ThoughtWorks®

# TECHNOLOGY RADAR VOL. 21

คู่มือนำทางสู่ปลายขอบเทคโนโลยี ตามแบบฉบับของเรา

thoughtworks.com/radar
#TWTechRadar

# ผู้ร่วมสร้างสรรค์

*เรดาร์เทคโนโลยีเกิดจากการเรียบเรียง*
*โดยคณะกรรมการที่ปรึกษาด้านเทคโนโลยี*

—

เรดาร์เทคโนโลยีฉบับนี้เขียนขึ้นมาจากการประชุมของคณะกรรมการที่ปรึกษาด้านเทคโนโลยี
ในซานฟรานซิสโก (San Francisco) ในเดือนตุลาคม พ.ศ. 2562

Rebecca Parsons (CTO)

Martin Fowler (Chief Scientist)

Bharani Subramaniam

Erik Dörnenburg

Evan Bottcher

Fausto de la Torre

Hao Xu

Ian Cartwright

James Lewis

Jonny LeRoy

Ketan Padegaonkar

Lakshminarasimhan Sudarshan

Marco Valtas

Mike Mason

Neal Ford

Ni Wang

Rachel Laycock

Scott Shaw

Shangqi Liu

Zhamak Dehghani


*© ThoughtWorks, Inc. All Rights Reserved.*

# เกี่ยวกับ เรดาร์เทคโนโลยี

พวกเรา Thoughtworkers ล้วนมีความหลงใหลในเทคโนโลยี เราสร้าง วิจัย ทดสอบ เปิดโอเพนซอร์ส ผลิตงานเขียน และมีเป้าหมายที่จะยกระดับเทคโนโลยีให้ดีขึ้นอย่างต่อเนื่องเพื่อทุกคนและทุกธุรกิจ พันธกิจของพวกเราคือ “ผลักดันความ เป็นเลิศทางซอฟต์แวร์ และปฏิวัติอุตสาหกรรมไอทีให้ดีขึ้นกว่าเดิม” การจัดทำและแบ่งปันเทคโนโลยีเรดาร์ ก็เพื่อสนับสนุนพันธกิจนี้

คณะกรรมการที่ปรึกษาของ ThoughtWorks ซึ่งประกอบไปด้วยผู้นำและผู้เชี่ยวชาญในเทคโนโลยีหลากหลายด้านที่รวมตัวกันเป็นประจำเพื่อพูดคุยแลกเปลี่ยนถึงแผนยุทธศาสตร์ทางเทคโนโลยีภายใน ThoughtWorks เอง และแนวโน้ม ของเทคโนโลยีทีกำลังมีบทบาทสำคัญ ในอุตสาหกรรมโลกขณะนั้น

การรวบรวมผลการประชุมดังกล่าว ถูกจัดทำขึ้นเป็นเรดาร์เทคโนโลยีฉบับนี้ โดยนำเสนอในรูปแบบที่เป็นประโยชน์กับทุกคนในวงการ ตั้งแต่นักพัฒนาจนถึงผู้บริหารระดับสูง ตัวเนื้อหานั้น เราเจตนาสรุปให้กระชับได้ใจความ หากอยากทราบถึงรายละเอียดเพิ่มเติม เราขอส่งเสริมให้คุณทดลองใช้เทคโนโลยีเหล่านั้นด้วยตัวเอง

เรดาร์เทคโนโลยีใช้แผนภาพวงกลมในการนำเสนอข้อมูล โดยแบ่งพื้นที่ออกเป็น 4 กลุ่มดังนี้ เทคนิค เครื่องมือ แพลตฟอร์ม ภาษาและเฟรมเวิร์ก ในกรณีที่บางเรื่องสามารถจัดลงได้หลายกลุ่มเราจะจัดลงในกลุ่มที่เหมาะสมที่สุด นอกจากนั้นเรายังจัดกลุ่มเรื่องต่างๆ แล้วแบ่งตามวงแหวนออกเป็น 4 ชั้นเพื่อสะท้อนมุมมองตำแหน่งของเทคโนโลยีตามความคิด เห็นของเรา

หากสนใจประวัติเพิ่มเติมเกี่ยวกับเรดาร์เทคโนโลยีสามารถดูเพิ่มเติมได้ที่ thoughtworks.com/radar/faq

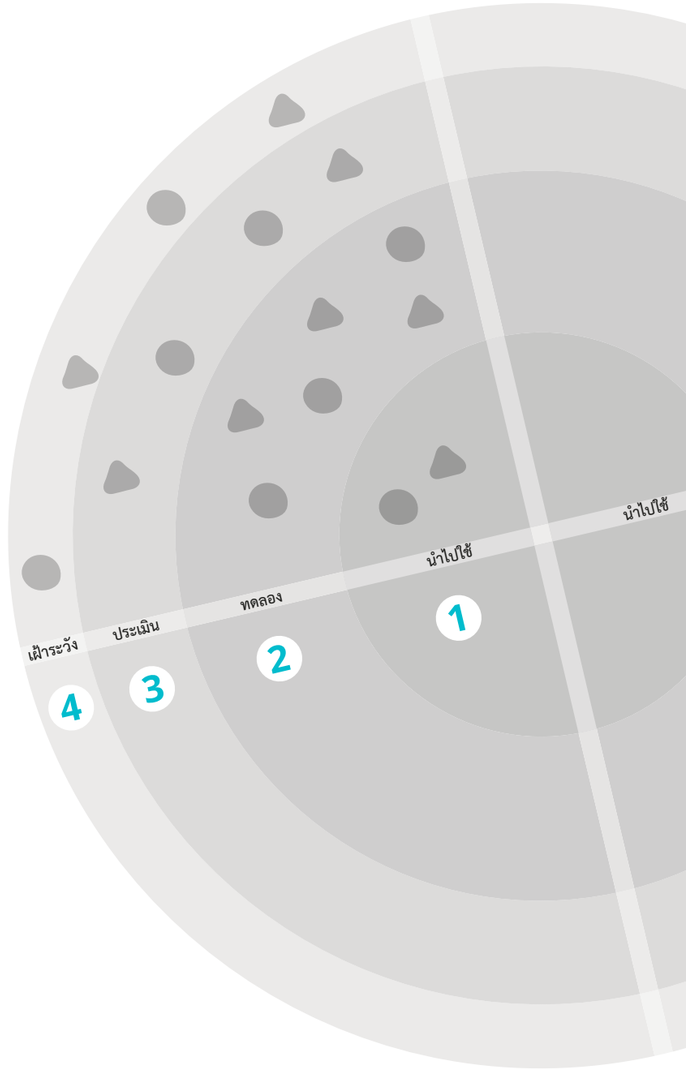
# ภาพรวม ของเรดาร์

**1 นำไปใช้ (Adopt)**
เราสนับสนุนให้นำเทคโนโลยีเหล่านี้ไปใช้ได้ทันที ซึ่งเราเองได้นำไปใช้แล้วในโครงการต่างๆ ที่มีความเหมาะสม

**2 ทดลอง (Trial)**
เรามองเห็นว่ามีความคุ้มค่าที่จะศึกษาเพิ่มเติม เห็นความสำคัญว่าควรจะใช้ความสามารถทางด้านนี้ได้อย่างไร องค์กรขนาดใหญ่ควรทดลองใช้เทคโนโลยีนี้ในโครงการที่สามารถรองรับความเสี่ยงได้

**3 ประเมิน (Assess)**
มีความคุ้มค่าที่จะสำรวจ เพื่อทำความเข้าใจว่ามีผลต่อองค์กรอย่างไร

**4 เฝ้าระวัง (Hold)**
ควรดำเนินการด้วยความระมัดระวัง




**▲ มาใหม่หรือเปลี่ยนแปลง**
**● คงเดิม**

เทคโนโลยีที่มาใหม่หรือที่มีการเปลี่ยนแปลงอย่างเห็นได้ชัดจากเรดาร์ฉบับทีแล้วจะถูกแสดงด้วยเครื่องหมายสามเหลี่ยม ในขณะที่เทคโนโลยีที่คงเดิมไม่เปลี่ยนแปลงจะถูกแสดงด้วยเครื่องหมายวงกลมเราพยายามให้เกิดการเรียนรู้ในเทคโนโลยีใหม่ๆ ดังนั้นเทคโนโลยีที่ไม่มีความเคลื่อนไหวจะไม่ถูกพูดถึงในฉบับนี้ ซึ่งไม่ได้หมายความว่าเทคโนโลยีนั้นๆ ไม่มีคุณค่า แต่เนื่องด้วยพื้นที่ของเรดาร์นั้นมีจำกัด

! เราพยายามให้เกิดการเรียนรู้ในเทคโนโลยีใหม่ๆ ดังนั้นเทคโนโลยีที่ไม่มีความเคลื่อนไหวจะไม่ถูกพูดถึงในฉบับนี้ ซึ่งไม่ได้หมายความว่าเทคโนโลยีนั้นๆไม่มีคุณค่า แต่เนื่องด้วยพื้นที่ของ radar มีจำกัด


*© ThoughtWorks, Inc. All Rights Reserved.*

# มีอะไรใหม่?

*ประเด็นที่โดดเด่นในฉบับนี้*

## คลาวด์ : ยิ่งทำมากยิ่งได้น้อย?

ผู้ให้บริการคลาวด์รายใหญ่ต่างมีบริการหลักที่มีความสามารถทัดเทียมกัน ทำให้การแข่งขันมุ่งเน้นไปที่การให้บริการเสริมซึ่งส่งเสริมให้พวกเขาออกบริการใหม่ด้วยความเร็วที่ไม่ปลอดภัยเพื่อเอาชนะคู่แข่ง เราเห็นบริการใหม่ถูกดันออกสู่ตลาดแบบไม่สมบูรณ์หรือมีความสามารถไม่ครบ

การไปให้ความสำคัญกับความเร็วและการเพิ่มจำนวนบริการอย่างรีบเร่ง ผ่านการเข้าซื้อกิจการหรือการเร่งสร้างบริการใหม่ๆ มักนำไปสู่บริการที่ไม่เพียงแต่มีข้อบกพร่องเท่านั้น แต่มีทั้งเรื่องเอกสารที่ไม่เรียบร้อย ระบบที่ทำให้เป็นอัตโนมัติได้ยาก และความไม่สมบูรณ์ของการทำงานร่วมกันระหว่างบริการอื่นๆ ของผู้ให้บริการเองอีกด้วย สิ่งเหล่านี้นำความขัดใจมาสู่ทีมที่พยายามจะส่งมอบซอฟต์แวร์ที่ใช้ความสามารถต่างๆ ที่ผู้ให้บริการคลาวด์สัญญาไว้ แต่กลับพบอุปสรรคตลอดทาง

หลายบริษัทเลือกผู้ให้บริการคลาวด์จากหลายปัจจัย และการตัดสินใจมักมาจากคนระดับสูงในองค์กรที่ไม่ทราบรายละเอียด เพราะฉะนั้นคำแนะนำของเราสำหรับแต่ละทีมคือ อย่าเหมารวมว่าทุกบริการในคลาวด์ของคุณจะมีคุณภาพเท่ากัน ทดสอบขีดความสามารถหลักของบริการก่อนเสมอ และเปิดรับทางเลือกที่เป็นโอเพนซอร์สไว้พิจารณา หรือหากจำเป็นก็เลือกใช้กลยุทธ์ผสมผู้ให้บริการคลาวด์หลายเจ้าเข้าด้วยกันถ้าประเมินแล้วคุ้มค่า ระหว่างการออกผลิตภัณฑ์ให้ทันตลาดได้เร็วขึ้น เทียบกับค่าใช้จ่ายและพลังงานที่ต้องเสียไปในการจัดการคลาวด์หลายเจ้าพร้อมกัน

## การปกป้องห่วงโซ่อุปทานซอฟต์แวร์

องค์กรควรต่อต้านการกำกับดูแลที่ไม่สนใจสภาพปัญหาหรือความต้องการของคนทำงาน ซึ่งการกำกับแบบนี้ใช้เวลานานและต้องใช้คนเพื่อตรวจสอบและอนุมัติ ในทางตรงกันข้าม การทำให้การกำกับดูแลต่างๆ อย่างการปกป้องไลบรารีที่พึงพา การกำหนดนโยบายความมั่นคงด้วยโค้ด หรือการบริหารค่าใช้จ่ายเป็นอัตโนมัติ กลับช่วยปกป้องส่วนที่สำคัญ แต่ไม่ ฉุกเฉิน ของโครงการซอฟต์แวร์เอาไว้

เราเห็นวิวัฒนาการของกระบวนการพัฒนาซอฟต์แวร์กำลังมุ่งสู่การทำให้ทุกอย่างเป็นอัตโนมัติมากขึ้น เริ่มตั้งแต่การมีชุดทดสอบที่เป็นอัตโนมัติ การเชื่อมต่อกันอย่างต่อเนื่อง (continuous integration), การส่งมอบอย่างต่อเนื่อง (continuous delivery), ความสามารถในการกำหนดโครงสร้างพื้นฐานด้วยโค้ด (infrastructure-as-code) และยุคปัจจุบันคือการกำกับดูแลแบบอัตโนมัติ (automated governance)

## การตีความกล่องดำของแมชชีนเลิร์นนิง

แมชชีนเลิร์นนิงมักจะค้นพบคำตอบสำหรับปัญหาที่มนุษย์ไม่สามารถแก้ได้ โดยใช้เทคนิค pattern matching หรือ back propagation หรือเทคนิคอื่นๆ ที่เป็นที่รู้จักกันดี อย่างไรก็ตามแม้จะมีพลังมากมาย โมเดลหลายตัวกลับคลุมเครือโดยธรรมชาติ หมายความว่าผลลัพธ์ของมันไม่สามารถอธิบายออกมาเป็นเหตุเป็นผลได้

นี่เป็นปัญหาในสถานการณ์ที่มนุษย์ควรมีสิทธิ์รู้ว่าการตัดสินใจที่เกิดขึ้น มีที่มาที่ไปอย่างไรหรือเมื่อความเสี่ยงจากการแทรก อคติ กลุ่มตัวอย่าง อัลกอริทึม หรือความลำเอียงในรูปแบบอื่น เข้าไปในโมเดลจะส่งผลกระทบต่อสังคมในวงกว้าง

เราเห็นการก่อตัวของเครื่องมือต่างๆ เช่น What-If และเทคนิคต่างๆ อย่าการทดสอบความลำเอียงทางจริยธรรม ที่ช่วยเราหาข้อจำกัดและทำนายผลลัพธ์ของโมเดลเหล่านี้ ขณะที่พัฒนาการของการตีความกำลังมุ่งไปสู่ทางที่ถูกต้องแล้ว แต่การอธิบายนิวรัลเนตเวิร์คเชิงลึกยังคงเป็นเป้าหมายที่ยากที่จะไปถึง ด้วยเหตุนี้นักวิทยาศาสตร์ข้อมูลจึงเริ่มให้ความสำคัญถึงความสามารถในการอธิบายเป็นเกณฑ์ลำดับหนึ่งในการเลือกโมเดล

## งานพัฒนาซอฟต์แวร์ดั่งทีมกีฬา

นับตั้งแต่เทคโนโลยีเรดาร์ฉบับแรกๆ เราเคยเตือนให้เห็นถึงผลเสียของเครื่องมือและเทคนิคที่พยายามแยกสมาชิกของทีมพัฒนาซอฟต์แวร์ออกจากกัน หรือที่ขัดขวางการรับฟังคำแนะนำและการร่วมมือระหว่างกันมาแล้ว

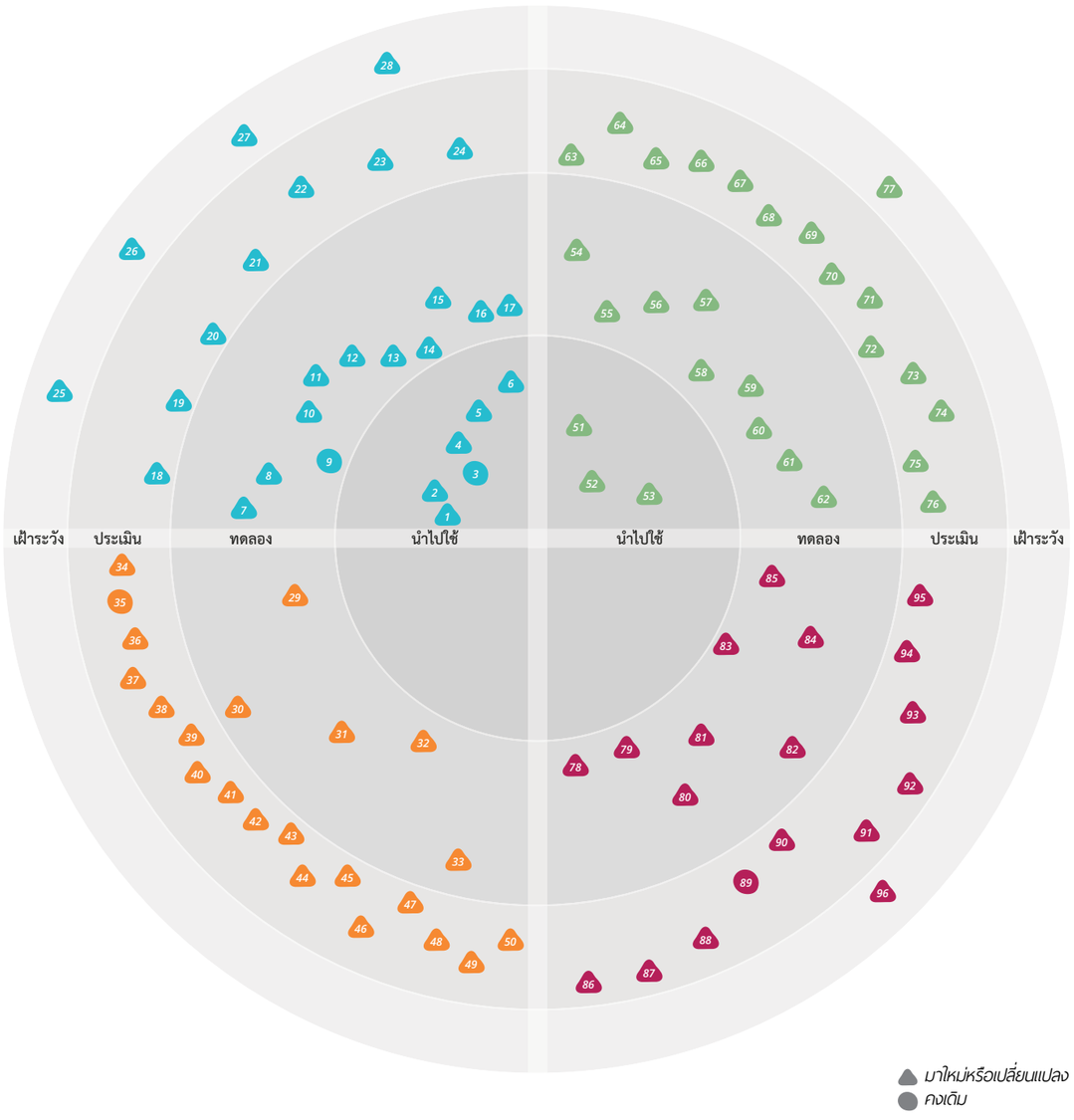
บ่อยครั้งเราจะได้ยินคำอ้างจากผู้เชี่ยวชาญที่เข้ามาใหม่ ผู้ปฏิบัติงาน ผู้รับจ้าง หรือเครื่องมือที่แนะนำว่าควรแยกงานพัฒนาบางส่วนออกมา เพื่อหลบเลี่ยงความโกลาหลในชีวิตประจำวันของงานพัฒนาซอฟต์แวร์ เราไม่เห็นด้วยกับคำอ้างดังกล่าวและคอยหาวิธีที่ดีกว่าอยู่ตลอด ที่จะทำให้ความสัมพันธ์ในทีมพัฒนาซอฟต์แวร์เป็นดั่งทีมกีฬา

การรับฟังคำแนะนำและความคิดเห็นเป็นสิ่งจำเป็นอย่างยิ่ง หากต้องการสร้างสิ่งที่ซับซ้อนอย่างซอฟต์แวร์ แม้มีโครงการที่ต้องการความเชี่ยวชาญเฉพาะด้านมากขึ้นอย่างต่อเนื่อง เรายังพยายามจัดความเชี่ยวชาญเหล่านั้น ให้อยู่กับความร่วมมือและการรับฟังความคิดเห็นของกันและกัน

เราไม่ชอบมีม “วิศวกรสิบเท่า” ที่อยู่ในกระแสสังคม แต่เราชอบสร้างหรือเปิดทางให้เกิด “ทีมสิบเท่า” มากกว่า เราเห็นสิ่งนี้เกิดขึ้นแล้วจากการรวมเอา การออกแบบ วิทยาศาตร์ข้อมูล และความมั่นคง ไปใส่ในทีมที่ทำงานข้ามฟังก์ชันกันได้ จนสิ่งเหล่านี้ถูกทำให้เป็นอัตโนมัติและวางใจได้ ความท้าทายถัดไปคือการรวมกิจกรรมที่เกี่ยวข้องกับการกำกับดูแลและการปฏิบัติตามข้อกำหนดเข้ามาเป็นส่วนหนึ่งของความร่วมมือระหว่างกัน


*© ThoughtWorks, Inc. All Rights Reserved.*

# เรดาร์เทคโนโลยี




## เทคนิค

### นำไปใช้
1. Container security scanning
2. Data integrity at the origin
3. Micro frontends
4. Pipelines for infrastructure as code
5. Run cost as architecture fitness function
6. Testing using real device

### ทดลอง
7. Automated machine learning (AutoML)
8. Binary attestation
9. Continuous delivery for machine learning (CD4ML)
10. Data discoverability
11. Dependency drift fitness function
12. Design systems
13. Experiment tracking tools for machine learning
14. Explainability as a first-class model selection criteria
15. Security policy as code
16. Sidecars for endpoint security
17. Zhong Tai

### ประเมิน
18. BERT
19. Data mesh
20. Ethical bias testing
21. Federated learning
22. JAMstack
23. Privacy-preserving record linkage (PPRL) using Bloom filter
24. Semi-supervised learning loops

### เฝ้าระวัง
25. 10x engineers
26. Front-end integration via artifact
27. Lambda pinball
28. Legacy migration feature parity

## แพลตฟอร์ม

### นำไปใช้

### ทดลอง
29. Apache Flink
30. Apollo Auto
31. GCP Pub/Sub
32. Mongoose OS
33. ROS

### ประเมิน
34. AWS Cloud Development Kit
35. Azure DevOps
36. Azure Pipelines
37. Crowdin
38. Crux
39. Delta Lake
40. Fission
41. FoundationDB
42. GraalVM
43. Hydra
44. Kuma
45. MicroK8s
46. Oculus Quest
47. ONNX
48. Rootless containers
49. Snowflake
50. Teleport

### เฝ้าระวัง

## เครื่องมือ

### นำไปใช้
51. Commitizen
52. ESLint
53. React Styleguidist

### ทดลอง
54. Bitrise
55. Dependabot
56. Detekt
57. Figma
58. Jib
59. Loki
60. Trivy
61. Twistlock
62. Yocto Project

### ประเมิน
63. Aplas
64. asdf-vm
65. AWSume
66. dbt
67. Docker Notary
68. Facets
69. Falco
70. in-toto
71. Kubeflow
72. MemGuard
73. Open Policy Agent (OPA)
74. Pumba
75. Skaffold
76. What-If Tool

### เฝ้าระวัง
77. Azure Data Factory for orchestration

## ภาษาและเฟรมเวิร์ก

### นำไปใช้

### ทดลอง
78. Arrow
79. Flutter
80. jest-when
81. Micronaut
82. React Hooks
83. React Testing Library
84. Styled components
85. Tensorflow

### ประเมิน
86. Fairseq
87. Flair
88. Gatsby.js
89. GraphQL
90. KotlinTest
91. NestJS
92. Paged.js
93. Quarkus
94. SwiftUI
95. Testcontainers

### เฝ้าระวัง
96. Enzyme


*© ThoughtWorks, Inc. All Rights Reserved.*

# เทคนิค

## การพิเคราะห์คอนเทเนอร์เพื่อความมั่นคง
*นำไปใช้*

การนำคอนเทเนอร์ (โดยเฉพาะ Docker) มาใช้อย่างต่อเนื่องเพื่อการดีพลอย ทำให้เทคนิคพิเคราะห์คอนเทเนอร์เพื่อความมั่นคงกลายเป็นเทคนิคที่ขาดไม่ได้ไปเสีย เราจึงย้ายเทคนิคนี้เข้าสู่หมวดนำไปใช้เพื่อสะท้อนให้เห็นถึงความสำคัญนี้

เมื่อคอนเทเนอร์เปิดหนทางใหม่ๆ ให้ปัญหาด้านความมั่นคงเข้ามาได้ จึงเป็นเรื่องจำเป็นที่ต้องใช้เครื่องมือพิเคราะห์ และตรวจสอบคอนเทเนอร์ระหว่างการดีพลอยทุกครั้ง เราชอบให้เครื่องมือพิเคราะห์ต่างๆ ทำงานอย่างอัตโนมัติ โดยเป็นส่วนหนึ่งของไปป์ไลน์ในการดีพลอย

## การทำข้อมูลให้สมบูรณ์ตั้งแต่ต้นกำเนิด
*นำไปใช้*

ในปัจจุบันหากต้องการปลดล็อกการเข้าถึงข้อมูลเพื่อการวิเคราะห์ คำตอบของหลายๆ องค์กรคือการสร้างไปป์ไลน์ข้อมูลที่ซับซ้อนขึ้นมา ไปป์ไลน์นั้นดึงข้อมูลจากหลายๆ แหล่งข้อมูลด้วยกัน นำข้อมูลมาทำความสะอาด จากนั้นแปลงรูปร่างและเคลื่อนย้ายมันไปยังที่ใหม่เพื่อการใช้งานต่อไป

การจัดการข้อมูลด้วยวิธีนี้ มักจะทิ้งงานยากให้กับไปป์ไลน์ที่จะนำข้อมูลไปใช้ต่อที่เหลือ เช่น งานตรวจสอบความสมบูรณ์ของข้อมูลที่เข้ามา และงานทำความสะอาดข้อมูล ที่ต้องสร้างตรรกะซับซ้อนขึ้นมาเพื่อให้ได้ระดับคุณภาพที่ต้องการ

