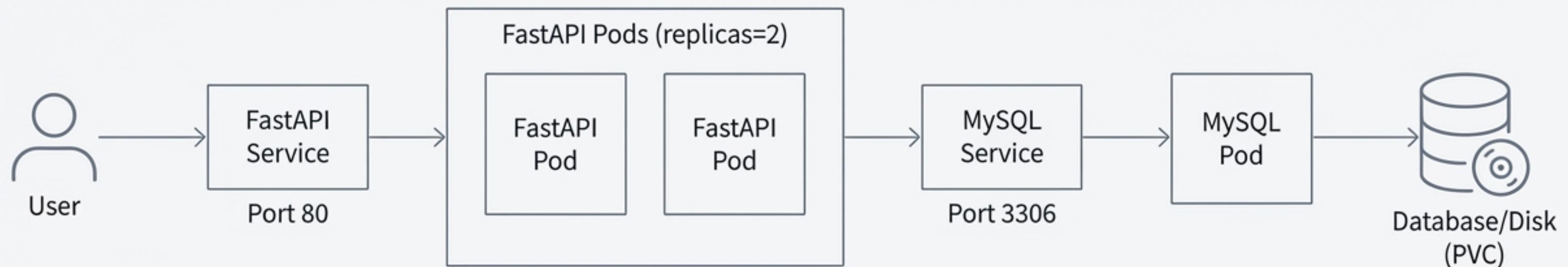


בונים מערכת מלאה על Kubernetes

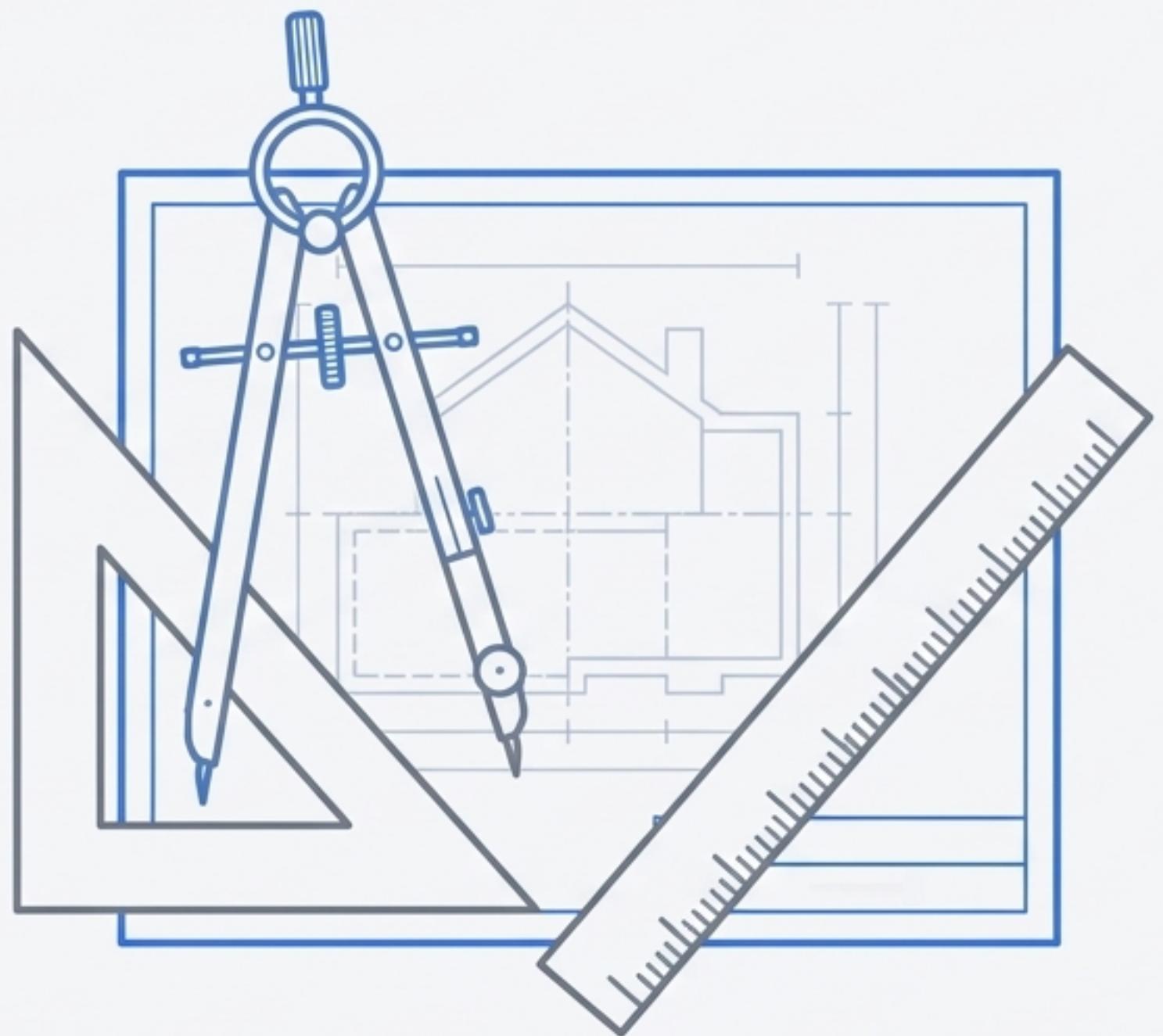
מדריך צעד-אחר-צעד: מ-FastAPI ו-MySQL למ-לאסטן חי

