Model of a Subsystem for Securing E-Mail Against Loss using Mail Transport Agents based on Containerized Environments

Bohdan Leshchenko[1], Tetiana Vakaliuk[2,3,4], Andrii Yefimenko[2], Viacheslav Osadchyi[5], and Dmytro Antoniuk[2]

[1] *Sana Commerce Ukraine, 20 Capitulnyi lane, Zhytomyr, 10003, Ukraine*
[2] *Zhytomyr Polytechnic State University, 103 Chudnivsyka str., Zhytomyr, 10005, Ukraine*
[3] *Institute for Digitalisation of Education of the NAES of Ukraine, 9M. Berlynskoho str., Kyiv, 04060, Ukraine*
[4] *Kryvyi Rih State Pedagogical University, 54 Gagarin ave., Kryvyi Rih, 50086, Ukraine*
[5] *Borys Grinchenko Kyiv University, 18/2 Bulvarno-Kudriavska str., Kyiv, 04053, Ukraine*

Abstract

The article describes the development of a functional model and the application of an enterprise email system security subsystem. This addresses the problem of email delivery and ensures the prevention of email loss. The article presents the results of a case study on a company's infrastructure, identifying gaps in the existing system. It discusses the selection of an infrastructure and cloud provider, considering the benefits of containerization and cloud computing. The process of selecting a mail authorization server tailored to the organization's unique email requirements is explored. The article's conclusion underlines the usefulness and significance of the developed method for guarding against email loss in businesses that rely largely on email to effectively communicate with clients and partners.

Keywords

Email security; email reliability; containerized environments; cloud-based email services.

1. Introduction

Email is an important communication tool in today's digital world, and reliable email delivery is essential for the efficient functioning of businesses and organizations. However, periodic outages of Internet DNS services and interruptions in the operation of email service providers emphasize the need for organizations to have their email delivery infrastructure [1, 2].

The loss of emails while they are being sent from containerized environments is a serious problem that can significantly impact communication efficiency and business processes [3, 4]. Despite the growing popularity of containerization, email providers are not completely reliable, and this can lead to the loss of emails on the way from the application to the email provider.

This problem has important implications for many organizations that rely on email to communicate with customers, partners, and internal departments. Losing emails can lead to data loss, missed deadlines, poor stakeholder communication, and ultimately financial loss [5].

However, the solution to this problem is not so simple. Developing a mechanism for creating a mail queue in an application is not cheap and involves certain technical challenges. In addition, containerized environments may contain applications whose source code is not accessible, which complicates the process of identifying the causes of lost emails.

CPITS-2023-II: Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2023, Kyiv, Ukraine
EMAIL: zogyyy1@gmail.com (B. Leshchenko); tetianavakaliuk@gmail.com (T. Vakaliuk); yefimenko.andrii@gmail.com (A. Yefimenko); v.osadchyi@kubg.edu.ua (V. Osadchyi); dmitry_antonyuk@yahoo.com (D. Antoniuk)
ORCID: 0009-0001-5781-3518 (B. Leshchenko); 0000-0001-6825-4697 (T. Vakaliuk); 0000-0003-2128-4797 (A. Yefimenko); 0000-0001-5659-4774 (V. Osadchyi); 0000-0001-7496-3553 (D. Antoniuk)

The goal of this study is to come up with a method to stop emails from being lost when sent from containerized environments and to address this issue. We propose to conduct research aimed at developing an effective mechanism that will reduce the risk of email loss. The study will analyze the existing problems of email delivery, technologies, and methods of preventing email loss, taking into account the technical limitations of containerization.

1.1. Theoretical Background

Emails have become an effective initial vector of infection because virtually every company uses email, and employees receive a large volume of emails every day. Because of the overwhelming volume of emails, employees have very little time to review and analyze each one, which might give them a false sense of security. Cybercriminals are effectively exploiting this situation by organizing phishing attacks, which are becoming even more widespread and sophisticated with the advent of cloud-based email services. Consequently, email security is becoming a critical task for businesses and employees to avoid the consequences of these evolving threats.

Simple Mail Transfer Protocol (SMTP) [6] is the most used protocol for sending and receiving email messages on websites today. While Sendmail has traditionally been a popular choice for SMTP servers, it has faced various issues over the years. The monolithic architecture of Sendmail has been a major contributor to security vulnerabilities, making it challenging to configure and maintain.

In response to these shortcomings, Postfix was developed as an alternative to Sendmail. Postfix is designed to address many of the security concerns associated with Sendmail. Additionally, Postfix simplifies the management of email server installations by employing a straightforward approach. Administrators can conveniently handle Postfix through two configuration files, reducing complexity.

One of the standout features of Postfix is its ability to function effectively even in demanding situations. It is not uncommon for software systems, including email servers, to face unexpected conditions such as running out of memory or disk space. Postfix, however, excels at detecting and handling such conditions without worsening the problem. This robustness ensures stable and reliable email operations, even during challenging circumstances.

Of course, like any system that started a long time ago, the email system has evolved alongside and in parallel with the development of the entire Internet.

So, just as the infrastructure of enterprises has been changing first towards hydride and then towards fully cloud-based, the infrastructure also needs to evolve. As of today, more and more applications are migrating to Kubernetes clusters hosted in different cloud environments, which means that not only the applications themselves have to be redesigned for such a containerized environment, but also all the accompanying software.

1.1.1. Principles of Organizing E-mail Exchange

The main purpose of e-mail is to exchange messages (information that is delivered asynchronously). The main messaging protocol is SMTP, which aims to ensure reliable and efficient exchange of letters.

The SMTP protocol is used to send all email in the world. According to official documentation [7], the process of transferring information using this protocol is quite simple. A visual representation of the protocol's scheme is provided in Fig. 1.

