/thoughtworks

TECHNOLOGY RADAR

คู่มือนำทางสู่ปลายขอบเทคโนโลยี
ตามแบบฉบับของเรา

Volume 24

#TWTechRadar
thoughtworks.com/radar

ผู้ร่วมสร้างสรรค์

เรดาร์เทคโนโลยีถูกจัดทำขึ้น โดยคณะกรรมการที่ปรึกษาด้านเทคโนโลยี

คณะกรรมการที่ปรึกษาด้านเทคโนโลยีประกอบด้วยผู้นำและนักเทคโนโลยีผู้มากประสบการณ์ 20 คนของ Thoughtworks พวกเขาพบปะกันซึ่งหน้าประจำปีละ 2 ครั้งและทางโทรศัพท์ในทุกๆ 2 สัปดาห์ โดยมีหน้าที่สำคัญคือให้คำแนะนำ และเป็นที่ปรึกษาให้ประธานเจ้าหน้าที่บริหารฝ่ายเทคโนโลยี Rebecca Parsons

คณะกรรมการที่ปรึกษาจะทำหน้าที่เป็นผู้พิจารณาเทคโนโลยีที่กำลังมีบทบาทสำคัญในอุตสาหกรรมโลกขณะนั้น และมีผลกระทบต่อนักเทคโนโลยีของ Thoughtworks เพื่อกำหนดแผนยุทธศาสตร์ภายในเนื่องจากสถานการณ์โรคระบาดที่โลกกำลังประสบอยู่ คู่มือฉบับนี้จึงเกิดจากการประชุมแบบเสมือนจริงอีกครั้งหนึ่ง

Rebecca Parsons (CTO)

Martin Fowler (Chief Scientist)

Bharani Subramaniam

Birgitta Böckeler

Brandon Byars

Camilla Crispim

Cassie Shum

Erik Dörnenburg

Evan Bottcher

Fausto de la Torre

Hao Xu

Ian Cartwright

James Lewis

Lakshminarasimhan Sudarshan

Mike Mason

Neal Ford

Perla Villarreal

Rachel Laycock

Scott Shaw

Shangqi Liu

Zhamak Dehghani

ทีมแปลภาษาไทย:
พิรดา ทิพย์รัตน์, ณัฐพงศ์ อินทรักษ์, สันทบ บัวแก้ว, วรพล ช่วยพันธ์, พิชญุตม์ จิตรศิลป์ฉายากุล, ธีรศักดิ์ ขอพุทธพรชัย, ศอลาฮุดดีน เฉลิมไทย, ธนกฤต จูฑะมงคล, สุธัม ธรรมวงศ์, วณิชนันท์ สินพิทักษ์, ชญานิษฐ์ นิลรัตน์, Jose Barbosa, ฐปพงศ์ เศรษฐ์พิทักษ์, ชาคริต ลิขิตขจร, คณิศร สุธรรม, พชร ตันทนิส, ไกวัลยวิชย์ ฉวรรณกุล, ภควัต อเนกวิโรจน์, ปัทมา ทวนชัยศรี, ณัฐวรรณ วงษ์ตั้ง

เกี่ยวกับเรดาร์เทคโนโลยี

พวกเรา Thoughtworkers ล้วนมีความหลงใหลในเทคโนโลยี เราสร้าง วิจัย ทดสอบ เปิดโอเพนซอร์ส ผลิตงานเขียน และมีเป้าหมาย ที่จะยกระดับเทคโนโลยีให้ดีขึ้นอย่างต่อเนื่องเพื่อทุกคนและทุกธุรกิจ พันธกิจของพวกเราคือ “ผลักดันความเป็นเลิศทางซอฟต์แวร์ และปฏิวัติอุตสาหกรรมไอทีให้ดีขึ้นกว่าเดิม” การจัดทำและแบ่งปันเทคโนโลยีเรดาร์ก็เพื่อสนับสนุนพันธกิจนี้

คณะกรรมการที่ปรึกษาของ Thoughtworks ซึ่งประกอบไปด้วยผู้นำและผู้เชี่ยวชาญในเทคโนโลยีหลากหลายด้านที่มารวมตัวกันเป็นประจำในทุกปี เพื่อพูดคุยแลกเปลี่ยนถึงแผนยุทธศาสตร์ทางเทคโนโลยีภายในของ Thoughtworks เอง และแนวโน้มของเทคโนโลยีที่กำลังมีบทบาทสำคัญในอุตสาหกรรมโลกในขณะนั้น

การรวบรวมผลการประชุมดังกล่าวถูกจัดทำขึ้นเป็นเรดาร์เทคโนโลยีฉบับนี้ โดยนำเสนอในรูปแบบที่เป็นประโยชน์กับทุกคนในวงการ ตั้งแต่นักพัฒนาจนถึงผู้บริหารระดับสูง ตัวเนื้อหานั้นเราเจตนาสรุปให้กระชับได้ใจความ หากอยากทราบถึงรายละเอียดเพิ่มเติมเราขอส่งเสริมให้คุณทดลองใช้เทคโนโลยีเหล่านั้นด้วยตัวคุณเอง

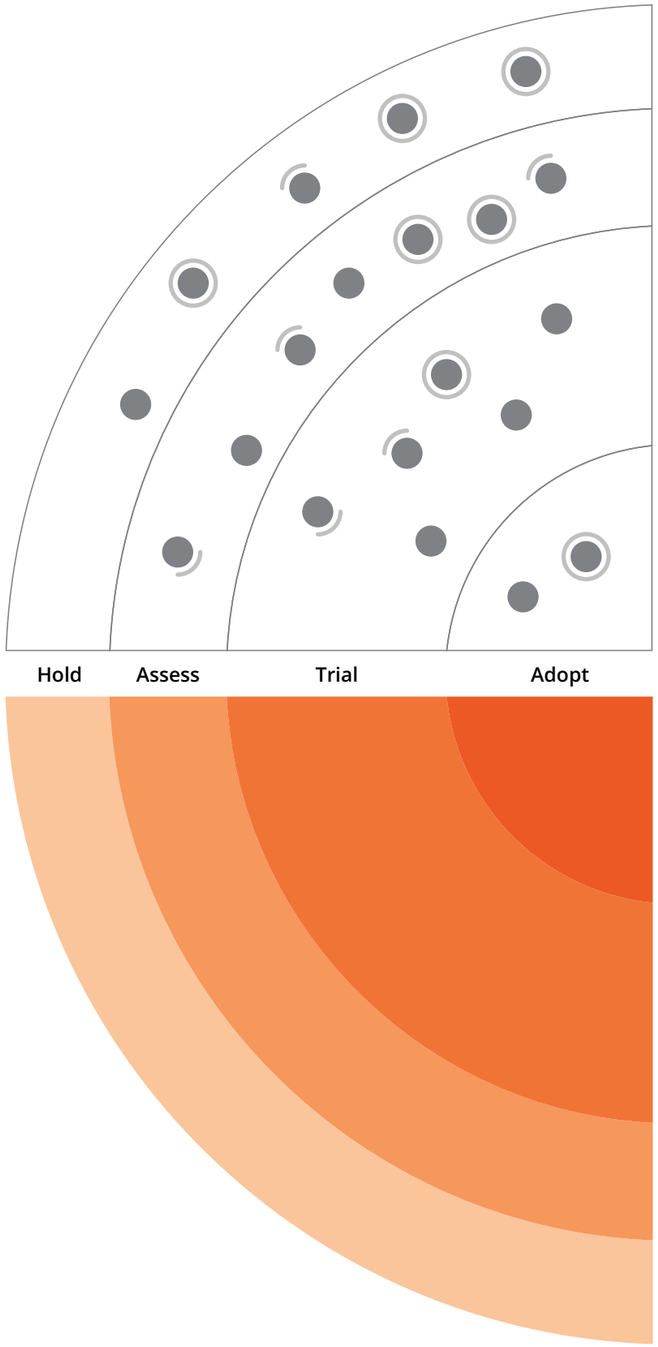
เรดาร์เทคโนโลยีใช้แผนภาพวงกลมในการนำเสนอข้อมูล โดยแบ่งพื้นที่ออกเป็น 4 กลุ่มดังนี้ เทคนิค เครื่องมือ แพลตฟอร์ม ภาษาและเฟรมเวิร์ค ในกรณีที่บางเรื่องสามารถจัดลงได้หลายกลุ่มเราจะจัดลงในกลุ่มที่เหมาะสมที่สุด นอกจากนั้นเรายังจัดกลุ่มเรื่องต่างๆ แล้วแบ่งตามวงแหวน ออกเป็น 4 ระดับเพื่อสะท้อนมุมมองตำแหน่งของเทคโนโลยี ตามความคิดเห็นของเรา

หากสนใจประวัติเพิ่มเติมเกี่ยวกับเรดาร์เทคโนโลยีสามารถดูเพิ่มเติมได้ที่ thoughtworks.com/radar/faq

ภาพรวมของเรดาร์

เรดาร์ คือการติดตามเทคโนโลยีที่น่าสนใจซึ่งเราเรียกว่าหัวข้อ เรานำเสนอเรดาร์เทคโนโลยีฉบับนี้เป็นแผนภาพวงกลม โดยแบ่งประเภทออกตามเสี้ยวและวงแหวน โดยเสี้ยวสื่อถึงประเภทของหัวข้อต่างๆ ในขณะที่วงแหวนสื่อถึงระดับชั้นของการนำเทคโนโลยีนั้นไปใช้ตามความเห็นของเรา

หัวข้อต่างๆ สื่อถึงเทคโนโลยีหรือเทคนิคที่มีบทบาทสำคัญต่อการพัฒนาซอฟต์แวร์ หัวข้อเป็นสิ่งที่เคลื่อนไหวได้ เมื่อระดับชั้นในวงแหวนมีการเปลี่ยนตำแหน่งไป ปกติเมื่อมีการขยับขึ้นจะสื่อถึงระดับความมั่นใจกับเทคโนโลยีที่เพิ่มมากขึ้นตาม



ใหม่

เลื่อนเข้า/ออก

ไม่มีการเปลี่ยนแปลง

เรดาร์ของเรามีการเปลี่ยนแปลงตลอดเวลาเพื่อนำเสนอสิ่งใหม่ เทคโนโลยีที่ไม่มีการเปลี่ยนแปลงจะถูกทำให้จางลงซึ่งไม่ได้หมายความว่าเทคโนโลยีนั้นๆ ไม่มีคุณค่าแต่เนื่องด้วยพื้นที่ของเรดาร์นั้นมีจำกัด

นำไปใช้

เราสนับสนุนให้นำเทคโนโลยีเหล่านี้ไปใช้ได้ทันทีซึ่งเราเองได้นำไปใช้แล้วในโครงการต่าง ๆ ที่มีความเหมาะสม.

ทดลอง

เราเห็นถึงความคุ้มค่าที่จะศึกษาเพิ่มเติม มันสำคัญที่จะเข้าใจวิธีการได้มาซึ่งความสามารถนี้ องค์กรควรทดลองใช้เทคโนโลยีนี้ในโครงการที่รับความเสี่ยงได้

ประเมิน

มีความคุ้มค่าที่จะสำรวจเพื่อทำความเข้าใจว่ามีผลต่อองค์กรอย่างไร

เฝ้าระวัง

ควรดำเนินการด้วยความระมัดระวัง

ประเด็นเด่นในฉบับนี้

ไปได้ไวกว่าด้วยแพลตฟอร์มทีม

หลายองค์กรได้หันมาสร้างทีมแพลตฟอร์มกันมากขึ้น ซึ่งคือทีมที่ตั้งขึ้นมาเฉพาะเพื่อทำให้ความสามารถบางอย่างกลายเป็นแพลตฟอร์ม ซึ่งแพลตฟอร์มนี้เองจะคอยให้บริการทีมอื่น ๆ ภายในองค์กร ให้การพัฒนาแอปพลิเคชันทำได้รวดเร็ว ไม่สิ้นเปลืองค่าใช้จ่ายในการดำเนินงาน และช่วยให้การออกผลิตภัณฑ์ไปสู่ตลาดทำได้เร็วยิ่งขึ้น เราเคยกล่าวถึงเรื่องนี้ไปแล้วในเรดาร์ปี 2017 แล้วเราก็ยินดีที่ได้เห็นเทคนิคนี้เติบโตและมีความพร้อมมากขึ้น ขณะเดียวกันเราก็เห็นถึงวิธีการผิด ๆ ในการสร้างทีมแพลตฟอร์มที่ควรหลีกเลี่ยงด้วยเช่นกัน อย่าง การสร้าง “แพลตฟอร์มเดียวครอบจักรวาล” ก็อาจจะไม่ใช่วิธีที่ให้ประสิทธิภาพสูงสุด หรือแพลตฟอร์มประเภท “เล่นใหญ่ไว้ก่อน” ก็อาจจะใช้เวลาเป็นปีกว่าจะเห็นผล หรือการ “สร้างแพลตฟอร์มไว้ก่อนเถอะ เดี๋ยวมีคนมาใช้เอง” ก็อาจเป็นการสิ้นเปลืองโดยใช่เหตุ เป็นต้น ตรงกันข้ามกัน หากนำแนวคิดการสร้างผลิตภัณฑ์มาใช้สร้างแพลตฟอร์มกลับจะช่วยให้แต่ละทีมรู้ว่าใครคือลูกค้าของตน แล้วสิ่งใดคือคุณค่าที่ผู้มาใช้งานต้องการ เป็นที่น่าเสียดายที่บางบริษัทกลับนำระบบการเปิดตั๋วที่นิยมใช้กันในการบริหารงานแบบเก่ามาขั้นระหว่างทีมแพลตฟอร์ม ทำให้ไม่ได้รับประโยชน์อย่างเต็มที่ แถมยังสืบทอดปัญหาเดิม ๆ ที่เกิดจากการทำงานแบบไซโลมาด้วย อย่างการตอบสนองที่ช้าเกินไป การแย่งกันใช้ทรัพยากรที่มีจำกัด เป็นต้น ส่วนในเรื่องดีเราก็พบเครื่องมือใหม่ ๆ และวิธีการแบ่งทีมและการประสานงานกัน ที่จะทำให้ได้ประสิทธิภาพดีกว่ามาอีกด้วย

เลือกสิ่งที่สะดวกมากกว่าสิ่งที่ดีที่สุด

การทำให้ระบบเป็นอัตโนมัติ และการรองรับการขยายทีมได้กลายเป็นสิ่งที่เป็นพื้นฐานของหลาย ๆ องค์กรไปแล้ว เราจึงเห็นแพลตฟอร์มต่าง ๆ พยายามรวบรวมเครื่องมือสำหรับนักพัฒนาเข้าด้วยกันและทำเป็นบริการคลาวด์ แต่ก่อนนักพัฒนามักจะเป็นผู้คัดสรรและร้อยเครื่องมือต่าง ๆ เข้าด้วยกันเอง (เช่น ระบบที่มีตัวจัดเก็บโค้ด ที่เชื่อมต่อกับไปป์ไลน์ ที่เชื่อมต่อกับตัวจัดเก็บแพ็คเกจ ที่เชื่อมต่อกับระบบวิกิ ฯลฯ) แต่ปัจจุบันแพลตฟอร์มสำหรับนักพัฒนาอย่าง Azure DevOps หรือระบบนิเวศของ GitHub ได้เข้ามากินส่วนแบ่งตลาดมากขึ้น แม้ว่าแต่ละแพลตฟอร์มอาจจะไม่สมบูรณ์แบบและมีความพร้อมแตกต่างกันไป แต่ก็ต้องยอมรับว่า ความเป็น “ร้านค้าครบวงจร” ของแพลตฟอร์มพวกนี้ก็ดึงดูดใจเหล่านักพัฒนาได้ดีทีเดียว โดยรวมแล้ว เราสังเกตุเห็นว่านักพัฒนาดูเหมือนจะเลือกความสะดวกสบายและความคล่องตัวมาก่อนเรื่องอื่น แต่ก็ต้องยอมรับว่าเครื่องมือที่ได้อาจจะไม่ได้สมบูรณ์ที่สุดไปทุกอย่าง

แล้วมันก็ “ซับซ้อนเกินกว่าจะสรุปให้สั้น” อีกครั้ง

ในกระบวนการสรรหาหัวข้อมาสู่เรดาร์ในแต่ละฉบับจะผ่านการถกเถียงกันอย่างหนัก บางหัวข้อที่ซับซ้อนมาก ๆ ก็มักจะจบสถานภาพลงที่ “TCTB - หรือ ซับซ้อนเกินกว่าจะสรุปให้สั้น” กล่าวคือ หัวข้อพวกนี้ถูกจัดกลุ่มอยู่ในสิ่งที่มีข้อดีและข้อเสียเต็มไปหมด หรือมีความละเอียดอ่อนเกินกว่าจะสรุปออกมาสั้นได้ในไม่กี่บรรทัด บ่อยครั้งที่หัวข้อพวกนี้ไปอยู่ในสื่อรูปแบบอื่นแทน เช่น บทความ หรือ พอดคาสต์ ทุกครั้งที่คุยกันถึงเรื่องพวกนี้ทีไร ก็จะถกกันนานและต้องนัดประชุมกันใหม่เพื่อหาข้อสรุปอยู่หลายรอบ ซึ่งเราจริงจังกันมาก บ่อยครั้งก็ต้องนัดประชุมกับทีมที่อยู่หน้างานเพื่อหาข้อมูลเพิ่มเติม และหลายครั้งเราก็หาข้อสรุปไม่ได้อยู่ดี ยกตัวอย่างเรื่องพวกนี้ เช่น เรื่องการรวมซอร์สโค้ดไว้ที่เดียวกัน (monorepo) เรื่องโมเดลการแตกบรานช์แต่ละแบบ หรือแนวทางที่ดีในการควบคุมระบบในสถาปัตยกรรมแบบกระจายตัว เป็นต้น บางคนอาจจะเคยสงสัยว่าทำไมหัวข้อสำคัญเหล่านี้ไม่เคยถูกยกมาพูดในเรดาร์เสียที ก็โปรดรับรู้ไว้เถิดว่า ไม่ใช่เราไม่รู้จัก หรือไม่อยากเขียนถึงหรอก แต่เป็นเพราะมีหลายหัวข้อในแวดวงการพัฒนาซอฟต์แวร์ที่มีข้อดีแลกข้อเสียมากเกินกว่าจะสรุปออกมาเป็นคำแนะนำที่ชัดเจนไม่คลุมเครือ บางครั้งหัวข้อย่อยที่แคบลงก็มีโอกาสลงเอยในเรดาร์ได้ง่ายกว่า ส่วนหัวข้อใหญ่ก็คงต้องรอบทสรุปกันต่อไปจนกว่าจะมีความชัดเจนมากขึ้น

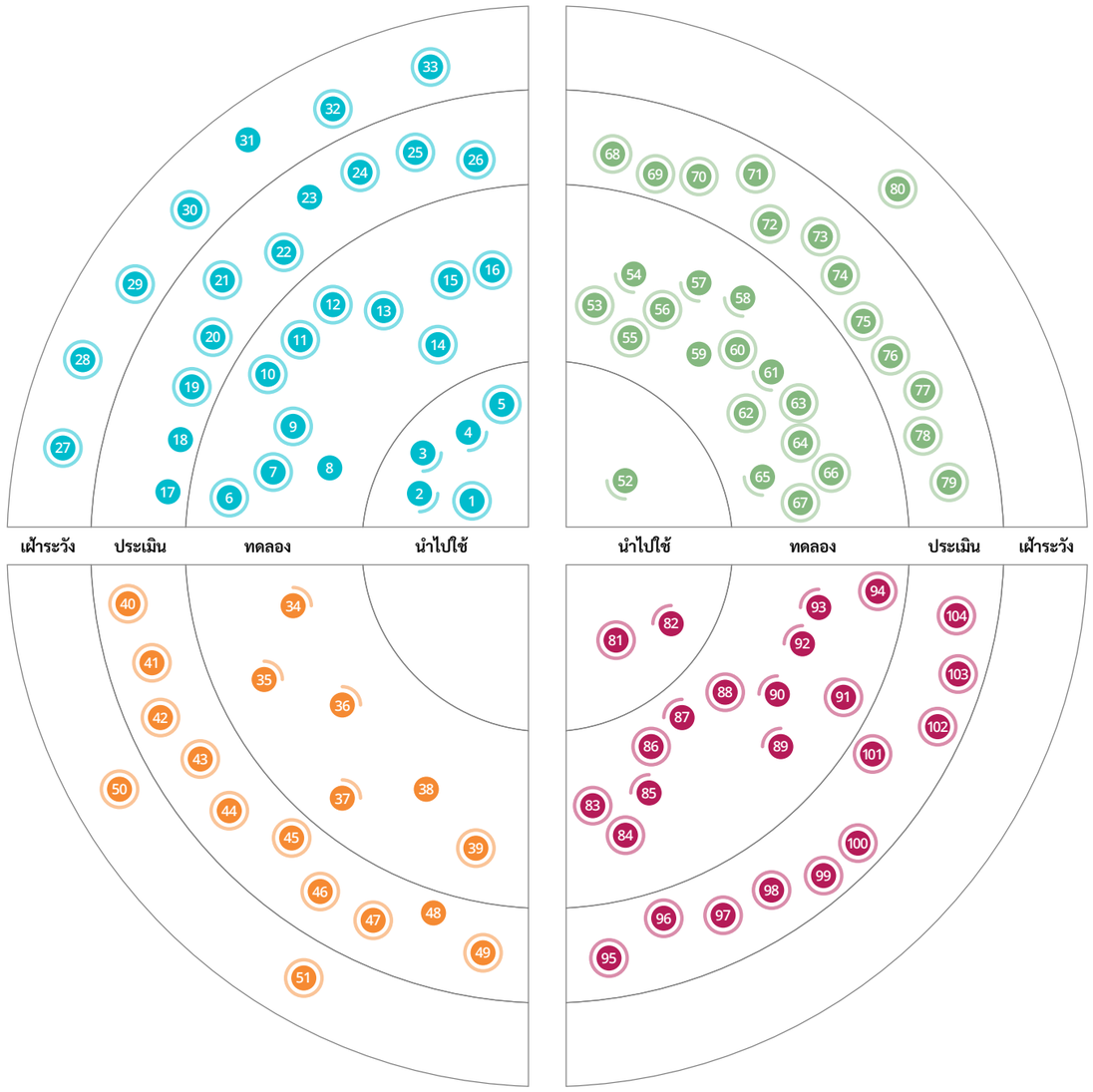
การทำความเข้าใจถึงระดับความขึ้นต่อกันในสถาปัตยกรรม

อีกหนึ่งเรื่องที่โผล่ขึ้นมาบ่อย ๆ ในการถกกันภายในของเราคือหัวข้อที่ว่าชิ้นส่วนต่าง ๆ ในสถาปัตยกรรมใดควรจะมีระดับความขึ้นต่อกัน (coupling) แบบไหน จะเป็นระหว่างภายในไมโครเซอร์วิสด้วยกันก็ดี หรือระหว่างคอมโพเนนต์ API เกตเวย์ กับฟรอนต์เอนด์ ฯลฯ ก็ตาม เรียกได้ว่าเกือบจะทุกครั้งที่ชิ้นส่วนซอฟต์แวร์สองส่วนต้องมาคุยกัน นักพัฒนาต้องมาคุยกันถึงเรื่องนี้อยู่ร่ำไป การจะยึดเอาคำแนะนำทั่วไปที่ชอบบอกให้เราแยกขาดกันให้มากที่สุด ก็ทำได้ยากในทางปฏิบัติเพราะจะทำงานกันไม่สะดวกเท่าไรนัก

มีหลายปัจจัยในการพิจารณาว่าควรจะตั้งระดับความขึ้นต่อกันแบบไหน คือดูว่ามันร้อยต่อกันอย่างไร หรือไปทำความเข้าใจความสัมพันธ์ระหว่างโดเมนนั้น ๆ หรือวิธีที่ทั้งสองใช้ในการสื่อสาร หรือดูปัจจัยด้านการทำทรานแซคชั่น แล้วบางครั้งก็ต้องดูเรื่องการรองรับการขยายประกอบไปด้วย โดยธรรมชาติแล้ว ซอฟต์แวร์ไม่อาจเกิดขึ้นได้ถ้าไม่มีความขึ้นต่อกันอยู่บ้างเลย เพราะฉะนั้นการจะตัดสินใจใช้ความขึ้นต่อกันระดับไหน ก็คือต้องพิจารณาถึงข้อดีข้อเสียประกอบกันเสมอ ซึ่งก็เป็นทักษะที่ต้องมีสำหรับการวางสถาปัตยกรรมในโลกสมัยใหม่

ในวงการเราก็เห็นวิธีปฏิบัติที่ไม่ดีแต่นิยมทำกัน เช่นการให้ซอฟต์แวร์สร้างโค้ดขึ้นมาเองแล้วนำไปใช้เป็นไคลแอนต์ไลบรารี แล้ววิธีการปฏิบัติที่ดีก็มีเหมือนกัน เช่นการตั้งใจใช้ BFF แพทเทิร์น อย่างไรก็ตาม จะร้ายหรือดี สำหรับเรื่องนี้แล้วมันไม่มีสูตรสำเร็จ และการตามคำแนะนำทั่วไปโดยไม่คิดนั้นก็เปล่าประโยชน์ ฉะนั้นแล้วการทุ่มเวลาและแรงงานเพื่อเข้าใจปัจจัยต่าง ๆ ที่เกี่ยวข้องกับกรณีนั้น ๆ ย่อมดีกว่าการแสวงหาวิธีสำเร็จรูปแต่ขาดประสิทธิภาพ

เรดาร์เทคโนโลยี



เทคนิค

นำไปใช้

1. การขยายสัญญาของ API
2. การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML)
3. Design systems
4. ทีมพัฒนาแพลตฟอร์มดั่งทีมพัฒนาผลิตภัณฑ์
5. การสับเปลี่ยนบัญชีผู้ใช้บริการ

ทดลอง

6. คลาวด์แซนด์บ็อกซ์
7. Contextual bandits
8. Docker อิมเมจที่ไม่มีลีนุกซ์ดิสโตร
9. Ethical Explorer
10. การใช้สมมุติฐานเป็นตัวขับเคลื่อนการปรับปรุงระบบ
11. การร้องขอข้อคิดเห็นด้วยวิธีที่เรียบง่าย
12. แมชชีนเลิร์นนิงที่เรียบง่ายที่สุดเท่าที่จะเป็นไปได้
13. SPA injection
14. ภาระการรับรู้ของทีม
15. จัดการไฟล์ Xcodeproj ด้วยเครื่องมือ
16. การใช้ type ร่วมกันระหว่าง UI/BFF

ประเมิน

17. แพลตฟอร์มเขียนโค้ดน้อยเฉพาะทาง
18. เอกลักษณ์กระจายศูนย์
19. Deployment drift radiator
20. การเข้ารหัสแบบโฮโมมอร์ฟิก
21. Hotwire
22. อิมพอร์ตแมพ สำหรับ ไมโครฟรอนท์เอนด์
23. Open Application Model (OAM)
24. Privacy-focused web analytics
25. การเขียนโปรแกรมแบบม็อบระยะไกล
26. Secure multiparty computing

เฝ้าระวัง

27. GitOps
28. ทีมแพลตฟอร์มที่ถูกแบ่งตามเลเยอร์ของเทคโนโลยี
29. นโยบายความซับซ้อนของรหัสผ่านที่คิดไปเองว่าดี
30. พูลรีเควสเป็นวิธีการเดียวในการตรวจทานโค้ดของเพื่อนร่วมงาน
31. SAFe™
32. แบ่งแยกทีมในการดูแลโค้ดกับไปป์ไลน์
33. แพลตฟอร์มที่ใช้การร้องขอในการดำเนินงาน

แพลตฟอร์ม

นำไปใช้

ทดลอง

34. AWS Cloud Development Kit
35. Backstage
36. Delta Lake
37. Materialize
38. Snowflake
39. ฟอนต์ที่เปลี่ยนแปลงรูปแบบได้ (Variable fonts)

ประเมิน

40. Apache Pinot
41. Bit.dev
42. DataHub
43. Feature Store
44. JuiceFS
45. การใช้งาน Kafka API โดยไม่ต้องใช้ Kafka
46. NATS
47. Opstrace
48. Pulumi
49. Redpanda

เฝ้าระวัง

50. Azure Machine Learning
51. เครื่องมือโครงสร้างพื้นฐานด้วยโค้ด (IaC) ฉบับทำเอง

เครื่องมือ

นำไปใช้

52. Sentry

ทดลอง

53. axe-core
54. dbt
55. esbuild
56. Flipper
57. Great Expectations
58. k6
59. MLflow
60. OR-Tools
61. Playwright
62. Prowler
63. Pyright
64. Redash
65. Terratest
66. Tuple
67. Why Did You Render

ประเมิน

68. Buildah และ Podman
69. GitHub Actions
70. Graal Native Image
71. HashiCorp Boundary
72. imgcook
73. Longhorn
74. Operator Framework
75. Recommender
76. Remote - WSL
77. Spectral
78. Yelp detect-secrets
79. Zally

เฝ้าระวัง

80. AWS CodePipeline

ภาษา & เฟรมเวิร์ค

นำไปใช้

81. Combine
82. LeakCanary

ทดลอง

83. Angular Testing Library
84. AWS Data Wrangler
85. Blazor
86. FastAPI
87. io-ts
88. Kotlin Flow
89. LitElement
90. Next.js
91. On-demand modules
92. Streamlit
93. SWR
94. TrustKit

ประเมิน

95. .NET 5
96. bUnit
97. Dagster
98. Flutter สำหรับเว็บ
99. Jotai และ Zustand
100. Kotlin Multiplatform Mobile
101. LVGL
102. React Hook Form
103. River
104. Webpack 5 Module Federation

