

רשימת מושגים

1 עקרון SOC, מודולציה, צימוד ולכידות
2 ביזור, N-Tier, Scale UP vs Scale Out, מודל CAP
3 מהו עיצוב תוכנה ומהי ארכיטקטורת תוכנה
4 מודל DIKW
4 ייצוג ידע מבוסס רכישה ממוחמה באמצעות Ontology
4 ייצוג ידע על בסיס מודל נלמד מנתונים
4 מהו מחשוב ענן
4 מהו שירות תוכנה
4 IaaS, PaaS, SaaS, XaaS
5 Microservices, Pros and Cons
6 HTTP
6 REST
7 מהו פרוטוקול WebSocket ומה ייעודו
7 מהו Docker ומרכיביו (Image, Container, Engine, Registry), מהו Docker File
8 סוגי מסדי נתונים NoSQL, הסיבות להופעתן, דוגמאות לתשתיות
8 מאפייני מסד נתונים מבוסס Key-Value, ומאפייני מסד נתונים מבוסס מסמכים
8 מהי Transaction ותכונותיה (ACID), מהו מדד TPS
9 BASE
10 הגדרות Big Data, ההבדל בינו לבין Small Data
10 מודל ה-Vים לתאור אתגרי ביג דאטא
10 מהו Data Pipeline ושלביו
11 מהו Apache Hadoop, מרכיביו, ומטרותיו
11 מהו Map-Reduce (האלגוריתם ואסטרטגיית עיבוד, לא רק מנוע העיבודים של Hadoop)
12 מהו Spark, מרכיביו, ההבדל העיקרי בינו לבין Apache Hadoop
13 מהו מתווך מסרים, מאפייני Kafka
13 הגדרה מהי למידה
13 ההבדל העיקרוני בין תכנות קלאסי ללמידת מכונה
14 2 עקרונות לסוג למידה מפוקחת, מהי למידה מפוקחת (עץ החלטה, רגרסיה לינארית)
14 2 עקרונות לסוג למידה לא מפוקחת, מהי למידה מפוקחת (ניתוח אשכולות, חוקי אסוציאציה)
14 הגדרה מהי אפלקציה מונוליטית
14 ארכיטקטורת למבדה, ארכיטקטורת קאפקה וההבדל העיקרי ביניהם

עקרון SOC, מודולציה, צימוד ולכידות

עיצוב של תוכנה: תכנון של פתרונות בהינתן סט של אילוצים

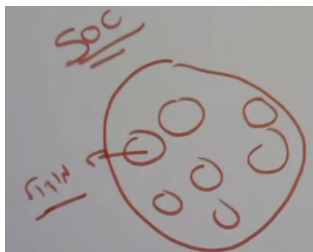
ארכיטקטורה: כל ההחלטות שקשה לסגת מהם בלי לשלם מחיר כבד עד כדי בלתי אפשרי

אפשר להגיד שארכיטקטורה זה סוג של עיצוב תוכנה

Spiral of Concerns (SOC) – עקרון שאומר לפצל את השלם לחלקיו

כלומר חלוקת השלם לחלקיו ויצירת מודלים, בדרך זו יותר קל לנהל את המכלול,

זהו מפתח לכל יצירה של תוצרים.



מודולציה – חלוקה טובה למודלים

חלוקה טובה למודולים על פי המדדים הבאים: (מדד)

1. Coupling (צימוד) – נסתכל כאן על זוגות של מודולים

כמה המודולים (Modules) תלויים ביניהם, נרצה מינימלית

נרצה מערכת שבה המודולים הם *tightly coupled* במקום *lightly coupled*

על מנת לפתור צימוד (*tightly coupled*) אפשר להוסיף שכבה אבסטרקטית שנקראת ממשק, Interface.

ממשק – מחיצה אבסטרקטית אשר אומרת מה צריך ולא איך

2. לכידות (פיקוס) – הסתכלות על מודל ספציפי, האם הוא מטפל בנושא אחד בלבד או בכמה דברים

פתרונות לדף המושגים הרלוונטיים לבחינה

יוצרת בעיות של תחזוקה, הבאת קוד לא רלוונטית ועוד.

ניהול Dependencies (האופן שבו מודלים תלויים אחד בשני)

עיצוב תוכנה זה בעצם ניהול תלויות, אם מנהלים נכון את התלות בין כל מודולים אז כל השאר יותר קל.

המשמעות של עיצוב תוכנה זה ניהול Dependencies \Leftrightarrow

תוכנה – אוסף של שירותים דיגיטליים שמשרתים מטרה כלשהי

יש שירותים שהם כללים כמו ניהול זיכרון, קבלת קלט וכו' וכאשר אוספים אוסף של שירותים שכאלה ליעוד מסוים

אז זה נקרא אפליקציה (אוסף של שירותים יעודים למשימה מסוימת)

שירות תוכנה – אבולוציה של המושג אובייקט, יכול לעבוד עבור כל פלטפורמה

חומים – נניח לעשות אינטגרציה בין מערכים בלינקדס לווידוס, יש בעיה בחיבור הלוגי.

ביזור, N-Tier, Scale UP vs Scale Out, מודל CAP

ביזור: סוג של מודל מחשוב, כמות של מחשבים (≤ 2) אשר עובדים **יחד** על מנת לבצע משימה כלשהי, מערך עם

$n \in \mathbb{Z}^+$ מחשבים נקראת מערכת $n - tier$.

