rather than email, directly to their unit for high priority notifications that require fast action and short turn around times.

The past approach to issuing targeted SMS text message notifications relied on having contact information manually entered and stored on an Excel spreadsheet. When an alert needed to be sent, the Group Commander manually copied and pasted emails and phone numbers into Outlook for issuance via SMS text message. One limitation is that Outlook only allows about 1700 recipients per message while also requiring the manual entry of the cell phone carrier specific to each phone number. The legacy approach worked for alerts that happen once or twice per month, but the base now needs to scale to reach a dynamic and ever-growing roster of more than 3000 base personnel.

[Approach] - As of today, Luke AFB has launched PROJECT HERMES, a new flexible and scalable text message notification system to issue broad or targeted notifications to authenticated military personnel. PROJECT HERMES can issue notifications to all 3,000 credentialed personnel in just a few clicks and can scale to thousands more if needed while maintaining a constantly updated roster of verified recipients. This ensures that actionable notifications get to the right people at the right time. Outside of a reasonable fee to keep the platform ready, the cost of the solution is pay as you go, meaning only the messages sent incur cost.

[Customer testimonial] *"In the past, sending group text message alerts would take either too long to send or would never send at all due to the manual nature of the solution and limitations on the numbers we could text."* Said Wedge Antilles, Operations Manager at Luke AFB. *"Now, new personnel are directed to register for notifications at the new PROJECT HERMES web portal. From there base command can issue custom base wide text message notifications in just a few minutes and can even target specific subgroups of personnel based on their commanders. We have a*

governance process in place to make sure that the system doesn't get overused so personnel don't tune out the notifications."

[Customer experience] New personnel are required by their Group Commander to register for PROJECT HERMES upon arrival and orientation. Using the web portal they enter their basic information including name, rank, title, email and phone number. Their immediate commander is automatically notified of the sign-up and verifies the enrollment list to ensure that only the right personnel have access to the system. *"I received an emergency notification text the other day"*. Said Aayla Secure, Airmen E-2. *"It let me know that there was a hurricane headed to East Florida where I have family members. Thanks to this alert system, I was able to call them early to make sure they are ok and boarding up the house."* Each year, Aayla re-registers to receive notifications of alerts occurring on or off the base, which allows the system to have updated information as people change their deployment status or get promoted.

[Leader quote] *"Our communication systems are important for our day to day operations, when an emergency occurs, we need a quick way to alert many personnel at once."* Said Msgt. Biggs Darklighter. *"Having a system in place that can dynamically do this would massively improve our communication flow"*

[Call to action] – To learn more about PROJECT HERMES and to receive instantaneous alerts at any time of the day with a quick and easy sign up process, visit www.lukeafbmnotif.com

Frequently Asked Questions

User

Q: What kinds of alerts will I be notified about?

A: (1) Base-wide emergency messages such as Chemical Spills and Active Shooter (2) Commander's Calls where your direct leadership can message your subgroup directly.

Q: What do I have to do to sign up?

A: You will have to provide rank, name (first, last), section squadron group, telephone number, and military email address.

Q: Can I sign up with both my personal phone and my government issued device?

A: Personal device. Not all personnel have government issued devices.

Q: Can my spouse and kids sign up for this as well? They live on base.

A: No, we only want base personnel to receive these alerts.

Q: Will I still get an alert if I'm off base or is this a location based capability?

A: Yes, you can still receive a notification off base.

Q: Will I still get these alerts once 6 months from now after I re-deploy? I already get enough text messages and irrelevant spam.

A: No, each year we will reset our list and require all personnel to sign up again. That way contact info, deployment status, and hierarchy roles are updated regularly.

Q: How are you going to keep my contact information secure?

A: Only authorized personnel will be able to see your information. The information we store will be encrypted upon initial storage to provide an additional layer of security.

Q: Do I have to use my CAC card to sign up?

A: No you do not need the CAC card to sign up, just your us.af.mil email address.

Q: What other data are you collecting on me and do I have a choice in what is collected and retained?

A: No other information collected besides the ones listed above, our goal is to limit the information available to public knowledge.

Q: If I move off base, do I keep getting messages or can I unsubscribe

A: No you cannot unsubscribe, this is to ensure at least 75% of our personnel are registered and in the loop of potential events or threats.

Q: Can I unsubscribe if I no longer wish to receive these alerts?

A: Everyone is required to receive alerts.

Q: Do I need to filter out specific alerts myself?

A: No filtering system for the recipients, designated groups must receive their messages.

Q: What happens if a false alert is sent? How will that be handled?

A: We have a prompt before sending a message to make sure all information being sent is correct. This prompt will include all information given by the user to ensure they entered correct information. Our goal is to avoid false alerts.

Q: Can I submit an alert request if an emergency happens? How can we support this alert system?

A: Unauthorized recipients should notify those who are authorized and the word will travel quickly to personnel with the capabilities to send alerts. The goal is to keep this process simple and avoid false alerts.

