Deploying Testbed Docker-based application for Encryption as a Service in Kubernetes

Amir Javadpour*[||], Forough Ja'fari[§]**, Tarik Taleb[¶][††], Chafika Benzaïd[‡][‡‡], Luis Rosa[†], Pedro Tomás[†], and Luis Cordeiro[†]

*ICTFICIAL Oy, Espoo, Finland [†]OneSource, Coimbra, Portugal
[‡]Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland
[§]Department of Computer Engineering, Sharif University of Technology, Iran
[¶]Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany
[||]a.javadpour87@gmail.com (Corresponding Author) **azadeh.mth@gmail.com
[††]tarik.taleb@rub.de [‡‡]chafika.benzaid@oulu.fi

***Abstract*—The scalability of today's networking infrastructures, such as Kubernetes, has increased the demand for everything-as-a-service concepts, including encryption services. This paper details deploying an Encryption as a Service (EaaS) framework on Kubernetes. In our current implementation, a web platform is dedicated to the subscription processes and managing the requested services. Service providers subscribe to receive encryption and decryption services for their clients, and a token is assigned to them. The client, or better to say, the devices covered by that service provider, can use encryption/decryption to specify that token. As our implemented web platform is written in Django, through this paper, we have discussed the deployment of Django applications on Kubernetes and the role of two other services (i.e., Database and Nginx) to make it available. The services are executed on Docker containers and deployed on Kubernetes pods. We have also explained the steps to build Docker containers, including the details of Dockerfiles. We have deployed this framework on both local and remote Kubernetes environments. The former deployment was performed by setting up a local Minikube cluster, while the latter assumes a remote Kubernetes cluster, and we are connecting to it through a VPN. We have tested the availability and accuracy of the encryption/decryption services and checked the logs to ensure that these services work correctly. We aim to present this article to assist researchers wishing to conduct effective testing or a *Testbed* in the real world.**

***Index Terms*—Encryption as a Service (EaaS), Kubernetes, Internet of Things (IoT), NGINX, Deploying Docker-based, TestBed.**

I. Introduction

Encryption as a Service (EaaS) is the process of providing all cryptographic services to the end users, and it overcomes the resource limitation issues of the end devices. With EaaS, organizations can outsource the complex and time-consuming tasks of encryption and key management to third-party providers who specialize in these processes. This allows them to focus on their core business operations while maintaining the security and integrity of their sensitive data. EaaS offers numerous benefits, including easy implementation and scalability, reduced costs of ownership and maintenance, and enhanced security measures [1, 2, 3].

One of the needful users of EaaS solutions is the Internet of Things (IoT) devices. They have limited resources, and due to their large scale and the popularity of such networks among modern digital societies, a scalable EaaS framework is needed. However, managing and maintaining an EaaS platform for this demand is challenging [4, 5, 6]. An EaaS framework deployed on Kubernetes environments improves data security across other services provided by this environment. It focuses on real-time encryption, easy integration, and improved access controls [2, 7].

To communicate with the world outside the Kubernetes cluster, clients first connect to Nginx, which acts as an entry point and then routes requests to appropriate internal services, such as the *Django/Gunicorn* service. NGINX offers high performance, scalability, and reliability. NGINX is flexible and can be used for various applications, from serving static content to load balancing and caching dynamic web applications. NGINX Ingress Controller is a popular and powerful tool for managing external access to Kubernetes services. NGINX works with third-party modules for load balancing, caching, security, and monitoring, allowing users to extend the capabilities of their web servers while maintaining reliability. NGINX Service Mesh provides a trustful environment with strict control over allowed connections.

Minikube is a platform that allows running a single-node Kubernetes cluster in a local environment, such as a laptop or PC. This tool allows developers and DevOps engineers to install Kubernetes quickly for testing, development, or learning. Minikube includes all the core components of Kubernetes, including Deployments, Services, and ConfigMaps, and is compatible with common tools such as *kubectl*. In Kubernetes, a cluster may contain one or more nodes. Each node can host multiple pods, and each pod can host one or more containers. Each node has its resources, such as CPU, memory, and storage. When pods are deployed in the cluster, Kubernetes automatically determines which nodes are best suited to run the pods. To check the sufficient resources in the nodes, tools such as *kubectl* describe nodes and *kubectl* get pods -o wide are used to monitor the cluster and check how the pods are distributed.

