ThoughtWorks®

TECHNOLOGY RADAR

คู่มือนำทางสู่ปลายขอบเทคโนโลยี ตามแบบฉบับของเรา

Vol.22

thoughtworks.com/radar

#TWTechRadar

ผู้ร่วมสร้างสรรค์

เรดาร์เทคโนโลยีถูกจัดทำขึ้นโดยคณะกรรมการที่ปรึกษาด้านเทคโนโลยี

คณะกรรมการที่ปรึกษาด้านเทคโนโลยีประกอบด้วยผู้นำและนักเทคโนโลยีผู้มากประสบการณ์ที่มากกว่า 20 คนของ ThoughtWorks พวกเขารวมตัวประจำแบบเจอหน้ากัน 2 ครั้งต่อปี และแบบทางไกลทุกๆ 2 สัปดาห์ โดยมีหน้าที่สำคัญคือให้คำแนะนำ และเป็นที่ปรึกษาให้ประธานเจ้าหน้าที่บริหารฝ่ายเทคโนโลยี Rebecca Parsons

คณะกรรมการที่ปรึกษาจะทำหน้าที่เป็นผู้พิจารณาเทคโนโลยีที่กำลังมีบทบาทสำคัญในอุตสาหกรรมโลกขณะนั้น และมีผลกระทบต่อนักเทคโนโลยีของ ThoughtWorks เพื่อกำหนดแผนยุทธศาสตร์ภายใน ตามปกติแล้วเราจัดทำ Techonology Radar จากการรวมตัวในสถานที่เดียวกัน แต่เนื่องจากสถานการณ์โรคระบาดที่โลกกำลังประสบอยู่ คู่มือฉบับนี้จึงเกิดจากการประชุมแบบเสมือนจริง


Rebecca Parsons (CTO)


Martin Fowler (Chief Scientist)


Bharani Subramaniam


Birgitta Böckeler


Camilla Crispim


Erik Dörnenburg


Evan Bottcher


Fausto de la Torre


Hao Xu


Ian Cartwright


James Lewis


Jonny LeRoy


Lakshminarasimhan Sudarshan


Mike Mason


Neal Ford


Ni Wang


Rachel Laycock


Scott Shaw


Shangqi Liu


Zhamak Dehghani

ทีมแปลภาษาไทย: คณิศร สุธรรม, ณัฐพงศ์ อินทรักษ์, ดิษทัต เพิ่มพูนวิวัฒน์, ธนกฤต จูฑะมงคล, ธนพร เบญจพลกุล, ปัทมา ทวนชัยศรี, พชร ตันทนิส, พิชญุตม์ จิตรศิลป์ฉายากุล, พิรดา ทิพย์รัตน์, พิสินี สกุลวานิชพร, พีรพัฒน์ จั่นขาว, ภควัต อเนกวิโรจน์, วณิชนันท์ สินพิทักษ์, วรุณ เกียรติดุริยกุล, ศรัทธา เชาว์เจริญรัตน์, ศอลาฮุดดีน เฉลิมไทย, ศิโรรัตน์ สุนทรสุข, อนุชิต ประเสริฐสังข์, อพินยา โพธิพันธ์, Naren Katakam

เกี่ยวกับเรดาร์เทคโนโลยี

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

คณะกรรมการที่ปรึกษาของ ThoughtWorks ซึ่งประกอบไปด้วยผู้นำและผู้เชี่ยวชาญในเทคโนโลยีหลากหลายด้านที่มารวมตัวกันเป็นประจำในทุกปี เพื่อพูดคุยแลกเปลี่ยนถึงแผนยุทธศาสตร์ทางเทคโนโลยีภายในของ ThoughtWorks เอง และแนวโน้มของเทคโนโลยีที่กำลังมีบทบาทสำคัญในอุตสาหกรรมโลกในขณะนั้น

การรวบรวมผลการประชุมดังกล่าวถูกจัดทำขึ้นเป็นเรดาร์เทคโนโลยีฉบับนี้ โดยนำเสนอในรูปแบบที่เป็นประโยชน์กับทุกคนในวงการ ตั้งแต่นักพัฒนาจนถึงผู้บริหารระดับสูง ตัวเนื้อหานั้นเราเจตนาสรุปให้กระชับได้ใจความ หากอยากทราบถึงรายละเอียดเพิ่มเติมเราขอส่งเสริมให้คุณทดลองใช้เทคโนโลยีเหล่านั้นด้วยตัวคุณเอง

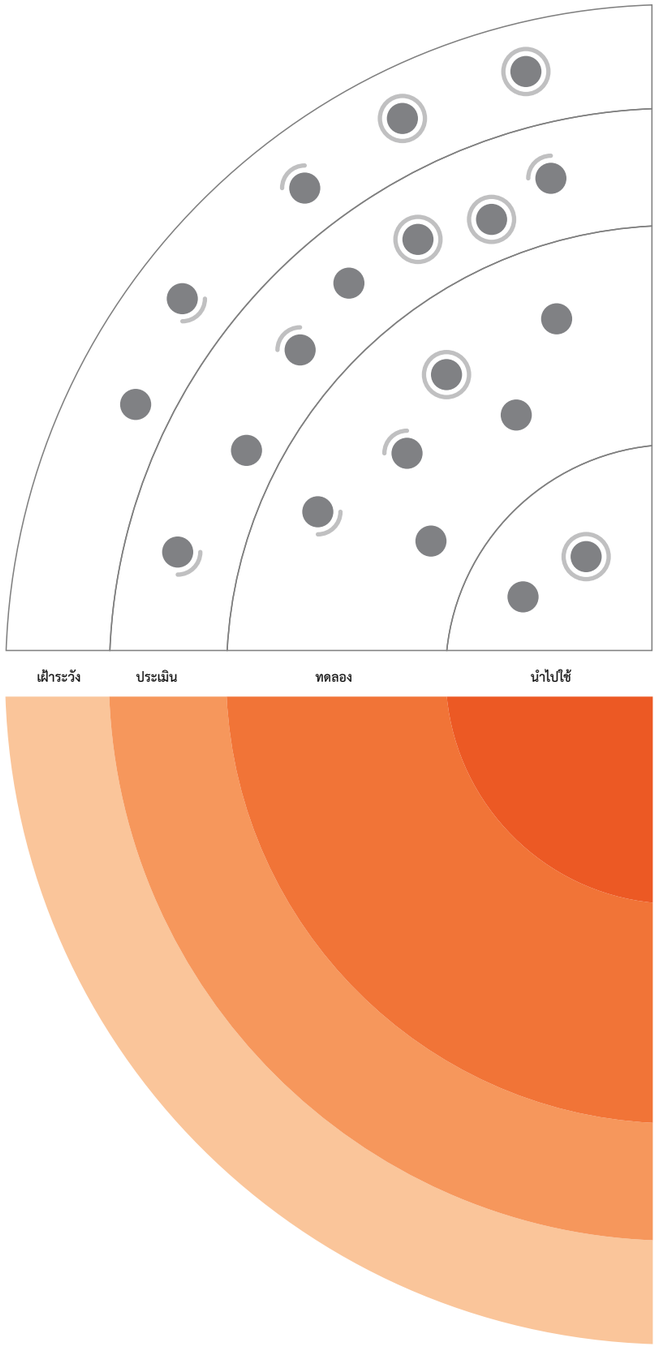
เรดาร์เทคโนโลยีใช้แผนภาพวงกลมในการนำเสนอข้อมูล โดยแบ่งพื้นที่ออกเป็น 4 กลุ่มดังนี้ เทคนิค เครื่องมือ แพลตฟอร์ม ภาษาและเฟรมเวิร์ค ในกรณีที่บางเรื่องสามารถจัด ลงได้หลายกลุ่มเราจะจัดลงในกลุ่มที่เหมาะสมที่สุด นอกจากนั้นเรายังจัดกลุ่มเรื่องต่างๆ แล้วแบ่งตามวงแหวน ออกเป็น 4 ระดับเพื่อสะท้อนมุมมองตำแหน่งของเทคโนโลยี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

หากสนใจประวัติเพิ่มเติมเกี่ยวกับเรดาร์เทคโนโลยีสามารถดูเพิ่มเติมได้ที่ thoughtworks.com/radar/faq.

ภาพรวม ของเรดาร์

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

เทคโนโลยีท่ามกลางวิกฤตการณ์

ดังคำกล่าวที่ว่า *“สถานการณ์สร้างวีรบุรุษฉันใด ความจำเป็นสร้างนวัตกรรมฉันนั้น”*

ที่ผ่านมาบางบริษัทอาจเคยทดลองทำงานทางไกลอยู่บ้าง จากที่เทคโนโลยีด้านการสื่อสารทางไกลเริ่มมีความพร้อมมากพอ แต่ทันใดนั้นการเกิดโรคระบาดครั้งใหญ่บังคับให้บริษัททั่วโลกต้องเร่งปรับตัวและเปลี่ยนพื้นฐานการทำงานไปอย่างสิ้นเชิงเพื่อรักษาธุรกิจไว้ มาในวันนี้หลายคนสังเกตเห็นแล้วว่า “การทำงานแบบ WFH” กับ “การถูกบังคับให้ WFH เพราะมีโรคระบาด” มันแตกต่างอย่างสิ้นเชิง เราเชื่อว่าในอนาคตโลกต้องใช้เวลาอีกนานกว่าจะปรับตัวเพื่อให้การทำงานทางไกลทำได้เต็มประสิทธิภาพเหมือนเคย ก่อนหน้านี้แม้แต่เราเองก็ไม่เชื่อว่าการสร้างเรดาร์แบบทางไกลเป็นเรื่องที่เป็นไปได้ แต่ตอนนี้เราก็ทำมันแล้ว เรดาร์ฉบับนี้เป็นฉบับแรกที่จัดทำขึ้นโดยไม่เจอตัวกันเลย

เพราะเราไม่อาจนิ่งเฉยไม่พูดถึงวิกฤติที่กำลังเกิดขึ้น เราถึงสื่อมันผ่านหัวข้อต่างๆ ที่พยายามเป็นตัวช่วยให้กับความต้องการอันเร่งด่วนนี้ เพื่อทำให้การทำงานร่วมกันแบบทางไกลเกิดประสิทธิภาพสูงสุด แต่หัวข้อที่ว่า “การจะทำงานร่วมกันแบบทางไกลให้ดีที่สุดต้องทำอย่างไร” เป็นหัวข้อที่ลึกและมีรายละเอียดปลีกย่อยมาก แน่นอนว่าการจัดทำเรดาร์ในรูปเล่มเช่นนี้มีข้อจำกัด เราจึงมีช่องทางอื่นนอกจากเรดาร์ฉบับนี้ เช่น รายการ พอดแคสต์ ที่เราคุยแลกเปลี่ยนประสบการณ์การสร้างเรดาร์จากทางไกล อีกทั้งยังมีบทความเกี่ยวกับประสบการณ์ต่างๆ ที่รวมคำแนะนำเกี่ยวกับการทำงานทางไกลอย่างมีประสิทธิภาพเอาไว้ มีงานสัมมนาออนไลน์ที่พูดถึงกลยุทธ์การทำงานในช่วงวิกฤต และมีลิงค์เชื่อมโยงไปถึงเอกสารอื่นๆ ของ ThoughtWorks ซึ่งหนึ่งในนั้นคือคู่มือการทำงานทางไกล

ทุกอย่างคือซอฟต์แวร์

บ่อยครั้งที่เราคอยสนับสนุนให้ส่วนอื่นๆ ในระบบนิเวศของการพัฒนาซอฟต์แวร์นำหลักปฏิบัติทางวิศวกรรมที่ทีมพัฒนาซอฟต์แวร์แบบอไจล์เป็นผู้บุกเบิกไปปรับใช้ เราหวนกลับมาพูดเรื่องนี้อยู่บ่อยครั้งเพราะเราพบกลุ่มใหม่ในระบบนิเวศนี้อยู่เรื่อยๆ ที่นำคำแนะนำนี้ไปปรับใช้ได้ไม่ถึงไหนเลย สำหรับเรดาร์ฉบับนี้เราจึงตัดสินใจเรียกร้องมันอีกครั้งหนึ่งผ่านหัวข้อต่างๆ เช่น การกำหนดโครงสร้างพื้นฐานด้วยโค้ด การกำหนดไปป์ไลน์ด้วยโค้ด หัวข้อมากมายด้านการปรับแต่งโครงสร้างพื้นฐาน หัวข้อที่เกี่ยวกับไปป์ไลน์แมชชีนเลิร์นนิง และมีหัวข้อที่เกี่ยวข้องกับการปรับใช้หลักวิศวกรรมซอฟต์แวร์อื่นๆ อีก

เรายังพบว่าทีมที่เป็นเจ้าของเรื่องนั้นไม่กล้านำหลักปฏิบัติทางวิศวกรรม เช่น การทำให้กระบวนการเป็นอัตโนมัติ การเชื่อมต่อกันอย่างต่อเนื่อง การสร้างชุดทดสอบ ไปประยุกต์ใช้อย่างจริงจัง ซึ่งเราเข้าใจดีว่าพวกเค้ามีปัจจัยมากมายคอยขัดขวางไม่ให้การเคลื่อนไหวนี้ทำได้เร็ว เช่น ปัญหาความซับซ้อน (ที่เกิดจากโดเมนปัญหาเอง หรือที่ไม่ตั้งใจให้เกิด) การขาดความรู้ ปัญหาการเมืองภายใน การขาดเครื่องมือสนับสนุน และยังมีเหตุผลอื่นอีกมาก อย่างไรก็ตามประโยชน์ที่องค์กรจะได้รับเมื่อปรับใช้หลักปฏิบัติทางวิศวกรรมแบบอไจล์มันชัดเจนและคุ้มค่าที่จะลงทุนเพื่อให้ได้มา