משמעות (למה ביזור?): לעשות מקבילות

Scaling: להגדיל את התפוקה של המערכת, בלי שינויים בתוכנה אלא רק בחומרה, יש 2 סוגים

Scale UP: לקחת מחשב, tier ובעצם לשנות אותו מבחינת חומרה, לדוג' להוסיף זיכרון או מעבדים לאותו מחשב,

שינוי זה אינו מומלץ היות ויכול להיות צוואר בקבוק כלשהו כך שהתפוקה לא תמשיך לגדול

Scale Out: להוסיף מחשבים, שרתים וכו', להגדיל את הקיבולת של המערכת (**הרבה יותר רציני ומשמעותי מ-UP**)

Scale UP vs Scale Out

הדמיה:

Scale Out



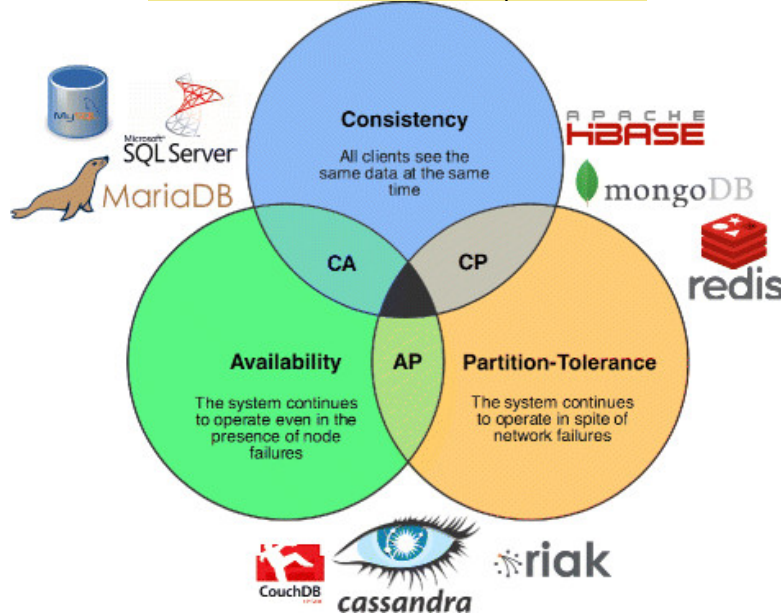
Scale Up



ההבדלים: מוצג למעלה בהגדרות מה היתרונות והחסרונות

מודל CAP

פתרונות לדף המושגים הרלוונטיים לבחינה



Consistency: כל פעם שאנחנו מבצעים קריאה כלשהי, אנחנו **תמיד** מקבלים את המידע הכי עדכני או שנקבל שגיאה, מדובר על כל הצמתים אשר מחוברים למערכת מבוצרת, אם יש מערכת אחת שלא הצליחה אז נבצע ROLL BACK

Availability: כל פעם שאנחנו מבצעים קריאה כלשהי, אז נקבל **איזשהי** תשובה, או שהיא הכי מעודכנת או שלא, במילים אחרות, לכל בקשה של הקליינט, נקבל תשובה כלשהי, לכל עיקרון זה מחייב שתתבצע תגובה בזמן סביר

Partition Tolerance: המערכת יכולה להתנהל על כמה מחשבים שונים כאשר יש **רשת ביניהם שהיא לא מאובטחת, ולא מובטח עליה שום דבר**, ולכן יכול להיות שיש הודעות שיגיעו באיחור או לא יגיעו בכלל, המערכת תעבוד כשהיא מבוצרת.

חשיבות תיאוריית CAP - מאז 2005 כמות המידע באינטרנט גדל באופן **אקספוננציאלי**. לכן הדרך היחידה להתמודד עם זה הייתה Scale Out וכך בעצם נוצרה רשת מבוצרת. כדי לשמור על הרשת תקינה חייבים ליישם את עקרונות CAP. אולם אי אפשר לקיים את שלושת העקרונות בו זמנית אלא רק שניים. לכן כל מערכת תבחר איזה 2 עקרונות ליישם בו זמנית בהתאם לדרישות ולסדר העדיפויות הספציפית שלה.

שפות NoSQL יכולות לקיים רק 2 מהם בו זמנית, ומקיימות מודל **BASE**, בעוד ששפות SQL מקיימות מודל **ACID** (סיכום מסדי נתונים)

מהו עיצוב תוכנה ומהי ארכיטקטורת תוכנה

עיצוב תוכנה:

תהליך של פתרון בעיות ובתכנון של פיתוח תוכנה, ז"א פיתוח התכנון ודרך העבודה של התוכנה, נעשה לאחר שנקבעים מטרות והמפרט. התהליך של עיצוב התוכנה משלב בתוכו מרכיבים של תכנות Low-Level, יישום אלגוריתמים, ארכיטקטורת תוכנה ותכנון מופשט של מערכי התוכנה. **עיצוב הוא אחד מהשלבים בפיתוח תוכנה, תכנון של פתרון בהינתן סט של אילוצים**

ארכיטקטורת תוכנה:

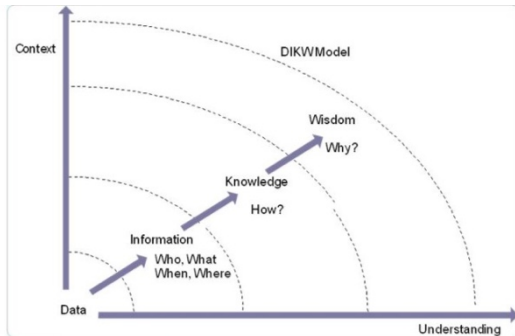
ארכיטקטורה היא התחום העוסק בתכנון מערכות תוכנה. המונח ארכיטקטורה בהנדסת תוכנה פירושו ייצוג היבטים שונים של התוכנה באופן מופשט. ארכיטקטורה של מערכות תוכנה היא לפיכך תכנון מופשט של ההיבטים השונים של התוכנה, היחסים בין המרכיבים השונים של התוכנה והחוקים החלים עליהם.

פתרונות לדף המושגים הרלוונטיים לבחינה

החלטות עיצוביות קרדינליות מרכזיות שמקבלים ושקשה לסגת מהם בלי לשלם מחיר כבד עד כדי כך שלא ניתן לסגת מהם.

מודל DIKW

בבסיס הפירמידה קיימים נתונים, מעליהם מידע, מעל ידע ובראש החכמה. הפירמידה מוצגת בצורה הבאה :



:Data

אוסף של נתונים שאינם מאורגנים או מסודרים ולכן לא באמת נוכל להשתמש בהם

:Information

בשביל זה נוסף קרשים וסידור בסיסי של הנתונים, ובכך נוכל לענות על שאלות כמו "מי, מה, איפה ומתי" (מה שאנחנו מחפשים)

:Knowledge שלב זה מתבצע ע"י אנליזת המידע שיש לנו, ובכך נוכל לדעת איך להשתמש במידע

:Wisdom מאפשר לנו להבין עקרונות מה נכון ומה לא נכון. (זה העתיד, כל השאר זה העבר)

אינפורמציה: מנגנון שמוריד מאי וודאות (מדד ע"י אנטרופיה)

2 דרכים לרכוש ולייצג ידע:

ייצוג ידע מבוסס רכישה ממומחה באמצעות Ontology

תורת האנשים (ללכת למומחה)

ייצוג ידע על בסיס מודל נלמד מנתונים.