We also have another concept called microservices, which can solve problems related to the growth of projects. They are often used with container management tools and services and deployed on Kubernetes-managed cloud platforms. Deploying microservices requires considering API versions and integration testing across multiple domains, and automated monitoring is critical to ensure that each component is working properly [8, 9, 10]. As mentioned above, NGINX is a powerful load-balancing solution trusted by some of the world's most popular websites, including Dropbox, Netflix, and Zynga. By providing dynamic reconfiguration for simple service management and easy integration with popular microservices management tools like Kubernetes, NGINX makes it easy for leading companies like Netflix to use NGINX at the core of microservices deployments. NGINX Controller provides application delivery management for NGINX microservices solutions and makes it easy to manage and monitor microservices architectures at scale. NGINX Service Mesh provides management solutions for our containerized microservices and provides solutions for bridging heterogeneous microservice environments. To deploy EaaS on Kubernetes, we need to partition the environment into different parts on different nodes. We also need to consider the hardware requirements for each node to consider the optimal performance and scalability of the EaaS deployment (Figure 1).

As shown in Figure 1 (Part A), EaaS deployment in a Kubernetes environment should have different partitions on different nodes, according to hardware requirements, to optimize performance and scalability. The details of each item for EaaS deployment in organizing Kubernetes containers based on NGINX service are shown in Figure 1 (Part B).

This paper focuses on implementing an EaaS framework on Kubernetes and provides its details. Figure 2 shows the implementation architecture, containing three main services: Django, Gunicorn, and Nginx. Django is a popular high-level web development framework in the Python programming language. This framework has attracted the attention of many web developers due to its Model-View-Controller (MVC) architecture, powerful tools for database management, and a fast and efficient development environment. The end-devices connect to the web platform (Django) through Nginx, as a proxy. Nginx forwards traffic from the end-devices to the Django service, and vice versa. The Django service is also exposed by the Gunicorn service. It is worth noting that a PostgreSQL database service [11] is also provided to store the data related to the EaaS framework.

Each service is encapsulated in Docker containers, defined by Dockerfiles, and orchestrated using Docker Compose with environment configurations managed through ".env" files. The deployment process on Kubernetes involves creating deployments for each service, ensuring communication between them, and efficiently managing pods. This setup exemplifies a robust infrastructure for a scalable EaaS framework.

The remainder of this paper is organized as follows. section II provides details on dockerizing the main EaaS services and the required configurations. section III presents the steps of deploying the created Docker containers on the Kubernetes environment. The performance of the deployed EaaS framework is then evaluated in section IV. Finally, section V gives this paper's summary and conclusion.

II. Dockerizing the Services

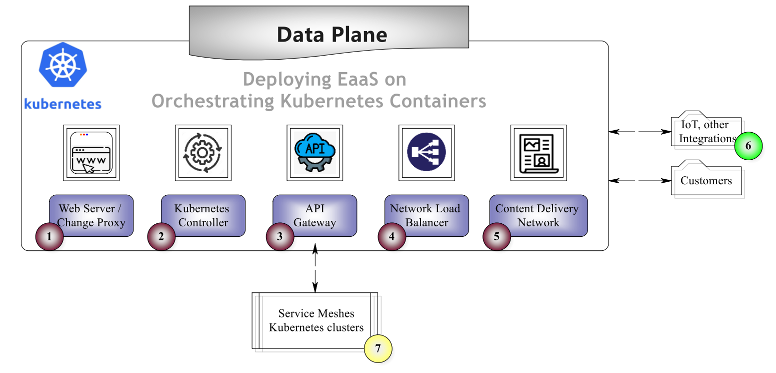
This section describes how to dockerize the main services to build the EaaS framework. Three services are needed to be executed on the Kubernetes based on their Docker container: a PostgreSQL service for the database, a Django service for the web platform, and an Nginx service for the proxy.

The steps to build Docker containers will be provided in the following section. This includes creating a Dockerfile for each service. For the web service, the Dockerfile includes installing dependencies, copying project files, and setting up Gunicorn. The Dockerfile consisted of copying the Nginx config file and setting the executable command for the Nginx service. Also, it explains how to build a Docker image using the docker build command and test it with docker run. Docker Compose is a tool that allows us to manage different Docker services with a single YAML file. We looked at how to define services in Docker Compose and map ports. Also, dependencies between services and settings related to networking and storage are explained. To configure Nginx, we looked at creating an upstream to direct traffic to the web service. We also discussed settings for Nginx listening on port 80 and how to configure paths for static and media files.