รากฐานของปัญหานี้เกิดจากที่แหล่งกำเนิดข้อมูลไม่มีแรงจูงใจ หรือความรู้สึกรับผิดชอบที่จะทำให้ข้อมูลมีคุณภาพออกมาตั้งแต่ต้น ด้วยเหตุนี้เราจึงสนับสนุนแนวคิดการทำข้อมูลให้สมบูรณ์ตั้งแต่ต้นกำเนิดอย่างแข็งขัน ซึ่งหมายความว่า แหล่งข้อมูลใดที่จัดสรรข้อมูลออกมาที่สามารถนำไปใช้งานต่อได้ ต้องระบุมาตรการด้านคุณภาพของข้อมูลที่ตัวเองรับผิดชอบอย่างชัดเจน และต้องรับประกันมาตรการเหล่านั้นด้วย

เหตุผลหลักที่อยู่เบื้องหลังแนวคิดนี้ คือ ระบบที่ให้กำเนิดและทีมที่ดูแลย่อมมีความคุ้นเคยอย่างใกล้ชิดกับข้อมูลของพวกเขาที่สุด และต้องอยู่ในตำแหน่งที่ดีที่สุด ที่จะแก้ปัญหาจากต้นตอ

สถาปัตยกรรม Data Mesh เก่งกว่านี้ไปอีกขั้น โดยมันเปรียบข้อมูลที่สามารถใช้งานต่อได้เป็นดั่งผลิตภัณฑ์ ที่คุณภาพของข้อมูลและเป้าหมายของข้อมูลเป็นคุณลักษณะจำเป็นของทุกๆ ชุดข้อมูลที่แบ่งปันออกไป

## Micro frontends
*นำไปใช้*

ที่ผ่านมาเราเห็นประโยชน์อย่างมากจากการแนะนำสถาปัตยกรรมแบบไมโครเซอร์วิส ซึ่งมันทำให้ทีมสามารถขยายการส่งมอบได้มากขึ้น จากการที่เราสามารถดีพลอยและบริหารเซอร์วิสต่างๆ แยกกันอย่างเป็นอิสระต่อกัน

น่าเสียดายที่เรายังคงเห็นหลายๆ ทีมประยุกต์ใช้ไมโครเซอร์วิสเฉพาะที่ระบบแบ็กเอนด์เท่านั้น ส่วนแอปพลิเคชันฟรอนท์เอนด์กลับอยู่ในรูปแบบโมโนลิธ ที่มีขนาดใหญ่และทำงานพันกันอยู่ในเว็บแอปพลิเคชันเดียวกัน ซึ่งทำให้ประโยชน์ของไมโครเซอร์วิสถูกลดทอนไปอย่างมาก

ไมโครฟรอนท์เอนด์ได้รับความนิยมอย่างต่อเนื่องตั้งแต่ครั้งแรกที่ถูกแนะนำไป เราเห็นหลายทีมได้นำบางรูปแบบของสถาปัตยกรรมนี้ไปใช้จัดการความซับซ้อนของแอพลิเคชัน ที่มีทีมหลายทีมหรือคนหลายคนพยายามเข้าแก้ไขพร้อมกัน

ในเดือนกันยายนที่ผ่านมา หนึ่งในผู้คิดค้นเทคนิคนี้ได้ตีพิมพ์บทความแนะนำ ที่กลายเป็นแหล่งอ้างอิงสำหรับการทำไมโครฟรอนท์เอนด์ ในบทความแสดงให้เห็นว่า รูปแบบสถาปัตยกรรมนี้สามารถทำให้เป็นจริงได้โดยใช้หลากหลายกลไกในการเขียนเว็บเข้าช่วยเหลือ และยังมีตัวอย่างแอปพลิเคชันที่พัฒนาด้วย React.js ที่ทำไว้ให้อ้างอิงอีกด้วย

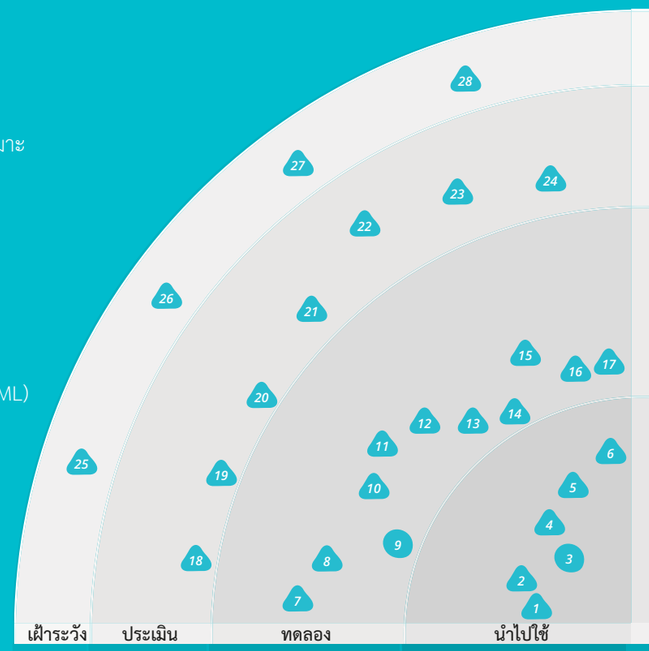
เรามั่นใจว่าสถาปัตยกรรมรูปแบบนี้จะได้รับความนิยมมากขึ้น เมื่อองค์กรขนาดใหญ่พยายามจะถอดงานพัฒนาส่วนต่อ

### นำไปใช้
1. การพิเคราะห์คอนเทเนอร์เพื่อความมั่นคง
2. การทำข้อมูล ให้สมบูรณ์ตั้งแต่ต้นกำเนิด
3. Micro frontends
4. ไปป์ไลน์สำหรับโครงสร้างพื้นฐานด้วยโค้ด
5. ค่าใช้จ่ายในการดำเนินการเป็นฟังก์ชันเพื่อวัดความเหมาะสม ทางสถาปัตยกรรม
6. ทดสอบโดยใช้อุปกรณ์จริง

### ทดลอง
7. แมชชีนเลิร์นนิงแบบอัตโนมัติ (AutoML)
8. การพิสูจน์ไบนารี
9. การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML)
10. ความสามารถในการค้นพบข้อมูล
11. ฟังก์ชันเพื่อวัดความเหมาะสมของการเบี่ยงเบนของสิ่งที่พึ่งพาอ้างอิง
12. Design systems
13. เครื่องมือติดตามการทดลองสำหรับแมชชีนเลิร์นนิง
14. ใช้ความสามารถในการอธิบายเป็นเกณฑ์ลำดับแรกในการเลือกโมเดล
15. นโยบายความมั่นคงด้วยโค้ด
16. ไซด์คาร์สำหรับความมั่นคงของเอนด์พอยท์
17. จงไท่

### ประเมิน
18. BERT
19. Data mesh
20. การทดสอบความลำเอียงทางจริยธรรม
21. Federated learning
22. JAMstack
23. การเชื่อมโยงประวัติข้อมูลแบบเก็บรักษาความเป็นส่วนตัวโดยอาศัยเทคนิคการกรองแบบ Bloom
24. ห่วงการเรียนรู้แบบกึ่งสอน

### เฝ้าระวัง
25. วิศวกรสิบเท่า
26. การประกอบฟรอนท์เอนด์เข้าด้วยกันผ่านแพ็กเกจ
27. แลมด้าเล่นชิ่ง
28. การย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด





*© ThoughtWorks, Inc. All Rights Reserved.*

# เทคนิค

*เครื่องมือแมชชีนเลิร์นนิงแบบอัตโนมัติ ช่วยปิดช่องว่างของอุปสงค์อุปทานความต้องการนักวิทยาศาสตร์ข้อมูลที่เชี่ยวชาญด้านแมชชีนเลิร์นนิง เครื่องมือประเภทนี้มีประโยชน์ตอนเริ่มต้น แต่ให้ผลลัพธ์ที่ดีกว่าหากให้ผู้เชี่ยวชาญเป็นผู้ใช้*

*(AutoML)*

*การพิสูจน์ไบนารี เป็นเทคนิคที่ทำให้การควบคุมความมั่นคงขณะดีพลอยเป็นจริงได้ โดยใช้การเข้ารหัสเพื่อพิสูจน์ว่าอิมเมจไบนารี ใดๆ มีสิทธิ์จะดีพลอยหรือไม่*

*(การพิสูจน์ไบนารี)*

ประสานผู้ใช้ออกจากกันเพื่อกระจายมันสู่ทีมต่างๆ

## ไปป์ไลน์สำหรับโครงสร้างพื้นฐานด้วยโค้ด

*นำไปใช้*

การใช้ไปป์ไลน์การส่งมอบเพื่อควบคุมกระบวนการออกซอฟต์แวร์ กลายเป็นเรื่องที่ปฏิบัติกันอย่างแพร่หลายทั่วไป แต่เรายังสามารถนำเครื่องมือ CI/CD มาใช้ทดสอบระบบโครงสร้างพื้นฐานได้อีกด้วย เช่น ทดสอบการตั้งค่าของเซิร์ฟเวอร์ (เช่น Chef cookbooks, Puppet modules, Ansible playbooks), ทดสอบการสร้างอิมเมจของเซิร์ฟเวอร์ (เช่น Packer), ทดสอบการสร้างสภาพแวดล้อม (เช่น Terraform, CloudFormation) และทดสอบการเชื่อมต่อระหว่างสภาพแวดล้อมต่างๆ

การใช้ไปป์ไลน์สำหรับโครงสร้างพื้นฐานด้วยโค้ดทำให้คุณทดสอบและพบข้อบกพร่องได้ก่อนที่การเปลี่ยนแปลงจะมีผลกับสภาพแวดล้อมนั้นๆ ซึ่งรวมทั้งสภาพแวดล้อมสำหรับการพัฒนา และสภาพแวดล้อมสำหรับการทดสอบ ด้วยนอกจากนี้ มันยังสร้างความมั่นใจให้เราว่า เครื่องมือสำหรับโครงสร้างพื้นฐานเหล่านี้จะทำงานอย่างคงเส้นคงวาเสมอ เมื่อทำงานผ่าน CI / CD แทนการสั่งงานผ่านเครื่องของ นักพัฒนาเอง สำหรับพวกเราแล้วการนำเทคนิคนี้มาใช้ได้ผลลัพธ์ที่น่าพึงพอใจ

## ค่าใช้จ่ายในการดำเนินการเป็นฟังก์ชันเพื่อวัดความเหมาะสมทางสถาปัตยกรรม

*นำไปใช้*

การประเมิน การติดตาม และการวางแผนค่าใช้จ่ายการดำเนินการบนโครงสร้างพื้นฐานแบบคลาวด์อย่างอัตโนมัติเป็นเรื่องที่จำเป็นสำหรับองค์กรในปัจจุบัน รูปแบบการกำหนดราคาอันหลักแหลมของผู้ให้บริการคลาวด์ ผนวกกับการงอกขยายของตัวแปรที่เกี่ยวข้องกับราคาได้เรื่อยๆ และธรรมชาติของสถาปัตยกรรมในปัจจุบันที่มีความยืดหยุ่นสูง สามารถนำไปสู่ค่าใช้จ่ายการดำเนินการที่แพงจนน่าตกใจยกตัวอย่างเช่น บริการ serverless ที่ราคาคำนวณจากจำนวนการเรียกใช้ API หรือโซลูชันสำหรับ event-streaming ที่ราคาคำนวณจากปริมาณการรับส่งข้อมูลที่เกิดขึ้น หรือคลัสเตอร์สำหรับประมวลผลข้อมูลที่ราคาคำนวณจากจำนวนงานที่ทำอยู่ บริการทั้งหมดนี้ล้วนแล้วมีธรรมชาติที่ยืดหยุ่น ซึ่งเปลี่ยนแปลงไปตามเวลา เช่นเดียวกับสถาปัตกรรมที่วิวัฒน์ได้

เมื่อทีมของเราจัดการโครงสร้างพื้นฐานแบบคลาวด์ การใช้ค่าใช้จ่ายในการดำเนินการเป็นฟังก์ชันเพื่อวัดความเหมาะสมทางสถาปัตยกรรม (run cost as architecture fitness function) คือหนึ่งในสิ่งแรกๆ ที่พวกเขาทำ นั่นหมายความว่าทีมของเราสามารถสังเกตค่าใช้จ่ายของบริการที่ทำงานอยู่ เปรียบเทียบกับคุณค่าที่ถูกส่งมอบ

เมื่อพวกเขาเห็นการเบี่ยงเบนออกจากราคาที่คาดหวังหรือยอมรับได้ ก็จะมีการปรึกษาหารือกันว่าต้องวิวัฒน์สถาปัตยกรรมไปในทิศทางใดให้ดีขึ้น โดยที่การสังเกตและการคำนวณของค่าใช้จ่ายในการดำเนินการทั้งหมดนั้นถูกทำให้เป็นอัตโนมัติ

## ทดสอบโดยใช้อุปกรณ์จริง

*นำไปใช้*

เมื่อนำการส่งมองอย่างต่อเนื่องมาปฏิบัติใช้จนสำเร็จ ทีมจะพยายามทำให้ทุกสภาพแวดล้อมเพื่อการทดสอบมีความใกล้เคียงกับสภาพแวดล้อมโปรดักชันมากที่สุดเท่าที่จะมากได้ การทำเช่นนี้ช่วยให้ทีมค้นพบข้อบกพร่องได้ล่วงหน้าก่อนที่ปัญหาจะเกิดในโปรดักชัน

ไม่ใช่แค่กับซอฟต์แวร์ทั่วไป แต่แนวคิดนี้สามารถประยุกต์ใช้กับซอฟต์แวร์ที่ต้องยุ่งกับฮาร์ดแวร์ได้เช่นเดียวกัน ยกตัวอย่างเช่น อุปกรณ์ IoT หรืออุปกรณ์ฝังตัว ดั้งนั้นการทดสอบด้วยอุปกรณ์จริงจะช่วยหลีกเลี่ยงปัญหาดังกล่าวได้ โดยนำมันมาเป็นส่วนหนึ่งของไปป์ไลน์

## แมชชีนเลิร์นนิงแบบอัตโนมัติ (AutoML)

*ทดลอง*

พลังและสัญญาของแมชชีนเลิร์นนิงทำให้ตลาดงานมีอุปสงค์ความต้องการผู้เชี่ยวชาญในด้านนี้สูงกว่าอุปทานอย่างมากเพื่อหลีกปัญหาดังกล่าว เราเห็นการก่อตัวของเครื่องมือประเภทแมชชีเลิร์นนิงแบบอัตโนมัติ (Automated machine learning: AutoML) มากขึ้น โดยเครื่องมือเหล่านี้อ้างว่าสามารถช่วยคนที่ไม่ใช่ผู้เชี่ยวชาญให้ทำงานด้านนี้ได้ง่ายขึ้น โดยทำให้กระบวนการสร้างโมเดลทั้งวงจรเป็นไปอย่างอัตโนมัติ ตัวอย่างเครื่องมือประเภทนี้ เช่น Google's AutoML DataRobot และ H2O AutoML

แม้ผลลัพธ์ที่ได้จากเครื่องมือประเภทนี้ดูมีแนวโน้มที่ดีก็ตาม แต่เราขอเตือนภาคธุรกิจว่า อย่าเข้าใจไปว่าโมเดลที่ได้นั้นสมบูรณ์สิ้นสุดแล้ว ทางเว็บไซต์ H2O ได้กล่าวไว้ว่า "เพื่อสร้างโมเดลที่ทรงประสิทธิภาพ องค์ความรู้และพื้นฐานทางวิทยาศาสตร์ข้อมูลบางส่วนยังคงมีความจำเป็น"
การเชื่ออย่างสนิทใจกับผลลัพธ์ที่ได้จากเทคนิคนี้ อาจนำมาซึ่งความเสี่ยงที่จะเกิดความลำเอียงทางจริยธรรม หรือการตัดสินใจที่ให้ผลเสียกับคนกลุ่มน้อยของสังคม ภาคธุรกิจอาจใช้เครื่องมือเหล่านี้สร้างโมเดลที่มีประโยชน์ตอนเริ่มต้นได้ แต่เราขอส่งเสริมให้ปรึกษาผู้เชี่ยวชาญที่มีประสบการณ์เพื่อตรวจสอบความสมบูรณ์และปรับปรุงโมเดลที่ได้อีกทีหนึ่ง

## การพิสูจน์ไบนารี

*ทดลอง*

จากการใช้คอนเทเนอร์ในการดีพลอยเซอร์วิสเป็นกลุ่มใหญ่โดยทีมต่างๆ ที่ทำงานเป็นอิสระต่อกัน และจากความเร็วที่เพิ่มขึ้นของการส่งมอบซอฟต์แวร์อย่างต่อเนื่องจนกลายเป็นเรื่องปกติของหลายๆ องค์กร ทำให้ความต้องการซอฟต์แวร์ด้านความมั่นคง ที่สามารถทำงานได้อย่างอัตโนมัติขณะดีพลอย เกิดขึ้น

การพิสูจน์ไบนารี เป็นเทคนิคที่ทำให้การควบคุมความมั่นคงขณะดีพลอยเป็นจริงได้ โดยใช้การเข้ารหัสเพื่อพิสูจน์ว่าอิมเมจไบนารีใดๆ มีสิทธิ์จะดีพลอยหรือไม่

เมื่อใช้เทคนิคนี้ การวางตราประทับเพื่อรับรองไบนารีอาจมาจากกระบวนการประกอบซอฟต์แวร์ที่เป็นอัตโนมัติทั้งหมด หรือจะให้ทีมผู้ดูแลความมั่นคงเป็นผู้ลงชื่อรับรองตัวไบนารี หลังจากซอฟต์แวร์ผ่านการทดสอบ การตรวจสอบคุณภาพ และการรับรองสิทธิ์การดีพลอยจากทีมอื่นๆ ก็ได้

บริการอย่าง GCP Binary Authorization โดย Grafeas และเครื่องมืออย่าง in-toto และ Docker Notary รองรับการประทับตราและการพิสูจน์ตราประทับของอิมเมจก่อนที่จะดีพลอย


*© ThoughtWorks, Inc. All Rights Reserved.*

## การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML)
*ทดลอง*

ด้วยความนิยมที่เพิ่มขึ้นของแอปพลิเคชันที่มีพื้นฐานเป็นแมชชีนเลิร์นนิง และความซับซ้อนทางเทคนิคในการสร้างมัน ทีมเราใช้ การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (continuous delivery for machine learning: CD4ML) หนักมาก เพื่อส่งมอบแอปพลิเคชันที่มีคุณลักษณะที่มั่นคง รวดเร็ว และยั่งยืน

CD4ML นำหลักการและแนวปฏิบัติของ CD มาใช้สร้างแอปพลิเคชันที่เกี่ยวกับแมชชีนเลิร์นนิง โดยช่วยลดรอบเวลาที่ยาวนานของการทำงานทั้งหมดลง มันทำเช่นนั้นได้โดยการกำจัดการส่งงานต่อด้วยมือระหว่างทีมต่างๆ ตั้งแต่ทีมพัฒนา สู่วิศวกรข้อมูล สู่นักวิทยาศาสตร์ข้อมูล และสู่วิศวกรแมชชีนเลิร์นนิง ครอบคลุมกระบวนการตั้งแต่ต้นจนจบของการสร้างโมเดล ไปจนถึงการดีพลอยโมเดลในโปรดักชัน

ทีมงานของเราประสบความสำเร็จในการนำ CD4ML มาทำให้ทุกขั้นตอนเป็นอัต โนมัติ ตั้งแต่ การกำหนดเวอร์ชัน การทดสอบ และการดีพลอยทุกส่วนประกอบของแอปพลิเคชันที่มีพื้นฐานเป็นแมชชีนเลิร์นนิง ตั้งแต่ ส่วนข้อมูล ตัวโมเดล และโค้ดที่เกี่ยวข้อง

## ความสามารถในการค้นพบข้อมูล
*ทดลอง*

ส่วนที่ยากลำบากที่สุด ในขั้นตอนการทำงานของนักวิทยาศาสตร์ข้อมูลและนักวิเคราะห์ คือ ขั้นตอนการระบุตำแหน่งของข้อมูลที่ ต้องการ แล้วพยายามทำความเข้าใจกับมัน เพื่อประเมินความน่าเชื่อถือของข้อมูลว่าควรนำมาใช้ต่อหรือไม่

ขั้นตอนเหล่านี้เป็นเรื่องที่ยากลำบาก เพราะข้อมูลดิบที่ได้มาไม่มีอภิภัณฑ์ข้อมูลเสริม (meta-data) ถึงแหล่งที่มาของข้อมูลติดมาด้วย การขาดความสามารถที่เพียงพอในการค้นหาและระบุตำแหน่งของข้อมูล ก็เป็นอีกเหตุผลหนึ่ง

พวกเราอยากส่งเสริมให้ทีมที่ทำหน้าที่ให้บริการข้อมูลเพื่อการวิเคราะห์ หรือคนที่กำลังจะสร้างแพลตฟอร์มข้อมูลขึ้นมาใหม่ คำนึงถึงความสามารถในการค้นพบข้อมูล ให้เป็นความสามารถสำคัญที่พวกเขาควรทำได้ในสภาพแวดล้อมนั้นๆ เพื่อให้การระบุตำแหน่งข้อมูลทำได้โดยง่าย สามารถตรวจสอบคุณภาพของข้อมูล สามารถทำความเข้าใจโครงสร้างหรือแหล่งที่มาและสามารถเข้าถึงมันได้

แต่เดิมแล้ว ความสามารถนี้จะอยู่ในโซลูชันที่ใช้รูปแบบแคตตาล็อคข้อมูลขนาดใหญ่ในการค้นหาแต่ในช่วงไม่กี่ปีที่ผ่านมา การเติบโตของโครงการโอเพนซอร์สต่างๆ กำลังปรับปรุงประสบการณ์ของนักพัฒนาให้ดีขึ้น ทั้งฝั่งผู้ให้ข้อมูลเองและฝั่งผู้รับข้อมูลไปใช้งานต่อ เพื่อให้การค้นพบข้อมูลทำได้โดยง่าย เครื่องมืออย่าง Amundsen โดย Lyft และ WhereHows โดย LinkedIn เป็นตัวอย่างของเครื่องมือประเภทนี้

สิ่งที่เราอยากจะเห็น คือการเปลี่ยนพฤติกรรมของผู้ให้ข้อมูลโดยเจตนาให้ข้อมูลอภิภัณฑ์เสริมที่ช่วยทำให้ข้อมูลถูกค้นพบได้ง่าย แทนการพึ่งพาเครื่องมือค้นหาที่อนุมานข้อมูลอภิภัณฑ์บางส่วนจากฐานข้อมูลของแอปพลิเคชันใดๆ

## ฟังก์ชันเพื่อวัดความเหมาะสมของการเบี่ยงเบนของสิ่งที่พึ่งพาอ้างอิง
*ทดลอง*

ทีมและองค์กรจำนวนมากไม่มีแบบแผนหรือวิธีที่แน่นอนในการติดตามสิ่งที่ซอฟต์แวร์ของเค้าพึ่งพาต่อกันทางเทคนิค (technical dependency) ประเด็นต่างๆ จะไม่แสดงตัวจนกว่าซอฟต์แวร์มีความจำเป็นต้องเปลี่ยนแปลงแก้ไขซึ่ง ณ จุดดังกล่าวทั้งไลบรารี, API ต่างๆ หรือส่วนประกอบอื่นๆ ที่ ล้าสมัยไปแล้วอาจทำให้เกิดปัญหาหรือทำให้การดำเนินการล่าช้ากว่ากำหนด

ฟังก์ชันเพื่อวัดความเหมาะสมของการเบี่ยงเบนของสิ่งที่พึ่งพาอ้างอิง (Dependency drift fitness function) เป็นเทคนิคเฉพาะในการทำฟังก์ชันเพื่อวัดความเหมาะสมทางสถาปัตยกรรมเชิงวิวัฒน์ เพื่อติดตามสิ่งที่พึ่งพาอ้างอิงทั้งหลายในระบบตลอดช่วงอายุ ฉะนั้นมันสามารถบ่งชี้ถึงงานที่อาจต้องทำเพิ่มในอนาคต และบอกได้ว่าความเสี่ยงต่างๆ มันดีขึ้นหรือแย่ลง

## Design systems
*ทดลอง*

เมื่อการพัฒนาแอปพลิเคชันมีความซับซ้อนและเปลี่ยนแปลงได้ตลอด การส่งมอบอย่างมีประสิทธิผล โดยคงรักษาคุณสมบัติการเข้าถึงได้ดีของผู้ใช้ และการทำให้ผลิตภัณฑ์มีหน้าตาโดยรวมที่สอดคล้องกันเป็นเรื่องที่ยาก

Design systems เป็นการกำหนดการออกแบบ รูปแบบคอมโพเนนต์ไลบรารี ดีไซน์ที่ดี และแนวปฏิบัติทางวิศวกรรมที่ยอมรับร่วมกัน เพื่อทำให้เรามั่นใจว่าการพัฒนาผลิตภัณฑ์มีความสอดคล้องไปในทำนองเดียวกัน

เราพบว่า design system เป็นเครื่องมือเสริมที่มีประโยชน์มากในงานพัฒนาผลิตภัณฑ์ เวลาที่ต้องทำงานข้ามทีมหรือระหว่างบทบาทหน้าที่กัน เพราะทีมสามารถเพ่งความสนใจไปกับสิ่งท้าทายที่เป็นยุทธศาสตร์ของตัวผลิตภัณฑ์เอง โดยไม่เสียความสนใจหรือเสียแรงเพื่อเริ่มสร้างทุกอย่างจากศูนย์ทุกครั้งที่ต้องการคอมโพเนนต์ใหม่

ทั้งนี้ประเภทของคอมโพเนนต์และเครื่องมือที่ช่วยสร้าง design systems นั้นผันแปรได้อย่างมาก

## เครื่องมือติดตามการทดลองสำหรับแมชชีนเลิร์นนิง
*ทดลอง*

งานในแต่ละวันของแมชชีนเลิร์นนิงมักจะวนเวียนอยู่กับการทดลองปรับแต่งตัวแปรต่างๆ เช่น ปรับวิธีเลือกโมเดล ปรับรูปแบบการเชื่อมต่อของเนตเวิร์ค ปรับข้อมูลที่จะนำมาใช้ฝึกสอน ปรับปรุงประสิทธิภาพการทำงาน หรือปรับแต่งโมเดล

เนื่องจากหลายๆ ครั้ง โมเดลที่ใช้ยังยากเกินกว่าจะแปลออกมาหรืออธิบายได้ นักวิทยาศาสตร์ข้อมูลจึงจำเป็นต้องใช้ประสบการณ์และสัญชาติญาณในการตั้งสมมุติฐานเพื่อปรับแต่งค่าแล้ววัดว่าผลกระทบของการปรับแต่งนั้นส่งผลอย่างไรกับประสิทธิภาพโดยรวมของโมเดล

