Contents

- **ทำความเข้าใจ Kubernetes และการใช้งานเบื้องต้น**
  - Pod
  - ReplicaSet
  - Deployment
  - Service
  - Configmap and Secret
  - Ingress
  - Namespace and Quota

Image Registry

Registry คือพื้นที่เก็บ source code และ dependencies ที่ build ให้อยู่ในรูป binary เรียกว่า Image เมื่อต้องการจะ deploy application ซึ่ง Container Engine จะทำการดึง Image จาก Registry ไป Run

Docker Hub

Github Container registry

Docker registry

Gitlab Container registry

Harbor

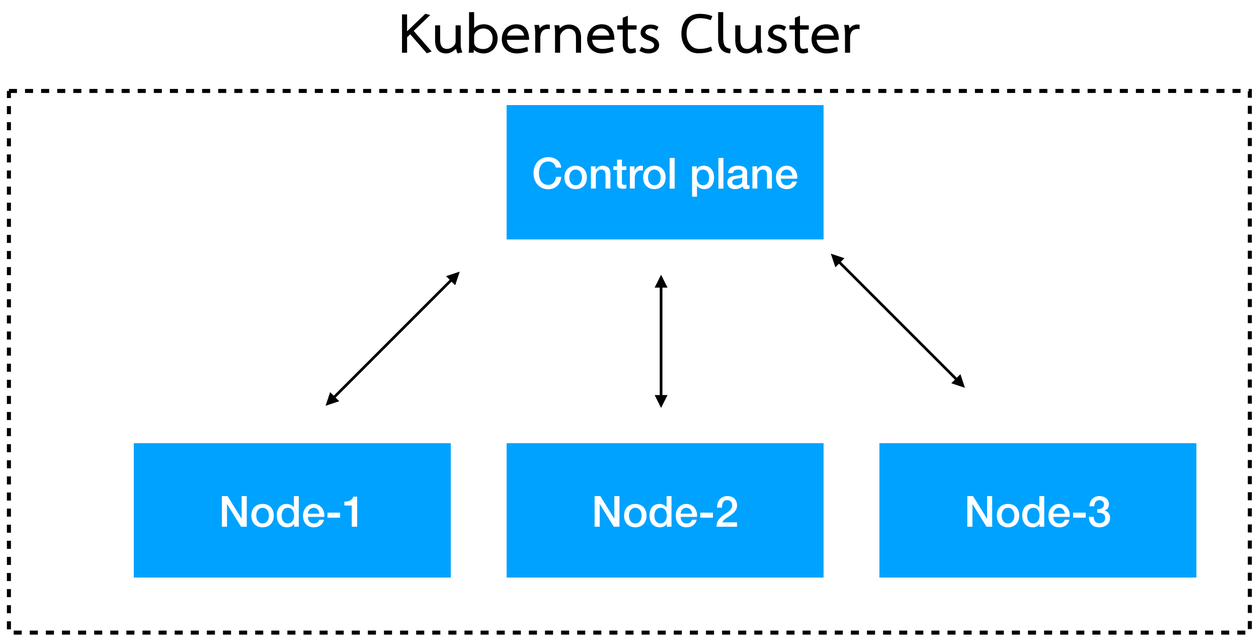
Nexus repository

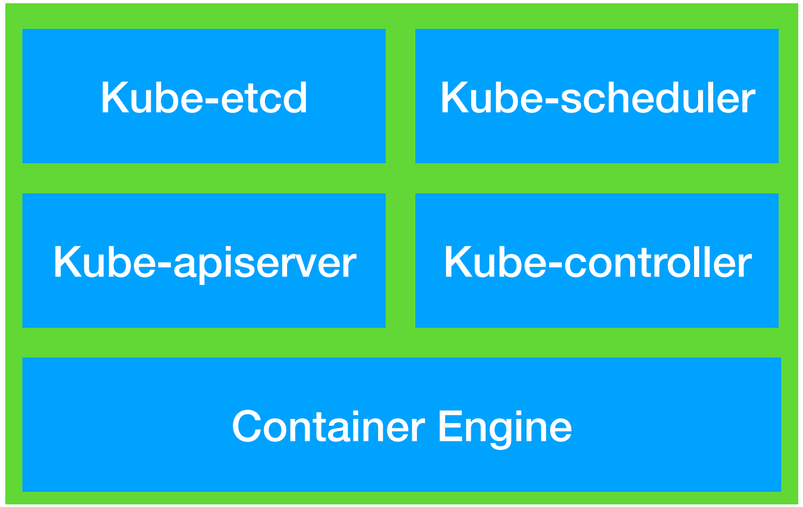
Google Container registry

ทำความเข้าใจ Kubernetes และการใช้งานเบื้องต้น

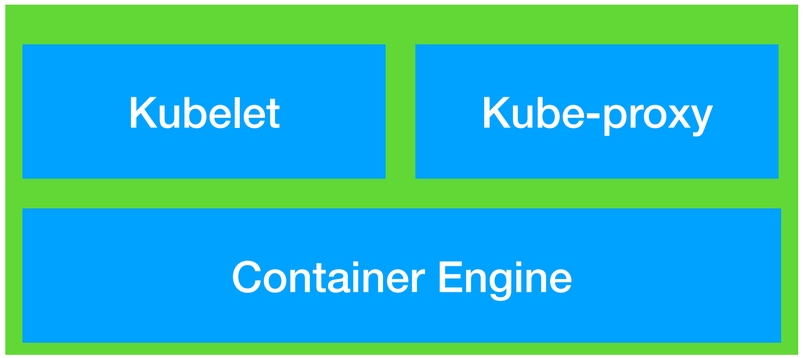
Kubernetes เป็นเครื่องมือที่ช่วยบริหารจัดการ Container จำนวน ๆ ในหลาย ๆ ที่ โดยทำเรื่องยากให้กลายเป็นเรื่องง่ายโดยเพียงไม่กี่คำสั่ง และมีความสามารถเพิ่มเติมเข้ามา เช่น การ scalling เพื่อรองรับ traffic ตามความต้องการ หรือ เรื่องการรักษาความผิดพลาดที่ผิดปกติด้วยตัวมันเอง เป็นต้น

Kubernetes



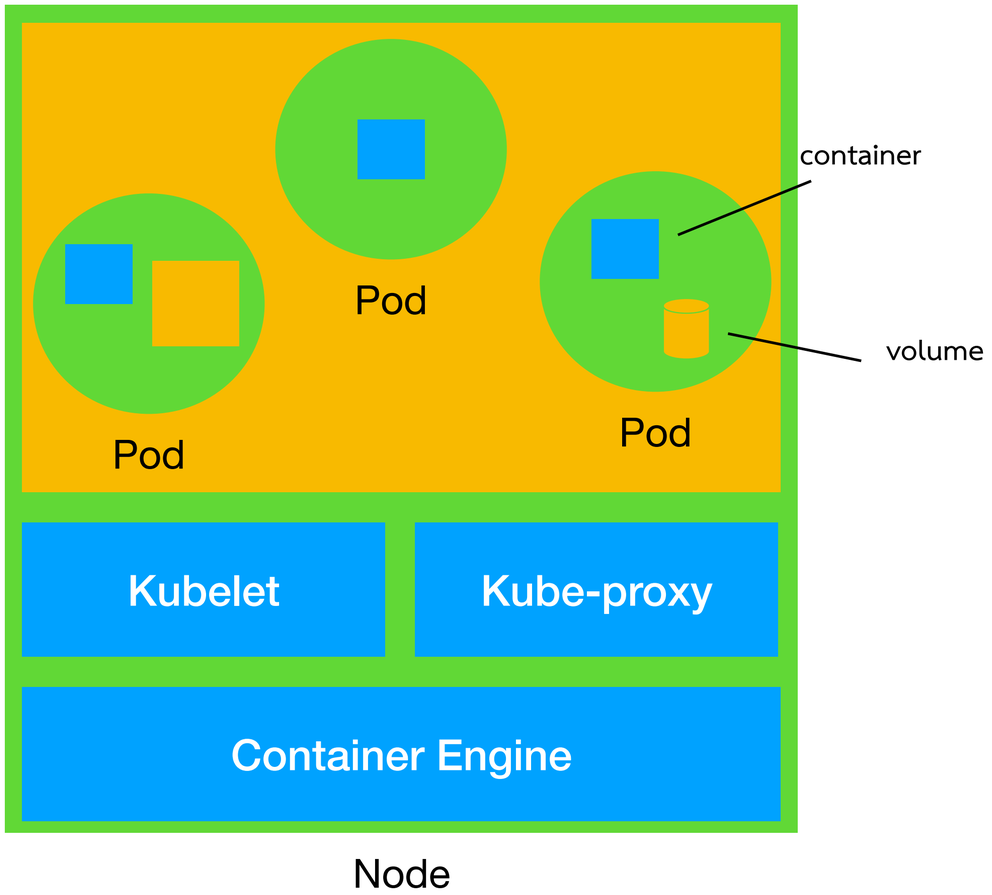


Control plane



Node

Kubernetes



Pod คือ หน่วยการทำงานของ Applications ซึ่ง Pod สามารถบรรจุ Container มากกว่า 1 ขึ้นไป โดยทั่วไป Pod จะมีสถานะเป็น stateless และสามารถตายได้

Volume คือการ mount พื้นที่ภายใน container ไปสู่ storage ภายนอกเพื่อเก็บรักษาข้อมูลไว้เมื่อ Pod ตายข้อมูลภายใน Pod จะไม่หายไป ส่วนพื้นที่ภายนอก เช่น S3, NFS, GlusterFS และ Storage ที่ support CSI Driver เช่น HPE Alletra และ 3PAR เป็นต้น