לשפוך נתונים לתוך מכונה ולהוציא נתונים (למידת מכונה)

יש פה התקשרות ל-Associations Rules שזה קשר בין אובייקטים, ע"י קשר אפשר לדעת מה לעשות ואיפה למקם נתונים/מידע מסויים בהסתברות גבוהה. לדוגמא כל מי שקונה פסטה קונה גם רוטב לפסטה, אז נשים אותם אחד ליד השני, או לחלופין ב-2 מקומות רחוקים ככה שהוא יעבור את כל הסופר בדרך ויזרוק לעגלה מלא דברים לא קשורים ויבזבז כסף (רווח לסופר)

מהו מחשוב ענן

הגדרה ראשונה: מודל מחשוב המשלב ביזור, עיבוד מקבילי, עלויות תפעול נמוכות ויעילות כללית בניצול משאבים עם יכולות נגישות חוצות פלטפורמות והסתלמות (Scalability) ניכרות, לצד מיקוד הלקוחות בצריכת שירותי מחשוב ישירות וללא פעילויות גלוות הכרוכות בייצור או תחזוקה.

הגדרה שניה: שם גג לצורת הפצה (Delivery) של משאבי מחשוב בצורת שירותים המבוססים על רשתות ומרכזי נתונים (Data Centers)

מהו שירות תוכנה

התחברות לשירות ללא ידע באיזה שפה או איך אלא שולחים את הנתונים שצריך ומקבלים את מה שרצינו. מתחלק לשני חלקים או קבלת נתונים או עיבוד נתונים. (לא מחייב ענן). להשלים

IaaS, PaaS, SaaS, XaaS

Paas – Platform As A Service

פתרונות לדף המושגים הרלוונטיים לבחינה

שירות המספק פלטפורמה להרצת אפליקציות או לפיתוח אפליקציות של המשתמש. במקרה זה, מקבל המשתמש גם משאבי מחשב וגם תוכנות תשתית הנדרשות לצורך הרצת או פיתוח מערכת.

- אירוח שירותי ענן
- אחסון וניהול נתונים
- שירותי מסרים
- שירותי ניהול משתמשים וזהויות

IaaS – Infrastructure as a Service

שירותים בהם המשתמש מקבל משאבי מחשב לשימוש דוגמאות למשאבי מחשב שאפשר לקבל:

- מכונות וירטואליות
- אחסון
- נתבים וירטואליים

ההבדל בין IaaS ל-PaaS:

ב-IaaS נותנים לך חומרים כמו מלט, לבנים וכו' לבניית בית, כאן מקבלים את הגמישות להפוך את הבית לאיך שאנחנו רוצים, דוגמא לשירותים: Microsoft Azure, AWS וכו' בעוד שב-PaaS, הבית בנוי עבורך, אתה רק צריך לרדת אותו, נוכל להריץ פה דברים שהבית תומך בהם, לא מקבלים את הגמישות, לדוגמא Google Apps Engine, הרוקו, וכו'.

SaaS – Software as a Service

תוכנה המספקת למשתמש שירותים הניתנים באמצעות אירוח באתר הספק, במקום רכישת מוצר תוכנה והתקנתו בשרתי הארגון הרוכש. הפעלת שירותי התוכנה מאתר הספק היא דרך רשת תקשורת, בדרך כלל האינטרנט. לדוגמא:

- Office 365
- Google Docs
- Salesforce

XaaS - Anything as a Service

מתייחס לגיוון ההולך וגדל של שירותים באינטרנט דרך חישוב ענן אשר לא מסופק באופן מקומי שיטה זו בין היתר תומכת בשיטת "שלם כאשר אתה משתמש" וכשאתה לא משתמש תפסיק לשלם

Microservices, Pros and Cons

המושג Microservices מייצג סגנון של ארכיטקטורת תוכנה בו אפליקציות מורכבות בנויות מיחידות קטנות ועצמאיות המתקשרות ביניהן באמצעות language-agnostic APIs השירותים קטנים, נבדלים זה מזה ומיועדים לביצוע משימות מוגדרות ומצומצמות.


יתרונות:

- נותן לפתח את החופש לתכנת בתוכנה עצמאית
- יכול להיות מפותח ע"י קבוצה קטנה של אנשים
- אפשר לכתוב בשפות שונות כל מיקרוסרווס
- קל לעשות אינטגרציה ע"י Jenkins, CircleCI וכו'
- קל להבנה ושינוי, קל עבור מפתח חדש בצוות
- מאורגן בצורה טובה יותר
- כאשר דרוש שינוי כלשהו, משנים רק את המיקרוסרוויס שמתייחס אליו ולא צריך לשנות את כל האפליקציה
- קל ל-Scaling
- קל לעשות אינטגרציה עם צד שלישי

פתרונות לדף המושגים הרלוונטיים לבחינה

- אין התחייבות, אפשר לייצר חדש בקלילות
- אמינות והתמודדות עם שגיאות טובה יותר, אם יחידה אחת נפלה אז לא כל המערכת נופלת

חסרונות:

- בגלל שהמערכת מבוזזת, הבדיקות יכולות להיות מסובכות
- ככל שכמות הסרוויסים עולה, יש יותר שגיאות של Network latency וכו'
- שהמערכת מבוזזת נגרם מאמץ מוכפל
- כאשר מספר המיקרו-סרוויסים גדל, ביצוע האינטגרציה וניהול כל המוצרים יכול להסתבך
- מפתחים צריכים לתת מאמץ נוסף על מנת ליישם מכניזם של תקשורת בין השירותים
- חילוק האפליקציה למיקרו-סרוויסים הוא סוג של אומנות 
- קושי בלמצוא את הבעיה