เฝ้าระวัง

TECHNOLOGY RADAR

เทคนิค

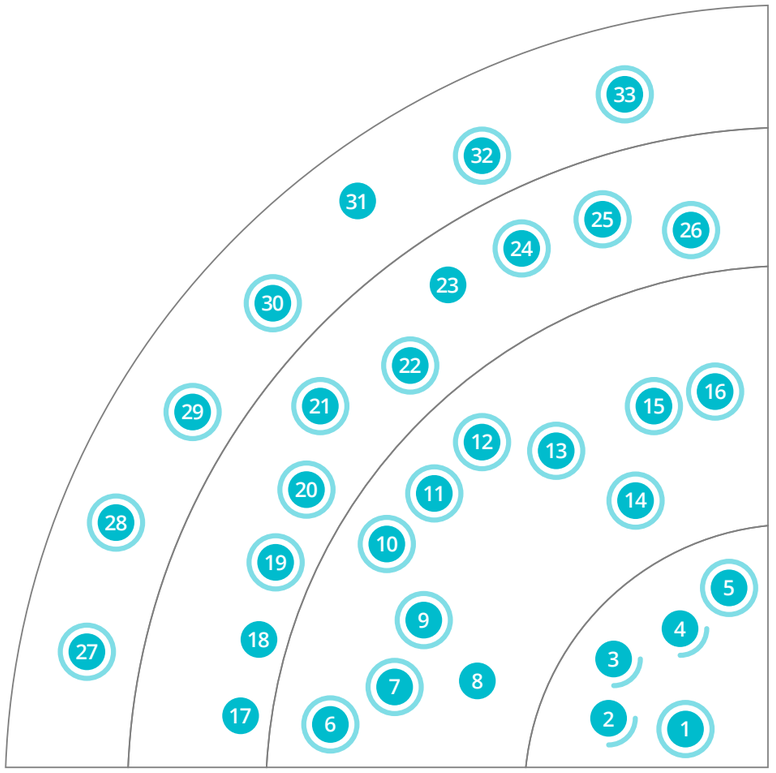
เทคนิค

การขยายสัญญาของ API
นำไปใช้

รูปแบบการขยายสัญญาของ API (API expand-contract) ซึ่งบางครั้งเรียกว่าการเปลี่ยนแปลงแบบขนาน น่าจะเป็นสิ่งที่หลาย ๆ คนคุ้นเคยโดยเฉพาะในบริบทฐานข้อมูลหรือโค้ด อย่างไรก็ตามเราเห็นการนำไปใช้กับ API ค่อนข้างน้อย ในบางกรณี เราเห็นปัญหานี้ถูกแก้โดยการกำหนดเวอร์ชันของ API ที่ซับซ้อน หรือการแก้ไข API โดยไม่เข้ากันกับ API เก่า (breaking change) ที่ความจริงแล้วสามารถแก้ไขด้วยการขยายสัญญาก็พอเพียงแล้ว กล่าวคือขั้นตอนแรกของการขยายสัญญา ให้เริ่มจากการเติมส่วนที่ต้องการลงใน API ในขณะเดียวกันก็ยังรักษาส่วนที่จะเลิกใช้ไว้อยู่ และจะลบมันออกได้ก็ต่อเมื่อผู้ใช้ API ทุกคนสลับไปใช้สคีมาใหม่กันครบแล้ว วิธีนี้จำเป็นต้องมีการประสานงานระหว่างผู้ใช้และผู้สร้าง API ระดับหนึ่ง โดยอาจทำผ่านเทคนิคเช่น การทดสอบคอนแทรคท์ที่ขับเคลื่อนด้วยผู้บริโภค (consumer-driven contract)

การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML)
นำไปใช้

เราเห็นเทคนิค การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML) เป็นจุดเริ่มต้นที่ดีสำหรับโซลูชันแมชชีนเลิร์นนิงใด ๆ ที่กำลังปรับเพื่อการใช้งานจริง หลายองค์กรต่างไว้วางใจโซลูชันแมชชีนเลิร์นนิงมากขึ้น ไม่ว่าจะเป็นการเสนอสินค้าและบริการตามรายบุคคล หรือการนำมาใช้ดำเนินงานต่าง ๆ ภายในองค์กร ดังนั้นจึงมีความเหมาะสมทางธุรกิจที่จะใช้บทเรียนและแนวทางปฏิบัติที่ดีของการส่งมอบอย่างต่อเนื่อง (CD) มาใช้งานกับโซลูชันแมชชีนเลิร์นนิง



Design systems
นำไปใช้

เมื่อการพัฒนาแอปพลิเคชันมีการเปลี่ยนแปลงและความซับซ้อนเพิ่มขึ้นอยู่เสมอ การส่งมอบผลิตภัณฑ์ที่เข้าถึงได้สามารถใช้งานได้ และหน้าตาคงที่ จึงเป็นความท้าทายโดยเฉพาะอย่างยิ่งในองค์กรขนาดใหญ่ที่มีหลายทีมทำงานกับผลิตภัณฑ์ที่แตกต่างกัน Design Systems เป็นการกำหนดชุดของรูปแบบการออกแบบ (design patterns) ไลบรารี่คอมโพเนนท์ (component libraries) การออกแบบและหลักปฏิบัติทางวิศวกรรมที่ดี ทำให้ได้ผลิตภัณฑ์ดิจิทัลที่คงเส้นคงวา Design Systems สร้างขึ้นจากแนวทางการออกแบบในอดีต นำเสนอไลบรารี่และเอกสารที่ใช้ร่วมกันซึ่งง่ายต่อการค้นหาและใช้งาน โดยทั่วไปคำแนะนำจะถูกเขียนเป็นโค้ดและอยู่ภายใต้การควบคุมเวอร์ชัน เพื่อให้คำแนะนำมีความคลุมเครือน้อยที่สุดและบำรุงรักษาง่ายกว่าเอกสารธรรมดา Design Systems ได้กลายเป็นวิธีมาตรฐานเมื่อทำงานข้ามทีมและกลายเป็นข้อบังคับในการพัฒนาผลิตภัณฑ์ เพราะพวกมันจะช่วยให้ทีมมีสมาธิและช่วยให้ทีมสามารถมุ่งความสนใจไปที่ความท้าทายเชิงกลยุทธ์ของตัวผลิตภัณฑ์ โดยไม่ต้องเริ่มต้นสร้างจากศูนย์ใหม่ทุกครั้งที่มีสร้างคอมโพเนนท์การแสดงผล.

นำไปใช้
1. การขยายสัญญาของ API
2. การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML)
3. Design systems
4. ทีมพัฒนาแพลตฟอร์มดั่งทีมพัฒนาผลิตภัณฑ์
5. การสับเปลี่ยนบัญชีผู้ใช้บริการ

ทดลอง
6. คลาวด์แซนด์บ็อกซ์
7. Contextual bandits
8. Docker อิมเมจที่ไม่มีลีนุกซ์ดิสโตร
9. Ethical Explorer
10. การใช้สมมุติฐานเป็นตัวขับเคลื่อนการปรับปรุงระบบ
11. การร้องขอข้อคิดเห็นด้วยวิธีที่เรียบง่าย
12. แมชชีนเลิร์นนิงที่เรียบง่ายที่สุดเท่าที่จะเป็นไปได้
13. SPA injection
14. ภาระการรับรู้ของทีม
15. จัดการไฟล์ Xcodeproj ด้วยเครื่องมือ
16. การใช้ type ร่วมกันระหว่าง UI/BFF

ประเมิน
17. แพลตฟอร์มเขียนโค้ดน้อยเฉพาะทาง
18. เอกลักษณ์กระจายศูนย์
19. Deployment drift radiator
20. การเข้ารหัสแบบโฮโมมอร์ฟิก
21. Hotwire
22. อิมพอร์ตแมพ สำหรับ ไมโครฟรอนท์เอนด์
23. Open Application Model (OAM)
24. Privacy-focused web analytics
25. การเขียนโปรแกรมแบบม็อบระยะไกล
26. Secure multiparty computing

เฝ้าระวัง
27. GitOps
28. ทีมแพลตฟอร์มที่ถูกแบ่งตามเลเยอร์ของเทคโนโลยี
29. นโยบายความซับซ้อนของรหัสผ่านที่คิดไปเองว่าดี
30. พูลรีเควสเป็นวิธีการเดียวในการตรวจทานโค้ดของเพื่อนร่วมงาน
31. SAFe™
32. แบ่งแยกทีมในการดูแลโค้ดกับไปป์ไลน์
33. แพลตฟอร์มที่ใช้การร้องขอในการดำเนินงาน

เทคนิค

คำว่า bandits มีที่มาจากฉายาของสล็อตแมชชีนในคาสิโนนั่นเอง โดยหลักการแล้วอัลกอริธึมนี้จะสำรวจทางเลือกใหม่ ๆ เพื่อเรียนรู้เพิ่มเติมเกี่ยวกับผลลัพธ์ที่ได้ ขณะเดียวก็จะรักษาสมดุลโดยเลือกใช้ตัวเลือกที่ทราบว่าให้ผลลัพธ์ที่ดีควบคู่กันไประหว่างทาง

(Contextual bandits)

ทีมสร้างแพลตฟอร์มในรูปแบบผลิตภัณฑ์

นำไปใช้

ตามที่ได้กล่าวถึงในธีมของฉบับนี้ แวดวงการพัฒนาซอฟต์แวร์ได้มีประสบการณ์มากขึ้นต่อ ทีมสร้างแพลตฟอร์มในรูปแบบผลิตภัณฑ์ ที่สร้างและดูแลแพลตฟอร์มที่ใช้ภายใน โดยแพลตฟอร์มเหล่านี้ถูกใช้โดยทีมทั่วทั้งองค์กร และช่วยให้ทีมพัฒนาแอปพลิเคชันได้อย่างรวดเร็ว อีกทั้งยังลดความซับซ้อนของการดำเนินการและลดเวลาการออกสู่ตลาด จากการที่เทคนิคนี้ถูกนำไปใช้มากขึ้น ก็ทำให้เราเห็นถึงวิธีการที่ดีและไม่ดีของแนวทางนี้อย่างชัดเจนขึ้นเช่นกัน ในการสร้างแพลตฟอร์มนั้นจำเป็นอย่างยิ่งที่จะต้องระบุตัวลูกค้า และผลิตภัณฑ์ที่จะได้ประโยชน์จากมันอย่างชัดเจนแทนที่จะสร้างมันขึ้นมาจากความว่างเปล่า และเรายากให้ระมัดระวังการทำ ทีมสร้างแพลตฟอร์มแบบแบ่งชั้น ที่คงการแบ่งกลุ่มตามเทคโนโลยีที่ใช้อยู่เดิม แต่แค่เปลี่ยนชื่อทีมเป็น “ทีมแพลตฟอร์ม” และเรายังต่อต้านรูปแบบแพลตฟอร์มที่ดำเนินงานผ่านระบบตั๋ว (ticket-driven) ด้วยเช่นกัน

เรายังคงเป็นแฟนตัวยงของการใช้หลักการจาก Team Topologies ซึ่งเราคิดถึงวิธีการจัดรูปแบบของทีมแพลตฟอร์มที่ดีที่สุด เรามองว่าทีมสร้างแพลตฟอร์มในรูปแบบผลิตภัณฑ์ จะเป็นแนวทางมาตรฐาน และเป็นสิ่งสำคัญที่ทำให้ระบบไอทีมีสมรรถภาพสูงขึ้นได้

การสับเปลี่ยนบัญชีผู้ใช้บริการ

นำไปใช้

เราแนะนำอย่างยิ่งให้แต่ละองค์กรตรวจสอบให้แน่ใจว่าได้สลับรหัสรับรองตัวตนผู้ใช้อยู่เสมอ เมื่อไปใช้งานบริการบัญชีผู้ใช้ของผู้ให้บริการคลาวด์ (cloud service account) เจ้าต่าง ๆ การสับเปลี่ยนเป็นหนึ่งใน การรักษาความปลอดภัย 3R เพราะมันง่ายมากที่องค์กรจะลืมว่าเคยมีบัญชีเหล่านั้นอยู่ ซึ่งกว่าจะรู้ตัวก็มีเหตุการณ์ที่ไม่พึงประสงค์เกิดขึ้นไปแล้ว สิ่งนี้นำไปสู่ปัญหาอื่น ๆ อีก เช่น การมีบางบัญชีที่กำหนดสิทธิ์ผู้ใช้ไว้กว้างเกินจำเป็น หรือเปิดไว้เป็นระยะเวลานานโดยไม่ได้สับเปลี่ยนเลย นอกจากนั้นแล้ว การประยุกต์ใช้หลักการสับเปลี่ยนบัญชีผู้ใช้บริการอย่างสม่ำเสมอ เป็นจุดเริ่มต้นสำคัญที่นำไปสู่หลักการกำหนดสิทธิ์ให้น้อยที่สุดเท่าที่จำเป็น (least privilege principle) อีกด้วย

คลาวด์แซนด์บ็อกซ์

ทดลอง

จากที่การใช้งานคลาวด์เป็นเรื่องสามัญมากขึ้น และการสร้าง คลาวด์แซนด์บ็อกซ์ ขึ้นมาใช้ เป็นเรื่องที่ทำได้ง่ายขึ้น แม้จะใช้ในงานขนาดใหญ่ก็ตาม ในการพัฒนางานทีมของเราจึงชอบการใช้สภาพแวดล้อมที่ไปใช้คลาวด์ล้วน (ซึ่งตรงข้ามกับการพัฒนาบนเครื่องนักพัฒนา) เพราะมันช่วยลดความซับซ้อนในการบำรุงรักษา เราเห็นว่าเครื่องมือที่พยายามจำลองบริการคลาวด์เนทีฟขึ้นมาบนเครื่องนักพัฒนาเองนั้นมีข้อจำกัดในความน่าเชื่อถือ เมื่อนักพัฒนาต้องการบิลด์และทดสอบซอฟต์แวร์ ดังนั้นเราไปตั้งใจทำให้คลาวด์แซนด์บ็อกซ์เป็นมาตรฐาน แทนการจำลองคลาวด์เนทีฟบนเครื่องนักพัฒนาดีกว่า ซึ่งวิธีการนี้จะผลักดันให้เกิดการกำหนดโครงสร้างพื้นฐานด้วยโค้ด อีกทั้งทำให้เกิดกระบวนการออนบอร์ดที่ดีสำหรับการจัดเตรียมสภาพแวดล้อมแซนด์บ็อกซ์ให้นักพัฒนา อย่างไรก็ดี การเปลี่ยนมาใช้คลาวด์แซนด์บ็อกซ์มีความเสี่ยงหลายประการเนื่องจากนักพัฒนาจะต้องพึ่งพาความพร้อมใช้งานของสภาพแวดล้อมระบบคลาวด์อย่างแท้จริง และอาจทำงานช้าลงเพราะต้องรอผลลัพธ์เมื่อไม่สามารถติดต่อกับคลาวด์ได้ เราจึงอยากแนะนำให้ใช้การกำกับดูแลที่ไม่ระเบียบพิธีเยอะมาใช้ เพื่อตกลงสร้างมาตรฐานกลางให้กับคลาวด์แซนด์บ็อกซ์ขององค์กร โดยเฉพาะอย่างยิ่งในด้านที่เกี่ยวกับความปลอดภัย IAM หรือการติดตั้งในระดับภูมิภาค (regional deployments)

Contextual bandits

ทดลอง

Contextual bandits เป็นการเรียนรู้แบบเสริมกำลัง (reinforcement learning) ประเภทหนึ่งที่เหมาะสมสำหรับโจทย์ที่ต้องการหาความสมดุลระหว่างการสำรวจหาโอกาสใหม่ ๆ หรือใช้ประโยชน์จากสิ่งที่มีอย่างเต็มที่ ซึ่งคำว่า bandits มีที่มาจากฉายาของสล็อตแมชชีนในคาสิโนนั่นเอง โดยหลักการแล้วอัลกอริธึมนี้จะสำรวจทางเลือกใหม่ ๆ เพื่อเรียนรู้เพิ่มเติมเกี่ยวกับผลลัพธ์ที่ได้ ขณะเดียวก็จะรักษาสมดุลโดยเลือกใช้ตัวเลือกที่ทราบว่าให้ผลลัพธ์ที่ดีควบคู่กันไประหว่างทาง เราได้ใช้เทคนิคนี้ในสถานการณ์ที่เรามีข้อมูลเพียงเล็กน้อยในการฝึกและการติดตั้งกับโมเดลแมชชีนเลิร์นนิงอื่น ๆ จากการที่เราสามารถเพิ่มบริบทต่าง ๆ ให้กับโจทย์การรักษาสมดุลเช่นนี้ ทำให้สามารถประยุกต์ใช้อัลกอริธึมนี้กับงานหลากหลายรูปแบบด้วยกัน ซึ่งรวมทั้งงานทดสอบทางเลือก A/B งานให้คำแนะนำ หรืองานเพิ่มประสิทธิภาพของเลย์เอาต์ก็ดี

อิมเมจที่ไม่มีลีนุกซ์ดิสโตร

ทดลอง

การที่จะสร้าง Docker อิมเมจเพื่อครอบแอปพลิเคชันของเรา จะมีอยู่สองเรื่องที่ต้องให้ความสำคัญคือ เรื่องความมั่นคง และเรื่องขนาดของอิมเมจ โดยปกติ เราจะใช้เครื่องมือพิเคราะห์คอนเทเนอร์เพื่อความมั่นคง (container security scanning) เพื่อตรวจจับและอุดช่องโหว่ต่าง ๆ ที่ถูกเปิดเผย และใช้ลีนุกซ์ดิสโตรขนาดเล็กเป็นพื้นฐาน เช่น Alpine Linux เพื่อจัดการเรื่องขนาดและทรัพยากร แต่ด้วยภัยคุกคามด้านความมั่นคงที่เพิ่มขึ้น ทำให้การกำจัดช่องทางการโจมตีที่เป็นไปได้มีความสำคัญมากยิ่งขึ้นกว่าเดิม นี่เป็นสาเหตุว่าทำไมการใช้งาน Docker อิมเมจที่ไม่มีลีนุกซ์ดิสโตรได้กลายเป็นทางเลือกหลักของการพัฒนาแอปบนคอนเทเนอร์ การนำลีนุกซ์ดิสโตรออกไปทำให้ลดฟุตพริ้นท์และสิ่งที่พึ่งพา (dependency) ที่เกิดขึ้นจากลีนุกซ์ดิสโตร เทคนิคนี้ยังช่วยลดข้อผิดพลาดของการตรวจพิเคราะห์ ลดพื้นที่การโจมตีให้แคบลง แถมยังมีช่องโหว่ให้อุดน้อยลงด้วย ส่วนขนาดที่เล็กลงก็ทำให้ประหยัดทรัพยากรได้มากขึ้น เป็นต้น Google ได้เตรียม คอนเทเนอร์อิมเมจที่ไม่มีลีนุกซ์ดิสโตร ของแต่ละภาษาไว้ให้ใช้ หรือคุณจะสร้างเอง ผ่านเครื่องมืออย่าง Bazel ของ Google หรือจะใช้การสร้างมัลติสเตจใน Docker ก็ได้ หมายเหตุไว้ว่า ถ้าใช้อิมเมจที่ไม่มีลีนุกซ์ดิสโตรจะไม่มีเชลล์ติดมาด้วย ทำให้ดีบั๊กได้ยาก อย่างไรก็ดี มันแก้ได้ไม่ยาก โดยการหาอิมเมจเวอร์ชันที่ดีบั๊กได้ในโลกออนไลน์ เช่น BusyBox shell ทั้งนี้ การใช้อิมเมจที่ไม่มีลีนุกซ์ดิสโตรติดมาด้วย เป็นเทคนิคที่ Google เป็นผู้บุกเบิกมาตั้งแต่ต้น และจากประสบการณ์ของเราพบว่าอิมเมจส่วนมากยังคงมี

ที่มาจาก Google อยู่ หากมีผู้ให้บริการหลายรายให้เลือกมากขึ้น เราคงจะสะดวกสบายมากกว่านี้ นอกจากนี้ โปรดระมัดระวังเมื่อใช้ Trivy หรือเครื่องมือพิเคราะห์ช่องโหว่ที่คล้ายกัน เนื่องจากคอนเทเนอร์อิมเมจที่ไม่มีลีนุกซ์ดิสโตรนั้นได้รับการสนับสนุนในรุ่นล่าสุดเท่านั้น

Ethical Explorer

ทดลอง

กลุ่มที่อยู่เบื้องหลัง Ethical OS คือ Omidyar Network ซึ่งเป็นบริษัทร่วมทุนเพื่อการเปลี่ยนแปลงทางสังคมที่ถูกสร้างโดย Pierre Omidyar ผู้ก่อตั้ง eBay กลุ่มนี้ได้เผยแพร่การตีพิมพ์ครั้งใหม่ที่เรียกว่า Ethical Explorer โดยดึงบทเรียนที่เรียนรู้จากการใช้เฟรมเวิร์คจริยธรรมชุดเก่าอย่าง Ethical OS และเพิ่มคำถามสำหรับทีมผลิตภัณฑ์เข้าไปอีกด้วย โดยมันสามารถดาวน์โหลดได้ฟรี และสามารถนำมาทำเป็นการ์ดเพื่อกระตุ้นให้เกิดการสนทนาที่เกี่ยวข้องกับจริยธรรม ชุดเครื่องมือนี้มีคำถามปลายเปิดสำหรับการแจ้งเตือน “ความเสี่ยง” ทางเทคนิคหลายประการ ประกอบไปด้วย การเฝ้าระวัง (มีใครสามารถใช้ผลิตภัณฑ์หรือบริการของเราเพื่อติดตามหรือระบุผู้ใช้รายอื่นได้หรือไม่) การให้ข้อมูลในทางที่ผิด การกีดกัน ความไม่เป็นกลางของอัลกอริทึม การเสพติด การควบคุมข้อมูล ผู้ร้าย และการให้อำนาจมากเกินไป เป็นต้น อีกทั้งยังมีคู่มือภาคสนามที่มีกิจกรรมและการประชุมเชิงปฏิบัติการ แนวคิดสำหรับการเริ่มต้นการสนทนา และเคล็ดลับสำหรับการนำเข้าไปใช้งานในองค์กร ในขณะที่เรายังมีหนทางอีกยาวไกลในอุตสาหกรรม เพื่อเป็นตัวแทนทางจริยธรรมของสังคมดิจิทัลของเราให้ดีขึ้น เราได้มีการสนทนาที่มีประสิทธิผลโดยใช้ Ethical Explorer ไปบ้างแล้ว และเราได้รับการสนับสนุนจากการรับรู้ถึงความสำคัญของการตัดสินใจผลิตภัณฑ์ในการแก้ไขปัญหาสังคม

การใช้สมมุติฐานเป็นตัวขับเคลื่อนการปรับปรุงระบบ

ทดลอง

เรามักถูกขอให้ปรับปรุงแก้ไขระบบเดิมที่เราไม่ได้สร้างไว้ บางครั้งเราพบปัญหาทางเทคนิคที่ต้องใช้ความใส่ใจ เช่น การปรับปรุงประสิทธิภาพหรือความน่าเชื่อถือ วิธีทั่วไปวิธีหนึ่งในการแก้ไขปัญหาเหล่านี้คือการสร้าง “เรื่องราวทางเทคนิค” (technical stories) โดยใช้รูปแบบเดียวกับ “เรื่องราวของผู้ใช้” (user story) โดยมองที่ผลลัพธ์ทางเทคนิคแทนที่จะเป็นทางธุรกิจ แต่งานด้านเทคนิคเหล่านี้มักจะยากที่จะประเมินและใช้เวลานานกว่าที่คาดไว้ หรือไม่ได้ผลลัพธ์ที่ต้องการ อีกวิธีหนึ่งที่ประสบความสำเร็จมากขึ้นคือ การใช้สมมุติฐานเป็นตัวขับเคลื่อนการปรับปรุงระบบ แทนการลงมือพัฒนาเรื่องราวของผู้ใช้ที่รออยู่ ทีมสามารถเข้ามาร่วมกันสร้างสมมุติฐานให้กับปัญหาเพื่อกำหนดผลลัพธ์เชิงเทคนิคที่วัดผลได้ จากนั้นพวกเขาแบ่งการทำงานออกเป็นรอบย่อยๆ และกำหนดกรอบเวลาในการทดลอง เพื่อตรวจสอบหรือหักล้างสมมติฐานแต่ละสมมติฐานตามลำดับความสำคัญ ผลลัพธ์ที่ได้จากเวิร์กโฟลว์นี้จะเป็นการปรับปรุงเพื่อลดความไม่แน่นอนแทนที่จะทำตามแผน ส่งผลให้สามารถคาดการณ์ผลลัพธ์ได้มากขึ้น

การร้องขอข้อคิดเห็นด้วยวิธีที่เรียบง่าย

ทดลอง

ในขณะที่หลากหลายองค์กรต่างขับเคลื่อนไปสู่ evolutionary architecture การเก็บรวบรวมข้อมูลการตัดสินใจเกี่ยวกับการออกแบบ สถาปัตยกรรม เทคนิคและวิธีการทำงานของทีมจึงเป็นสิ่งสำคัญ กระบวนการรวบรวมและสรุปผลข้อเสนอแนะที่จะนำไปสู่การตัดสินใจเหล่านี้เริ่มต้นด้วยการร้องขอข้อคิดเห็น (Request for Comments - RfCs) RfCs เป็นเทคนิคในการรวบรวมบริบทการออกแบบและแนวคิดทางสถาปัตยกรรม โดยทำงานร่วมกันกับทีมเพื่อเป้าหมายคือการตัดสินใจบนบริบทและข้อมูลที่เกี่ยวข้อง เราขอแนะนำให้องค์กรต่าง ๆ ใช้ การร้องขอข้อคิดเห็นด้วยวิธีที่เรียบง่าย โดยใช้เทมเพลตมาตรฐานที่ไม่สลับซับซ้อนร่วมกันหลาย ๆ ทีมรวมถึงการควบคุมเวอร์ชันของ RfCs
การเก็บบันทึกข้อมูลในการตรวจสอบการตัดสินใจเหล่านี้เป็นสิ่งที่สำคัญอย่างมาก เพราะจะเป็นประโยชน์ต่อสมาชิกของทีมในอนาคตและเป็นการเก็บข้อมูลวิวัฒนาการทางเทคนิคและธุรกิจขององค์กร องค์กรขนาดใหญ่จำนวนมากได้นำ RfCs มาใช้กับทีมที่มีการทำงานอย่างอิสระด้วยตนเองเพื่อผลักดันการสื่อสารและการทำงานร่วมกันที่ดีขึ้นโดยเฉพาะในเรื่องที่เกี่ยวข้องกับการตัดสินใจร่วมกันของหลาย ๆ ทีม

แมชชีนเลิร์นนิงที่เรียบง่ายที่สุดเท่าที่จะเป็นไปได้

ทดลอง

ผู้ให้บริการคลาวด์รายใหญ่ทั้งหลายนำเสนอโซลูชันแมชชีนเลิร์นนิง (ML) อันน่าตื่นตาตื่นใจ เครื่องมือที่ทรงพลังเหล่านี้สามารถใช้ประโยชน์ได้มากมาย แต่ทุกอย่างล้วนมีข้อแลกเปลี่ยน โดยผู้ให้บริการคลาวด์จะเรียกเก็บค่าใช้จ่ายสำหรับบริการเหล่านี้ นอกจากนี้ยังมีค่าใช้จ่ายในการดำเนินงานอีกด้วย เครื่องมือที่ซับซ้อนเหล่านี้จำเป็นต้องทำความเข้าใจและต้องการความชำนาญในการใช้งาน ในทุกเครื่องมือใหม่ที่เพิ่มลงในสถาปัตยกรรมจะเพิ่มภาระค่าดำเนินการเหล่านี้ จากประสบการณ์ของเรา ทีมงานมักจะเลือกเครื่องมือที่ซับซ้อนเกินความต้องการ เพราะพวกเขาประเมินพลังของเครื่องมือที่เรียบง่ายต่ำเกินไป เช่น การถดถอยเชิงเส้น ปัญหาแมชชีนเลิร์นนิงจำนวนมากไม่จำเป็นต้องใช้งาน GPU หรือเครือข่ายเส้นประสาท ด้วยเหตุนี้เราจึงสนับสนุนการใช้งานแมชชีนเลิร์นนิงที่เรียบง่ายที่สุดเท่าที่จะเป็นไปได้ โดยใช้เครื่องมือและโมเดลที่เรียบง่ายเช่นโค้ด Python ไม่กี่ร้อยบรรทัดบนแพลตฟอร์มการประมวลผลที่คุณมีอยู่ จงเข้าถึงเครื่องมือที่ซับซ้อนเฉพาะเมื่อคุณต้องการความสามารถของพวกมันเท่านั้น

เทคนิค

เมื่อเราต้องการปรับปรุงระบบดั้งเดิมที่เป็นซิงเกิลเพจแอปพลิเคชัน (SPA) แทนที่จะห่อระบบเดิมเราจะฝังจุดเริ่มต้นของ SPA ใหม่ลงในเอกสาร HTML หน้าเก่าและค่อย ๆ แก้ไขมันจนทำงานแทนของเก่าในที่สุด

(SPA Injection)

เทคนิค

ปฏิสัมพันธ์ระหว่างทีมนั้นเป็นหนึ่งใน ปัจจัยสำคัญที่กำหนดว่าทีมจะสามารถส่งมอบคุณค่าให้กับลูกค้าได้รวดเร็วแค่ไหน ผู้เขียนหนังสือ Team Topology ได้พัฒนาการประเมินสำหรับการวัดปฏิสัมพันธ์นี้ซึ่งเราเรียกว่าภาระการรับรู้ของทีม

(ภาระการรับรู้ของทีม)

