ການປຽບທຽບປະສິດທິພາບລະຫວ່າງ Microservices ແລະ Monolithic ໃນການພັດທະນາລະບົບ IT Helpdesk ດ້ວຍ ASP: ກໍລະນີສຶກສາດ້ານຄວາມໄວ ແລະ ປະສິດທິພາບໃນການນຳໃຊ້ຊັບພະຍາກອນ

ຄຳຈັນ ພົມມະລີ , ອັດສະວິນ ເທບພະກັນ, ຄຳພັດ ບຸນນະດີ, ວິມົນທາ ຂຽວວົງພະຈັນ,
ຄະນະວິສະວະກຳສາດ ມະຫາວິທະຍາໄລແຫ່ງຊາດ, ສປປ ລາວ

ບົດຄັດຫຍໍ້

ການຄົ້ນຄວ້າຄັ້ງນີ້ມີຈຸດປະສົງເພື່ອ 1) ປຽບທຽບປະສິດທິພາບລະຫວ່າງສະຖາປັດຕະຍະກຳຊອບແວແບບ Microservices ແລະ Monolithic ໃນການພັດທະນາລະບົບ IT Helpdesk ດ້ວຍພາສາ ASP. ການປຽບທຽບໄດ້ສຸມໃສ່ 3 ດ້ານຫຼັກຄື: 1. ຄວາມໄວໃນການຕອບສະໜອງ (Response Time) 2. ການນຳໃຊ້ຊັບພະຍາກອນໜ່ວຍຄວາມຈຳ (RAM) 3.ການນຳໃຊ້ຊັບພະຍາກອນໜ່ວຍປະມວນຜົນກາງ (CPU) ໂດຍການວັດແທກຜົນໄດ້ຖືກດຳເນີນໃນ 4 ໜ້າວຽກຫຼັກຂອງລະບົບຄື:ສ້າງເຊີວິສຣິເຄວສ, ການຕອບເຊີວິສຣິເຄວສ, ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄວສ, ການສະແດງກຣາຟ. ຜູ້ຄົ້ນຄວ້າໄດ້ສ້າງລະບົບ IT Helpdesk ຂຶ້ນມາສອງແບບ: ແບບໜຶ່ງໃຊ້ສະຖາປັດຕະຍະກຳ Monolithic ແລະ ອີກສະບັບໜຶ່ງໃຊ້ Microservices. ຈາກນັ້ນ, ໄດ້ທຳການທົດສອບໂດຍການຈຳລອງການນຳໃຊ້ງານໃນປະລິມານຂໍ້ມູນທີ່ແຕກຕ່າງກັນ (100, 1,000, 50,000 ແລະ 100,000 requests) ແລະ ເກັບກຳຂໍ້ມູນປະສິດທິພາບດ້ວຍເຄື່ອງມື Resource Monitor. ຂໍ້ມູນທີ່ໄດ້ຖືກນຳມາວິເຄາະຫາຄ່າສະເລ່ຍ ແລະ ປຽບທຽບຄວາມແຕກຕ່າງ. ຜົນການຄົ້ນຄວ້າພົບວ່າ ດ້ານຄວາມໄວ (Response Time) ສະຖາປັດຕະຍະກຳ Monolithic ມີຄວາມໄວໃນການຕອບສະໜອງໄວກວ່າ Microservices ໃນທຸກໆການທົດສອບ, ໂດຍສະເລ່ຍແລ້ວໄວກວ່າປະມານ 58%. ດ້ານການໃຊ້ຊັບພະຍາກອນ (RAM ແລະ CPU) ສະຖາປັດຕະຍະກຳ Microservices ມີການນຳໃຊ້ RAM ແລະ CPU ສູງກວ່າ Monolithic ຢ່າງຊັດເຈນ ໂດຍສະເລ່ຍໃຊ້ RAM ຫຼາຍກວ່າ 84% ແລະ CPU ຫຼາຍກວ່າ 70% ເນື່ອງຈາກຕ້ອງຈັດການຫຼາຍເຊີວິສທີ່ແຍກຈາກກັນ. ເຖິງວ່າ Monolithic ຈະມີປະສິດທິພາບດ້ານຄວາມໄວ ແລະ ການໃຊ້ຊັບພະຍາກອນທີ່ດີກວ່າໃນລະບົບຂະໜາດນ້ອຍຫາປານກາງ, ແຕ່ Microservices ມີຂໍ້ດີໃນດ້ານຄວາມຍືດຢຸ່ນ, ການຂະຫຍາຍລະບົບ (Scalability) ແລະ ການບຳລຸງຮັກສາທີ່ງ່າຍກວ່າໃນໄລຍະຍາວ.

ຄຳສັບສຳຄັນ: *ໂມໂນລິທິກ, ໄມໂຄເຊີວິສ, ສະຖາປັດຕະຍະກຳ, ເອສໂອເອພີ, ເອເອັສພີ*

**ຕິດຕໍ່ພົວພັນ: ຄຳຈັນ ພົມມະລີ; ໂທ: 020 52457714; ອີເມວ: khaamchanh@gmail.com*

A Comparative Analysis of Microservices and Monolithic Architectures in the Development of an IT Helpdesk System Using ASP: A Case Study on Performance Speed and Resource Utilization

Khamchanh PHOMMARY, Atsawin THEPPHAKAN, Khampheth BOUNNADY and Vimontha KHIEOVONGPHACHANH*

Faculty of Engineering, National University of Laos, Laos PDR

Abstract

This research aimed to compare the performance of microservices and monolithic software architectures in developing an IT Helpdesk system using ASP. The comparison focused on three main aspects including response time, memory (RAM) usage, and CPU usage. The evaluation covered four core system tasks: (a) creating a service request, (b) responding to a service request, (c) confirming the service request response, and (d) displaying graphs. Two versions of the IT Helpdesk system were developed: one using a monolithic architecture and the other using a microservices architecture. Performance testing was conducted by simulating workloads with varying volumes of data (100, 1,000, 50,000, and 100,000 requests). Resource usage was monitored using the Resource Monitor tool, and the results were analyzed to calculate averages and compare differences. The study found that in terms of response time, the monolithic architecture outperformed the microservices architecture in all tests, with response times approximately 58% faster on average. In terms of resource usage, the microservices architecture consumed significantly more resources, using about 84% more RAM and 70% more CPU on average, largely due to the overhead of managing multiple independent services. While the monolithic architecture demonstrated superior speed and resource efficiency in small- to medium-sized systems, the microservices architecture offered greater flexibility, scalability, and ease of maintenance, making it more suitable for long-term system growth.

Key words: *Monolith, Microservice, Architecture, SOAP, ASP*

Correspondence: Khamchanh PHOMMARY; Tel: 020 52457714; Email: khaamchanh@gmail.com

1. ພາກສະເໜີ

1.1 ຄວາມເປັນມາ ແລະ ຄວາມສຳຄັນຂອງບັນຫາ

ສະຖາປັດຕະຍະກຳຊອບແວປະກອບເປັນກະດູກສັນຫຼັງຂອງແອບພລິເຄຊັນໃດໆ ໂດຍກຳນົດກອບການງານທາງເຕັກນິກ ແລະ ຄວາມສາມາດໃນການທຳງານຂອງແອບພລິເຄຊັນນັ້ນໆ ສະຖາປັດຕະຍະກຳຊ໊ອບແອມີບົດບາດສຳຄັນໃນການເພີ່ມປະສິດທິພາບຄຸນລັກສະນະທີ່ສຳຄັນເຊັ່ນ: ປະສິດທິພາບ ຄວາມສາມາດໃນການຂະຫຍາຍຂະໜາດ ຄວາມໜ້າເຊື່ອຖື ຄວາມສາມາດໃນການຈັດການ ຄວາມສາມາດໃນການປັບປ່ຽນ ແລະຄວາມສາມາດການນຳໄປໃຊ້ ດັ່ງນັ້ນ ການເລືອກສະຖາປັດຕະຍະກຳທີ່ເໝາະສົມໃນໄລຍະຕົ້ນຂອງການພັດທະນາແມ່ນສຳຄັນຕໍ່ຄວາມສຳເລັດຂອງຊ໊ອບແອ ໃນປັນດາແນວທາງສະຖາປັດຕະຍະກຳຕ່າງໆທີ່ມີຢູ່ ສະຖາປັດຕະຍະກຳ Monolithic ແລະ Microservices ແມ່ນສອງໂມເດລທີ່ໄດ້ຮັບການຍອມຮັບຢ່າງກ້ວ້າງຂວ້າງທີ່ສຸດ ການສຶກສານີ້ສຸມໃສ່ການປຽບທຽບປະສິດທິພາບຂອງສອງສະຖາປັດຕະຍະກຳທັງສອງນີ້ຢູ່ໃນບໍລິບົດຂອງລະບົບ IT Helpdesk.

ໃນຍຸກດິຈິຕອນທີ່ທຸກຢ່າງຂັບເຄື່ອນດ້ວຍເຕັກໂນໂລຊີ, ປະສິດທິພາບຂອງແອັບພລິເຄຊັນໄດ້ກາຍເປັນປັດໄຈສຳຄັນທີ່ສຸດ. ຜູ້ໃຊ້ງານຄາດຫວັງລະບົບທີ່ມີຄວາມໄວ, ໝັ້ນຄົງ ແລະ ເຊື່ອຖືໄດ້. ຫົວໃຈຫຼັກທີ່ກຳນົດຄຸນນະພາບເຫຼົ່ານີ້ກໍຄື ສະຖາປັດຕະຍະກຳຊອບແວ (Software Architecture) ເຊິ່ງເປັນເໝືອນໂຄງສ້າງພື້ນຖານທີ່ກຳນົດທິດທາງການພັດທະນາ ແລະ ຄວາມສາມາດຂອງລະບົບໃນໄລຍະຍາວ. ການເລືອກສະຖາປັດຕະຍະກຳທີ່ເໝາະສົມຕັ້ງແຕ່ເລີ່ມຕົ້ນ ຈຶ່ງເປັນສິ່ງຈຳເປັນຕໍ່ຄວາມສຳເລັດຂອງໂຄງການ.

1) ຄວາມເປັນມາຂອງ Monolithics, Microservices ແລະ IT Help Desk

- ສະຖາປັດຕະຍະກຳແບບ Monolithic: ເປັນຮູບແບບດັ້ງເດີມທີ່ໄດ້ຮັບຄວາມນິຍົມມາຢ່າງຍາວນານໃນການພັດທະນາຊອບແວ. ແນວຄິດຫຼັກຄືການສ້າງທຸກໜ້າທີ່ການເຮັດວຽກ (Functions) ຂອງລະບົບໃຫ້ລວມກັນຢູ່ໃນໂປຣແກຣມດຽວ, ເຊື່ອມຕໍ່ກັນຢ່າງແໜ້ນໜາ ແລະ ໃຊ້ຖານຂໍ້ມູນຮ່ວມກັນ. ວິທີນີ້ເຮັດໃຫ້ການພັດທະນາໃນໄລຍະເລີ່ມຕົ້ນມີຄວາມງ່າຍດາຍ ແລະ ບໍ່ຊັບຊ້ອນ.

- ສະຖາປັດຕະຍະກຳແບບ Microservices: ເປັນແນວຄິດທີ່ເກີດຂຶ້ນມາເພື່ອແກ້ໄຂຂໍ້ຈຳກັດຂອງ Monolithic ໂດຍສະເພາະເມື່ອລະບົບມີຂະໜາດໃຫຍ່ ແລະ ຊັບຊ້ອນຂຶ້ນ. ແນວຄິດນີ້ຄືການແບ່ງລະບົບອອກເປັນໜ່ວຍບໍລິການ (Service)

ຍ່ອຍໆ ທີ່ເຮັດວຽກເປັນອິດສະຫຼະຕໍ່ກັນ, ແຕ່ລະເຊີວິສຮັບຜິດຊອບໜ້າທີ່ສະເພາະເຈາະຈົງ ແລະ ສາມາດມີຖານຂໍ້ມູນຂອງຕົນເອງ. ການສື່ສານລະຫວ່າງເຊີວິສຈະຜ່ານ API (Application Programming Interface) ເຮັດໃຫ້ການພັດທະນາ, ປັບປຸງ ແລະ ຂະຫຍາຍລະບົບມີຄວາມຍືດຫຍຸ່ນສູງ.

- ລະບົບ IT Help Desk ເປັນລະບົບທີ່ສຳຄັນໃນອົງກອນ ເພື່ອເປັນສູນກາງໃນການຮັບແຈ້ງບັນຫາ, ຂໍຄວາມຊ່ວຍເຫຼືອ ແລະ ໃຫ້ບໍລິການດ້ານໄອທີ. ໃນອະດີດ, ລະບົບເຫຼົ່ານີ້ອາດເປັນພຽງລະບົບຮັບແຈ້ງບັນຫາແບບງ່າຍດາຍ ແຕ່ໃນປັດຈຸບັນ ມັນໄດ້ກາຍເປັນເຄື່ອງມືສຳຄັນທີ່ຕ້ອງຮອງຮັບຜູ້ໃຊ້ຈຳນວນຫຼາຍ, ປະມວນຜົນຂໍ້ມູນຢ່າງໄວວາ ແລະ ໃຫ້ບໍລິການທີ່ບໍ່ຕິດຂັດ ເພື່ອຮັກສາປະສິດທິພາບການເຮັດວຽກຂອງອົງກອນ.

2) ຄວາມສຳຄັນຂອງ Monolithics, Microservices ແລະ IT Help Desk

- ຄວາມສຳຄັນຂອງ Monolithic ມີຄວາມສຳຄັນໃນໂຄງການຂະໜາດນ້ອຍຫາປານກາງທີ່ບໍ່ມີຄວາມຊັບຊ້ອນຫຼາຍ. ຈຸດແຂງຂອງມັນຄື ຄວາມງ່າຍດາຍໃນການພັດທະນາ ແລະ ການນຳໄປຕິດຕັ້ງ (Deployment) ໃນເບື້ອງຕົ້ນ ເພາະທຸກຢ່າງລວມຢູ່ໃນທີ່ດຽວ.

- ຄວາມສຳຄັນຂອງ Microservices ມີຄວາມສຳຄັນຢ່າງຍິ່ງສຳລັບລະບົບຂະໜາດໃຫຍ່ທີ່ຕ້ອງການ ຄວາມສາມາດໃນການຂະຫຍາຍໂຕ (Scalability) ຄວາມໜ້າເຊື່ອຖື (Reliability) ແລະ ຄວາມງ່າຍໃນການບຳລຸງຮັກສາ (Maintainability) ເນື່ອງຈາກແຕ່ລະເຊີວິສເປັນອິດສະຫຼະ, ຖ້າມີເຊີວິສໃດໜຶ່ງລົ້ມເຫຼວ ກໍຈະບໍ່ສົ່ງຜົນກະທົບຕໍ່ລະບົບທັງໝົດ. ນອກຈາກນີ້, ທີມພັດທະນາສາມາດອັບເດດ ຫຼື ປັບປຸງແຕ່ລະເຊີວິສໄດ້ໂດຍບໍ່ຕ້ອງຢຸດການເຮັດວຽກຂອງລະບົບທັງໝົດ.