โลกของข้อมูลที่กำลังพร้อมและขยายออก

แนวทางหนึ่งที่กระจายตัวไปตามหัวข้อต่างๆ ในเรดาร์ฉบับนี้ จับประเด็นไปที่เรื่องความพร้อมของโลกข้อมูล โดยให้ความสำคัญไปที่เทคนิคและเครื่องมือสำหรับใช้วิเคราะห์ข้อมูลและแมชชีนเลิร์นนิงโดยเฉพาะ เริ่มจากที่เราสังเกตเห็นว่าในงานด้านการประมวลผลภาษาธรรมชาติ (NLP) ต่างมีพัฒนาการและนวัตกรรมออกใหม่อย่างต่อเนื่อง

เรารู้สึกยินดีที่เห็นเครื่องมือสำหรับสร้างแมชชีนเลิร์นนิงประเภทที่ทำงานได้อย่างครบวงจรเกิดขึ้นใหม่อยู่ตลอดและที่มีอยู่เดิมก็เติบโตอย่างต่อเนื่อง ซึ่งชุดเครื่องมือเหล่านี้เกิดจากการนำหลักปฏิบัติทางวิศวกรรมที่เข้มแข็งมาใช้ร่วมกับเครื่องมือต่างๆที่ทำงานได้ดีกับกระบวนการทำงานแบบอไจล์ อันเป็นการพิสูจน์ให้เห็นว่า “แมชชีนเลิร์นนิงก็เป็นซอฟต์แวร์เหมือนกัน” สามารถใช้หลักการพัฒนาแบบเดียวกันได้

ในส่วนของสถาปัตยกรรมแบบกระจายศูนย์อย่างไมโครเซอร์วิส เราเห็นความสนใจที่เพิ่มขึ้นอย่างมากที่จะนำสถาปัตยกรรมดาต้าเมช (data mesh) มาใช้ร่วมด้วย ซึ่งคือสถาปัตยกรรมที่รองรับการขยายของข้อมูลและทำให้การแลกเปลี่ยนข้อมูลเชิงวิเคราะห์ระหว่างเซอร์วิสทำได้อย่างมีประสิทธิภาพ จากที่อุตสาหกรรมเริ่มตั้งคำถามอย่างจริงจังว่าข้อมูลควรถูกจัดการอย่างไรในระบบสมัยใหม่ และจากทิศทางโดยรวมและมุมมองที่เปิดกว้างที่เป็นอยู่ เราจึงยินดีและคาดหวังจะได้เห็นนวัตกรรมใหม่ๆ ออกมาในอนาคตอันใกล้นี้

Kubernetes กับระบบนิเวศอันสมบูรณ์

จากการแผ่อำนาจเหนือใครในตลาดนี้ของ Kubernetes ระบบนิเวศที่อยู่รอบข้างก็ได้รับอนิสงค์ความเจริญไปด้วยอย่างช่วยไม่ได้ การที่หลายหัวข้อในเรดาร์ฉบับนี้กล่าวถึงสิ่งที่อยู่รอบตัว Kubernetes นั้นแสดงให้เห็นถึงอิทธิพลที่แผ่กว้างของมัน ยกตัวอย่าง เช่น Lens และ k9s ที่อยากช่วยให้การจัดการคลัสเตอร์ง่ายขึ้น หรือ kind ที่ต้องการช่วยปรับปรุงวิธีการทดสอบบนเครื่องนักพัฒนา และ Gloo ที่นำเสนอตัวเองเป็นเอพีไอเกตเวย์ทางเลือกหนึ่ง รวมถึง Hydra เป็นเซิร์ฟเวอร์สำหรับทำ OAuth ที่ถูกปรับแต่งมาสำหรับ Kubernetes โดยเฉพาะ และ ArgoCD ที่ใช้ความสามารถพื้นฐานของ Kubernetes เพื่อสร้างไปป์ไลน์การส่งมอบอย่างต่อเนื่อง

สิ่งเหล่านี้แสดงให้เห็นว่า Kubernetes วางตำแหน่งตัวเองได้อย่างเหมาะสม เพื่อสร้างระบบนิเวศของการสนับสนุนให้เกิดขึ้นรอบข้างตัวเอง โดยการมีความสามารถต่างๆ ที่จำเป็นมาให้อย่างครบครันแต่ก็ให้รายละเอียดสูงจนใช้ยากเกินไปสำหรับคนทั่วไป ช่องว่างความซับซ้อนนี้จึงเป็นพื้นที่ให้เกิดระบบนิเวศรอบข้างเพื่อทำให้การปรับแต่ง Kubernetes ทำงานได้ง่าย หรือเป็นการเสริมความสามารถใหม่ที่ขาดไป หาก Kubernetes ยังครองความเป็นผู้นำตลาดต่อไปเช่นนี้ เราจะเห็นระบบนิเวศที่ขยายและเติบโตต่อไปที่ใช้ประโยชน์จากจุดแข็งหรือที่พยายามปิดจุดอ่อนที่มี และเมื่อไหร่ที่ระบบนิเวศนี้พร้อมเต็มที่ เราคาดหวังว่าการวิวัฒน์ไปสู่เครื่องมือที่ระดับสูงขึ้น ที่ยังคงคุณประโยชน์ของ Kubernetes โดยไม่ทำให้ผู้ใช้สับสนกับการปรับแต่งมากมายที่เป็นไปได้

เรดาร์เทคโนโลยี

เทคนิค

นำไปใช้
1. จัดการแพลตฟอร์มภายในเหมือนจัดการผลิตภัณฑ์
2. การกำหนดโครงสร้างพื้นฐานด้วยโค้ด
3. ไมโครฟรอนท์เอนด์
4. การกำหนดไปป์ไลน์ด้วยโค้ด
5. การแพร่กันทางไกลที่ใช้การได้จริง
6. การเปิด/ปิดฟีเจอร์ที่เรียบง่ายที่สุดที่จะเป็นไปได้

ทดลอง
7. การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML)
8. การทดสอบความลำเอียงทางจริยธรรม
9. การใช้ GraphQL เพื่อรวบรวมข้อมูลทางฝั่งเซิร์ฟเวอร์
10. ไมโครฟรอนท์เอนด์สำหรับการพัฒนาแอปพลิเคชันบนมือถือ
11. ทีมวิศวกรการผลิตแพลตฟอร์ม
12. นโยบายความมั่นคงด้วยโค้ด
13. ขั้นตอนการเรียนรู้แบบกึ่งสอนแบบวนซ้ำ
14. การถ่ายทอดการเรียนรู้สำหรับ NLP
15. ใช้กระบวนการหรือวิธีที่รับรองการทำงานแบบทางไกลอย่างเป็นธรรมชาติ
16. สถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero Trust Architecture)

ประเมิน
17. Data mesh
18. เอกลักษณ์แบบกระจายศูนย์
19. การนิยามไปป์ไลน์ข้อมูลแบบประกาศ
20. DeepWalk
21. การจัดการระบบที่จดจำสถานะข้อมูลด้วยเครื่องมือบริหารควบคุมคอนเทเนอร์
22. การรันบิลด์ในสนามซ้อม

เฝ้าระวัง
23. การย้ายขึ้นคลาวด์แบบยกวางทั้งระบบ
24. การย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด
25. การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ
26. GitFlow และการสร้างบรานช์ที่มีช่วงชีวิตยาวนาน
27. การทดสอบแบบบันทึกภาพแต่เพียงอย่างเดียว

แพลตฟอร์ม

นำไปใช้
28. .NET Core
29. Istio

ทดลอง
30. Anka
31. Argo CD
32. Crowdin
33. eBPF
34. Firebase
35. Hot Chocolate
36. Hydra
37. OpenTelemetry
38. Snowflake

ประเมิน
39. Anthos
40. Apache Pulsar
41. Cosmos
42. Google BigQuery ML
43. JupyterLab
44. Marquez
45. Matomo
46. MeiliSearch
47. Stratos
48. Trillian

เฝ้าระวัง
49. การใช้งาน Node สำหรับทุกสิ่ง

ภาษาและไลบรารี

นำไปใช้
50. Cypress
51. Figma

ทดลอง
52. Dojo
53. DVC
54. เครื่องมือติดตามการทดลองสำหรับแมชชีนเลิร์นนิง
55. Goss
56. Jaeger
57. k9s
58. kind
59. mkcert
60. MURAL
61. Open Policy Agent (OPA)
62. Optimal Workshop
63. Phrase
64. ScoutSuite
65. เครื่องมือสำหรับทดสอบการเปลี่ยนแปลงด้านภาพแสดงผล
66. Visual Studio Live Share

ประเมิน
67. Apache Superset
68. AsyncAPI
69. ConfigCat
70. Gitpod
71. Gloo
72. Lens
73. Manifold
74. Sizzy
75. Snowpack
76. tfsec

เฝ้าระวัง

นำไปใช้
77. React Hooks
78. React Testing Library
79. Vue.js

ทดลอง
80. CSS-in-JS
81. Exposed
82. GraphQL Inspector
83. Karate
84. Koin
85. NestJS
86. PyTorch
87. Rust
88. Sarama
89. SwiftUI

ประเมิน
90. Clinic.js Bubbleprof
91. Deequ
92. ERNIE
93. MediaPipe
94. Tailwind CSS
95. Tamer
96. Wire
97. XState

เฝ้าระวัง
98. Enzyme

TECHNOLOGY RADAR Vol. 22

เทคนิค

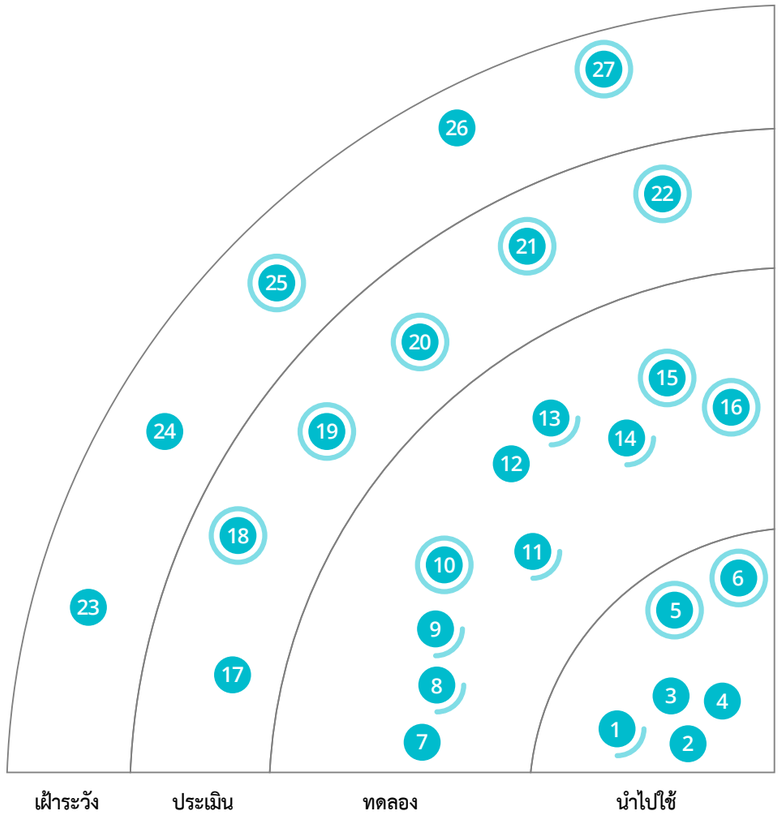
เทคนิค

จัดการแพลตฟอร์มภายในเหมือนจัดการผลิตภัณฑ์
นำไปใช้

องค์กรจำนวนมากพยายามสร้างแพลตฟอร์มภายในขึ้นมาเพื่อลดความซับซ้อนทางเทคนิคลงทำให้การพัฒนาผลิตภัณฑ์ดิจิทัลต่างๆ ทำได้รวดเร็วและมีประสิทธิภาพยิ่งขึ้น บริษัทที่ประสบความสำเร็จกับการใช้กลยุทธ์นี้ ต่างก็ประยุกต์แนวคิด **จัดการแพลตฟอร์มภายในเหมือนจัดการผลิตภัณฑ์** เข้ากับองค์กรของตน ด้วยการเอาใจใส่ลูกค้าภายในของตน ซึ่งในที่นี้คือบรรดาทีมพัฒนานั่นเอง เพื่อร่วมมือกันออกแบบพัฒนาแพลตฟอร์มที่ตอบสนองความต้องการ มีผู้จัดการผลิตภัณฑ์ทำหน้าที่วางแผนกลยุทธ์โดยมีเป้าหมายอยู่ที่การส่งมอบแพลตฟอร์มที่เพิ่มคุณค่าทางธุรกิจและทำให้ประสบการณ์ของนักพัฒนาดีขึ้น ขณะเดียวกันเป็นที่น่าเสียดายเพราะมีอีกหลายองค์กรไม่ประสบความสำเร็จกับภารกิจนี้ เพราะแพลตฟอร์มถูกสร้างจากทีมที่แยกตัวไม่สื่อสารกัน ทำงานบนสมมติฐานที่ขาดการตรวจสอบและขาดผู้ใช้งานจริง ถึงแม้ใช้แผนการเชิงรุกในการลงทุนเพื่อสร้างให้เสร็จก็ตามที ก็จะได้แพลตฟอร์มที่ไม่คุ้มค่าการลงทุน ที่สูบเอาพลังงานและความสามารถมหาศาลขององค์กรไปกับการให้ได้มา อย่างที่ทราบกันดีว่า หัวใจของการจัดการผลิตภัณฑ์ที่ดีคือการสร้างผลิตภัณฑ์ที่เป็นที่รักของผู้ใช้ ซึ่งไม่เว้นแม้แต่ผลิตภัณฑ์ภายในองค์กร

