國立暨南國際大學資訊工程學系

碩士論文

採用 Kubernetes 架構的 O-RAN 雲基礎設施管理服務 (IMS) 介面的設計和實現

The Design and Implementation of Open RAN Cloud Infrastructure Management Services Interfaces with Kubernetes Architecture

指導教授:吳坤熹博士

研究生:周以恆

中華民國 111 年 8 月

doi:10.6837/ncnu202200101

國立暨南國際大學碩士論文考試審定書

資訊工程 學系（研究所）
研究生 周以恆 所提之論文

The Design and Implementation of Open RAN Cloud Infrastructure Management Services Interfaces with Kubernetes Architecture
採用 Kubernetes 架構的 O-RAN 雲基礎設施管理服務（IMS）介面的設計和實現

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

劉一宇 委員兼召集人

許名宏 委員

吳坤熹 委員

張碩杰 委員

中華民國 111 年 7 月 25 日

doi:10.6837/ncnu202200101

致謝詞

首先，我一定要感謝從大學三年級開始一直到碩士班，都指導、幫助我的 吳坤熹 老師，在這一路上老師給我的幫助數不勝數，也是在老師的鼓勵下，讓我下定決心要繼續讀研究所，可以說這一切的成就要是少了老師，都無法達成，甚至不會去嘗試。在就學期間遇到的無數障礙，也都是老師在背後不斷的督促和幫助之下，一個個渡過。每當在遇到挫折受到打擊後，也是有老師的打氣，讓我恢復信心繼續前進努力下去，最終才能準時完成了五年一貫。

接著，在實驗室中的訓練，老師不僅是教會我們要怎麼去做研究，更是教會我們如何做人以及做事，像是引導著我自己去找到答案，而不是一昧的等著教科書或是別人來告訴你答案。這些技能也都在我有幸到企業實習時，一一在遇到各種問題時成功派上用場，曾經的教誨和督促都在這一刻應驗。而我相信這些技巧，也會在未來的職場生涯中發揮它們的作用，幫助我度過更多的難關。也希望老師能夠帶著這股教學的熱情，繼續幫助更多的學弟妹完成人生中的各個關卡。

此外，也要感謝我的家人，無論我做出各種決定，從來都是尊重我。對於我的決定都是鼓勵，在背後默默支持我，在身心俱疲時回家，也總是能將我的能量充滿，回來繼續面對眼前的課題。

最後，我想要勉勵自己，別忘記在大學這期間的所學，帶著積累下來的技能和勇氣在未來繼續努力。或許未來再也沒有人能在後面逼著自己往前，但每當想要停止進步，就要回頭望向過去這段期間努力的自己，以及在一路上幫助自己的貴人們，帶著大家給予自己的期許，往更前一步去邁進。

doi:10.6837/ncnu202200101

論文名稱：採用 Kubernetes 架構的 O-RAN 雲基礎設施管理服務(IMS) 介面的設計和實現

校院系：國立暨南國際大學科技學院資訊工程學系　　頁數：40

畢業時間：中華民國 111 年 8 月　　學位別：碩士

研究生：周以恆　　指導教授：吳坤熹博士

摘要

隨著 5G 通訊服務的發展，目前電信營運商遇到的問題之一是眾多數量的基地台導致佈建成本過高。基地台的技術以往都掌握在電信設備的製造廠商手上，而 O-RAN 這個組織想要將其中的介面都標準化，依此讓基地台的每個元件由不同廠商來提供，甚至將許多功能，例如: RAN Intelligence Controller(RIC), O-CU, O-DU, 以 Virtualized Network Function(VNF)或是 Containerized Network Function(CNF)的形式來提供，並提出 O-Cloud 的概念將 CNFs 運行在雲端平台上。而其中 O-Cloud Infrastructure Management Services (IMS) 的角色尤其重要，它負責控管整個 O-Cloud 上的資源，並且向最上層的 Service Management and Orchestration Framework (SMO) 來提供 API 服務。IMS 和 SMO 目前已有部分標準與實作釋出，不過許多流程細節和實作工具都未定義。目前也仍未有完整的 O-Cloud 及 O2 介面 (O2 Interface) 實作，本研究因此著重於 SMO 與 O-Cloud 兩者之間的介面 O2 Interface 設計與實作，提供一個將此介面功能實作的方法，以及一個在 O-RAN 中完整的測試平台。

關鍵字: O2 介面，Open RAN，基礎設施管理服務

doi:10.6837/ncnu202200101

Title of Thesis: The Design and Implementation of Open RAN Cloud Infrastructure Management Services Interfaces with Kubernetes Architecture
Name of Institute: Department of Computer Science and Information Engineering, College of Science and Technology, National Chi Nan University
Pages: 40
Graduation Time: 08/2022 Degree: Master
Student Name: Yi-Heng Zhou Advisor Name: Dr. Quincy Wu

Abstract

With rapid growing of 5G wireless communication services, one of the challenges is the soaring cost of base station deployment. To reduce dependency on a single vendor for radio access network (RAN) technologies, the goal of the O-RAN organization is to standardize all the interfaces, so that each element of the base station can be provided by independent manufacturers. Furthermore, each Network Function, such as RAN Intelligence Controller (RIC), O-CU, O-DU, can be provided as a Virtualized Network Function (VNF) or Containerized Network Function (CNF). Therefore, O-RAN proposed the concept of O-Cloud to run CNFs on cloud platforms. Among them, the role of O-Cloud Infrastructure Management Services (IMS) is particularly important. It is responsible for controlling resources on the entire O-Cloud and providing API services to the uppermost Service Management and Orchestration Framework (SMO). So far, some standards and implementations have been released, but many details and tools for implementation have not been defined. This thesis focuses on the O2 Interface between the SMO and O-Cloud, providing an elegant solution for design and implementation, and demonstrating a complete platform for testing O-RAN functions.

Keywords: O2 Interface, Open RAN, Infrastructure Management Services

doi:10.6837/ncnu202200101

目次

doi:10.6837/ncnu202200101

doi:10.6837/ncnu202200101

表目次

doi:10.6837/ncnu202200101

圖目次

doi:10.6837/ncnu202200101

doi:10.6837/ncnu202200101

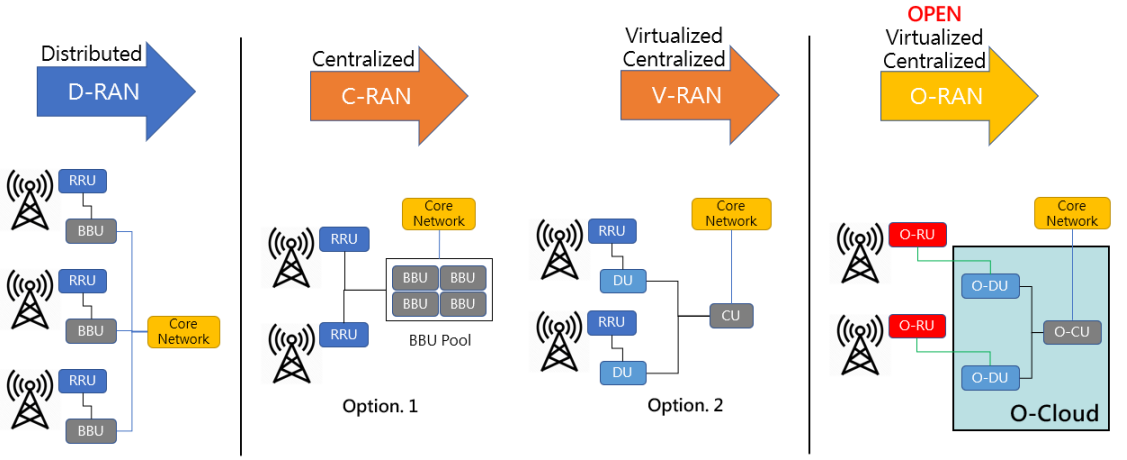
第一章　緒論

第一節 研究動機

資訊服務逐漸佔據我們的日常生活，行動網路的需求也因此快速成長，根據愛立信在 2022 年公告的數據[1]，我們就可以看見全球行動網路數據在 2022 年第一季已經達到了 93 EB (ExaByte, 10^{18} Byte)，相較去年的第一季足足成長了 40%。於此同時，在報告中也特別指出長期以來的數據成長原因，主要是源於訂閱用戶的增加。其中 5G 服務的訂閱用戶成長速度高於以往 4G 問世時的成長速度，預期在 2027 年能夠超過 44 億的訂閱用戶。5G 的系統提出了三大主要服務: Enhanced Mobile Broadband (eMBB)、Ultra-reliable and Low Latency Communications (URLLC)、Massive Machine Type Communications (mMTC)，這三項服務分別規範了行動網路的多樣且高傳輸範圍、大頻寬、低延遲以及更多數量的連網裝置，這些高度要求的服務皆是為了能滿足未來更多的需求。

於此同時，5G 的這些需求導致基地台的容量與密度更高，也就推動了無線電存取網路 (Radio Access Network, RAN) 技術的進步，網路運營商在搭建更多的基地台，涵蓋更多範圍增加用戶以及提供更快速的網路最佳化服務時，降低搭建 (Capital Expenditure, CAPEX) 或是維運 (Operating Expense, OPEX) 基地台的成本也是十分重要的考量。過去，基地台的設備技術皆被掌握在各家設備製造商手上，當中各個元件之間不開放的介面也導致了設備只能向同一家設備製造商購買的綁定狀況，採購費用也因此居高不下。

doi:10.6837/ncnu202200101



圖一 RAN 技術的演進過程

RAN 技術的演進，如圖一 RAN 技術的演進過程圖一所示，可以從 Distributed RAN (D-RAN) 這個最簡單的設計開始說起，每個基地台都是由一組的遠端射頻模組 (Remote Radio Unit, RRU)、基頻模組 (Baseband Unit, BBU) 以及天線所組成。由於基地台的容量是為了區域最高峰的需求所配置，導致有許多資源在平時被閒置造成浪費。於是後來的 Centralized RAN (C-RAN) 提出了兩種選項。第一種是將 BBU 給集中在一起稱為 BBU Pool，這個改動使得多個 RRU 可以動態共享 BBU Pool 資源，大幅降低 CAPEX 與 OPEX。而第二種選項則是將 BBU 切割成兩個更細的單元: CU (Centralized Unit) 以及 DU (Distributed Unit)，讓 DU 放在靠近 RRU 的位置，而 CU 則是能夠選擇與 DU 放在一起或者放得更靠近核心網路 (Core Network)。Virtualized RAN (V-RAN)則是維持 C-RAN 的架構，不過透過將虛擬化網路功能 (Virtualized Network Function, VNF) 或是容器化網路功能 (Containerized Network Function, CNF) 的技術將硬體與軟體進行解藕，CU、DU 和 BBU 等網路功能就可以運行在 x86 的通用硬體設備，因此這不僅將實體的資源運用到最大化也同時降低成本。

最後，Open RAN (O-RAN) 是由 O-RAN 聯盟 (O-RAN Alliance)[2] 所提倡，其中包含了 AT&T[3] 等 31 間電信商以及數百個會員，主張將 CU、DU 與 RRU 之間的介面公開標準化，這些元件稱之為 O-RAN Central Unit (O-CU)、O-RAN

