**הדרך שעברתי**

1. יצרתי ריפו של GITHUB בקישור שמצויין בREADME
2. רציתי להעלות GKE אך העלתי MINIKUBE למען הפשטות. בהמשך כשראיתי שיש TEMPLATE בGITHUB לGKE אמרתי לעצמי שזאת אחלה הזדמנות ללמוד איך לעשות את זה גם בGKE(ובאופן כללי בGCP) אז הרמתי שם.
3. כחלק מהשימוש בGKE יש להם כלי בשם GCLOUD שמשנה את הkubeconifg ככה שיפנה לקלאסטר המתאים, דיי נוח
4. כתבתי בFLASK תוכנית פשוטה שמדפיסה hello world ב/
5. אחרי מחשבה האם להשתמש בflask run שרץ כdev mode או כPROD על web server, החלטתי להמשיך עם flask run בגלל שאני לא רואה בהמשך המטלה בקשות לשימוש בסביבות שונות וקדומה ולכן אני מבין שאין כוונה לקחת את זה לכיוון הזה. הdockerfile עצמו דיי פשוט, מסביר את עצמו + יש שם הסברים בכל מקרה
6. בהתחלה כשהשתמשתי בMinikube אז דחפתי לhub.docker אבל החלפתי את זה לארטיפקט מנגר שהעלתי על הGCP שמחובר לGKE
7. טוב אז התחלתי מלהשתמש בTEMPLATE שGITHUB מביאים(בחרתי GITHUB כי לא יצא לי לבנות שם PIPELINE לפני ואמרתי עוד הזדמנות טובה + אם כבר הכל בGITHUB אז לא נסבך את המצב, יהיה יותר קל לכם גם לעקוב ולראות את הכל במקום אחד)

כדי שהGIT יוכל לתקשר עם GKE יש צורך לקנפג workload identity provider(או בדרך פחות מגניבה של credentials ששומרים בSECRETS אבל workload identity provider היה יותר כיף)

העברתי דברים לENV וSECRETS ושיניתי שיעבוד עם DOCKER COMPOSE.

הוספתי שלב לTEST

הורדתי את השימוש בסביבה בגיט כי כפי שאמרתי קודם אנחנו הרי על סביבה אחת אז מיותר.

שומר על הpipeline כמה שיותר גנרי

1. עם הלמעלה
2. שמתי את הבדיקות באותו IMAGE שגם יבוצע לו PUBLISH ולאחר שיצרנו אותו פשוט עשיתי docker compose run שיריץ pytest שבתכלס בודק אם חזרה לנו תשובה והיא מה שמצופה(hello world). יש מלא דרכים לבצע את זה(סוגי בדיקות שונות, imageים שונים, multi stageים, docker composeים שונים וכו וכו) אבל בסוף החלטתי לשמור על הפשטות ולא לבצע OVERENGINEERING.
3. בTEMPLATE של גיט הם השתמשו בkustomize שלא הכרתי ונורא הסתקרנתי אז למדתי והשתמשתי בו גם