การกำหนดโครงสร้างพื้นฐานด้วยโค้ด
นำไปใช้

ถึงแม้ว่า **การกำหนดโครงสร้างพื้นฐานด้วยโค้ด** จะเป็นเทคนิคที่มีมานานพอสมควรแล้ว (เราได้พูดถึงเทคนิคนี้ในเรดาร์เมื่อปี 2011) แต่ในยุคของคลาวด์เทคโนโลยีสมัยใหม่ที่การสร้างและติดตั้งโครงสร้างพื้นฐานเกิดจากการส่งคำสั่งการตั้งค่าไปยังแพลตฟอร์มคลาวด์เท่านั้น เทคนิคนี้กลับยิ่งมีความสำคัญมากขึ้นไปอีก จากที่เราเลือกใช้คำว่า “ด้วยโค้ด” จึงหมายรวมการนำหลักปฏิบัติที่ดีที่เราเรียนรู้จากการพัฒนาซอฟต์แวร์มาใช้ในบริบทของการสร้างโครงสร้างพื้นฐานด้วย ตั้งแต่การกำหนดเวอร์ชันให้โค้ด



การยึดมั่นต่อหลักการ DRY การแบ่งโค้ดออกเป็นโครงสร้างแบบแยกส่วนประกอบ การรักษาโค้ดให้อ่านง่ายและปรับแก้ได้ง่าย การทดสอบและติดตั้งระบบได้อย่างอัตโนมัติ สิ่งเหล่านี้ล้วนเป็นหลักปฏิบัติที่สำคัญอย่างยิ่ง เพราะฉะนั้น เป็นหน้าที่ของผู้ที่เข้าใจการพัฒนาซอฟต์แวร์และโครงสร้างพื้นฐานอย่างลึกซึ้งที่ต้องเอาใจใส่และให้ความช่วยเหลือแก่ผู้ร่วมงานคนอื่นที่ยังไม่เข้าใจ เพราะการพูดว่า “ให้ดูแลโครงสร้างพื้นฐานเหมือนที่เราดูแลโค้ด” นั้นยังไม่เพียงพอ เราต้องทำให้มั่นใจว่าสิ่งที่เราเรียนรู้และปฏิบัติกันในโลกของการพัฒนาซอฟต์แวร์จะถูกประยุกต์ใช้อย่างสม่ำเสมอตั้งแต่ต้นจนจบในโลกของการกำหนดโครงสร้างพื้นฐานด้วยเช่นกัน

ไมโครฟรอนท์เอนด์
นำไปใช้

เราเห็นถึงประโยชน์ของการใช้สถาปัตยกรรมแบบไมโครเซอร์วิสอย่างมาก ซึ่งมันเปิดโอกาสให้ทีมต่างๆ สามารถขยายการพัฒนาเซอร์วิสออกไปได้มากขึ้น อันเนื่องจากแต่ละเซอร์วิสสามารถติดตั้งและดูแลแยกกันได้อย่างอิสระไม่ขึ้นต่อกัน น่าเสียดายที่เรายังคงเห็นหลายๆ ทีมประยุกต์ใช้ไมโครเซอร์วิสเฉพาะที่ระบบแบ็กเอนด์เท่านั้น ส่วนแอปพลิเคชันฟรอนท์เอนด์กลับอยู่ในรูปแบบโมโนลิธที่มีขนาดใหญ่และซับซ้อนพันกันเป็นแอปพลิเคชันเดียว ซึ่งเป็นการลดทอนประโยชน์ของไมโครเซอร์วิสไปอย่างมาก ขณะนี้เราเห็นการปรับใช้สถาปัตยกรรมแบบ **ไมโครฟรอนท์เอนด์** อย่างต่อเนื่องในรูปแบบที่แตกต่างกันไป เพื่อใช้จัดการความซับซ้อนของแอปพลิเคชันฟรอนท์เอนด์ขนาด

นำไปใช้
1. จัดการแพลตฟอร์มภายในเหมือนจัดการผลิตภัณฑ์
2. การกำหนดโครงสร้างพื้นฐานด้วยโค้ด
3. ไมโครฟรอนท์เอนด์
4. การกำหนดไปป์ไลน์ด้วยโค้ด
5. การแพร่กันทางไกลที่ใช้การได้จริง
6. การเปิด/ปิดฟีเจอร์ที่เรียบง่ายที่สุดที่จะเป็นไปได้

ทดลอง
7. การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML)
8. การทดสอบความลำเอียงทางจริยธรรม
9. การใช้ GraphQL เพื่อรวบรวมข้อมูลทางฝั่งเซิร์ฟเวอร์
10. ไมโครฟรอนท์เอนด์สำหรับการพัฒนาแอปพลิเคชันบนมือถือ
11. ทีมวิศวกรการผลิตแพลตฟอร์ม
12. นโยบายความมั่นคงด้วยโค้ด
13. ขั้นตอนการเรียนรู้แบบกึ่งสอนแบบวนซ้ำ
14. การถ่ายทอดการเรียนรู้สำหรับ NLP
15. ใช้กระบวนการหรือวิธีที่รับรองการทำงานแบบทางไกลอย่างเป็นธรรมชาติ
16. สถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero Trust Architecture)

ประเมิน
17. Data mesh
18. เอกลักษณ์แบบกระจายศูนย์
19. การนิยามไปป์ไลน์ข้อมูลแบบประกาศ
20. DeepWalk
21. การจัดการระบบที่จดจำสถานะข้อมูลด้วยเครื่องมือบริหารควบคุมคอนเทเนอร์
22. การรันบิลด์ในสนามซ้อม

เฝ้าระวัง
23. การย้ายขึ้นคลาวด์แบบยกวางทั้งระบบ
24. การย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด
25. การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ
26. GitFlow และการสร้างบรานช์ที่มีช่วงชีวิตยาวนาน
27. การทดสอบแบบบันทึกภาพแต่เพียงอย่างเดียว

เทคนิค

เราเชื่อในการทำแพร์โปรแกรมมิ่ง แต่โลกหลังวิกฤติโควิด-19 คือโลกแห่งการทำงานทางไกล การแพร์ที่ประสบความสำเร็จจำเป็นต้องใช้วิธีการต่างๆ ที่เหมาะสมในทางปฏิบัติเพื่อให้เกิดประสิทธิภาพสูงสุด

(การแพร์กันทางไกลที่ใช้การได้จริง)

เราสนับสนุนให้ใช้วิธีการเปิด/ปิดฟีเจอร์ที่เรียบง่ายที่สุดเท่าที่เป็นไปได้ แทนการใช้วิธีการเปิด/ปิดที่ซับซ้อน

(การเปิด/ปิดฟีเจอร์ที่เรียบง่ายที่สุดที่จะเป็นไปได้)

ใหญ่ ที่ต้องใช้หลายทีมในการแก้ไขดูแลพร้อมกัน ในเดือนมิถุนายนที่ผ่านมา หนึ่งในผู้คิดค้นเทคนิคนี้ได้ตีพิมพ์บทความแนะนำ ที่กลายเป็นแหล่งอ้างอิงสำหรับการทำไมโครฟรอนท์เอนด์ ในบทความแสดงให้เห็นว่ารูปแบบสถาปัตยกรรมนี้สามารถสร้างจากกลไกต่างๆ ที่แตกต่างกัน และยังมีตัวอย่างแอปพลิเคชันที่พัฒนาด้วย React.js ที่ทำไว้เป็นตัวอย่างอ้างอิงอีกด้วย เรามั่นใจว่าสถาปัตยกรรมรูปแบบนี้จะได้รับความนิยมสูงขึ้นไปอีกในอนาคต จากที่หลายองค์กรต่างพากันแตกงานพัฒนาส่วนต่อประสานผู้ใช้ออกจากกันมากขึ้น เพื่อกระจายการดูแลสู่ทีมต่างๆ นั่นเอง

การกำหนดไปป์ไลน์ด้วยโค้ด

นำไปใช้

การกำหนดไปป์ไลน์ด้วยโค้ด เป็นเทคนิคที่เน้นยำว่า ทุกการตั้งค่าให้แก่ไปป์ไลน์เพื่อการส่งมอบซอฟต์แวร์ใดๆ ควรอยู่ในรูปแบบของโค้ดเสมอ ตั้งแต่ การบิลด์ การทดสอบ และการติดตั้งแอปพลิเคชันหรือโครงสร้างพื้นฐาน โดยมันควรจะถูกกำหนดเวอร์ชัน แยกเป็นส่วนประกอบที่สามารถนำมาใช้ซ้ำภายหลัง สามารถทดสอบและติดตั้งได้อย่างอัตโนมัติ จากที่หลายองค์กรปรับตัวเข้าสู่การจัดทีมให้อยู่ในรูปแบบกระจายตัวกันและเป็นอิสระต่อกันมากขึ้นเพื่อสร้างไมโครเซอร์วิสหรือไมโครฟรอนท์เอนด์ ทำให้หลักปฏิบัติทางวิศวกรรมนี้กลายเป็นสิ่งจำเป็นยิ่งขึ้นไปอีก เพื่อทำให้เกิดความสอดคล้องทั่วทั้งองค์กรในการสร้างและติดตั้งซอฟต์แวร์ ด้วยความต้องการนี้เองจึงเกิดเครื่องมือและเทมเพลตในการกำหนดไปป์ไลน์ขึ้นมาเพื่อช่วยสร้างมาตรฐานให้การบิลด์และติดตั้งซอฟต์แวร์ ซึ่งเครื่องมือเหล่านี้จะรองรับการใช้ _ไปป์ไลน์การส่งมอบแบบประกาศ_ ซึ่งเป็นเสมือนเอกสารพิมพ์เขียวที่กำกับให้งานที่อยู่ตามขั้นตอนต่างๆ ของไปป์ไลน์เริ่มทำงาน วิธีนี้เป็นการซ่อนรายละเอียดของวิธีการทำงานภายในเอาไว้ด้วย เพราะฉะนั้นการเลือกเครื่องมือเพื่อสร้างไปป์ไลน์ใดๆ ควรคำนึงถึงความสามารถในการจัดการไปป์ไลน์ด้วยโค้ดเป็นปัจจัยหนึ่งเสมอ

การแพร์กันทางไกลที่ใช้การได้จริง

นำไปใช้

พวกเราเชื่ออย่างจริงจังว่าการ แพร์โปรแกรมมิ่ง ช่วยพัฒนาให้โค้ดมีคุณภาพดีขึ้น ช่วยกระจายความรู้ภายในทีมได้มากกว่า และทำให้การส่งมอบซอฟต์แวร์ในภาพรวมนั้นเร็วขึ้น โลกหลังวิกฤตการณ์โควิด-19 หลายๆ ทีมพัฒนาซอฟต์แวร์จะหันมาใช้การทำงานแบบกระจายสถานที่กันหรือทำงานทางไกลกันมากขึ้น ในสถานการณ์เช่นนี้ เราขอแนะนำให้ใช้ **การแพร์กันทางไกลที่ใช้การได้จริง** ซึ่งหมายถึงการปรับใช้การแพร์ริ่งให้เหมาะสมกับเครื่องมือที่มีอยู่ในขณะนั้น โดยพิจารณาใช้เครื่องมืออย่าง Visual Studio Live Share ที่ให้ประสิทธิภาพดีกว่าและลดความหน่วงของการทำงานทางไกล เลือกใช้เครื่องมือประเภทแชร์หน้าจอก็ต่อเมื่อผู้เข้าร่วมทุกคนอยู่ในพื้นที่ใกล้เคียงกันและเชื่อมต่ออยู่บนอินเทอร์เน็ตความเร็วสูง เลือกแพร์กันระหว่างนักพัฒนาที่อยู่ในเขตเวลาใกล้เคียงกันมากกว่าจะคาดหวังกับการแพร์กับผู้เข้าร่วมจากสถานที่ห่างไกล ถ้าหากไม่สามารถแพร์ได้ด้วยเหตุผลทางโลจิสติกส์ ควรกลับไปใช้แนวทางปฏิบัติอื่นๆ เช่น การเขียนโปรแกรมคนเดียวเสริมด้วยการรีวิวโค้ด การทำงานร่วมกันผ่านการส่ง pull-request (แต่ต้องระวังเรื่อง Gitflow และการสร้างบรานช์ที่มีช่วงชีวิตยาวนาน) หรือใช้การแพร์กันระยะสั้นสำหรับโค้ดส่วนที่สำคัญ เราแพร์กันแบบทางไกลมาหลายปีแล้วและพบว่าได้ผลดีถ้าปรับใช้อย่างเหมาะสมในทางปฏิบัติ