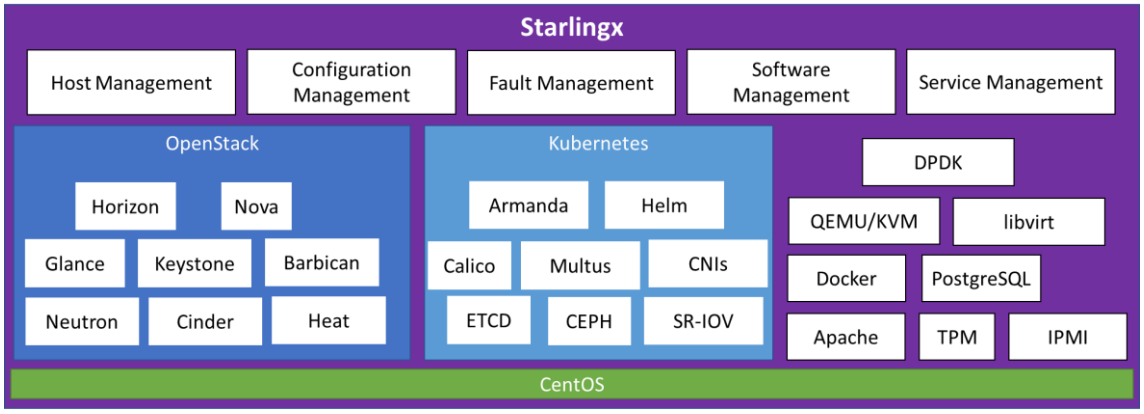
doi:10.6837/ncnu202200101

Distributed Unit (O-DU) 和 O-RAN Radio Unit (O-RU)[4]，以往之間的介面都由各家廠商自行把握，導致電信運營商只能向同一設備商購買一整組的功能。標準化讓各家廠商都能開發各自功能，並且能夠任意組合在一起，提供運營商更多的選項。除此之外，O-RAN 定義了上層的管理層 Near Real-Time RAN Intelligence Controller (Near-RT RIC)、Non Real-Time RAN Intelligence Controller (Non-RT RIC) 和 Service Management and Orchestration (SMO) 等功能，以及用來佈署網路功能的 O-Cloud。O-Cloud 可說是虛擬化的基底，負責管理雲端的實體設備。所有的 VNF/CNF 佈署都得依靠下層的實體設備來提供，因此在整體架構中十分重要。 其中包含負責管理實體設備的 Infrastructure Management Service (IMS) 以及管理 VNF 佈署服務的 Deployment Management Service (DMS)。 分別對應上層 SMO 的 Federated O-Cloud Orchestration and Management (FOCOM) 以及 Network Function Orchestration (NFO) 進行底層 O-Cloud 的實際管理服務。在這之間的介面，我們稱為 O2 介面 (O2 Interface)，O2 介面為一個 SMO 對多個 O-Clouds 的管理介面，現在已經陸續有相關的 API (Application Programming Interface)以及使用情境(Use-Case) 被標準定義[5, 6]。 不過在這些使用情境被定義的同時，流程中仍有不少實作細節未被設計。

doi:10.6837/ncnu202200101

第二節 研究目的

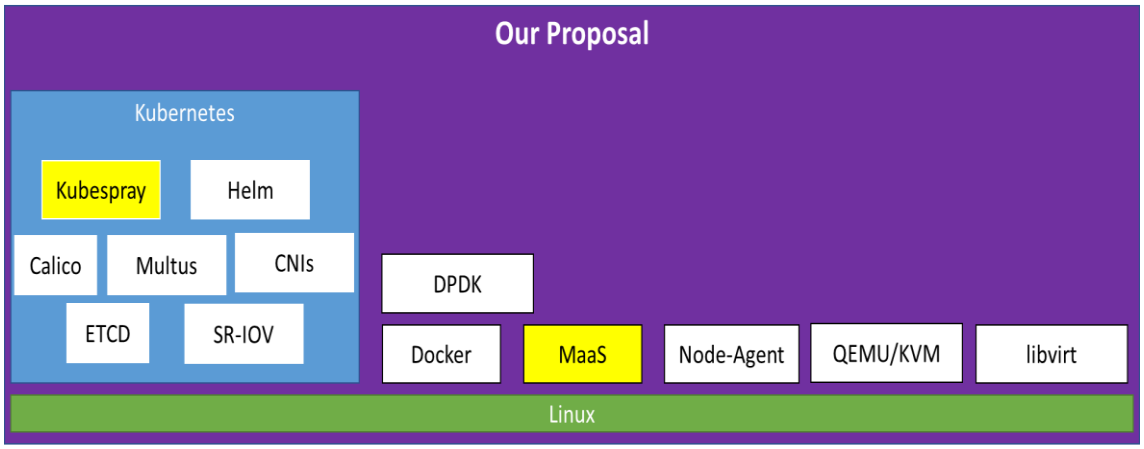
本篇論文的重點，即為了因應往後相繼開發的 VNF/CNF 可有一個適宜佈署的雲端平台，在此提出一個符合目前規範標準並且能夠實作出來的基礎設施管理服務 (IMS)，用來建設並管理實體設備，並且用數個使用情境來做展示。



圖二 Starlingx 所使用的服務

O-RAN Software Community[7] 原有一套正在開發中的 O2 Interface 的專案，不過其底層高度依賴 StarlingX 這個將大量產品整合的平台管理服務，如圖二所示。整合大量服務的缺點就是增加系統負擔，而 CU、DU 這些 VNF/CNF 都對實體資源高度依賴，有額外的服務來搶資源並不是個好的設計，並且在執行而且需要使用專案提供的環境才能執行。因此我們選擇將 StarlingX 給替換掉，佈署 CNF 的功能我們將依靠 Kubernetes[8]，當中服務都能夠容易的被替代。而在其他規範中所定義的使用情境所需功能，如圖三，我們也將避免多種工具做無法切割的整合，降低往後需要增添或改變功能的難度。

doi:10.6837/ncnu202200101



圖三 我們所用到的服務

doi:10.6837/ncnu202200101

第三節 論文架構

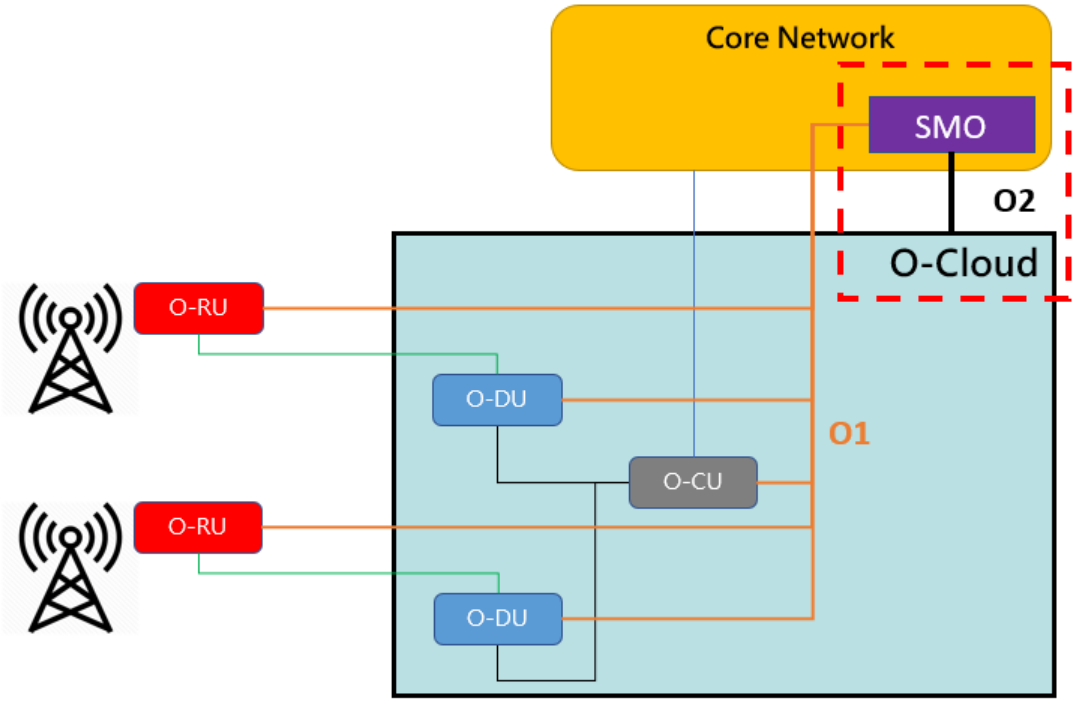
本篇論文的內容分為七章，第二章我們將介紹一些背景知識；在第三章講解 O-Cloud 自動佈署與刪除；第四章則會說明 IMS 功能的設計實作；第五章是我們如何實作整個系統；第六章則會展示我們實驗的數據；在最後的第七章下結論以及未來展望。

doi:10.6837/ncnu202200101

第二章　研究背景

本章著重在 O-Cloud 本身與 SMO 之間的 O2 介面之功能，在第二章我們將會在第一節從 O-RAN 開始介紹，第二節說明一個 O-Cloud 的架構大致為何，並在第三節講解 SMO 與 O-Cloud 之間的原件以及 O2 介面的功能，讓讀者對於整個介面的架構有基本的理解。而在第四節與第五節分別介紹我們會使用到的服務，分別是 Kubernetes 以及 Metal-as-a-Service，這兩者皆在整個 O-Cloud 的功能中提供了重要的服務。

第一節 O-RAN



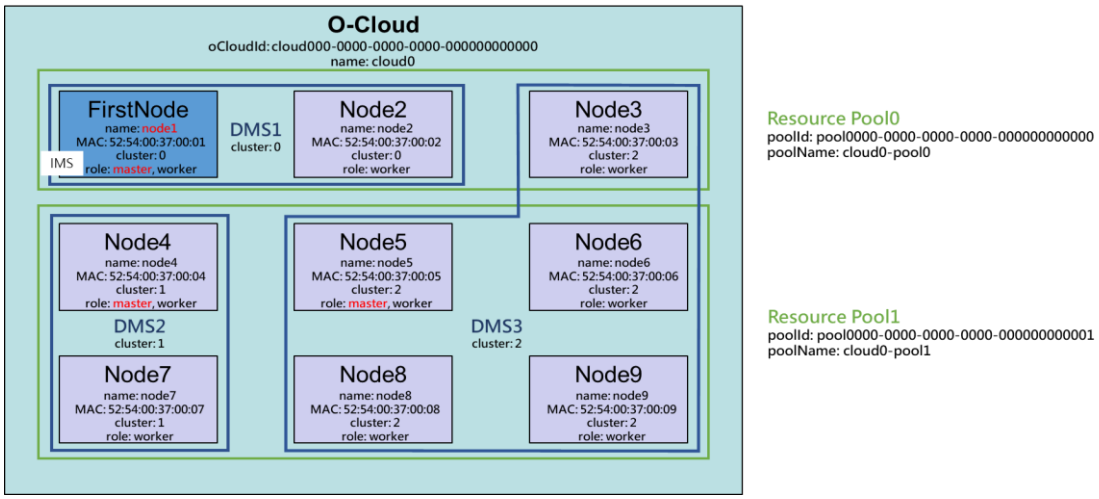
圖四 在 O-RAN 中的 O2 介面

在 O-RAN 的架構中，圖四，O-Cloud 是由一群實體設備的組成的雲端運算平台，負責來提供 VNF/CNF 佈署在實體設備上。而對於最上層的管理規劃元件 SMO 來說，底下的 O-Cloud 在邏輯上可以視為一個簡單的單一平台，但實體上 O-Cloud 可以由多個分散在不同地點的機房所構成。將 O-Cloud 分布在不同地點的好處有兩個，第一個是可租用不同的實體提供商，而第二個是對於需要較低延遲的元件，

doi:10.6837/ncnu202200101

佈署在離提供服務越近的地方越有利。在標準當中也有定義不一樣的使用情境，可以將不同的元件 Near RT RIC, O-CU, O-DU 或是 O-RU 亦可根據需求被虛擬化佈署在不同的 O-Cloud 上。

第二節 O-Cloud



圖五 一個 O-Cloud 的架構圖

圖五是一個 O-Cloud 的架構圖，裡面的每個紫色底色的框框就是指一台機器。標為藍色底色的是一台特別的機器，這是整個 O-Cloud 裡第一且唯一要先被手動安裝且設定好的 FirstNode；關於這個 FirstNode 的功用，我們會在第三章裡詳細敘述。每台機器的資訊主要包含機器名稱、MAC (Media Access Control) Address、所屬的 Kubernetes 群集 (K8s Cluster) 以及角色。

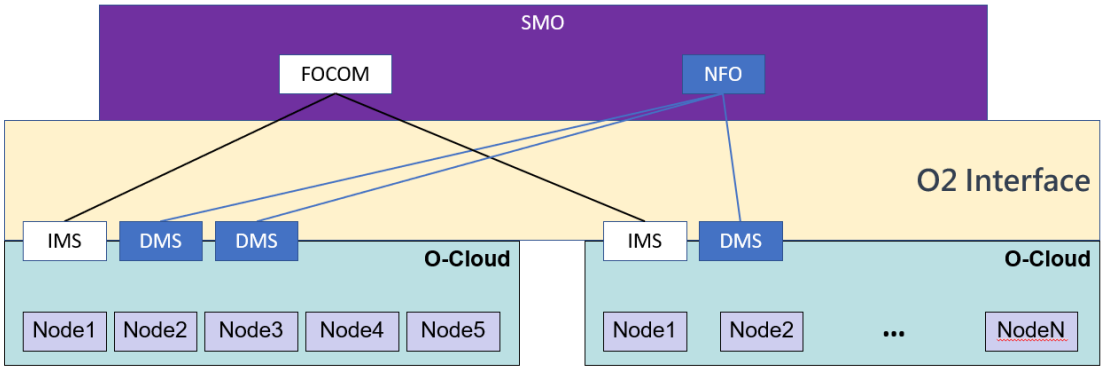
藍色線框起來的為 DMSs，也就是 K8s Clusters，在一個 O-Cloud 中可以劃分多個 DMS，而在規劃時，SMO 也需要根據每台機器的資源來決定這台機器在 Cluster 角色為何，主要分成兩類 Master 以及 Worker，更詳細的敘述我們會在本章的第四節來介紹。