Q: How will I know who is sending a message?

A: All messages will also include the subject of the message and the name of the sender following the message they sent.

Q: How long will an alert message be?

A: Most messages will be short and to the point (160 to 300 characters). However, the system supports longer messages should the sender choose to do so.

Q: Will I (as a recipient) need to reply to any of these messages?

A: No, not at this time.

Stakeholder

Q: How much control do you need over this new system?

A: As much control is possible with regards to configuration, message viewing, etc. while still utilizing a low-cost and low-maintenance system.

Q: What level of classification/privacy will you be sharing?

A: Not sharing classified information, mostly informative broadcast/emergency information. However, we will have the option to include more sensitive, yet non-classified information in the long-term.

Q: How much flexibility do you need and in what?

A: Adding new members, removing, changing roles, editing information, creating and reconfiguring the military hierarchy

Q: How often will you be using it

A: Roughly 1-2 times a month at the very least in the short-term, with the potential for more frequent use in the future.

Q: Is it only for on-base activities or do you need messaging capabilities during off-base times for off-duty officers?

A: Need messaging capabilities for all personnel at base even if off-base.

Q: What type of devices do you use for the input/output (sending and receiving)?

A: Government-issued computers and mobile devices for both sending and receiving.

Q: Do you want to be able to segregate messages by role type?

A: Yes, we would like the ability to send messages to specific roles.

Q: Do you want to have the ability to upload spreadsheets and automatically populate the recipient list?

A: Individuals have to be able to input their own information *without* the admins having to fill out an excel sheet.

Q: What types of roles do you need?

A: Specific roles like rank or existence within the military hierarchy (group, squadron, etc.)

Q: How many people are able to send, manage, change settings, etc?

A: Many user admins (20 - 60 people to send messages) as well as one root admin.

Q: Do you need a way to quickly see the history of what is sent/when/to whom?

A: Would like but not need at the moment.

Q: Will this be a public facing website? If so, how are we going to enable access through our firewall to government owned devices and networks?

A: As of right now the web-interface itself is public facing. However, from there participants need to log-in with credentials and provide verification codes sent to their military emails which adds many layers of security. The website itself will need to go through the ATO process to be accessible via the DoD internet.

Q: How do we prevent 'bad actors' from exploiting this? Can anyone sign up? What kind of user authentication is there?

A: There are multiple virtual "checkpoints" which provide multiple layers of security. Firstly, all users (people who are authorized to send messages) must (a) create an account with verification from their secure military email account and (b) get approval within the system from the root admin. All recipients will be required to register with a verification code sent to their secure military email account. Users can also kick-out recipients from the system. The back-end of the system utilizes Amazon Web Services GovCloud, a cloud computing platform which is FedRAMP High compliant.

Q: How do we get ATO (Authorization to Operate) and what does c-ATO (Continuous Authorization to Operate) look like once it is operational?

A: No insight just yet, it seems like there is ATO but c-ATO will be a bigger question down the line of development.

Q: What Impact Level (IL) is this?

A: The system meets requirements to handle IL 4 data, which include base personnel PII and hierarchy information.

Q: Will this integrate with any of our existing systems?

A: Completely new, no integration with an existing system.

Q: What is this going to cost? Who would own, operate, and maintain this?

A: Have to double check periodically as we go.

Q: What does access look like for local law enforcement/fire departments/etc. Do they need it?

A: No, other entities will notify the civilian sector, a system for police, fire is already set up. This system is meant for internal communications between base personnel.

The Current Solution

Our plan is to have a serverless web application backed by Amazon Web Services that will handle the messaging dynamically.

In Amazon Web Services, there are different types of accounts we could use:

- Commercial
- GovCloud

To provide better security, we believe that our development under the GovCloud is essential for the base's operations.

For the front-end, we will develop a React Web Application that will seamlessly facilitate the user's ability to send an alert message to thousands of base personnel in an instant.

Potential Solution:

- Web Application that is able to send messages to recipients.
- **Recipients** will be stored in a database.
    - Rank
    - Name
    - Section Squadron Group
    - Phone Number (Optional)
    - Email (Optional)
    - Must have either Email or SMS or both.
- Process an **authenticated user** (Group Commander for example) will go through:
    - Prepare and write alert message
        - Specify category of recipient (Everyone, Group, Rank, etc.)
        - Fill in message box
        - Send message
        - Receive confirmation of messages being sent
    - View Recipients
        - Recipients can be removed if not authorized
    - Modify/Add Categories to Recipients
        - Modify/Add Groups
        - Modify/Add Squadrons
        - Modify/Add ...

- Process the **application** will go through:
    - Receive the message from the authenticated user
    - Process who this message will be sent to
    - Select all the people this message will be sent to
    - As we go through each recipient
        - If email is provided, grab email
        - If phone number provided, grab phone number
        - If either is provided, give preference to SMS.
    - Once we grab all the data, we will send the message to the corresponding recipients who match the specifications given.
    - If errors present (someone didn't get a message), notify the user.
    - If no errors occurred, return a success message!