- ຄວາມສຳຄັນຂອງ IT Help Desk ເປັນລະບົບທີ່ສົ່ງຜົນກະທົບໂດຍກົງຕໍ່ປະສິດທິພາບ ແລະ ຄວາມພໍໃຈຂອງຜູ້ໃຊ້ໃນອົງກອນ. ລະບົບ IT Help Desk ທີ່ມີປະສິດທິພາບສູງ ຕອບສະໜອງໄວ ແລະ ຈັດການບັນຫາໄດ້ຢ່າງເປັນລະບົບ ຈະຊ່ວຍຫຼຸດຜ່ອນເວລາທີ່ຜູ້ໃຊ້ຕ້ອງເສຍໄປກັບບັນຫາດ້ານໄອທີ ແລະ ເຮັດໃຫ້ການດຳເນີນງານຂອງທຸລະກິດເປັນໄປຢ່າງຣາບລື່ນ.

ຈາກການສຶກສາຄົ້ນຄວ້າທີ່ຜ່ານມາ ໄດ້ມີການປຽບທຽບສະຖາປັດຕະຍະກຳທັງສອງໃນຫຼາຍໆດ້ານ ຕົວຢ່າງ ເຊັ່ນ: ຄຳໄພ ຄຸນນະວົງສາ (2023) ໄດ້ສຶກສາການນຳໃຊ້ Microservice ເພື່ອຍົກລະດັບ API ພົບວ່າ Microservice ສາມາດເຮັດວຽກໄດ້ໄວກວ່າ Monolithic ຢ່າງຊັດເຈນ, ໂດຍສະເພາະເມື່ອມີຜູ້ໃຊ້ງານຈຳນວນຫຼາຍພ້ອມກັນ, ເຊັ່ນ: ການເຮັດທຸລະກຳ Insert ຂໍ້ມູນ ສາມາດເຮັດໄດ້ໄວກວ່າເຖິງ 93.45%. ຢ່າງໃດກໍຕາມ, ການສຶກສານີ້ຍັງຊີ້ໃຫ້ເຫັນວ່າ Microservice ມີການນຳໃຊ້ຊັບພະຍາກອນຂອງເຊີບເວີຂ້ອນຂ້າງຫຼາຍກວ່າ. Villamizar, et al. (2016) ໄດ້ປຽບທຽບຄ່າໃຊ້ຈ່າຍໃນການນຳໃຊ້ລະບົບເທິງຄລາວ (Cloud) ພົບວ່າ ສະຖາປັດຕະຍະກຳແບບ Microservices ສາມາດຫຼຸດຄ່າໃຊ້ຈ່າຍໄດ້ຢ່າງຫຼວງຫຼາຍເມື່ອທຽບກັບ Monolithic ໂດຍສະເພາະເມື່ອໃຊ້ຮ່ວມກັບເທັກໂນໂລຢີເຊັ່ນ AWS Lambda.

ການສຶກສາເຫຼົ່ານີ້ສະແດງໃຫ້ເຫັນວ່າການເລືອກສະຖາປັດຕະຍະກຳມີຜົນກະທົບທັງດ້ານປະສິດທິພາບ (ຄວາມໄວ) ແລະ ຄ່າໃຊ້ຈ່າຍ.

ເຖິງແມ່ນວ່າຈະມີການຄົ້ນຄວ້າປຽບທຽບລະຫວ່າງ Monolithic ແລະ Microservices ມາແລ້ວ, ແຕ່ການສຶກສາສ່ວນໃຫຍ່ມັກຈະເນັ້ນໃສ່ເທັກໂນໂລຢີ ແລະ ກໍລະນີສຶກສາທີ່ແຕກຕ່າງກັນອອກໄປ ເຊັ່ນ: ການໃຊ້ Java, Spring Framework ຫຼື ການວັດແທກດ້ານຄ່າໃຊ້ຈ່າຍ. ຍັງມີຊ່ອງຫວ່າງຂອງການຄົ້ນຄວ້າທີ່ປຽບທຽບປະສິດທິພາບໂດຍກົງໃນບໍລິບົດຂອງ ການພັດທະນາລະບົບ IT Helpdesk ດ້ວຍພາສາ ASP ເຊິ່ງເປັນເທັກໂນໂລຢີທີ່ຍັງຄົງໄດ້ຮັບຄວາມນິຍົມໃນຫຼາຍອົງກອນ.

ບັນຫາທີ່ເກີດຂຶ້ນຄື ຜູ້ພັດທະນາ ແລະ ຜູ້ຕັດສິນໃຈໃນອົງກອນ ຂາດຂໍ້ມູນເຊິ່ງປະຈັກເພື່ອປະກອບການຕັດສິນໃຈວ່າຄວນເລືອກສະຖາປັດຕະຍະກຳໃດທີ່ເໝາະສົມທີ່ສຸດສຳລັບລະບົບ IT Helpdesk ຂອງຕົນ. ການເລືອກທີ່ຜິດພາດອາດນຳໄປສູ່ບັນຫາດ້ານປະສິດທິພາບ, ຄວາມຫຍຸ້ງຍາກໃນການບຳລຸງຮັກສາ ແລະ ການຂະຫຍາຍລະບົບໃນອະນາຄົດ.

ດັ່ງນັ້ນ, ຈຶ່ງມີຄວາມຈຳເປັນທີ່ຈະຕ້ອງເຮັດບົດຄົ້ນຄວ້ານີ້ ເພື່ອ ປຽບທຽບປະສິດທິພາບລະຫວ່າງ Microservices ແລະ Monolithic ຢ່າງເປັນລະບົບ ໃນກໍລະນີສຶກສາການພັດທະນາລະບົບ IT Helpdesk ດ້ວຍພາສາ ASP ໂດຍເນັ້ນວັດຜົນໃນດ້ານທີ່ສຳຄັນຄື: ຄວາມໄວໃນການຕອບກັບ (Response Time) ແລະ ການນຳໃຊ້ຊັບພະຍາກອນ (CPU ແລະ RAM). ຜົນການຄົ້ນຄວ້າຈະເປັນຂໍ້ມູນອ້າງອີງທີ່ສຳຄັນ ແລະ ເປັນປະໂຫຍດສຳລັບນັກພັດທະນາ, ຜູ້ຈັດການໂຄງການ ແລະ ອົງກອນຕ່າງໆ ໃນການເລືອກສະຖາປັດຕະຍະກຳທີ່ສາມາດຕອບໂຈດທັງດ້ານປະສິດທິພາບ, ການບຳລຸງຮັກສາ ແລະ ການເຕີບໂຕຂອງລະບົບໄດ້ຢ່າງດີທີ່ສຸດ.

1.2. ຄຳຖາມຂອງການຄົ້ນຄ້ວາ

1. ລະຫວ່າງສະຖາປັດຕະຍະກຳ Microservices ແລະ Monolithic ສະຖາປັດຕະຍະກຳໃດມີຄວາມໄວໃນການຕອບກັບ (Response Time) ໄວກວ່າກັນ ໃນການພັດທະນາລະບົບ IT Helpdesk ດ້ວຍພາສາ ASP?

2. ການນຳໃຊ້ຊັບພະຍາກອນໜ່ວຍຄວາມຈຳ (RAM) ຂອງສະຖາປັດຕະຍະກຳ Microservices ແລະ Monolithic ມີຄວາມແຕກຕ່າງກັນຄືແນວໃດ?

3. ສະຖາປັດຕະຍະກຳໃດໃຊ້ຊັບພະຍາກອນໜ້ອຍກວ່າກັນ ໃນສະພາບການຂອງລະບົບ IT Helpdesk ທີ່ພັດທະນາດ້ວຍພາສາ ASP?

1.3. ຈຸດປະສົງຂອງການຄົ້ນຄ້ວາ

1. ເພື່ອປຽບທຽບປະສິດທິພາບດ້ານຄວາມໄວໃນການຕອບກັບ (Response Time) ລະຫວ່າງ Microservices ກັບ Monolithic ຂອງ IT Helpdesk ທີ່ພັດທະນາດ້ວຍພາສາ ASP

2. ເພື່ອປຽບທຽບການນຳໃຊ້ຊັບພະຍາກອນຫນ່ວຍຄວາມຈຳ (RAM) ລະຫວ່າງ Microservices ກັບ Monolithic ຂອງ IT Helpdesk ທີ່ພັດທະນາດ້ວຍພາສາ ASP.

2. ບົດຄົ້ນຄວ້າທີ່ກ່ຽວຂ້ອງ

ຄຳໄພ ຄຸນນະວົງສາ (2023) ໄດ້ສຶກສາການນຳໃຊ້ Microservice ເຂົ້າໃນການຍົກລະດັບການເຮັດວຽກຂອງ API

ຜົນການວິໄຈພົບວ່າ: 1) ເປົ້າໝາຍ ແລະ ເທັກໂນໂລຊີ: ການສຶກສານີ້ມີຈຸດປະສົງເພື່ອປຽບທຽບປະສິດທິພາບຂອງ API ລະຫວ່າງສະຖາປັດຕະຍະກຳແບບ Microservice ແລະ Monolithic ໂດຍໃຊ້ Java, Spring Framework, MySQL ແລະ Docker ໃນການພັດທະນາ ແລະ ຕິດຕັ້ງລະບົບ. 2) ຜົນການປຽບທຽບຄວາມໄວທົ່ວໄປ: ໃນການທົດລອງທົ່ວໄປ, ສະຖາປັດຕະຍະກຳແບບ Microservice ເຮັດວຽກໄດ້ໄວກວ່າ Monolithic ໃນທຸກທຸລະກຳ ເຊັ່ນ: ການຖອນເງິນ (Insert) ໄວກວ່າ 27.59%, ການກວດສອບຍອດເງິນ (Select) ໄວກວ່າ 9.39% ແລະ ການຍົກເລີກທຸລະກຳ (Update) ໄວກວ່າ 12.8%. 3) ຜົນການປຽບທຽບເມື່ອມີຜູ້ໃຊ້ງານຈຳນວນຫຼາຍ: ຈຸດທີ່ແຕກຕ່າງຊັດເຈນທີ່ສຸດແມ່ນເມື່ອມີຜູ້ໃຊ້ງານຈຳນວນຫຼາຍພ້ອມກັນ, Microservice ມີປະສິດທິພາບສູງກວ່າ Monolithic ຢ່າງມະຫາສານ: ການຖອນເງິນໄວກວ່າ 93.45%, ການກວດສອບຍອດເງິນໄວກວ່າ 92.68% ແລະ ການຍົກເລີກທຸລະກຳໄວກວ່າ 83.77%. 4) ສະຖາປັດຕະຍະກຳ Microservice ເໝາະສົມກັບລະບົບຂະໜາດໃຫຍ່ທີ່ມີຜູ້ໃຊ້ງານໜາແໜ້ນ, ໃນຂະນະທີ່ Monolithic ເໝາະກັບລະບົບຂະໜາດນ້ອຍທີ່ມີຜູ້ໃຊ້ບໍ່ຫຼາຍ. ຢ່າງໃດກໍຕາມ, Microservice ໃຊ້ຊັບພະຍາກອນຂອງເຊີບເວີຫຼາຍກວ່າ Monolithic.

Villamizar, Garcés, Ochoa, Castro, Salamanca, Verano, Lang (2016) ໄດ້ສຶກສາ Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservices Architectures ໂດຍມີຈຸດປະສົງເພື່ອສຶກສາ ແລະ ປຽບທຽບປະສິດທິພາບດ້ານຄ່າໃຊ້ຈ່າຍຂອງ 3 ສະຖາປັດຕະຍະກຳທີ່ແຕກຕ່າງກັນ (Monolithic, Microservices, ແລະ AWS Lambda) ເມື່ອໃຊ້ງານເທິງລະບົບ Amazon Web Services (AWS). ຜົນການຄົ້ນຄວ້າພົບວ່າ Microservices ປະຢັດກວ່າ Monolithic ສະຖາປັດຕະຍະກຳແບບ Microservices ສາມາດຊ່ວຍຫຼຸດຄ່າໃຊ້ຈ່າຍລົງໄດ້ປະມານ 9.5% ຫາ 13.8% ເມື່ອທຽບໃສ່ສະຖາປັດຕະຍະກຳແບບ Monolithic. ສ່ວນສະຖາປັດຕະຍະກຳແບບ AWS Lambda (Serverless) ສະແດງໃຫ້ເຫັນການປະຢັດຄ່າໃຊ້ຈ່າຍທີ່ສູງທີ່ສຸດ, ໂດຍສາມາດຫຼຸດຄ່າໃຊ້ຈ່າຍໄດ້ຫຼາຍກວ່າ 50% ແລະ ໃນບາງກໍລະນີສູງເຖິງ 77% ເມື່ອທຽບກັບສະຖາປັດຕະຍະກຳແບບ Monolithic. ບົດຄວາມວິໄຈເລື່ອງ Workload Characterization for Microservices ມີຈຸດປະສົງເພື່ອປຽບທຽບປະສິດທິພາບລະຫວ່າງສະຖາປັດຕະກຳສອງແບບຄື Monolith ແລະ Microservices. ບົດຄົ້ນຄວ້າດັ່ງກ່າວໃຊ້ວິທີການທົດລອງ ໃຊ້ແອັບພລິເຄຊັນຈຳລອງ (Acme Air) ທີ່ພັດທະນາດ້ວຍພາສາ Node.js ແລະ Java ພ້ອມທັງທົດສອບໃນສະພາບແວດລ້ອມທີ່ແຕກຕ່າງກັນ (Bare metal, Docker-Host, Docker-Bridge) ໂດຍໃຊ້ JMeter ເປັນເຄື່ອງມືສ້າງໂຫຼດ. 3) ຜົນສະຫຼຸບດ້ານປະສິດທິພາບ (Throughput) ສະຖາປັດຕະກຳແບບ Microservices ເຮັດໃຫ້ປະສິດທິພາບ (Throughput) ຫຼຸດລົງຢ່າງຫຼວງຫຼາຍເມື່ອທຽບກັບ Monolith (ຫຼຸດລົງ 79.2% ໃນ Node.js ແລະ 70.2% ໃນ Java). 4) ຜົນສະຫຼຸບດ້ານການໃຊ້ງານ CPU: Microservices ມີການໃຊ້ຄຳສັ່ງ CPU (Path length) ຕໍ່ການຮ້ອງຂໍ (Request) ສູງກວ່າແບບ Monolith ເຖິງ 3–3.05 ເທົ່າ, ເຊິ່ງໝາຍເຖິງການໃຊ້ຊັບພະຍາກອນ CPU ຫຼາຍຂຶ້ນ.

