Kubernetes S01E01: Introduction และ Concept

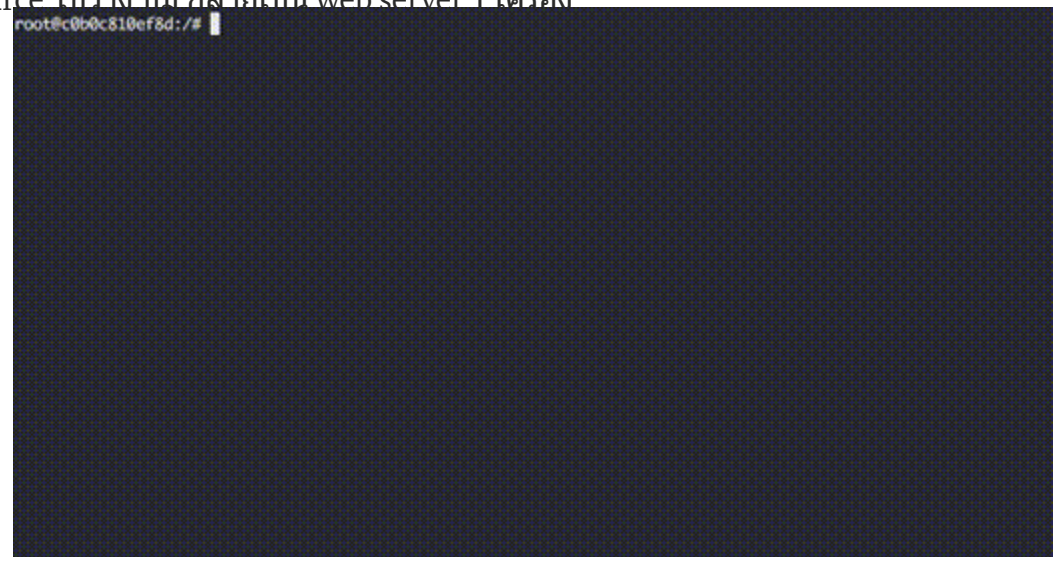
Nut P. Follow
Aug 6, 2017 · 4 min read

Source: kubernetes.io

ช่วงนี้หลายๆ คนอาจจะได้ยินชื่อ Kubernetes บ่อยๆ ว่าเป็น tools ที่ช่วยให้ deploy software ได้ง่าย สเกลได้ดี บลาๆๆ และปัจจุบันที่ Jitta.com เองก็ใช้ kubernetes เป็นเครื่องมือหลักมาเกือบสองปีกว่าแล้วในการ deploy app (kubernetes ออกมาประมาณ 3 ปีที่แล้ว)
เนื่องในโอกาสที่ kubernetes ได้เข้า Cloud Native Computing Foundation (CNCF) พอดี การได้เข้า CNCF เนี่ยแสดงให้เห็นว่า kubernetes เป็นปัจจุบันและอนาคตของ cloud แน่นอน เลยมาเขียนเล่าซะหน่อยครับ
ตอนที่ผมศึกษาเกี่ยวกับเจ้าตัว kubernetes ครั้งแรกๆ เนี่ย ก็รู้สึกว่ามันเข้าใจยากเหลือเกิน ใช้เวลานานกว่าจะเห็นภาพว่าอะไรคืออะไร
blog นี้เลยลองพยายามอธิบาย kubernetes เป็นภาษาที่ง่ายขึ้น เข้าใจง่ายขึ้นเท่าที่จะทำได้นะ
ก่อนจะเข้าเรื่อง kubernetes ลองมาดูก่อนว่าทำไมถึงควรลองใช้

ทำการ Deploy แบบต่างๆ

ยกตัวอย่างด้วยการจะติดตั้ง web app ซักตัวละกัน
แบบทำเองหมดเลย ก็เช่า cloud มาซักเครื่อง ssh เข้าไป ลง web server (เช่น nginx/apache) ลงนู่นนั่นนี่ ก๊อป source ไปวาง มัน กลายเป็น web server 1 เครื่อง