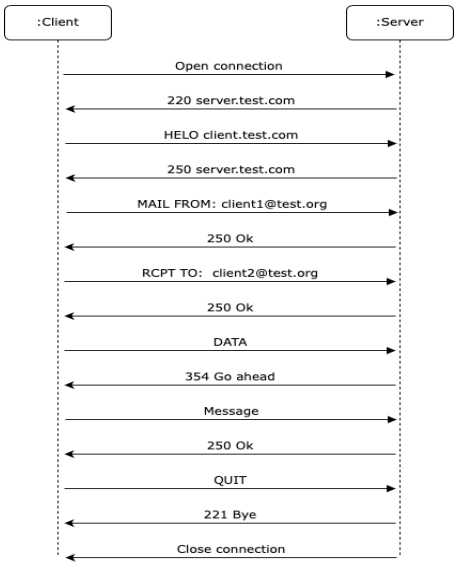

Figure 1: Operation of the SMTP protocol diagram

The network on which SMTP is implemented consists of hosts that communicate using the TCP protocol and operate on the public Internet, an isolated internal TCP/IP network, or other local or wide area networks using another transport layer protocol other than TCP. With SMTP, a process can transmit electronic messages to another process on the same network or a different network using a relay or gateway that is accessible to both networks.

Thus, an electronic message can pass through several intermediate relays or gateways on its way from the sender to the final recipient. This ensures efficient routing and delivery of emails to their destination [7].

Computer programs that allow you to create, send, receive, and view emails are called email clients. Users can access email using an interface made by an email client. Both locally on a user's device and remotely on a web server are capable of running email clients. For the first case, examples include Microsoft Outlook, Apple Mail, and Mozilla Thunderbird—these programs are installed directly on the user's computer or mobile device and provide the ability to manage email without a direct connection to the Internet.

In the case of web servers, examples of email clients are Google Gmail and Yahoo! Mail—these services are provided in the cloud and are accessible to users through a web browser. In particular, a user can access his or her email from any device with an Internet connection without having to install separate software on each device.

The Simple Messaging Protocol system consists of many different elements, the main elements of the system are shown in Fig. 2.

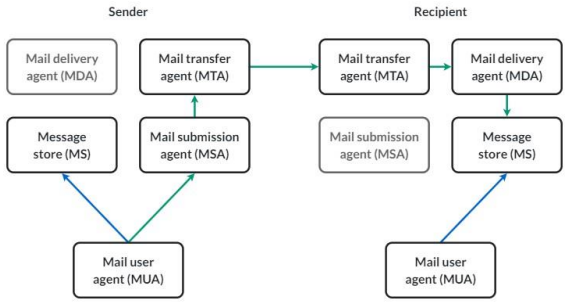

Figure 2: The basic architecture of email [8]

Separate operations that are in charge of sending and receiving mail are necessary for an exchange system to function successfully. The incoming mail agent and the outbound mail agent must typically run on separate servers in large commercial enterprises.

To send messages to recipients and to receive new messages from the user's mailbox, the mail client creates two connections: one to the outgoing mail server and one to the incoming mail server.

The email client connects to the incoming mail server and the outgoing mail server using various interfaces. For communicating with particular mail agents that are used to process incoming and outgoing emails, these can be distinct protocols and ports.

This separation of functionality between servers and different interfaces allows for efficient and reliable email management and ensures optimal interaction with email clients.

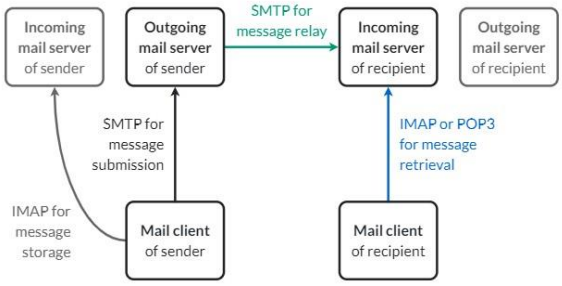

Figure 3: The recipient's mail client establishes a connection with the incoming mail server using a distinct port and protocol compared to the outgoing mail server [8]

The outgoing mail server serves several important functions:

1. Message Queue: It receives messages from mail clients and places them in a queue for further delivery.
2. Routing: The outgoing mail server determines the appropriate incoming mail server for each recipient and transfers the messages to that server for final delivery.
3. Client-Server Interaction: Acting as a server, it interacts with mail clients when receiving messages, but functions as a client when relaying messages to incoming mail servers.
4. Connection Initiation: The outgoing mail server ensures that connections are always initiated by email clients.
5. Bounce Messages: If the outgoing mail server is unable to deliver a message, it sends a bounce message back to the user who sent the original message.

6. Sender Information: It adds information about the sender, but does not alter the content of the message.
7. Authentication: Before accepting a message, the outgoing mail server typically authenticates the user, often requiring a username and password [9].

An incoming mail server operates by listening for connections from outgoing mail servers. When an outgoing mail server connects to send a message, the incoming mail server records the message and session details, including the sender's IP address [10].

Incoming messages can be rejected by an email server for various reasons, such as:

- Non-existent email recipient.
- Full mailbox of the recipient.
- Message size exceeding the server's limit.
- Unreliable or inauthentic sender.

In case of rejection, the incoming mail server may try to resend the message later or notify the user of the failed delivery.

Once the incoming mail server receives a message, it takes charge of its subsequent delivery. If there are any issues with delivery, such as the need for forwarding, the incoming mail server informs the message sender.

After concluding the session with the outgoing mail server, the incoming mail server includes the gathered session details in the received message. It then assesses the message to determine if it is spam or legitimate. Depending on this assessment, the message may be:

- Delivered to the recipient's inbox if it is deemed legitimate and does not contain any spam indicators.
- Moved to the recipient's spam folder if it is detected as unsolicited content.
- Deleted without notification to the sender if the inbound mail server is convinced that it is spam and should not be delivered.
- The use of filters helps to identify spam and other unwanted messages, as well as generate automatic responses in certain situations [10].

For authorization, the email client provides identification data, such as an email address and password. The inbound mail server verifies this information to ensure that it corresponds to a valid user and matches the data in the database correctly.

In some cases, the inbound mail server may support an alternative authentication method such as OAuth. This allows the email client to obtain a special access token that allows access to only certain functions or messages within a limited scope. This approach gives users more control over which applications or services have access to their mailboxes [11].

