הרצאה 2:

Amazon Web Services - AWS

היא יחידה עסקית של חברת אמזון האמריקאית, העוסקת באספקת שירותי מחשוב ענן ליחידים, לחברות ולגופים ממשלתיים ונחשבת לבעלת שירותים מתקדמים בתחום. הטכנולוגיה מאפשרת ללקוחות של AWS לשכור מיקום לאחסון ממוחשב, אשר זמין בכל עת באמצעות האינטרנט.

Amazon Elastic Compute Cloud (EC2)

שירות AWS המאפשר להשכיר מחשבים וירטואליים ובכך לחסוך בהוצאות על חומרה. באמצעות EC2 הלקוח יכול להריץ מכונות וירטואליות לצורך שימושו.

סביבה וירטואלית של EC2 נקראת instance וישנם חמישה סוגים עיקריים:

- General Purpose
- Compute Optimized
- Memory Optimized
- Storage Optimized
- Accelerated Computing

ביצירת instance לוקחים בחשבון (בין היתר) את מאפייני החומרה הבאים:

- CPU (ARM, Intel, AMD)
- GPU (Nvidia, Intel)
- TPU
- Metal instances on AWS
- FPGA-based nodes

Amazon Elastic Block Store (Amazon EBS)

מתפקד ככונן אחסון עבור instances של EC2.

Amazon Simple Storage Service (Amazon S3)

שירות שמאחסן נתונים עבור משתמשים בענן.

Amazon Relational Database Service (Amazon RDS)

מאפשר למפתחים ליצור ולנהל מסדי נתונים יחסיים בענן. Amazon RDS מאפשר למפתחים לעקוב אחר כמויות גדולות של נתונים ולארגן ולחפש באמצעותם ביעילות.

Testing

בדיקות תוכנה הן תהליך שנועד להעריך את איכותה של תוכנה ועמידתה בדרישות שהוצבו לה. בדיקות תוכנה מהוות חלק אינטגרלי מתהליכי הנדסת תוכנה והבטחת איכות תוכנה.

Testing types

בדיקות יחידה (Unit) בדיקות ברמת יחידת תוכנה (לרוב פונקציה, מודול).

בדיקות אינטגרציה (Integration) בדיקת שילוב יחידות תוכנה בהיקפים שונים, החל משתי יחידות ועד לכלל היחידות במערכת.

בדיקות מערכת (System) בדיקת המערכת בכללותה, בדרך כלל בראיית המשתמש של יכולות המערכת.

בדיקות קבלה (Acceptance) בדיקות הנעשות על ידי המשתמש או הלקוח במטרה לוודא כי המערכת פועלת בהתאם לדרישות שהוגדרו במסמך הדרישות המקורי ובשינויי דרישה (change request) שהועברו במהלך מחזור חיי הפיתוח.

בנוסף מקובל לסווג את הבדיקות גם לפי החלוקה שמתבססת על אופי הבנת המערכת ורמת הירידה של הבודק לפרטים:

בדיקות קופסה לבנה (White Box) בדיקות אלו מתבססות על הכרת קוד המקור של התוכנה ובניית תוכניות בדיקה המותאמות לנתיבי הזרימה האפשריים של הקוד. בדיקות יחידה עשויות להיות בדיקות מסוג קופסה לבנה. בסוג בדיקות זה, על הבודק להכיר את הלוגיקה של הקוד, וכן, עליו להיות בעל ידע בשפת התכנות בה כתובה התוכנה.

בדיקות קופסה שחורה (Black Box) בדיקות אלו אינן מכירות את המבנה הפנימי של המערכת ומתבססות על בדיקת הפלט הצפוי לקלט מסוים בהתאם לתכנון מוקדם כלשהו. בדיקות קבלה מתבצעות בשיטה זו בדרך כלל. בסוג בדיקות זה, הבודק חייב לדעת את פירוט דרישות המערכת, וכן, עליו לדעת לאיזה פלט מהתוכנה עליו לצפות עבור קלט מסוים. עם זאת, הבודק אינו חייב להכיר את הלוגיקה של הבעיה או אפילו לדעת את שפת התכנות בה היא כתובה.

בדיקות קופסה אפורה (Gray Box) בדיקות אלו מכירות במבנה הפנימי של המערכת אך משתמשות בידע זה על מנת לבצע בדיקות בסגנון קופסה שחורה. כך לדוגמה שינוי של מאגרי הנתונים לבדיקת פלט מסוים או שימוש בהנדסה הפוכה על מנת לאתר את גבולות הפעולה של המערכת, אלו הן דוגמאות נפוצות לבדיקות בשיטה זו.