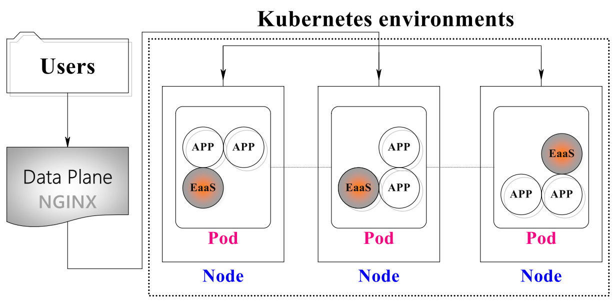
A Dockerfile is a file that defines a set of commands to build a Docker container. This file should specify what images will be used as a base, how dependencies will be installed, and how the project will run. Table I shows a sample dockerfile that we have used for the Django service. This file specifies the required dependencies to be installed and the related commands to be executed before the container is deployed. To serve static and media files, specific paths are provided to access CSS, JavaScript, images, and other static resources.

To run Nginx as a standalone service in Docker, we must create a special dockerfile for Nginx. Also, to properly configure Nginx, we need to include the appropriate settings in the Nginx config file. These settings include the upstream to direct traffic to the web service and the ports Nginx uses to listen for incoming requests. First, a Dockerfile for Nginx must be created in a separate path. This file contains the necessary commands to configure and run Nginx. In this Dockerfile, we need to copy the Nginx config file to the container, which will start Nginx. The Nginx config file contains upstream and routing settings. In this file, we must define how Nginx will route incoming traffic to the web service and the ports Nginx will listen to. These settings allow Nginx to connect to the web service and handle incoming traffic properly.

After that, the Dockerfiles are generated, and a docker-compose.yml file is used to define them as services. It is worth noting that for the database service, a default one is presented by Docker Hub, and we can easily build it on Kubernetes by calling the version and the path to its default container.



A) The details of Deploying EaaS on Orchestrating Kubernetes Containers based on NGINX Service



B) The EaaS deployment in a Kubernetes environment should have different parts on different nodes, with consideration given to the hardware requirements to optimize performance and scalability.

Fig. 1. The details of Deploying EaaS and presenting different parts on different nodes in Kubernetes Containers based on NGINX Service