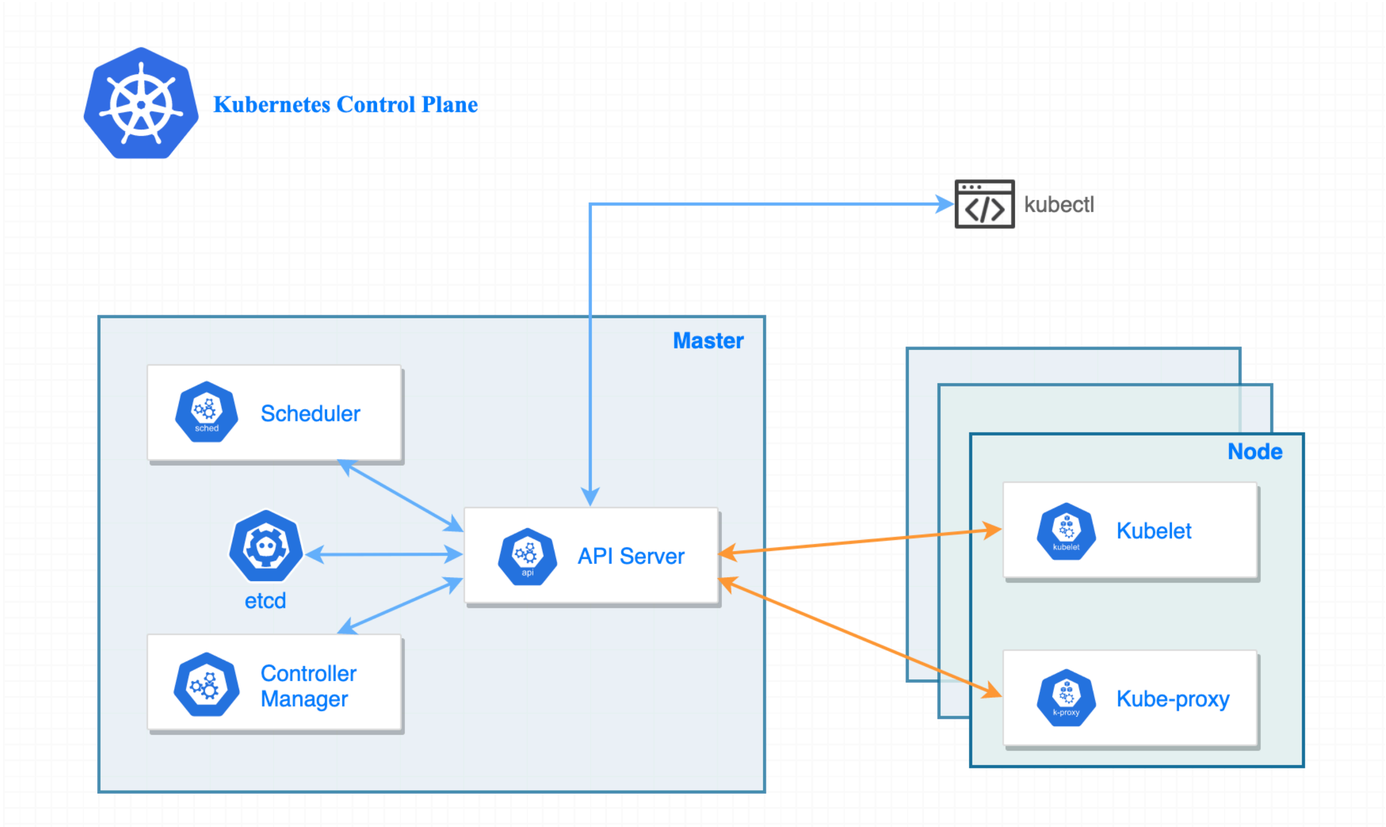
v1.20 Kubernetes deprecated Docker

v1.24 Kubernetes removed Docker

Looking new Container Engine

cri-o

Kubernetes



https://kubesphere.io/blogs/monitoring-k8s-control-plane/

Kube-apiserver ทำหน้าที่ ในการ validate และเรื่องการ config ข้อมูลที่เกี่ยวข้องกับ Pods, Services, Deployment เป็นต้น ซึ่ง kube-apiserver ให้บริการในรูปแบบ RES API

Kube-scheduler ทำหน้าที่ในการสังเกตการณ์คำสั่งในการสร้าง Pod ใหม่จาก kube-apiserver ซึ่งจะนำไปคำนวนเพื่อหาว่า Pod ที่จะถูกสร้างควรจะอยู่ที่ Node อะไร

Kube-etcd เป็น memory-storage ทำหน้าที่ในการเก็บค่าสถานะต่าง ๆ เช่น actual state ที่เป็นจำนวนปัจจุบันของ Pod ที่มีสถานะ Running, desire state คือจำนวน Pod ที่ต้องมีสถานะ Running ตามที่ตั้งไว้ใน configure และ เก็บ configure ต่าง ๆ เป็นต้น

Kube-controller Manager ทำหน้าที่ในการสังเกตการณ์ค่าสถานะภายใน Cluster ว่ามีค่าไหนที่เปลียนไปและไม่ตรงกับค่าที่ต้องการตาม configure กำหนดไว้ถ้าพบว่าไม่ตรง kube-controller จะทำการสร้างคำสั่งส่งไปที่ kube-apiserver เพื่อทำให้เท่ากัน ยกตัวอย่างบาง kube-controllers เช่น

- Replicas controller รับผิดชอบในการทำให้จำนวนของ Pod ณ ปัจจุบันเท่ากับจำนวน Pod ที่ต้องการที่ Configure กำหนดไว้
- Node controller รับผิดชอบในการแจ้งเตือนเมื่อ Node ตาย

Kubelet เป็น Agent Node เพื่อใช้ในการติดต่อกับ kube-apiserver ค่อยรับคำสั่งต่าง ๆ เช่น สร้าง Pod, ลบ Pod, restart Pod, ตรวจสอบสถานะ Pod โดยจะนำคำสั่งเหล่านี้ไปส่งไปยัง Container Engine ภายใน Node เพื่อดำเนินงานต่ออีกที

Kube-proxy เป็น Network จำลองที่ติดตั้งอยู่ทุก ๆ Node โดยแต่ละ Node สามารถติดต่อหากันได้ผ่าน Network จำลองนี้รวมไปถึง Client ที่ต้องการจะเรียกใช้ App บน K8S จะผ่าน kube-proxy เช่นกัน และช่วยในเรื่อง forwarding และ roundrobin

Kubernetes

Kind Objects

https://medium.com/devops-mojo/kubernetes-objects-resources-overview-introduction-understanding-kubernetes-objects-24d7b47bb018

Kubernetes - Pod

Pod คือ หน่วยการทำงานของ Applications ซึ่ง Pod สามารถบรรจุ Container มากกว่า 1 ขึ้นไป โดยทั่วไป Pod จะมีสถานะเป็น stateless แต่สามารถทำเป็น Statefull ได้เช่นกันโดยการทำ Volume และ Pod สามารถตายได้ตลอดเวลา

resource limit

เราสามารถกำหนด ความต้องการในการใช้งานของ CPU และ Memory ได้
โดยกำหนด ค่าพื้นฐาน (request) และ ค่าจำกัด (limit) ของการใช้งานให้กับ
Container ของ Pod

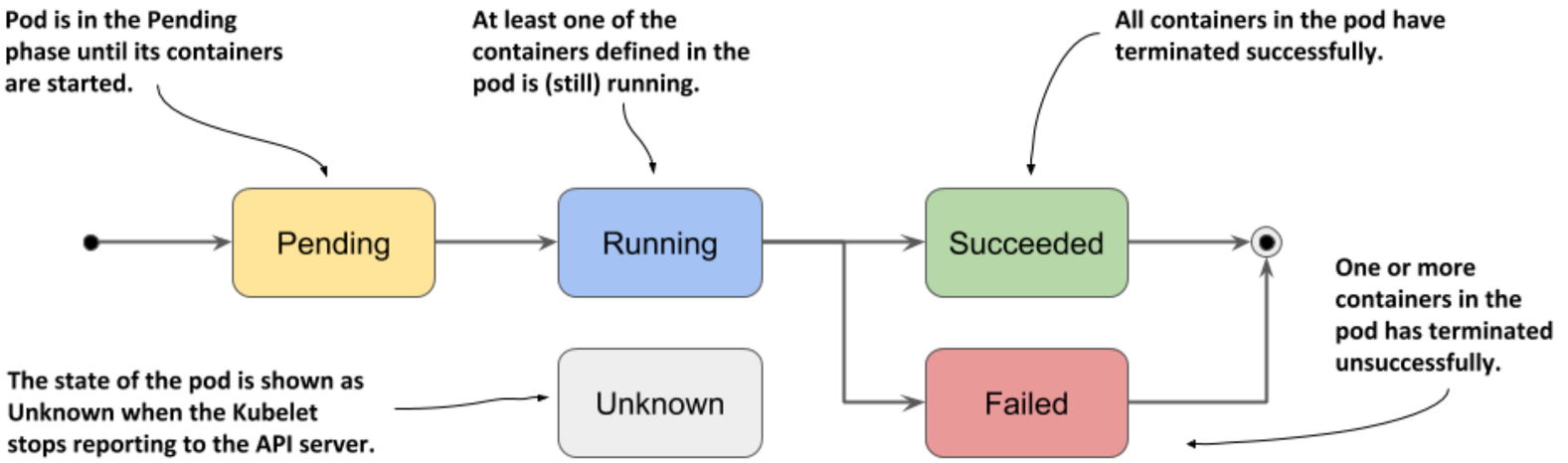
การกำหนด CPU ต้องเป็นจำนวนเต็มบวกหรือทศนิยมมีหน่วยเป็น milicore ตัวอย่าง ดังนี้

1) 10mi หมายถึง 10 mili-core
2) 1000mi หรือ 1 หมายถึง 1 core
3) 0.1 หมายถึง 100 mili-core

การกำหนด Memory ต้องเป็นจำนวนเต็มบวกหรือทศนิยมมีหน่อยเป็น Mi, Gi, Ti, Pi, Ei เป็นต้น ตัวอย่าง ดังนี้

1) 10Mi หมายถึง 10 Mebibytes
2) 10M หมายถึง 10 Megabytes
3) 1Gi หมายถึง 1 Gibibytes

Pod Life Cycle



LAB1 - Pod

Ref : https://github.com/phyze/k8s/tree/main/pod

ReplicaSet

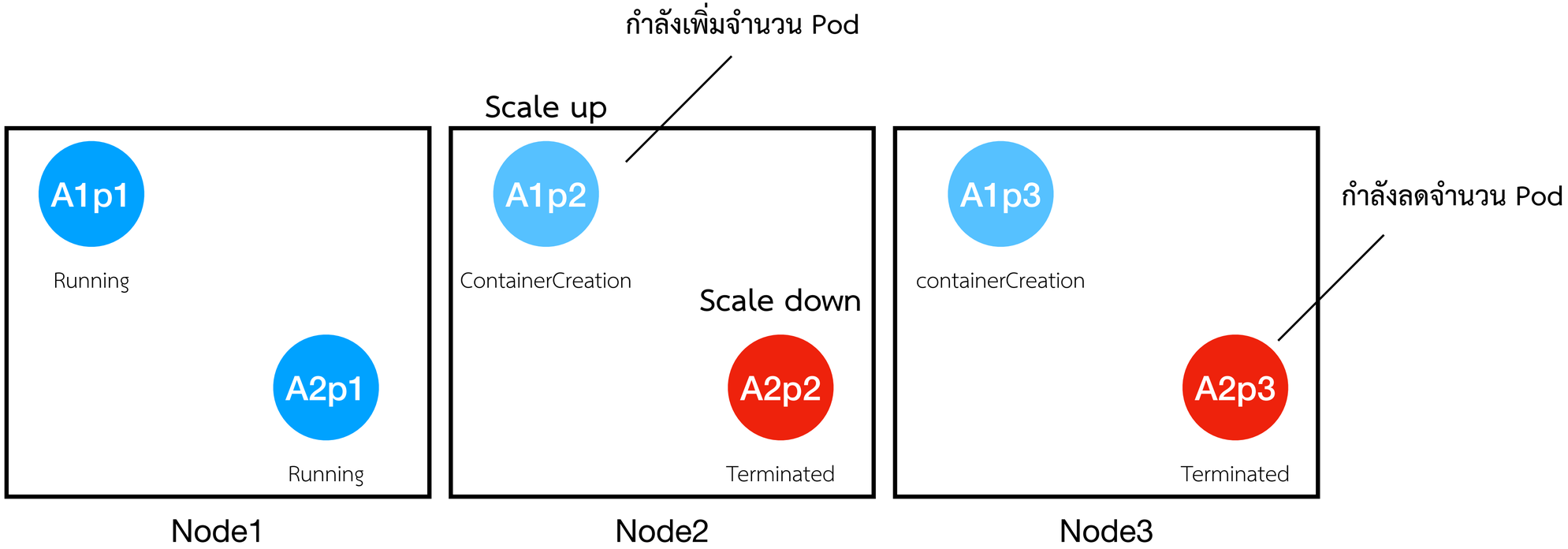
ด้วยความสามารถของ Pod เองไม่สามารถ scale up หรือ scale down ได้ และไม่สามารถรักษาตัวเองเมื่อเกิด crash ขึ้นทำให้ Pod ตาย ดังนั้นจึงนำ ReplicaSet เข้ามาช่วยในการ Scaling และ Self-healing

Twitter/@hameed_shahir

Scaling

Scaling แบ่งออกเป็น 2 ประเภท ดังนี้

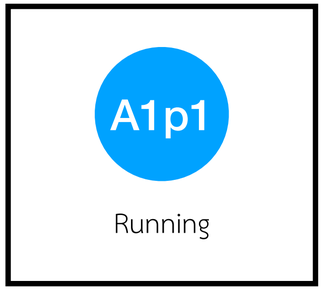
1) **Scale up** คือการเพิ่มจำนวน Pod ที่เป็น Pod ชนิดเดียวกันออกไปอยู่ในแต่ละ Nodes
2) **Scale down** คือการลดจำนวน Pod ที่ถูกสำเนา