ງານວິໄຈນີ້ ໄດ້ນຳສະເໜີການປຽບປະສິດທິພາບຂອງສະຖາປັດຕະກຳແບບ Monolith ແລະ Microservices ໂດຍເຮັດການປຽບທຽບປະສິດທິພາບການເຮັດວຽກລະຫວ່າງ Node.js ແລະ Java ໂດຍການຈຳລອງແອບພລິເຄຊັນ ທີ່ຊື່ວ່າ Acme Air ເຊິ່ງການປຽບທຽບແບ່ງອອກເປັນການຕິດຕັ້ງຊັອບແວຣ໌ແບບ Bare mental Docker-Host Configuration ແລະ Docker-Bridge Configuration ເຊິ່ງການທົດລອງນີ້ໃຊ້ Apache JMeter ໃນການສ້າງໂຫຼດເຂົ້າສູ່ແອບພລິເຄຊັນ.

ຜົນການວິໄຈພົບວ່າ: 1) ຈຸດປະສົງຂອງງານວິໄຈ ເພື່ອປຽບທຽບປະສິດທິພາບຂອງສະຖາປັດຕະກຳແບບ Monolith ແລະ Microservices ທີ່ພັດທະນາດ້ວຍພາສາ Node.js ແລະ Java. 2) ວິທີການທົດລອງ ໃຊ້ແອັບພລິເຄຊັນຈຳລອງຊື່ "Acme Air" ພາຍໃຕ້ການຕິດຕັ້ງ 2 ຮູບແບບ (Bare metal Docker-Host ແລະ Docker-Bridge) ແລະ ໃຊ້ Apache JMeter ເພື່ອສ້າງໂຫຼດໃນການທົດສອບ. 3) ຜົນດ້ານ Throughput ສະຖາປັດຕະກຳ Microservices ເຮັດໃຫ້ Throughput (ປະລິມານການຮອງຮັບວຽກ) ຫຼຸດລົງຢ່າງຊັດເຈນ ເມື່ອທຽບກັບ Monolith, ໂດຍຫຼຸດລົງ 79.2% ສຳລັບ Node.js ແລະ 70.2% ສຳລັບ Java. 4) ຜົນດ້ານ CPU Overhead: Microservices ຕ້ອງການຄຳສັ່ງຈາກ CPU (Path length) ຫຼາຍກວ່າ Monolith ເຖິງ 3 - 3.05 ເທົ່າຕໍ່ໜຶ່ງ Request, ເຊິ່ງສະແດງເຖິງການໃຊ້ງານ CPU ທີ່ໜັກກວ່າ.ດຍງານວິໄຈນີ້ ສະຫຼຸບຜົນວ່າການນຳສະຖາປັດຕະກຳ Microservices ມາໃຊ້ເຊິ່ງພັດທະນາໂດຍ ພາສາ Node.js ແລະ Java ນັ້ນສິ່ງຜົນໃຫ້ Throughput ລົດລົງເຖິງ 79.2% ແລະ 70.2% ຕາມລຳດັບແລະ ຍິ່ງໄປກວ່ານັ້ນ Microservices ມີ Path length ຫຼືຈຳນວນຄັ້ງທີ່ CPU ສັ່ງການຫຼາຍກວ່າ Monolith 3-3.05 Times/Request (Ueda, Nakaike, & Ohara, 2016).

ບົດຄວາມວິໄຈເລື່ອງ Leveraging Microservices architecture by using Docker technology ຜົນການວິໄຈ

ພົບວ່າ: 1) ເພີ່ມຄວາມໄວໃນການເຮັດວຽກອັດຕະໂນມັດ (Accelerate Automation) Docker ເໝາະສົມກັບການສ້າງ ຂະບວນການອັດຕະໂນມັດ ເຊັ່ນ ການທົດສອບ ຫຼື ການສົ່ງມອບ ແອັບພລິເຄຊັນ ໃຫ້ໄວ ແລະ ມີປະສິດທິພາບ. 2) ເພີ່ມຄວາມ ເປັນອິດສະຫຼະ (Accelerate Independency) Container ມີ ສະພາບແວດລ້ອມທີ່ແຍກອອກຈາກກັນຢ່າງຊັດເຈນ, ຊ່ວຍໃຫ້ ແຕ່ລະທີມສາມາດພັດທະນາ ແລະ ຈັດການແຕ່ລະສ່ວນຂອງ ລະບົບ (service) ໄດ້ຢ່າງອິດສະຫຼະ ໂດຍບໍ່ກະທົບກັນ. 3) ສະດວກໃນການເຄື່ອນຍ້າຍ (Accelerate Portability) ແອັບພ ລິເຄຊັນທີ່ຢູ່ໃນ Container ສາມາດນຳໄປໃຊ້ງານໄດ້ໃນທຸກ ສະພາບແວດລ້ອມ (ເຊັ່ນ ເຄື່ອງຂອງນັກພັດທະນາ, ເຊີບເວີທົດ ສອບ) ໂດຍບໍ່ເກີດບັນຫາຄວາມເຂົ້າກັນໄດ້. 4) ໃຊ້ຊັບພະຍາກອນ ໄດ້ຢ່າງຄຸ້ມຄ່າ (Accelerate Resource Utilization): Docker ໃຊ້ຊັບພະຍາກອນ (ເຊັ່ນ CPU, RAM) ໜ້ອຍກວ່າການໃຊ້ Virtual Machine (VM) ແບບດັ້ງເດີມ, ເຮັດໃຫ້ສາມາດສ້າງ ແລະ ໃຊ້ງານຫຼາຍໆບໍລິການ (services) ພ້ອມກັນໄດ້ໃນເຄື່ອງ ດຽວ.

ສະຫຼຸບວ່າ Microservices ມີທັງຂໍ້ດີ ແລະ ຂໍ້ຈຳກັດ. ມັນເ ໝາະສົມກັບ ລະບົບຂະໜາດໃຫຍ່ທີ່ມີຜູ້ໃຊ້ໝາແໜ້ນ ໂດຍ ສາມາດເພີ່ມຄວາມໄວ ແລະ ປະຢັດຄ່າໃຊ້ຈ່າຍໄດ້. ແຕ່ໃນທາງ ກັບກັນ, ມັນອາດເຮັດໃຫ້ ປະສິດທິພາບ (Throughput) ຫຼຸດລົງ ແລະ ໃຊ້ຊັບພະຍາກອນ (CPU) ຫຼາຍຂຶ້ນ ໃນບາງສະຖານະ ການ. ໃນຂະນະທີ່ Monolithic ເໝາະກັບ ລະບົບຂະໜາດນ້ອຍ ທີ່ມີຜູ້ໃຊ້ບໍ່ຫຼາຍ. ການເລືອກໃຊ້ຈຶ່ງຂຶ້ນກັບຄວາມຕ້ອງການຂອງ ໂຄງການເປັນຫຼັກ.

3. ວິທີດຳເນີນການຄົ້ນຄວ້າ

3.1 ການອອກແບບການຄົ້ນຄວ້າ

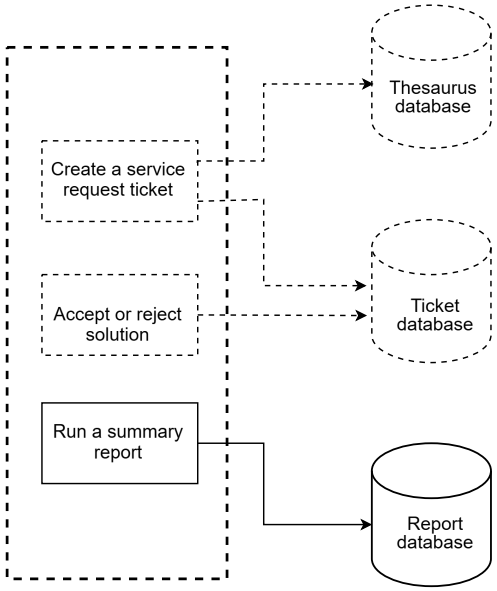
ງານວິໄຈນີ້ປັນການຄົ້ນຄວ້າປຽບທຽບການນຳໃຊ້ Microservices ກັບ Monolithic ຂອງ IT Helpdesk ທີ່ ພັດທະນາດ້ວຍ ພາສາ ASP ໃນສ້າງເຊີວິສຣີເຄວສ, ການຕອບເຊີ ວິສຣີເຄວສ, ຢືນຢັນຄຳຕອບເຊີວິສຣີເຄວສ, ການສະແດງກຣາຟ ໂດຍ ສະຖາປັດຕະຍະກຳທັງສອງ ແມ່ນຈະຖືກພັດທະນາດ້ວຍ ພາສາ ASP ຕິດຕັ້ງຢູ່ເທິງ Server ໂດຍການສຶກສາປຽບທຽບໃຫ້ ເຫັນຄວາມແຕກຕ່າງທາງດ້ານການຈັດການຂໍ້ມູນລະຫວ່າງ ໃຊ້ Microservices ກັບ Monolithic ຂອງ IT Helpdesk.

ການດຳເນີນງານທົດລອງແມ່ນເນັ້ນຈະນຳສະເໜີການ ອອກແບບ ແລະ ການພັດທະນາລະບົບ IT Helpdesk ພາຍໃຕ້ ການອອກແບບດ້ວຍສະຖາປັດຕະຍະກຳ Microservices ແບ່ງ ອອກໄດ້ເປັນ 5 ຂັ້ນຕອນ 1)ອອກແບບລະບົບ IT Helpdesk ພາຍໃຕ້ສະຖາປັດຕະຍະກຳ Microservices 2) ພັດທະນາໜ້າ ຈໍ ແລະ ຖານຂໍ້ມູນຂອງ IT Helpdesk ເພື່ອໃຫ້ຜູ້ໃຊ້ງານ ສາມາດສ້າງ ແລະ ເບິ່ງຂໍ້ມູນຂອງເຊີວິສຣີເຄວສ 3) ອອກແບບ ແລະ ພັດທະນາຖານຂໍ້ມູນຂອງຄຳສັບສຳພັນເພື່ອໃຫ້ລະບົບ ສາມາດແບ່ງແຍກໝວດໝູ່ຂອງເຊີວິສຣີເຄວສ 4) ພັດທະນາເຊີ ວິສ classification service, ticket service, feedback ticket service ໃຫ້ສາມາດຮັບຄ່າ ແລະ ສົ່ງຄ່າຜ່ານ API Gateway 5) ພັດທະນາສ່ວນຕໍ່ຂະຫຍາຍຂອງ analysis service ເພື່ອນຳມາເປັນສ່ວນຂະຫຍາຍຂອງລະບົບ IT Helpdesk 6) ທົດລອງໃຊ້ວຽກຈິງໃນການສ້າງເຊີວິສຣີເຄວສ ແລະ ການຈັດໝວດໝູ່ຄຳສັບສຳພັນ 7) ທົດລອງເພີ່ມ analysis service ໃນຂະນະທີ່ລະບົບ IT Helpdesk ກຳລັງເຮັດວຽກ.

ຫຼັງຈາກທີ່ໄດ້ຮັບຜົນການທົດລອງ ຜົນໄດ້ຮັບລະຫວ່າງ Microservices ກັບ Monolithic ຂອງ IT Helpdesk ເຮົາກໍ່ ຈະມີການປຽບທຽບຜົນດັ່ງກ່າວໃນສາມຮູບແບບຄື: ຄວາມໄວໃນ ການຕອບກັບຂໍ້ມູນ (Response Time), ການນຳໃຊ້ໜ່ວຍ ຄວາມຈຳຊົ່ວຄາວ (RAM), ການນຳໃຊ້ໜ່ວຍປະມວນຜົນກາງ (CPU).

3.1.1 ອອກແບບລະບົບ IT Helpdesk ພາຍໃຕ້ສະຖາປັດຕະ ຍະກຳ Monolithic

ການອອກແບບລະບົບ IT Helpdesk ນັ້ນຈະອອກແບບໜ້າ ຕາຕິດຕໍ່ແລ້ວເຂົ້າຖານຂໍ້ມູນເລີຍໂດຍທົ່ວໄປຈະໃຊ້ຖານຂໍ້ມູນໂຕ ດຽວແຕ່ໃນທີ່ນີ້ເພື່ອການປຽບທຽບຈະໃຊ້ຖານຂໍ້ມູນ 3 ຖານ ດັ່ງ ຮູບທີ 1.

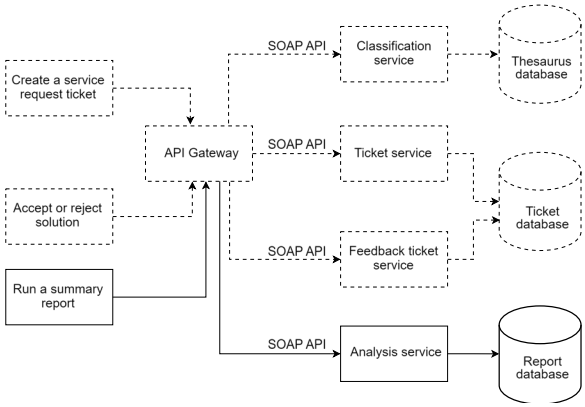


ຮູບທີ 1 ສະຖາປັດຕະຍະກຳ IT helpdesk ແບບ Monolithic Architecture

ການອອກແບບ Monolithicຈະປະກອບດ້ວຍຫຼາຍຟັງ ຊັນເຊິ່ງແຕກຕ່າງກັບ Microservices ປະກອບດ້ວຍຫຼາຍເຊີວິສ.

3.1.2 ອອກແບບລະບົບ IT Helpdesk ພາຍໃຕ້ສະຖາປັດຕະ ຍະກຳ Microservices

ການອອກແບບລະບົບ IT Helpdesk ນັ້ນຈະແບ່ງ ອອກເປັນແຕ່ລະເຊີວິສ ໂດຍແຍກຕາມລັກສະນະການດຳເນີນງານ ດັ່ງຮູບທີ 2 ທີ່ສະແດງເຖິງຮູບແບບຄວາມສຳພັນຂອງແຕ່ລະຟັງ ຊັນງານ ແຕ່ລະເຊີວິສຈະຕິດຕໍ່ກັນຜ່ານທາງ API Gateway.