After studying the principles of exchange, we move on to study the problems of secure mail sending.

1.1.2. Researching the Problems of Secure Mail Sending

Classifying messages as spam based on their content and origin is an important task in spam filtering systems. Probabilistic classifiers, such as naive Bayesian spam filtering [12], use statistical models to calculate the probability that an incoming message is spam.

To make a classification decision, it is necessary to convert the continuous probability into a binary value. If the probability exceeds a certain threshold, the message is rejected as spam. This approach allows you to divide messages into two categories: spam and non-spam (legitimate messages).

When classifying spam, it is important to ensure that the false positive rate (classifying legitimate messages as spam) is close to zero. This helps to avoid users missing important messages. On the other hand, the false negative rate (classifying spam as legitimate messages) can be somewhat higher to prevent spam from getting into the inbox [13].

To determine a message's authenticity and potential delivery to the recipient's inbox, the message's origin holds significant importance. When sender domain authentication is unavailable, the message's delivery is reliant on the IP address's reputation.

Social engineering based on exploiting people's gullibility has become a common technique. In addition, phishing attacks often use spoofed sender addresses, including sending emails to victims that look like they came from their addresses. Also, spoofed email addresses can be misused in the context of email authentication, for example, when trying to unsubscribe from mailing lists [14]. Thus, spoofing emails makes it possible to bypass

restrictions and spam mailing lists. Preventing such attacks and protecting emails are becoming important tasks to ensure the safety and reliability of users.

Modern email protocols are designed to ensure confidentiality, authenticity, and integrity during message transmission without the need for user intervention. The use of Transport Layer Security (TLS) in IMAP, POP, and SMTP protocols allows you to ensure the confidentiality of messages even when an attacker acts in the middle, trying to capture information. Additionally, the DomainKeys Identified Mail (DKIM) mechanism along with domain name system security extensions guarantees the authenticity and integrity of messages [15].

Support for TLS is common, but certificate validation is not always used, which can compromise protection against active attacks by an attacker [15]. However, the use of Sender Policy Framework (SPF) and DKIM is also common, which helps reduce the risk of receiving spam and fake messages. These technologies help improve email security, but you need to make sure they are properly configured and used to maximize user protection.

2. Results

2.1. Description of the Company's Infrastructure and Identification of Gaps in the Existing System

The case study enterprise, from the very beginning of its application development, had a distributed infrastructure, and applications were hosted on physical servers in various data centers. With the development of cloud computing, application orchestration, and containerization technologies, the company began moving its infrastructure to containers hosted by cloud providers.

At present, the company has not implemented a mechanism for storing and processing e-mail, which in turn leads to its loss and potential interception by a third party.

In this case, the loss or transfer of information or the contents of the emails is critical, as the emails contain information about orders placed and other sensitive information.

We have a situation in which the main applications have already been moved to the cloud, but emails are sent directly to recipients without proper protection and minimal operations to hide the content of emails. The main problem, of course, is the loss of emails in the chain between the application and the recipient. The main task is to develop a cluster-embedded solution that combines encryption, message queuing, and logging of the number of sent and unsent messages.

Since there are a large number of applications in the infrastructure, auditing how many users have sent emails is also a critical factor. At the moment, an end user who sends only a few emails and a user who sends hundreds of thousands of emails pay the same because there is currently no control mechanism.

2.2. Selecting an Infrastructure and Cloud Provider

A container is a standardized unit of software that stores code along with all its dependencies and allows it to be easily ported and run in different environments standardization by including dependencies means that you will get the same behavior wherever you run it [16].

Transform your monolithic code into lightweight modules to enhance manageability and connectivity. By doing so, you can avoid the risk of one small module breaking your entire program. This approach grants you more precise control over your code, but it does introduce multiple moving parts to your platform.

However, managing numerous moving parts can be challenging. When one container connects to another, it becomes essential to remember to update each one for platform stability. With dozens of containers, this can lead to code management complexities.

Kubernetes greatly simplifies the process of deploying containers and helps to reduce a large number of costs associated with it. Thanks to it, deployment periods can be reduced from a full day to a much shorter time, as Kubernetes automates code compilation, testing, and checking for updates to all services. Previously, there were several automation tools to help with the deployment process, but they were mostly designed for

monolithic architectures, while Kubernetes provides more efficient and scalable container management. Orchestration and automation offer solutions to the challenges encountered when deploying infrastructure manually. With these processes, you can eliminate the risk of forgetting files, ensure updates are applied across all servers, and easily undo changes as needed [17].

One distinguishing factor between Kubernetes and other deployment options is the continuous nature of container deployment. There's no need to wait for sequential compilation and deployment of binaries. Instead, Kubernetes constantly incorporates new changes into containers and deploys them in the background. This enables swift code deployment without disrupting performance during specific periods. With Kubernetes and containers, there's no need for developer intervention to manage services that require frequent updates.

Kubernetes and containers are well-suited for cloud environments due to their portability and lightweight nature. They can be deployed on cloud platforms as well as local servers, making them versatile for a multi-cloud strategy across various service providers. Containers offer an attractive option for reducing risk when implementing a microservice architecture in cloud computing.

When considering the best cloud hosting service out of the three options (AWS, Azure, GCP), it is important to analyze the benefits and drawbacks of each. AWS, being the leader in the cloud hosting market, should be given priority.

AWS offers three container environments: ECS, EKS, and Fargate. If developers have limited experience with containers and already use AWS to host their services, ECS is the recommended option. This "containers-as-a-service" solution automates deployments directly in the cloud using Amazon AWS CloudFormation, making it a great starting point to determine if containers are suitable for an organization.

For a more comprehensive use of Kubernetes and containers, Amazon EKS is a suitable choice. EKS can move an existing on-premises Kubernetes deployment to the cloud. Advantages and surveys show that EKS is positioned to become the most popular container management method, with 63% of container users surveyed by Kubernetes preferring AWS.