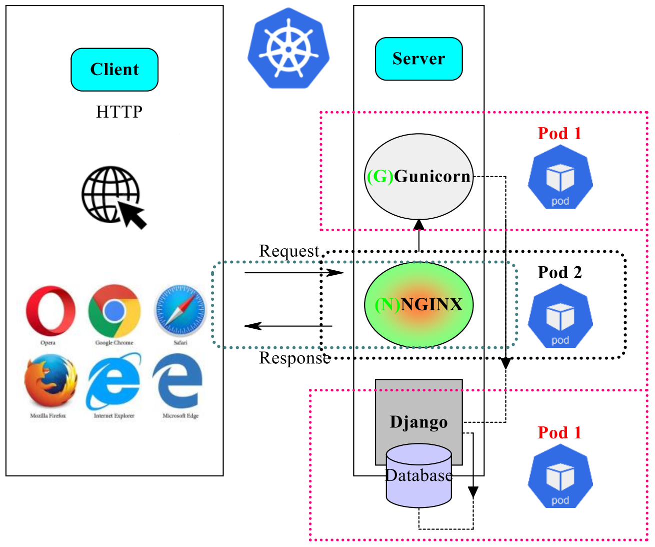
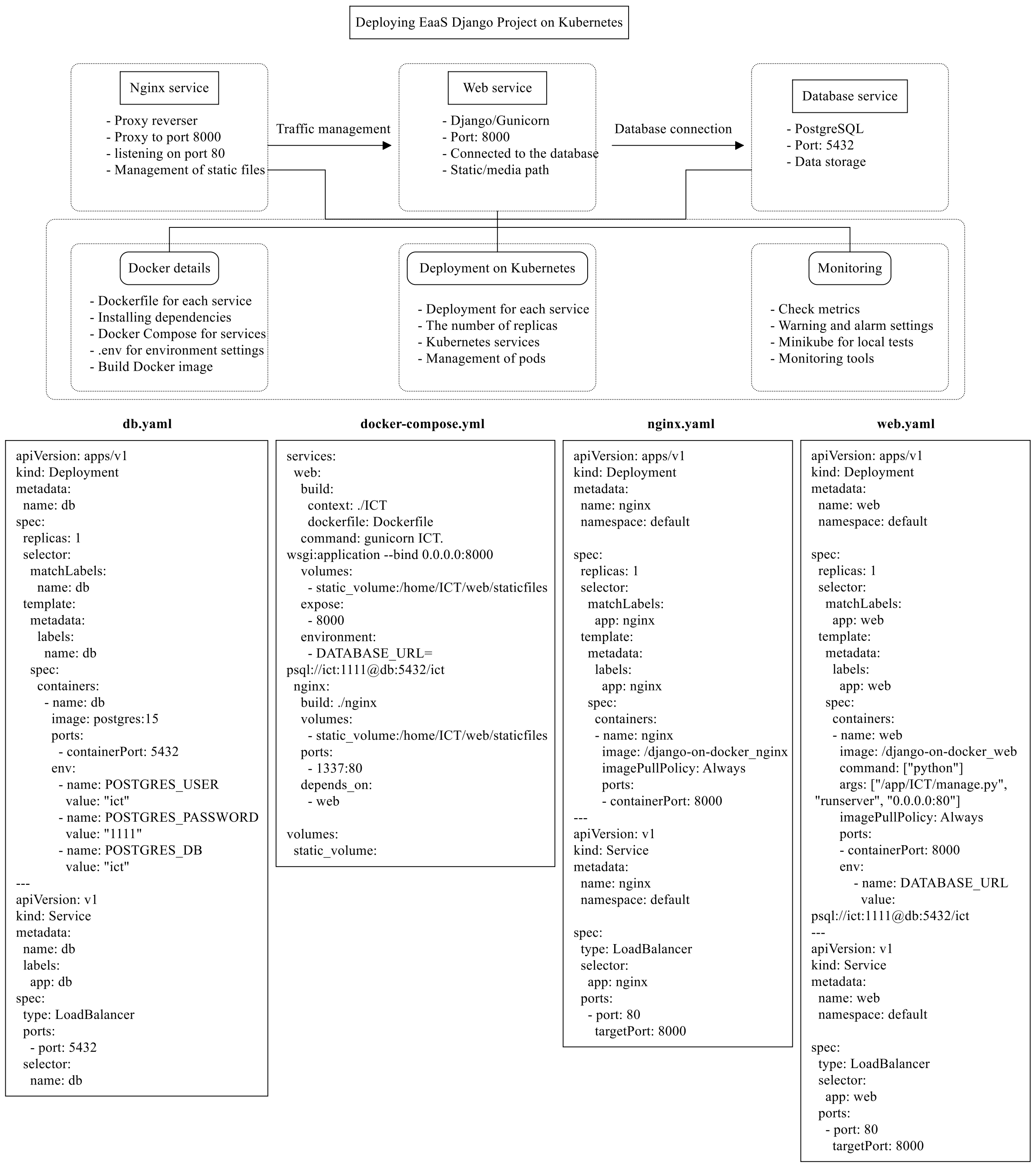


Fig. 2. The implementation architecture used for deploying the EaaS framework on Kubernetes.

Hence, dockerizing the Django platform and the Nginx proxy are discussed here.

The docker-compose.yml file shown in Figure 3 presents the configurations for dockerizing web (i.e., Django service) and nginx (i.e., Nginx proxy). The database URL is also specified to show the web service where to communicate for storage processes. Moreover, a Gunicorn command is defined to make the Django service accessible. Gunicorn is used as a WSGI server to handle requests. It is worth noting that we have passed the environment variables directly in the docker-compose.yml file in this configuration. Another common way to manage environment variables is to create a .env file that contains important information such as database URLs, API keys, and other sensitive settings. This file should be placed



db.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
spec:
  replicas: 1
  selector:
    matchLabels:
      name: db
  template:
    metadata:
      labels:
        name: db
    spec:
      containers:
        - name: db
          image: postgres:15
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: "ict"
            - name: POSTGRES_PASSWORD
              value: "1111"
            - name: POSTGRES_DB
              value: "ict"
---
apiVersion: v1
kind: Service
metadata:
  name: db
  labels:
    app: db
spec:
  type: LoadBalancer
  ports:
    - port: 5432
  selector:
    name: db
```

docker-compose.yml