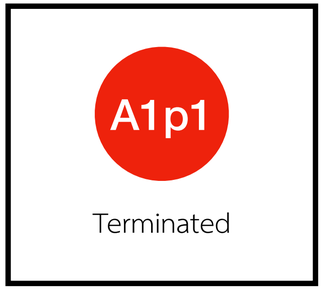
Self-healing

Self-healing คือกลไกลในการรักษาตัวเอง (Pod) ของ K8S ในกรณีที่เกิด crash หรือ Pod ถูกลบ K8S จะทำการสร้าง Pod ขึ้นมาใหม่ให้เท่ากับจำนวนที่ต้องการจาก config file หรือผ่าน command kubectl

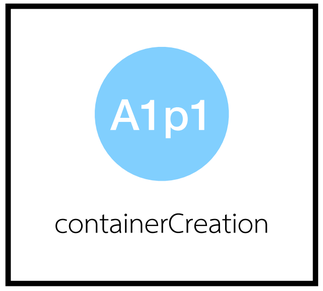
Step 1 - ก่อนโดนลบ Pod อยู่ในสถานะทำงานปกติ



Step 2 - โดนลบ Pod อยู่ในสถานะกำลังปิดตัว



Step 3 - หลังลบหลังจากที่ Pod ปิดตัว K8S ทำการสร้าง Pod ขึ้นมาใหม่ให้จำนวนเท่ากับของเดิม



LAB2 - ReplicaSet

Ref : https://github.com/phyze/k8s/tree/main/replicaSet

Deployment

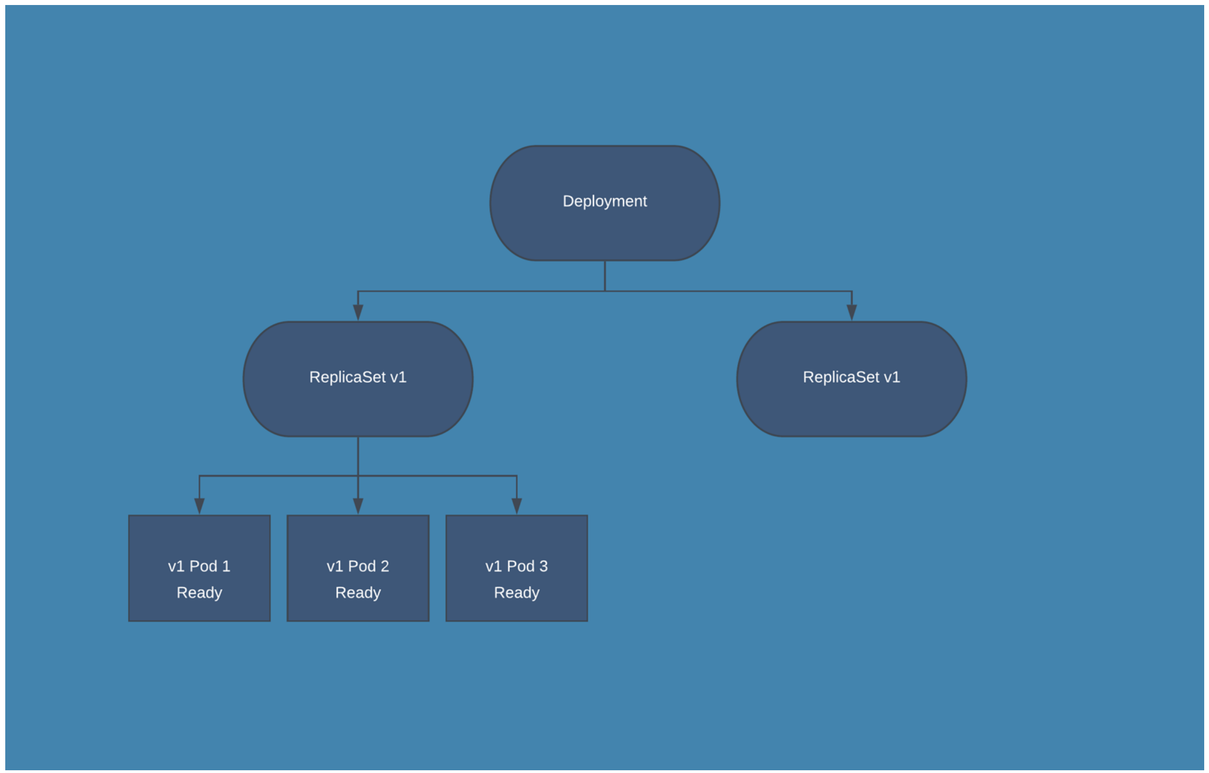
เมื่อต้องการ Update version ของ Application ซึ่ง replicaSet ไม่สามารถทำได้โดยตรงจากการเปลียน tag ของ image ดังนั้นจึงนำ kind Deployment เข้ามาช่วยในการ Update version ของ Application โดยการสร้าง replicaSet ใหม่ขึ้นแล้วสร้าง Pod ที่เป็น version ใหม่ภายใต้ replicaSet ใหม่ เรียกวิธีนี้ว่า Rolling Update

และยังมีความสามารถของ Deployment ดังนี้

- Rollback
- Strategy
- Liveness
- Readiness

Kubernetes - Deployment

Rolling Update



https://www.bluematador.com/blog/kubernetes-deployments-rolling-update-configuration

สร้างเกตุว่า มีการสร้าง ReplicaSet v2 ขึ้นมาใหม่ จากนั้นทำการ terminate pod ที่ ReplicaSet v1 และสร้าง Pod ที่ ReplicaSet v2 จนครบจำนวนที่กำหนดไว้ใน config เรียกว่า Desire state และทุกการ rolling update มีการเก็บชุด ReplicaSet เก่าไว้สำหรับ rollback

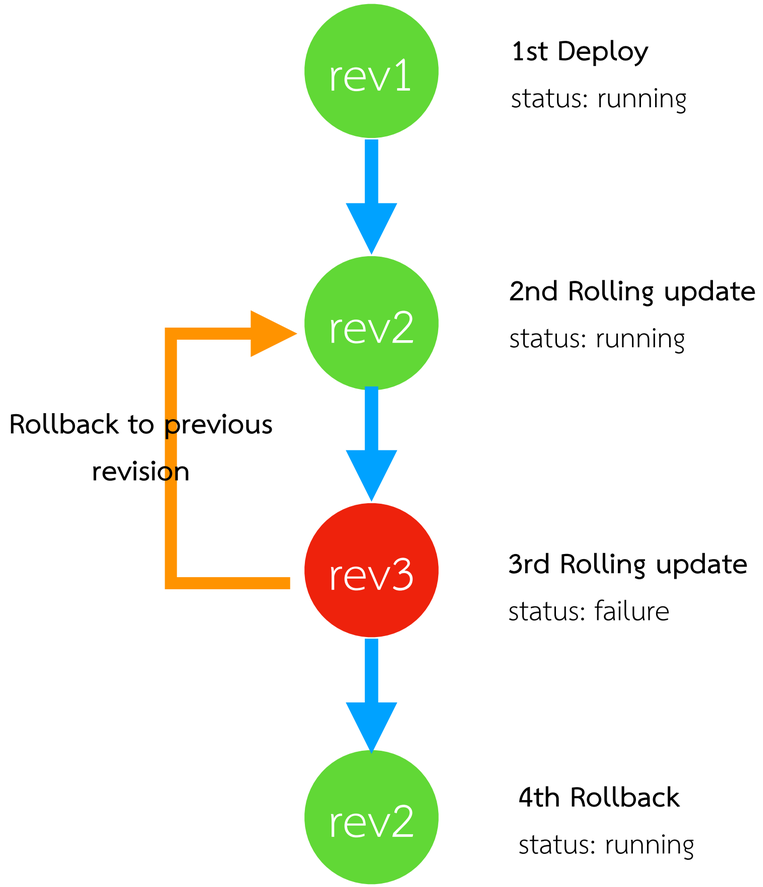
NOTE: Desire state คือ ค่าที่ถูกกำหนดไว้ใน configuration file แบบชัดเจน เช่น YAML, kubectl CMD ที่มีการระบุค่าที่ต้องการไว้เป็นจำนวนหนึ่ง ๆ ยกตัวอย่าง กำหนดให้ app A มี 3 replicas แสดงว่าเมื่อทำการ apply แล้วต้องมี 3 pods ที่มีสถานะ running และ pod ต้อง healty ถึงจะบอกได้ว่า desire state คือ 3

Rollback

เมื่อการ deploy ไม่ได้เป็นไปตามที่คาดหวังการ update app ให้เป็น version ล่าสุดอาจเกิดข้อผิดพลาดจำเป็นต้องการย้อนกลับไปเป็น version เก่าและทุก ๆ ครั้งที่มีการ deploy เกิดขึ้นจะมีการเก็บ history ของ replicaSet ก่อนหน้าไว้ (revision) เพื่อสำหรับการทำ Rollback

วิธีที่ใช้ในการ Rollback มีสองวิธี ดังนี้

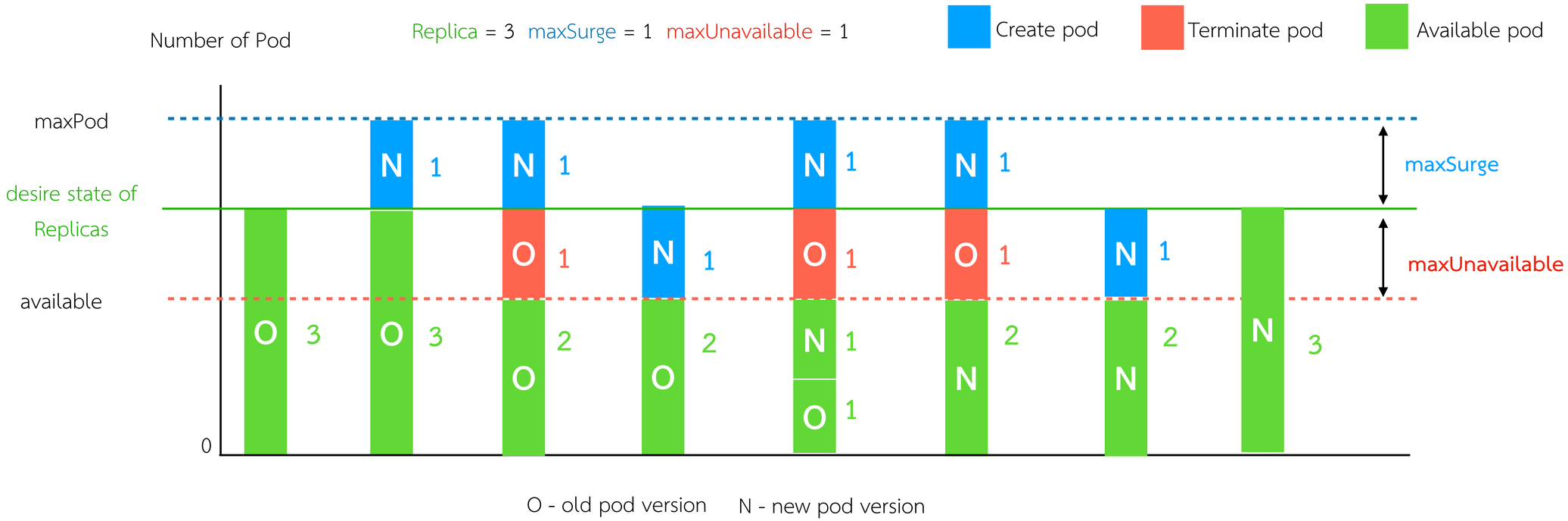
1) วิธี rollback ผ่าน kubectl CLI ด้วย rollout undo โดย default จะถอยกลับไป 1 revision แต่ถ้าต้องการกลับไป 2 revision ก่อนหน้าต้องใส่ --to-revision=2 ต่อท้าย
2) วิธี rollback ผ่าน config yaml ด้วยการเปลียน tag ของ image เป็น tag ก่อนหน้า