เมื่อโมเดลเหล่านี้ถูกนำมาใช้จนกลายเป็นเรื่องทั่วไปมากขึ้นในระบบธุรกิจ เราเลยเห็นเครื่องมือติดตามการทดลอง

# เทคนิค

*นิวรัลเนตเวิร์คเชิงลึก ได้สาธิตความสามารถในการจดจำที่น่าทึ่ง และความแม่นยำที่โดดเด่นผ่านโจทย์ที่หลากหลาย แต่ยิ่งมีการนำไปใช้มากขึ้น ยิ่งต้องให้ความสำคัญกับการบอกได้ว่าการตัดสินใจของมันมีที่ไปที่มาอย่างไร*

*(ใช้ความสามารถในการอธิบายเป็นเกณฑ์ลำดับแรกในการเลือกโมเดล)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# เทคนิค

*ความซับซ้อนของภาพรวมเทคโนโลยีในปัจจุบัน เรียกร้องให้เราปฏิบัติกับนโยบายความมั่นคงด้วยโค้ด คือ การกำหนดและบันทึกมันในเวอร์ชันคอนโทรล ตรวจสอบ ดีพลอยมันได้อย่างอัตโนมัติ รวมไปถึงการวัดประสิทธิภาพ*

*(นโยบายความมั่นคงด้วยโค้ด)*

*แก่นของจงไท่คือ วิธีการส่งมอบโมเดลทางธุรกิจที่พร้อมนำไปใช้ต่อ ถูกออกแบบมาเพื่อช่วยให้ธุรกิจเล็กๆ สามารถส่งมอบบริการชั้นเลิศ โดยไม่จำเป็นต้องลงทุนพัฒนาโครงสร้างพื้นฐานของตัวเอง เหมือนองค์กรใหญ่*

*(จงไท่)*

สำหรับแมชชีนเลิร์นนิงก่อตัวขึ้น ซึ่งมันช่วยให้ผู้ทำการทดลองสามารถติดตามผลและทำงานอย่างมีระบบแบบแผน

ถึงแม้จะยังไม่มีผู้ชนะที่ชัดเจน แต่เครื่องมืออย่าง MLflow หรือ Weights & Biases และแพลตฟอร์มอย่างComet หรือ Neptune ได้นำความเคร่งครัดและความสามารถในการทำซ้ำมาสู่กระบวนการคิดและการทำงานของนักพัฒนาแมชชีนเลิร์นนิง

เครื่องมือเหล่านี้ยังช่วยสนับสนุนให้เกิดความร่วมมือระหว่างกันและกัน เปลี่ยนวิธีการทำงานของวิทยาศาสตร์ข้อมูลจากที่อาศัยความมุมานะโดยลำพัง สู่การทำงานร่วมกันเป็นทีม

## ใช้ความสามารถในการอธิบายเป็นเกณฑ์ลำดับแรกในการเลือกโมเดล

*ทดลอง*

นิวรัลเนตเวิร์คเชิงลึกได้สาธิตความสามารถของมันในการจดจำที่น่าทึ่งและความแม่นยำที่โดดเด่น ผ่านการประยุกต์ใช้กับโจทย์ที่หลากหลาย

ถ้าให้การฝึกสอนที่มากพอและรูปแบบการเชื่อมต่อที่เหมาะสมแล้ว โมเดลเหล่านี้สามารถเทียบเคียง หรือก้าวข้ามขีดความสามารถของมนุษย์ในบางขอบเขตปัญหาเลยทีเดียว

อย่างไรก็ตาม โดยธรรมชาติของนิวรัลเนตเวิร์คนั้นคลุมเครือต่อความเข้าใจของมนุษย์ แม้บางส่วนของโมเดลจะสามารถนำกลับมาใช้ใหม่ได้ผ่านเทคนิคการถ่ายทอดการเรียนรู้ น้อยครั้งที่ความรู้เหล่านั้นจะมีเหตุผลพอที่มนุษย์จะเข้าใจความหมายของมันได้

ในทางตรงกันข้าม โมเดลที่สามารถอธิบายได้ คือโมเดลที่ทำให้เราบอกได้ว่าการตัดสินใจมีที่มาอย่างไร ตัวอย่างเช่น จากการใช้ต้นไม้ตัดสินใจ (desicion tree) เราสามารถตีความผลลัพธ์ที่ได้เพื่ออธิบาย กระบวนการที่ใช้จำแนกประเภทข้อมูล ความสามารถในการอธิบายกลาย เป็นเรื่องที่จำเป็นขึ้นในบางอุตสาหกรรมที่ต้องถูกดูแล หรือเมื่อเกิดผลกระทบทางจริยธรรมจากการตัดสินใจเป็นเรื่องที่ต้องกังวล เนื่องจากโมเดลเหล่านี้ได้เข้าไปอยู่ในระบบธุรกิจที่สำคัญอย่างกว้างขวาง จึงเป็นเรื่องสำคัญที่เราควรจะคำนึงถึงความสามารถในการอธิบายเป็นเกณฑ์ลำดับแรกในการเลือกโมเดล

แม้จะทรงพลังแค่ไหนก็ตาม แต่นิวรัลเนตเวิร์คเชิงลึกอาจไม่ใช่ทางเลือกที่เหมาะสมเมื่อความสามารถในการอธิบายเป็นข้อกำหนดที่เข้มงวด

## นโยบายความมั่นคงด้วยโค้ด

*ทดลอง*

นโยบายความมั่นคงคือกฎและขั้นตอนที่ช่วยป้องกันระบบของเราจากภัยคุกคามและการรบกวนต่างๆ ตัวอย่างเช่น นโยบายด้านสิทธิ์การเข้าถึงสามารถกำหนดและบังคับว่า ใครสามารถเข้าถึงเซอร์วิสและทรัพยากรใด ในสถานการณ์ไหนได้บ้าง หรือนโยบายความมั่นคงของระบบเครือข่ายสามารถจำกัดอัตราการรับส่งข้อมูลของแต่ละเซอร์วิสได้อย่างยืดหยุ่น

ความซับซ้อนของภาพรวมเทคโนโลยีทั้งหมดในปัจจุบัน เรียกร้องให้เราปฏิบัติกับนโยบายความมั่นคงด้วยโค้ดแทน ซึ่งคือการที่เราสามารถกำหนดนโยบายในรูปแบบโค้ด บันทึกมันไว้เป็นหลักฐานในเวอร์ชันคอนโทรล ซึ่งทำให้การตรวจสอบหรือการดีพลอยนโยบายทำได้อย่างอัตโนมัติ รวมไปถึงการเฝ้าติดตามวัดผลประสิทธิภาพของมัน

เครื่องมือเช่น Open Policy Agent หรือแพลตฟอร์มอย่าง Istio ช่วยให้การกำหนดนโยบายและกลไกการบังคับใช้ทำได้อย่างยืดหยุ่น และรองรับแนวปฏิบัติแบบนโยบายรักษาความมั่นคงด้วยโค้ด

## ไซด์คาร์สำหรับความมั่นคงของเอนด์พอยท์

*ทดลอง*

โซลูชันที่เราทำขึ้นทุกวันนี้ ต่างก็ทำงานอยู่ในสภาพแวดล้อมที่ความซับซ้อนเพิ่มขึ้นไม่หยุด อย่าง polycloud หรือ hybrid-cloud ซึ่งแต่ละเซอร์วิสเองและแต่ละส่วนประกอบต่างก็อยู่กันแบบกระจายตัว

ในสภาพการณ์เช่นนี้แล้ว เพื่อรักษาความมั่นคงเราจึงประยุกต์สองหลักการสำคัญตั้งแต่เริ่มลงมือ ได้แก่ หลักการไม่ไว้ใจเนตเวิร์คและต้องพิสูจน์ตัวตนก่อนเสมอ (zero trust network) และหลักการให้สิทธิ์ให้น้อยที่สุดเท่าที่จำเป็นเท่านั้น (least privilege)

ไซด์คาร์สำหรับความมั่นคงของเอนด์พอยท์ เป็นเทคนิคที่เราใช้อยู่เป็นประจำเพื่อประยุกต์หลักการดังกล่าว เพื่อบังคับใช้การควบคุมด้านความมั่นคงไปในทุกเอนด์พอยท์ของทุกส่วนประกอบเลย ตั้งแต่ API ของเซอร์วิส, API ของที่เก็บข้อมูล และ API ของตัวควบคุม Kubernetes เป็นต้น

เราทำได้โดยใช้เทคนิคไซด์คาร์ที่อยู่นอกโพรเซส (out-of-process sidecar) (เป็นโพรเซสหรือคอนเทเนอร์ที่ถูกดีพลอยควบคู่ไปกับแต่ละเซอร์วิสหลัก โดยทั้งคู่จะใช้ execution context, host และ identity ร่วมกัน) ซึ่งมีเครื่องมืออย่าง Open Policy Agent และ Envoy ที่ช่วยให้เทคนิคนี้เป็นจริงได้

ไซด์คาร์สำหรับความมั่นคงของเอนด์พอยท์ ช่วยให้วงของความน่าเชื่อถือแคบลงอยู่ที่ระดับเอนด์พอยท์ใดๆ ที่สนใจเท่านั้น แทนที่จะกระจายวงกว้างไปทั่วทั้งเครือข่าย พวกเราอยากเห็นแต่ละทีมรับผิดชอบการตั้งค่าความมั่นคงของเอนด์พอยท์ของไซด์คาร์กันเอง แทนที่จะมาจากทีมส่วนกลาง

## จงไท่

*ทดลอง*

จงไท่ (Zhong Tai) เป็นที่พูดถึงในวงการ IT ของจีนมาเป็นเวลาหลายปีแล้ว แต่ก็ยังไม่ได้เป็นที่สนใจมากนักในฝั่งตะวันตก แก่นของจงไท่คือวิธีการส่งมอบโมเดลทางธุรกิจที่พร้อมนำไปใช้ต่อ ซึ่งถูกออกแบบมาเพื่อช่วยให้ธุรกิจเล็กๆ สามารถส่งมอบบริการชั้นเลิศโดยไม่จำเป็นต้องลงทุนพัฒนาโครงสร้างพื้นฐานของตัวเอง เหมือนองค์กรใหญ่ และช่วยองค์กรที่มีธุรกิจอยู่แล้วนำเสนอบริการล้ำสมัยออกสู่ตลาดได้เร็วจนน่าตกใจ

กลยุทธจงไท่ ถูกนำเสนอเป็นครั้งแรกโดยบริษัทอาลีบาบา และต่อมาบริษัทอินเตอร์เนตของจีนหลายแห่งก็ได้นำแนวคิดนี้ไปปรับใช้กันอย่างแพร่หลาย เนื่องจากโมเดลทางธุรกิจของพวกเขามีความเป็นดิจิตอลโดยกำเนิดซึ่งสามารถนำไปทำซ้ำกับตลาดใหม่ๆ และกลุ่มธุรกิจอื่นๆ ได้ ทุกวันนี้บริษัทต่างๆ ในจีนเริ่มนำจงไท่ไปใช้เป็นเครื่องมือเพื่อเปลี่ยนแปลงตัวเองให้เข้าสู่ความเป็นดิจิตอลมากขึ้น


*© ThoughtWorks, Inc. All Rights Reserved.*

## BERT
*ประเมิน*

BERT (Bidirectional Encoder Representations from Transformers) เป็นวิธีการใหม่ในการฝึกสอนเพื่อสร้างตัวแทนของภาษาธรรมชาติ ซึ่งผลงานชิ้นนี้ถูกตีพิมพ์โดยนักวิจัยที่ Google เมื่อเดือนตุลาคม 2018 จากผลการทดสอบอันยอดเยี่ยมในงานต่างๆ ของภาษาธรรมชาติ BERT ทำให้คนเห็นภาพรวมและความเป็นไปได้ของเทคโนโลยีการประมวลผลภาษาธรรมชาติเปลี่ยนไปอย่างมีนัยสำคัญ

ด้วยสถาปัตยกรรมแบบแปลงรูป (Transformer) ทำให้ระหว่างการฝึกสอน BERT สามารถเรียนรู้บริบทของคำได้ โดยตีความจากคำรอบข้างทั้งฝั่งซ้ายและขวาของคำนั้นๆ
นอกจากนี้ Google ได้ปล่อยโมเดลที่สร้างด้วย BERT สำหรับการใช้งานทั่วไปให้คนที่สนใจนำไปใช้งานต่อ โดยใช้ฐานข้อความที่ยังไม่ถูกให้ความหมายขนาดใหญ่ในการฝึกสอน ซึ่งหนึ่งในนั้นมี Wikipedia รวมอยู่ด้วย นักพัฒนาสามารถนำโมเดลนี้ไปใช้หรือปรับแต่งให้ดีขึ้น โดยเอาไปฝึกต่อกับฐานข้อความสำหรับงานเฉพาะใดๆ เพื่อให้ได้ผลลัพธ์ที่ดียิ่งขึ้น

ในเรดาร์ฉบับเมษายน 2019 เราได้กล่าวถึงวิธีการถ่ายทอดการเรียนรู้ใน NLP ไปแล้ว BERT และสิ่งที่ถูกพัฒนาต่อยอดจากมันจะเดินหน้าทำให้การถ่ายทอดการเรียนรู้ใน NLP เป็นพื้นที่ที่น่าตื่นเต้นยิ่งขึ้น เนื่องจากมันช่วยลดภาระการจำแนกประเภทข้อความลงอย่างมีนัยสำคัญ

## Data mesh
*ประเมิน*

Data mesh เป็นมุมมองแนวคิดการออกแบบสถาปัตยกรรมเพื่อปลดล็อคการเข้าถึงข้อมูลเพื่อการวิเคราะห์และรองรับการขยายตัวของข้อมูลที่มีปริมาณมหาศาลได้

ชุดโดเมนข้อมูลในระบบแบบกระจาย โดยธรรมชาติจะมีปริมาณเพิ่มขึ้นอย่างต่อเนื่อง สถาปัตยกรรมแบบ Data mesh ช่วยให้การเข้าถึงข้อมูลเหล่านี้ทำได้อย่างรวดเร็ว เพื่อรองรับสภาพการณ์ที่ความต้องการเข้าถึงข้อมูลมีสูงขึ้นได้ไม่รู้จบ และการขอข้อมูลมาได้จากหลายแหล่งที่มาในระบบ เช่น จากระบบแมชชีนเลิร์นนิง ระบบวิเคราะห์ข้อมูล หรือจากแอปพลิเคชันที่ต้องประมวลผลข้อมูลจำนวนมากอย่างรวดเร็ว

ด้วยมุมมองแนวคิดแบบใหม่ ทำให้ Data mesh มีคำตอบให้กับปัญหาที่มักพบเจอกับ data lake แบบดั้งเดิมที่ข้อมูลอยู่รวมศูนย์กัน หรือปัญหาจากสถาปัตยกรรมแบบแพลตฟอร์มข้อมูลได้ทั้งหมด

Data mesh มองสถาปัตยกรรมในมุมใหม่โดยอาศัยคุณสมบัติจากสถาปัตยกรรมแบบกระจายยุคใหม่ ที่พิจารณาให้เรื่องโดเมนมีความสำคัญเป็นลำดับแรก ประยุกต์แนวคิดแบบคิดเชิงแพลตฟอร์มเพื่อวางโครงสร้างพื้นฐานทางข้อมูลที่ผู้ใช้สามารถให้บริการตนเองได้ มองข้อมูลดั่งผลิตภัณฑ์ และทำงานบนมาตรฐานเปิดเพื่อสร้างระบบนิเวศ ที่ยอมให้ผลิตภัณฑ์ข้อมูลในระบบกระจายสามารถแลกเปลี่ยนข้อมูลจากกันและกันได้สะดวก

## การทดสอบความลำเอียงทางจริยธรรม
*ประเมิน*

ในปีที่ผ่านมา เราเห็นผู้คนให้ความสนใจแมชชีนเลิร์นนิงในอีกแง่มุม โดยเฉพาะในหัวข้อที่เกี่ยวข้องกับนิวรัลเนตเวิร์คเชิงลึก ความน่าตื่นตาตื่นใจจากความสามารถที่น่าทึ่งของโมเดลเหล่านี้ได้ขับเคลื่อนให้เกิดพัฒนาการของเครื่องมือและเทคนิคต่างๆ แต่ในระยะหลังเริ่มมีความกังวลก่อตัวขึ้น ว่าโมเดลที่ได้อาจสร้างผลร้ายทางสังคมโดยไม่เจตนา ตัวอย่างเช่น ในธุรกิจขอสินเชื่อเมื่อโมเดลผ่านการฝึกสอนให้ตัดสินใจเพื่อผลกำไรสูงสุดทางธุรกิจ มันอาจเลือกตัดสิทธิ์ผู้สมัครที่ตกอยู่กลุ่มผู้ด้อยโอกาสทิ้งไปทั้งหมด

กระนั้นยังมีเรื่องน่ายินดี เราเห็นการพูดถึงวิธีทดสอบความลำเอียงทางจริยธรรม (Ethical bias testing) กันมากขึ้น ซึ่งจะช่วยเปิดเผยถึงความเป็นไปได้ที่จะเกิดความไม่เท่าเทียมกัน

เครื่องมือเช่น lime, AI Fairness 360 หรือ What-if ช่วยเผยถึงความคลาดเคลื่อนที่อาจเกิดขึ้นได้จากการมีกลุ่มข้อมูลที่น้อยเกินไปในชุดข้อมูลที่ใช้ฝึกสอนโมเดล ส่วนเครื่องมือ อย่าง Google Facets หรือ Facets Dive ช่วยให้เราเห็นภาพ และค้นพบกลุ่มข้อมูลกลุ่มย่อยที่อาจคาดไม่ถึงภายในแหล่งข้อมูลที่ใช้ฝึกสอน อย่างไรก็ตามหัวข้อนี้เป็นสิ่งที่กำลังพัฒนากันอยู่ เราคาดหวังว่าเมื่อเวลาผ่านไป จะมีมาตราฐานและแนวปฏิบัติ เพื่อใช้ทดสอบความลำเอียงทางจริยธรรมออกมามากขึ้น

## Federated learning
*ประเมิน*

การจะฝึกสอนโมเดล โดยทั่วไปแล้ว ต้องรวบรวมข้อมูลจากแหล่งข้อมูลต่างๆ แล้วส่งไปยังที่ส่วนกลางเพื่อทำการฝึกสอน วิธีนี้กลับเป็นปัญหาโดยเฉพาะกับกรณีที่ข้อมูลที่จะส่งนั้นประกอบไปด้วยข้อมูลส่วนบุคคลที่สามารถระบุตัวตนได้

เรารู้สึกยินดีกับการก่อตัวขึ้นของเทคนิค Federated learning ซึ่งเป็นวิธีการเก็บรักษาความเป็นส่วนตัวในการฝึกสอนกับชุดข้อมูลขนาดใหญ่ ที่มีความหลากหลายและเกี่ยวข้องกับข้อมูลส่วนบุคคล

เทคนิคนี้จะอนุญาตให้ข้อมูลถูกเก็บอยู่บนอุปกรณ์ของผู้ใช้ภายใต้การควบคุมของแต่ละบุคคลใดๆ ในขณะเดียวกันก็มีวิธีสนับสนุนข้อมูลให้กับคลังข้อมูลรวมที่ใช้ฝึกสอนโดยไม่เสียความเป็นส่วนตัว

ตัวอย่างหนึ่งของเทคนิคนี้ คือ ให้อุปกรณ์ของผู้ใช้เป็นผู้ถือข้อมูลและอัปเดตโมเดลได้อย่างอิสระ จากนั้นอุปกรณ์จะรวบรวมข้อมูลพารามิเตอร์ของโมเดลแล้วส่งไปยังส่วนกลางแทนการส่งตัวข้อมูลโดยตรง เทคนิคนี้มีเรื่องอัตรารับส่งข้อมูลของเครือข่ายและความสามารถในการคำนวณของอุปกรณ์เป็นข้อจำกัดและความท้าทายสำคัญ แต่กระนั้นก็ตาม เรายังชอบที่ Federated learning อนุญาตให้ผู้ใช้งานมีอำนาจควบคุมข้อมูลส่วนบุคคลของตนเองได้

## JAMstack
*ประเมิน*

แนวความคิดที่เริ่มต้นเมื่อหลายปีก่อน จากการทำ backend as a service เพื่อรองรับแอปพลิเคชันมือถือแบบเนทีฟ ปัจจุบันแนวคิดดังกล่าว เริ่มได้รับความนิยมมากขึ้นสำหรับการใช้งานกับเว็บแอปพลิเคชัน

เราเห็นเฟรมเวิร์คหลายตัวที่นำแนวคิดนี้มาใช้ อย่างเช่น Gatsby.js ที่รองรับเทคนิคการสร้างเว็บไซต์แบบแสดงผลอย่างเดียว (static site generation) ผสมกับเทคนิคการเรนเดอร์ที่ฝั่งไคลแอนต์ (client-side rendering) ที่ไปเรียกใช้ API ภายนอกโดยตรง ซึ่งมีชื่อเรียกการรวมกันเช่นนี้ว่า Jamstack (JavaScript, API, and Markup) วิธีการนี้ ทำให้

# เทคนิค

*เรารู้สึกยินดีกับการก่อตัวขึ้นของเทคนิค Federated learning ซึ่งเป็นวิธีการเก็บรักษาความเป็นส่วนตัวในการฝึกสอนกับชุดข้อมูลขนาดใหญ่ที่หลากหลายและเกี่ยวข้องกับข้อมูลส่วนบุคคล*

*(Federated learning)*

*Jamstack ทำให้เราสร้างประสบการณ์ผู้ใช้ได้อย่างเต็มที่ให้กับเว็บแอปพลิเคชันที่พึ่งพาการเรียก API และ SaaS เป็นส่วนมาก*

*(JAMstack)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# เทคนิค

*ในประสบการณ์ของเรา วิศวกรที่ยอดเยี่ยมไม่ได้ถูกขับเคลื่อนด้วยผลผลิตส่วนบุคคล แต่ถูกขับเคลื่อนจากการทำงานกับทีมที่น่าอัศจรรย์มากกว่า ซึ่งมันให้ประสิทธิผลดีกว่ามาก*

*(วิศวกรสิบเท่า)*

เราสร้างประสบการณ์ผู้ใช้ได้อย่างเต็มที่ ให้กับเว็บแอปพลิเคชันที่พึ่งพาการเรียก API และ SaaS เป็นส่วนมาก

เนื่องจากแอปพลิเคชันจะมี HTML ที่ถูกเรนเดอร์ ณ ตอนบิลด์ไทม์ หรือเฉพาะในเบราว์เซอร์ เลยทำให้การดีพลอยไม่ต่างกับการดีพลอยเว็บไซต์แบบแสดงผลอย่างเดียวทั่วไป และเรายังได้อานิสงส์ข้อดีของมันมาด้วยทั้งหมด นั่นคือ ลดพื้นที่ให้โจมตีที่ฝั่งเซิร์ฟเวอร์ลง และได้ประสิทธิภาพความเร็วมาโดยใช้ทรัพยากรน้อยมาก

เมื่อเป็นเช่นนั้นแล้ว มันจึงเหมาะอย่างยิ่งกับการดีพลอยไปที่ CDN อันที่จริงเราได้ลองตั้งชื่อเล่นให้กับเทคนิคนี้ว่า แอปพลิเคชันที่ให้ความสำคัญกับ CDN มาเป็นอันดับหนึ่ง (CDN first application)

## การเชื่อมโยงประวัติข้อมูลแบบเก็บรักษาความเป็นส่วนตัว (PPRL) โดยอาศัยเทคนิคการกรองแบบ Bloom

*ประเมิน*

การเชื่อมโยงประวัติข้อมูลผู้ใช้จากหลายแหล่งข้อมูลเข้าด้วยกันไม่ใช่เรื่องยาก หากเชื่อมโยงโดยอาศัยกุญแจข้อมูลที่มีค่าตรงกันทุกฝ่าย อย่างไรก็ดี ถ้ามองในมุมความเป็นส่วนตัวของผู้ใช้แล้ว แม้ทุกแหล่งข้อมูลจะมีกุญแจข้อมูลตรงกัน อาจไม่ใช่ความคิดที่ดีนักหากแต่ละแหล่งข้อมูลจะเปิดเผยมันออกนอกระบบของตัวเอง

การเชื่อมโยงประวัติข้อมูลแบบเก็บรักษาความเป็นส่วนตัวโดยอาศัยเทคนิคการกรองแบบ Bloom (Privacy-preserving record linkage: PPRL) เป็นเทคนิคที่ถูกยอมรับกันแพร่หลาย โดยใช้ความน่าจะเป็นมาเชื่อมโยงประวัติข้อมูลระหว่างแหล่งข้อมูลที่ต่างกัน โดยไม่ต้องเปิดเผยข้อมูลส่วนตัวที่ใช้ระบุตัวตนผู้ใช้แต่อย่างใด

ตัวอย่างเช่น เมื่อต้องการจะเชื่อมโยงประวัติจากแหล่งข้อมูลสองที่ ให้ต่างฝ่ายต่างเข้ารหัสเขตข้อมูลส่วนที่สามารถใช้ระบุตัวตนผู้ใช้ได้ ด้วยเทคนิคการกรองแบบ Bloom เสียก่อน จากนั้นให้ทั้งคู่ส่งกุญแจที่ถูกเข้ารหัสไว้ ผ่านช่องทางที่มั่นคง คุณสามารถนำประวัติทั้งสองส่วนมาเชื่อมโยงกันสำเร็จ โดยการเปรียบเทียบคะแนนความคล้ายคลึงกันระหว่างข้อมูลทั้งสองชุด และไม่จำเป็นต้องรู้ถึงข้อมูลส่วนตัวของผู้ใช้เลย

ในบรรดาเทคนิค PPRL ทั้งหลายเราพบว่าการใช้เทคนิคการกรองแบบ Bloom นั้น รองรับการขยายตัวได้ดีเมื่อมีชุดข้อมูลที่มากขึ้น

## ห่วงการเรียนรู้แบบกึ่งสอน

*ประเมิน*

ห่วงการเรียนรู้แบบกึ่งสอน (Semi-supervised learning loops) เป็นประเภทหนึ่งของขั้นตอนการทำงานทางแมชชีนเลิร์นนิงที่ขั้นตอนสามารถทำวนซ้ำได้ มันนำข้อดีของความสัมพันธ์ที่หาได้จากข้อมูลที่ไม่มีป้ายกำกับมาใช้ให้เกิดประโยชน์