REST

קונצפט אשר מציג ארכיטקטורה למערכות רשת, מקל עלינו לעשות מערכת שמבוססת על הרשת. יחידות קצה אשר מדברות מעל רשת מדברות בעזרת **REST API**

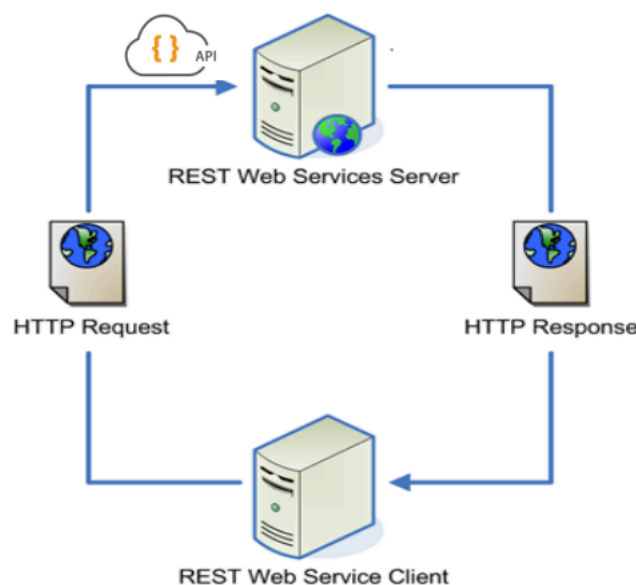
REST API – משתמש בעקרונות של REST על מנת לייצר API אוסף של קריאות ממערכת אחרת כדי לקבל ממנה שירותים, ממשק שנותן לי קטע של קוד או פונקציונליות לתפעל אותו.

5 כללים לארכיטקטורת REST:

1. לכל יחידה חייב להיות מזהה יחודי
2. שימוש בסט קטן של פונקציות
3. חייבים לחבר את כל היחידות ביחד
4. יכולת לעבוד עם סוגים שונים של מידע Content-Type
5. מקבלים בקשה ומחזירים תשובה ושוכחים (Statelessly)

HTTP

מימוש של REST (זהו פרוטוקול, כלומר סט של חוקים אשר מגדיר תקשורת בין יחידות עיבוד) **מחזור:**




בכללי יש פקודות POST, GET, DELETE, PUT, PATCH

מכיל Header ו-Body, אפשר לקרוא על זה עוד אבל זה בגדול חוזר מקורס **תקשורת** עם עמית דביר

פתרונות לדף המושגים הרלוונטיים לבחינה

מהו פרוטוקול WebSocket ומה ייעודו

פרוטוקול תקשורת מחשבים המספקת ערוצי תקשורת Full Duplex (מאפשר לתקשר אחד עם השני בשני כיוונים בו-זמנית, כמו פלאפון שאפשר גם לשמוע וגם לדבר) פרוטוקול זה מאפשר לבצע חיבור בין דפדפני Web לבין שרתי Web, בנוסף הוא מספק חיבור יציב (Persistent) במודל CS כך ששני הצדדים יכולים להתחיל לשלוח מידע בכל זמן, ה-Client מקים את חיבור ה-WebSocket דרך תהליך שנקרא  WebSocket Handshake, תהליך זה מתחיל בשליחת של בקשת HTTP מ-C ל-S.

מהו Docker ומרכיביו (Image, Container, Engine, Registry), מהו Docker File

דוקר הוא פרויקט קוד פתוח המאפשר אוטומציה של התקנת והרצת יישומים בתוך מכולות תוכנה כל הרעיון זה למנוע שיש מלא סביבות עבודה שונות, פעם היה צריך להכין טבלה ענקית של כל המכשירים בעולם × האפליקציה שלנו ולראות שיש ויי בכל תא, זה בעייתי. בעולם ההובלות פתרו את זה, הריי אם אני רוצה למכור גלשן אז אני לא צריך לחשוב איך חברת ההובלה תוביל אותו, זה קורה אוטומטית כי יש בעצם קונטיינר שעוטף את מה שאני רוצה למכור, אז אותו דבר כשאני מייצר אפליקציה, אני עוטף אותה בסביבת עבודה.

Image

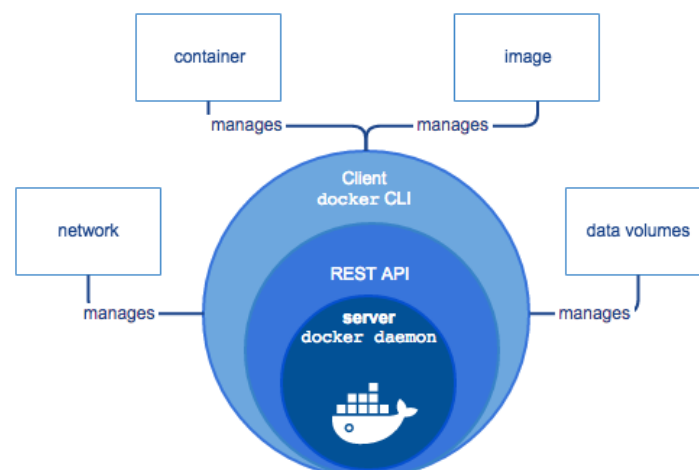
זוהי תבנית Read-only עם הוראות הפעלה ליצירת Docker Container, בד"כ תמונות מבוססות על תמונות אחרות, לדוגמא תמונה כלשהי מבוססת על מע"ה ubuntu

Container

גרסה רצה של Image, ניתן לעצור אותה, ליצור אותה, להזיז אותה או למחוק אותה ע"י Docker API/CLI. בכללי k8s מנהל קונטיינרים (k8s מחוץ לסקופ של הקורס)

Engine

מייצר, מאחסן ומריץ קונטיינר של דוקר, נתינת לפריסה ללא הקשר ב-Infrastructure, כלומר במקומי, פיזי, וירטואלי, מוד נתונים או ענק.



Registry

מאחסן Docker Images, לדוגמא Docker Hub הוא Docker Registry שכל אחד יכול להסתכל.

Docker File

סוג של "תבנית" שבתוכו אפשר להכניס הוראות שונות שיעבדו ממש כאילו אנחנו בתוך, משתמש עבור תמונה.