```
services:
  web:
    build:
      context: ./ICT
      dockerfile: Dockerfile
    command: gunicorn ICT.
wsgi:application --bind 0.0.0.0:8000
    volumes:
      - static_volume:/home/ICT/web/staticfiles
    expose:
      - 8000
    environment:
      - DATABASE_URL=
psql://ict:1111@db:5432/ict
  nginx:
    build: ./nginx
    volumes:
      - static_volume:/home/ICT/web/staticfiles
    ports:
      - 1337:80
    depends_on:
      - web

volumes:
  static_volume:
```

nginx.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default

spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: /django-on-docker_nginx
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default

spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 8000
```

web.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  namespace: default

spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web
        image: /django-on-docker_web
        command: ["python"]
        args: ["/app/ICT/manage.py",
 "runserver", "0.0.0.0:80"]
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
        env:
            - name: DATABASE_URL
              value:
psql://ict:1111@db:5432/ict
---
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: default

spec:
  type: LoadBalancer
  selector:
    app: web
  ports:
    - port: 80
      targetPort: 8000
```

Fig. 3. Deploying a Django project on Kubernetes using Docker.

in the main project folder and not uploaded to version control systems.

The configurations for the Nginx service should be added, and necessary configurations should be done in the docker-compose.yml file.

III. Deploying on Kubernetes

This section provides details on pods and services for Kubernetes deployment. The database service, web, and Nginx are defined as the project's main components in this configuration. Docker images are used for each service, and required settings are provided by using environment variables. The Kubernetes infrastructure to deploy these services is shown in Figure 1. When the Docker containers are ready, we can deploy them on the Kubernetes pods. Figure 3 shows how the three containers related to our three main services are deployed. The Nginx service is a proxy reverser, and Gunicorn handles incoming requests through port 8000.

TABLE I
THE DOCKERFILE USED FOR DOCKERIZING THE DJANGO WEB SERVICE.

