מדריך מפורט לפריסת הפרויקט שלך ב-KIND

הבנת התמונה הגדולה

נראה שהפרויקט שלך מחולק לשלושה חלקים עיקריים:

Backend: החלק האחורי של האפליקציה, ככל הנראה כתוב ב-Python ומטפל בלוגיקת העסקים.

Database: מסד הנתונים MongoDB, שמשמש לאחסון נתונים.

Frontend: החלק הקדמי של האפליקציה, כנראה כתוב ב-Python ומטפל בממשק המשתמש.

כל חלק רץ בתוך קונטיינר דוקר משלו, ויש לך תמונות Docker מוכנות לכל חלק.

KIND הוא כלי שמאפשר לך להריץ קלאסטר קוברנטיס מקומי, קטן וקליל, בתוך קונטיינר דוקר. זה מצוין לסביבות פיתוח ובדיקות.

מה זה אומר לפרוס ב-KIND?

כשאתה פורס את הפרויקט שלך ב-KIND, אתה למעשה מעביר את האפליקציות שלך (הקונטיינרים) לסביבת קוברנטיס. קוברנטיס ידאג להריץ את הקונטיינרים האלה, לוודא שהם זמינים, להקצות להם משאבים ולנהל את התקשורת ביניהם.

שלבים לפריסה:

התקנת KIND:

התקנת Docker: ודאי ש-Docker מותקן ומריץ במחשב שלך.

התקנת KIND: הורד והתקן את KIND בהתאם להוראות הרשמיות.

יצירת קלאסטר: הפעל את הפקודה הבאה כדי ליצור קלאסטר חדש:

Bash

kind create cluster

יצירת מניפסטים של קוברנטיס:

Deployment עבור כל שירות:

YAML

apiVersion: apps/v1

kind: Deployment

metadata:

name: backend

spec:

replicas: 1 # שינוי מספר הרפליקטים לפי הצורך

selector:

matchLabels:

app: backend

template:

metadata:

labels:

app: backend

spec:

containers:

- name: backend

image: your-backend-image:1.0.1

ports:

- containerPort: 5000 # החלף לפורט הנכון

חזור על הפקודה הזו עבור frontend ו-MongoDB, והחלף את הפרמטרים בהתאם.

Service עבור כל שירות:

YAML

apiVersion: v1

kind: Service

metadata:

name: backend

spec:

selector:

app: backend

ports:

- protocol: TCP

port: 5000

targetPort: 5000

חזור על הפקודה הזו עבור frontend ו-MongoDB.

Persistent Volume Claim (PVC) ל-MongoDB: אם MongoDB דורש אחסון מתמשך, תצטרך ליצור PVC.

יישום המניפסטים:

Bash

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

# וכן הלאה עבור כל מניפסט

מה קרה עכשיו?

Deployment: הגדרנו את מספר העותקים של כל שירות, את התמונה שתשימש ואת הפורטים.

Service: יצרנו שירותים וירטואליים שמאפשרים גישה לפודים של השירותים.

PVC: (אם רלוונטי) הקצינו נפח אחסון מתמשך ל-MongoDB.

גישה לאפליקציה

מציאת כתובת ה-IP:

Bash

kubectl get services

גישה לאפליקציה: פתח דפדפן והקלד את כתובת ה-IP ואת הפורט של השירות הקדמי.

נקודות נוספות

NetworkPolicy: ניתן להגדיר חומות אש לרשת כדי לשלוט בתקשורת בין השירותים.

Ingress: אם תרצי להגיע לאפליקציה שלך מחוץ לקלאסטר, תצטרכי להגדיר Ingress Controller.

ConfigMaps: ניתן להשתמש ב-ConfigMaps כדי להזריק קונפיגורציות לאפליקציות.

Secrets: ניתן להשתמש ב-Secrets כדי לאחסן מידע רגיש כמו סיסמאות.

טיפים נוספים:

Helm: כלי נוח לניהול צברים של קוברנטיס, יכול להקל על תהליך הפריסה.

Kustomize: כלי ליצירת קונפיגורציות קוברנטיס מותאמות אישית.

Istio: שכבה נוספת של ניהול תנועה ברשת בתוך קוברנטיס.

הערה: זהו רק מבוא בסיסי. קוברנטיס הוא נושא רחב ומורכב, ויש עוד הרבה מה ללמוד.

אני רוצה בעצם שמבחוץ רק הFRONT יהיה חשוף, והFRONT והBACK יוכלו לתקשר, והBACK והDB יוכלו לתקשר. שהFRONT לא יהיה חשוף לDB וכן להיפך.