ซึ่งเทคนิคประเภทนี้ช่วยปรับปรุงประสิทธิภาพของโมเดลโดยการนำชุดข้อมูลที่มีและไม่มีป้ายกำกับมาผสมรวมกันหลากหลายรูปแบบ หรือไม่ก็ใช้วิธีเปรียบเทียบโมเดลที่ถูกฝึกสอนหลายๆ แบบด้วยข้อมูลซับเซ็ตที่แตกต่างกันไป

เทคนิคนี้ต่างจากเทคนิคการเรียนรู้ที่ไม่มีการสอนจากมนุษย์ (unsupervised learning) ที่เครื่องจะอนุมานการจัดกลุ่มจากข้อมูลที่ไม่มีป้ายกำกับ แล้วก็ต่างจากเทคนิคการเรียนรู้ที่มีการสอน (supervised learning) ที่ข้อมูลที่ใช้ฝึกสอนถูกแปะป้ายไว้ทั้งหมดแล้ว

โดยการเรียนรู้แบบกึ่งสอนนำเอาข้อดีของทั้งคู่มาใช้ โดยใช้ข้อมูลชุดเล็กๆ จากข้อมูลที่มีป้ายกำกับ และข้อมูลชุดที่ใหญ่กว่ามากจากข้อมูลที่ไม่มีป้ายกำกับ เทคนิคนี้ยังมีความเกี่ยวข้องอย่างใกล้ชิดกับเทคนิคการเรียนรู้แบบแข็งขัน (active learning) ที่มนุษย์ถูกกำกับให้เลือกแปะป้ายให้กับข้อมูลที่มีความกำกวมอีกด้วย

จากที่มนุษย์ผู้เชี่ยวชาญ ที่สามารถแปะป้ายให้กับข้อมูลได้อย่างแม่นยำ เป็นทรัพยากรที่มีจำกัด และการแปะป้ายข้อมูลก็เป็นกิจกรรมที่กินเวลานานที่สุดในขั้นตอนการทำงานของแมชชีนเลิร์นนิงทั้งหมด การเรียนรู้แบบกึ่งสอนจึงช่วยลดค่าใช้จ่ายและเวลาในการฝึกสอน และทำให้แมชชีนเลิร์นนิงคุ้มค่าและเปิดความเป็นไปได้กับผู้ใช้อีกประเภทหนึ่ง

## วิศวกรสิบเท่า

*เฝ้าระวัง*

คำศัพท์ที่มีการกล่าวถึงอย่างวิศวกรสิบเท่า (10x engineer) ได้ถูกนำมาไตร่ตรองใหม่อีกครั้งในช่วงหลายเดือนที่ผ่านมา หลังจากมีกระทู้ในทวิตเตอร์ที่ถูกพูดถึงเรื่องนี้ในวงกว้าง ที่สรุปเป็นคำเชิญชวนได้ว่า เพื่อเก็บรักษาวิศวกรระดับเทพบางคนไว้ ที่มีผลผลิตส่วนบุคคล ในสายตาคนอื่นสูงกว่าวิศวกรทั่วไปหลายเท่า บริษัทควรมองข้ามอย่าถือโทษพฤติกรรมการต่อต้านสังคม หรือพฤติกรรมที่สร้างความเสียหายของเขาเหล่านั้นไปบ้าง

ต้องขอบคุณที่หลายคนบนสื่อสังคมออนไลน์มองเรื่องนี้เป็นเรื่องตลก แต่ทัศนคติของสังคมต่อเรื่อง “ฮีโร่นักพัฒนาระดับเทพ” นั้นยังมีกันอย่างแพร่หลาย

ในประสบการณ์ของเรา วิศวกรที่ยอดเยี่ยมไม่ได้ถูกขับเคลื่อนด้วยผลผลิตส่วนบุคคล แต่ถูกขับเคลื่อนจากการทำงานกับทีมที่น่าอัศจรรย์มากกว่า ซึ่งมันให้ประสิทธิผลดีกว่ามาก โดยสร้างทีมจากการนำแต่ละคนที่มีความสามารถมารวมกัน ผสมระดับประสบการณ์ที่แตกต่างและภูมิหลังที่หลากหลายเข้าด้วยกัน และให้การสนับสนุนที่พอเหมาะเพื่อให้เกิด ความร่วมมือ การเรียนรู้ และการพัฒนาอย่างต่อเนื่อง

ทีมสิบเท่า (10x team) แบบนี้ สามารถที่จะเคลื่อนที่ได้ไวกว่า ขยายขนาดได้เร็วกว่า และมีความยืดหยุ่นสูงกว่ามาก โดยไม่จำเป็นต้องให้ท้ายพฤติกรรมที่ไม่เหมาะสม


*© ThoughtWorks, Inc. All Rights Reserved.*

## การประกอบฟรอนท์เอนด์เข้าด้วยกันผ่านแพ็กเกจ

*เฝ้าระวัง*

เมื่อทีมนำแนวคิดของไมโครฟรอนท์เอนด์ไปใช้ พวกเขามีวิธีที่จะประกอบคอมโพเนนต์ต่างๆ เข้ามาเป็นแอปพลิเคชันเดียวกันได้หลายรูปแบบ และแน่นอนว่าย่อมมีรูปแบบที่ไม่ควรอยู่ในนั้นเช่นกัน กรณีที่พบได้ทั่วไป คือการประกอบฟรอนท์เอนด์เข้าด้วยกันผ่านแพ็กเกจ

แพ็กเกจจะถูกสร้างขึ้นจากไมโครฟรอนท์เอนด์แต่ละส่วน ซึ่งส่วนมากอยู่ในรูปแบบแพ็กเกจชนิด NPM แล้วถูกส่งต่อไปเก็บที่ระเบียน จากนั้น แพ็คเกจแต่ละชิ้นจะถูกรวบรวมเข้าด้วยกัน เป็นเพ็คเกจสุดท้ายที่บรรจุทุกไมโครฟรอนท์เอนเอาไว้ทั้งหมด

หากมองจากมุมทางเทคนิคอย่างเดียว การประกอบไมโครฟรอนท์เอนด์เข้าด้วยกัน ณ ตอนบิลด์ไทม์ลักษณะนี้ ให้ผลลัพธ์เป็นแอปพลิเคชันที่ใช้งานได้ปกติดี

อย่างไรก็ตาม การประกอบเข้าด้วยกันผ่านการใช้แพ็กเกจบอกเป็นนัยว่า ทุกครั้งที่เกิดการเปลี่ยนแปลง แพ็กเกจหลักทั้งชิ้นจะต้องถูกสร้างขึ้นมาใหม่ ซึ่งต้องใช้เวลา มีโอกาสอย่างมากที่จะส่งผลเสียกับประสบการณ์การพัฒนา

แย่ไปกว่านั้นรูปแบบของการประกอบฟรอนท์เอนด์เข้าด้วยกันยังทำให้ระหว่างไมโครฟรอนท์เอนด์แต่ละตัวเกิดการพึ่งพาอ้างอิงต่อกันตรงๆ ณ บิลด์ไทม์ เป็นเหตุให้ต้องเสียแรงไม่น้อยไปกับการประสานงานกัน

## แลมด้าเล่นชิ่ง

*เฝ้าระวัง*

พวกเราพัฒนาโครงการที่ใช้สถาปัตยกรรม serverless มาสองปีแล้ว และพบว่ามันค่อนข้างง่ายที่จะตกหลุมพรางของการสร้างระบบให้กลายเป็นโมโนลิธแบบกระจายตัวแทน

สถาปัตยกรรมแบบแลมด้าเล่นชิ่ง (Lambda pinball) โดยบุคลิกของมันแล้วทำให้คนหลุดการเอาใจใส่เรื่องตรรกะของโดเมนที่สำคัญไป เพราะมัวยุ่งกับการจัดการความยุ่งเหยิงของรีเควสต่างๆ ระหว่าง lambda, bucket และ queue ที่ชิ่งกันไปกันมาระหว่างกัน เกิดเป็นกราฟความสัมพันธ์ที่ซับซ้อนขึ้นเรื่อยๆ ระหว่างบริการต่างๆ ในคลาวด์ ซึ่งทำให้การทดสอบระดับยูนิตก็ทำได้ยาก และต้องทำการทดสอบแอปพลิเคชันผ่านการประกอบทุกส่วนเข้าด้วยกันทั้งหมดเท่านั้น

รูปแบบหนึ่งที่ช่วยหลีกเลี่ยงการสร้างสถาปัตยกรรมแบบนี้ได้ คือ ให้แยกความแตกต่างระหว่าง ส่วนต่อประสานที่เป็นสาธารณะกับส่วนต่อประสารนที่ไว้เผยแพร่ ออกจากกัน โดยเริ่มจากการประยุกต์เทคนิคการแบ่งขอบเขตของโดเมนที่เราคุ้นเคยกันดี แล้วค่อยทำให้แต่ละโดเมนใช้ส่วนต่อประสานที่ไว้เผยแพร่ในการสื่อสารระหว่างกัน

## การย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด

*เฝ้าระวัง*

เพื่อรับมือกับความต้องการของลูกค้า (ทั้งลูกค้าภายในและภายนอกองค์กร) เราพบว่ามีองค์กรที่ต้องการแทนที่ระบบเก่าที่ล้าสมัยด้วยระบบใหม่เพิ่มขึ้น หนึ่งในพฤติกรรมไม่ดีที่พบเห็นได้บ่อย คือการย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด (legacy migration feature parity) ซึ่งมาจากความเสียดายฟีเจอร์ในระบบเก่านั่นเอง

เรากลับมองว่านี่เป็นการเสียโอกาสอย่างมาก เพราะเมื่อเวลาผ่านไประบบเก่ามักจะบวมขึ้นเสมอ มีฟีเจอร์มากมายที่ผู้ใช้ไม่ได้ใช้แล้ว (50% ตามรายงานของ Standish Group ปี 2014) และกระบวนการในการทำงานทางธุรกิจได้วิวัฒน์ไปจากเดิมแล้ว การย้ายฟีเจอร์เหล่านี้ไปด้วยเป็นการสูญเปล่าโดยใช่เหตุ

คำแนะนำของเราสำหรับเรื่องนี้คือ ลองโน้มน้าวลูกค้าของคุณให้ถอยหนึ่งก้าวเพื่อทำความเข้าใจความต้องการของลูกค้าในปัจจุบันให้ดี แล้วจัดลำดับความสำคัญของความต้องการเหล่านั้นกับผลลัพธ์ทางธุรกิจและตัวชี้วัดที่เหมาะสม

ซึ่งประโยคนี้มักพูดง่ายแต่ทำยาก เพราะต้องลงทุนทำการวิจัยผู้ใช้และประยุกต์แนวปฏิบัติของการพัฒนาผลิตภัณฑ์สมัยใหม่ลงไปก่อน แทนการย้ายฟีเจอร์เก่ามาใช้ตรงๆ เลย

# เทคนิค

*เมื่อเวลาผ่านไป*
*ระบบเก่าจะบวมขึ้นเสมอ*
*มีฟีเจอร์มากมายไม่ได้ใช้แล้ว*
*ธุรกิจก็ได้วิวัฒน์ไปจากเดิม*
*การย้ายฟีเจอร์เหล่านั้น*
*ไประบบใหม่ทั้งหมด*
*เป็นการสูญเปล่าโดยใช้เหตุ*

*(การย้ายไประบบใหม่*
*โดยคงฟีเจอร์ไว้ทั้งหมด)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# แพลตฟอร์ม

## Apache Flink

*ทดลอง*

Apache Flink มีผู้ใช้มากขึ้นจากที่เราให้ประเมินไว้เมื่อปี 2016 มันได้รับการยอมรับในฐานะผู้นำเอนจิ้นสำหรับประมวลผลข้อมูลในรูปแบบสตรีม แล้วยังเติบโตขึ้นในด้านการประมวลผลแบบแบตช์และด้านแมชชีนเลิร์นนิงเช่นกัน

หนึ่งในข้อแตกต่างของ Flink เมื่อเทียบกับเอนจิ้นสำหรับประมวลผลข้อมูลในรูปแบบสตรีมอื่นๆ คือความสามารถในการบันทึกสถานะของแอปพลิเคชันไว้เป็นระยะอย่างต่อเนื่อง หากแอปพลิเคชันเกิดล้มเหลวขึ้นมา แอปพลิเคชันจะถูกรีสตาร์ทและสถานะของมันจะถูกฟื้นฟูกลับมาจากบันทึกล่าสุดที่เก็บไว้ ดังนั้นแอปพลิเคชันสามารถประมวลผลได้อย่างต่อเนื่อง ราวกับว่าความล้มเหลวไม่เคยเกิดขึ้น วิธีนี้ช่วยนักพัฒนาลดความซับซ้อนของการสร้างและการบริหารระบบภายนอกให้มีความทนทานรองรับความล้มเหลว

เราเห็นหลายบริษัทนำ Flink ไปสร้างแพลตฟอร์มในการประมวลผลข้อมูลมากขึ้นอย่างต่อเนื่อง

## Apollo Auto

*ทดลอง*

แม้ในอดีตเทคโนโลยียานพาหนะไร้คนขับจะถูกจำกัดแค่ในกลุ่มบริษัทเทคโนโลยียักษ์ใหญ่เท่านั้น แต่ปัจจุบัน Apollo Auto ได้แสดงให้เห็นแล้วว่าเทคโนโลยีนี้ไม่ใช่เรื่องยากขนาดนั้นอีกต่อไป

เป้าหมายของโครงการที่ Baidu ผู้เป็นเจ้าของ ได้วางไว้ คือต้องการให้ Apollo กลายเป็นดั่งระบบปฏิบัติการ Andriod สำหรับอุตสาหกรรมยานพาหนะไร้คนขับ

แพลตฟอร์ม Apollo มีส่วนประกอบมากมาย เช่น ส่วนรับรู้ ส่วนจำลอง ส่วนวางแผน และส่วนควบคุมอัจฉริยะที่ทำให้บริษัทรถยนต์ต่างๆ สามารถเชื่อมต่อระบบขับเคลื่อนด้วยตนเองกับฮาร์ดแวร์ของรถยนต์ได้ แม้ชุมชนนักพัฒนายังใหม่อยู่ แต่ผู้จำหน่ายยานพาหนะหลายเจ้าได้เข้าร่วมโครงการเพื่อผลักดันให้เกิดจุดเชื่อมต่อมากขึ้น

หนึ่งในโครงการของเราได้ช่วยให้ลูกค้าผ่านการทดสอบใบอนุญาตยานพาหนะไร้คนขับ ซึ่งส่วนระบบขับเคลื่อนด้วยตนเองถูกพัฒนาบนพื้นฐานของ Apollo นอกจากนี้ Apollo ยังจัดสรรวิธีการเพิ่มฟีเจอร์ขั้นสูงเข้าไปได้เรื่อยๆ ตามแนวคิดสถาปัตยกรรมเชิงวิวัฒน์ ที่เปิดให้เราเพิ่มเซนเซอร์และความสามารถต่างๆ เข้าไปได้ตลอด ตามวิถีแบบเอไจล์

## GCP PubSub

*ทดลอง*

GCP Pub/Sub เป็นแพลตฟอร์มสำหรับทำ event-streaming ของ Google มันเป็นชิ้นส่วนทางโครงสร้างพื้นฐานที่เราหยิบมาใช้บ่อยในสถาปัตยกรรมที่ทำงานบน Google Cloud Platform เพื่อช่วยนำเข้าข้อมูลอีเวนท์ในปริมาณสูงๆ, ใช้สื่อสารระหว่างงาน serverless ต่างๆ, และใช้สตรีมข้อมูลในขั้นตอนงานการประมวลผลข้อมูล เป็นต้น

หนึ่งในฟีเจอร์ที่เป็นเอกลักษณ์คือการรองรับการติดตามข้อมูลได้ทั้งแบบผลักและดึง กล่าวคือ ข้อมูลสามารถถูกผลักให้ผู้สนใจ เช่น การส่งต่อข้อมูลใดๆ ไปที่เอนด์พอยท์หนึ่ง หรือผู้สนใจเลือกดึงข้อมูลเองเมื่อต้องการ เช่น การรอและดึงข้อมูลเป็นชุดเพื่อประมวลผลข้อมูลพร้อมกัน ทีมของเราพอใจในความน่าเชื่อถือและรองรับการขยายตัวได้ดีเหมือนที่มันได้โฆษณาไว้

## Mongoose OS

*ทดลอง*

Mongoose OS ยังคงเป็นหนึ่งในระบบปฏิบัติการไมโครคอนโทรเลอร์ที่เป็นโอเพนซอร์ส และเป็นเฟรมเวิร์คสำหรับพัฒนาเฟิร์มแวร์ของระบบฝังตัวที่เราถูกใจ Mongoose OS เติมเต็มช่องว่างเพื่อนักพัฒนาซอฟต์แวร์ระบบฝังตัว คือ ช่องว่างระหว่างเฟิร์มแวร์ของ Arduino ที่เหมาะแก่การทำผลิตภัณฑ์ต้นแบบ กับ SDK แบบเนทีฟของแบร์เมทัลไมโคร


*© ThoughtWorks, Inc. All Rights Reserved.*

### นำไปใช้

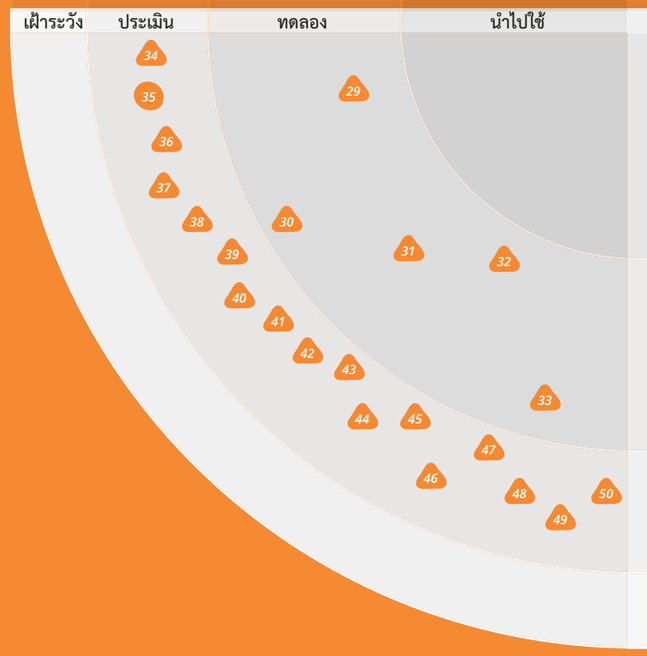
### ทดลอง

29. Apache Flink
30. Apollo Auto
31. GCP PubSub
32. Mongoose OS
33. ROS

### ประเมิน

34. AWS Cloud Development Kit
35. Azure DevOps
36. Azure Pipelines
37. Crowdin
38. Crux
39. Delta Lake
40. Fission
41. FoundationDB
42. GraalVM
43. Hydra
44. Kuma
45. MicroK8s
46. Oculus Quest
47. ONNX
48. Rootless containers
49. Snowflake
50. Teleport

### เฝ้าระวัง



# แพลตฟอร์ม

*Apache Flink ได้รับการยอมรับในฐานะผู้นำเอนจิ้นสำหรับประมวลผลข้อมูลในรูปแบบสตรีม แล้วยังเติบโตขึ้นในด้านการประมวลผลแบบแบตช์และด้านแมชชีนเลิร์นนิงเช่นกัน*

*(Apache Flink)*

*เป้าหมายของโครงการที่วางไว้คือต้องการให้ Apollo กลายเป็นดั่งระบบปฏิบัติการ Andriod สำหรับอุตสาหกรรมยานพาหนะไร้คนขับ*

*(Apollo Auto)*

คอนโทรลเลอร์

จากประสบการณ์ ทีมของเราประสบความสำเร็จในการใช้ mDash ซึ่งเป็นแพลตฟอร์มจัดการอุปกรณ์แบบครบวงจรของ Cesanta เพื่อพัฒนาโครงการขนาดเล็กที่เริ่มทำจากศูนย์ได้สำเร็จ

ในปัจจุบันผู้ให้บริการคลาว์ดแพลต์ฟอร์มสำหรับอุปกรณ์ IoT รายใหญ่ทั้งหลายได้รองรับเฟรมเวิร์คการพัฒนาของ Mongoose OS แล้ว เพื่อนำไปใช้ในเรื่องต่างๆ เช่น การจัดการอุปกรณ์ การเชื่อมต่อสัญญาณ หรือการอัปเกรดเฟิร์มแวร์ทางอากาศ (over--the-air: OTA)

ตั้งแต่เรารายงาน Mongoose OS ไปคราวที่แล้ว จำนวนบอร์ดและไมโครคอนโทรลเลอร์ที่รองรับมันได้เพิ่มมากขึ้น ซึ่งรวมทั้งฮาร์ดแวร์จาก STM, Texas Instruments และ Espressif ก็เช่นกัน เรายังคงถูกใจกับความสามารถในการอัปเดตเฟิร์มแวร์ทางอากาศแบบไม่มีสะดุด และการที่มันสามารถฝังการรักษาความมั่นคงมาในระดับอุปกรณ์แต่ละตัวเลย

## ROS
*ทดลอง*

ROS (Robot Operating System) คือชุดของไลบารีและเครื่องมือที่ช่วยนักพัฒนาซอฟต์แวร์ในการสร้างแอปพลิเคชันหุ่นยนต์ มันคือเฟรมเวิร์คสำหรับการพัฒนาที่ได้จัดเตรียมเครื่องมือต่างๆ ให้ เช่น แอบแสตรกชันเพื่อจำลองฮาร์ดแวร์ ไดร์เวอร์ของอุปกรณ์ ไลบารีช่วยเหลือตัวแสดงผล ระบบการส่งผ่านข้อความ ตัวจัดการแพ็คเกจ และอื่นๆ อีกมากมาย

ตัว Apollo Autoเองก็พัฒนามาจาก ROS เช่นกัน ในอีกโครงการหนึ่งของเราที่ใช้งาน ADAS เพื่อพัฒนาระบบจำลองการขับขี่ เราก็ใช้ระบบส่งข้อความของ ROS (bag) เทคโนโลยีนี้ไม่ใช่เทคโนโลยีใหม่แต่อย่างใด แต่มันได้รับความสนใจอีกครั้งจากพัฒนาการของ ADAS ที่เกิดขึ้น

## AWS Cloud Development Kit
*ประเมิน*

หลายๆ ทีมของเรามักเลือก Terraform เป็นตัวเลือกแรกเมื่อต้องการกำหนดโครงสร้างพื้นฐานของคลาวด์ อย่างไรก็ดีมีบางทีมที่ได้เริ่มทดลองใช้งาน AWS Cloud Development Kit (AWS CDK) แล้ว และค่อนข้างพอใจกับสิ่งที่ได้เจอมา โดยเฉพาะการรองรับการใช้ภาษาโปรแกรมมิ่งเป็นวิธีหลักในการกำหนดค่า แทนที่จะใช้ไฟล์คอนฟิคอย่างปกติทั่วไป

การทำเช่นนี้ได้ ทำให้พวกเขาสามารถประยุกต์ใช้เครื่องมือ วิธีการทดสอบ และทักษะที่มีอยู่แล้วได้ทันที แต่เช่นเดียวกับเครื่องมืออื่นๆ ในตระกูลนี้ การดูแลใส่ใจเพื่อรับรองว่าการดีพลอยยังง่ายต่อความเข้าใจและการบำรุงรักษายังเป็นเรื่องที่ต้องทำ

แม้ว่าการรองรับภาษา C# และภาษา Java ยังต้องรอต่อไป และหากสามารถมองข้ามฟังก์ชันบางอย่างที่ยังไม่สมบูรณ์ในตอนนี้ไปก่อน เราคิดว่า AWS CDK นั้นน่าจับตามองในฐานะเครื่องมือทางเลือกอื่นของการใช้ไฟล์คอนฟิค

## Azure DevOps
*ประเมิน*

บริการของ Azure DevOps ประกอบไปด้วยชุดของบริการที่ช่วยสนับสนุนการพัฒนา ที่แพลตฟอร์มบริหารจัดการให้ ยกตัวอย่างบริการ เช่น ที่เก็บซอร์สโค้ดสำหรับ Git, CI/CD ไปป์ไลน์ เครื่องมือสำหรับทดสอบแบบอัตโนมัติ เครื่องมือจัดการงานคงค้าง และคลังสำหรับเก็บแพ็กเกจ

ไปป์ไลน์ของ Azure DevOps มีพัฒนาการขึ้นตามเวลาที่ผ่านมา สิ่งที่เราชอบเป็นพิเศษคือความสามารถในการกำหนดไปป์ไลน์ด้วยโค้ด และระบบนิเวศของส่วนต่อขยายของมันที่อยู่บน Azure DevOps marketplace กระนั้นในขณะที่เขียนบทความนี้ ทีมงานของเราพบว่ามีฟีเจอร์บางส่วนยังไม่สมบูรณ์นัก ยกตัวอย่างเช่น ขาดส่วนแสดงผลที่มีประสิทธิภาพ ที่ผู้ใช้สามารถเห็นภาพไปป์ไลน์และท่องไปรอบๆ ได้ง่าย รวมถึงยังขาดสามารถในการสั่งไปป์ไลน์ให้ทำงานจากแพ็กเกจ หรือจากไปป์ไลน์อื่นๆ ได้

## Azure Pipelines
*ประเมิน*

Azure Pipelines ที่ให้เราสามารถเป็นหนึ่งในผลิตภัณฑ์ของ Azure DevOps ที่ให้เราสามารถสร้างไปป์ไลน์ด้วยโค้ดบนคลาวด์ สำหรับโครงการที่ใช้ Git เซิร์ฟเวอร์ของ Azure DevOps เอง หรือจากผู้ให้บริการ Git เจ้าอื่นๆ เช่น GitHub หรือ Bitbucket

ความน่าสนใจของมันคือความสามารถในการสั่งสคริปต์ให้ทำงานในเอเย่นต์ต่างระบบปฏิบัติการต่างๆ ได้ ตั้งแต่ Linux, MacOS รวมถึง Windows โดยไม่ต้องจัดการเวอร์ชวลแมชชีนเอง นี่เป็นความก้าวหน้าที่ดีโดยเฉพาะอย่างยิ่งกับทีมที่ต้องใช้โซลูชัน .NET Framework ที่ต้องทำงานบนระบบปฏิบัติการ Windows เรากำลังประเมินการใช้บริการนี้สำหรับทำการส่งมอบซอฟต์แวร์อย่างต่อเนื่องบนระบบ iOS ด้วยเช่นกัน

