**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. אני מתייחס לכל התהליך כאילו אנחנו מעלים לסביבת DEV כדי למנוע סיבוכים מיותרים כשאין לנו ריבוי סביבות ולכן אני מעלה את הFLASK כflask run ובלי gunicorn וngnix . 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

הוספתי גוב נוסף לROLLBACK שיוסבר בחלק הרלוונטי בהמשך

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. אשמח לקבל הבהרה בנוגע לסעיף, כי אין ביכולות הכלי לנטר את הPIPELINE עצמו(הניטור מתבצע בשני דרכים, או שמריצים משהו על AGENT או ששולחים את המידע מתוך סקריפט PYTHON שרץ. זאת אומרת שאין איך לנטר את הPIPELINE שהוא בתכלס ריצה על המכונה בGITHUB ואם אני מחבר TASK לשרת FLASK הרי שבכל מקרה לא נקבל את המידע מהריצה של הPIPELINE אלא רק כשיעשה DEPLOY וגם אז אנחנו נמשיך לקבל באופן קבוע.. הכלי לא מותאם לזה)
4. בTEMPLATE של גיט הם השתמשו בkustomize שלא הכרתי ונורא הסתקרנתי אז למדתי והשתמשתי בו במקום HELM
5. אין פה יותר מדי אחרי שיוצרים את הkustomiz, עושים לו BUILD וAPPLY
6. הוספתי HPA לkustomiz, שמרתי עליו פשוט ממש, האפליקציה שלנו לא באמת צורכת משאבים אז אין באמת שינוים
7. טוב אז פה היה תהליך מעניין דווקא. בגלל שהשתמשתי בGKE AUTOPILOT(אני במשתמש החינמי אז רציתי לחסוך עליות) הוא מגיע מובנה עם managed Prometheus for gke אבל זה לא באמת מגיע out of the box. העלתי את כל הרכיבים שלו על namespace בשם prom ואפשר לראות את הרכיבים הקיימים בgke-gmp-system. היה צורך להתקין את prom עצמו, podmonitoring, service account, ui כדי שנוכל ליחצן לGRAFANA. היו הרבה סיבוכים קטנים בדרך שהגיעו מעצם זה שהGKE הוא במצב AUTOPILOT והייתי צריך לברור הרבה בדוקומנטציה כדי להבין מה בדיוק גורם להם ולסדר את זה.

אחרי זה העלתי GRAFANA חינמית באתר שלהם, חיברתי את הPROM ובגלל ששוב אנחנו בAUTOPILOT אז לא יכולתי להשתמש בdashboardים הקיימים כי כולם מבקשים לבחור NODEים(הAUTOPILOT לא מייחצן את זה בכלל אז הPROM כאילו ולא קיימים NODEים בכלל) אז יצרתי DASHBOARD בשם MAIN שיש בו סתם שני מטריקות פשוטות כדי לראות שזה עובד.

1. הוספתי JOB שהמטרה שלו היא הROLLBACK, בתכלס רציתי לשמור על פשטות כי יש מיליון ואחד דברים שאפשר לעשות אז אחרי שאנחנו עושים deploy בגוב הקודם, הגוב שלנו בודק שאין שגיאות בלוגים בDEPLOYMENT ואם יש הוא מבצע kubectl rollout undo ומחזיר אותנו אחורה(ומקריס את הגוב שנראה שהוא לא צלח בGIT שזה אומר שיש שגיאות)