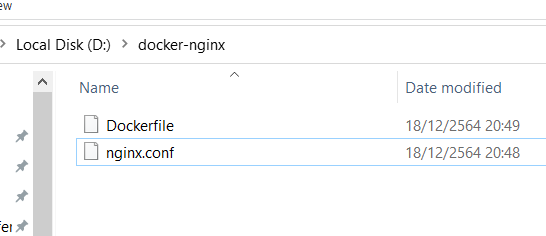


reverse proxy เป็นเซิร์ฟเวอร์ที่อยู่ระหว่างแอปพลิเคชันภายในและไคลเอ็นต์ภายนอก ส่งต่อคำขอของไคลเอ็นต์ไปยังเซิร์ฟเวอร์ที่เหมาะสม เนื่องจาก NGINX มีคุณลักษณะ การโหลดบาลานซ์ ความปลอดภัย และการเร่งความเร็วขั้นสูงจำนวนมากที่แอปพลิเคชัน เฉพาะทางส่วนใหญ่ไม่มี การใช้ NGINX เป็นพร็อกซีย้อนกลับทำให้เราสามารถเพิ่ม

คุณลักษณะเหล่านี้ในแอปพลิเคชันใดก็ได้ ในโพสต์นี้ เราจะตั้งค่า reverse proxy ด้วย NGINX และจะตั้งค่าสองแอปพลิเคชัน (อันหนึ่งบน NGINX และอีกอันบน apache)



**สร้าง Dockerfile**

FROM nginx:alpine

COPY nginx.conf /etc/nginx/nginx.conf

RUN apk update && apk add bash

**สร้างไฟล์ nginx.conf**

worker\_processes 1;

events { worker\_connections 1024; }

http {

sendfile on;

upstream docker-nginx {

server nginx:80;

}

upstream docker-apache {

server apache:80;

}

server {

listen 8080;

location / {

proxy\_pass http://docker-nginx;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Host $server\_name;

}

}

server {

listen 8081;

location / {

proxy\_pass http://docker-apache;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Host $server\_name;

}

}

}

#ต่อมาสร้าง Images ด้วยคำสั่ง

$ docker build -t nginx-alpine .

#เมื่อเสร็จแล้ว รันคำสั่ง

docker run -t -i nginx-alpine /bin/bash

bash-4.4# nginx -v

nginx version: nginx/1.19.3

#ถึงตอนนี้เราได้ Image สำหรับการรัน Service Container มาแล้ว 1 Container และต่อมาเราจะทำการสร้าง Images สำหรับ reverse proxy

docker build -t reverseproxy .

#แสดงข้อมูลของ Images File

docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nginx-alpine latest 741681cbc05a 11 minutes ago 27MB

reverseproxy latest 741681cbc05a 11 minutes ago 27MB

worker\_connections กำหนดจำนวนสูงสุดของการเชื่อมต่อพร้อมกันที่สามารถเปิดได้โดยกระบวนการของผู้ปฏิบัติงาน (default=1024)

โดยปกติแล้ว sendfile จำเป็นอย่างยิ่งในการเร่งความเร็วเว็บเซิร์ฟเวอร์ใดๆ ผ่านการส่งพอยน์เตอร์ (โดยไม่ต้องคัดลอกอ็อบเจกต์ทั้งหมด) ไปยังตัวอธิบายซ็อกเก็ตโดยตรง อย่างไรก็ตาม เราใช้ NGINX เป็น reverse proxy เพื่อให้บริการเพจจากเซิร์ฟเวอร์แอปพลิเคชัน เราสามารถปิดใช้งานได้

คำสั่งต้นน้ำใน ngx\_http\_upstream\_module กำหนดกลุ่มของเซิร์ฟเวอร์ที่สามารถฟังบนพอร์ตต่างๆ ดังนั้น คำสั่งต้นน้ำจึงถูกใช้เพื่อกำหนดพูลของเซิร์ฟเวอร์

Nginx สามารถร้องขอพร็อกซีไปยังเซิร์ฟเวอร์ที่สื่อสารโดยใช้ http(s), FastCGI, SCGI และ uwsgi หรือโปรโตคอล memcached ผ่านชุดคำสั่งแยกต่างหากสำหรับพร็อกซีแต่ละประเภท (โมดูล ngx\_http\_upstream\_module)

หลังจากกำหนดเซิร์ฟเวอร์ต้นน้ำแล้ว เราจำเป็นต้องบอก NGINX ถึงวิธีการฟังและวิธีตอบสนองต่อคำขอ

พร็อกซีที่ตรงไปตรงมาที่สุดเกี่ยวข้องกับการส่งคำขอไปยังเซิร์ฟเวอร์ที่สามารถสื่อสารโดยใช้ http พร็อกซีประเภทนี้เรียกว่า "proxy pass" ทั่วไป และได้รับการจัดการโดย proxy\_pass directive