การเปิด/ปิดฟีเจอร์ที่เรียบง่ายที่สุดที่จะเป็นไปได้

นำไปใช้

เป็นเรื่องน่าเสียดายที่เทคนิคการเปิด/ปิดฟีเจอร์ (feature toggles) ไม่ได้รับความนิยมเท่าที่เราอยากให้มันเป็น และเรามักเห็นการใช้งานผิดจุดประสงค์และประเภทของมันอยู่บ่อยครั้งเพื่อรับประโยชน์จากการเชื่อมต่อกันอย่างต่อเนื่อง เราพบว่าทีมมักเลือกใช้แพลตฟอร์มขนาดใหญ่ อย่าง LaunchDarkly เพื่อได้มาซึ่งความสามารถนี้ ทั้งที่จริงการใช้เงื่อนไข ถ้า/แล้ว (if/else) ธรรมดาก็เพียงพอแล้ว เพราะฉะนั้น หากการทดสอบเพื่อเลือกระหว่าง A/B หรือการปล่อยซอฟต์แวร์แบบแคนารี (Canary release) หรือการให้ฝั่งธุรกิจเป็นผู้รับผิดชอบในการปล่อยฟีเจอร์ออกสู่ตลาดไม่ใช่สิ่งจำเป็น เราแนะนำอย่างยิ่งให้ใช้ **การเปิด/ปิดฟีเจอร์ที่เรียบง่ายที่สุดเท่าที่จะเป็นไปได้** แทนที่การใช้เฟรมเวิร์คการเปิด/ปิดฟีเจอร์ที่ซับซ้อน

การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML)

ทดลอง

การใช้แมชชีนเลิร์นนิงเพื่อเพิ่มความฉลาดให้กับแอปพลิเคชันและบริการทางธุรกิจนั้นไม่ได้มีแค่การฝึกสอนโมเดลแล้วก็นำมันไปใช้เท่านั้น แต่ยังหมายถึงการสร้างวัฏจักรที่ครบวงจรที่สามารถทำซ้ำได้ เริ่มตั้งแต่การฝึกสอน การทดสอบ การติดตั้ง ไปจนถึงการวัดผลการทำงานของโมเดล เพราะฉะนั้น **การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง (CD4ML)** จึงเป็นเทคนิคที่ทำให้วัฏจักรการพัฒนา ติดตั้ง และวัดผลโมเดลแมชชีนเลิร์นนิงมีความน่าเชื่อถือ กลุ่มเทคโนโลยีที่เข้ามาช่วยให้เกิด CD4ML ประกอบไปด้วยหลายองค์ประกอบ เช่น เครื่องมือสำหรับเข้าถึงและวิเคราะห์ข้อมูล เครื่องมือไว้กำหนดบันทึกเวอร์ชันให้กับผลของการทดลอง (เช่น ข้อมูล โมเดลและโค้ดที่ใช้) เครื่องมือไว้สร้างไปป์ไลน์สำหรับการส่งมอบอย่างต่อเนื่อง เครื่องมือไว้สร้างสภาพแวดล้อมแบบอัตโนมัติสำหรับทำการติดตั้งและทดลอง เครื่องมือไว้วัดผลและติดตามประสิทธิภาพของโมเดล และเครื่องมือไว้เฝ้าตรวจการปฏิบัติการต่างๆ ที่เกิดขึ้นกับโมเดล บริษัทต่างๆ สามารถเลือกใช้เครื่องมือที่แตกต่างกันได้ขึ้นอยู่กับเทคโนโลยีที่มีอยู่แล้วในบริษัทนั้นๆ CD4ML จะเน้นไปที่การทำให้ทุกอย่างเป็นอัตโนมัติและลดปฏิบัติการที่ต้องทำด้วยมือลง สำหรับพวกเราแล้ว CD4ML คือวิธีการพัฒนาโมเดลแมชชีนเลิร์นนิงที่เราใช้ในทางปฏิบัติมาเสมอ

การทดสอบความลำเอียงทางจริยธรรม

ทดลอง

ในปีที่ผ่านมา เราเห็นผู้คนให้ความสนใจแมชชีนเลิร์นนิงในอีกแง่มุมโดยเฉพาะหัวข้อที่เกี่ยวข้องกับนิวรัลเน็ตเวิร์กเชิงลึก ความสามารถที่น่าทึ่งของโมเดลเหล่านี้ได้ขับเคลื่อนให้เกิดการพัฒนาเครื่องมือและเทคนิคมากมาย แต่ในระยะหลังเริ่มมีความกังวลว่าโมเดลเหล่านั้นอาจก่อให้เกิดผลร้ายต่อสังคมโดยไม่เจตนา ตัวอย่างเช่น ในธุรกิจสินเชื่อหนึ่ง เมื่อโมเดลผ่านการฝึกสอนให้ตัดสินใจอนุมัติการขอสินเชื่อโดยเห็นแก่ผลกำไรสูงสุดทางธุรกิจ มันอาจเลือกตัดสิทธิ์ผู้ขออนุมัติซึ่งเป็นผู้ด้อยโอกาสทิ้งไปทั้งหมด เรายังโชคดีที่ผู้คนเริ่มให้ความสนใจวิธี**การทดสอบความลำเอียงทางจริยธรรม (Ethical bias testing)** มากขึ้น ซึ่งช่วยเผยถึงความเป็นไปได้ของความไม่เท่าเทียมในการตัดสินใจออกมา

เครื่องมือเช่น lime, AI Fairness 360 หรือ What-If Tool ช่วยเผยถึงความคลาดเคลื่อนที่อาจเกิดขึ้นได้จากการมีกลุ่มข้อมูลที่น้อยเกินไปในชุดข้อมูลที่ใช้ฝึกสอนโมเดล ส่วนเครื่องมืออย่าง Google Facets หรือ Facets Dive ช่วยให้เราเห็นภาพและค้นพบข้อมูลกลุ่มย่อยที่คาดไม่ถึงภายในแหล่งข้อมูลที่ใช้ฝึกสอน เราใช้เทคนิค lime (local interpretable model-agnostic explanations) ร่วมกับเทคนิคนี้ เพื่อช่วยให้เราเข้าใจผลการคาดการณ์ของตัวจำแนกใดๆ ในแมชชีนเลิร์นนิง และเพื่อให้รู้ว่ามันทำงานอย่างไร

การใช้ GraphQL เพื่อรวบรวมข้อมูลทางฝั่งเซิร์ฟเวอร์

ทดลอง

เราเห็นเครื่องมือ เช่น Apollo Federation ออกมามากขึ้นที่สามารถรวมเอนด์พอย์นต่างๆ ของ GraphQL มาสร้างเป็นกราฟเดียวกันได้ ด้วยเหตุนี้เราถึงอยากเตือนให้ระวังการใช้ GraphQL อย่างผิดจุดประสงค์ โดยเฉพาะการใช้มันเป็นโปรโตคอลสำหรับสื่อสารระหว่างเซิร์ฟเวอร์กับเซิร์ฟเวอร์กันเอง สิ่งที่เราปฏิบัติคือ **ใช้ GraphQL เพื่อรวบรวมข้อมูลทางฝั่งเซิร์ฟเวอร์** เท่านั้น ซึ่งรูปแบบนี้ทำให้ไมโครเซอร์วิสยังคงทำหน้าที่ประกาศเป็น RESTFul API ออกมาเหมือนปกติ ส่วนเบื้องหลัง แบกเอนด์สำหรับฟรอนท์เอนด์ (Backend for Frontends, BFF) หรือเซอร์วิสสำหรับรวบรวมผลจะร้อยเรียงข้อมูลจากเซอร์วิสต่างๆ เข้าด้วยกัน ผ่าน GraphQL อีกทีหนึ่ง ส่วนการได้มาซึ่งรูปกราฟที่ถูกต้องเกิดจากการออกแบบร่วมกันระหว่างทีมกับผู้เชี่ยวชาญในโดเมน เพื่อให้ภายในกราฟใช้ภาษาทางธุรกิจที่ถูกต้องภายใต้บริบทนั้นๆ การใช้เทคนิคนี้ช่วยทำให้การสร้างเซอร์วิสเพื่อรวบรวมข้อมูลหรือการสร้าง BFF ทำได้ง่ายขึ้น ในขณะเดียวกันก็เป็นการสนับสนุนให้ออกแบบเซอร์วิสให้ถูกต้องเพื่อหลีกเลี่ยงการสร้าง RESTful API ที่ขาดความสมบูรณ์

ไมโครฟรอนท์เอนด์สำหรับการพัฒนาแอปพลิเคชันบนมือถือ

ทดลอง

ตั้งแต่เรากล่าวถึง micro frontends ในเรดาร์ฉบับปี 2016 เราก็พบเห็นการนำเทคนิคนี้ไปใช้กับการพัฒนาเว็บไซต์อย่างแพร่หลาย อย่างไรก็ตาม เมื่อเร็วๆ นี้เราเริ่มเห็นโครงการที่ต่อยอดสถาปัตยกรรมนี้เพื่อสร้าง **ไมโครฟรอนท์เอนด์สำหรับการพัฒนาแอปพลิเคชันบนมือถือ** แล้วเช่นกัน เมื่อแอปพลิเคชันเริ่มมีขนาดใหญ่และซับซ้อนมากถึงระดับหนึ่ง เราจึงจำเป็นต้องกระจายการพัฒนาออกเป็นหลายๆ ทีม แต่วิธีนี้ก็นำมาซึ่งความท้าทายในการที่ทีมแต่ละทีมพยายามจะรักษาความอิสระในการทำงานของตนเอง ในขณะเดียวกันก็ต้องพยายามเชื่อมต่องานกับทีมอื่นบนแอปพลิเคชันเดียวกัน ถึงแม้ส่วนมากเราพบว่าทีมต่างๆ เลือกที่จะสร้างเฟรมเวิร์คส่วนตัวขึ้นมาใช้เองเพื่อรองรับวิธีการทำงานเช่นนี้ แต่ก็เริ่มมีเฟรมเวิร์คออกมาเพื่อแบ่งโค้ดออกเป็นส่วนย่อยๆ แล้วเช่นกัน อย่างเช่น Atlas and Beehive ซึ่งช่วยให้การเชื่อมรวมงานจากหลายๆ ทีมเป็นแอปพลิเคชันเดียวกันมีกระบวนการที่เรียบง่ายขึ้น

ทีมวิศวกรการผลิตแพลตฟอร์ม

ทดลอง

การเปิดรับการใช้คลาวด์และวัฒนธรรมแบบ DevOps ที่มากขึ้น แม้จะช่วยเพิ่มประสิทธิภาพของทีมให้มีความคล่องตัวกว่าเดิม จากการลดการพึ่งพิงทีมดูแลระบบจากส่วนกลางและงานโครงสร้างพื้นฐานไปได้ แต่ก็ทำให้เกิดข้อจำกัดแก่ทีมที่ขาดทักษะที่จะดูแลแอปพลิเคชันได้ด้วยตัวเอง แบบครบวงจร บางองค์กรเลือกแก้ปัญหานี้โดยการสร้าง **ทีมวิศวกรการผลิตแพลตฟอร์ม** ขึ้นมา ซึ่งมีหน้าที่ดูแลแพลตฟอร์มภายใน โดยมีเป้าหมายให้ทีมส่งมอบอื่นๆ สามารถติดตั้งและดูแลระบบได้ด้วยตัวเองโดยใช้เวลาในการส่งมอบที่สั้นลงและลดความซับซ้อนของชุดเทคโนโลยีที่ใช้ลง โดยทีมนี้จะมุ่งเน้นไปที่การสร้าง API เพื่อให้ทีมอื่นสามารถให้บริการตัวเองได้ หรือสร้างเครื่องมืออำนวยความสะดวกต่างๆ เพื่อให้ทีมส่งมอบสามารถรับผิดชอบและรองรับงานของตนในแพลตฟอร์มได้โดยไม่ติดขัด ทั้งนี้องค์กรที่กำลังพิจารณาจัดตั้งทีมนี้ขึ้นมานั้นต้องระมัดระวังอย่างมากไม่ให้เกิดทีม DevOps แยกเป็นอีกแผนก หรือการเปลี่ยนแค่ชื่อแผนก หรือการยกโครงสร้างแผนกปฏิบัติการเดิมที่มีอยู่แล้ว มาตั้งใหม่ หากคุณกำลังสงสัยว่าแล้วทีมแพลตฟอร์มที่ดีต้องมีหน้าตาอย่างไร เราใช้แนวคิดจาก Team Topologies เพื่อแยกทีมแพลตฟอร์มในโครงการของเราออกเป็นทีมสร้างความพร้อมเพื่อให้คำปรึกษาและช่วยเหลือในช่วงเริ่มต้น ทีมหลักที่ทำงานอยู่ภายในแพลตฟอร์มอีกทีหนึ่งและทีมที่มุ่งเน้นไปที่การสร้างคุณค่าให้แพลตฟอร์ม

นโยบายความมั่นคงด้วยโค้ด

ทดลอง

นโยบายความมั่นคงคือกฎและขั้นตอนที่ช่วยป้องกันระบบของเราจากภัยคุกคามและการรบกวนต่างๆ ตัวอย่างเช่น นโยบายด้านสิทธิ์การเข้าถึงสามารถกำหนดและบังคับว่าใครสามารถเข้าถึงเซอร์วิสและทรัพยากรใด ในสถานการณ์ไหนได้บ้าง หรือนโยบายความมั่นคงของระบบเครือข่ายที่สามารถจำกัดอัตราการรับส่งข้อมูลของแต่ละเซอร์วิสได้อย่างยืดหยุ่น ความซับซ้อนของภาพรวมเทคโนโลยีทั้งหมดในปัจจุบัน เรียกร้องให้เราใช้ นโยบายความมั่นคงด้วยโค้ด แทนการกำหนดด้วยกระบวนการทำงานของคน ซึ่งคือการที่เรากำหนดนโยบายลงในโค้ด ที่สามารถบันทึกเก็บเป็นหลักฐานด้วยเวอร์ชันคอนโทรล สามารถตรวจสอบนโยบายและเปิดใช้งานนโยบายต่างๆ ได้อย่างอัตโนมัติ รวมไปถึงการเฝ้าติดตามวัดผลประสิทธิภาพของมัน เครื่องมือเช่น Open Policy Agent หรือแพลตฟอร์มอย่าง Istio ช่วยให้การกำหนดนโยบายและกลไกการบังคับใช้ทำได้อย่างยืดหยุ่น และรองรับแนวปฏิบัติการจัดการนโยบายรักษาความมั่นคงด้วยโค้ด

