# ECE1779 – Introduction to Cloud Computing (Fall 2025)

## Project Proposal:

**Title: Cloud-Native Network Health Dashboard**

Author: Diana Chamorro Zuleta

Deployment Provider: DigitalOcean

Orchestration: Kubernetes (DOKS)

Database: PostgreSQL with Persistent Volume

Advanced Features: Authentication (JWT) and CI/CD Pipeline with Backup & Recovery

## 1. Motivation

### Problem Statement

I currently work full-time as a Network Engineer at Rogers Communications, where I’ve spent the past four years working on large-scale network design, deployment, and performance optimization projects. Through this experience, I’ve had the opportunity to see how major telecom operators monitor and maintain network reliability using enterprise-grade tools and complex infrastructure. However, I’ve also noticed a consistent gap in the industry: smaller organizations and startups often lack access to affordable, user-friendly tools for monitoring their own network performance.

While large corporations rely on platforms like Datadog or Cisco ThousandEyes to track latency, uptime, and packet loss, these tools are expensive, and require significant setup and licensing costs. Smaller teams, local businesses, and even independent developers rarely have the resources or technical staff to implement such systems. Yet, reliable connectivity is just as important for them as it is for large companies; a few minutes of downtime can impact their operations, reputation, or online sales.

That’s where my project idea comes from. I want to build a lightweight, cloud-based **Network Health Dashboard** that gives smaller users some of the same visibility that big enterprises have but in a simpler, more accessible way. The idea is to create a web application that continuously checks the health of network endpoints (such as websites, APIs, or servers) and provides clear, visual insights on their performance. Users will be able to log in, register the endpoints they care about, and view metrics such as response time, uptime percentage, and latency trends through a clean, intuitive dashboard.

The system will run entirely in the cloud, using modern tools like **Docker** and **Kubernetes**, which means it can stay online 24/7 without depending on any local computer. Data will be stored safely in a **PostgreSQL database**, and monitoring results will persist across system restarts.

In short, this project combines my professional background with the technical skills developed in this course. It’s designed to demonstrate my understanding of containerization, orchestration, state management, and monitoring but more importantly, it aims to fill a real-world need. I want to create something practical and meaningful that smaller organizations can use to monitor their own services without needing an enterprise budget or a full IT department.

### ****Target Users****

This project is aimed at people who need a simple, affordable way to track how their websites or online services are performing. It’s especially useful for **developers and DevOps students** who want to monitor the uptime and response time of their APIs or small applications.

It can also benefit **network engineering students**who want a hands-on way to explore cloud-based monitoring and data visualization. Lastly, it’s practical for **independent creators or small online companies** who host projects on platforms like DigitalOcean and want clear, transparent insights into how their sites are doing without paying for expensive tools.

### ****Existing Solutions and Limitations****

Popular tools like **Datadog, Pingdom**, and **UptimeRobot**already offer network monitoring, but they are often **costly, closed-source, and overly complex** for small-scale use. This project takes a **simpler and open approach**, using containerized cloud technologies to give users transparency, control, and flexibility, all while keeping costs low and setup easy.

## 2. Objective and Key Features

The goal of this project is to build and deploy a **cloud-based web application** that keeps track of how well different network endpoints, such as websites or servers, are performing. It will **measure and display metrics** like response time and uptime, **store the results in a database,** and send **alerts** when performance drops below a certain threshold.

The system will be **fully containerized** using Docker and managed with **Kubernetes on DigitalOcean,** ensuring it runs reliably and can scale when needed. It will also include built-in **monitoring and automation pipelines**, showing how modern cloud technologies work together in a real, end-to-end setup.

### ****Core Features****

* **Containerization & Local Development:**  
  The entire application, including the backend, worker, frontend, and database, will run inside Docker containers. For local development, Docker Compose will be used to manage all components together, making it easy to test and debug before deployment.
* **Database & State Management:**  
  All user information, endpoint data, and measurement results will be stored in a PostgreSQL database. The setup will use persistent volumes so that data is never lost, even if the containers are restarted or updated.
* **Backend API:**  
  The backend will be built with FastAPI. It will handle user authentication through JWT tokens, manage endpoints using standard create/read/update/delete (CRUD) operations, and provide REST APIs for retrieving metrics.
* **Worker / Scheduler:**  
  A background Python job will run automatically at regular intervals (using a Kubernetes CronJob) to check each registered endpoint and save the latest performance data into the database.
* **Frontend Dashboard:**  
  The web interface, built with React (using Vite), will display each endpoint’s status and performance through simple charts and summaries, allowing users to easily track uptime and latency over time.
* **Deployment & Orchestration:**  
  The complete system will be deployed on **DigitalOcean Kubernetes (DOKS).** It will use Kubernetes Deployments, Services, Persistent Volume Claims, and an HTTPS Ingress for secure and reliable access.
* **Monitoring & Observability:**  
  The application will include monitoring for CPU, memory, and disk usage using DigitalOcean’s built-in tools. Additional dashboards will be created with Prometheus and Grafana to visualize custom performance metrics.
* **Advanced Feature #1 – Authentication & Security:**  
  User access will be secured with JWT authentication, password hashing, and HTTPS connections, ensuring that data and credentials are protected.
* **Advanced Feature #2 – CI/CD Pipeline:**  
  A GitHub Actions workflow will automate the build and deployment process, creating Docker images, pushing them to the DigitalOcean Container Registry, and applying updates directly to the Kubernetes cluster.
* **Optional (if time allows) Advanced Feature – Backup & Recovery:**  
  Automatic daily backups of the PostgreSQL database will be stored in **DigitalOcean Spaces** to protect against data loss and allow for easy recovery.

