

NCDR 108-T29

微服務架構探討與建置

Microservice Discussion and Development



國家災害防救科技中心

National Science and Technology Center
for Disaster Reduction

國家災害防救科技中心

中華民國 109 年 01 月

NCDR 108-T29

微服務架構探討與建置

Microservice Discussion and Development

陳俊元



國家災害防救科技中心

中華民國 109 年 01 月

摘要

單體式架構是過去常見的應用程式架構，此架構結構十分簡潔且各項功能可以直接進行構通，對於開發者而言容易上手，但後期維護因服務之間過度緊密，導致更新修改有所困難。相對於單體式架構而言，微服務架構係以單一且功能小的服務為基礎，透過模組化方式組合成複雜的大型應用系統，由於早期建置規劃不易，導致早年少有應用程式採用此架構。近年來隨著虛擬化與容器服務的發展，微服務架構的概念再度被提倡與推廣。本文將探討以微服務架構進行網站系統建置、管理之可行性分析。

關鍵字：微服務、容器

Abstract

Monolithic architecture is a common application architecture in the past. This architecture is very concise and its functions can be directly constructed. In addition, this architecture is easy for developers to getting started, but later maintenance is too difficult due to excessive closeness between services. Relative to monolithic architecture, microservice architecture is based on small services, and combined modules to build a complex large-scale systems. However, due to the difficulty of early implementation planning, few applications adopted this architecture in early years. In recent years, the concept of microservice architecture is promoted and promoted again by the development of virtualization and container services. In this study, we will explore the feasibility analysis of website system construction and management with microservice architecture.

Keyword: Microservice, Container

目錄

摘要.....	1
Abstract.....	2
目錄.....	3
圖目錄.....	5
第一章 簡介.....	6
第二章 相關技術發展.....	7
2.1 虛擬化.....	8
2.2 容器.....	9
2.2.1 Docker.....	9
2.2.2 Kubernetes	10
2.2.3 DevOps	11
第三章 系統架構與環境設置.....	12
3.1 環境需求.....	12
3.2 軟體環境.....	13
3.2.1 系統安裝.....	13
3.2.2 平台建置.....	13
第四章 實驗.....	14
4.1 部署服務.....	14
4.2 成果展示.....	18

第五章 結論與未來工作.....	19
參考文獻.....	20
附件-CentOS 安裝作業.....	21
附件-Kubernetes 安裝作業	24



圖目錄

圖 2.1	DevOps 與傳統方法的比較	11
圖 3.1	系統架構圖	12
圖 3.2	Kubernetes 安裝架構	13
圖 4.1	程式碼管理平台部署情況	17
圖 4.2	圖磚系統之系統畫面	18
圖 6.1	CentOS 7 安裝	21
圖 7.2	Kubernetes Join Token	30
圖 7.3	master01 節點狀態	31
圖 7.4	K8S 集群狀態	36



第一章 簡介

近年來軟硬體架構的發展，使微服務架構的議題開始被討論與應用。本研究探討微服務架構於對外服務系統應用，並建置一簡易系統檢視微服務架構所帶來的優缺點。章節架構安排如下：第二章相關技術發展，說明現行微服務架構常見的相關背景技術的探討；第三章系統架構與環境設置，說明以微服務架構為基礎的系統架構以及建置系統的步驟與方法；第四章實驗，則是以一簡易系統展示本研究成果；第五章為結論與未來工作。



第二章 相關技術發展

本節將簡易的說明與微服務相關技術的發展。在過去單體式架構 (Monolithic Architecture) 廣為許多系統及應用所採用，其優點是直覺、為人所知、受多種開發工具支援，深受工程師所喜愛及推薦。然而隨著開發及維運時間的拉長，其缺點逐一浮現，像是因系統複雜龐大造成建構與部署的時間較長；任何一個小變動，就得須重新部署整個應用系統等缺點。而微服務 (Microservices) 則有別於單體式架構，採用微小服務設計使系統簡單化、容易進行開發、修改及部署。微服務 (Microservices) 一詞首次出現於 2011 年的威尼斯的一場軟體架構工作坊 [2]。隔年 2012 年，Microservices 一詞正式被確認最合適的詞彙；同年 James Lewis 發表第一個微服務架構的實現 [3]。2014 年由 Lewis 及 Fowler 共同提出了微服務的概念 [2]，包含下列幾項特性：

- 微服務是由以單一應用程式構成的小服務；
- 自己擁有自己的行程與輕量化處理；
- 服務依業務功能設計；
- 以全自動的方式部署；
- 與其他服務使用 HTTP API 通訊；
- 服務可以用不同的程式語言與資料庫等元件實作。

簡言之，微服務的概念在於將複雜的系統以多個單一且功能專一的小服務取代，各功能區塊則是以 API 相互進行通訊。儘管此概念可直接應用於一般伺服器上，但效果卻差異不大。隨著虛擬化、容器化、Kubernetes 的興起，得使微服務的概念與應用彰顯其成效。目前像是亞馬遜 (amazon)、Twitter、可口可樂 (Coca Cola)、Netflix、Uber 等企業都採用微服務架構 [1]。