AWS Fargate, Amazon's latest release for container users, enables container deployment without the need to manage servers or clusters. Fargate also works with AWS EKS, providing multiple options and combinations based on individual needs.

Hosted Kubernetes with AWS is particularly attractive for developers who are new to the container environment. It serves as an excellent starting point to experiment with containers and gauge their compatibility with the development environment. However, EKS is considered difficult to set up and requires technical experience with containers. On the plus side, it offers full scalability and configurability, empowering companies to control Kubernetes and its integration with local development processes [18].

Let's now analyze the advantages and disadvantages of Microsoft Azure. IT professionals primarily working with the Windows software environment will find deploying to Azure intuitive and straightforward. Although Azure is a relatively new container service, launched in 2015, Microsoft continues to enhance its offerings.

While Microsoft is known for its Windows operating system, Azure can also work with various Linux and Unix distributions. This means that it is not limited to Windows-only applications, but it does have limitations regarding hybrid container support, unlike AWS, which does support hybrid deployments.

Azure introduced Azure Kubernetes Service (AKS) in October 2017, a service similar to AWS EKS. Deploying AKS on an Azure virtual machine has the added advantage of being free, with payment only required for the resources used on Azure's virtual machines.

The main disadvantage of Azure is that while AKS was introduced before AWS EKS, Kubernetes is more adapted to AWS and GCP. As a result, Azure lags behind both AWS and Google's Kubernetes engine when it comes to updating to the latest versions of Kubernetes. However, if Azure is already a part of your existing architecture and you need to implement containers, it simplifies deployment and provides detailed analytics to assess if the platform meets your needs. Additionally, Azure offers a competing service

to AWS ECS called Service Fabric, which is also worth considering.

Now, let's analyze the advantages and disadvantages of Google Cloud Platform (GCP). Given that Google is the original creator of Kubernetes, working with GCP provides several advantages. Any new versions and deployments are immediately available, while other platforms may have a time lag. Moreover, GCP offers excellent opportunities for working with big data, machine learning, and Artificial Intelligence (AI) technologies.

However, GCP faces challenges as it is not as popular as AWS or Azure in terms of infrastructure as a service solution. GCP lacks the small business cloud offerings that make AWS and Azure more appealing for corporate integration into internal networks. Additionally, GCP does not have features like Active Directory integration (Azure) or Identity and Access Management (AWS).

Google promotes its platform as the ideal choice for working with the DevOps methodology. DevOps teams consist of specialists who simultaneously work on automating deployment and application development tasks. For such teams, GCP offers advantages for automating deployments.

Comparing costs becomes difficult due to the pay-as-you-go pricing policies of these platforms. The price an individual developer experimenting with Kubernetes and containers pays will not be the same as what an enterprise requires for powerful computing resources. Costs also depend on the resources utilized, with each platform having a minimum cluster allocation [19].

Ultimately, there is no clear-cut answer regarding the best service to use with Kubernetes. The choice depends on the specific project requirements. If you are already experienced with Azure or AWS, it is advisable to stick with the same platform. However, if you need to work with AI and machine learning, GCP offers an attractive and cost-effective solution.

2.3. Selecting a Mail Authorization Server

Organizations have different email needs. While some businesses only need a way to send straightforward marketing newsletters, others need a more sophisticated email infrastructure for high-volume and transactional uses.

Let's now examine the technical details of SendGrid and Mailgun, the two top email platforms. Both systems provide great ways to communicate large amounts of data, complete transactions, and guarantee first-rate delivery quality.

Being the "Email Service for Developers," Mailgun offers robust APIs that make sending, receiving, and tracking emails simple. Mailgun provides a sophisticated API, Mailjet for marketing, email verification, bulk sending, delivery issue prediction, and other services to over 200,000 enterprises with a 99.99% uptime guarantee.

Let's now evaluate SendGrid's advantages and disadvantages. SendGrid, one of the top transactional email systems on the market, also provides email marketing capabilities including user-friendly registration forms and design templates. SendGrid, which has more than 80,000 customers' trust, enables the safe delivery of more than 70 billion emails each month.

Since its founding in 2009, SendGrid has maintained an advantage in transactional email services. They stand out from the competition because of their emphasis on availability, scalability, and considerable email experience. Clients that use SendGrid's services enjoy an average delivery rate of 97%, exceeding the sector average of 85%, according to the company's data [20]. This reflects their service's high level of effectiveness and dependability in assuring successful email delivery.

To counter phishing and spoofing attempts, SendGrid provides sophisticated authentication with SPF and DKIM. Furthermore, they include AI-based technologies like their adaptive communication engine, which makes use of AI to improve availability and react to changing ISP requirements.

In addition, Mailgun offers proactive email monitoring, continuing advising, managed delivery services with API support, and management of IP and domain reputation.

A 97% average delivery success rate is what Mailgun advertises. Additionally, they tout an industry-lower bounce rate of 0.4% on average, compared to the industry standard of 2%.

2.4. Researching Email Security Mechanisms

Email security helps protect organizations and recipients from data breaches and other threats.

Use secure login authentication methods.

The traditional use of just a password and login to authenticate a user is considered outdated, but many people still use it. Modern authentication methods require two or more factors to verify a user's identity.

To use Two-Factor Authentication (2FA), users must combine three different types of authentication factors: knowledge (something you know, like a PIN or password), possession (something you have, like a debit card or cell phone), and physical uniqueness (something that only you have, like a fingerprint). For instance, you have to show your debit card and enter your PIN to use an ATM (possession factor and knowledge component, respectively). Temporary one-time passwords or SMS are frequently used for this kind of authentication.

Users can access numerous applications using Single Sign-On (SSO) after authenticating their identities with an identity provider. SSO and 2FA are frequently coupled. Centralized access with SSO lessens the possibility of account intrusion, streamlines the login procedure, and aids in enforcing strong passwords [20].

Application Programming Interfaces (APIs) are unique codes that identify and authenticate users. They provide a more secure alternative to logging in with a username and password. Developers commonly use APIs to control access to services like email APIs.