- Process a **recipient** will go through:
    - Register themselves into the system their af.mil email address
    - Verifying their phone number is correct
    - Once authenticated, provide information
        - Rank
        - Name
        - Group
        - Squadron
        - Phone Number(Optional)
        - Email (Optional)
        - Must have either Email or SMS or both.
    - Receive any messages if under the category/role specified by the message being sent.

Given the requirements and designed solution, the following are the implemented pages and features which will be displayed on the front-end React website. There are three account types, each of which has different access to the website and its functionality. The Root Admin has similar capabilities to the User account, the only difference being that there is only one Root Admin account and they have access to all information being passed through the system. This includes all logs, the ability to see/define/edit all hierarchy, and the ability to send a message to the entire base. There can be any number of User accounts and they have the ability to see their logs and those of any Users under their jurisdiction.

Root Admin: This is the main system administrator account for the entire base. There will be one (1) account which will be shared by a small group who will be locally managing the system.

1. MESSAGE: On this page the Root Admin can
    a. Select recipient groups they are looking to send their message to (as the Root User is the "top of the pyramid," they can send messages to everyone listed in the hierarchy)
    b. Type the subject of their message in a text box
    c. Craft their message in a text box
    d. Hit "submit" to send their message.
2. LOGS: On this page the Root Admin can
    a. See logs of all messages across the entire system that include the Date/Time, Sent by, Subject, Message, and Sent to
    b. Search through message logs using keywords
3. ADMIN: On this page the Root Admin can
    a. See all recipients under their jurisdiction (in the case of the Root User, that is everyone).
    b. Confirm information and kick-out recipients if need be
    c. Edit the hierarchy at any level (including adding/removing/merging categories)
4. LOG OUT

Users: These accounts are for anyone who is authorized to send a message. These are people in leadership roles (Group commanders, Section leaders, etc.) and there can be any number of User accounts. The role of Users is similar to that of the Root Admin, except Users can only send messages and see logs of individuals downstream of them in the hierarchy. They can also only edit the hierarchy downstream of them.

1. MESSAGE: On this page the User can
    a. Select recipient groups they are looking to send their message to
    b. Type the subject of their message in a text box
    c. Craft their message in a text box
    d. Hit "submit" to send their message.
2. LOGS: On this page the User can
    a. See logs of all messages under their jurisdiction that include the Date/Time, Sent by, Subject, Message, and Sent to
    b. Search through message logs using keywords
3. ADMIN: On this page the User can
    a. See all recipients under their jurisdiction
    b. Confirm information and kick-out recipients if need be
    c. Edit the hierarchy at any level downstream of them (including adding/removing/merging categories)
4. LOG OUT

Recipients: All Recipients who will be receiving messages via this system will be required to sign-up initially and re-register every year (this is after the database wipes itself. By having Recipients re-register annually, we can guarantee period information updates).

More information to come on this specific section.

Those pages will serve as the frontend of our application. Users and recipients looking to register will only see those pages if authenticated. Now, this a potential structure for the backend:

- AWS DynamoDB
    - Store all the user data. This may include
        - Recipients
        - Roles - They will give a list
        - ...

- AWS Simple Notification Service (SNS)
    - Sends a message to recipients in a database which is essentially a group of users/emails/phone numbers.

- AWS Lambda Functions
    - Handles the logic of an application infrastructure. Will process data, make necessary changes, and call other services to send this data.

- AWS API Gateway
    - Application to Application communication that will enable users to send messages through a web application rather than through the development console of Amazon Web Services.