2.1 虛擬化

虛擬化的概念為 1960 年代由 IBM 所提出，此概念是一種將電腦各種實體資源進行抽象化並分割、組合成一個或多個環境供不同用途使用，此項技術解決當時一台大型主機無法提供多人同時使用的問題。隨著 1980 年代起，多工的作業系統以及個人電腦的普及，使得虛擬化的優勢不在，導致發展停滯一段時間。近年來隨著硬體發展迅速導致硬體效能過剩，以及雲端的需求，使得虛擬化的相關技術再度被提倡與推廣。

透過虛擬化的技術可將 CPU、記憶體、網路卡等資源模擬成多個獨立且安全的虛擬電腦，例如在一台具有 16 核心 CPU、64GB 記憶體、1TB 硬碟空間的主機上，透過虛擬作業平台 (Hypervisor) 可虛擬成多個 2 核心、2GB 記憶體、100GB 硬碟空間的虛擬機電腦提供各種不同系統使用。

2.2 容器

容器是一種作業系統層虛擬化技術，此項技術是將應用程式打包成一個軟體容器，其包含軟體本身的執行程式及所需的作業系統核心和函式庫。由於此項技術不需要虛擬作業平台，因此具有輕量化、迅速啟動的特性。Docker、qemu/kvm、chroot、OpenVZ、libvirt-lxc、libvirt-sandbox 等皆為容器技術的實作。

2.2.1 Docker

Docker 是直接運行在作業系統之上的容器技術，透過沙箱機制虛擬出一完整系統。Docker 所虛擬出的容器彼此之間不會有任何接口，可完全隔離容器與作業系統及容器與容器之間的隔離。Docker 包含了三個主要部份：

- Docker 映像檔

Docker 映像檔是一個唯讀的環境模板，此模板包含服務的程式 (包括應用程式、設定檔) 以及作業系統所需的相關函式庫。

- Docker 容器

依照 Docker 映像檔所建立的實例 (instance) 稱為 Docker 容器，一個映像檔可以建立多個容器，每個容器都是隔離不相互影像的。

- Docker 倉庫

Docker 倉庫為存放 Docker 映像檔的倉庫。目前 Docker Hub 為現今最大公開 (Public) 的 Docker 倉庫，供使用者下載各式 Docker 映像檔使用；使用者亦可透過私有的倉庫進行 Docker 映像檔管理與使用。

2.2.2 Kubernetes

Kubernetes(簡稱 K8s) 是一套容器的自動部署、擴展和管理的開源系統，由 Google 設計並捐贈給 Cloud Native Computing Foundation。

基本的 Kubernetes 對象包含：

- Pod

Pod 為 K8s 執行的最小單元，每個 Pod 之間允許執行多個容器。此外 Pods 是有生命週期的。他們可以被創建，而且銷毀不會再啟動。

- Deployment

透過 Deployment 可以動態創建和銷毀 Pod。

- Service

透過 Service 可允許外部使用者訪問 Pod。

- Volume

為 Pod 提供儲存空間使用。

- Namespace 透過 Namespace 可以隔離不同專案、環境或客戶。

2.2.3 DevOps

DevOps 是一種文化、實務及工具結合於一身的概念，相對於傳統，此種做法可提升整體開發速度、交付速度及服務水準的做法。在 DevOps 的框架下，開發與安全團隊及營運團隊會透過各種自動化工具緊密的整合在一起，使團隊具有高度競爭力。圖 2.1 呈現 DevOps 與傳統方法的比較，由圖可了解儘管初期導入 DevOps 成本較高，但具有較短的開發週期，並透過大量的自動化工具可進行源碼的檢測、系統測試、系統部署、資安弱掃及系統監控等工作，使得開發、維運上都獲得不錯的收益。

	傳統	導入DevOps
開發週期	以周或月為開發週期	以分鐘或小時為開發週期
源碼檢測	屬不同部門需跨單位進行 約1~3日	使用工具自動檢測 約1~10分鐘
測試	人工測試 約0.5日	使用工具自動檢測 約1~10分鐘
部署	人工部署 約1~7日	自動部署 如有異常可隨時退回上一版本 約10~30分鐘
資安弱掃	外部弱掃機構進行檢測 約2~10日	內部弱掃工具定期自動檢測 約1~10分鐘
監控	花費大量人力進行監控及處理日誌	由工具收集及分析日誌，過濾出主要異常問題，並依事件自動觸發對應機制，人工僅需處理意外事件
成本	開發、部署、維運成本都很大	前期導入微服務成本較大，後期開發成本下降，部署維運成本亦低於傳統開發

圖 2.1 DevOps 與傳統方法的比較

第三章 系統架構與環境設置

在本研究中，微服務架構係基於 Kubernetes 服務之上，並於將 App 打包成容器進行服務。此外為有效管理 Kubernetes，Kubernetes 係透過虛擬機形式建置於實體主機之上，如圖 3.1 系統架構所示。此系統架構由三台主機所構成，每台主機上透過虛擬機執行多個 K8s，再由 K8s 組成一個叢集 (K8s Cluster)，最後由此叢集進行微服務架構的執行與管理。本研究於台北、台中及 Google 雲端等三地，建構三套上述系統，其詳細建置方式如下列章節說明。