SPA injection

ทดลอง

การใช้ท่ากาฝากล้อมต้น หรือ Strangler Fig Pattern มักจะเป็นทางเลือกแรก ๆ ที่นิยมนำมาใช้กัน เมื่อต้องการจะเปลี่ยนถ่ายจากระบบเก่าไปสู่ระบบใหม่ โดยที่โค้ดชุดใหม่จะถูกสร้างล้อมรอบโค้ดชุดเก่าไว้ จนโค้ดชุดใหม่จะค่อย ๆ แทนที่ความสามารถที่ต้องการแก้ไขจนครบทั้งหมด ซึ่งเป็นกลยุทธการจัดการแบบ “ภายนอกสู่ภายใน” ที่ใช้งานได้ดีกับงานปรับปรุงระบบเก่าโดยส่วนใหญ่ ปัจจุบันเรามีประสบการณ์กับการพัฒนาซิงเกิลเพจแอปพลิเคชัน (SPA) มาพอสมควร ซึ่งมันมากพอจนทำให้แอปพลิเคชัน SPA บางตัวจำเป็นต้องถูกปรับปรุงแล้วด้วยซ้ำ เราจึงได้เห็นวิธีการที่ตรงกันข้ามในการเปลี่ยนแปลง นั่นคือการทำแบบ “ภายในสู่ภายนอก” ซึ่งวิธีนี้ แทนที่จะห่อระบบเดิม เราจะฝังจุดเริ่มต้นของ SPA ใหม่ลงในเอกสาร HTML หน้าเก่าและค่อย ๆ แก้ไขมันจนทำงานแทนของเก่าในที่สุด วิธีนี้ไม่จำเป็นต้องเลือกเฟรมเวิร์คเดียวกันก็ได้ หากผู้ใช้งานรับได้กับความเร็วในการแสดงผลที่ช้าลง เช่น เราสามารถฝังแอปพิเคชัน React ลงไปในระบบเก่าที่เป็น AngularJS วิธีการ SPA injection ช่วยให้คุณสามารถค่อย ๆ ถอดโค้ด SPA เก่าออกไปที่ละนิด จนกระทั่ง SPA ใหม่เข้ามาแทนที่อย่างสมบูรณ์ ขณะที่ท่ากาฝากล้อมต้น จะเปรียบได้กับกาฝากที่อาศัยพื้นผิวภายนอกที่มั่นคงของต้นไม้ที่อาศัยเพื่อสนับสนุนตัวเองจนกระทั่งมันสามารถหยั่งรากลึกและปล่อยให้ต้นไม้ที่อาศัยตายไปในที่สุด แต่วิธีการนี้เปรียบเหมือนการฉีดสิ่งแปลกใหม่เข้าสู่ร่างผู้อาศัยแต่แรก โดยยังจะพึ่งพิงความสามารถของร่างผู้อาศัยจนกว่าของใหม่จะเข้าครอบงำร่างทั้งหมดได้อย่างสมบูรณ์

ภาระการเรียนรู้ภายในทีม

ทดลอง

สถาปัตยกรรมของระบบนั้นมักจะล้อกับโครงสร้างขององค์กรเสมอ ฉะนั้นการวางรูปแบบการปฏิสัมพันธ์ระหว่างทีมให้เป็นไปในแบบที่ต้องการ จึงไม่ใช่เรื่องแปลกแต่อย่างใด ซึ่งสามารถอ่านเพิ่มเติมได้จากตัวอย่างของ The Inverse Conway Maneuver เราพบว่าปฏิสัมพันธ์ระหว่างทีมนั้นเป็นหนึ่งในปัจจัยสำคัญที่กำหนดว่าทีมจะสามารถส่งมอบคุณค่าให้กับลูกค้าได้รวดเร็วแค่ไหน หรือง่ายดายเพียงใด ซึ่งในที่สุดเราก็ได้พบวิธีวัดระดับปฏิสัมพันธ์นี้แล้ว โดยใช้แบบประเมินผลของผู้เขียนหนังสือ Team Topologies ในการประเมินว่าทีมสามารถสร้าง ทดสอบ และดูแลรักษาระบบของตนได้ยากง่ายเพียงใด เมื่อวัดระดับ ภาระการเรียนรู้ภายในทีม แล้วเราจะสามารถแนะนำวิธีการปรับเปลี่ยนโครงสร้างทีมและพัฒนาปฏิสัมพันธ์ระหว่างทีมของลูกค้าได้ต่อไป

จัดการไฟล์ Xcodeproj ด้วยเครื่องมือ

ทดลอง

นักพัฒนาแอปพลิเคชัน iOS ของเราหลายคนมักจะต้องปวดหัวทุกครั้งที่ต้องใช้ Xcode เนื่องจากไฟล์ Xcodeproj จะเปลี่ยนแปลงทุกครั้งเวลาที่มีอะไรบางอย่างในโปรเจกต์เปลี่ยนแปลง ด้วยความที่ไฟล์ Xcodeproj ไม่ได้ออกแบบมาให้คนอ่านได้ง่าย จึงทำให้การพยายามแก้ไขความทับซ้อน (merge conflicts) เป็นเรื่องที่ค่อนข้างยุ่งยาก อาจส่งผลให้ประสิทธิภาพในการทำงานลดลง และเสี่ยงที่จะทำให้เกิดความผิดพลาดในระดับโปรเจกต์ นั่นคือหากมีความผิดพลาดอะไรบางอย่างกับไฟล์นี้อาจส่งผลให้ Xcode ทำงานได้ไม่ถูกต้องและมีความเป็นไปได้สูงที่การทำงานของนักพัฒนาทุกคนในทีมจะถูกบล็อค ดังนั้นแทนที่จะพยายามแก้ไขความทับซ้อนของไฟล์นี้ด้วยตัวเองหรือพยายามจัดการเวอร์ชันของมัน เราขอแนะนำให้คุณใช้วิธีการจัดการไฟล์ Xcodeproj ด้วยเครื่องมือ ทำได้โดยระบุการตั้งค่าโปรเจกต์ Xcode ของคุณด้วยภาษา YAML (XcodeGen, Struct) Ruby (Xcake) หรือ Swift (Tuist) แล้วเครื่องมือเหล่านี้จะช่วยสร้างไฟล์ Xcodeproj ให้โดยอัตโนมัติโดยดูจากไฟล์สำหรับตั้งค่าและโครงสร้างของโปรเจกต์ของคุณ ด้วยวิธีนี้ ปัญหาการทับซ้อนของไฟล์ Xcodeproj จะกลายเป็นแค่เรื่องในอดีต และต่อไปนี้หากเกิดมีปัญหาการทับซ้อนขึ้น มันก็จะเกิดขึ้นในไฟล์สำหรับตั้งค่าซึ่งง่ายต่อการอ่านและจัดการมากกว่า

การใช้ type ร่วมกันระหว่าง UI/BFF

ทดลอง

จากการที่ TypeScript กลายมาเป็นภาษาสามัญสำหรับการพัฒนาฟรอนต์เอนด์ และ Node.js กลายเป็นที่นิยมสำหรับเทคโนโลยี BFF ทำให้เราได้เห็น การใช้ type ร่วมกันระหว่าง UI/BFF เพิ่มมากขึ้น เทคนิคนี้จะมีชุดของนิยามของประเภทข้อมูลที่ใช้ร่วมกันทั้งอ็อบเจกต์ข้อมูลที่ได้จากคำขอของฟรอนท์เอนด์ และข้อมูลที่ตอบกลับจากเซิร์ฟเวอร์แบ็คเอนด์ ปกติแล้วเราจะให้ระวังแนวปฏิบัติแบบนี้ เพราะมันจะสร้างพัทธะผูกพันระหว่างฟรอนท์เอนด์และแบ็คเอนด์โดยไม่จำเป็น อย่างไรก็ตาม หลายทีมพบว่าข้อดีของแนวทางนี้มีน้ำหนักมากกว่าความเสี่ยงของการเกิดพันธะผูกพัน เนื่องด้วยรูปแบบของ BFF จะทำงานได้ดีเมื่อทั้งโค้ด UI และ BFF มีเจ้าของเป็นทีมเดียวกัน และบ่อยครั้งจะเก็บทั้งสองส่วนไว้ในที่จัดเก็บซอร์สโค้ดเดียวกัน ทำให้สามารถมองคู่ของ UI และ BFF นั้นเป็นระบบเดียวกันได้ การที่ BFF สนับสนุนการเรียกดูข้อมูลแบบรู้ไทป์ที่แน่นอนทำให้เราสามารถปรับเปลี่ยนผลลัพธ์ให้ตรงกับความต้องการที่เฉพาะเจาะจงกับฟรอนต์เอนด์ แทนที่จะใช้ข้อมูลสำหรับวัตถุประสงค์ทั่วไป ที่จำเป็นต้องรองรับความต้องการของผู้ใช้งานหลากหลายและมีฟิลด์ข้อมูลมากกว่าที่ต้องการใช้จริง ซึ่งจะเป็นการลดความเสี่ยงในการเปิดเผยข้อมูลที่ผู้ใช้งานไม่ควรเห็นอย่างไม่ตั้งใจ และเป็นการป้องกันการแปลความหมายของข้อมูลที่ส่งกลับไปอย่างไม่ถูกต้อง อีกทั้งยังทำให้การเรียกดูข้อมูลมีความชัดเจนมากยิ่งขึ้น แนวปฏิบัตินี้มีประโยชน์โดยเฉพาะเมื่อใช้ร่วมกับ io-ts ในการบังคับใช้ไทป์อย่างปลอดภัยในช่วงเวลาที่โปรแกรมทำงาน

แพลตฟอร์มเขียนโค้ดน้อยเฉพาะทาง

ประเมิน

หลาย ๆ องค์กรขณะนี้ต้องเผชิญหน้ากับการตัดสินใจที่ดูเหมือนเล็ก แต่มีผลอย่างมากต่ออนาคต นั่นคือการตัดสินใจว่าจะใช้แพลตฟอร์มจำพวกที่ไม่ต้องเขียนโค้ด (no-code platform) หรือเขียนโค้ดน้อย (low-code platform) ซึ่งแพลตฟอร์มประเภทนี้จะออกแบบมาเพื่อแก้ปัญหาเฉพาะทางในโดเมนที่จำกัดมาก ปัจจุบันมีผู้ให้บริการจำนวนมากพยายามผลักดันผู้ใช้ไปในทิศทางนี้อย่างหนัก ความกังวลที่เรามีต่อแพลตฟอร์มเหล่านี้ คือ ความยากในการประยุกต์ใช้หลักปฏิบัติทางวิศวกรรมที่ดี อย่างเช่น การทำเวอร์ชันหรือการทดสอบ แต่อย่างไรก็ตาม เราสังเกตเห็นผู้เล่นหน้าใหม่ที่น่าสนใจในตลาดนี้เช่นกัน ทั้ง Amazon Honeycode ที่เป็นแพลตฟอร์มช่วยให้การสร้างแอปพลิเคชันเอาไว้จัดการอีเวนท์หรืองานง่าย ๆ ทำได้โดยไม่ต้องเขียนโค้ด และ Parabola ที่เป็นแพลตฟอร์มจัดการขั้นตอนงานบนคลาว์ด ที่มีลักษณะความสามารถคล้าย ๆ กับ IFTTT

ทั้งหมดนี้จึงเป็นสาเหตุให้เราหยิบยกเรื่อง แพลตฟอร์มเขียนโค้ดน้อยเฉพาะทาง (bounded low-code platform) มาไว้ในเรดาร์ฉบับนี้อีกครั้ง กระนั้นแล้ว เรายังเคลือบแคลงความสามารถของมันอยู่ว่าจะทำได้ดีแค่ไหนหากปรับใช้ในวงกว้าง ด้วยที่ว่าเครื่องมือเหล่านี้เปรียบเหมือนกับผักตบชวาในประเทศไทย ที่สามารถเจริญเติบโตออกนอกกรอบได้เสมอ และพร้อมจะพันเกี่ยวทุกสิ่งรอบตัวเข้าด้วยกัน นั่นเป็นเหตุที่เราต้องเตือนให้ระมัดระวังอย่างมากหากจะนำแพลตฟอร์มประเภทนี้มาใช้งาน

เอกลักษณ์กระจายศูนย์

ประเมิน

ในปี 2016 Christopher Allen ผู้มีส่วนร่วมสำคัญต่อมาตรฐาน SSL/TLS เขียนบทความ the path to self-sovereign identity ที่ให้แรงบันดาลใจแก่เรา ถึงหลักการ 10 ข้อที่จะหนุนให้เกิดเอกลักษณ์ดิจิทัลรูปแบบใหม่ และวิถีทางจะไปถึงสู่จุดนั้น อธิปไตยเอกลักษณ์ของตนเอง หรือเอกลักษณ์แบบกระจายศูนย์ (decentralized identity) มีความหมายตามมาตรฐาน Trust over IP ว่าเป็น “เอกลักษณ์ของบุคคล องค์กร หรือสิ่งของ ที่จะไม่มีวันหมดอายุ เคลื่อนย้ายได้ ไม่ขึ้นกับหน่วยงานผู้มีอำนาจกลางใด และไม่สามารถถูกพรากไปได้” การสร้างและการปรับใช้ระบบเอกลักษณ์กระจายศูนย์ กำลังได้รับกระแสและใกล้จะสำเร็จเต็มที เราเห็นการใช้งานต่าง ๆ เพื่อปกป้องความเป็นส่วนตัว เช่น ระบบสุขภาพลูกค้า ระบบโครงสร้างพื้นฐานด้านสุขภาพของรัฐบาล และ ระบบอัตลักษณ์ทางกฎหมายองค์กร หากคุณต้องการเริ่มต้นเรียนรู้ระบบเอกลักษณ์กระจายศูนย์อย่างรวดเร็ว คุณสามารถประเมินเครื่องมือต่าง ๆ เช่น Sovrin Network, Hyperledger Aries และ Indy OSS หรือดูมาตรฐาน การระบุเอกลักษณ์กระจายศูนย์ และ ข้อมูลรับรองตัวตนที่พิสูจน์ได้ (verifiable credentials) เรากำลังจับตามองเรื่องนี้อย่างใกล้ชิด จากที่เรากำลังช่วยลูกค้าวางตำแหน่งทางกลยุทธ์ในยุคเอกลักษณ์ดิจิทัลรูปแบบใหม่นี้

Deployment drift radiator

ประเมิน

Deployment drift radiator หรือตัวสะท้อนที่สามารถบอกได้ว่าเวอร์ชันของซอฟต์แวร์ที่ติดตั้งในสภาพแวดล้อมต่าง ๆ ในองค์กรมีความแตกต่างมากน้อยเพียงใด แม้บางองค์กรจะติดตั้งซอฟต์แวร์อัตโนมัติแล้ว แต่ยังต้องการการอนุมัติจากผู้คุมในการติดตั้งในบางสภาพแวดล้อม โดยเฉพาะสภาพแวดล้อมที่เข้าใกล้โปรดักชัน ด้วยเหตุนี้สภาพแวดล้อมเหล่านั้นอาจมีซอฟต์แวร์เวอร์ชั่นที่ล้าหลังกว่าเวอร์ชันในปัจจุบันอยู่หลายรุ่น เทคนิคนี้ทำให้ความล่าช้านี้มองเห็นได้ง่าย ๆ ผ่านแดชบอร์ด โดยจะแสดงให้เห็นว่าในแต่ละสภาพแวดล้อมมีชิ้นส่วนใดบ้างที่ยังตามหลังเวอร์ชั่นล่าสุดอยู่ สิ่งนี้ช่วยสะท้อนให้เห็นถึงว่าเราเสียโอกาสไปเท่าไหร่อันเนื่องจากซอฟต์แวร์รุ่นล่าสุดยังไปไม่ถึงโปรดักชัน นอกจากนี้ยังช่วยแสดงถึงความเสี่ยงด้านความมั่นคงเพราะจะสะท้อนได้ว่าซอฟต์แวร์ที่มีการแก้ไขด้านความมั่นคงแล้วตัวใดยังไม่ถูกติดตั้งบ้าง

การเข้ารหัสแบบโฮโมมอร์ฟิก

ประเมิน

การเข้ารหัสแบบโฮโมมอร์ฟิก (HE) แบบสมบูรณ์หมายถึงวิธีการเข้ารหัสรูปแบบหนึ่งโดยสามารถนำข้อมูลที่ถูกเข้ารหัสแล้วมาประมวลผลต่อโดยตรงได้อย่างเช่น การค้นหาหรือการผ่านกระบวนการทางคณิตศาสตร์ ผลจากการประมวลผล ยังอยู่ในรูปแบบการเข้ารหัสและสามารถถอดรหัสเพื่อดูข้อมูลภายหลังได้แม้ว่าจะถูกเสนอตั้งแต่ปี 1978 แต่แนวคิดนี้ได้ถูกนำมาพัฒนาในปี 2009 ด้วยขีดความสามารถการประมวลผลที่เพิ่มขึ้นและการที่มีไลบรารีโอเพ่นซอร์สที่ง่ายต่อการใช้งาน อย่างเช่น SEAL Lattigo HElib และการเข้ารหัสแบบโฮโมมอร์ฟิกแบบบางส่วนใน Python ทำให้การเข้ารหัสแบบโฮโมมอร์ฟิกมีความเป็นไปได้ในการใช้งานจริงมากขึ้น

สถานการณ์ที่จูงใจให้ใช้การเข้ารหัสแบบโฮโมมอร์ฟิกนั้นรวมถึง การใช้เพื่อรักษาความเป็นส่วนตัวเมื่อต้องส่งข้อมูลให้ฝ่ายอื่นที่ไม่เชื่อถือทำการประมวลผล ตัวอย่างเช่น การประมวลผลข้อมูลที่ถูกเข้ารหัสบนคลาวด์ หรือการที่ยอมให้บุคคลอื่นเป็นตัวกลางในการรวบรวมผลลัพธ์ที่ถูกเข้ารหัสแบบโฮโมมอร์ฟิกของการทำแมชชีนเลิร์นนิงแบบหลายฝ่าย (federated machine learning)

นอกจากนี้นี้วิธีการเข้ารหัสแบบโฮโมมอร์ฟิกยังถือว่าปลอดภัยจากควอนตัมคอมพิวเตอร์และมาตรฐานสำหรับการเข้ารหัสแบบโฮโมมอร์ฟิกนั้นก็อยู่ในระหว่างการดำเนินการถึงแม้ด้วยข้อจำกัดในปัจจุบันทั้งในด้านประสิทธิภาพและความเป็นไปได้ในแต่ละประเภทการประมวลผล การเข้ารหัสแบบโฮโมมอร์ฟิกก็ยังควรค่าต่อความสนใจ

Hotwire

ประเมิน

Hotwire (HTML over the wire) เป็นเทคนิคในการสร้างเว็บแอปพลิเคชัน ซึ่งเพจถูกสร้างจากหลาย ๆ คอมโพเนนต์ต่างจาก SPAs สมัยใหม่ที่ HTML ของคอมโพเนนต์จะถูกสร้างที่ฝั่งเซิร์ฟเวอร์ แล้วส่ง “ผ่านสาย” ไปยังเบราว์เซอร์ แอปพลิเคชันสามารถประกอบส่วน HTML เข้าด้วยกันได้ด้วยโค้ด JavaScript เพียงเล็กน้อยบนเบราว์เซอร์ ทีมของเราและทีมอื่น ๆ ได้ทดลองใช้เทคนิคนี้นับแต่คำขอเว็บแบบไม่ประสานเวลา (asynchronous web request) สนับสนุนการใช้งานข้ามเบราว์เซอร์ช่วงปี 2005 แต่ด้วยเหตุผลหลายประการมันไม่ได้รับการผลักดันมากนัก

วันนี้ Hotwire ใช้เว็บเบราว์เซอร์ที่ทันสมัยและความสามารถต่าง ๆ ของ HTTP เพื่อให้ได้ความรวดเร็ว การตอบสนองที่ดี และความคล่องตัวทางธรรมชาติของแอพหน้าเดียว (SPAs) มันโอบรับการออกแบบเว็บแอปพลิเคชันที่เรียบง่าย ด้วยการเก็บการทำงานส่วนตรรกะไว้ที่ฝั่งเซิร์ฟเวอร์ และคงความเรียบง่ายของโค้ดในฝั่งไคลแอนต์ ทีมที่ Basecamp ได้เปิดตัวเฟรมเวิร์ค Hotwire สองสาม

เทคนิค

แม้บางองค์กรจะติดตั้งซอฟต์แวร์อัตโนมัติแล้ว แต่ยังต้องการการอนุมัติจากผู้คุมในการติดตั้งในบางสภาพแวดล้อม โดยเฉพาะสภาพแวดล้อมที่เข้าใกล้โปรดักชัน ด้วยเหตุนี้สภาพแวดล้อมเหล่านั้นอาจมีซอฟต์แวร์เวอร์ชั่นที่ล้าหลังกว่าเวอร์ชันที่กำลังพัฒนาอยู่หลายรุ่น Deployment drift radiator ทำให้ความล่าช้านี้มองเห็นได้ง่าย ๆ ผ่านแดชบอร์ด

(Deployment drift radiator)

เทคนิค

Secure multiparty computing (MPC) เป็นเทคนิคการประมวลผลร่วมกันระหว่างหน่วยงาน โดยที่แต่ละฝ่ายไม่ต้องเปิดเผยข้อมูลส่วนตัวให้อีกฝ่ายรู้ หรือใช้คนกลางมารับรองความถูกต้อง

(Secure multiparty computing)

เฟรมเวิร์คที่เพิ่มความสามารถให้กับ แอปพลิเคชัน ของพวกเขา รวมถึง Turbo และ Stimulus ซึ่ง Turbo มีชุดของเทคนิคและเฟรมเวิร์คเพื่อเพิ่มความเร็วในการตอบสนองของแอปพลิเคชัน โดยการป้องกันการโหลดหน้าเว็บทั้งหมด แสดงตัวอย่างหน้าเพจจากแคช และการแยกโหลดชิ้นส่วนหน้าจอตามคำขอการปรับค่า Stimulus ถูกออกแบบมาเพื่อปรับปรุง HTML แบบคงที่ในเบราว์เซอร์ให้ดีขึ้นโดยการเชื่อมต่อวัตถุ JavaScript กับอิลลิเมนต์บน HTML

อิมพอร์ตแมพ สำหรับ ไมโครฟรอนท์เอนด์

ประเมิน

การประกอบแอปพลิเคชันจากหลาย ๆ ไมโครฟรอนท์เอนด์ นั้นจะต้องมีส่วนของระบบที่ต้องตัดสินใจว่าจะโหลดไมโครฟรอนท์เอนด์ไหนและจากที่ใด และจนถึงตอนนี้ เราต้องเลือกระหว่างการสร้างโซลูชันขึ้นมาเอง หรือพึ่งพาเฟรมเวิร์คที่มีความครอบคลุมกว่าอย่าง single-spa แต่ในตอนนี้ ได้มีมาตรฐานใหม่ นั่นคือ อิมพอร์ตแมพ ซึ่งจะมาช่วยทั้งสองกรณี ประสบการณ์แรกของเราแสดงให้เห็นว่าการใช้ อิมพอร์ตแมพ สำหรับ ไมโครฟรอนท์เอนด์ ทำให้การแบ่งแยกเป็นระเบียบ โค้ดจาวาสคริปท์จะเป็นตัวระบุสิ่งที่จะอิมพอร์ต ส่วนแท็กสคริปท์เล็ก ๆ ใน HTML ที่ได้รับมาในตอนเริ่มต้น จะกำหนดว่าต้องโหลดโปรแกรมฟรอนท์เอนด์มาจากที่ใด แน่นอนว่า HTML ที่ว่านั้นถูกสร้างขึ้นในฝั่งเซิร์ฟเวอร์ ทำให้สามารถเปลี่ยนแปลงค่ากำหนดต่าง ๆ ในระหว่างการแสดงผลได้ และในหลาย ๆ ด้าน เทคนิคนี้ทำให้เรานึกถึง ตัวสร้างลิงค์หรือตัวโหลดเส้นทางสำหรับ Unix ไลบรารีแบบไดนามิก ในขณะนี้ ตัวอิมพอร์ตแมพนั้นรองรับแค่บราวเซอร์ Chrome แต่ด้วยการใช้โปรแกรมเสริม SystemJS ทำให้สามารถนำไปใช้กับบราวเซอร์ได้กว้างขวางขึ้น

Open Application Model (OAM)

ประเมิน

Open Application Model (OAM) เป็นความพยายามที่จะนำเสนอมาตรฐานของแพลตฟอร์มทางด้านโครงสร้างพื้นฐานในรูปแบบผลิตภัณฑ์ ไว้ให้นักพัฒนาสามารถอธิบายแอพพลิเคชันโดยไม่ขึ้นกับแพลตฟอร์มผ่านการใช้แอบแสตรกชันของคอมโพเนนท์ คอนฟิก สโคป และเทรต ในขณะเดียวกันผู้ทำแพลตฟอร์มก็สามารถกำหนดแพลตฟอร์มของพวกเขาด้วยการอธิบายเวิร์คโหลด สโคป และเทรต ได้ เช่นกัน จากที่เราเคยกล่าวไปครั้งที่แล้ว เราได้ติดตามแพลตฟอร์มตัวหนึ่งที่นำเทคนิคนี้ไปพัฒนา ซึ่งก็คือ KubeVela ที่ใกล้จะออกเวอร์ชั่น 1.0 แล้ว และเราสงสัยว่า KubeVela จะสามารถพิสูจน์ไอเดียของ OAM ได้หรือไม่

Privacy-focused web analytics

ประเมิน

การวิเคราะห์เว็บที่เน้นความเป็นส่วนตัว เป็นเทคนิคการวิเคราะห์เว็บโดยไม่รุกล้ำความเป็นส่วนตัวของผู้ใช้งานด้วยการทำให้ผู้ใช้งานเป็นบุคคลนิรนาม ผลจากการปฏิบัติตามข้อบังคับทั่วไปเกี่ยวกับการคุ้มครองข้อมูล (GDPR) ที่น่าประหลาดใจอย่างหนึ่งคือ หลายองค์กรเลือกจะทำให้ประสบการณ์ของผู้ใช้งานย่ำแย่ลง ด้วยการเพิ่มขั้นตอนการยอมรับคุกกี้ที่ยุ่งยาก โดยเฉพาะเมื่อผู้ใช้งานไม่ยอมรับค่าตั้งต้นของคุกกี้ในทันที การวิเคราะห์เว็บที่เน้นความเป็นส่วนตัวที่ทั้งมีประโยชน์ในแง่ของการทำตามตัวบทกฎหมายและเจตจำนงของ GDPR และยังสามารถหลีกเลี่ยงความจำเป็นในการนำเสนอขั้นตอนการยอมรับคุกกี้ที่ดูคุกคาม หนึ่งในแพลตฟอร์มที่นำวิธีการนี้ไปใช้คือ Plausible

การเขียนโปรแกรมแบบม็อบระยะไกล

ประเมิน

ม็อบโปรแกรมมิ่ง เป็นหนึ่งในเทคนิคที่ทีมงานของเราพบว่าทำได้ง่ายขึ้นเมื่อทำงานทางไกล การม็อบโปรแกรมมิ่งทางไกล ช่วยให้คนในทีมสามารถรุมกันเพื่อแก้ปัญหาและดูส่วนของโค้ดต่าง ๆ ด้วยกันโดยไม่ติดข้อจำกัดทางกายภาพของสถานที่ ที่รองรับคนได้จำนวนจำกัด ทีมจะสามารถประสานงานกันและทำงานร่วมกันในปัญหาหรือโค้ดได้อย่างรวดเร็ว โดยไม่ต้องเชื่อมต่อกับจอแสดงผลขนาดใหญ่หรือเสียเวลาจองห้องประชุมและเตรียมไวท์บอร์ด

Secure multiparty computing

ประเมิน

Secure multiparty computing (MPC) เป็นเทคนิคการประมวลผลร่วมกันระหว่างหน่วยงาน โดยที่แต่ละฝ่ายไม่ต้องเปิดเผยข้อมูลส่วนตัวให้อีกฝ่ายรู้ หรือใช้คนกลางมารับรองความถูกต้อง ซึ่งเหมาะกับกรณีที่ทั้งคู่ไม่เชื่อใจกัน โดยที่ทั้งคู่จะร่วมกันประมวลผลข้อมูลด้วยกฎข้อเดียวกัน แต่ผลลัพธ์ที่ได้จะไม่ถูกส่งต่อให้อีกฝ่ายทราบ โจทย์คลาสิคของ MPC คือ ปัญหาของมหาเศรษฐี ที่เศรษฐีสองคนต้องการรู้ว่าใครรวยกว่ากันโดยมีเงื่อนไขที่ว่าจะต้องไม่เปิดเผยมูลค่าทรัพย์สินรวมให้อีกฝ่าย รวมถึงคนกลางได้รับรู้ เป็นต้น ในทางปฏิบัติการสร้าง MPC ทำได้หลายวิธี เช่น การใช้โปรโตคอลการแชร์ความลับ (secret sharing) โปรโตคอลการส่งข้อมูลแบบหลงลืม (oblivious transfer) โปรโตคอลวงจรที่บิดเบือน (garbled circuits) หรือเทคนิคการเข้ารหัสแบบโฮโมมอร์ฟิก (homomorphic encryption) ส่วนในตลาดก็มีโซลูชัน MPC เชิงพาณิชย์ที่เพิ่งเปิดตัวไป อย่าง Antchain Morse ที่อ้างว่าสามารถแก้ปัญหาการแชร์ความลับ หรือการทำแมชชีนเลิร์นนิงระหว่างต่างหน่วยงานได้ เช่น งานตรวจสอบเครดิตร่วมกันของหลายฝ่าย หรืองานแลกเปลี่ยนข้อมูลเวชระเบียนของคนไข้ระหว่างโรงพยาบาล แม้ว่าแพลตฟอร์มเหล่านี้จะดูน่าสนใจในทางการตลาด แต่เราต้องรอดูว่าสิ่งนี้จะมีประโยชน์จริง ๆ ในทางปฏิบัติหรือไม่

GitOps

เฝ้าระวัง

เราอยากเตือนให้ใช้ GitOps อย่างระมัดระวัง โดยเฉพาะในประเด็นของการใช้บรานช์ โดยทั่วไปแล้ว GitOps สามารถมองเป็นวิธีหนึ่งในการกำหนดโครงสร้างพื้นฐานด้วยโค้ด ให้เป็นจริงได้ โดยจะนำโค้ดโครงสร้างพื้นฐานจาก Git ไปรวมและประยุกต์ใส่สภาพแวดล้อมต่าง ๆ อย่างต่อเนื่อง