## Crowdin
*ประเมิน*

โครงการที่ต้องรองรับผู้ใช้หลายภาษาส่วนมากเริ่มต้นด้วยการพัฒนาฟีเจอร์ในภาษาเดียวก่อนและค่อยจัดการแปลภาษาที่เหลือแบบออฟไลน์ผ่านการใช้อีเมลหรือสเปรดชีต แม้ว่ากระบวนการเหล่านี้จะง่ายและใช้ได้ดี แต่ความยุ่งยากอาจเกิดขึ้นได้ในเวลาอันรวดเร็วคุณอาจจะต้องตอบคำถามเดิมซ้ำๆ กับผู้แปลในแต่ละภาษา สูญเสียพลังงานจากการประสานงานระหว่างนักแปล นักพิสูจน์อักษร และตัวทีมพัฒนาเอง

Crowdin เป็นหนึ่งในไม่กี่แพลตฟอร์มที่ช่วยปรับปรุงกระบวนการแปลในโครงการของเราให้เป็นระเบียบ ด้วยเครื่องมือนี้ไม่ขัดจังหวะการทำงาน โดยจะทำให้ทีมพัฒนาสามารถพัฒนาฟีเจอร์ต่อไปเรื่อยๆ โดยแพลตฟอร์มจะจัดระเบียบข้อความที่ต้องถูกแปลไปสร้างเป็นขั้นตอนงานแบบออนไลน์เตรียมไว้

เราชอบที่ Crowdin มีฟีเจอร์ที่คอยกระตุ้นทีมให้ช่วยกัน


*© ThoughtWorks, Inc. All Rights Reserved.*

แปลอย่างสม่ำเสมอและต่อเนื่อง แทนที่จะจัดการการแปลเป็นชุดใหญ่ชุดเดียวตั้งแต่ต้นจนจบ

### Crux
*ประเมิน*

Crux เป็นฐานข้อมูลแบบเอกสาร (document database) ที่มีความสามารถในการการสืบค้นข้อมูลแบบกราฟตามเวลาแบบสองมิติ (bitemporal graph queries)

ฐานข้อมูลโดยส่วนใหญ่จะเก็บข้อมูลเวลาเพียงมิติเดียว ซึ่งหมายความว่าเราสามารถโมเดลข้อเท็จจริงต่างๆ ไปพร้อมกับเวลาที่ข้อเท็จจริงนั้นเกิดขึ้นได้ แต่ฐานข้อมูลตามเวลาแบบสองมิติ นอกจากจะสามารถโมเดลเวลาที่ข้อเท็จจริงเกิดขึ้น (valid time) แล้ว ยังสามารถโมเดลเวลาที่ระบบยอมรับข้อเท็จจริงนั้น (transaction time) อีกด้วย

ถ้าคุณต้องการฐานข้อมูลแบบเอกสารที่มีความสามารถในการสืบค้นแบบกราฟในการค้นหาข้อมูล เราอยากให้ลอง Crux ดู แม้ขณะนี้มันยังอยู่ในช่วงอัลฟาและยังไม่รองรับการสืบค้นด้วย SQL ก็ตาม แต่คุณสามารถใช้ Datalog ในการสืบค้นเพื่ออ่านข้อมูลและท่องไปในความสัมพันธ์ของข้อมูล

### Delta lake
*ประเมิน*

Delta Lake เป็นชั้นเก็บข้อมูลโอเพนซอร์สที่พัฒนาโดย Databricks มันพยายามนำแนวคิดทรานแซคชันมาสู่ระบบประมวลผลข้อมูลขนาดใหญ่ ซึ่งหนึ่งในปัญหาที่พบได้บ่อยเวลาใช้ Apache Spark คือขาดความสามารถการทำทรานแซคชันที่รองรับ ACID

Delta Lake เชื่อมต่อกับ Spark API และจัดการปัญหานี้โดยใช้ประโยชน์จากบันทึกของทรานแซคชัน และไฟล์ Parquet ที่มีการกำหนดเวอร์ชันไว้

นอกจากนี้ด้วยการรองรับการแยกระดับของทรานแซคชันแบบจัดลำดับ (serializable) ช่วยให้การอ่านและเขียนไฟล์ไปที่ Parquet สามารถทำได้พร้อมๆ กัน มันยังมีฟีเจอร์ที่น่าสนใจอื่นๆ เช่น สามารถบังคับใช้โครงสร้างข้อมูลขณะเขียน และสามารถกำหนดเวอร์ชันได้ ซึ่งช่วยให้เราสืบค้นและย้อนคืนข้อมูลเวอร์ชันเก่าๆ ถ้าจำเป็น เราเริ่มใช้ Delta Lake ในบางโครงการของเราและค่อนข้างพอใจกับมัน

### Fission
*ประเมิน*

ระบบนิเวศของ serverless ใน Kubernetes กำลังเติบโตขึ้น เราเคยกล่าวถึง Knative ในเรดาร์ที่แล้ว ในครั้งนี้เราก็เห็นถึงเฟรมเวิร์คตัวหนึ่งที่กำลังได้รับความสนใจ ซึ่งนั่นก็คือ Fission

Fission ทำให้นักพัฒนาสามารถโฟกัสไปที่การเขียน serverless ฟังก์ชันที่มีอายุสั้น และผูกตัวฟังก์ชันเข้ากับ http request ที่ต้องการ ขณะที่ตัวเฟรมเวิร์คเองจะจัดการงานส่วนที่เหลือให้อยู่เบื้องหลัง ทั้งการเชื่อมต่อระบบเข้าด้วยกันและจัดการทรัพยากร Kubernetes ให้อัตโนมัติ

นอกจากนี้ Fission ยังทำให้เราสามารถร้อยเรียงฟังก์ชันหลายตัวเข้าด้วยกัน หรือเชื่อมต่อกับผู้ให้บริการภายนอกผ่านเว็บฮุก (web hooks) หรือแม้กระทั่งทำให้การจัดการโครงสร้างพื้นฐานของ Kubernetes เป็นไปอย่างอัตโนมัติ

### FoundationDB
*ประเมิน*

FoundationDB เป็นฐานข้อมูลโอเพนซอร์สประเภทที่โมเดลมีได้หลากหลายรูปแบบ ซึ่งถูกพัฒนาโดยแอปเปิ้ลโดยแอปเปิ้ลซื้อกิจการมาในปี 2015 และเปิดเผยซอร์สโค้ดในปี 2018

หัวใจหลักของ FoundationDB คือความสามารถในการจัดเก็บค่าในรูปแบบ key-value ในระบบแบบกระจายโดยรองรับทรานแซคชันชนิด “จัดลำดับแบบเข้มงวด” (strict serializability) ได้ด้วย

อีกมุมหนึ่งที่น่าสนใจของ FoundationDB คือแนวคิด “ระดับชั้น” ซึ่งเป็นการสร้างข้อมูลเสริมให้กับโมเดลที่มีอยู่ ซึ่งระดับชั้นเหล่านี้เป็นเพียงคอมโพเนนต์ที่ไม่ถือค่าของตัวเอง แต่ทำงานร่วมกับ key-value หลักอีกทีหนึ่ง ตัวอย่างของระดับชั้น ได้แก่ Record layer และ Document layer

FoundationDB ได้ตั้งมาตรฐานไว้สูง ด้วยการใช้การทดสอบแบบ simulation testing โดยทุกวันจะทดสอบเพื่อจำลองสถานการณ์ให้ระบบล้มเหลวในรูปแบบต่างๆ เพื่อดูว่าระบบสามารถกู้ตัวเองกลับมาเป็นปกติได้หรือไม่ ด้วยประสิทธิภาพการทำงานที่ยอดเยี่ยม การผ่านบททดสอบอย่างหนักหน่วง และความง่ายในการบริหาร ทำให้ FoundationDB ไม่ใช่แค่ฐานข้อมูลธรรมดา แต่เป็นทางเลือกที่ดีสำหรับผู้ที่กำลังพิจารณาสร้างระบบแบบกระจาย โดยสามารถใช้ FoundationDB เป็นแกนพื้นฐานของระบบนั้น

### GraalVM
*ประเมิน*

GraalVM เป็นเวอร์ชวลแมชชีนครอบจักรวาลที่พัฒนาโดย Oracle สำหรับสั่งงานแอปพลิเคชันที่เขียนด้วยภาษาที่มีพื้นฐานบน JVM, JavaScript, Python Ruby, R รวมถึงภาษา C/C++ และภาษาที่มีพื้นฐานบน LLVM ด้วย

GraalVM ในรูปฟอร์มที่เรียบง่ายที่สุด คือเวอร์ชวลแมชชีนที่มีประสิทธิภาพสูงกว่าเวอร์ชวลแมชชีนทั่วไป สำหรับทำงานร่วมกับภาษาในตระกูล JVM หรือกับภาษาอื่นๆ นอก JVM ที่รองรับ มันยังอนุญาตให้เราสร้างแอปพลิเคชันที่ผสมหลายภาษาในแอปพลิเคชันเดียวกันได้ โดยเสียผลกระทบทางประสิทธิภาพเพียงเล็กน้อย

เครื่องมือสารพัดประโยชน์ของมันอย่าง Native Image (ปัจจุบันมีแค่รุ่นไลเซนส์แบบ Early Adopter Technology ให้เลือกใช้) สามารถคอมไพล์โค้ดภาษา Java แบบล่วงหน้าเป็นไฟล์ที่สามารถทำงานได้ด้วยตัวเอง ที่ใช้เวลาในการเริ่มต้นทำงานสั้นกว่า และใช้พื้นที่ในหน่วยความจำน้อยกว่า

GraalVM ได้สร้างความตื่นเต้นอย่างมากในชุมชนนักพัฒนา Java ซึ่งเฟรมเวิร์คต่างๆ รวมทั้ง Micronaut, Quarkus และ Helidon ต่างก็ใช้ประโยชน์จากข้อดีของมัน

## แพลตฟอร์ม

*GraalVM ได้สร้างความตื่นเต้นอย่างมากในชุมชนนักพัฒนา Java ซึ่งเฟรมเวิร์คต่างๆ รวมทั้ง Micronaut, Quarkus และ Helidon ต่างก็ใช้ประโยชน์จากข้อดีของมัน*

*(GraalVM)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# แพลตฟอร์ม

*ความสามารถในการทำงานร่วมกันระหว่างเครื่องมือและเฟรมเวิร์คในระบบนิเวศของนิวรัลเนตเวิร์คยังเป็นความท้าทายที่มีมาเสมอ ซึ่ง ONNX สามารถช่วยได้*

*(ONNX)*

*Kuma เป็น service mesh ที่ไม่ขึ้นกับแพลตฟอร์มใดๆ เราสามารถนำไปใช้บน Kubernetes บนเวอร์ชวลแมชชีน หรือบนเครื่องเซิร์ฟเวอร์เปล่าก็ได้*

*(Kuma)*

## Hydra
*ประเมิน*

การติดตั้งโซลูชัน OAuth2 ของตัวเอง อาจไม่ใช่เรื่องจำเป็นของทุกองค์กร แต่ถ้าจำเป็นขึ้นมา เราพบว่า Hydra เป็นเครื่องมือที่น่าสนใจ

Hydra เป็นโอเพนซอร์สโซลูชัน สำหรับทำ OAuth2 เซิฟเวอร์และให้บริการ OpenID Connect

Hydra ตั้งใจไม่มีระบบจัดการเอกลักษณ์ (identity management) มาให้ในตัว ซึ่งเป็นสิ่งที่เราพอใจมาก เพราะไม่ว่าจะใช้จัดการเอกลักษณ์แบบไหนอยู่ เราสามารถนำมาเชื่อมต่อกับ Hydra ผ่าน API ที่ออกแบบมาอย่างดี

การแยกการจัดการเอกลักษณ์ออกจากเฟรมเวิร์ค OAuth2 เช่นนี้ ช่วยให้ง่ายกว่าในการนำ Hydra ไปใช้ในองค์กรที่มีระบบพิสูจน์ตัวตนอยู่แล้ว

## Kuma
*ประเมิน*

Kuma เป็น service mesh ที่ไม่ขึ้นกับแพลตฟอร์มใดๆ เราสามารถนำไปใช้บน Kubernetes บนเวอร์ชวลแมชชีน หรือบนเครื่องเซิร์ฟเวอร์เปล่าก็ได้

Kuma ถูกสร้างขึ้นเพื่อทำหน้าที่เป็นหอบังคับการบิน (control plane) ที่ทำงานซ้อนอยู่บน Envoy อีกชั้นหนึ่ง มันเปิดโอกาสให้เราดัดแปลงแก้ไขข้อมูลที่วิ่งอยู่ระหว่างเนตเวิร์คในระดับชั้นที่ 4 และชั้นที่ 7 ได้

เมื่อเป็นเช่นนั้นแล้ว เราสามารถเชื่อมโยงข้อมูล สังเกตตรวจตรา รักษาความมั่นคง หรือเติมแต่งข้อมูลที่วิ่งไปมาระหว่างเซอร์วิสได้

Service mesh ตัวอื่นๆ มักถูกพัฒนาขึ้นมาเพื่อทำงานกับ Kubernetes โดยเฉพาะ ซึ่งไม่ใช่เรื่องเลวร้ายอะไร แต่หากองค์กรยังไม่พร้อมเปลี่ยนไปใช้ Kubernetes โอกาสการรับเอา service mesh ไปใช้จะถูกจำกัดลง

แทนที่จะรอให้การเปลี่ยนแพลตฟอร์มขนาดใหญ่เช่นนั้นเสร็จสมบูรณ์ การนำ Kuma มาใช้ในสภาพแวดล้อมปัจจุบันเลย เป็นการยกระดับระบบโครงสร้างพื้นฐานทางเนตเวิร์คให้ทันสมัยในทันที

## MicroK8s
*ประเมิน*

เราเคยแนะนำ Kubernetes ไปแล้ว และมันยังคงเป็นตัวเลือกตั้งต้นเพื่อจัดการหรือดีพลอยคอนเทเนอร์ลงกลุ่มคลัสเตอร์ในโปรดักชัน

อย่างไรก็ตาม การดีพลอย Kubernetes บนเครื่องของนักพัฒนาเองเป็นเรื่องที่ยากขึ้นเรื่อยๆ แล้ว หากต้องการได้ประสบการณ์ที่สมบูรณ์ระดับเดียวกับสภาพแวดล้อมในโปรดักชัน

จากตัวเลือกที่มีอยู่ เราพบว่า MicroK8s เป็นตัวเลือกที่มีประโยชน์ การดีพลอย MicroK8s snap ทำได้ง่ายโดยแค่เลือกเวอร์ชันที่ต้องการ แล้วคุณจะได้ Kubernetes ที่พร้อมใช้งานในไม่กี่คำสั่ง คุณยังสามารถตรวจสอบติดตามเมื่อมีเวอร์ชันสำคัญออกใหม่ และเลือกให้อัปเกรดเองอย่างอัตโนมัติหากต้องการ

## Oculus Quest
*ประเมิน*

เราได้ติดตามเทคโนโลยี AR/VR (Augmented/Virtual Reality) ในเรดาร์ของเรามาพักใหญ่แล้ว เราพบว่าความน่าสนใจของมันถูกจำกัดจากแพลตฟอร์มที่เฉพาะเจาะจงเกินไป และทางเลือกในการเชื่อมต่อสัญญาณที่ยุ่งยาก

แต่ Oculus Quest ได้เปลี่ยนเกมการแข่งขัน โดยเป็นผลิตภัณฑ์ VR แบบสวมศรีษะเจ้าแรกสำหรับลูกค้าทั่วไปที่สามารถทำงานได้ด้วยตัวเอง โดยไม่ต้องพึ่งพาสายเชื่อมต่อหรืออุปกรณ์เสริมนอกเหนือจากโทรศัพท์มือถือ ด้วยศักยภาพของอุปกรณ์นี้ จะเปิดโอกาสให้คนทั่วไปสามารถเข้าถึงแอปพลิเคชัน VR ได้ง่ายขึ้น ซึ่งความต้องการที่สูงขึ้นจะส่งผลให้ตลาดพัฒนานวัตกรรมออกมาแข่งขันกันอย่างเข้มข้น พวกเราขอชื่นชมอุปกรณ์นี้ที่มันช่วยให้การเข้าถึง VR เป็นเรื่องง่าย และตื่นเต้นที่จะได้เห็นสิ่งที่กำลังจะตามมาจากนี้

## ONNX
*ประเมิน*

ระบบนิเวศของเครื่องมือและเฟรมเวิร์คสำหรับนิวรัลเนตเวิร์คมีพัฒนาการที่เร็วมาก แต่ความสามารถในการทำงานร่วมกันระหว่างเครื่องมือต่างๆ ยังเป็นความท้าทายที่มีมาเสมอ

เป็นเรื่องไม่แปลกในวงการแมชชีนเลิร์นนิงที่เราจะสร้างโมเดลต้นแบบขึ้นมาอย่างฉับไว ฝึกสอนโมเดลด้วยเครื่องมือหนึ่ง แล้วนำโมเดลที่ได้ไปดีพลอยในอีกเครื่องมือหนึ่งเพื่อใช้วิจัยและสรุปผลอีกที

แต่ด้วยรูปแบบข้อมูลภายในที่ไม่เข้ากัน ทำให้เราต้องสร้างและบริหารตัวแปลงโมเดลที่ยุ่งเหยิงขึ้นเอง รูปแบบไฟล์ ONNX (Open Neural Network Exchange) จึงเกิดขึ้นเพื่อจัดการกับปัญหานี้

ภายใน ONNX นั้น นิวรัลเนตเวิร์คโมเดลจะถูกนำเสนออยู่ในรูปแบบกราฟข้อมูลที่อ้างอิงตามข้อกำหนดของโอเปอเรเตอร์มาตรฐาน พอรวมกับที่มันกำหนดวิธีการแปลงข้อมูลค่าน้ำหนักที่ผ่านการฝึกสอนไว้ด้วย ทำให้ ONNX สามารถถ่ายทอดโมเดลนิวรัลเนตเวิร์คจากเครื่องมือหนึ่งไปสู่อีกเครื่องมือหนึ่งได้

สิ่งนี้เปิดโอกาสและความเป็นไปได้อีกมากมาย ตัวอย่างเช่น โครงการ Model Zoo ที่ได้รวบรวมชุดโมเดลต่างๆ ที่ผ่านการฝึกสอนแล้วในรูปแบบข้อมูล ONNX ไว้แล้ว


*© ThoughtWorks, Inc. All Rights Reserved.*

## คอนเทเนอร์ที่ไม่มีสิทธิ์รูท

*ประเมิน*

การจัดการและสั่งงานคอนเทเนอร์ในอุดมคตินั้นควรดำเนินการโดยตัวจัดการคอนเทเนอร์ไม่มีสิทธิ์รูท ซึ่งในทางปฏิบัติไม่ใช่สิ่งที่ทำได้ตรงตัวนัก แต่หากทำได้สำเร็จก็จะลดพื้นที่ให้โจมตีลงจากผู้ไม่หวังดีและหลีกเลี่ยงปัญหาด้านความมั่นคงออกไปได้ชุดใหญ่เลย โดยเฉพาะความเสี่ยงที่สิทธิ์ถูกยกออกภายนอกคอนเทเนอร์

ชุมชนนักพัฒนามีการถกเถียงกันเรื่องการทำคอนเทอร์ที่ไม่มีสิทธิ์รูทมาได้สักพัก และเรื่องนี้เป็นหนึ่งในข้อกำหนดที่ Open Container ได้กำหนดไว้ โดยมี runc เป็นรันไทม์มาตรฐานที่ทำตามข้อกำหนดนี้ ซึ่ง Kubernetes เองก็ใช้งานมันอยู่เบื้องล่าง

ปัจจุบัน Docker เวอร์ชัน 19.03 มีฟีเจอร์คอนเทเนอร์ที่ไม่มีสิทธิ์รูทให้ทดลองใช้แล้ว แม้มันจะทำงานได้จริงแต่ฟีเจอร์นี้ยังไม่สามารถทำงานร่วมกับอีกหลายฟีเจอร์ได้ เช่น cgroup ส่วนควบคุมทรัพยากร และตัวจัดการประวัติข้อมูลด้านความมั่นคงของ AppArmor

## Snowflake

*ประเมิน*

เมื่อเรานึกถึงการสร้างคลังข้อมูล (data warehouse) เรามักนึกถึงงานสร้างโครงสร้างพื้นฐานเพื่อรวมศูนย์ข้อมูลที่ยากที่จะจัดการหรือขยายเพราะต้องรับมือกับความต้องการเข้าถึงข้อมูลที่มากขึ้นเรื่อยๆ

แต่ Snowflake ต่างจากนั้น มันคือโซลูชันที่ถูกพัฒนาขึ้นเพื่อให้บริการคลังข้อมูลชนิด SQL บนคลาวด์โดยเฉพาะ ประกอบไปด้วยฟีเจอร์ที่คัดสรรมาอย่างดี เช่น สามารถสร้างทรานแซคชันที่รับประกันความถูกต้องของข้อมูลที่ครอบคลุมทั้งฐานข้อมูล รองรับการสร้างรูปแบบข้อมูลแบบมีโครงสร้างหรือกึ่งมีโครงสร้าง มีฟังก์ชันการวิเคราะห์ข้อมูลในฐานข้อมูลโดยตรง และที่สำคัญที่สุด มันถูกออกแบบมาให้แยกส่วนจัดเก็บ ส่วนประมวลผล และส่วนงานบริการออกจากกันอย่างชัดเจน ผลคือ Snowflake สามารถจัดการปัญหาท้าทายแทบทั้งหมดที่มักพบในงานบริหารคลังข้อมูล

## Teleport

*ประเมิน*

เป็นด่านตรวจด้านความมั่นคงสำหรับตรวจสอบสิทธิ์การเข้าถึงในโครงสร้างพื้นฐานแบบคลาวด์เนทีฟ หนึ่งในฟีเจอร์ที่น่าสนใจคือมันสามารถทำตัวเสมือนเป็นผู้ออกใบรับรอง (CA) สำหรับโครงสร้างพื้นฐาน ทำให้สามารถออกใบรับรองระยะสั้นให้กับระบบที่ต้องการ และกำหนดสิทธิ์การเข้าถึงตามบทบาท (role-based access control) ได้หลากหลายเพื่อใช้ภายในโครงสร้างพื้นฐาน Kubernetes ของคุณ (หรือเพื่อใช้บริหารแค่ SSH อย่างเดียวก็ได้)

ในโครงสร้างพื้นฐานที่ต้องการเพิ่มความเข้มงวดด้านความมั่นคง ความสามารถด้านการติดตามการเปลี่ยนแปลงที่เกิดขึ้นอย่างตลอดเวลาเป็นเรื่องสำคัญ อย่างไรก็ตาม ไม่ใช่ทุกเหตุการณ์จะต้องใช้ความเข้มงวดในการตรวจสอบที่เท่ากันด้วย Teleport คุณสามารถใช้การบันทึกการเปลี่ยนแปลงที่เกิดขึ้นกับเหตุการณ์ส่วนมาก และเพิ่มระดับความเข้มงวดเป็นพิเศษโดยการบันทึกหน้าจอของผู้ใช้งานที่ขอใช้สิทธิ์รูทที่สูงกว่า

# แพลตฟอร์ม

*เป็นด่านตรวจด้านความมั่นคงสำหรับตรวจสอบสิทธิ์การเข้าถึงในโครงสร้างพื้นฐานแบบคลาวด์เนทีฟ*

*(Teleport)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# เครื่องมือ

## Commitizen
*นำไปใช้*

Commitizen เป็นเครื่องมือง่ายๆ ที่ช่วยจัดระเบียบการคอมมิตเมื่อใช้ GIT มันจะคอยถามและให้คุณกรอกเฉพาะข้อมูลที่จำเป็นเท่านั้น แล้วก็จัดรูปแบบข้อความในคอมมิตนั้นๆ ให้อย่างเหมาะสม

มันยังมีแม่แบบข้อความตามสถานการณ์ต่างๆ ให้เลือกใช้มากมาย หรือคุณจะเพิ่มแม่แบบของคุณเองก็ได้ผ่านการสร้างตัวแปลงขึ้นเอง เครื่องมืออันเรียบง่ายนี้ช่วยประหยัดเวลาและป้องกันการปฏิเสธจากคอมมิตฮุกที่อาจเกิดได้ในภายหลัง

## ESLint
*นำไปใช้*

เราใช้ ESLint ในหลายๆ โครงการในฐานะเครื่องมือมาตรฐานเพื่อวิเคราะห์ซอร์สโค้ดสำหรับภาษา JavaScript มันมีชุดกฎเกณฑ์ให้เลือกใช้มากมาย มีชุดมาตรฐานที่แนะนำให้ใช้และส่วนขยายต่อเติมเพื่อรองรับเฟรมเวิร์คหรือ Javascript ในรูปแบบต่างๆ

เราเห็นมันถูกใช้ประโยชน์อย่างหนักเพื่อเป็นการช่วยทีมสร้างและบังคับใช้บรรทัดฐานในการเขียนโค้ด โดยมันคอยวิเคราะห์และรายงานผลให้ตลอดเวลาระหว่างการพัฒนา

เราสามารถสร้างมาตรฐานร่วมกัน โดยบังคับใช้แนวปฏิบัติที่ดีที่สุด หรือสไตล์การเขียนโค้ดให้เป็นแนวทางตามที่ตกลงกันไว้ และยังช่วยชี้ช่องโหวในโค้ดของเราให้อีกด้วย

ESLint เชื่อมต่อเข้ากับ IDE ส่วนใหญ่ และคอยให้คำแนะนำระหว่างเขียนโค้ดแบบทันทีทันใด โดยเฉพาะกฎที่เกี่ยวกับสไตล์การเขียนโค้ด มันสามารถแก้ไขส่วนที่ผิดกฎให้เองโดยอัตโนมัติ ช่วยให้กระบวนการพัฒนาลื่นไหลและมีประสิทธิภาพโดยไม่เสียค่าใช้จ่ายในการพัฒนาอื่นๆ เพิ่มนักพัฒนาสามารถเข้าใจกฎต่างๆ ได้อย่างรวดเร็วจากเอกสารที่ชุมชนช่วยกันทำขึ้นมา ซึ่งเอกสารได้อธิบายรูปแบบต่างๆ ในการเขียนโค้ดเอาไว้อย่างดี

เมื่อ ESLint กลายเป็นสิ่งที่ใช้กันทั่วไปและทรงพลังมากขึ้น จึงทำให้มันเป็นที่สนใจมากขึ้นในอุตสาหกรรม ซึ่งเห็นได้จากการที่ทีมพัฒนาภาษา TypeScript ได้ย้ายมาสนับสนุนและทำงานร่วมกับ ESLint แล้ว แทนที่จะลงทุนต่อใน TSLint