הסבר קצת יותר מקיף:

<https://github.com/ZviMints/Summaries/blob/master/%D7%AA%D7%A2%D7%A9%D7%99%D7%99%D7%94%D7%95%D7%94%D7%9E%D7%A1%D7%91%D7%99%D7%91/%D7%AA%D7%A7%D7%A6%D7%99%D7%A8%20Docker.pdf>

סוגי מסדי נתונים NoSQL, הסיבות להופעתן, דוגמאות לתשתיות

מדובר במסדי נתונים בעל נפתים גדולים, השומרים מידע בכמות עצומה במהירויות גבוהות מידע זה אינו מאורגן לפי שיטה כלשהי (Unstructured), הוא מגוון מאוד כך **שלא** ניתן לסדר אותו בטבלאות מסדי נתונים NoSQL הופיעו כחלופה פופולרית למסד נתונים טבלאי/רלציוני, כאשר יישומי האינטרנט הפכו למורכבים יותר ויותר, נתונים Horizontal Scalability (Scale Out), שימוש חכם במטמון וביצוע מקבילי. **הערה:** ב-SQL מקבילים ACID וב-NoSQL מקבילים Base.

סוגי מסדי נתונים הם:

1. Key-Value

דוגמא: Redis, Amazon DynamoDB הן מערכות פשוטות לניהול מסדי נתונים שמאחסנות זוגות של ערכי מפתח ומספקות פונקציות בסיסיים לאחזור הערך המשויך למפתח ידוע, זה אומר את מערכת הניהול ל-Embedded Database כאשר הנתונים המאוחסנים אינם מורכבים ולכן המהירות היא בעלת חשיבות עליונה, טוב ל-Caching

2. Wide-Column

3. Document

דוגמא: MongoDB, Couchbase אשר הן מערכות שלא בנויות על סכימה אלא מאחסנות נתונים בצורה של מסמכים, כאן המסמך הוא המפתח והתוכן של המסמך הוא הערך

4. Graph

הבדלים בין SQL לבין NoSQL:

- RDBMS איטי ויקר
- RDBMS עושה Scale-up ע"י קנייה של מכונות גדולות יותר
- NoSQL עושה Scale-out ע"י מערכת מבוזזת על כמה הוסטים
- RDBMS מאחסן Big-Data באופן לא טוב בגלל שאי אפשר לעשות Scale טוב, ולכן מערכות NoSQL כמו Hadoop תומכות ב-"Big-Data" בצורה הרבה יותר טובה
- NoSQL גמיש, מותאם להתעסקות עם BigData, יותר זול לתחזוק, מבנה יכול להשתנות עם הזמן.
- מסדי נתונים רלציוניים מתקשים לעבור ביזור ולממש מקבול לעיבודים.

מאפייני מסד נתונים מבוסס Key-Value, ומאפייני מסד נתונים מבוסס מסמכים

מהי Transaction ותכונותיה (ACID), מהו מדד TPS

Transaction

היא פעולה לוגית לשינוי נתונים המורכבת מסדרת פעולות בדידות. תנועה מכונה לעיתים גם יחידת עבודה לוגית, מכיוון שכל הפעולות הבדידות המרכיבות אותה חייבות להתבצע כיחידה אחת, או לא להתבצע כלל. **תכונה זו נקראת אטומיות.**

מטרה: כדי לשמור על עקביות הנתונים העסיקיים, לדוגמא בפעולת של העברת כספים הסכום הוחסר אבל הוא הועבר, תנועה שלא בוצעה בשלומתה מבוטלת באמצעות Rollback, כדי להחזיר את המערכת והנתונים למצב יציב

פתרונות לדף המושגים הרלוונטיים לבחינה

שהיה לפני, בסיום הפעולה היא הופכת להיות Committed Transaction, תנועה יכולה להתבטל באמצעות הסגה כל עוד לא קובעה, אם אני לא טועה יש תמיכה רק ממוגו 4 בטרנזקציות.

מדד Transaction per second

כמות הפעולות האומטיות המובצעות ע"י אישיות כלשהי, כמות הטרנזקציות המבצעות כל שנייה

טרנזקציה נכנסת למידע בעזרת פעולות קריאה וכתובה (Read&Write) במטרה לשמור על עקביות במסד הנתונים לפני ואחרי טרנזקציה צריכים להתקיים מאפיינים מסוימים מאפיינים אלו נקראים ACID (יחסים)

המונח **ACID** (קורה ב-SQL, רלציוני) הוא ר"ת של Durability, Isolation, Consistency, Atomicity תכונות אלה הן אבן הפינה של מסדי נתונים ובלעדיהן לא ניתן להבטיח את שלמות הנתונים במערכת אלה.

Atomicity

כל פעולה מתבצעת במלואה או לא מתבצעת בכלל
לפי יוסי: כל טרנזקציה מתבצעת בפעם אחת או שלא מתבצעת בכלל

Consistency

הבסיס נתונים תמיד נשאר עקבי, אם פעולה לא חוקית (השמה של מחרוזת במקום מספר, השמה של מספר שלא זהה לחוק שהוגדר עבור עמודה) מתבצעת, אז בסיס נתונים לא מאפשר ביצוע של אותה הפעולה.
לפי יוסי: מסד נתונים חייב להיות עקבי לפני ואחרי טרנזקציה

Isolation

בסיס הנתונים מאפשר לבצע פעולות רבות במקביל, כל עוד התוצאה זהה לביצוע הפעולות באופן טורי
לפי יוסי: מספר הטרנזקציות מתבצעות באופן ב"ת אחת בשנייה וללא הפרעות

Durability

פעולה שמתבצעת תמיד נשארת במסד נתונים, אפילו אם היה תקלה באמצע, אם זה הפסקת חשמל וכו'. פעולות לא הולכות לאיבוד.
לפי יוסי: השינויים של טרנזקציה מוצלות מתבצעים אפילו אם יש שגיאות מערכת

לעומת זאת ב-NoSQL נקבל Base

Acid & Base הינם מודלים לייצוג עקביות, Acid – מייצג תכונות הנדרשות מטרנזאקציה במסד נתונים רלציוני בכדי לסיים יחידת עדכון באופן תקין, Base – מייצג זאת ב-NoSQL Databases.