ลงเองก็ต้อง apt-get รัวๆ

ซึ่งก็ดีถ้าแอพเราขนาดเล็ก แต่ข้อเสียก็มีเยอะเลยถ้าต้องทำกับแอพที่ใหญ่ขึ้น เช่น

- ถ้า version ของต่างๆ ในเครื่องเปลี่ยนละ แอพจะบึ้มมั้ย
- สเกลเครื่องเพิ่ม/ย้ายเครื่องก็ต้องมาลงแบบนี้อีกรอบ
- เครื่องดับทำไง
- อยากได้ environment เหมือนเดิมเป๊ะ

เลยเป็นที่มาของยุคต่อไปคือ **Containerized Application**

พอใช้ container อย่าง docker เข้ามาช่วย ก็ช่วยแก้ปัญหาข้างบนได้ระดับนึง ทำให้เราสามารถกำหนด env ของแอพเราได้เป๊ะๆ กำหนด version ได้เป๊ะๆ ทำไม่ต้องกังวลเรื่องนี้อีก

```
FROM node:8.1.2

# dependency
WORKDIR /app
ADD package.json /app/package.json
ADD yarn.lock /app/yarn.lock
RUN yarn --prod

ADD . /app
```

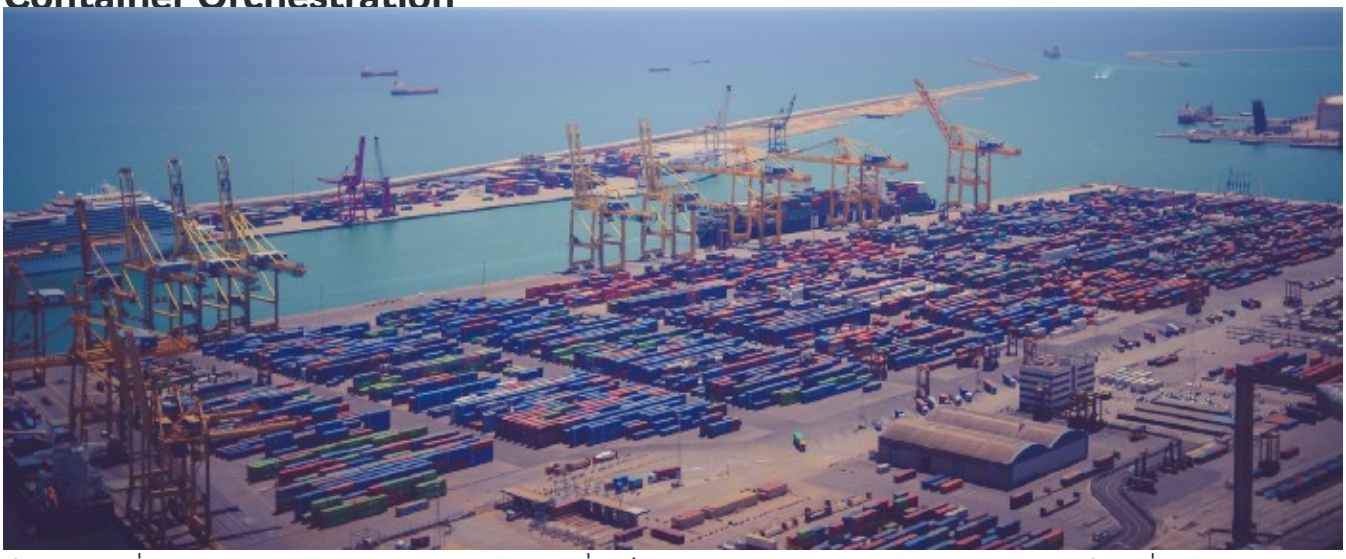
ตัวอย่าง Dockerfile

นอกจากนี้ตัว docker เองยังมีลักษณะเป็น **Infrastructure-as-Code** คือแทนที่เราจะต้องมานั่งลงนั่นนี่เอง เราเขียนนิยามแทนว่าแอพเราเวลา deploy แล้วต้องเป็นยังไง ทำให้จัดการการตั้งค่าต่างๆได้ง่าย
พอเราห่อแอพเป็น container ถ้าเราอยาก scale หรือย้ายเครื่อง เราก็สามารถเอาทั้ง container ไปวางในที่ใหม่ได้เลย ไม่ต้องมา setup อะไรเยอะให้ยุ่งยาก
แต่ก็ยังมีเรื่องปวดหัวบางเรื่อง เช่น

- จะสเกล เราก็ยังต้องไปเปิดเครื่องใหม่ แล้วเอา container ไปวาง ทำตรงนี้เป็น as-a-code ได้มั้ย
- networking ระหว่าง container / service discovery / อื่นๆ…

เลยมี Tools ที่มาช่วยจัดการ container อีกที เรียกว่า…

Container Orchestration