ຮູບທີ 2 ສະຖາປັດຕະຍະກຳ IT helpdesk ແບບ Microservices Architecture

ໂດຍໃນລະບົບ IT helpdesk ສາມາດແບ່ງອອກເປັນ 4 ເຊີວິສຕາມຟັງຊັນການເຮັດວຽກ ດັ່ງຕໍ່ໄປນີ້:

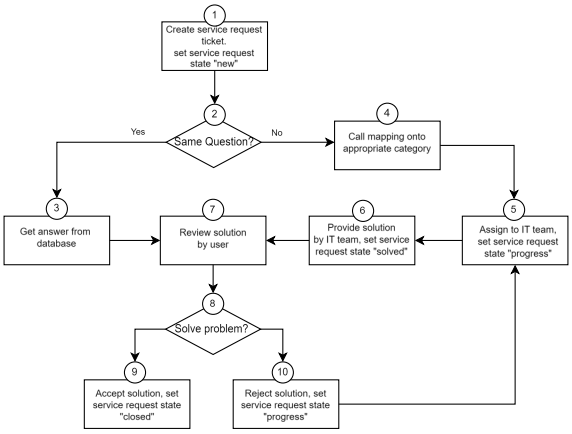
1. Classification service ເມື່ອຜູ້ໃຊ້ງານທຳການສ້າງເຊີວິສຣິເຄວສ ເຊີວິສຈະຖືກຈຳແນກເຊີວິສຣິເຄວສໄປຕາມໝວດໝູ່ຂອງງານຕ່າງໆທີ່ຖືກຕ້ອງ ໂດຍແບ່ງໝວດໝູ່ຕາມຖານຂໍ້ມູນຄຳສັບສຳພັນ.

2. Ticket service ເມື່ອເຊີວິສຣິເຄວສ ຖືກແຍກຕາມໝວດໝູ່ຂອງງານຕ່າງໆຮຽບຮ້ອຍແລ້ວ ທາງລະບົບຈະທຳການເອີ້ນເຊີວິສ ເພື່ອບັນທຶກຂໍ້ມູນໄປທີ່ຖານຂໍ້ມູນເຊີວິສຣິເຄວສ.

3. Feedback ticket service ເມື່ອທາງໜ່ວຍງານໄອທີທີ່ໄດ້ທຳການແກ້ໄຂບັນຫາຂອງແຕ່ລະເຊີວິສຣິເຄວສນັ້ນໆ ທາງຜູ້ໃຊ້ງານຕ້ອງທຳການຍອມຮັບຈາກ (ຄັ້ງ) ຫຼື ປະຕິເສດ (reject) ວ່າວິທີການແກ້ໄຂບັນຫາທີ່ທາງຫນ່ວຍງານໄອທີສະເໜີສາມາດແກ້ໄຂເຊີວິສຣິເຄວສໄດ້ແທ້ ຫຼື ບໍ່ ໂດຍຫຼັງຈາກຍອມຮັບ ຫຼື ປະຕິເສດການແກ້ໄຂເຊີວິສຣິເຄວສ ທາງລະບົບຈະທຳການເອີ້ນ API ເພື່ອໃຊ້ວຽກ Feedback ticket service ເກັບຄ່າຍອມຮັບ ຫຼື ປະຕິເສດເຊີວິສຣິເຄວສນັ້ນໆ.

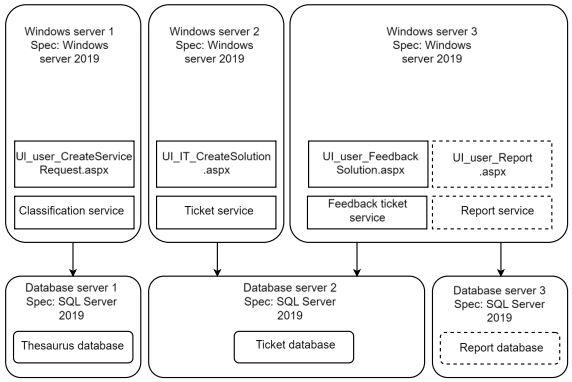
4. Analysis service ເມື່ອທາງຜູ້ໃຊ້ວຽກທຳການເອີ້ນຂໍ້ມູນຂອງລາຍງານກ່ຽວກັບລະຍະເວລາການຕອບສະໜອງຂອງໜ່ວຍງານໄອທີ ແລະ ສັດສ່ວນຂອງຈຳນວນເຊີວິສຣິເຄວສໃນແຕ່ລະໝວດໝູ່ ທາງລະບົບຈະທຳການຕິດຕໍ່ຫາ API ເພື່ອເອີ້ນໃຊ້ເຊີວິສໃນການສະແດງຜົນຂອງບົດລາຍງານ.

ໂດຍເຊີວິສຣິເຄວສຈະມີລຳດັບຂອງວຽກຄື ເລີ່ມຕົ້ນຈາກຜູ້ໃຊ້ງານທຳການສ້າງໃບເຊີວິສຣິເຄວສຂຶ້ນ ຜ່ານລະບົບ IT Helpdesk ເຊິ່ງໃນການສ້າງໃບເຊີວິສຣິເຄວສຈະມີຖານະເປັນ New ທາງລະບົບຈະນຳໃບເຊີວິສຣິເຄວສນັ້ນ ສົ່ງຕໍ່ໄປຫາ classification service ເພື່ອໃຫ້ລະບົບທຳການໝວດໝູ່ ຫຼັງຈາກລະບົບໄດ້ທຳການແຍກໝວດໝູ່ຂອງເຊີວິສຣິເຄວສແລ້ວ ລະບົບຈະເອີ້ນ ticket service ເພື່ອບັນທຶກຂໍ້ມູນຂອງເຊີວິສຣິເຄວສນັ້ນໆ ລວມເຖິງໝວດໝູ່ທີ່ທາງລະບົບໄດ້ທຳການວິເຄາະມາໃຫ້ ແລະ ສົ່ງໄປຍັງໜ່ວຍງານໄອທີ ທີ່ຮັບຜິດຊອບດ້ວຍສະຖານະ progress ທາງໜ່ວຍງານໄອທີຈະທຳການຫາວິທີແກ້ໄຂບັນຫາໃຫ້ກັບຜູ້ໃຊ້ງານ ໂດຍບັນທຶກວິທີການແກ້ໄຂປັນຫາລົງໄປຍັງເຊີວິສຣິເຄວສນັ້ນໆຜ່ານ feedback ticket service ດ້ວຍສະຖານະ solved ແລະ ທາງຜູ້ໃຊ້ງານຈະເປັນຄົນກວດສອບວ່າວິທີແກ້ໄຂບັນຫາທີທາງໜ່ວຍງານພະແນກໄອທີ ຕອບກັບມາສາມາດແກ້ໄຂບັນຫາໄດ້ ຫຼື ບໍ່ ຜ່ານທາງ feedback ticket service ໂດຍຍອມຮັບວິທີການແກ້ໄຂບັນຫາ ຖ້າວິທີການແກ້ໄຂບັນຫາໃຊ້ວຽກໄດ້ ທາງລະບົບຈະທຳການປ່ຽນສະຖານະຂອງເຊີວິສຣິເຄວສເປັນ closed ເພື່ອຈົບໃບເຊີວິສຣິເຄວສ ຫຼື ປະຕິເສດວິທີການແກ້ໄຂໃນກໍລະນີທີ່ບໍ່ສາມາດນຳໄປແກ້ໄຂບັນຫາໄດ້ ເຊິ່ງຖ້າຜູ້ໃຊ້ງານປະຕິເສດວິທີການແກ້ໄຂບັນຫາທາງລະບົບຈະສົ່ງໃບເຊີວິສຣິເຄວສກັບໄປໃຫ້ກັບທາງໜ່ວຍງານໄອທີດ້ວຍສະຖານະ progress ໃໝ່ອີກເທື່ອຫນຶ່ງເພື່ອໃຫ້ກວດສອບ ແລະ ສົ່ງວິທີແກ້ໄຂປັນຫາກັບມາໃຫມ່ ດັ່ງຮູບທີ 3.



ຮູບທີ 3 ລຳດັບ ແລະ ສະຖານະຂອງເຊີວິສຣິເຄວສ

ການຕິດຕັ້ງເຊີເວີເພື່ອການວັດປະສິດທິພາບດັ່ງຮູບທີ 4.



ຮູບທີ 4 ການຕິດຕັ້ງລະບົບ IT Helpdesk ຕາມສະຖາປັດຕະຍະກຳ Microservices

1. Windows server 1 ຕິດຕັ້ງໜ້າຈໍເຊີວິສຣິເຄວສສຳຫຼັບຜູ້ໃຊ້ງານ UI_user_CreateServiceRequest.aspx ແລະ Classification service.

2. Windows server 2 ຕິດຕັ້ງໜ້າຈໍວິທີການແກ້ໄຂເຊີວິສຣິເຄວສສຳຫຼັບໜ່ວຍງານໄອທີ UI_IT_CreateSolution.aspx ແລະ Ticket service.

3. Windows server 3 ຕິດຕັ້ງໜ້າຈໍສະແດງວິທີການຢືນຢັນຄຳຕອບການແກ້ໄຂເຊີວິສຣິເຄວສສຳຫຼັບຜູ້ໃຊ້ງານ UI_user_FeedbackSolution.aspx ແລະ Feedback ticket

service ແລະ ນຳກຣາຟມາສະແດງ UI_user_Report.aspx ແລະ Report service.

3.1.3 ການອອກແບບສະຖາປັດຕະຍະກຳ

ງານວິໄຈນີ້ແມ່ນແນ່ເນັ້ນໃສ່ການສະເຫນີລະບົບ IT Helpdesk ທີ່ພັດທະນາ ແລະ ອອກແບບພາຍໃຕ້ສະຖາປັດຕະຍະກຳ Microservices ພ້ອມທັ້ງການນຳໃຊ້ເຕັກນິກຄຳສັບສຳພັນມາໃຊ້ ເພື່ອຈັດໝວດໝູ່ຂອງເຊີວິສຣິເຄວສ ໂດຍລະບົບ IT Helpdesk ຈະມີສ່ວນປະກອບຫຼັກຄື ສ່ວນໜ້າຈໍສະແດງຜົນ ສອນເອພີໄອ ແລະ ເຊີວິສ ສ່ວນຂອງຖານຂໍ້ມູນ ເຊິ່ງແຕ່ລະສ່ວນຈະມີການພັດທະນາທີ່ແຍກອອກຈາກັນ.

3.1.4 ການອອກແບບໜ້າຈໍຕິດຕໍກັບຜູ້ໃຊ້

ໃນກະບວນການອອກແບບປະກອບດ້ວຍ ໜ້າຈໍການສ້າງເຊີວິສຣິເຄວສ, ໜ້າຈໍວິທີແກ້ໄຂເຊີວິສຣິເຄວສ ສຳລັບໜ່ວຍງານໄອທີ, ໜ້າຈໍສະແດງຢືນຢັນຄຳຕອບແກ້ໄຂເຊີວິສຣິເຄວສ ຕອບໂດຍຜູ້ໃຊ້ອງກ ແລະ ໜ້າຈໍສະແດງກຣາຟບົດລາຍງານຂອງເຊີວິສຣິເຄວສ

3.1.5 ສ່ວນເອພີໄອແລະເຊີວິສ

Classification service ເປັນເວັບເຊີວິສທີ່ພາຍໃນປະກອບດ້ວຍຟັງຊັນ ClassificationCategory ເຊິ່ງເປັນຟັງຊັນທີ່ໃຊ້ໃນການຈັດໝວດໝູ່ຂອງເຊີວິສຣິເຄວສ ໂດຍຟັງຊັນນີ້ຖືກຮຽກໃຊ້ຈາກໜ້າເຊີວິສຣິເຄວສສຳຫຼັບຜູ້ໃຊ້ງານ.

Ticket service ເປັນເວັບເຊີວິສທີ່ປະກອບດ້ວຍຟັງຊັນ CreateService ເພື່ອໃຊ້ໃຫ້ໃນການບັນທຶກຂໍ້ມູນຂອງເຊີວິສຣິເຄວສເຂົ້າໃນຖານຂໍ້ມູນ ໂດຍການເຊື່ອມຕໍ່ກັບໜ້າຈໍເຊີວິສຣິເຄວສສຳລັບຜູ້ໃຊ້ງານ ໂດຍເຊີວິສນີ້ຈະເຮັດຫຼັງຈາກເຊີວິສ classification service ໄດ້ທຳການຄືນຄ່າໝວດໝູ່ ແລະ ໜ່ວຍງານທີ່ຮັບຜິດຊອບຂອງເຊີວິສ.

Feedback ticket service ເປັນເຊີວິສທີ່ຮວບຮວມຟັງຊັນກ່ຽວກັບການແກ້ໄຂປັນຫາທັງໝົດຂອງລະບົບໂດຍປະກອບດ້ວຍຟັງຊັນ ServiceDetail, AcceptService, RejectService ແລະ CreateTicketSolution ໂດຍໜ້າຈໍທີ່ເອີ້ນໃຊ້ເວັບເຊີວິສນີ້ແມ່ນໜ້າຈໍວິທີແກ້ໄຂເຊີວິສຣິເຄວສ ສຳລັບໜ່ວຍງານໄອທີ ແລະ ໜ້າຈໍຢືນຢັຍຄຳຕອບສຳຫຼັບຜູ້ໃຊ້.

Analysis service ເປັນເວັບເຊີວິສທີ່ປະກອບດ້ວຍ ReportPerformanceByResponseTime ແ ລ ະ ReportTicketByCategory ໂດຍທັງຄູ່ບໍ່ຕ້ອງຮັບຄ່າເພື່ອນຳໄປປະມວນຜົນ.

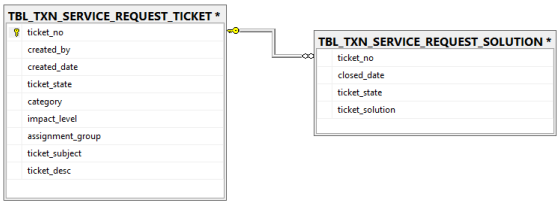
3.1.6 ສ່ວນຂອງຖານຂໍ້ມູນ

ໃນລະບົບ IT Helpdesk ຈະແບ່ງຖານຂໍ້ມູນອອກເປັນ 3 ຖານຂໍ້ມູນ ໄດ້ແກ່ thesaurus database, ticket database, report database ໂດຍຖານຂໍ້ມູນ Thesaurus ຈະມີຕາຕະລາງ TBL_MST_THESAURUS ເພື່ອເກັບຂໍ້ມູນຂອງຄຳສັບສຳພັນ ແລະ ເພື່ອໃຊ້ໃນການແບ່ງໝວດໝູ່ຂອງເຊີວິສຣິເຄວສ ຈຶ່ງມີການປັບປ່ຽນຖັນເພື່ອໃຫ້ເຂົ້າກັບລະບົບ IT Helpdesk ໂດຍຂໍ້ມູນຕາມຕະຕະລາງທີ 1.

