ThoughtWorks®

TECHNOLOGY RADAR

คู่มือนำทางสู่ปลายขอบเทคโนโลยี
ตามแบบฉบับของเรา

Volume 23

#TWTechRadar
thoughtworks.com/radar

ผู้ร่วมสร้างสรรค์

เรดาร์เทคโนโลยีถูกจัดทำขึ้นโดยคณะกรรมการที่ปรึกษาด้านเทคโนโลยี

คณะกรรมการที่ปรึกษาด้านเทคโนโลยีประกอบด้วยผู้นำและนักเทคโนโลยีผู้มากประสบการณ์ที่มากกว่า 21 คนของ ThoughtWorks พวกเขารวมตัวประจำแบบเจอหน้ากัน 2 ครั้งต่อปี และทุกๆ 2 สัปดาห์ทางโทรศัพท์ โดยมีหน้าที่สำคัญคือให้คำแนะนำ และเป็นที่ปรึกษาให้ประธานเจ้าหน้าที่บริหารฝ่ายเทคโนโลยี Rebecca Parsons

คณะกรรมการที่ปรึกษาจะทำหน้าที่เป็นผู้พิจารณาเทคโนโลยีที่กำลังมีบทบาท สำคัญในอุตสาหกรรมโลกขณะนั้น และผลกระทบต่อนักเทคโนโลยีของ ThoughtWorks เพื่อกำหนดแผนยุทธศาสตร์ภายใน ตามปรกติแล้วเราจัดทำเรดาร์เทคโนโลยีจากการรวมตัวในสถานที่เดียวกัน แต่เนื่องจากสถานการณ์โรคระบาดที่โลกกำลังประสบอยู่ 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

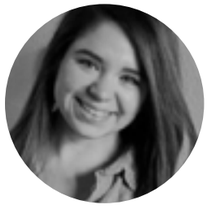
James Lewis

Lakshminarasimhan Sudarshan

Mike Mason

Neal Ford

Ni Wang



Perla Villarreal

Rachel Laycock

Scott Shaw

Shangqi Liu

Zhamak Dehghani

ทีมแปลภาษาไทย:
ฐปพงศ์ เศรษฐ์พิทักษ์, ณัฐพงศ์ อินทรักษ, ดิษทัต เพิ่มพูนวิวัฒน์, ธนกฤต จูฑะมงคล, ปัทมา ทวนชัยศรี, พิชญุตม์ จิตรศิลป์ฉายากุล, พิรดา ทิพย์รัตน์, พิสินี สกุลวานิชพร, พิสินี สกุลวานิชพร, มารัช ตรีคุณประภา, ยศ อนงค์เลขา, วิสุทธิ์ วงศ์กำชัย, ศอลาฮุดดีน เฉลิมไทย

เกี่ยวกับเรดาร์เทคโนโลยี

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

คณะกรรมการที่ปรึกษาของ ThoughtWorks ซึ่งประกอบไปด้วยผู้นำและผู้เชี่ยวชาญในเทคโนโลยีหลากหลายด้านที่มารวมตัวกันเป็นประจำในทุกปี เพื่อพูดคุยแลกเปลี่ยนถึงแผนยุทธศาสตร์ทางเทคโนโลยีภายในของ ThoughtWorks เอง และแนวโน้มของเทคโนโลยีที่กำลังมีบทบาทสำคัญในอุตสาหกรรมโลกในขณะนั้น

การรวบรวมผลการประชุมดังกล่าวถูกจัดทำขึ้นเป็นเรดาร์เทคโนโลยีฉบับนี้ โดยนำเสนอในรูปแบบที่เป็นประโยชน์กับทุกคนในวงการ ตั้งแต่นักพัฒนาจนถึงผู้บริหารระดับสูง ตัวเนื้อหานั้นเราเจตนาสรุปให้กระชับได้ใจความ หากอยากทราบถึงรายละเอียดเพิ่มเติมเราขอส่งเสริมให้คุณทดลองใช้เทคโนโลยีเหล่านั้นด้วยตัวคุณเอง

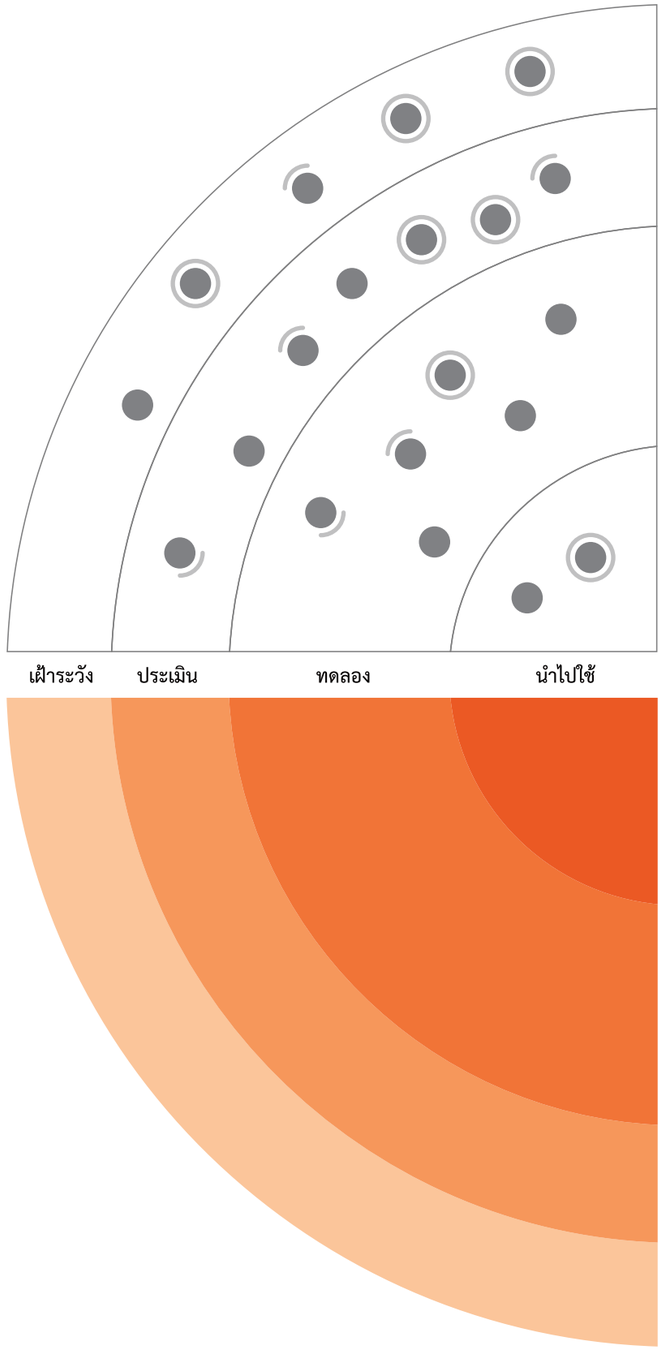
เรดาร์เทคโนโลยีใช้แผนภาพวงกลมในการนำเสนอข้อมูล โดยแบ่งพื้นที่ออกเป็น 4 กลุ่มดังนี้ เทคนิค เครื่องมือ แพลตฟอร์ม ภาษาและเฟรมเวิร์ค ในกรณีที่บางเรื่องสามารถจัด ลงได้หลายกลุ่มเราจะจัดลงในกลุ่มที่เหมาะสมที่สุด นอกจากนั้นเรายังจัดกลุ่มเรื่องต่างๆ แล้วแบ่งตามวงแหวน ออกเป็น 4 ระดับเพื่อสะท้อนมุมมองตำแหน่งของเทคโนโลยี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

หากสนใจประวัติเพิ่มเติมเกี่ยวกับเรดาร์เทคโนโลยีสามารถดูเพิ่มเติมได้ที่ thoughtworks.com/radar/faq

ภาพรวมของเรดาร์

เรดาร์ คือการติดตามเทคโนโลยีที่น่าสนใจซึ่งเราเรียกว่า หัวข้อ เรานำเสนอเรดาร์เทคโนโลยีฉบับนี้เป็นแผนภาพวงกลม โดยแบ่งประเภทออกตามเสี้ยวและวงแหวน โดยเสี้ยวสื่อถึงประเภทของหัวข้อต่างๆ ในขณะที่วงแหวนสื่อถึงระดับชั้นของการนำเทคโนโลยีนั้นไปใช้ตามความเห็นของเรา

หัวข้อต่างๆ สื่อถึงเทคโนโลยีหรือเทคนิคที่มีบทบาทสำคัญต่อการพัฒนาซอฟต์แวร์ หัวข้อเป็นสิ่งที่เคลื่อนไหวได้ เมื่อระดับชั้นในวงแหวนมีการเปลี่ยนตำแหน่งไป ปกติเมื่อมีการขยับขึ้นจะสื่อถึงระดับความมั่นใจกับเทคโนโลยีที่เพิ่มมากขึ้นตาม



ใหม่

เลื่อนเข้า/ออก

ไม่มีการเปลี่ยนแปลง

เรดาร์ของเรามีการเปลี่ยนแปลงตลอดเวลาเพื่อนำเสนอสิ่งใหม่ เทคโนโลยีที่ไม่มีการเปลี่ยนแปลงจะถูกทำให้จางลงซึ่งไม่ได้หมายความว่าเทคโนโลยีนั้นๆ ไม่มีคุณค่าแต่เนื่องด้วยพื้นที่ของเรดาร์นั้นมีจำกัด

นำไปใช้ (Adopt)

เราสนับสนุนให้นำเทคโนโลยีเหล่านี้ไปใช้ได้ทันทีซึ่งเราเองได้นำไปใช้แล้วในโครงการต่างๆ ที่มีความเหมาะสม

ทดลอง (Trial)

เราเห็นถึงความคุ้มค่าที่จะศึกษาเพิ่มเติม มันสำคัญที่จะเข้าใจวิธีการได้มาซึ่งความสามารถนี้ องค์กรควรทดลองใช้เทคโนโลยีนี้ในโครงการที่รับความเสี่ยงได้

ประเมิน (Assess)

มีความคุ้มค่าที่จะสำรวจเพื่อทำความเข้าใจว่ามีผลต่อองค์กรอย่างไร

เฝ้าระวัง (Hold)

ควรดำเนินการด้วยความระมัดระวัง

ประเด็นเด่นในฉบับนี้

GraphQL กับความยิ่งใหญ่ที่อาจจะมากเกินไป?

เราเห็นความนิยมใน GraphQL ที่เพิ่มสูงขึ้นในหมู่คน รวมไปถึงระบบนิเวศรอบ ๆ ตัวมันที่เติบโตขึ้นอย่างรวดเร็ว GraphQL ช่วยแก้ปัญหาบางประการที่มักพบได้ทั่วไปในสถาปัตยกรรมกระจายตัวอย่างไมโครเซอร์วิส เพราะเมื่อนักพัฒนาแตกระบบเป็นเซอร์วิสย่อย ๆ บ่อยครั้งที่พวกเขาก็ต้อง รวมข้อมูลกลับเข้าด้วยกันอีกครั้ง อยู่ดี เพื่อตอบโจทย์ความต้องการทางธุรกิจ แม้ว่า GraphQL จะสามารถแก้ปัญหาพื้นฐานนี้ได้อย่างสะดวก แต่ก็ไม่ต่างกับทุกแนวคิดที่ทรงพลังที่มาพร้อมกับข้อแลกเปลี่ยนที่ต้องพิจารณาอย่างระมัดระวังเพื่อหลีกเลี่ยงข้อเสียที่จะตามมาในระยะยาว ตัวอย่างเช่น เราเห็นบางคนเปิดเผยข้อมูลภายในมากเกินไปผ่านเครื่องมือรวบรวมข้อมูล อันนำไปสู่ปัญหาทางสถาปัตยกรรมที่เปราะบางในเวลาต่อมา อีกกรณีหนึ่ง ที่เกิดจากความพยายามแก้ปัญหาระยะสั้น แต่ส่งผลให้เกิดความลำบากในระยะยาว คือการที่คนไปใช้เครื่องมือรวบรวมข้อมูลบางตัว แล้วพยายามสร้างโมเดลศูนย์กลางของทั้งองค์กรขึ้นมา ให้ครอบคลุมทุกบริบทที่เป็นไปได้ ซึ่งในทางปฏิบัติจะทำให้สำเร็จได้ยาก ทั้งนี้ เรายังคงส่งเสริมให้คนใช้ GraphQL รวมถึงชุดเครื่องมือที่กำลังขยายกว้างรอบตัวมันต่อไป เพียงแต่เราอยากเตือนให้ระวังการนำเทคโนโลยีที่ออกแบบมาเพื่อแก้ปัญหาเฉพาะจุด ไปใช้แก้ไขหลายปัญหามากเกินไป

อุปสรรคบนเบราว์เซอร์ที่ยังต้องฝ่าฟันต่อไป

เดิมทีเบราว์เซอร์มีไว้สำหรับท่องดูเอกสารเท่านั้น ไม่ได้คิดเผื่อการรองรับแอปพลิเคชันเหมือนที่เป็นอยู่ทุกวันนี้ จากความไม่ตรงกันนั้นส่งผลต่อนักพัฒนาเสมอมา เพื่อจะเอาชนะอุปสรรคดังกล่าว นักพัฒนาต้องทบทวนและท้าทายสิ่งที่เป็นอยู่ซ้ำแล้วซ้ำเล่า เพื่อคิดหาวิธีการหรือเทคโนโลยีใดที่เหมาะสม สำหรับการสร้างแอปพลิเคชันให้สามารถทดสอบได้ มีวิธีจัดการสถานะข้อมูลที่ดี ตอบสนองได้เร็ว และอุดมไปด้วยความสามารถ ในเรดาร์ฉบับนี้เราจึงอยากให้เห็นแนวโน้มของเรื่องพวกนี้ว่ามีการเปลี่ยนแปลงไปอย่างไรบ้าง

ซึ่งเรื่องแรก เกี่ยวกับ Redux เมื่อปี 2017 เรายกให้มันอยู่ในหมวด “ให้นำไปใช้” เพื่อสื่อว่า Redux เป็นไลบรารีตั้งต้นสำหรับจัดการสถานะข้อมูลเมื่อใช้กับแอปพลิเคชัน React ใด ๆ ปัจจุบันเราเห็นว่านักพัฒนาหลายคนต่างเริ่มมองหาทางเลือกอื่นกันแล้ว (Recoil) หรือไม่ก็ชะลอการใช้ไลบรารีใดไลบรารีหนึ่งจนกว่าจะจำเป็น

เรื่องที่สอง Svelte ที่กำลังได้รับความสนใจมากขึ้นอย่างต่อเนื่อง มันได้เข้ามาท้าทายแนวคิดหลักตัวหนึ่ง ที่เฟรมเวิร์คยอดนิยมต่างก็ใช้กัน ทั้ง React และ Vue.js นั่นคือ Virtual DOM

เรื่องที่สาม คือ เราเริ่มเห็นเครื่องมือใหม่ ๆ ที่จะเข้ามาช่วยให้การทดสอบบนเบราว์เซอร์ทำได้ง่ายขึ้น Playwright เป็นเครื่องมือทดสอบ UI ที่พยายามปรับปรุงเรื่องนี้ให้ดีขึ้น หรือ Mock Service Worker ก็ใช้เทคนิคใหม่ที่น่าสนใจ เพื่อแยกการทดสอบระบบหลังบ้านออกจากหน้าบ้าน

เรื่องที่สี่ คือ เรายังเห็นปัญหาความไม่สมดุลกันระหว่างประสิทธิภาพและความคล่องตัวของนักพัฒนา เราจึงได้พูดถึง browser-tailored polyfills ที่มีเจตนาช่วยให้เรื่องนี้มีสมดุลมากขึ้น

ทำทุกอย่างให้เห็นภาพ

ในเรดาร์ฉบับนี้ เราพูดถึงหัวข้อมากมายจากหลายสาขาเทคโนโลยีด้วยกัน มีสิ่งหนึ่งที่คล้ายกันอยู่ในหัวข้อพวกนั้น นั่นคือเรื่องการแสดงแผนภาพข้อมูล (visualization) คุณจะได้พบกับเรื่องนี้ในหัวข้อสาขาต่าง ๆ ทั้งในด้านโครงสร้างพื้นฐาน วิทยาศาสตร์ข้อมูล ทรัพยากรบนคลาวด์ มีการพูดถึงนวัตกรรมใหม่ ๆ ของเครื่องมือสำหรับแสดงแผนภาพที่แปลกใหม่ รวมไปถึงเทคนิคต่าง ๆ เพื่อแสดงข้อมูลบางเรื่องที่ปกติจะยากต่อการอธิบาย ให้สามารถเห็นเป็นภาพได้อย่างเห็นผล นอกจากนี้ คุณจะได้อ่านข้อคิดเห็นเกี่ยวกับเครื่องมือในกลุ่มนี้ที่เกี่ยวกับด้านวิทยาศาสตร์ข้อมูล แดชบอร์ดต่าง ๆ เช่น Dash, Bokeh และ Streamlit รวมถึงเครื่องมือสำหรับแสดงแผนภาพของโครงสร้างพื้นฐาน อย่าง Kiali ที่เอาไว้ใช้แสดงแผนภาพเน็ตเวิร์คของเซอร์วิสเมช ในสถาปัตยกรรมไมโครเซอร์วิส ปัจจุบันระบบนิเวศรอบตัวนักพัฒนามีความซับซ้อนมากขึ้น การใช้รูปมาช่วยทำให้เห็นภาพเป็นวิธีการสื่อสารที่มีประสิทธิภาพ ที่จะช่วยให้เราอยู่กับความซับซ้อนที่ถาโถมเข้ามาโดยไม่บ้าไปเสียก่อน

การก้าวสู่วัยรุ่นของโครงสร้างพื้นฐานด้วยโค้ด

ทุกวันนี้การทำให้โครงสร้างพื้นฐานด้วยโค้ด กลายเป็นสิ่งที่ปฏิบัติกันทั่วไป จากที่องค์กรต่าง ๆ เข้าใจคุณค่าของการทำให้งานโครงสร้างพื้นฐานเป็นเรื่องอัตโนมัติ ขณะเดียวกันการใช้งานที่มากขึ้นนี้ ได้กลายเป็นเสียงตอบรับชั้นดีให้ผู้สร้างเครื่องมือและเฟรมเวิร์คนำไปสร้างนวัตกรรมใหม่ออกมา อันจะเห็นได้จากเครื่องมือยุคใหม่อย่าง CDK และ Pulumi และอีกมากมาย ที่มีความสามารถที่หลากหลายกว่าเครื่องมือในยุคแรกมาก ความแตกต่างระหว่างเครื่องมือทั้งสองรุ่นนี้ มีมากเสียจนเรามีความเห็นว่าการจัดการโครงสร้างพื้นฐานด้วยโค้ด ได้เข้าสู่ช่วง “วัยรุ่น” เต็มตัวแล้ว ทั้งในความหมายเชิงบวกและเชิงลบของคำนี้ ตั้งแต่เรื่องน่ายินดีที่เห็นได้ผ่านหัวข้อที่เกี่ยวข้องกระจายอยู่ทั่วกลุ่มเรดาร์ อันเป็นการสะท้อนว่าระบบนิเวศนี้มีความพร้อมมากขึ้น แล้วเราก็ได้พูดถึงเรื่องท้าทายที่หลายที่กำลังเผชิญกันอยู่เช่นกัน ทั้งในประเด็นการขาดรูปแบบการใช้งานที่ตกผลึกสมบูรณ์ และในแง่ความยุ่งยากจากความพยายามใช้ความสามารถนี้ให้เต็มประสิทธิภาพสูงสุด แต่ทั้งหมด ล้วนเป็นสัญญาณที่ดีของการเติบโตก่อนจะเข้าสู่วัยที่พร้อมสมบูรณ์ เราหวังอย่างยิ่งว่าชุมชนนักพัฒนาโครงสร้างพื้นฐานจะนำเทคนิคการออกแบบซอฟต์แวร์มาปรับใช้ โดยเฉพาะในเรื่องของการสร้างโครงสร้างพื้นฐานที่ดีพลอยได้ที่อิสระต่อกัน

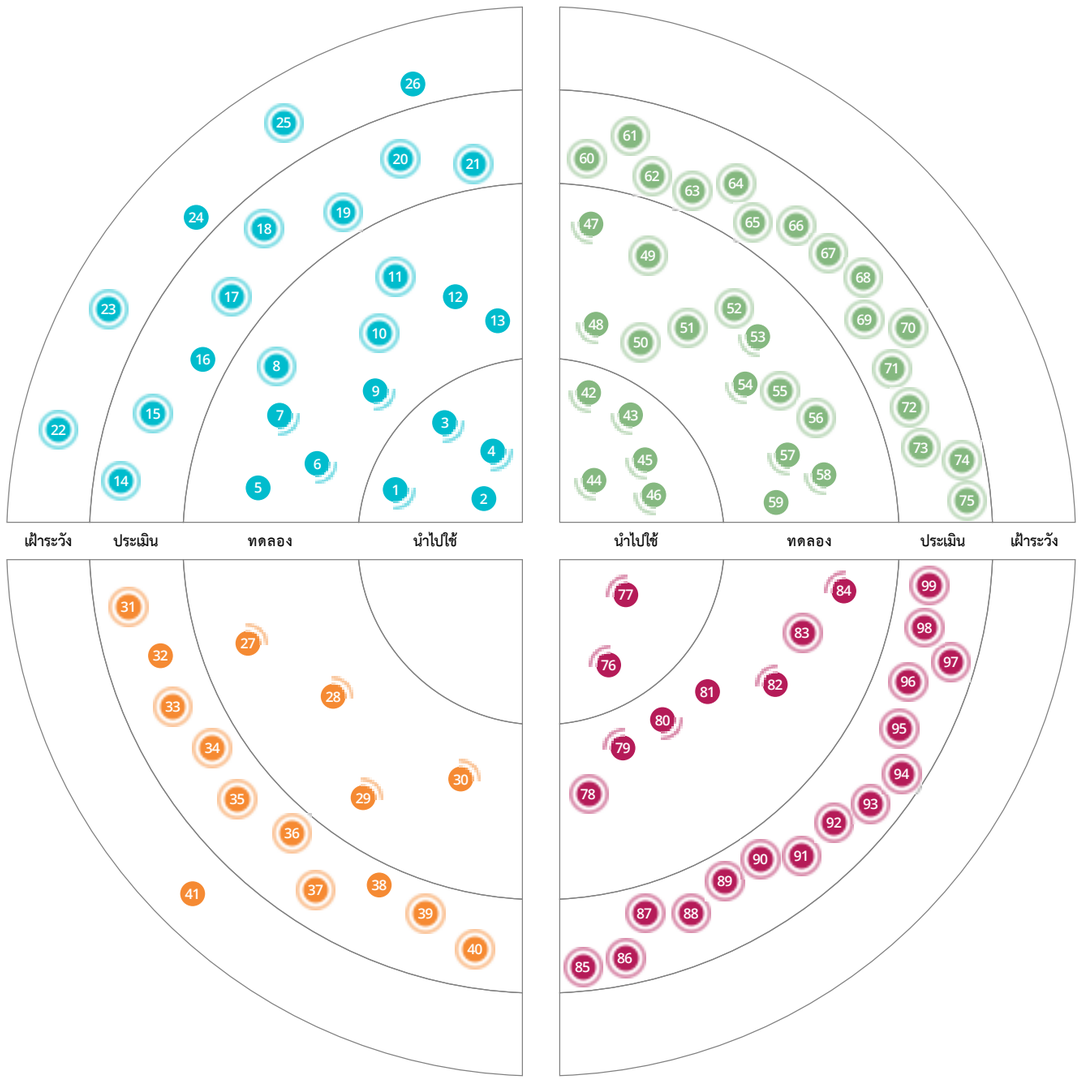
ความเท่าเทียมกันทางการเขียนโปรแกรม

ในหลาย ๆ ประเด็นที่เราเคยพูดถึงกันไปนั้นมักจะมุ่งเน้นไปที่ เครื่องมือและเทคนิคที่จะช่วยสนับสนุนให้เกิดความเท่าเทียมทางการเขียนโปรแกรม ซึ่งช่วยให้ผู้ที่เขียนโปรแกรมไม่ได้สามารถทำงานที่โปรแกรมเมอร์เท่านั้นถึงจะทำได้ ตัวอย่างเช่น IFTTT และ Zapier ต่างเป็นโซลูชันที่ได้รับความนิยมจากคนกลุ่มนี้

เรายังสังเกตเห็นเครื่องมือประเภทนี้มากขึ้น เช่น Amazon Honeycode ซึ่งเป็นเครื่องมือช่วยสร้างแอปพลิเคชันเชิงธุรกิจที่ไม่ซับซ้อน มันเตรียมสภาพแวดล้อมสำหรับการพัฒนาไว้ในแบบที่ผู้ใช้แทบไม่ต้องเขียนโค้ดเป็นเลย แม้ว่าเครื่องมือเหล่านี้จะเตรียมสภาพแวดล้อมสำหรับการพัฒนาให้ตรงกับวัตถุประสงค์ แต่ก็มีความท้าทายเมื่อเราต้องการขยับมันขึ้นไปสู่ระดับโปรดักชัน

เรื่องนี้ไม่ใช่เรื่องใหม่แต่อย่างใด ระหว่างนักพัฒนาซอฟต์แวร์และผู้ใช้สเปรดชีตอย่างซ่ำซองก็เคยเจอความขัดแย้งนี้มาก่อนแล้ว สุดท้ายพวกเขาก็หาวิธีพบกันครึ่งทางได้ในที่สุด การถือกำเนิดของเครื่องมือสมัยใหม่ในกลุ่มนี้ ได้จุดให้ประเด็นดังกล่าวเป็นที่ถกเถียงกันอีกครั้งในโดเมนต่าง ๆ ที่กว้างขึ้น รวมทั้งประเด็นด้านข้อดีและข้อเสีย ก็กลับมาเช่นกัน

เรดาร์เทคโนโลยี



เทคนิค

นำไปใช้

1. ฟังก์ชันประเมินความเหมาะสมด้านความเบี่ยงเบนของสิ่งพึ่งพา
2. ฟังก์ชันความเหมาะสมทางสถาปัตยกรรมในด้านรายจ่ายค่าบริหารงาน
3. นโยบายความมั่นคงด้วยโค้ด
4. แม่แบบเซอร์วิสของตนเอง

ทดลอง

5. การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML)
6. ดาต้าเมช
7. การนิยามไปป์ไลน์ข้อมูลแบบประกาศ
8. การสร้างแผนภาพด้วยโค้ด
9. อิมเมจที่ไม่มีสิ่งมีชีวิตระดับสูงสุด
10. การดักจับอีเวนท์
11. การเทียบผลลัพธ์แบบคู่ขนาน
12. ใช้กระบวนการหรือวิธีการที่รองรับการทำงานทางไกลอย่างเป็นธรรมชาติ
13. สถาปัตยกรรมแบบไร้ความเชื่อใจ

ประเมิน

14. แพลตฟอร์มชนิดลงมือโค้ดน้อยเฉพาะทาง
15. Browser-tailored polyfills
16. เอกลักษณ์กระจายศูนย์
17. การจัดการบริการบนคลาวด์ด้วย Kubernetes
18. Open Application Model (OAM)
19. Secure enclaves
20. การทดลองแบบสลับไปมา
21. ข้อมูลรับรองตัวตนที่พิสูจน์ได้

เฝ้าระวัง

22. Apollo Federation
23. ESB ในคราบ API Gateway
24. การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ
25. ไมโครฟรอนท์เอนด์เพื่ออนาธิปไตย
26. สมุดโน๊ตเสมือนในงานโปรดักชัน

แพลตฟอร์ม

นำไปใช้

ทดลอง

27. Azure DevOps
28. Debezium
29. Honeycomb
30. JupyterLab

ประเมิน

31. Amundsen
32. AWS Cloud Development Kit
33. Backstage
34. Dremio
35. DuckDB
36. K3s
37. Materialize
38. Pulumi
39. Tekton
40. Trust over IP stack