โดยเจ้าเครื่องมือ Container Orchestration เนี่ย ก็จะเข้ามาจัดการตัว container อีกที โดยที่เรากำหนดโครงสร้างของระบบที่เราต้องการ เช่น รัน container A 3 ชุด container B 2 ชุด (ศัพท์เทคนิกเรียกว่า

เครื่องที่เป็นเครื่อง physical จริงๆมากนัก แถมพวก orchestration มักจะมาพร้อมกับคุณสมบัติ self-healing คือถ้าเครื่องจริงๆเครื่องนึงมันตายไป เดี๋ยวมันจัดแจงย้าย container ไปไว้ที่อื่นให้เองให้แอพเราทำงานได้ปกติตามโครงสร้างที่เรากำหนดไว้ เราแทบไม่ต้องสนใจเลยว่าจริงๆ แล้วมันเกิดอะไรในระดับ physical เย่
ข้อดีอีกอันนึงคือ Portability: สมมติวันนึงอยากย้าย Cloud จาก AWS ไป Google หรือย้ายมาทำ Private server การที่เราสามารถเขียนโครงสร้างของระบบเป็น config ได้ ทำให้เราย้ายไปมาได้สะดวกขึ้น

Kubernetes

(ด้วยความขี้เกียจ ต่อไปนี้เขียนว่า k8s)

Source: kubernetes.io

เว็บจั่วหัวไว้บนเว็บไซต์เลยว่าเป็น

> ***Production-Grade Container Orchestration:*** *Automated container deployment, scaling, and management*

แปลเป็นไทยก็คือเป็น platform ที่ช่วยจัดการ deploy และ scale container ของเรานั่นเอง ซึ่งข้อดีของ k8s คือมีความยืดหยุ่นสูง (รันบน cloud ไหนก็ได้ หรือ server ไหนก็ได้), มี API ให้เล่นได้, และมี Self-healing (ฮีลตัวเองให้กลับมาเป็นโครงสร้างแบบที่เราบอกมันได้)
การสร้างของต่างๆ ใน k8s จะทำโดยที่เราระบุ **Desired state** ว่าควรจะเป็นยังไง (แปลไทย: ระบุว่าเวลาที่ทุกอย่างปกติดี ควรจะมีอะไรทำงานยังไงบ้าง) แล้วตัว k8s จะพยายามจัดการของต่างๆให้เป็นไปตาม desired state ตลอดเวลา (Self-healing!)