接著用綠色線框起來的一群機器，我們稱它們是在同一個 Resource Pool 底下，Resource Pool 的劃分主要是方便 SMO 在規劃 O-Cloud 時的分類方法，而通常同個 Resource Pool 也意味著在相同的地理位置，或是根據用途來分類。在規劃整個 O-

doi:10.6837/ncnu202200101

Cloud 時 SMO 就需要根據需求來設計類似圖五的架構圖，並照著它來佈署。對 O-Cloud 本身，Resource Pool 的劃分並沒有實質意義，它只是在對 SMO 提供資訊時，方便 SMO 自行去分類使用。

第三節 O2 介面



圖六 O2 介面中 SMO 與 O-Cloud 的元件

圖六展示的，是在 SMO 與 O-Cloud 之間的 O2 介面，O2 介面主要被分割成兩個部分，SMO 與 O-Cloud 也各有兩個元件來負責處理這兩個部分的服務，而接下來我們將依序介紹這四個元件以及它們的功用。

第一小節 Federated O-Cloud Orchestration and Management (FOCOM)

FOCOM 負責統一管理底下的多個 O-Clouds 提供 SMO 對實體資源的單一介面，主要和 IMS 溝通，需要統整底下所有的實體資源，並且追蹤這些實體資源的狀態。

第二小節 Infrastructure Management Service (IMS)

O-Cloud 中最重要的就是 IMS，每個 O-Cloud 中都有一個 IMS，IMS 的主要工作包含控管整個實體資源，向 SMO 中的 FOCOM 提供管理/監視資源的 API ，如果佈署的 CNF 需要特殊的硬體支援，這項管理資源的功能就十分重要。

除此之外，在許多 O-Cloud 相關的流程中，如: 佈署 O-Cloud、在 O-Cloud 中新增機器等，IMS 皆扮演了相當重要且不可或缺的角色。

doi:10.6837/ncnu202200101

第三小節 Network Function Orchestration (NFO)

相對於 FOCOM 負責和管理實體設備的服務，NFO 就是用來與 DMS 溝通管理 CNF 的介面，同樣對 SMO 提供單一的 CNF 管理介面。

第四小節 Deployment Management Services (DMS)

管理在 O-Cloud 上佈署的 CNF，需要實體資源時會向 IMS 申請。 DMS 就像是 Kubernetes 或是 Openstack 這樣的系統，S 提供一個統一底下不同系統的介面來向上提供服務。O-CU、O-DU 或是 O-RU 亦可根據需求被虛擬化佈署在不一樣的 O-Cloud 上。

第四節 Kubernetes (K8s)

CNF 的容器(Container) 化技術逐漸在取代 VNF 在虛擬機 (Virtual Machine, VM) 的佈署方式，而 K8s 就是用來管理這些 CNF 的一項服務，提供的功能可以做到，自動且快速的將容器佈署到機器上，根據需求的快速服務容器自動擴展，管理偵測容器狀態並自動回復。而在將 VNF 轉換成 CNF 時也會遇到不少挑戰，例如: 資源的佔用預留、服務的穩定性、大量客製化的設定等，K8s 內部的資源就可以替我們提供方便且穩定的服務。

K8s 將一群機器劃分在同個群集裡面，運用並監控這些機器的狀態，所以在 O-Cloud 中 K8s 的功能就十分適合用來擔任 DMS 的角色，不論是對於 CNF 提供方便且靈活的控制或是對於服務狀態監控，以及已經被開發的可靠套件 (SR-IOV, Multus)，這些特點皆能為整個系統的功能提供許多幫助。

在 K8s 中主要分成兩種角色，Master 與 Worker，一台機器可以同時擔任這兩個身分。Master 是負責這個 Cluster 的管理層 (Control Plane)，負責的服務以及他們的名稱主要分為四項為: 儲存 Cluster 的資訊 (etcd)、接收前端的指令 (kube-api-server)、監控每台機器以及服務的狀況 (kube-controller-manager) 和負責佈署工作的 (kube-scheduler)。而 Worker 則是準備讓 CNF 佈署在自己身上，也運行著幾個服務: 與 Master 溝通 (kubelet)、網路代理 (kube-proxy)以及用來執行容器的

doi:10.6837/ncnu202200101

(Container Runtime)。

第五節 Metal-as-a-Service (MaaS)

MaaS[9] 為管理實體或虛擬機器的服務，能夠控制每台機器的狀態，包括開關機等服務，另外也提供一個獨立且完整的 PXE(Preboot eXecution Environment) 服務[10]，提供我們對於實體機足夠的掌控。

PXE 這項服務主要是提供藉由網路介面卡，來幫助在同個網路中的機器安裝作業系統的工具，而在一個 O-Cloud 中，要將整個資料中心 (Data Center)初始化時，每台機器皆是剛安裝好的狀態。透過此項服務，我們只需要提前蒐集好我們所需要的資訊，像是網路介面卡上的 MAC Address，就可以透過 PXE 來將每台實體機的作業系統安裝好，並開始下一階段的環境設定及軟體安裝。

在此值得注意的是，在 PXE 服務中一開始的引導是依靠 DHCP 來做，意即在 DHCP Server (Dynamic Host Configuration Protocol Server) 需要先行設定，MaaS 也有提供自己的 DHCP Server 可以使用，不過得注意在相同的區域網路只能有一台 DHCP Server。

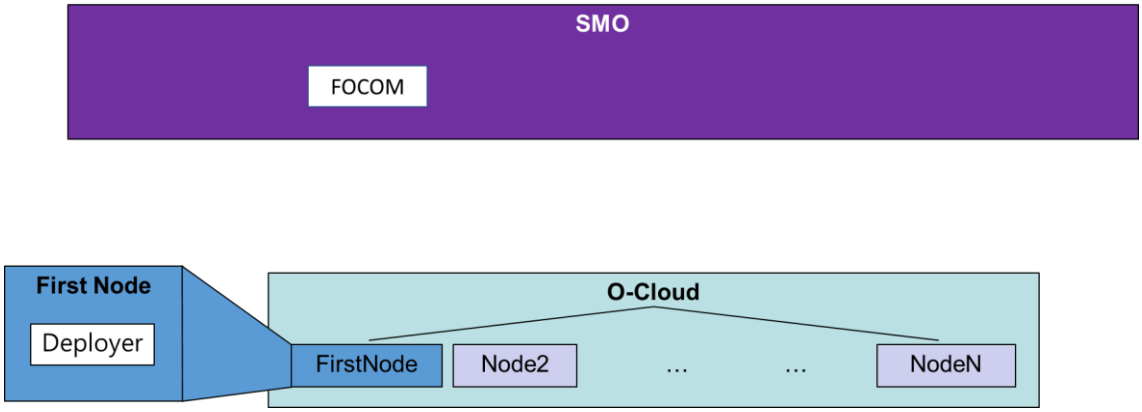
doi:10.6837/ncnu202200101

第三章　O-Cloud

在上一章對於 O-Cloud 有充分的了解過後，我們將來介紹整個 O-Cloud 最開始也是最重要的基本功能，自動化佈署。我們將在接下來的第一節先講解自動佈署的細節，以及如何設計與實作當中的步驟。在第二節介紹由我們提出用以重置 O-Cloud 的功能。最後，詳細介紹我們自己開發的服務 Deployer，以及如何使用它的 API。

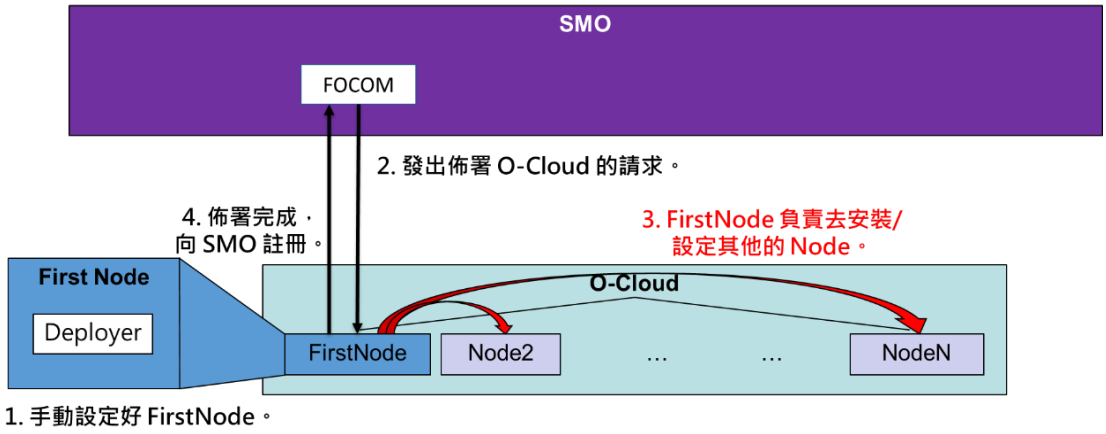
第一節 O-Cloud 自動化佈署

當一個 O-Cloud 在硬體環境剛搭建好，而所有機器都還沒被設定甚至沒有安裝 OS 的時候，要如何讓這些機器自動設定好就是個很重要的功能，這是被定義在 [6] 的使用情境，不過在 O-Cloud 中怎麼去設定與安裝每台機器的步驟並沒有定義。請先參考圖七，在一開始我們需要手動先設定好一台機器，當作我們與這個 O-Cloud 溝通的介面，而這台機器我們稱之為 FirstNode。這個 FirstNode 上面需要運行一個服務，用來提供整個自動化佈署的服務。這個服務要能夠接受我們的要求，並且依此來佈署整個 O-Cloud，這個我們自行開發的服務稱之為 Deployer。



圖七 O-Cloud 在佈署前的狀態

doi:10.6837/ncnu202200101



圖八 O-Cloud 自動化佈署步驟 Overview

圖八為 O-Cloud 自動化佈署大概的步驟，我們可以大概分為 4 步，在接下來會在講解裡面的細節。第 1 步，我們需要手動將 FirstNode 設定好，並且將剛才所提到的 Deployer 安裝好並執行，等待 SMO 下一步的指令。第 2 步，SMO 中的 FOCOM 傳送佈署 O-Cloud 的請求，並且帶著所需的所有資訊。用紅色字體標註的第 3 步，即為 FirstNode 將其他機器都進行安裝和設定的步驟，而我們也將在這裡提供一個我們所設計的流程。最後，當整個 O-Cloud 的佈署皆完成以後，我們也需要向 SMO 進行通知，所以第 4 步會向 SMO 進行註冊。

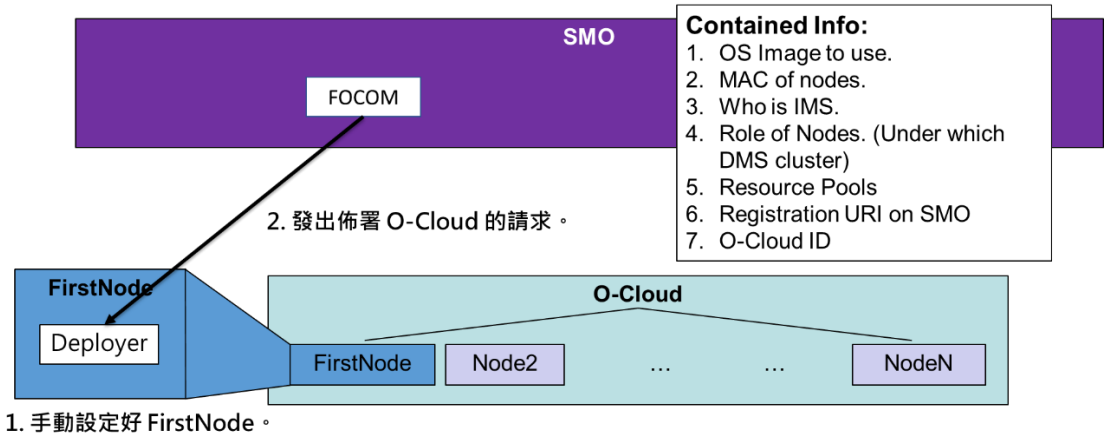
圖九為佈署中的第 1 步與第 2 步。首先，第 1 步的手動設定，我們設計的精神就是盡可能的將手動的部分簡化，所以 FirstNode 在設定時只需要將網路卡資訊設定好，安裝 Deployer 並執行，手動設定的部分就算完成。