圖 3.1 系統架構圖

3.1 環境需求

每個節點 (Node) 最低環境需求建議採用下列規格：

- 2 核心 CPU

- 16GB 記憶體
- 160GB 硬碟空間

3.2 軟體環境

3.2.1 系統安裝

本研究採用 CentOS 作為基礎環境作業系統，參考安裝方式請參閱附件-CentOS 安裝作業。

3.2.2 平台建置

建構 Kubernetes multi master - multi work node 容器平台，本研究建置 3 個 master 節點及 9 個 worker node 節點，master 節點含 K8S 主要核心之微服務容器，配置如圖 3.2 所示。詳細安裝方式請參閱附件-Kubernetes 安裝作業。

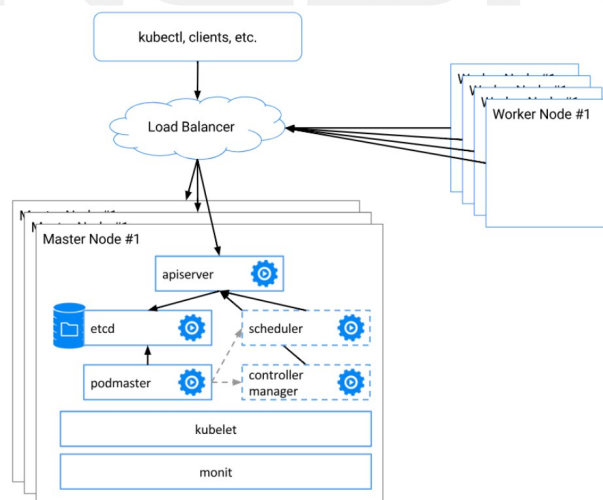


圖 3.2 Kubernetes 安裝架構

第四章 實驗

圖磚服務常用於防災相關的展示系統中，用以展示空間化的地圖資訊或影像資訊，因此本研究透過圖專服務平台展示本研究之微服務平台。本次部署的圖磚系統微服務架構，於台北東七機房、台中文心機房兩地的私有雲以及一地的 Google Cloud Platform(GCP) 雲端伺服器等三地共佈設 9 台伺服器，每個機房可依資源考量(如網路頻寬、公有雲平台費用) 分別設定不同服務部署數量，其中東七機房部屬 4 台伺服器，文心機房部屬 3 台伺服器，GCP 雲端機房部屬 2 台伺服器。

4.1 部署服務

本研究案例原始碼存放於程式碼管理平台上 (<https://git.cbe.tw/website/tw.cbe.test>)，透過程式碼管理平台，進行微服務的部署，部署語法如下顯示。

1. Ingress

負責控制網址導向的部分。

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: tw-cbe-test-ingress
  labels:
    app: tw-cbe-test
spec:
```

```
tls:
- hosts:
  - test.cbe.tw
  secretName: tw-cbe-secret
- hosts:
  - test1.cbe.tw
  secretName: tw-cbe-secret
- hosts:
  - test2.cbe.tw
  secretName: tw-cbe-secret
- hosts:
  - test3.cbe.tw
  secretName: tw-cbe-secret
rules:
- host: test.cbe.tw
  http:
    paths:
      - path: "/"
      backend:
        serviceName: tw-cbe-test-service
        servicePort: 80
- host: test1.cbe.tw
  http:
    paths:
      - path: "/"
      backend:
        serviceName: tw-cbe-test-service
        servicePort: 80
- host: test2.cbe.tw
  http:
    paths:
      - path: "/"
      backend:
        serviceName: tw-cbe-test-service
        servicePort: 80
- host: test3.cbe.tw
  http:
    paths:
      - path: "/"
      backend:
        serviceName: tw-cbe-test-service
        servicePort: 80
```

2. Ingress

負責控制服務導向的部分。

```
---
apiVersion: v1
kind: Service
metadata:
  name: tw-cbe-test-service
  labels:
    app: tw-cbe-test
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  selector:
    app: tw-cbe-test
  type: LoadBalancer
```

3. StatefulSet

負責控制服務建立的部分。

```
---
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: tw-cbe-test
  labels:
    app: tw-cbe-test
spec:
  serviceName: "tw-cbe-test"
  updateStrategy:
    type: RollingUpdate
  replicas: 3
  template:
    metadata:
      labels:
        app: tw-cbe-test
    spec:
      imagePullSecrets:
      - name: registrykey-cbe-tw
```