> ***fun fact:*** *ก่อนจะเป็น k8s เนี่ย Google ใช้ container management system ที่ชื่อ Borg เพื่อจัดการ data center มากว่า 15 ปี แล้วเอาประสบการณ์และความเจ็บปวดมายำรวมกันพัฒนาต่อเป็น k8s*

องค์ประกอบของ k8s

(ชื่อหัวข้อฟังดูเหมือนมาจากหนังสือเรียน 555+)

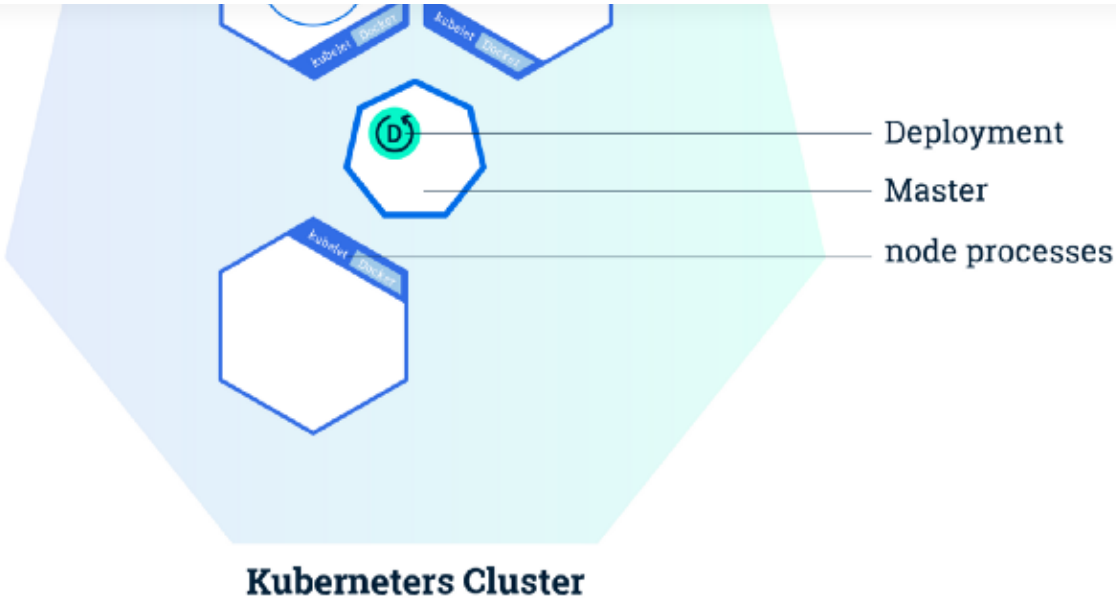
Cluster

ก็คือกลุ่มของ server (a.k.a node) ที่เอาไว้รัน container นั่นแหละ ถ้ามองภาพง่ายๆ คือ k8s มาครอบ cluster เอาไว้เสมือนเป็น server ใหญ่ๆ เครื่องเดียว โดยที่ตัว k8s จะจัดการให้เองว่าจริงๆ แล้วอะไรรันอยู่ที่ไหน
ใน cluster จะประกอบด้วย master และ node ปกติ

- master มีหน้าที่จัดการ node ว่าต้องทำอะไรบ้าง เวลาเราจะ deploy หรือแก้อะไรเราก็จะสั่งผ่าน master
- แต่ละ node มี process ชื่อ *kubelet* ไว้คุยกับ master และมี docker ในตัวเพื่อรัน container

Deployment

พอเราจะเอา app ของเราไปรันใน k8s เราจะสร้างสิ่งที่เรียกว่า **Deployment**



Source: kubernetes.io

เจ้าตัว deployment เนี่ย เป็น configuration ที่ไว้สร้าง(หรืออัพเดท) app และบอกว่า app เราจะรันยังไง (กำหนด desired state ของ app) โดยที่ k8s จะพยายามดูแลให้เป็นไปตาม config ตลอดเวลา (self-healing)
พอเราสร้าง deployment เพื่อกำหนด config ของแอพเราแล้ว มันจะสร้างและเอาแอพของเราไปรันอยู่ในสิ่งที่เรียกว่า **Pod**