而在第 2 步中，所需要帶的資訊細節也如圖九 O-Cloud 自動佈署的步驟 1 與 2 圖九所示，需要攜帶的資訊有七項

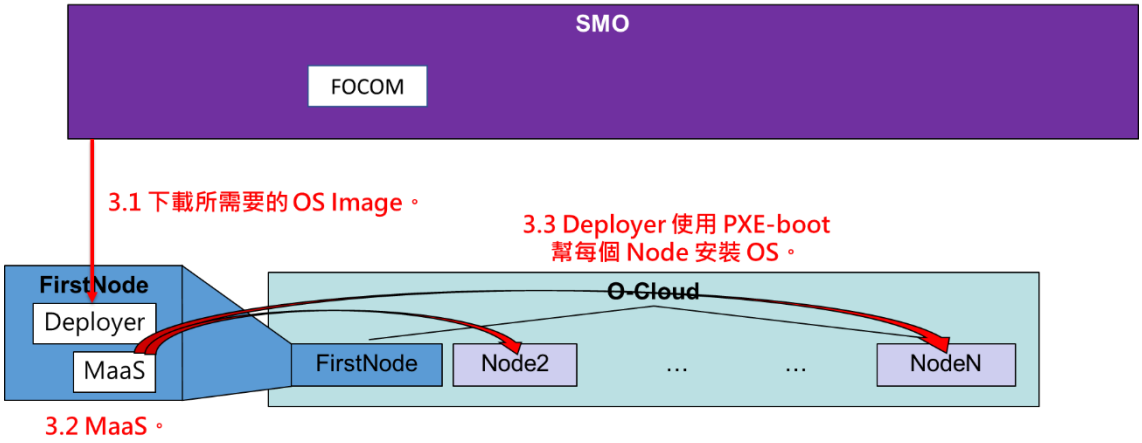
1. OS Image: 由 SMO 提供可下載的完整 URI。
2. Node MAC Address: 在安裝設備時就需要將這此記錄下來。
3. IMS 身分: 整個 O-Cloud 必須有一台機器擔任 IMS，這個是由 SMO 來決定，而這個 IMS 其實也負責整個 O2 介面 API 的運行，與 IMS 和 DMS 的溝通都是經過這個 API。

doi:10.6837/ncnu202200101

4. DMS 資訊: Node 所屬 K8s Cluster 和 Master Node、Worker Node 身份。
5. Resource Pool: Resource Pool 的劃分。
6. SMO Registration URI: O-Cloud 佈署完成後向 SMO 的註冊位置。
7. O-Cloud ID: 由 SMO 提供的 ID，以便 O-Cloud 向 SMO 溝通時的身分驗證用。



圖九 O-Cloud 自動佈署的步驟 1 與 2



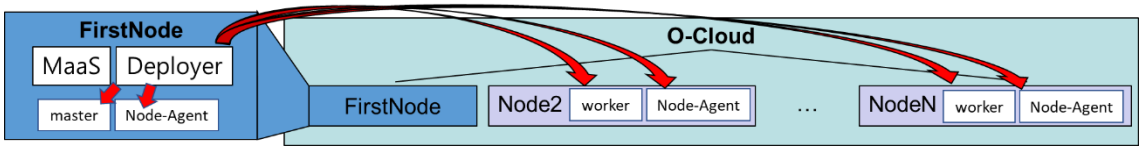
圖十 FirstNode 幫其他機器安裝 OS

圖十所示為 FirstNode 替其他機器安裝 OS 的步驟，在此我們細分為 3.1 到 3.3。首先第 3.1 步，Deployer 會根據先前第二步 SMO 提供的 URI 下載所需的 OS Image，等待下一步的取用。第 3.2 步，Deployer 會在本身安裝 MaaS 服務。第 3.3 步，MaaS 利用在第二步收集到的 MAC Address 資訊，使用 MaaS 對其他所有的機器進行 PXE

doi:10.6837/ncnu202200101

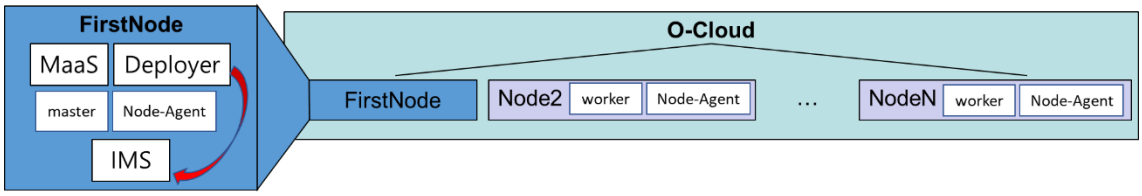
boot 的動作安裝 OS。在 MaaS 幫機器安裝 OS 時，會同時在機器上設定好 Secure Shell (SSH)，之後就可以利用此來幫其他機器進行設定與安裝。

3.4 Deployer 幫每個 Node 安裝所需軟體。



圖十一 Deployer 幫其他機器安裝所需軟體

下一步 3.4 中，圖十一所示的為 Deployer 幫其他機器安裝所需軟體，在我們的設計中，基本必要的軟體只有兩個，K8s 以及 Node-Agent，圖十一中的 Master 及 Worker 即是 K8s 中的角色，在此圖例中，只有 FirstNode 一人為 K8s Master，其餘的機器皆為 Worker。而 Node-Agent 是我們用來蒐集各台機器上的實體資訊所自行開發的服務，會在下一章有更詳細的介紹。

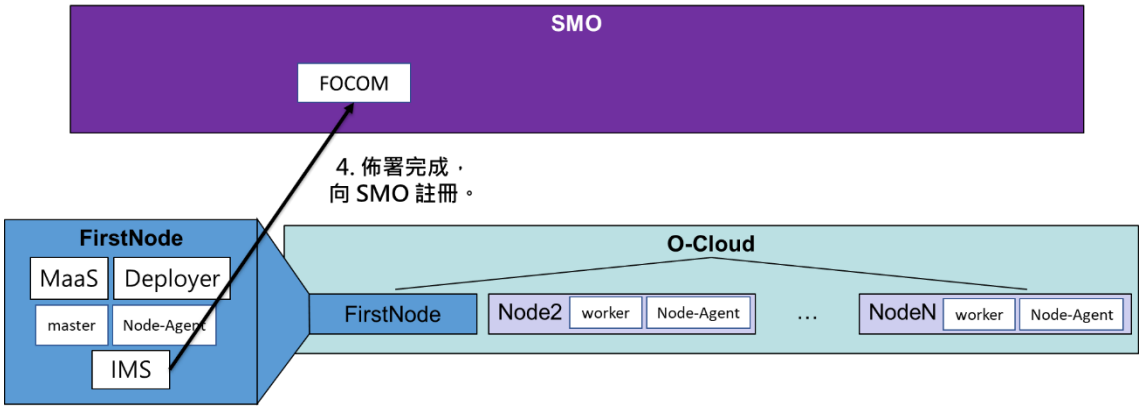


3.5 Deployer 安裝 IMS，並將 O-Cloud 資訊傳給 IMS。

圖十二 Deployer 安裝 IMS

圖十二下一步則為安裝 IMS，在此步驟會將 O-Cloud 相關資訊以及 DMS (K8s) 的相關 Token 一併傳給 IMS，IMS 會利用這個 Token 來與 K8s 串接，先前提過 IMS 同時也代表整個 O2 介面的 API，DMS 等功能都是從這個介面進行存取使用，API 細項會在第四章進行介紹。

doi:10.6837/ncnu202200101



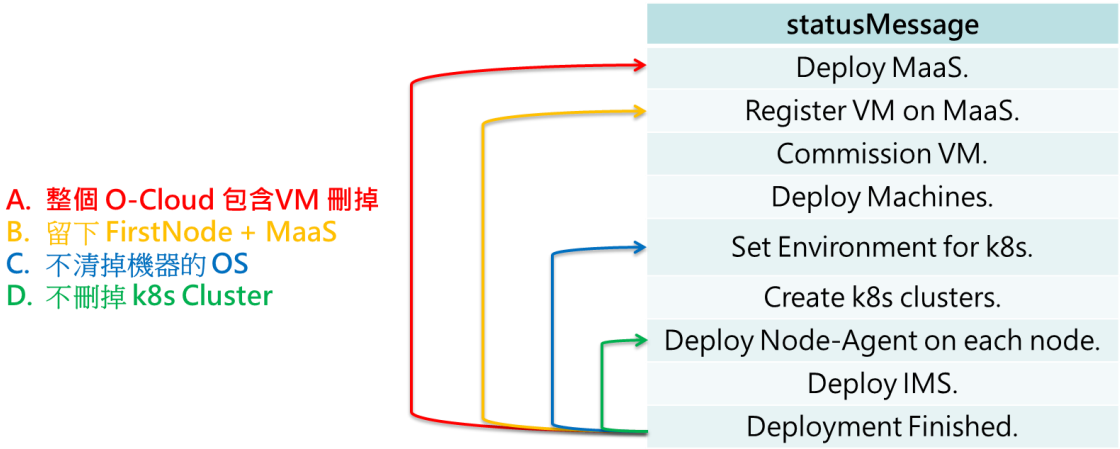
圖十三 佈署完成向 SMO 的 FOCOM 進行註冊

圖十三，最後一步即為 IMS 向 SMO 中的 FOCOM 進行最後的註冊，告知 SMO 這個 O-Cloud 已經完成佈署，以一般的設計而言，從 SMO 發出請求直到這一步，SMO 才會知道目前的狀況，而我們的 Deployer 則提供 SMO 可以主動查詢佈署狀態的功能，詳細作法會在本章的第三節講解 Deployer 的用法。

佈署 O-Cloud 的步驟到此已經完成。依 O-Cloud 大小規模不同，整體佈署花費的時間略有差異，細節會在第六章的實驗數據呈現。

第二節 O-Cloud 刪除

在 O-RAN 規範中並沒有規範 O-Cloud 的刪除情境，所以針對 O-Cloud 的刪除我們先列出四種不同的實作選項，如圖十四。



圖十四 四種不同的 O-Cloud 刪除選項

doi:10.6837/ncnu202200101

在圖十四中列出的四個實作選項為

A. 刪除整個 O-Cloud 包括 FirstNode: 再重新佈署時，需要連手動設定 FirstNode 都重來。

B. 留下 FirstNode+MaaS: 減去安裝 MaaS 的時間。

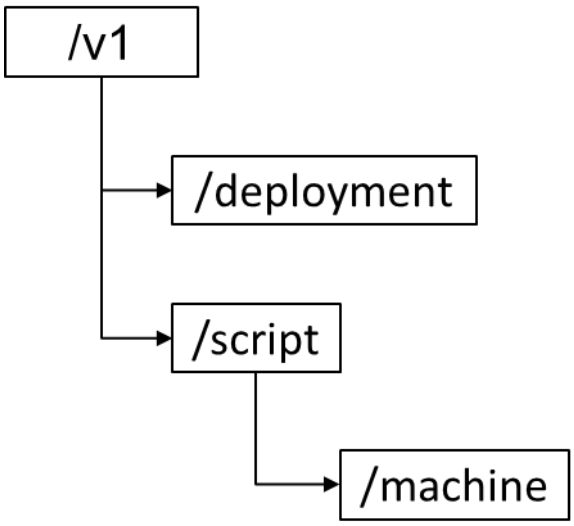
C. 不刪除機器: 機器不再需要重新安裝 OS，不過仍需要設定與安裝 K8s 與 Node-Agent。

D. 不刪除 K8s Cluster: 佈署 K8s Cluster 非常花時間，所以留下 K8s Cluster 可以加速重新佈署的時間。

最後我們選擇時做的為選項 B，原因為第一將機器完全關閉能夠，也能與未來節能的使用情境相關聯，例如: 在一個地區對於行動網路的需求下降後，刪除這個地區的 O-Cloud，重新佈署一個較小的 O-Cloud，或是直接將此區域的用戶轉移至其它 O-Cloud。刪除 O-Cloud 的 API 介面同樣會在第四章詳細介紹。

第三節 Deployer

Deployer 是我們開發來進行整個 O-Cloud 佈署與刪除的服務，這個 API 採用 REST 架構[11]，而物件架構如下面圖十五所示，接下來我們會將整個 API 的表格列出並說明如何使用這個 API。



圖十五 Deployer API 物件架構

doi:10.6837/ncnu202200101

表一 Deployer Script 物件 API

API	METHOD	URI	Example	Response Code	Response Body

Create Script	POST	/v1/scripts	curl -X 'POST' \
			'http://<first_node_ip_addr>:8080/v1/scripts' \
			-H 'accept: application/json' \
			-H 'Content-Type: application/json' \
			-d '{
			"name": "test script",
			"description": "Test on create a new script.",
			"machines": [],
			"imageURI": "/path/to/image",
			"registrationURI": "/path/to/registration"
			"resourcePools": [{"poolId": "pool0000-0000-0000-0000-000000000000",
			"poolName": "cloud0-pool0"}]
			}'
				201	{
					"scriptId": "string"
					}