ขั้นตอนการเรียนรู้แบบกึ่งสอนแบบวนซ้ำ

ทดลอง

ขั้นตอนการเรียนรู้แบบกึ่งสอนแบบวนซ้ำ (semi-supervised learning loops) เป็นประเภทหนึ่งในวิธีการทำแมชชีนเลิร์นนิงที่ ขั้นตอนสามารถทำวนซ้ำได้เรื่อยๆ โดยนำความสัมพันธ์ที่หาได้จากข้อมูลที่ไม่มีป้ายกำกับ (label) มาใช้ให้เกิดประโยชน์ ซึ่งเทคนิคนี้อาจช่วยให้ประสิทธิภาพของโมเดลดีขึ้นได้จากการนำข้อมูลที่มีและไม่มีป้ายกำกับมาผสมรวมกันในรูปแบบต่างๆ ซึ่งโดยพื้นฐานแล้วมันใช้การเปรียบเทียบโมเดลที่ถูกฝึกสอนหลายๆ แบบด้วยซับเซ็ตของข้อมูลที่แตกต่างกันไป เทคนิคนี้ต่างจากเทคนิคการเรียนรู้ที่ไม่มีการสอนจากมนุษย์ (unsupervised learning) ที่เครื่องจะอนุมานการจัดกลุ่มจากข้อมูลที่ไม่มีป้ายกำกับ แล้วก็ต่างจากเทคนิคการเรียนรู้ที่มีการสอน (supervised learning) ที่ข้อมูลที่ใช้ฝึกสอนถูกแปะป้ายไว้หมดแล้ว โดยการเรียนรู้แบบกึ่งสอนนำเอาข้อดีของทั้งคู่มาใช้ โดยใช้ข้อมูลชุดเล็กๆ จากข้อมูลที่มีป้ายกำกับ และข้อมูลชุดที่ใหญ่กว่ามากจากข้อมูลที่ไม่มีป้ายกำกับมาผสมกัน เทคนิคนี้ยังมีความคล้ายกับเทคนิคการเรียนรู้ที่มีการสอนแบบมีผู้กระทำ (active learning) ซึ่งมีมนุษย์ช่วยแปะป้ายให้กับข้อมูลที่กำกวมอีกด้วย จากที่ผู้มีความชำนาญที่สามารถแปะป้ายให้กับข้อมูลได้อย่างแม่นยำเป็นทรัพยากรจำกัด และการแปะป้ายข้อมูลก็เป็นขั้นตอนที่กินเวลานานที่สุดในบรรดา

เทคนิค

มีเรื่องกังวลที่กำลังพุ่งขึ้นมาว่าบางโมเดลอาจก่อให้เกิดผลร้ายต่อสังคมโดยไม่เจตนา โชคยังดีที่เราเห็นผู้คนให้ความสนใจวิธีการทดสอบความลำเอียงทางจริยธรรมมากขึ้น ซึ่งช่วยเผยถึงความเป็นไปได้ของความไม่เท่าเทียมในการตัดสินใจออกมา

(การทดสอบความลำเอียงทางจริยธรรม)

เทคนิคไมโครฟรอนท์เอนด์ถูกนำไปใช้กับการพัฒนาเว็บไซต์อย่างแพร่หลาย เรากำลังเห็นการวางสถาปัตยกรรมเช่นนี้สำหรับโลกมือถือเช่นกัน

(ไมโครฟรอนท์เอนด์สำหรับการพัฒนาแอปพลิเคชันบนมือถือ)

เทคนิค

สถาปัตยกรรมแบบไร้ความเชื่อใจได้เน้นย้ำถึงความสำคัญของการรักษาความมั่นคงจากทุกการสื่อสารและทุกการเข้าถึง การบังคับใช้นโยบายความมั่นคงทำด้วยโค้ดบนพื้นฐานการให้สิทธิ์ต่ำที่สุดเท่าที่จำเป็น

(สถาปัตยกรรมแบบไร้ความเชื่อใจ)

ขั้นตอนการทำงานของแมชชีนเลิร์นนิงทั้งหมด การเรียนรู้แบบกึ่งสอนจะช่วยลดค่าใช้จ่ายและเวลาในการฝึกสอน และทำให้แมชชีนเลิร์นนิงคุ้มค่าและเปิดความเป็นไปได้กับผู้ใช้อีกประเภทหนึ่ง เราเริ่มที่จะเห็นการใช้งานของเทคนิคในอีกรูปแบบ ที่การสอนเกิดขึ้นแบบอ่อนๆ (weakly supervised) มากขึ้น ที่เครื่องเป็นผู้แปะป้ายให้กับข้อมูลเอง ซึ่งข้อมูลชุดนั้นจะน่าเชื่อถือน้อยกว่าข้อมูลที่ถูกกำกับโดยมนุษย์โดยตรง

การถ่ายทอดการเรียนรู้สำหรับ NLP

ทดลอง

เราเคยให้เทคนิคนี้อยู่ในกลุ่มประเมินในเรดาร์ฉบับก่อน และนวัตกรรมในโลกของการประมวลผลภาษาธรรมชาติ (NLP) ยังคงเติบโตไปข้างหน้าด้วยความเร็วสูง ซึ่งเราสามารถใช้ประโยชน์จากนวัตกรรมเหล่านี้กับหลายๆ โครงการของเรา ซึ่งต้องขอบคุณเทคนิค **การถ่ายทอดการเรียนรู้สำหรับ NLP** เมื่อเทียบผลการทดสอบกับมาตรฐานการวัดผล GLUE (ซึ่งเป็นชุดรวมการทดสอบเกี่ยวกับความเข้าใจภาษา) เราได้เห็นถึงพัฒนาการที่ดีขึ้นอย่างก้าวกระโดดในช่วงสองปีที่ผ่านมา จากคะแนนเฉลี่ยในวันที่เปิดตัวที่ 70.0 จนมาถึงเดือนเมษายนปี 2020 ที่คะแนนของกลุ่มผู้นำได้กันเกิน 90.0 ไปแล้ว แล้วในบรรดาโครงการที่เกี่ยวข้องกับ NLP ของเราทั้งหลาย การนำโมเดลที่ถูกฝึกสอนมาล่วงหน้าของ ELMo, BERT, ERNIE มาทำการปรับปรุงต่อให้เหมาะกับความต้องการของแต่ละโครงการมีพัฒนาการที่ดีกว่ามากเมื่อเทียบกับการใช้โมเดลแบบอื่น

ใช้กระบวนการหรือวิธีที่รับรองการทำงานแบบทางไกลอย่างเป็นธรรมชาติ

ทดลอง

ทีมที่ทำงานแบบต่างสถานที่กันมีได้หลายรูปแบบ เราพบว่าการได้ทำงานร่วมกันในสถานที่เดียวกันแบบ 100% กลายเป็นเรื่องพิเศษไปเสียแล้ว ทีมของเราส่วนใหญ่จะอยู่ในรูปแบบกระจายกันอยู่ในหลายสถานที่ทำงาน หรือไม่ก็มีสมาชิกบางคนที่ต้องทำงานนอกสถานที่อยู่เสมอ ดังนั้น**การใช้กระบวนการหรือวิธีที่รับรองการทำงานแบบทางไกลอย่างเป็นธรรมชาติ** เป็นทางเลือกแรกจะช่วยให้เกิดความคล่องตัวและให้ประสิทธิภาพดีกว่า โดยเริ่มจากการที่ทุกคนในทีมสามารถเข้าถึงระบบต่างๆ ที่จำเป็นจากทางไกลได้ ยิ่งกว่านั้น การใช้เครื่องมืออย่างเช่น Visual Studio Live Share, MURAL หรือ Jamboard อยู่ตลอด จะเปลี่ยนพฤติกรรมการทำเวิร์คช็อปและการแพร่โปรแกรมมิ่งให้อยู่ในรูปแบบออนไลน์แทน จนทำให้สิ่งเหล่านี้กลายเป็นกิจวัตรประจำของทีมไป ซึ่งดีกว่าการใช้มันเป็นครั้งคราวที่เราไม่คุ้นชินแล้วลดทอนประสิทธิภาพการทำงานลง การรับรองการทำงานแบบทางไกลอย่างเป็นธรรมชาตินั้นไปไกลกว่าการยกเอาการทำงานแบบเดิมเข้าสู่โลกดิจิทัลตรงๆ เพราะมันเปิดและยอมรับการสื่อสารแบบอะซิงโครนัส ที่คนอาจทำงานไม่พร้อมกัน มันต้องการวินัยที่สูงขึ้นในการจดบันทึกการตัดสินใจใดๆ และการที่ทุกคนประชุมแบบทางไกลอยู่เสมอแม้จะอยู่ในสถานที่เดียวกันก็เป็นอีกวิธีที่ทีมของเราเลือกปฏิบัติเพื่อให้สอดคล้องกับการทำงานจากที่ใดก็ได้

สถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero Trust Architecture)

ทดลอง

ณ วันนี้ที่ภาพรวมเทคโนโลยีของหลายหน่วยงานมีความซับซ้อนมากขึ้น จากสินทรัพย์ทางเทคโนโลยีซึ่งอยู่ในรูปแบบที่ต่างกันไป ไม่ว่าจะเป็นข้อมูล ฟังก์ชัน โครงสร้างพื้นฐาน และผู้ใช้ สิ่งเหล่านี้กระจายตัวในแหล่งที่มีขอบเขตความมั่นคงที่แตกต่างกัน ซึ่งอาจจะอยู่ภายในองค์กร หรืออยู่กับผู้ให้บริการคลาวด์รายต่างๆ และบริการซอฟต์แวร์ผ่านคลาวด์ ปัจจัยเหล่านี้ได้ขับเคลื่อนให้องค์กรต้องมีสถาปัตยกรรมของระบบและการวางแผนด้านความมั่นคงขององค์กรแบบใหม่ที่ต่างจากเดิมอย่างสิ้นเชิง จากวิธีการจัดการนโยบายความมั่นคงแบบเดิมๆ ที่กำหนดกันเป็นค่าคงที่ เปลี่ยนแปลงไม่บ่อย และบังคับใช้อย่างหยาบๆ โดยอาศัยการกำหนดขอบเขตความเชื่อใจและการควบคุมด้วยเน็ตเวิร์ก ไปสู่วิธีการจัดการแบบใหม่ที่การเปลี่ยนแปลงค่าเกิดขึ้นได้อย่างยืดหยุ่นตลอดเวลา และการบังคับใช้นโยบายทำได้อย่างละเอียด โดยอาศัยการให้สิทธิ์การเข้าถึงแบบชั่วคราว **สถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero trust architecture (ZTA))** เป็นกลยุทธ์และการเดินทางขององค์กร ไปสู่การทำให้แนวคิดความมั่นคงแบบไร้ความเชื่อใจเป็นจริงสำหรับทุกสินทรัพย์ ไม่ว่าจะเป็นเครื่องใช้อุปกรณ์ โครงสร้างพื้นฐาน เซอร์วิส ข้อมูล และผู้ใช้ รวมไปถึงการสร้างหลักปฏิบัติด้านความมั่นคงต่างๆ ภายในองค์กร เช่น บังคับให้ทุกการเข้าถึงและทุกการสื่อสารต้องอยู่ภายใต้กรอบความมั่นคงเสมอโดยไม่คำนึงถึงที่ตั้งของเครือข่าย การบังคับใช้นโยบายความมั่นคงทำด้วยโค้ดบนพื้นฐานการให้สิทธิ์ต่ำที่สุดเท่าที่จำเป็นและละเอียดที่สุดเท่าที่จะเป็นไปได้ การมีระบบสอดส่องเฝ้าระวังตลอดเวลา และการจัดการภัยคุกคามที่เข้ามาได้อย่างอัตโนมัติ หลายๆ หัวข้อในเรดาร์ของเราสะท้อนให้เห็นถึงเทคนิคต่างๆ ที่ช่วยให้เกิดสถาปัตยกรรมนี้ ได้แก่ นโยบายความมั่นคงด้วยโค้ด เทคนิคไซด์คาร์ สำหรับสร้างความมั่นคงให้จุดเชื่อมต่อ และ BeyondCorp ถ้าคุณกำลังอยู่ระหว่างการเดินทางไปสู่ ZTA ลองศึกษาบทความตีพิมพ์ของกูเกิลที่ BeyondProd และของ NIST ZTA เพื่อศึกษาเพิ่มเติมถึงหลักแนวคิด วิธีการปรับตัวเข้าสู่สถาปัตยกรรมนี้ และชุดเทคโนโลยีต่างๆ ที่ช่วยสนับสนุน

ดาต้าเมช (Data mesh)