```

containers:
- name: tw-cbe-test-web
  imagePullPolicy: Always
  image: dkr.cbe.tw/website-test
  ports:
  - containerPort: 80
  env:
  - name: LETSENCRYPT_PROXYPASS_URL
    value: "http://ssl.cbe.tw/.well-known/acme-
      challenge/ connectiontimeout=15 timeout
      =30"
  - name: LETSENCRYPT_PROXYPASSREVERSE_URL
    value: "http://ssl.cbe.tw/.well-known/acme-
      challenge/"
  volumeMounts:
  - name: localtime
    mountPath: /etc/localtime
volumes:
- name: localtime
  hostPath:
    #realpath /etc/localtime
    path: /usr/share/zoneinfo/Asia/Taipei

```

將部署語法上傳至程式碼管理平台後，可於程式碼管理平台上查看部署的情況(如圖 4.1所示)。

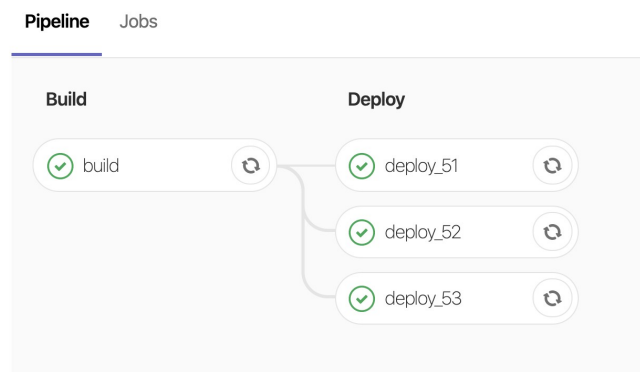


圖 4.1 程式碼管理平台部署情況

4.2 成果展示

圖 4.2 畫面上成功展示不同圖磚，是來自不同機房的服務，其中 Server: X-Y 表示機房 X 內的第 Y 個服務，畫面上 Server: 1-3 則表示此服務來自於東七機房內的第三台伺服器所提供的圖磚服務。若有需修改部署數量，僅需修改部署語法中的 replicas 參數數值，既可動態變更服務的數量。

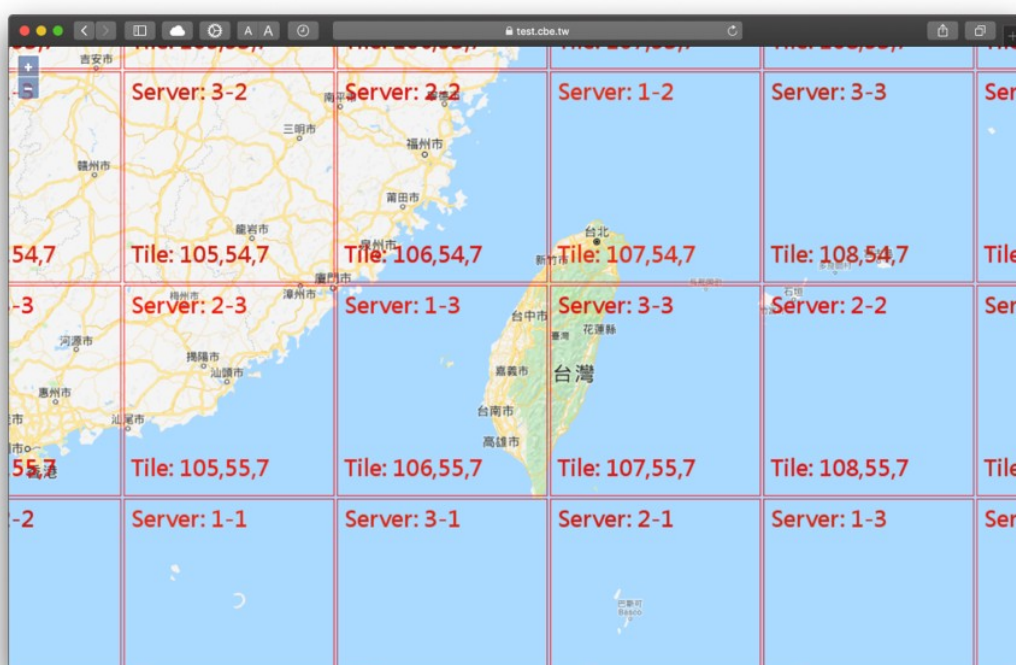


圖 4.2 圖磚系統之系統畫面

第五章 結論與未來工作

本文以 Gitlab 整合了 Docker 以及 Kubernetes 來展現微服務架構及 DevOps 的實例，透過 Docker 封裝 App 應用程式，並透過 GitLab 程式碼管理平台於 Kubernetes 上進行容器的部署。並且僅需透過調整 replicas 參數而不需多做額外複雜的設定，既可動態改變部署數量。透過此案例可以探究微服務架構的便利性，然而仍有許多可研究的議題，例如使用者權限管理、系統日誌、高可用性資料庫的建置與資料同步等等，都是未來需要克服的議題。



參考文獻

- [1] Aleksandra Kwiecień. <https://reurl.cc/A1N10p>, 2019. <https://divante.com/blog/10-companies-that-implemented-the-microservice-architecture-and-paved-the-way-for-others>.
- [2] Fowler M. Lewis, J. Microservices. <https://martinfowler.com/articles/microservices.html>, 2014.
- [3] J. Lewis. Micro Services - Java the Unix way. 33rd Degree Conference for Java Masters, 2012. <http://2012.33degree.org/pdf/JamesLewisMicroServices.pdf>.



附件-CentOS 安裝作業

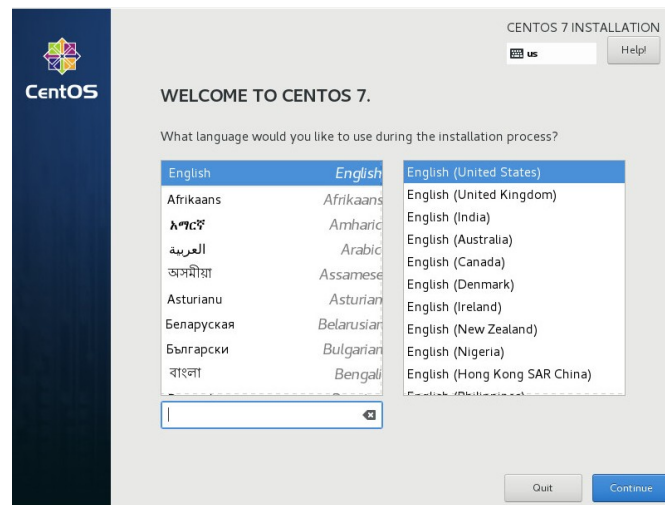


圖 6.1 CentOS 7 安裝

1. 安裝 CentOS 7.5(使用官網 CentOS-7-x86_64-DVD-1804.iso)，安裝畫面如圖 6.1。

2. 更新 CentOS 7.5