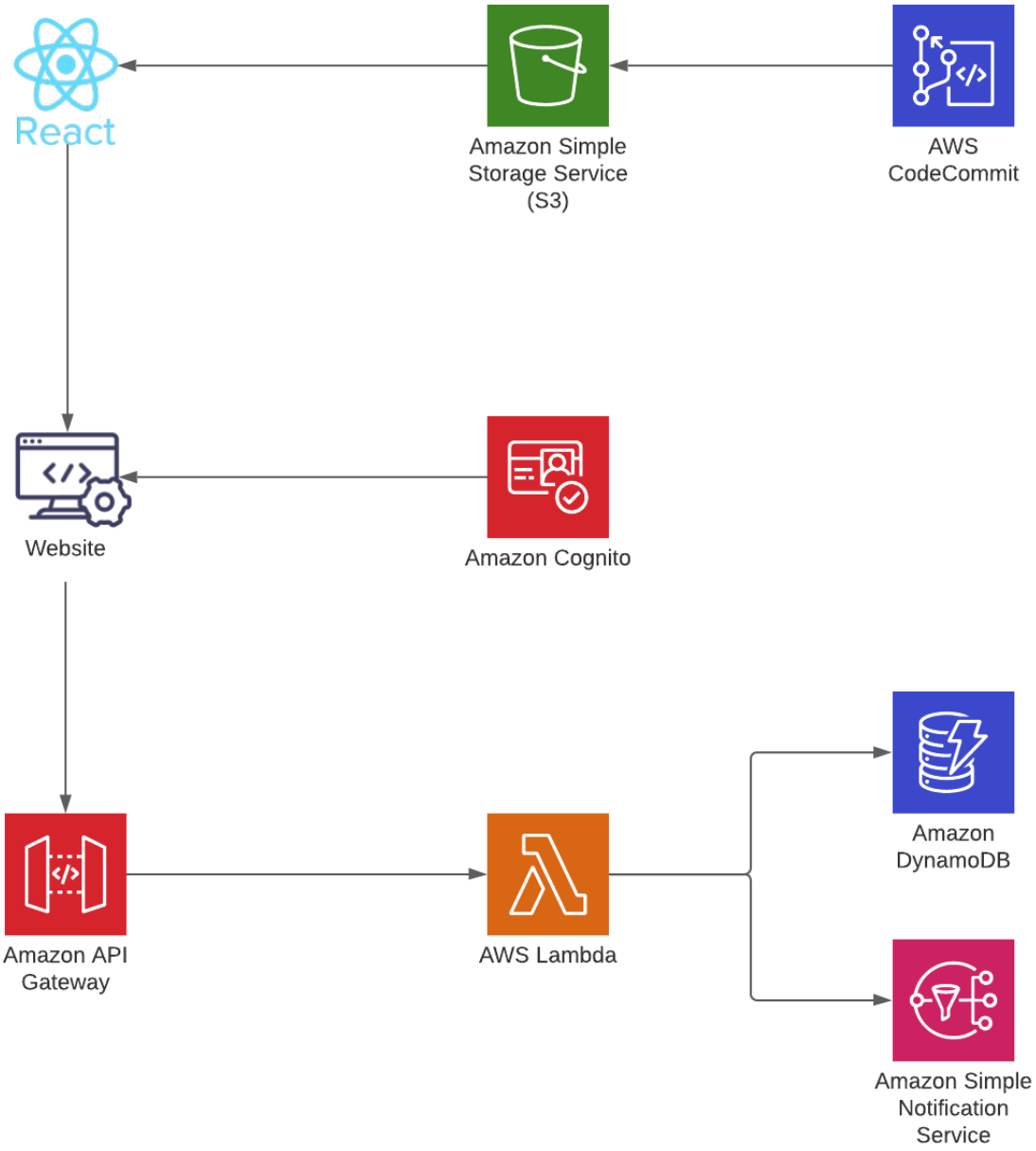
- AWS Cognito User Pools
    - Holds authentication information which will verify requests coming from a specific user.

- AWS S3 w/ React
    - Static web hosting
    - Set of tools and services that can be used together or on their own, to help front-end web and mobile developers build scalable full stack applications