เมื่อใช้ GitOps ร่วมกับแนวคิดที่กำหนดให้หนึ่งบรานช์เป็นตัวแทนของสภาพแวดล้อมใด ๆ แล้ว การเลื่อนการเปลี่ยนแปลงจากสภาพแวดล้อมหนึ่งไปสู่อีกสภาพแวดล้อมหนึ่ง จะทำได้ด้วยการรวมโค้ดจากต่างบรานช์เข้าด้วยกันนั่นเอง ซึ่งพิจารณาดูเผิน ๆ แล้ว การกำหนดให้โค้ดเป็นแหล่งข้อมูลจริงเพียงหนึ่งเดียว (single source of truth) ก็เป็นเหตผลที่เข้าท่า แต่วิธีนี้อาจนำไปสู่สถานการณ์ที่แย่ลงได้ หากการรวมโค้ดมีปัญหาระหว่างทาง หรือไม่เกิดการรวมโค้ดขึ้นมา จนทำให้สภาพแวดล้อมต่าง ๆ ค่อย ๆ เพี้ยนออกจากกัน จนในที่สุดสภาพแวดล้อมใด ๆ จำเป็นต้องมีลักษณะเฉพาะของตัวเอง สถานการณ์นี้คล้าย ๆ กับเรื่องที่เราเคยเตือนเมื่อครั้งก่อน ที่เรากล่าวถึง GitFlow และการสร้างบรานช์ที่มีช่วงชีวิตยาวนาน

ทีมแพลตฟอร์มที่ถูกแบ่งตามเลเยอร์ของเทคโนโลยี

เฝ้าระวัง

การที่องค์กรต่าง ๆ หันมาสนใจสร้างแพลตฟอร์มซอฟต์แวร์กันมากขึ้น ได้ตอบแทนคุณค่ามากมายกลับสู่องค์กร แต่วิธีการพัฒนาแพลตฟอร์มที่เหมาะสมนั้นเต็มไปด้วยหลุมพรางมากมาย เป็นเรื่องปกติที่พอมีแนวคิดใหม่เกิดขึ้นเราก็เห็นเทคโนโลยีเก่าพยายามชุบตัวแล้วมาพร้อมกับคำพูดเก๋ ๆ ที่อาจทำให้บางคนหลงลืมไปว่าทำไมเทคโนโลยีเหล่านั้นไม่เป็นที่นิยมแล้วในปัจจุบัน ดังที่เห็นตัวอย่างการชุบตัว ที่เราเคยยกมาคุยผ่านหัวข้อ ESB ในคราบ API Gateway ในเรดาร์ฉบับก่อน ๆ เช่นเดียวกัน เราก็เห็นตัวอย่างการแบ่งทีมตามเลเยอร์ของเทคโนโลยีแบบเดิม ๆ แล้วเรียกพวกเขาเป็นแพลตฟอร์มทีมต่าง ๆ ให้เห็นอยู่ทั่วไป ในอดีตที่ผ่านมาการจะสร้างแอปพลิเคชันสักตัวหนึ่ง มันเป็นเรื่องปกติที่จะแบ่งเป็นทีมฟรอนท์เอนด์ แยกจากแบ็คแอนด์ หรือทีมที่บริหารฐานข้อมูล เรากลับเห็นรูปแบบการแบ่งทีมในลักษณะนี้ เกิดขึ้นซ้ำกับบางองค์กรที่พยายามจะสร้างแพลตฟอร์มของตัวเองขึ้นมา ต้องขอบคุณ Conway's Law ที่ทำให้เรารู้ว่า การจัดวางทีมพัฒนาแฟลตฟอร์มตามความสามารถทางธุรกิจ นั้นเป็นโมเดลที่มีประสิทธิภาพกว่ามาก เพราะทำให้ทีมหนึ่งสามารถดูแลและเป็นเจ้าของความสามารถนั้นแบบครบวงจร รวมถึงสิทธิ์ในการดูแลข้อมูลด้วย ซึ่งเรื่องนี้ลดภาระการพูดคุยกันระหว่างเลเยอร์ หากเปรียบเทียบกับ ทีมแพลตฟอร์มที่ถูกแบ่งตามเลเยอร์ของเทคโนโลยี ที่ทีมพัฒนาฟรอนต์เอนด์จะต้องรอทีมแบ็คแอนด์ ที่ต้องรอทีมข้อมูลเป็นทอด ๆ กันไป

นโยบายความซับซ้อนของรหัสผ่านที่คิดไปเองว่าดี

เฝ้าระวัง

การกำหนดนโยบายของรหัสผ่านเป็นมาตรการพื้นฐานสำหรับหลาย ๆ องค์กรในปัจจุบัน อย่างไรก็ตามเรายังคงเห็นหลายองค์กรมีการบังคับใช้รหัสผ่านที่ประกอบไปด้วยสัญลักษณ์ ตัวเลข ตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก และยังรวมไปถึงการใส่อักขระพิเศษ สิ่งเหล่านี้เรียกว่า นโยบายความซับซ้อนของรหัสผ่านที่คิดไปเองว่าดี ซึ่งนำไปสู่การรับรู้ในด้านความปลอดภัยที่ไม่ถูกต้อง เนื่องจากมันยิ่งทำให้ผู้ใช้งานเลือกใช้รหัสผ่านที่มีความปลอดภัยต่ำเพราะว่าตัวเลือกที่ให้นั้นยากต่อการจดจำและการพิมพ์ จาก คำแนะนำของ NIST ปัจจัยหลักของความเข้มแข็งของรหัสผ่านนั้นคือความยาวของรหัสผ่าน ดังนั้นผู้ใช้งานควรเลือกใช้รหัสผ่านที่มีลักษณะเป็นกลุ่มคำโดยมีความยาวสูงสุดได้ 64 ตัวอักษรรวมช่องว่าง รหัสผ่านแบบกลุ่มคำมีความปลอดภัยสูงกว่าและสามารถจดจำได้ง่ายกว่า

พูลรีเควสเป็นวิธีการเดียวในการตรวจทานโค้ดของเพื่อนร่วมงาน

เฝ้าระวัง

บางองค์กรชอบคิดไปเองว่า พูลรีเควสเป็นวิธีการเดียวในการตรวจทานโค้ดของเพื่อนร่วมงาน เราพบว่าวิธีนี้มีแต่จะสร้างคอขวดให้กับการพัฒนา หนำซ้ำยิ่งทำให้คุณภาพของความคิดเห็นแย่ลงกว่าเดิม อันเนื่องจากผู้ตรวจทานที่มีอยู่จำกัดจะเริ่มปฏิเสธคำขอเมื่อมีงานอยู่ล้นมือ แม้ว่าข้อโต้แย้งอาจกล่าวได้ว่าพูลรีเควสเป็นวิธีหนึ่งในการตรวจทานโค้ด "ตามกฎระเบียบขององค์กร" แต่หนึ่งในลูกค้าของเราก็เคยถูกปฏิเสธเรื่องนี้มาแล้วเช่นกัน เพราะไม่มีหลักฐานใดยืนยันได้ว่ามีการตรวจทานเกิดขึ้นจริงก่อนที่จะยอมรับ เราเชื่อว่าพูลรีเควสเป็นเพียงหนึ่งในวิธีการจัดการเวิร์กโฟลว์ของการตรวจทานโค้ด เราจึงอยากให้ผู้คนพิจารณาวิธีการอื่น ๆ โดยเฉพาะอย่างยิ่งเมื่อการฝึกสอนและส่งต่อความคิดเห็นให้กับเพื่อนร่วมงานเป็นเรื่องที่จำเป็น

SAFe™

เฝ้าระวัง

"การจะทำอไจล์ได้ ต้องเริ่มที่การเป็นอไจล์เสียก่อน" เป็นจุดยืนที่เรายึดถือมานานแล้ว ซึ่งไม่ใช่เรื่องแปลกใหม่แต่อย่างใด แต่จากรายงานประจำเดือนพฤษภาคมปี 2019 ของ Gartner ที่ได้นำเสนอว่า SAFe™ (The Scaled Agile Framework®) นั้นเป็นอไจล์เฟรมเวิร์คที่ได้รับความสนใจและถูกใช้งานมากที่สุดในระดับองค์กร กอปรกับที่มีหลายองค์กรมากขึ้นกำลังอยู่ในช่วงปรับตัวสู่รูปแบบการทำงานใหม่ ๆ เราจึงคิดว่านี่เป็นช่วงเวลาสำคัญที่จะหยิบเรื่องนี้มาเน้นย้ำอีกครั้งหนึ่ง เราเจอกับองค์กรมากมายที่ต้องลำบากทำตามกระบวนการที่ SAFe กำหนด ที่เทอะทะเกินไป และที่ต้องมีผู้คุมอยู่ทุกขั้นตอน กระบวนการเหล่านี้สร้างความลำบากในสายโครงสร้างองค์กร และส่งผลกระทบต่อการทำงานในแต่ละวัน นอกจากนี้ SAFe ยังส่งเสริมการทำงานแบบแยกส่วนกันภายในองค์กร (Silo) อีกทั้งยังขัดขวางไม่ให้เกิดแพลตฟอร์มที่ส่งเสริมความสามารถทางธุรกิจอย่างแท้จริง ส่วนการควบคุมจากเบื้องบนลงสู่ด้านล่างก็ก่อให้เกิดความสิ้นเปลืองในสายธารคุณค่า (value stream) โดยใช่เหตุ เป็นการกีดกันความคิดสร้างสรรค์ทางด้านทางวิศวกรรมของพนักงาน ในขณะเดียวกันก็จำกัดความเป็นอิสระและการทดลองในทีม เราจึงอยากแนะนำว่า แทนที่จะเสียเวลาไปกับการวัดว่าใครลงแรงไปเท่าไหร่ หรือยึดเอาพิธีกรรมในการทำงานเป็นที่ตั้ง องค์กรต่าง ๆ ควรไปใช้กระบวนการอื่นที่มีความคล่องตัวกว่า อย่าง เฟรมเวิร์ค EDGE ที่วิธีการและการกำกับดูแลมุ่งเน้นไปที่คุณค่าที่จะได้รับ ที่ส่งเสริมการเปลี่ยนแปลงโดยลดขั้นตอนยุ่งยากภายในองค์กรลง นอกจากนี้เราอยากแนะนำให้ประเมินภาระความรู้ความเข้าใจของทีม ประกอบกัน เพื่อระบุประเภทของทีม และกำหนดวิธีที่ทีมต่าง ๆ จะประสานงานกัน

Scaled Agile Framework® และ SAFe™ เป็นเครื่องหมายการค้าของบริษัท Scaled Agile

เทคนิค

บางองค์กรชอบคิดไปเองว่า พูลรีเควสเป็นวิธีการเดียวในการตรวจทานโค้ดของเพื่อนร่วมงาน เราพบว่าวิธีนี้มีแต่จะสร้างคอขวดให้กับการพัฒนา หนำซ้ำยิ่งทำให้คุณภาพของความคิดเห็นแย่ลงกว่าเดิม

(พูลรีเควสเป็นวิธีการเดียวในการตรวจทานโค้ดของเพื่อนร่วมงาน)

แบ่งแยกทีมในการดูแลโค้ดกับไปป์ไลน์

เฝ้าระวัง

โดยอุดมคติแล้วเราอยากให้ทีมที่เป็นเจ้าของโค้ดกับไปป์ไลน์ที่ติดตั้งโค้ดนั้นเป็นทีมเดียวกัน โดยเฉพาะอย่างยิ่งทีมที่กำลังเสริมสร้างวัฒนธรรมแบบ DevOps แต่ก็เป็นเรื่องน่าเสียดายที่เรายังคงพบเห็นบางองค์กรที่แบ่งแยกทีมในการดูแลโค้ดกับไปป์ไลน์ โดยจะยกหน้าที่สำหรับดูแลและจัดการไปป์ไลน์ให้กับทีมที่ดูแลโครงสร้างพื้นฐาน ส่งผลให้เกิดความล่าช้าเวลาต้องการที่จะเปลี่ยนแปลงบางอย่างและเป็นอุปสรรคในการพัฒนาไปป์ไลน์ให้ดีขึ้น รวมถึงส่งผลให้ทีมนักพัฒนาขาดความรู้สึกเป็นเจ้าของและรู้สึกไม่มีส่วนร่วมในขั้นตอนการติดตั้ง สาเหตุหนึ่งที่ทำให้เกิดสิ่งนี้ขึ้นก็น่าจะมาจากการพยายามแบ่งแยกทีมออกจากกัน และอีกสาเหตุหนึ่งที่เป็นไปได้ก็คือความต้องการที่จะให้มีขั้นตอนหรือผู้ที่ทำหน้าที่เป็น “ผู้เฝ้าประตู” ซึ่งถึงแม้วิธีการพยายามควบคุมกำกับดูแลแบบนี้จะฟังดูมีเหตุผลตามหลักเกณฑ์ดั้งเดิมในการนำมาใช้ เราก็ยังพบว่าโดยทั่วไปมันก็ยังทำให้เกิดความยากลำบากและไม่เป็นประโยชน์อยู่ดี

แพลตฟอร์มที่ใช้การร้องขอในการดำเนินงาน

เฝ้าระวัง

หนึ่งในเป้าหมายสูงสุดของการเป็นแพลตฟอร์มที่ดี คือลดกระบวนการที่ต้องทำเรื่องร้องขอเข้ามาให้น้อยที่สุด เนื่องจากมันทำให้เกิดการรอในกระบวนการส่งมอบคุณค่า เป็นเรื่องน่าเศร้าที่เรายังเห็นหลายองค์กรยังไม่สามารถผลักดันแนวคิดนี้ให้พ้นตัวไปได้เสียที ส่งผลให้เกิดแพลตฟอร์มที่ใช้การร้องขอในการดำเนินงาน ขึ้น มันเป็นเรื่องน่าเบื่อหน่ายเข้าไปอีก เมื่อบางองค์กรนำกระบวนการเปิดคำร้องมาใช้ขั้นตรงกลางระหว่างแพลตฟอร์มคลาว์ดที่มี API อยู่แล้ว แถมผู้ใช้ยังสามารถบริการตัวเองได้เอง มันอาจจะยากและไม่จำเป็น ที่จะทำแพลตฟอร์มที่ทีมสามารถให้บริการตนเองได้เลยตั้งแต่วันแรก แต่มันควรต้องเป็นปลายทางที่ทุกคนอยากไปให้ถึง
ระบบที่พึ่งพาการอนุมัติเกินความจำเป็น และการขาดความไว้วางใจระหว่างกัน เป็นสาเหตุหนึ่งที่ขัดขืนไม่ให้เกิดการเปลี่ยนแปลง ดังนั้นวิธีหนึ่งในการลดขั้นตอนจากกระบวนการอนุมัติ คือการใช้ระบบตรวจสอบอัตโนมัติและเพิ่มการแจ้งเตือนที่ดีลงไปในแพลตฟอร์ม ยกตัวอย่างเช่น การเปิดเผยให้ทีมเห็นค่าใช้จ่ายในการดำเนินการ จะทำให้ทีมสามารถสร้างระบบอัตโนมัติที่สามารถจำกัดงบประมาณเพื่อกันไม่ให้จ่ายเกินกว่าที่กำหนด นอกจากนี้ การใช้นโยบายความมั่นคงด้วยโค้ด และการใช้เครื่องมือตรวจจับการกำหนดค่า หรือตัววิเคราะห์เช่น Recommender ก็เป็นตัวช่วยที่ดีให้ทีมทำสิ่งที่ถูกต้อง

TECHNOLOGY RADAR

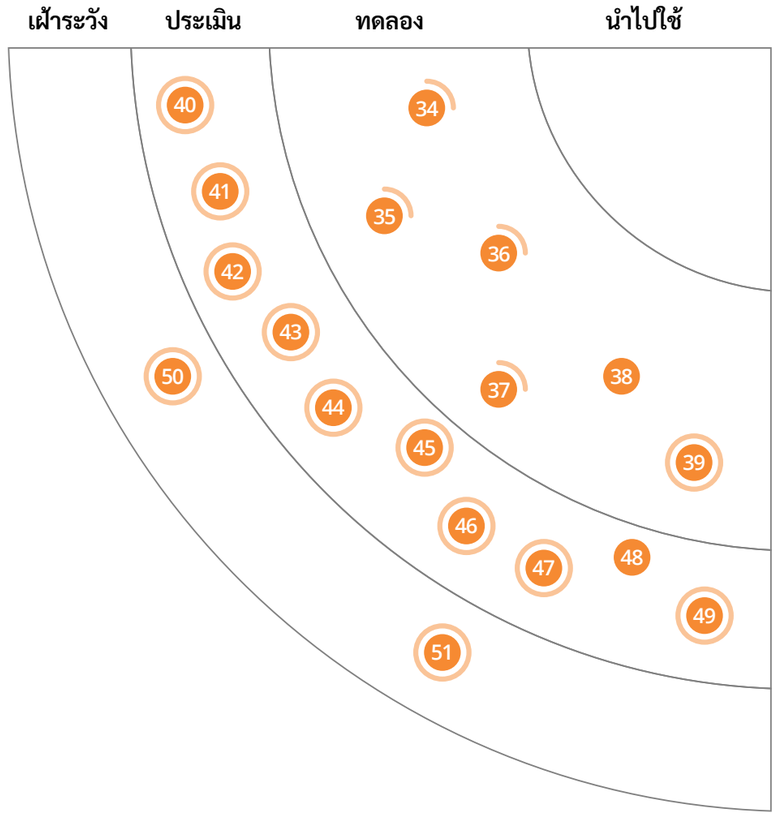
แพลตฟอร์ม

แพลตฟอร์ม

AWS Cloud Development Kit

ทดลอง

หลายทีมของเราที่ได้ใช้งาน AWS แล้วพบว่า AWS Cloud Development Kit (AWS CDK) เป็นเครื่องมือเริ่มต้นของ AWS ที่เหมาะสมสำหรับการเปิดใช้งานการจัดเตรียมโครงสร้างพื้นฐาน โดยเฉพาะอย่างยิ่งพวกเขาชอบใช้เพราะมันสนับสนุนการเขียนโปรแกรมเป็นหลักแทนการใช้ไฟล์กำหนดค่า ซึ่งมันทำให้พวกเขาสามารถใช้เครื่องมือแนวทางการทดสอบ และทักษะที่พวกเขามีอยู่แล้วได้ ทั้งนี้เช่นเดียวกับเครื่องมืออื่นที่มีลักษณะคล้ายกัน ความระมัดระวังยังเป็นสิ่งที่ต้องมีเพื่อทำให้มั่นใจว่าการติดตั้งจะยังคงง่ายต่อการทำความเข้าใจและการดูแลรักษา ในปัจจุบันชุดเครื่องมือในการพัฒนานี้สามารถรองรับการใช้งานกับ TypeScript, JavaScript, Python, Java, C# และ .NET ผู้ให้บริการรายใหม่กำลังถูกเพิ่มเข้าไปในแพ็คเกจหลักของ CDK และเรายังได้ใช้งานทั้ง AWS Cloud Development Kit และ HashiCorp's Cloud Development Kit for Terraform เพื่อสร้างการกำหนดค่าของ Terraform และเปิดใช้งานการจัดเตรียมแพลตฟอร์ม Terraform อย่างประสบความสำเร็จ



Backstage

ทดลอง

เรายังเห็นความสนใจและการใช้งาน Backstage ที่มากขึ้น รวมถึงการนำแนวคิดศูนย์ส่งเสริมความรู้สำหรับนักพัฒนามาใช้ (developer portal) เพื่อให้การพัฒนาภายในมีสภาพแวดล้อมระหว่างทีมเป็นไปในทิศทางเดียวกัน สืบเนื่องจากที่เครื่องมือและเทคโนโลยีมีมากขึ้นทุกวัน การกำหนดมาตรฐานบางอย่างขึ้นมาภายในกลายเป็นเรื่องที่ต้องให้ความสำคัญ เพื่อสร้างความสอดคล้องระหว่างทีม และเพื่อให้นักพัฒนาง่วนอยู่กับการสร้างนวัตกรรมและผลิตภัณฑ์ใหม่ ๆ โดยไม่เสียเวลากับการสร้างสิ่งที่ซ้ำซ้อนขึ้นมา การมีศูนย์ส่งเสริมความรู้สำหรับนักพัฒนา จึงช่วยให้นักพัฒนาสามารถค้นหาบริการที่ทีมอื่น ๆ ได้ทำไว้แล้ว หรือเป็นที่ ๆ แชร์เทคนิคในการแก้ปัญหาของแต่ละทีม Backstage เป็นแพลตฟอร์มโอเพนซอร์สสำหรับสร้างศูนย์ส่งเสริมนักพัฒนาที่สร้างโดย Spotify โดยเราสามารถสร้างแม่แบบซอฟต์แวร์ขึ้นมาเพื่อให้การขึ้นโครงการใหม่ทำได้สะดวก ที่ภายในได้เตรียมเครื่องมือด้านโครงสร้างพื้นฐานไว้ให้พร้อม เพื่อสร้างความเป็นหนึ่งเดียวระหว่างทีม และมีคลังเอกสารทางเทคนิคต่าง ๆ ให้ใช้งาน Backstage มีสถาปัตยกรรมการออกแบบที่รองรับการต่อเติมขยาย จึงสามารถเพิ่มขยายความสามารถ หรือปรับแต่งระบบให้เข้ากับระบบนิเวศขององค์กรนั้น ๆ ได้อย่างเหมาะสม

Delta Lake

ทดลอง

Delta Lake เป็นโอเพ่นซอร์สชั้นจัดเก็บข้อมูล ซึ่งถูกพัฒนาโดย Databricks ซึ่งพยายามนำทรานแซกชัน ACID มาสู่การประมวลผลข้อมูลขนาดใหญ่ ในโปรเจคที่ต่อยอดการใช้งาน Databricks ทั้ง ดาต้าเลค หรือ ดาต้าเมช ทีมงานของเรายังคงเลือกใช้ที่เก็บข้อมูล Delta Lake มากกว่า

นำไปใช้

ทดลอง

34. AWS Cloud Development Kit
35. Backstage
36. Delta Lake
37. Materialize
38. Snowflake
39. ฟอนต์ที่เปลี่ยนแปลงรูปแบบได้ (Variable fonts)

ประเมิน

40. Apache Pinot
41. Bit.dev
42. DataHub
43. Feature Store
44. JuiceFS
45. การใช้งาน Kafka API โดยไม่ต้องใช้ Kafka
46. NATS
47. Opstrace
48. Pulumi
49. Redpanda

เฝ้าระวัง

50. Azure Machine Learning
51. เครื่องมือโครงสร้างพื้นฐานด้วยโค้ด (IaC) ฉบับทำเอง

แพลตฟอร์ม

LinkedIn ได้ค่อย ๆ พัฒนาปรับปรุงโครงการ WhereHows ของตนให้กลายเป็นแพลตฟอร์มที่มีความสามารถในการสำรวจหาข้อมูลสำเร็จ โดยทำจากระบบที่ข้อมูลเมทาดาต้าสามารถเพิ่มขยายได้

(DataHub)

การใช้พื้นที่จัดเก็บไฟล์โดยแบบปกติ เช่น S3 หรือ ADLS แน่นอนว่า Delta Lake ยังมีข้อจำกัดโดยสนับสนุนเฉพาะโปรเจคที่ใช้แพลตฟอร์มที่รองรับการจัดเก็บไฟล์ในรูปแบบไฟล์ Parquet เท่านั้น Delta Lake อำนวยความสะดวกในการอ่าน/เขียนข้อมูลพร้อมกันในกรณีที่จำเป็นต้องมีการทำทรานแซกชันระดับไฟล์ เราพบว่าการผสานรวมที่ราบรื่นของ Delta Lake กับ API ในการทำงานแบบแบทช์ และการทำงานแบบไมโครแบทช์ ของ Apache Spark ได้ก่อให้เกิดประโยชน์อย่างยิ่ง โดยเฉพาะฟีเจอร์เช่น การเดินทางข้ามเวลา ที่ช่วยเราการเข้าถึงข้อมูล ณ เวลาใดเวลาหนึ่ง หรือการย้อนกลับของคอมมิท เช่นเดียวกับ วิวัฒนาการของ schema ที่สนับสนุนการเขียนข้อมูล แม้ว่าจะมีข้อจำกัดบางประการเกี่ยวกับฟีเจอร์เหล่านี้ก็ตาม

Materialize

ทดลอง

Materialize เป็นระบบฐานข้อมูลแบบ Streaming ที่สามารถช่วยให้เราประมวลผลข้อมูลเฉพาะส่วนที่เพิ่มเข้ามาโดยไม่จำเป็นต้องใช้ดาต้าไปป์ไลน์ที่ซับซ้อน การประมวลผลสามารถทำได้โดยสร้างวิวด้วยภาษา SQL มาตรฐาน และเชื่อม Materialize เข้ากับแหล่งข้อมูลสตรีม โดยที่ภายใน Materialize ใช้เอนจิ้นแบบ differential data flow ประมวลผลข้อมูลเฉพาะส่วนที่เพิ่มเข้ามา เพื่อให้ได้ผลลัพธ์ที่ถูกต้อง คงเส้นคงวา และมีความหน่วงน้อยที่สุด ซึ่งยังต่างจากระบบฐานข้อมูลแบบดั้งเดิม ตรงที่มันไม่มีข้อจำกัดในการสร้าง Materialized Views และการประมวลผลนั้นเกิดขึ้นแบบเรียลไทม์ เรายังได้ใช้ Materialize ร่วมกับ Spring Cloud Stream และ Kafka เพื่อสืบค้นสตรีมเหตุการณ์สำหรับข้อมูลเชิงลึกในระบบแบบกระจายที่ขับเคลื่อนด้วยเหตุการณ์ และเราค่อนข้างชอบภาพรวมของระบบดังกล่าว

Snowflake

ทดลอง

จากที่เราได้กล่าวถึง Snowflake บนเรดาร์ที่ผ่านมา เราได้รับประสบการณ์มากขึ้นเช่นเดียวกับ Data mesh ที่เป็นอีกทางเลือกหนึ่งสำหรับดาต้าแวร์เฮาส์ หรือดาต้าเลค Snowflake ยังคงสร้างความประทับใจด้วยฟีเจอร์อย่างฟีเจอร์ time travel ซึ่งทำให้ดูข้อมูลในอดีตได้ ฟีเจอร์การโคลนแบบ zero-copy ฟีเจอร์การแบ่งปันข้อมูล และฟีเจอร์ตลาดข้อมูล (Snowflake Marketplace) ทำให้นั่นเป็นเหตุผลที่เราชอบ Snowflake มากกว่าตัวเลือกอื่นในท้องตลาด Redshift กำลังพยายามแยกการจัดเก็บ (storage) และการประมวลผล (compute) ซึ่งเป็นจุดแข็งของ Snowflake แต่ถึงแม้จะใช้ Redshift ร่วมกับ Redshift Spectrum ก็ไม่สะดวกและไม่ยืดหยุ่นเท่า ส่วนหนึ่งเป็นเพราะมันผูกมัดกับ Postgres (แต่เรายังคงชอบ Postgres อยู่นะ) คำค้นหาที่ติดต่อกับภายนอก (Federated queries) อาจเป็นเหตุผลที่ควรจะใช้ Redshift แต่เมื่อพูดถึงการใช้งาน Snowflake จะง่ายกว่ามาก อีกหนึ่งตัวเลือกอย่าง BigQuery ก็ใช้งานง่ายมากเช่นกัน แต่ในการติดตั้งคลาวด์หลาย ๆ รายผสมผสานกัน Snowflake ยังเป็นตัวเลือกที่ดีกว่า เพราะเราประสบความสำเร็จในการใช้งาน Snowflake กับ GCP, AWS และ Azure

ฟอนต์ที่เปลี่ยนแปลงรูปแบบได้ (Variable fonts)

ทดลอง

Variable fonts เป็นวิธีแก้ปัญหาในการค้นหาและรวบรวมไฟล์ฟอนต์ที่จำเป็นต้องมีหลาย ๆ ไฟล์ เพื่อให้ได้ฟอนต์แต่ละแบบที่มีน้ำหนักและสไตล์ที่แตกต่างกัน Variable fonts ทำให้ทุกอย่างรวมอยู่ในไฟล์ฟอนต์เพียงไฟล์เดียว ซึ่งสามารถกำหนดรูปแบบและน้ำหนักที่ต้องการได้ แม้ว่านี่จะไม่ใช่เรื่องใหม่ แต่เรายังคงเห็นหลาย ๆ เว็บไซต์และโปรเจกต์ที่อาจจะได้รับประโยชน์จากแนวทางง่าย ๆ นี้ หากคุณมีหน้าเว็บที่มีฟอนต์เดียวกันแต่มีการใช้งานหลายรูปแบบ เราขอแนะนำให้ลองนำ Variable fonts ไปปรับใช้