				400	{
					"message": "string"
					}

Get All Scripts	GET	/v1/scripts	curl -X 'GET' \ 'http://<first_node_ip_addr>:8080/v1/scripts' \ -H 'accept: application/json'	200	[**ScriptObject1, ScriptObject2…**]

Get Script	GET	/v1/script/{scriptID}	curl -X 'GET' \
			'http://<first_node_ip_addr>:8080/v1/script/04c1aa02-3a58-41e5-b4f7-c940a0959fc5' \
			-H 'accept: application/json'
				200	{
					"scriptId": "string",
					"name": "string",
					"description": "string",
					"machines": [],
					"imageURI": "string",
					"registrationURI": "string"
					}

				404	{
					"message": "string"
					}

Delete Script	DELETE	/v1/script/{scriptID}	curl -X 'DELETE' \ 'http://<first_node_ip_addr>:8080/v1/script/04c1aa02-3a58-41e5-b4f7-c940a0959fc5' \ -H 'accept: application/json'	204	

				404	{
					"message": "string"
					}

首先，表一為 Script 物件的 API 用法以及可能收到的回覆表格，Script 這個物件是用來裝類似於圖五的整個 O-Cloud 藍圖的資訊，而裡面每個值的意義描述如下:

1. scriptId: 用來辨識 Script 物件的 Universally Unique Identifier (UUID)[12] ，在 POST 後成功後會回傳。
2. name: 此 Script 物件的名稱。
3. description: 用來描述 Script 物件。
4. machines: 此 O-Cloud 內的所有機器資訊，可於創建 Script 物件時一併帶入，或是之後另外新增。Machine 物件的詳細內容請看下方表二描述。
5. imageURI: 由 SMO 提供下載 OS Image 的位址。
6. registrationURI: O-Cloud 佈署完成後要向 SMO 中的 FOCOM 註冊位址。
7. resourcePools: 會在這個 O-Cloud 中出現的 Resource Pool 需在此指定，後面在新增 Machine 時會進行檢查，Resource Pool 物件為兩個值所組成的

doi:10.6837/ncnu202200101

物件，”poolId” 以及 “poolName”。

表二 Deployer Machine 物件 API

API	METHOD	URI	Example	Response Code	Response Body

Create Machine	POST	/v1/script/{scriptID}/machines	curl -X 'POST' \
			'http://<first_node_ip_addr>:8080/v1/scripts/1/machines' \
			-H 'accept: application/json' \
			-H 'Content-Type: application/json' \
			-d '{
			"name": “node1",
			"MAC": "52:54:00:37:00:01",
			"IMS": true,
			“cluster": 0,
			“master": true,
			“worker": true,
			“poolId": “pool0000-0000-0000-0000-000000000000”,
			“poolName": “cloud0-pool0”,
			}'
				201	{
					"machineId": "string"
					}

				400	{
					"message": “string”
					}

Get All Machines	GET	/v1/script/{scriptID}/machines	curl -X 'GET' \ 'http://<first_node_ip_addr>:8080/v1/script/1/machines' \ -H 'accept: application/json'	200	[MachineObject1, MachineObject2…]

Get Machine	GET	/v1/script/{scriptId}/machine/{machineID}	curl -X 'GET' \
			'http://<first_node_ip_addr>:8080/v1/script/1/machine/0f52c978-3cd9-4564-aac3-bfcd9c936b96' \
			-H 'accept: application/json'
				200	{
					"machineId": "string",
					"name": "string",
					"MAC": "string",
					"cluster": 0,
					"IMS": true,
					"master": true,
					"worker": true,
					"poolId": "string",
					"poolName": "string",
					"statusMessage": "BareMetal",
					"IP": "null"
					}

				404	{
					"message": “string”
					}

Delete Machine	DELETE	/v1/script/{scriptId}/machine/{machineID}	curl -X 'DELETE' \ 'http://192.168.10.2:8080/v1/script/1/machine/0f52c978-3cd9-4564-aac3-bfcd9c936b96' \ -H 'accept: application/json'	204	

				404	{
					"message": “string”
					}

表二為 Machine 物件的相關 API，正如剛才所提，Machine 物件可以在創造 Script 物件時一併創造，亦或者在 Script 創建完後新增或刪除，而 Machine 物件的相關值如下:

1. machineId: 用來辨識 Machine 物件的 UUID，在 POST 成功後會回傳。
2. name: 此 Machine 物件的名稱。
3. MAC: 這台機器的 MAC Address，至少一個，供 PXE 使用。
4. cluster: 這台機器所屬的 K8s Cluster 編號，編號為數字，同個數字意謂在同一個 Cluster。
5. IMS: 這台主機是否為 IMS，整個 Script 物件中應只有一個 Machine 的 IMS 值為 ”True”。
6. master: 是否為所屬 K8s Cluster 中的 Master。

doi:10.6837/ncnu202200101

7. worker: 被是否為所屬 K8s Cluster 中的 Worker。
8. poolId: 被劃分的 Resource Pool ID。
9. poolName: 被劃分的 Resource Pool 名稱。
10. statusMessage: MaaS 幫這台機器安裝的狀態，依序為:"BareMetal"、"Commissioning" 、"Testing" 、"Ready"、"Deploying"以及" Deployed"，如圖十六。
11. IP: 在 MaaS 幫這台機器安裝完成後，Deployer 會把這台機器的 IP Address 紀錄起來，以便後續透過 SSH 存取這台機器。

圖十六 Machine statusMessage 總覽

doi:10.6837/ncnu202200101

表三 Deployer Deployment 物件 API

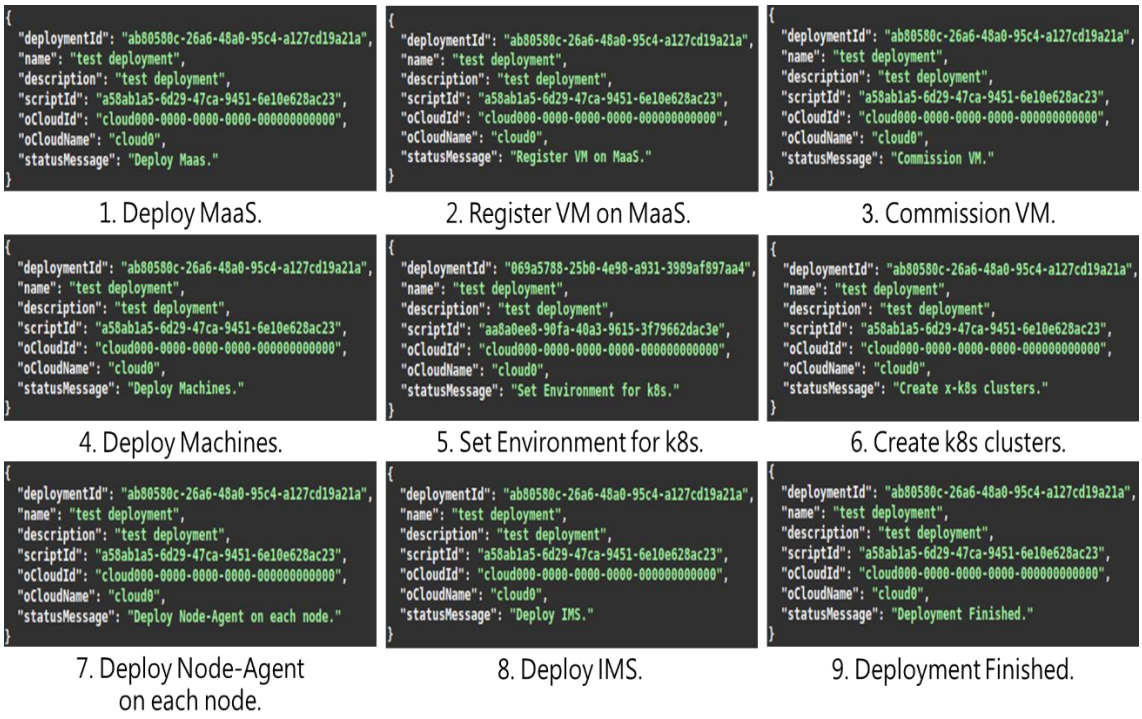
API	METHOD	URI	Example	Response Code	Response Body
Create Deployment	POST	/v1/deployments	curl -X 'POST' \ 'http://<first_node_ip_addr>:8080/v1/deployments' \ -H 'accept: application/json' \ -H 'Content-Type: application/json' \ -d '{ "name": "DeployOCloudFirst", "description": "Test create a deployment.", "scriptId": "04c1aa02-3a58-41e5-b4f7-c940a0959fc5", "oCloudId": "cloud000-0000-0000-0000-000000000000", "oCloudName": "cloud0", }'	201	{ "deploymentId": "string" }
				400	{ "message": "string" }
Get All Deployment	GET	/v1/deployments	curl -X 'GET' \ 'http://<first_node_ip_addr>:8080/v1/deployments' \ -H 'accept: application/json'	200	[**DeploymentObject1, DeploymentObject2,**]
Get Deployment	GET	/v1/deployment/{deploymentID}	curl -X 'GET' \ 'http://<first_node_ip_addr>:8080/v1/deployment/f6a7781c-a511-4ede-a6a9-df47b00b7de5' \ -H 'accept: application/json'	200	{ "deploymentId": "string", "name": "string", "description": "string", "scriptId": "string", "oCloudId": "string", "oCloudName": "string", "statusMessage": "Deployment Created." }
				404	{ "message": "string" }
Delete Deployment	DELETE	/v1/deployment/{deploymentID}	curl -X 'DELETE' \ 'http://<first_node_ip_addr>:8080/v1/deployment/f6a7781c-a511-4ede-a6a9-df47b00b7de5' \ -H 'accept: application/json'	204	
				404	{ "message": "string" }

Deployment 物件代表其為一次的佈署，此物件的新增與刪除就分別代表 O-Cloud 的佈署與刪除，而有關 Deployment 物件的值描述如下:

1. deploymentId: 用來辨識 Machine 物件的 UUID，在 POST 後成功後會回傳。
2. name: 此 Deployment 物件的名稱。
3. description: 用來描述 Deployment 物件。
4. scriptId: 儲存 O-Cloud 資訊的 Script 物件。
5. oCloudId: 這個 O-Cloud 的 ID，由 SMO 製造並維護。
6. oCloudName: 這個 O-Cloud 的名稱，由 SMO 製造並維護。
7. statusMessage: 佈署的狀態，依序為: ” Deploy MaaS.”、” Register VM on MaaS.”、” Commission VM.”、” Deploy Machines.”、” Set Environment for k8s.”、” Create k8s clusters.”、” Deploy Node-Agent on each node.”、” Deploy

doi:10.6837/ncnu202200101

IMS.” 以及 ” Deployment Finished.”，如圖十七。



圖十七 Deployment statusMessage 總覽

doi:10.6837/ncnu202200101

第四章　O2 介面功能設計與實作

在對整個 O-RAN 以及 O-Cloud 架構有理解後，我們就可以開始專注在 O2 介面所提供的服務上面，O2 介面可以分成兩塊：IMS 與 DMS。我們在此分成三節說明，第一節為 Infrastructure Inventories，第二節是 IMS 提供的資源訂閱服務，最後一節則為 DMS 利用串接的 K8s 佈署 CNF 的功能。

第一節 Infrastructure Inventories

表四 IMS API

Prefix: /o2ims_infrastructureInventory/v1/

	Resource URI	HTTP method	Name
O-Cloud Info	/	GET	O-Cloud Description
Resources	/resourceTypes	GET	Resource Type List
	/resourceTypes/{resourceTypeId	GET	Resource Type Description
	/resourcePools	GET	Resource Pool List
	/resourcePools/{resourcePoolId}	GET	Resource Pool Description
	/resourcePools/{resourcePoolId} /resources	GET	Resource List
	/resourcePools/{resourcePoolId} /resources/{resourceId}	GET	Resource Description
DMS Info	/deploymentManagers	GET	Deployment Manager List
	/deploymentManagers/{deploymentManagerId}	GET	Deployment Manager Description
Subscriptions	/subscriptions	GET	Inventory Subscription List
	/subscriptions	POST	Inventory Subscription List
	/subscriptions/{subscriptionId}	GET	Inventory Subscription Description
	/subscriptions/{subscriptionId}	DELETE	Inventory Subscription Description

表四為 IMS 的相關 API 以及它們的功能名稱，IMS 的 API 主要分為四類，分為: O-Cloud 資訊、Resources (實體資源相關)、DMS 資訊以及實體資源訂閱服務，我們將在這節一一介紹。