## React Styleguidist
*นำไปใช้*

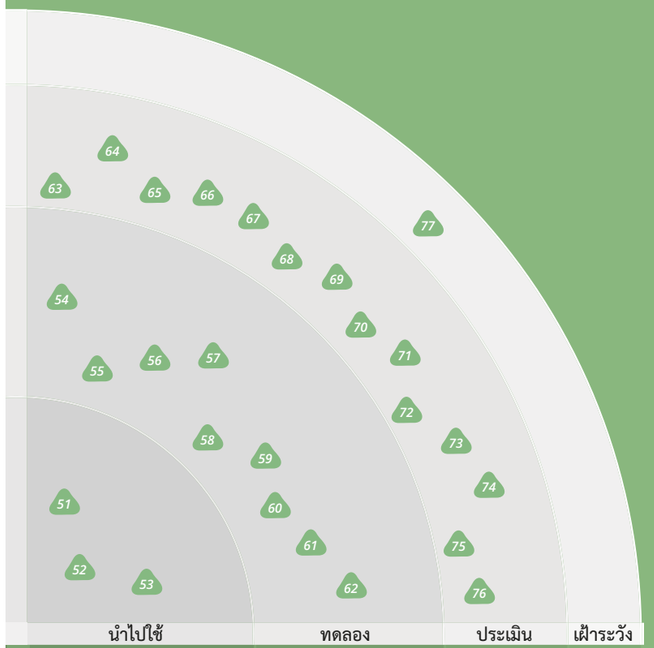
React Styleguidist เป็นเครื่องมือที่ใช้สำหรับการพัฒนา React คอมโพเนนต์ ซึ่งประกอบไปด้วยเซิร์ฟเวอร์สำหรับใช้ระหว่างการพัฒนา ที่มีความสามารถในการรีโหลดหน้าได้อย่างอัตโนมัติเมื่อมีการแก้ไขโค้ดเกิดขึ้น และมันยังสร้างคู่มือการใช้สไตล์ในรูปแบบเอกสาร HTML เพื่อใช้ร่วมกันกับทีม คู่มือนี้จะแสดงคอมโพเนนต์เวอร์ชันล่าสุดไว้ที่เดียวกันทั้งหมด พร้อมเอกสารวิธีการใช้งานคอมโพเนนต์ใดๆ และ props ที่เกี่ยวข้อง

เราเคยกล่าวถึง React Styleguidist ใน**สภาพแวดล้อมในการพัฒนาส่วนต่อประสานผู้ใช้** มาก่อนหน้านี้ และเมื่อเวลาผ่านไป มันได้กลายเป็นทางเลือกแรกในกลุ่มเครื่องมือประเภทเดียวกันนี้

## Bitrise
*ทดลอง*

การประกอบ การทดสอบ และการดีพลอยแอปพลิเคชันมือถือ นำมาซึ่งความซับซ้อนในขั้นตอนต่างๆ อย่างหลีกเลี่ยงไม่ได้ โดยเฉพาะเมื่อเราพิจารณาดูทั้งไปป์ไลน์ตั้งแต่การดึงโค้ดออกมาจากที่เก็บซอร์สโค้ดจนถึงการดีพลอยขึ้นแอปสโตร์

แม้ว่าขั้นตอนทั้งหมดสามารถทำให้เป็นอัตโนมัติได้ด้วยการเขียนสคริปต์ หรือสร้างไปป์ไลน์การประกอบโดยใช้




**นำไปใช้**
51. Commitizen
52. ESLint
53. React Styleguidist

**ทดลอง**
54. Bitrise
55. Dependabot
56. Detekt
57. Figma
58. Jib
59. Loki
60. Trivy
61. Twistlock
62. โครงการ Yocto

**ประเมิน**
63. Aplas
64. asdf-vm
65. AWSume
66. dbt
67. Docker Notary
68. Facets
69. Falco
70. in-toto
71. Kubeflow
72. MemGuard
73. Open Policy Agent (OPA)
74. Pumba
75. Skaffold
76. What-If tool

**เฝ้าระวัง**
77. Azure Data Factory
สำหรับควบคุมไปป์ไลน์


*© ThoughtWorks, Inc. All Rights Reserved.*

# เครื่องมือ

*Figma มีความสามารถที่เหมือนกันกับโปรแกรมออกแบบอย่าง Sketch หรือ Invision แต่มันช่วยให้เราทำงานร่วมกันได้ดีกว่า*

*(Figma)*

*ถ้าคุณกำลังพัฒนาแอปพลิเคชันด้วยภาษา Java และใช้งาน Docker อยู่แล้ว แออยากให้ลองพิจารณาใช้ Jib จาก Google ดูได้*

*(jib)*

เครื่องมือ CI/CD ทั่วไป อย่างไรก็ตาม ทีมเราพบ Bitrise ซึ่งเป็นเครื่องมือ CD ที่ถูกสร้างมาเพื่อแอปพลิเคชันมือถือโดยเฉพาะ มันเหมาะมากกับแอปพลิเคชันมือถือที่มองว่าการเชื่อมต่อกับไปป์ไลน์ของระบบหลังบ้านไม่ใช่เรื่องจำเป็น

Bitrise ง่ายในการดีพลอย และจัดสรรชุดขั้นตอนสำเร็จรูปที่งานพัฒนาแอปพลิเคชันมือถือส่วนใหญ่ต้องการมาให้อย่างครบครัน

## Dependabot

*ทดลอง*

การทำไลบรารีที่แอปพลิเคชันของเราพึ่งพาอ้างอิงให้ทันสมัยอยู่เสมอเป็นงานที่ไม่สนุก แต่ด้วยเหตุผลด้านความมั่นคง มันสำคัญที่ต้องคอยอัปเกรดไลบรารีเหล่านั้นให้ทันสมัยอยู่เสมอ

มีเครื่องมือหลายตัวที่ช่วยให้กระบวนการนี้เจ็บปวดน้อยที่สุดและเป็นอัตโนมัติที่สุด จากประสบการณ์ของเรา เราพบว่า Dependabot ทำได้ดีในเรื่องนี้

โดย Dependabot จะเชื่อมต่อกับที่เก็บซอร์สโค้ดใน Github และคอยตรวจสอบไลบรารีที่แอปพลิเคชันของเราพึ่งพาอ้างอิงให้อัตโนมัติ เมื่อมันพบว่ามีไลบรารีใดที่ควรได้รับการอัปเกรด Dependabot จะขอเปิด pull request พร้อมกับแจ้งเวอร์ชันที่ควรอัปเกรดให้เราตรวจสอบ

## Detekt

*ทดลอง*

Detekt เป็นเครื่องมือวิเคราะห์ซอร์สโค้ดสำหรับภาษา Kotlin มันสามารถรายงานความซับซ้อนอิงตามชุดของกฎที่ปรับแต่งได้อย่างละเอียด และวิเคราะห์หาโค้ดที่เป็นปัญหาด้านการออกแบบ เราสามารถเรียกมันทำงานจากบรรทัดคำสั่งหรือใช้งานผ่านส่วนต่อขยายของ Gradle, SonarQube และ IntelliJ ก็ได้ พวกเราพบว่า Detekt เป็นเครื่องมือที่ดีสำหรับใช้ดูแลซอร์สโค้ดให้มีคุณภาพสูง

ทั้งนี้ เมื่อการวิเคราะห์และการสร้างรายงานถูกรวมเข้าสู่ไปป์ไลน์การประกอบแล้ว การตรวจสอบรายงานเป็นประจำนั้นเป็นเรื่องสำคัญที่ชัดเจนอยู่แล้ว ทีมงานควรจัดสรรเวลาเพื่อจัดการกับปัญหาที่พบอยู่เสมอ

## Figma

*ทดลอง*

หนึ่งในความเจ็บปวดของการออกแบบ การตอบสนองการใช้งาน และ การออกแบบหน้าตาของผลิตภัณฑ์ คือการขาดเครื่องมือที่ช่วยให้เกิดความร่วมมือระหว่างกัน และนี่คือจุดที่ Figma เข้ามาเติมเต็ม

Figma มีความสามารถที่เหมือนกันกับโปรแกรมออกแบบอย่าง Sketch หรือ Invision แต่ด้วยความสามารถในการทำงานร่วมกับผู้อื่นได้ในเวลาเดียวกัน ช่วยให้เราพบไอเดียใหม่ๆ ร่วมกันแบบเรียลไทม์

ทีมของเราพบว่า Figma มีประโยชน์มาก โดยเฉพาะการทำงานแบบรีโมทหรือกับทีมแบบกระจายสถานที่ นอกจากความสามารถในการทำงานร่วมกันแล้ว Figma ยังมี API ที่ช่วยให้เราพัฒนากระบวนการ DesignOps ให้ดีขึ้นอีกด้วย

## Jib

*ทดลอง*

การทำแอปพลิเคชันให้เป็นคอนเทเนอร์อาจต้องการการปรับแต่งค่าที่ซับซ้อนที่สภาพแวดล้อมการพัฒนาหรือในบิลด์เอเยนต์เองก็ตาม

ถ้าคุณกำลังพัฒนาแอปพลิเคชันด้วยภาษา Java และใช้งาน Docker อยู่แล้ว อยากให้ลองพิจารณา Jib จาก Google ดูได้ มันเป็นส่วนต่อขยายสำหรับ Maven และ Gradle ที่เป็นโอเพนซอร์ส

โดยส่วนต่อขยายนี้จะใช้ข้อมูลจากไฟล์บิลด์คอนฟิคโดยตรง เพื่อสร้างแอปพลิเคชันออกมาเป็นอิมเมจของ Docker โดยไม่ต้องมี Dockerfile หรือ Docker daemon เลยก็ตาม

Jib ปรับแต่งประสิทธิภาพการแบ่งชั้นของอิมเมจให้ด้วย ซึ่งโดยทางทฤษฎีจะทำให้การบิลด์ครั้งถัดไปเร็วขึ้น

## Loki

*ทดลอง*

Loki เป็นเครื่องมือสำหรับทดสอบส่วนต่อประสานผู้ใช้มีการเปลี่ยนแปลงจากเดิมหรือไม่ (visual regression) โดยมันทำงานร่วมกับ Storybook ซึ่งเราเคยกล่าวถึงมันในบริบทของสภาพแวดล้อมในการพัฒนาส่วนต่อประสานผู้ใช้ไปแล้ว เพียงปรับแต่งไม่กี่บรรทัด เราสามารถใช้ Loki ทดสอบคอมโพเนนต์ส่วนต่อประสานผู้ใช้ที่มีอยู่ อย่างง่ายได้

เราพบว่า ทางที่ดีควรทำการทดสอบผ่านเบราว์เซอร์ Chrome ภายใต้ Docker คอนเทเนอร์อีกชั้นหนึ่ง เพื่อหลีกเลี่ยงผลการทดสอบที่อ่อนไหวอันเกิดจากการแสดงผลที่ผิดเพี้ยนเล็กน้อยระหว่างสภาพแวดล้อมที่ต่างกัน

จากประสบการณ์ของเรา พบว่าผลทดสอบของ Loki นั้นมีเสถียรภาพ การอัปเกรดเวอร์ชันของ Storybook มีโอกาสทำให้ผลการทดสอบผิดพลาดไปบ้างเล็กน้อย ข้อจำกัดที่ต้องระวังอีกอย่างคือ หากต้องการทดสอบคอมโพเนนต์ที่ใช้ position: fixed นั้นทำไม่ได้ตรงตัว แต่สามารถหลีกเลี่ยงปัญหานี้โดยการห่อคอมโพเนนต์นั้นไว้ก่อน

## Trivy

*ทดลอง*

ในไปป์ไลน์การประกอบที่ใช้สร้างและดีพลอยคอนเทเนอร์ควรมีขั้นตอนการพิเคราะห์คอนเทเนอร์เพื่อความมั่นคงอยู่ด้วย ทีมเราชอบ Trivy เป็นพิเศษ ซึ่งเป็นเครื่องมือไว้ตรวจจับช่องโหว่ของคอนเทเนอร์ การที่มันเป็นไบนารีที่ใช้งานได้ด้วยตัวเองทำให้มันง่ายในการดีพลอยกว่าเครื่องมือตัวอื่น นอกจากนี้ ประโยชน์ของ Trivy คือมันเปิดโอเพนซอร์สและรองรับคอนเทเนอร์ประเภทดิสโตรเลส (distroless)

## Twistlock

*ทดลอง*

Twistlock เป็นผลิตภัณฑ์เชิงพาณิชย์ที่สามารถตรวจจับและป้องกันความเสี่ยงด้านความมั่นคงทั้งตอนบิลด์ไทม์และตอนรันไทม์


*© ThoughtWorks, Inc. All Rights Reserved.*

ความสามารถเหล่านี้ยังปกป้องครอบคลุมไปถึงเวอร์ชวลแมชชีน คอนเทเนอร์ ตัวควบคุมคอนเทเนอร์ ที่เก็บระเบียน และที่เก็บซอร์สโค้ดต่างๆ ที่แอปพลิเคชันพึ่งพาอ้างอิง

Twistlock ช่วยให้ทีมของเราพัฒนาแอปพลิเคชันที่ต้องถูกกำกับดูแลได้เร็วขึ้น ที่โครงสร้างพื้นฐานและสถาปัตยกรรมของแอปพลิเคชันต้องปฏิบัติตามกฎและข้อบังคับต่างๆ เช่น มาตรฐานอุตสาหกรรมบัตรเครดิต (PCI) และ มาตรฐานคุ้มครองข้อมูลที่เกี่ยวข้องกับผู้ป่วย (HIPAA)

ทีมของเราชื่นชอบประสบการณ์ในการพัฒนาที่ Twistlock มอบให้ ยกตัวอย่างเช่น ความสามารถในการจัดสร้างสภาพแวดล้อมด้วยโค้ดได้ ความง่ายในการเชื่อมต่อเข้ากับแพลตฟอร์มสำหรับการเฝ้าสังเกตการณ์ต่างๆ และชุดทดสอบประสิทธิภาพที่มีมาให้ในตัว ที่สามารถใช้วัดประสิทธิภาพของโครงสร้างพื้นฐานของเราเทียบกับมาตรฐานที่อุตสาหกรรมวางไว้

เราใช้งาน Twistlock ร่วมกับเครื่องมือพิเคราะห์แอปพลิเคชันอื่นๆ บนคลาวด์ของเราแบบรันไทม์ โดยเฉพาะกับแอปพลิเคชันที่ จำเป็นจะต้องปฏิบัติตามกฎและข้อบังคับอย่างเคร่งครัด

## โครงการ Yocto

*ทดลอง*

เราเห็นอุปกรณ์ Internet of Things ที่มากด้วยความสามารถแล้วเลือกใช้ลินุกซ์เป็นระบบปฏิบัติการเพิ่มขึ้นแทนการใช้ระบบปฏิบัติการแบบฝังตัวที่ทำขึ้นพิเศษสำหรับอุปกรณ์นั้นๆ

และเพื่อเป็นการประหยัดทรัพยากรและลดพื้นที่การโจมตี มันจึงสมเหตุสมผลพอที่จะใช้ลินุกซ์ที่ถูกออกแบบและปรับแต่งเฉพาะตัวสำหรับอุปกรณ์ใดๆ โดยติดตั้งเครื่องมือที่จำเป็นเพียงพอให้ซอฟต์แวร์ทำงานบนอุปกรณ์นั้นเท่านั้น

ในบริบทนี้จึงทำให้ โครงการ Yocto กลับมามีประโยชน์อีกครั้งในฐานะเครื่องมือช่วยออกแบบและปรับแต่งลินุกซ์เพื่อการแจกจ่าย

ด้วยความที่เครื่องมือนี้ใช้พื้นฐานความรู้ค่อนข้างมากและมีความยืดหยุ่นสูง จึงมีโอกาสผิดพลาดได้ง่าย อย่างไรก็ตามช่วงหลายปีที่ผ่านไป โครงการ Yocto ได้ดึงดูดให้เกิดชุมชนนักพัฒนาที่กระตือรือร้นที่พร้อมให้ความช่วยเหลือ

เมื่อเปรียบเทียบกับเครื่องมือในลักษณะเดียวกัน Yocto สามารถนำไปใช้ในกระบวนการส่งมอบซอฟต์แวร์อย่างต่อเนื่องได้ง่ายกว่า และไม่ยึดติดกับระบบนิเวศอันใดอันหนึ่ง เหมือน Android Things หรือ Ubuntu core

## Aplas

*ประเมิน*

ส่วนมากแล้ว การบริหารสินทรัพย์ทางซอฟต์แวร์ของเราเป็นเรื่องยากมากเมื่อมันมีมากและซับซ้อนขึ้นเรื่อยๆ Aplas เป็นเครื่องมือทำแผนภาพซอฟต์แวร์เพื่อให้เราเห็นภาพรวมทางซอฟต์แวร์ที่ถูกใช้อยู่ภายในองค์กรในรูปแบบแผนที่

เครื่องมือทำงานโดยการนำเข้าอภิพันธุ์ข้อมูล (meta-data) ของระบบปัจจุบัน แล้วแสดงผลเป็นแผนที่ออกมา โดยเราสามารถแสดงแผนที่จากมุมมองที่หลากหลาย ซึ่งการนำเข้าข้อมูลสินทรัพย์สามารถทำได้ทั้งแบบป้อนข้อมูลด้วยมือหรือแบบผ่านการเรียก API อย่างอัตโนมัติ เราค่อนข้างตื่นเต้นที่ได้เห็นวิวัฒนาการของผลิตภัณฑ์นี้ และความเป็นไปได้ที่จะเกิดขึ้นจากการรวมรวบอภิพันธุ์ข้อมูลได้อย่างอัตโนมัติ

ตัวอย่างเช่น มันควรเป็นไปได้ที่จะเปิดเผยฟังก์ชันความเหมาะสมทางสถาปัตยกรรม (architectural fitness function) ของเราออกมา เช่น ค่าใช้จ่ายในการดำเนินการ แล้วใช้ Aplas ทำให้เห็นภาพว่าเราจ่ายกับโครงสร้างพื้นฐานของคลาวด์ไปเท่าไหร่แล้ว

การทำความเข้าใจว่าระบบไหนสื่อสารกับระบบไหนบ้างด้วยเทคโนโลยีอะไร เป็นปัญหาหนึ่งที่เราเจออยู่ประจำ ซึ่ง Aplas สามารถแสดงมันให้เราเห็นได้

## asdf-vm

*ประเมิน*

asdf-vm เป็นเครื่องมือประเภทรับบรรทัดคำสั่ง ที่ช่วยบริหารเวอร์ชันของระบบรันไทม์ในแต่ละโครงการ รองรับภาษาโปรแกรมได้หลากหลาย มันทำงานคล้ายกับเครื่องมือ เช่น rvm ของ Ruby และ nvm ของ Node

ด้วยข้อดีของการใช้สถาปัตยกรรมที่ออกแบบให้รองรับการต่อเติมขยายเลยทำให้มีภาษาและเครื่องมืออยู่ในรายการส่วนขยาย มากมาย เช่น Bazel หรือ tflint ซึ่งเป็นเครื่องมือที่ผู้ใช้ต้องบริหารรันไทม์เวอร์ชันเป็นโครงการต่อโครงการไปเหมือนกัน

## AWSume

*ประเมิน*

AWSume เป็นสคริปต์สำหรับอำนวยความสะดวก ที่ช่วยให้เราจัดการโทเค่นสำหรับเข้าใช้งาน AWS และไว้ใช้สวมรหัสยืนยันตัวตนตามบทบาท (assume role credentials) ต่างๆ ผ่านบรรทัดคำสั่งได้

เราพบว่า AWSume ใช้งานค่อนข้างสะดวกเมื่อต้องใช้งานร่วมกับหลายๆ บัญชีพร้อมกัน แทนที่จะต้องระบุโปรไฟล์ที่ต้องการใช้งานในทุกๆ คำสั่ง ตัวสคริปต์อ่านค่าจากแคชของบรรทัดคำสั่งไว้ และส่งออกค่าเหล่านั้นไปยังตัวแปรระดับสภาพแวดล้อม (environment variables) ซึ่งส่งผลให้ทั้งการออกคำสั่งผ่านบรรทัดคำสั่งและ AWS SDK สามารถเลือกหยิบรหัสยืนยันตัวตนที่ถูกต้องมาใช้ได้อย่างสะดวก

## dbt

*ประเมิน*

การแปลงรูปข้อมูล (data transformation) เป็นส่วนสำคัญของขั้นตอนงานการประมวลผลข้อมูล ตั้งแต่ การกรอง การจัดกลุ่ม หรือการรวมข้อมูลจากหลายๆ แหล่งข้อมูลเพื่อจัดรูปแบบข้อมูลให้เหมาะสมสำหรับนำไปวิเคราะห์ หรือเพื่อนำเข้าข้อมูลให้แมชชีนเลิร์นนิงโมเดล

dbt เป็นโอเพนซอร์สและผลิตภัณฑ์เชิงพาณิชย์ประเภท SaaS สำหรับนักวิเคราะห์ข้อมูล เพื่อช่วยให้การแปลงรูปข้อมูลเป็นเรื่องง่ายและมีประสิทธิภาพ

เฟรมเวิร์คและเครื่องมือสำหรับการแปลงรูปข้อมูลในปัจจุบันสามารถแบ่งออกเป็นสองกลุ่ม คือกลุ่มที่ทรงพลังและยืดหยุ่น แต่ต้องการความเข้าใจอย่างลึกซึ้งในรูปแบบการเขียนโปรแกรมและภาษาของแต่ละเฟรมเวิร์ค อย่างเช่น

# เครื่องมือ

*โครงการ Yocto กลับมามีประโยชน์อีกครั้ง ในฐานะเครื่องมือช่วยออกแบบ และปรับแต่งลินุกซ์เพื่อการ แจกจ่ายที่สร้างขึ้นพิเศษ สำหรับแต่ละเงื่อนไข เช่น อุปกรณ์ Internet of Things*

*(โครงการ Yocto)*

*เป็นเครื่องมือทำแผนภาพ ซอฟต์แวร์เพื่อให้เราเห็นภาพรวมทางซอฟต์แวร์ที่ถูกใช้อยู่ภายใน องค์กรในรูปแบบแผนที่*

*(Aplas)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# เครื่องมือ

*จากการที่ Kubernetes ถูกนำไปใช้เป็นระบบควบคุมคอนเทเนอร์มากขึ้น ชุดเครื่องมือด้านความมั่นคงสำหรับคอนเทเนอร์และ Kubernetes เลยมีวัฒนาการอย่างรวดเร็ว Falco เป็นหนึ่งในเครื่องมือที่ทำมาเพื่อคอนเทเนอร์โดยเฉพาะ เพื่อจัดการด้านความมั่นคงขณะรันไทม์*

*(Falco)*

Apache Spark หรือกลุ่มเครื่องมือประเภทหน้าจอลากและวาง ที่ก็ไม่เข้ากันกับแนวปฏิบัติด้านวิศวกรรมที่ช่วยเสริมความน่าเชื่อถือของระบบ อย่างเช่น การทดสอบอย่างอัตโนมัติ และการดีพลอย

dbt ช่วยเติมเต็มช่องว่างนี้ มันใช้ภาษา SQL ในการสื่อสารซึ่งเป็นภาษาที่เข้าใจกันอย่างกว้างขวาง สามารถโมเดลการแปลงรูปข้อมูลที่อยู่ในรูปแบบแบตช์ ในขณะเดียวกันก็รองรับการใช้งานผ่านบรรทัดคำสั่ง ที่ส่งเสริมกันกับแนวปฏิบัติด้านวิศวกรรมที่ดี เช่นการกำหนดเวอร์ชัน การทดสอบอย่างอัตโนมัติ และการดีพลอย โดยแก่นหลักของมันแล้ว มันต้องการเป็นเครื่องมือสำหรับโมเดลการแปลงรูปข้อมูลด้วยโค้ด ในรูปแบบ SQL นั่นเอง

dbt ในขณะนี้รองรับข้อมูลจากหลายๆ **แหล่งข้อมูล** อย่างเช่น Snowflake และ Postgres และมี**ตัวเลือกในการสั่งงาน**ให้หลากหลาย อย่างเช่น Airflow และระบบคลาวด์ของมันเอง ความสามารถในการแปลงรูปข้อมูลถูกจำกัดตามข้อจำกัดของภาษา SQL และมันยังไม่รองรับการแปลงข้อมูลแบบเรียล์ไทม์ ณ ขณะนี้ที่บทความนี้ถูกเขียน

## Docker Notary

*ประเมิน*

Docker Notary เป็นซอฟต์แวร์โอเพนซอร์สสำหรับใช้ประทับสินทรัพย์ทางดิจิตอลต่างๆ เช่น ไฟล์ อิมเมจ และคอนเทเนอร์ นั่นหมายความว่าเราสามารถพิสูจน์แหล่งที่มาของสินทรัพย์ได้ ซึ่งเป็นแนวปฏิบัติที่ดีกว่าสำหรับทุกที่ที่นำไปใช้ และมีประโยชน์อย่างมากในสภาพแวดล้อมที่ต้องถูกกำกับดูแล

ตัวอย่างเช่น หลังจากคอนเทเนอร์ถูกสร้าง เราสามารถใช้กุญแจส่วนตัวของผู้สร้างและค่าแฮชของอิมเมจนั้นร่วมกันเพื่อประทับตัวตนของผู้สร้างกับอิมเมจเข้าด้วยกัน จากนั้นเราสามารถพิสูจน์แหล่งที่มาโดยตรวจสอบค่าแฮชร่วมกับกุญแจสาธารณะของผู้สร้างอีกที

เพื่อบริหารข้อมูลกุญแจสาธารณะต่างๆ ที่มีในระบบเราสามารถใช้ระเบียนสาธารณะที่รองรับ Notary ที่น่าเชื่อถือ เช่น Docker Trusted Registry หรือเราจะติดตั้งและบริหารระเบียนส่วนตัวเองก็ได้เช่นกัน

บางทีมของเราที่บริหารเซิฟเวอร์ Notary ด้วยตัวเอง ได้รายงานถึงความไม่เสถียรในบางกรณี ฉะนั้นเราจึงแนะนำให้ใช้ระเบียนที่มี Notary ติดตั้งมาพร้อมใช้ในตัวถ้าเป็นไปได้

## Facets

*ประเมิน*

อันเนื่องจากข้อมูลขนาดใหญ่ถูกใช้ในการตัดสินใจสำคัญนั้นเพิ่มขึ้นอย่างต่อเนื่อง ไม่ว่าจะใช้ตัดสินใจตรงๆ หรือใช้เพื่อฝึกสอนโมเดลแมชชีนเลิร์นนิงก็ตาม มันสำคัญที่ต้องเข้าใจช่องว่าง ข้อบกพร่อง และความเอนเอียงต่างๆ ที่มีโอกาสเกิดในชุดข้อมูลของคุณ