```
FROM python:3.8.10-slim-buster as builder
WORKDIR /usr/src/ICT
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
RUN apt-get update && apt-get install -y –no-install-recommends gcc
RUN pip install –upgrade pip
RUN pip install flake8==6.0.0
COPY . /usr/src/ICT/
COPY ./requirements.txt .
RUN pip wheel –wheel-dir /usr/src/ICT/wheels -r requirements.txt
FROM python:3.8.10-slim-buster
RUN mkdir -p /home/ICT
RUN addgroup –system ict && adduser –system –group ict
ENV HOME=/home/ICT
ENV APP_HOME=/home/ICT/web
RUN mkdir $APP_HOME
RUN mkdir $APP_HOME/staticfiles
WORKDIR $APP_HOME
RUN apt-get update
RUN apt-get install -y –no-install-recommends netcat
COPY –from=builder /usr/src/ICT/wheels /wheels
COPY –from=builder /usr/src/ICT/requirements.txt .
RUN pip install –upgrade pip
RUN pip install –no-cache /wheels/*
COPY . $APP_HOME
RUN chown -R ict:ict $APP_HOME
USER ict
```

To bring up a local Kubernetes cluster, we have used Minikube, and *Kubectl* allows us to connect to the cluster, create new resources, and monitor the status of the cluster and pods. *kubectl* is the primary tool for Kubernetes cluster management.

We can use containers such as PostgreSQL to create and deploy an SQL database. For example, if we want to have a PostgreSQL version 15, we can call the version associated with this version from Docker Hub. YAML files are commonly used to create and run a database in Kubernetes. These files contain Kubernetes resource definitions for developing and managing pods, deployments, and services. The command "*kubectl* apply -f db-deployment.yaml -f db-service.yaml" applies the YAML files and creates the required deployment and service. This way, we will deploy one replica of PostgreSQL and a service listening on the specified port in Figure 3 part ***db.yaml***.

Data monitoring and services are critical for maintaining operational integrity and security in deploying Encryption-as-a-Service on Kubernetes. Each service is carefully monitored using tools to track health, performance, and resource usage. Logs from each container are collected and analyzed to detect anomalies and troubleshoot problems. Kubernetes' native monitoring capabilities are enhanced with tools like Prometheus and Grafana for metrics collection and visualization. Additionally, Kubernetes' logging capabilities play a vital role in monitoring for auditing access and identifying potential security breaches. PostgreSQL is used as the database service, configured through Dockerfiles, and managed using Kubernetes Secrets and ConfigMaps for sensitive information. Data replication, backup, and recovery strategies are implemented to safeguard against data loss and ensure high availability.

IV. EVALUATION RESULTS

The deployment process involved setting up Kubernetes services to facilitate communication between the web service, database, and Nginx. Environment configurations were managed through .env files, ensuring sensitive information was securely handled. Testing with Minikube in a local environment confirmed that the deployment works as intended, with services communicating correctly and efficiently handling HTTP/HTTPS traffic. Using *Kubectl* to manage resources and check logs has been instrumental in verifying the deployment's stability and performance. Initial results are promising, with the deployed system showing robust performance and scalability. Also, in our current implementation, we have implemented the EaaS and the subscription part to prepare tokens for new clients. We also discussed various topics related to deploying and configuring Django projects on Kubernetes and Docker. First, we discussed the deployment of Django on Kubernetes and the role of the three primary services (web, database, and Nginx). The database service was defined with PostgreSQL, the web service with *Django/Gunicorn*, and the Nginx service to handle HTTP traffic. To build Docker containers, we explained the steps to create a Dockerfile and the methods of creating a Docker image. The Dockerfile included steps to install dependencies, copy project files, and run the app with Gunicorn. Also, we examined how to configure the Django project using environment files (.env) and methods of managing environment variables. Next, the structure of the Docker Compose file, how to define different services, map ports, and the settings required to connect services were discussed. Also, Nginx configurations and upstream settings, ports, and traffic routing to the web service were explained. Also, we covered setting up Minikube and how to test Kubernetes deployments in a local environment. We ran Kubernetes files, tested services with curl, and checked logs to ensure services worked correctly.