BASE

Basically Available

המידע בד"כ זמין

Soft State

המצב יכול להשתנות גם ללא עדכונים בגלל עדכונים ישנים

Eventual consistency

אם ניתן מספיק זמן, בסוף המידע יהיה עקבי.

העבודה משתנה שלא מחייבת אותנו לעבוד כמו **ACID** ובכך מצד אחד אנחנו לא תמיד הכי מעודכנים אבל זה יוצר אצלנו את האפשרויות של הרחבה וחוסן של המערכת.

פתרונות לדף המושגים הרלוונטיים לבחינה

מי שמשמש בכללי BASE מעדיף **זמינות** אבל לא מחייב עדכון באותו הזמן לכולם אלא בהמשך.

הגדרות Big Data, ההבדל בינו לבין Small Data

הגדרה: הוא מונח המתייחס למאגר מידע הכולל נתונים מבוזרים, שאינם מאורגנים לפי שיטה כלשהי, שמגיעים ממקורות רבים, בכמויות גדולות, בפורמטים מגוונים, ובאיכויות שונות.

- יש המון מידע שנשמר
 - יש לנו כוח חישובי
 - יש את הטכנולוגיה
- המטרה:** לדעת להסביר, לחזות

ההבדלים: בין SmallData לBigData :

1. **מטרה:** נאסף למטרה מוגדרת **אחת לעומת** אבולוצית מטרות עם הזמן
2. **מקום:** מקום אחד בפורמט **אחיד לעומת** מבוזר על גבי מספר פלטפורמות ופורמטים.
3. **מבניות:** יש מבנה, סכמה **לעומת** יש נתונים בלי סכמה.
4. **אורך חיים:** נאגר ורלוונטי למספר **שנים לעומת** מצטבר לאורך זמן לא ידוע.
5. **יחידות מידה:** אחיד ומוגדר **לעומת** רבגוני.
6. **שחזור:** ניתן ליצירה מחודשת **לעומת** לא ניתן ליצירה מחודשת.
7. **אינטרוספקציה:** נתונים המתארים עצמם **בבהירות (מיהו האובייקט נמדד, מה המימד ומהו הערך)** **לעומת** מגוון, חזרות, כפילויות וחוסרים הכנות קשות ואינטנסיביות
8. **יכולת ניתוח:** **בבת אחת לעומת** צורך להכין, לצמצם, לפצל, להמיר.

מודל ה-V-ים לתאור אתגרי ביג דאטא

נתוני העתק מוגדרים לפי שלושה עקרונות עיקריים:

1. נפח Volume

כמות גדולה מאוד של מידע שיש לנהל אותו, יתרה מזה, ככל שאנו אוספים יותר ויותר מידע – זה עוזר אף יותר לנהל אותו ולהפיק תובנות ממנו.

2. מהירות Velocity (נרצה זמן קרוב ל-Realtime)
קצב של המידע שזורם

3. גיוון Variety:

מגוון של סוגי דאטא. קודם כל מדובר כאן במידע שהוא – Unstructured מידע לא מובנה, שלא ניתן להכניס אותו לטבלה רגילה. זהו מידע ממקורות שונים, מסוגים שונים, בפורמטים שונים.



תרשים 1: נתוני עתק לפי מודל V⁶

פתרונות לדף המושגים הרלוונטיים לבחינה

איך מזהים שיש בעיות ב-BigData, ישנם 4-Vים: (מהסיכום של 2018)

1. **Volume** - נפח הנתונים.
2. **Velocity** - קצב קבלת הנתונים מהירה, מוצפים בנתונים. (מקסימום היכולת לקבל)
3. **Variety** - נתונים מגעים בכל מיני פורמטים, עם סכמה, בלי סכמה....
4. **Veracity** - איכות ואמינות הנתונים לא וודאית. נתונים חלקים, נתונים בעלי משמעות כפולה, נתונים שמגיעים בעיבוד, נתוני שקר שמופצים.

מהו Data Pipeline ושלביו

זרימת נתונים ממקור אחד לאחר היא אחת התהליכים הקריטיים לארגונים מונעים מידע. זרימת נתונים יכולה להיות רעועה כי ישנם כל כך הרבה דברים **שיכולים להשתבש** באמצע, וכלל שמורכבות הדרישות גדלה ומספר מקורות המידע מתרבה, בעיות אלה גדלות בהיקף ובהשפעה. **Data Pipeline** היא תוכנה שמבטלת הרבה תהליכים ידניים ומאפשרת זרימה חלקה של מידע מתחנה אחת לאחרת. (נבנה עבור יעילות) היא מתחילה בהגדרת מה, איפה ואיך נאספים הנתונים והיא מבצעת אוטומציה לתהליכים: **חילוף, טרנספורמציה שילוב, אימות וטעינה של המידע להמשך ניתוח וויזואליזציה**. היא מספקת מהירות end to end ע"י ביטול שגיאות וטיפול במקרי צוואר בקבוק. היא יכולה לעבד הרבה זרמי נתונים בבת אחת. בקיצור היא בהחלט נצרכת לחברות מבוססות מידע.

מהו Apache Hadoop, מרכיביו, ומטרותיו

Hadoop – היא פלטפורמה לעיבוד מידע בנפחים גדולים, Open source, יש לה 2 מודולים, אחד זה Map Reduce והשני זה HDFS

מרכיביו:

HDFS – Hadoop Distributed File System

זה מערכת קבצים שהיא חלק מהפלטפורמה, שומרת מידע ובעלת אמינות גבוהה המידע מחולק לבלוקים, ומפוזר ל Nodes-מרוברים. לכל בלוק של מידע יש יותר מעותק אחד, כדי שאם פתאום Node אחד נפל ולא זמין – יש עוד Node עם אותו המידע והוא זמין בכל מקרה. מערכת שיש לה שרידות גבוהה כך שאם יהיו בעיות המערכת את תוכל להמשיך. וקלה להתחברות, מבחינת המשתמש הוא רואה את כלל המערכת ללא הבנה או ראייה של כל החלוקות

Map/Reduce – (תהליך עיבוד מבוזר) עם כל ההתפתחויות שקרו בשנים אחרונות בתחום זה, Map/Reduce נחשב למנוע הכי נפוץ שיש, יציב ומוכח בתעשייה. יחד עם זאת, עם המשך התפתחות בתחום וכמויות דאטה שהולכות וגדלות, פותח בשנים אחרונות מנוע חדש – Spark מנוע ושיטת עיבוד לעבודה מקבילים, כאשר העיבוד יהיה מפוצל על כמה מחשבים. העבודה איתו לוקחת זמן (Batch Processing) ואינה Real-time/Online, ניתן להפעיל על כל מחשב.