โครงการ Facets ของ Google มอบเครื่องมือมีที่มีประโยชน์ให้แก่เราสองตัวด้วยกัน ได้แก่ Facets Overview และ Facets Drive

Facets Overview แสดงภาพการกระจายของค่าสำหรับแต่ละคุณลักษณะในชุดข้อมูล สามารถแสดงการเบี่ยงเบนของชุดข้อมูลทั้งที่ใช้ฝึกสอนและใช้ตรวจสอบ และสามารถใช้เปรียบเทียบชุดข้อมูลหลายๆ ชุดได้

Facets Drive ใช้สำหรับเจาะลึกและแสดงภาพข้อมูลแต่ละจุดในชุดข้อมูลขนาดใหญ่ ด้วยการใช้มิติที่มองเห็นได้ต่างๆ ในการสำรวจความสัมพันธ์ระหว่างคุณลักษณะ

เครื่องมือทั้งสองมีประโยชน์มากสำหรับการทำ**การทดสอบความลำเอียงทางจริยธรรม**

## Falco

*ประเมิน*

จากการที่ Kubernetes ถูกนำไปใช้เป็นระบบควบคุมคอนเทเนอร์มากขึ้น ชุดเครื่องมือด้านความมั่นคงสำหรับคอนเทเนอร์และ Kubernetes เลยมีวัฒนาการอย่างรวดเร็ว Falco เป็นหนึ่งในเครื่องมือที่ทำมาเพื่อคอนเทเนอร์โดยเฉพาะเพื่อจัดการด้านความมั่นคงขณะรันไทม์
Falco ใช้ประโยชน์จาก**เครื่องมือตรวจจับ Linux kernel** ของ Sysdig และการตรวจประวัติจาก system call ทำให้เราสามารถเข้าถึงข้อมูลเชิงลึกของระบบ และช่วยให้เราตรวจพบกิจกรรมไม่ปกติใน แอปพลิเคชัน คอนเทเนอร์ เครื่องเซิร์ฟเวอร์ หรือแม้กระทั่งตัวควบคุมของ Kubernetes เองก็ได้ เราชอบความสามารถของ Falco ที่มันสามารถตรวจจับการบุกรุกได้โดยไม่ต้องพึ่งพาโค้ดภายนอก หรือการพ่วงคอนเทเนอร์ไว้ด้านข้างแต่อย่างใด

## in-toto

*ประเมิน*

เราเห็นเทคนิค**การพิสูจน์ใบนารี** ถูกใช้เพิ่มมากขึ้นในการรักษาความมั่นคงให้กับห่วงโซ่อุปทานซอฟต์แวร์ โดยเฉพาะอย่างยิ่งในอุตสาหกรรมที่มีการกำกับดูแล ซึ่งแนวทางที่ได้รับความนิยมในปัจจุบันมีทั้งการสร้างระบบสำหรับการพิสูจน์ใบนารีขึ้นมาใช้เองหรือพึ่งพาบริการบนคลาวด์ พวกเรารู้สึกยินดีที่ได้เห็นโอเพนซอร์สที่ชื่อว่า in-toto เข้ามามีบทบาทในพื้นที่นี้

in-toto เป็นเฟรมเวิร์คที่ใช้เทคนิคการเข้ารหัสในการพิสูจน์ มันจะพิสูจน์ทุกส่วนประกอบและทุกขั้นตอนตลอดทางจนถึงโปรดักชันของแพ็กเกจใดๆ มันยังมีจุดเชื่อมต่อให้หลากหลายครอบคลุมเครื่องมือประเภทต่างๆ เช่น เครื่องมือช่วยประกอบ เครื่องมือตรวจสอบคอนเทเนอร์ และเครื่องมือช่วยดีพลอย

เป็นไปได้ที่เครื่องมือสำหรับห่วงโซ่อุปทานซอฟต์แวร์จะเป็นเครื่องมือส่วนที่วิกฤติที่สุดในชุดเครื่องมือรักษาความมั่นคงขององค์กร เราจึงพอใจที่มันเป็นโครงการโอเพนซอร์ส และที่ in-toto มีพฤติกรรมโปร่งใสตรวจสอบได้ แม้กระทั่งความสมบูรณ์ถูกต้องและห่วงโซ่อุปทานของตัวเองก็ยังสามารถถูกพิสูจน์ได้โดยชุมชน เราต้องมารอดูกันว่ามันจะดึงดูดผู้ใช้และผู้มีส่วนร่วมพัฒนาได้มากพอหรือไม่ เพื่อแข่งขันในตลาดนี้

## Kubeflow

*ประเมิน*

Kubeflow น่าสนใจด้วยเหตุผลสองประการ ประการแรกมันนำเอา Kubernetes Operators มาใช้อย่างชาญฉลาด ซึ่งเราได้พูดให้ความสำคัญในเทคโนโลยีเรดาร์ ฉบับเดือนเมษายน 2019

ประการที่สองคือ Kubeflow เปิดให้เราสามารถฝังรหัส


*© ThoughtWorks, Inc. All Rights Reserved.*

และการกำหนดเวอร์ชันของขั้นตอนงานของแมชชีนเลิร์นนิงได้ ทำให้เราย้ายขั้นตอนงานจากสภาพแวดล้อมหนึ่งไปสู่อีกสภาพแวดล้อมหนึ่งได้ง่าย

Kubeflow ประกอบไปด้วยคอมโพเนนต์หลากหลาย เช่น Jupyter notebook, ไปป์ไลน์ข้อมูล และเครื่องมือควบคุมต่างๆ

เมื่อนำคอมโพเนนต์เหล่านี้มาจัดเป็นแพ็กเกจในรูปแบบ Kubernetes operator แล้ว เราสามารถดึงความสามารถจาก Kubernetes มาใช้โต้ตอบกับเหตุการณ์ต่างๆ ที่ pods สร้างขึ้น เพื่อสร้างเป็นลำดับขั้นต่างๆ ของขั้นตอนงาน เมื่อแต่ละแพ็กเกจแยกคอนเทเนอร์ของโปรแกรมและข้อมูลออกจากกันแล้ว ขั้นตอนงานทั้งหมดสามารถถูกย้ายจากสภาพแวดล้อมหนึ่งไปยังอีกที่หนึ่งได้

สิ่งนี้เป็นประโยชน์มากในสถานการณ์ที่ขั้นตอนงานถูกพัฒนาขึ้นบนคลาวด์หนึ่ง แต่ความต้องการการประมวลผลมีสูงขึ้นจึงต้องย้ายไปสู่อีกสภาพแวดล้อมหนึ่ง เช่น ซูเปอร์คอมพิวเตอร์ที่ทำขึ้นเอง หรือคลัสเตอร์ที่ใช้หน่วยประมวลผลเทนเซอร์ (TPU) แทน

## MemGuard

*ประเมิน*

แอปพลิเคชันที่ต้องรับมือกับข้อมูลที่ละเอียดอ่อน (เช่น ข้อมูลกุญแจที่ใช้ในการเข้ารหัส) เมื่อข้อมูลเหล่านั้นถูกจัดเก็บลงหน่วยความจำแบบไม่เข้ารหัสแล้ว มีความเสี่ยงที่ผู้ไม่ประสงค์ดีจะใช้จุดอ่อนนี้ อ่านค่าและเข้าถึงเครื่องอื่นๆ ในเนตเวิร์ค หรือนำข้อมูลไปใช้ในทางไม่ดี

คลาว์ดโซลูชันส่วนมาก มักเลือกใช้เทคนิค hardware security modules (HSM) เพื่อหลีกเลี่ยงการโจมตีลักษณะนี้ อย่างไรก็ตาม หากคุณต้องทำสิ่งเดียวกันบนศูนย์ข้อมูลของตนเองที่ไม่มี HSM ให้ใช้แล้วละก็ MemGuard เป็นอีกทางเลือกหนึ่งที่มีประโยชน์

MemGuard เป็นแอปพลิเคชันที่ทำงานที่ระดับหน่วยความจำ มันทำหน้าที่เสมือนกับกล่องนิรภัยที่หุ้มข้อมูลสำคัญไว้ แม้ MemGuard จะแทนที่การใช้ HSM ไม่ได้ซะทีเดียว แต่มันก็ติดตั้งความสามารถด้านความมั่นคงหลายอย่างมาให้ เช่น สามารถป้องกันการโจมตีในลักษณะ cold boot คอยช่วยระวังไม่ให้ข้อมูลสำคัญถูกรบกวนจากการทำ garbage collection และเป็นเสมือนป้อมปราการที่ทำงานร่วมกับ guard page เพื่อลดโอกาสที่ข้อมูลที่ละเอียดอ่อนจะถูกเปิดเผยสู่ภายนอก

## Open Policy Agent (OPA)

*ประเมิน*

การกำหนดและบังคับใช้นโยบายด้านความมั่นคงให้เหมือนกันท่ามกลางความหลากหลายของเทคโนโลยีเป็นเรื่องยาก

แม้แต่แอปพลิเคชันง่ายๆ เรายังต้องควบคุมสิทธิ์การเข้าถึงของส่วนประกอบต่างๆ ของมัน โดยต้องทำการปรับแต่งนโยบายความมั่นคงและบังคับใช้ผ่านกลไกที่ส่วนประกอบแต่ละส่วนมีให้เอง ทั้งตัวควบคุมคอนเทเนอร์ ตัวเซอร์วิส และแหล่งเก็บข้อมูลของเซอร์วิสใดๆ เป็นต้น

พวกเราตื่นเต้นกับ Open Policy Agent (OPA) ซึ่งเป็นเทคโนโลยีโอเพนซอร์สที่พยายามจะแก้ปัญหานี้ OPA ให้คุณกำหนดนโยบายสิทธิ์การเข้าถึงได้อย่างละเอียดและยืดหยุ่นผ่านโค้ด โดยใช้ภาษาสำหรับกำหนดนโยบายที่ชื่อว่า Rego มันมีพฤติกรรมการบังคับใช้นโยบายแบบกระจายตัว และทำการควบคุมอยู่ภายนอกไม่รุกล้ำเข้าไปยุ่งกับโค้ดของแอปพลิเคชัน

ขณะที่กำลังเขียนบทความนี้อยู่ OPA ได้ทำให้การกำหนดนโยบายและการบังคับใช้ต่างๆ มีความยืดหยุ่นและเป็นรูปแบบเดียวกัน เพื่อรักษาความมั่นคงการเข้าถึง API ของ Kubernates, API ของไมโครเซอร์วิสผ่านการพ่วงไซด์คาร์กับ Envoy และ Kafka มันยังสามารถใช้เป็นไซด์คาร์พ่วงเข้ากับเซอร์วิสใดๆ ที่ต้องการตรวจสอบสิทธิ์การเข้าถึงหรือกรองข้อมูล http response ที่ออกไปได้ด้วย

Styra บริษัทที่อยู่เบื้องหลัง OPA ยังมีผลิตภัฑณ์เชิงพาณิชย์ที่ไว้จัดการนโยบายที่กระจายตัวอยู่ให้มองเห็นเสมือนรวมศูนย์อยู่ที่เดียว พวกเราอยากที่จะเห็น OPA เติบโตอย่างมั่นคงขึ้นผ่านโปรแกรมบ่มเพาะของ CNCF (CNCF incubation program) และพัฒนาต่อไปเพื่อรองรับการบังคับใช้นโยบายในสถานการณ์ที่ท้าทายมากยิ่งขึ้น เช่น สถานการณ์ที่มีแหล่งข้อมูลที่หลากหลาย

## Pumba

*ประเมิน*

Pumba เป็นเครื่องมือทดสอบความโกลาหล (chaos testing) และเครื่องมือจำลองเนตเวิร์ค (network emulation) สำหรับ Docker

Pumba สามารถสั่งคอนเทเนอร์ให้หยุดชั่วคราว หยุดถาวร ถอนตัว หรือตายไปได้ แล้วยังจำลองเนตเวิร์คโดยสร้างเหตุการณ์ล้มเหลวในรูปแบบต่างๆ ได้ เช่น ข้อมูลมาล่าช้า ข้อมูลสูญหาย และข้อมูลถูกจำกัดอัตรารับส่ง

Pumba ใช้เครื่องมือ tc ในการจำลองเนตเวิร์ค ดังนั้นในคอนเทเนอร์ของเราต้องมี tc ติดตั้งไว้แล้ว หรือเราอาจจะเปิดการทำงานของ Pumba แยกเป็นคอนเทเนอร์ต่างหากแล้วพ่วงมันกับคอนเทเนอร์หลักอีกที

Pumba มีประโยชน์อย่างมากเมื่อต้องการทดสอบความโกลาหลอย่างอัตโนมัติกับระบบแบบกระจาย โดยเราจะจำลองและทดสอบบนกลุ่มคอนเทเนอร์ในเครื่องส่วนตัวของนักพัฒนาหรือในไปป์ไลน์การประกอบก็ได้

## Skaffold

*ประเมิน*

Google นำเสนอ Skaffold ออกมา ซึ่งเป็นเครื่องมือโอเพนซอร์สที่ไว้ทำให้ขั้นตอนการพัฒนาบนเครื่องนักพัฒนาเป็นไปอย่างอัตโนมัติ ซึ่งรวมไปถึงการดีพลอยงานไปที่ Kubernetes

มันช่วยตรวจจับการเปลี่ยนแปลงของซอร์สโค้ดและเรียกขั้นตอนต่างๆ ตั้งแต่การประกอบ ติดป้าย และดีพลอยเข้าไปในคลัสเตอร์ของ K8s แล้วยังนำบันทึกการเปลี่ยนแปลงของแอปพลิเคชันออกสู่บรรทัดคำสั่งให้อีกด้วย ขั้นตอนเหล่านี้สามารถเชื่อมเข้ากับเครื่องมือในการประกอบและดีพลอยอื่นๆ ได้ แต่ค่าเริ่มต้นที่เตรียมไว้นั้นถูกกำหนดตามความเห็นส่วนตัวของผู้สร้างเพื่อให้ง่ายต่อการเริ่มต้นใช้งาน

# เครื่องมือ

*Pumba เป็นเครื่องมือทดสอบความโกลาหลและช่วยจำลองสถานการณ์เนตเวิร์ค มีประโยชน์อย่างมากเมื่อต้องการทดสอบระบบแบบกระจาย โดยสามารถทดสอบในกลุ่มคอนเทเนอร์บนเครื่องส่วนตัวของนักพัฒนา หรือบนไปป์ไลน์การประกอบก็ได้*

*(Pumba)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# เครื่องมือ

*What-if Tool ช่วยให้นักวิทยาศาสตร์ข้อมูลขุดลึกลงไปในพฤติกรรมของโมเดล และทำให้เห็นว่าฟีเจอร์หรือชุดข้อมูลใด มีผลกระทบอย่างไรกับผลลัพธ์ที่ออกมา*

*(What-If Tool)*

## What-If tool

*ประเมิน*

โลกของแมชชีนเลิร์นนิงเริ่มเปลี่ยนมุมมองสิ่งที่ให้ความสำคัญไปเล็กน้อย จากการเน้นสำรวจดูว่าโมเดลสามารถฉลาดขึ้นได้แค่ไหน ไปสู่เรื่องว่ามันทำอย่างไร

ความกังวลเรื่องการสร้างความลำเอียงให้โมเดล หรือการพยายามทำให้โมเดลใช้งานได้ทั่วถึงจนเกินไปทำให้เกิดเครื่องมือที่น่าสนใจ เช่น What-if Tool (WIT) ขึ้นมา

เครื่องมือตัวนี้ช่วยให้นักวิทยาศาสตร์ข้อมูลขุดลงลึกไปในพฤติกรรมของโมเดลและทำให้เห็นว่าฟีเจอร์ หรือชุดข้อมูลใดมีผลกระทบอย่างไรกับผลลัพธ์ที่ออกมา

WIT ถูกแนะนำโดย Google และเปิดให้ใช้ผ่าน Tensorboard หรือ Jupyter notebook มันช่วยทำให้งานต่างๆ ง่ายขึ้น เช่น การเปรียบเทียบระหว่างโมเดล การแบ่งชุดข้อมูล การแสดงแผนภาพในแง่มุมต่างๆ และการแก้ไขข้อมูลดิบ ถึงแม้ว่า WIT จะช่วยให้งานวิเคราะห์เหล่านี้ง่ายขึ้น แต่มันยังต้องการความเข้าใจคณิตศาสตร์และทฤษฎีเบื้องหลังแต่ละโมเดลพวกนี้อย่างลึกซึ้ง

ผู้ใช้ทั่วไปไม่ควรจะคาดหวังเครื่องมือตัวใดจะช่วยกำจัดความเสี่ยงหรือลดความเสียหายที่เกิดขึ้นไปแล้วจากการประยุกต์ใช้โมเดลผิดประเภทหรือใช้อัลกอริทึมการสอนที่แย่

## Azure Data Factory สำหรับควบคุมไปป์ไลน์

*เฝ้าระวัง*

ปัจจุบัน Azure Data Factory (ADF) เป็นผลิตภัณฑ์หลักของ Azure สำหรับใช้ควบคุมไปป์ไลน์การประมวลผลข้อมูล

มันรองรับการนำเข้าข้อมูล การคัดลอกข้อมูลไปมาระหว่างแหล่งเก็บข้อมูลต่างๆ ทั้งใน Azure เองหรือศูนย์บริการข้อมูลข้างนอก และการสั่งโปรแกรมตรรกะการแปลงรูปให้ทำงาน

แม้เราจะมีประสบการณ์ที่ดีในระดับหนึ่งกับการใช้ ADF เพื่อย้ายข้อมูลง่ายๆ จากศูนย์ข้อมูลข้างนอกไปยังคลาวด์ เรากลับรู้สึกไม่อยากแนะนำให้ใช้ Azure Data Factory สำหรับควาบคุมไปป์ไลน์ของงานประมวลผลข้อมูลที่ซับซ้อน

มีหลายปัจจัยที่ทำให้ประสบการณ์ใช้งานของเราถูกท้าทาย เช่น ข้อจำกัดที่ความสามารถหลายๆ อย่างไม่สามารถทำผ่านโค้ดก่อน ดูเหมือนว่า ADF จะให้ความสำคัญกับ**การสร้างแพลตฟอร์มที่โค้ดน้อย**ไว้ก่อนเสียมากกว่า, ไม่สะดวกนักในการตรวจจับและรายงานข้อบกพร่องที่พบ, ความสามารถในการเฝ้าสังเกตที่ถูกจำกัดเนื่องจาก ADF ไม่สามารถนำบันทึกการเปลี่ยนแปลงของตัวเองไปเชื่อมกับบริการอื่นๆ ของ Azure ได้ เช่นบริการ Azure Data Lake หรือ DataBricks ทำให้ยากต่อการเฝ้าสังเกตระบบตั้งแต่ต้นน้ำถึงปลายน้ำ และกลไก source-triggering ของข้อมูลก็มีให้ใช้แค่ในบางภูมิภาคของคลาว์ดเท่านั้น

ในตอนนี้เราขอส่งเสริมให้ใช้เครื่องมือทางเลือกที่เป็นโอเพนซอร์ส (เช่น Airflow) ในการจัดการไปป์ไลน์ข้อมูลที่มีความซับซ้อนไปก่อน และจำกัดการใช้งาน ADF เฉพาะการคัดลอกธรรมดาหรือการบันทึกภาพรวมเท่านั้น

เราหวังว่า ADF จะนำความกังวลเหล่านี้ไปปรับปรุงเพื่อรองรับขั้นตอนงานประมวลผลข้อมูลที่มีความซับซ้อนมากยิ่งขึ้น และให้ความสำคัญกับการใช้งานความสามารถต่างๆ ผ่านโค้ดไว้ก่อนมากขึ้นเช่นกัน


*© ThoughtWorks, Inc. All Rights Reserved.*

# ภาษาและเฟรมเวิร์ก

## Arrow
*ทดลอง*

Arrow เป็นไลบรารีเสริมสำหรับเขียนโปรแกรมเชิงฟังก์ชันสำหรับภาษา Kotlin โดยถือกำเนิดจากการควบรวมระหว่างสองไลบรารียอดนิยมเข้าด้วยกัน (kategory และ funKTionale)

ในขณะที่ภาษา Kotlin นั้นจัดสรรโครงสร้างพื้นฐานสำหรับการเขียนโปรแกรมเชิงฟังก์ชันไว้ให้แล้ว Arrow เข้ามาเติมเต็มโดยเป็นไลบรารีเสริมที่มาพร้อมกับชุดเครื่องมือที่พร้อมใช้ ที่มีแอบสแตรกชันระดับสูงกว่าเพื่อให้นักพัฒนาแอปพลิเคชันเอาไปประยุกต์ใช้ได้ทันที

Arrow นั้นมีให้ทั้ง data type, type class, effect, optic และ functional pattern อื่นๆ รวมถึงการเชื่อมต่อกับไลบรารียอดนิยมต่างๆ ให้ใช้

ความประทับใจที่ได้จาก Arrow ในแรกพบ ถูกยืนยันอีกครั้งผ่านการใช้มันสร้างแอปพลิเคชันหลากหลายที่ทำงานอยู่ในโปรดักชันในปัจจุบัน

## Flutter
*ทดลอง*

หลายๆ ทีมของเราใช้ Flutter แล้วชื่นชอบเป็นอย่างมาก มันคือเฟรมเวิร์คข้ามแพลตฟอร์มที่ทำให้คุณพัฒนาแอปพลิเคชันมือถือแบบเนทีฟด้วยภาษา Dart ดังนั้นมันจึงได้รับผลประโยชน์จากความสามารถของ Dart โดยตรง เช่น มันสามารถคอมไพล์เป็นโค้ดเนทีฟที่สื่อสารกับแพลตฟอร์มเป้าหมายโดยปราศจากตัวกลาง และนักพัฒนาไม่ถูกรบกวนทางความคิดจากการสลับระหว่างภาษาระหว่างพัฒนาเหมือนเครื่องมือตัวอื่น

ความสามารถฮอตรีโหลดของมันยังคงน่าประทับใจ เมื่อแก้ไขโค้ดเราจะเห็นผลลัพธ์โดยทันที เรามั่นใจที่จะแนะนำให้คุณลอง Flutter ในสักโครงการของคุณดู

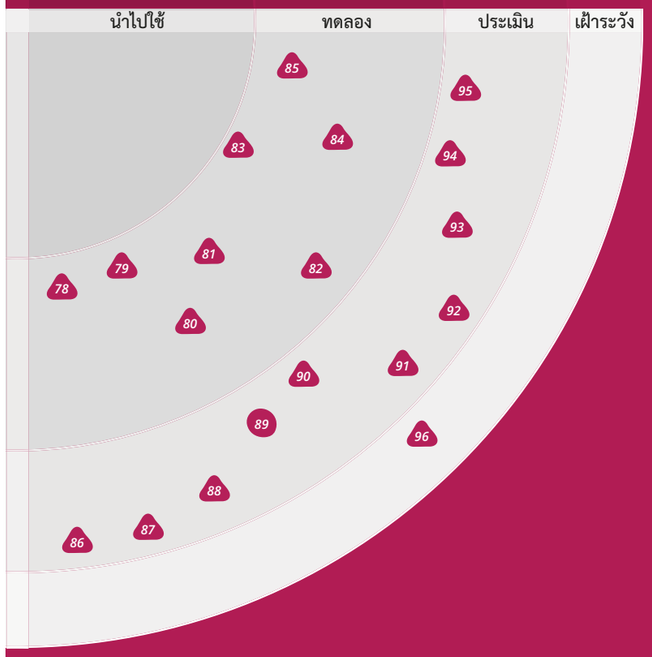
## jest-when
*ทดลอง*

jest-when คือ JavaScript ไลบรารีขนาดเล็ก ซึ่งเข้ามาเติมเต็ม Jest ในด้านการจับคู่เปรียเทียบอาร์กิวเมนต์ของฟังก์ชันจำลอง Jest เป็นเครื่องมือที่เยี่ยมยอดสำหรับการทดสอบอยู่แล้ว ในขณะที่ jest-when ช่วยให้คุณสามารถทดสอบอาร์กิวเมนต์ที่เฉพาะเจาะจงของฟังก์ชันจำลองได้ ดังนั้นคุณจะเขียนการทดสอบระดับยูนิตแต่ละส่วนงานที่ทนทานขึ้น

## Micronaut
*ทดลอง*

Micronaut เป็นเฟรมเวิร์คสำหรับสร้างไมโครเซอร์วิสที่ทำงานบน JVM โดยรองรับทั้งภาษา Java, Kotlin และ Groovy

มันแตกต่างกับเฟรมเวิร์คอื่น ตรงที่มันใช้หน่วยความจำน้อยกว่า และใช้เวลาในการเริ่มต้นทำงานที่เร็วกว่า สาเหตุที่เป็นเช่นนี้ได้ เกิดจากการออกแบบสถาปัตยกรรมภายในเพื่อทำ dependency injection (DI) ที่แตกต่างโดยไม่มีการใช้เทคนิครันไทม์รีเฟลคชันและการสร้างพร็อกซี่คลาสเลย ซึ่งเป็นเทคนิคที่มักพบได้บ่อยครั้งและเป็นจุดอ่อนของเฟรมเวิร์คอื่น ซึ่ง Micornaut ใช้เทคนิคที่ยอมให้การทำ DI/AOP เกิดขึ้นได้ ณ ตอนคอมไพล์ไทมเท่านั้น





**นำไปใช้**

**ทดลอง**

78. Arrow
79. Flutter
80. jest-when
81. Micronaut
82. React Hooks
83. React Testing Library
84. Styled components
85. Tensorflow

**ประเมิน**

86. Fairseq
87. Flair
88. Gatsby.js
89. GraphQL
90. KotlinTest
91. NestJS
92. Paged.js
93. Quarkus
94. SwiftUI
95. Testcontainers

**เฝ้าระวัง**

96. Enzyme



*© ThoughtWorks, Inc. All Rights Reserved.*

# ภาษาและเฟรมเวิร์ก

*Micronaut เป็นเฟรมเวิร์คสำหรับสร้างไมโครเซอร์วิสที่ทำงานบน JVM โดยรองรับทั้งภาษา Java, Kotlin และ Groovy มันแตกต่างจากเฟรมเวิร์คอื่นตรงที่มันใช้หน่วยความจำน้อยกว่าและใช้เวลาในการเริ่มต้นทำงานที่เร็วกว่า*

*(Micronaut)*

*React Testing Library เป็นตัวอย่างที่ดีของเฟรมเวิร์คที่ยิ่งใช้ลึกลงไปเท่าไหร่ ยิ่งบดบังทางเลือกอื่นมากขึ้น และได้กลายเป็นตัวเลือกแรกเมื่อต้องทดสอบระบบฟรอนท์เอนด์ที่พัฒนาด้วย React*