第一小節 O-Cloud 資訊

首先，IMS 所提供的 O-Cloud 資訊如圖十八，提供 O-Cloud ID、名稱等資訊。

doi:10.6837/ncnu202200101

```
{
  "oCloudId": "cloud000-0000-0000-0000-000000000000",
  "globalCloudId": null,
  "name": "cloud0",
  "description": "An ocloud",
  "infrastructureManagementServiceEndpoint": "http://localhost:5005/o2ims_infrastructureInventory/v1/"
}
```


圖十八 IMS API 提供的 O-Cloud 資訊

第二小節 實體資源資訊


```
{
  "resourcePoolId": "pool0000-0000-0000-0000-000000000000",
  "name": "cloud0-pool0",
  "globalLocationId": "pool0000-0000-0000-0000-000000000000",
  "location": "hsinchu1",
  "description": "A Resource Pool"
},
```


圖十九 IMS API 提供的 Resource Pool 資訊

圖十九為從 IMS 上獲得的 Resource Pool 資訊，我們可以使用這個 Resource Pool 的 ID 來搜尋底下的所有實體資源，搜尋結果可看到圖二十。在此範例中我們可以看到三台實體主機的 Resource ID，而我們可以再進一步搜尋這台主機上的資源。在下方的圖二十一與圖二十二中，分別顯示的是 Node1 上的 CPU 以及網路介面卡的資源，我們也可以看見這兩張圖中 ”parentId” 的欄位與圖二十中第一個 Resource 的 ”resourceId” 都同樣是”a05acb2e-45fe-4883-851b-08bddcf7b0b8”。

doi:10.6837/ncnu202200101

```
{
  "resourceId": "a05acb2e-45fe-4883-851b-08bddcf7b0b8",
  "resourceTypeId": "0c96ddfc-eef1-44b3-b20d-b0f6c1a1f077",
  "resourcePoolId": "pool0000-0000-0000-0000-000000000000",
  "name": "node1",
  "parentId": "pool0000-0000-0000-0000-000000000000",
  "description": "A physical server resource"
},
{
  "resourceId": "8bc7e79d-41b7-4c02-940b-06b0a412e740",
  "resourceTypeId": "0c96ddfc-eef1-44b3-b20d-b0f6c1a1f077",
  "resourcePoolId": "pool0000-0000-0000-0000-000000000000",
  "name": "node10",
  "parentId": "pool0000-0000-0000-0000-000000000000",
  "description": "A physical server resource"
},
{
  "resourceId": "0688f267-01be-4512-aab1-6a1f78b8896c",
  "resourceTypeId": "0c96ddfc-eef1-44b3-b20d-b0f6c1a1f077",
  "resourcePoolId": "pool0000-0000-0000-0000-000000000000",
  "name": "node2",
  "parentId": "pool0000-0000-0000-0000-000000000000",
  "description": "A physical server resource"
},
```


圖二十 IMS Resource Pool 底下的實體主機


```
{
  "resourceId": "52b1e9fd-ec07-3bca-9548-f75315947076",
  "resourceTypeId": "ea3dac25-7d17-41d6-8642-9cf53b36fa6c",
  "resourcePoolId": "pool0000-0000-0000-0000-000000000000",
  "name": "a05acb2e-45fe-4883-851b-08bddcf7b0b8-cpu-0",
  "parentId": "a05acb2e-45fe-4883-851b-08bddcf7b0b8",
  "description": "A CPU resource of the physical server"
},
```


圖二十一 Node1 上的 CPU 資源


```
{
  "resourceId": "d158c20c-8b53-3f4f-8ad9-95d7c67fbebd",
  "resourceTypeId": "93ef92a6-cf89-480a-8a5c-3f05a6f04835",
  "resourcePoolId": "pool0000-0000-0000-0000-000000000000",
  "name": "a05acb2e-45fe-4883-851b-08bddcf7b0b8-if-cali4bdd3fee2b7",
  "parentId": "a05acb2e-45fe-4883-851b-08bddcf7b0b8",
  "description": "An interface resource of the physical server"
},
```


圖二十二 Node1 上的網路介面卡資源

doi:10.6837/ncnu202200101

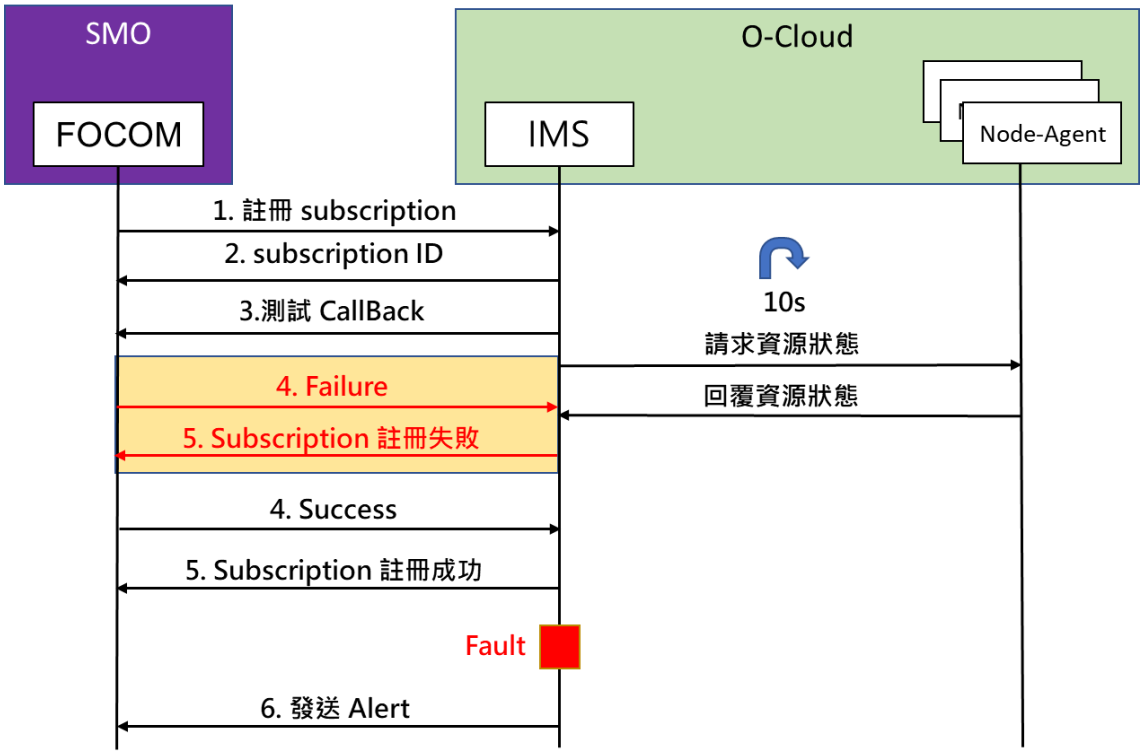
第三小節 DMS 資訊


```
"deploymentManagerId": "cc2a782d-c151-36e0-9f58-96cf07543c4e",
"name": "k8s-cluster0",
"description": "A DMS",
"deploymentManagementServiceEndpoint": "http://localhost:5005/o2dms/v1/cc2a782d-c151-36e0-9f58-96cf07543c4e",
```


圖二十三 從 IMS 獲取的 DMS 資訊

圖二十三是從 IMS 上所獲取的 DMS 訊息，在這裡我們主要可以獲取 DMS 的 ID，在 O2 介面的 DMS API 使用這個 ID 來指定 DMS 進行動作。

第四小節 Subscription



圖二十四 訂閱實體資源 (Subscription) 信令圖

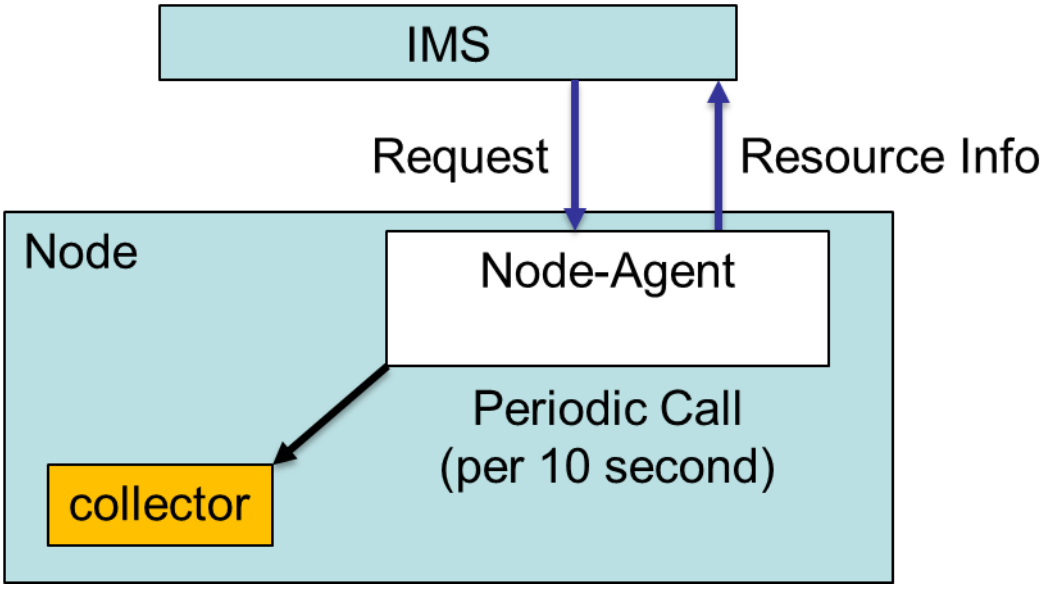
訂閱實體資源 (Subscription)的功能由於牽扯到較多步驟，我們在這裡使用信令圖來展示，先決條件為 IMS 會定期(每 10 秒)向各台機器查詢目前的實體資源狀態。

1. 首先，由 SMO 的 FOCOM 向 IMS 發出訂閱的請求，並在請求中帶入如果發生 Alert 要呼叫的 CallBack 位址。
2. IMS 先行測試 CallBack 的功能。

doi:10.6837/ncnu202200101

3. FOCOM 回復 CallBack 的請求為失敗或成功。
4. 如果 FOCOM 回覆 CallBack 成功，則 IMS 會告知 FOCOM 回覆註冊成功。失敗的話 IMS 則直接取消此次的訂閱。
5. 當有 Fault 發生時，如資源突然故障或者使用超過限額，則 IMS 就會觸發 CallBack 通知 FOCOM。

第二節 Node-Agent



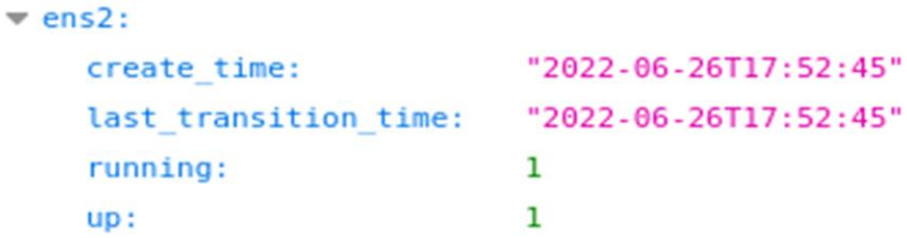
圖二十五 Node-Agent 的運作架構

為了獲得每台機器的實體資源資訊，Node-Agent 會執行在每台機器上，並且提供 IMS 隨時獲取實體資源狀態的服務（如: CPU 的使用率、網路介面卡的狀態）。由於獲得 CPU 使用率這種資源會有阻塞 (Blocking) 的狀況，而我們不希望 IMS 在獲取資訊時還需要慢慢等待，於是 Node-Agent 的存在就是定期(10 秒)獲得自身機器的資訊，並隨時等待 IMS 來獲取，如圖二十五。