```
yum -y update
```

3. 設定 hostname

```
echo node31-master01 > /etc/hostname
```

4. 設定 IP

```
cat <<EOF > /etc/sysconfig/network-scripts/ifcfg-ens160
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
IPADDR=10.31.100.61
NETMASK=255.255.0.0
```

```
GATEWAY=10.31.254.254
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
NAME=ens160
DEVICE=ens160
ONBOOT=yes
ZONE=
DNS1=8.8.8.8
DNS2=8.8.4.4
EOF
```

5. 編輯 /etc/host

```
cat <<EOF >> /etc/hosts
10.31.100.61 node31-master01
10.31.100.62 node31-master02
10.31.100.63 node31-master03
10.31.100.100 node31-vip
10.31.100.101 node31-node01
10.31.100.102 node31-node02
10.31.100.103 node31-node03
10.31.100.104 node31-node04
10.31.100.105 node31-node05
10.31.100.106 node31-node06
10.31.100.107 node31-node07
10.31.100.108 node31-node08
10.31.100.109 node31-node09
EOF
```

6. 編輯 /etc/host

```
cat <<EOF >> /etc/hosts
10.31.100.61 node31-master01
10.31.100.62 node31-master02
10.31.100.63 node31-master03
10.31.100.100 node31-vip
10.31.100.101 node31-node01
10.31.100.102 node31-node02
10.31.100.103 node31-node03
10.31.100.104 node31-node04
10.31.100.105 node31-node05
10.31.100.106 node31-node06
```

```
10.31.100.107 node31-node07
10.31.100.108 node31-node08
10.31.100.109 node31-node09
EOF
```

7. 設置 SELINUX

```
sed -i s/^SELINUX=.*$/SELINUX=disabled/ /etc/selinux/
config
systemctl disable firewalld.service
systemctl disable libvirtd.service
```



附件-Kubernetes 安裝作業

• 前置作業

Kubernetes 安裝的前置步驟如下：

1. Disabling Swap

```
cat <<EOF >> /etc/hosts
swapoff -a
sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

2. Forward Policy

```
iptables -P FORWARD ACCEPT
```

3. Kernel 參數調整

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
vm.swappiness=0
EOF
sysctl -p /etc/sysctl.d/k8s.conf
```

4. 加載 ipvs 模組

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
vm.swappiness=0
EOF
sysctl -p /etc/sysctl.d/k8s.conf
```

5. NTP 設定

```
yum install ntp -y
```

```
ntpdate time.stdtime.gov.tw
hwclock -w
echo "server time.stdtime.gov.tw" >> /etc/ntp.conf
systemctl start ntpd
systemctl enable ntpd
```

6. 安裝 docker-ce

```
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install docker-ce-18.06.0.ce -y
systemctl enable docker && systemctl start docker
```

• K8S 安裝作業

1. 配置 yum 設定檔

```
cat << EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=http://yum.kubernetes.io/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
        https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
yum update -y
```

2. 安裝 K8S 軟體

```
yum install -y kubelet kubeadm kubectl ipvsadm
sed -i '/KUBELET_EXTRA_ARGS=/s//KUBELET_EXTRA_ARGS=--fail-swap-on=false/g' /etc/sysconfig/kubelet
systemctl enable kubelet.service
```

• 建立 K8S Cluster 以東七機房為例。

1. 設定 ssh trust

```
ssh-keygen -t rsa
cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
## 編輯所有節點之 public key 至所有節點 authorized_keys
   檔
scp authorized_keys node31-master02:~/.ssh/
   authorized_keys
ssh node31-master02 ls
```

2. 準備 HAProxy 設定檔

(在所有 master 節點進行下列設定)

```
mkdir -p /etc/haproxy
cat <<EOF > /etc/haproxy/haproxy.cfg
global
    log 127.0.0.1 local0
    log 127.0.0.1 local1 notice
    tune.ssl.default-dh-param 2048

defaults
    log global
    mode http
    option dontlognull
    timeout connect 5000ms
    timeout client 600000ms
    timeout server 600000ms

listen stats
    bind :9090
    mode http
    balance
    stats uri /haproxy_stats
    stats auth admin:admin123
    stats admin if TRUE

frontend kube-apiserver-https
    mode tcp
    bind :8443
    default_backend kube-apiserver-backend

backend kube-apiserver-backend
```

```
mode tcp
balance roundrobin
stick-table type ip size 200k expire 30m
stick on src
server node31-master01 10.31.100.61:6443 check
server node31-master02 10.31.100.62:6443 check
server node31-master03 10.31.100.63:6443 check
EOF
```