When configuring API keys, different permission levels can be assigned to restrict access to specific parts of an account.

Sender authentication protocols are essential for proving the authenticity of the sender and preventing spam or spoofing. The following authentication protocols should be used:

- SPF: Identifies mail servers allowed to send emails from a specific domain.
- DKIM: Verifies that the sender is responsible for the email's content and domain.
- Domain-based Message Authentication, Reporting & Conformance (DMARC): Specifies how to handle emails not authenticated by SPF or DKIM.

Using an SMTP server is crucial for email delivery. The SMTP server processes and sends emails to the recipient's inbox. It also verifies the email's origin and protects recipients from illegitimate senders.

SMTP authentication enhances email security by requiring a login using a supported authentication mechanism.

Data protection is vital, especially for sensitive information. Encryption, such as TLS, safeguards email content during transmission. Mail providers prioritize TLS-encrypted connections when delivering emails, preventing unauthorized access to email content [21–23].

To enhance data security, it is important to restrict access to data, especially recipient data, to only those employees who require it. This minimizes the risk of email leaks becoming more serious if the data is compromised. Here are some ways to achieve this:

- Limit the number of users to only those employees who regularly need access to your ESP. Regularly review the user list to ensure there are no inactive users.
- Implement different levels of permissions based on each user's role. This allows team-members to access the necessary features for their functions while safeguarding sensitive customer data.

Sender Policy Framework

SPF is an email authentication method that verifies which mail servers are authorized to send emails from a specific domain [24]. This helps ISPs identify when spammers try to send malicious emails from a domain they don't own. With SPF, email recipients can trust that emails are coming from the sender they expect, and senders can be confident that their brand is not being used to send phishing emails. SPF records are short lines of text added to a domain's TXT record in the DNS and are checked early in the SMTP conversation to establish a TCP connection between the sender and the receiving server. The SPF record specifies the IP addresses that are authorized to send emails to that domain. To confirm that

an SPF record is configured correctly, one can use various tools such as Scott Kitterman's SPF Testing Tools, OpenSPF.org, SPF Record Check, or SPF Wizard.

DomainKeys Identified Mail

Cisco and Yahoo jointly created the encryption method known as DKIM. Its function is to give senders the option to "sign" their messages, enabling receivers to confirm the legitimacy and authorization of the domain that sent the email [24]. Popular email service providers like Gmail and Microsoft might restrict or prevent the delivery of such messages in the absence of a DKIM signature.

DKIM is a straightforward method for email authentication that confirms the sender is in control of the domain and responsible for the email's content. There are two stages to the DKIM process:

- A private key is added by the sender to their email servers, and this key is used to sign the email.
- The receiving server verifies the sender's signature using the public key that is kept in the text record dkimselector._domainkey.domain.com.

DKIM is essential for establishing the sender's legitimacy, promoting confidence, and lowering the dangers of spam and phishing assaults. Inbox providers may block emails if DKIM is not appropriately implemented, preventing them from getting to the intended recipients. While a few blocked messages might not seem like much, they can have a big impact on businesses.

There are numerous internet resources for checking DKIM. A DKIM analyzer or DKIM validation tool can assist in making sure that your DKIM record has been published correctly. Before putting any modifications to SPF or DKIM data into effect, they must be verified.

Domain-based Message Authentication, Reporting & Conformance

SPF and DKIM are two methods used by the DMARC protocol to confirm the legitimacy of an email message.

Internet Service Providers (ISPs) can successfully resist fraudulent email tactics like domain spoofing, which try to trick recipients and steal their personal information, by using DMARC records.

DMARC enables email senders to declare what to do with messages that are unable to authenticate with SPF or DKIM. Senders can either direct these emails to a garbage folder or completely block them. With fewer false positives because of this proactive method, carriers may more precisely detect spammers and safeguard customers' inboxes. Additionally, it produces thorough authentication reports for increased market transparency.

Implementing DMARC is highly recommended for the following reasons:

Reputation: By stopping unauthorized parties from sending emails from a domain that is not their own, publishing a DMARC record protects your brand. Sometimes, improving your reputation simply by releasing a DMARC record is enough.

Visibility: By identifying the senders who are using your domain to send emails, DMARC reports give you important information about your email program.

Security: DMARC aids in establishing a uniform protocol for processing unauthenticated emails across the email community. The entire email ecosystem's security and dependability are strengthened by this group effort.

A crucial addition to email authentication, DMARC data show how email senders and ISPs work together to secure the email channel.

Transport Layer Security

A technology called TLS protects digital communications between two parties. It makes sure that no communication may be intercepted, eavesdropped on, or altered while data is transmitted between a server and a client [25]. TLS configuration can be difficult to get right, and a bad configuration can give users a false impression of security.

The Secure Sockets Layer (SSL) protocol was replaced by the TLS protocol, which is now officially deprecated and regarded as insecure. It is strongly advised against using any SSL version. Additionally, the IETF declared in RFC 8996 that TLS versions 1.1 and 1.0 are no longer supported and should not be utilized. The NCSC advises upgrading TLS 1.1 or 1.0-based government systems to either TLS 1.3 or 1.2.

To improve data security, TLS provides three main services:

Authentication: This enables each side to verify the other party's identity throughout the communication.

Data between the user agent and the server is transmitted encrypted to prevent reading or decryption by unauthorized parties.

Integrity: TLS protects data against loss, damage, manipulation, or fabrication by ensuring that it is unchanged and unaltered throughout encryption, transmission, and decryption.

The handshake phase of a TLS connection is where the client and server create a shared secret and talk about crucial factors, including cipher settings, to ensure a secure communication channel.

2.5. Development of a Functional Model and Application of the Enterprise Email System Security Subsystem

According to the study of the requirements for the design of the security subsystem, we have a problem with email delivery.

Figure 4: Illustration of the situation with the place where letters were lost

It has been determined that emails may not reach the final recipient for the following reasons:

- SendGrid may become unavailable.
- The SendGrid API key may expire or be incorrectly configured.
- IP address restrictions may be configured incorrectly.