เฝ้าระวัง

41. Node overload

เครื่องมือ

นำไปใช้

42. Airflow
43. Bitrise
44. Dependabot
45. Helm
46. Trivy

ทดลอง

47. Bokeh
48. Concourse
49. Dash
50. jscodeshift
51. Kustomize
52. MLflow
53. Pitest
54. Sentry
55. ShellCheck
56. Stryker
57. Terragrunt
58. tfsec
59. Yarn

ประเมิน

60. CML
61. Eleventy
62. Flagger
63. gossm
64. Great Expectations
65. k6
66. Katran
67. Kiali
68. LGTM
69. Litmus
70. Opacus
71. OSS Index
72. Playwright
73. pnpm
74. Sensei
75. Zola

เฝ้าระวัง

ภาษาและเฟรมเวิร์ค

นำไปใช้

76. Arrow
77. jest-when

ทดลอง

78. Fastify
79. Immer
80. Redux
81. Rust
82. single-spa
83. Strikt
84. XState

ประเมิน

85. Babylon.js
86. Blazor
87. Flutter Driver
88. HashiCorp Sentinel
89. Hermes
90. io-ts
91. Kedro
92. LitElement
93. Mock Service Worker
94. Recoil
95. Snorkel
96. Streamlit
97. Svelte
98. SWR
99. Testing Library

เฝ้าระวัง

TECHNOLOGY RADAR

เทคนิค

เทคนิค

ฟังก์ชันประเมินความเหมาะสมด้านความเบี่ยงเบนของสิ่งพึ่งพา (Dependency drift fitness function)

นำไปใช้

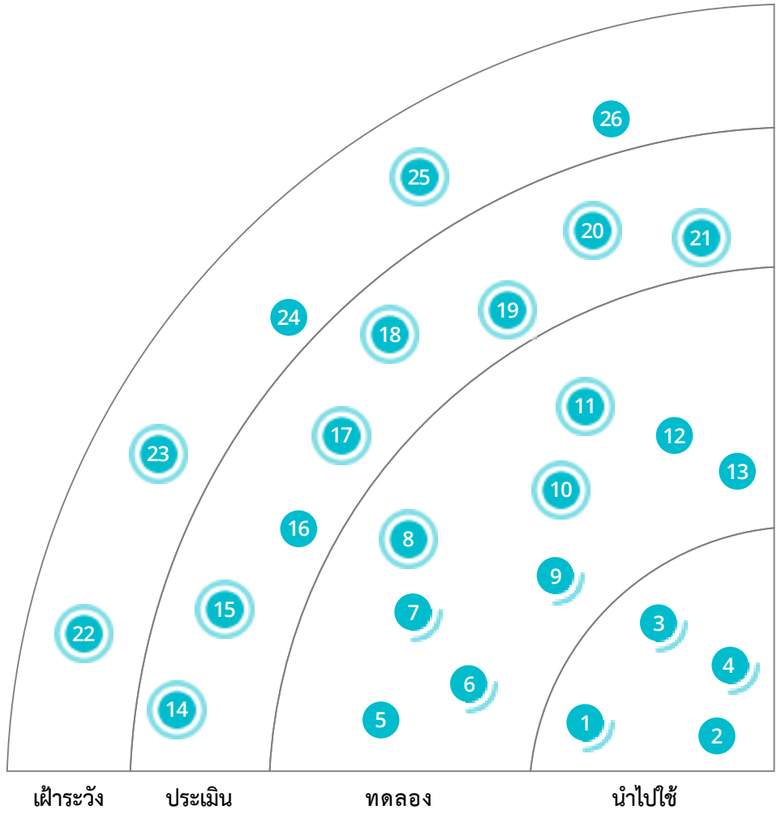
ฟังก์ชันประเมินความเหมาะสม (Fitness functions) เป็นแนวคิดที่ถูกนำเสนอจาก สถาปัตยกรรมเชิงวิวัฒนาการ (evolutionary architecture) ซึ่งถูกยืมมาจาก การคำนวณเชิงวิวัฒนาการ (evolutionary computing) อีกทีหนึ่ง คือ ฟังก์ชันที่สามารถคำนวณหาได้ว่า แอปพลิเคชันหรือสถาปัตยกรรมของเราออกห่างจากคุณลักษณะเด่นที่เราต้องการหรือไม่ ซึ่งในทางปฏิบัติก็มักจะอยู่ในรูปแบบชุดการทดสอบที่เป็นส่วนหนึ่งของไปป์ไลน์ หนึ่งในคุณลักษณะสำคัญของแอปพลิเคชันที่ดี คือคุณลักษณะด้านความทันสมัยของสิ่งที่ระบบเราพึ่งพา เช่น ไลบรารี API หรือคอมโพเนนท์ใด ๆ ดังนั้น ฟังก์ชันประเมินความเหมาะสมด้านความเบี่ยงเบนของสิ่งพึ่งพา จะคอยจับตาและแจ้งเตือนเมื่อพบว่ามีสิ่งพึ่งพาที่เก่าเกินไปอยู่ในระบบ ด้วยปัจจุบันมีเครื่องมือที่มากขึ้นและพร้อมขึ้นที่สามารถตรวจสอบเรื่องนี้ได้ เช่น Dependabot หรือ Snyk เราจึงรวมฟังก์ชันประเมินความเหมาะสมด้านความเบี่ยงเบนของสิ่งพึ่งพา เป็นส่วนหนึ่งของกระบวนการเผยแพร่ซอฟต์แวร์ได้ไม่ยาก เพื่อการแก้ไขจะเกิดขึ้นอย่างทันท่วงที

ฟังก์ชันความเหมาะสมทางสถาปัตยกรรมในด้านรายจ่ายค่าบริหารงาน

นำไปใช้

องค์กรต่าง ๆ ต้องสามารถประเมิน ติดตาม และทำนายค่าใช้จ่ายสำหรับสร้างและดูแลรักษาโครงสร้างพื้นฐานบนคลาวด์ได้อย่างอัตโนมัติ เพราะปัจจุบัน ผู้ให้บริการหลายเจ้าวางกฎเกณฑ์การตั้งราคาไว้อย่างแยบยล เมื่อรวมกับตัวแปรด้านราคาที่มีแต่จะเพิ่มขึ้นเรื่อย ๆ และธรรมชาติของสถาปัตยกรรมสมัยใหม่ที่เพิ่มลดขนาดได้ตลอดเวลา เราอาจจะตกใจได้เมื่อเห็นค่าใช้จ่ายที่ออกมาทีหลัง เพื่อให้เห็นภาพมากขึ้น ขอยกตัวอย่าง เช่น ราคาค่าใช้จ่ายของบริการ serverless ขึ้นอยู่กับจำนวน API ที่ถูกเรียกใช้งาน ส่วนระบบอีเวนต์สตรีมมิ่งราคาจะขึ้นกับปริมาณข้อมูลที่วิ่งผ่าน หรือส่วนคลัสเตอร์เพื่อประมวลผลข้อมูล ราคาจะขึ้นกับจำนวนงานที่เปิดใช้ สิ่งเหล่านี้ล้วนแต่มีความไดนามิกสูง มีโอกาสเปลี่ยนแปลงได้ตลอดเมื่อสถาปัตยกรรมวิวัฒน์ไป

เมื่อทีมของเราต้องบริหารจัดการโครงสร้างพื้นฐานของคลาวด์ เราจะทำกิจกรรมหนึ่งเพื่อกำหนดหา ฟังก์ชันความเหมาะสมทางสถาปัตยกรรมในด้านรายจ่ายค่าบริหารงาน (run cost as architecture fitness function) ตั้งแต่เนิ่น ๆ เสมอ เมื่อเป็นเช่นนั้น ทีมของเราจะสามารถติดตาม เปรียบเทียบ ค่าใช้จ่ายของบริการที่ใช้กับคุณค่าที่เราส่งมอบ เมื่อพวกเขาเห็นว่ามีการเบี่ยงเบนออกจากค่าที่ตั้งไว้ หรือเกินกว่าค่าที่ยอมรับได้ พวกเขาจะพูดคุยตกลงกัน ว่านี่ถึงเวลาที่จะต้องปรับเปลี่ยนสถาปัตยกรรมแล้วหรือยัง ซึ่งการติดตามและการคำนวณค่าใช้จ่ายค่าดูแลรักษา จะถูกสร้างเป็นฟังก์ชันที่ทำงานอย่างอัตโนมัติ



นำไปใช้

1. ฟังก์ชันประเมินความเหมาะสมด้านความเบี่ยงเบนของสิ่งพึ่งพา (Dependency drift fitness function)
2. ฟังก์ชันความเหมาะสมทางสถาปัตยกรรมในด้านรายจ่ายค่าบริหารงาน
3. นโยบายความมั่นคงด้วยโค้ด
4. แม่แบบเซอร์วิสของตนเอง

ทดลอง

5. การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML)
6. ดาต้าเมช (Data mesh)
7. การนิยามไปป์ไลน์ข้อมูลแบบประกาศ
8. การสร้างแผนภาพด้วยโค้ด
9. อิมเมจที่ไม่มีลีนุกซ์ดิสโตร
10. การดักจับอีเวนท์
11. การเทียบผลลัพธ์แบบคู่ขนาน
12. ใช้กระบวนการหรือวิธีการที่รองรับการทำงานทางไกลอย่างเป็นธรรมชาติ
13. สถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero trust architecture)

ประเมิน

14. แพลตฟอร์มชนิดลงมือโค้ดน้อยเฉพาะทาง
15. การเติมโพลีฟิลเฉพาะส่วนที่เบราว์เซอร์ขาด
16. เอกลักษณ์กระจายศูนย์
17. การจัดการบริการบนคลาวด์ด้วย Kubernetes
18. Open Application Model (OAM)
19. Secure enclaves
20. การทดลองแบบสลับไปมา
21. ข้อมูลรับรองตัวตนที่พิสูจน์ได้

เฝ้าระวัง

22. Apollo Federation
23. ESB ในคราบ API Gateway
24. การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ
25. ไมโครฟรอนท์เอนด์เพื่ออนาธิปไตย
26. สมุดโน๊ตเสมือนในงานโปรดักชันนำไปใช้

เทคนิค

ดาต้าเมช (Data mesh) เป็นการปรับมุมมองความคิด โดยตั้งคำถามว่าเราควรจัดการกับข้อมูลวิเคราะห์ขนาดใหญ่อย่างไร โดยพุ่งเป้าไปที่ปัญหาเดียวกันกับที่วิธีการจัดการข้อมูลวิเคราะห์แบบรวมศูนย์ได้พยายามแก้ไขอยู่ แต่อย่างไรก็ตาม มันก็ยังมีช่องว่างทางเครื่องมือที่กว้างมาก ทั้งที่เป็นโอเพนซอร์สก็ดีหรือที่เป็นเชิงพาณิชย์ก็ตาม

(ดาต้าเมช)

นโยบายความมั่นคงด้วยโค้ด

นำไปใช้

ในปัจจุบันภาพรวมของเทคโนโลยีซับซ้อนขึ้นกว่าแต่ก่อนมาก จึงต้องการวิธีการจัดการเรื่องความมั่นคงแบบใหม่ โดยการทำเรื่องนี้ให้มันเป็นอัตโนมัติมากที่สุด และต้องนำหลักปฏิบัติทางวิศวกรรมมาใช้มากขึ้น พอเราจะสร้างระบบใด ๆ ขึ้นมา เราจำเป็นต้องเอาเรื่องนโยบายความมั่นคง (security policy) มาพิจารณาด้วยเสมอ ซึ่งก็คือกฎเกณฑ์และนโยบายที่จะคอยปกป้องระบบของเรา จากภัยคุกคามและการทำลายที่มีโอกาสเกิดขึ้น ยกตัวอย่างเช่น การกำหนดนโยบายการเข้าถึง (access control policy) เป็นการกำหนดว่าใครสามารถที่จะเข้าถึงเซอร์วิส หรือรีซอร์สใดได้บ้าง ในโอกาสใดบ้าง หรือการมีนโยบายความมั่นคงทางเน็ตเวิร์ค (network security policy) ก็จะสามารถควบคุมจำกัดปริมาณข้อมูลของเซอร์วิสใด ๆ ได้

หลายทีมของเราได้ใช้วิธีการกำหนด นโยบายความมั่นคงด้วยโค้ด แล้วได้ผลดี ซึ่งคำว่า *ด้วยโค้ด* เราไม่ได้หมายความว่า จะต้องเขียนนโยบายพวกนี้ลงในไฟล์เท่านั้น แต่รวมความหมายแฝงอื่นด้วย นั่นคือการใช้แนวทางปฏิบัติทางวิศวกรรมต่าง ๆ กับนโยบายความมั่นคง ไม่ว่าจะเป็น การจัดการและควบคุมเวอร์ชันนโยบาย การตรวจสอบความถูกต้องของนโยบายในไปป์ไลน์ หรือการดีพลอยนโยบายลงสภาพแวดล้อมใด ๆ อย่างอัตโนมัติ รวมถึง สามารถเฝ้าสังเกตและติดตามดูประสิทธิภาพของนโยบายที่ติดตั้งไปแล้วด้วย จากประสบการณ์ของเรา และความพร้อมของเทคโนโลยีในกลุ่มนี้ที่ดีมาก —- เช่น Open Policy Agent และแพลตฟอร์มอย่าง Istio ที่รองรับการกำหนดและบังคับใช้นโยบายอย่างยืดหยุ่น ที่เข้ากันดีกับแนวคิด นโยบายความมั่นคงด้วยโค้ด —- เราจึงแนะนำอย่างยิ่ง ให้ใช้เทคนิคนี้กับงานของคุณได้แล้ว

แม่แบบเซอร์วิสของตนเอง (Tailored service templates)

นำไปใช้

จากครั้งที่เราได้พูดถึงเทคนิคการสร้าง แม่แบบเซอร์วิสของตนเอง (Tailored service templates) ไว้ ถึงตอนนี้เราเห็นการประยุกต์ใช้เทคนิคนี้กันมากขึ้น เพื่อใช้ปูทางขององค์กรไปสู่สถาปัตยกรรมไมโครเซอร์วิส จากที่เทคโนโลยีด้านโครงสร้างพื้นฐานมีการพัฒนาอย่างต่อเนื่องไม่หยุด ตั้งแต่เครื่องมือเฝ้าสังเกตการณ์ ระบบควบคุมคอนเทนเนอร์ หรือแม้แต่ไซด์คาร์ของเซอร์วิสเมชก็ดี การมีแม่แบบเซอร์วิสไว้ ช่วยร่นเวลาการติดตั้งเซอร์วิสในส่วนการเชื่อมต่อกับโครงสร้างพื้นฐานต่าง ๆ เพราะได้กำหนดค่าตั้งต้นที่ดี ในตัวเซอร์วิสที่จะสร้างเลย

เราได้ใช้หลักการบริหารผลิตภัณฑ์ภายใน กับแม่แบบเซอร์วิสแล้วได้ผลดี ทำโดยการใส่ใจทีมนักพัฒนาผู้มาใช้งาน เสมือนเป็นลูกค้าคนหนึ่ง บริการให้พวกเขานำโค้ดขึ้นโปรดักชันได้ง่าย ดูแลบริหารกันเองได้ โดยมีเครื่องมือเฝ้าสังเกตการณ์ที่เหมาะสมให้ การใช้แม่แบบเซอร์วิสยังเปิดโอกาสให้เกิดการกำกับดูแลเทคโนโลยีระหว่างกันแบบไม่ต้องลงแรงมาก ซึ่งดีกว่าการใช้หน่วยงานกลางเป็นผู้ตัดสินใจอนุมัติเทคโนโลยีใด ๆ

การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML)

ทดลอง

เมื่อประมาณทศวรรษที่แล้ว เราได้แนะนำให้รู้จักเทคนิคการส่งมอบอย่างต่อเนื่อง (Continuous Deilvery - CD) ซึ่งเป็นวิธีหลักที่เราใช้ส่งมอบโซลูชันซอฟต์แวร์เสมอมา ทุกวันนี้ หลายโซลูชันมีการนำโมเดลแมชชีนเลิร์นนิงเข้ามาใช้ด้วย เราเองก็ไม่เว้นที่จะปรับใช้หลักปฏิบัติ CD กับโมเดลพวกนี้เหมือนกัน ซึ่งเราเรียกเทคนิคนี้ว่า การส่งมอบแมชชีนเลิร์นนิงอย่างต่อเนื่อง (CD4ML) แม้ว่าหลักการของ CD ยังคงเดิม แต่แนวปฏิบัติและเครื่องมือที่ใช้ เพื่อครอบคลุมทั้งกระบวนการของแมชชีนเลิร์นนิง ตั้งแต่การฝึก การทดสอบ การดีพลอย จนไปถึงการติดตามโมเดล นั้นแตกต่าง เช่น การควบคุมเวอร์ชัน ไม่ได้มีแค่ส่วนข้อมูลที่ต้องถูกจัดเก็บ แต่รวมไปถึงตัวโมเดลเอง และตัวแปรพารามิเตอร์ต่าง ๆ ด้วย ส่วนการทดสอบให้ครอบคลุมทั้งสามเหลี่ยมพีระมิด (testing pyramid) ก็ต้องคำนึงถึงเรื่องการทดสอบความลำเอียงของโมเดล (model bias) ความเป็นธรรมของโมเดล (fairness) รวมไปถึงการตรวจสอบความถูกต้องของฟีเจอร์และข้อมูลด้วย ส่วนการดีพลอย ก็ต้องพิจารณาว่าจะเลื่อนขั้นหรือวัดประสิทธิภาพของโมเดลใหม่ เทียบกับโมเดลแชมป์ปัจจุบันอย่างไร ในขณะที่โลกเพิ่งจะตื่นเต้นกับคำว่า MLOps อยู่เลย เราเชื่อว่า CD4ML นั้นเป็นวิธีการที่ครอบคลุมทั้งกระบวนการ เพื่อให้โมเดลแมชชีนเลิร์นนิงมีความน่าเชื่อถือ และสามารถเก่งขึ้นตลอดเวลา ตั้งแต่เริ่มไอเดียจนถึงโปรดักชัน

ดาต้าเมช (Data mesh)

ทดลอง

ดาต้าเมช (Data mesh) เป็นการปรับมุมมองความคิดทางสถาปัตยกรรมและองค์กร โดยตั้งคำถามว่าเราควรจัดการกับข้อมูลวิเคราะห์ขนาดใหญ่อย่างไรให้ดีกว่าที่เป็นอยู่ ซึ่งแนวคิดนี้ประกอบไปด้วยหลักการ 4 ข้อด้วยกัน ได้แก่ (1) ให้กระจายความเป็นเจ้าของข้อมูล โดยกระจายสถาปัตยกรรมตามโดเมนของข้อมูล (2) ให้โดเมนใด ๆ บริการข้อมูลเสมือนเป็นผลิตภัณฑ์ของโดเมนนั้น (3) มีแพลตฟอร์มโครงสร้างพื้นฐาน ให้ใครมาใช้ข้อมูลได้สะดวก เพื่อส่งเสริมให้เกิดทีมตามโดเมนที่สามารถบริหารดูแลตนเองได้ (4) กำกับดูแลข้อมูลร่วมกัน เพื่อส่งเสริมให้เกิดระบบนิเวศและการแลกเปลี่ยนข้อมูลระหว่างโดเมน แม้ว่าหลักการเหล่านี้จะฟังดูตรงตัว และพุ่งเป้าไปที่ปัญหาเดียวกันกับที่วิธีการจัดการข้อมูลวิเคราะห์แบบรวมศูนย์ได้พยายามอยู่ก็ตาม แต่ผลลัพธ์ที่ได้กลับต่างกันมาก หลังจากที่พวกเราได้พัฒนาดาต้าเมชให้กับลูกค้าหลาย ๆ เจ้า โดยต่อยอดจากเครื่องมือเดิมที่มีอยู่ เราได้เรียนรู้ว่า (ก) ยังมีช่องว่างทางเครื่องมือที่กว้างมาก ทั้งที่เป็นโอเพนซอร์สก็ดีหรือที่เป็นเชิงพาณิชย์ก็ตาม ที่จะช่วยเร่งการพัฒนาดาต้าเมชให้เร็วยิ่งขึ้น ขนาดเรายังต้องพัฒนาเครื่องมือเฉพาะให้แก่ลูกค้าหลายส่วน และ (ข) แม้ว่าจะมีช่องว่างอยู่ก็ตาม แต่มันก็เป็นไปได้ที่จะใช้เทคโนโลยีที่มีอยู่ในปัจจุบันเป็นพื้นฐานเพื่อต่อยอด

จริงที่ การเลือกเทคโนโลยีที่เหมาะสม เป็นส่วนประกอบหลักของการพัฒนากลยุทธ์ข้อมูลบนดาต้าเมชขององค์กร แต่จะทำให้สำเร็จได้นั้น ต้องปรับโครงสร้างองค์กรเพื่อกระจายทีมดาต้าแพลตฟอร์มออกไป แล้วสร้างบทบาทหน้าที่ เจ้าของผลิตภัณฑ์ข้อมูล (data product owner) ประจำแต่ละโดเมนขึ้นมา โดยวางโครงสร้างระบบผลตอบแทนที่จำเป็น เพื่อให้แต่ละโดเมนเกิดความรู้สึกอยากเป็นเจ้าของ และอยากจะแบ่งปันข้อมูลเชิงวิเคราะห์

เสมือนว่าข้อมูลนี้เป็นผลิตภัณฑ์ของตน

การนิยามไปป์ไลน์ข้อมูลแบบประกาศ
ทดลอง

ไปป์ไลน์ข้อมูลส่วนมาก จะเขียนเป็นไฟล์สคริปต์ขนาดใหญ่ และนิยมเขียนกันด้วยภาษา Python หรือ Scala โดยจะอธิบายขั้นตอนการทำงานด้วยโค้ด ทำให้ภายในสคริปต์เองจะปนไปด้วยโค้ดที่เล่าวิธีการทำงานแต่ละขั้นตอน และโค้ดที่จะร้อยขั้นตอนต่าง ๆ เข้าด้วยกัน ซึ่งจัดการได้ยุ่งยากมาก สถานการณ์นี้เคยเกิดขึ้นกับคนที่เคยเขียนชุดทดสอบกับ Selenium มาแล้ว ในตอนนั้นนักพัฒนาได้ค้นพบวิธีการจัดระเบียบโค้ดด้วยท่าเพจออบเจกต์ (Page Object pattern) และในภายหลังเฟรมเวิร์คในกลุ่มบีดีดี (Behavior-Driven Development, BDD) ก็ได้เริ่มแยกส่วนที่ร้อยเรียงขั้นตอนต่าง ๆ กับส่วนที่อธิบายวิธีการทำงานแต่ละขั้นตอนออกจากกันมาให้ในตัว

ปัจจุบันเริ่มมีบางทีมกำลังทดลองนำแนวคิดเดียวกันมาสู่วิศวกรรมข้อมูลแล้ว เกิดเป็นแนวคิด การนิยามไปป์ไลน์ข้อมูลแบบประกาศ ซึ่งการนิยามอาจประกาศในรูปแบบไฟล์ YAML ที่จะระบุเพียงขั้นตอนและลำดับการทำงานเท่านั้น มีการระบุชุดข้อมูลขาเข้าและขาออกที่แต่ละขั้นตอนต้องการ และจะอ้างอิงไปที่สคริปต์ก็ต่อเมื่อกรณีนั้นมีตรรกะที่ซับซ้อนเท่านั้น ยกตัวอย่าง เช่น A La Mode เป็นเครื่องมือที่เพิ่งออกมาได้ไม่นาน และสร้างภาษาเฉพาะ (DSL - Domain Specific Language) ขึ้นมา เพื่อใช้กำหนดไปป์ไลน์ ส่วน airflow-declarative ก็น่าสนใจเช่นกัน เพราะสามารถสร้างกราฟแบบไม่วนทิศ (DAG - Directed Acyclic graphs) ด้วยการประกาศในไฟล์ YAML แล้วแปลงเป็นตัวควบคุมการทำงานของ Airflow ได้ทันที ซึ่งเครื่องมือตัวนี้ดูเหมือนจะเป็นที่นิยมกว่าใครเพื่อนในตอนนี้

การสร้างแผนภาพด้วยโค้ด
ทดลอง

เราเริ่มเห็นเครื่องมือที่สามารถวาดสถาปัตยกรรมซอฟต์แวร์และ แผนภาพด้วยโค้ด ออกมามากขึ้น การใช้เครื่องมือประเภทนี้มีข้อดีเหนือกว่าการใช้เครื่องมือแผนภาพแบบจัดเต็มทั่ว ๆ ไป ตรงที่มันควบคุมเวอร์ชันได้ง่าย และสามารถสร้างและรวม DSL จากหลายแหล่งเข้าด้วยกัน เครื่องมือในกลุ่มนี้ที่เราชอบ ก็มี Diagrams, Structurizr DSL AsciiDoctor Diagram และพวกที่เสถียรไว้ใจได้ ก็มี WebSequenceDiagrams, PlantUML โดยมี Graphviz เป็นตัวที่อาวุโสที่สุดในกลุ่ม ทั้งนี้การสร้างรูป SVG ด้วยตัวเองไม่ใช่เรื่องยากเหมือนสมัยก่อนแล้ว ฉะนั้นอย่าตัดทางเลือกการสร้างเครื่องมือขึ้นมาใช้เองออกไป หนึ่งในผู้เขียนยังเคยเขียนสคริปต์ Ruby เพื่อสร้างรูป SVG ขึ้นมาใช้เองด้วยเหมือนกัน

อิมเมจที่ไม่มีลีนุกซ์ดิสโตร
ทดลอง

เมื่อเราจะสร้าง Docker อิมเมจเพื่อครอบแอปพลิเคชันของเรา จะมีอยู่สองเรื่องที่เราต้องให้ความสำคัญ คือ เรื่องความมั่นคง และเรื่องขนาดของอิมเมจ โดยปกติเราจะใช้เครื่องมือ พิเคราะห์คอนเทนเนอร์เพื่อความมั่นคง (container security scanning) เพื่อตรวจจับและอุดช่องโหว่ต่าง ๆ ที่ถูกเปิดเผย และใช้ลีนุกซ์ดิสโตรขนาดเล็กเป็นพื้นฐาน เช่น Alpine Linux เพื่อจัดการเรื่องขนาดและทรัพยากร จากที่ได้ใช้ อิมเมจที่ไม่มีลีนุกซ์ดิสโตร (distroless Docker images) มามากขึ้นจนมั่นใจ เราก็พร้อมจะแนะนำให้ไปใช้ต่อ เพื่อเป็นเกราะป้องกันความมั่นคงอีกชั้นหนึ่งให้กับแอปพลิเคชัน เทคนิคนี้ช่วยลดข้อผิดพลาดของการตรวจพิเคราะห์ ลดพื้นที่การโจมตีให้แคบลง แถมยังมีช่องโหว่ให้อุดน้อยลงด้วย ส่วนขนาดที่เล็กลงก็ทำให้ประหยัดทรัพยากรได้มากขึ้น Google ได้เตรียม คอนเทนเนอร์อิมเมจสำหรับดิสโตรเลส ของแต่ละภาษาไว้ให้ใช้ หรือคุณจะสร้างดิสโตรเลสเอง ผ่านเครื่องมืออย่าง Bazel ของ Google หรือจะใช้การสร้างมัลติสเตจใน Docker ก็ได้ หมายเหตุไว้ว่า ถ้าใช้อิมเมจที่ไม่มีลีนุกซ์ดิสโตรจะไม่มีเชลล์ติดมาด้วย ทำให้ดีบั๊กได้ยาก อย่างไรก็ดี มันแก้ได้ไม่ยากโดยการหาอิมเมจเวอร์ชันที่ดีบั๊กได้ในโลกออนไลน์ เช่น BusyBox shell ทั้งนี้ การใช้อิมเมจที่ไม่มีลีนุกซ์ดิสโตรติดมาด้วย เป็นเทคนิคที่ Google เป็นผู้บุกเบิกมาตั้งแต่ต้น และจากประสบการณ์ของเราพบว่าอิมเมจส่วนมากยังคงมีที่มาจาก Google อยู่ เราหวังว่าเทคนิคนี้จะถูกใช้มากขึ้นนอกระบบนิเวศนี้