Kubernetes - Deployment

Strategy

คือ กลยุทธ์ในการควบคุมการ Rolling update ระหว่างลบ pod ที่ replica เก่าและสร้าง pod ที่ replica ใหม่
โดยมี 2 ตัวแปรในการกำหนดทิศทางของการ Rolling update ดังนี้
maxSurge คือ สามารถมี Pod ได้เกินจำนวน replicas ที่ตั้งไว้ได้กี่ Pod ในระหว่างการ update
maxUnavailable คือ สามารถลบ Pod ได้สูงสุดกี่ Pod และต้องเหลือจำนวนที่สามารถทำงานได้โดยไม่ต่ำไปกว่านี้

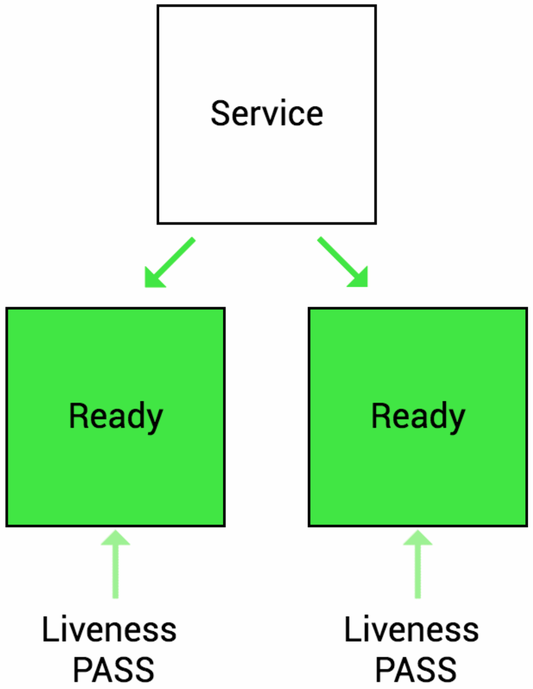


Liveness probe

เป็นกลวิธีที่ที่ใช้ในการตรวจสอบสุขภาพหรือสถานะการทำงานของ Application เพื่อให้แน่ใจว่ายังสามารถทำงานได้ปกติ ซึ่ง Application จำเป็นต้องกำหนดวิธีตรวจสอบและเงื่อนไขลงไปด้วยบางส่วน หากตรวจสอบแล้วไม่ตรงเงื่อนไขทาง K8S จะทำการ restart Pod เพื่อให้ Application กลับมาทำงานได้ตามปกติ

Liveness probe มี parameter หลัก ๆ อยู่ 2 อย่างคือ
initialDelaySeconds เป็นการตั้งค่าเวลา delay ก่อนเริ่มการตรวจสอบหลังจาก Pod ถูกสร้าง
periodSeconds เป็นการตั้งค่าเวลาให้ทำการตรวจสอบทุก ๆ ในช่วงเวลาที่กำหนด

ตัวอย่าง
กำหนดให้ initialDelaySeconds เท่ากับ 10 วินาที และ periodSeconds เท่ากับ 3 วินาที หมายความว่าหลังจากที่ Pod ถูกสร้าง initialDelaySecond ก็เริ่มทำการนับเวลาจนถึง 10 วินาที และทำการตรวจสอบ Liveness และนับต่อไปอีก 3 วินาทีจะทำการตรวจสอบอีกรอบวนไปเรื่อย ๆ



https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-setting-up-health-checks-with-readiness-and-liveness-probes

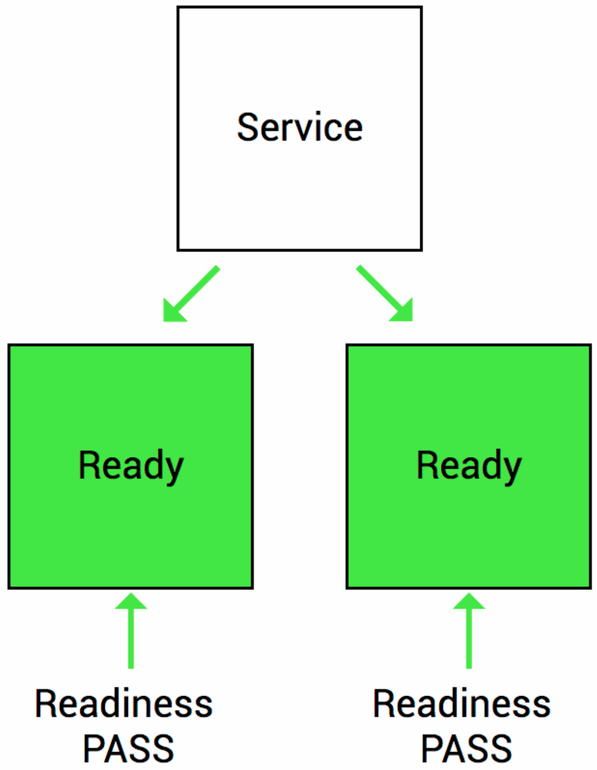
Readiness probe

เป็นกลวิธีที่ที่ใช้ในการตรวจสอบความพร้อมในการทำงานของ Application หาก Application ยังไม่พร้อม จะไม่มีการปล่อย Traffic เข้ามาเพื่อป้องการไม่ให้เกิดความผิดพลาดของการทำงาน อะไรจะเกิดขึ้นหาก Application มีการติดต่อกับ Database แต่ Database ไม่สามารถทำงานได้ แต่ผู้ใช้ยังสามารถเรียกทำธุรกรรมบางอย่างกับ Application ได้จนถึงขั้นตอนที่สำคัญเมื่อผู้ใช้ทำการยืนยัน ข้อมูลถูกส่งไปบันทึกลง Database แต่ไม่สามารถบันทึกได้และผู้ใช้ได้รับ Message error ดังนั้นการใช้ Readiness probe จะช่วยลดความเสี่ยงที่จะเกิดขึ้นหากมีระบบภายในหรือระบบภายนอกไม่สามารถเชื่อมต่อหรือใช้งานได้

แต่ผู้พัฒนาจำเป็นต้องออกแบบวิเคราะห์ว่า Application มีการเชื่อมต่อกับใครและต้องตรวจสอบอะไรบ้างถึงสามารถเรียกว่า พร้อม

Readiness probe มี parameter หลัก ๆ อยู่ 2 อย่างคือ
initialDelaySeconds เป็นการตั้งค่าเวลา delay ก่อนเริ่มการตรวจสอบหลังจาก Pod ถูกสร้าง
periodSeconds เป็นการตั้งค่าเวลาให้ทำการตรวจสอบทุก ๆ ในช่วงเวลาที่กำหนด

ตัวอย่าง
กำหนดให้ initialDelaySeconds เท่ากับ 10 วินาที และ periodSeconds เท่ากับ 3 วินาที หมายความว่าหลังจากที่ Pod ถูกสร้าง initialDelaySecond ก็เริ่มทำการนับเวลาจนถึง 10 วินาที และทำการตรวจสอบ Readiness และนับต่อไปอีก 3 วินาทีจะทำการตรวจสอบอีกรอบวนไปเรื่อย ๆ



https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-setting-up-health-checks-with-readiness-and-liveness-probes

LAB3 -Deployment

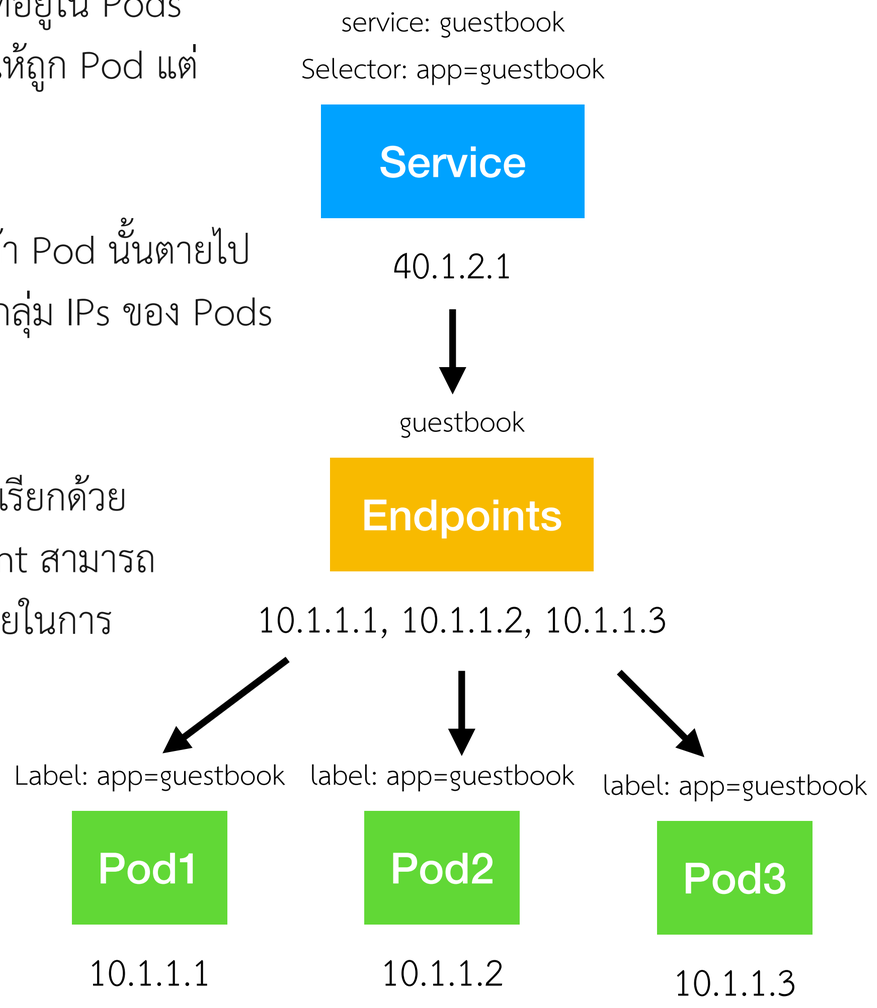
Ref : https://github.com/phyze/k8s/tree/main/deployment

Service

เป็นกลไกที่ใช้ในการเปิดช่องทางให้ Client ทั้งภายในและภายนอก Cluster สามารถติดต่อกับ Application ที่อยู่ใน Pods โดยใช้วิธีการ label selector เพื่อจัดกลุ่ม Pod ที่ได้มีการประกาศ label ตรงกันเพื่อที่จะ forward request ให้ถูก Pod แต่ก็ยังไม่สามารถ forward ได้ทันที Service จำเป็นต้องระบุ IP ให้ชัดเจนในการ forward ไปยังปลายทาง

Pod มี IP เป็นของตัวเองตั้งแต่โดนสร้างขึ้น แต่การที่จะให้ Service พูดคุยกับ Pod โดยตรงก็สามารถทำได้แต่ถ้า Pod นั้นตายไป จำเป็นต้องมากำหนดค่ากันใหม่เพราะ IP จะไม่คงเดิม ดังนั้นจึงนำ Endpoints Object เข้ามาช่วยทำในการจัดกลุ่ม IPs ของ Pods ที่ match กับ label ที่กำหนดไว้ หากมี Pod ใดตาย หรือ เกิดใหม่จะถูก Update ลงไปใน Endpoints

Service เมื่อถูกสร้างจะได้รับ Vitual IP ที่ไม่เปลี่ยนแปลงจนกว่า Service นั้นจะถูกทำราย แต่การที่ให้ client เรียกด้วย IP คงไม่สะดวก ดังนั้น K8S นำ IP ไปจับคู่กับ ชื่อ Service และบันทึกไว้ใน DNS ภายใน Cluster วิธีนี้เอง Client สามารถเรียก Service ด้วยชื่อคล้ายกับ ชื่อ domain บน web browser และนอกจากนี้ การติดต่อผ่าน Service ยังช่วยในการกระจาย request ไปสู่ Pod อย่างเท่า ๆ กันอีกด้วย (Load balancing)