That's why it was decided to add a mail intermediary to this scheme, which will make a mail queue and prevent mail loss.

Figure 5: Simplified diagram of the new infrastructure configuration

Since we have a multi-cluster environment, we need to take care of working in this scenario.

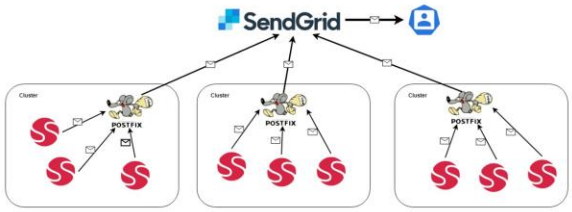

Figure 6: Simplified scheme of mail queue operation in a multi-cluster environment

So, each cluster has two redundant Postfix applications running synchronously and on different feeds. On each cluster, Postfix will have separate settings and its own SendGrid API key.

After researching the available software, the following resources were selected:

- Azure cluster (AKS).
- Postfix as a mail transfer agent.
- Fluent-bit for collecting logs from other containers.
- Grafana-Loki-SQL database for logging processing.
- Azure managed disks for persistent storage to keep the mail queue even when something happens to the deployment.

Develop a system that will store the mail sent by the application, check if the mail is possible, and only then send the mail. You also need to develop a system for monitoring the number of sent emails.

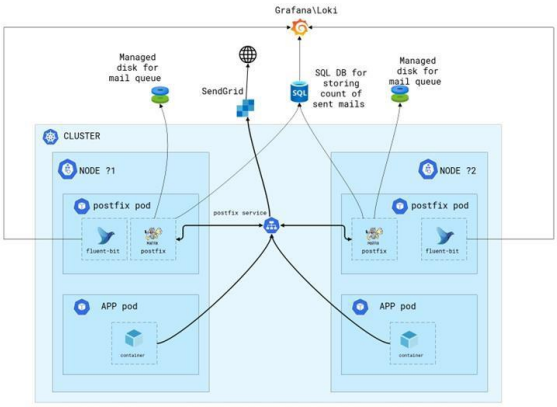

Figure 7: Detailed scheme of applications in the cluster according to the previously defined requirements

We develop Helm configuration files for deploying infrastructure in cluster environments.

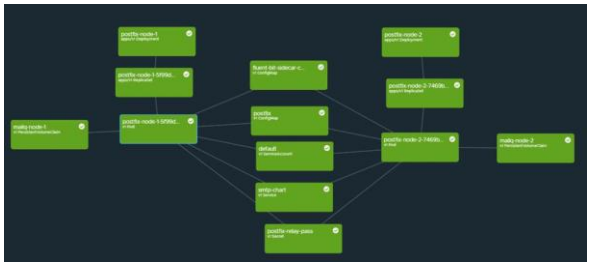

Figure 8: The structure of the Helm chart

So, in one cluster we have:

- Two Postfix pods that include two containers with Fluent-bit and Postfix.
- Two PersistentVolumeClaim.
- Two ReplicaSets.
- One ConfigMap for FluentBit.
- One ConfigMap for Postfix.
- One ServiceAccount.
- One Service.
- One Secret.

The Postfix application was chosen as the basis of the email system for the project. The main problem with Postfix, in our case, is that its development began a long time ago and was not designed for the modern infrastructure of Internet technologies. Postfix has proven to be a simple and reliable system for sending and queuing mail, but it was not designed for containerized environments and has a log system that is quite difficult to work with. For optimal work with logs and system monitoring in general, it was decided to create an application for parsing and transforming logs].

As an alternative to syslog (which is still the default), Postfix has its logging system. Postfix version 3.4 or later supports this. When Postfix is operating in a container, logging to stdout is advantageous since it removes the syslogd dependence.

Python, a general-purpose high-level programming language, was used to create the solution. Its design philosophy places a strong emphasis on the readability of the code by using substantial indentation.

The most recent Alpine base container with the parser and Postfix was chosen as the foundation for the container with the parser. Alpine is currently the standard for using it as the basis for your applications. It is constantly updated, constantly receives modern solutions to security problems, has basic functionality, and, despite all of the above, has a fairly small size of ~5Mb.

However, for Postfix to work properly and use all the parser's functions, you need to install the software. For the parser to work, you must first install Python from the repository, the pyodbc library, msodbcsql, mssql-tool. For Postfix to work with secure authorization, you need to install Postfix and the packages postfix-pcre, libsasl, cyrus-sasl. In addition to all these packages, you need to install some additional ones.

For Postfix authorization to work properly, you need to create a system user and add information about it to the Postfix configuration files.

The final step in the Dockerfile is to run the script for initial setup and start the parser.

The functions of the initial startup script are to configure Postfix parameters and then start the parser. The shell used is ash, which is a simplified version of the shell from the Alpine Linux developers. The parameters used for configuration are taken from environment variables, which is a safe approach.

We have also implemented functionality when it is necessary to transfer non-standard settings when deploying a solution. For this purpose, there is a separate environment variable POSTFIX_CUSTOM_CONFIG, to which non-standard parameters are passed through the ";" symbol.

Today, many cloud service providers provide services for renting their computing capabilities and using Kubernetes clusters based on their equipment. In this case, the architecture of the solution is based on the AKS from Microsoft. Despite this, the development of automated templates based on Helm Charts is not dependent on service providers in most cases and is universal, that is, it can be used on almost any environment, including a local computer.

Helm is a Kubernetes package manager**.** Helm is used to build "charts", which are packages of Kubernetes resources used to deploy applications to a cluster. There are different types of storage and distribution of these packages: they can be stored in archives or just directories with files, distributed through private or public repositories.

The main part of any Helm chart is the Deployment type, which describes what the Pod will consist of, what containers should be included in it, what files should be attached to

it, and what environment variables should be in the containers.