- הוא אינו DB. מאפשר יותר פונצקיונאליות
- מאפשר עיבוד ואחסון על נתונים
- ישנו חלק שאפשר להוסיף שהוא יהיה DB לארגון HBase
- הוא שומר נתונים בצד אחד ובצד השני הוא מחשב אחרים

עובדות על Hadoop:

- מהפרויקטים המובילים של Apache
- אחת הסביבות הכי מבוקשות של עולם הביג דאטא
- framework
- יש מס' שיטות לעיבוד מקבילי
- סקילבלי, גמיש (ללא Scheme), לא עולה כסף (זה open source)

פתרונות לדף המושגים הרלוונטיים לבחינה

מהו Map-Reduce (האלגוריתם ואסטרטגיית עיבוד, לא רק מנוע העיבודים של Hadoop)

מודל תכנותי שמולווה בתשתית עובדת המיועד לאיבוד של DATASATE גדולים.

היתרון: פיצול משימה לתתי משימות כך עובדים בצורה מקבילית.

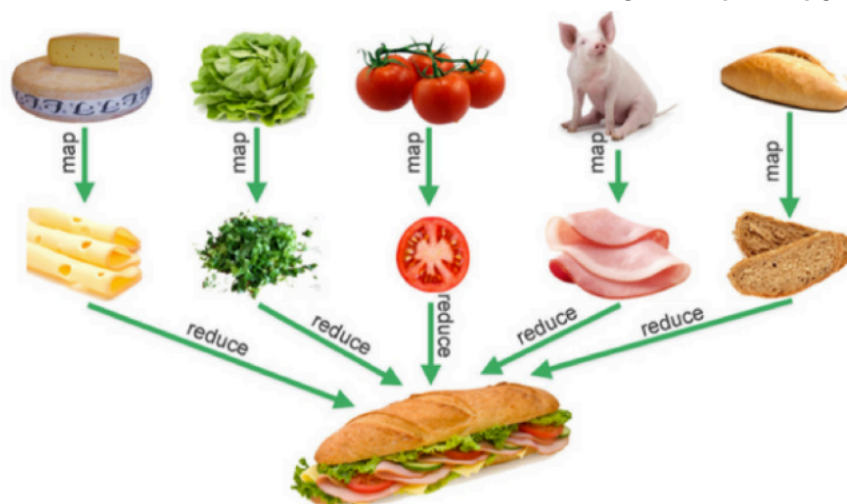
שלבבים:

שלב א: כל המחשבים מקבלים משימות ומחולקות לתפקידים

שלב ב: ערבוב, בכל מחשב בודקים את כל התוצאות אבל עכשיו אנחנו נרצה לחלק תפקידים הספציפיים עבור

המחשב אחד יהיה אחראי על A ומחשב אחר מנהל את B.

שלב ג': קיצוב הנתונים וסכמה מכל המחשבים



מהו Spark, מרכיביו, ההבדל העיקרי בינו לבין Apache Hadoop

Spark – זהו למעשה הדור הבא של עיבוד מידע, הרבה יותר יעיל מ-Map/Reduce, הוא יותר מהיר, יודע לנצל זכרון

בצורה מיטבית, הרבה יותר קל ונוח לפתח בו, ניתן לעשות בו עיבודים של Machine Learning.

מנוע עיבודים, המטרה היא עיבוד מאוד מהיר. מנועה שמיועד לכמות נתונים בינונית. כאשר אנחנו רוצים

להפעיל אלגוריתמים שחוזרים על עצמם מספר פעמים (איטרציות). (בניגוד ל Map שהוא מפצל מנתח ומחזיר

(ללא חזרה)

פיצ'רים:

פתרונות לדף המושגים הרלוונטיים לבחינה

- מאפשר בניה של מבנים לעיבוד ומאפשר לקחת נתונים ולנתח אותם במספר מעבדים.
- יש לו שאילתות.
- עיבוד לנתונים בזמן אמת.
- יודע להציג גרפים.
- יודע להשתמש בשפה R.
- מנועה שרץ מהזיכרון וגם מדיסק. (מאבד מהירות שהולך לדיסק).
- מאפשר לשמור תוצאות ביניים.
- RDD – המבנה של הנתונים הבסיסי, חזק מבזר. דומה ל MONGO.
- RDD – המבנה של הנתונים נשאר, לא משנים.
- RDD – שני פעולות הוא עושה. 1. טרנספורם, מחזיר RDD חדש. 2. מחזיר ערך חדש.

השילוב בין SPARK & MONGO

- לפעמים נרצה שה DB יהיה של MONGO מצד שני הפעולות שנרצה לבצע יהיו של ספרק כדי להגיע למקסום.
- MONGO מאפשר ניתוח נתונים בזמן אמת.
- ספרק יש 100 אפשרויות של אלגוריתם לעיבוד נתונים.
- ניתן לשלב עם יצירת אינדקס של MONGO ובכך לעזור ל SPARK.

השוואה בין HADOOP ל SPARK

- הבדל פי 100 במהירות.
- SPARK מתאים ויעיל כאשר הנתונים מתאימים לזיכרון כאשר HADOOP כאשר הזיכרון לא מתאים ומסודר אז היתרון שלו עולה.

יתרונות של Spark:

גמישות – האפשרות לייצר מאפס פתרון מותאם צורך.

תומך בשליפות SQL – Spark SQL

מספק **עיבוד בזמן אמת** (Spark Streaming – real-time streaming analytics)

יכול לרוץ על Hadoop Cluster או בענן, או על Cluster משל עצמו.

יכול לגשת למקורות מידע מאוד מגוונים: HDFS, Apache Cassandra, Apache HBase, Amazon S3 Cloud Storage

מהו מתווך מסרים, מאפייני Kafka

מתווה מסרים: דפוס שליחת הודעות כאשר השולח לא שולח הודעה ללקוח ספציפי.