Service

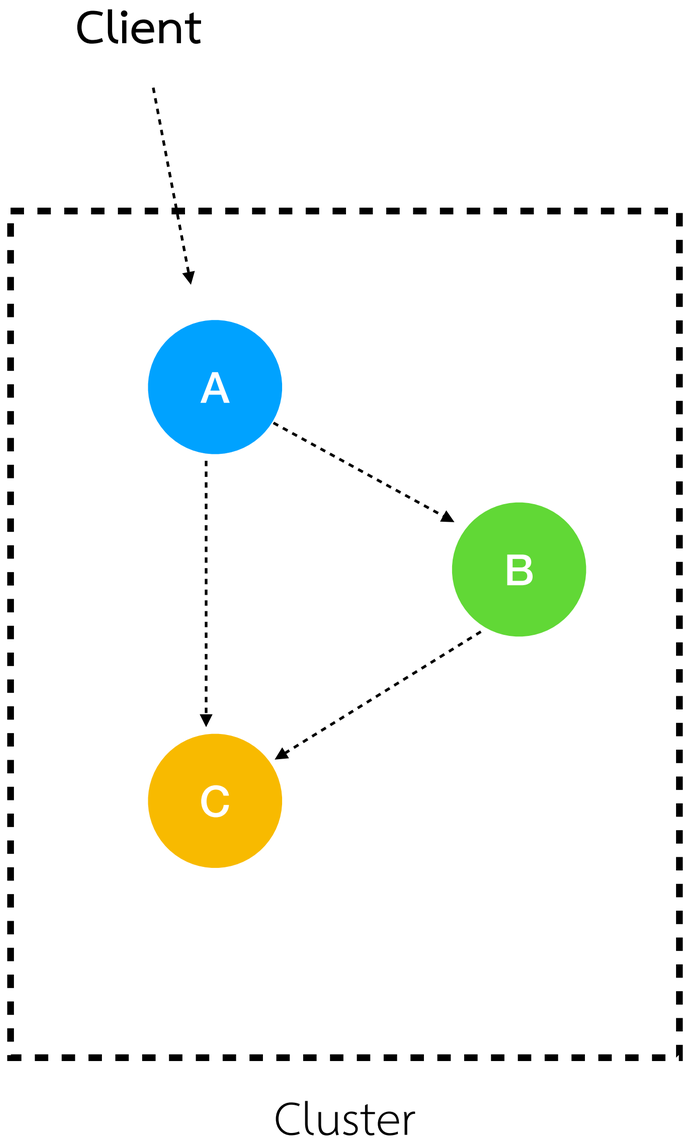
Service มีประเภทของการเปิดช่องทาง (Expose) ที่ใช้กันบ่อย ๆ อยู่ 4 อย่าง ดังนี้

ClusterIP ใช้สำหรับภายในติดต่อสื่อสารกันเองเท่านั้นซึ่งจะได้รับชุด Virtual IP เมื่อ Service ถูกสร้าง

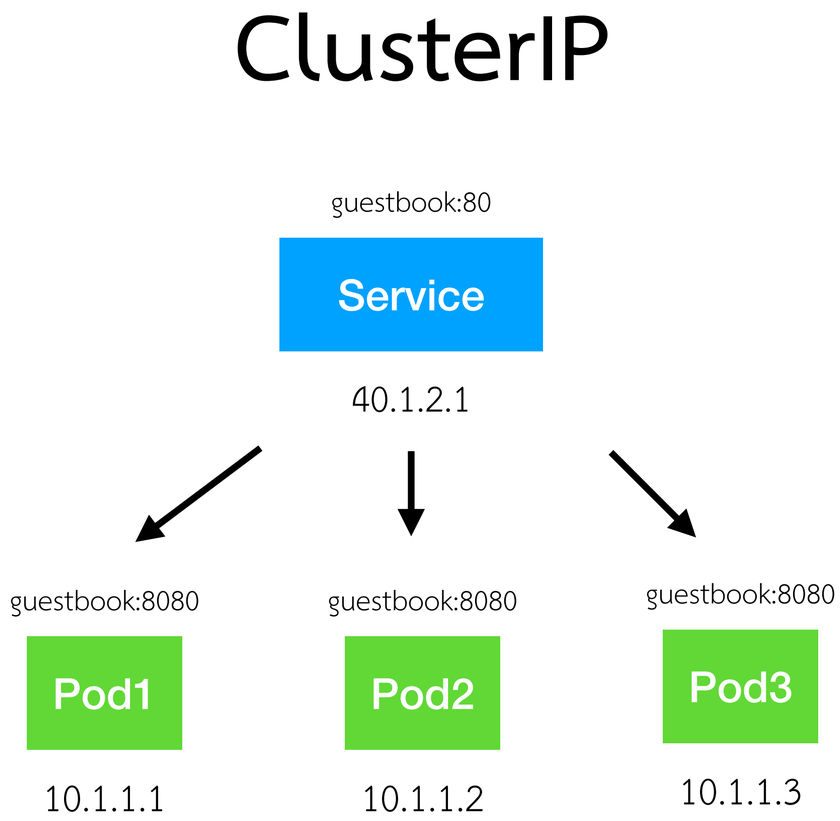
NodePort เหมือนกับ ClusterIP แต่เพิ่มเติมคือภายนอกสามารถติดต่อสื่อสารเข้ามาได้และตอนที่ Service ถูกสร้างขึ้น Port ที่ใช้ติดต่อจะถูก random ที่อยู่ในช่วง 30000-32767 โดย Default แต่ก็สามารถกดหนด port เองได้เช่นกัน

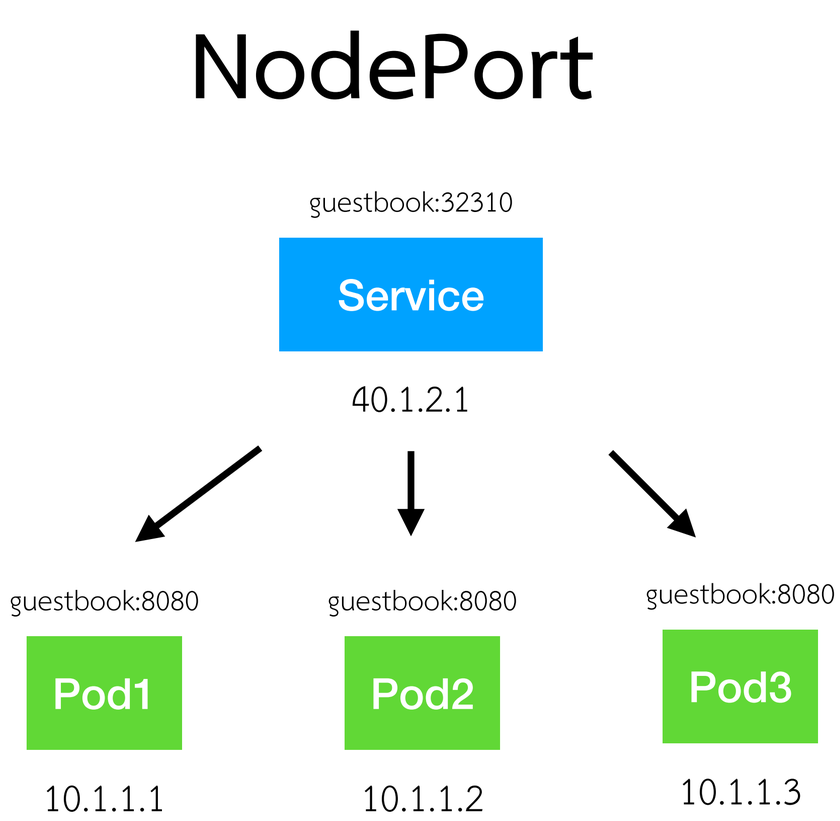
LoadBalancer ความสามารถเหมือนกับ ClusterIP และ NodePort แต่ที่เพิ่มเติมคือ IP ที่ได้รับเมื่อ Service ถูกสร้างไม่ใช่ Virtual IP แต่เป็น Public IP ซึ่ง Service ประเภทนี้จะใช้ได้บน Cloud เช่น Azure Kubernetes Service , Google Kubernetes Engine และ Amazon Elastic Kubernetes Service เป็นต้น

ExternalName เป็นการ Map domain name ของ 3rd party ภายนอกเข้ากับชื่อ Service ภายใน cluster



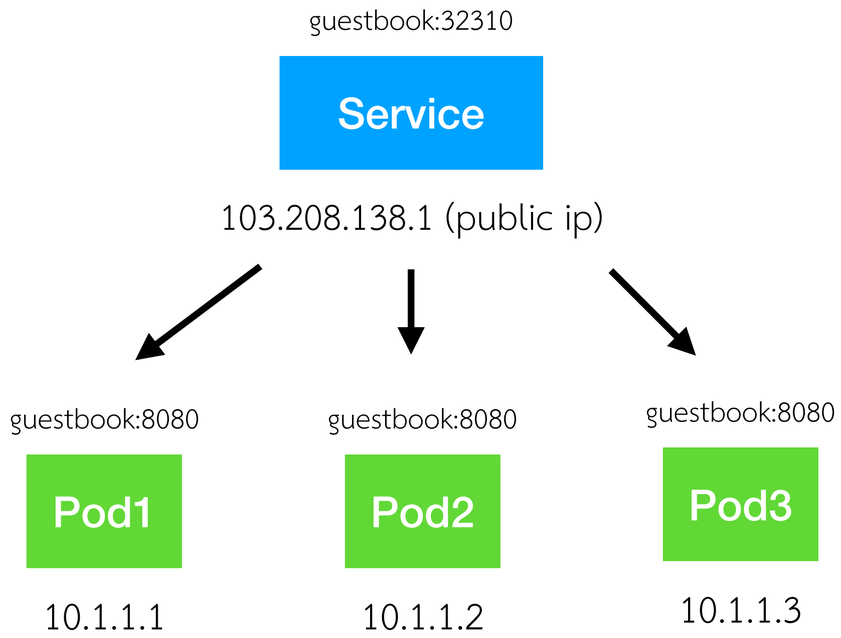
Kubernetes - Service



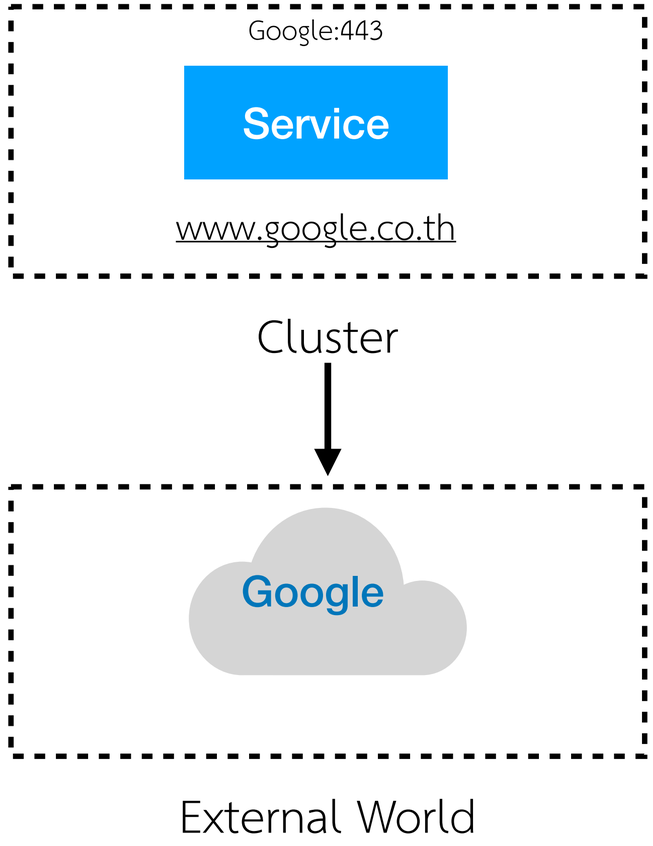


Kubernetes - Service

LoadBalancer



ExternalName



LAB4 - Service

Ref : https://github.com/phyze/k8s/tree/main/service

ConfigMap

เมื่อมีการเปลียนแปลง Config ของ App สิ่งหนึ่งที่ต้องหลีกเลี่ยงไม่ได้คือการ Build Image ใหม่ ดังนั้นเพื่อต้อง Build ทุกครั้งและง่ายต่อการปรับเปลียน Config ในแต่ละ Environment เช่น dev, uat, staging ไม่จำเป็นต้อง Build ใหม่ทุกรอบ แนะนำให้ใช้ ConfigMap แทน