ประเมิน

ดาต้าเมช (Data mesh) เป็นมุมมองการออกแบบสถาปัตยกรรมและโครงสร้างองค์กรที่ท้าท้ายความเชื่อแบบดั้งเดิมว่า เราต้องรวมศูนย์ข้อมูลเชิงวิเคราะห์ก่อนเสมอถึงจะใช้มันได้ โดยข้อมูลทั้งหมดต้องถูกนำมากองรวมกันในที่เดียว หรือถูกจัดการโดยทีมจัดการข้อมูลที่มาจากส่วนกลางขององค์กร ดาต้าเมช กล่าวไว้ว่า การจะนำบิ๊กดาต้าไปต่อยอดเพื่อสร้างนวัตกรรมใหม่ได้นั้น กรรมสิทธิ์ของข้อมูลต้องถูกกระจายตัวไปสู่บรรดาเจ้าของข้อมูลในแต่ละโดเมนเสียก่อน ซึ่งคือผู้ที่มีส่วนได้ส่วนเสียในการให้บริการข้อมูลนั้นเสมือนข้อมูลในโดเมนเป็นดั่งผลิตภัณฑ์ของตนเอง (ร่วมกับการสนับสนุนที่ดีจากแพลตฟอร์มข้อมูล เพื่อให้การดึงข้อมูลทำได้ง่าย ผู้รับสามารถให้บริการตัวเองได้ โดยการตัดความซับซ้อนทางเทคนิคที่เป็นไปได้ออกไป) รวมทั้งต้องการวิธีการบริหารปกครองข้อมูลร่วมกันแบบใหม่ที่ทุกสิ่งเป็นไปอย่างอัตโนมัติ เพื่อเปิดให้ข้อมูลต่างโดเมนสามารถแลกเปลี่ยนทำงานร่วมกันได้ ฉะนั้นแล้ว ด้วยการกระจายข้อมูล บวกกับการมีความสามารถในการทำงานร่วมกันที่ดีและการให้ความสำคัญกับประสบการณ์ของผู้ใช้ข้อมูล สิ่งเหล่านี้เป็นกุญแจสำคัญสู่การทำให้เกิดนวัตกรรมใหม่อย่างเท่าเทียมทั่วถึงจากการใช้ข้อมูลหากองค์กรของคุณมีโดเมนจำนวนมากอยู่แล้ว ที่มีระบบหรือทีมทำหน้าที่ผลิตข้อมูลอยู่มากมาย หรือมีกรณีการใช้และการเข้าถึงข้อมูลที่หลากหลาย ที่เกิดจากการขับเคลื่อนของข้อมูล เราแนะนำให้คุณลองประเมินการใช้งานดาต้าเมชดู การจะสร้างองค์กรและสถาปัตยกรรมในรูปแบบดาต้าเมชได้ ต้องการการลงทุนในการสร้างดาต้าแพลตฟอร์มที่สามารถให้บริการข้อมูลด้วยตนเอง และต้องปรับองค์กรเพื่อรับกับโดเมน ให้เกิดโครงสร้างที่กรรมสิทธิ์ของผลิตภัณฑ์ข้อมูลสามารถถือกันแบบระยะยาว รวมถึงต้องการโครงสร้างรางวัลที่กระตุ้นให้เกิดการสร้างและใช้งานผลิตภัณฑ์ข้อมูลอีกด้วย

เอกลักษณ์แบบกระจายศูนย์

ประเมิน

ตั้งแต่อินเตอร์เน็ตเกิดขึ้นมา โลกของเทคโนโลยีก็ถูกเร่งให้มีวิวัฒนาการเข้าสู่แนวคิดแบบกระจายตัวมาโดยตลอด แม้ว่าโปรโตคอลต่างๆ อย่าง HTTP หรือรูปแบบสถาปัตยกรรมอย่าง ไมโครเซอร์วิส หรือ ดาต้าเมช ได้ช่วยให้ระบบกระจายตัวเกิดขึ้นอย่างมากมาย แต่วิธีจัดการเอกลักษณ์ (identity management) ในปัจจุบันยังคงถูกรวมศูนย์อยู่เหมือนเดิม การเกิดขึ้นของเทคโนโลยีบัญชีธุรกรรมแบบกระจายตัว (Distributed ledger technology, DLT) เป็นตัวจุดฉนวนให้แนวคิด เอกลักษณ์แบบกระจายศูนย์ เป็นจริงขึ้นมา ซึ่งในระบบการจัดการเอกลักษณ์แบบกระจายศูนย์ แต่ละสิ่งต้องมีอัตลักษณ์ไว้ระบุตัวตนเสมอและสามารถเลือกใช้รากความน่าเชื่อที่มีร่วมกันใดๆ ได้อย่างอิสระ ซึ่งต่างจากโลกทั่วไปที่ใช้ระบบจัดการเอกลักษณ์ที่มีพื้นฐานอยู่บนหน่วยงานกลางที่ได้รับความน่าเชื่อถือและระบบระเบียนต่างๆ เช่น เทคโนโลยีไดเรกทอรีเซอร์วิส ผู้ให้บริการรับรองสากล (CA) และหน่วยงานให้บริการชื่อโดเมน เป็นต้น เรารู้สึกตื่นเต้นที่เห็นการพัฒนาของ มาตรฐานระบบจัดการเอกลักษณ์แบบกระจายศูนย์ ซึ่งเป็นมาตรฐานสำคัญที่ทำให้ระบบเอกลักษณ์แบบกระจายศูนย์เป็นไปได้ ถึงแม้ระบบจัดการเอกลักษณ์แบบกระจายศูนย์ที่รองรับการขยายของข้อมูลยังมีอยู่น้อยมาก โดยเราเริ่มปรับใช้หลักคิดเดียวกันในสถาปัตยกรรม สำหรับผู้ที่สนใจการทดลองล่าสุดและการร่วมมือกันในอุตสาหกรรม สามารถเข้าไปที่ สมาคมเอกลักษณ์แบบกระจายศูนย์

การนิยามไปป์ไลน์ข้อมูลแบบประกาศ

ประเมิน

ไปป์ไลน์ของข้อมูลส่วนมากมักจะเขียนอยู่ในรูปแบบสคริปต์ขนาดใหญ่ ที่ไม่มากก็น้อยถูกเขียนในเชิงคำสั่งด้วยภาษา Python หรือ Scala ซึ่งในสคริปต์เต็มไปด้วยโค้ดที่อธิบายรายละเอียดวิธีการทำงานของแต่ละขั้นตอนที่ยุ่งเหยิงและยากต่อการดูแล ในอดีตเมื่อพบสถานการณ์ที่คล้ายกันเวลาสร้างชุดทดสอบด้วย Selenium นักพัฒนาได้คิดค้นการจัดการโค้ดในรูปแบบเพจออบเจกต์ (Page Object pattern) ขึ้นมา โดยในภายหลังเฟรมเวิร์คสำหรับสร้างชุดสอบที่ขับเคลื่อนจากพฤติกรรมที่เห็น (Behavior-Driven Development, BDD) ต่างก็ออกแบบมาโดยสามารถแยกขั้นตอนการทำงานออกเป็นส่วน เพื่อเรียงร้อยมันเข้าด้วยกันทีหลัง บางทีมของเรากำลังทดลองนำแนวคิดเดียวกันมาปรับใช้กับวิศวกรรมข้อมูลอยู่ จึงเกิด **การนิยามไปป์ไลน์ข้อมูลแบบประกาศ** ขึ้นมา ที่การนิยามถูกแยกออกมาต่างหาก ซึ่งอาจอยู่ในรูปแบบไฟล์ YAML ที่ในนั้นจะระบุเพียงขั้นตอนและลำดับการทำงานเท่านั้น โดยอ้างอิงไปที่ชุดข้อมูลขาเข้าและขาออกที่ใช้ และจะอ้างอิงไปที่สคริปต์ของโค้ดเฉพาะกรณีที่ตรรกะมีความซับซ้อนสูงเท่านั้น เราพบว่า A La Mode เป็นเครื่องมือโอเพนซอร์สตัวแรกที่เข้ามาจัดการเรื่องดังกล่าว

DeepWalk

ประเมิน

DeepWalk เป็นอัลกอริทึมสำหรับช่วยนำแมชชีนเลิร์นนิงมาใช้กับกราฟ ความยากอย่างหนึ่งเมื่อต้องทำงานกับชุดข้อมูลในรูปแบบกราฟอยู่ที่วิธีการสกัดเอาฟีเจอร์ออกจากกราฟ ซึ่ง DeepWalk สามารถช่วยได้ มันทำงานโดยใช้อัลกอริทึม SkipGram เพื่อสร้างข้อมูลฝังตัวของโหนด (node embedding) โดยมองกราฟเป็นเหมือนภาษาธรรมชาติ โดยให้โหนดแต่ละโหนดเปรียบเสมือนคำแต่ละคำที่ไม่ซ้ำกันในภาษานั้น การสร้างรูปประโยคจึงเกิดจากการเดินแบบสุ่มบนกราฟแบบจำกัดระยะทาง ข้อมูลฝังตัวที่ได้เหล่านี้จะสามารถนำไปใช้ต่อกับโมเดลแมชชีนเลิร์นนิงที่หลากหลาย DeepWalk เป็นหนึ่งในเทคนิคที่พวกเรากำลังทดลองใช้ในหลายๆ โครงการที่ต้องทำงานแมชชีนเลิร์นนิงบนข้อมูลในรูปแบบกราฟ

การจัดการระบบที่จดจำสถานะข้อมูลด้วยเครื่องมือบริหารควบคุมคอนเทเนอร์

ประเมิน

เราขอเตือนให้ระวัง **การจัดการระบบที่จดจำสถานะข้อมูลด้วยเครื่องมือบริหารควบคุมคอนเทเนอร์** อย่างเช่น Kubernetes ระบบฐานข้อมูลบางตัวไม่ได้ถูกออกแบบมาเพื่อทำงานภายใต้สภาพแวดล้อมที่ตัวเองจะถูกบริหารควบคุม เช่น มันไม่เคยคาดหวังว่าจู่ๆ จะถูกสั่งให้หยุดการทำงานลงในทันที หรือต้องย้ายไปทำงานบนเครื่องอื่นได้ ซึ่งการสร้างเซอร์วิสประเภทฐานข้อมูลที่มีเสถียรภาพสูงที่ทำงานได้ตลอดเวลานั้นไม่ใช่เรื่องเล็ก ดังนั้นเรายังคงแนะนำให้ติดตั้งใช้งานมันบนเครื่องเวอร์ชวลแมชชีนหรือเครื่องเปล่าตรงๆ มากกว่าพยายามฝืนติดตั้งมันในแพลตฟอร์มบริหารควบคุมคอนเทเนอร์

การรันบิลด์ในสนามซ้อม

ประเมิน

แม้เราจะสนับสนุนอย่างแข็งขันให้ใช้วิธีการเชื่อมต่อกันอย่างต่อเนื่อง มากกว่าการใช้กระบวนการทำงานแบบ Gitflow ที่การเชื่อมต่อขาดความต่อเนื่อง เราก็ทราบดีว่าหากทีมมีขนาดใหญ่เกินไปหรือในทีมขาดวินัยที่จะรันชุดทดสอบบนเครื่องตัวเองเสียก่อน การจะคอมมิตตรงๆ ไปที่มาสเตอร์บรานช์ หรือการรันไปป์ไลน์บนมาสเตอร์บรานช์นั้นจะทำให้ประสิทธิภาพการทำงานลดลง เพราะมันทำให้กระบวนการบิลด์ช้า ไม่น่าเชื่อถือ และหากมีบิลด์เสียนานๆ อาจบล็อคไม่ให้นักพัฒนาคนอื่นทำงานต่อได้ ซึ่งแทนที่จะแก้ปัญหาที่ต้นเหตุ (เช่น ทำให้ชุดทดสอบทำงานได้เร็วขึ้น หรือทำให้สามารถทดสอบได้บนเครื่องตัวเอง หรือแก้ไขด้วยการเปลี่ยนสถาปัตยกรรมออกจากโมโนลิธ) หลายทีมเลือกที่จะใช้การแตกฟีเจอร์บรานช์ออกมาเพื่อหลบเลี่ยงปัญหาเหล่านี้ ซึ่งอย่างที่ทราบกันว่าเราไม่สนับสนุนให้ใช้ฟีเจอร์บรานช์มาตลอด เพราะว่ามันทำให้ทีมเสียแรงไปกับการแก้ปัญหาเวลาเกิดความทับซ้อนของงาน ใช้เวลานานกว่าจะได้ผลลัพธ์ และอาจทำให้เผลอสร้างข้อผิดพลาดระหว่างการแก้ไขความทับซ้อน ดังนั้นเราจึงอยากเสนอให้พิจารณา **การรันบิลด์ในสนามซ้อม (preflight build)** เป็นอีกทางเลือก ซึ่งมันเป็นวิธีที่ทำงานด้วยการสร้าง Pull request สำหรับบรานช์อายุสั้น โดยบรานช์จะเปิดรับทุกคอมมิตและมีอายุอยู่แค่ระหว่างช่วงการบิลด์เท่านั้น และเพื่อทำให้กระบวนการนี้เป็นไปอย่างอัตโนมัติ เราพบว่ามีระบบหุ่นยนต์ อย่าง Bors ที่สามารถช่วยให้การรวมบรานช์ไปที่มาสเตอร์และการลบบรานช์ที่บิลด์สำเร็จแล้วเกิดขึ้นอย่างอัตโนมัติ เรากำลังประเมินกระบวนการทำงานแบบนี้อยู่ ซึ่งเราก็แนะนำให้คุณลองประเมินดูเช่นเดียวกัน แต่โปรดอย่าใช้มันเพื่อแก้ผิดปัญหา เพราะมันอาจนำไปสู่การใช้บรานช์อย่างผิดๆ และก่อให้เกิดผลเสียมากกว่าผลดี