### Database Schema (Initial draft - work in progress):

To manage data efficiently and ensure the system remains stateful, the application will use a **relational database structure** built on **PostgreSQL.** The design focuses on reliability, clarity, and easy scalability. Each table has a specific purpose and connects logically to the others, forming the backbone of the monitoring system. Together, they allow the app to store users, the endpoints they want to monitor, performance measurements, and any alerts that are generated when performance thresholds are not met.

Below is a breakdown of the main tables:

* **Users**  
  This table stores information about each registered user, including their unique ID, email address, encrypted password, and the date their account was created. It ensures that each user can securely log in and manage only their own monitored endpoints.  
  (Columns:*id*,*email*,*password\_hash*,*created\_at*)
* **Endpoints**  
  Each endpoint represents a website, server, or API that a user wants to monitor. This table links directly to the Users table through a user\_id foreign key, ensuring each endpoint is tied to its owner. It stores the endpoint’s name, URL, and creation date.  
  (Columns:*id*,*user\_id*,*name*,*url*,*created\_at*)
* **Measurements**  
  This is the heart of the monitoring system. Each time the background worker runs, it records metrics for every endpoint, such as latency (in ms), current status (“up” or “down”), and a timestamp. Over time, this table builds a historical log that powers the performance charts and trends on the dashboard.  
  (Columns:*id*,*endpoint\_id*,*latency\_ms*,*status*,*timestamp*)
* **Alerts**  
  Whenever an endpoint’s performance falls outside acceptable limits, the system generates an alert. This table stores those alerts with a message, timestamp, and a flag indicating whether the issue has been resolved. It helps users quickly identify and respond to recurring or critical problems.  
  (Columns:*id*,*endpoint\_id*,*message*,*timestamp*,*resolved*)

The relationships between users, endpoints, and measurements make it easy to extend the system later, for example, by adding teams, regions, or notification preferences.

### Alignment with Course Requirements

This project was designed specifically to meet all of the technical requirements outlined in the ECE1779 course description. It uses Docker and Docker Compose for local development, ensuring that every part of the system, from the backend API to the database and frontend, can run in isolated containers. The use of PostgreSQL with persistent volumes demonstrates proper state management, guaranteeing that data is not lost during restarts or redeployments.

For orchestration, the application will be deployed on DigitalOcean Kubernetes (DOKS), which aligns with the course’s emphasis on modern container orchestration and scalability. Kubernetes Deployments, Services, and PersistentVolumeClaims will be used to manage components in a structured and reliable way. Monitoring and observability are addressed through both DigitalOcean’s built-in metrics and a Prometheus + Grafana setup for deeper insights into system performance.

Finally, the project includes at least two advanced features: secure authentication using JWT tokens, and a CI/CD pipeline with GitHub Actions that automates building and deploying containers to the cloud. These components collectively demonstrate an understanding of end-to-end cloud-native development.

**Project Scope and Feasibility**

The project scope has been intentionally defined to be achievable for one person within the two-month project window. Each component is small enough to be developed and tested independently but meaningful enough to demonstrate full functionality when integrated.

The technologies chosen, FastAPI, React, PostgreSQL, and Kubernetes, are lightweight, well-documented, and work seamlessly together. With around 30 hours of focused work, it is realistic to complete all core features and at least two advanced ones while leaving time for testing and documentation. The modular structure also allows future improvements, such as email alerts or anomaly detection, without increasing the current workload. Overall, the scope is ambitious enough to showcase technical skill but narrow enough to ensure a polished, working final product.

## 3. Tentative Plan

Because I am completing this project individually, I will approach it as a series of structured development phases. Each phase builds on the previous one to ensure consistent progress and maintain a clear path from design to deployment.

In the **first phase,** I will focus on setting up the local development environment using **Docker Compose.** This includes creating the FastAPI backend, defining the PostgreSQL schema, and testing database connectivity. I will also prepare a basic React frontend to confirm that the full stack communicates correctly.

The **second phase**will involve preparing the **Kubernetes configuration** for deployment on DigitalOcean (DOKS). This includes writing the necessary manifests for Deployments, Services, and Persistent Volume Claims, and verifying that the application runs reliably in the cloud environment.

During the **third phase**, I will integrate **monitoring and automation**. I plan to use Prometheus and Grafana for detailed performance insights, along with DigitalOcean’s built-in monitoring tools. I will also set up a **CI/CD pipeline** using GitHub Actions to automate image builds and deployments.

The **final phase** will focus on **security, optimization, and documentation**. I will implement JWT authentication, ensure HTTPS is enabled through an Ingress controller, test all major workflows, and prepare the final report with screenshots and architecture diagrams.

The modular structure also means that each component can be built and validated independently before being integrated into the full system.

## References

* **ECE1779 Course Project Handout** – University of Toronto, Department of Electrical & Computer Engineering.  
  <https://www.eecg.utoronto.ca/~cying/courses/ece1779-cloud/project/handout>
* **FastAPI Documentation** – Official documentation for building APIs with Python.  
  https://fastapi.tiangolo.com
* **DigitalOcean Kubernetes (DOKS) Documentation** – Guides and best practices for deploying containerized applications in the cloud.  
  https://docs.digitalocean.com/products/kubernetes/
* **Grafana and Prometheus Helm Charts** – Community Helm charts for monitoring and observability.  
  <https://github.com/prometheus-community/helm-charts>
* **GitHub Actions for DigitalOcean Deployments** – Reference for building CI/CD pipelines using GitHub Actions and DigitalOcean Container Registry.  
  https://docs.digitalocean.com/reference/doctl/