ຕາຕະລາງທີ 1 ໂຄງສ້າງຕະຕະລາງ TBL_MST_THESAURUS

Column	Data description
type	ແບ່ງເປັນ subject, description
category	ປະເພດຂອງໝວດໝູ່ເຊີວິສຣິເຄວສ
keyword	ກຸ່ມສັບຄຳທີ່ມີຄວາມກ່ຽວຂ້ອງກັບໝວດໝູ່ງານນັ້ນໆ
Assign_group	ໜ່ວຍງານທີ່ຮັບຜິດຊອບໝວດໝູ່ງານນັ້ນໆ

ຖານຂໍ້ມູນ ticket database ມີ 2 ຕະຕະລາງ ຄື TBL_TXN_SERVICE_REQUEST_TICKET ໃຊ້ເກັບຂໍ້ມູນຂອງເຊີວິສຣິເຄວສ ແລະ TBL_TXN_SERVICE_REQUEST_SOLUTION ເພື່ອໃຊ້ເກັບວິທີການແກ້ໄຂປັນຫາ ໂດຍ 2 ຕະຕະລາງນີ້ມີຄວາມສຳພັນກັນດ້ວຍໝາຍເລກເຊີວິສຣິເຄວສ ເພື່ອໃຊ້ອ້າງອີງໃນການນຳຂໍ້ມູນຈາກ 2 ຕະຕະລາງມາລວມກັນເວລາສະແດງຂໍ້ມູນຜ່ານໜ້າຈໍ ດັ່ງຮູບທີ 5.



ຮູບທີ 5 ໂຄງສ້າງຄວາມສຳພັນຂອງຖານຂໍ້ມູນ ticket database

ໃນຕະຕະລາງຂອງ TBL_TXN_SERVICE_REQUEST_TICKET ຈະເກັບລວບລວມຂໍ້ມູນທີ່ກ່ຽວກັບເຊີວິສຣິເຄວສ ສ່ວນທີ່ມາຈາກຜູ້ໃຊ້ງານທັງໝົດ ໄດ້ແກ່ໝາຍເລກເຊີວິສຣິເຄວສ (ticket_no), ຜູ້ໃຊ້ງານທີ່ທຳການສ້າງເຊີວິສຣິເຄວສ (created_by), ວັນທີສ້າງເຊີວິສຣິເຄວສ (created_date), ສະຖານະຂອງເຊີວິສຣິເຄວສ (ticket_state), ໝວດໝູ່ຂອງເຊີວິສຣິເຄວສ (category), ລະດັບຜົນກະທົບຂອງເຊີວິສຣິເຄວສທີ່ມີຕໍ່ລະບົບ (impact_level), ໜ່ວຍງານໄອທີທີ່ເບິ່ງແຍງໝວດໝູ່ງານນັ້ນໆ(assignment_group), ຫົວຂໍ້ຂອງເຊີວິສຣິເຄວສ (ticket_subject) ແລະ ລາຍລະອຽດຂອງເຊີວິສຣິເຄວສ (ticket_desc) ສ່ວນຕາຕະລາງຂອງTBL_TXN_SERVICE_ REQUEST_SOLUTION ຈະເກັບຂໍ້ມູນທີ່ກ່ຽວກັບວິທີການແກ້ໄຂປັນຫາຈາກທາງໜ່ວຍງານໄອທີທັງໝົດ ໄດ້ແກ່ໝາຍເລກເຊີວິສຣິເຄວສ (ticket_no), ວັນທີໜ່ວຍງານໄອທີທຳການແກ້ໄຂປັນຫາ (closed_date), ສະຖານະຂອງເຊີວິສຣິເຄວສ (ticket_state) ແລະ ວິທີການແກ້ໄຂປັນຫາ (ticket_solution) ຖານຂໍ້ມູນ report database ມີ 1 ຕະຕະລາງ ຄື ຕະຕະລາງTBL_TXN_SERVICE _REQUEST _HISTORY ໂດຍນຳເອົາຖັນຂອງຕະຕະລາງໃນ ticket database ໄດ້ແກ່ TBL_TXN_SERVICE_ REQUEST_ TICKET ມາລວມກັບຕະຕະລາງ TBL_TX N_ SERVI

CE_REQUEST_SOLUTION ເພື່ອທຳລາຍງານສະຫຼຸບ ເວລາການຕອບສະໜອງຕໍ່ເຊີວິສຣິເຄວສ ແລະ ລາຍງານສະຫຼຸບ ຜົນການປຽບທຽບຈຳນວນຂອງເຊີວິສຣິເຄວສໃນລະບົບ.

3.2 ເຄື່ອງມືທີ່ໃຊ້ໃນການເກັບກຳຂໍ້ມູນ

ໃນການທົດລອງເພື່ອເກັບຂໍ້ມູນຜູ້ທົດລອງເອງແມ່ນໄດ້ ຂຽນຄຳສັ່ງທີ່ເປັນພາສາ ASP ສ້າງເວັບເຊີວິສ ແລະ SQL Server ເປັນຖານຂໍ້ມູນ ເພື່ອເປັນເຄື່ອງມືໃນການທົດສອບ (Front End) ໂດຍຂຽນໃນແບບ Monolithic ແລະ Microservices ຂອງ IT HelpDesk ແລະ (Back End) ແລະ ໃຊ້ຄຳສັ່ງໃນASPເພື່ອອັດຄວາມໄວການຕອບສະໜອງ, ໃຊ້ Resource Monitor*(resmon.exe)* ອັດການນຳໃຊ້ ຊັບພະຍາກອນ RAM, CPU.

3.2.1 Resource Monitor (resmon.exe)

ເປັນເຄື່ອງມືທີ່ມີປະສິດທິພາບສູງທີ່ຕິດມາພ້ອມກັບລະບົບ ປະຕິບັດການ Windows. ມັນຊ່ວຍໃຫ້ຜູ້ໃຊ້ສາມາດຕິດຕາມ ການນຳໃຊ້ຊັບພະຍາກອນຂອງຄອມພິວເຕີໄດ້ແບບລະອຽດ ແລະ ທັນທີ (real-time) ເຊິ່ງໃຫ້ຂໍ້ມູນທີ່ເລິກເຊິ່ງກວ່າ Task Manager.

1. ການນຳໃຊ້ເພື່ອຕິດຕາມ CPU (Central Processing Unit) ໃນແຖບ CPU, Resource Monitor ສະແດງຂໍ້ມູນສຳຄັນດັ່ງນີ້: ກຣາຟການໃຊ້ງານ CPU ລວມ: ສະແດງເປີເຊັນການໃຊ້ງານ CPU ທັງໝົດໃນປັດຈຸບັນ ແລະ ຍ້ອນຫຼັງໄປປະມານ 60 ວິນາທີ.

- ລາຍຊື່ໂປຣແກຣມ (Processes): ສະແດງລາຍຊື່ຂອງທຸກ ໂປຣແກຣມ ແລະ ເຊີວິດທີ່ກຳລັງເຮັດວຽກຢູ່.
- ການໃຊ້ງານ CPU ຂອງແຕ່ລະໂປຣແກຣມ: ສິ່ງທີ່ສຳຄັນທີ່ ສຸດຄື ມັນບອກວ່າແຕ່ລະໂປຣແກຣມກຳລັງໃຊ້ CPU ຢູ່ຈັກ ເປີເຊັນ. ປະໂຫຍດຫຼັກ ກໍຄື ຖ້າຄອມພິວເຕີຊ້າ ຫຼື ຄ້າງ, ເຮົາ ສາມາດເຂົ້າມາເບິ່ງໄດ້ທັນທີວ່າໂປຣແກຣມໃດກຳລັງໃຊ້ CPU ໜັກຜິດປົກກະຕິ ແລະ ເປັນສາເຫດຂອງບັນຫາ.

2. ການນຳໃຊ້ເພື່ອຕິດຕາມ RAM (Random Access Memory)

ໃນແຖບ Memory, Resource Monitor ໃຫ້ຂໍ້ມູນທີ່ຊ່ວຍໃຫ້ ເຂົ້າໃຈການໃຊ້ໜ່ວຍຄວາມຈຳໄດ້ດີຂຶ້ນ:

• ກຣາຟສະແດງສະຖານະຂອງ RAM: ແບ່ງໜ່ວຍຄວາມຈຳ ອອກເປັນສ່ວນຕ່າງໆ:

- In Use (ກຳລັງໃຊ້ງານ): RAM ທີ່ກຳລັງຖືກໃຊ້ໂດຍໂປຣ ແກຣມ, ລະບົບ, ແລະ ໄດຣເວີ.
- Standby (ລໍຖ້າໃຊ້ງານ): RAM ສ່ວນນີ້ເກັບຂໍ້ມູນທີ່ເຄີຍ ໃຊ້ແລ້ວໄວ້ຊົ່ວຄາວ. ຖ້າໂປຣແກຣມຕ້ອງການຂໍ້ມູນນັ້ນອີກ, ມັນ ຈະຖືກເອີ້ນໃຊ້ໄດ້ໄວຂຶ້ນ. ແຕ່ຖ້າມີໂປຣແກຣມໃໝ່ຕ້ອງການ RAM ສ່ວນນີ້ກໍພ້ອມທີ່ຈະຖືກນຳໄປໃຊ້ທັນທີ.
- Free (ວ່າງ): RAM ທີ່ບໍ່ໄດ້ຖືກໃຊ້ງານເລີຍ.

• Hard Faults/sec: ເປັນຄ່າທີ່ສຳຄັນຫຼາຍ. ມັນໝາຍເຖິງ ຈຳນວນຄັ້ງຕໍ່ວິນາທີທີ່ລະບົບຕ້ອງໄປດຶງຂໍ້ມູນຈາກຮາດດິດ (Page File) ເພາະຂໍ້ມູນນັ້ນບໍ່ໄດ້ຢູ່ໃນ RAM. ຖ້າຄ່ານີ້ສູງ ຕະຫຼອດເວລາ, ມັນເປັນສັນຍານວ່າຄອມພິວເຕີກຳລັງຂາດ RAM ແລະ ເປັນສາເຫດຫຼັກທີ່ເຮັດໃຫ້ເຄື່ອງຊ້າ.

• ລາຍຊື່ໂປຣແກຣມ ແລະ ປະລິມານ RAM ທີ່ໃຊ້: ສະແດງໃຫ້ ເຫັນວ່າແຕ່ລະໂປຣແກຣມໃຊ້ RAM ຫຼາຍປານໃດ, ຊ່ວຍໃຫ້ ຊອກຫາໂປຣແກຣມທີ່ກິນ RAM ຫຼາຍເກີນໄປໄດ້.

• ຜູ້ສ້າງ (Creator): ບໍລິສັດ ໄມໂຄຣຊອບ (Microsoft Corporation).

• Resource Monitor ເປັນເຄື່ອງມືທີ່ຖືກພັດທະນາຂຶ້ນໂດຍ Microsoft ແລະ ຖືກລວມເຂົ້າມາເປັນສ່ວນໜຶ່ງຂອງລະບົບ ປະຕິບັດການ Windows ເລີ່ມຕັ້ງແຕ່ລຸ້ນ Windows Vista (ເປີດໂຕປີ 2007) ເປັນຕົ້ນມາ ແລະ ມີຢູ່ໃນ Windows ທຸກລຸ້ນ ຫຼັງຈາກນັ້ນ (ເຊັ່ນ: Windows 7, 8, 10, ແລະ 11). ດັ່ງນັ້ນ, ມັນຈຶ່ງເປັນຜະລິດຕະພັນຂອງບໍລິສັດ Microsoft ໂດຍກົງ.

3.3 ວິທີເກັບກຳຂໍ້ມູນ

3.3.1ການທົດລອງດ້ານຄວາມໄວ ໃນການຕອບກັບຂໍ້ມູນ (Response Time)

ການທົດລອງການເກັບກຳຂໍ້ມູນສ້າງເຊີວິສຣິເຄວສ ທີ່ມີຂໍ້ ມູນ 100 ເຊີວິສຣິເຄວສ 1000 ເຊີວິສຣິເຄວສ 50,000 ເຊີວິສຣິ ເຄວສ 100,000 ເຊີວິສຣິເຄວສ ຕາມລຳດັບດ້ວຍ Font End ທີ່ ສ້າງຂຶ້ນແລ້ວນຳຄ່າທີ່ໄດ້ໄປບັນທຶກໄວ້ໃນຕາຕະລາງຜ່ານ Store Procedure ເກັບກຳຂໍ້ມູນໂດຍຫົວໜ່ວຍທີ່ໃຊ້ໃນການເກັບກຳ ແມ່ນເປັນ Second (S) ເຊິ່ງການທົດລອງນີ້ແມ່ນຈະເຮັດວົນ ເປັນຈຳນວນ 5 ຄັ້ງ.

ການທົດລອງການເກັບກຳຂໍ້ມູນການຕອບເຊີວິສຣິເຄວສ ຂໍ້ ມູນຈຳນວນ 100 ຕອບເຊີວິສຣິເຄວສ 1000 ຕອບເຊີວິສຣິ ເຄວສ 50,000 ຕອບເຊີວິສຣິເຄວສ 100,000 ຕອບ ຕາມລຳດັບ ດ້ວຍ Back End ທີ່ສ້າງຂຶ້ນແລ້ວນຳຄ່າທີ່ໄດ້ໄປບັນທຶກໄວ້ໃນ ຕາຕະລາງຜ່ານ Store Procedure ເກັບກຳຂໍ້ມູນໂດຍຫົວໜ່ວຍ ທີ່ໃຊ້ໃນການເກັບກຳແມ່ນເປັນ Second (S) ເຊິ່ງການທົດລອງນີ້ ແມ່ນ ຈະເຮັດວົນເປັນຈຳນວນ 5 ຄັ້ງ.

ການທົດລອງການເກັບກຳຂໍ້ມູນຢືນຢັນຄຳຕອບເຊີວິສຣິເຄວສ ສາມາດໃຊ້ໄດ້ຈິງຫຼືບໍ່ໃຊ້ໄດ້ ທີ່ມີຂໍ້ມູນ 100 ຄັ້ງ 1000 ຄັ້ງ 50,000 ຄັ້ງ 100,000 ຄັ້ງ ຕາມລຳດັບດ້ວຍ Font End ທີ່ສ້າງ ຂຶ້ນ ແລ້ວນຳຄ່າທີ່ໄດ້ໄປບັນທຶກໄວ້ໃນຕາຕະລາງເກັບກຳຂໍ້ມູນຜ່ານ Store Procedure ໂດຍຫົວໜ່ວຍທີ່ໃຊ້ໃນການເກັບກຳແມ່ນ ເປັນ Second (S) ເຊິ່ງການທົດລອງນີ້ແມ່ນ ຈະເຮັດວົນຊ້ຳ ເປັນຈຳນວນ 5 ຄັ້ງ.

ການທົດລອງການເກັບກຳຂໍ້ມູນການສະແດງກຣາຟໂດຍນຳຂໍ້ ມູນໃນ 100 ຄັ້ງ 1000 ຄັ້ງ 50,000 ຄັ້ງ 100,000 ຄັ້ງ ຕາມລຳ ດັບດ້ວຍ Back End ທີ່ສ້າງຂຶ້ນ ແລ້ວນຳຄ່າທີ່ໄດ້ໄປສະແດງໂດຍ ຫົວໜ່ວຍທີ່ໃຊ້ໃນການເກັບກຳແມ່ນເປັນ Second (S) ເຊິ່ງ ການທົດລອງນີ້ ແມ່ນຈະເຮັດວົນເປັນຈຳນວນ 5 ຄັ້ງ.