คำสั่ง proxy\_pass ส่วนใหญ่จะพบในบริบทของตำแหน่ง และกำหนดโปรโตคอลและที่อยู่ของพร็อกซีเซิร์ฟเวอร์ เมื่อคำขอตรงกับตำแหน่งที่มีคำสั่ง proxy\_pass อยู่ภายใน คำขอจะถูกส่งต่อไปยัง URL ที่กำหนดโดยคำสั่ง

คำสั่ง proxy\_pass คือสิ่งที่ทำให้การกำหนดค่าเป็น reverse proxy ระบุว่าคำขอทั้งหมดที่ตรงกับบล็อกตำแหน่ง (ในกรณีนี้คือรูท / เส้นทาง) ควรส่งต่อไปยังพอร์ตเฉพาะบนโฮสต์ที่ระบุซึ่งแอปกำลังทำงานอยู่

ในข้อมูลโค้ดการกำหนดค่าข้างต้น จะไม่มีการกำหนด URI ที่ส่วนท้ายของเซิร์ฟเวอร์ในการกำหนด proxy\_pass สำหรับคำจำกัดความที่เหมาะสมกับรูปแบบนี้ URI ที่ลูกค้าร้องขอจะถูกส่งไปยังเซิร์ฟเวอร์ต้นน้ำตามที่เป็นอยู่

ดังนั้น หากเราพยายามเข้าถึงเครื่องโฮสต์ผ่านพอร์ต 8080 NGINX จะทำหน้าที่เป็นพร็อกซีย้อนกลับและให้บริการทุกอย่างที่อยู่ในคำจำกัดความของ proxy\_pass ในสถานการณ์ข้างต้น เรามี docker-nginx ซึ่งเป็นชื่อของหนึ่งในเซิร์ฟเวอร์อัปสตรีมของเรา ซึ่งหมายความว่าจะให้บริการ nginx

คำขอที่มาจาก NGINX ในนามของลูกค้าจะดูแตกต่างจากคำขอที่มาจากไคลเอ็นต์โดยตรง ส่วนใหญ่คือส่วนหัวที่สอดคล้องกับคำขอ เมื่อ NGINX พร็อกซี่คำขอ จะทำการปรับเปลี่ยนส่วนหัวของคำขอที่ได้รับจากลูกค้าโดยอัตโนมัติ:

NGINX กำจัดส่วนหัวที่ว่างเปล่า

ส่วนหัว "โฮสต์" ถูกเขียนใหม่เป็นค่าที่กำหนดโดยตัวแปร $proxy\_host นี่จะเป็นที่อยู่ IP หรือชื่อและหมายเลขพอร์ตของอัปสตรีม ตามที่กำหนดโดยคำสั่ง proxy\_pass โดยตรง

#ต่อมาเพิ่มส่วน Header

worker\_processes 1;

events { worker\_connections 1024; }

http {

sendfile on;

upstream docker-nginx {

server nginx:80;

}

upstream docker-apache {

server apache:80;

}

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Host $server\_name;

server {

listen 8080;

location / {

proxy\_pass http://docker-nginx;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Host $server\_name;

}

}

server {

listen 8081;

location / {

proxy\_pass http://docker-apache;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Host $server\_name;