การดักจับอีเวนท์
ทดลอง

จากที่หลาย ๆ บริษัทกำลังหาทางย้ายออกจากระบบเก่าไปสู่ระบบใหม่ เราจึงอยากแนะนำทางเลือกอื่นนอกจากเทคนิคการดักจับข้อมูลที่เปลี่ยนแปลง (change data capture - CDC) ไว้ด้วย อันเป็นเทคนิคหนึ่งที่เราเคยแนะนำไว้แล้ว แต่เมื่อปี 2004 Martin Fowler เคยให้นิยามอีกเทคนิคหนึ่งเอาไว้ คือเทคนิค การดักจับอีเว้นท์ (event interception) ซึ่งเป็นการดักจับรีเควสต์ที่เข้ามาสู่ระบบเก่าแล้วส่งไปที่ใหม่ เพื่อที่จะค่อย ๆ แทนที่ของเก่าด้วยของใหม่ได้ในที่สุด ซึ่งในทางปฏิบัติมันทำได้ ด้วยการคัดลอกชุดของอีเวนท์ หรือข้อความ ออกมาเป็นอีกชุดหนึ่ง หรือหากเป็นโปรโตคอล HTTP ก็ใช้การดักจับรีเควสต์ที่เข้ามาได้เช่นเดียวกัน ยกตัวอย่างเช่น ให้คัดลอกชุดของอีเวนท์จากระบบหนึ่งก่อนที่คำสั่งจะถูกใช้ในการแก้ไขเมนเฟรม หรือการคัดลอกรายการชำระเงินก่อนที่จะถูกเขียนลงฐานข้อมูลในระบบแกนหลักของแบงค์ (core banking) ซึ่งทั้งสองตัวอย่างนำไปสู่วิธีการที่เราสามารถค่อย ๆ ถอดเอาระบบเก่าออกไปได้ เราเห็นว่าเทคนิคการดักจับการเปลี่ยนแปลงตั้งแต่ต้นทางเช่นนี้ ไม่ค่อยเป็นที่พูดถึงเท่าไหร่นัก เป็นเหตุผลว่าทำไมเราถึงอยากจะยกมันมาพูดถึงในฉบับนี้

การเทียบผลลัพธ์แบบคู่ขนาน
ทดลอง

การจะย้ายระบบเก่าขนาดใหญ่ไปใหม่ไม่เคยเป็นเรื่องง่ายเลย แต่วิธีหนึ่งที่ประสิทธิภาพสูงมากคือ การใช้เทคนิค การเทียบผลลัพธ์แบบคู่ขนาน (Parallel run with Reconcillation) กล่าวคือ เพื่อจะพิสูจน์ว่าโค้ดชุดเก่ากับโค้ดชุดใหม่ทำงานได้ผลลัพธ์ตรงกัน ให้เรานำโค้ดทั้งสองชุดมาประมวลผลด้วยข้อมูลชุดเดียวกันไปพร้อม ๆ กัน แล้วดูผลลัพธ์สุดท้ายเปรียบเทียบกัน แม้เทคนิคนี้จะมีมานานแล้ว แต่เราก็เริ่มเห็นการประยุกต์ใช้กับเครื่องมือใน

เทคนิค

เราเริ่มเห็นเครื่องมือที่สามารถวาดสถาปัตยกรรมซอฟต์แวร์ และแผนภาพด้วยโค้ด ออกมามากขึ้น ข้อดีคือมันควบคุมเวอร์ชันได้ง่าย และสามารถสร้างและรวม DSL จากหลายแหล่งเข้าด้วยกัน

(การสร้างแผนภาพด้วยโค้ด)

การดักจับอีเวนท์ เป็นอีกทางเลือกที่น่าสนใจเมื่อต้องการดึงข้อมูลออกจากระบบเก่าไปสู่ระบบใหม่ ซึ่งเทคนิคการดักจับการเปลี่ยนแปลงตั้งแต่ต้นทาง แทนที่จะไปสร้างข้อมูลขึ้นมาใหม่โดยอาศัยการดักจับข้อมูลที่เปลี่ยนแปลง (change data capture - CDC) นั้น มักจะไม่ค่อยเป็นที่พูดถึงเท่าไหร่นัก

(การดักจับอีเวนท์)

เทคนิค

แพลตฟอร์มจำพวกที่ไม่ต้องเขียนโค้ด (no-code platform) หรือเขียนบ้างแต่น้อย (low-code platform) มักจะถูกออกแบบมาเพื่อแก้ปัญหาเฉพาะทางในโดเมนที่จำกัดมาก กระนั้นแล้ว เรายังเคลือบแคลงความสามารถของมันอยู่ว่าจะทำได้ดีแค่ไหนหากปรับใช้ในวงกว้าง

(แพลตฟอร์มชนิดลงมือโค้ดน้อยเฉพาะทาง)

กลุ่ม CD ต่าง ๆ เช่น การจะทำแคนารีรีลีส หรือการเปิด/ปิดฟีเจอร์ใด ๆ (feature toggles) ก็จะใช้เทคนิคนี้ประกอบด้วยเพื่อทำการทดลองและวิเคราะห์ข้อมูลโดยใช้ข้อมูลจริงประกอบ นอกจากนี้ เรายังเคยใช้เทคนิคนี้เพื่อเปรียบเทียบข้อมูลในเชิงคุณภาพอีกด้วย เช่น การเปรียบเทียบความเร็วในการทำงาน แม้เราจะเคยใช้ท่านี้หลายต่อหลายครั้ง ผ่านเครื่องมือที่เราทำขึ้นมาเอง แต่เครื่องมือ อย่าง Scientist ของ GitHub ก็ดีมากจนเราต้องปรบมือให้ พวกเขาใช้เครื่องมือนี้ปรับชิ้นส่วนสำคัญมากในระบบภายในของตัวเองได้สำเร็จ ปัจจุบันเครื่องมือตัวนี้ได้รับความนิยมและถูกคัดลอกไปสู่ภาษาอื่น ๆ ด้วย

ใช้กระบวนการหรือวิธีการที่รองรับการทำงานทางไกลอย่างเป็นธรรมชาติ

ทดลอง

ขณะที่โรคระบาดยังยืดเยื้อต่อไป การทำงานแบบกระจายตัวกัน ดูเหมือนจะเป็นสิ่งธรรมดาใหม่ไปเสียแล้ว อย่างน้อยก็สำหรับช่วงเวลานี้ ตลอดช่วงหกเดือนที่ผ่าน เราเองก็ได้เรียนรู้อย่างมาก ถึงวิธีทำงานทางไกลอย่างไรให้มีประสิทธิภาพ ซึ่งในทางที่ดี มันผลักดันให้เราค้นพบเครื่องมือใหม่ ๆ ที่จะมาช่วยให้การทำงานร่วมกันแบบทางไกลสะดวกขึ้นกว่าเดิม เช่น นักพัฒนาสามารถไว้ใจให้ Visual Studio Live Share และ GitHub Codespaces ช่วยอำนวยความสะดวกให้ทำงานกันเป็นทีม และเพิ่มผลิตผลของงานส่วนในทางที่ไม่ค่อยดีนัก การทำงานทางไกลก็ทำให้รู้สึกหมดไฟได้เหมือนกัน มีหลายต่อหลายคนที่ถูกนัดประชุมทางไกลติด ๆ กันทั้งวัน พอมากเกินไปมันก็ส่งผลกระทบที่ไม่ดีตามมา แม้เครื่องมือออนไลน์ที่ช่วยแสดงภาพการทำงานจะทำให้การร่วมมือกันง่ายขึ้นจริง แต่ความซับซ้อนก็เกิดง่ายขึ้นด้วย อีกทั้งในการเผยแพร่งานจากเครื่องมือพวกนี้ก็อาจมีประเด็นเรื่องความมั่นคงให้ต้องระวัง คำแนะนำของเราคือเราอยากให้หยุดคิด และคุยกันในทีมให้มากขึ้น ประเมินดูว่าอะไรดีและอะไรไม่ดี และปรับกระบวนการหรือเครื่องมือตามที่เห็นสมควร

สถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero trust architecture)

ทดลอง

ขณะที่เทคโนโลยีคอมพิวเตอร์และระบบข้อมูลในองค์กรเปลี่ยนไปไม่หยุดนิ่ง — จากสถาปัตยกรรมโมโนลิธ สู่ไมโครเซอร์วิส จากการจัดเก็บข้อมูลรวมศูนย์ด้วยดาต้าเลค สู่ การเก็บข้อมูลกระจายศูนย์ด้วยดาต้าเมช (data mesh) จากโฮสต์ที่ตั้งอยู่ในองค์กร สู่การโฮสบนคลาวด์ พร้อมกันหลายเจ้า และจากความพร้อมของอุปกรณ์เชื่อมต่อต่าง ๆ ที่มากขึ้นทุกวัน — แต่วิธีที่หลายองค์กรใช้รักษาความมั่นคงกลับไม่ค่อยเปลี่ยนไปจากเดิมเลย ยังคงพึ่งพิงเทคนิคการกำหนดขอบเขตเน็ตเวิร์คความไว้ใจอย่างหนัก โดยการลงทุนปกป้องของหวงเหล่านี้ด้วยการทำให้ไฟร์วอลล์หนาขึ้น ใช้ท่อสัญญาณส่วนตัว และการตั้งค่าระบบไฟร์วอลล์ที่จุกจิก รวมถึงการตั้งกฎและกระบวนการรักษาความมั่นคงที่ยุ่งยากไม่ยืดหยุ่น ซึ่งล้าหลังไม่เหมาะสมกับโจทย์ที่เจอในสถานการณ์ปัจจุบันไปเสียแล้ว ทั้งหมดนี้ทำให้พวกเราอยากกล่าวถึงสถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero Trust Architecture - ZTA) อีกครั้งหนึ่ง

ZTA เป็นการเปลี่ยนมุมมองครั้งใหญ่ของการวางสถาปัตยกรรมและกลยุทธ์ด้านความมั่งคง ที่วางอยู่บนความเชื่อว่า เราไม่สามารถใช้ขอบเขตเน็ตเวิร์คความไว้ใจ เป็นตัวแทนขอบเขตความมั่นคงได้อีกต่อไป และไม่ควรใช้ความไว้ใจโดยปริยายใดเป็นตัวกำหนดสิทธิ์ของผู้ใช้หรือเซอร์วิส เพียงแค่พวกเขาอยู่ภายใต้เน็ตเวิร์คหรือสถานที่เดียวกัน ปัจจุบันมีทรัพยากรและเครื่องมือต่างๆ มากขึ้น ที่จะทำให้ ZTA เป็นจริงในแง่มุมต่าง ๆ เช่น เราสามารถกำหนด นโยบายความมั่นคงด้วยโค้ด โดยใช้หลักการตั้งค่าสิทธิ์ของผู้ใช้เฉพาะที่จำเป็นที่สุด ให้ละเอียดที่สุดเท่าที่จะเป็นไปได้ ที่ติดตามการใช้งานได้ตลอด และสามารถผ่อนภัยต่าง ๆ ที่เข้ามาได้อย่างอัตโนมัติ หรือใช้สถาปัตยกรรมเซอร์วิสเมช (service mesh) เพื่อควบคุมการติดต่อกันแบบจุดต่อจุดเท่านั้น ระหว่างแอปพลิเคชันถึงเซอร์วิส หรือระหว่างเซอร์วิสด้วยกันเอง หรือติดตั้ง ตัวพิสูจน์ไบนารีไฟล์ เพื่อยืนยันแหล่งกำเนิดของไบนารีไฟล์นั้นได้ หรือใช้เทคนิค การล้อมกรอบความมั่นคง เสริมกับวิธีการเข้ารหัสแบบทั่วไป เพื่อบังคับให้ข้อมูลมั่นคงเสมอให้ครบทั้งสามเสาหลัก อันได้แก่ ระหว่างเดินทาง ขณะอยู่กับที่ และเมื่ออยู่ในหน่วยความจำ สำหรับการหาข้อมูลเพิ่มเติมเกี่ยวกับ ZTA เราแนะนำให้คุณศึกษางานตีพิมพ์ของ NIST ZTA และของ Google บนเว็บไซต์ BeyondProd

แพลตฟอร์มชนิดลงมือโค้ดน้อยเฉพาะทาง

ประเมิน

หลาย ๆ องค์กรขณะนี้ต้องเผชิญหน้ากับการตัดสินใจที่ดูเหมือนเล็กแต่มีผลอย่างมากต่ออนาคต นั่นคือการตัดสินใจว่าจะใช้แพลตฟอร์มจำพวกที่ไม่ต้องเขียนโค้ด (no-code platform) หรือเขียนบ้างแต่น้อย (low-code platform) หรือไม่ ซึ่งแพลตฟอร์มประเภทนี้จะออกแบบมาเพื่อแก้ปัญหาเฉพาะทางในโดเมนที่จำกัดมาก ปัจจุบันมีผู้ให้บริการจำนวนมากพยายามผลักดันผู้ใช้ไปในทิศทางนี้อย่างหนัก ความกังวลที่เรามีต่อแพลตฟอร์มเหล่านี้ คือ ความยากในการประยุกต์ใช้หลักปฏิบัติทางวิศวกรรมที่ดี อย่างเช่น การทำเวอร์ชัน หรือการทดสอบ แต่อย่างไรก็ตาม เราสังเกตเห็นผู้เล่นหน้าใหม่ที่น่าสนใจในตลาดนี้เช่นกัน ทั้ง Amazon Honeycode ที่เป็นแพลตฟอร์มช่วยให้การสร้างแอปพลิเคชันเอาไว้จัดการอีเวนท์หรืองานง่าย ๆ ทำได้โดยไม่ต้องเขียนโค้ด และ

Parabola ที่เป็นแพลตฟอร์มจัดการขั้นตอนงานบนคลาวด์ ที่มีลักษณะความสามารถคล้าย ๆ กับ IFTTT ทั้งหมดนี้จึงเป็นสาเหตุให้เราหยิบยกเรื่อง แพลตฟอร์มชนิดลงมือโค้ดน้อยเฉพาะทาง (bounded low-code platform) มาไว้ในเรดาร์ฉบับนี้ กระนั้นแล้ว เรายังเคลือบแคลงความสามารถของมันอยู่ว่าจะทำได้ดีแค่ไหนหากปรับใช้ในวงกว้าง ด้วยที่ว่าเครื่องมือเหล่านี้เปรียบเหมือนกับผักตบชวาในประเทศไทย ที่สามารถเจริญเติบโตออกนอกกรอบได้เสมอ และพร้อมจะพันเกี่ยวทุกสิ่งรอบตัวเข้าด้วยกัน นั่นเป็นเหตุที่เราต้องเตือนให้ระมัดระวังอย่างมากหากจะนำแพลตฟอร์มประเภทนี้มาใช้งาน

การเติมโพลีฟิลเฉพาะส่วนที่เบราว์เซอร์ขาด (Browser-tailored polyfills)

ประเมิน

โพลีฟิล (Polyfill) เป็นเทคนิคการเติมฟังก์ชันสมัยใหม่ให้กับเบราว์เซอร์ที่ยังไม่มี ซึ่งมีประโยชน์อย่างสูงต่อการพัฒนาเว็บให้ก้าวไปข้างหน้าได้โดยไม่ต้องพะวงหลัง แต่บ่อยครั้งที่เว็บแอปพลิเคชันได้เติมโพลีฟิลให้กับเบราว์เซอร์รุ่นใหม่ที่ไม่ต้องการมัน เป็นการสิ้นเปลืองการดาวน์โหลดและการประมวลผลโดยไม่จำเป็น ข้อเสียนี้เริ่มเด่นชัดขึ้นจากสถานการณ์ในปัจจุบันที่เหลือเบราว์เซอร์เอนจิ้นในตลาดเพียงไม่กี่ตัว การใช้โพลีฟิลก็มุ่งประสงค์ไปหาเอนจิ้นเดียวเท่านั้น คือเอนจิ้น Trident ของ IE11 หนำซ้ำ ส่วนแบ่งทางการตลาดของ IE11 ก็กำลังถดถอย เนื่องจาก การสนับสนุนที่กำลังจะหยุดลง ในอีกไม่ถึงปีข้างหน้า

ดังนั้น พวกเราจึงแนะนำให้ผู้พัฒนาหันมาใช้เทคนิคการเติมโพลีฟิลเฉพาะส่วนที่ขาดของเบราว์เซอร์นั้น (browser-tailored polyfills) จะดีกว่า ซึ่งเทคนิคนี้สามารถทำเป็นระบบบริการได้ด้วยซ้ำ เช่น บริการของ Polyfill.io

เอกลักษณ์กระจายศูนย์

ประเมิน

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

การจัดการบริการบนคลาวด์ด้วย Kubernetes

ประเมิน

ผู้ให้บริการคลาวด์เจ้าต่าง ๆ เริ่มทยอยรองรับการเรียกใช้บริการของตน ผ่าน API ของ Kubernetes กันมากขึ้นเรื่อยๆ ด้วยเทคนิคการนิยามรีซอร์สส่วนเสริม (custom resource definitions - CRD) โดยส่วนใหญ่ บริการคลาวด์เหล่านี้จะเป็นแกนสำคัญของระบบโครงสร้างพื้นฐาน ซึ่งมักจะเห็นการใช้เครื่องมือ อย่าง Terraform หรือ Pulumi ในการสร้าง แต่ด้วย CRD ประเภทนี้ อันได้แก่ (ACK ของ AWS หรือ Azure Service Operator ของ Azure และ Config Connectors ของ GCP) คุณสามารถสร้างและจัดการบริการคลาวด์ต่าง ๆ ผ่าน Kubernetes โดยตรงได้เลย ข้อดีหนึ่งของ การจัดการบริการบนคลาวด์ด้วย Kubernetes คือ คุณสามารถใช้ประโยชน์จากส่วนควบคุมของ Kubernetes ตัวเดียวกัน ในการควบคุมสถานะของระบบ ทั้งส่วนแอปพลิเคชันและระบบโครงสร้างพื้นฐาน ส่วนข้อเสียคือ มันจะเป็นการผูกคลัสเตอร์ Kubernetes ของคุณเข้ากับระบบโครงสร้างพื้นฐานอย่างแน่นหนา ดังนั้น เราจึงกำลังประเมินวิธีการนี้อย่างระมัดระวัง และคุณเองก็ควรจะทำเช่นเดียวกัน

Open Application Model (OAM)

ประเมิน

บ่อยครั้งที่เราได้พูดถึงประโยชน์ของการตั้งทีมผลิตภัณฑ์วิศวการแพลตฟอร์ม (platform engineering product teams) เพื่อสนับสนุนทีมพัฒนาผลิตภัณฑ์อื่น ๆ แต่ในทางปฏิบัติมันก็ไม่ได้ทำกันง่าย ๆ อยู่ดี มีหลายสิ่งที่วงการนี้พยายามหาทางปรับปรุงให้ง่ายขึ้น หนึ่งในนั้นคือการกำหนดโครงสร้างพื้นฐานด้วยโค้ด

แม้เครื่องมืออย่าง Terraform และ Helm จะเดินมาถูกทางแล้ว แต่เครื่องมือพวกนี้ก็ยังมุ่งสนใจกับการจัดการโครงสร้างพื้นฐาน มากกว่าเรื่องการพัฒนาแอปพลิเคชันอยู่ดี เราจึงเห็นการขยับตัวเข้าสู่แนวคิด การกำหนดโครงสร้างพื้นฐานด้วยซอฟต์แวร์ (infrastructure as software) กันบ้างแล้ว ผ่านเครื่องมืออย่าง Pulumi และ CDK ที่เพิ่งออกมา Open Application Model (OAM) นั้นเป็นความพยายามที่จะนำมาตรฐานมาสู่เครื่องมือกลุ่มนี้ โดยตั้งนิยามแนวคิดต่าง ๆ เช่น คอมโพเนนท์ คอนฟิก สโคป และเทรต ไว้ให้นักพัฒนาสามารถอธิบายแอปพลิเคชันของตนโดยไม่ขึ้นกับแพลตฟอร์มใด ๆ ส่วนคนพัฒนาแพลตฟอร์มก็สามารถอธิบายแพลตฟอร์มของตนในรูปแบบแนวคิด เวิร์คโหลด สโคป และเทรต ได้เช่นกัน

คงต้องติดตามกันต่อไปว่า มาตรฐาน OAM จะเป็นที่ยอมรับแพร่หลายแค่ไหน แต่เราขอแนะนำให้จับตามองแนวคิดอันน่าสนใจและกำลังเป็นที่ต้องการนี้ไว้ให้ดี

กรอบล้อมความมั่นคง (Secure enclaves)

ประเมิน

กรอบล้อมความมั่นคง หรือที่เรียกกันว่า สภาพแวดล้อมการประมวลผลที่ปลอดภัย (Trusted Execution Environments - TEE) เป็นเทคนิคการแยกสภาพแวดล้อม — ตัวประมวลผล หน่วยความจำ และที่เก็บข้อมูล — ออกจากส่วนอื่น ให้ภายในมีระดับความมั่นคงอย่างแน่นหนา โดยจำกัดการแลกเปลี่ยนข้อมูลกับระบบแวดล้อมภายนอกที่ไม่เชื่อถือ ยกตัวอย่าง เช่น หากมีกรอบล้อมความมั่นคงที่ระดับฮาร์ดแวร์และระบบปฏิบัติการ จะสามารถสร้างและเก็บกุญแจส่วนตัว (private key) ไว้ภายใน เพื่อประมวลงานที่เกี่ยวข้องกับกุญแจนั้น เช่น การเข้ารหัส หรือการ

เทคนิค

ทุกวันนี้ ข้อมูลรับรองตัวตนดิจิทัลส่วนใหญ่เป็นเพียงแค่บันทึกข้อมูลง่าย ๆ ในระบบสารสนเทศ ซึ่งง่ายต่อการแก้ไขปลอมแปลง และบ่อยครั้งก็เปิดเผยข้อมูลที่ไม่จำเป็นออกมาด้วย ช่วงไม่กี่ปีที่ผ่านมา เราเห็นถึงความพร้อมขึ้นเรื่อย ๆ ของวิธีที่จะมาแก้ปัญหานี้ ที่เรียกกันว่า ข้อมูลรับรองตัวตนที่พิสูจน์ได้ (Verifiable Credentials)

(ข้อมูลรับรองตัวตนที่พิสูจน์ได้)

เทคนิค

ไม่ว่าเครื่องมือนั้นจะชื่อว่าอะไรก็ตาม แต่การนำตรรกะทางธุรกรรมไปรวมศูนย์ไว้ในเครื่องมือหนึ่งเป็นความคิดที่ไม่ดีเลย มีแต่จะสร้างพันธะผูกพัน ขาดความโปร่งใส และทำให้หลุดไม่พ้นจากผู้ให้บริการรายนั้น โดยไม่มีข้อดีที่ชัดเจนอะไรเลย

(ESB ในคราบ API Gateway)

พิสูจน์ซิกเนเจอร์ โดยที่กุญแจส่วนตัวไม่หลุดออกนอกกรอบ หรือถูกนำเข้าไปที่หน่วยความจำของแอปพลิเคชันที่ไม่ไว้ใจเลย โดยที่กรอบล้อมความมั่นคงจะมีชุดคำสั่งที่จำกัดให้ใช้ เพื่อสั่งให้ทำงานที่ไว้ใจแยกออกจากส่วนที่ไม่ไว้ใจ

เทคนิคนี้ไม่ใช่เรื่องใหม่แต่อย่างใด หลาย ๆ อุปกรณ์ฮาร์ดแวร์และระบบปฏิบัติการ ต่างก็รองรับเทคนิคนี้ เช่น Apple แต่ส่วนใหญ่จะเห็นการใช้งานในอุปกรณ์ IoT และแอปพลิเคชันที่ประมวลผลอยู่ไกลศูนย์ (edge application) แต่เพียงไม่นานนี้ เทคนิคนี้เริ่มได้รับความสนใจเพื่อใช้กับงานองค์กรและกับแอปพลิเคชันบนระบบคลาวด์มากขึ้น ทำให้ผู้ให้บริการคลาวด์หลายเจ้า เริ่มนำเสนอบริการด้าน การประมวลผลแบบไว้วางใจได้ (confidential computing) ออกมา เช่น กรอบล้อมความมั่นคงโดยใช้ฮาร์ดแวร์ของ Azure หรือ Azure confidential computing infrastructure จะรองรับเวอร์ชวลแมชชีนที่เปิดใช้งาน TEE และสามารถใช้งานกรอบได้ผ่านไลบรารี Open Enclave SDK ที่เป็นโอเพนซอร์ส ส่วนฝั่ง GCP ก็มีเช่นกัน ผ่านบริการ GCP Confidential VMs and Compute Engine ที่เพิ่งเปิดทดลองในโหมดเบต้าอยู่ขณะนี้ โดยยอมให้ใช้เวอร์ชวลแมชชีนกับการเข้ารหัสข้อมูลในหน่วยความจำ หรือบริการ AWS Nitro Enclaves ก็มีบริการคล้าย ๆ กันกำลังจะเปิดให้ลองใช้บริการเร็ว ๆ นี้ จากการนำเสนอบริการลักษณะนี้บนคลาวด์ กับการประมวลผลแบบลับ (confidential computing) ทั้งหมดนี้ทำให้คุณสามารถยกระดับการปกป้องข้อมูลครบทุกองค์ประกอบ ทั้งตอนเก็บอยู่ ระหว่างรับ-ส่ง และตอนอยู่ในหน่วยความจำ

แม้ปัจจุบันจะเป็นเพียงช่วงเริ่มต้นของการใช้กรอบล้อมความมั่นคงกับงานระดับองค์กร เราขอส่งเสริมให้พิจารณาใช้เทคนิคนี้ ขณะเดียวกันก็คอยติดตามดู ช่องโหว่ที่มีการประกาศออกมา ที่อาจจะทำให้กรอบล้อมความมั่นคงของผู้ให้บริการอุปกรณ์ฮาร์ดแวร์อ่อนแอลงได้

การทดลองแบบสลับไปมา

ประเมิน

ในการทดลองปิดใด ๆ เทคนิคการทดสอบแบบ A/B เป็นข้อมูลประกอบการตัดสินใจที่ดีเยี่ยมเมื่อใช้กับการพัฒนาผลิตภัณฑ์ แต่ผลการทดสอบจะได้ไม่ตรงนัก หากไม่สามารถทำให้ A/B เป็นอิสระต่อกันได้ เช่น เมื่อเพิ่มคนในกลุ่มหนึ่ง จะส่งผลกระทบต่ออีกกลุ่มหนึ่งเสมอ ซึ่งปัญหานี้สามารถแก้ไขด้วยวิธี การทดลองแบบสลับไปมา (Switchback experimentation) โดยแก่นหลักของแนวคิดนี้บอกไว้ว่า ให้สลับการทดลองไปมาระหว่างทั้งสองโหมด ณ กลุ่มสถานที่เดียวกันแต่ต่างเวลาออกไป แทนที่จะทดลองไปทั้งสองโหมดในเวลาเดียวกัน จากนั้นถึงเปรียบเทียบดูว่า ลูกค้าชอบวิธีไหนมากกว่ากัน รวมถึงค่าชี้วัดอื่น ๆ ที่สนใจ เราได้ลองวิธีการนี้ในบางโปรเจคแล้ว ซึ่งก็ได้ผลดี และคิดว่ามันเป็นเทคนิคที่ดีที่เราจะเก็บมันไว้ใช้ในครั้งถัดไป