Apache Pinot

ประเมิน

Apache Pinot เป็นฐานข้อมูลประเภท OLAP ที่รองรับการเก็บข้อมูลแบบกระจายตัว โดยสร้างขึ้นมาเพื่องานวิเคราะห์ข้อมูลแบบทันทีทันใดที่ต้องการความหน่วงต่ำ มันรองรับการนำเข้าข้อมูลเป็นชุด ๆ ผ่านแหล่งข้อมูล อย่าง Hadoop HDFS, Amazon S3, Azure ADLS หรือ Google Cloud Storage หรือแหล่งข้อมูลแบบสตรีม อย่าง Apache Kafka ก็ได้ ปกติแล้วหากเป็นงานวิเคราะห์ข้อมูลที่ผู้ใช้สามารถโต้ตอบได้ทันที จะต้องการระบบที่มีความหน่วงต่ำ ซึ่งวิธีประเภท SQL-on-Hadoop ไม่สามารถทำความเร็วระดับนี้ได้ แต่ในเอนจิ้น OLAP สมัยใหม่ เช่น Apache Pinot หรือ Apache Druid และ Clickhouse จะให้ความหน่วงต่ำกว่ามาก และเหมาะอย่างยิ่งกับงานวิเคราะห์จากแหล่งข้อมูลที่ไม่เปลี่ยนแปลงค่าแล้ว (immutable data) ที่ต้องการผลอย่างรวดเร็ว เช่นงานรวมผลข้อมูล (Aggregation) ที่ชุดข้อมูลไหลมาจากแหล่งข้อมูลแบบทันทีทันใด Apache Pinot นั้นถูกสร้างขึ้นโดย LinkedIn และถูกมอบให้ Apache เป็นผู้ดูแลต่อในปลายปี 2018 จากนั้นเป็นต้นมา Apache Pinot ได้เพิ่มสถาปัตยกรรมปลั๊กอินเข้าไป และรองรับการทำงานผ่าน SQL รวมไปถึงความสามารถที่สำคัญอื่น ๆ ด้วย อย่างไรก็ตาม Apache Pinot มีความซับซ้อนในการบริหารจัดการอยู่พอสมควร หากคุณมีปริมาณข้อมูลขนาดใหญ่พอ และมองหาความสามารถในการค้นหาข้อมูลโดยใช้ความหน่วงต่ำ เราอยากแนะนำให้คุณลองประเมิน Apache Pinot ดู

Bit.dev

ประเมิน

Bit.dev เป็นคลาวด์แพลตฟอร์มสำหรับร่วมมือกันสร้าง UI คอมโพเนนท์ โดยใช้ความสามารถของ Bit ในการแยกคอมโพเนนท์และแยกเป็นโมดูลออกจากกัน เพื่อการนำกลับมาใช้ใหม่ แม้เทคโนโลยีเว็บคอมโพเนนท์ จะอยู่มาสักพักใหญ่แล้ว แต่การจะนำคอมโพเนนท์ใด ๆ จากต่างโครงการกันมาผสมรวมกันเป็นแอปพลิเคชันใหม่ก็ไม่ใช่เรื่องที่ทำได้ง่ายอยู่ดี Bit จึงเข้ามาช่วยได้ในเรื่องนี้ มันเองเป็นเครื่องมือบรรทัดคำสั่ง (cli) ที่สามารถแยกคอมโพเนนท์ใด ๆ ออกจากไลบรารีหรือโครงการต่าง ๆ ที่มีอยู่แล้วได้ คุณสามารถจะสร้างบริการขึ้นมาเองจาก Bit เพื่อให้คอมโพเนนท์ทำงานร่วมกัน หรือเลือกใช้บริการของ Bit.dev โดยตรงก็ได้

DataHub
ประเมิน

จากครั้งแรกที่เรดาร์ของเราได้พูดถึง ความสามารถในการสำรวจหาข้อมูล ไป นับตั้งแต่นั้นมาบริษัท LinkedIn ได้ค่อย ๆ พัฒนาปรับปรุงโครงการ WhereHows ของตนให้กลายเป็นแพลตฟอร์มที่มีความสามารถในการสำรวจหาข้อมูลสำเร็จ โดยทำจากระบบที่ข้อมูลเมทาดาต้าสามารถเพิ่มขยายได้ ปัจจุบันโครงการนี้มีชื่อใหม่ว่า DataHub ซึ่ง DataHub จะกำกับให้แต่ละคอมโพเนนท์ในระบบส่งข้อมูลเมทาดาต้าของตนไปยังแพลตฟอร์มส่วนกลางผ่าน API หรือการสตรีมเสมอ แทนการให้ส่วนกลางเป็นฝ่ายดึงหรือกวาดข้อมูลเมทาดาต้าไปใช้เอง วิธีการนี้ช่วยเปลี่ยนความเป็นเจ้าของข้อมูล จากทีมข้อมูลส่วนกลางไปให้แต่ละทีมที่ดูแลคอมโพเนนท์ใด ๆ รับผิดชอบต่อข้อมูลของตน จากที่หลาย ๆ บริษัทต่างต้องการจะเป็นองค์กรที่ขับเคลื่อนด้วยข้อมูลกันมากขึ้น การมีระบบที่สามารถสำรวจหาข้อมูล ทำความเข้าใจถึงคุณภาพและที่มาที่ไปของข้อมูล เป็นสิ่งจำเป็นอย่างมาก เราจึงอยากแนะนำให้คุณประเมิน DataHub จากความสามารถเหล่านั้น

Feature Store
ประเมิน

Feature Store เป็นดาต้าแพลตฟอร์มเฉพาะทาง ที่พยายามจะแก้ปัญหาสำคัญในกระบวนการฟีเจอร์เอนจิเนียริ่งในปัจจุบัน โดยมีแก่นความสามารถสำคัญอยู่สามอย่าง ได้แก่ (1) เมื่อมีข้อมูลใหม่เข้ามา จะสามารถนำข้อมูลไปวิ่งผ่านบริการดาต้าไปปไลน์ต่าง ๆ อย่างอัตโนมัติ ทั้งนี้เพื่อลดความยุ่งยากในการจัดการไปป์ไลน์ลง (2) สามารถบันทึกและจัดเก็บฟีเจอร์ข้อมูลเป็นหมวดหมู่ เพื่อส่งเสริมให้ฟีเจอร์ใด ๆ ในโมเดลหนึ่งเป็นที่รู้จักและนำไปใช้ข้ามโมเดลอื่นได้ (3) ให้บริการฟีเจอร์ข้อมูลที่สอดคล้องกันเสมอระหว่างการฝึกและการทดสอบการรบกวนของโมเดล

ตั้งแต่ Uber เปิดเผยถึง แพลตฟอร์ม Michelangelo ของตนออกมา องค์กรและสตาร์ทอัพต่างก็หันมาสร้างแพลตฟอร์มฟีเจอร์สโตร์ของตัวเองกันมากขึ้น เช่น Hopsworks, Feast และ Tecton เราเห็นถึงศักยภาพที่น่าสนใจของ Feature Store และแนะนำให้คุณประเมินมันอย่างรอบคอบก่อนนำไปใช้

JuiceFS
ประเมิน

JuiceFS เป็นฟอร์แมตไฟล์โอเพนซอร์สที่รองรับมาตรฐาน POSIX และการจัดเก็บข้อมูลแบบกระจายตัว ซึ่งมันอาศัยเทคโนโลยีอย่าง Redis และฐานข้อมูลแบบออบเจกต์เป็นกลไกสำคัญในการทำงาน ปกติแล้วหากใครจะขึ้นแอปพลิเคชันใหม่ เราจะแนะนำให้ใช้ฐานข้อมูลแบบออบเจกต์ตรง ๆ ในการเก็บข้อมูลโดยไม่ต้องมีเลเยอร์ใดมาคั่นกลาง อย่างไรก็ตาม JuiceFS เป็นตัวเลือกที่น่าสนใจ หากนำมาใช้ในกรณีที่ต้องการย้ายแอปพลิเคชันเก่า ๆ ไปยังคลาวด์ ที่ยังต้องพึ่งพิงฟอร์แมตไฟล์เดิม ๆ อย่าง POSIX อยู

การใช้งาน Kafka API โดยไม่ต้องใช้ Kafka
ประเมิน

เมื่อธุรกิจจำนวนมากหันไปพึ่งพาอีเวนท์เพื่อแบ่งปันข้อมูลระหว่างไมโครเซอร์วิส หรือใช้รวบรวมข้อมูลเพื่อการวิเคราะห์ หรือการส่งข้อมูลเข้าดาต้าเลค Apache Kafka จึงกลายเป็นแพลตฟอร์มยอดนิยมที่ผู้คนมักเลือกใช้เพื่อทำสถาปัตยกรรมที่ขับเคลื่อนด้วยอีเวนท์ (event-driven architecture) ถึงแม้ว่า Kafka จะปฏิวัติแนวคิดการส่งข้อความที่สามารถรองรับโหลดจำนวนมากได้ แต่ตัวมันเองก็มีส่วนประกอบหลายอย่าง ทั้ง ZooKeeper โบรกเกอร์ พาร์ทิชัน และมิเรอร์ แม้ดูเป็นเรื่องยุ่งยากในการดำเนินการและบริหารจัดการ แต่ว่ามันตอบโจทย์ด้านความยืดหยุ่นและให้ประสิทธิภาพที่ดีเมื่อต้องการใช้งาน เหมาะอย่างยิ่งกับระบบในองค์กรขนาดใหญ่ จากความยุ่งยากในการเริ่มใช้งานระบบนิเวศของมันทั้งหมดนั้น เรายินดีเมื่อได้เห็นแพลตฟอร์มใหม่ ๆ ที่นำ Kafka API มาใช้งานโดยไม่ต้องใช้ Kafka ทั้งแบบที่เป็นสถาปัตยกรรมทางเลือก เช่น Kafka on Pulsar และ Redpanda หรือแบบใช้ API ร่วมกับ Kafka ทั้งตัวอ่าน (consumer) และเขียน (producer) เช่น Azure Event Hubs for Kafka แต่คุณสมบัติบางอย่างของ Kafka เช่น สตรีมไคลเอนต์ไลบรารี ไม่สามารถใช้งานร่วมกับโบรกเกอร์ทางเลือกเหล่านี้ได้ จึงยังเป็นเหตุผลที่ดีในการใช้ Kafka ต่อไป อย่างไรก็ตาม เรายังต้องติดตามกันต่อไปว่านักพัฒนาใช้กลยุทธ์นี้จริง ๆ หรือเป็นเพียงความพยายามของคู่แข่งเพื่อชวนผู้ใช้มาใช้งานบริการของตน ไม่ว่าอย่างไรก็ตาม เราอาจจะมองได้ว่า ส่วนที่ดีที่สุดส่วนหนึ่งของ Kafka น่าจะเป็นโปรโตคอลและ API ที่ใช้งานได้อย่างสะดวก

NATS
ประเมิน

NATS เป็นระบบการจัดคิวข้อความที่รวดเร็ว ปลอดภัย มีคุณสมบัติที่หลากหลาย และมีความเป็นไปได้ในการติดตั้งมากมาย ตอนแรกคุณอาจจะสงสัยว่าทำไมโลกยังต้องการระบบคิวข้อความอื่นอีก ต้องบอกว่าเรามีระบบคิวข้อความมากมายหลากหลายรูปแบบตั้งแต่เริ่มมีการใช้คอมพิวเตอร์ทางธุรกิจและได้ผ่านการปรับแต่งและการเพิ่มประสิทธิภาพปีแล้วปีเล่าเพื่อให้รองรับงานต่าง ๆ แต่ NATS นั้นมีลักษณะที่น่าสนใจหลายประการ ความสามารถเฉพาะตัวในการรองรับการขยายขนาดตั้งแต่ใช้บนระบบเอ็มเบ็ดคอนโทรลเลอร์ไปจนถึงระบบซูเปอร์คลัสเตอร์บนคลาวด์ที่ใหญ่ระดับทั่วโลก เราให้ความสนใจเป็นพิเศษกับความตั้งใจของ NATS ที่จะรองรับการสตรีมข้อมูลจากอุปกรณ์มือถือ IoT และเครือข่ายของระบบต่าง ๆ ที่เชื่อมต่อกัน อย่างไรก็ตาม ปัญหายุ่งยากบางอย่างจำเป็นต้องได้รับการแก้ไข เช่นการทำให้ผู้บริโภคเห็นเฉพาะข้อความและหัวข้อที่พวกเขาได้รับอนุญาตให้เข้าถึง โดยเฉพาะอย่างยิ่งเมื่อใช้งานบนเครือข่ายครอบคลุมขอบเขตระดับองค์กร ซึ่งในจุดนี้ NATS 2.0 ได้เริ่มแนะนำเฟรมเวิร์คที่สามารถรักษาความปลอดภัยและสามารถควบคุมระดับการเข้าถึง ซึ่งความสามารถนี้สนับสนุนคลัสเตอร์แบบหลายบัญชี โดยที่แต่ละบัญชีจะจำกัดการเข้าถึงคิว (queue) และหัวข้อ (topics) NATS ถูกเขียนด้วยภาษา Go และได้รับการดูแลโดยชุมชนนักพัฒนาเป็นอย่างดี ถึงแม้จะมีไคลแอนต์ที่ถูกเขียนจากหลาย ๆ ภาษาที่ใช้กันอย่างแพร่หลาย แต่ไคลแอนต์ Go นั้นก็ได้รับความนิยมมากที่สุด อย่างไรก็ตามนักพัฒนาของเราบางคนพบว่าไลบรารีไคลแอนต์ภาษาทั้งหมดมีแนวโน้มที่จะสะท้อนถึงโค้ดต้นฉบับที่เขียนด้วย Go การเพิ่มแบนด์วิดธ์และพลังการประมวลผลบนอุปกรณ์ไร้สายขนาดเล็กที่เพิ่มขึ้นหมายความว่าปริมาณข้อมูลทางธุรกิจจะต้องใช้แบบเรียลไทม์จะเพิ่มขึ้นเป็นเงาตามตัว เราประเมิน NATS เป็นแพลตฟอร์มที่เป็นไปได้สำหรับการสตรีมข้อมูลทั้งภายในองค์กรและระหว่างธุรกิจ

แพลตฟอร์ม

เมื่อธุรกิจจำนวนมากหันไปพึ่งพาอีเวนท์เพื่อแบ่งปันข้อมูลระหว่างไมโครเซอร์วิส หรือใช้รวบรวมข้อมูลเพื่อการวิเคราะห์ หรือการส่งข้อมูลเข้าดาต้าเลค Apache Kafka จึงกลายเป็นแพลตฟอร์มยอดนิยม จากความยุ่งยากในการเริ่มใช้งานระบบนิเวศของมันทั้งหมดนั้น เรายินดีเมื่อได้เห็นแพลตฟอร์มใหม่ ๆ ที่นำ Kafka API มาใช้งานโดยไม่ต้องใช้ Kafka

(การใช้งาน Kafka API โดยไม่ต้องใช้ Kafka)

แพลตฟอร์ม

บางครั้งองค์กรต่าง ๆ ก็ชอบสร้างเฟรมเวิร์คหรือแอบแสตรกชันของตนขึ้นมาโดยนำผลิตภัณฑ์หรือเฟรมเวิร์คภายนอกมาครอบ และหวังว่าสิ่งนี้จะมีประโยชน์กว่าการใช้งานผลิตภัณฑ์ภายนอกเหล่านั้นตรง ๆ โดยพวกเขาไม่ได้คิดเผื่อเวลาเพื่อการบำรุงรักษาให้สิ่งนี้ทันสมัย ซึ่งไม่นานหลังจากนั้น พวกเขาก็ตระหนักได้เองว่าเครื่องมือที่ถูกครอบไว้อันที่จริงแล้วเก่งกว่าเครื่องมือที่ทำขึ้นเอง

(ผลิตภัณฑ์โครงสร้างพื้นฐานแบบโค้ด (IaC) ของตนเอง)

Opstrace

ประเมิน

Opstrace เป็นแพลตฟอร์มสำหรับงานสังเกตการณ์ (observability platform) แบบโอเพนซอร์ส ที่มีเจตนาทำขึ้นสำหรับติดตั้งภายในเน็ตเวิร์คขององค์กร บางหน่วยงานไม่อยากใช้โซลูชันเชิงพาณิชย์ที่มีอยู่ในตลาด อย่าง Datadog (ซึ่งอาจเป็นเพราะเหตุผลด้านค่าใช้จ่าย หรือเป็นห่วงความมั่นคงของข้อมูลก็ตาม) ซึ่งทางเลือกที่เป็นไปได้คือการทำเอง โดยนำเครื่องมือโอเพนซอร์สหลาย ๆ ตัวมาเรียงร้อยเข้าด้วยกัน ซึ่งก็ไม่ใช่งานน้อย ๆ เลย Opstrace เห็นถึงปัญหานี้เช่นกัน และพยายามจะทำให้เรื่องนี้สะดวกยิ่งขึ้น โดยมันใช้ API และการเชื่อมต่อแบบโอเพนซอร์สทั้งหมด กล่าวคือ มันนำ Prometheus และ Grafana มาต่อกันแล้วเพิ่มความสามารถบางอย่างลงไป เช่น ฟีเจอร์ด้านการยืนยันตัวตน หรือการรองรับมาตรฐาน TLS เป็นต้น ส่วนแกนหลักภายในของ Opstrace นั้นใช้คลัสเตอร์ของ Cortex เพื่อสร้าง API ของ Prometheus ที่รองรับการขยายตัวได้ และใช้งานคลัสเตอร์ของ Loki สำหรับการจดล็อคบันทึกเหตุการณ์ ทั้งนี้แพลตฟอร์มตัวนี้ยังค่อนข้างใหม่ และหากเทียบกับเครื่องมืออย่าง Datadog หรือ SignalFX มันยังขาดความสามารถบางอย่าง กระนั้นแล้ว มันก็เป็นเครื่องมือที่หน้าฝากความหวัง และน่าจับตามอง

Pulumi

ประเมิน

เราเห็นความสนใจใน Pulumi ค่อย ๆ โตขึ้นอย่างต่อเนื่อง Pulumi เป็นเครื่องมือที่มาเติมเต็มช่องว่างที่มีในตลาดการกำหนดโครงสร้างพื้นฐานด้วยโค้ด ที่ Terraform เป็นเจ้าตลาดอยู่ แม้ Terraform เป็นทางเลือกที่พิสูจน์ตัวเองแล้ว แต่ด้วยธรรมชาติการกำหนดเชิงประกาศของมัน ผู้ใช้จะต้องทนอยู่กับการไม่มีวิธีสร้างแอบแสตรกชันที่ดีพอ และการทดสอบที่ทำได้อย่างจำกัด ปกติแล้วการใช้ Terraform จะพอเพียงหากโครงสร้างพื้นฐานไม่ค่อยเปลี่ยนแปลง แต่ถ้าเปลี่ยนกันตลอดเช่นนั้น การจัดการด้วยภาษาโปรแกรมจริง ๆ จะเหมาะกว่า Pulumi แตกต่างที่การกำหนดสามารถเขียนด้วยภาษา TypeScript/JavaScript, Python และ Go ไม่ต้องใช้ภาษามาร์กอัพ (markup language) หรือเทมเพลตใด ๆ มันมุ่งเน้นไปที่การจัดการสถาปัตยกรรมแบบคลาวด์เนทีฟ รองรับคอนเทนเนอร์ เซิร์ฟเวอร์เลสฟังก์ชัน และบริการด้านข้อมูล ของแต่ละผู้ให้บริการ อีกทั้งยังรองรับ Kubernetes ได้เป็นอย่างดี แม้ว่าเร็ว ๆ นี้ AWS CDK จะเข้ามาท้าชิงตลาด แต่ Pulami ยังคงเป็นเครื่องมือเดียวที่ไม่ขึ้นกับผู้ให้บริการรายใด เราคาดว่า Pulumi จะเป็นที่นิยมกว่านี้ และหวังที่จะได้เห็นเครื่องมือและความรู้ใหม่เกิดขึ้นรอบ ๆ เป็นระบบนิเวศสนับสนุนกันและกัน

Redpanda

ประเมิน

Redpanda เป็นแพลตฟอร์มสตรีมมิ่งที่มี API ที่เหมือนกับของ Kafka ซึ่งช่วยให้เราได้รับประโยชน์จากระบบนิเวศของ Kafka โดยไม่ต้องลำบากกับความซับซ้อนของการติดตั้ง Kafka เช่น การดูแลจัดการ Redpanda นั้นง่ายขึ้น เพราะระบบมีเพียงไบนารีเพียงไฟล์เดียว เราไม่จำเป็นต้องพึ่งพาบริการภายนอกที่ซับซ้อนดูแลลำบากอย่าง ZooKeeper เพราะ Redpanda เลือกใช้โปรโตคอล Raft ในการจัดการความคงเส้นคงวาของข้อมูล และยังมีการใช้ชุดทดสอบที่ครอบคลุมเพื่อตรวจสอบว่า Raft ทำงานได้อย่างถูกต้องแทน หนึ่งในความสามารถของ Redpanda (พร้อมใช้งานสำหรับลูกค้าองค์กรเท่านั้น) คือความสามารถในการแปลงข้อมูลในสตรีมด้วย WebAssembly (WASM) โดยใช้เอ็นจิน WASM ที่ฝังไว้ สิ่งนี้ช่วยให้นักพัฒนาสามารถสร้างตัวแปลงข้อมูลในภาษาใดก็ได้ที่สามารถคอมไพล์เป็น WASM นอกจากนี้ Redpanda ยังลดการหน่วงเวลาลงได้มากและเพิ่มอัตราการประมวลผลข้อมูลมากขึ้นเพราะมีการพัฒนาประสิทธิภาพอย่างต่อเนื่อง Redpanda เป็นทางเลือกที่น่าตื่นเต้นสำหรับการใช้งานร่วมกับ Kafka และควรค่าแก่การพิจารณา

Azure Machine Learning

เฝ้าระวัง

เราพบว่าผู้ให้บริการคลาวด์ทั้งหลาย ต่างพากันผลักดันบริการใหม่ ๆ เข้าสู่ตลาดมากขึ้นอย่างต่อเนื่อง เราเคยกล่าวถึงข้อกังวลนี้ไปแล้วว่า บางครั้งบางบริการก็ถูกเปิดใช้งานทั้งที่ยังขาดความพร้อมอยู่มาก จากประสบการณ์ของเรา เป็นที่น่าเสียดายว่า Azure Machine Learning เป็นแพลตฟอร์มหนึ่งที่เข้าข่ายกรณีนี้ Azure ML เป็นหนึ่งในหลายเทคโนโลยีใหม่ ในกลุ่มแพลตฟอร์มเขียนโค้ดน้อยเฉพาะทางที่สัญญาว่าจะทำให้นักวิทยาศาสตร์ข้อมูลทำงานได้สะดวกขึ้น แต่ที่สุดแล้วมันดูจะไม่เป็นเช่นนั้นเท่าไหร่ นักวิทยาศาสตร์ข้อมูลของเรายังรู้สึกว่าการใช้ Python นั้นยังง่ายกว่า นอกจากนี้เราพบว่าการรองรับการขยายของระบบก็ทำได้อย่างยากลำบาก แม้เราจะพยายามอย่างถึงที่สุดแล้ว และการขาดเอกสารที่เพียงพอเป็นอีกจุดหนึ่งที่ทำให้เราตัดสินใจย้าย Azure ML ไปยังวงแหวน เฝ้าระวัง

เครื่องมือโครงสร้างพื้นฐานด้วยโค้ด (IaC) ฉบับทำเอง

เฝ้าระวัง

ผลิตภัณฑ์ที่ดี หากได้รับการสนับสนุนจากหน่วยงานหรือชุมชน ย่อมไม่หยุดนิ่งและจะต้องปรับปรุงให้ทันสมัยอยู่เรื่อย ๆ บางครั้งองค์กรต่าง ๆ ก็ชอบสร้างเฟรมเวิร์คหรือแอบแสตรกชันของตนขึ้นมา โดยนำผลิตภัณฑ์หรือเฟรมเวิร์คภายนอกมาครอบ และหวังว่าสิ่งนี้จะมีประโยชน์กว่าการใช้งานผลิตภัณฑ์ภายนอกเหล่านั้นตรง ๆ หลายที่ได้สร้าง เครื่องมือโครงสร้างพื้นฐานด้วยโค้ด (IaC) ฉบับทำเอง ขึ้นมา โดยพวกเขาไม่ได้คิดเผื่อเวลาเพื่อการบำรุงรักษาให้สิ่งนี้ทันสมัย ซึ่งไม่นานหลังจากนั้น พวกเขาก็ตระหนักได้เองว่าเครื่องมือที่ถูกครอบไว้อันที่จริงแล้วเก่งกว่าเครื่องมือที่ทำขึ้นเอง บางอันที่แย่หน่อยก็ไปลดทอนความสามารถที่ต้นฉบับมีด้วยซ้ำ เอาละ แม้จะมีบางองค์กรที่ประสบความสำเร็จกับการสร้างผลิตภัณฑ์ขึ้นมาใช้เองอยู่บ้าง เราก็อยากเตือนถึงเรื่องนี้อยู่ดี ว่ามันต้องใช้เวลาและความพยายามไม่น้อย และต้องการวิสัยทัศน์ในระยะยาวที่ดี หากจะทำผลิตภัณฑ์ที่ทำออกมาได้ดีกว่า

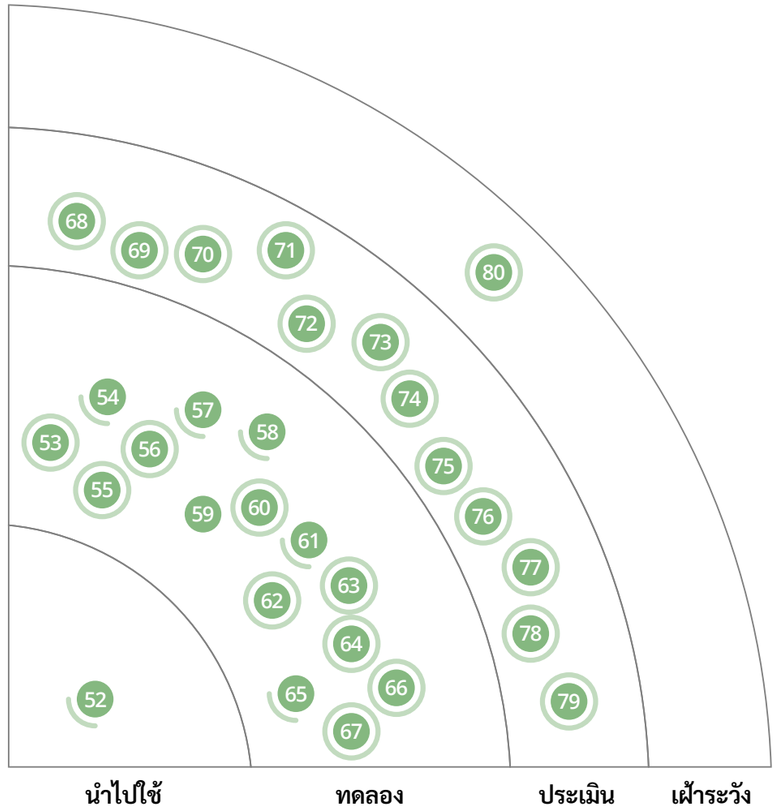
TECHNOLOGY RADAR

เครื่องมือ

เครื่องมือ

Sentry
นำไปใช้

Sentry ได้กลายเป็นตัวเลือกที่หลายทีมของเรานึกถึงก่อนเสมอ ในฐานะเครื่องมือสำหรับรายงานข้อผิดพลาดของแอปพลิเคชันฟร็อนเอนด์ การมีความสามารถอย่างการจัดกลุ่มข้อผิดพลาดไว้เป็นหมวดหมู่ หรือความสามารถกำหนดรูปแบบข้อผิดพลาดที่ไม่ได้สนใจไว้ได้ ก็ช่วยได้มากในสถานการณ์ที่มีรายงานข้อผิดพลาดจากผู้ใช้ทะลักเข้ามาอย่างมหาศาล และหากรวม Sentry ให้เป็นส่วนหนึ่งของไปป์ไลน์ก็มีข้อดีเช่นกัน เพราะจะสามารถแนบซอร์สแมพมาด้วย ซึ่งจะช่วยให้การดีบักหาข้อผิดพลาดมีประสิทธิภาพมากยิ่งขึ้น ทั้งนี้ยังช่วยทวนสอบหาว่าข้อผิดพลาดเกิดขึ้นที่เวอร์ชันไหนของซอฟต์แวร์ นอกจากนี้เรายังชื่นชม Sentry ที่แม้จะมีธุรกิจหลักเป็นบริการทางด้านซอฟต์แวร์ (SaaS) ก็จริง แต่ก็ยังเลือกที่จะเปิดเผยซอร์สโค้ดต่อสาธารณะ และอนุญาตให้นำไปใช้งานได้ฟรีสำหรับการใช้ในงานขนาดเล็ก หรือใช้กับ โฮสติ้งของตนเอง



axe-core
ทดลอง

การทำเว็บไซต์ให้รองรับการใช้งานของทุกกลุ่มคนนั้นต้องการความใส่ใจอย่างจริงจัง ในทุกขั้นตอนการส่งมอบจะต้องมีการพิจารณาและตรวจสอบเรื่องนี้อยู่ตลอดทั้งสาย แต่เครื่องมือทดสอบการเข้าถึงที่มีชื่อเสียงหลายตัวก็ถูกออกแบบมาให้ทดสอบกับเว็บไซต์ที่เสร็จสมบูรณ์แล้วเป็นส่วนใหญ่ ซึ่งกว่าจะพบปัญหาบางทีก็สายไปแล้ว การแก้ไขก็ทำได้ยาก แล้วในที่สุดก็ถูกทิ้งไว้ต่อไป เมื่อไม่นานมานี้ เราได้นำเครื่องมือทดสอบด้านการเข้าถึง อย่าง axe-core มาใช้ในกระบวนการสร้างเว็บไซต์ภายในของธอทเวิร์คเองโดย axe-core เป็นเอ็นจิ้นสำหรับทดสอบความสามารถในการเข้าถึงที่ทำงานในขั้นตอนการบิลด์เลย ทีมพัฒนาของเราจึงสามารถปรับแก้ให้ถูกต้องตามกฎต่าง ๆ ได้อย่างรวดเร็ว ตั้งแต่เว็บไซต์ยังอยู่ระหว่างการพัฒนา ทั้งนี้เครื่องมืออัตโนมัติประเภทนี้ก็ไม่ได้สมบูรณ์แบบ เพราะบางปัญหาก็ไม่สามารถตรวจจับได้ นอกจากนั้นยังมีเครื่องมืออื่น ๆ ในรุ่นเชิงพาณิชย์ให้ใช้ผ่าน axe DevTools ซึ่งจะแนะแนวทางการทดสอบการเข้าถึงให้กับทีมเมื่อเจอกับปัญหาส่วนใหญ่

dbt
ทดลอง

นับแต่ที่เราได้กล่าวถึง dbt ในเรดาร์ก่อนหน้านี้ เราได้ใช้มันในบางโครงการและรู้สึกประทับใจในสิ่งที่เห็น เช่น ผู้ใช้ข้อมูลสามารถเข้าถึงส่วนของการปรับเปลี่ยนรูปแบบข้อมูลในไปป์ไลน์ ELT ได้ง่ายกว่าเมื่อเทียบกับไปป์ไลน์ทั่วไปที่ถูกสร้างโดยวิศวกรข้อมูล ในขณะเดียวกันก็ส่งเสริมแนวปฏิบัติทางวิศวกรรมที่ดี เช่น การกำหนดเวอร์ชัน การทดสอบอัตโนมัติ และการติดตั้งอัตโนมัติ จนถึงตอนนี้ SQL ยังคงเป็นภาษากลาง (Lingua Franca) ของโลกข้อมูล (รวมไปถึงฐานข้อมูล คลังข้อมูล เครื่องมือสืบค้นข้อมูล ดาต้าเลค และแพลตฟอร์มการวิเคราะห์) และระบบข้อมูลส่วนใหญ่นั้นรองรับ SQL ในระดับหนึ่ง ทำให้นักพัฒนาสามารถใช้ dbt กับระบบข้อมูลเหล่านี้เพื่อปรับเปลี่ยนรูปแบบข้อมูลได้โดยง่ายเพียงแค่สร้างแดปเตอร์เท่านั้น มีการเติบโตขึ้นของจำนวนตัวเชื่อมต่อแบบเนทีฟ ซึ่งรวมถึง Snowflake BigQuery Redshift และ Postgres นอกจากนี้ยังมีปลั๊กอินชุมชน จำนวนมากให้ใช้งานอีกด้วย เราเล็งเห็นว่าเครื่องมือ