3. 建立 HAProxy

(在所有 master 節點進行下列設定)

```
mkdir -p /etc/kubernetes/manifests
cat <<EOF > /etc/kubernetes/manifests/haproxy.yaml
kind: Pod
apiVersion: v1
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  labels:
    component: haproxy
    tier: control-plane
    name: kube-haproxy
    namespace: kube-system
spec:
  hostNetwork: true
  priorityClassName: system-cluster-critical
  containers:
  - name: kube-haproxy
    image: docker.io/haproxy:1.7-alpine
    resources:
      requests:
        cpu: 100m
    volumeMounts:
    - name: haproxy-cfg
      readOnly: true
      mountPath: /usr/local/etc/haproxy/haproxy.cfg
    volumes:
    - name: haproxy-cfg
      hostPath:
        path: /etc/haproxy/haproxy.cfg
        type: FileOrCreate
```



```
EOF
```

4. 準備 Keepalived 設定檔

```
cat <<EOF > /etc/kubernetes/manifests/keepalived.yaml
kind: Pod
apiVersion: v1
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  labels:
    component: keepalived
    tier: control-plane
    name: kube-keepalived
    namespace: kube-system
spec:
  hostNetwork: true
  priorityClassName: system-cluster-critical
  containers:
  - name: kube-keepalived
    image: docker.io/osixia/keepalived:1.4.5
    env:
    - name: KEEPALIVED_VIRTUAL_IPS
      value: 10.31.100.100
    - name: KEEPALIVED_INTERFACE
      value: ens160
    - name: KEEPALIVED_UNICAST_PEERS
      value: "#PYTHON2BASH:['10.31.100.61',
        '10.31.100.62', '10.31.100.63']"
    - name: KEEPALIVED_PASSWORD
      value: d0cker
    - name: KEEPALIVED_PRIORITY
      value: "100"
    - name: KEEPALIVED_ROUTER_ID
      value: "51"
  resources:
    requests:
      cpu: 100m
  securityContext:
    privileged: true
    capabilities:
      add:
      - NET_ADMIN
```

EOF

5. Kubernetes Control Plane

透過 kubeadm 來建立 control plane 元件

- 準備 kubeadm-config

(在 master01 節點進行下列設定)

```
cat <<EOF > /etc/kubernetes/manifests/kubeadm-
  config.yaml
apiVersion: kubeadm.k8s.io/v1alpha2
kind: MasterConfiguration
kubernetesVersion: v1.12.2
apiServerCertSANS:
- "10.31.100.100"
api:
  controlPlaneEndpoint: "10.31.100.100:8443"
etcd:
  local:
  extraArgs:
    listen-client-urls: "https://127.0.0.1:2379,
      https://10.31.100.61:2379"
    advertise-client-urls: "https
      ://10.31.100.61:2379"
    listen-peer-urls: "https
      ://10.31.100.61:2380"
    initial-advertise-peer-urls: "https
      ://10.31.100.61:2380"
    initial-cluster: "node31-master01=https
      ://10.31.100.61:2380"
  serverCertSANS:
    - node31-master01
    - 10.31.100.61
  peerCertSANS:
    - node31-master01
    - 10.31.100.61

controllerManagerExtraArgs:
  node-monitor-grace-period: 10s
  pod-eviction-timeout: 10s
```

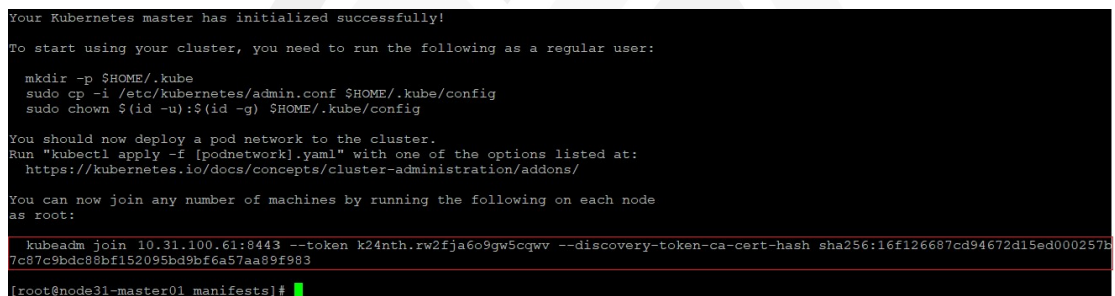
```
networking:
  podSubnet: "10.244.0.0/16"
EOF
```

- 套用 kubeadm-config

(在 master01 節點進行下列設定)。

約需 1 分鐘，完成後會出現 Kubernetes Join Token，如圖 7.2。

```
cd /etc/kubernetes/manifests
kubeadm init --config kubeadm-config.yaml
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/
config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```



```
Your Kubernetes master has initialized successfully!
To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

kubeadm join 10.31.100.61:8443 --token k24nth.rw2fja6o9gw5cqvw --discovery-token-ca-cert-hash sha256:16f126687cd94672d15ed000257b7c87c9bdc88bf152095bd9bf6a57aa89f983

[root@node31-master01 manifests]#
```

圖 7.2 Kubernetes Join Token。安裝完第一台 Kubernetes 後出現的 Kubernetes Join Token。

- 確認 master01 節點狀態 (master01 節點)，如圖 7.4。

```
kubectl get node
kubectl get pod -n kube-system
```

6. 同步 CA 與 Certs 至其它 master 節點上以供使用。