3.3.2 ການທົດລອງດ້ານການນຳໃຊ້ຫນ່ວຍຄວາມຈຳຊົ່ວຄາວ (RAM) ແລະ ຫນ່ວຍປະມວນຜົນກາງ (CPU)

ການທົດລອງການເກັບກຳຂໍ້ມູນສ້າງເຊີວິສຣິເຄວສ ທີ່ມີຂໍ້ມູນ 100 ເຊີວິສຣິເຄວສ 1000 ເຊີວິສຣິເຄວສ 50,000 ເຊີວິສຣິ

ເຄອສ 100,000 ເຊີວິສຣິ ຕາມລຳດັບດ້ວຍ Font End ທີ່ສ້າງ ຂຶ້ນແລ້ວນຳຄ່າທີ່ໄດ້ໄປບັນທຶກໄວ້ໃນຕາຕະລາງຜ່ານ Store Procedure ເກັບກຳຂໍ້ມູນຄ່າທີ່ໄດ້ຮັບຈາກການທົດລອງແມ່ນຈະ ຄິດເປັນເປີເຊັນນີ້(%) ສຳຫຼັບຂໍ້ມູນ CPU ແລະ ເປັນເມັກກະໄບ (MB) ສຳຫຼັບຂໍ້ມູນ RAM ແມ່ນຈະເຮັດວຽນເປັນຈຳນວນ 5 ຄັ້ງ.

ການທົດລອງການເກັບກຳຂໍ້ມູນການຕອບເຊີວິສຣິເຄອສ ຂໍ້ ມູນຈຳນວນ 100 ຕອບເຊີວິສຣິເຄອສ 1000 ຕອບເຊີວິສຣິ ເຄອສ 50,000 ຕອບເຊີວິສຣິເຄອສ 100,000 ຕອບເຊີວິສຣິ ເຄອສ ຕາມລຳດັບດ້ວຍ Back End ທີ່ສ້າງຂຶ້ນ ແລ້ວນຳຄ່າທີ່ໄດ້ ໄປບັນທຶກໃວ້ໃນຕາຕະລາງຜ່ານ Store Procedure ເກັບກຳຂໍ້ ມູນຄ່າທີ່ໄດ້ຮັບຈາກການທົດລອງແມ່ນຈະຄິດເປັນເປີເຊັນນີ້ (%) ສຳຫຼັບຂໍ້ມູນ CPU ແລະ ເປັນເມັກກະໄບ (MB) ສຳຫຼັບຂໍ້ມູນ RAM ແມ່ນຈະເຮັດວຽນເປັນຈຳນວນ 5 ຄັ້ງ.

ການທົດລອງການເກັບກຳຂໍ້ມູນຢືນຢັນຄຳຕອບເຊີວິສຣິເຄອສ ສາມາດໃຊ້ໄດ້ຈິງຫຼືບໍ່ໃຊ້ໄດ້ ທີ່ມີຂໍ້ມູນ 100 ຄັ້ງ 1000 ຄັ້ງ 50,000 ຄັ້ງ 100,000 ຄັ້ງ ຕາມລຳດັບ ດ້ວຍ Font End ທີ່ສ້າງ ຂຶ້ນ ແລ້ວນຳຄ່າທີ່ໄດ້ໄປບັນທຶກໄວ້ໃນຕາຕະລາງເກັບກຳຂໍ້ມູນຜ່ານ Store Procedure ເກັບກຳຂໍ້ມູນ ຄ່າທີ່ໄດ້ຮັບຈາກການທົດລອງ ແມ່ນຈະຄິດເປັນເປີເຊັນນີ້(%) ສຳຫຼັບຂໍ້ມູນ CPU ແລະ ເປັນ ເມັກກະໄບ (MB) ສຳຫຼັບຂໍ້ມູນ RAM ແມ່ນຈະເຮັດວຽນເປັນຈຳ ນວນ 5 ຄັ້ງ.

ການທົດລອງການເກັບກຳຂໍ້ມູນການສະແດງກຣາຟໂດຍນຳຂໍ້ ມູນໃນ 100 ຄັ້ງ 1000 ຄັ້ງ 50,000 ຄັ້ງ 100,000 ຄັ້ງ ຕາມລຳ ດັບ ດ້ວຍ Back End ທີ່ສ້າງຂຶ້ນ ແລ້ວນຳຄ່າທີ່ໄດ້ໄປສະແດງ ເກັບກຳຂໍ້ມູນ ຄ່າທີ່ໄດ້ຮັບຈາກການທົດລອງແມ່ນຈະຄິດເປັນ ເປີເຊັນນີ້ (%) ສຳຫຼັບຂໍ້ມູນ CPU ແລະ ເປັນເມັກກະໄບ (MB) ສຳຫຼັບຂໍ້ມູນ RAM ແມ່ນຈະເຮັດວຽນເປັນຈຳນວນ 5 ຄັ້ງ.

3.4 ການວິເຄາະ ແລະ ການຕີຄວາມຫມາຍຂໍ້ມູນ

ການວິເຄາະ ແລະ ການຕີຄວາມຄວາມຂອງຂໍ້ມູນຜູ້ທົດລອງ ເອງແມ່ນພວກເຮົາຈະສົມທຽບຈາກການນຳໃຊ້ຂໍ້ມູນຕົວຈິງທີ່ໄດ້ ຮັບຈາກການທົດລອງ ໂດຍຜົນທີ່ໄດ້ຈາກ Monolithic ແລະ Microservices (ເຊີວິສຣິເຄອສ , ຕອບເຊີວິສຣິເຄອສ, ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄອສ, ການສະແດງກຣາຟ) ແລ້ວຫາຄ່າ ສະເລ່ຍຄວາມໄວໃນການຕອບກັບຂໍ້ມູນມີຫົວໜ່ວຍເປັນ ວິນາທີ (S), ການໃຊ້ໜ່ວຍຄວາມຈຳຊົ່ວຄາວມີຫົວໜ່ວຍເປັນເມກະ ໄບ (MB) ແລະ ການໃຊ້ໜ່ວຍປະມວນຜົນກາງມີຫົວໜ່ວຍ ເປັນເປີເຊັນ (%).

ເຊິ່ງວ່າໃນການທົດລອງ (ເຊີວິສຣິເຄອສ , ຕອບເຊີວິສຣິ ເຄອສ, ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄອສ, ການສະແດງກຣາຟ) ກັບ ຂໍ້ມູນ 100 ຣິເຄອສ 1000 ຣິເຄອສ 50,000 ຣິເຄອສ 100,000 ຣິເຄອສ ຕາມລຳດັບເປັນຈຳນວນ 5 ຄັ້ງ ຂອງ Monolithic ແລະ Microservices ຂອງ IT HelpDesk ແລ້ວມາຫາຄ່າສະເລ່ຍ ເກັບໄວ້ເພື່ອປຽບທຽບໂດຍຄ່າສະເລ່ຍຕົວໃດທີ່ມີປະສິດທິພາບ ດີກວ່າແມ່ນຈະໃຫ້ຄະແນນເປັນ 1 ສ່ວນ ຄ່າສະເລ່ຍໃດທີ່ມີ ປະສິດທິພາບຕ່ຳກວ່າແມ່ນຈະໃຫ້ຄະແນນເປັນ 0 ເພື່ອໃຫ້ເປັນ ການງ່າຍໃນການສະຫຼຸບໃນຕອນທ້າຍ.

3.4.1 ສູດຄຳນວນຫາຄ່າສະເລ່ຍ

ສູດຄິດໄລ່ຫາຄ່າສະເລ່ຍໃນການທົດລອງຂອງຄວາມໄວໃນ ການຕອບກັບຂໍ້ມູນ (Response Time), ການນຳໃຊ້ໜ່ວຍ ຄວາມຈຳຊົ່ວຄາວ (RAM), ການນຳໃຊ້ໜ່ວຍປະມວນຜົນກາງ (CPU) ລະຫວ່າງ Monolithic ແລະ Microservices ທີ່ເຮັດ ວຽກຢູ່ໃນສະພາບແວດລ້ອມຄືກັນແມ່ນຈະຄິດໄລ່ຫາຄ່າສະເລ່ຍ ຕາມຫຼັກຄະນິດສາດດັ່ງລຸ່ມນີ້:

ສູດຄິດໄລ່ຫາຄ່າສະເລ່ຍ : $AV_x = \frac{\sum x}{n}$

- AV_x ແມ່ນແທນໃຫ້ກັບຄ່າສະເລ່ຍຂອງ.

- $\sum x$ ແມ່ນຜົນບວກທັງຫມົດຂອງຂໍ້ມູນ.

- n ແມ່ນແທນໃຫ້ກັບຈຳນວນຄັ້ງທີ່ທົດລອງ.

ທີ່ມາແນວຄິດນີ້ມີມາຕັ້ງແຕ່ສະໄໝບູຮານ, ໂດຍມີການ ບັນທຶກການນຳໃຊ້ໂດຍ ຊາວບາບີໂລນ (Babylonians) ແລະ ຊາວກຣີກ (Greeks) ໃນການຄິດໄລ່ທາງດາລາສາດ ແລະ ການ ຄ້າ. ມັນເປັນແນວຄິດທີ່ພັດທະນາຂຶ້ນຕາມທຳມະຊາດເພື່ອຫາ "ຄ່າກາງ" ຫຼື "ຕົວແທນ" ຂອງກຸ່ມຂໍ້ມູນ.

ອ້າງອີງຈາກ "ຫຼັກການພື້ນຖານຂອງຄະນິດສາດ ແລະ ສະຖິຕິ ສາດ (Fundamental principle of mathematics and statistics)".

3.4.2 ສູດຄຳນວນຫາຄວາມແຕກຕ່າງ

ສູດຄິດໄລ່ຫາຄວາມແຕກຕ່າງໃນການທົດລອງຂອງຄວາມ ໄວໃນການຕອບກັບຂໍ້ມູນ (Response Time), ການນຳ ໃຊ້ໜ່ວຍຄວາມຈຳຊົ່ວຄາວ (RAM), ການນຳໃຊ້ໜ່ວຍ ປະມວນຜົນກາງ (CPU) ລະຫວ່າງ Monolithic ແລະ Microservices ທີ່ເຮັດວຽກຢູ່ໃນສະພາບເວດລ້ອມຄືກັນແມ່ນ ຈະ ຄິດໄລ່ຫາຄ່າສະເລ່ຍຕາມຫຼັກຄະນິດສາດດັ່ງລຸ່ມນີ້:

ສູດຄຳນວນຫາຄວາມແຕກຕ່າງ $AV_{xy} = A - B$

- AV_{xy} ແມ່ນແທນໃຫ້ກັບຄວາມແຕກຕ່າງຂໍ້ມູນຕາມ ຫົວໜ່ວຍ.

ທີ່ມາ: ຫຼັກການພື້ນຖານຂອງວິຊາເລຂາຄະນິດ (Fundamental Principle of Arithmetic)

ການອ້າງອີງ: "ຈາກຫຼັກການຄິດໄລ່ທາງຄະນິດສາດພື້ນຖານ".

- A ແມ່ນແທນໃຫ້ກັບຄ່າທີ 1

- B ແມ່ນແທນໃຫ້ກັບຄ່າທີ 2

ສູດຄຳນວນຫາຄວາມແຕກຕ່າງເປັນສ່ວນຮ້ອຍ

$$AV_y = \frac{A - B}{(A + B)/2} \times 100$$

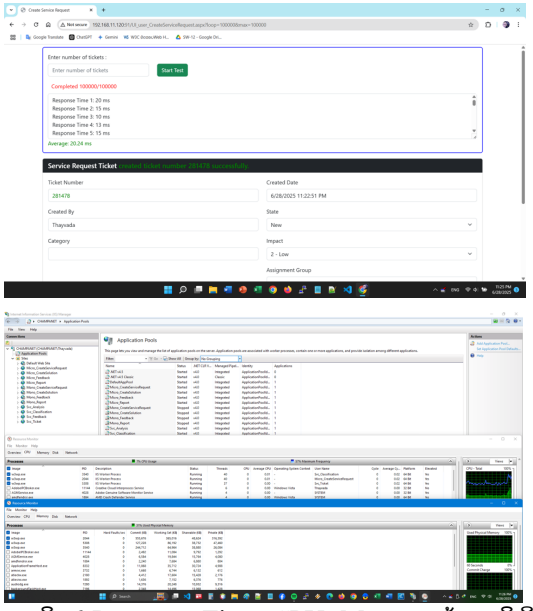
- AV_y ແມ່ນແທນໃຫ້ກັບຄ່າເປີເຊັນຄວາມແຕກຕ່າງ.

- A ແມ່ນແທນໃຫ້ກັບຄ່າທີ 1

- B ແມ່ນແທນໃຫ້ກັບຄ່າທີ 2

ທີ່ມາປຶ້ມຕຳລາຄະນິດສາດ ຫຼື ສະຖິຕິ (Mathematics or Statistics Textbooks)

ການອ້າງອີງ: ເວັບໄຊຂອງສະຖາບັນການສຶກສາ, Wolfram MathWorld ຫຼື Khan Academy.