- בקורס זה נבצע בדיקות יחידה ואינטגרציה. לצורך בדיקות יחידה ניתן להשתמש בpytest.

Continuous Integration/Continuous Delivery - CI/CD

הוא מכלול שיטות העבודה, הכלים והאוטומציות המהווים את עמוד השדרה הטכני של פיתוח התוכנה הזריז. כלי CI/CD מאפשרים פיתוח תוכנה רציף, המצמצם ככל האפשר את הזמן העובר בין הוספת תכונה או יצירת שינוי בקוד התוכנה, לבין הגשת גרסה חדשה ויציבה של התוכנה ללקוח.

בעצם זוהי אינטגרציה רציפה של הקוד החדש לקוד המקור הקיים של התוכנה, ומסירה רציפה ללקוח של גרסת התוכנה החדשה.

במתודולוגיית הפיתוח האג'ילי ובתרבות הדבאופס, עקרונות הCI/CD מהווים מרכיב מרכזי, וכוללים תמיכה במערכות המאפשרות פיתוח, בדיקות, אינטגרציה ופריסה רציפים ומתמשכים של התוכנה, לאורך כל מחזור חיי הפיתוח שלה.

Pipeline

במהלך פיתוח תוכנה, ישנם תהליכים מוגדרים רבים שעל קוד המקור של התוכנה לעבור, לפני שתוכל התוכנה להימסר ללקוח.

מטרת כלי ותהליכי CI/CD היא לעשות אוטומטיזציה לתהליכים אלה, ולחבר בין תהליכים שונים, כך שהתוצר של כל תהליך הוא בסיס העבודה של התהליך שלאחריו, וסופו של כל תהליך מניע את הפעלת התהליך הבא. מכלול התהליכים האוטומטיים שעובר קוד המקור של התוכנה נקרא CI/CD pipeline, או הפייפליין.

CI/CD מגשר על הפערים בין צוותים שונים, על ידי אכיפת אוטומציה בבניית, בדיקת ומסירת התוכנה (build, test and deploy).

תהליכים אוטומטיים שעשויים להיות חלק מתהליך הפייפליין הינם:

- בנייה של קוד המקור של התוכנה יחד עם השינוי החדש, ווידוא שהשינוי איננו יוצר בעיות קומפילציה. (push / pull request)
- מיזוג הקוד המכיל את השינוי החדש בגרסת התוכנה המיועדת לשחרור ללקוח. (merge)
- התקנת התוכנה בסביבה המתאימה והרצת בדיקות אוטומטיות על המוצר.
- התקנה של התוכנה החדשה על שרתי בדיקות ידניות, שישמשו את אנשי ה QA בתהליך עבודתם ביום העבודה הבא.

כישלון בביצוע אחד מן השלבים ימנע את מעבר התוצר לשלב הבא, ויחייב את המפתח או את הבודק לתקן את החלק המכשיל את הפייפליין. לאחר בדיקות התוכנה בכללותה, וקבלת אישור מצוותי בקרת האיכות, תהליכים נוספים עשויים להיות מופעלים כדי להתקין את עדכוני התוכנה אצל הלקוח.

GitHub Actions

פלטפורמה של גיטהב לצורך CI/CD המאפשרת ליצור build, test, and deployment pipeline.
באמצעות פלטפורמה זו ניתן ליצור workflows עבור תהליכי התוכנה השונים ביניהם Pull Request, merge, push ועוד.

FastAPI

סביבת מערכת ווב (Web framework) לצורך פיתוח מוצרי RESTful APIs בפייתון.
מאפשרת שליחה ועיבוד של בקשות מרובות בו זמנית.
בעצם יהווה כשרת שלנו בקורס זה.

REST

התפיסה הארכיטקטונית ב-REST היא תפיסת שרת-לקוח. תפיסה זו מחייבת קיום לקוח ושרת. לקוחות יוזמים פניות המכילות בקשות לשרתים. השרת מעבד את הפנייה, ומחזיר תגובות מתאימות. הייחוד של REST הוא בכך שהתוכן המועבר מהלקוח לשרת הוא ייצוג של משאב(בד"כ בXML, HTML או JSON).

בכל מצב נתון הלקוח יכול להיות בתהליך של שינוי מצב או במצב מנוחה (rest). במצב של מנוחה הלקוח יכול להיות באינטראקציה עם המשתמש, אבל אינו תופס משאבים בשרת. הלקוח שולח פניות כאשר הוא מוכן לעבור למצב חדש. כאשר קיימת פנייה אחת או יותר שטרם הסתיים הטיפול בהן, הלקוח נמצא במצב של מעבר ממצב למצב. כאשר הלקוח נמצא בתהליך של שינוי מצב, השרת מעבד את המסמך המתאר את מצב הלקוח. המסמך עשוי להכיל קישורים, שעשוי להיות בהם שימוש בפעם הבאה שהלקוח יבקש לשנות את מצבו.

