SISTEME TOLERANTE LA DEFECTE
Tema #1 Site web cu chat și IA peste Kubernetes

Termen de predare: Ultimul laborator

Obiective

Scopul acestei teme este de a implementa, folosindu-vă de Kubernetes si de diferite tehnologii pentru backend, baze de date și frontend, un site web ce conține un chat și o componentă ce va permite folosirea unei tehnologii de IA pusă la dispoziție de Azure.

Enunț

Această temă este personalizată pentru **BADEA ALEXANDRU-GABRIEL**.

Obiectivul temei este de a crea un website ce conține o aplicație de chat și o aplicație de IA. Tema va fi implementată folosind mai multe deployment-uri gestionate de un cluster de Kubernetes. Arhitectura acestei aplicații va cuprinde mai multe elemente:

1. Site-ul web efectiv va fi ținut pe un content management system (CMS).

- Acestei teme îi este asignată **Vveb** cu **5** replici.
- Site-ul va fi expus pe portul **80**.
- Puteți folosi webgui-ul/sau abordarea code-first pentru a construi un mic site pentru un magazin, o pizzerie, o postare de tip articol blog. Alegerea e a voastra (nu pierdeți mult timp dar nici nu lăsați pagina goală).
- Sistemul de CMS va folosi o bază de date proprie, pe aceasta o puteți identifica în documentația specifică. Unele CMS-uri permit mai multe variante de baze de date. Alegerea este la latitudinea voastră.

3. Sistemul de chat va fi introdus în pagina web a CMS-ului folosind un iframe html.

- Implementarea backend-ului de chat se va realiza folosind protocolul **WebSocket**. Codul pentru acesta se va afla pe un server web. Acestei teme îi este asignată folosirea **Go+Nginx** cu **2** replici.
- Serverul de chat va fi expus pe portul **88**.
- Partea de client va fi implementată folosind un framework de frontend. Acestei teme îi este asignată folosirea **React** cu 1 replică.
- Clientul de chat va fi expus pe portul **90**.
- Pentru stocarea mesajelor in baza de date, se vor salva următoarele: numele utilizatorului sursă, mesajul în format text, ASCII; timestamp-ul trimiterii mesajului.
- Pagina de chat va conține un formular în care se introduce un mesaj text, și va avea un buton de trimitere. Vor fi afișate și mesajele din trecut (aflate în baza de date) în ordine cronologică.
- Chatul va fi introdus în pagina CMS-ului printr-un **iframe**.

4. Aplicația de IA va fi introdusă în pagina web a CMS-ului folosind un iframe html.

- Această aplicație va reprezinta o pagină web ce va permite upload-ul unui fișier care va fi procesat apoi de un sistem de IA.
- Partea de client va fi implementată folosind un framework de frontend. Acestei teme îi este asignată folosirea **React** cu 1 replică.
- Se va menține un istoric cu toate cererile realizate și rezultatele obținute.
- Fișierele vor fi stocate folosind **blob storage-ul Azure**.
- Informațiile despre fișiere (nume, adresa blob, timestamp, rezultat procesare) vor fi stocate într-o **bază de date SQL hostată în Azure**.
- Fișierele vor fi procesate folosind un serviciu de IA. Acestei teme îi este asignat serviciul **OCR**.
- Aplicația de IA va fi introdusă în pagina CMS-ului printr-un **iframe**.

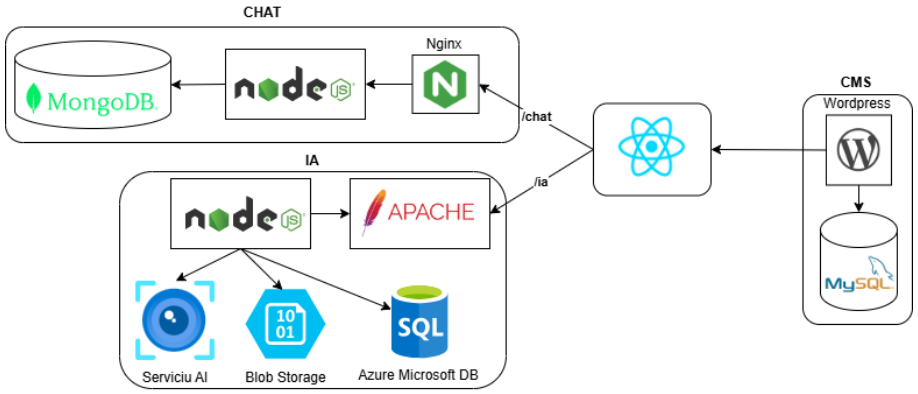

Figure 1: Exemplu de arhitectura pentru aplicatia in discutie

Figura **??** prezintă o arhitectură în care layer-ul de prezentare este bazat pe **Wordpress** cu baza de date **MySql**. Backend-urile pentru chat și aplicația de IA sunt dezolvate în **Node.js**. Chat-ul folosește ca bază de date **MongoDB** și expus prin **Nginx**. Aplicația de IA are un backend implementat în **Node.js** expus prin **Apache**, și folosește serviciul de tip **Computer vision - form recognizer**, iar stocarea persistentă a rezultatelor se realizează cu ajutorul unui **Blob Storage** și a unei baze de date **Azure SQL Database**. Funcționalitățile ambelor aplicații sunt expuse printr-o interfață grafică dezvoltată în **React** ce este integrată în cadrul CMS-ului.

Detalii implementare

Pentru majoritatea componentelor este recomandat să folosiți containere de pe Dockerhub. În general le puteți folosi cu doar mici modificări. Este posibil să fie nevoie să vă creați propriile imagini. Se vor avea în vedere Dockerfile-urile de tip multi-stage pentru reducerea dimensiunilor.

Pentru containerul care ține serverul web ce expune chat-ul va trebui să implementați voi codul și să creați o imagine custom. Atenție, orice imagine custom va fi stocată de un **registry privat** al clusterului.

Tema va consta într-o serie de fișiere de tip .yaml (și alte fișiere de configurare inițiale). Întreaga arhitectură va trebui pornită dintr-o singură comandă **apply**. Odată executată comanda **apply** totul va trebui să funcționeze. **Când prezentați nu va fi permis să faceți nicio modificare după ce dați apply, nici măcar la CMS**.

Pentru fiecare componentă se va crea un folder în care se va afla fișierul Dockerfile alături de oricare alte fișiere (cod, export bază de date, ș.a.m.d.).

Prezentare și punctare

În ziua prezentării, mașinile Kubernetes (două) trebuie să fie pregătite și pornite. La fel și addon-urile (de ex. registry). Containerele ce necesită acest lucru trebuie să fie deja build-uite și puse în registry.

În momentul prezentării va trebui să:

- Arătați că nu e niciun obiect creat pe cluster;
- Aplicați .yaml-ul pe cluster;
- Intrați pe site și arătați că toate aplicațiile funcționează corect;
- **NU** aveți voie să faceți nicio configurație la niciuna din componente (nici cms).

Distribuția punctajului este următoarea:

- 20 puncte - Chat-ul funcționează conform specificațiilor;
- 20 puncte - Aplicația IA funcționează conform specificațiilor;
- 30 puncte - CMS-ul integrează tot și funcționează conform specificațiilor;
- 20 puncte - Veți primi câteva întrebări și cerințe cu instrucțiuni cu operații peste cluster din linia de comandă; Punctajul se primește doar dacă este evident că știți ce faceți și dați comenzile rapid. Va exista un timer. Acest punctaj se dă doar dacă tema funcționează.

O arhivă .zip cu toate fișierele va fi pusă la dispoziție pentru verificări anti-plagiat.