ข้อมูลรับรองตัวตนที่พิสูจน์ได้

ประเมิน

ข้อมูลรับรองตัวตน (credential) อยู่ไปทุกที่ในชีวิตเรา ทั้งหนังสือเดินทาง ใบขับขี่ หรือใบประกาศทางการศึกษาก็ดี อย่างไรก็ตาม ทุกวันนี้ ข้อมูลรับรองตัวตนดิจิทัลส่วนใหญ่เป็นเพียงแค่บันทึกข้อมูลง่าย ๆ ในระบบสารสนเทศ ซึ่งง่ายต่อการแก้ไข ปลอมแปลง และบ่อยครั้งก็เปิดเผยข้อมูลที่ไม่จำเป็นออกมาด้วย ช่วงไม่กี่ปีที่ผ่านมา เราเห็นถึงความพร้อมขึ้นเรื่อย ๆ ของวิธีที่จะมาแก้ปัญหานี้ ที่เรียกกันว่าข้อมูลรับรองตัวตนที่พิสูจน์ได้ (Verifiable Credentials) ซึ่ง มาตรฐาน W3C ได้ให้นิยามเทคนิคนี้ว่า เป็นการสร้างชุดข้อมูลที่รักษาความมั่นคงด้วยการเข้ารหัส ที่เคารพความเป็นส่วนตัวของผู้ถือ และสามารถใช้เครื่องพิสูจน์ความถูกต้องได้ โดยรูปแบบการใช้งานจะคำนึงถึงผู้ถือข้อมูลเป็นศูนย์กลางเสมอ เสมือนวิธีที่เราจัดการกับข้อมูลรับรองตัวตนในชีวิตจริง ที่ผู้ใช้สามารถบรรจุข้อมูลรับรองตัวตนที่พิสูจน์ได้ใส่ในกระเป๋าเงินดิจิทัล แล้วใช้แสดงต่อใครก็ตาม โดยไม่ต้องขออนุญาตกับผู้ออกใบรับรอง แนวทางแบบกระจายศูนย์เช่นนี้ ส่งผลให้ผู้ใช้งานสามารถจัดการข้อมูลของตนได้ดีกว่าเดิม และเลือกที่จะเปิดเผยเฉพาะข้อมูลที่ต้องการ ซึ่งเป็นการยกระดับการปกป้องความเป็นส่วนตัวของข้อมูลไปอีกขั้น ตัวอย่างเช่น เมื่อใช้เทคนิคการพิสูจน์ความจริงโดยไม่ต้องเปิดเผยข้อมูล (zero-knowledge proof) คุณสามารถสร้างข้อมูลรับรองตัวตนเพื่อที่จะยืนยันว่าคุณเป็นผู้ใหญ่ โดยไม่ต้องเปิดเผยข้อมูลวันเกิดของคุณเลย ปัจจุบันชุมชนนักพัฒนาได้จัดทำตัวอย่างการใช้งาน ข้อมูลรับรองตัวตนที่พิสูจน์ได้ในรูปแบบต่าง ๆ ไว้ให้ศึกษากัน พวกเราเองก็พัฒนาข้อมูลรับรองสุขภาพเกี่ยวกับ COVID ขึ้นมาเช่นกัน โดยอ้างอิงจาก COVID-19 Credentials Initiative (CCI) แม้ว่าเทคนิคนี้จะไม่ขึ้นกับเทคโนโลยีบล็อกเชน หรือ เอกลักษณ์แบบกระจายศูนย์ (decentralized identity) แต่ในทางปฏิบัติเรามักจะเห็นมันทำงานร่วมกับเอกลักษณ์แบบกระจายศูนย์อยู่เสมอ และใช้บล็อกเชนเพื่อลงทะเบียนข้อมูลที่ใช้พิสูจน์ นอกจากนี้ มักจะเห็นเฟรมเวิร์คด้านเอกลักษณ์แบบกระจายศูนย์ ถูกผนวกรวมกับข้อมูลรับรองตัวตนที่พิสูจน์ได้มาเลย

Apollo Federation

เฝ้าระวัง

เมื่อครั้งแรกที่เราพูดถึง GraphQL ในเรดาร์ เราได้เตือนว่าการใช้มันอย่างผิด ๆ จะเผลอไปทำตามรูปแบบที่ไม่ดีที่คนนิยมทำกัน ซึ่งหากใช้มันหนักขึ้นจะก่อให้เกิดโทษมากกว่าคุณ อย่างไรก็ดี เราก็เห็นหลายทีมให้ความสนใจใน GraphQL มากขึ้น ซึ่งในนั้นก็รวมทีมของพวกเราด้วย จากที่มันสามารถรวบรวมข้อมูลจากหลายแหล่งเข้าด้วยกันได้ ครั้งนี้เราอยากจะเตือนว่าให้ใช้ Apollo Federation อย่างระมัดระวัง จากที่มันชอบแนะนำให้สร้างข้อมูลกราฟขนาดใหญ่ (unified data graph) ที่ครอบคลุมทุกโดเมนขององค์กรขึ้นมา แนวคิดนี้ ถ้าฟังเผิน ๆ ก็คงฟังดูน่าสนใจที่เดียว แต่เราก็ควรระลึกถึงความพยายามในทำนองเดียวกัน ที่เคยเกิดขึ้นมาแล้วกับอุตสาหกรรมนี้เช่นกัน อย่างเช่น แนวคิด MDM หรือการบัญญัติโมเดลข้อมูล (canonical data model) และอื่น ๆ ที่แสดงให้เห็นถึงกับดักของความคิดนี้ เรื่องนี้จะทวีความร้ายแรงขึ้น หากว่าโดเมนที่เรากำลังยุ่งอยู่ด้วยนั้น มีความสลับซับซ้อนมากพอที่จะสร้างเป็นโมเดลเฉพาะของตัวเอง

ESB ในคราบ API Gateway

เฝ้าระวัง

นานมาแล้ว เราเคยเตือนให้ระวังถึง อีเอสบีแบบรวมศูนย์ หรือ centralized enterprise services buses แล้วบอกว่าแนวคิด “ความฉลาดไปอยู่ที่ปลายท่อ ส่วนความโง่ไปอยู่ในท่อ” ต่างหาก ที่เป็นหนึ่งในคุณสมบัติที่ดีของไมโครเซอร์วิส ที่น่าห่วงคือ เราสังเกตเห็นว่ากลุ่มผลิตภัณฑ์ ESB เก่ากำลังปรับภาพลักษณ์ผลิตภัณฑ์ตัวเองใหม่ ให้กลายเป็น ESB ในคราบ API Gateway ที่จะชอบเชื้อเชิญให้ผู้ใช้ออกแบบ API Gateway ให้เก่งเกินตัวกว่าที่ควร ได้โปรดอย่าให้คำโฆษณาพวกนี้หลอกคุณได้ เพราะไม่ว่าเครื่องมือนั้นจะชื่อว่าอะไรก็ตาม แต่การนำตรรกะทางธุรกรรมไปรวมศูนย์ไว้ในเครื่องมือหนึ่งเป็นความคิดที่ไม่ดีเลย มีแต่จะสร้างพันธะผูกพัน ขาดความโปร่งใส และทำให้หลุดไม่พ้นจากผู้ให้บริการรายนั้น โดยไม่มีข้อดีที่ชัดเจนอะไรเลย ทั้งนี้ เราเชื่อว่า API Gateway มีประโยชน์เสมอ หากใช้มันจัดการเรื่องความต้องการที่ตัดขวางทั้งระบบ (cross-cutting concern) แต่ความฉลาดต่าง ๆ ก็ควรไปอยู่ที่ API ข้างนอกไม่ใช่ที่ตัวมัน

การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ

เฝ้าระวัง

หลายปีก่อน เป็นช่วงที่แพลตฟอร์มด้านเก็บรวบรวมล็อกบันทึกเหตุการณ์ (log aggregation platform) ยุคใหม่ทยอยเกิดขึ้นมา แพลตฟอร์มพวกนี้สามารถจัดเก็บและค้นหาข้อมูลจากล็อกบันทึกเหตุการณ์จำนวนมาก คนจึงนิยมนำมาใช้วิเคราะห์ข้อมูลการทำงานของระบบ เพื่อสกัดหาแนวโน้มและความรู้ใหม่ที่ซ่อนอยู่ ซึ่ง Splunk เป็นหนึ่งในกลุ่มเครื่องมือประเภทนี้ที่โดดเด่นกว่าใคร การที่เครื่องมือเหล่านี้ช่วยให้คนสามารถเห็นภาพรวมด้านความมั่นคงและการทำงานของระบบ จึงกลายเป็นสิ่งที่นักพัฒนาและผู้ดูแลระบบขาดไม่ได้ กระแสการใช้งานนี้ได้แพร่หลายไปสู่ฝั่งธุรกิจด้วยเช่นกัน นำมาซึ่งความคิดที่ว่า เราน่าจะใช้ การเก็บรวบรวมล็อกบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ ได้ด้วยเหมือนกัน

แต่ในความเป็นจริงแล้ว ความต้องการทางธุรกิจมักจะล้ำหน้าและเปลี่ยนแปลงอย่างรวดเร็ว เกินกว่าที่เครื่องมือประเภทนี้จะตอบสนองได้ เพราะด้วยธรรมชาติการจดบันทึกเหตุการณ์นั้น มีไว้สำหรับการเฝ้าสังเกตการณ์ในเชิงเทคนิคเท่านั้น และมักให้ข้อมูลไม่เพียงพอหากจะใช้ทำความเข้าใจลูกค้าในเชิงลึก ฉะนั้นแล้ว หากคุณต้องการวิเคราะห์หาข้อมูลลูกค้าอย่างจริงจัง เราแนะนำให้ใช้เครื่องมือและตัวชี้วัดที่สร้างขึ้นสำหรับงานประเภทนี้โดยเฉพาะ หรือออกแบบระบบให้รองรับการสังเกตการณ์โดยธรรมชาติ โดยใช้วิธีการขับเคลื่อนพฤติกรรมจากเหตุการณ์ที่เกิดขึ้น (event-driven approach) ซึ่งวิธีนี้ ทำให้เหตุการณ์ทั้งในแง่ธุรกิจและการใช้งานจะถูกรวบรวมและจัดเก็บด้วยกลวิธีที่สามารถนำเหตุการณ์เหล่านี้กลับมาฉายซ้ำ หรือประมวลผลด้วยเครื่องมือที่สร้างขึ้นมาเพื่อวัตถุประสงค์นี้โดยเฉพาะ

ไมโครฟรอนท์เอนด์เพื่ออนาธิปไตย

เฝ้าระวัง

ตั้งแต่ที่เราได้นิยามคำนี้ไว้ในปี 2016 ไมโครฟรอนท์เอนด์ก็ได้รับความนิยมขึ้นและเป็นที่แพร่หลาย ซึ่งมันเองก็โชคร้ายไม่ต่างกับเทคนิคอื่น ๆ ที่ดันมีชื่อติดหู นั่นคือโดนใช้อย่างผิด ๆ หรือมากเกินควร ซึ่งประเด็นที่เรากังวลเป็นพิเศษ คือสถาปัตยกรรมนี้ถูกนำไปสร้างความชอบธรรมในการผสมใช้เทคโนโลยี เครื่องมือ หรือเฟรมเวิร์คต่าง ๆ ที่ทับซ้อนไว้ด้วยกัน อันนำไปสู่ภาวะที่เรียกว่า ไมโครฟรอนท์เอนด์เพื่ออนาธิปไตย หนึ่งในรูปแบบของอาการร้ายแรงที่มักพบเห็นกัน คือการผสมเอาหลายเฟรมเวิร์คทางฟรอนท์เอนด์ เช่น ใช้ React.js และ Angular ในหน้าแอปพลิเคชันเดียวกัน แม้จะทำได้ในทางเทคนิค แต่ก็ไม่แนะนำให้ทำเลย ถ้าไม่ได้เป็นส่วนหนึ่งของกลยุทธ์การเปลี่ยนถ่ายเทคโนโลยีที่ไตร่ตรองกันไว้อย่างดี แต่หากจะทำจริง ๆ ทีมควรจะสร้างความสอดคล้องกันในคุณสมบัติต่าง ๆ อย่างวิธีการจัดรูปแบบเว็บ เช่น ตกลงกันว่าจะใช้ CSS-in-JS หรือ CSS modules และกลไกที่จะใช้เชื่อมคอมโพเนนท์ต่าง ๆ เข้าด้วยกัน เช่น จะเลือกใช้ iFrames หรือ Web Components กันดี ยิ่งไปกว่านั้น องค์กรควรจะตัดสินใจว่าจะกำกับดูแลกันอย่างไร จะตั้งมาตรฐานขึ้นมา หรือจะปล่อยให้ทีมต่าง ๆ ตัดสินใจกันเอง ว่าจะใช้เครื่องมือบริหารจัดการสถานะข้อมูล วิธีการดึงข้อมูล เครื่องมือช่วยบิลด์ ไลบรารีเพื่อการวิเคราะห์ ฯลฯ ตัวใดในแอปพลิเคชันไมโครฟรอนท์เอนด์

สมุดโน๊ตเสมือนในงานโปรดักชัน

เฝ้าระวัง

ตลอดหลายสิบปีมานี้ แนวคิด computational notebooks หรือ สมุดโน๊ตเสมือนเพื่อการคำนวณ ที่ Wolfram Mathematica คิดค้น ได้เติบโตกลายเป็นเครื่องมือสำคัญที่ใช้กับงานวิจัยทางวิทยาศาสตร์ งานทดลอง และงานด้านวิชาการเสมอมา ก่อนจะถูกใช้กับงานวิทยาศาสตร์ข้อมูลในเวลาต่อมา

จากความนิยมของ Jupyter notebooks และ Databricks notebooks มันจึงกลายเป็นคู่หูคนโปรดของนักวิทยาศาสตร์ข้อมูลไปเลย แต่การใช้งานก็จะจำกัดอยู่กับการเป็นสื่อกลางการสื่อสารระหว่างกัน หรือใช้เป็นเครื่องมือคิดค้นทดลองนวัตกรรมสมัยใหม่เท่านั้น จนกระทั่งเมื่อเร็ว ๆ นี้ เราเริ่มเห็นแนวโน้มใหม่ ที่สนับสนุนให้ใช้สมุดโน๊ตเสมือนกับงานระดับโปรดักชันได้เลย ผ่านการโฆษณาชวนเชื่อของผู้ให้บริการบางราย กรณีนี้แม้เจตนาอาจจะดี — เพื่อให้เกิดความเท่าเทียมและความร่วมมือกันเขียนโปรแกรมระหว่างนักวิทยาศาสตร์ข้อมูล — แต่ผลลัพธ์ที่ได้กลับแย่ โดยวิธีการนี้จะสูญเสียคุณสมบัติดี ๆ ที่จำเป็นต้องมีในงานโปรดักชันไปเสียหมดทั้งเรื่อง ความสามารถในการขยายตัว การดูแลรักษา ความยืดหยุ่นต่อความล้มเหลว ฯลฯ เราจึงอยากแนะนำว่า อย่าใช้ สมุดโน๊ตเสมือนในงานโปรดักชัน แต่จงส่งเสริมให้นักวิทยาศาสตร์ข้อมูลสามารถสร้างโค้ดคุณภาพดีได้ด้วยตัวเอง ด้วยเครื่องมือและเฟรมเวิร์คที่เหมาะสม และให้ไปใช้แพลตฟอร์มสำหรับแมชชีนเลิร์นนิงที่ทำงานได้ครบวงจร ที่ช่วยให้การกระบวนการการส่งมอบอย่างต่อเนื่องทำได้ง่าย และช่วยซ่อนความซับซ้อนออกไปให้แล้ว

เทคนิค

ประเด็นที่เรากังวลเป็นพิเศษคือการนำสถาปัตยกรรมไมโครฟรอนท์เอนด์ไปสร้างความชอบธรรมในการผสมใช้เทคโนโลยี เครื่องมือ หรือเฟรมเวิร์คต่าง ๆ ที่ทับซ้อนไว้ด้วยกันในหน้าแอปพลิเคชันเดียวกัน แม้จะทำได้ในทางเทคนิค แต่เราก็ไม่แนะนำให้ทำ

(ไมโครฟรอนท์เอนด์เพื่ออนาธิปไตย)

TECHNOLOGY RADAR

แพลตฟอร์ม

แพลตฟอร์ม

Azure DevOps

ทดลอง

บริการของ Azure DevOps เป็นบริการชุดเครื่องมือสำหรับการพัฒนาซอฟต์แวร์ โดยมีบริการที่เก็บซอร์สโค้ดสำหรับ Git ระบบไปป์ไลน์ CI/CD เครื่องมือช่วยทดสอบอย่างอัตโนมัติ เครื่องมือบริหารงานคงค้าง และบริการคลังเก็บแพ็กเกจ

เราพบว่า ทีมของเราเลือกใช้บริการ Azure DevOps กันมากขึ้น ซึ่งพวกเขาก็พอใจกับผลลัพธ์ที่ได้ นี่เป็นสัญญาณที่ดีว่า Azure DevOps มีความพร้อมกว่าเดิม เรานั้นชอบใจในความยืดหยุ่นของมัน ที่ยอมให้ผสมบริการต่าง ๆ เข้าด้วยกันอย่างอิสระ แม้ว่าบริการเหล่านั้นจะเป็นของผู้ให้บริการภายนอกก็ตาม เช่น ในขณะที่ใช้บริการไปป์ไลน์ของ Azure DevOps อยู่ เราสามารถใช้ที่เก็บซอร์สโค้ดจากผู้ให้บริการอีกเจ้าได้ ทีมของเราฮือฮากับบริการ Azure DevOps Pipelines เป็นที่สุด อย่างไรก็ดีบริการอื่น ๆ ของ Azure DevOps ก็สร้างความประทับใจที่ดีกับนักพัฒนาที่ใช้งานเช่นกัน และช่วยส่งมอบคุณค่าให้เราอย่างต่อเนื่อง

Debezium

ทดลอง

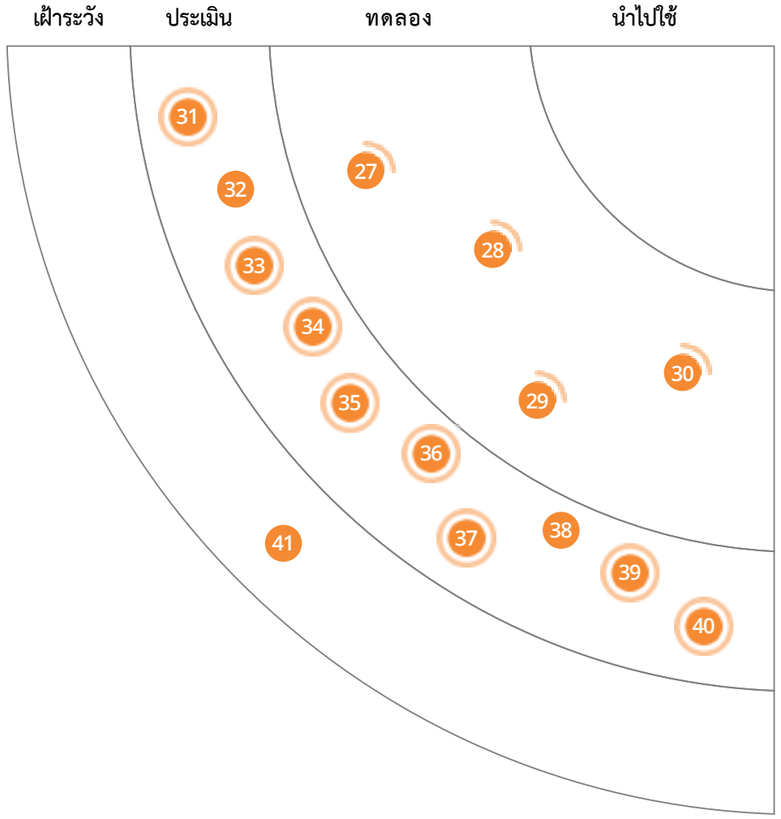
Debezium เป็นแพลตฟอร์มดักจับการเปลี่ยนแปลงของข้อมูล (change data capture: CDC) ซึ่งมันสามารถจับการเปลี่ยนแปลงที่เกิดขึ้นในฐานข้อมูลแล้วส่งต่อไปยังทอปปิกของ Kafka

เทคนิคการทำ CDC สามารถประยุกต์กับสถานการณ์หลายรูปแบบ เช่น ใช้ทำสำเนาข้อมูลระหว่างฐานข้อมูล ใช้ป้อนข้อมูลเข้าสู่ระบบวิเคราะห์ ใช้จัดการข้อมูลเมื่อต้องแตกสถาปัตยกรรมโมโนลิธออกเป็นไมโครเซอร์วิส หรือใช้หักล้างข้อมูลที่อยู่ในแคช Debezium ทำงานโดยการอ่านไฟล์ล็อกบันทึกเหตุการณ์ของฐานข้อมูลเพื่อดักจับการเปลี่ยนแปลงที่เกิดขึ้น มีหัวเชื่อมต่อที่รองรับฐานข้อมูลหลายเจ้า ทั้ง Postgres, MySQL, Oracle และ MongoDB เราใช้งาน Debezium กับหลายโครงการ และมันทำงานได้ดีสมใจเรา

Honeycomb

ทดลอง

Honeycomb เป็นบริการด้านการสังเกตการณ์ข้อมูล ที่รองรับการนำเข้าข้อมูลปริมาณมหาศาลจากระบบต่าง ๆ ในโปรดักชันแล้วทำให้มันจัดการได้ ผ่านเทคนิคการสุ่มกลุ่มตัวอย่างข้อมูลแบบไดนามิก นักพัฒนาจะสามารถเขียนล็อกบันทึกเหตุการณ์ที่สมบูรณ์ไว้ก่อน แล้วค่อยคิดว่าจะหั่นและเชื่อมข้อมูลพวกนี้กันอย่างไรทีหลัง ซึ่งเหมาะกับระบบกระจายตัวขนาดใหญ่ที่เป็นอยู่ในปัจจุบัน เพราะเราอยู่ในยุคที่ไม่สามารถจะคาดเดาได้ทั้งหมด ว่าในอนาคตจะมีข้อมูลใดที่เราอยากรู้เพิ่มเติมบ้าง ทีมงานของ Honeycomb กำลังพัฒนาบริการนี้ให้ใช้ร่วมกับภาษาและเฟรมเวิร์คต่าง ๆ ซึ่งขณะนี้รองรับทั้ง Go, Node, Java และ Rails ฯลฯ มีการเพิ่มฟีเจอร์ใหม่เข้ามาอย่างรวดเร็ว อีกทั้งอัตราค่าบริการก็ถูกปรับให้เข้าถึงได้ง่ายขึ้นเพื่อเพิ่มความน่าสนใจ และทีมของเราก็ชอบมันมากด้วย



นำไปใช้

ทดลอง

27. Azure DevOps
28. Debezium
29. Honeycomb
30. JupyterLab

ประเมิน

31. Amundsen
32. AWS Cloud Development Kit
33. Backstage
34. Dremio
35. DuckDB
36. K3s
37. Materialize
38. Pulumi
39. Tekton
40. Trust over IP stack

เฝ้าระวัง

41. Node overload

แพลตฟอร์ม

สภาพแวดล้อมที่โต้ตอบกับผู้ใช้ของ JupyterLab เป็นวิวัฒนาการมาจาก Jupyter Notebook โดยมันเอาความสามารถเดิมที่มี มาเพิ่มเติมความสามารถใหม่เข้าไป ให้สามารถลากวางเซลล์ต่าง ๆ ได้ หรือกดแท็บเพื่อช่วยเติมคำที่เหลือให้เต็ม นอกจากนั้นยังมีอีกหลายฟีเจอร์ที่น่าสนใจที่ยังไม่ได้กล่าวถึง

(JupyterLab)

Backstage เป็นแพลตฟอร์มโอเพนซอร์สสำหรับสร้างศูนย์ส่งเสริมนักพัฒนา (developer portal) ที่สร้างโดย Spotify มันช่วยจัดตั้งมาตรฐานของเทคโนโลยีและเครื่องมือต่าง ๆ ที่ทีมกำลังใช้งานอยู่ และป้องกันไม่ให้ระบบนิเวศของซอฟต์แวร์นั้นกระจัดกระจายและมีความความซับซ้อน

(Backstage)

JupyterLab
ทดลอง

ตั้งแต่ฉบับที่แล้ว ที่เราแนะนำให้ประเมิน JupyterLab ไป ในตอนนี้ มันได้กลายเป็นเว็บ UI หลักสำหรับใช้งาน Project Jupyter ของหลาย ๆ คนไปแล้ว ซึ่งความนิยมของ JupyterLab กำลังแซง Jupyter Notebook ไปอย่างรวดเร็วมาก และไม่ช้าก็เร็วน่าจะมาแทนที่ในที่สุด หากคุณยังใช้ Jupyter Notebook อยู่ ก็น่าจะลอง JupyterLab ดูได้แล้ว สภาพแวดล้อมที่โต้ตอบกับผู้ใช้ของมันก็เป็นวิวัฒนาการมาจาก Jupyter Notebook โดยมันเอาความสามารถเดิมที่มี มาเพิ่มเติมความสามารถใหม่เข้าไป ให้สามารถลากวางเซลล์ต่าง ๆ ได้ หรือกดแท็บเพื่อช่วยเติมคำที่เหลือให้เต็ม นอกจากนั้นยังมีอีกหลายฟีเจอร์ที่น่าสนใจที่ยังไม่ได้กล่าวถึง

Amundsen
ประเมิน

นักวิทยาศาสตร์ข้อมูลใช้เวลาส่วนใหญ่ไปกับการสำรวจข้อมูล (data discovery) หากมีเครื่องมือใหม่ ๆ มาช่วยได้ก็ย่อมจะทำให้พวกเขาตื่นเต้นขึ้นมาได้บ้าง ทุกวันนี้ โครงการ Apache Atlas ได้ถูกใช้เป็นเครื่องมือหลักเพื่อใช้จัดการข้อมูลอภิพันธุ์ (metadata management) ไปแล้ว แต่การจะสำรวจข้อมูลก็ยังทำได้ไม่ง่ายนักอยู่ดี ขอเสนอ Amundsen เครื่องมือที่สามารถติดตั้งและทำงานควบคู่ไปกับ Apache Atlas ที่จะทำให้หน้าจอการค้นหาเพื่อการสำรวจข้อมูลนั้นน่าใช้ยิ่งขึ้น

ชุดเครื่องมือสำหรับพัฒนาบนคลาวด์ของ AWS
ประเมิน

Terraform กลายเป็นตัวเลือกหลักของพวกเราหลายคนเพื่อใช้กำหนดโครงสร้างพื้นฐานบนคลาวด์ แต่ก็มีบางคนได้ไปทดลองใช้ AWS Cloud Development Kit (AWS CDK) แล้วชอบใจไม่เบาเหมือนกัน โดยเฉพาะการที่พวกเขาสามารถใช้ภาษาโปรแกรมมิ่งยอดนิยมกำหนดค่า แทนการใช้ไฟล์คอนฟิคแบบเดิม จึงสามารถประยุกต์ใช้ทักษะ เครื่องมือ และวิธีการทดสอบ ที่คุ้นเคยได้ทันทีกับการกำหนดโครงสร้างพื้นฐาน แต่ทั้งนี้การใช้ AWS CDK ก็ไม่ต่างจากเครื่องมือตัวอื่น ๆ ที่ต้องให้ความใส่ใจ เพื่อให้การติดตั้งจัดการเป็นสิ่งที่เข้าใจและดูแลรักษาได้ง่าย โดยขณะนี้ มันรองรับการเขียนด้วยภาษา TypeScript, JavaScript, Python, Java, C# และ .NET เราจะยังคงจับตาดูมันต่อไป โดยเฉพาะ หลังจากที่ทีมของ AWS ร่วมกับ HashiCorp จับมือกันเปิดตัว ชุดเครื่องมือ Cloud Development Kit สำหรับ Terraform ฉบับทดลอง ซึ่งสามารถผลิตค่ากำหนดของ Terraform และควบคุมการสร้างโครงสร้างพื้นฐานจากแพลตฟอร์มของ Terraform ผ่าน CDK ได้