זו המערכת שנבנה ייחד, צעד אחר צעד.



אנחנו בונים מערכת מלאה: מסד נתונים (MySQL) ואפליקציה (FastAPI) שמתקשרים אחד עם השני. דיאגרמה זו היא המפה שלנו למסע.

חלק 1: היסודות - הגדרות וסודות



לפני שבונים את הקירות, חיבים
להגדר את הבסיס.

בשלב זה, ניצור את שני קבצי
התשתית: אמידע רגיש (סימאות)
ואחד להגדרות כלליות (שם
משתמש, כתובת).

כיספת המידע: ניהול `Secret` סיסמאות עם

קובץ זה מחזיק מידע רגיש.
(Base64, Kubernetes מקודד (Base64, Kubernetes
ולא נשמר כטקסט פשוט, מה שמוסיף
שכבת אבטחה.

```
secret.yaml
apiVersion: v1
kind: Secret
  name: mysql-secret
  type: Opaque
  stringData:
    MYSQL_PASSWORD: password123
    MYSQL_ROOT_PASSWORD: rootpassword123
```

סוג האובייקט: 'SID'.

השם שבו השתמש כדי
למשוך את הסיסמאות.

דרך נוחה לכתוב סידות.
Kubernetes מקודד
אותם ל-Base64
ברקע.

הרטוטים: הגדרות כלירות עם `ConfigMap`

כואנ שומרים מידע לא רגיש שהמערכת צריכה כדי לפעול, כמו שמות משתמשים, שמות מסדי נתונים, וכתובות רשת פנימיות.

configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
data:
  MYSQL_DATABASE: testdb
  MYSQL_USER: user
  MYSQL_HOST: mysql-service
```

סוג האובייקט: מפת הגדרות.

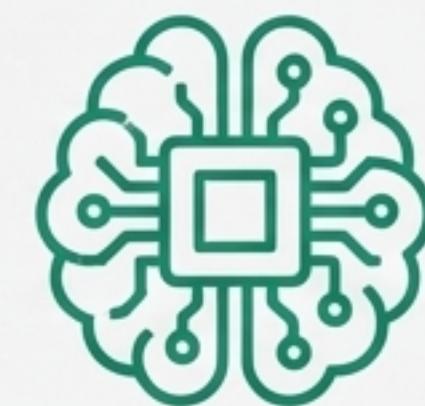
חשיבות: זו הכתובת הפנימית שבה האפליקציה תשמש כדי למצוא את מסד הנתונים.

חלק 2: חדר המכוונות - בניית מסד הנתונים

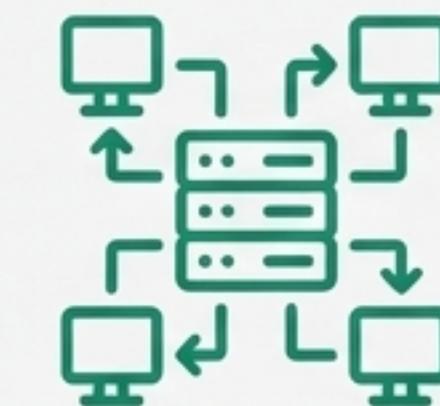
עכשו כישר לנו יסודות, נבנה את ליבת המערכת - מסד הנתונים. התהילה מורכב משלושה חלקים הכרחיים:



אחסון (PVC): דיסק
קשיח וירטואלי שישמר
את המידע שלנו.



ניהול (Deployment):
המוח שודאג שהקונטינר
של MySQL ירוז תמיד.



רשת (Service):
כתובת רשת קבועה כדי
שאפשר יהיה לדבר איתו.

זיכרון קבוע: הבטחת שרידות המידע עם PVC

קונטינרים (Pods) ב-Kubernetes הם זמינים. אם הם קורסים, כל המידע שבתוכם נמחק. כדי שמסד הנתונים שלן ישמר מידע למשך זמן רב, אנחנו צריכים לחבר אליו 'דיסק חיצוני' באמצעות PersistentVolumeClaim.

אנו מודדים את הדרישות של MySQL בclaim (Claim) שטח אחסון מהקלאסטר.

מצב גישה: רק פוד אחד יכול לכתוב לדיסק בו-זמן (אידיאלי ל-MYSQL).

הגודל המבוקש: 1 גיגה-בייט.

pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

המוח של מסד הנתונים: ה`Deployment` של MySQL

זהו האובייקט שמנהל את הקונטינר של MySQL. הוא אחראי לוודא שתמיד ירוץ עותק תקין שלו, וידוע לחבר אליו את כל ההגדרות והאחסון שהכנו מראש.

mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
spec:
  replicas: 1
  template:
    spec:
      containers:
        - name: mysql
          image: mysql:8.0
          envFrom:
            - configMapRef:
                name: mysql-config
            - secretRef:
                name: mysql-secret
      volumeMounts:
        - mountPath: /var/lib/mysql
          name: mysql-data
      volumes:
        - name: mysql-data
          persistentVolumeClaim:
            claimName: mysql-pvc
```