ຮູບທີ 6 Response Time, CPU, Memory ສ້າງເຊີວິສຣິເຄວສຂອງ Monolithic ແລະ Microservices

4. ຜົນການຄົ້ນຄວ້າ

4.1. ຜົນການເກັບຂໍ້ມູນດ້ານຄວາມໄວ (Response Time)

ໃນການປຽບທຽບປະສິດທິພາບດ້ານຄວາມໄວດ້ວຍການທົດລອງການ ສ້າງເຊີວິສຣິເຄວສ , ການຕອບເຊີວິສຣິເຄວສ , ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄວສ , ການສະແດງກຣາຟ ການທົດລອງທັງຫມົດແມ່ນ 5 ຄັ້ງ ແລ້ວນຳຄ່າທີ່ໄດ້ມາສະເລ່ຍ ແລະ ຫາຄ່າສ່ວນຕ່າງເພື່ອປຽບທຽບຫາປະສິດທິພາບໃນການປະມວນຜົນຄວາມໄວໃນການຕອບກັບຂໍ້ມູນຂອງ Monolithic ແລະ Microservices ໂດຍມີຫົວຫນ່ວຍເປັນວິນາທີ (ms) ແລະ ສ້າງກຣາຟສະແດງຜົນທີ່ໄດ້ຮັບເຊິ່ງມີລາຍລະອຽດດັ່ງລຸ່ມນີ້:

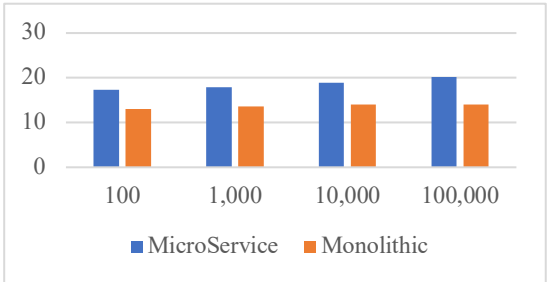
- ປຽບທຽບປະສິດທິພາບດ້ານຄວາມໄວ (Response Time) CPU, Memory ສ້າງເຊີວິສຣິເຄວສ, ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄວສ, ການສະແດງກຣາຟ

ໃນການເກັບຂໍ້ມູນການທົດລອງການບັນເພີ່ມຂໍ້ມູນ ສ້າງເຊີວິສຣິເຄວສ,ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄວສ, ການສະແດງກຣາຟ ຂໍ້ມູນຈຳນວນ 100 ຄັ້ງ 1,000 ຄັ້ງ 10,000 ຄັ້ງ 100,000 ຕາມລຳດັບ ເພື່ອວັດປະສິດທິພາບດ້ານຄວາມໄວ (Response Time) CPU, Memory ສ້າງເຊີວິສຣິເຄວສ ທີ່ມີການເພີ່ມຂຶ້ນຂອງຈຳນວນເຊີວິສຣິເຄວສ ຂໍ້ມູນແຕ່ລະໄລຍະວ່າມີການປ່ຽນແປງຫນ້ອຍຫຼາຍພຽງໃດ.

ຕາຕະລາງທີ 2 ຜົນການທົດລອງປະສິດທິພາບດ້ານຄວາມໄວ (Response Time) ສ້າງເຊີວິສຣິເຄວສ

ຈຳນວນຄັ້ງ	Microservices ຫົວຫນ່ວຍ (ms)	Monolithic ຫົວຫນ່ວຍ (ms)	ສ່ວນຕ່າງ ຫົວຫນ່ວຍ(ms)
1000	17.3	13.04	4.26
1,000	17.9	13.55	4.35
10,000	18.86	13.99	4.87
100,000	20.24	14.04	6.2
ລວມ	74.3	54.62	19.68
ສະເລ່ຍ	18.56	13.66	4.90

ຈາກຕາຕະລາງການທົດລອງເກັບກຳຂໍ້ມູນດ້ານເທິງສາມາດເຫັນໄດ້ວ່າປະສິດທິພາບໃນການປະມວນຜົນການເພີ່ມຂໍ້ມູນຂອງ Microservices ແລະ Monolithic ຂອງ IT HelpDesk.



ຮູບທີ 7 ຄ່າສະເລ່ຍຄວາມໄວ (Response Time) ສ້າງເຊີວິສຣິເຄວສ

- ປຽບທຽບປະສິດທິພາບດ້ານຄວາມໄວ (Response Time) ໃນການຕອບເຊີວິສຣິເຄວສ

ຕາຕະລາງທີ 3 ສະຫຼຸບຜົນການຜົນການປຽບທຽບປະສິດທິພາບຄວາມໄວ (Response Time)

ຄ່າສະເລ່ຍຄວາມໄວໃນການຕອບກັບຂໍ້ມູນ (Response Time) ຫົວຫນ່ວຍເປັນວິນາທີ (s)				
ຮູບແບບ	**Microservices ຫົວຫນ່ວຍ (ms)**	**Monolithic ຫົວຫນ່ວຍ (ms)**	**ສ່ວນຕ່າງ ຫົວຫນ່ວຍ (ms)**	**ສ່ວນຕ່າງ ຫົວຫນ່ວຍ (%)**
ສ້າງເຊີວິສຣິເຄວສ	18.56	13.66	4.9	30.42
ຕອບເຊີວິສຣິເຄວສ	53.45	49.09	4.36	8.50
ຢືນຢັນຄຳຕອບເຊີວິສຣິເຄວສ	65.33	46.99	18.34	32.66
ການນຳຂໍ້ມູນມາສະແດງກຣາຟ	3.28	0.36	2.92	160.44
ສະເລ່ຍ	35.16	27.53	7.63	58.00

ຄ່າສະເລ່ຍຄວາມໄວໃນການໃຊ້ຫນ່ວຍຄວາມຈຳຊົ່ວຄາວ (RAM) (MB)				
ຮູບແບບ	**Microservices ຫົວຫນ່ວຍ (MB)**	**Monolithic ຫົວຫນ່ວຍ (MB)**	**ສ່ວນຕ່າງ ຫົວຫນ່ວຍ (MB)**	**ສ່ວນຕ່າງ ຫົວຫນ່ວຍ (%)**

ສ້າງເຊີວິສຣິເถอส	543.91	204.53	339.38	90.69
ภาบตอบเชิวิสธิเถอส	412.66	176.03	236.63	80.39
ยืบยันคำตอบเชิวิสธิเถอส	537.92	173.45	364.47	102.47
ภาบบำข้ำูบมาสะแดงภธาพ	427.7	220.44	207.26	63.96
สะเล่ย	480.55	193.61	286.94	84.38

ค่าสะเล่ยภาบบำใช้หบ่อยปะมอบผิบภาๆ (CPU) ทิอหบ่อย(%)				
รูบแบบ	**Microservices ทิอหบ่อย (%)**	**Monolithic ทิอหบ่อย (%)**	**ส่อบต่างๆ ทิอหบ่อย (%)**	**ส่อบต่างๆ ทิอหบ่อย (%)**
ส้างเชิวิสธิเถอส	0.46	0.16	0.3	96.77
ภาบตอบเชิวิสธิเถอส	0.30	0.15	0.15	66.67
ยืบยันคำตอบเชิวิสธิเถอส	0.24	0.12	0.12	66.67
ภาบบำข้ำูบมาสะแดงภธาพ	0.33	0.19	0.14	53.85
สะเล่ย	0.33	0.16	0.18	70.99

5. อะพิปายผิบ

ถอามไอในภาบตอบสะหฆอๆ (Response Time / Performance) สะถาปัดตะยะกำแบบ Monolithic มิถอามไอในภาบตอบสะหฆอๆ (Response Time) ไอกว่า Microservices ในทุกๆภาบทิดสอบ (สร้างเชิวิสธิเถอส , ตอบเชิวิสธิเถอส, ยืบยันคำตอบเชิวิสธิเถอส, ภาบสะแดงภธาพ) โดยสะเล่ยไอกว่าปะมาบ 58%. ผิบภาบถิ้บพิบนี้ขัดแย้งกับ ถำไพ ถุบนะวิงสา (2023) ที่พิบว่า Microservices ไอกว่า Monolithic โดยสะเพาะเมื่อมิผู้ใช้งาบจำบวบหูาย (ไอกว่า 83-93%). แบอใดกำตาม Ueda, et al. (2016) สึกสาก่ยวกับ Workload Characterization for Microservices ได้ผิบภาบถิ้บถอ้าสอดถ่องกับกับบิดถิ้บถอ้านี้ ที่ว่า Microservices เธ็ดให้ปะสิดทิพาบ (Throughput) หูดลิ่ง ย่างหูวงหูาย (70-79%) เมื่อทยบกับ Monolithic. ภาบที่ Throughput หูดลิ่ง กำหมายถอามว่าละบิบธองธับอยภได้ห้อยลิ่งในเอลาเท่ากับ, เชิ่งสอดถ่องกับภาบที่ Response Time สูงขึ้บในบิดอิใจนี้. บิดอิใจของ ถำไพ ใช้ Java, Spring แล Docker เชิ่งถูกออกแบบมาเพื่อธองธับภาบขะหยายโต (Scaling) ได้ดิ. ถอามไอของ Microservices จะเห็บผิบชัดเจบเมื่อละบิบมิภาบโหูดสูง (High Load) แล ต้องภาบขะหยายแต่ละส่อบเป็บอิดสะหูะ.นี้อาดเอ้ำได้ว่าในภาบเธ็ดอยภพื้บถาบที่บ่ซับซ้อบ, Monolithic อาดไอกว่าเพาะบ่มี Overhead จากภาบสื่สาบผ่าบเถือข่าย (Network API calls) ถึกับ Microservices. แต่ในละบิบใหย่ที่มิภาบใช้งาบหับภา, Microservices สามาดสะแดงปะสิดทิพาบที่เหบือกว่าได้.

สำลับด้าบภาบบำใช้ซับพะยากอบ (Resource Utilization: RAM & CPU) สะถาปัดตะยะกำแบบ Microservices ใช้ซับพะยากอบสูงกว่า ย่างชัดเจบ, โดยใช้ RAM หูายกว่าสะเล่ย 84% แล ใช้ CPU หูายกว่าสะเล่ย 70%. ผิบภาบถิ้บถอ้านี้สอดถ่องกับ ถำไพ ถุบนะวิงสา (2023) เชิ่งละบุว่า "Microservice ใช้ซับพะยากอบของเชิบเอิหูายกว่า Monolithic". เชุ่บดยวกับกับ Ueda, et al. (2016) ที่พิบว่า Microservices มิภาบใช้คำสั่ง CPU สูงกว่า Monolithic เถิง 3-3.05 เท่า ต่ request. ทุกๆภาบอิใจที่อ้างอิงมาใช้ผิบลับในทิดทางดยวกับ. นี้เป็บข้อสะหฺบที่หม้าเชื่อถือได้ว่า Microservices มิ Overhead สูงกว่า Monolithic. เหดผิบหลักภาคือ แต่ละ Service ต้องเธ็ดอยภใบ Process ของตับเอง, มิ Runtime Environment ของตับเอง, แล มิภาบสื่สาบผ่าบ Network เชิ่งทั้งหมดบี้ล้อบแต่ต้องภาบ RAM แล CPU เพิ่มขึ้บ.

สำลับด้าบอื่บๆ (ค่าใช้จ่าย, ภาบบำลุงธักสา, ถอามยืดยุ่บ) เห็บว่าเถิง Monolithic จะมิปะสิดทิพาบดิกว่า, แต่ Microservices มิข้อดิด้าบถอามยืดยุ่บ (Flexibility), ภาบขะหยายละบิบ (Scalability) แล ภาบบำลุงธักสา (Maintenance) ที่ง่ายกว่าในไลยะยาอ เชิ่งสอดถ่องกับ Villamizar, et al. (2016) ที่พิบว่า Microservices สามาดหูดค่าใช้จ่าย ในละบิบ Cloud ได้, เชิ่งเป็บผิบมาจากถอามสามาดในภาบขะหยายละบิบ (Scalability) ได้ย่างมิปะสิดทิพาบ (ขะหยายสะเพาะส่อบที่จำเป็บ). สำลับบิดถิ้บถอ้าในหัวข้อ Leveraging Microservices architecture by using Docker technology ยังอะทิบายเพิ่มติ่มอิกว่า Docker ช่อยให้ Microservices มิถอามเป้บอิดสะหูะ, ง่ายต่ภาบเถื่อบย้าย แล ง่ายต่ภาบเธ็ดอยภอัดตะโบมัด, เชิ่งทั้งหมดบี้ภาถือปัดใจที่เธ็ดให้ภาบบำลุงธักสาง่ายขึ้บ.

ในด้าบปะสิดทิพาบดิบ (Raw Performance), Monolithic ขะบะย่างชัดเจบในภาละบิสีภาสาบี้. ผิบภาบทิดลองสะแดงให้เห็บว่าสะถาปัดตะยะกำแบบ Monolithic มิถอามไอในภาบตอบสะ หฆอๆ (Response Time) ไอกว่า Microservices โดยสะเล่ยเถิง 58% แล ใช้ซับพะยากอบ (RAM แล CPU) หัอยกว่าย่างมะหาสาบ. เหดผิบหลักแม่บย้อบถอามซับซ้อบที่เพิ่มขึ้บของ Microservices. ในขะบะที่ Monolithic ปะมอบผิบทุกย่างพายในโปธแกธมดยว, Microservices ต้องมิภาบสื่สาบข้ามเถื่อข่ายละหว่างๆแต่ละเชิวิด (API calls) เชิ่งส้าง “Overhead” (พาละอยภที่เพิ่มเติม) ขึ้บมา, เธ็ดให้มับช้าลิ่ง แล ใช้ซับพะยากอบหูายขึ้บเพื่อจัดภาบแต่ละเชิวิดที่แยกจากกับ.แบอใดกำตาม ปะสิดทิพาบบ่แม่บถำตอบสุดท้าย, ถอามยืดยุ่บ แล ภาบบำลุงธักสาในไลยะยาวสำคับ

ກວ່າ. ເຖິງແມ່ນວ່າ Monolithic ແຕ່ Microservices ມີຂໍ້ດີທີ່ສຳຄັນກວ່າໃນໄລຍະຍາວ, ໂດຍສະເພາະສຳລັບລະບົບທີ່ອາດຈະຕ້ອງເຕີບໂຕໃນອະນາຄົດ. ການຂະຫຍາຍລະບົບ (Scalability): ສາມາດເລືອກຂະຫຍາຍສະເພາະເຊີວິດທີ່ໃຊ້ງານໜັກໄດ້, ຕ່າງຈາກ Monolithic ທີ່ຕ້ອງຂະຫຍາຍທັງລະບົບ. ສ່ວນການບຳລຸງຮັກສາ (Maintainability): ການແກ້ໄຂ ຫຼື ປັບປຸງລະບົບເຮັດໄດ້ງ່າຍກວ່າ ເພາະແຕ່ລະເຊີວິດມີຂະໜາດນ້ອຍ ແລະ ແຍກຈາກກັນ. ດ້ານຄວາມໜ້າເຊື່ອຖື (Reliability) ກໍເຊັ່ນດຽວກັນ ຖ້າມີເຊີວິດໃດໜຶ່ງລົ້ມເຫຼວ, ມັນຈະບໍ່ສົ່ງຜົນກະທົບຕໍ່ລະບົບທັງໝົດ, ຕ່າງຈາກ Monolithic ທີ່ຖ້າສ່ວນໃດໜຶ່ງຜິດພາດ, ທັງລະບົບອາດຈະຢຸດເຮັດວຽກ. ສະນັ້ນ, ການເລືອກສະຖາປັດຕະຍະກຳທີ່ຖືກຕ້ອງຂຶ້ນຢູ່ກັບ "ບໍລິບົດ" ແລະ "ເປົ້າໝາຍຂອງໂຄງການ" ການເລືອກທີ່ດີທີ່ສຸດແມ່ນຂຶ້ນຢູ່ກັບຄວາມຕ້ອງການຂອງລະບົບນັ້ນໆ.