ארכיטקטורת REST פותחה במקביל ל-HTTP 1.1. למרות זאת REST היא ארכיטקטורה כללית הניתנת למימוש גם בסביבות אחרות ולא רק תחת HTTP.

Asyncio

מודול בפייתון אשר מספק framework לפיתוח single-threaded בצורה אסינכרונית (שמזכיר מעט multi-threaded development).

Monolithic application

תיאור מערכת אשר מפותחת במקום אחד (צד שרת, לקוח, גישה לDB).
דוגמאות למערכות מונוליטיות: הקרנל של לינוקס, WSL, Django.
כלומר, כל הקומפוננטות באותו המקום אך מפותחות באופן מודולרי.

Microservices

מערכת שמבוזרת לservices אשר מתקשרים ביניהם באמצעות אינטרפייס.
כלומר כל קומפוננטה היא מוצר בפני עצמה שבסופו של דבר היא מודולרית ותהפוך לדוקר קונטיינר.

עקרונות של הנדסה נכונה

- **Measure twice and cut once** עקרון האומר שלפני כתיבת פונקציונלית של התוכנה, יש לחשוב היטב על בחירת הבעיה המתאימה, הגישה להתמודדות עם הבעיה, בחירת הכלים הנכונים לפתרון הבעיה, תכנון הצוות המתאים וביצוע כל המדידות הנדרשות לפני שמתחילים לכתוב קוד.
- **Don’t Repeat Yourself (DRY)** עקרון האומר שאין לשכפל קוד ולעבוד באופן מודולרי, גנרי ותקין. ככל שמתרגלים לגישה הזו מוקדם יותר, כך זוכים לתוצאות טובות ויעילות יותר במהלך הפיתוח כולו.
- **Keep It Simple Stupid (KISS)** עקרון האומר לשמור על הפשטות כמה שניתן.

- **You Aren't Gonna Need It (YAGNI)** עקרון האומר לפתח רק את הנדרש. למפתחים ישנה נטייה לממש הכל רק שלא תמיד יש לזה צורך (בזבוז זמן ויעילות).
- **Avoid Premature Optimization** עקרון האומר שיש להימנע מאופטימיזציות לפני מימוש הלוגיקה.
- **Principle Of Least Astonishment** עקרון האומר שהקוד צריך להיות ברור ומובן מאליו גם עבור מפתחים אחרים.
- **S.O.L.I.D.** עקרון האומר שכל מודול (מחלקה, או ספריה) עושה דבר אחד בלבד, שיהיה פתוח להרחבות אך סגור לשינויים של מודול קיים. הירושה מהמחלקות לא תדרוס התנהגות קיימת אלא תוסיף את הלוגיקה המשלימה למחלקת האב. השורה התחתונה שיש ליצור קוד שקל לתחזק לאורך זמן רב.
- **Law of Demeter** עקרון האומר שיש לחלק את אזורי האחריות בין מחלקות ולבצע אנקפסולציה של הלוגיקה באמצעות מחלקות, מתודות או מבנה נתונים אחר. שני עקרונות הבאים לידי ביטוי כאן הם: **Decoupling** האומר שיש לצמצם כמה שניתן את מספר החיבורים בין מחלקות וישויות שלא קשורות אחת לשנייה. **Cohesion** האומר שיש לשמור מחלקות שיש ביניהן לוגיקה משותפת תחת package יחיד ומשותף.

NGINX

הוא שרת אינטרנט שמתמקד בעיקר בשימוש בזיכרון מועט ובביצועים גבוהים. בנוסף, הוא יכול לשמש כשרת פרוקסי הפוך עבור הפרוטוקולים HTTP, SMTP, HTTPS, POP3 ו-IMAP.

Apache

הוא תוכנת השרת HTTP הנפוצה בעולם.

שרת האפאצ'י הוא אחד ממוצרי הדגל המוקדמים של קהילת הקוד הפתוח. עיצובו המודולרי מאפשר רמה גבוהה של התאמה לצורכי כל אתר אינטרנט. אפאצ'י תומך במגוון רחב של הרחבות ("מודולים") ועיצובו ידידותי למפתח ולמנהל האתר. שרת האפאצ'י מסוגל לפעול על מערכות הפעלה רבות.