ודא שתמיד ירוץ עותק אחד בדיק.

הקסם קורה כאן:
שואבים את כל משתני הסביבה מה-`ConfigMap` ו-`Secret` שיצרנו.

מחברים את הדיסק (mysql-pvc) לתיקיה mysql-data /var/lib/mysql/ בתוך הקונטינר, שם MySQL שומר את המידע.

`Service` מtan גישה ל-MySQL עם

ה-IP של Pod יוכל להשתנות. כדי שהאפליקציה קבוצה שלנו תוכל למצוא את מסד הנתונים באופן אמין, אנחנו יוצרים `Service` שמעניק לו כתובות DNS פנימית ויציבה.

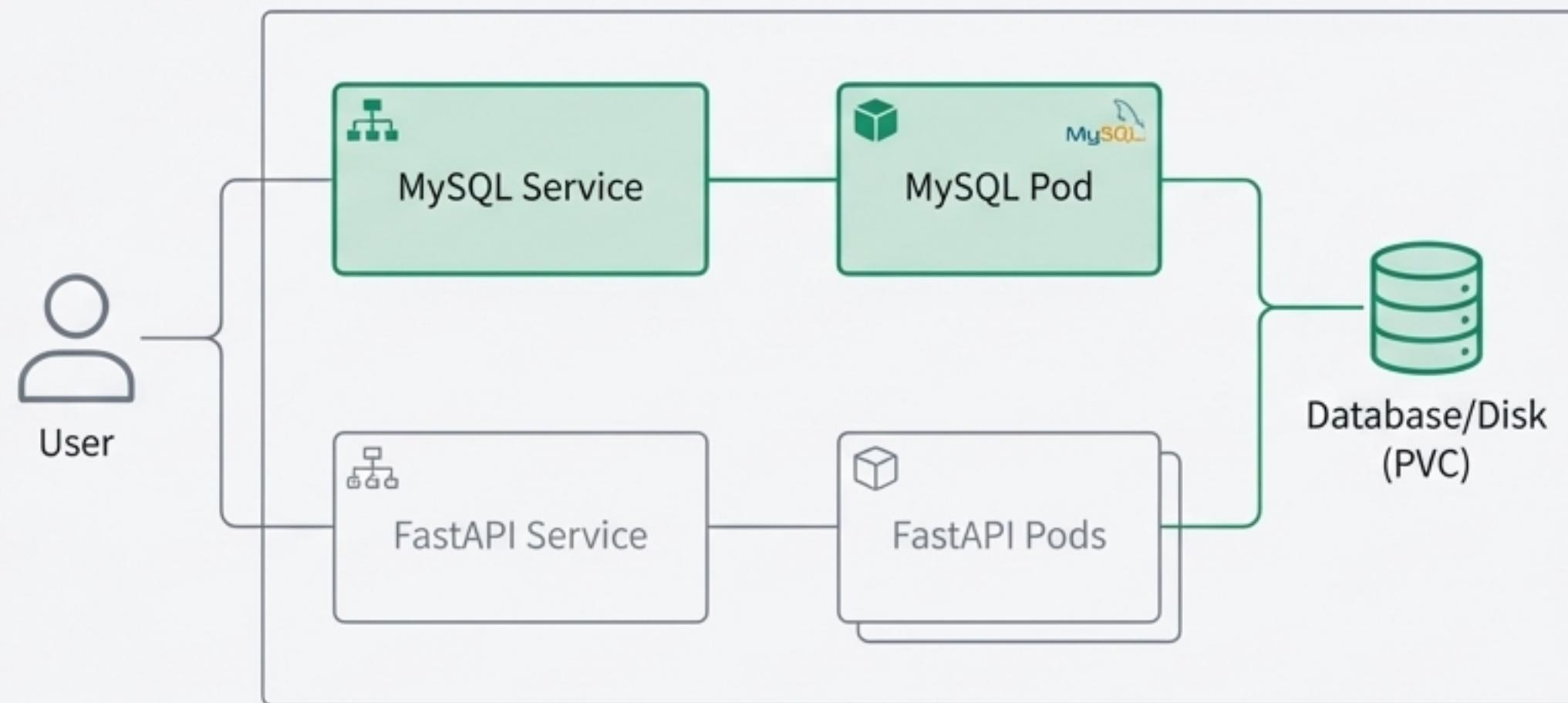
mysql-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    app: mysql
  ports:
  - port: 3306
    targetPort: 3306
```

זו הכתובת שהאפליקציה משתמש בה (זכרים את `MYSQL_HOST` מה-ConfigMap).

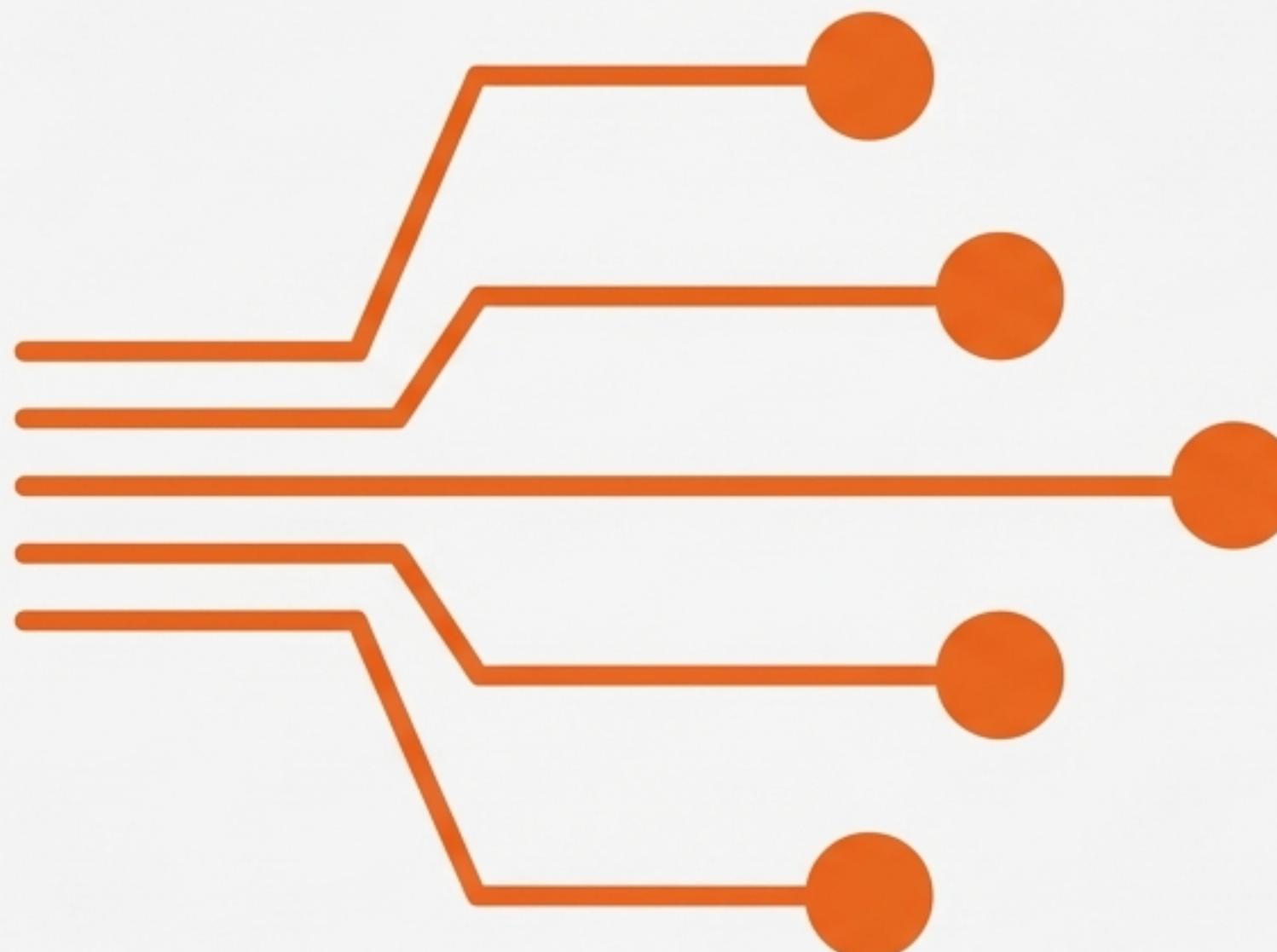
ה-Service יודע להעביר תובורה לכל Pod עם התווית `app: mysql` (שהוגדרה ב-Deployment).

נקודות ביניים: בניית שכבה מסד הנתונים



יצרנו אחסון קבוע, הגדרנו את מסד הנתונים, ונתנו לו כתובת רשות יציבה. חדר המכונות שלנו מוכן. השלב הבא: בניית האפליקציה שתשתמש בו.

חלק 3: הממשק לעולם - בניית שכבות האפליקציה



עם ממד נתונים יציב מאחורי הקלעים,
אנחנו יכולים להרים את שרת האפליקציה
של API.

התהlixir דומה: נגידר `Deployment` כדי לנהל
את האפליקציה, ו-`Service` כדי לחשוף
אותה לעולם.

האפליקציה בפועל: ה-Deployment של API

בדוגמה ל-MYSQL, גם האפליקציה מנוהלת על ידי `Deployment`. שמו לב להבדל המרכז: CAN CAN מרכיבים שני עותקים במקביל.

העוצמה של Kubernetes: CAN CAN מבקשת 2 עותקים. אם אחד נופל, השני נמצא נופל, השני ממשיר לעבוד והשירות לא נפגע.

בדיוק כמו ב-MYSQL, CAN CAN 'מצוריקים' את אותן הגדרות וסודות כדי שהאפליקציה תדע איך להתחבר למסד הנתונים.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fastapi
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: api
          image: fastapi-mysql:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8000
          envFrom:
            - configMapRef:
                name: mysql-config
            - secretRef:
                name: mysql-secret
```