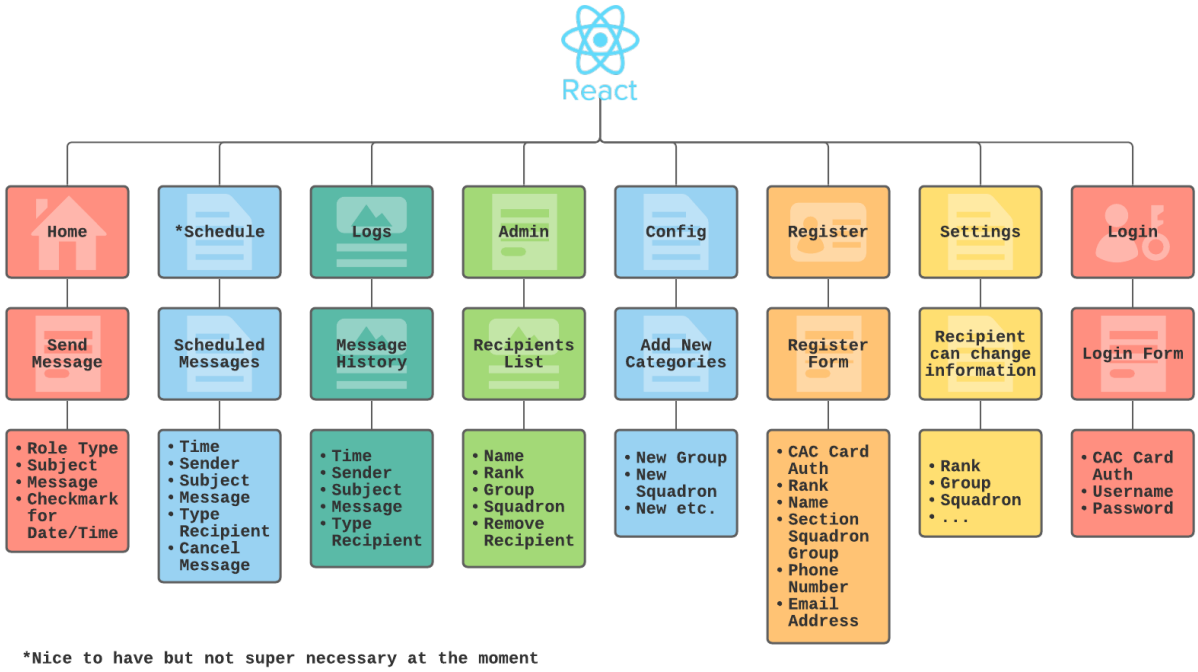
The Diagrams

Here are some diagrams that represent the foundation of the infrastructure we seek to create. The goal is to have deep specifications and a good understanding of the architecture before moving into the implementation phase. This will benefit us in being able to cover all edge cases and avoiding getting stuck in the middle of development.

Here is a general overview of the application architecture:



Here is an overview of the frontend which is the website. We will use the React js framework to create a seamless website that is easy to navigate and work with from the user's perspective. This framework will also work nicely with the backend implementation to further strengthen the message sending capabilities.



Home, Schedule, Logs, Admin, Config, Register, Settings, and Login are all pages that will be in the website, each of the boxes below them are components, specific blocks of elements that contain the logic and design of a bigger element like a form that has inputs for sending a message. Components make it much more efficient for us to create multiple elements at once with little code since it is object-oriented. This helps us in being more efficient by handling all elements from one source!

Complications

As we were developing our solution, specifically the front-end, the **Digital Transformation Hub (DxHub)** brought to light a big underlying issue in our implementation: Who is going to maintain this application and how will the data of recipients remain consistent?

In terms of maintainability, we will try our best to make the application as dynamic as possible in the sense that no coding will be needed if small changes need to be made. However, under the assumption that a service stops working or the application has an error, who will maintain this once this fellowship is over? That is one complication we need to address soon in order to prepare the right tools and documentation for Luke Air Force Base.

In terms of data consistency, one big road block is the type of data we are dealing with and how we go about storing/accessing this type of data. The current solution we have is to make all recipients register to the alert system each year by providing their Name, Rank, Group, Squadron, Email, and Phone Number. With the type of data we are requesting, this falls under the IL 4 data categorization under the Department of Defense's Cloud Computing Security Requirements Guide. But with this solution, we are essentially shifting the underlying problem.

Initially, the Group Commanders had to copy all the emails/phone numbers to send a message to the recipients. Now, recipients have to register to receive messages each year. While this new solution makes the commander's job easier, it will not be easier for those having to register and update their information. Here are some examples on how our solution can be inefficient:

- Recipient is deployed on January 1st, cannot unsubscribe because of the application's requirements of resetting the list of recipients each year, so they have to wait a year to be taken off the list.
- Recipients received a new rank, now they have to go into the application to update their rank.
- Recipient was moved to a different group, now they have to change that information in the application.
- Recipient was moved to a different squadron, now they have to change that information in the application.
- Recipient was moved to a different sub-group in the squadron, now they have to change that information in the application.
- Recipient was moved to a different group, squadron, and sub-group in the squadron, now they have to change that information in the application.

With these examples, it is apparent that with each new change in a recipient's personal information, the recipient has to manually go into the application to change and update that information. It's a possible task, but:

- Recipients might take a while to register to the alert system or won't immediately update their information once a change occurs.
- Some recipients might not even register to the alert system or won't update their information at all.

While the prior solution is possible, it is not ideal nor efficient as the work being done is dependent on the recipients who may or may not be updating their information or registering to the system. By making recipients the dependency of our system, we fall in a very awkward position as the application will not be useful if (in a worse case scenario) no recipients register at all.

Another complication our current solution brings is the idea of having outdated data. As soon as a recipient re-ranks, our system will automatically be outdated. Meaning, our system will always be one step behind Luke AFB's system. **The Digital Transformation Hub (DxHub)** brought this issue to light in our meeting and made us realize that there is far more potential for a solution that will truly solve the underlying problem of Luke Air Force Base's notification system. There are many roadblocks ahead, but we believe approaching the "dream" solution will truly provide an effective system to Luke Air Force Base.

The Dream Solution

The easter egg or dream solution that will make this system more efficient is to find a way to connect our application to Luke AFB's server or database that can provide up-to-date information on each of the recipients.

If we have some form of repository of information we could use, we will immediately have a much better solution and eliminate the problem at hand. Since we're already dealing with IL 4 data through having recipients register and provide their information, the only difference is the fact that we are accessing resources from Luke AFB to depend on rather than recipients.

This solution essentially shifts our dependency from recipients to Luke AFB's data which in our opinion could be more reliable. Here are some of the advantages the dream solution will bring:

- Up-to-date information on recipients.
    - No need for **Register Page**
    - No need for **Settings Page**
    - No need for **Config Page**
- No need for recipients to register at all.
- No need for recipients to update their information.
- No need to reset the recipients list each year.
- If a recipient re-ranks, our system will automatically be updated.
- If a recipient is deployed, our system will automatically remove them from our recipients list.