במקום זאת, השולח מסווג את ההודעות לסוג מסויים והלקוח רושם איזה סוג של הודעות הוא מוכן לקבל.

Kafka:

היא פלטפורמת תוכנה לעיבוד זרם נתונים (stream processing) קוד פתוח המפותחת במסגרת קרן התוכנה אפאצ' ונכתבת

בשפות **Scala** ו-Java. הפרויקט נועד לספק פלטפורמה מאוחדת, עם תפוקה גבוהה וזמני השהיה קצרים לטיפול בזרם

נתונים **בזמן אמת**. שכבת האחסון היא בעצם **תור הודעות גדול ממדים בתבנית עיצוב יצרן-צרכן (Pub/Sub) עם**

ארכיטקטורה **מבוזרת ניתנת להגדלה (scalable)**

מאפייני Kafka:

1. סוג של מסווג הודעות, שומר את ההודעות המוזנות בקטגוריות שנקראות topics.
2. תהליך שמפרסם הודעות ל Kafka Topics נקרא **יצרן**.
3. תהליך שמעבד ומקבל הודעות שמתפרסמות בtopics נקרא **צרכן**.
4. Kafka מורכבת מ-cluster שמכיל בתוכו מספר שרתים, כל שרת נקרא **broker**.
- ה-broker אחראי על קבלת הודעות מיצרנים ועל טיפול בקשות צרכנים לקבלת הודעות.
5. תקשורת בין כל הרכבים מתבצעת באמצעות API ופרוטוקול TCP.

הגדרה מהי למידה

למידה אומר שרמת הביצועים של תכנית במשימה מסוימת יעלה ביחד עם כמות הניסיון כלומר, תכנית מחשב תיקרא "תכנית לומדת" אם הביצועים שלה במשימה כלשהי השתפרו ככל שהיא צברה יותר ניסיון.

ההבדל העיקרוני בין תכנות קלאסי ללמידת מכונה

להשלים

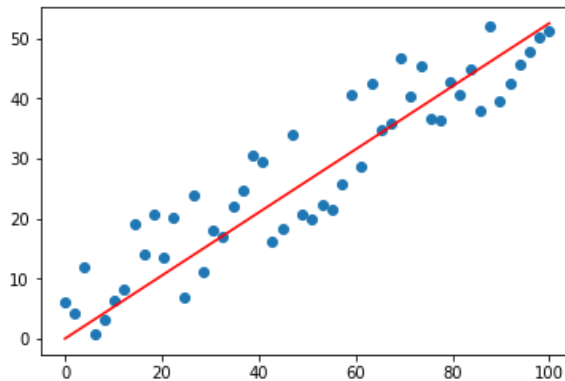
פתרונות לדף המושגים הרלוונטיים לבחינה

2 עקרונות לסוג למידה מפוקחת, מהי למידה מפוקחת (עץ החלטה, רגרסיה לינארית) למידה מפוקחת:

המכונה מקבל מקבץ דוגמאות שקרו ובנוסף מקבלת מהו הפלט הרצוי עבור הדוגמאות הללו, לדוגמא: המייל מכתובת מסוימת הינו מייל זבל. בעצם יש מפקח שמספר למכונה מה היא אמורה ללמוד מכל אחת מהדוגמאות. מהמכונה אנו מצפים לייצר סט חוקים שממפים כל מקרה לתוצאה.

רגרסיה לינארית:

- מחפשים יחס בין משתנה תלוי מספרי לבין משתנים (אחד או יותר) בלתי תלויים
- תכונת המטרה היא נומרית
- נועד לביצוע חיזוי לערכו של המשתנה התלוי על בסיס ערכי תכונות אחרות



עצי החלטה:

- צומת פנימית מייצגת פיצול על ערכי תכונה
- קשת מייצגת תוצאת פיצול
- עלים מייצגים תיג או התפלגות
- עצי החלטה מחלקים את מרחב התכונות למלבנים, ומעניקים לכל מלבן תיג מתוך האופציות הנתונות (אדום וכחול בדוגמא)
- שיטת יצירת העץ היא לרוב בגישת $Top \Rightarrow Down$

2 עקרונות לסוג למידה לא מפוקחת, מהי למידה לא מפוקחת (ניתוח אשכולות, חוקי אסוציאציה) למידה לא מפוקחת:

כאן אין מורה או מפקח שמדריך את המכונה, המכונה מקבל מקבץ של מקרים ואנו מצפים מהמכונה ללמוד לבד מה הקשר בין המקרים, לזהות דפוסי התנהגות וכדומה.

ניתוח אשכולות:

להשלים

חוקי אסוציאציה:

להשלים

הגדרה מהי אפלקציה מונוליתית

בנויה כיחידה אוטונומית אחת. דבר זה גורם לכך ששינויים באפליקציה איטיים יותר כיוון שזה משפיע על התוכנית כולה (צריך לשנות את כל התכנית). שינוי לחלק קטן בקוד עלול לדרוש בניה והקמה מחדש של גרסה חדשה לתוכנה.

ארכיטקטורת למבדה, ארכיטקטורת קאפקה וההבדל העיקרי ביניהם

ארכיטקטורה למבדה: תבנית לעיצוב מערכות ביג דאטה

תבנית עם 2 ערוצים של הנתונים:

פתרונות לדף המושגים הרלוונטיים לבחינה

- **ערוץ של עיבוד מהיר** - אשר בו הנתונים עוברים עיבוד מהיר וישר מועברים לתצוגה אצל המשתמש.
- **ערוץ של עיבוד איטי** - הנתונים עוברים עיבוד יותר מעמיק ולכן יותר איטי לדוגמא יכול לעבור machine learning ורק לבסוף עובר לתצוגה אצל המשתמש.

ארכיטקטורה קאפה: תבנית לעיצוב מערכות ביג דאטה אשר בה יש רק ערוץ מהיר אשר בו הנתונים עוברים עיבוד מהיר וישר מועברים לתצוגה אצל המשתמש.

Scalability: היכולת של תהליך, רשת, תוכנה או ארגון להתמודד עם כמות הולכת וגדלה של עבודה ע"י הוספת משאבי מערכת