เทคนิค

เทคโนโลยีบัญชีธุรกรรมแบบกระจายตัว (DLT) ที่เป็นรากฐานของ BitCoin ได้จุดฉนวนให้เกิดแนวคิดของการระบุเอกลักษณ์แบบกระจายศูนย์

(เอกลักษณ์แบบกระจายศูนย์)

ความยากอย่างหนึ่งเมื่อต้องทำงานกับชุดข้อมูลในรูปแบบกราฟอยู่ที่วิธีการสกัดเอาฟีเจอร์ออกจากกราฟ ซึ่ง DeepWalk สามารถช่วยได้

(DeepWalk)

เทคนิค

โดยทั่วไปแล้วบันทึกเหตุการณ์เพื่อการเฝ้าสังเกตการณ์ในเชิงเทคนิคให้ข้อมูลไม่พอเพียงสำหรับใช้ทำความเข้าใจลูกค้าในเชิงลึก

(การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ)

การทดสอบแบบบันทึกภาพมีประโยชน์อย่างปฏิเสธไม่ได้เมื่อใช้สำหรับทดสอบระบบเก่าที่ล้าสมัย แต่มันไม่ควรถูกใช้เป็นกลไกหลักสำหรับทดสอบระบบใดๆ

(การทดสอบแบบบันทึกภาพแต่เพียงอย่างเดียว)

การย้ายขึ้นคลาวด์แบบยกวางทั้งระบบ

เฝ้าระวัง

มันน่าประหลาดใจที่แม้ว่าจะผ่านประสบการณ์มามากกว่า 10 ปีในการย้ายระบบไปบนคลาวด์ เรายังรู้สึกจำเป็นที่จะต้องพูดถึงแนวคิด **การย้ายขึ้นคลาวด์แบบยกวางทั้งระบบ (Cloud lift and shift)** ที่คลาวด์ถูกมองเป็นเพียงแค่โซลูชันสำหรับโฮสต์งานเท่านั้น โดยที่เรื่องอื่นๆ ที่มีอยู่แล้ว เช่น ระบบสถาปัตยกรรม แนวปฏิบัติด้านความมั่นคง วิธีการดูแลระบบของไอที ยังคงเป็นแบบเดิมต่อไป แนวคิดนี้ทำให้พลาดประโยชน์จากความคล่องตัวและนวัตกรรมที่ควรจะได้รับจากการใช้คลาวด์ การจะย้ายระบบไปบนคลาวด์ต้องการการเปลี่ยนแปลงอย่างตั้งใจในหลายๆ ด้าน เพื่อจะก้าวไปสู่องค์กรที่ใช้คลาวด์ได้อย่างเป็นธรรมชาติ ยกตัวอย่างเช่น สถาปัตยกรรมของระบบเป็นหนึ่งในเสาหลักสำคัญเพื่อให้เกิดการส่งมอบที่คล่องตัวก็ต้องมีการเปลี่ยนแปลง ทั้งนี้ขึ้นอยู่กับสถานการณ์ที่แตกต่างกันของแต่ละองค์กร ผลลัพธ์ของการย้ายไปบนคลาวด์อาจไปตกอยู่ตรงไหนสักแห่งในสเปกตรัม ระหว่างการได้คลาวด์แบบยกวางหรือการได้คลาวด์แบบธรรมชาติ วิธีการย้ายแบบยกวางทั้งระบบโดยใช้คอนเทเนอร์ ไปบนคลาวด์นั้นน่าเย้ายวนเป็นอย่างมาก แม้มันจะช่วยร่นเวลาในการย้ายได้จริง แต่วิธีนี้ไม่ตอบโจทย์ในการสร้างความคล่องตัวให้กับองค์กร และไม่ทำให้การส่งมอบทั้งฟีเจอร์และคุณค่าดีขึ้นเลย ส่วนวิธีการบริหารความมั่นคงขององค์กรบนคลาวด์ก็ใช้พื้นฐานที่แตกต่างอย่างสิ้นเชิงเมื่อเทียบกับการบริหารความมั่นคงแบบดั้งเดิมที่มักใช้ไฟร์วอลล์และการแบ่งเขตโซนเท่านั้น เพราะต้องการการเปลี่ยนแปลงเพื่อมุ่งสู่การสร้างสถาปัตยกรรมแบบไร้ความเชื่อใจ (Zero trust architecture) ส่วนวิธีการดูแลระบบของไอทีก็ควรจะมีการเปลี่ยนรูปแบบไปเช่นกัน เพื่อให้มุ่งสู่การมีแพลตฟอร์มที่สามารถให้บริการคลาวด์ได้อย่างเป็นอัตโนมัติ ที่ทีมสามารถให้บริการตนเองได้อย่างปลอดภัย ส่งเสริมให้ทีมรับผิดชอบต่องานดูแลระบบด้วยตนเองมากขึ้น และมีอำนาจตัดสินใจได้อย่างอิสระ สุดท้ายแต่ไม่ท้ายสุด องค์กรต้องสร้างรากฐานเพื่อรองรับการเปลี่ยนแปลงที่เข้ามาอย่างต่อเนื่อง อย่างเช่น การสร้างไปป์ไลน์ที่ทดสอบแอปพลิเคชันและโครงสร้างพื้นฐานไว้เป็นส่วนหนึ่งของกระบวนการย้ายขึ้นคลาวด์ วิธีนี้จะส่งผลให้ได้ระบบที่คงทนกว่า ถูกแบ่งออกเป็นส่วนๆ อย่างดี และให้แนวทางแก่องค์กรในการวิวัฒน์และปรับปรุงระบบให้ดีขึ้นต่อไป

การย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด

เฝ้าระวัง

เราพบว่ามีหลายองค์กรมากขึ้น ที่ต้องการแทนที่ระบบเก่าที่ล้าสมัยด้วยระบบใหม่เพื่อรองรับกับความต้องการของลูกค้าที่สูงขึ้น (ทั้งลูกค้าภายในและภายนอกองค์กร) หนึ่งในแบบแผนที่ไม่ควรแต่มักทำกันประจำ คือ **การย้ายไประบบใหม่โดยคงฟีเจอร์เก่าไว้ทั้งหมด (legacy migration feature parity)** ซึ่งเกิดจากความเสียดายฟีเจอร์ในระบบเก่านั่นเอง เรากลับเห็นว่า การตัดสินใจเช่นนั้นเป็นการเสียโอกาสอย่างมาก เนื่องจากระบบเก่ามักจะบวมขึ้นตามกระบวนการทางธุรกิจที่วิวัฒน์เปลี่ยนแปลงตามเวลา ทำให้มีฟีเจอร์หลายส่วนไม่ได้ถูกใช้งานแล้ว (ซึ่งอาจจะสูงถึง 50% ของฟีเจอร์ทั้งหมด ตามรายงานของ Standish Group ปี 2014) การแทนที่ฟีเจอร์ทั้งหมดจึงเป็นการสูญเปล่าโดยใช่เหตุ คำแนะนำของเราสำหรับเรื่องนี้คือ ลองโน้มน้าวลูกค้าของคุณให้หยุดคิดและพิจารณา ทำความเข้าใจความต้องการของผู้ใช้ในปัจจุบันให้ดี และจัดลำดับความสำคัญของความต้องการเหล่านั้นใหม่ให้สอดคล้องกับผลลัพธ์ทางธุรกิจ ด้วยตัวชี้วัดที่เหมาะสม ซึ่งคำแนะนำนี้มักพูดง่ายแต่ทำได้ยาก เพราะแทนที่จะนำฟีเจอร์เก่ามาใช้ตรงๆ บริษัทต้องลงทุนทำวิจัยพฤติกรรมผู้ใช้ และประยุกต์เทคนิควิธีพัฒนาผลิตภัณฑ์แบบสมัยใหม่เข้าไป

การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ

เฝ้าระวัง

เมื่อหลายปีก่อนแพลตฟอร์มสำหรับการเก็บรวมรวมบันทึกเหตุการณ์แบบใหม่ได้ถือกำเนิดขึ้น ที่สามารถจัดเก็บและค้นหาบันทึกเหตุการณ์จำนวนมากเพื่อค้นพบเทรนด์และข้อมูลเชิงลึกจากข้อมูลการทำงานของระบบ ซึ่งมี Splunk เป็นหนึ่งในเครื่องมือที่โดดเด่นที่สุดในด้านนี้ ด้วยความที่แพลตฟอร์มประเภทนี้ช่วยให้มองเห็นภาพรวมของความมั่นคงและการทำงานของระบบในทุกส่วน จึงกลายเป็นเครื่องมือที่ผู้ดูแลระบบและนักพัฒนาพึ่งพิงจนขาดไม่ได้ กระแสการใช้งานนี้ได้แพร่ไปสู่ฝั่งธุรกิจด้วยเช่นกันจนนำมาซึ่งความคิดที่ว่า เราน่าจะใช้**การเก็บรวบรวมบันทึกเหตุการณ์เพื่อการวิเคราะห์เชิงธุรกิจ**ได้ด้วยเหมือนกัน อย่างไรก็ตามความต้องการทางธุรกิจมักจะล้ำหน้าและเปลี่ยนแปลงอย่างรวดเร็วไปกว่าที่ความสามารถของเครื่องมือประเภทนี้จะรองรับไหว เพราะโดยทั่วไปแล้วบันทึกเหตุการณ์เพื่อการเฝ้าสังเกตการณ์ในเชิงเทคนิคให้ข้อมูลไม่พอเพียงสำหรับใช้ทำความเข้าใจลูกค้าในเชิงลึก หากคุณต้องการที่จะวิเคราะห์หาข้อมูลของลูกค้าอย่างจริงจัง เราแนะนำให้ใช้เครื่องมือและตัวชี้วัดที่สร้างขึ้นมาเพื่องานประเภทนี้โดยเฉพาะ หรือออกแบบระบบให้รองรับการสังเกตการณ์โดยธรรมชาติ โดยใช้วิธีการขับเคลื่อนพฤติกรรมจากเหตุการณ์ที่เกิดขึ้น (event-driven appraoch) ซึ่งวิธีนี้ทำให้เหตุการณ์ทั้งในแง่ธุรกิจและการใช้งานจะถูกรวบรวมและจัดเก็บด้วยกลวิธีที่สามารถนำเหตุการณ์เหล่านี้กลับมาฉายซ้ำหรือประมวลผลด้วยเครื่องมือที่สร้างขึ้นมาเพื่อวัตถุประสงค์นี้โดยเฉพาะ

GitFlow และการสร้างบรานช์ที่มีช่วงชีวิตยาวนาน

เฝ้าระวัง

เราได้ชูประเด็นปัญหาเรื่อง **GitFlow และการสร้างบรานช์ที่มีช่วงชีวิตยาวนาน** ไว้เมื่อห้าปีก่อน เพราะโดยแก่นแล้วการสร้างบรานช์ที่มีช่วงชีวิตยาวนานนั้นขัดแย้งกับแนวคิดที่ว่าเราควรผสานโค้ดกันอย่างสม่ำเสมอ ซึ่งจากประสบการณ์ของเรา การผสานโค้ดอย่างต่อเนื่องนั้นเป็นวิธีการพัฒนาซอฟแวร์ที่ดีกว่าแนวทางอื่นๆ หลังจากนั้นเราได้เพิ่มคำเตือนไปยังทีมที่ใช้ Gitflow เพราะเรามักจะเห็นมันถูกใช้งานควบคู่กับการสร้างบรานช์ที่มีช่วงชีวิตยาวนานอยู่ตลอด ในปัจจุบันเรายังคงเห็นทีมที่ตั้งเป้าที่จะพัฒนาระบบเว็บไซต์โดยใช้การส่งมอบอย่างต่อเนื่องนั้นถูกดึงดูดเข้าสู่การใช้ท่านี้ เรารู้สึกยินดีที่ได้เห็นผู้เผยแพร่ GitFlow ได้เพิ่มข้อความลงในบทความของเขาที่อธิบายว่า Gitflow ไม่ได้ถูกสร้างมาเพื่อจุดประสงค์ดังกล่าว

การทดสอบแบบบันทึกภาพแต่เพียงอย่างเดียว

เฝ้าระวัง

การทดสอบแบบบันทึกภาพ (snapshot testing) มีประโยชน์อย่างปฏิเสธไม่ได้เมื่อใช้สำหรับทดสอบระบบเก่าที่ล้าสมัย โดยใช้ยืนยันว่าการแก้ไขเปลี่ยนแปลงใดๆ ยังคงให้ผลลัพธ์การทำงานเหมือนที่บันทึกไว้ อย่างไรก็ตามเราพบผลเสียจากการใช้เทคนิคนี้อย่างไม่ถูกต้อง โดยการใช้**การทดสอบแบบบันทึกภาพแต่เพียงอย่างเดียว**เป็นกลไกหลักสำหรับทดสอบระบบใดๆ จากที่การทดสอบประเภทนี้จะตรวจที่ผลลัพธ์สุดท้ายว่าให้ผลของ DOM เหมือนกับที่บันทึกไว้หรือไม่เท่านั้นโดยไม่ได้มุ่งทดสอบที่พฤติกรรมว่าทำงานอย่างไร ฉะนั้นมันจึงค่อนข้างเปราะบางและขาดความน่าเชื่อถือ และเป็นการส่งเสริมหลักปฏิบัติที่ไม่ดีที่คิดแบบมักง่ายว่า “ก็แค่ลบภาพที่บันทึกทิ้งแล้วบันทึกใหม่อีกครั้ง” ซึ่งวิธีที่ดีกว่าคือการทดสอบตรรกะและพฤติกรรมของคอมโพเนนท์โดยจำลองจากการใช้งานของผู้ใช้ โดยวิธีคิดแบบนี้ถูกโปรโมตให้ใช้ด้วยเครื่องมือในตระกูล Testing Library