ซึ่งเป็นกลวิธีทีที่เก็บ Config file ไว้บน K8S Cluster และเมื่อ App ต้องการใช้ Config เพียงแค่ทำการ mapping path เข้าไปใน Container ภายใน Pod และอ้างถึงชื่อของ ConfigMap ซึ่งการ Upload ConfigMap ขึ้นไปเก็บไว้บน K8S Cluster

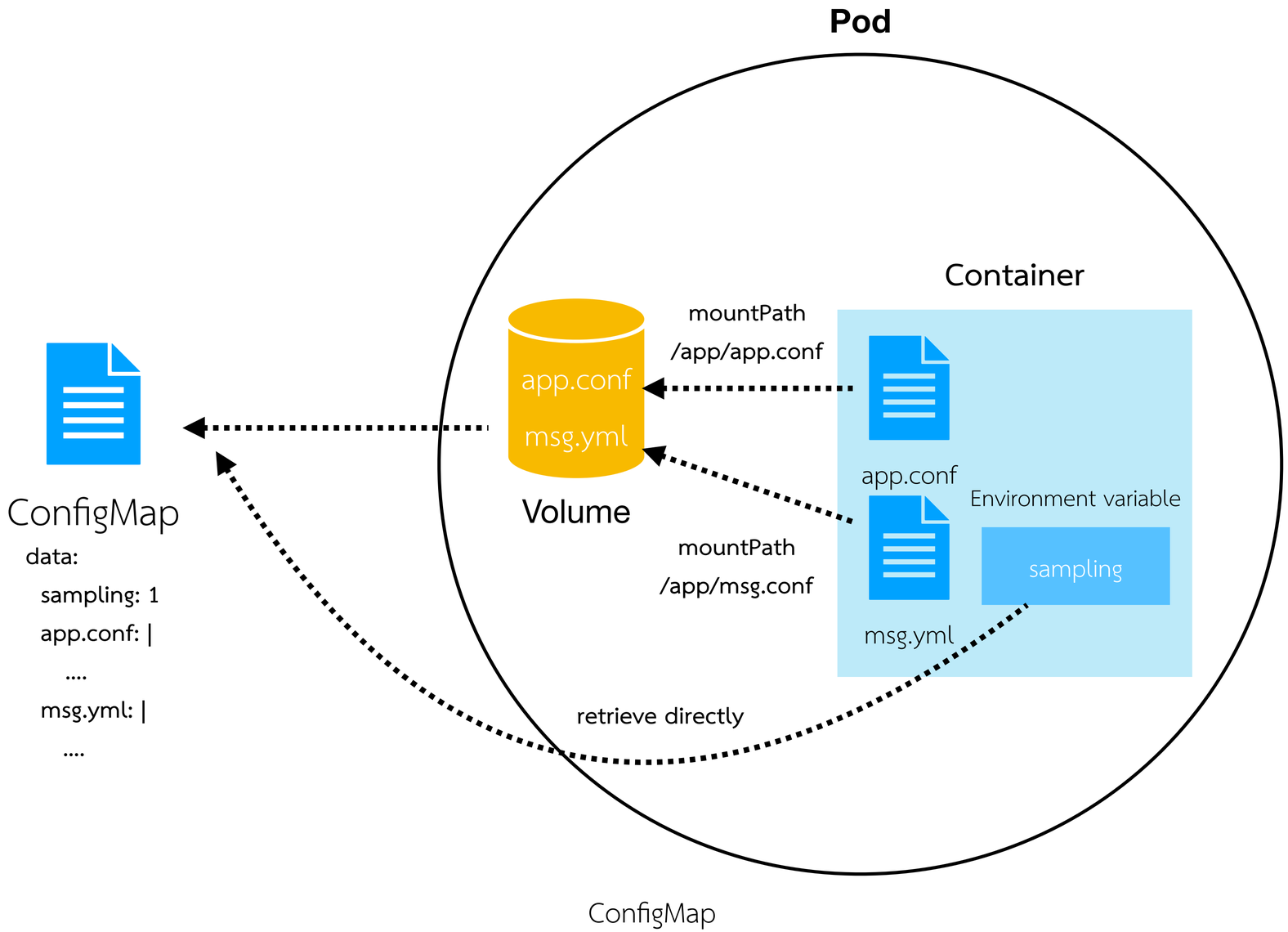
ลักษณะการใช้งานหลัก ๆ มีอยู่ 2 วิธี

1) ใช้ในรูปแบบ Environment variable keys
2) ใช้ในรูปแบบ Config file keys

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # Env keys
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

Kubernetes - ConfigMap and Secret



ConfigMap

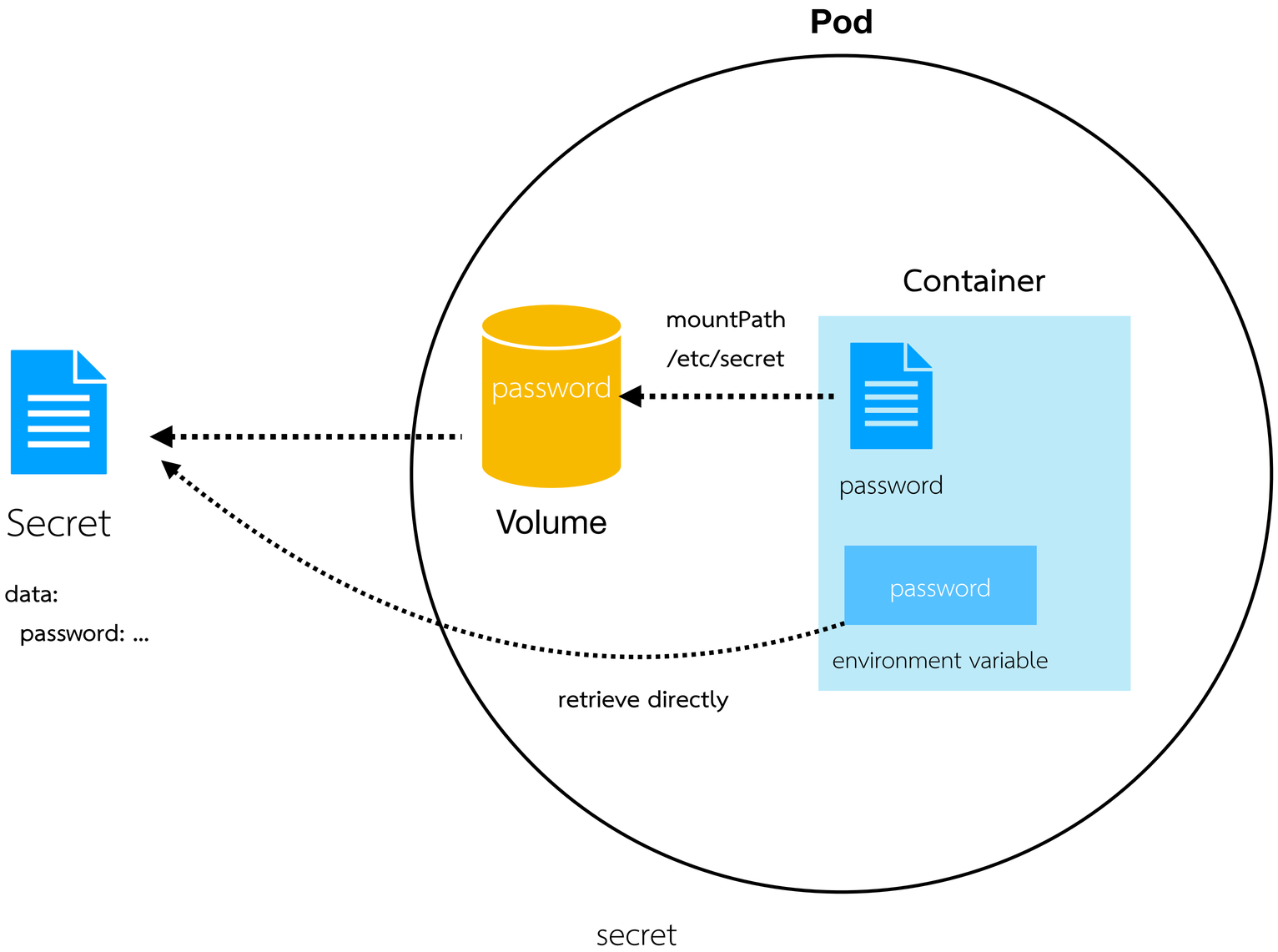
Secret

ความปลอดภัยของข้อมูลเป็นเรื่องสำคัญอันดับต้น ๆ ของการพัฒนา Application การที่ใส่ Username หรือ Password ที่ใช่เชื่อมต่อกับฐานข้อมูลหรืออะไรก็ตาม ที่ Sensitive data ลงใน Config file เป็นเรื่องที่ต้องระวัง K8S จึงแนะนำให้ใช้วิธีการเก็บ data ที่ Sensitive นำไปไว้บน K8S Cluster เมื่อต้องการจะใช้ก็ให้อ้างถึงชื่อของ Secret คล้ายกับวิธี Configmap และโดย default แล้ว secret ไม่ได้มีการ Encrypt แต่อย่างใดซึ่งจำเป็นต้องการทำ RBAC เพื่อป้องกันการเข้าถึงหรือมองเห็น หรือ integration กับ 3rd party เช่น Hashicorp vault