In this case, the Deployment consists of a Pod that includes two containers: Fluent-Bit of the latest version and the container with the parser and Postfix. The container with Fluent-Bit contains log storage and a configuration file.

```
- name: fluent-bit
  image: fluent/fluent-bit
  volumeMounts:
    - name: log-storage
      mountPath: /mnt/log/
      readOnly: true
    - name: fluent-bit-config
      mountPath: /fluent-bit/etc/
```

Figure 9: An example of mounting storage to a container

A persistent memory disk is attached to the container with Postfix to store logs in case of a cluster or container reboot. In Deployment, checks are additionally configured to see if the container has been successfully powered on and if it is still available.

If you need to have multiple copies of Postfix on different host machines for greater reliability, Deployment is configured to prevent the installation of two copies on one host.

```
spec:
  topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: kubernetes.io/hostname
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
        matchLabels:
          app_selector: app_selector
```

Figure 10: An example of mounting storage to a container

Also, environment variables that are stored in the ConfigMap are passed to the container. The ConfigMap.yaml file describes two ConfigMap resources: the first one stores environment variables that are passed to the container from Postfix and the parser, and the second one contains configuration files for Fluent-Bit.

The FluentBit settings describe where to get the information, how to process it, how to filter only the necessary data, and where to send the generated output information. It also contains information on how to work with the information we receive from the developed parser.

To store passwords, certificates, etc., the Secret type is used, which stores information in a hashed form. It's worth noting that it's exactly hashed, not encrypted. In the Secret of our Helm Chart, we store the password to the MSSQL database and the key to the SMTP service provider. This is to ensure that other services in the cluster, namely applications that send mail, have a network connection to Postfix, you need to create a Service dash resource. It specifies the port-forwarding parameters.

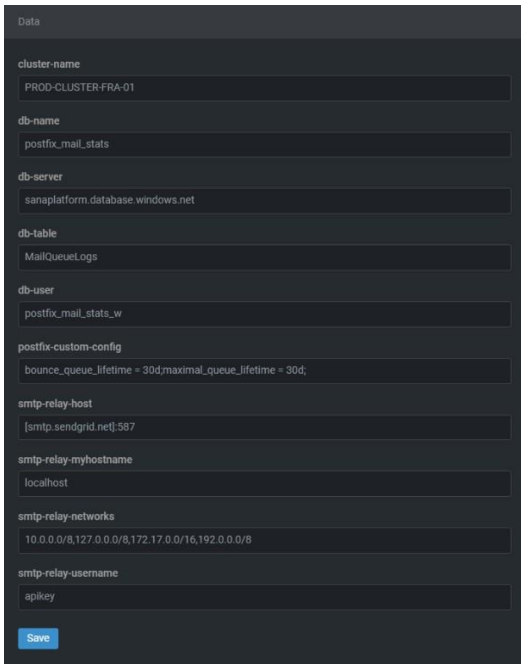

Figure 11: ConfigMap that contains environment variables

The name, version, and version of the program, as well as a brief description of the Helm Chart, are all included in the Chart.yaml file, which also contains some basic information about the Helm Chart itself.

Variables can be passed to Helm Chart using various methods, but the best practice is to store them in a YAML file and keep them in a version-controlled git repository to prevent configuration drift.

It is recommended to deploy one Postfix Deployment per cluster but with a different number of Postfix replicas, at least two.

At this point, the development of the template for automatic deployment of the solution can be considered completed. Further, the received files will be automatically processed by the helm utility, which is part of the Azure DevOps Pipelines portal plugins.

After the system has been successfully developed, and the operability of several systems has been constructed, it is necessary to have complete information about the operation of these systems, error visualization, and complete statistics on successfully delivered mail.

There are many systems for monitoring, but the choice was made to use free open-source solutions—Grafana and Grafana Loki.

Grafana Loki is an aggregator of log records, in this case, received from FluentBit.

Grafana is a modern data visualizer that can work with almost any data source. We need to visualize data from Grafana Loki and MSSQL.

Several deployment options are available for Grafana and Grafana Loki, including installation as a service, operation in a container, and deployment in Kubernetes. We chose to use Kubernetes since it is currently the most cutting-edge method of deploying Grafana.

With the help of the IDE for Kubernetes—Lens, we will deploy the above services with standard settings, but add a certificate for secure connection to web resources.

In the Grafana data sources settings, we configure Grafana Loki and MSSQL.

Grafana has a fairly wide range of tools for displaying information, which means that operators will be able to get all the necessary data in the form they need.

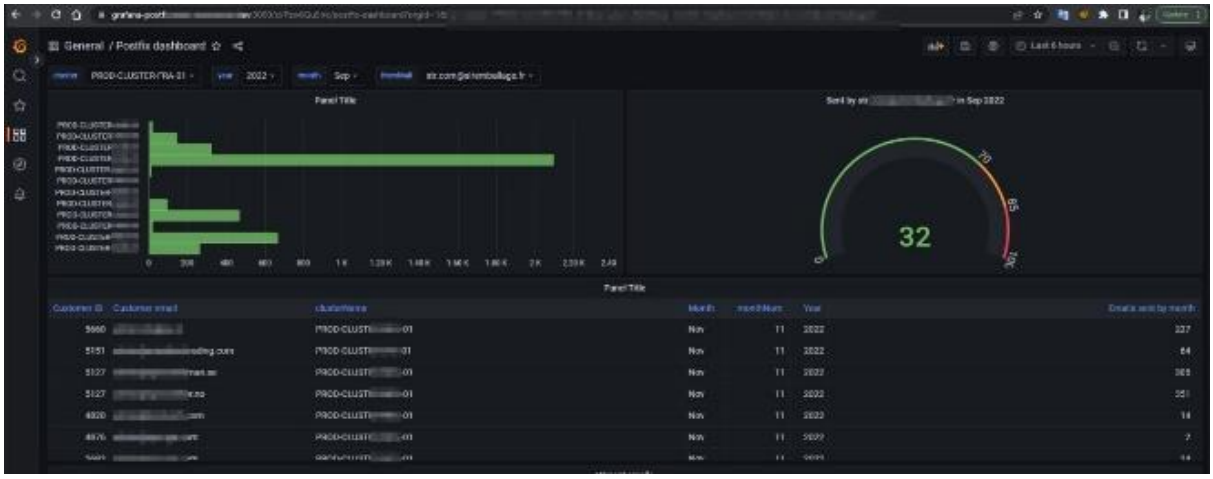

Figure 12: An example of displaying information about sent emails using Grafana

Monitoring systems are installed on the cluster environment and dashboards are developed for easy access to the necessary information.