השער לעולם החיצון: ה-'Service' של API-Fast

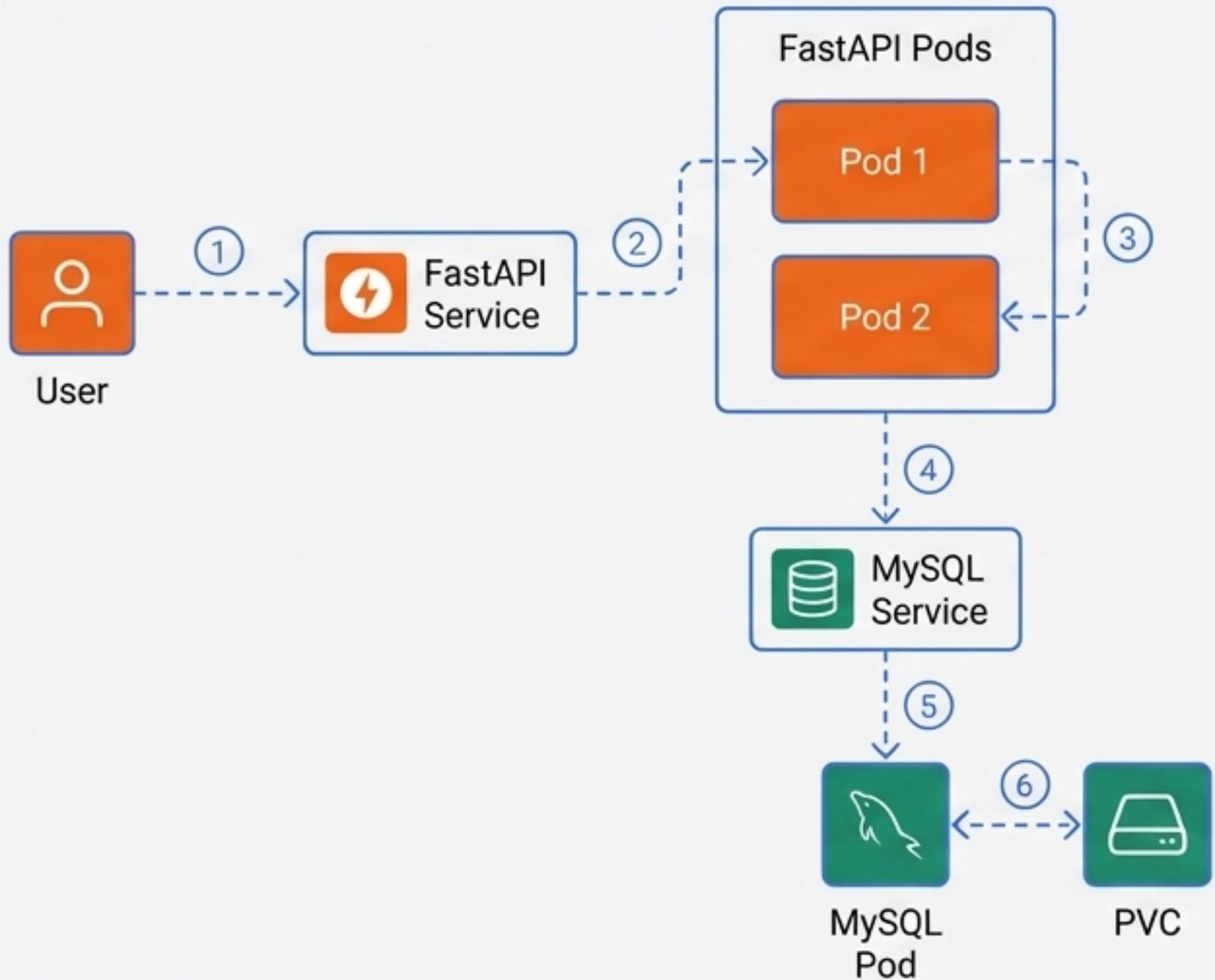
כדי שימושים יכולים לגשת לאפליקציה שלנו, אנחנו צריכים 'Service' שיקבל תעבורת מבחן וינתב אותה לפודים של ה-API.

ה-'Service' מאזין בפורט 80, פורט ה-HTTP הסטנדרטי.

הוא מעביר את הבקשות לפורט 8000 בתוך הוא מעביר את הבקשות לפורט 8000 הפודים, שם האפליקציה שלנו רצתה.

```
apiVersion: v1
kind: Service
metadata:
  name: fastapi-service
spec:
  port: 80
  selector:
    app: fastapi
  ports:
    - port: 80
      targetPort: 8000
```

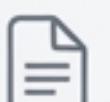
המערכת בפועל: קר הכל מתחבר



1. **משתמש פונה ל-'fastapi-service' בפורט 80.**
2. **FastAPI Service** מעביר את הבקשה לאחד מהפודים של API (פורט 8000).
3. **קוד ה-API.** קורא את משתני הסביבה כדי למצוא את כתובת מד הנתונים.(mysql-service)
4. הקוד פונה ל-'MySQL Service' בפורט 3306.
5. **MySQL Service** מעביר את הבקשה לפוד של MySQL.
6. **MySQL Pod** קורא או כותב מידע ל-**PVC** (הDisk הקשיח), והמידע נשמר.

התמונה המלאה: כל רכיבי המערכת שלר

בנינו יחד מערכת שלמה, רכיב אחר רכיב. להלן כל הקבצים שייצרנו:

-  `secret.yaml`
-  `configmap.yaml`
-  `pvc.yaml`
-  `mysql-deployment.yaml`
-  `mysql-service.yaml`
-  `fastapi-deployment.yaml`
-  `fastapi-service.yaml`

הצעד הבא שלר: להריץ את כל הקבצים האלה בסדר הנכון כדי להקים את המערכת.