Backstage
ประเมิน

หลายองค์กรสรรหาวิธีสนับสนุนและปรับปรุงสภาพแวดล้อมการพัฒนาของตนให้มีความสะดวกยิ่งขึ้น ด้วยการตั้งศูนย์หรือแพลตฟอร์มสำหรับนักพัฒนาขึ้นมา ยิ่งองค์กรมีเทคโนโลยีและเครื่องมือมากเท่าไหร่ ก็ควรให้ความสำคัญกับการจัดตั้งมาตรฐานระหว่างกันสักรูปแบบหนึ่งขึ้นมา เพื่อให้องค์กรเดินหน้าไปในทิศทางเดียวกัน ทั้งนี้ก็เพื่อให้นักพัฒนาสนใจไปที่การสร้างนวัตกรรมและผลิตภัณฑ์เป็นหลัก ได้ใช้ประโยชน์จากงานที่คนอื่นทำกันไว้ ไม่เสียเวลาเริ่มสร้างอะไรใหม่จากศูนย์ ฉะนั้นการมีศูนย์ส่งเสริมนักพัฒนา (developer portal) จะช่วยให้นักพัฒนารู้ว่ามีบริการและแนวทางปฏิบัติอะไรให้ใช้บ้างภายในหน่วยงาน Backstage เป็นแพลตฟอร์มโอเพนซอร์สสำหรับสร้างศูนย์ส่งเสริมนักพัฒนาที่สร้างโดย Spotify ที่รองรับการสร้างแม่แบบซอฟต์แวร์เพื่อให้การขึ้นโครงการใหม่ทำได้ง่าย ช่วยรวบรวมเครื่องมือด้านโครงสร้างพื้นฐานไว้ด้วยกัน และเป็นคลังเก็บเอกสารทางเทคนิคต่าง ๆ Backstage มีสถาปัตยกรรมการออกแบบที่รองรับการต่อเติมขยาย จึงทำให้การเพิ่มความสามารถใหม่ หรือการปรับแต่งระบบให้เข้ากับระบบนิเวศขององค์กรนั้น เป็นเรื่องที่ทำได้

Dremio
ประเมิน

Dremio เป็นคลาวด์ดาต้าเลคเอนจิ้น ที่สามารถประมวลผลงานคิวรีข้อมูลเชิงตอบโต้กับดาต้าเลคบนคลาวด์ได้โดยตรง เมื่อคุณใช้งาน Dremio เพื่อพยากรณ์แนวโน้ม คุณไม่ต้องเสียแรงบริหารไปป์ไลน์เพื่อสกัดและแปลงข้อมูลไปเก็บที่ดาต้าแวร์เฮาส์แยกต่างหาก มันช่วยแบ่งข้อมูลออกเป็นชุดเสมือนรูปแบบการนำเข้าข้อมูลสู่ดาต้าเลค และช่วยปรับมุมมองผู้ใช้ข้อมูลให้มีรูปแบบเดียวกัน แม้ว่าเอนจิ้นอย่าง Presto จะเป็นผู้เผยแพร่เทคนิคการแยกแหล่งเก็บข้อมูลออกจากหน่วยประมวลผลข้อมูล ให้เป็นที่รู้จัก แต่ Dremio นำเทคนิคดังกล่าวไปต่อยอดแล้วทำได้ดีกว่า ทั้งในด้านประสิทธิภาพ และการลดค่าใช้จ่ายด้านบริหารงานให้ถูกลงไปอีก

DuckDB
ประเมิน

เป็นระบบฐานข้อมูลแบบคอลัมน์ (columnar database) สามารถฝังตัวเข้ากับโปรแกรมอื่นได้ โดยออกแบบมาเพื่องานด้านวิทยาศาสตร์ข้อมูลและงานวิเคราะห์ข้อมูลโดยเฉพาะ เนื่องจากว่า นักวิเคราะห์ข้อมูลมักใช้เวลาส่วนสำคัญกับการเตรียมข้อมูลให้สะอาดและการจำลองข้อมูลให้ดีบนเครื่องตัวเอง ก่อนจะขยับขยายนำงานไปประมวลต่อบนเครื่องเซิร์ฟเวอร์ทีหลัง

แม้ในอดีต เทคโนโลยีระบบฐานข้อมูลถูกพัฒนามานานหลายทศวรรษแล้วก็ตาม แต่ส่วนใหญ่ทำมาเพื่อใช้กับงานระหว่างไคลแอนต์-เซิร์ฟเวอร์เป็นหลัก ไม่ได้ถูกออกแบบให้

เหมาะสำหรับเป็นฐานข้อมูลบนเครื่องนักพัฒนา เพื่อรองรับงานประเภทประมวลคิวรีเชิงโต้ตอบ นักวิเคราะห์จึงหาทางหลีกเลี่ยงข้อจำกัดดังกล่าว ด้วยการใช้เครื่องมือที่สามารถประมวลผลข้อมูลจากหน่วยความจำแทน ผ่านเครื่องมือ เช่น Pandas หรือ data.table แม้เครื่องมือเหล่านี้จะให้ผลลัพธ์เป็นที่น่าพอใจก็จริง แต่ก็มีข้อจำกัดสำคัญคือข้อมูลที่นำมาวิเคราะห์จะต้องมีขนาดใหญ่ไม่เกินหน่วยความจำที่มีในเครื่อง ซึ่งเราเห็นว่า DuckDB แก้ข้อจำกัดนี้อย่างลงตัว จากการเป็นระบบฐานข้อมูลแบบเชิงคอลัมน์แบบฝังตัว ที่ปรับแต่งประสิทธิภาพให้เหมาะสำหรับงานวิเคราะห์ข้อมูลบนเครื่องนักพัฒนาโดยเฉพาะ และยังรองรับข้อมูลที่มีขนาดใหญ่กว่าหน่วยความจำบนเครื่องได้ด้วย

K3s

ประเมิน

K3s เป็น Kubernetes ขนาดเล็กที่สร้างมาเพื่อฝังในอุปกรณ์ IoT หรือใช้กับระบบประมวลผลไกลศูนย์ (edge computing)

จากที่ K3s อยู่ในรูปแบบไบนารีเดี่ยวที่สามารถทำงานด้วยตัวเอง และแทบไม่พึ่งพิงไลบรารีใดจากระบบปฏิบัติการ เลยทำให้ง่ายต่อการใช้งานและดูแล มันสลับเอา sqlite3 มาใช้เป็นตัวจัดเก็บข้อมูลตั้งต้น แทนการใช้ etcd แบบปกติ มันตัดสินใจลดขนาดการใช้หน่วยความจำลง ด้วยการนำเอาทุกส่วนงานสำคัญมาประมวลไว้ในโปรเซสเดียวเท่านั้น และเจตนาลดขนาดไบนารีลง ด้วยการตัดเอาไดรเวอร์ตัวจัดเก็บข้อมูลของผู้ให้บริการภายนอก และไลบรารีที่เกี่ยวข้องกับผู้ให้บริการคลาวด์ ที่ไม่จำเป็นกับการใช้งาน K3s ออกไปทั้งหมด ทำให้ K3s เป็นตัวเลือกที่ดีและน่าพิจารณามากสำหรับใช้ในสภาพแวดล้อมที่ทรัพยากรมีจำกัด

Materialize

ประเมิน

Materialize เป็นระบบฐานข้อมูลแบบสตรีมมิ่ง (streaming database) ที่ช่วยให้ผู้ใช้ประมวลผลข้อมูลเฉพาะส่วนที่เพิ่มเข้ามา (incremental computation) โดยไม่ใช้ไปป์ไลน์ประมวลข้อมูลซับซ้อน ผู้ใช้เพียงเขียนคำสั่งสร้างวิวในภาษา SQL มาตรฐาน และเชื่อม Materialize เข้ากับฐานข้อมูลที่ต้องการจะสตรีมข้อมูลออกมาเท่านั้น ภายใน Materialize ใช้เอนจิ้นแบบ differential data flow ประมวลผลข้อมูลเฉพาะส่วนที่เพิ่มเข้ามา เพื่อให้ได้ผลลัพธ์ที่ถูกต้อง สอดคล้องกัน และใช้เวลาประมวลน้อยที่สุด มันยังต่างจากระบบฐานข้อมูลทั่ว ๆ ไป ตรงที่มันไม่มีข้อจำกัดทางการสร้างวิว และการประมวลผลนั้นเกิดขึ้นแบบเรียลไทม์

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 Pulumi ยังคงเป็นเครื่องมือเดียวที่ไม่ขึ้นกับผู้ให้บริการรายใด เราคาดว่า Pulumi จะเป็นที่นิยมกว่านี้ และหวังที่จะได้เห็นเครื่องมือและความรู้ใหม่เกิดขึ้นรอบ ๆ เป็นระบบนิเวศสนับสนุนกันและกัน

Tekton

ประเมิน

Tekton เป็นแพลตฟอร์มน้องใหม่สำหรับจัดการไปป์ไลน์การส่งมอบซอฟต์แวร์ (CD pipeline) บน Kubernetes แบบเนทีฟ ไม่เพียงแค่การติดตั้งหรือการใช้งานที่ทำมาเพื่อ Kubernetes โดยเฉพาะเท่านั้น ส่วนการเขียนไปป์ไลน์ยังเป็น (รีซอร์สเสริม) (custom resource) ชนิดหนึ่งของ Kubernetes เลย นั่นหมายความว่าคุณสามารถควบคุมไปป์ไลน์ผ่านเครื่องมือไคลแอนต์ของ Kubernetes โดยตรง (ทั้ง CLI หรือ API) หรือใช้ประโยชน์จากความสามารถด้านการจัดการทรัพยากรของ Kubernetes อย่างการย้อนกลับเวอร์ชันเก่าได้ เป็นต้น Tekton มีฟอร์แมตที่มีความยืดหยุ่นสามารถกำหนดขั้นตอนงานต่าง ๆ เป็นเงื่อนไข กำหนดงานคู่ขนานกัน หรือกำหนดงานสุดท้ายเพื่อทำหน้าที่เก็บกวาดได้ ฯลฯ ส่งผลให้เราสามารถสร้างไปป์ไลน์เพื่อการดีพลอยที่ซับซ้อนหรือผสมผสานกันได้อย่างหลากหลาย รองรับการย้อนกลับเวอร์ชันเก่า หรือการทยอยปล่อยเวอร์ชั่นใหม่ และอีกมากมาย Tekton เปิดโอเพนซอร์ส และมี บริการบน GCP ให้เลือกใช้ แม้คู่มือใช้งานจะยังไม่ดีที่สุด และชุมชนผู้ใช้ยังไม่ใหญ่มาก แต่การใช้ Tekton ของเรากับงานระดับโปรดักชันบน AWS ก็สำเร็จไปด้วยดี

แพลตฟอร์ม

K3s เป็น Kubernetes ขนาดเล็กที่สร้างมาเพื่อฝังในอุปกรณ์ IoT หรือใช้กับระบบประมวลผลไกลศูนย์ (edge computing) โดยอาศัยการตัดเอาไดรเวอร์ตัวจัดเก็บข้อมูลของผู้ให้บริการภายนอกและไลบรารีที่เกี่ยวข้องกับผู้ให้บริการคลาวด์ ที่ไม่จำเป็นกับการใช้งานออกไปทั้งหมด

(K3s)

Materialize เป็นระบบฐานข้อมูลแบบสตรีมมิ่ง (streaming database) ที่ช่วยให้ผู้ใช้ประมวลผลข้อมูลเฉพาะส่วนที่เพิ่มเข้ามา (incremental computation) โดยไม่ใช้ไปป์ไลน์ประมวลข้อมูลซับซ้อน

(Materialize)

แพลตฟอร์ม

มาตรฐานนี้ถูกแบ่งออกเป็น 4 ลำดับชั้นทางด้านเทคนิคและทางด้านการกำกับดูแล โดยมุ่งเป้าไปที่การสร้างบรรทัดฐานของการจัดการเอกลักษณ์แบบกระจายศูนย์ ให้อยู่ในรูปแบบที่สามารถแลกเปลี่ยนข้อมูลระหว่างกันได้

(ระดับขั้นของ Trust over IP)

Node.js เป็นเทคโนโลยีที่ได้รับความนิยมสูง แต่นั่นก็ไม่ทำให้มันเหมาะสมที่จะใช้สำหรับทุกอย่าง ตัวอย่างเช่น งานที่ต้องใช้การประมวลผลทางซีพียูสูง ๆ ซึ่ง Node.js ก็เป็นตัวเลือกที่ไม่ดีเอาเสียเลย เราจึงอยากเตือนเกี่ยวกับพฤติกรรมการใช้ Node.js โดยไม่ไตร่ตรองให้ถี่ถ้วน หรือใช้มันด้วยเหตุผลผิด ๆ

(การใช้งาน Node อย่างพร่ำเพรื่อ)

Trust over IP stack

ประเมิน

ปัจจุบัน มีโจทย์ใหม่ ๆ ที่เข้ามาท้าทายวิธีการที่องค์กรและบุคคลต่าง ๆ สร้างความเชื่อใจระหว่างกันในโลกดิจิทัล ผลักดันทำให้เกิดเป็นแนวคิดใหม่เพื่อปรับปรุงวิธีการพิสูจน์ตัวตน วิธีแบ่งปันและพิสูจน์คุณลักษณะที่จำเป็นต่อการสร้างความไว้วางใจ และวิธีการทำธุรกรรมที่มั่นคง ที่ดีขึ้นกว่าเดิม เรดาห์ของเรามีการกล่าวถึงเทคโนโลยีพื้นฐานที่เกี่ยวข้องกับเรื่องนี้ เช่น เอกลักษณ์แบบกระจายศูนย์ และข้อมูลรับรองตัวตนที่พิสูจน์ได้ ซึ่งเป็นกุญแจสำคัญเพื่อเปิดทางสู่รูปแบบความเชื่อใจทางดิจิทัลยุคใหม่ที่กำลังจะมาถึง

อย่างไรก็ดี ปฏิเสธไม่ได้ว่าถ้าจะให้คนทั้งโลกยอมรับวิธีการใหม่เช่นนี้ จำเป็นต้องมีมาตรฐานกลางเกิดขึ้นเพื่อกำกับดูแลทางเทคนิคให้เกิดการแลกเปลี่ยนกับมาตรฐานรอบข้างได้ มูลนิธิ Trust over IP Foundation อันเป็นส่วนหนึ่งของ Linux Foundation จึงเกิดมาเพื่อผลักดันให้สิ่งนี้เกิดขึ้น พวกเขาได้แรงบันดาลใจมาจากมาตรฐานโปรโตคอล TCP/IP ที่คอยเป็นสื่อกลางให้อินเตอร์เน็ตสามารถแลกเปลี่ยนข้อมูลกันได้ข้ามอุปกรณ์นับล้าน ๆ เครื่อง ทางกลุ่มจึงพยายามร่างมาตรฐานโดยแบ่งออกเป็น 4 ลำดับชั้นทางด้านเทคนิคและทางด้านการกำกับดูแล ซึ่งสามารถดูเพิ่มเติมได้ที่ ระดับขั้นของ Trust over IP หากองค์กรใดกำลังทบทวนเรื่องระบบเอกลักษณ์ และวิธีการสร้างความเชื่อใจทางดิจิทัลกับระบบรอบข้างอยู่ละก็ เราแนะนำให้พิจารณา ToIP และเครื่องมือสนับสนุนของมัน อย่าง Hyperledger Aries

การใช้งาน Node อย่างพร่ำเพรื่อ

เฝ้าระวัง

เทคโนโลยีที่ได้รับความนิยมสูง มักมีแนวโน้มที่จะถูกนำไปใช้อย่างพร่ำเพรื่อจนเกินไป ดังเช่นกระแส การใช้งาน Node อย่างพร่ำเพรื่อ ที่กำลังเกิดขึ้นอยู่ตอนนี้ อันเป็นพฤติกรรมการใช้ Node โดยไม่ไตร่ตรองให้ถี่ถ้วน หรือใช้มันด้วยเหตุผลผิด ๆ ซึ่งมีสองคำอ้างที่มักถูกยกขึ้นมาใช้อยู่ประจำ

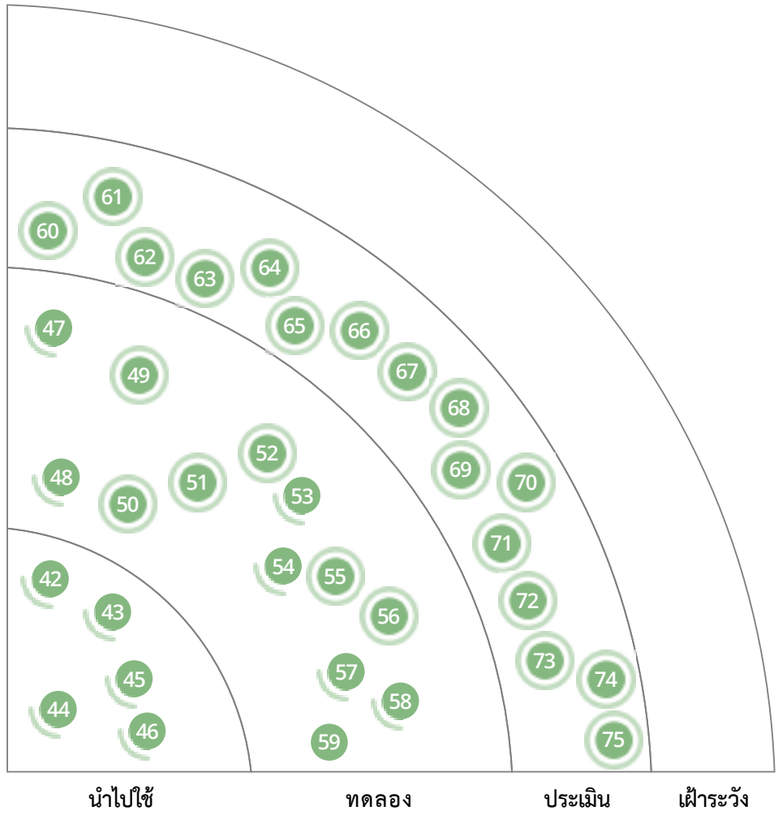
คำอ้างแรก คือการใช้ Node ทั้งระบบเป็นเรื่องดีเพราะจะได้ดูแลรักษาง่าย เป็นการใช้ภาษาโปรแกรมมิ่งเดียวกันทั้งระบบ ซึ่งเราเห็นต่างว่า วิธีผสมใช้หลายภาษาโปรแกรมมิ่งเข้าด้วยกัน (Polyglot programming) ตามความเหมาะสมของงานเป็นวิธีที่ดีกว่า ซึ่ง Node เองเป็นเทคโนโลยีที่ดีเช่นกัน เพียงแต่ต้องใช้อย่างเหมาะสม

เหตุผลที่สองที่มักจะถูกยกมาสนับสนุน คือเหตุผลด้าน “ประสิทธิภาพ” แม้ในปัจจุบันจะมีผลการทดสอบทางประสิทธิภาพที่น่าเชื่อถือออกมามากมายให้เปรียบเทียบ แต่ความเชื่อในสมัยเริ่มแรกที่ Node ได้รับความนิยมก็ยังฝังใจผู้คนอยู่ ในขณะนั้น มันเป็นเฟรมเวิร์คแรกที่นำเทคนิคการเขียนโปรแกรมแบบ ไม่บล็อคหน่วยประมวลผลย่อย (Nonblocking Programming Model) มาใช้อย่างจริงจัง เทคนิคนี้ให้ประสิทธิภาพที่ดีมาก เมื่อใช้กับงานที่มีการติดต่อกับ I/O มาก ๆ (ซึ่งเราเองได้กล่าวถึงข้อดีนี้ไปแล้วในงานเขียนเรื่อง Node.js ในปี 2012) แต่กับงานที่ใช้การประมวลผลทางซีพียูสูง ๆ มันกลับไม่ใช่ทางเลือกที่ดีเอาเสียเลย เพราะธรรมชาติของรันไทม์ในภาษา JavaScript นั้นจะมีหน่วยประมวลผลย่อยเพียงหน่วยเดียว (Single-threaded) เท่านั้น ซึ่งหากมองกลับมาที่ปัจจุบัน แพลตฟอร์มหรือเฟรมเวิร์กอื่น ๆ ต่างก็มีความสามารถการเขียนโปรแกรมแบบไม่มีการบล็อคออกมาแล้วเช่นกัน ซึ่งบางตัวยังมี API ที่เรียบง่ายและทันสมัยกว่าอีกด้วย การยกประเด็นด้าน “ประสิทธิภาพ” จึงไม่ใช่เหตุผลที่ฟังขึ้นอีกต่อไป

TECHNOLOGY RADAR

เครื่องมือ

เครื่องมือ



Airflow
นำไปใช้

Airflow ยังคงเป็นเครื่องมือติดตามขั้นตอนงาน (workflow management tool) แบบโอเพนซอร์ส สำหรับสร้างเป็นไปป์ไลน์ประมวลข้อมูลในรูปแบบกราฟแบบไม่วนทิศ (Directed Acyclic Graph - DAG) ที่เราโปรดและใช้กันแพร่หลายที่สุด โดยเครื่องมือประเภทนี้แข่งขันกันหนักขึ้นทั้งจากกลุ่มที่เป็นโอเพนซอร์สด้วยกัน เช่น Luigi และ Argo หรือจากกลุ่มธุรกิจผู้ให้บริการ เช่น Azure Data Factory หรือ AWS Data Pipeline อย่างไรก็ดี Airflow วางตำแหน่งแตกต่างจากตัวอื่น ตรงที่สามารถกำหนดขั้นตอนงานด้วยการเขียนโปรแกรม แทนการกำหนดค่าในไฟล์แบบไม่เขียนโค้ด อีกทั้งยังรองรับการทดสอบแบบอัตโนมัติ เปิดโอเพนซอร์ส ใช้งานได้หลายแพลตฟอร์ม มีช่องทางเชื่อมต่อมากมายกับระบบนิเวศข้อมูลต่าง ๆ และมีกลุ่มผู้ใช้ขนาดใหญ่คอยช่วยเหลือ แต่ทั้งนี้ Airflow ยังไม่รองรับสถาปัตยกรรมข้อมูลแบบกระจายศูนย์ เช่น ดาต้าเมช เนื่องจากมันควบคุมขั้นตอนงานแบบรวมศูนย์นั่นเอง

Bitrise
นำไปใช้

Bitrise เป็นเครื่องมือด้าน CD ที่ทำขึ้นเฉพาะแอปพลิเคชันมือถือ และยังมีประโยชน์อย่างมากในกระบวนการพัฒนาแอปพลิเคชันมือถือของเรา ที่เราอยากจะแนะนำให้ใช้กันมากกว่านี้ Bitrise ช่วยให้การพัฒนาแอปพลิเคชันมือถือทำได้สะดวก ตั้งแต่การบิลด์ การทดสอบ และการดีพลอย โดยเริ่มช่วยเหลือตั้งแต่กระบวนการแรก ที่โค้ดยังอยู่ในเครื่องนักพัฒนา ไปจนถึงกระบวนการสุดท้าย ที่จะเอาแอปพลิเคชันขึ้นแอปสโตร์ มันติดตั้งไม่ยากเลย และยังเตรียมขั้นตอนต่าง ๆ ที่จำเป็นสำหรับการพัฒนาแอปพลิเคชันบนมือถือ ไว้ให้พร้อมแล้ว

Dependabot
นำไปใช้

ในบรรดาเครื่องมือที่ช่วยอัพเดทไลบรารีที่เราพึ่งพา (dependencies) ให้ทันสมัยทั้งหมด Dependabot เป็นตัวเลือกแรกที่เราจะนึกถึงก่อนเสมอ มันเชื่อมต่อและทำงานร่วมกับ GitHub ได้อย่างแนบเนียน โดยมันจะคอยตรวจสอบว่าไลบรารีใดควรได้รับการอัพเดทบ้าง แล้วจะส่งคำขอแก้ไข (pull request) มาให้อย่างอัตโนมัติ สามารถเปิดใช้กับทุกโครงการภายในองค์กร ทำให้ทีมต่าง ๆ สามารถรับการอัพเดทได้อย่างง่ายดาย หากไม่ได้ใช้ Github ก็ไม่เป็นไร คุณยังใช้ไลบรารีของมันในบิลด์ไปป์ไลน์ได้อยู่ แต่หากคุณสนใจตัวเลือกอื่น อาจลองพิจารณา Renovate ที่ครอบคลุมการเชื่อมต่อกับผู้ให้บริการต่าง ๆ ที่มากกว่า ซึ่งรวมทั้ง GitLab, Bitbucket และ Azure DevOps

Helm
นำไปใช้

Helm เป็นตัวจัดการแพ็กเกจของ Kubernetes ที่มาพร้อมกับ Chart Repository อันเป็นคลังเก็บแอปพลิเคชัน Kubernetes อย่างเป็นทางการที่ได้รับการดูแลอย่างดี จากครั้งล่าสุดที่ได้กล่าวถึงไป ระหว่างนั้น Helm 3 ก็ได้ออกมา ซึ่งมีการเปลี่ยนแปลงครั้งสำคัญคือการเอา Tiller ออก ซึ่งเป็นคอมโพเนนท์ไว้ควบคุมฝั่งเซิร์ฟเวอร์ที่เคยอยู่ใน Helm 2 ซึ่งการตัดออกเช่นนี้ เป็นการบังคับให้การเปลี่ยนแปลงคลัสเตอร์ทำได้จากฝั่งไคลแอนต์เท่านั้น อันมีข้อดีคือการแก้ไขจะทำได้ตามสิทธิ์ของผู้ใช้ที่เรียกคำสั่ง Helm เข้ามา เราได้ใช้ Helm กับงานของลูกค้าหลายโครงการด้วยกัน ซึ่งความสามารถด้านการจัดการสิ่งพึ่งพา (dependency management) การทำเทมเพลต และกลไกฮุกที่มันมีให้ ได้

เครื่องมือ

Trivy เป็นเครื่องมือตรวจจับช่องโหว่ของคอนเทนเนอร์ ซึ่งเป็นไบนารีเดี่ยวที่ทำงานได้เลย ทำให้ติดตั้งได้ง่ายกว่าเครื่องมือตัวอื่น

(Trivy)

jscodeshift ช่วยลดความน่าปวดหัวของการดูแลโค้ดเบส JavaScript ขนาดใหญ่ ซึ่งเราเคยใช้มันตอนที่เราพยายามจะดูแลระบบ design systems แล้วพบว่ามันมีประโยชน์ดี

(jscodeshift)

ช่วยให้การจัดการแอปพลิเคชันตลอดทั้งช่วงชีวิตของมันใน Kubernetes เป็นเรื่องง่ายขึ้นมาก

Trivy
นำไปใช้

ไปป์ไลน์สำหรับการสร้างและการดีพลอยคอนเทนเนอร์ใด ๆ ควรรวมขั้นตอนการสแกนความมั่นคงของคอนเทนเนอร์เข้าไปด้วยเสมอ ในบรรดาเครื่องมือที่ดีหลายตัวในกลุ่มนี้ Trivy เป็นเครื่องมือตรวจจับช่องโหว่ของคอนเทนเนอร์ที่ทีมของเราชื่นชอบเป็นพิเศษ เราได้ลองใช้ทั้ง Clair และ Anchore Engine ไปแล้ว สิ่งที่ทำให้ Trivy แตกต่างจาก Clair คือมันไม่เพียงแต่ตรวจสอบคอนเทนเนอร์เท่านั้น แต่ยังตรวจสอบไปถึงไลบรารีที่พึ่งพาในระดับโค้ดอีกด้วย การที่มันเป็นไบนารีเดี่ยวที่ทำงานได้เลย ทำให้ติดตั้งได้ง่ายกว่าเครื่องมือตัวอื่น นอกจากนี้ ข้อดีของ Trivy คือมันเปิดโอเพนซอร์ส และรองรับคอนเทนเนอร์ประเภทดิสโตรเลส (distroless) อีกด้วย

Bokeh
ทดลอง

Bokeh เป็นหนึ่งในไลบรารีสำคัญในภาษา Python สำหรับวาดกราฟ แสดงแผนภาพข้อมูล และสามารถแสดงผลบนเบราว์เซอร์ได้ ผ่าน JavaScript มันทำให้การใช้โค้ดซ้ำเป็นเรื่องง่ายหากเทียบกับเครื่องประเภทเดียวกันบนเดสก์ทอป เหมาะกับงานที่ต้องลองผิดลองถูกบนเว็บแอปพลิเคชัน อีกทั้งมันยังเป็นไลบรารีที่มีความสมบูรณ์ และมีฟังก์ชันครบครัน สิ่งที่เราชอบเกี่ยวกับ Bokeh ได้แก่ การคงความสนใจอยู่กับการแสดงผลให้ดี และไม่พยายามก้าวก่ายเรื่องอื่นที่ไม่ใช่หน้าที่ อย่างเรื่อง การรวบรวมข้อมูล (ดู ggplot) หรือ การสร้างเว็บแอปพลิเคชัน (เช่น Shiny หรือ Dash การใช้ Bokeh จึงเป็นเรื่องที่สะดวกมาก หากการแยกการจัดการออกจากกันเช่นนี้ เป็นสิ่งสำคัญสำหรับคุณ มันยังมีวิดเจ็ตสำหรับแสดงผลบนเว็บมาให้ และยังสามารถใช้งานในโหมดเซิร์ฟเวอร์ได้ด้วย แต่คุณจะเลือกใช้หรือไม่ใช่ความสามารถเหล่านี้ก็ได้ตามที่เห็นเหมาะสม Bokeh นั้นยืดหยุ่น จากที่มันไม่อนุมานว่าเราจะใช้งานมันอย่างไร และเพื่อให้ทำงานได้ มันก็ไม่พึ่งพาไลบรารีอื่นมากมาย (เหมือนอย่าง pandas หรือ notebooks)