*(React Testing Library)*

ความสามารถนี้ไม่เพียงแต่ดึงดูดให้มันถูกนำไปใช้สร้างไมโครเซอร์วิสเท่านั้น แต่ถูกนำไปใช้ในบริบทอื่น เช่น ซอฟต์แวร์สำหรับงาน IoT แอนดรอยด์แอปพลิเคชัน หรือสร้าง serverless ฟังก์ชัน

Micronaut ทำงานอยู่บน Netty เซิร์ฟเวอร์อีกที ทำให้การเขียนโปรแกรมในเชิงปฏิกิริยานั้นทำได้ตรงตัว มีเครื่องมือช่วยเหลือชั้นเลิศ

นอกจากนี้ยังมีฟีเจอร์ที่แนบมาให้ในตัว เช่น service discovery และ circuit breaker ซึ่งเหมาะกับสถาปัตยกรรมยุคใหม่ที่อยู่ในรูปแบบคลาวด์เนทีฟ

Micronaut ถือว่าเป็นผู้ท้าชิงหน้าใหม่ที่น่าสนใจในตลาดเฟรมเวิร์คที่ทำงานบน JVM เราเห็นการใช้งานมันมากขึ้นเรื่อยๆ ในโปรดักชัน ทำให้เราขยับมันขึ้นมาอยู่ในหมวดหมู่น่าทดลอง

## React Hooks

*ทดลอง*

เมื่อต้นปีที่ผ่านมา React Hooks ได้ถูกเพิ่มเข้าเป็นส่วนหนึ่งของเฟรมเวิร์ก React อันโด่งดัง ซึ่งช่วยเพิ่มความสามารถที่จะใช้สถานะภายในและฟีเจอร์อื่นๆ ของ React โดยที่ไม่ต้องสร้างคลาส มันนำเสนอวิธีที่ดูสะอาดกว่าเทคนิคการสร้างคอมโพเนนต์ระดับสูง (higher-order component) หรือเทคนิค render-props

ไลบรารีหลายตัว อย่างเช่น Material UI และ Apollo ก็ได้เปลี่ยนมาใช้มันเป็นที่เรียบร้อยแล้ว แต่ก็ยังมีปัญหาอยู่บ้างกับการทดสอบคอนโพเนนต์ที่ใช้งาน Hooks โดยเฉพาะกับการใช้ร่วมกับ Enzyme ซึ่งเหตุการณ์นี้ก็นำไปสู่การประเมิน Enzyme ใหม่ ถึงความเหมาะสมของการเป็นเครื่องมือที่ควรใช้

## React Testing Library

*ทดลอง*

โลกของ JavaScript เปลี่ยนแปลงเคลื่อนไหวค่อนข้างรวดเร็ว และจากที่เรามีประสบการณ์มากขึ้นจากการใช้งานเฟรมเวิร์ค คำแนะนำของเราจึงเปลี่ยนไปเช่นกัน React Testing Library เป็นตัวอย่างที่ดีของเฟรมเวิร์คที่ยิ่งใช้ลึกลงไปเท่าไหร่ ยิ่งบดบังทางเลือกอื่นมากขึ้น และได้กลายเป็นตัวเลือกแรกเมื่อต้องทดสอบระบบฟรอนท์เอนด์ที่พัฒนาด้วย React

ทีมของเราชอบความจริงที่ว่า ชุดทดสอบที่เขียนด้วยเฟรมเวิร์คนี้เปราะบางน้อยกว่าเฟรมเวิร์คทางเลือกอื่น อย่าง Enzym เพราะคุณถูกส่งเสริมให้แยกทดสอบความสัมพันธ์ต่างๆ ระหว่างคอมโพเนนต์จากกัน ซึ่งตรงกันข้ามกับการทดสอบรายละเอียดวิธีทำภายใน

## Styled Component

*ทดลอง*

styled components ทำให้เราสามารถเขียน CSS เพื่อกำหนดสไตล์ของ React component ในโค้ดของคอมโพเนนต์ได้ตรงๆ เลย โดยใช้เทคนิค tagged template literal ของ JavaScript เข้าช่วย ซึ่งลดความลำบากในการจัดการกับ CSS ไปอย่างมาก และทำให้ไม่จำเป็นที่ต้องใช้วิธีตั้งชื่อตามธรรมเนียม (naming convention) หรือวิธีการอื่นๆ เพื่อหลีกเลี่ยงการตั้งชื่อชนกันใน CSS นอกจากนี้ยังไม่ต้องคอยจำ CSS จำนวนมหาศาลว่ามีอะไรอยู่ที่ไหนอีกด้วย

แต่แน่นอนว่าการนำ CSS เข้าไปใส่ไว้ใน Javascript โดยตรงเช่นนี้ ก็อาจทำให้ยากขึ้นต่อการมองภาพรวมและการจัดการสไตล์ที่อยู่ในคอมโพเนนต์ต่างๆ เราจึงอยากแนะนำว่าถ้าจะนำมันไปใช้ ให้เข้าใจว่ากำลังแลกกับอะไรเสียก่อน

## Tensorflow

*ทดลอง*

จากการปล่อยเวอร์ชัน 2.0 ออกมา ทำให้ TensorFlow ยังรักษาความเป็นผู้นำในอุตสาหกรรมของการเป็นเฟรมเวิร์คสำหรับทำแมชชีนเลิร์นนิง

TensorFlow เริ่มต้นจากการเป็นไลบรารีในการประมวลผลเชิงตัวเลข จากนั้นค่อยๆ ขยายกลายเป็นไลบรารีที่สนับสนุนวิธีการและสภาพแวดล้อมการประมวลผลที่หลากหลายในการทำแมชชีนเลิร์นนิง ตั้งแต่ระดับซีพียูสำหรับโทรศัพท์มือถือ ไปจนถึงระดับคลัสเตอร์ของจีพียูขนาดใหญ่ ระหว่างนั้นก็มีเฟรมเวิร์คเกิดขึ้นมากมายเพื่อทำให้งานสร้างเครือข่ายและการฝึกสอนง่ายขึ้น ในช่วงเวลานั้นเองเฟรมเวิร์คอื่นๆ โดยเฉพาะ PyTorch ได้นำเสนอโมเดลการเขียนโปรแกรมเชิงคำสั่ง (imperative programming) ออกมา ทำให้การดีบั๊กและการสั่งงานเป็นเรื่องที่ง่ายขึ้น

ปัจจุบัน TensorFlow 2.0 กำหนดให้การเขียนโปรแกรมเชิงคำสั่ง (แบบดำเนินการทันที) เป็นค่าเริ่มต้นแล้ว และเลือกใช้ Keras เป็น API ระดับสูงเพียงตัวเดียว แม้การเปลี่ยนแปลงนี้จะทำให้การใช้งาน TensorFlow ทันสมัยมากขึ้นและทำให้มันแข่งขันกับ PyTorch ได้ แต่มีหลายส่วนถูกเขียนขึ้นใหม่อย่างมีนัยสำคัญ ซึ่งบ่อยครั้งทำให้ส่วนต่างๆ ไม่สามารถเข้ากันได้กับของเก่าที่มี ทั้งเครื่องมือและเฟรมเวิร์คต่างๆ ที่อยู่ในระบบนิเวศของ Tensorflow เดิมจะไม่สามารถทำงานกับเวอร์ชันใหม่ได้ทันที

ดังนั้นช่วงเวลานี้ลองพิจารณาว่าคุณอยากจะออกแบบและทำการทดลองใน TensorFlow 2.0 แต่ย้อนมันกลับไปใช้เวอร์ชันแรก เพื่อให้บริการและรันโมเดลในโปรดักชันหรือไม่

## Fairseq

*ประเมิน*

Fairseq เป็นชุดเครื่องมือสำหรับการสร้างโมเดลแบบลำดับสู่ลำดับ (sequence-to-sequence) โดย Facebook AI Research

ซึ่งทำให้นักวิจัยและนักพัฒนาสามารถนำโมเดลที่ทำขึ้นเฉพาะมาฝึกสอนเพื่อสร้างโมเดลที่รองรับ งานแปล สรุปความ ออกแบบโมเดลภาษา และงานอื่นๆ ทางภาษาธรรมชาติ

สำหรับผู้ใช้ PyTorch นี่เป็นตัวเลือกที่ดี เนื่องจาก Fairseq ให้โค้ดตัวอย่างสำหรับอ้างอิงโมเดลแบบลำดับสู่ลำดับหลากหลายรูปแบบด้วยกัน รองรับการฝึกสอนโมเดลแบบกระจายข้ามหลาย GPU และหลายเครื่อง ต่อขยายได้ง่าย และมีกลุ่มโมเดลที่ผ่านการฝึกสอนมาล่วงหน้าให้เลือกใช้ หนึ่งในนั้นคือ RoBERTa ซึ่งเป็นโมเดลที่ปรับปรุงประสิทธิภาพต่อยอดจาก BERT อีกที


*© ThoughtWorks, Inc. All Rights Reserved.*

## Flair
*ประเมิน*

Flair เป็นเฟรมเวิร์คที่เรียบง่ายสำหรับงานประมวลผลภาษาธรรมชาติ มันมีพื้นฐานอยู่บนภาษา Python และครอบคลุมงานมาตรฐานในภาษาธรรมชาติ เช่น การรู้จำชื่อ (named entity recognition: NER), วจีวิภาค (part-of-speech tagging: PoS), การแก้ปัญหาความกำกวม (word-sense disambiguation), และการจัดแบ่งประเภท และยังทำงานได้แม่นยำในงานต่างๆ ด้วยเช่นกัน

Flair มี API ที่เรียบง่าย และพยายามรวมเอมเบดดิงของคำและเอกสารหลากหลายรูปแบบไว้เป็นหนึ่งเดียว ซึ่งรวมไปถึง BERT, Elmo และเอ็มเบ็ดดิงของ Flair เองด้วย

นอกจากนี้ Flair ยังรองรับการประมวลหลายภาษาธรรมชาติ ตัวเฟรมเวิร์คเองถูกสร้างขึ้นบน Pytorch อีกทีหนึ่ง พวกเราใช้ Flair อยู่ในบางโครงการ และชอบในความง่ายในการใช้งานและความเรียบง่ายที่ทรงพลังของมัน

## Gatsby
*ประเมิน*

Gatsby.js เป็นเฟรมเวิร์คที่ใช้ในการเขียนเว็บแอปพลิเคชันในสถาปัตยกรรมรูปแบบ JAMstack

ส่วนหนึ่งของแอปพลิเคชันจะถูกสร้างตั้งแต่ตอนตอนบิลด์ไทม์และถูกดีพลอยในลักษณะของเว็บไซต์ที่แสดงผลได้อย่างเดียว (static site) ความสามารถส่วนที่เหลือจะถูกจัดการต่อด้วย Progressive web application (PWA) ที่ทำงานบนเบราว์เซอร์ แอปพลิเคชันลักษณะนี้สามารถทำงานได้โดยไม่ต้องพึ่งโค้ดฝั่งเซิร์ฟเวอร์เลย เพื่อจัดการเนื้อหาของเว็บไซต์ โดยทั่วไปแอปพลิเคชันจะเรียก API จากเซอร์วิสภายนอกหรือโซลูชัน SaaS โดยตรงแทน

ในกรณีของ Gatsby.js โค้ดทั้งหมดจะถูกเขียนด้วย React ซึ่งเฟรมเวิร์คได้ปรับแต่งประสิทธิภาพความเร็วมาให้ในตัว เพื่อผู้ใช้จะรู้สึกลื่นในการใช้งาน โดยมันแบ่งโค้ดและข้อมูลออกจากกันให้ในตัว เพื่อลดเวลาในการโหลดหน้าเพจลง และดึงข้อมูลหน้าถัดไปเตรียมไว้ก่อน เพื่อเพิ่มประสิทธิภาพความเร็วระหว่างการท่องเว็บแอปพลิเคชัน

เรายังสามารถเรียก API ต่างๆ ผ่าน GraphQL และมีส่วนต่อขยายเพิ่มเติมหลายตัวที่ช่วยให้การเชื่อมต่อกับระบบอื่นเรียบง่ายขึ้น

## GraphQL
*ประเมิน*

หลายโครงการของเราประสบความสำเร็จในการนำ GraphQL ไปใช้งาน เราได้พบบางรูปแบบการใช้งานที่น่าสนใจอีกด้วย หนึ่งในนั้นคือการใช้ GraphQL เป็นตัวรวบรวมข้อมูลทางฝั่งเซิร์ฟเวอร์

ถึงอย่างนั้นก็ตาม เรากังวลว่ามันจะถูกนำไปใช้งานอย่างผิดๆ หรืออาจมีบางปัญหาเกิดขึ้นได้ ตัวอย่างเช่น ปัญหาประสิทธิภาพที่เกิดจากงานคิวรี่ประเภท N+1 หรือความซับซ้อนที่เกิดจากการเพิ่มโค้ดตั้งต้นจำนวนมากเมื่อต้องการเพิ่มโมเดลใหม่เข้าไป ซึ่งก็มีทางหลบปัญหาเหล่านี้ด้วยกันหลายวิธี เช่นการทำคิวรีแคชชิง

แม้มันจะไม่ใช่คำตอบของทุกปัญหา แต่ก็น่าลองประเมินดูว่าควรนำ GraphQL ไปใช้ในส่วนไหนของสถาปัตยกรรมของคุณ

## KotlinTest
*ประเมิน*

KotlinTest คือเครื่องมือที่อยู่ได้ด้วยตัวเองเพื่อใช้ทดสอบสำหรับระบบนิเวศของภาษา Kotlin ที่ทีมของพวกเราชื่นชอบ มันสามารถทำ property-based testing ได้ ซึ่งเป็นเทคนิคที่เราพูดให้ความสำคัญไปในเรดาร์ครั้งก่อน ข้อแตกต่างที่สำคัญ คือ มันรองรับวิธีการเขียนการทดสอบหลากหลายรูปแบบเพื่อจัดโครงสร้างชุดทดสอบให้เหมาะสม และมีตัวจับคู่เปรียบเทียบ (matchers) ที่ครบครันช่วยให้การเขียนชุดทดสอบสามารถเล่าเป็นเรื่องราวผ่าน DSL ที่งดงาม

## NestJS
*ประเมิน*

NestJS คือ Node.js เฟรมเวิร์คสำหรับทำงานที่ฝั่งเซิร์ฟเวอร์ ซึ่งพัฒนาด้วยภาษา TypeScript ด้วยการเชื่อมระบบนิเวศที่อุดมสมบูรณ์ของชุมชน node ต่างๆ เข้าด้วยกัน ทำให้ NestJS เตรียมสถาปัตยกรรมที่มีพร้อมทุกอย่างสำหรับสร้างเว็บแอปพลิเคชัน

NestJS ถูกพัฒนาขึ้นมาด้วยความตั้งใจให้นักพัฒนาใช้เวลาในการเรียนรู้น้อยตัวเฟรมเวิร์คน้อย มันยังรองรับโปรโตคอลหลากหลาย เช่น GraphQL Websocket และไลบรารี ORM ของค่ายต่างๆ

## Paged.js
*ประเมิน*

เมื่อใช้ HTML และเทคโนโลยีที่เกี่ยวข้องในการสร้างหนังสือและสิ่งพิมพ์แล้ว การจัดหน้าให้เหมาะสมเป็นสิ่งที่ต้องคำนึงถึง ซึ่งรวมถึงการใส่เลขหน้า การนับจำนวนหน้า องค์ประกอบในแต่ละหน้าที่ซ้ำกันในส่วนหัวและส่วนท้ายของหน้า รวมไปถึงกลไกต่างๆ เพื่อหลีกเลี่ยงกรณีขึ้นหน้าใหม่แบบแปลกๆ
Paged.js เป็นโอเพนซอร์สไลบรารีที่เติมเต็มส่วนที่ขาดหายสำหรับมาตรฐาน Paged Media และมาตรฐาน Generated Content for Paged Media ของ CSS module แม้ว่า Paged.js ยังอยู่ในช่วงของการทดลอง แต่ก็สามารถช่วยเติมเต็มช่องว่างที่สำคัญ เรื่อง “เขียนครั้งเดียว ตีพิมพ์ได้ทุกที่” ของ HTML

## Quarkus
*ประเมิน*

Quarkus เป็นเฟรมเวิร์คสำหรับสร้างแอปพลิเคชันด้วยภาษา Java ที่พัฒนาโดย Red Hat ออกแบบโดยให้ความสำคัญกับคอนเทเนอร์ไว้ก่อนและรองรับสถาปัตยกรรมแบบคลาวด์เนทีฟ มันใช้เวลาสั้นมากในการเริ่มต้นทำงาน (ระดับสิบมิลลิวินาที) และใช้พื้นที่หน่วยความจำน้อย ทำให้เหมาะมากกับ FaaS (Function as a Service)

# ภาษาและเฟรมเวิร์ก

*Gatsby.js เป็นเฟรมเวิร์คที่ใช้ในการเขียนเว็บแอปพลิเคชันในสถาปัตยกรรมรูปแบบ JAMstack มันแบ่งโค้ดและข้อมูลออกจากกันให้ในตัวเพื่อลดเวลาในการโหลดหน้าเพจลง และดึงข้อมูลหน้าถัดไปเตรียมไว้ก่อน เพื่อเพิ่มประสิทธิภาพความเร็วระหว่างการท่องเว็บแอปพลิเคชัน*

*(Gatsby.js)*

*Quarkus เป็นเฟรมเวิร์คสำหรับสร้างแอปพลิเคชันด้วยภาษา Java ที่พัฒนาโดย Red Hat ออกแบบโดยให้ความสำคัญกับคอนเทเนอร์ไว้ก่อน มันใช้เวลาสั้นมากในการเริ่มต้นทำงาน และใช้พื้นที่หน่วยความจำน้อย*

*(Quarkus)*


*© ThoughtWorks, Inc. All Rights Reserved.*

# ภาษาและเฟรมเวิร์ก

*การทดสอบแบบอัตโนมัติกับการสร้างสภาพแวดล้อมที่ไว้ใจได้เป็นประเด็นที่อยู่กันเสมอมา Testcontainers เป็นไลบรารีที่ช่วยจัดการปัญหานี้ทำให้สิ่งพึ่งพาอ้างอิงถูกหุ้มด้วยคอนเทเนอร์*

*(Testcontainers)*

หรือเซอร์วิสที่มีการปรับขยายขึ้นหรือลงบ่อยๆ จากระบบควบคุมคอนเทเนอร์

Quarkus นั้นเหมือนกับ Micronaut ตรงที่สามารถใช้เทคนิคคอมไพล์ล่วงหน้าในการทำ dependency injection ณ ตอนคอมไพล์ไทม์ และสามารถหลีกเลี่ยงการเสียค่าใช้จ่าย ณ รันไทม์จากการใช้รีเฟลคชัน มันยังสามารถทำงานร่วมกับอิมเมจของ GraalVM Native ได้ดี ซึ่งยิ่งช่วยให้ลดเวลาในการเริ่มต้นทำงานลงได้อีก

Quarkus รองรับโมเดลการเขียนโปรแกรมทั้งเชิงปฏิกิริยา (reactive programming) หรือเชิงคำสั่ง (imperative programming) เคียงข้างกับ Micronaut และ Helidon ขณะนี้ Quarkus กำลังเป็นผู้นำยุคใหม่ของเฟรมเวิร์คสำหรับภาษา Java ที่พยายามปรับปรุงประสิทธิภาพของเวลาในการเริ่มต้นทำงานและการใช้หน่วยความจำ โดยไม่ทำให้นักพัฒนาเสียประสิทธิผลการทำงาน

มันได้รับความสนใจจากชุมชนนักพัฒนามากขึ้นเรื่อยๆ และน่าจับตามองไว้

## SwiftUI

*ประเมิน*

Apple เลือกเปลี่ยนแปลงครั้งใหญ่ด้วยการนำเสนอ SwiftUI ซึ่งเป็นเฟรมเวิร์คที่ใช้สร้างส่วนต่อประสานผู้ใช้สำหรับแพลตฟอร์ม macOS และ iOS

เราชอบการตัดสินใจของ SwiftUI ที่เลือกทิ้งความสัมพันธ์ที่งุ่มง่ามระหว่าง Interface Builder และ Xcode ไปใช้วิธีใหม่ที่เป็นการประกาศ ที่มีความสอดประสานระหว่างกันโดยมีโค้ดเป็นศูนย์กลาง คุณสามารถเทียบโค้ดและการแสดงผลของมันแบบเคียงข้างกันใน XCode 11 ทำให้ประสบการณ์การพัฒนาดีขึ้นกว่าเดิมมาก

SwiftUI เฟรมเวิร์คได้รับแรงบันดาลใจมาจากโลก React.js ที่เป็นผู้นำด้านการพัฒนาเว็บไซต์ในช่วงหลายปีมานี้ การรองรับการใช้ค่าข้อมูลที่ไม่เปลี่ยนรูป (Immutable values) ในโมเดลของการแสดงผล (view model) และการรองรับกลไกการเปลี่ยนแปลงข้อมูลแบบไม่พร้อมกัน (asynchronous update) ทำให้มันรวมเป็นหนึ่งเดียวกับการเขียนโปรแกรมเชิงปฏิกิริยา ทำให้นักพัฒนามีทางเลือกอื่นที่เป็นแบบเนทีฟ นอกเหนือจากเฟรมเวิร์คประเภทเดียวกัน อย่าง React Native หรือ Flutter

แม้ SwiftUI จะเป็นตัวแทนที่ Apple วางไว้สำหรับอนาคตการพัฒนาส่วนต่อประสานผู้ใช้ แต่มันยังค่อนข้างใหม่และต้องให้เวลากว่ามันจะเข้าที่ เราตั้งตารอที่จะเห็นคู่มือการใช้งานที่ดีขึ้นและการเกิดของชุมชนนักพัฒนาที่ร่วมกันตั้งแนวปฏิบัติต่างๆ สำหรับทำการทดสอบ SwiftUI และห่วงทางวิศวกรรมอื่นๆ

## Testcontainers

*ประเมิน*

การทดสอบแบบอัตโนมัติกับการสร้างสภาพแวดล้อมที่ไว้ใจได้ เป็นประเด็นที่อยู่คู่กันมาเสมอ โดยเฉพาะอย่างยิ่งเมื่อจำนวนส่วนประกอบที่ระบบสมัยใหม่ต้องพึ่งพาอ้างอิงกลับมีสูงขึ้นเรื่อยๆ Testcontainers เป็นไลบรารีสำหรับภาษา Java ที่ช่วยจัดการปัญหานี้ด้วยการทำให้สิ่งที่พึ่งพาอ้างอิงนั้นถูกหุ้มด้วยคอนเทเนอร์อีกทีก่อนจะทำการทดสอบ

สิ่งนี้มีประโยชน์อย่างมากสำหรับเปิดและปิดฐานข้อมูลอยู่เรื่อยๆ เพื่อใช้ในการทดสอบ แต่ไม่ใช่แค่นั้นมันสามารถนำไปใช้หุ้มเบราว์เซอร์ได้ด้วยเพื่อทำการทดสอบส่วนต่อประสานผู้ใช้ ทีมของเราพบว่าไลบรารีตัวนี้ช่วยทำให้การทดสอบการเชื่อมต่อของแต่ละระบบน่าเชื่อถือมากขึ้น อีกทั้งยังสามารถเขียนโปรแกรมสั่งการได้ มีขนาดเล็กและเมื่อใช้งานเสร็จแล้วสามารถโยนคอนเทเนอร์ทิ้งได้เลย

## Enzyme

*เฝ้าระวัง*

ส่วนใหญ่เรามักไม่ย้ายเครื่องมือที่ถูกแทนที่แล้วไปที่กลุ่มเฝ้าระวังในเรดาร์ แต่ทีมของเรารู้สึกอย่างยิ่งว่า Enzyme (เครื่องมือไว้ทดสอบระดับยูนิตของคอมโพเนนต์ส่วนต่อประสานผู้ใช้ของ React) ควรถูกแทนที่โดย React Testing Library ได้แล้ว เนื่องจากหลายทีมพบว่า Enzyme มุ่งเน้นไปที่การทดสอบสถานะภายในคอมโพเนนต์ ซึ่งนำไปสู่การสร้างชุดทดสอบที่เปราะบางและยากในการบริหารเปลี่ยนแปลง


*© ThoughtWorks, Inc. All Rights Reserved.*

***ต้องการข่าวสารและข้อมูลเชิงลึกล่าสุด***
***เกี่ยวกับเรดาร์เทคโนโลยี***

สามารถติดตามบนช่องทางต่างๆ
ตามที่คุณถนัด หรือ สมัครรับข่าวสารผ่านทางอีเมล์

*กดติดตามเลย*

**ThoughtWorks®**

ThoughtWorks คือบริษัทให้คำปรึกษาและวางแผนทางด้านเทคโนโลยีระดับโลก
เราเป็นชุมชนของนักสร้างสรรค์ ผู้ที่มีเป้าหมายและความหลงไหลในสิ่งที่เหมือนกัน
เราช่วยลูกค้าของเราประยุกต์ใช้เทคโนโลยีเพื่อช่วยขับเคลื่อนธุรกิจ
พร้อมทั้งสร้างซอฟต์แวร์ที่มีความสำคัญให้กับพวกเขาอีกด้วย
นอกจากนี้การใช้ซอฟต์แวร์เพื่อขับเคลื่อนสังคมก็เป็นอีกภารกิจสำคัญของเรา
เราร่วมมือกับหลากหลายองค์กรที่มีวัตถุประสงค์เดียวกัน
เพื่อการสร้างความเปลี่ยนแปลงที่ดีให้เกิดขึ้นกับสังคม

กว่า 25 ปีที่ ThoughtWorks
ได้เติบโตเป็นบริษัทที่มีผู้เข้าร่วมกว่า 7 ,000 คนทั่วโลก
เป็นการรวมกลุ่มของคนที่หลากหลาย
ที่แต่ละคนมีภารกิจในการคิดค้นเครื่องมือใหม่
เพื่อตอบโจทย์การพัฒนาวงการเทคโนโลยีระดับโลก
ThoughtWorks มีสำนักงานทั้งหมด 43 แห่ง
กระจายอยู่ 14 ประเทศทั่วโลก ได้แก่ ประเทศไทย,
ออสเตรเลีย, บราซิล,
แคนาดา, ชิลี, จีน, เอกวาดอร์, เยอรมัน,
อินเดีย, อิตาลี, สิงคโปร์, สเปน,
สหราชอาณาจักร และ สหรัฐอเมริกา

*thoughtworks.com*


*© ThoughtWorks, Inc. All Rights Reserved.*

ThoughtWorks®

*thoughtworks.com/radar*

*#TWTechRadar*