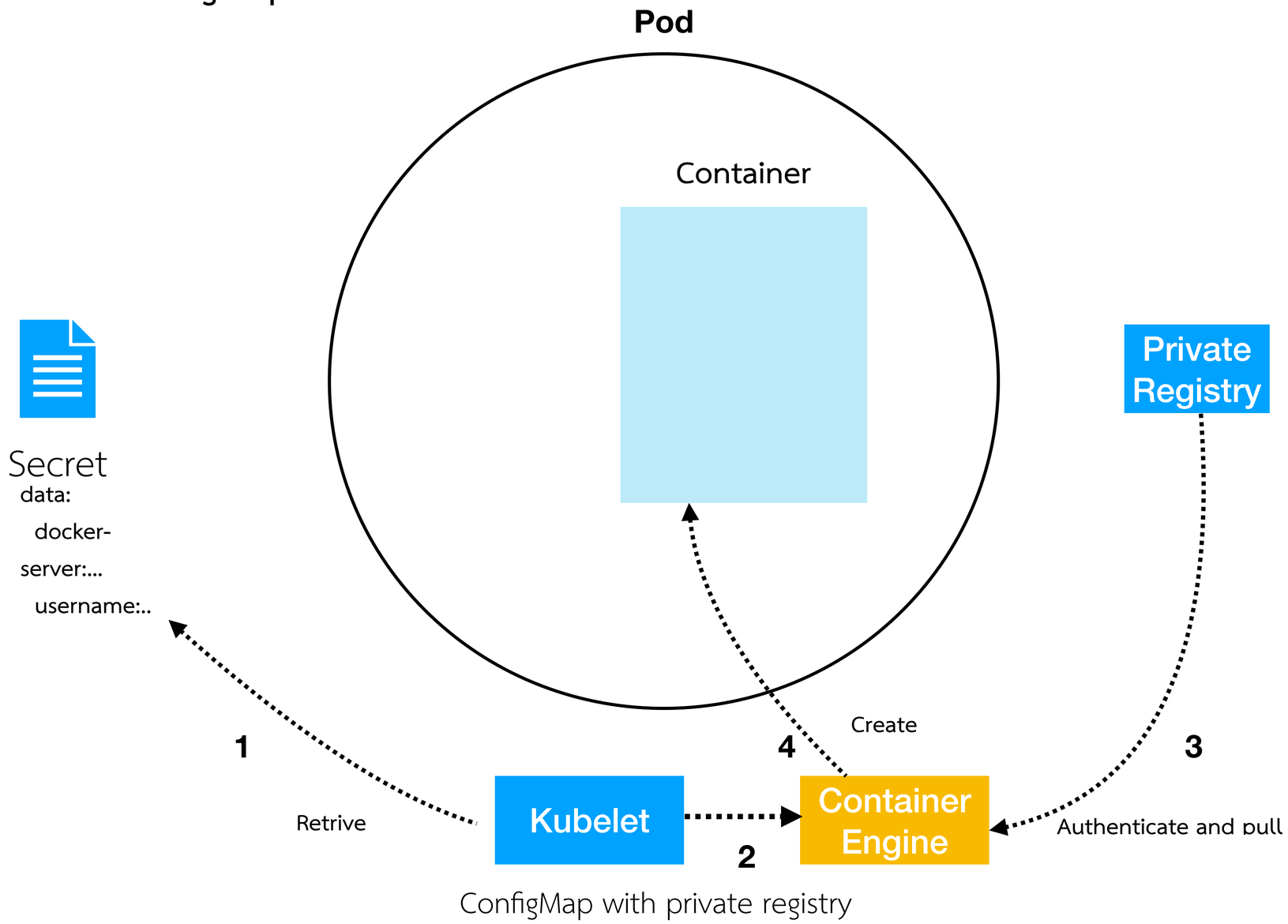
ลักษณะการใช้งานหลัก ๆ มี 3 วิธี

1) ใช้ในรูปแบบ environment variable
2) ใช้ในรูปแบบ file ภายใต้การ volume และ mount เข้ากับ path ใน container
3) นำไปใช้กับ container ที่ใช้ image ที่มาจาก private registry ส่วนการ authenticate, pull image ทาง kubelet จะเป็นคนสั่ง container engine อีกที

Kubernetes - ConfigMap and Secret



Kubernetes - ConfigMap and Secret



ConfigMap with private registry

1) ดึง secret เพื่อนำไปใช้ยืนยันตัวตน
2) ส่งคำสั่งในการสร้าง container
3) ทำการยืนยันตัวตนและดึง image
4) สร้าง container

LAB5 - ConfigMap and Secret

https://github.com/phyze/k8s/tree/main/configmap

https://github.com/phyze/k8s/tree/main/secret

Ingress

การเปิดช่องทางให้ภายนอก Cluster สามารถเรียกใช้งาน Applications ภายในได้มีอยู่สองวิธีคือ NodePort และ LoadBalancer ให้เข้าถึง Applications โดยตรงก็ดูไม่ค่อยแย่เท่าไหร่ในเรื่องการบริหารจัดการเส้นทาง ผู้เรียกใช้งานจากภายนอกก็แค่ระบุ IP กับ Port ให้ตรง Applications ถ้ามี 10 ก็ต้องระบุ IP และ Port ถึง 10 แบบ

ถ้า Applications มากขึ้นถึง 30 แบบนี้เริ่มที่จะจัดการแยกแล้ว IP กับ Port ที่ต้องใช้เต็มไปหมด ดังนั้น K8S จึงเสนอวิธีให้มีตัวกลางในการบริหารจัดการเส้นทางโดยให้มีเส้นทางเดียวเพื่อให้ง่ายต่อการเข้าถึง Applications และตัวกลางนี้จะทำหน้าที่ในการส่งต่อให้กับ Application ที่อยู่ภายในให้เองโดยที่ผู้เรียกใช้งานจอกภายนอกไม่จำเป็นต้องรู้ว่า IP อะไร Port อะไร ซึ่งวิธีที่ใช้แยกว่า request ที่เข้ามาต้องการไปที่ Applications ไหนนั้นใช้วิธีตรวจสอบกฎอยู่ 2 คือ base-host และ base-path เป็นต้น

ตัวกลางที่ทำหน้าที่ในการบริหารจัดการที่รับ Request จากภายนอก เรียกว่า **Ingress Controller** และมีความสามารถในเรื่อง SSL/TLS, LoadBalancer, Healthy Check ส่วน การส่งไปยังเส้นทางที่ถูกต้องและประมวลผลว่าต้องส่งต่อไป Applications ใดนั้นเรียกว่า **Ingress Resource** ที่สามารถตั้งกฎการเส้นทางและเขียนทับเส้นทางใหม่ได้

Ingress Controller

Kubernetes (K8S) เองไม่มี Default Ingress Controller ผู้พัฒนาต้อง Add-on เข้ามาเองไม่ว่าจะติดตั้งผ่าน helm หรือ kubectl ก็ตาม product ในตลาดตอนนี้มีหลายเจ้ามากที่ เช่น Nginx, Avi, Kong, traefik, Istio และ Contour เป็นต้น

K8S Cluster อนุญาตให้สามารถติดตั้ง Ingress Controller ได้หลาย Product ขึ้นอยู่กับว่าผลิตภัทธ์ที่พัฒนาต้องการใช้ Ingress Controller ยี่ห้อไหน ซึ่ง Ingress resource จำเป็นต้องใส่ชื่อของ Ingress Controller ลงไปเพื่อให้ ingress Controller ทำงานตามRule ที่ตั้งไว้ให้ถูกต้อง สิ่งนี้เรียกว่า **IngressClass**

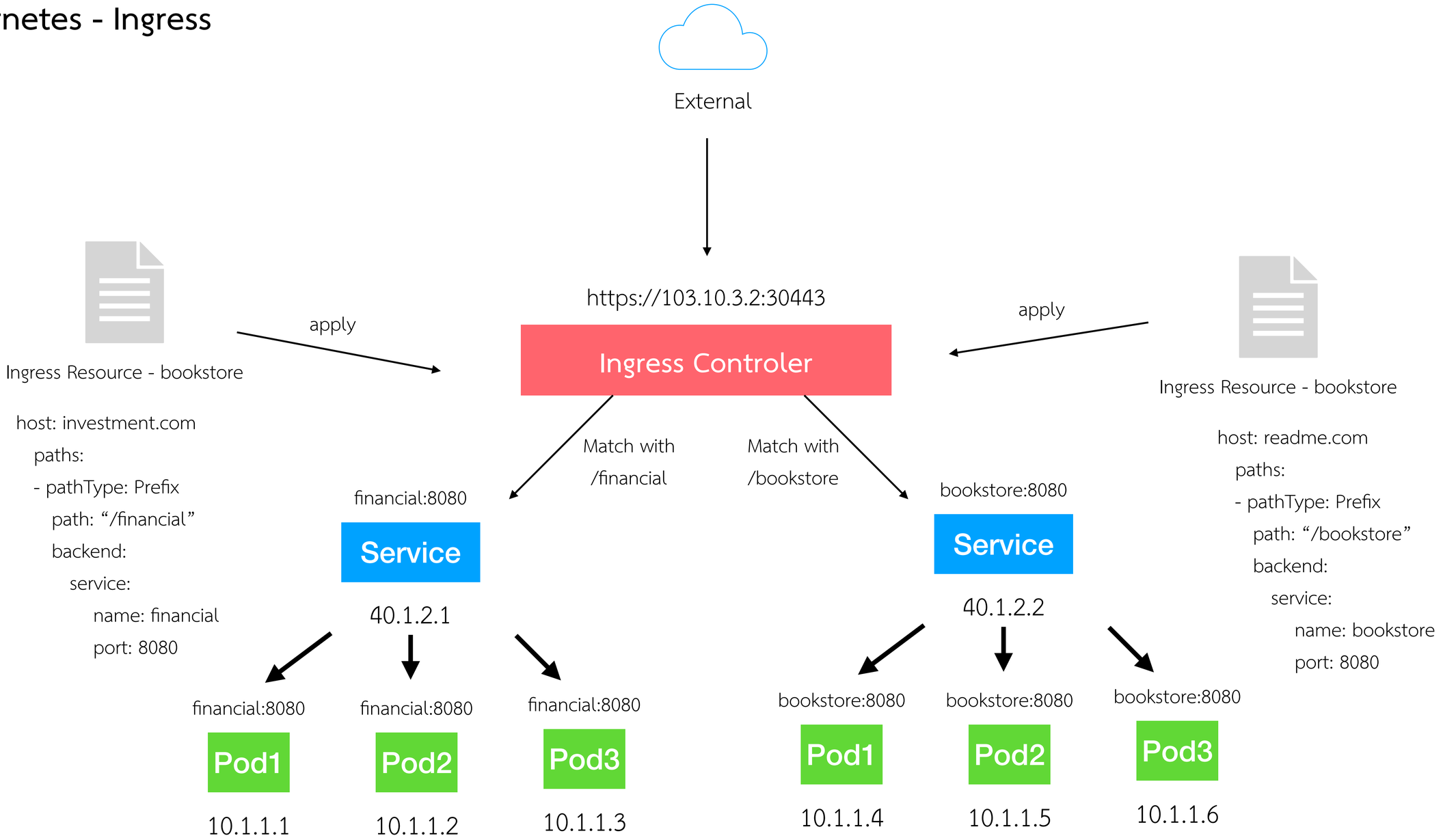
ในกรณที่ K8S ที่เป็น commercial การที่จะ expose Ingress Controller เป็น LoadBalancer เพื่อให้ได้ Public IP นั้นง่ายมาก แต่ถ้าเป็น On-premise ต้องใช้คนละวิธีคือต้องตั้ง LoadBalance ไม่ว่าจะเป็นแบบ standalone หรือ HIgh Availability (HA) มากั้นข้างหน้า K8S Cluster เพื่อกระจาย Traffic สู่ K8S Nodes และทำการ Mapping Public IP กับ IP ของ LoadBalancer

Ingress Resource

คือ config file ที่กำหนดแผนการดำเนินงานของ request ที่เข้ามาสู้ Ingress controller ว่าจะทำยังไงต่อไปและจะต้องส่งต่อไปที่ services ไหน ซึ่งภายใน config file มี parameter ให้กำหนด แบบคร่าว ๆ ดังนี้

- กำหนดให้รองรับ HTTP/HTTPS
- กำหนดให้ใช้ Ingress Controler ยี่ห้ออะไร
- กำหนดให้ส่งต่อ request ไปที่ services อะไรเมื่อ match กับ rules ที่ตั้งไว้
- กำหนดให้แก้ไข header หรือ เพิ่ม header เป็นต้น

Kubernetes - Ingress



LAB6 - Ingress

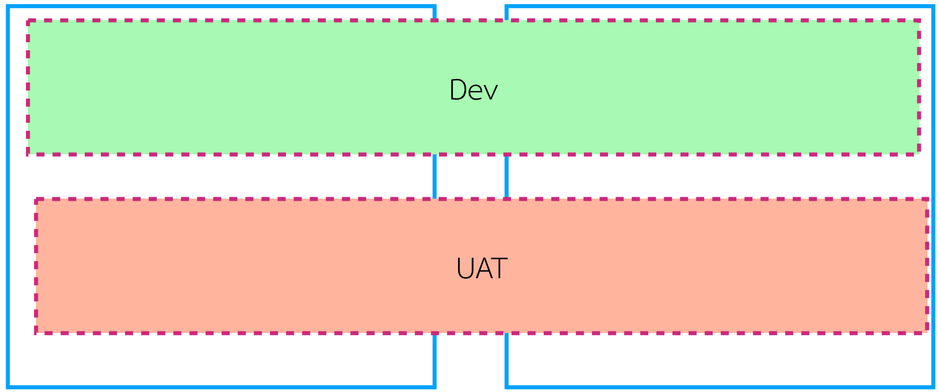
Ref : https://github.com/phyze/k8s/tree/main/ingress