We started by building Docker containers to package our custom services. This involved setting up Dockerfiles to create container images that included everything needed for the services. Once the containers were built, we deployed them to Kubernetes, where they were run as pods, the basic building blocks in Kubernetes. This made our services accessible and ready for users. Then, we tested Encryption as a service platform. This involved checking that the different parts could connect, looking at logs to spot errors, and ensuring the services were fast and responsive. This final step aimed to ensure the platform was ready for production and that users would have a smooth experience. We need to ensure that users connected to the EaaS platform have access. Various platforms are available in the EaaS sector, such as a Django-type web service. This web service provides a page to the customer and subscription encryption services. Upon joining and requesting a service, the user receives a token (for example, 15a4d), which he or his devices can use to access EaaS. When a user registers, his information is stored in the database along with the token. The EaaS platform has several components,

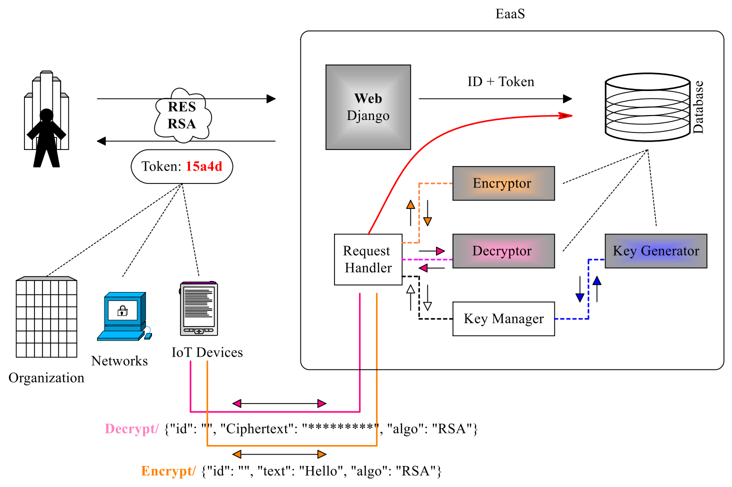


Fig. 4. Make an HTTP post request and include the authorization token received from the site

including the request handler (RH), Decryptor, Encryptor, key manager (KM), and key generator (KG).

In Figure 4, we see an Internet of Things (IoT) device sending a request to RH, which wants to encrypt the word ”Hello” using RSA encryption. The device also has a token. When RH receives a request, it first searches the database for the token and user information to verify if the service can be provided. If allowed, RH communicates with KM to obtain the key generated by KG for the encryption process. Then, RH sends the encrypted content back to the IoT client (***curl -X POST -H ”Content-Type: application/json” -H ’Authorization: Bearer aaaa’ -d ’”id”: ”1”, ”text”: ”hello”, ”algo”: ”rsa”’ http://10.1.0.100:30303/encrypt/***). The request is sent to RH for decryption, which contains the ID and encrypted text, and assuming that the cipher is “bbbb”, to decrypt it, the command is used (***curl -X POST -H ”Content-Type: application/json” -H ’Authorization: Bearer aaaa’ -d ’”id”: ”1”, ”cipher”: ”bbbb”, ”algo”: ”rsa”’ http://10.1.0.100:30303/decrypt/***)

V. Conclusion and Future Work

In this paper, we implemented an encryption as a service framework on Kubernetes. The current Testbed has been deployed on OneSource (OneSource, Consultoria Informática, Lda.) company assets as part of the RIGOUROUS ”secuRe desIGn and deplOyment of trUsthwoRthy cOntinUum computing 6G Services” project. We have covered deploying Django on Kubernetes, building Docker containers, configuring Django projects using environment files, setting up Minikube, and testing Kubernetes deployments in a local environment. Further integration with advanced monitoring and logging tools will be pursued to enhance the system’s observability. This includes implementing more granular logging and real-time alerts to identify and resolve issues proactively. Enhancing security measures should be considered for future work, such as incorporating advanced authentication mechanisms and ensuring compliance with emerging data protection regulations.

In our future projects, We plan to enhance the security of our Docker-based application deployed on Kubernetes by integrating a network honeypot. Additionally, we will implement virtual network embedding, utilizing machine learning algorithms to optimize resource allocation. It acts as a decoy to lure and analyze potential attackers, significantly enhancing our security measures [12, 6, 13, 14, 15]. Administrators can detect and monitor unauthorized access attempts by deploying a honeypot and collecting real-time data on emerging threats.

Deploying EaaS framework on Kubernetes revealed several key insights regarding performance, scalability, and security. Integrating Django, PostgreSQL, and Nginx into Docker containers managed by Kubernetes has shown promising results, especially in handling HTTP/HTTPS traffic and interservice communications. Initial experiments using Minikube in a local environment confirmed the robustness and potential scalability of the framework to meet the needs of IoT devices, which require secure and efficient communication channels despite limited resources. However, several areas require further exploration and improvement, especially in terms of security improvements. While the current implementation uses standard encryption methods, there is significant potential to enhance security through advanced authentication mechanisms and strict compliance with emerging data protection regulations.Also, The deployment process emphasized the importance of careful setup and configuration, especially when handling sensitive information via environment (.env) files. The use of Dockerfiles and Docker Compose is essential in managing container services, facilitating effective deployment in a controlled environment. However, moving to a production environment may present additional challenges, such as managing API versions and ensuring comprehensive integration testing across multiple domains. Addressing these challenges will be critical for wider adoption and real-world application of the EaaS framework.

Another critical area identified is the need for advanced monitoring and reporting tools. The current implementation uses basic tools to verify the stability and performance of the deployment, which is sufficient for initial testing. For production environments, more sophisticated monitoring solutions are necessary to provide detailed reporting and real-time alerts.