As you can see, no work would be needed from the recipient's side. In fact, the recipient wouldn't have to access the application at all. Our architecture would handle all the registration instead. In this case, the only ones who need to access the notification system are the commanders looking to send a message. This solves the problem for both parties as we completely automate the system. This does not mean the dream solution is perfect, there are still some measures to take to ensure our architecture is secure, operational, reliable, efficient, and cost-effective.

A roadblock is receiving ATO to handle this data. This will be our biggest and most challenging obstacle to get through. It will be our job to efficiently communicate our dream solution and provide the necessary specifications to ensure our Problem Sponsors and Luke AFB are confident in our provided solution.

If this solution cannot make it to the development phase, then we hope that at least the conversations we have with Luke AFB bring some light to the underlying issue.

AWS Well-Architected

For both our current and dream solution, we will aim to follow the AWS Well-Architected Framework, a framework that follows "*key concepts, design principles, and architectural best practices for designing and running workloads in the cloud.*" - AWS

This framework separates our architecture into five pillars:

- Operational Excellence
- Security
- Reliability
- Performance
- Cost Optimization

Operational Excellence focuses on running and monitoring systems to deliver business value, or in our case, military value. This pillar is all about automating changes, responding to events in the application, and defining standards to manage daily operations like sending a message.

Security focuses on the protection of information and systems. Confidentiality and integrity of data, privilege management, and protection of systems are key concepts that are useful in establishing best security practices.

Reliability focuses on ensuring our architecture performs its intended function correctly and consistently when expected to. This means our architecture should be resilient and quickly recover from failures to meet demands. Distributed system design, recovery planning, and how to handle change are key concepts in this area.

Performance Efficiency is heavily focused on using resources effectively. Being able to select the right resource types and sizes based on workload requirements, monitoring performance, and making informed decisions to maintain efficiency as needs evolve.

Cost Optimization is focused on avoiding unnecessary costs. Being able to understand and control where resources are being used will affect the way money is being spent on the overall infrastructure. Our goal is to select the right number of resources, analyze spending, and scale to meet the needs effectively.

Operational Excellence

In order for our architecture to be operationally excellent, we need to fulfill five design principles to ensure all key aspects of our architecture meet the requirements and best practices of AWS Well-Architected.

Operations as Code

We define our entire workload or architecture as code and update it with code. This will allow us to implement our operations procedures as code and automate its execution, limiting human error and enabling consistent responses to events.

We will do this by defining our whole architecture through **CloudFormation templates** in **YAML files**. By defining our architecture as code, we can immediately replicate, update, and even move our whole infrastructure in a matter of seconds. This is much better than having to manually implement these changes to our architecture.

Frequent, Small, Reversible Changes

This principle is about designing workloads that allow components to be updated regularly and be able to make changes in small increments that can be reversed if they fail. To achieve this principle, we will use CodeCommit to store our application code and CloudFormation Stacks for our architecture structure. Through these services, we will be able to make frequent changes in small increments.

CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. This will give us the ability to make small frequent updates or changes to our application code and also allow us to reverse changes if things fail.

CloudFormation is a service that will help us define and model our AWS architecture through code-based templates so that we don't have to create and configure individual resources one at a time.

Refining Operations Procedures Frequently

Looking for opportunities to improve our operations procedures is critical. As we evolve our architecture, we need to evolve our procedures as well. We can do this by running testing phases to validate all procedures and make sure all parties are familiar with them.

Anticipating Failure

Another critical principle is to identify potential sources of failure. We can do this in our testing phase by performing “pre-mortem” exercises where we test our failure scenarios and validate the impact of each.

By testing our response procedures, we ensure effectiveness and familiarity with each potential source of failure so that they can be removed or fixed.

Learning From All Operational Failures

Within each failure in our operational events, we will document and share what is learned so that this failure cannot be replicated in the future. Some examples may include:

- Edge case that threatens authentication.
- Edge case that threatens message sending capabilities.
- Edge case that threatens scheduling or log viewing.

Security

In order for our architecture to be secure, we need to fulfill seven design principles to ensure all key aspects of our architecture meet the requirements and best practices of AWS Well-Architected.

Strong Identity Foundation

Both our architecture and application will use the principle of least privilege and will enforce separation of duties. Least privilege is a principle where a user or service only has the necessary permissions needed to perform their tasks.

This means that when an user is created, that user will not have permissions whatsoever until specific roles or permissions are given. In the case of our AWS architecture, those needing access to an AWS account will be given an IAM Account with no permissions. Until an administrator gives permissions to this new account, the user will not be able to perform any actions. In the case of our application, when a commander is added to send messages, they will not have any permissions to do so until verified and approved by the higher up who created the account.