Namespace and Quotas

Kubernetes (K8S) อนุญาตให้ผู้ใช้งานสามารถจัดสรรพื้นที่แบ่งเป็นส่วนเล็ก ๆ สำหรับการใช้งานให้เหมาะสมกับโครงการ นั้นหมายความว่าการพัฒนาโครงการใดสักโครงการไม่จำเป็นต้องใช้ทรัพยากรของ K8S Cluster ทั้งหมดซึ่งสามารถแบ่งเป็นพื้นที่ของแต่ละโครงการหรือ Cluster ต้องมีการแบ่งปันให้กับหลาย ๆ ทีมใช้งานร่วมกันผู้ดูแลสามารถใช้กลวิธีการกำจัดจัดสรรทรัพยากรเพื่อเข้ามาช่วยแบ่งปันพื้นที่ให้แต่ละโครงการหรือทีมได้

การจำกัดทรัพยากรแบบกำหนดระดับความสำคัญและแต่ละความสำคัญจะถูกกำหนด CPU และ Memory ไว้ ร่วมไปถึงจำกัดจำนวน Pod ในแต่ความสำคัญ เรียกว่า Resource Quotas

การจำกัดทรัพยากรแบบกำหนดช่วงการใช้ทรัพยากรโดยเริ่มจาก ต่ำสุด ถึง สูงสุดของ resource ที่ Pod หรือ Container ใช้งาน เรียกว่า LimitRange ซึ่งเป็นกฎบังคับใช้ ถ้าไม่กำหนดการใช้ทรัพยากรให้กับ Pod หรือ Container ทาง validator ของ Kubernetes API จะไม่ให้ดำเนินงานต่อพร้อมแจ้งเหตุผลที่ error



Dev

Class: high
- Pods : สามาใช้ได้ไม่ได้ 10 Pods
- Memory : สามารถใช้ได้ไม่เกิน 100 Gi
- CPU: สามารถใช้ได้ไม่เกิน 20 cores

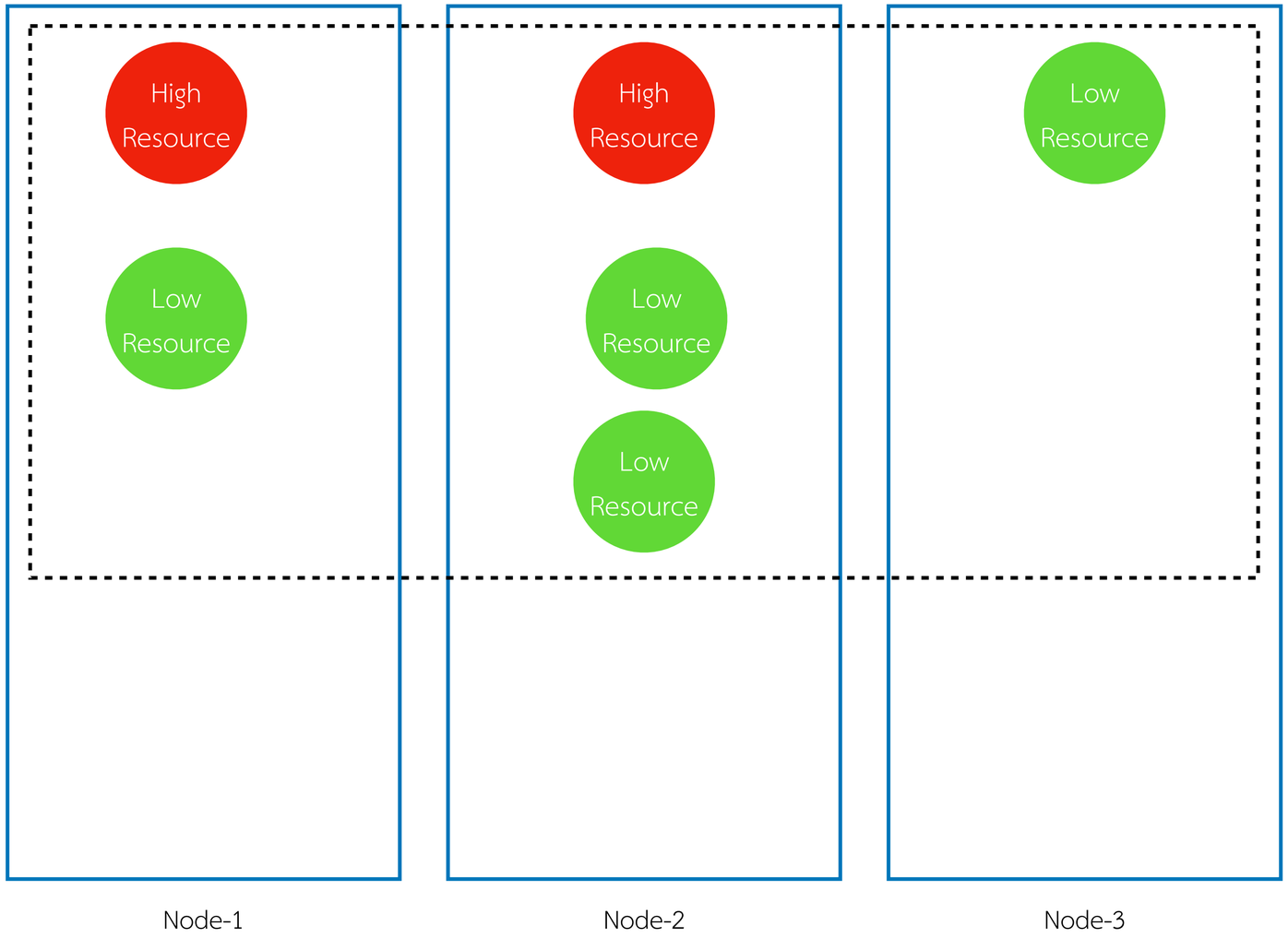
Class: low
- Pods : สามาใช้ได้ไม่ได้ 10 Pods
- Memory : สามารถใช้ได้ไม่เกิน 10 Gi
- CPU: สามารถใช้ได้ไม่เกิน 10 cores

UAT

Class: high
- Pods : สามาใช้ได้ไม่ได้ 10 Pods
- Memory : สามารถใช้ได้ไม่เกิน 500 Gi
- CPU: สามารถใช้ได้ไม่เกิน 50 cores

Class: low
- Pods : สามาใช้ได้ไม่ได้ 10 Pods
- Memory : สามารถใช้ได้ไม่เกิน 100 Gi
- CPU: สามารถใช้ได้ไม่เกิน 20 cores

Kubernetes - Namespace and Quotas



ตัวอย่าง.

Resource Quotas ที่ใช้ 2 ประเภท

High Resource:

สามารถใช้ CPU ได้ไม่เกิน 3 cores
สามารถใช้ Memory ได้ไม่เกิน 3 Gi
สามารถมี Pods ได้ไม่เกิน 2

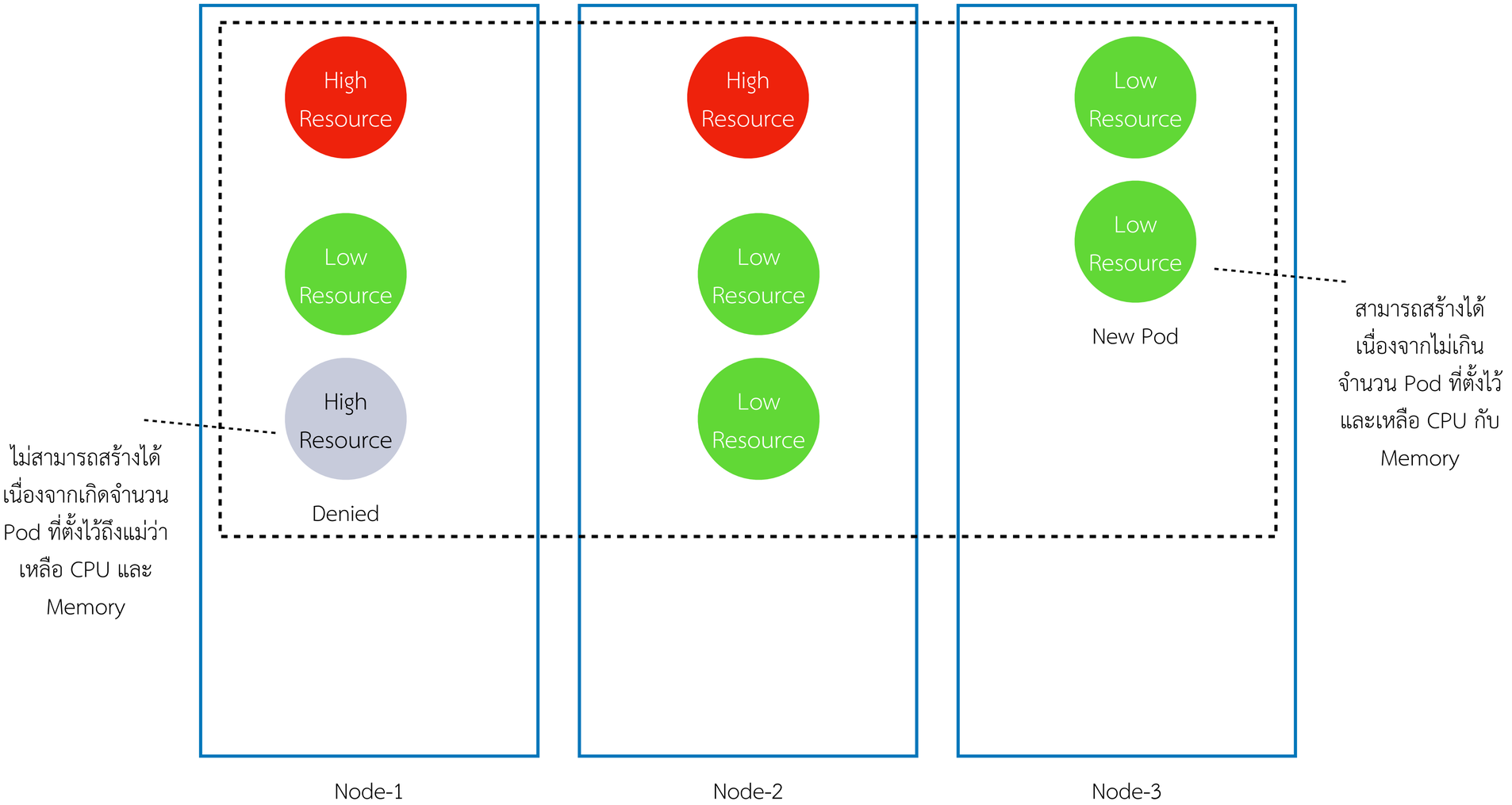
Low Resource:

สามารถใช้ CPU ได้ไม่เกิน 2 cores
สามารถใช้ Memory ได้ไม่เกิน 1 Gi
สามารถมี Pods ได้ไม่เกิน 6

	ขั้นต่ำที่ Pod ต้องการใช้	ใช้ได้ไม่เกิน
	CPU : 100mi	CPU : 400mi
	Memory : 20Mi	Memory : 100Mi

	ขั้นต่ำที่ Pod ต้องการใช้	ใช้ได้ไม่เกิน
	CPU : 400mi	CPU : 1000mi
	Memory : 100Mi	Memory : 1Gi

Kubernetes - Namespace and Quotas



LAB7 - Namespace and Quotas

Ref : https://github.com/phyze/k8s/tree/main/namespace_quota