นำไปใช้
52. Sentry

ทดลอง
53. axe-core
54. dbt
55. esbuild
56. Flipper
57. Great Expectations
58. k6
59. MLflow
60. OR-Tools
61. Playwright
62. Prowler
63. Pyright
64. Redash
65. Terratest
66. Tuple
67. Why Did You Render

ประเมิน
68. Buildah และ Podman
69. GitHub Actions
70. Graal Native Image
71. HashiCorp Boundary
72. imgcook
73. Longhorn
74. Operator Framework
75. Recommender
76. Remote - WSL
77. Spectral
78. Yelp detect-secrets
79. Zally

เฝ้าระวัง
80. AWS CodePipeline

เครื่องมือ

OR-Tools เป็นชุดซอฟต์แวร์โอเพ่นซอร์สสำหรับแก้ปัญหาการหาค่าเหมาะสมที่สุดเชิงการจัด (combinatorial optimization problem) โดยปัญหาประเภทนี้จะมีวิธีการแก้ปัญหาที่เป็นไปได้จำนวนมาก เครื่องมือแบบนี้จึงมีประโยชน์มากในการเสาะหาวิธีที่คุ้มค่าที่สุด

(OR-Tools)

เช่น dbt จะช่วยให้แพลตฟอร์มข้อมูลมีความเป็นแพลตฟอร์ม “บริการตนเอง” มากขึ้น

esbuild

ทดลอง

เรามองหาเครื่องมือที่สามารถร่นฟีดแบคของการพัฒนาซอฟต์แวร์แต่ละรอบให้เร็วขึ้นอยู่เสมอ และ esbuild ก็เป็นหนึ่งในตัวอย่างที่ดีของเรื่องนี้ เมื่อไหร่ก็ตามที่โค้ดฟรอนท์เอนด์มีขนาดใหญ่ขึ้น การสร้างแพ็คเกจแต่ละครั้งก็อาจใช้เวลานานเป็นนาที แต่ตัว esbuild นั้นปรับแต่งมาเพื่อเพิ่มความเร็วเรื่องนี้โดยเฉพาะ จึงสามารถลดเวลาการสร้างแพ็คเกจลงไปได้ตั้งแต่ 10 ถึง 100 เท่าเลยทีเดียว esbuild ถูกเขียนด้วยภาษา Golang และใช้วิธีการที่มีประสิทธิภาพกว่าในกระบวนการพาร์ซซิ่ง การปรินต์ และการทำซอร์ซแมป จึงสามารถสร้างแพ็คเกจได้เร็วกว่าเครื่องมือช่วยบิวด์อื่น อย่าง Webpack หรือ Parcel หลาย ๆ ทีมของเราได้เปลี่ยนไปใช้ esbuild เป็นตัวเลือกหลักแล้ว แม้ว่ามันจะมีตัวแปลงไวยกรณ์ภาษา JavaScript ไม่ครอบคลุมเท่ากับเครื่องมืออื่นก็ตาม

Flipper

ทดลอง

Flipper เป็นดีบักเกอร์สำหรับดีบักแอปพลิเคชันมือถือที่สามารถพัฒนาต่อยอดได้ มันรองรับการวิเคราะห์การใช้ทรัพยากร การตรวจสอบเค้าโครงแบบโต้ตอบได้ ตัวแสดงล็อกบันทึกเหตุการณ์ และตัวสำรวจการทำงานของเครือข่ายสำหรับแอปพลิเคชัน iOS, Android และ React Native เมื่อเทียบกับดีบักเกอร์สำหรับแอปพลิเคชันมือถือเจ้าอื่น ๆ เราพบว่า Flipper มีขนาดเล็กแต่มีคุณสมบัติที่หลากหลายและติดตั้งได้ง่าย

Great Expectations

ทดลอง

เราได้กล่าวถึง Great Expectations ไว้ในเรดาร์ฉบับที่แล้ว ซึ่งในฉบับนี้เราก็ยังชื่นชอบมันอยู่จนตัดสินใจย้ายเครื่องมือนี้เข้ามาในหมวดหมู่ทดลองใช้งาน ตัว Great Expectations เป็นเฟรมเวิร์คที่ช่วยให้คุณสามารถสร้างตัวควบคุมต่าง ๆ ขึ้นมา เพื่อใช้ตรวจจับความผิดปกติหรือปัญหาคุณภาพในไปป์ไลน์ข้อมูล นอกจากนั้นมันยังทำหน้าที่ตรวจสอบข้อมูลในไปป์ไลน์ข้อมูลในระหว่างที่ข้อมูลทำงานอยู่ เสมือนเป็นยูนิตเทสต์ที่ทำงานในบิลด์ไปป์ไลน์ เราชอบความเรียบง่ายและการใช้งานที่สะดวกของมัน อันเนื่องจากกฎระเบียบที่ตั้งไว้ จะถูกบันทึกเก็บไว้ในรูปแบบไฟล์ JSON ซึ่งผู้เชี่ยวชาญด้านโดเมนข้อมูลใด ๆ สามารถแก้ไขได้โดยตรง ไม่จำเป็นต้องมีทักษะด้านวิศวกรรมข้อมูลแต่อย่างใด

k6

ทดลอง

ตั้งแต่ที่เราได้กล่าวถึง k6 เป็นครั้งแรกในเรดาร์ฉบับก่อน เรามีประสบการณ์เพิ่มมากขึ้นอีกเล็กน้อยในการใช้งานมันเพื่อทดสอบประสิทธิภาพ และได้ผลลัพธ์ที่ดีอีกด้วย ทีมของเราพอใจที่เครื่องมือเน้นประสบการณ์การใช้งานของนักพัฒนา และมีความยืดหยุ่น แม้ว่าจะเป็นเรื่องง่ายที่จะเริ่มใช้งาน k6 ทั้งหมดด้วยตัวเอง แต่จุดที่โดดเด่นยิ่งกว่านั้นคือความเรียบง่ายในการต่อรวมกับระบบนิเวศผู้พัฒนา ตัวอย่างเช่น การใช้งานร่วมกับ Datadog adapter ที่ทีมหนึ่งสามารถแสดงผลประสิทธิภาพของระบบกระจายตัวและสามารถระบุจุดน่ากังวลที่สำคัญ ก่อนที่จะเอาระบบขึ้นโปรดักชัน ส่วนอีกทีมหนึ่งที่ใช้งาน k6 เวอร์ชันพาณิชย์สามารถใช้ Azure pipelines marketplace extension เพื่อนำการทดสอบประสิทธิภาพมารวมในไปป์ไลน์การส่งมอบอย่างต่อเนื่องของทีม และสร้างรายงานของ Azure DevOps ได้โดยใช้ระยะเวลาเพียงเล็กน้อยเท่านั้น และตั้งแต่ที่ k6 สามารถตรวจสอบผลลัพธ์เทียบกับเกณฑ์ได้โดยอัตโนมัติ มันเลยง่ายมากที่จะเพิ่มขั้นตอนนี้บนไปป์ไลน์เพื่อตรวจจับประสิทธิภาพที่ลดลงจากการเปลี่ยนแปลงที่เราพัฒนาเพิ่มในระบบได้ทันที เกิดกลไกการป้อนกลับที่ทรงพลังสำหรับนักพัฒนา

MLflow

ทดลอง

MLflow เป็นเครื่องมือโอเพนซอร์สสำหรับ ติดตามการทดลองของแมชชีนเลิร์นนิง และการจัดการโมเดลแมชชีนเลิร์นนิงตลอดทั้งกระบวนการ การจะพัฒนาและวิวัฒน์โมเดลแมชชีนเลิร์นนิงประกอบไปด้วยขั้นตอนต่าง ๆ ตั้งแต่เริ่มทำการทดลอง ติดตามผลของการทดลอง และคอยติดตามและปรับแต่งประสิทธิภาพของโมเดล ซึ่ง MLFLow ช่วยอำนวยความสะดวกตลอดทั้งกระบวนการ ผ่านการทำงานร่วมกับมาตรฐานเปิดต่าง ๆ และการเชื่อมต่อกับเครื่องมืออื่น ๆ ในระบบนิเวศ นอกจากนี้ บริการ MLFlow บนคลาวด์ที่บริหารโดย Databricks ก็มีและรองรับทั้ง AWS และ Azure ซึ่งบริการนี้สมบูรณ์ขึ้นมากในเวลาอันรวดเร็วจากที่เราได้ใช้ก็ให้ผลดี มันเป็นเครื่องมือที่ดีทีเดียวสำหรับจัดการและติดตามโมเดล และรองรับวิธีการใช้งานผ่านทั้ง API และ UI อย่างไรก็ตาม ความกังวลเดียวของเราที่มี คือดูเหมือน MLflow พยายามจะเป็นทุกอย่างมากเกินไปในแพลตฟอร์มเดียวกัน เช่น มันสามารถให้บริการและให้คะแนนโมเดลได้อีกด้วย

OR-Tools

ทดลอง

OR-Tools เป็นชุดซอฟต์แวร์โอเพ่นซอร์สสำหรับแก้ปัญหาการหาค่าเหมาะสมที่สุดเชิงการจัด (combinatorial optimization problem) โดยปัญหาประเภทนี้จะมีวิธีการแก้ปัญหาที่เป็นไปได้จำนวนมากแต่อาจไม่คุ้มค่าที่สุด การใช้เครื่องมืออย่าง OR-Tools จึงมีประโยชน์มากในการเสาะหาวิธีที่คุ้มค่าที่สุดนี้ คุณสามารถเลือกโมเดลโจทย์ขึ้นมาด้วยภาษาใดภาษาหนึ่งที่ OR-Tools รองรับ ซึ่งได้แก่ Python, Java, C# หรือ C++ แล้วจึงเลือกตัวแก้โจทย์ในขั้นตอนถัดไป ซึ่งมีทั้งแบบโอเพนซอร์สและแบบเชิงพาณิชย์ให้เลือกใช้ เราประสบความสำเร็จในการใช้ OR-Tools กับหลายโครงการด้วยกัน ทั้งกับรูปแบบการกำหนดการจำนวนเต็มธรรมดาและแบบผสม (mixed-integer programming)

Playwright

ทดลอง

Playwright เป็นเครื่องมือช่วยทดสอบ UI ของเว็บ ที่ API ของมันสามารถเข้าควบคุมเบราว์เซอร์สำคัญ ๆ ที่อยู่ต่างค่ายกันได้ทั้งหมด ทั้ง Chromium, Firefox และ Webkit ที่มันทำเช่นนี้ได้ เป็นเพราะมันไปใช้เบราว์เซอร์ Firefox และ Webkit รุ่นดัดแปลงที่แนบมาด้วยกับตัวไลบรารี จากความสามารถดังกล่าวนี้ ทำให้ Playwright กลายเป็นเครื่องมือที่น่าจับตามองมากขึ้น เรายังคงได้รับรายงานความคิดเห็นเชิงบวกกับ Playwright อยู่เสมอ โดยเฉพาะในด้านความเสถียร ทีมของเรายังพบว่าการย้ายจาก Puppeteer ไปเป็น Playwright เป็นเรื่องไม่ยากเนื่องจากทั้งคู่มี API ที่คล้ายกันมาก

Prowler

ทดลอง

เรายินดีเสมอเมื่อได้เห็นเครื่องมือที่ทำหน้าที่ตรวจสอบความถูกต้องของการตั้งค่าระบบโครงสร้างพื้นฐาน (infrastructure configuration scanning) เพิ่มมากขึ้นหรือมีพัฒนาการมากขึ้นในตลาด ซึ่ง Prowler คือหนึ่งในเครื่องมือกลุ่มนี้ ที่จะช่วยให้การตั้งค่าโครงสร้างพื้นฐานบน AWS มีความมั่นคงยิ่งขึ้นหากทีมพิจารณาปรับใช้คำแนะนำที่ได้จากการตรวจสอบ แม้ว่าโครงการ Prowler จะมีมาสักพักใหญ่ แต่ในช่วงหลายปีนี้มันได้ถูกพัฒนาไปอย่างมาก จากการใช้งานมาเราพบว่าเครื่องมือนี้มีประโยชน์มาก เพราะช่วยให้ทีมต่าง ๆ สามารถรับผิดชอบเรื่องความมั่นคงด้วยตัวเองโดยไม่ต้องเสียเวลามากเพื่อรอผลลัพธ์ในแต่ละรอบ Prowler แบ่งเกณฑ์มาตรฐาน CIS ของ AWS ออกเป็นหมวดหมู่ย่อย ๆ (IAM, การบันทึกเหตุการณ์, การเฝ้าสังเกตการณ์, เน็ตเวิร์ก, CIS ระดับ 1, CIS ระดับ 2, EKS-CIS) แล้วยังมีการตรวจสอบอื่น ๆ ที่ช่วยให้ได้ข้อมูลเชิงลึกในด้านมาตรฐานความปลอดภัยของข้อมูลบัตรชำระเงิน (PCI DSS) และระเบียบการคุ้มครองข้อมูลทั่วไปแห่งสหภาพยุโรป (GDPR) อีกด้วย

Pyright

ทดลอง

แม้ว่าการทำ duck typing จะถูกมองว่าเป็นฟีเจอร์ของภาษาในสายตาของโปรแกรมเมอร์หลายคนที่เขียน Python ก็ตาม แต่ในบางครั้งโดยเฉพาะกับโค้ดเบสขนาดใหญ่ การมีเครื่องมือตรวจสอบประเภทไทป์ (type checking) ก็มีประโยชน์ไม่น้อย ด้วยเหตุนี้จึงมีใครหลายคนพยายามจะนำเสนออะโนเทชัน (annotation) เพื่อระบุไทป์ขึ้นมาผ่าน Python Enchancement Proposal (PEPs) และ Pyright เป็นเครื่องมือตรวจสอบไทป์ที่ทำงานร่วมกับอะโนเทชันเหล่านั้น นอกจากนี้ Pyright มีความสามารถในการอนุมานประเภทไทป์ (type inference) และการสร้างการ์ด (guard) ป้องกันการใช้ไทป์ผิดพลาด โดยมันเข้าใจโครงสร้างโค้ดเมื่ออยู่ภายใต้เงื่อนไขที่ต่างกันไป มันถูกออกแบบโดยคำนึงถึงการทำงานร่วมกับโค้ดเบสขนาดใหญ่ และทำงานได้เร็ว อีกทั้งยังมีโหมดที่คอยติดตามไฟล์ที่เปลี่ยนแปลง (watch mode) เพื่อตรวจสอบเฉพาะกับไฟล์นั้นเท่านั้น จึงช่วยทำให้การตรวจสอบทำได้รวดเร็วยิ่งขึ้น ผู้ใช้สามารถใช้ Pyright โดยตรงผ่านบรรทัดคำสั่ง หรือใช้ผ่านเครื่องมือพัฒนาอย่าง VS Code, Emacs, vim, Sublime ฯลฯ จากการใช้งานของเรา เราชอบ Pyright มากกว่าเครื่องมืออื่นที่คล้าย ๆ กันนี้อย่าง mypy

Redash

ทดลอง

การที่ทีมนำแนวคิดของ DevOps ที่ว่าด้วย “คุณสร้างมันคุณก็ใช้มัน” มาใช้ หมายความว่าทีมได้ให้ความสนใจในตัวชี้วัดทางเทคนิคและธุรกิจจากระบบที่พวกเขาติดตั้ง บ่อยครั้งที่เราพบว่าเครื่องมือวิเคราะห์นั้นเข้าถึงได้ยากสำหรับนักพัฒนาส่วนใหญ่ดังนั้นงานในการตรวจจับและนำเสนอตัวชี้วัดจึงถูกทิ้งไว้ให้กับทีมอื่น ๆ ภายหลังจากส่งมอบฟีเจอร์ไปยังผู้ใช้แล้ว ทีมงานของเราพบว่า Redash มีประโยชน์อย่างมากในการค้นหาเมตริกสำหรับผลิตภัณฑ์และการสร้างแดชบอร์ด ในลักษณะที่นักพัฒนาทั่วไปสามารถทำได้ด้วยตนเอง ส่งผลให้ทีมสามารถร่นเวลารอผลลัพธ์ในแต่ละรอบและมุ่งเน้นกับผลลัพธ์ทางธุรกิจ

เครื่องมือ

Tuple เป็นเครื่องมือที่ค่อนข้างใหม่ ที่ออกแบบมาเพื่อให้การแพร์โปรแกรมมิ่งระยะไกลนั้นสะดวกยิ่งขึ้น ซึ่งออกแบบมาเพื่อปิดช่องว่างในตลาด หลังจากที่ Slack เลิกพัฒนา Screenhero ต่อ

(Tuple)

เครื่องมือ

imgcook เป็นบริการ SaaS ของอาลีบาบาที่สามารถแปลงไฟล์ภาพการออกแบบอย่าง ไฟล์ Sketch, PSD, หรือไฟล์ภาพทั่วไป ไปเป็นโค้ดฟรอนท์เอนด์ได้อย่างชาญฉลาด

(imgcook)

Terratest

ทดลอง

เราจับตาดู Terratest ในฐานะเครื่องมือทางเลือกสำหรับทดสอบโครงสร้างพื้นฐานมาสักระยะหนึ่งแล้ว มีหลาย ๆ ทีมของเราที่หันมาใช้งานมันมากขึ้น แล้วก็ชื่นชอบในการใช้งานและความเสถียรที่ได้ Terratest เป็นไลบรารีในภาษา Golang ที่ช่วยให้เราสามารถเขียนชุดทดสอบโค้ดโครงสร้างพื้นฐานแบบอัตโนมัติได้ง่ายขึ้น เราสามารถใช้เครื่องมือกำหนดโครงสร้างพื้นฐานด้วยโค้ดอย่าง Terraform ในการสร้างโครงสร้างพื้นฐานของจริงขึ้นมา (ยกตัวอย่างเช่น เซิร์ฟเวอร์ ไฟร์วอลล์ หรือเครื่องมือโหลดบาลานซ์) แล้วลองติดตั้งแอปพลิเคชันของเราลงไป จากนั้นเราสามารถใช้ Terratest ตรวจสอบพฤติกรรมของโครงสร้างพื้นฐานเพื่อยืนยันความถูกต้อง เมื่อสิ้นสุดการทดสอบ Terratest ยังช่วยถอดถอนแอปพลิเคชันที่ติดตั้งลงไป หรือทำลายโครงสร้างพื้นฐานที่ถูกสร้างขึ้นระหว่างการทดสอบให้ด้วย ซึ่งมีประโยชน์อย่างมากในการทดสอบโค้ดโครงสร้างพื้นฐานในสภาพแวดล้อมของจริง

Tuple

ทดลอง

Tuple เป็นเครื่องมือที่ค่อนข้างใหม่ ที่ออกแบบมาเพื่อให้การแพร์โปรแกรมมิ่งระยะไกลนั้นสะดวกยิ่งขึ้น ซึ่งออกแบบมาเพื่อปิดช่องว่างในตลาด หลังจากที่ Slack เลิกพัฒนา Screenhero ต่อ แม้จะมีข้อจำกัดในการใช้งานอยู่บ้าง เนื่องจากมันยังรองรับแค่แพลตฟอร์ม Mac OS (จะสนับสนุน Linux เร็ว ๆ นี้) และ UI ที่ใช้งานแปลก ๆ ที่ยังต้องแก้ต่อไป แต่ภายใต้ข้อจำกัดเหล่านั้นเรายังได้รับประสบการณ์การใช้งานที่ดีอยู่ ตัว Tuple เองแตกต่างจากเครื่องมือแบ่งปันวิดีโอและหน้าจอทั่วไป อย่าง Zoom ตรงที่มันรองรับการควบคุมหน้าจอร่วมกันผ่านเคอร์เซอร์เมาส์สองตัว และยังแตกต่างจากตัวเลือกอย่าง Visual Studio Live Share ตรงที่มันไม่ยึดติดกับ IDE ใด ๆ นอกจากนี้ Tuple ยังรองรับการสื่อสารด้วยเสียงและวิดีโอ การแชร์คลิปบอร์ดระหว่างกัน มีการตอบสนองที่ดี ไม่หน่วงเท่ากับเครื่องมือทั่วไป อีกทั้งความสามารถในการวาดและลบภาพในหน้าจอของคู่แพร์ของคุณ ยิ่งทำให้ Tuple เป็นเครื่องมือที่น่าใช้งานและเป็นมิตรที่ดีกับนักพัฒนา

Why Did You Render

ทดลอง

เวลาที่เราใช้ React เรามักจะเจอสถานการณ์ที่บางหน้าเว็บเพจแสดงผลได้ช้ากว่าปกติ เพราะบางคอมโพเนนต์ถูกวาดใหม่ซ้ำ ๆ เกินความจำเป็น Why Did You Render เป็นไลบรารีที่ทำงานโดยการเข้าไปแก้ไขโค้ด React บางส่วน ทำให้เราสามารถตรวจสอบสาเหตุการวาดใหม่ของคอมโพเนนต์ได้ เราได้ใช้ไลบรารีนี้กับบางโครงการเพื่อหาต้นตอของปัญหาทางด้านประสิทธิภาพ แล้วมันช่วยแก้ปัญหานี้ได้เป็นอย่างดีเลยทีเดียว

Buildah และ Podman

ประเมิน

แม้ว่า Docker ได้กลายเป็นตัวเลือกหลักที่เราใช้เมื่อต้องทำงานร่วมกับเทคโนโลยีคอนเทนเนอร์ แต่เราก็เห็นผู้เล่นหน้าใหม่ที่น่าสนใจเข้ามาตลอด ซึ่ง Buildah และ Podman ก็อยู่ในกลุ่มนี้ ทั้งคู่เป็นโครงการที่เติมเต็มซึ่งกันและกันเองได้อย่างลงตัว กล่าวคือ เราสามารถใช้ Buildah เพื่อสร้างอิมเมจ และใช้ Podman ในการรันคอนเทนเนอร์ บนดิสโทรลีนุกซ์หลากหลายเจ้า โดยไม่ต้องการสิทธิ์รูท แต่อย่างใด นอกจากนี้ Podman ได้นำเสนอเอนจิ้นในการรันคอนเทนเนอร์ชนิดที่ไม่ต้องรันโปรเซสเดมอนทิ้งไว้ ซึ่งเป็นวิธีที่น่าสนใจเมื่อเปรียบเทียบกับ Docker แล้วการที่ Podman สามารถใช้ได้ทั้งอิมเมจประเภท Open Container Initiative (OCI) ที่สร้างโดย Buildah หรือใช้อิมเมจของ Docker ยิ่งทำให้เครื่องมือนี้ง่ายและน่าสนใจกว่าเดิม

GitHub Actions

ประเมิน

เซิร์ฟเวอร์ CI และเครื่องมือบิลด์เป็นสิ่งที่เก่าแก่ที่สุดและใช้กันอย่างแพร่หลายในชุดเครื่องมือของเรา พวกมันถูกรันอยู่บนบริการคลาวน์โฮสต์ดิ้งรูปแบบง่าย ๆ จนไปถึงรูปแบบที่ซับซ้อนอย่างไปป์ไลน์เซิฟเวอร์ที่มีชุดโค้ดกำหนดซึ่งรองรับการใช้บิลด์แมชชีนจำนวนมาก จากประสบการณ์ของเราและตัวเลือกที่หลากหลายที่มีอยู่ในตลาด เราเริ่มสงสัยเมื่อ Github Actions ถูกนำมาใช้เป็นกลไกในการจัดการเวิร์กโฟลว์สำหรับการสร้างและการรวมระบบ การที่ Github Actions เพิ่มโอกาสสำหรับนักพัฒนาในการเริ่มต้นการพัฒนาและปรับแต่งโครงการขนาดเล็กได้ง่าย หมายความว่า Github Actions กำลังก้าวไปสู่การเป็นทางเลือกหลักสำหรับโครงการขนาดเล็ก การมีเครื่องมือบิลด์รวมอยู่ในที่เก็บซอร์สโค้ดโดยตรงเป็นสิ่งที่สะดวกสบายมาก เหล่านักพัฒนาต่างมาสนใจฟีเจอร์นี้ทำให้มีเครื่องมือและเวิร์กโฟลว์เกิดขึ้นมากมายพร้อมสำหรับใช้งาน และผู้พัฒนาเครื่องมือเองก็กำลังส่งผลิตภัณฑ์ขึ้นไปบน Github Marketplace อย่างไรก็ตามเรายังคงแนะนำให้คุณดำเนินการด้วยความระมัดระวัง แม้ว่าโค้ดและ Git จะสามารถเปลี่ยนไปใช้ Git รีโมตอื่นได้ แต่เวิร์กโฟลว์การพัฒนาที่ยึดตาม Github Actions นั้นไม่สามารถย้ายไปได้ นอกจากนี้ โปรดพิจารณาว่า โครงการมีขนาดใหญ่หรือซับซ้อนพอที่จะใช้ไปป์ไลน์แยกแล้วหรือยัง แต่สำหรับการเริ่มต้นใช้งานอย่างรวดเร็วในโครงการขนาดเล็กควรพิจารณา Github Actions และระบบนิเวศที่กำลังเติบโตรอบตัวพวกมัน

Graal Native Image

ประเมิน

Graal Native Image เป็นเทคโนโลยีที่คอมไพล์โค้ด Java ให้เป็นไฟล์ที่รันได้หรือไลบรารีสำหรับใช้ร่วมกัน โดยสามารถทำงานบนระบบปฏิบัติการโดยตรง ไม่ต้องผ่านเวอร์ชวลแมชชีน เนทีฟอิมเมจได้รับการปรับให้เหมาะสมเพื่อลดหน่วยความจำและเวลาเริ่มต้นของแอปพลิเคชัน ทีมงานของเราประสบความสำเร็จในการใช้ Graal ทำงานเป็นคอนเทเนอร์ Docker ขนาดเล็กใน สถาปัตยกรรมแบบไร้เซิร์ฟเวอร์ ซึ่งเน้นการลดเวลาเริ่มต้น แม้ว่ามันจะออกแบบมาสำหรับใช้งานกับภาษาเช่น Go หรือ Rust ซึ่งคอมไพล์แบบเนทีฟ และต้องการขนาดไบนารีที่เล็กลงและเวลาเริ่มต้นที่สั้นลง แต่ Graal เนทีฟอิมเมจอาจมีประโยชน์สำหรับทีมที่มีความต้องการนำไปใช้งานแบบในอื่น ๆ และต้องการใช้กับภาษาที่ทำงานบน JVM โดย Graal Native Image Builder รองรับภาษาที่ทำงานบน JVM เช่น Java Scala Clojure และ Kotlin และสร้างไฟล์ที่รันได้ (executable) บนระบบปฏิบัติการต่าง ๆ เช่น MacOS Windows และ หลาย ๆ ระบบปฏิบัติการในเครือ Linux เนื่องจากมันใช้สมมติฐานแบบปิด (closed-world

assumption) ที่ต้องทราบโค้ดทั้งหมดที่ใช้ในเวลาคอมไพล์ ทำให้ฟีเจอร์ reflection หรือ dynamic class loading จำเป็นต้องมีการตั้งค่าเพิ่มเติม เพื่อช่วยให้สามารถหาประเภทของคลาสได้

HashiCorp Boundary
ประเมิน

HashiCorp Boundary เป็นเครื่องมือที่รวมความสามารถที่จำเป็นในการจัดการเน็ตเวิร์กที่มั่นคงและการระบุตัวตนเข้าด้วยกัน เพื่อจะได้เป็นตัวกลางในการติดต่อกับโฮสต์และเซอร์วิสต่าง ๆ ได้ในตัวเอง แม้เซอร์วิสนั้นจะอยู่ที่ต่างผู้ให้บริการคลาวด์หรืออยู่ภายในองค์กรของคุณเอง นอกจากนี้ คุณสามารถใช้ระบบจัดการคีย์ที่เลือกเอง ของผู้ให้บริการคลาวด์ หรือจะใช้ของ HashiCorp Vault เองก็ได้