Pod

Source: kubernetes.io

Pod เป็นหน่วยที่เล็กที่สุดในบรรดา k8s object ซึ่งแทนกลุ่มของ container ของเรา บวกกับ resource ต่างๆ ที่จำเป็นสำหรับ container นั้นๆ เช่น storage, network ip และยังเก็บ config ต่างๆเกี่ยวกับ container แต่ละอัน เช่น ต้องรันยังไง ต้องคุยผ่าน port ไหน
ดังนั้นของที่อยู่ใน pod เดียวกัน จะเปรียบเสมือนว่าลงอยู่ในเครื่อง host เดียวกัน (เหมือนเวลาเราลง PHP+MySQL+Apache ในเครื่องเดียวกันหมด) ดังนั้นแต่ละ container ก็จะคุยกันได้ผ่าน localhost เลย
เปรียบเทียบง่ายๆ คือเหมือนเซลล์ในร่างกาย เป็นหน่วยเล็กที่สุดยังไงอย่างงั้น ในเซลล์ก้อมีหลายๆ organism ที่สื่อสารกันเองได้ในเซลล์ (ใช่ปะ? ไม่ค่อยได้ตั้งใจเรียนชีวะเท่าไหร่ 55)

Source: kubernetes.io

พอเราส่ง deployment ที่มี config ของ pod เข้าไปแล้ว ตัว master ของ k8s ก็จะไปจัดแจงเองว่า pod จะไปอยู่ที่ node ไหนยังไง
ตัว pod เองก็จะมี lifecycle ของมัน เกิดแก่เจ็บตายเป็นเรื่องธรรมดา deployment ก็จะมีลูกน้องมือขวาอีกตัวนึงที่มาดูแล pod ให้มีจำนวนเท่าที่เรากำหนดบอกมัน ชื่อว่า…

Replica Set

ตอนที่เราส่ง deployment เข้าไปใน k8s เนี่ย นอกจากมันจะสร้าง pod แล้วมันยังสร้าง Replica Set ขึ้นมาด้วย
หน้าที่หลักของ Replica Set คือทำให้ pod มีจำนวนเท่ากับที่เราบอกมัน
ตัวอย่างเช่นเราบอกว่าอยากได้ pod myapp ทั้งหมด 4 replica ถ้ามีเกิน ตัว Replica Set มันก็ไปไล่เตะทิ้งให้ ถ้ามีน้อยกว่ามันก็จะสร้างเพิ่มให้ (self-healing แฮ่!)
นอกจากจะบอกจำนวนเป๊ะๆ ตัว Replica Set ยังทำ Autoscale ได้ เช่น ถ้า CPU Usage เกิน80% ให้สร้าง Pod เพิ่มอีก (ชื่อทางการคือ Horizontal Pod Autoscalers (HPA))

🎯 Bullet Points

- Deployment = Replica Set + Pod
- 1 Deployment จะมี 1 Replica Set และมี pod ได้ 1 แบบ แต่มีหลายๆ pod ได้จากการ scale (รูปข้างบนอาจจะทำให้เข้าใจผิดว่า 1 deployment มี pod หลายๆแบบได้ 😬)
- ถ้าสมมติจะทำ microservices (มีหลายๆ service) ก็จะใช้วิธีเขียนหลายๆ Deployment เอา
- เปรียบเทียบแบบชีวะ: pod = เซลล์แต่ละประเภท เช่น เซลล์ลูกตา เซลล์สมอง // Replica Set นี่คือคุมจำนวนเซลล์ // Deployment คืออวัยวะแต่ละอัน ลูกตา สมอง (1 function = 1 deployment)

To be continued

EP หน้า (S01E02) จะมาต่อเรื่องของ Services ที่ช่วยในการสเกลแอพ และลองเอา App ง่ายๆขึ้น k8s กันตัดจบ!
つづく

Programming Software Development Kubernetes Cloud Computing Infrastructure