Future work also aims to further secure Docker-based applications deployed on Kubernetes based on Moving Target Defence to shuffle the Pods [16, 17]. This includes the aforementioned integration of a network honeypot and the implementation of advanced monitoring and reporting tools to improve system visibility. Ensuring compliance with data protection regulations and incorporating advanced authentication mechanisms will be critical. These improvements not only enhance the security of the framework, but also ensure its durability and reliability in various real-world applications, ultimately driving its adoption across industries.

Acknowledgment

This research is partially supported by the European Union's Horizon Europe research and innovation program under the RIGOUROUS project (Grant No. 101095933). It is also partially supported by the Research Council of Finland (formerly the Academy of Finland) through the 6G Flagship program (Grant No. 346208). The paper reflects only the authors' views, and the European Commission bears no responsibility for any utilization of the information contained herein.

References

[1] A. Alqarni, "Enhancing cloud security and privacy with zero-knowledge encryption and vulnerability assessment in kubernetes deployments," Ph.D. dissertation, Middle Tennessee State University, 2023.

[2] A. Javadpour, F. Ja'fari, T. Taleb, Y. Zhao, Y. Bin, and C. Benzaïd, "Encryption as a service for iot: Opportunities, challenges and solutions," *IEEE Internet of Things Journal*, 2023.

[3] M. Zhang, J. Cao, Y. Sahni, Q. Chen, S. Jiang, and T. Wu, "Eaas: A service-oriented edge computing framework towards distributed intelligence," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2022, pp. 165–175.

[4] D. Unal, A. Al-Ali, F. O. Catak, and M. Hammoudeh, "A secure and efficient internet of things cloud encryption scheme with forensics investigation compatibility based on identity-based encryption," *Future Generation Computer Systems*, vol. 125, pp. 433–445, 2021.

[5] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[6] A. Javadpour, F. Ja'fari, T. Taleb, M. Shojafar, and C. Benzaïd, "A comprehensive survey on cyber deception techniques to improve honeypot performance," *Computers & Security*, p. 103792, 2024.

[7] A. Javadpour, F. Ja'fari, and T. Taleb, "Encryption as a service: A review of architectures and taxonomies," in *Distributed Applications and Interoperable Systems*, R. Martins and M. Selimi, Eds. Cham: Springer Nature Switzerland, 2024, pp. 36–44.

[8] T. Z. Benmerar, T. Theodoropoulos, D. Fevereiro, L. Rosa, J. Rodrigues, T. Taleb, P. Barone, G. Giuliani, K. Tserpes, and L. Cordeiro, "Towards establishing intelligent multi-domain edge orchestration for highly distributed immersive services: a virtual touring use case," *Cluster Computing*, pp. 1–31, 2024.

[9] T. Taleb, A. Boudi, L. Rosa, L. Cordeiro, T. Theodoropoulos, K. Tserpes, P. Dazzi, A. I. Protopsaltis, and R. Li, "Toward supporting xr services: Architecture and enablers," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3567–3586, 2022.

[10] T. Z. Benmerar, T. Theodoropoulos, D. Fevereiro, L. Rosa, J. Rodrigues, T. Taleb, P. Barone, K. Tserpes, and L. Cordeiro, "Intelligent multi-domain edge orchestration for highly distributed immersive services: an immersive virtual touring use case," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 2023, pp. 381–392.

[11] H. Schönig, *Mastering PostgreSQL 15: Advanced techniques to build and manage scalable, reliable, and fault-tolerant database applications*. Packt Publishing, 2023. [Online]. Available: https://books.google.nl/books?id=ZBOrEAAAQBAJ

[12] A. Javadpour, F. Ja'Fari, T. Taleb, and C. Benzaïd, "A mathematical model for analyzing honeynets and their cyber deception techniques," in *2023 27th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2023, pp. 81–88.

[13] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "Enhancing 5g network slicing: Slice isolation via actor-critic reinforcement learning with optimal graph features," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 31–37.

[14] A. Javadpour, F. Ja'fari, T. Taleb, and C. Benzaïd, "Reinforcement learning-based slice isolation against ddos attacks in beyond 5g networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3930–3946, 2023.

[15] C. Benzaïd, T. Taleb, A. Sami, and O. Hireche, "Fortisedos: A deep transfer learning-empowered economical denial of sustainability detection framework for cloud-native network slicing," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 2818–2835, 2024.

[16] A. Javadpour, F. Ja'fari, T. Taleb, M. Shojafar, and B. Yang, "Scema: An sdn-oriented cost-effective edge-based mtd approach," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 667–682, 2023.

[17] A. Javadpour, F. Ja'fari, T. Taleb, and M. Shojafar, "A cost-effective mtd approach for ddos attacks in software-defined networks," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 4173–4178.