Concourse
ทดลอง

การจะสร้างบิลด์ไปป์ไลน์ ที่สามารถบิลด์และดีพลอยซอฟต์แวร์ไปสู่สภาพแวดล้อมต่าง ๆ ได้หลากหลาย ต้องการเครื่องมือที่ให้ความสำคัญกับเรื่อง “ไปป์ไลน์และแพ็กเกจที่ได้” มาก่อนเรื่องอื่น นานมาแล้วเราใช้ Concourse เราก็ถูกใจหลายอย่าง ตั้งแต่วิธีการอันเรียบง่ายและยืดหยุ่น กับหลักการที่ว่าทุกบิลด์ต้องสร้างเป็นคอนเทนเนอร์เสมอ และการบังคับให้กำหนด ไปป์ไลน์ด้วยโค้ด (pipeline as code) เท่านั้น ในปัจจุบันมันได้ปรับปรุงให้สะดวกต่อการใช้งานมากขึ้น ส่วนวิธีการอันเรียบง่ายของมัน ก็ได้ผ่านบทพิสูจน์ของเวลาแล้วว่าดี ที่ผ่านมามีหลายทีมของเราและลูกค้าที่ได้ติดตั้งไปป์ไลน์ขนาดใหญ่ด้วย Concourse แล้วประสบผลสำเร็จ นอกจากนั้น บ่อยครั้งที่เราใช้ประโยชน์จากความยืดหยุ่นของตัวเวิร์คเกอร์ ที่สามารถทำงานจากที่ใดก็ได้ เช่น กรณีการทดสอบการทำงานร่วมกันฮาร์ดแวร์นั้นต้องติดตั้งบนเครื่องที่เชื่อมต่ออยู่เท่านั้น

Dash
ทดลอง

ในเรดาร์ฉบับนี้ เราได้แนะนำเครื่องมือหลายตัวด้วยกัน ที่สามารถสร้างเว็บแอปพลิเคชันเพื่อให้ผู้ใช้เห็นภาพหรือโต้ตอบกับข้อมูลได้ เครื่องมือในกลุ่มนี้เก่งกว่าไลบรารีสำหรับสร้างกราฟทั่วไปอย่าง D3 เพราะช่วยทุ่นแรงการสร้างแอปพลิเคชันวิเคราะห์ข้อมูล จากข้อมูลที่มีอยู่แล้ว Dash โดย Plotly กำลังได้รับความในนิยมในหมู่นักวิทยาศาสตร์ข้อมูล จากการเป็นเครื่องมือที่สามารถสร้างแอปพลิเคชันด้านการวิเคราะห์ด้วยภาษา Python มันทำงานโดยเข้าไปเพิ่มความสามารถใหม่ให้กับดาต้าไลบรารีใน Python คล้ายกับที่ Shiny ทำกับภาษา R แอปพลิเคชันพวกนี้ที่ถูกสร้างขึ้น มักจะถูกเรียกกันอย่างติดปากว่าเป็น “แดชบอร์ด” แต่ความสามารถของมันทำได้มากกว่าที่สื่อไปไกล โดย Dash เก่งพอจะสร้างแอปพลิเคชันที่รองรับการต่อขยาย และพร้อมที่จะใช้งานได้จริงบนระบบโปรดักชัน ต่างไปจากเครื่องมือในกลุ่มเดียวกันอย่าง Streamlit หากคุณต้องนำเสนอข้อมูลเชิงวิเคราะห์ที่สลับซับซ้อนให้กับผู้ใช้ฝั่งธุรกิจ เราอยากให้ลองพิจารณา Dash แทนการใช้โซลูชันที่ไม่อาศัยการเขียนโค้ด หรือที่เขียนได้แต่เพียงน้อย อย่างที่ Tableau มีให้

jscodeshift
ทดลอง

การจะดูแลโค้ดเบสขนาดใหญ่ไม่เคยเป็นเรื่องง่าย โดยเฉพาะการไมเกรตไปหาโค้ดใหม่ ที่บางฟีเจอร์ไม่รองรับกับของเดิม (breaking change) ถึงแม้ IDE หลาย ๆ ตัวจะสามารถรีแฟคเตอร์โค้ดได้ แต่ก็ใช้ได้เฉพาะกับงานที่ไม่ซับซ้อนนัก หากเจอกรณียาก ๆ ก็อาจจะช่วยไม่ไหว ต้องมีมนุษย์เข้ามาดูเอง เช่น ตอนที่โค้ดของเราเป็นไลบรารีที่มีใครมาเรียกใช้อย่างกว้างขวาง ทุกครั้งที่จะแก้ไขอะไร และมีโค้ดส่วนที่ไม่รองรับกับของเดิมอยู่ด้วย นักพัฒนาจะต้องตามไปแก้โค้ดของส่วนที่มาเรียกใช้ให้ถูกต้องตาม ซึ่งเป็นงานที่ผิดได้ง่ายและใช้เวลา ความลำบากนี้จะบรรเทาลงได้ถ้าใช้ jscodeshift ซึ่งเป็นชุดเครื่องมือสำหรับการรีแฟคเตอร์โค้ดในภาษา JavaScript และ TypeScript มันทำงานโดยการคอมไพล์เป็นกราฟเอเอสที (Abstract Syntax Trees - AST) ที่เราสามารถเข้าไปแก้ไขเปลี่ยนแปลงรูปข้อมูลนี้ ผ่านการใช้ตัวแปลงต่าง ๆ ที่มันมี API เตรียมไว้ให้ (ยกตัวอย่าง เช่น คุณสามารถเพิ่ม ลบ เปลี่ยนชื่อ พร็อพเพอร์ตี้ใด ๆ จากคอมโพเนนท์ที่มีอยู่แล้วได้) จากนั้นมันจะแปลง AST ที่ผ่านการแก้ไขแล้ว กลับไปเป็นโค้ดชุดใหม่ให้ นอกจากนี้ มันยังมาพร้อมกับเครื่องมือช่วยเหลือการทดสอบเป็นยูนิต ทำให้คุณสามารถทดสอบงานไมเกรตโค้ดด้วย jscodeshift แบบเขียนการทดสอบก่อนลงมือทำ (test-driven development) เราเคยใช้มันตอนที่เราพยายามจะดูแลระบบ design systems แล้วพบว่ามันมีประโยชน์ดี

Kustomize
ทดลอง

Kustomize เป็นเครื่องมือไว้จัดการและแก้ไขไฟล์มานิเฟสต์ (manifest files) ของ Kubernetes ที่คุณสามารถเลือกดัดแปลงรีซอร์สพื้นฐานใด ๆ ของ Kubernetes ก่อนที่มันจะถูกติดตั้งลงในสภาพแวดล้อมต่าง ๆ ในปัจจุบันคำสั่ง kubectl ก็รองรับ Kustomize โดยตรงแล้วด้วย เราชอบที่มันช่วยรักษาโค้ดให้ ไม่ซ้ำซ้อนกัน (DRY) หากจะเปรียบเทียบกับ Helm ที่พยายามจะเป็นหลายอย่าง ไม่ว่าจะเป็นตัวจัดการแพ็คเกจ ตัวจัดการเวอร์ชัน เป็นต้น เราพบว่า Kustomize นั้นดำเนินตามรอยปรัชญาของ Unix มากกว่า นั่นคือ การทำสิ่งหนึ่งเพียงสิ่งเดียวให้ดี และการเคารพว่าเอาท์พุตของโปรแกรมหนึ่ง สามารถต่อเป็นอินพุตของอีกโปรแกรมหนึ่งได้เสมอ

MLflow
ทดลอง

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

Pitest
ทดลอง

วิธีการทดสอบทั่วไปจะมุ่งไปที่การตรวจสอบว่าโค้ดโปรดักชันของเราทำงานถูกต้องตามที่ควรหรือไม่ อย่างไรก็ดี บางทีเราอาจจะเผลอ สร้างโค้ดการทดสอบที่ไม่สมบูรณ์หรือไร้ประโยชน์ขึ้นมา จนมีความมั่นใจผิด ๆ ว่าโค้ดเรายังทำงานได้ถูกต้องอยู่ ซึ่งการทดสอบแบบมิวเทชัน (mutation testing) สามารถตรวจจับปัญหานี้ได้ โดยมันจะทำหน้าที่ประเมินคุณภาพของโค้ดชุดทดสอบ และช่วยให้พบกรณีแปลก ๆ ที่ยากจะสังเกต การทดสอบมิวเทชันทำงานโดยการเข้าแก้ไขโค้ดโปรดักชันของเรา แล้วลองเรียกทดสอบดู โดยหวังว่าการทดสอบจะตรวจพบข้อผิดปกติที่ใส่เข้าไปได้ แต่หากผลการทดสอบยังคงผ่านอยู่ แสดงว่าคุณภาพชุดทดสอบนั้นยังไม่ดีพอ ต้องได้รับการปรับปรุง ทีมของเราได้ใช้ Pitest มาสักระยะแล้วได้ผลดี จึงอยากแนะนำให้ใช้ต่อ เพื่อใช้ประเมินคุณภาพของชุดทดสอบของคุณ มันเหมาะแก่การใช้กับโครงการที่เขียนด้วยภาษา Java แต่หากคุณใช้ภาษาอื่นอยู่ ก็ลองทางเลือกอย่าง Stryker ดูก็ได้

Sentry
ทดลอง

Sentry เป็นเครื่องมือสำหรับมอนิเตอร์แอปพลิเคชัน ที่สนใจไปที่การรายงานความผิดปกติของระบบ เครื่องมือประเภทนี้ แตกต่างจากเครื่องมือจัดการล็อกบันทึกเหตุการณ์ทั่วไป อย่าง ELK Stack ตรงที่มันจะเน้นความสามารถไปที่การค้นพบ สืบหา และแก้ไขข้อผิดปกติโดยเฉพาะ Sentry เป็นแอปพลิเคชันที่มีมาสักพักใหญ่ และรองรับหลายภาษาและเฟรมเวิร์ค เราได้ใช้มันในหลาย ๆ โครงการ ซึ่งเราก็ได้ใช้ประโยชน์จากการติดตามความผิดปกติ มันช่วยบอกได้ว่าคอมมิทใด เป็นตัวแก้ไขความผิดปกตินั้น ๆ และคอยแจ้งเตือนเมื่อพบว่า มีความผิดปกติใดที่หายไปแล้วกลับขึ้นมาอีก อันเนื่องจากข้อผิดพลาดในระบบ

ShellCheck
ทดลอง

แม้เครื่องมือช่วยเหลือด้านโครงสร้างพื้นฐานจะดีขึ้นมากเพียงใด แต่การใช้เชลล์สคริปต์ก็ยังเหมาะกับบางสถานการณ์อยู่ดี เป็นความจริงที่ว่า ไวยกรณ์ของภาษาเชลล์สคริปต์เรียกได้ว่าเป็นภาษาลึกลับเลยทีเดียว อาจเป็นเพราะว่า เราไม่ค่อยจะได้ใช้มันบ่อยเท่าไหร่นัก นั่นทำให้เราชื่นชอบ ShellCheck ซึ่งเป็นเครื่องมือตรวจสอบรูปแบบและไวยากรณ์ (linter) สำหรับเชลล์คริปต์ สามารถสั่งการได้ผ่านชุดบรรทัดคำสั่ง หรือทำงานเป็นส่วนหนึ่งของบิลด์ หรือที่ดีไปกว่านั้น สามารถใช้เป็นส่วนเสริมร่วมกับ IDE เจ้าดังต่าง ๆ ShellCheck สามารถตรวจสอบรายละเอียดได้หลายร้อยปัญหา โดยปัญหาเหล่านั้นได้ถูกระบุรายละเอียดไว้ในวิกิเพจ เพื่อความสะดวก เครื่องมือและ IDE ต่าง ๆ ก็จะมีทางเชื่อมโยงไปหาวิกิเพจเหล่านั้นให้ตอนพบเจอปัญหา

Stryker
ทดลอง

Stryker เป็นน้องใหม่ในกลุ่มเครื่องมือทดสอบคุณภาพของชุดทดสอบ (mutation testing) มีความสามารถคล้ายกับ Pitest ที่สามารถใช้เป็นเครื่องมือวัดคุณภาพชุดทดสอบที่มีอยู่ ทีมของเราใช้มันกับโครงการต่าง ๆ ที่พัฒนาด้วย JavaScript แล้วได้ผลเป็นที่น่าพอใจ มันยังรองรับภาษาอื่นอย่าง C# และ Scala ด้วยเช่นกัน Stryker ใช้งานง่ายและปรับแต่งได้หลากหลาย จากที่ได้ใช้งาน มันช่วยเพิ่มความครอบคลุมของชุดทดสอบ และความมั่นใจในแอปพลิเคชันที่เรากำลังส่งมอบให้ลูกค้า

Terragrunt
ทดลอง

เราใช้ Terraform อย่างหนัก เพื่อสร้างและจัดการโครงสร้างพื้นฐานของคลาวด์ จากการใช้งานมันกับงานขนาดใหญ่ ที่ต้องแบ่งงานออกเป็นโมดูลย่อยแล้วค่อยร้อย

เครื่องมือ

Kustomize เป็นเครื่องมือไว้จัดการและแก้ไขไฟล์มานิเฟสต์ (manifest files) ของ Kubernetes ที่คุณสามารถเลือกดัดแปลงรีซอร์สพื้นฐานใด ๆ ของ Kubernetes ก่อนที่มันจะถูกติดตั้งลงในสภาพแวดล้อมต่าง ๆ

(Kustomize)

Sentry เป็นเครื่องมือสำหรับมอนิเตอร์แอปพลิเคชัน ที่สนใจไปที่การรายงานความผิดปกติของระบบ มันช่วยให้ทีมของเราสามารถติดตามความผิดปกติบอกได้ว่าคอมมิทใดเป็นตัวแก้ไขความผิดปกตินั้น ๆ และคอยแจ้งเตือนเมื่อพบว่า มีความผิดปกติใดที่หายไปแล้วกลับขึ้นมาอีก อันเนื่องจากข้อผิดพลาดในระบบ

(Sentry)

เครื่องมือ

Flagger เป็นเครื่องมือที่ช่วยในการปรับแต่งปริมาณโหลดที่วิ่งไปหาเซอร์วิสใด ๆ เพื่อทดสอบความพร้อมของเซอร์วิสนั้นตอนที่มีเวอร์ชันใหม่ออกมา ซึ่งจะสะดวกมากเมื่อต้องทำงานที่เกี่ยวกับเซอร์วิสเมช และเอพีไอเกตเวย์

(Flagger)

รวมกันทีหลัง ทีมเราเจอว่า มีโค้ดบางส่วนระหว่างโมดูลที่เขียนซ้ำซ้อนกันอย่างหลีกเลี่ยงไม่ได้ อันเกิดจากข้อจำกัดของ Terraform เอง ซึ่งปัญหานี้ทีมเราแก้ด้วยการใช้ Terragrunt ซึ่งเป็นไลบรารีขนาดเล็กที่ครอบ Terraform ไว้บาง ๆ ที่เพิ่มความสามารถตามแนวปฏิบัติที่ Yevgeniy Brikman แนะนำไว้ในหนังสือ *Terraform: Up and Running* ที่เขาเป็นคนแต่ง เราพบว่า Terragrunt ช่วยได้มาก เพราะมันสนับสนุนให้เรากำหนดโมดูลเป็นเวอร์ชันและการสร้างโมดูลที่ใช้ซ้ำได้ในสภาพแวดล้อมที่แตกต่างกัน ส่วนความสามารถด้านฮุก (lifecycle hook) เป็นอีกคุณสมบัติที่มีประโยชน์และช่วยเพิ่มความยืดหยุ่นในการทำงาน ส่วนในเรื่องการทำเป็นแพ็กเกจ Terragrunt มีข้อจำกัดเดียวกันกับ Terraform ที่ว่า มันไม่มีวิธีชัด ๆ ในการกำหนดแพ็กเกจ หรือการอ้างอิงระหว่างแพ็กเกจที่เกี่ยวข้องกัน ซึ่งก็สามารถแก้ปัญหาเฉพาะหน้า ด้วยการทำเป็นโมดูลย่อย ๆ แล้วระบุเวอร์ชันผ่านการใช้แท็กของ Git

tfsec

ทดลอง

ความมั่นคงเป็นหน้าที่ของทุกคนไม่ใช่ของใครคนใดคนหนึ่ง และการตรวจพบความเสี่ยงตั้งแต่เนิ่น ๆ ย่อมดีกว่าการพบปัญหาในภายหลังเสมอ

เราได้กล่าวถึงเทคนิค การกำหนดโครงสร้างพื้นฐานด้วยโค้ด ไปบ่อยครั้ง ซึ่ง Terraform เป็นตัวเลือกแรก ๆ สำหรับใช้บริหารสภาพแวดล้อมบนคลาวด์ เราสามารถนำ tfsec มาใช้ร่วมกับ Terraform เพื่อพิเคราะห์ตรวจหาความเสี่ยงด้านความมั่นคงที่มีโอกาสเป็นไปได้ จากที่เราใช้งานมา มันให้ผลค่อนข้างเป็นที่น่าพอใจเลยทีเดียว เครื่องมือนี้สามารถติดตั้งและใช้งานได้สะดวก จึงเป็นทางเลือกอันยอดเยี่ยมสำหรับทีมที่มุ่งมั่นจะปิดความเสี่ยงด้านความมั่นคง เพื่อป้องกันความเสียหายที่อาจเกิด

tfsec เตรียมกฎการตรวจสอบสำเร็จรูปสำหรับการทำงานร่วมกับผู้ให้บริการคลาวด์เจ้าต่าง ๆ ให้พร้อม รองรับทั้ง AWS และ Azure หากทีมกำลังใช้งาน Terraform อยู่แล้ว การใช้ tfsec ร่วมด้วย จึงเป็นการส่งเสริมกันดี

Yarn

ทดลอง

Yarn ยังคงเป็นเครื่องมือบริหารจัดการแพ็คเกจในดวงใจของหลาย ๆ ทีม ขณะเดียวกัน Yarn 2 ที่เพิ่งออกก็ดูน่าสนใจเอามาก มีการปรับปรุงและพัฒนาความสามารถไปอย่างมากมายจากเวอร์ชันก่อนหน้า นอกจากที่มันปรับปรุงให้เวิร์คเสปซใช้งานง่ายขึ้นแล้ว ยังนำเสนอคอนเซ็ปต์ใหม่ *zero-installs* เข้ามา ซึ่งช่วยให้นักพัฒนาสามารถรันงานได้ทันทีหลังจากโคลนโปรเจกต์ลงมา โดยไม่ต้องดาวน์โหลดไลบรารีก่อน ทั้งนี้ ก็ต้องระวังไว้ด้วย เพราะ Yarn 2 มีการเปลี่ยนแปลงบางอย่างที่ไม่รองรับความสามารถที่มีในเวอร์ชันก่อน ทำให้การอัพเกรดนั้นค่อนข้างยุ่งยากไม่ตรงตัว นอกจากนั้น มันยังเปิดใช้งานสภาพแวดล้อมแบบใหม่ ที่เป็นแบบ plug'n'play (PnP) มาให้ตั้งแต่ต้น ซึ่งขณะนี้ React Native ยังไม่สามารถทำงานร่วมกับสภาพแวดล้อมแบบ PnP ได้ ซึ่งทีมมีทางเลือกคือปิดการใช้งาน PnP ไปก่อน หรือใช้ Yarn 1 ต่อไป แต่หากจะใช้ Yarn 1 จริง ๆ ก็ต้องตระหนักไว้ด้วยว่า ขณะนี้ Yarn 1 ได้เข้าสู่โหมดบำรุงรักษา (maintenance mode) เรียบร้อยแล้ว

CML

ประเมิน

ในเรดาร์ฉบับที่แล้ว เราพูดถึงเทคนิคการ ส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML) ไปแล้ว ในฉบับนี้เราอยากให้ความสนใจกับเครื่องมือตัวใหม่ที่ดูมีแวว ที่ชื่อว่า Continuous Machine Learning หรือ CML ที่ถูกพัฒนาโดยกลุ่มเดียวกันที่สร้าง DVC ขึ้นมา CML มุ่งเป้าจะนำหลักปฏิบัติทางวิศวกรรมในด้าน CI/CD มาใช้กับงาน AI และ ML แล้วเลือกใช้ชุดเครื่องมือเดิมที่นักพัฒนาคุ้นชินอยู่แล้ว เอามาจัดระเบียบใหม่ ทำเป็นโครงสร้างพื้นฐานสำหรับทำ MLOps แทนการสร้าง AI แพลตฟอร์มขึ้นมาใหม่โดยเฉพาะ เราชอบที่มันให้ความสำคัญกับการรองรับ DVC อันเป็นสัญญาณที่ดีว่าเครื่องมือตัวนี้กำลังเติบโต

Eleventy

ประเมิน

พวกเราชอบแนวคิดของพวก เครื่องมือก่อกำเนิดเว็บไซต์สแตติก (static site generator) มานานแล้ว เราจะใช้มันเสมอหากเหมาะโอกาส เพราะมันช่วยหลีกเลี่ยงความซับซ้อนและให้ประสิทธิภาพที่ดี Eleventy เป็นเครื่องมือที่อยู่มาหลายปีแล้ว แต่เพิ่งจะเข้าตาเรา จากที่มันมีความสมบูรณ์มากขึ้น และจากที่เครื่องมือที่เราเคยโปรดอย่าง Gatsby.js แสดงปัญหาด้านการรองรับการขยายออกมา โดย Eleventry นั้นเรียนรู้ได้ง่าย ช่วยขึ้นเว็บไซต์ได้เร็ว เรายังชอบความง่ายในการสร้างมาร์คอัพที่มีความหมาย (semantic markup) ผ่านเทมเพลตที่มันเตรียมให้ (ซึ่งทำให้ผู้ใช้เข้าถึงเว็บไซต์ได้สะดวกขึ้น) และมีวิธีแบ่งหน้าการแสดงผล (pagination) ที่เรียบง่ายแต่ให้ผลดี

Flagger

ประเมิน

การใช้ เซอร์วิสเมช และเอพีไอเกตเวย์ ช่วยให้การกระจายโหลดภายในคลัสเตอร์ของไมโครเซอร์วิสใด ๆ ทำได้สะดวก Flagger ใช้ความสามารถนี้ปรับแต่งปริมาณโหลดที่วิ่งไปหาเซอร์วิสใด ๆ เพื่อทดสอบความพร้อมของเซอร์วิสนั้นตอนที่มันมีเวอร์ชันใหม่ออกมา ซึ่งความสามารถนี้เป็นสิ่งที่เทคนิคอย่าง แคนารีรีลีส (canary releases) และการดีพลอยแบบบลูกรีน (blue/green deployment) นั้นต้องการ โดย Flagger สามารถทำงานร่วมกับพร็อกซียอดนิยมต่าง ๆ (เช่น Envoy และ Kong) เพื่อทยอยป้อนรีเควสเข้าสู่เซอร์วิส และรายงานผลความสำเร็จของเซอร์วิสใหม่ เราชอบที่

มันทำให้แนวปฏิบัติที่ดีเช่นนี้เป็นจริงได้ง่ายขึ้น คนจะได้หันมาใช้กันมากกว่านี้ และแม้ว่ามันจะถูกพัฒนาและสนับสนุนโดย Weaveworks แต่มันทำงานได้อย่างอิสระไม่ขึ้นกับเครื่องมืออื่น ๆ ของ Weaveworks แต่อย่างใด

gossm

ประเมิน

เมื่อต้องการเชื่อมต่อกับเซิร์ฟเวอร์บน AWS เป็นที่แนะนำกันว่าควรเชื่อมต่อผ่านเครื่องบาสเตียน (bastion host) แทนการเชื่อมต่อตรง ๆ

แต่การจะต้องสร้างเครื่องบาสเตียนขึ้นมาเพียงเพื่อจุดประสงค์นี้เท่านั้น อาจดูไม่คุ้มและยุ่งยากเกินไป เป็นสาเหตุที่ AWS ถึงมีบริการ AWS Systems Manager's Session Manager ให้คุณสามารถมุดเข้าไปที่เครื่องเซิร์ฟเวอร์ใด ๆ ที่ต้องการได้สะดวกยิ่งขึ้น gossm เป็นเครื่องมือบรรทัดคำสั่ง (cli) โอเพนซอร์สที่ช่วยให้การใช้งาน Session Manager นั้นง่ายขึ้นไปอีก

โดยคุณสามารถใช้ประโยชน์จากระบบความมั่นคงที่มีอยู่ใน Session Manager กับกฎเกณฑ์ IAM ที่ประกาศไว้ เพื่อเรียกใช้เครื่องมือคำสั่ง `ssh` และ `scp` ในเทอร์มินัลของเราโดยตรง นอกจากนี้ยังมีคำสั่งแสดงรายการเซิร์ฟเวอร์ที่มีและสามารถเชื่อมต่อกับ SSH ได้โดยตรง ซึ่งเป็นคำสั่งที่ขาดไปจากชุดบรรทัดคำสั่งมาตรฐานของ AWS

Great Expectations

ประเมิน

จากความตื่นตัวเรื่อง CD4ML ที่มากขึ้น ทำให้แง่มุมต่าง ๆ ของการดูแลรักษาข้อมูลของนักวิทยาศาสตร์ข้อมูลหรือวิศวกรข้อมูล นั้นได้รับความสนใจมากขึ้นไปด้วย ซึ่งเรื่องการกำกับดูแลข้อมูลอย่างอัตโนมัติ (automated data governance) ก็เป็นอีกแง่มุมหนึ่งที่ได้รับการพัฒนามากขึ้นเช่นกัน Great Expectations เป็นเฟรมเวิร์คที่คุณสามารถสร้างตัวควบคุมต่าง ๆ ขึ้นมา เพื่อใช้ตรวจจับความผิดปกติหรือปัญหาคุณภาพในไปป์ไลน์ข้อมูล คล้ายกับที่การทดสอบต่าง ๆ จะถูกทดสอบในบิลด์ไปป์ไลน์ Great Expectations จะทำหน้าที่ตรวจสอบข้อมูลในไปป์ไลน์ข้อมูลไม่ต่างกัน นอกจากการเป็น ตัวหยุดสายพาน ของไปป์ไลน์ที่ดีแล้ว มันยังมีประโยชน์อย่างอื่นคือ เป็นตัวช่วยทำให้แน่ใจว่าอัลกอริทึมที่อาศัยโมเดลต่าง ๆ จะอยู่ในขอบเขตที่ถูกต้องเสมอ ที่กำหนดโดยชุดข้อมูลที่จะใช้ฝึก การควบคุมอย่างเป็นอัตโนมัติเช่นนี้ ช่วยให้ข้อมูลอยู่กระจายตัวกันได้ ทำให้เกิดความเท่าเทียมด้านการเข้าถึงข้อมูล และความรับผิดชอบต่อข้อมูล นอกจากนี้ Great Expectations ยังมาพร้อมกับเครื่องมือช่วยวิเคราะห์การทำงาน (profiler tool) ที่ช่วยให้เราทำความเข้าใจคุณภาพของข้อมูลเฉพาะจุดที่สนใจ และช่วยหาขอบเขตข้อมูลที่เหมาะสม

k6

ประเมิน

k6 เป็นเครื่องมือสำหรับทดสอบประสิทธิภาพที่เพิ่งออกมาได้ไม่นานที่พวกเราต่างพูดถึงกันใหญ่ มีจุดเด่นตรงที่มันให้ความใส่ใจต่อประสบการณ์การใช้งานของนักพัฒนา เราสามารถใช้ k6 ผ่านชุดบรรทัดคำสั่ง (command line) ด้วยการส่งสคริปต์การทดสอบที่เขียนด้วยภาษา JavaScript เข้าไป แล้วปรับแต่งระยะเวลาและจำนวนผู้ใช้ที่ต้องการจำลองในระบบทดสอบ มันยังมี ความสามารถขั้นสูง อีกมากให้ปรับแต่ง เช่น สั่งให้แสดงค่าสถิติระหว่างการทดสอบทันที ไม่ต้องรอให้การทดสอบสิ้นสุดจึงแสดงผล สั่งปรับเพิ่มลดจำนวนผู้ใช้ที่ต้องการจำลองระหว่างการทดสอบ หรือสั่งพักหรือสั่งเดินหน้าการทดสอบ ผลลัพธ์ของการทดสอบจะประกอบไปด้วยคะแนนค่าชี้วัดที่ปรับแต่งได้ สามารถแปลงค่าเพื่อส่งไปแสดงผลที่ Datadog หรือเครื่องมืออื่น ๆ ที่ใช้สำหรับงานสังเกตการณ์