```

[root@node31-master01 manifests]# kubectl get node
NAME                STATUS    ROLES    AGE     VERSION
node31-master01    NotReady master   9m59s   v1.12.2
[root@node31-master01 manifests]# kubectl -n kube-system get pod
NAME                                READY   STATUS              RESTARTS   AGE
coredns-576cbf47c7-bz7nr           0/1    ContainerCreating   0           11m
coredns-576cbf47c7-x7nbn           0/1    ContainerCreating   0           11m
etcd-node31-master01               1/1    Running              0           11m
kube-apiserver-node31-master01     1/1    Running              0           11m
kube-controller-manager-node31-master01  1/1    Running              0           11m
kube-haproxy-node31-master01       1/1    Running              0           11m
kube-keepalived-node31-master01     1/1    Running              0           11m
kube-proxy-729q9                   1/1    Running              0           11m
kube-scheduler-node31-master01     1/1    Running              0           11m
[root@node31-master01 manifests]#

```

圖 7.3 master01 節點狀態。

```

export DIR=/etc/kubernetes/
for NODE in node31-master02 node31-master03; do
  echo "----- ${NODE} -----"
  ssh ${NODE} "mkdir -p ${DIR}/pki/etcd"
  scp ${DIR}/pki/ca.crt ${NODE}:${DIR}/pki/ca.crt
  scp ${DIR}/pki/ca.key ${NODE}:${DIR}/pki/ca.key
  scp ${DIR}/pki/sa.key ${NODE}:${DIR}/pki/sa.key
  scp ${DIR}/pki/sa.pub ${NODE}:${DIR}/pki/sa.pub
  scp ${DIR}/pki/front-proxy-ca.crt ${NODE}:${DIR}/
  pki/front-proxy-ca.crt
  scp ${DIR}/pki/front-proxy-ca.key ${NODE}:${DIR}/
  pki/front-proxy-ca.key
  scp ${DIR}/pki/etcd/ca.crt ${NODE}:${DIR}/pki/etcd
  /ca.crt
  scp ${DIR}/pki/etcd/ca.key ${NODE}:${DIR}/pki/etcd
  /ca.key
  scp ${DIR}/admin.conf ${NODE}:${DIR}/admin.conf
done

```

7. 增加 master2 至 K8S 集群。

在 master02 節點進行下列設定：

```

cat <<EOF > /etc/kubernetes/manifests/kubeadm-config.
yaml
apiVersion: kubeadm.k8s.io/v1alpha2
kind: MasterConfiguration
kubernetesVersion: v1.12.2
apiServerCertSANS:

```

```

- "10.31.100.100"
api:
  controlPlaneEndpoint: "10.31.100.100:8443"
etcd:
  local:
  extraArgs:
    listen-client-urls: "https://127.0.0.1:2379,
      https://10.31.100.62:2379"
    advertise-client-urls: "https
      ://10.31.100.62:2379"
    listen-peer-urls: "https://10.31.100.62:2380"
    initial-advertise-peer-urls: "https
      ://10.31.100.62:2380"
    initial-cluster: "node31-master01=https
      ://10.31.100.61:2380,node31-master02=https
      ://10.31.100.62:2380"
    initial-cluster-state: existing
  serverCertSANS:
    - node31-master02
    - 10.31.100.62
  peerCertSANS:
    - node31-master02
    - 10.31.100.62

controllerManagerExtraArgs:
  node-monitor-grace-period: 10s
  pod-eviction-timeout: 10s

networking:
  podSubnet: "10.244.0.0/16"
EOF
cd /etc/kubernetes/manifests/
kubeadm alpha phase certs all --config kubeadm-config.
  yaml
kubeadm alpha phase kubelet config write-to-disk --
  config kubeadm-config.yaml
kubeadm alpha phase kubelet write-env-file --config
  kubeadm-config.yaml
ubeadm alpha phase kubeconfig kubelet --config kubeadm
  -config.yaml
systemctl restart kubelet

export CP0_IP=10.31.100.61
export CP0_HOSTNAME=node31-master01

```

```

export CP1_IP=10.31.100.62
export CP1_HOSTNAME=node31-master02
kubeadm alpha phase etcd local --config kubeadm-config
  .yaml #可觀察 pod 變化
export KUBECONFIG=/etc/kubernetes/admin.conf

#此動作會造成 etcd cluster 變成 unavailable, 等待約20秒
  , 當新節點 join 成功會自動恢復正常
kubectl exec -n kube-system etcd-${CP0_HOSTNAME} --
  etcdctl --ca-file /etc/kubernetes/pki/etcd/ca.crt
  --cert-file /etc/kubernetes/pki/etcd/peer.crt --key
  -file /etc/kubernetes/pki/etcd/peer.key --endpoints
  =https://${CP0_IP}:2379 member add ${CP1_HOSTNAME}
  https://${CP1_IP}:2380

kubeadm alpha phase kubeconfig all --config kubeadm-
  config.yaml
kubeadm alpha phase controlplane all --config kubeadm-
  config.yaml
kubeadm alpha phase kubelet config annotate-cri --
  config kubeadm-config.yaml
kubeadm alpha phase mark-master --config kubeadm-
  config.yaml

mkdir -p $HOME/.kube
cp -rp /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config

```

8. 增加 master3 至 K8S 集群。

在 master02 節點進行下列設定：

```

cat <<EOF > /etc/kubernetes/manifests/kubeadm-config.
  yaml
apiVersion: kubeadm.k8s.io/v1alpha2
kind: MasterConfiguration
kubernetesVersion: v1.12.2
apiServerCertSANs:
- "10.31.100.100"
api:
  controlPlaneEndpoint: "10.31.100.100:8443"
etcd:
  local:

```