HashiCorp Boundary กำลังเพิ่มจำนวนผู้ให้บริการยืนยันตัวตนที่รองรับมากขึ้นเรื่อย ๆ เมื่อเราใช้มันแล้ว เราจะสามารถกำหนดสิทธิ์การเข้าถึงทั้งในระดับโฮสต์และระดับเซอร์วิสเลย ตัวอย่าง เช่น มันเปิดโอกาสให้เราควบคุมสิทธิ์การเข้าถึงคลัสเตอร์ Kubernetes ได้อย่างละเอียด และจะมีฟีเจอร์สำหรับดึงรายชื่อเซอร์วิสต่าง ๆ ได้อย่างไดนามิกจากหลายแหล่งเร็ว ๆ นี้ การทำงานของมันทั้งหมดเกิดขึ้นเบื้องหลัง ผู้ใช้งานยังทำงานผ่านระบบเบื้องหน้าที่คุ้นเคยต่อไป ที่เป็นเช่นนี้ได้ เป็นเพราะมันเชื่อมต่อกันผ่านเลเยอร์การจัดการเน็ตเวิร์กของ Boundary เอง

imgcook
ประเมิน

ถ้าใครยังจำโครงการวิจัย pix2code กันได้อยู่ ที่ผู้ใช้เพียงป้อนภาพหน้าจอ GUI เข้าไประบบก็จะสร้างโค้ดหน้าจอขึ้นมาให้อย่างอัตโนมัติ ปัจจุบันเราได้เห็นผลิตภัณฑ์ที่ใช้เทคนิคดังกล่าวนี้เกิดขึ้นจริงแล้ว ซึ่ง imgcook เป็นบริการ SaaS ของอาลีบาบาที่สามารถแปลงไฟล์ภาพการออกแบบอย่าง ไฟล์ Sketch, PSD, หรือไฟล์ภาพทั่วไป ไปเป็นโค้ดฟรอนท์เอนด์ได้อย่างชาญฉลาด โครงการนี้เกิดขึ้นจากที่อาลีบาบาต้องง่วนอยู่กับการปรับหน้าแคมเปญจำนวนมากให้เข้ากับช่วงเทศกาลชอปปิ้งสิบเอ็ดเดือนสิบเอ็ด ซึ่งส่วนใหญ่เป็นหน้าเว็บที่ใช้ครั้งเดียวทิ้ง จึงต้องการเครื่องมือช่วยเหลือให้การผลิตหน้าเหล่านี้ทำได้อย่างรวดเร็ว ด้วยเทคโนโลยีดีปเลิร์นนิง นักออกแบบสามารถส่งไฟล์ต้นฉบับขึ้นไปให้มันประมวลผล โค้ดผลลัพธ์ที่ได้มาสามารถส่งต่อให้นักพัฒนานำไปตรวจสอบความเรียบร้อย ซึ่งทีมงานของเรากำลังประเมินเทคโนโลยีนี้อยู่ แม้ imgcook จะประมวลผลภาพที่ฝั่งเซิร์ฟเวอร์และเราจะติดต่อกับมันผ่านหน้าเว็บเป็นหลัก แต่มันก็ได้จัดเตรียมชุด เครื่องมือเสริม ไว้สำหรับเชื่อมต่อกับเครื่องมือต่าง ๆ เพื่อให้การทำงานระหว่างนักออกแบบและนักพัฒนาเป็นไปได้อย่างลื่นไหล ทั้งนี้เราสามารถกำหนดได้ว่าต้องการให้มันสร้างโค้ดออกมาในรูปแบบไหนผ่านภาษา DSL ที่มีให้ เทคโนโลยีนี้ไม่ได้สมบูรณ์แบบ นักออกแบบจำเป็นต้องออกแบบตามข้อตกลง (specifications) บางอย่างเพื่อทำให้ imgcook สร้างโค้ดออกมาได้ดีขึ้น แล้วนักพัฒนายังคงต้องปรับแก้โค้ดในภายหลังอยู่ดี เรามักจะเตือนให้ระวังการใช้งานโค้ดที่ถูกสร้างขึ้นอย่างอัตโนมัติ เพราะส่วนใหญ่โค้ดที่ได้มักจะยากต่อการบำรุงรักษาในระยะยาว ซึ่ง imgcook ก็เข้าข่ายด้วยเช่นกัน แต่หากจะนำมาใช้งานที่เฉพาะกิจ เช่น การสร้างหน้าแคมเปญแบบใช้ครั้งเดียวแบบนี้ ก็คุ้มค่าที่จะลอง

Longhorn
ประเมิน

Longhorn เป็นเครื่องมือจัดการการเก็บข้อมูลชนิดบล็อกแบบกระจายตัวสำหรับ Kubernetes ท่ามกลางทางเลือกมากมายในการเก็บข้อมูลแบบถาวรของ Kubernetes สิ่งที่ทำให้ Longhorn โดดเด่นคือมันถูกสร้างขึ้นมาเพื่อรองรับการทำสำเนาเฉพาะส่วน (incremental snapshot) และการสำรองข้อมูล (backup) มันจึงทำให้การเพิ่มเสถียรภาพให้ตัวเก็บข้อมูลสำรองบน Kubernetes ที่ไม่ได้อยู่บนคลาวด์ เป็นเรื่องง่ายขึ้น และในการทดลองเปิดใช้ฟีเจอร์ ReadWriteMany (RWX) ทำให้หลายเครื่องเซิร์ฟเวอร์ (node) สามารถอ่านและเขียนมายังแหล่งเก็บข้อมูลเดียวกัน ในการเลือกตัวจัดการการเก็บข้อมูลสำหรับ Kubernetes เป็นสิ่งที่ต้องตระหนักเป็นอย่างดี หากระบบต้องใช้การจัดเก็บข้อมูลแบบบล็อก เราแนะนำให้พิจารณา Longhorn เป็นอีกทางเลือกหนึ่ง

Operator Framework
ประเมิน

Operator Framework เป็นชุดเครื่องมือโอเพนซอร์สที่ช่วยลดความยุ่งยากในการสร้างและจัดการวงจรชีวิตของโอเปอเรเตอร์ของ Kubernetes การกำหนดโอเปอเรเตอร์เป็นรูปแบบไว้ ถูกนำเสนอครั้งแรกโดยทีม CoreOS ซึ่งเป็นการรวบรวมเทคนิคและเป็นการซ่อนความซับซ้อนในการจัดการแอปพลิเคชัน Kubernetes ในรูปแบบต่าง ๆ โดยอาศัยความสามารถพื้นฐานของ Kubernetes เอง ซึ่งในแต่ละรูปแบบจะประกอบไปด้วยการกำหนด resources ที่จะถูกบริหารจัดการ และ controller code ที่ทำหน้าที่คอยตรวจตราทำให้ทรัพยากรตรงกับสถานะเป้าหมายที่กำหนด วิธีนี้จึงมักถูกนำมาใช้ต่อยอดความสามารถของ Kubernetes ให้รองรับการจัดการ แอปพลิเคชันในรูปแบบต่างกันไป โดยเฉพาะอย่างยิ่งการนำมาใช้จัดการแอปพลิเคชันที่จำสถานะ (stateful) เป็นต้น Operator Framework มีองค์ประกอบหลักสามส่วนด้วยกัน ได้แก่ ส่วนที่หนึ่ง Operator SDK ที่ช่วยลดความยุ่งยากในการพัฒนา ทดสอบ และจัดทำโอเปอเรเตอร์เป็นแพ็คเกจขึ้นมา ส่วนที่สอง Operator lifecycle manager ที่ช่วยให้การติดตั้ง จัดการ และอัปเกรดโอเปอเรเตอร์ทำได้สะดวก และส่วนสุดท้ายคือ แค็ตตาล็อก ที่ทำหน้าที่รวบรวมโอเปอเรเตอร์ภายนอกไว้ให้ใช้งาน ทีมงานของเราพบว่า Operator SDK เป็นเครื่องมือที่ช่วยทุ่นแรงมากทำให้การพัฒนาแอปพลิเคชัน Kubernetes-native ทำได้อย่างรวดเร็ว

Recommender
ประเมิน

ในขณะที่บริการต่าง ๆ ของคลาวด์เจ้าใหญ่ทั้งหลายมี

เครื่องมือ

Recommender เป็นบริการบน Google Cloud ที่วิเคราะห์ทรัพยากรที่ใช้งานอยู่ และให้คำแนะนำถึงวิธีการเพิ่มประสิทธิภาพให้เหมาะสมตามการใช้งานจริงของผู้ใช้นั้น ๆ

(Recommender)

เครื่องมือ

Zally เป็นเครื่องมือตรวจสอบรูปแบบและไวยากรณ์ของมาตรฐาน OpenAPI แบบกะทัดรัด ที่จะช่วยให้เรามั่นใจว่า API ของเราสอดคล้องกับข้อแนะนำสไตล์ที่ทีมกำหนดไว้

(Zally)

จำนวนเพิ่มสูงขึ้นเรื่อย ๆ ความสะดวกสบายและความสมบูรณ์ของเครื่องมือต่าง ๆ ที่คอยช่วยเหลือให้ผู้ใช้ได้ใช้บริการเหล่านั้นอย่างเต็มประสิทธิภาพและมั่นคงสูงสุด ก็สูงขึ้นเช่นกัน Recommender เป็นบริการบน Google Cloud ที่วิเคราะห์ทรัพยากรที่ใช้งานอยู่ และให้คำแนะนำถึงวิธีการเพิ่มประสิทธิภาพให้เหมาะสมตามการใช้งานจริงของผู้ใช้นั้น ๆ บริการนี้เตรียม “ผู้ให้คำแนะนำ” ตามด้านต่าง ๆ ไว้ให้ใช้ เช่น ด้านความมั่นคง ด้านปริมาณการใช้งานและการประมวลผล หรือด้านการประหยัดต้นทุน ยกตัวอย่าง เช่น ผู้ให้คำแนะนำ IAM Recommender จะทำหน้าที่ช่วยให้ผู้ใช้สามารถกำหนดเฉพาะสิทธิ์ที่จำเป็นให้ดีขึ้น ด้วยการชี้ให้เห็นถึงสิทธิ์บางสิทธิ์ที่ไม่เคยถูกใช้งาน ซึ่งอาจจะมีสาเหตุมาจากการที่ผู้ใช้วางสิทธิ์ไว้กว้างเกินไป

Remote - WSL

ประเมิน

ในช่วงไม่กี่ปีที่ผ่านมาเราได้กล่าวถึงระบบย่อย Windows สำหรับ Linux (WSL) ในบทสนทนาเป็นบางครั้ง แม้ว่าเราจะชอบ WSL รวมถึงการปรับปรุงใน WSL 2 แต่มันไม่เคยถูกนำมาใส่ในเรดาร์ ในเรดาร์ฉบับนี้เราจะพูดถึงตัวส่วนขยายสำหรับ Visual Studio Code ที่ช่วยปรับปรุงประสบการณ์การทำงานกับ WSL เป็นอย่างมาก แม้ว่าเดิมที Editor บน Windows สามารถเข้าถึงไฟล์ในระบบไฟล์ใน WSL ได้เสมอ แต่มันก็ไม่ได้ตระหนักรับรู้ถึงสภาพแวดล้อมของ Linux ที่แยกออกมา ด้วยส่วนขยาย Remote - WSL Visual Studio Code จะตระหนักถึง WSL ทำให้นักพัฒนาสามารถเปิดใช้เชลล์ Linux ได้ อีกทั้งทำให้สามารถดีบักไบนารี่ที่รันใน WSL จาก Windows ได้ และนอกจากนี้ IntelliJ ของ JetBrains ก็ได้รับการปรับปรุงในส่วนของการรองรับ WSL อย่างต่อเนื่อง

Spectral

ประเมิน

หัวข้อประเภทหนึ่งที่มักจะโผล่มาในทุก ๆ เรดาร์ นั่นคือการกล่าวถึงเครื่องมือในกลุ่มตรวจสอบความถูกต้องของเอกสาร ซึ่งมักจะมีเครื่องมือประเภทนี้ออกมากมาย ทุก ๆ ครั้งที่เกิดภาษาใหม่ขึ้นมา เครื่องมือเหล่านี้เป็นที่รู้จักกันทั่วไปในนามว่า ลินเตอร์ (linter) ซึ่งชื่อนี้มีที่มาจากคำสั่ง lint ของ Unix ที่เอาไว้ใช้ตรวจสอบโค้ดภาษา C เราชื่นชอบเครื่องมือประเภทนี้เสมอ เพราะมันช่วยแจ้งเตือนข้อผิดพลาดให้ทราบก่อนที่โค้ดจะถูกคอมไพล์เลย ด้วยซ้ำ ซึ่งเครื่องมือล่าสุดในกลุ่มนี้ได้แก่ Spectral อันเป็นเครื่องมือไว้ตรวจสอบความถูกต้องของเอกสารที่จัดเก็บในรูปแบบ YAML และ JSON แม้ Spectral ดูเหมือนจะเป็นเครื่องมือทั่วไปที่ใช้ตรวจสอบไฟล์ฟอร์แมตดังกล่าว แต่เป้าหมายที่แท้จริงของมัน คือ การนำไปใช้ตรวจสอบไฟล์ที่ทำตามข้อกำหนดของ OpenAPI (ที่วิวัฒน์มาจากข้อกำหนด Swagger) และข้อกำหนด AsyncAPI โดยมันเตรียมชุดของกฎที่พร้อมใช้งานไว้ให้แล้ว ทำให้ชีวิตของนักพัฒนาง่ายขึ้นในการออกแบบและพัฒนา API เช่น มีกฎที่จะตรวจสอบว่า API ขาดพารามิเตอร์บางตัวไปหรือไม่ หรือกฎที่คอยเตือนหากลืมกำหนดค่าฟิลด์ license ไป แม้จะเป็นเรื่องน่ายินดีที่มีเครื่องมือใหม่ให้ใช้งานในการพัฒนา API แต่เราก็อดตั้งคำถามไม่ได้ว่า จริงๆ แล้ว การเขียนข้อกำหนดที่ไม่สามารถตรวจสอบได้ว่าทำงานได้จริง ควรจะซับซ้อนจนต้องใช้เครื่องมือประเภทนี้เลยหรือไม่ ถ้าจะซับซ้อนขนาดนี้ ข้อกำหนดเหล่านี้ควรอยู่ในรูปแบบโค้ดมากกว่าเป็นไฟล์เอกสารหรือไม่

Yelp detect-secrets

ประเมิน

Yelp detect-secrets เป็นโมดูลหนึ่งในภาษา Python ที่สามาถตรวจจับรหัสลับ (secret) ภายในโค้ดใด ๆ ซึ่งมันไปไล่สแกนไฟล์ต่าง ๆ ในไดเรกทอรีเพื่อหาดูว่ามีสิ่งที่อาจเป็นรหัสลับฝังอยู่หรือไม่ เราสามารถใช้โมดูลนี้ร่วมกับฮุกของ Git ให้มันทำงานก่อนที่การคอมมิทจะเกิดขึ้น หรือวางมันไว้เป็นส่วนหนึ่งของไปป์ไลน์ CI/CD ก็ได้ โดยที่เราสามารถเริ่มต้นใช้งานได้ทันที เพราะมันมากับค่าตั้งต้นที่พร้อมใช้งานแล้ว และหากต้องการจะปรับแต่งก็ทำได้สะดวกเช่นกัน นอกจากนี้ยังมีระบบปลั๊กอินส่วนเสริมเพื่อเพิ่มความสามารถในการค้นหารหัสลับได้อีกด้วย หากเปรียบเทียบเครื่องมือนี้กับเครื่องมืออื่น ๆ ในทำนองเดียวกัน Yelp สามารถตรวจจับรหัสลับได้มากกว่าเพื่อน แม้ยังไม่ได้ปรับแต่งค่าอะไรเลย

Zally

ประเมิน

เมื่อระบบนิเวศของข้อกำหนดการใช้งาน API เติบโตขึ้น เราพบว่ามีเครื่องมือที่ใช้ตรวจสอบสไตล์โดยอัตโนมัติเพิ่มมากขึ้นเรื่อย ๆ เช่นกัน Zally เป็นเครื่องมือตรวจสอบรูปแบบและไวยากรณ์ของมาตรฐาน OpenAPI แบบกะทัดรัด ที่จะช่วยให้เรามั่นใจว่า API ของเราสอดคล้องกับข้อแนะนำสไตล์ที่ทีมกำหนดไว้ โดยที่มันสามารถตรวจสอบกับกฎที่สร้างขึ้นมาสำหรับคำแนะนำการเขียน API ในรูปแบบของ Zalando ได้ทันทีหลังติดตั้ง และมันก็ยังสนับสนุนการใช้งานส่วนต่อขยายของ Kotlin เพื่อสร้างกฎที่จะใช้ตรวจสอบได้เองอีกด้วย Zally ประกอบไปด้วยเว็บ UI ที่มีอินเตอร์เฟสเพื่อแสดงผลการละเมิดกฎของสไตล์ต่าง ๆ และมี CLI ที่ช่วยให้การนำ Zally ไปใช้บนไปป์ไลน์การส่งมอบอย่างต่อเนื่องของคุณนั้นง่ายขึ้น

AWS CodePipeline

เฝ้าระวัง

จากประสบการณ์ของทีมธอทเวิร์คหลาย ๆ ทีม เราแนะนำว่าให้ใช้ AWS CodePipeline อย่างระมัดระวัง เราพบว่ามันใช้ยากขึ้นมาทันทีเมื่อทีมต้องการสร้างไปป์ไลน์ที่ซับซ้อนกว่าไปป์ไลน์พื้นฐาน เมื่อทีมเริ่มใช้งาน AWS ในช่วงแรกมันอาจจะดูเหมาะสม และเริ่มต้นได้ง่ายดี แต่เราอยากให้ลองถอยมาดูภาพกว้างขึ้น แล้วพิจารณาให้ดีว่า AWS CodePipeline ยังเหมาะสมอยู่หรือไม่ในสถานการณ์ระยะยาว ยกตัวอย่าง เช่น วันหนึ่งไปป์ไลน์จะต้องรองรับการกระจายงานออก และการรวมงานเข้าด้วยกันหรือไม่ หรือระบบต้องใช้วิธีการติดตั้งและทดสอบที่ซับซ้อนกว่าการใช้ทริกเกอร์ปกติหรือไม่

TECHNOLOGY RADAR

ภาษา & เฟรมเวิร์ค

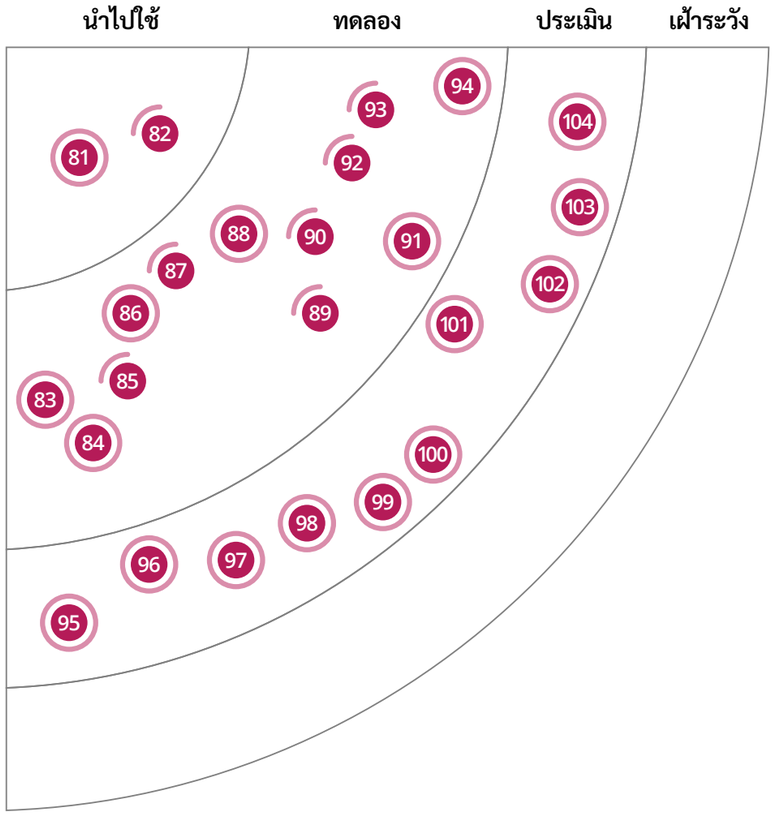
ภาษา & เฟรมเวิร์ค

Combine
นำไปใช้

เมื่อนานมาแล้ว เราได้ยกโครงการ ReactiveX ไว้ในหมวดหมู่ “ให้นำไปใช้” ซึ่งมันเป็นโครงการโอเพนซอร์สที่เป็นต้นตระกูลของเฟรมเวิร์คโอเพนซอร์สตัวอื่น ๆ ให้สามารถเขียนโปรแกรมเชิงรีแอคทีฟได้ ต่อมาในปี 2017 เราก็ได้กล่าวเพิ่มเติมถึงเฟรมเวิร์ค RxSwift ที่นำความสามารถการเขียนโปรแกรมเชิงรีแอคทีฟมาสู่โลกการพัฒนา iOS ด้วยภาษา Swift มาวันนี้ Apple ได้ออกเฟรมเวิร์คสำหรับการเขียนโปรแกรมเชิงรีแอคทีฟของตัวเองมาแล้วเช่นกัน ในนามของเฟรมเวิร์ค Combine จากนั้นมามันก็ได้เป็นตัวเลือกแรกหากเราต้องการจะพัฒนาแอปพลิเคชันมือถือใด ๆ ที่รองรับ iOS รุ่น 13 ขึ้นไป เฟรมเวิร์คตัวนี้เรียนรู้ง่ายกว่า RxSwift และทำงานร่วมกับ SwiftUI ได้เป็นอย่างดี หากใครกำลังวางแผนที่จะแปลงแอปพลิเคชันที่ใช้ RxSwift ไปเป็น Combine หรือพยายามใช้เฟรมเวิร์คทั้งคู่ในโครงการเดียวกัน คุณอาจสนใจศึกษาไลบรารี RxCombine เพิ่มเติม

LeakCanary
นำไปใช้

ทีมนักพัฒนาของเราพบว่า LeakCanary เป็นตัวเลือกเริ่มต้นที่ดีสำหรับเริ่มต้นการพัฒนาแอพพลิเคชั่นบน Android เพราะมันสามารถช่วยตรวจจับการรั่วไหลของหน่วยความจำที่น่ารำคาญได้ จากการมันสามารถเชื่อมต่อได้ง่าย และมีการแจ้งเตือนที่ชัดเจน ทำให้สามารถติดตามกลับไปยังสาเหตุของการรั่วไหลของหน่วยความจำได้อย่างมีประสิทธิภาพ สิ่งเหล่านี้ช่วยประหยัดเวลาให้คุณได้อย่างมากในการหาสาเหตุความผิดพลาดจากอาการที่แอพพลิเคชั่นใช้หน่วยความจำจนหมดเกลี้ยงในหลาย ๆ อุปกรณ์



เราจึงอยากแนะนำให้คุณได้เพิ่ม LeakCanary ลงในชุดเครื่องมือของคุณ

Angular Testing Library
ทดลอง

จากที่เรายังคงพัฒนาเว็บแอปพลิเคชันอย่างเป็นประจำด้วยภาษา JavaScript เราก็ยังเพลิดเพลินกับวิธีการทดสอบเว็บแอปพลิเคชันที่ Testing Library นำเสนอ นอกเหนือจาก React Testing Library ที่เราใช้เป็นประจำ เรายังมีโอกาสได้ไปสำรวจส่วนอื่น ๆ ของไลบรารีนี้เพิ่มเติม ซึ่ง Angular Testing Library ก็ได้นำข้อดีของ Testing Library มาใช้กับ Angular ทำให้เราสามารถทดสอบ UI คอมโพเนนต์จากมุมมองของผู้ใช้งาน ที่ทดสอบพฤติกรรมของคอมโพเนนต์แทนการทดสอบวิธีการทำงานภายใน นำมาซึ่งชุดทดสอบที่บำรุงรักษาได้ง่าย ถึงแม้จะมีเอกสารค่อนข้างน้อย แต่ Angular Testing Library ก็มีตัวอย่างการทดสอบที่เข้าใจได้ง่าย ตามลักษณะงานต่าง ๆ เลยช่วยให้เราเริ่มต้นได้อย่างรวดเร็ว เราประสบความสำเร็จอย่างมากกับการใช้งานไลบรารีนี้ในโปรเจกต์ Angular และขอแนะนำให้คุณลองใช้แนวทางเช่นนี้ในการเขียนชุดทดสอบ

AWS Data Wrangler
ทดลอง

AWS Data Wrangler เป็นไลบรารีโอเพ่นซอร์สที่ต่อยอด

นำไปใช้
81. Combine
82. LeakCanary

ทดลอง
83. Angular Testing Library
84. AWS Data Wrangler
85. Blazor
86. FastAPI
87. io-ts
88. Kotlin Flow
89. LitElement
90. Next.js
91. On-demand modules
92. Streamlit
93. SWR
94. TrustKit

ประเมิน
95. .NET 5
96. bUnit
97. Dagster
98. Flutter สำหรับเว็บ
99. Jotai และ Zustand
100. Kotlin Multiplatform Mobile
101. LVGL
102. React Hook Form
103. River
104. Webpack 5 Module Federation

เฝ้าระวัง

ภาษา& เฟรมเวิร์ค

FastAPI เป็นเว็บเฟรมเวิร์ครุ่นใหม่ที่เหมาะกับงานสร้าง API ที่ทำงานได้เร็ว รองรับโหลดหนัก ๆ ได้ดี ซึ่งต้องใช้ร่วมกับ Python รุ่น 3.6 ขึ้นไปเท่านั้น

(FastAPI)

ความสามารถของไลบรารีอย่าง Pandas ให้สามารถเชื่อมต่อดาต้าเฟรมกับบริการด้านข้อมูลต่าง ๆ ของ AWS ได้สะดวกยิ่งขึ้น นอกจากนี้แล้ว ไลบรารีนี้ยังใช้ความสามารถของ Apache Arrow และ Boto3 เพื่อสร้างเป็น API ต่าง ๆ สำหรับใช้โหลด แปลงรูป และบันทึกข้อมูลจากดาต้าเลค และดาต้าแวร์เฮาส์อีกด้วย ข้อจำกัดที่สำคัญของไลบรารีนี้คือมันไม่สามารถใช้ทำไปป์ไลน์ข้อมูลแบบกระจายตัวขนาดใหญ่ได้ แต่ทั้งนี้เราสามารถใช้บริการด้านข้อมูลต่าง ๆ ของ AWS เช่น Athena Redshift และ Timestream จัดการเรื่องที่ยุ่งยาก และการดึงข้อมูลไว้ก่อน เพื่อขั้นตอนถัดมาเราจะสามารถเขียนอธิบายวิธีการแปลงรูปข้อมูลที่ซับซ้อนให้กับดาต้าเฟรมนั้น ๆ ได้อย่างเหมาะสม จากที่เราได้ใช้ AWS Data Wrangler ในงานโปรดักชัน เราชอบที่มันช่วยให้เรามุ่งเน้นกับงานแปลงข้อมูลเป็นสำคัญ ไม่เสียเวลามากเกินไปกับการเชื่อมต่อกับบริการด้านข้อมูลต่าง ๆ ของ AWS

Blazor

ทดลอง

แม้ทุกวันนี้โลกการพัฒนาเว็บ UI ยังคงเป็น Javascript และระบบนิเวศรอบตัวมันที่ครองอิทธิพลสูงสุดอยู่ก็ตาม แต่การเติบโตของเทคโนโลยี WebAssembly ก็ทำให้มีนวัตกรรมใหม่ ๆ ปรากฏขึ้นมาบ้างแล้วเช่นกัน Blazor ยังคงดึงดูดความสนใจของเรา มันให้ผลลัพธ์ที่ดีกับทีมของเราในการสร้างส่วนเชื่อมต่อกับผู้ใช้แบบโต้ตอบได้ที่หลากหลายโดยการใช้ภาษา C# ที่ทำงานบน WebAssembly ความจริงที่ว่าทีมของเราสามารถเขียน C# ในฟรอนท์เอนด์ได้นั้นช่วยให้แบ่งปันโค้ดและนำไลบรารีที่มีอยู่มาใช้ใหม่ได้ และด้วยเครื่องมือที่มีอยู่สำหรับการดีบักและการทดสอบ เช่น bUnit ทำให้มันเป็นโอเพนซอร์สเฟรมเวิร์คที่คุ้มค่าน่าลอง

FastAPI

ทดลอง

เราเห็นหลาย ๆ ทีมนำ Python มาใช้เป็นภาษาหลักในการสร้างระบบ ไม่เพียงเฉพาะกับงานด้านวิทยาศาสตร์ข้อมูลเท่านั้น แต่ยังรวมไปถึงงานสร้างแบ็คเอนด์เซอร์วิสอีกด้วย ซึ่งในบริบทนี้ เราได้ใช้ FastAPI แล้วค่อนข้างประทับใจ มันเป็นเว็บเฟรมเวิร์ครุ่นใหม่ที่เหมาะกับงานสร้าง API ที่ทำงานได้เร็ว รองรับโหลดหนัก ๆ ได้ดี ซึ่งต้องใช้ร่วมกับ Python รุ่น 3.6 ขึ้นไปเท่านั้น นอกจากนั้นแล้ว เฟรมเวิร์คตัวนี้ยังมีระบบนิเวศที่ดีให้ใช้งาน เช่น มีความสามารถด้านการสร้างเอกสาร API ตามมาตรฐาน OpenAPI ซึ่งช่วยให้ทีมงานมุ่งเน้นไปที่การพัฒนาระบบ และผลิต REST API ออกมาได้อย่างรวดเร็ว ด้วยความสามารถที่ดีเช่นนี้ ทำให้เรามอง FastAPI เป็นทางเลือกที่ดีอีกทางหนึ่ง

io-ts

ทดลอง

เราเพลิดเพลินกับการใช้ TypeScript มาตลอดและหลงใหลความปลอดภัยที่ได้จากภาษาที่ช่วยตรวจสอบเรื่องไทป์ (strong type) อย่างไรก็ดี ในกรณีที่ข้อมูลมาจากภายนอกกรอบที่ระบบไทป์คุ้มกันไว้อยู่ (เช่น ข้อมูลได้มาจากเซอร์วิสแบ็คเอนด์ ณ ตอนรันไทม์) ก็อาจเกิดข้อผิดพลาดว่าข้อมูลไม่ตรงกับไทป์ได้อยู่ดี หนึ่งในไลบรารีที่ช่วยแก้ปัญหานี้คือ io-ts ซึ่งช่วยปิดข้อจำกัดของ TypeScript ที่การตรวจไทป์ทำได้เฉพาะตอนคอมไพล์เท่านั้น ให้สามารถตรวจสอบ ณ รันไทม์ได้ด้วยเมื่อเจอข้อมูลที่มาจากระบบภายนอกผ่านฟังก์ชันแปลงรหัส (encode) และฟังก์ชันถอดรหัส (decode) นอกจากนี้ เราสามารถสร้างการ์ดป้องกันไทป์ (type guard) ของเราเองได้ด้วย ยิ่งเราใช้ io-ts มากขึ้นเท่าไหร่ ยิ่งย้ำกับเราว่าไลบรารีนี้มีประโยชน์มากจริง ๆ และเรายังคงชื่นชอบวิธีการแก้ปัญหาที่งดงามที่มันทำออกมาได้ตรงจุดพอดี