從 Node-Agent 上得到的 CPU 以及網路介面卡資訊，分別為圖二十六及圖二十七，可以看到 CPU 的各個核心 (Core) 的使用率和網路介面卡是否正在運作或啟用。

doi:10.6837/ncnu202200101

圖二十六 Node-Agent 上的 CPU 資源使用率資訊



圖二十七 Node-Agent 上的網路介面卡狀態資訊

第三節 DMS

表五 DMS API

Prefix: /o2dms/v1/{deploymentManagerID}

	Resource URI	HTTP method	Name
DMS Info	/	GET	DMS
NF Deployment	/O2dms_DeploymentLifecycle/NfDeployment	GET	NF Deployment List
	/O2dms_DeploymentLifecycle/NfDeployment	POST	Create a NF Deployment
	/O2dms_DeploymentLifecycle/NfDeployment/{nfDeploymentId}	GET	NF Deployment Description
	/O2dms_DeploymentLifecycle/NfDeployment/{nfDeploymentId}	PUT	Update NF Deployment
	/O2dms_DeploymentLifecycle/NfDeployment/{nfDeploymentId}	DELETE	Delete NF Deployment
NF Deployment Descriptor	/O2dms_DeploymentLifecycle/NfDeploymentDescriptor	GET	NF Deployment Descriptor Description
	/O2dms_DeploymentLifecycle/NfDeploymentDescriptor	POST	Create a NF Deployment Descriptor List
	/O2dms_DeploymentLifecycle/NfDeploymentDescriptor/{nfDeploymentDescriptorId}	GET	Deployment Manager Description
	/O2dms_DeploymentLifecycle/NfDeploymentDescriptor/{nfDeploymentDescriptorId}	PUT	Create a NF Deployment Descriptor Description
	/O2dms_DeploymentLifecycle/NfDeploymentDescriptor/{nfDeploymentDescriptorId}	DELETE	Delete Create NF Deployment Descriptor

DMS 提供 NF 相關的功能，DMS

doi:10.6837/ncnu202200101

表五所列為 DMS API，分為三類：DMS 資訊、NF Deployment 及 NF Deployment Descriptor。DMS 資訊與 IMS 上的相同，在此就不贅述。

NF Deployment Descriptor 則是 NF Deployment 的資訊，如圖二十八，K8s 使用的是 Helm[13] 來達成 NF 的佈署。我們只需要建立好 Helm Chart 並且放在 Github[14]上，在 ”artifactRepoUrl” 帶入 Github Repository 的網址，並且指定 Helm Chart 的名稱 ” artifactName”，NF Deployment Descriptor 就成功建立好了。接著使用這個 NF Deployment Descriptor 的 ID 創建新的 NF Deployment，CNF 就會被佈署在 DMS 上。

圖二十八 NF Deployment Descriptor 資訊

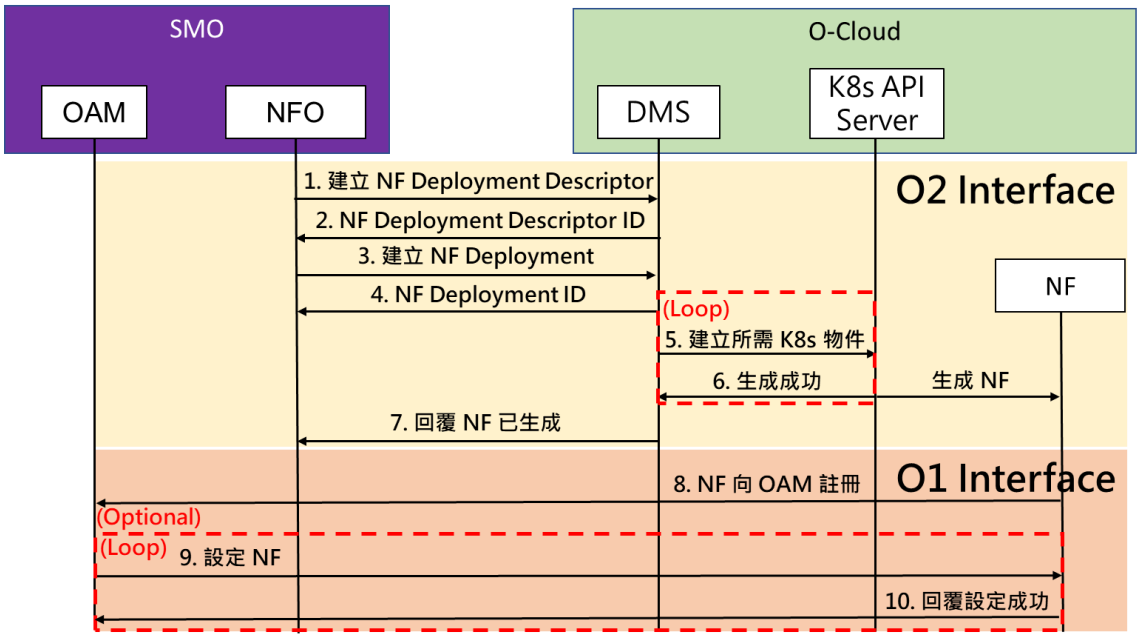
接下來，我們將使用信令圖來展示 NF Deployment 的佈署，如圖二十九。

1. NFO 向 DMS 發出建立 NF Deployment Descriptor 的請求。
2. DMS 回覆所建立的 NF Deployment Descriptor 物件 ID。
3. 接著，NFO 再向 DMS 使用剛才建立的 NF Deployment Descriptor 的 ID 來 NF Deployment 物件。
4. DMS 回覆所建立的 NF Deployment 物件 ID。
5. DMS 根據 NF Deployment Descriptor 的物件，向 K8s 的 API Server 建立 NF 所需的所有物件 (K8s 物件可為多個)。
6. K8s API Server 回覆物件建立成功 (5. 和 6. 等同於生成此 NF)。
7. DMS 回覆 NFO，NF 已經被生成。
8. NF 向 OAM (Operations, Administration and Maintenance) 註冊。

doi:10.6837/ncnu202200101

9. (Optional) OAM 可以透過 O1 介面對 NF 進行設定 (設定可分為多次)。

10. (Optional) NF 回覆 OAM 的設定成功。



圖二十九 NF Deployment 流程圖

而刪除 NF Deployment 的流程就相對簡單，如圖三十。

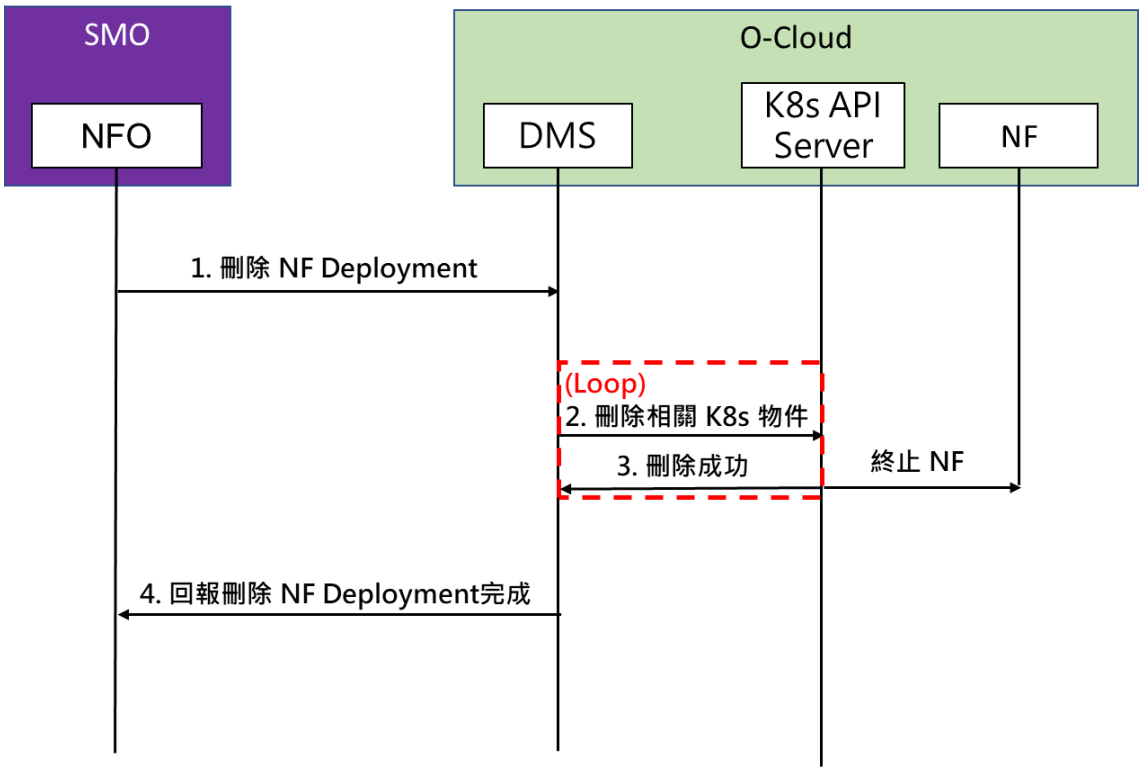
1. NFO 向 DMS 發出刪除 NF Deployment Descriptor 的請求。

2. DMS 根據 NF Deployment ID 找到相對應的 NF，並向 K8s 的 API Server 刪除相關的 K8s 物件 (K8s 物件可為多個)。

3. K8s API Server 回覆物件刪除成功的訊息 (2. 和 3. 的過程，等於在終止此 NF)。

4. DMS 向 NFO 回報 NF 已刪除。

doi:10.6837/ncnu202200101



圖三十 刪除 NF Deployment 流程圖

doi:10.6837/ncnu202200101

第五章　系統實作

實驗環境為了節省涉入的實體主機數量，我們採用了虛擬機 (VM) 的方式來做實驗，整體流程仍與實體環境相同。本文的虛擬機採用 QEMU+KVM[15] 的方式架設。本章將拆解成兩節，第一節說明我們如何架設實驗的網路環境。在第二節講解如何利用 QEMU+KVM 架設這些虛擬機，並模擬實體環境的使用情境。

第一節 實驗網路環境

在 O-Cloud 中由於 PXE 服務的限制，我們所需的網路環境需要沒有 DHCP Server，但保留可以對外連線的能力，於是我們需要一個 NAT (Network Address Translation) 網路。我們使用了 lxd Bridge 來實作此網路環境，為使用 ”lxc network show lxdbr0” 顯示的 Bridge 設定。在下一節建立虛擬機時，在指令中指定此 Bridge 即可將機器加入這個 NAT Network。除此之外，在建立虛擬機器時我們還可以指定機器的網路介面的 MAC Address，而這也更符合實際的使用情景，在這裡我們使用 52:54:00:37:00:XX，最後的一個 Byte “XX” 為機器的編號。例: Node1 的 MAC Address 即為 52:54:00:37:00:01，依此類推。

```
ubuntu@node1:~$ sudo lxc network show lxdbr0
config:
  ipv4.address: 192.168.10.1/24
  ipv4.dhcp: "false"
  ipv4.nat: "true"
  ipv6.address: none
description: ""
name: lxdbr0
type: bridge
used_by: []
managed: true
status: Created
locations:
- none
```

圖三十一 使用 ”lxc network show lxdbr0” 顯示的 Bridge 設定

doi:10.6837/ncnu202200101

第二節 實驗環境

表六 實驗主機規格

HOST	
CPU	Intel(R) Xeon(R) Gold 6152 CPU @ 2.10GHz 22 Cores 44 Threads
memory	220 GB

表六為實驗主機的規格，而表七與表八分別為 FirstNode 以及其餘每台虛擬機(Node) 的規格，由於 FirstNode 要擔任 O-Cloud 佈署者的工作，因此本文建議 FirstNode 的規格不能太差，以一個有 20 個 Node 的 O-Cloud 來說最好要有 16 個 vCPU 以及 8GB 的記憶體。

表七 FirstNode 虛擬機規格

FirstNode	
Hypervisor	QEMU 2.11.1 + KVM
vCPU	16
memory	8 GB

doi:10.6837/ncnu202200101

表八 一般虛擬機規格

Node	
Hypervisor	QEMU 2.11.1 + KVM
vCPU	2
memory	2 GB

先前提到，我們將用 QEMU+KVM 來架設虛擬機，我們可以先利用 Virt-install[16] 這個工具來生成虛擬機的 XML 檔，再用 Libvirt[17] 的工具 Virsh 來建立虛擬機。圖三十二為使用 Virt-install 的範例指令，其中要特別注意的有:

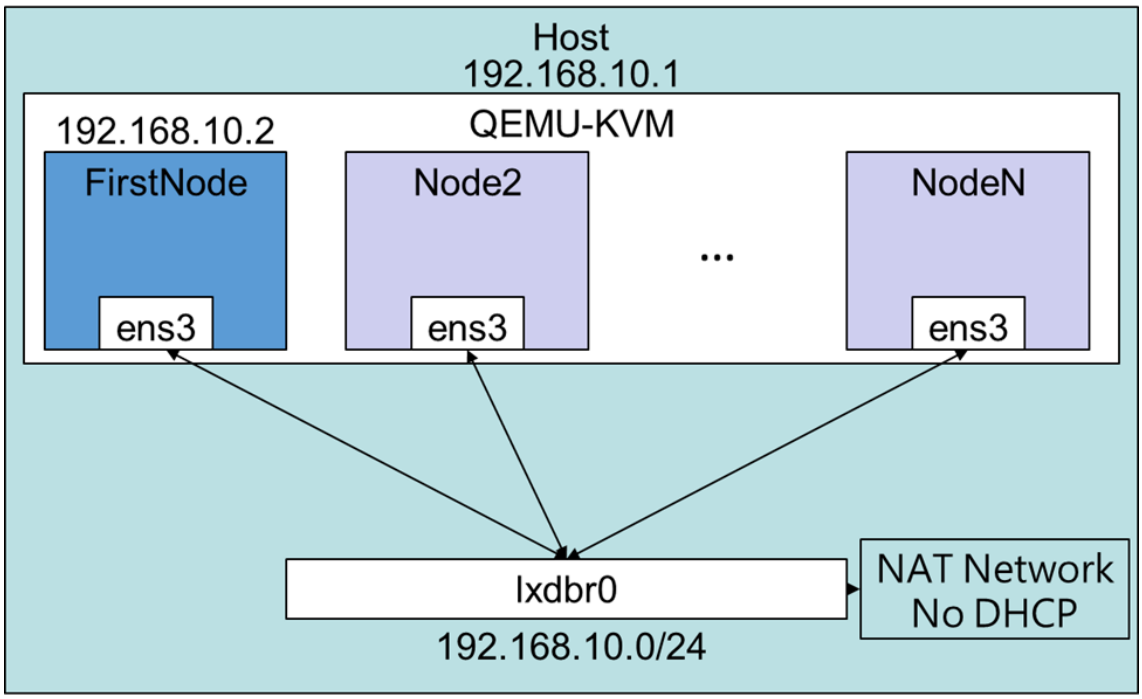
1. ”--boot=network”: 標記此虛擬機會使用 PXE 開機服務
2. 以及 ” --network bridge:lxdbr0,mac=52:54:00:37:00:02,model=virtio”: 在這個選項，我們標註了使用的 Bridge 為上一節中所設定的 ”lxdbr0” Bridge，以及設定了此介面的 MAC Address，而最後的 “model=virtio” 也是值得注意的一點，否則易有大量 Collision，詳細請見附錄一。

```
virt-install --virt-type kvm \
--name node2 \
--vcpu=2 \
--ram 2048 \
--boot=network \
--disk path=/path/to/disk.qcow2,format=qcow2,size=20 \
--network bridge:lxdbr0,mac=52:54:00:37:00:02,model=virtio \
--graphics none \
--noautoconsole \
--print-xml
```

圖三十二 Virt-install 範例指令

doi:10.6837/ncnu202200101

整體設定完成後，我們的實驗環境大致如圖三十三所示。



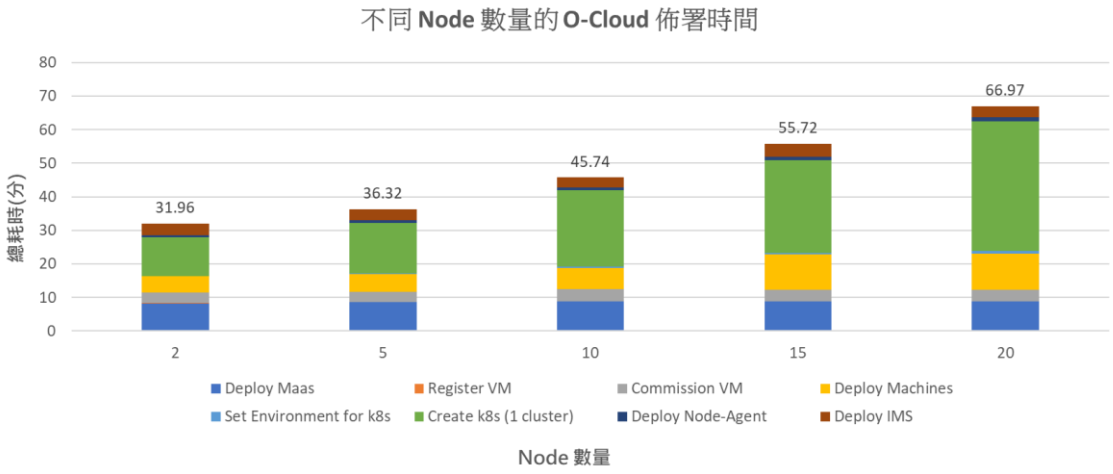
圖三十三 實驗環境架構圖

doi:10.6837/ncnu202200101

第六章實驗結果

了解整體實作流程以及實驗環境架設後，本章將會分別在第一節與第二節展示 O-Cloud 自動佈署以及刪除的耗時。

第一節 O-Cloud 自動佈署耗時



圖三十四 不同 Node 數量的 O-Cloud 佈署時間

圖三十四為不同機器 (Node) 數量的 O-Cloud 佈署時間(只有一個 DMS，也就是一個 K8s Cluster)，在圖中我們可以看見，其中主要耗時都為建立 K8s 時所消耗，而其它步驟的耗時並沒有明顯的增長。以整體的耗時時間來說，Node 數量與耗時的成長曲線接近等差的成長。

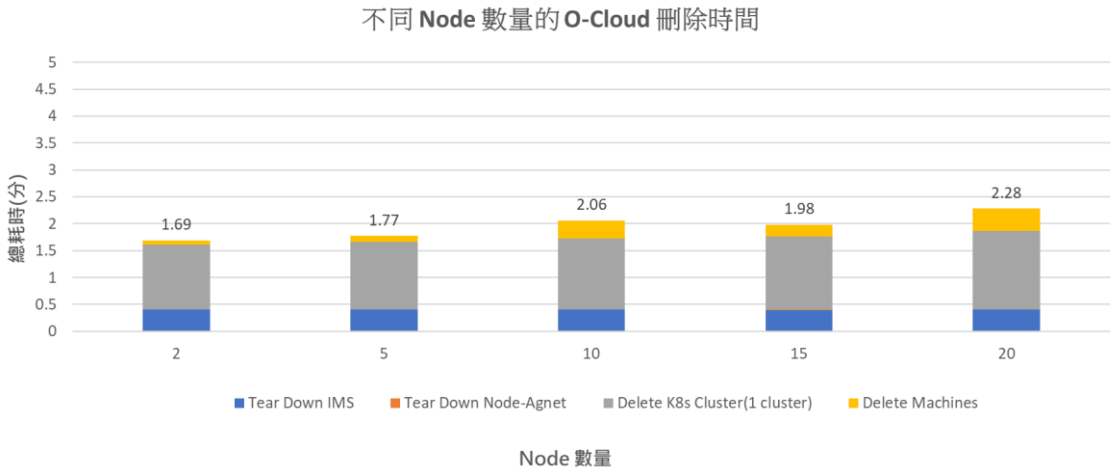
其中大部分的步驟皆為平行化進行，而如果在未來想要縮短佈署時間，則應該要從建立 K8s 的部分開始下手。而在建立 K8s Cluster 時，我們也特別觀察過資源的使用率，無論是 CPU、網路傳輸量 (Network Throughput)、記憶體或者硬碟的寫入 (Disk I/O)，在大部分時間都沒有大量占用或不夠的現象，所以大部分的時間都用在等待安裝套件的過程，絕對還有改善的空間。

第二節 O-Cloud 刪除耗時

圖三十五為不同 Node 數量的 O-Cloud 刪除時間，基本上可以說 Node 數量對

doi:10.6837/ncnu202200101

O-Cloud 刪除的耗時並沒有影響，基本耗時皆在 2 分鐘附近就可以完成。



圖三十五 不同 Node 數量的 O-Cloud 刪除時間

doi:10.6837/ncnu202200101

第七章　結論與未來展望

第一節 結論

隨著 O-RAN 的標準與使用情境發展，相關的實作必然會被相繼提出，而在測試 CNF 功能時就會需要一個能夠實作的使用環境。當前的標準仍在發展當中，細節也未被描述，在嘗試架設測試環境時必然受到大量阻礙，而官方目前的進度由於高度依賴單一產品 (Starlingx)，過於耗用資源也會導致在佈署時的效果不彰。

而本文提出一個不需高度依賴單一產品的 O-Cloud 實作方法，有完整的佈署實作方法與流程，其中包含多個使用情境的實現，亦符合目前標準所規範。並且提供可在虛擬環境上佈署 O-Cloud 的優良實驗環境。對於相繼提出的 O-CU、O-DU 等實作，實際利用 O-Cloud 佈署的實驗必不可少。

第二節 未來展望

目前 O-RAN 仍有許多未被定義的使用情境，如 Monitoring 等服務，O-Cloud 在未來的用途增加，補充這些功能的實作提供完整的測試平台，是值得再繼續努力的方向。

而除了官方的使用情境，本文提出的刪除 O-Cloud 之情境，也可以有效運用在節能的功能上，而如何最大化的節能[18]，同時留下需要的資源，必然也會是值得研究的問題。

除此之外，在未來如果有能替 FOCOM 整合多個 O-Cloud 的工具，也是相當值得開發的工具，相當期待在未來有更多 O-Cloud 以及 O2 介面功能的發展。

doi:10.6837/ncnu202200101

參考文獻

[1] *Ericsson Mobility Report*, Ericsson, 2022.

[2] "O-RAN ALLIANCE," https://www.o-ran.org/.

[3] "AT&T Inc.," https://www.att.com/.

[4] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, and L. Zeng, “OpenRAN: a software-defined ran architecture via virtualization,” *ACM SIGCOMM computer communication review,* vol. 43, no. 4, pp. 549-550, 2013.

[5] O.-R. ALLIANCE, "O-RAN.WG6.O2IMS-INTERFACE-v01.01," 2022.

[6] O.-R. ALLIANCE, "O-RAN.WG6.ORCH-USE-CASES-v03.00," 2022.

[7] "O-RAN Software Community," https://wiki.o-ran-sc.org/.

[8] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *IEEE cloud computing,* vol. 1, no. 3, pp. 81-84, 2014.

[9] "Metal As A Service (MAAS)," https://maas.io/.

[10] I. Corporation, "Preboot Execution Environment (PXE) specification v2.1," 1999.

[11] L. Richardson, and S. Ruby, *RESTful web services*: " O'Reilly Media, Inc.", 2008.

[12] P. Leach, M. Mealling, and R. Salz, *A universally unique identifier (uuid) urn namespace,* 2070-1721, 2005.

[13] S. Gokhale, R. Poosarla, S. Tikar, S. Gunjawate, A. Hajare, S. Deshpande, S. Gupta, and K. Karve, "Creating Helm Charts to ease deployment of Enterprise Application and its related Services in Kubernetes." pp. 1-5.

[14] "Github," https://github.com/.

[15] I. Habib, “Virtualization with KVM,” *Linux Journal,* vol. 2008, no. 166, pp. 8, 2008.

[16] M. J. Hammel, “Managing kvm deployments with virt-manager,” *Linux Journal,* vol. 2011, no. 201, pp. 7, 2011.

[17] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt." pp. 574-579.

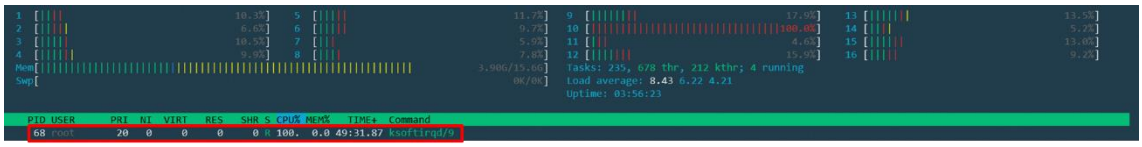
[18] J. Shuja, K. Bilal, S. A. Madani, and S. U. Khan, “Data center energy efficient resource scheduling,” *Cluster Computing,* vol. 17, no. 4, pp. 1265-1277, 2014.

doi:10.6837/ncnu202200101

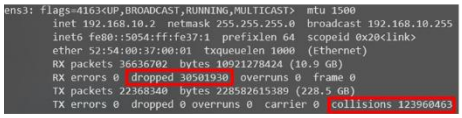
附錄

附錄一 KVM 的網路介面設定

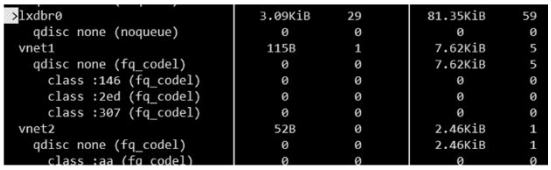
在使用 virt-install 工具創建 VM 時，須注意網路介面的 driver model 要設為 ”virtio” 而非 ” rtl8139”，否會有大量 Collision 導致 Packet Loss 的情況發生，如附圖一。



VM 上發生大量 Soft Interrupt



大量 Tx collision 和 Rx dropped



流量也不高 (觀察最高流量≈ 200M)

附圖一 當使用 rtl8139 driver 會導致大量 Packet Loss

doi:10.6837/ncnu202200101