เราสามารถทำให้การทดสอบประสิทธิภาพเป็นส่วนหนึ่งของไปป์ไลน์ CI/CD ได้ง่าย ๆ โดยเพิ่ม checks เข้าไปในสคริปต์ด้วย หากต้องการเร่งการทดสอบด้านประสิทธิภาพ คุณอาจพิจารณาใช้ k6 Cloud อันเป็นเวอร์ชันทางการค้าของมัน ที่มีความสามารถการขยายการทดสอบไปทำงานในคลาวด์ สามารถแสดงผลลัพธ์ออกมาเป็นแผนภาพที่หลากหลายกว่าเวอร์ชันโอเพนซอร์ส

Katran

ประเมิน

Katran เป็นเครื่องมือโหลดบาลานซ์ระดับ L4 ที่ให้ประสิทธิภาพสูง มันอาจไม่จำเป็นสำหรับงานทั่วไปนัก แต่ถ้าใครที่มองหาเครื่องมือโหลดบาลานซ์อีกตัวมาสำรองเครื่องมือโหลดบาลานซ์ระดับ L7 (อย่างเช่น HAProxy หรือ NGINX) ที่มีอยู่ หรือถ้าต้องการขยายงานโหลดบาลานซ์ออกไปที่เครื่องที่สองหรือมากกว่านั้น เราแนะนำให้ประเมิน Kratan ดู เราเห็นว่าการใช้ Katran เป็นทางเลือกที่ยืดหยุ่นและให้ประสิทธิภาพดีกว่าการใช้ DNS เพื่อกระจายงานไปหากลุ่มเครื่องมือโหลดบาลานซ์ระดับ L7 หรือการกระจายด้วย IPVS ในเคอร์เนลของลีนุกซ์ ที่วิศวกรเน็ตเวิร์คมักใช้กันเพื่อแก้ปัญหานี้

Kiali

ประเมิน

จากความนิยมนำเซอร์วิสเมชมา ใช้ดีพลอยคอนเทนเนอร์ต่าง ๆ เป็นไมโครเซอร์วิสกันมากขึ้น แน่นอนว่าจะมีเครื่องมือใหม่ ๆ ออกมาด้วยเช่นกัน ที่จะมาช่วยให้การดูแลสถาปัตยกรรมนี้ง่ายขึ้น และเป็นอัตโนมัติมากขึ้น Kiali เองก็เป็นหนึ่งในนั้น โดยมันมี UI สำหรับสังเกตและควบคุมเครือข่ายเซอร์วิสที่ Istio เป็นผู้จัดการ ช่วยให้เรามองเห็นภาพความสัมพันธ์ของเซอร์วิสต่าง ๆ ในเน็ตเวิร์ค และเข้าใจปริมาณข้อมูลที่ไหลไปมาระหว่างกัน เช่น เมื่อใช้ควบคู่ไปกับ Flagger ขณะอยู่ในการรีลีสแบบแคนารี มันสามารถแสดงปริมาณรีเควสต์ที่กำลังถูกส่งไปยังเซอร์วิสใหม่ได้ เราชอบความสามารถหนึ่งของ Kiali เป็นพิเศษ ที่สามารถจำลองความล้มเหลวของเครือข่ายเข้าไปในเซอร์วิสเมช เพื่อทดสอบดูว่า ระบบมีความพร้อมแค่ไหนหากต้องรับมือกับภาวะเน็ตเวิร์คสะดุด ซึ่งการทดสอบเช่นนี้ มักจะถูกละเลยไปเสมอ เนื่องจากมันยากที่จะกำหนดค่าและทดสอบความล้มเหลวในไมโครเซอร์วิสที่มีโครงข่ายซับซ้อน

เครื่องมือ

Great Expectations เป็นเฟรมเวิร์คที่คุณสามารถสร้างตัวควบคุมต่าง ๆ ขึ้นมาเพื่อใช้ตรวจจับความผิดปกติหรือปัญหาคุณภาพในไปป์ไลน์ข้อมูล ช่วยให้คุณสามารถกำกับดูแลข้อมูลได้อย่างอัตโนมัติ (automated data governance)

(Great Expectations)

LGTM คือเครื่องมือพิเคราะห์โค้ด (static code analysis tool) ที่มุ่งเน้นตรวจสอบด้านความมั่นคงโดยเฉพาะ โดยอ้างอิงข้อมูลจากกฎเกณฑ์ด้านการเขียนโค้ดที่มั่นคงที่มันได้รวบรวมไว้

(LGTM)

เครื่องมือ

Sensei ช่วยให้การกำหนดและกระจายแนวทางการเขียนโค้ดที่มั่นคง (secure code) นั้นกลายเป็นเรื่องที่ง่าย

(Sensei)

LGTM
ประเมิน

การเขียนโค้ดที่มั่นคง (secure coding) เป็นเรื่องสำคัญกว่าเรื่องไหน แต่นักพัฒนาก็ยังมีอีกหลายเรื่องที่ต้องดูแลให้ความสำคัญ LGTM ทำหน้าที่รักษาความมั่นคงให้เราอีกชั้นหนึ่ง ในขณะเดียวกันก็พยายามเก็บเกี่ยวความรู้จากหลักปฏิบัติด้านการเขียนโค้ดที่มั่นคงจากโครงการต่าง ๆ มาไว้ที่มัน มันทำงานเป็นเครื่องมือพิเคราะห์โค้ด (static code analysis tool) ที่มุ่งเน้นตรวจสอบด้านความมั่นคงโดยเฉพาะ โดยอ้างอิงข้อมูลจากกฎเกณฑ์ด้านการเขียนโค้ดที่มั่นคงที่มันได้รวบรวมไว้ (ที่ส่วนหนึ่งเป็นโอเพนซอร์ส) ซึ่งการบังคับใช้กฎเกณฑ์ใด สามารถทำได้ผ่านการคิวรีไปที่โค้ดใด ๆ ด้วยภาษา CodeQL คุณสามารถนำ LGTM ไปเชื่อมต่อในไปป์ไลน์เพื่อตรวจสอบความมั่นคงของโค้ดได้เช่นกัน โดยมันรองรับทั้งภาษา Java, Go, JavaScript, Python, C# และ C/C++ ทั้งนี้ LGTM และ CodeQL เป็นส่วนหนึ่งของโครงการ Github Security Lab

Litmus
ประเมิน

Litmus เป็นเครื่องมือจำลองความผิดปกติให้กับระบบ (chaos engineering) ที่ใครก็เริ่มต้นใช้ทดสอบระบบได้ไม่ยาก มันสามารถจำลองเหตุการณ์ความผิดปกติหลากหลายรูปแบบให้กับคลัสเตอร์ของ Kubernetes เราประทับใจความครอบคลุมของ Litmus ที่ทำได้มากกว่าการไล่ทุบพ็อดทิ้ง แต่สามารถจำลองสถานการณ์ความผิดปกติอื่น ๆ ที่เกิดจากเน็ตเวิร์ก หน่วยประมวลผล หน่วยความจำ หรือจาก I/O ได้ด้วย นอกจากนั้น มันยังมีแบบจำลองสถานการณ์ความผิดปกติที่ทำขึ้นเฉพาะเพื่อทดสอบ Kafka และ Cassandra อีกด้วย

Opacus
ประเมิน

แนวคิดของ differential privacy เป็นเทคนิคการรักษาความเป็นส่วนตัวด้วยการปนเปื้อนข้อมูล ปรากฏตัวครั้งแรกในเรดาร์ปี 2016 แม้การละเมิดความเป็นส่วนตัวผ่านการคิวรี่ที่สามารถอนุมานหาคำตอบได้ด้วยโมเดลการคิดอย่างเป็นระบบ จะเป็นที่ทราบกันว่าเกิดขึ้นได้แต่โดยส่วนมากถูกมองว่าเป็นปัญหาทางทฤษฎีมากกว่า เนื่องด้วยแนวทางแก้ไขมีอยู่น้อยมาก แวดวงนี้ยังขาดเครื่องมือที่จะป้องกันไม่ให้สิ่งนี้เกิดขึ้น Opacus เป็นไลบรารีใหม่ของ Python ที่สามารถนำมาใช้เสริมกับ PyTorch เพื่อขัดขวางการโจมตีประเภทนี้ ถึงแม้นี่เป็นพัฒนาการที่มีแนวโน้มดี แต่การหาโมเดลที่ถูกต้องและชุดข้อมูลที่จะนำมาใช้ได้นั้นค่อนข้างท้าทาย ไลบรารีนี้ยังค่อนข้างใหม่มาก ดังนั้นเราจะคอยดูว่ามันจะได้รับการยอมรับอย่างไรต่อจากนี้

OSS Index
ประเมิน

มันเป็นเรื่องสำคัญสำหรับทีมพัฒนา ที่จะต้องระบุได้ว่าไลบรารีใดที่แอปพลิเคชันของเราพึ่งพาอยู่นั้น มีช่องโหว่หรือไม่ ซึ่งเราสามารถนำ OSS Index มาช่วยจัดการเรื่องนี้ OSS Index ได้รวบรวมและจัดโอเพนซอร์สคอมโพเนนท์และเครื่องมือสแกนต่าง ๆ ไว้เป็นหมวดหมู่ ให้นักพัฒนาใช้มันระบุหาช่องโหว่ ทำความเข้าใจความเสี่ยงที่เกี่ยวข้องและรักษาให้ซอฟต์แวร์มั่งคงอยู่เสมอ ทีมของเราได้เชื่อมต่อ OSS Index เป็นส่วนหนึ่งของไปป์ไลน์ด้วยวิธีที่ต่างกันไปตามภาษาที่ใช้ ผ่านเครื่องมืออย่าง AuditJS หรือ Gradle plugin เราพอใจกับผลลัพธ์ที่ได้ มันทำการตรวจสอบหาช่องโหว่ได้อย่างรวดเร็ว แม่นยำ และมีการแจ้งเตือนที่ไม่ใช่เรื่องจริงเพียงเล็กน้อยเท่านั้น

Playwright
ประเมิน

ตลาดของกลุ่มเครื่องมือช่วยทดสอบ UI ของเว็บไซต์ยังคงแข่งขันกันไม่หยุด Playwright ถูกพัฒนาจากประสบการณ์ความรู้ ของทีมผู้สร้าง Puppeteer กลุ่มหนึ่ง ที่ย้ายมาทำงานให้กับ Microsoft โดยมันมีความสามารถที่น่าสนใจคือ ผู้พัฒนาสามารถเขียนชุดทดสอบให้รองรับได้ทั้งเบราว์เซอร์ Chromium, Firefox และ Webkit ผ่าน API

ชุดเดียวกันทั้งหมด จากการรองรับทุกเอนจิ้นหลัก ๆ ได้ทั้งหมด ทำให้ Playwright ได้รับความสนใจจากชุมชนขึ้นมาทันที เคล็ดลับการทำงานข้ามแพลตฟอร์มของมัน อยู่ที่การทำงานร่วมกับ Firefox และ Webkit เวอร์ชันดัดแปลงที่มันแนบมาให้ด้วย เรายังต้องติดตามดูกันต่อไปว่าเครื่องมืออื่น ๆ จะตามความสามารถนี้ทันหรือไม่ จากที่หลายเจ้าหันมาใช้ Chrome DevTools Protocol เป็น API หลักมากขึ้นเรื่อย ๆ เพื่อสั่งการเบราว์เซอร์ ซึ่งโปรโตคอลนี้รองรับการสั่งการเบราว์เซอร์ได้หลายค่ายเช่นกัน

pnpm

ประเมิน

pnpm เป็นเครื่องมือจัดการแพ็คเกจดาวรุ่ง สำหรับ Node.js ที่เรากำลังจับตาดูอย่างใกล้ชิดจากความเร็วและประสิทธิภาพที่ดีกว่าเมื่อเทียบกับตัวจัดการแพ็คเกจอื่น ๆ โดยมันจะจัดเก็บไลบรารีที่พึ่งพา (dependencies) รวมไว้ที่เดียวกัน แล้วค่อยให้ไดเรกทอรี node_modules ที่เกี่ยวข้องสร้างลิงค์ไปหาแทน ทั้งนี้ pnpm ปรับแต่งประสิทธิภาพโดยละเอียดได้ถึงระดับไฟล์ มีพื้นฐาน API ที่ดีรองรับการขยายและการปรับแต่งเพิ่มเติม รองรับการใช้งานในโหมดเซิร์ฟเวอร์ ซึ่งจะทำให้การดาวน์โหลดไลบรารีต่าง ๆ เร็วยิ่งขึ้นอีก หากองค์กรของคุณมีหลายโครงการที่ใช้ไลบรารีเดียวกันซ้ำ ๆ เราอยากให้คุณลองพิจารณา pnpm ดู

Sensei

ประเมิน

Sensei จากทีมงาน Secure Code Warrior เป็นส่วนเสริมของ IDE ที่ช่วยให้ทีมสามารถกำหนดและกระจายแนวทางการเขียนโค้ดที่มั่นคง (secure code) ของภาษา Java ได้ ที่ ThoughtWorks เราจะสนับสนุนแนวคิด “ให้ใช้เครื่องมือก่อนการบังคับเสมอ” คือการส่งเสริมให้การทำเรื่องดีเป็นเรื่องง่าย มากกว่าการใช้กฎและขั้นตอนมาตรวจสอบกำกับ ซึ่งการใช้ Sensei เข้ากันดีกับปรัชญานี้ นักพัฒนาสามารถสร้างกฎกำกับไว้ แล้วส่งต่อให้เพื่อนร่วมทีมได้ใช้ร่วมกัน โดยเราสามารถเขียนกฎเป็นสูตรที่ง่ายหรือซับซ้อนอย่างไรก็ได้ เป็นภาษาคิวรีที่เล่นกับ AST ของภาษา Java ตัวอย่างของกฎที่มีให้อยู่แล้ว เช่น กฎการแจ้งเตือนเมื่อพบความเสี่ยงเรื่อง SQL injection หรือเมื่อพบวิธีการเข้ารหัสที่อ่อนหัดไป ฯลฯ อีกสิ่งหนึ่งที่เราชอบ คือ Sensei ทำงานตอนที่เรากำลังแก้ไขโค้ดใน IDE อยู่ ดังนั้นเราจะได้รับผลการตรวจสอบในทันที เร็วกว่าการใช้เครื่องมือเชิงพิเคราะห์ (static analysis tool) ทั่วไป

Zola

ประเมิน

Zola เป็นเครื่องมือสำหรับสร้างเว็บไซต์แบบสแตติก (static site generator) ที่เขียนด้วยภาษา Rust ดังนั้นมันจึงมาในรูปแบบไบนารีเดี่ยวที่ไม่พึ่งพิงไลบรารีภายนอกเลย ทำงานได้รวดเร็ว รองรับความสามารถทั่วไป ที่การสร้างเว็บไซต์ควรทำได้ เช่น รองรับการทำงานร่วมกับ Sass หรือจัดรูปแบบเนื้อหาในรูปแบบมาร์คดาว์น (Markdown) และการทำฮอตรีโหลด (Hot reloaded) พวกเราใช้ Zola สร้างเว็บไซต์สแตติกแล้วได้ผลดี เราชื่นชอบที่มันสามารถทำงานได้ดั่งใจ

TECHNOLOGY RADAR

ภาษา & เฟรมเวิร์ค

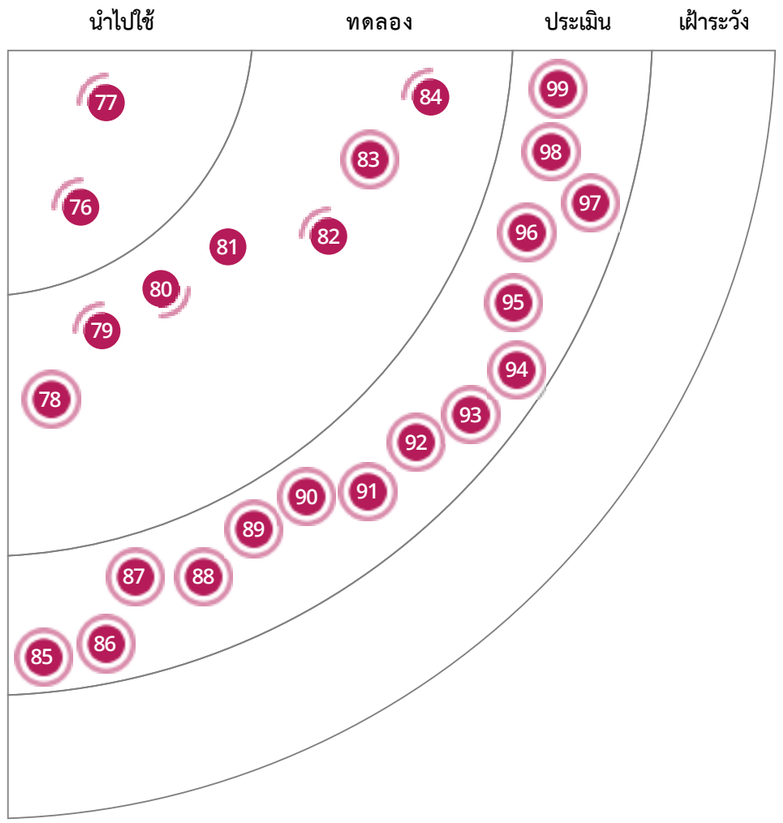
ภาษา & เฟรมเวิร์ค

Arrow
นำไปใช้

Arrow ถูกยกให้เป็นไลบรารีเสริมใน ไลบรารีมาตรฐานของ Kotlin ที่จะเพิ่มความสามารถการเขียนโปรแกรมเชิงฟังก์ชันให้ดียิ่งขึ้น ชุดฟังก์ชันแอบแสตรกชันระดับสูงที่ Arrow เตรียมไว้ได้พิสูจน์แล้วว่ามีประโยชน์อย่างมาก จนทีมของเราพิจารณาให้ใช้ Arrow เป็นไลบรารีตั้งต้นพื้นฐานเมื่อต้องทำงานกับภาษา Kotlin หมายเหตุ ไม่นานมานี้ Arrow มีการเปลี่ยนแปลงยกใหญ่เพื่อเตรียมออกเวอร์ชัน 1.0 มีการเพิ่มโมดูลใหม่เข้ามาหลายตัว พร้อมกันนี้ก็ได้ยกเลิกการสนับสนุนบางโมดูลและลบบางโมดูลออกไปด้วยเช่นกัน

jest-when
นำไปใช้

jest-when เป็นไลบรารีภาษา JavaScript ขนาดเล็กที่ส่งเสริมการทำงานร่วมกับ Jest ทำให้สามารถจำลองการทำงานของฟังก์ชันตามอาร์กิวเมนต์ที่ส่งเข้ามา Jest เป็นเครื่องมือทดสอบที่ยอดเยี่ยมอยู่แล้ว jest-when ยิ่งทำให้การทดสอบโมดูลที่พึ่งพิงอาร์กิวเมนต์หลายตัวมีความทนทานขึ้นไปอีก เพราะจะทดสอบได้ว่า อาร์กิวเมนต์ใดของฟังก์ชันจำลองนั้นถูกส่งค่าตรงกับค่าที่คาดหวังไว้หรือไม่ ทีมของเราจะเลือกใช้ jest-when ก่อนเสมอหากต้องการจะจำลองฟังก์ชันใด ๆ เพราะมันใช้งานง่ายและรองรับการเปรียบเทียบค่าได้หลากหลาย



Fastify
ทดลอง

เมื่อใดที่การใช้ Node.js เป็นเรื่องเหมาะสม เราเห็นว่า Fastify เป็นทางเลือกหนึ่งที่ทีมของเราใช้แล้วปลื้มมาก เฟรมเวิร์คตัวนี้มีความสามารถหลากหลาย ช่วยจัดการเรื่องการตรวจสอบความถูกต้องของรีเควสต์และเรสปอนส์ รองรับภาษา TypeScript แล้วก็มีระบบนิเวศของส่วนต่อขยายต่าง ๆ ที่สมบูรณ์ ช่วยให้ทีมของเราทำงานได้ง่ายขึ้นจริงอยู่ ที่มันเป็นทางเลือกที่ดีในระบบนิเวศของ Node.js แต่เรายังยืนยันคำแนะนำเดิมที่ให้ไปว่า อย่าเผลอใช้ Node อย่างพร่ำเพรื่อ

Immer
ทดลอง

เมื่อความซับซ้อนของเว็บแอปพลิเคชันเพิ่มสูงขึ้น การจัดการข้อมูลสถานะในระบบให้เข้าใจง่ายกลายเป็นสิ่งสำคัญกว่าเดิม ซึ่งการเขียนโปรแกรมโดยไม่เปลี่ยนแปลงค่าตัวแปร (immutability) เป็นเทคนิคที่สามารถช่วยให้แอปพลิเคชันของเรามีพฤติกรรมที่วางใจได้ แต่น่าเสียดายที่ภาษา JavaScript กลับไม่มีโครงสร้างภาษาที่รองรับการเขียนโปรแกรมในลักษณะนี้มาให้ในตัว (ดูเพิ่มเติมได้ที่ ES Record and Tuple proposal) Immer — ในภาษาเยอรมัน

นำไปใช้
76. Arrow
77. jest-when

ทดลอง
78. Fastify
79. Immer
80. Redux
81. Rust
82. single-spa
83. Strikt
84. XState

ประเมิน
85. Babylon.js
86. Blazor
87. Flutter Driver
88. HashiCorp Sentinel
89. Hermes
90. io-ts
91. Kedro
92. LitElement
93. Mock Service Worker
94. Recoil
95. Snorkel
96. Streamlit
97. Svelte
98. SWR
99. Testing Library

เฝ้าระวัง

ภาษา & เฟรมเวิร์ค

เราไม่ได้คิดว่า Redux เป็นระบบบริหารจัดการสถานะข้อมูล (state management) ทางเลือกแรกสำหรับ React อีกต่อไป มันเป็นเฟรมเวิร์คที่มีประโยชน์ แต่เมื่อเทียบกับแนวทางอื่น ๆ มันมักจะสร้างโค้ดที่ฟุ่มเฟือยกว่า และยากต่อการเข้าใจ

(Redux)

XState เป็นเฟรมเวิร์คที่เรียบง่ายสำหรับสร้างเครื่องสถานะจำกัด (finite state machine) ด้วยภาษา JavaScript และ TypeScript

(XState)

แปลว่า *เสมอ* — มันเป็นไลบรารีขนาดเล็กที่ช่วยให้เราเขียนโปรแกรมแบบไม่เปลี่ยนสถานะได้สะดวกยิ่งขึ้น มันอาศัยเทคนิคการทำงานแบบ copy-on-write และมี API ที่ไม่ซับซ้อน ทำงานร่วมกับออบเจกต์และอาร์เรย์พื้นฐานของ JavaScript ได้โดยตรง ซึ่งทำให้การเริ่มต้นใช้กับโครงการที่มีอยู่แล้ว ทำได้สะดวกไม่ต้องปรับโค้ดแบบยกใหญ่ หลายทีมของเราที่ได้ใช้มันแล้ว ชอบมันมากกว่า Immutable.js เสียอีก ซึ่งนี่เป็นเหตุผลที่เราขยับมันเข้ามาอยู่ในหมวดให้ทดลองใช้งาน

Redux

ทดลอง

เราตัดสินใจย้าย Redux กลับมาที่หมวดหมู่ “ทดลองใช้งาน” อีกครั้ง เพื่อสะท้อนความเห็นของเราว่า Redux ไม่ใช่ระบบบริหารจัดการสถานะข้อมูล (state management) ทางเลือกแรกสำหรับ React อีกต่อไป จากประสบการณ์ที่ผ่านมา Redux ยังคงเป็นเฟรมเวิร์คที่มีข้อดีสำหรับหลายกรณี แต่เมื่อเทียบกับแนวทางอื่น ๆ มันมักจะสร้างโค้ดที่ฟุ่มเฟือยกว่า และยากต่อการเข้าใจ ยิ่งถ้าเอา Redux Sagas เข้าไปผสมโรงด้วย ยิ่งจะไปกันใหญ่กว่าเดิม บ่อยครั้งที่วิธีการบริหารจัดการสถานะข้อมูลใน React เวอร์ชันหลัง ๆ ก็ให้ประสิทธิภาพเพียงพอแล้ว ไม่ต้องเสริมด้วยเฟรมเวิร์คอื่นเลย อย่างไรก็ดี เราอยากจะเน้นว่า เมื่อไหร่ที่การใช้วิธีจัดการสถานะข้อมูลแบบพื้นฐานเริ่มซับซ้อนขึ้น เป็นจังหวะที่ดีให้พิจารณาใช้ Redux หรือจะลอง Recoil ที่ Facebook เพิ่งประกาศออกมาดูก็ได้

Rust

ทดลอง

Rust เป็นภาษาที่กำลังเติบโตอย่างต่อเนื่องและเป็นที่นิยมมากขึ้นเรื่อย ๆ โดยมันถูกโหวตในเว็บไซต์ Stack Overflow ให้เป็นภาษา “ที่ถูกรักมากที่สุด” ของนักพัฒนาตลอด 5 ปีที่ผ่านมา เราเองก็ชอบมันเช่นกัน มันเป็นภาษาที่ทั้งเร็ว ปลอดภัย และเล่าเรื่องได้สวยงาม เราพบว่ายิ่งระบบนิเวศของมันโตขึ้นเท่าไหร่ ก็ยิ่งมีประโยชน์มากขึ้นเท่านั้น ดังที่เราเห็น Rust เริ่มถูกใช้ในงานวิทยาศาสตร์ข้อมูล และแมชชีนเลิร์นนิง เพราะมันสามารถ เพิ่มประสิทธิภาพความเร็วได้อย่างมีนัยสำคัญ รวมไปถึง Materialize ซึ่งเป็นระบบฐานข้อมูลแบบสตรีมมิ่งความหน่วงต่ำ ก็ถูกพัฒนาด้วย Rust เช่นกัน

single-spa

ทดลอง

single-spa เป็นเฟรมเวิร์ค JavaScript ที่ใช้สำหรับผสมไมโครฟรอนท์เอนด์ หลายตัวเข้าด้วยกันเป็นแอปพลิเคชันฟรอนท์เอนด์เดียวกัน แม้เราจะไม่แนะนำหลักการ ไมโครฟรอนท์เอนด์เพื่ออนาธิปไตย หรือการใช้ไมโครฟรอนท์เอนด์เป็นข้ออ้างในการผสมเฟรมเวิร์คต่าง ๆ เข้าด้วยกัน แต่ single-spa ก็รองรับความสามารถนี้ ซึ่งเราก็ยอมรับว่ามีกรณีอันสมควรให้ใช้อยู่บ้าง เช่น กรณีการอัปเกรดเวอร์ชันของเฟรมเวิร์คข้ามไมโครฟรอนท์เอนด์หลายตัวจำเป็นต้องเชื่อมเฟรมเวิร์คต่าง ๆ เข้าด้วยกัน singsle-spa ได้กลายเป็นเฟรมเวิร์คโปรดสำหรับพวกเรา หากจะเชื่อมไมโครฟรอนท์เอนด์เข้าด้วยกัน เราพบว่ามันทำงานเข้ากันดีกับ SystemJS โดยเฉพาะกรณีที่มีการพึ่งพิงไลบรารีตัวเดียวกัน แต่ต่างเวอร์ชันกัน