הגדרת תקשורת בין שירותים ב-Kubernetes באמצעות NetworkPolicy

הבנתי שאת רוצה להגדיר תקשורת ספציפית בין השירותים שלך: Frontend, Backend ו-MongoDB. הרעיון הוא לאפשר תקשורת ישירה רק בין Frontend ל-Backend, ובין Backend ל-MongoDB, תוך חסימת תקשורת ישירה בין Frontend ל-MongoDB.

NetworkPolicy ב-Kubernetes הוא הכלי המושלם למשימה הזו. הוא מאפשר לך להגדיר כללים מדויקים לגבי אילו פודים יכולים לתקשר עם אילו פודים אחרים.

שלבים לביצוע:

תיוג פודים:

ודא שכל פוד שלך מתויג בצורה ברורה. לדוגמה:

app: frontend

app: backend

app: mongodb

תוכל להשתמש בתגים נוספים לפי הצורך.

יצירת NetworkPolicy עבור Frontend:

YAML

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-frontend-to-backend

spec:

podSelector:

matchLabels:

app: frontend

policyTypes:

- Egress

egress:

- to:

- podSelector:

matchLabels:

app: backend

NetworkPolicy זו מאפשרת לפודים מסוג Frontend לשלוח תנועה רק לפודים מסוג Backend.

יצירת NetworkPolicy עבור Backend:

YAML

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-backend-to-frontend-and-mongodb

spec:

podSelector:

matchLabels:

app: backend

policyTypes:

- Egress

egress:

- to:

- podSelector:

matchLabels:

app: frontend

- podSelector:

matchLabels:

app: mongodb

NetworkPolicy זו מאפשרת לפודים מסוג Backend לשלוח תנועה לפודים מסוג Frontend ו-MongoDB.

יישום NetworkPolicies:

Bash

kubectl apply -f networkpolicy-frontend.yaml

kubectl apply -f networkpolicy-backend.yaml

הסבר:

podSelector: מגדיר לאילו פודים ה-NetworkPolicy חל.

policyTypes: מגדיר את סוג התנועה שברצוננו לשלוט בה (Ingress, Egress או שניהם).

egress: מגדיר את יעדי התנועה היוצאת מהפוד.

הסבר נוסף:

חסימת תנועה: אם לא תוגדר כלל Egress לפוד מסוים, כל התנועה היוצאת ממנו תהיה חסומה.

תנועה נכנסת: אם ברצונך לשלוט בתנועה הנכנסת לפוד, תוכל להגדיר את ה-policyType ל-Ingress.

תגים נוספים: ניתן להשתמש בתגים נוספים כדי להגדיר כללים מורכבים יותר. לדוגמה, תוכל להפריד בין סביבות פיתוח, בדיקה וייצור.

חשוב לזכור:

התחלה נקייה: לפני שאתה מגדיר את ה-NetworkPolicies, ודא שאין לך הגדרות רשת אחרות שעלולות להתנגש.

בדיקה: לאחר שתיישם את ה-NetworkPolicies, בדוק שהתקשורת בין השירותים פועלת כמצופה.

גמישות: NetworkPolicies מספקות גמישות רבה בהגדרת תקשורת בין פודים. ניתן ליצור כללים מורכבים יותר כדי להתאים לצרכים הספציפיים שלך.

לסיכום:

על ידי שימוש ב-NetworkPolicies, הצלחנו להגדיר תקשורת מבוקרת בין השירותים השונים שלך, תוך שמירה על האבטחה והבידוד הנדרשים.

מה ההבדל בין הפריסה של האפליקציה הזו בKUBERENTES לבין פריסה בKIND <?

ההבדל בין פריסה ב-Kubernetes לבין פריסה ב-KIND

Kubernetes ו-KIND הם שניהם פלטפורמות לניהול קונטיינרים, אך יש ביניהם הבדלים משמעותיים, בעיקר בהיקף ובמטרה.

Kubernetes

פלטפורמה מקיפה: Kubernetes היא פלטפורמה מובילה לניהול קלאסטרים של קונטיינרים בקנה מידה גדול. היא מספקת מגוון רחב של תכונות, כגון:

תזמור: הקצאת משאבים, ניהול מחזורי חיים של קונטיינרים והתאמתם לעומסים משתנים.

גילוי שירותים: גילוי אוטומטי של שירותים והתקשרות ביניהם.

ניהול נפח: ניהול נפחים מתמידים לאחסון נתונים.

רשת: הגדרת רשתות וירטואליות בין קונטיינרים.

אבטחה: מנגנוני אבטחה מגוונים להגנה על הקלאסטר.

סביבות ייצור: Kubernetes משמשת בדרך כלל לסביבות ייצור גדולות ומורכבות, שם היא מספקת זמינות גבוהה, יכולת התאמה וניהול מתקדם.

ספקים שונים: Kubernetes מוצעת על ידי ספקי ענן שונים (AWS, GCP, Azure) ועל ידי פתרונות מקומיים.