Enabling Traceability

Traceability will give us and Luke AFB the ability to monitor, alert, and audit actions and changes to our environment. Through the AWS CloudWatch and AWS CloudTrail services, those who have access will be able to receive logs and metric collections to view any activities in real-time. Some examples of this may be:

- A service like Lambda functions are continuously running when not supposed to be. CloudWatch will show this activity.
- A specific API is receiving too many unexpected calls. AWS CloudTrail will show a history of API calls being made to our account, hence giving us the ability to determine the source of this event.

In our application, the Logs page will show the administrators or commanders all the messages being sent by the subordinates. This will ensure that commanders can view the activity of their subordinates and be able to determine the cause of a specific event like:

- Too many messages are being sent from the account of a subordinate.
- An unauthorized user appeared and is sending malicious messages.

Security At All Layers

Our goal is to have both our architecture and application configured to have defense in depth with multiple security controls. In the case of our AWS architecture, we will ensure each of our services are authenticated correctly, have no security risks, and are only accessible by those who are authorized to do so.

In the case of our application, Multi-Factor Authentication will be required to access the dashboard. When logging in, the commander will have to provide their username and password. Once successful, they must provide their @af.mil.us email address, input the verification code sent to that email, and if successful they will be able to access the application.

Automating Security Best Practices

Some of our security mechanisms will require automation to avoid human error. This may include authentication, logs, or even the viewing of recipients lists. It is our goal to create a secure architecture that is able to have an implementation of controls that are defined and managed as code in version-controlled templates (CloudFormation).

Protecting Data in Transit and at Rest

All data needs to be categorized into sensitivity levels in order to provide the appropriate level of protection to ensure each category is secure and accessible.

In our case, we will be using Impact Levels to categorize our data based on the DOD Cloud Computing Security Requirements Guide:

- **Impact Level 2: Non-Controlled Unclassified Information**
    - All data cleared for public release.
    - Low Confidentiality & Moderate Integrity
    - Information not designated as Controlled Unclassified Information.
- **Impact Level 4: Controlled Unclassified Information**
    - Accommodates CUI and/or other mission critical data to include that is used in direct support of military or contingency operations.
    - CUI is information the Federal Government creates or possesses that a law, regulation, or Government-wide policy requires, or specifically permits, anb agency to handle by means of safeguarding or dissemination controls.
    - CUI requires protection from unauthorized disclosure.

Data we store on military personnel such as Name, Rank, Group, Squadron, Email and Phone Number is PII and under the DOD Cloud Computing SRG, it's considered Impact Level 4.

Data we store on communications such as messages being sent from commanders are told to be under Impact Level 2 due to the low confidentiality in the content of the messages. However, this requirement needs to be established in order to avoid highly sensitive data being transmitted to base personnel from commanders.

These are the two types of most-significant data we will be storing, messages sent and recipient information. One potential type of data is the hierarchy of Luke AFB as our level of authentication aims to follow this hierarchy to define what and who a user can send messages to. Our belief is that this type of information would also fall under Impact Level 4.

Based on these impact levels, our goal is to encrypt, tokenize, and define access control policies where appropriate so that this data is only accessible by those who are authorized. IL 2 and IL 4 may have different security needs, but all should be dealt with carefully to ensure that all data is secure.

Keeping People Away From Data

Our goal is to limit the need for users to access data straight from a database or other data storage service. Our application will have tables, dashboards, and configuration functionalities so that users can access the data they need without needing to look at a database when needing to send a message, look at logs, or change their groups.

By having these mechanisms and tools in place, we eliminate the need for direct access or manual processing of data and avoid the risk of mishandling or modifying raw data needed for our infrastructure to function properly.

Preparing for Security Events

In the case of an incident, it is our goal to prepare incident management and investigation policy and process procedures so that we can guarantee an efficient incident response.

No architecture or application is 100% secure, but with the right incident response plan, the probability of securing and strengthening an infrastructure is far greater than expected.

Reliability

In order for our architecture to be reliable, we need to fulfill five design principles to ensure all key aspects of our architecture meet the requirements and best practices of AWS Well-Architected.

Automatically Recovering From Failure

Recovering from failure is a critical concept of a reliable architecture. Throughout our architecture, we aim towards identifying key performance indicators in each of the services we use to trigger automation when an event occurs.

Doing this will allow for automatic notification and tracking of failures. An example of this may include:

- Setting alerts in CloudWatch to alert users when a threshold is being exceeded.
    - Sending messages
    - Too many requests
- Checking for errors in lambda functions and reporting those errors to the console for a developer to identify and fix

Our architecture could be far more reliable if we can anticipate failures and automatically fix them before they even happen.

Test Recovery Procedures

Testing is typically done to ensure an architecture/application is working properly under ideal conditions, but never to test its recovery procedures. Our goal is to automate and simulate different failure scenarios in order to test and validate our recovery procedures so that we can expose failure pathways and fix them before a real failure scenario occurs.