Kotlin Flow

ทดลอง

การรองรับการสร้างโครูทีนในภาษา Kotlin (coroutine) ได้เปิดโอกาสให้มีนวัตกรรมใหม่ ๆ เกิดขึ้นอย่างมากมาย ซึ่งหนึ่งในนั้นก็คือ Kotlin Flow ซึ่งเป็นไลบรารีที่ใช้ความสามารถของโครูทีนเต็ม ๆ Kotlin Flow เป็นไลบรารีที่ล้อตามข้อกำหนดของ Reactive Streams โดยใช้ความสามารถโครูทีนอยู่เบื้องหลัง ซึ่งต่างจากไลบรารีอย่าง RxJava ตรงที่มันเป็น API พื้นฐานของ Kotlin เลย ซึ่งเปรียบได้กับ API เรื่อง sequence ของ RxJava ที่มีเมธอด map และ filter ให้ใช้งาน นอกจากนั้น Kotlin Flow ก็ยังเหมือนไลบรารี sequence ตรงที่ตัวข้อมูลจะถูกสร้างก็ต่อเมื่อถูกเรียกใช้งานเท่านั้น ทั้งหมดนี้ทำให้การเขียนโค้ดที่เธรดต่าง ๆ ต้องทำงานร่วมกัน ดูเรียบง่ายและดูแลง่ายขึ้นกว่าวิธีอื่น ๆ นอกจากนี้ การทดสอบ Flow ก็ยังทำได้ง่ายผ่านการเรียกเมธอด toList ก็จะแปลง Flow ไปเป็น List

LitElement

ทดลอง

เมื่อปี 2014 เราเคยพูดถึง Web Components ไป จากนั้นมันก็มีพัฒนาการอย่างสม่ำเสมอ LitElement ซึ่งเป็นส่วนหนึ่งของ Polymer Project เป็นไลบรารีอันเรียบง่าย ที่เอาไว้สร้างเว็บคอมโพเนนท์ขนาดเล็ก อันที่จริงมันเป็นเพียงคลาสแม่ที่จัดการเรื่องทั่วไปให้หมดแล้ว ทำให้เวลาจะสร้างเว็บคอมโพเนนท์ขึ้นมาสักตัวสามารถทำได้ง่ายมาก เทคโนโลยีนี้มีความพร้อมสำหรับการใช้งานและเป็นที่ชื่นชอบ เราได้ใช้งาน LitElement และประสบความสำเร็จกับโปรเจคส่วนใหญ่ที่ทำงานแบบเว็บคอมโพเนนท์

Next.js

ทดลอง

นับจากที่เราเขียนถึง Next.js ในครั้งที่แล้ว เราได้มีประสบการณ์ในการใช้งานร่วมกับโค้ด React มากขึ้น Next.js เป็นเฟรมเวิร์คที่มีแนวคิดเป็นของตัวเอง ไม่ต้องตั้งค่า การทำเราท์ติ้งอย่างง่าย คอมไพล์อัตโนมัติ รวมโค้ดด้วย Webpack และ Babel ระบบฮอตรีโหลดที่รวดเร็วเพื่อให้สะดวกกับการทำงานของนักพัฒนา และก็ยังมีฟีเจอร์อื่นๆ อีก Next.js ให้การแสดงผลฝั่งเซิร์ฟเวอร์เป็นค่าเริ่มต้น ช่วยปรับปรุงการเพิ่มประสิทธิภาพของเครื่องมือค้นหา ลดเวลาในการโหลด และสนับสนุนการอัพเดตการแสดงผลฝั่งเซิร์ฟเวอร์อย่างต่อเนื่อง (incremental static generation) หลายทีมที่ใช้งาน Next.js บอกเราว่ามีประสบการณ์การใช้งานที่ดีและต่างตื่นเต้นกับการเติบโตของเฟรมเวิร์คนี้

On-demand modules
ทดลอง

On-demand modules เป็นเฟรมเวิร์คสำหรับแอนดรอยด์ที่ช่วยให้แอพพลิเคชันสามารถดาวน์โหลด หลาย ๆ APK ที่ถูกปรับแต่งและแยกไว้ โดยจะดาวน์โหลดเฉพาะฟังก์ชันที่จำเป็นเท่านั้น สิ่งนี้ควรค่าแก่การทดลองใช้สำหรับแอปขนาดใหญ่ซึ่งความเร็วในการดาวน์โหลดอาจเป็นปัญหา หรือหากผู้ใช้มีแนวโน้มที่จะใช้ฟังก์ชันบางอย่างในการติดตั้งครั้งแรกเท่านั้น นอกจากนี้ยังสามารถลดความซับซ้อนในการจัดการอุปกรณ์หลาย ๆ ประเภทโดยไม่ต้องใช้ APK ที่แตกต่างกัน ส่วนใน iOS ก็มี App Clips ที่เป็นเฟรมเวิร์คในลักษณะเดียวกัน

Streamlit
ทดลอง

Streamlit เป็นเฟรมเวิร์คโอเพนซอร์สที่นักวิทยาศาสตร์ข้อมูลสามารถนำไปใช้สร้างแอปพลิเคชันสำหรับแสดงข้อมูลเชิงโต้ตอบด้วยภาษา Python งานปรับแต่งโมเดลของแมชชีนเลิร์นนิงให้ออกมาดีเป็นกระบวนการที่กินเวลา ดังนั้นแทนที่จะเสียเวลาไปมากับการปรับแก้แล้วนำไปทดลองบนแอปพลิเคชันจริง Streamlit จึงมีประโยชน์มากสำหรับเราในการเก็บรวบรวมผลในแต่ละรอบ เพราะมันสามารถสร้างแอปพลิเคชันโปรโตไทป์ออกมาได้อย่างรวดเร็วเพื่อใช้ในการทดลอง Streamlit โดดเด่นกว่าคู่แข่งอย่าง Dash ตรงที่มันให้ความสำคัญกับการสร้างโปรโตไทป์ให้ออกมาได้ไวกว่า และรองรับไลบรารีการแสดงภาพที่หลากหลาย เช่น Plotly และ Bokeh จากการที่เราได้ใช้ Streamlit ในโครงการต่าง ๆ มาบ้างแล้ว เราประทับใจที่มันสามารถสร้างแผนภาพข้อมูลเชิงโต้ตอบออกมาได้โดยไม่ยากเย็นนัก

SWR
ทดลอง

ทีมงานของเราพบว่าเมื่อใช้ SWR ซึ่งเป็นไลบรารีหนึ่งที่ใช้ร่วมกับ React Hooks ในสถานการณ์ที่เหมาะสมแล้ว ส่งผลให้การแสดงผลมีประสิทธิภาพดีขึ้น โดยที่โค้ดก็ยังสะอาดดีอยู่ SWR นั้นรองรับมาตรฐานการแคชแบบ stale-while-revalidate อันเป็นหนึ่งในกลยุทธการแคชของมาตรฐาน HTTP ที่ข้อมูลจะถูกดึงจากแคชก่อนเสมอ แม้ข้อมูลจะไม่ล่าสุดแล้วก็ตาม (stale) ขณะเดียวกันก็วิ่งไปดึงข้อมูลจากระบบภายนอก (revalidate) เมื่อไหร่ที่ข้อมูลกลับมาจึงตามไปอัปเดตค่าในแคชตามหลัง อย่างไรก็ดี เราขอเตือนให้ใช้กลยุทธการแคชแบบนี้เฉพาะในกรณีที่แอปพิเคชันรับได้ที่ผู้ใช้จะเห็นข้อมูลไม่ล่าสุดปัจจุบัน ซึ่งในมาตรฐาน HTTP ทั่วไปกำหนดไว้ว่า แคชต้องตอบรีเควสต์ด้วยข้อมูลจากเรสปอนด์ล่าสุดเท่านั้น และเฉพาะกรณีพิเศษที่พิจารณาอย่างรอบคอบแล้วเท่านั้น ถึงจะอนุญาตให้ตอบกลับด้วยข้อมูลจากเรสปอนด์ที่เก่ากว่านั้นได้

TrustKit
ทดลอง

การใช้เทคนิคการฝังพับลิคคีย์ (SSL public key pinning) มีข้อต้องระวังหลายอย่าง เพราะหากกำหนดนโยบายการฝังพิน (pinning policy) ไว้ไม่ดีพอหรือไม่ได้สำรองพินเก็บไว้ จู่ ๆ แอปพลิเคชันอาจหยุดทำงานอย่างกระทันหันเมื่อไรก็ได้ ปัญหานี้เองที่ทำให้ TrustKit เกิดขึ้นมา โดยมันเป็นโอเพนซอร์สเฟรมเวิร์คที่เข้ามาช่วยให้การฝังพับลิคคีย์ลงในแอปพลิเคชันมือถือเป็นเรื่องง่ายขึ้น รองรับทั้งระบบปฏิบัติ iOS และ Android ส่วนการจะเลือกวิธีการฝังพับลิคคีย์แบบใดให้เหมาะสมเป็นเรื่องที่ต้องทำอย่างละเอียดอ่อน ซึ่งคุณสามารถอ่านรายละเอียดเพิ่มเติมได้ที่ คู่มือเริ่มต้นใช้งานที่ทางโครงการเตรียมไว้ให้ เราเองได้ใช้ TrustKit มาแล้วกับหลายโครงการในโปรดักชัน ซึ่งก็ใช้ได้ดีทีเดียว

.NET 5
ประเมิน

เราไม่ได้หยิบ .NET มาพูดถึงทุกครั้งเวลามันออกรุ่นใหม่มา แต่ .NET 5 ครั้งนี้เป็นก้าวที่สำคัญมาก เพราะเป็นรุ่นที่ได้ผสาน .NET Core และ .NET Framework เข้าด้วยกันเป็นแพลตฟอร์มเดียว ดังนั้นองค์กรต่าง ๆ ควรจะเริ่มวางแผนกลยุทธ์ในการค่อย ๆ ย้ายจากแพลตฟอร์มอื่น ๆ มาสู่แพลตฟอร์มเดียวกันได้แล้ว เพราะจะทำให้นักพัฒนามีสภาพแวดล้อมการทำงานที่เหมือนกัน โดยไม่ขึ้นกับสภาพแวดล้อมปลายทางที่จะไปติดตั้ง ซึ่งอาจจะอยู่บน Windows, Linux หรือแอปพลิเคชันมือถือที่ทำงานข้ามแพลตฟอร์ม ผ่าน Xamarin หรือเป็นเว็บไซต์ ผ่าน Blazor ก็ตาม แน่นอนว่าการพัฒนาโดยผสมใช้หลายภาษาเข้าด้วยกันเป็นสิ่งที่เหมาะสมกว่าสำหรับบริษัทใดที่มีวัฒนธรรมองค์กรที่ดีและความพร้อมกว่า แต่สำหรับบริษัทอื่น ๆ การวางมาตรฐานกลางโดยอิงที่ .NET แพลตฟอร์มเดียวกัน น่าจะทำให้จัดการได้ง่ายกว่า ณ ตอนนี้เรายังอยากที่จะให้ประเมิน .NET 5 ไว้ก่อน เพื่อคอยดูว่า .NET 6 ที่จะรวมทุกสิ่งไว้อย่างสมบูรณ์นั้นจะทำได้ดีแค่ไหน

bUnit
ประเมิน

bUnit เป็นไลบรารีสำหรับทดสอบ Blazor คอมโพเนนต์ คุณสามารถใช้ bUnit ร่วมกับเฟรมเวิร์คที่คุ้นเคยอย่าง NUnit, xUnit และ MSUnit ในการเขียนและทดสอบยูนิตเทสต์คอมโพเนนต์ต่าง ๆ ของ Blazor ได้โดยตรง โดยมันจะครอบคอมโพเนนต์นั้นไว้ ทำให้นักพัฒนาสามารถเขียนยูนิตเทสต์ด้วยวิธีที่คุ้นเคย นำมาซึ่งความรวดเร็วและความอิสระในการทดสอบคอมโพเนนต์ใด ๆ ดังนั้นหากคุณกำลังใช้ Blazor ในการพัฒนาระบบ เราแนะนำให้พก bUnit ไว้ในชุดเครื่องมือของคุณด้วย

Dagster
ประเมิน

Dagster เป็นโอเพ่นซอร์สเฟรมเวิร์คการบริหารจัดการข้อมูล (data orchestration) สำหรับแมชชีนเลิร์นนิง งานวิเคราะห์และ ไปป์ไลน์ข้อมูล ETL ทั่วไป จุดแตกต่างที่สำคัญระหว่าง Dagster และเฟรมเวิร์คที่ขับเคลื่อนด้วยงาน (task-driven) ตัวอื่น ๆ คือ Dagster นั้นตระหนักถึงเรื่องของการไหลของข้อมูลผ่านไปป์ไลน์ และยังป้องกันการใช้รูปแบบข้อมูลผิดพลาด (type safety) การที่มีมุมมองภาพรวมสำหรับไปป์ไลน์และของที่ผลิตทั้งหมด Dagster จึงสามารถจัดเวลาและบริหาร Pands, Spark, SQL และระบบใด ๆ ที่ Python สามารถเรียกใช้ได้ เฟรมเวิร์คตัวนี้ค่อนข้างใหม่เมื่อเทียบกับตัวอื่น และพวกเราแนะนำทุกคนให้ประเมินความสามารถของเฟรมเวิร์คตัวนี้สำหรับไปป์ไลน์ข้อมูลของคุณ

ภาษา& เฟรมเวิร์ค

Streamlit เป็นเฟรมเวิร์คโอเพนซอร์สที่นักวิทยาศาสตร์ข้อมูลสามารถนำไปใช้สร้างแอปพลิเคชันสำหรับแสดงข้อมูลเชิงโต้ตอบด้วยภาษา Python

(Streamlit)

ภาษา& เฟรมเวิร์ค

ทั้งคู่เป็นไลบรารีบริหารจัดการสถานะสำหรับ React โดยพุ่งเป้าไปที่การมีขนาดเล็ก และความเรียบง่ายในการใช้งาน และอาจจะไม่ใช่เรื่องบังเอิญที่ชื่อของทั้งสองนั้นแปลว่า สถานะ ในภาษาญี่ปุ่นและเยอรมันเหมือนกัน ตามลำดับ

(Jotai and Zustand)

Flutter for Web

ประเมิน

ตลอดเวลาที่ผ่านมาเฟรมเวิร์ค Flutter สนับสนุนแอพพลิเคชั่นเนทีฟบน iOS และ Android เป็นหลัก แต่ทีมพัฒนา Flutter มีวิสัยทัศน์ที่จะสนับสนุนในทุก ๆ แพลตฟอร์ม Flutter สำหรับเว็บ เป็นอีกหนึ่งก้าวสำคัญ เพราะมันช่วยให้สร้างแอพพลิเคชัน iOS Android และบนเบราว์เซอร์โดยใช้โค้ดเดียวกัน Flutter สำหรับเว็บ เวอร์ชันทดลองได้ถูกปล่อยออกมาเป็นระยะเวลา 1 ปีแล้ว และตอนนี้มันได้มีเวอร์ชันใช้งานได้จริงตามเป้าหมายแล้ว พร้อมกับการปล่อย Flutter 2.0 ในช่วงแรกของการรองรับการพัฒนาเว็บไซต์นั้น ทีมผู้พัฒนา Flutter ได้เน้นไปที่ progressive web apps single-page apps และการทำให้แอพพลิเคชันมือถือที่มีอยู่แล้วใช้ร่วมกับเว็บได้ โค้ดของเว็บแอพพลิเคชันและเฟรมเวิร์คที่เป็นภาษา Dart ทั้งหมดจะถูกคอมไพล์เป็นภาษา JavaScript ต่างจากแอพพลิเคชันมือถือที่จะถูกคอมไพล์ไปยังภาษาที่เฉพาะเจาะจงสำหรับแพลตฟอร์มนั้น ๆ Flutter เปิดโอกาสให้นักพัฒนาเลือกการแสดงผลได้ 2 แบบคือ แบบ HTML ทั่วไป ประกอบไปด้วย HTML CSS Canvas และ SVG หรืออีกแบบหนึ่งคือ CanvasKit ที่ใช้ WebAssembly ร่วมกับ WebGL ในการส่งชุดคำสั่ง Skia ไปยังบราวเซอร์เพื่อแสดงผลกราฟฟิกแบบ 2 มิติ ทีมของเราเริ่มใช้งาน Flutter for Web กันบ้างแล้ว และมันให้ผลลัพธ์เริ่มต้นที่น่าประทับใจ

Jotai and Zustand

ประเมิน

ในเรดาร์ฉบับก่อนหน้า เราได้ให้ความคิดเห็นเกี่ยวกับระบบบริหารจัดการสถานะข้อมูล (state management) ในแอพพลิเคชั่น React เราได้ย้าย Redux กลับไปอยู่ในหมวดหมู่ “ทดลองใช้งาน” โดยระบุว่า Redux ไม่ได้เป็นตัวเลือกแรกอีกต่อไป และเรายังมีการพูดถึง Recoil ของ Facebook อีกด้วย ส่วนในฉบับนี้เราต้องการพูดถึง Jotai และ Zustand ซึ่งทั้งคู่เป็นไลบรารีบริหารจัดการสถานะสำหรับ React โดยพุ่งเป้าไปที่การมีขนาดเล็ก และความเรียบง่ายในการใช้งาน และอาจจะไม่ใช่เรื่องบังเอิญที่ชื่อของทั้งสองนั้นแปลว่า สถานะ ในภาษาญี่ปุ่นและเยอรมันเหมือนกัน ตามลำดับ ทั้งคู่แตกต่างกันในเรื่องแนวทางการทำงาน โดยแนวทางของ Jotai จะใกล้เคียงกับ Recoil ที่สถานะจะประกอบด้วยอะตอม (atoms) ที่เก็บอยู่ภายในโครงสร้างคอมโพเนนท์ของ React ในทางตรงกันข้าม Zustand จะเก็บสถานะไว้ในที่เก็บสถานะตัวเดียวซึ่งอยู่นอก React คล้ายกับแนวทางที่ Redux ใช้ ผู้พัฒนา Jotai ได้จัดทำรายการตรวจสอบที่มีประโยชน์ในการตัดสินใจว่าเราควรจะใช้ไลบรารีตัวไหนเมื่อไหร่

Kotlin Multiplatform Mobile

ประเมิน

ตามแนวโน้มของการพัฒนามือถือแบบข้ามแพลตฟอร์ม Kotlin Multiplatform Mobile (KMM) ถือเป็นผู้เล่นหน้าใหม่ในเรื่องนี้ KMM เป็น SDK ที่ถูกสร้างขึ้นโดย JetBrains ซึ่งใช้ประโยชน์จากความสามารถในการใช้งานได้หลายแพลตฟอร์มของ Kotlin โดย KMM มาพร้อมกับเครื่องมือและฟีเจอร์ที่ออกแบบมาให้ผู้พัฒนาแอปพลิเคชันบนมือถือแบบข้ามแพลตฟอร์มตั้งแต่ต้นจนจบ มีประสบการณ์ที่สนุกและมีประสิทธิภาพมากสูงสุด ด้วย KMM คุณจะเขียนโค้ดเพียงชุดเดียวทั้งส่วนของตรรกะทางธุรกิจ (business logic) และส่วนของโค้ดหลัก (app core) ด้วยภาษา Kotlin จากนั้นสามารถนำโค้ดนั้นไปใช้สำหรับแอพพลิเคชันทั้งใน Android และ iOS โดยเขียนโค้ดที่เจาะจงสำหรับแต่ละแพลตฟอร์มเมื่อจำเป็นเท่านั้น ตัวอย่างเช่นการใช้ประโยชน์จาก อิลิเมนต์ UI ของแพลตฟอร์ม (native UI element) โดยโค้ดที่เขียนขึ้นเฉพาะแต่ละแพลตฟอร์มจะถูกแยกเก็บไว้คนละที่ แม้โครงการนี้จะยังอยู่ในขั้นต้น แต่ Kotlin Multiplatform Mobile ก็พัฒนาไปได้อย่างรวดเร็วมาก เราจะจับตาดูการพัฒนาของมันอย่างแน่นอนและคุณก็ควรทำเช่นนั้นเหมือนกัน

LVGL

ประเมิน

ด้วยความนิยมของสมาร์ทโฮมและอุปกรณ์สวมใส่ที่เพิ่มขึ้น อุตสาหกรรมก็ต้องการกราฟิกส่วนที่ติดต่อกับผู้ใช้ (GUI) ที่น่าใช้งานมากขึ้น อย่างไรก็ตาม หากใครเคยพัฒนาซอฟต์แวร์บนอุปกรณ์ฝังตัว (embedded device) ที่ไม่ใช่ Android หรือ iOS จะพบว่าการพัฒนา GUI นั้นต้องใช้พยายามอย่างมาก

LVGL เป็นไลบรารีโอเพนซอร์สสำหรับพัฒนากราฟิกบนอุปกรณ์ฝังตัว ที่ได้รับความนิยมเพิ่มขึ้นเรื่อย ๆ โดยมันทำงานร่วมกับแพลตฟอร์มฝังตัวชื่อดังทั้งหลายได้ดี เช่น NXP, STM32, PIC, Arduino และ ESP32 และต้องการหน่วยความจำที่น้อยมาก อันได้แก่ พื้นที่สำหรับแฟลช 64 kB และแรมอีก 8 kB ก็ทำงานได้แล้ว นอกจากนี้ มันทำงานได้อย่างราบรื่นบนอุปกรณ์ที่ใช้ MCUs พลังงานต่ำ อย่าง Cortex-M0 อีกด้วย LVGL ยังรองรับอินพุตที่หลากหลายอย่างเช่นหน้าจอสัมผัส เมาส์ ปุ่ม และมีคอนโทรลอื่น ๆ มากกว่า 30 ชนิด รวมถึงอินพุตแบบ TileView ที่นิยมใช้กับพวกนาฬิกาอัจฉริยะ LVGL ใช้ใบอนุญาตแบบ MIT ที่เป็นมิตรกับองค์กรและการนำไปใช้เชิงพาณิชย์ ทีมงานของเราได้ใช้เครื่องมือตัวนี้แล้วในงานโปรดักชันแล้วมีความรู้สึกเป็นบวก ซึ่งทำได้ดีในการผลิตในจำนวนน้อย

React Hook Form

ประเมิน

การสร้างฟอร์มเป็นเรื่องที่อยู่คู่กับการพัฒนาเว็บมาช้านาน โดยเฉพาะอย่างยิ่งหากเราเจาะจงไปที่การสร้างฟอร์มกับ React ทีมต่าง ๆ ของเราจึงนิยมใช้ Formik เพื่อทำให้การพัฒนาฟอร์มทำได้ง่ายขึ้น แต่ก็มีบางทีมเช่นกันที่เริ่มหันมาพิจารณาใช้เครื่องมืออื่นเป็นทางเลือก อย่าง React Hook Form มันถูกพัฒนาขึ้นหลังจากที่ React Hooks เกิดขึ้นมาแล้ว จึงสามารถนำ hooks มาใช้ประโยชน์ได้อย่างเต็มที่ โดยที่อิลลิเมนต์ต่าง ๆ ภายในฟอร์มจะถูกลงทะเบียนและติดตามจากเฟรมเวิร์คเสมือนเป็นคอมโพเนนต์ที่ไม่อยู่ภายนอกการควบคุม วิธีนี้จะช่วยลดการรีเรนเดอร์อิลิเมนต์เหล่านี้ลงได้อย่างมีนัยสำคัญ นอกจากนี้ไลบรารีนี้ยังมีขนาดเล็กแล้วตั้งค่าได้ง่าย

River

ประเมิน

หัวใจสำคัญของแนวทางการพัฒนาแมชชีนเลิร์นนิง คือการสร้างแบบจำลองจากข้อมูลที่ใช้ฝึกโมเดล เมื่อสร้างแบบจำลองแล้วเราจะสามารถนำโมเดลกลับมาใช้ซ้ำได้เรื่อย ๆ อย่างไรก็ตามโลกไม่ได้หยุดนิ่งและบ่อยครั้งที่แบบจำลองต้องปรับตัวเติบโตตามข้อมูลใหม่ที่เข้ามา การแก้ปัญหาเบื้องต้นโดยการสร้างแบบจำลองใหม่จากศูนย์ทุก ๆ ครั้งอาจจะช้าและมีต้นทุนที่สูง การเรียนรู้จากส่วนที่เพิ่มเข้ามา (Incremental learning) จะช่วยขจัดปัญหานี้ ทำให้สามารถเรียนรู้จากสตรีมข้อมูลที่เพิ่มขึ้นเพื่อตอบสนองต่อการเปลี่ยนแปลงได้เร็วขึ้น นอกจากนี้ เทคนิคนี้ยังมีผลพลอยได้คือความต้องการการประมวลผลและหน่วยความจำจะลดลงและสามารถคาดการณ์ได้ เรามีประสบการณ์ที่ดีในการใช้งานเฟรมเวิร์ค River แม้ว่าตอนนี้เรายังต้องเพิ่มการตรวจสอบแบบอัตโนมัติ หรือทดสอบเองในบางครั้ง หลังการปรับโมเดล

Webpack 5 Module Federation

ประเมิน

การเปิดตัวฟีเจอร์ Webpack 5 Module Federation ได้รับการจับตามองจากนักพัฒนาที่ใช้งานสถาปัตยกรรม micro frontend เป็นอย่างมาก ซึ่งเป็นฟีเจอร์ที่กำหนดมาตรฐานในการจัดการหรือโหลดโมดูลภายนอกหรือโค้ดส่วนกลางใด ๆ ที่ใช้ร่วมกันให้มีประสิทธิภาพมากยิ่งขึ้น โดยที่เราจะสามารถระบุข้อกำหนดของโมดูลที่ใช้ร่วมกันไว้ล่วงหน้า ซึ่งช่วยแก้ปัญหาสำคัญในโลก micro frontend ที่บางโมดูลมีโอกาสถูกโหลดขึ้นมาซ้ำ ๆ ให้เหลือการโหลดเพียงครั้งเดียว นอกจากนี้มันช่วยแยกแยะโมดูลภายในและภายนอกออกจากกัน โดยที่โมดูลภายนอกคือโมดูลที่ไม่ถูกรวมเข้ากับบิลด์หลัก แต่จะถูกโหลดเข้ามาทีหลังขณะเปิดหน้าเพจ หากเปรียบเทียบระหว่างการบิลด์ ณ คอมไพล์ไทม์โดยใช้ npm ตรง ๆ แล้ว วิธีนี้ช่วยทำให้กระบวนการอัปเดตโมดูลต้นน้ำที่มีโมดูลอื่นจำนวนมากมาพึ่งพิง นั้นทำได้ง่ายขึ้นอย่างมาก แต่ขอระวังสำคัญในการใช้ฟีเจอร์นี้คือ มันจะบังคับให้คุณต้องรวมทุก ๆ micro frontend เข้าด้วยกันผ่าน Webpack เท่านั้น ซึ่งตรงข้ามกับวิธีการอย่าง import maps ที่อาจจะกลายเป็นส่วนหนึ่งของ W3C standard ในการรวม micro frontend ในอนาคต

ภาษา& เฟรมเวิร์ค

บ่อยครั้งที่แบบจำลองต้องปรับตัวเติบโตตามข้อมูลใหม่ที่เข้ามา การเรียนรู้จากส่วนที่เพิ่มเข้ามา (Incremental learning) จะช่วยขจัดปัญหานี้ ทำให้สามารถเรียนรู้จากสตรีมข้อมูลที่เพิ่มขึ้นเพื่อตอบสนองต่อการเปลี่ยนแปลงได้เร็วขึ้น และเรามีประสบการณ์ที่ดีในการใช้งานเฟรมเวิร์ค River

(River)

/thoughtworks

Thoughtworks คือบริษัทให้คำปรึกษาและวางแผนทางด้านเทคโนโลยีระดับโลก เราคือชุมชนที่รวมปัจเจกชนผู้หลงใหลในการทำในสิ่งที่เราเชื่อ มากกว่า 9,000 คน จากสำนักงาน 48 แห่งที่กระจายอยู่ 17 ประเทศทั่วโลก ด้วยประสบการณ์มากกว่า 27 ปีที่เราใช้เทคโนโลยีสร้างความแตกต่างเพื่อช่วยแก้ปัญหาทางธุรกิจที่ซับซ้อนให้กับลูกค้า เมื่อการเปลี่ยนแปลงคือความแน่นอน เราเตรียมคุณให้พร้อมรับกับสิ่งไม่คาดฝัน

อยากติดตามข่าวสารและข้อมูลเชิงลึกล่าสุดเกี่ยวกับเรดาร์ใช่ไหม?

ติดตามเราบนช่องทางโซเชียลที่คุณถนัด
หรือ สมัครรับข่าวสารทางอีเมล์จากเรา

สมัครรับข่าวสารทันที

/thoughtworks

thoughtworks.com/radar

#TWTechRadar