3. Conclusion

The created system to prevent the loss of emails is relevant and important for organizations that rely on email for effective communication with customers and partners.

The efficiency of communication operations can be greatly increased and the risk of email loss can be dramatically decreased by implementing the security subsystem that has been created for a corporate email system based on containerized environments.

Taking into account the technical limitations of containerization and choosing the best technologies (for example, using Postfix, SendGrid, Microsoft Azure, and AKS) helps ensure the stability and reliability of the system.

The deployed data monitoring and visualization system allows us to quickly track mail-sending processes and respond to unforeseen events promptly.

The design of the email security subsystem demonstrates an increased level of reliability and visibility of email services in the enterprise, which can help improve business processes and mutual understanding with stakeholders.

Research and development of an email delivery system is a complex task, but given the importance of electronic communication in the modern world, investments in the development of such a system can be justified and bring positive results for organizations.

In general, the development and implementation of an email security subsystem for an enterprise is a step forward in ensuring reliable and efficient

communication, which will help improve interaction with customers and partners and reduce the risks of financial losses associated with data loss and delays in communication processes.

References

[1] V. Grechaninov, et al., Formation of Dependability and Cyber Protection Model in Information Systems of Situational Center, in: Workshop on Emerging Technology Trends on the Smart Industry and the Internet of Things, vol. 3149 (2022) 107–117.

[2] V. Grechaninov, et al., Decentralized Access Demarcation System Construction in Situational Center Network, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3188, no. 2 (2022) 197–206.

[3] F. Kipchuk, et al., Assessing Approaches of IT Infrastructure Audit, in: IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (2021). doi: 10.1109/picst54195.2021.9772181

[4] V. Buriachok, V. Sokolov, P. Skladannyi, Security Rating Metrics for Distributed Wireless Systems, in: Workshop of the 8th International Conference on "Mathematics. Information Technologies. Education": Modern Machine Learning Technologies and Data Science, vol. 2386 (2019) 222–233.

[5] Y. Sadykov, et al., Technology of Location Hiding by Spoofing the Mobile Operator IP Address, in: IEEE International Conference on Information and Telecommunication Technologies and Radio Electronics (2021) 22–25. doi: 10.1109/UkrMiCo52950.2021.9716700

[6] G. Howser, Simple Mail Transfer Protocol: Email, Comput. Netw. Internet (2020) 385–417. doi: 10.1007/978-3-030-34496-2_22.

[7] J. Klensin, Simple Mail Transfer Protocol, RFC (2001). doi:10.17487/rfc2821.

[8] Explained from First Principles, Email. URL:https://explained-from-first-principles.com/email/

[9] L. Enciso, et al., E-Mail Client Multiplatform for the Transfer of Information Using the SMTP Java Protocol Without Access to a Browser, Trends Adv. Inf. Syst. Technol. 745 (2018) 1124–1130. doi:10.1007/978-3-319-77 703-0_109.

[10] J. Klensin, Simple Mail Transfer Protocol, RFC (2008). doi:10.17487/rfc5321.

[11] K. Dilip, Review on SMTP Protocol for E-Mail Systems, Int. J. Res. Anal. Rev. 5(1) (2018) 127–132.

[12] M.B. Krishna, E-Mail Spam Classification using Naive Bayesian Classifier, Int. J. Res. Appl. Sci. Eng. Technol. 9(VI) (2021) 5209–5214. doi:10.22214/ijraset.2021.36153.

[13] R. Sureswaran, et al., Active E-mail System SMTP Protocol Monitoring Algorithm, 2nd IEEE International Conference on Broadband Network & Multimedia Technology (2009) 257–260, doi:10.1109/ICBNMT.2009.5348490.

[14] F. Mouton, L. Leenen, H. Venter, Social Engineering Attack Examples, Templates and Scenarios, Comput. Secur. 59 (2016) 186–209. doi:10.1016/j.cose.2016.03.004.

[15] I. Foster et al., Security by Any Other Name: On the Effectiveness of Provider Based Email Security, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (2015). doi:10.1145/2810103.2813607.

[16] Kubernetes, Kubernetes Documentation. URL:https://kubernetes.io/docs/home/

[17] S. Buchanan, J. Rangama, N. Bellavance, Helm Charts for Azure Kubernetes Service, Introd. Azure Kubernetes Serv. (2020) 151–189. doi:10.1007/978-1-4842-5519-3_8.

[18] B. Beach, et al., AWS Architecture Overview, Pro PowerShell for Amazon Web Services (2019) 1–8. doi:10.1007/978-1-4842-4850-8_1.

[19] K. Jangla, Containers, Accelerating Development Velocity Using Docker, (2018) 1–8. doi:10.1007/978-1-4842-3936-0_1.

[20] SendGrid, Email Delivery. URL: https://sendgrid.com/email-delivery/

[21] V. Jain, Analyzing Layer 2 and Layer 3 Traffic, Wireshark Fundamentals,

(2022) 79–134. doi:10.1007/978-1-4842-8002-7_3.
[22] P. Martin, Security, Kubernetes, Apress (2021) 115–156. doi:10.1007/978-1-4842-6494-2_11.
[23] K. Relan, V. Singhal, Pentest Ninja: XSS And SQLi Takeover Tool, Proceeding of the Second International Conference on Information and Communication Technology for Competitive Strategies 37 (2016) 1–2. doi:10.1145/2905055.2905243.
[24] S. Nightingale, Email authentication mechanisms: DMARC, SPF and DKIM (2017). doi:10.6028/nist.tn.1945.
[25] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, RFC (2018). doi:10.17487/rfc8446.