Scaling Horizontally

Scaling horizontally means dividing up a large resource into smaller resources to manage and avoid major failures in our architecture. Rather than having one big resource handle most if not all of the functionality of the application, we can split it into smaller microservices to ensure that a failure in one service does not affect the other services. Thus, increasing reliability in our architecture.

Not Guessing Capacity

Guessing capacity can be a big point of failure in specific architectures since miscalculating the necessary resources could result in either an abundance of resources (expensive) or in a shortage of resources (slow application).

To avoid this, our goal is to monitor and set specific optimizations in our architecture to handle events efficiently without a waste or need for resources. We can do this by setting specific threshold policies on specific services in the case of a big spike in traffic and defining the optimal or low number of resources needed when traffic is low.

Managing Change in Automation

Most if not all of our changes in our infrastructure will be automated to avoid human error. However, when changes to our automation need to be done, we will do this manually and track these changes so that no errors are present.

Performance Efficiency

In order for our architecture to be efficient in performance, we need to fulfill five design principles to ensure all key aspects of our architecture meet the requirements and best practices of AWS Well-Architected.

Democratizing Advanced Technologies

Using new technologies as a service provides us the opportunity to focus on our application development rather than resource provisioning and management.

Through AWS, we are given the ability to consume technologies as a service and leverage these technologies to build our architecture faster than on an on-premise architecture. Thus providing us with IT resources that are much more efficient than what a college student could develop!

Going Global in Minutes

In AWS GovCloud, there are 2 regions we are able to use for developing and hosting our architecture: us-gov-west-1 and us-gov-east-1. To ensure that our infrastructure remains reliable and efficient, our goal is to deploy our workload in us-gov-west-1 to provide lower latency and a better experience for the Luke AFB personnel. In the event of a failure in us-gov-west-1, we can also store our infrastructure in us-gov-east-1 to ensure availability at all times.

Using a Serverless Architecture

Developing a serverless architecture removes the need to maintain and run physical servers for traditional compute activities. It also benefits in having a microservice-based architecture that is not reliant on one specific resource.

Our main goal is to separate different functionalities of our application into separate services. By having separate independent services, we remove the operational burden of managing physical servers while simultaneously lowering transactional costs as we leverage services that operate at cloud scale.

Experimenting More Often

Due to the abundance of services we can use for our architecture, we get the ability to experiment and determine which services work best for our infrastructure.

As we develop, we will explore and make the necessary changes to ensure the services we are using have the right functionality for our needs. An example of this is the choice of using either AWS DynamoDB or AWS Relational Database Service for our data storage. Both are similar in the sense that they are databases, but each has their own unique set of functionalities that will either benefit or hinder our workload's performance.

Considering Mechanical Sympathy

By using the technology approach that aligns best with our workload goals, we enhance our understanding of how cloud services are consumed and how each service could best support us in our efforts.

The way we will do this is by asking the question "What functionality do we need and what limitations do we not need?". By asking this question, we identify what type of functionality we need for our application, we find a cloud service that can fulfill this need, and then we explore the limitations of said service to determine whether this service is ideal for our infrastructure or not.

If the service turns out to be too limited, we can look elsewhere for a service that fits our needs. This in turn reduces the risk of having to completely reshift our architecture in the middle of our development when we find out that the functionality we needed was never provided by the service we chose. Now that is not fun!

Cost Optimization

In order for our architecture to be cost optimized, we need to fulfill five design principles to ensure all key aspects of our architecture meet the requirements and best practices of AWS Well-Architected.

Implementing Cloud Financial Management

Our mission is to reach financial success by optimizing our architecture to be as cost efficient as possible. Through the use of services like AWS Cost Explorer, we can manage and view each service and determine inefficiencies that are threatening our cost performance.

Adopting a Consumption Model

During our development and testing phase, it is necessary that we adopt a strategy for not consuming resources inefficiently. In our case, sending unnecessary messages or even having a bug that infinitely sends messages could threaten our budget.

As we progress through our development and testing, we will monitor, fix, and reduce any cost inefficiencies that arise in each of the services we use.

Measuring Overall Efficiency

Once our solution is fully functional, we will measure the overall effectiveness of our application on a financial basis so that we can find better ways to implement our solution and reduce unnecessary costs. This will be a big part of our testing phase to make sure nothing is out of place in the functionality and cost of our workload.

Avoiding Spending Money on Undifferentiated Heavy Lifting

With the leverage of cloud services. We can focus more on the optimization of our architecture and cost efficiency of our application. With this, we benefit from not having to buy or sell additional servers we did or did not need.

Analyzing and Attributing Expenditure

Different AWS services provide expenditure and budget management functionalities for account owners to use. With these services, we can measure the expenditures of each individual service and determine whether a service is being cost effective or not.

One of our goals is to help facilitate this process for Luke AFB so that they can manage these costs and also identify areas where a lot of money is being spent unnecessarily. Through effective documentation, we believe this is possible!