```

extraArgs:
  listen-client-urls: "https://127.0.0.1:2379,
    https://10.31.100.63:2379"
  advertise-client-urls: "https
    ://10.31.100.63:2379"
  listen-peer-urls: "https://10.31.100.63:2380"
  initial-advertise-peer-urls: "https
    ://10.31.100.63:2380"
  initial-cluster: "node31-master01=https
    ://10.31.100.61:2380,node31-master02=https
    ://10.31.100.62:2380,node31-master03=https
    ://10.31.100.63:2380"
  initial-cluster-state: existing
serverCertSANS:
  - node31-master03
  - 10.31.100.63
peerCertSANS:
  - node31-master03
  - 10.31.100.63

controllerManagerExtraArgs:
  node-monitor-grace-period: 10s
  pod-eviction-timeout: 10s

networking:
  podSubnet: "10.244.0.0/16"
EOF
cd /etc/kubernetes/manifests/
kubeadm alpha phase certs all --config kubeadm-config.
  yaml
kubeadm alpha phase kubelet config write-to-disk --
  config kubeadm-config.yaml
kubeadm alpha phase kubelet write-env-file --config
  kubeadm-config.yaml
kubeadm alpha phase kubeconfig kubelet --config
  kubeadm-config.yaml
systemctl restart kubelet

export CP0_IP=10.31.100.61
export CP0_HOSTNAME=node31-master01
export CP2_IP=10.31.100.63
export CP2_HOSTNAME=node31-master03
kubeadm alpha phase etcd local --config kubeadm-config
  .yaml #可觀察 pod 變化

```

```
export KUBECONFIG=/etc/kubernetes/admin.conf

#此動作會造成 etcd cluster 變成 unavailable, 等待約20秒
, 當新節點 join 成功會自動恢復正常
kubectl exec -n kube-system etcd-${CP0_HOSTNAME} --
  etcdctl --ca-file /etc/kubernetes/pki/etcd/ca.crt
  --cert-file /etc/kubernetes/pki/etcd/peer.crt --key
  -file /etc/kubernetes/pki/etcd/peer.key --endpoints
  =https://${CP0_IP}:2379 member add ${CP2_HOSTNAME}
  https://${CP2_IP}:2380

kubeadm alpha phase kubeconfig all --config kubeadm-
  config.yaml
kubeadm alpha phase controlplane all --config kubeadm-
  config.yaml
kubeadm alpha phase kubelet config annotate-cri --
  config kubeadm-config.yaml
kubeadm alpha phase mark-master --config kubeadm-
  config.yaml

mkdir -p $HOME/.kube
cp -rp /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

9. 確認 K8S 集群狀態

在 master 任一節點進行下列設定：

```
kubectl get nodes
kubectl get pod -n kube-system
```



```
[root@node31-master03 manifests]# kubectl get nodes
NAME                STATUS    ROLES    AGE     VERSION
node31-master01    NotReady  master   31m     v1.12.2
node31-master02    NotReady  master   26m     v1.12.2
node31-master03    NotReady  master   7m23s   v1.12.2
[root@node31-master03 manifests]# kubectl get pod -n kube-system
NAME                                READY    STATUS              RESTARTS   AGE
coredns-576cbf47c7-94jw7            0/1     ContainerCreating   0           30m
coredns-576cbf47c7-zl5d9            0/1     ContainerCreating   0           30m
etcd-node31-master01                1/1     Running              0           30m
etcd-node31-master02                1/1     Running              4           19m
etcd-node31-master03                1/1     Running              6           6m43s
kube-apiserver-node31-master01       1/1     Running              0           30m
kube-apiserver-node31-master02       1/1     Running              0           16m
kube-apiserver-node31-master03       1/1     Running              0           3m37s
kube-controller-manager-node31-master01 1/1     Running              1           30m
kube-controller-manager-node31-master02 1/1     Running              0           16m
kube-controller-manager-node31-master03 1/1     Running              0           3m37s
kube-haproxy-node31-master01         1/1     Running              0           29m
kube-haproxy-node31-master02         1/1     Running              0           26m
kube-haproxy-node31-master03         1/1     Running              0           7m30s
kube-keepalived-node31-master01      1/1     Running              0           30m
kube-keepalived-node31-master02      1/1     Running              0           26m
kube-keepalived-node31-master03      1/1     Running              0           7m30s
kube-proxy-5b7fw                     1/1     Running              0           7m30s
kube-proxy-8l98v                     1/1     Running              0           26m
kube-proxy-bwrfd                     1/1     Running              0           30m
kube-scheduler-node31-master01        1/1     Running              1           30m
kube-scheduler-node31-master02        1/1     Running              0           16m
kube-scheduler-node31-master03        1/1     Running              0           3m37s
[root@node31-master03 manifests]#
```

圖 7.4 K8S 集群狀態

微服務架構探討與建置

發行人：陳宏宇

出版機關：國家災害防救科技中心

地址：新北市新店區北新路三段 200 號 9 樓

電話：02-8195-8600

報告完成日期：中華民國 108 年 12 月

出版年月：中華民國 109 年 01 月

版 次：第一版

非賣品



地址：23143新北市新店區北新路三段200號9樓

電話：++886-2-8195-8600

傳真：++886-2-8912-7766

網址：<http://www.ncdr.nat.gov.tw>