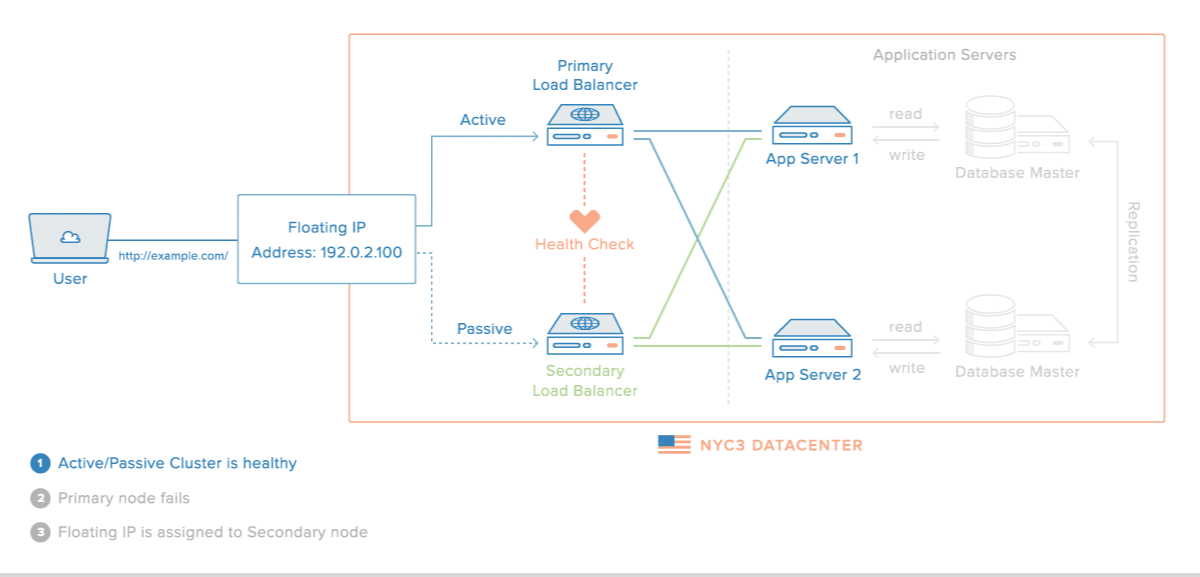
}

}

}

#เข้าไปดูและทำความเข้าใจได้ที่

<https://www.digitalocean.com/community/tutorials/understanding-nginx-http-proxying-load-balancing-buffering-and-caching>



#docker-compose.yml

ด้วย Compose เรากำหนดแอปพลิเคชันแบบหลายคอนเทนเนอร์ในไฟล์เดียว จากนั้นจึงหมุนแอปพลิเคชันของเราด้วยคำสั่งเดียวซึ่งทำทุกอย่างที่ต้องทำเพื่อให้ทำงานได้ หน้าที่หลักของ Docker Compose คือการสร้างสถาปัตยกรรมไมโครเซอร์วิส ซึ่งหมายถึงคอนเทนเนอร์และลิงก์ระหว่างกัน เมื่อเปรียบเทียบกับคำสั่ง docker คำสั่ง docker-compose ไม่เพียงแต่คล้ายคลึงกันเท่านั้น แต่ยังทำงานเหมือนกับคำสั่งของนักเทียบท่าอีกด้วย ข้อแตกต่างเพียงอย่างเดียวคือคำสั่ง docker-compose ส่งผลต่อสถาปัตยกรรมหลายคอนเทนเนอร์ทั้งหมดที่กำหนดไว้ในไฟล์คอนฟิกูเรชัน docker-compose.yml ไม่ใช่แค่คอนเทนเนอร์เดียว

มีสามขั้นตอนในการใช้ Docker Compose:

* กำหนดแต่ละบริการใน Dockerfile
* กำหนดบริการและความสัมพันธ์ซึ่งกันและกันในไฟล์ docker-compose.yml
* ใช้ docker-compose up เพื่อเริ่มระบบ

#สร้างไฟล์ docker-compose.yml:

services:

reverseproxy:

image: reverseproxy

ports:

- 8080:8080

- 8081:8081

restart: always

nginx:

depends\_on:

- reverseproxy

image: nginx:alpine

restart: always

apache:

depends\_on:

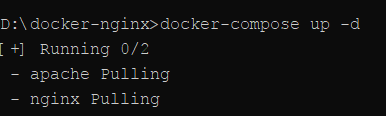
- reverseproxy

image: httpd:alpine

restart: always

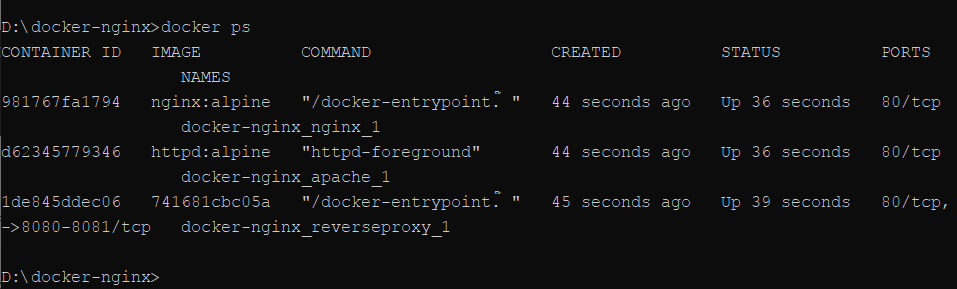
#เมื่อเสร็จแล้วรันคำสั่ง

$ docker-compose up -d

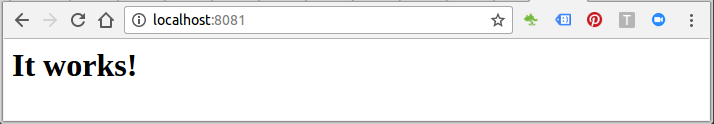


#เช็ค Process การทำงาน

$ docker ps



#เข้า Web Browser : localhost:8081



#เข้า Web Browser : localhost:8080



#เมื่อเสร็จแล้ว หากต้องการหยุด Container ก็ให้

$ docker-compose stop

#To remove (and stop) the container by docker-compose up,

docker-compose down