KIND

סביבת פיתוח מקומית: KIND היא יישום מקומי של Kubernetes, המיועד בעיקר לסביבות פיתוח ובדיקות.

פשטות: KIND קל להתקנה והפעלה, ומאפשר לך להריץ קלאסטר Kubernetes מקומי במהירות ובקלות.

גמישות: KIND מאפשר לך להתאים אישית את הקלאסטר לפי הצרכים שלך, אך ההתאמה היא בדרך כלל פחות מורכבת מאשר ב-Kubernetes המלא.

מגבלות: KIND מתאים בעיקר לפרויקטים קטנים ובינוניים. לקלאסטרים גדולים ומורכבים יותר, מומלץ להשתמש ב-Kubernetes המלא.

סיכום ההבדלים העיקריים

תכונה Kubernetes KIND

היקף פלטפורמה מקיפה לניהול קלאסטרים גדולים סביבת פיתוח מקומית

מורכבות מורכבת יותר להתקנה והגדרה פשוטה להתקנה והגדרה

שימוש סביבות ייצור, פיתוח ובדיקות בעיקר פיתוח ובדיקות

גמישות גמישות גבוהה גמישות מוגבלת יותר

מתי להשתמש במה?

Kubernetes:

פרויקטים גדולים ומורכבים.

סביבות ייצור הדורשות זמינות גבוהה.

צורך בתכונות מתקדמות של Kubernetes.

KIND:

פיתוח מקומי ובדיקות.

לימוד Kubernetes.

פרויקטים קטנים ובינוניים.

בניית צינורות CI/CD.

לסיכום:

KIND הוא כלי מצוין ללמוד ולהתנסות עם Kubernetes, אך עבור סביבות ייצור גדולות ומורכבות, Kubernetes המלא הוא הבחירה הטובה יותר

ההבדל העיקרי שתרגישי בפרויקט שלך כשאת פורסת אותו ב-KIND לעומת סביבת Kubernetes מלאה, הוא בעיקר בהיקף והמורכבות של הסביבה.

KIND הוא כמו חדר משחקים קטן ונוח לקונפיגורציה. הוא מצוין ל:

פיתוח מקומי: לבדוק את האפליקציה שלך בסביבה דומה לזו שתפעל בה בסופו של דבר.

ניסויים: לנסות דברים חדשים, להריץ גרסאות שונות ולראות איך הן מתנהגות.

לימוד: להבין את מושגי היסוד של Kubernetes בלי להתעמק בכל המורכבות.

Kubernetes המלא הוא כמו מרכז נתונים ענק. הוא מספק:

סביבת ייצור: להריץ את האפליקציה שלך בסביבה יציבה ואמינה.

תכונות מתקדמות: כמו ניהול תנועה מורכב, אבטחה ברמה גבוהה, והתאמה אישית עמוקה.

קנה מידה: להרחיב את האפליקציה שלך בקלות כדי לעמוד בעומסים משתנים.

אז איפה את תפגשי את ההבדל בפרויקט שלך?

גודל הקלאסטר: ב-KIND את תריצי בדרך כלל קלאסטר קטן עם מספר מצומצם של צמתים. ב-Kubernetes מלא, את יכולה להריץ קלאסטרים גדולים עם עשרות או מאות צמתים.

תכונות: ב-KIND לא יהיו לך כל התכונות של Kubernetes המלא. למשל, יכולות ניהול מתקדמות או תמיכה בספקי ענן שונים.

מורכבות: קונפיגורציה של Kubernetes מלאה יכולה להיות מורכבת יותר, עם הרבה אפשרויות להתאמה אישית. ב-KIND, הקונפיגורציה בדרך כלל פשוטה יותר.

לסיכום:

אם את רק מתחילה עם Kubernetes או שאת רוצה לבדוק רעיון חדש, KIND הוא כלי מצוין. הוא מאפשר לך להתחיל מהר ולראות תוצאות.

ככל שהפרויקט שלך יגדל והצרכים שלך יתפתחו, את תגלי שאת זקוקה לתכונות המתקדמות יותר של Kubernetes המלא.

דוגמאות ספציפיות:

ניהול נפחים: ב-KIND את יכולה להשתמש בנפחים מתמידים, אבל ב-Kubernetes המלא יש לך יותר אפשרויות לניהול נפחים, כמו למשל שימוש ב-Storage Classes.

רשת: ב-KIND הרשת בדרך כלל פשוטה יותר. ב-Kubernetes המלא יש לך יותר אפשרויות להגדיר רשתות מורכבות, כמו למשל שימוש ב-NetworkPolicies.

אבטחה: ב-Kubernetes המלא יש לך יותר כלים לאבטחת הקלאסטר, כמו למשל NetworkPolicies, Role-Based Access Control (RBAC) והצפנת תעבורה.