ກໍລະນີເລືອກ Monolithic ເມື່ອສ້າງລະບົບຂະໜາດນ້ອຍຫາປານກາງ, ໂຄງການທີ່ບໍ່ຊັບຊ້ອນ ຫຼື ຕ້ອງການພັດທະນາໃຫ້ໄວໃນເບື້ອງຕົ້ນ. ກໍລະນີເລືອກ Microservices ເມື່ອສ້າງລະບົບຂະໜາດໃຫຍ່, ຕ້ອງການຄວາມຢືດຢຸ່ນສູງ, ຄາດວ່າຈະມີຜູ້ໃຊ້ງານຈຳນວນຫຼາຍ, ແລະ ຕ້ອງການຄວາມງ່າຍໃນການບຳລຸງຮັກສາ ແລະ ອັບເດດໃນໄລຍະຍາວ. ນີ້ໝາຍຄວາມວ່າ ເວລາເລືອກລະບົບໃດໜຶ່ງ ນອກຈາກເບິ່ງຕົວເລກດ້ານຄວາມໄວ ແລະ ຊັບພະຍາກອນແລ້ວຄວນໃຫ້ຄວາມສຳຄັນກັບຄຸນລັກສະນະທາງສະຖາປັດຕະຍະກຳການຂະຫຍາຍລະບົບ, ຄວາມຢືດຢຸ່ນ, ແລະ ການບຳລຸງຮັກສາພ້ອມ ເຊິ່ງຈະສົ່ງຜົນໂດຍກົງຕໍ່ຄວາມສຳເລັດຂອງໂຄງການໃນໄລຍະຍາວ.

6. ສະຫຼຸບຜົນ

ຜົນການທົດສອບສະແດງໃຫ້ເຫັນວ່າ ສະຖາປັດຕະຍະກຳແບບ Monolithic ມີຄວາມໄວໃນການຕອບສະ ໜອງ (Response Time) ໄວກວ່າ Microservices ໃນທຸກໆໜ້າວຽກທີ່ທົດສອບ (ການສ້າງ, ການຕອບ, ການຢືນຢັນ ແລະ ການສະແດງກຣາຟ). ໂດຍສະເລ່ຍລວມແລ້ວ, Monolithic ສາມາດປະມວນຜົນໄດ້ໄວກວ່າປະມານ 58%. ການນຳໃຊ້ຊັບພະຍາກອນ RAM ແລະ CPU ມີຄວາມແຕກຕ່າງກັນ. ສະຖາປັດຕະຍະກຳແບບ Monolithic ໃຊ້ຊັບພະຍາກອນໜ້ອຍກວ່າ Microservices ຢ່າງຊັດເຈນ. Microservices ໃຊ້ໜ່ວຍຄວາມຈຳ (RAM) ຫຼາຍກວ່າ Monolithic ໂດຍສະເລ່ຍສູງເຖິງ 84%. Microservices ໃຊ້ໜ່ວຍປະມວນຜົນກາງ (CPU) ຫຼາຍກວ່າ Monolithic ໂດຍສະເລ່ຍປະມານ 70%. ແນວໃດກໍຕາມ ເຖິງວ່າຜົນການທົດສອບໃນກໍລະນີສຶກສາລະບົບ IT Helpdesk ນີ້ຈະຊີ້ວ່າ Monolithic ມີປະສິດທິພາບດ້ານຄວາມໄວ ແລະ ການໃຊ້ຊັບພະຍາກອນທີ່ດີກວ່າ, ແຕ່ການຕັດສິນໃຈເລືອກໃຊ້ສະຖາປັດຕະຍະກຳໃດໜຶ່ງບໍ່ສາມາດເບິ່ງແຕ່ຕົວເລກເຫຼົ່ານີ້ພຽງຢ່າງດຽວໄດ້. Monolithic ເໝາະສົມກັບໂຄງການຂະໜາດນ້ອຍຫາປານກາງທີ່ບໍ່ຊັບຊ້ອນ, ຕ້ອງການພັດທະນາໃຫ້ໄວ ແລະ ມີປະສິດທິພາບສູງສຸດໃນສະພາບແວດລ້ອມທີ່ຄວບຄຸມໄດ້. Microservices ເຖິງວ່າຈະຊ້າກວ່າ ແລະ ໃຊ້ຊັບພະຍາກອນຫຼາຍກວ່າໃນສະພາບການທົດລອງນີ້, ແຕ່ມັນມີຂໍ້ດີທີ່ສຳຄັນໃນໄລຍະຍາວດ້ານຄວາມຢືດຢຸ່ນ (Flexibility), ຄວາມສາມາດໃນການຂະຫຍາຍລະບົບ (Scalability) ແລະ ຄວາມງ່າຍໃນການບຳລຸງຮັກສາ (Maintainability). ຖ້າລະບົບ IT Helpdesk ມີແຜນທີ່ຈະຂະຫຍາຍໃນອະນາຄົດ, ມີຜູ້ໃຊ້ຈຳນວນມະຫາສານ, ຫຼື ຕ້ອງການໃຫ້ແຕ່ລະທີມພັດທະນາສ່ວນຕ່າງໆເປັນອິດສະຫຼະ Microservices ຈະເປັນຕົວເລືອກທີ່ເໜືອກວ່າ. ດັ່ງນັ້ນ, ການເລືອກລະຫວ່າງສອງສະຖາປັດຕະຍະກຳນີ້ຈຶ່ງເປັນການຊັ່ງນ້ຳໜັກລະຫວ່າງ "ປະສິດທິພາບໃນປະຈຸບັນ “(Monolithic) ແລະ "ຄວາມພ້ອມສຳລັບການເຕີບໂຕໃນອະນາຄົດ” (Microservices) ໂດຍຕ້ອງອີງໃສ່ເປົ້າໝາຍ ແລະ ຄວາມຕ້ອງການຂອງລະບົບເປັນສຳຄັນ.

7. ຂໍ້ສະເໜີແນະ

ສຳລັບໂຄງການຂະໜາດນ້ອຍຫາປານກາງ ຜູ້ພັດທະນາ ແລະ ຜູ້ຕັດສິນໃຈຄວນພິຈາລະນາເລືອກໃຊ້ສະຖາປັດຕະຍະກຳ Monolithic ສຳລັບລະບົບທີ່ບໍ່ມີຄວາມຊັບຊ້ອນສູງ, ມີທີມພັດທະນາຂະໜາດນ້ອຍ, ແລະ ບໍ່ໄດ້ຄາດຫວັງວ່າຈະມີການຂະຫຍາຍໂຕຢ່າງກ້າວກະໂດດໃນໄລຍະສັ້ນ. ໂດຍສະເພາະໃນອົງກອນທີ່ຍັງໃຊ້ ASP ເປັນຫຼັກ, Monolithic ຍັງຄົງເປັນຕົວເລືອກທີ່ມີປະສິດທິພາບສູງ ແລະ ຄຸ້ມຄ່າ.

ຖ້າຫາກໂຄງການມີແນວໂນ້ມທີ່ຈະຂະຫຍາຍໂຕໃນອະນາຄົດ ແລະ ຈຳເປັນຕ້ອງເລືອກ Microservices ເພື່ອຄວາມຢືດຢຸ່ນ, ອົງກອນຄວນກຽມພ້ອມ ແລະ ວາງແຜນດ້ານຊັບພະຍາກອນເຊີບເວີ (CPU, RAM) ໃຫ້ພຽງພໍ. ອົງກອນຄວນປະເມີນຄວາມຕ້ອງການຂອງຕົນເອງຢ່າງຮອບດ້ານ, ທັງດ້ານປະສິດທິພາບ, ຄ່າໃຊ້ຈ່າຍ, ການບຳລຸງຮັກສາ, ແລະ ການຂະຫຍາຍລະບົບໃນອະນາຄົດ ກ່ອນທີ່ຈະຕັດສິນໃຈເລືອກສະຖາປັດຕະຍະກຳໃດໜຶ່ງ.

ບົດຄົ້ນຄວ້າຕໍ່ໄປອາດສຶກສາປຽບທຽບດ້ວຍເທັກໂນໂລຊີອື່ນໆ: ເຮັດການຄົ້ນຄວ້າປຽບທຽບແບບດຽວກັນນີ້, ແຕ່ປ່ຽນພາສາ ແລະ ເຟຣມເວີກທີ່ໃຊ້ໃນການພັດທະນາ ເຊັ່ນ Node.js, Java (Spring Boot), Python (Django/Flask) ເພື່ອເບິ່ງວ່າຜົນດ້ານປະສິດທິພາບຈະປ່ຽນແປງໄປຫຼືບໍ່ ເມື່ອໃຊ້ເຄື່ອງມືທີ່ເໝາະສົມກັບການສ້າງ Microservices ໂດຍກົງ.

ການທົດສອບໃນສະພາບແວດລ້ອມທີ່ມີການຂະຫຍາຍໂຕ (Scalable Environment): ນຳລະບົບທັງສອງໄປທົດສອບເທິງສະພາບແວດລ້ອມແບບຄລາວ (Cloud) ຫຼື ໃຊ້ເຄື່ອງມືຈັດການ Container ເຊັ່ນ Docker Swarm/Kubernetes ເພື່ອທົດສອບປະສິດທິພາບເມື່ອມີການຂະຫຍາຍລະບົບ (Scale Out) ແບບອັດຕະໂນມັດ ເພື່ອວັດແທກປະສິດທິພາບໃນສະຖານະການທີ່ໃກ້ຄຽງກັບການໃຊ້ງານຈິງຂອງລະບົບຂະໜາດໃຫຍ່.

8. ຂໍ້ຈຳກັດຂອງການຄົ້ນຄ້ວາ

ຂໍ້ຈຳກັດດ້ານເທັກໂນໂລຊີທີ່ໃຊ້ (Technological Limitation): ຜົນການຄົ້ນຄວ້າສະເພາະເຈາະຈົງກັບການພັດທະນາດ້ວຍພາສາ ASP ເທິງລະບົບປະຕິບັດການ Windows. ຜົນລັບອາດຈະແຕກຕ່າງກັນຢ່າງສິ້ນເຊີງຫາກປ່ຽນໄປໃຊ້ເທັກໂນໂລຊີອື່ນໆທີ່ກຳລັງເປັນທີ່ນິຍົມໃນການສ້າງ Microservices ເຊັ່ນ Java (Spring Boot), Node.js ຫຼື Go ເຊິ່ງຖືກອອກແບບມາໃຫ້ມີປະສິດທິພາບສູງໃນສະພາບແວດລ້ອມແບບ Microservices.

ຂໍ້ຈຳກັດດ້ານ Network Overhead: ຜົນການວິໄຈທີ່ວ່າ Microservices ຊ້າກວ່າ ແລະ ໃຊ້ CPU/RAM ຫຼາຍກວ່າ ສ່ວນໜຶ່ງແມ່ນມາຈາກການທີ່ລະບົບຕ້ອງມີການສື່ສານຜ່ານເຄືອຂ່າຍ (API Calls) ລະຫວ່າງແຕ່ລະເຊີວິສ. ບົດຄົ້ນຄວ້ານີ້ໄດ້ວັດແທກປະສິດທິພາບໂດຍລວມ ແຕ່ບໍ່ໄດ້ແຍກວັດແທກເວລາທີ່ເສຍໄປກັບການສື່ສານຜ່ານເຄືອຂ່າຍ (Network Latency) ໂດຍສະເພາະ, ເຊິ່ງເປັນປັດໄຈສຳຄັນທີ່ເຮັດໃຫ້ Microservices ມີ Overhead ສູງຂຶ້ນ.

ຂໍ້ຈຳກັດຂອງກໍລະນີສຶກສາ (Case Study Limitation): ລະບົບ IT Helpdesk ທີ່ສ້າງຂຶ້ນເພື່ອການທົດລອງນີ້ ຍັງມີຄວາມຊັບຊ້ອນໃນລະດັບໜຶ່ງ. ຄຳສັບໃນການຈັດໝວດໝູ່ (Thesaurus) ແລະ ຈຳນວນເຊີວິສຍັງມີຈຳກັດ. ໃນລະບົບທີ່ໃຫຍ່ ແລະ ຊັບຊ້ອນກວ່ານີ້ຫຼາຍເທົ່າ, ຂໍ້ດີຂອງ Microservices ໃນດ້ານການບຳລຸງຮັກສາ ແລະ ການພັດທະນາແບບເປັນອິດສະຫຼະອາດຈະເດັ່ນຊັດຂຶ້ນຈົນສາມາດຊົດເຊີຍປະສິດທິພາບທີ່ຫຼຸດລົງໄດ້.

9. ເອກະສານອ້າງອີງ

ຂໍ້ມູນກ່ຽວກັບ (ອອນໄລ). (2006). *Web Services and the Microsoft Platform* ສືບຄົ້ນຈາກ: https://msdn.microsoft.com/en-us/library/aa480728.aspx.

ຄຳໄພ ຄຸນນະວົງສາ (2023) *ການນຳໃຊ້ Microservice ເຂົ້າໃນການຍົກລະດັບການເຮັດວຽກຂອງ API* ຄະນະວິສະວະກຳສາດ *ຈັດການ ຖານຂໍ້ມູນ My SQL* ຄະນະວິສະວະກຳສາດ ມະຫາວິທະຍາໄລແຫ່ງຊາດ ປະເທດລາວ.

ໄຕພົບ ພົມມະສອນ (2023) *ປຽບທຽບສິດທິພາບຂອງ REST API ກັບ GraphQL API ທີ່ພັດທະນາດ້ວຍພາສາ Java Script ໃນການ* ມະຫາວິທະຍາໄລແຫ່ງຊາດ ປະເທດລາວ.

Chris Richardson. (2015). *Introduction to Microservices* ສືບຄົ້ນຈາກ: *Kubernetes* Faculty of Science, Srinakharinwirot University Thailand.

Development. (2018). *Hyphenation and Thesaurus for NET* ສືບຄົ້ນຈາກ: http://www.crawler-lib.net/ nhunspell.

Divyanand Malavalli. (2015). *Scalable microservice based architecture for enabling DMTF profiles* in 2015 11th International Conference on Network and Service Management (CNSM), IEEE: Barcelona,Spain.

Fenopix. (2018). *HTML5 JavaScript Charts* ສືບຄົ້ນຈາກ: https://canvasjs.com/docs/ charts/basics-of-creating-html5-chart/. https://www.nginx.com/blog/introduction-to-microservices/.

Kilgarriff. (2004). *Thesauruses for natural language processing* in International Conference on Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003. 2004, IEEE: Beijing, China.

Mancuso. (2018). *Microservices 101* ສືບຄົ້ນຈາກ: http://bits.citrusbyte.com/microservices/.

Napawit Toomwong. (2019). *Performance Comparison Between Monolithic And Microservices Using Docker And* Villamizar, Garcés, Ochoa, Castro, Salamanca, Verano, Lang. (2016). *Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures* Paper presented at the 2016 16th IEEE/ACM InternationalSymposium on Cluster, Cloud and Grid Computing (CCGrid).