Strikt

ทดลอง

ระบบนิเวศรอบ ๆ ตัวภาษา Kotlin ยังคงเติบโตอย่างต่อเนื่อง มีไลบรารีหลายตัวที่หันมาใช้ประโยชน์จากคุณสมบัติของภาษา Kotlin เพื่อแข่งขันกับไลบรารีชนิดเดียวกันที่เขียนด้วยภาษา Java เช่นเดียวกับ Strikt ที่เป็นไลบรารีสำหรับการยืนยันความถูกต้อง (Assertion) ของโปรแกรม มันทำงานโดยอาศัยคุณสมบัติเรื่องบล็อคและแลมด้าฟังก์ชันในภาษา Kotlin เพื่อช่วยให้เราสามารถเขียนการทดสอบด้วยภาษาที่อ่านแล้วกระชับ ลื่นไหล นอกจากนี้เรายังสามารถสร้างกฎการยืนยันความถูกต้องขึ้นมาได้เอง ทำให้เราสามารถเขียนการทดสอบของเราด้วยภาษาที่อ่านเข้าใจง่าย ตรงกับโดเมนนั้น ๆ มากยิ่งขึ้น

XState

ทดลอง

ในเรดาร์ฉบับก่อน ๆ เราได้แนะนำไลบรารีสำหรับบริหารจัดการสถานะข้อมูล (state management) ไปหลายตัว แต่ XState มีแนวคิดบางอย่างที่น่าสนใจ ต่างจากตัวอื่น มันเป็นเฟรมเวิร์คที่เรียบง่ายสำหรับสร้างเครื่องสถานะจำกัด (finite state machine) ด้วยภาษา JavaScript และ TypeScript โดยรองรับมาตรฐานเครื่องสถานะจำกัดของ W3C และสามารถแสดงผลค่าสถานะต่าง ๆ ในรูปแบบแผนภูมิได้อีกด้วย

โดยเราสามารถใช้มันร่วมกับเฟรมเวิร์คยอดนิยม เช่น Vue.js, Ember.js, React.js และ RxJS หรือแปลงโมเดลของมันไปอยู่ในรูปแบบฟอร์แมตต่าง ๆ ได้ จากประสบการณ์การใช้เครื่องจำกัดสถานะในบริบทอื่น (โดยเฉพาะในบริบทการเขียนตรรกะให้เกมส์) เราพบว่าเป็นประโยชน์มาก หากสามารถแสดงค่าสถานะและความสัมพันธ์ของการเปลี่ยนแปลงระหว่างสถานะออกมาได้ ซึ่ง XState ก็ทำเช่นนั้นได้ ผ่านเครื่องมือ visualizer ที่มีมาให้

Babylon.js

ประเมิน

เมื่อหลายปีก่อน เราได้เขียนบทความเกี่ยวกับเทคโนโลยีเสมือนจริง (VR) กับเรื่องอื่น ๆ นอกจากเกมส์ ในบทความนั้นเราไม่ได้ทำนายว่าเทคโนโลยีนี้จะเป็นที่นิยมแค่ไหน รวดเร็วเพียงใด หรือไปได้ไกลแค่ไหนในตลาดอื่น ๆ นอกจากเกมส์ หากสะท้อนดูวันนี้ เราเห็นชัดว่าเทคโนโลยีนี้ได้รับความสนใจและถูกใช้มากขึ้น แต่การตอบรับก็ยังช้ากว่าที่บางคนของเราหวังเอาไว้ หากมองไปที่เอนจิ้นสำหรับใช้พัฒนาแอปพลิเคชัน VR ทั้ง Unity และ Unreal ต่างก็เป็นเครื่องมือที่มีความสมบูรณ์และเป็นที่ยอมรับในความสามารถ นอกจากนั้นยังมีเครื่องมืออื่น ๆ ที่เราเคยพูดถึง เช่น Godot ด้วย อย่างไรก็ดี ความลำบากอยู่ที่เครื่องมือเหล่านี้ เป็นสิ่งที่นักพัฒนาและทีมในองค์กรส่วนใหญ่ไม่คุ้นชิน พวกเราจึงสำรวจไปไกลขึ้น แล้วพบว่าเทคโนโลยีเสมือนจริงสำหรับเว็บนั้นมีพัฒนาการมาไกลแล้วเช่นกัน ซึ่งเรารู้สึกดีกับ Babylon.js มันถูกเขียนด้วยภาษา TypeScript และสามารถเรนเดอร์ตัวแอปพลิเคชันบนบราวเซอร์ได้ ซึ่งมันใช้วิธีการพัฒนาไม่ต่างจากวิธีที่นักพัฒนาส่วนใหญ่คุ้นเคยกันดี นอกจากนั้นแล้ว Babylon.js ยังเป็นซอฟต์แวร์โอเพนซอร์สที่มีความพร้อมสูง ได้รับการสนับสนุนทางการเงินเป็นอย่างดี จึงทำให้มันยิ่งน่าสนใจเข้าไปอีก

Blazor
ประเมิน

แม้ทุกวันนี้โลกการพัฒนาเว็บ UI ยังคงเป็น JavaScript และระบบนิเวศรอบตัวมันที่ครองอิทธิพลสูงสุดอยู่ก็ตาม แต่การเติบโตของเทคโนโลยี WebAssembly ก็ทำให้มีนวัตกรรมใหม่ ๆ ปรากฏขึ้นมาบ้างแล้วเช่นกัน หนึ่งในนั้นคือ Blazor มันเป็นเครื่องมือทางเลือกสำหรับสร้างเว็บ UI ด้วยภาษา C# ที่น่าสนใจ เราชื่นชอบโอเพนซอร์สเฟรมเวิร์คตัวนี้เป็นพิเศษ ที่สามารถแปลงโค้ดภาษา C# ไปทำงานบนเบราว์เซอร์ผ่านเทคโนโลยี WebAssembly ได้ ดังนั้นเราสามารถใช้ประโยชน์จาก .NET Runtime และระบบนิเวศรอบตัว รวมถึงไลบรารีต่าง ๆ ที่มีในภาษานี้ได้ทั้งหมด นอกจากนี้ Blazor สามารถสื่อสารไปมาระหว่างโค้ด JavaScript ที่ทำงานอยู่บนเบราว์เซอร์ได้อีกด้วย

Flutter Driver
ประเมิน

Flutter Driver เป็นไลบรารีสำหรับทดสอบแอปพลิเคชัน Flutter ในระดับการเชื่อมต่อ (integration test) เมื่อใช้ Flutter Driver คุณสามารถทำการทดสอบได้ทั้งบนอุปกรณ์จริง หรือใช้กับโปรแกรมจำลองอุปกรณ์ (emulator) ก็ได้ ทีมของเราสร้างชุดทดสอบระดับยูนิต (unit test) และชุดทดสอบระดับวิดเจ็ต (widget test) ไว้ก็เพื่อให้มั่นใจว่าแอปพลิเคชันที่กำลังพัฒนา สามารถทำงานตามฟังค์ชันทางธุรกิจได้ถูกต้อง เราจึงกำลังประเมินว่า Flutter Driver เหมาะแก่การเป็นเครื่องมือไว้ทดสอบการโต้ตอบของผู้ใช้ (user interaction) ด้วยหรือไม่ ซึ่งเราก็แนะนำให้คุณประเมินด้วยเช่นกัน

HashiCorp Sentinel
ประเมิน

ถึงแม้เราจะสนับสนุนการกำหนด นโยบายความมั่นคงด้วยโค้ด มาตลอด แต่หากว่ากันตามตรง เครื่องมือในกลุ่มนี้มีจำนวนจำกัดมาก หากคุณใช้ผลิตภัณฑ์ของ HashiCorp อยู่แล้ว เช่น Terraform หรือ Vault และไม่ติดใจที่จะซื้อไลเซนส์ระดับองค์กรมาใช้ คุณจะมี HashiCorp Sentinel เป็นอีกทางเลือกหนึ่ง ซึ่ง Sentinel เป็นภาษาโปรแกรมมิ่งสำหรับกำหนดและติดตั้ง นโยบายการตัดสินใจตามบริบท (Context-based policy decission) เช่น เราสามารถใช้ร่วมกับ Terraform เพื่อตรวจสอบว่ามีนโยบายใดที่กำลังถูกละเมิดอยู่หรือไม่ ก่อนจะเปลี่ยนแปลงโครงสร้างพื้นฐานใด ๆ หรือใช้ร่วมกับ Vault โดย Sentinel จะทำหน้าที่กำหนดสิทธิ์การเข้าถึงอย่างละเอียดให้กับ API ต่าง ๆ ซึ่งเราสามารถใช้ประโยชน์จากความสามารถของภาษาโปรแกรมมิ่งระดับสูง ทั้งเรื่อง การซ่อนรายละเอียดภายใน (Encapsulation) ความสามารถในการดูแลรักษา (Maintainability) ความง่ายในการเข้าใจ (Readability) ความสามารถในการต่อยอด (Extendsibility) ซึ่งวิธีการกำหนดเช่นนี้ เป็นทางเลือกที่น่าสนใจเมื่อเปรียบเทียบกับวิธีการประกาศนโยบายความมั่นคงแบบเดิม ๆ Sentinel จัดอยู่ในกลุ่มเครื่องมือประเภทเดียวกันกับ Open Policy Agent แต่เป็นเทคโนโลยีปิด ไม่เป็นโอเพนซอร์ส และใช้ได้กับผลิตภัณฑ์ของ HashiCorp เท่านั้น

Hermes
ประเมิน

Hermes เป็น JavaScript เอนจิ้นที่ถูกออกแบบพิเศษให้เริ่มโปรแกรมเร็วขึ้น สำหรับแอปพลิเคชัน React Native บนแพลตฟอร์ม Android โดยเฉพาะ เอนจิ้นอื่น ๆ เช่น V8 ใช้เทคนิคการคอมไพล์แบบทันเวลาพอดี (just-in-time, JIT) ซึ่งตอนรันไทม์เอนจิ้นจะวิเคราะห์โค้ด ก่อนจะสร้างเป็นชุดคำสั่งที่ให้ประสิทธิภาพสูงสุดออกมา แต่ Hermes ใช้เทคนิคที่ต่างไป โดยมันจะแปลงโค้ด JavaScript ให้เป็นไบต์โค้ดล่วงหน้า ก่อนที่แอปพลิเคชันจะเริ่มโปรแกรม (ahead-of-time, AOT) ด้วยซ้ำ ผลคือเราจะได้ไฟล์ APK ที่มีขนาดเล็กลง มีการใช้หน่วยความจำน้อยลง และเริ่มโปรแกรมได้เร็วขึ้น เราได้ทดสอบแอปพลิเคชันที่ใช้ Hermes อย่างระมัดระวังก่อนจะใช้งานมันอย่างจริงจัง ซึ่งเราก็แนะนำให้คุณทำเช่นนั้นเหมือนกัน

io-ts
ประเมิน

เราเพลิดเพลินกับการใช้ TypeScript มาตลอด และหลงใหลความปลอดภัยที่ได้จากภาษาที่ช่วยตรวจสอบเรื่องไทป์ (strong type) อย่างไรก็ดี ในกรณีที่ข้อมูลมาจากภายนอกกรอบที่ระบบไทป์คุ้มกันไว้อยู่ (เช่น ข้อมูลได้มาจากเซอร์วิสแบ็คเอนด์ ณ ตอนรันไทม์) ก็อาจเกิดข้อผิดพลาดว่าข้อมูลไม่ตรงกับไทป์ได้อยู่ดี หนึ่งในไลบรารีที่ช่วยแก้ปัญหานี้คือ io-ts ซึ่งช่วยปิดข้อจำกัด ที่การตรวจไทป์ทำได้เฉพาะตอนคอมไพล์ กับข้อมูลที่รับเข้ามาจากภายนอกตอนรันไทม์ ด้วยการเสนอฟังก์ชัน encode และ decode เข้ามา นอกจากนี้ เราสามารถใช้มันเป็นการ์ดป้องกันไทป์ (type guard) ที่เราสร้างขึ้นเฉพาะได้ด้วย โดยทีมของเราเห็นว่า การใช้ io-ts เป็นวิธีการแก้ปัญหาที่งดงาม ทำได้ตรงจุดพอดี

Kedro
ประเมิน

ปัจจุบันมีเครื่องมือที่พร้อมขึ้น สำหรับประยุกต์ใช้แนวปฏิบัติทางวิศวกรรมที่ดี กับงานวิทยาศาสตร์ข้อมูลแล้ว Kedro เป็นเครื่องมือที่ดีอีกตัวในกลุ่มนี้ โดยมันเป็นเฟรมเวิร์คสร้างขั้นตอนงานสำหรับพัฒนางานวิทยาศาสตร์ข้อมูล ช่วยกำหนดวิธีการมาตรฐานในการสร้างไปป์ไลน์ข้อมูลและแมชชีนเลิร์นนิง ที่พร้อมใช้กับงานโปรดักชัน พวกเราชอบที่ Kedro ให้ความสำคัญกับการใช้แนวปฏิบัติทางวิศวกรรมและการออกแบบที่ดี ด้วยการส่งเสริมให้เราทำการทดสอบก่อนลงมือทำ (TDD) แบ่งโค้ดออกเป็นส่วน ๆ มีวิธีควบคุมเวอร์ชัน และการมีวินัย ไม่เก็บข้อมูลความลับไว้ในโค้ด

LitElement
ประเมิน

เมื่อปี 2014 เราเคยพูดถึง Web Components ไป จากนั้นเรื่อยมามันก็มีพัฒนาการขึ้นมาอย่างช้า ๆ LitElement ซึ่ง

ภาษา & เฟรมเวิร์ค

Sentinel เป็นภาษาโปรแกรมมิ่งสำหรับกำหนดและติดตั้ง นโยบายการตัดสินใจตามบริบท (Context-based policy decission)

(HashiCorp Sentinel)

Kedro เป็นเฟรมเวิร์คสร้างขั้นตอนงานสำหรับพัฒนางานวิทยาศาสตร์ข้อมูล ช่วยกำหนดวิธีการมาตรฐานในการสร้างไปป์ไลน์ข้อมูลและแมชชีนเลิร์นนิง ที่พร้อมใช้กับงานโปรดักชัน

(Kedro)

ภาษา & เฟรมเวิร์ค

Snorkel ทำให้เราสามารถเขียนโปรแกรมจำแนกข้อมูลขนาดใหญ่สำหรับนำไปใช้ฝึกสอนโมเดลแมชชีนเลิร์น โดยมันถูกสร้างขึ้นที่มหาวิทยาลัย Stanford

(Snorkel)

เป็นส่วนหนึ่งของ Polymer Project เป็นไลบรารีอันเรียบง่าย ที่เอาไว้สร้างเว็บคอมโพเนนท์ขนาดเล็ก อันที่จริงมันเป็นเพียงเบสคลาสที่ภายในจัดการเรื่องทั่วไปให้หมดแล้ว ทำให้เวลาจะสร้างเว็บคอมโพเนนท์ขึ้นมาสักตัว สามารถทำได้ง่ายมาก เราเคยใช้ LitElement ในโครงการของเราตอนที่มันออกมาได้ใหม่ ๆ และได้ผลดี มาตอนนี้เราก็ตื่นเต้นที่ได้เห็นเทคโนโลยีนี้มีความพร้อมมากขึ้น

Mock Service Worker

ประเมิน

เว็บแอปพลิเคชันต่าง ๆ โดยเฉพาะที่ใช้กันภายในองค์กร มักจะถูกแบ่งออกเป็นสองส่วนด้วยกัน คือ ส่วนแสดงผลที่ทำงานบนเบราว์เซอร์ ที่อาจมีตรรกะทางธุรกิจ (business logic) รวมอยู่ด้วยบ้าง และส่วนที่ทำงานบนเซิร์ฟเวอร์ อันเป็นส่วนที่จัดการเรื่องตรรกะทางธุรกิจหลัก ดูแลสิทธิ์การเข้าถึงข้อมูล และจัดการเรื่องการบันทึกข้อมูลต่าง ๆ ทั้งสองส่วนมักสื่อสารกันด้วย JSON ผ่านทาง HTTP ส่วนใหญ่เอนด์พอยต์เหล่านี้เป็นเพียงรายละเอียดทางเทคนิค ไม่ควรสับสนว่าเป็น API จริง ๆ อย่างไรก็ดี รอยต่อนี้เป็นตัวเลือกที่ดี หากนำมาใช้แยกการทดสอบของทั้งสองระบบให้อิสระจากกัน กล่าวคือ หากจะทดสอบโค้ดที่ทำงานบนฝั่งเบราว์เซอร์ เราสามารถจำลองการทำงานของฝั่งเซิร์ฟเวอร์ขึ้นมาได้ โดยตัวจำลองจะทำงานที่ระดับเน็ตเวิร์ค ผ่านเครื่องมืออย่าง Mountebank หรือจะจำลองการทำงานของเซิร์ฟเวอร์ ด้วยการดักจับรีเควสต์ที่กำลังจะถูกส่งออกจากเบราว์เซอร์ไป เราชอบวิธีที่ Mock Service Worker เลือกใช้ โดยทำการการจำลองด้วย Service Worker ที่นักพัฒนาคุ้นเคยกันดี วิธีการนี้ส่งผลให้การตั้งค่าการทดสอบทำได้ง่าย และทดสอบได้เร็วขึ้น อย่างไรก็ตามเนื่องจากการทดสอบเหล่านี้ไม่ได้ทดสอบที่ระดับเน็ตเวิร์คจริง จึงต้องมีการทดสอบแบบทั้งระบบ (end-to-end) อีกชั้น เพื่อให้การทดสอบครอบคลุมทุกส่วน ตามหลักการทดสอบเป็นรูปพีระมิดที่ดี

Recoil

ประเมิน

มีหลายต่อหลายทีมกำลังประเมินกันใหม่ว่าควรจะใช้ระบบบริหารจัดการสถานะข้อมูล (state management) ใดกันดีสำหรับ React ซึ่งเป็นสิ่งที่เราแนะนำให้ทำเช่นกันในงานประเมิน Redux ครั้งใหม่ของเรา ไม่นานนี้ Facebook ผู้พัฒนา React ได้เปิดตัว Recoil ออกมา ซึ่งเป็นไลบรารีตัวใหม่สำหรับบริหารจัดการข้อมูลสถานะ ที่มีต้นกำเนิดมาจากแอปพลิเคชันภายในที่ต้องทำงานร่วมกับข้อมูลจำนวนมาก แม้เรายังไม่ผ่านการใช้งาน Recoil มากนัก แต่เราเล็งเห็นถึงศักยภาพและแนวโน้มที่ดีของมัน มันมี API ที่ไม่ซับซ้อนและเรียนรู้ได้ง่าย ใช้งานเข้ากับ React ได้อย่างเป็นธรรมชาติ และมีเทคนิคการแชร์สถานะข้อมูลร่วมกันภายในแอปพลิเคชันที่มีประสิทธิภาพและยืดหยุ่น แตกต่างจากวิธีอื่น โดยรองรับการสร้างข้อมูลสถานะขึ้นมาใหม่แบบไดนามิก จากการสืบทอดค่าข้อมูลหรือคิวรีของอีกสถานะหนึ่ง รองรับการเฝ้าติดตามสถานะข้อมูลที่เกิดขึ้นภายในแอปพลิเคชัน โดยไม่เสียความสามารถการแบ่งโค้ดเป็นส่วนย่อย เพื่อโหลดขึ้นส่วนแยกกัน

Snorkel

ประเมิน

แบบจำลอง ML สมัยใหม่มีความซับซ้อนมาก และยังต้องการกลุ่มข้อมูลที่จำแนกแล้ว (labeled training data set) จำนวนมากเพื่อจะใช้ในการฝึกสอน Snorkel เริ่มต้นมาจากแล็บ AI ของ Stanford โดยตระหนักดีว่า การจำแนกข้อมูลด้วยมือนั้นใช้ทรัพยากรสูง และบ่อยครั้งก็ไม่สามารถทำได้ Snorkel ทำให้เราสามารถจำแนกข้อมูลสำหรับฝึกสอนอย่างเป็นระบบด้วยการเขียนโปรแกรมสร้างเป็นฟังก์ชันการจำแนกข้อมูล Snorkel ใช้เทคนิคการเรียนรู้แบบมีผู้สอน เพื่อประเมินความแม่นยำและความเชื่อมโยงกันระหว่างฟังก์ชันการจำแนกข้อมูล จากนั้นก็ทำการปรับน้ำหนักใหม่ และรวบรวมผลการจำแนกจากฟังก์ชันเหล่านั้น เพื่อใช้เป็นข้อมูลเพื่อการฝึกสอนที่ถูกจำแนกอย่างดี ผู้สร้าง Snorkel ได้ออกแพลตฟอร์มเชิงพาณิชย์มา ชื่อว่า Snorkel Flow แม้ว่า Snorkel จะไม่ได้ถูกพัฒนาอย่างแข็งขันเช่นเดิมแล้ว แต่แนวคิดการจำแนกข้อมูลด้วยการสอนอย่างอ่อน (weakly supervised) ของมันก็ยังมีความสำคัญ

Streamlit

ประเมิน

Streamlit เป็นโอเพนซอร์สเฟรมเวิร์คสำหรับสร้างแอปพลิเคชันด้วยภาษา Python ที่นักวิทยาศาสตร์ข้อมูลชอบใช้กัน เพื่อแสดงแผนภาพข้อมูลออกเป็นเว็บที่น่าใช้ Streamlit โดดเด่นกว่าคู่แข่งอย่าง Dash ตรงที่มันเน้นให้สร้างแอปพลิเคชันต้นแบบได้ไว รองรับไลบรารีการแสดงภาพที่หลากหลาย เช่น Plotly และ Bokeh มันเป็นทางเลือกที่ดีมากสำหรับนักวิทยาศาสตร์ข้อมูลคนใดที่ต้องการการนำเสนอผลงานอย่างรวดเร็ว ในระหว่างรอบการทดลอง จากการใช้งานมันในบางโครงการ เราประทับใจที่สามารถสร้างแผนภาพข้อมูลเชิงโต้ตอบออกมาได้ในระยะเวลาอันสั้น

Svelte

ประเมิน

ท่ามกลางเฟรมเวิร์ค JavaScript สำหรับงานด้านฟรอนต์เอนด์ที่ออกมาใหม่อยู่เสมอ Svelte ถือเป็นคอมโพเนนท์เฟรมเวิร์คตัวหนึ่งที่โดดเด่นและน่าจับตามอง เฟรมเวิร์คอื่นมักจะใช้ประโยชน์จากเวอร์ชวลดอม (Virtual DOM) แต่ Svelt ไม่ใช้ แล้วกลับใช้เทคนิคการคอมไพล์โค้ดแทน โดยโค้ดที่ได้จะเป็น JavaScript ธรรมดาที่ไม่มีโค้ดเฟรมเวิร์คใด ๆ ผสมเลย ซึ่งมันจะเอาโค้ดชุดนี้ไปทำงานร่วมกับดอม (DOM) โดยตรง อย่างไรก็ตาม Svelte เป็นเพียงคอมโพเนนท์เฟรมเวิร์คเท่านั้น หากคุณมีแผนที่จะสร้างแอปพลิเคชันที่มีความสามารถซับซ้อน เราขอแนะนำให้คุณพิจารณาใช้ Sapper ร่วมกับ Svelte ด้วย

SWR

ประเมิน

SWR เป็นไลบรารีที่ทำงานร่วมกับ React Hooks สำหรับใช้ดึงข้อมูลจากภายนอก ภายใน SWR รองรับวิธีการแคชข้อมูลแบบ SWR อันเป็นกลยุทธการแคชหนึ่งในมาตรฐาน HTTP อันเป็นเทคนิคการใช้ข้อมูลจากแคชก่อนเสมอแม้ข้อมูลจะไม่ล่าสุดแล้ว (stale) ขณะเดียวกันก็วิ่งไปดึงข้อมูลจากระบบภายนอก (revalidate) เมื่อไหร่ที่ข้อมูลกลับมาจึงตามไปอัปเดตค่าในแคชตามหลัง ซึ่งคอมโพเนนท์ที่มาเรียกใช้งาน SWR จะได้รับข้อมูลสองชุดภายในหนึ่งสตรีม ชุดแรกเป็นข้อมูลจากแคช หลังจากนั้นจะได้รับข้อมูลล่าสุด ซึ่งทั้งหมดนี้เกิดขึ้นทันทีทันใดและเป็นอัตโนมัติ นักพัฒนาของเราประทับใจการใช้งาน SWR จากที่มันเปลี่ยนประสบการณ์ของผู้ใช้ไปอย่างมาก เพราะข้อมูลขึ้นมาเร็วและแสดงบนหน้าจออยู่เสมอ อย่างไรก็ดี เราขอเตือนการใช้กลยุทธการแคชแบบนี้เฉพาะกับข้อมูลที่เหมาะสม ที่การเห็นข้อมูลเก่าชั่วขณะเป็นสิ่งที่ยอมรับได้ หมายเหตุไว้ด้วยว่ามาตรฐาน HTTP กำหนดไว้ว่า แคชสามารถกระทำได้แต่ต้องแคชข้อมูลเรสปอนด์ล่าสุดของรีเควสต์นั้น ๆ เท่านั้น และเฉพาะกรณีพิเศษที่พิจารณาอย่างรอบคอบแล้วเท่านั้น ถึงจะอนุญาตให้ใช้ข้อมูลจากแคชที่เก่ากว่านั้นได้

Testing Library

ประเมิน

Testing Library เป็นชุดไลบรารีสำหรับทดสอบแอปพลิเคชันที่พัฒนาด้วยสารพันเฟรมเวิร์ค เช่น React, Vue, React Native, Angular ฯลฯ มันช่วยให้เรามีมุมมองการทดสอบคอมโพเนนท์ใด ๆ เสมือนว่าเป็นผู้ใช้งานเอง โดยสนับสนุนให้เราทดสอบพฤติกรรมที่แสดงออก มากกว่าทดสอบวิธีการภายในที่เราทราบ เช่น การทดสอบโดยคาดหวังว่า จะมีอิลิเมนต์ของ UI ใดปรากฏขึ้น เมื่อเกิดเหตุการณ์บางอย่าง ข้อดีหนึ่งของวิธีการนี้คือ เราจะได้ชุดทดสอบที่น่าเชื่อถือ ไม่พังง่าย ซึ่งคุณสมบัตินี้ทำให้ Testing Library พิเศษกว่าไลบรารีอื่น ถึงแม้พวกเราจะมีประสบการณ์ตรงแค่กับ React Testing Library และ Angular Testing Library เท่านั้น แต่ก็เป็นประสบการณ์ที่น่าประทับใจเลยทีเดียว

ThoughtWorks®

เราคือบริษัทให้คำปรึกษาและวางแผนทางด้านเทคโนโลยีระดับโลก เป็นชุมชนที่รวมปัจเจกชนผู้หลงใหลในการทำในสิ่งที่เราเชื่อมากกว่า 7,000 คน จากสำนักงาน 43 แห่งที่กระจายอยู่ 14 ประเทศทั่วโลก ด้วยประสบการณ์มากกว่า 25 ปีที่เราใช้เทคโนโลยีสร้างความแตกต่างและช่วยแก้ปัญหาทางธุรกิจที่ซับซ้อนให้กับลูกค้า เมื่อการเปลี่ยนแปลงคือความแน่นอน เราเตรียมคุณให้พร้อมรับกับสิ่งไม่คาดฝัน

อยากติดตามข่าวสารและข้อมูลเชิงลึกล่าสุดเกี่ยวกับเรดาร์ใช่ไหม?

ติดตามเราบนช่องทางโซเชียลที่คุณถนัด หรือสมัครรับข่าวสารทางอีเมล์จากเรา

สมัครรับข่าวสารทันที!

ThoughtWorks®

thoughtworks.com/radar

#TWTechRadar