TECHNOLOGY RADAR Vol. 22

แพลตฟอร์ม

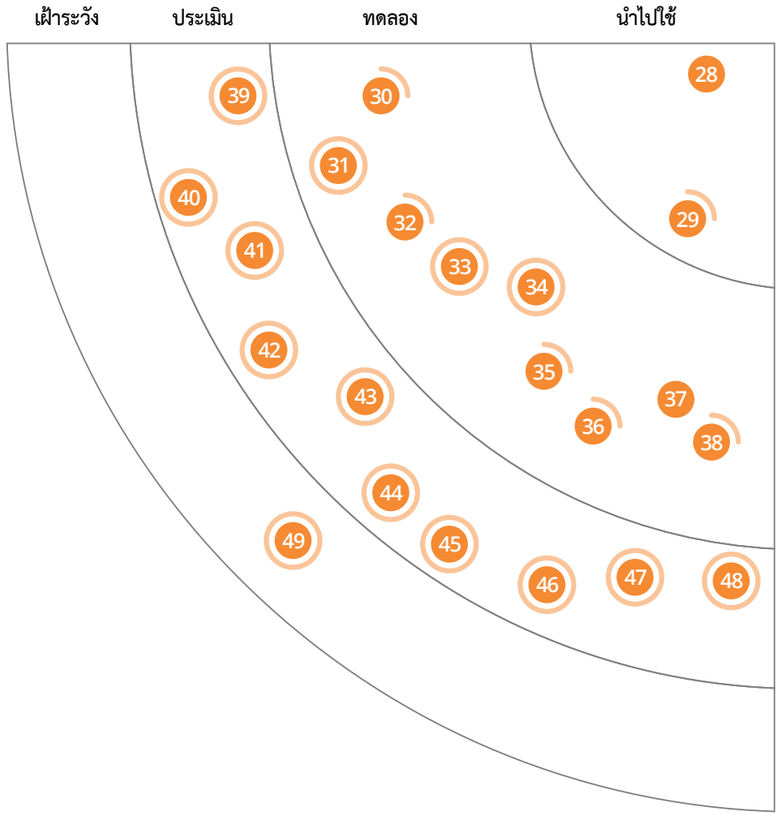
แพลตฟอร์ม

.NET Core
นำไปใช้

ที่ผ่านมาเราได้ยกให้ **.NET Core** อยู่ในกลุ่มควรนำไปใช้ไป ซึ่งบ่งบอกถึงการเป็นทางเลือกแรกของเราสำหรับใช้ในโครงการที่เป็น .NET ในฉบับนี้เราอยากจะยกมันมาพูดถึงอีกครั้งจากการเปลี่ยนแปลงที่เกิดขึ้นอย่างมาก เพราะนับตั้งแต่เวอร์ชัน 3._x_ ที่ออกมาปีที่แล้ว ได้มีฟีเจอร์มากมายถูกถ่ายโอนจาก .NET Framework มาสู่ .NET Core เพราะจากประกาศของไมโครซอฟต์เองที่บอกว่า นี่จะเป็นเวอร์ชันสุดท้ายของ .NET Framework ก็เป็นการเน้นย้ำว่า .NET Core คืออนาคตของ .NET ที่ผ่านมาไมโครซอฟต์ได้ทุ่มเทอย่างมากเพื่อทำให้ .NET Core สามารถทำงานร่วมกับคอนเทเนอร์ได้อย่างเป็นมิตร ซึ่งโครงการส่วนใหญ่ของเราที่ใช้ .NET Core ได้ทำงานอยู่บนระบบปฏิบัติการลีนุกซ์และส่วนมากอยู่ในรูปแบบคอนเทเนอร์แล้ว ส่วน .NET 5 ที่กำลังจะออกมาก็เป็นที่คาดหวังอย่างมาก ซึ่งเราก็ตั้งหน้าตั้งตารออยู่เช่นกัน

Istio
นำไปใช้

หากคุณกำลังสร้างหรือบริหารสถาปัตยกรรมไมโครเซอร์วิสขนาดใหญ่ภายใต้แพลตฟอร์มอย่าง Kubernetes อยู่ละก็ ควรอย่างยิ่งที่ต้องนำ เซอร์วิสเมช (service mesh) มาใช้เพื่อบริหารดูแลความต้องการประเภทที่กระจายตัวไปทั่วทั้งระบบ ในบรรดาตัวเลือกเซอร์วิสเมชทั้งหมด **Istio** เป็นหนึ่งในตัวเลือกที่ได้รับความนิยมอย่างสูง จากความสามารถที่หลากหลาย โดยสามารถให้บริการระบุที่อยู่ของเซอร์วิสอื่น บริหารการจราจรของข้อมูล รักษาความมั่นคงระหว่างเซอร์วิสด้วยกันหรือระหว่างต้นทางกับเซอร์วิส สามารถคอยเฝ้าสังเกตการณ์การทำงานของเซอร์วิส (ซึ่งรวมการติดตามร่องรอยระหว่างเซอร์วิส และการส่งสัญญาณต่างๆ ของเซอร์วิส) สามารถปล่อยอัพเดตเวอร์ชันใหม่แบบทะยอยเปลี่ยน และสามารถสร้างระบบที่ฟื้นฟูกลับสู่สภาวะปกติได้ ในเวอร์ชันล่าสุด Istio ได้ปรับปรุงประสบการณ์ผู้ใช้ด้วยการติดตั้งและการจัดวางแผงควบคุมที่ใช้งานง่าย จากประสบการณ์ในการส่งมอบระบบของเรา มันช่วยให้การสร้างระบบไมโครเซอร์วิสขนาดใหญ่ที่ต้องใช้การบริหารดูแลคุณภาพสูงมีความซับซ้อนน้อยลง แต่ต้องยอมรับว่าการจะบริหารดูแลระบบ Istio และ Kubernetes ด้วยตนเองนั้นจำเป็นต้องมีความรู้และทรัพยากรภายในที่มากพอ ซึ่งไม่เหมาะกับองค์กรที่ขาดความพร้อม



Anka
ทดลอง

Anka เป็นชุดเครื่องมือไว้สร้าง จัดการ แจกจ่าย บิลด์ และทดสอบ เวอร์ชวลแมชชีนสำหรับระบบปฏิบัติการ iOS และ MacOS มันนำประสบการณ์การใช้งานแบบ Docker มาสู่สภาพแวดล้อมที่เป็น MacOS เช่น เราสามารถสั่งสภาพแวดล้อมให้เปิดขึ้นในทันทีได้ มีชุดคำสั่งคอมมานด์ไลน์ให้สำหรับจัดการเวอร์ชวลแมชชีน มีรีจิสทรีสำหรับจัดการเวอร์ชัน และการติดป้ายกำกับให้กับเวอร์ชวลแมชชีนเพื่อการแจกจ่าย เราได้ลองใช้ Anka เพื่อสร้างคลาวด์ภายในองค์กรของลูกค้ามาแล้ว มันเป็นเครื่องมือที่คุ้มค่าแก่การพิจารณา หากคุณต้องการจำลองสภาพแวดล้อมของระบบปฏิบัติการ iOS และ MacOS ขึ้นมา

นำไปใช้
28. .NET Core
29. Istio

ทดลอง
30. Anka
31. Argo CD
32. Crowdin
33. eBPF
34. Firebase
35. Hot Chocolate
36. Hydra
37. OpenTelemetry
38. Snowflake

ประเมิน
39. Anthos
40. Apache Pulsar
41. Cosmos
42. Google BigQuery ML
43. JupyterLab
44. Marquez
45. Matomo
46. MeiliSearch
47. Stratos
48. Trillian

เฝ้าระวัง
49. การใช้งาน Node สำหรับทุกสิ่ง

แพลตฟอร์ม

แม้เทคโนโลยีนี้จะไม่ใช่เทคโนโลยีใหม่ แต่ปัจจุบันมันได้กลายเป็นส่วนสำคัญในโลกของไมโครเซอร์วิสที่อยู่ภายใต้การกำกับของตัวควบคุมคอนเทเนอร์

(eBPF)

การแยกออกจากกันอย่างชัดเจนระหว่างส่วนจัดการเอกลักษณ์และ OAuth2 เฟรมเวิร์คที่เหลือ ทำให้เป็นการง่ายที่จะเชื่อม Hydra เข้ากับระบบการยืนยันตัวตนที่มีอยู่แล้ว

(Hydra)

Argo CD

ทดลอง

แม้เราไม่อยากตัดสินเทคนิคอย่าง GitOps ว่าดีหรือไม่ แต่เราอยากจะพูดถึง Argo CD ในขอบเขตของการติดตั้งและติดตามการทำงานของแอปพลิเคชันลงบนแพลตฟอร์ม Kubernetes จากความสามารถของ Argo CD ที่ช่วยให้การติดตั้งแอปพลิเคชันทำได้ง่าย โดยการกำหนดสถานะของแอปพลิเคชันที่คาดหวังเอาไว้แล้วมันจะเปลี่ยนแปลงระบบปัจจุบันให้เข้าสู่สถานะนั้นอย่างอัตโนมัติ และการใช้มันเพื่อตรวจหาข้อบกพร่องและแก้ไขปัญหาที่เกิดจากการติดตั้ง รวมไปถึงการใช้มันเพื่อสืบค้นบันทึกเหตุการณ์และสถานะการติดตั้ง เราเลยอยากแนะนำให้คุณลองใช้ Argo CD แล้วคุณจะเห็นภาพว่ามีอะไรเกิดขึ้นบนคลัสเตอร์บ้าง คุณจะเห็นแบบเรียลไทม์ถึงการเกิดและตายของพ็อดต่างๆ เลยทีเดียว

Crowdin

ทดลอง

โครงการที่ต้องรองรับผู้ใช้งานได้หลายภาษาส่วนมากเริ่มต้นจากการพัฒนาฟีเจอร์ในภาษาใดภาษาหนึ่งก่อน แล้วค่อยจัดการแปลภาษาที่เหลือแบบออฟไลน์ผ่านการใช้อีเมลหรือสเปรดชีต แม้ว่ากระบวนการเหล่านี้จะง่ายและใช้ได้ดี แต่ความยุ่งยากอาจเกิดขึ้นได้ในเวลาอันรวดเร็ว คุณอาจจะต้องตอบคำถามเดิมซ้ำๆ กับผู้แปลในแต่ละภาษา และนั่นทำให้นักแปล นักพิสูจน์อักษร และนักพัฒนาซอฟต์แวร์เหนื่อยล้ากับการประสานงานระหว่างกันและกัน Crowdin เป็นหนึ่งในแพลตฟอร์มที่ช่วยปรับปรุงกระบวนการแปลให้เป็นระเบียบ ทีมพัฒนาซอฟต์แวร์สามารถมุ่งไปที่การพัฒนาฟีเจอร์ต่างๆ ต่อไป โดยที่แพลตฟอร์ม Crowdin จะเป็นตัวจัดการนำข้อความเข้าสู่ขั้นตอนการแปลภาษาแบบออนไลน์ให้อย่างอัตโนมัติ เราชอบที่ Crowdin มีฟีเจอร์ที่คอยกระตุ้นทีมให้ช่วยกันแปลภาษาอย่างสม่ำเสมอและต่อเนื่องแทนที่จะแปลงานเป็นชุดใหญ่ชุดเดียวตั้งแต่ต้นจนจบ

eBPF

ทดลอง

หลายปีที่ผ่านมาลีนุกซ์เคอร์เนลได้รวมเอาเวอร์ชวลแมชชีนสำหรับกรองเน็ตเวิร์กแพ็กเกต อย่าง **eBPF** (The extended Berkeley Packet Filter) มาในตัว ซึ่งทำให้สามารถแนบตัวกรองแพ็กเกตของ eBPF ไปที่ซอกเก็ตใดๆ ก็ได้ แต่ eBPF ทำได้มากกว่าการกรองแพ็กเกต มันยังอนุญาตให้ผู้ใช้สามารถกำหนดสคริปต์ของตนเอง โดยที่สคริปต์นั้นจะถูกให้ทำงานในเคอร์เนลทุกครั้งตามเหตุการณ์ต่างๆ ที่เกิดขึ้น โดยวิธีนี้จะสิ้นเปลืองทรัพยากรของเครื่องน้อยมาก แม้เทคโนโลยีนี้จะไม่ใช่เทคโนโลยีใหม่ แต่ปัจจุบันมันได้กลายเป็นส่วนสำคัญในโลกของไมโครเซอร์วิสที่อยู่ภายใต้การกำกับของตัวควบคุมคอนเทเนอร์ เพราะการสื่อสารระหว่างเซอร์วิสด้วยกันในระบบลักษณะนี้มักจะซับซ้อนมาก ทำให้ยากที่จะเชื่อมปัญหาด้านประสิทธิภาพหรือความหน่วงกลับไปที่ API ใดๆ ปัจจุบันเราจึงเห็นเครื่องมือต่างๆ ที่ถูกปล่อยออกมามีสคริปต์ eBPF ของตัวเองให้ด้วย เพื่อใช้สำหรับเก็บและแสดงภาพการจราจรของเน็ตเวิร์กแพ็กเกตที่เกิดขึ้น หรือใช้รายงานผลการบริโภคทรัพยากรของ CPU จากความนิยมของ Kubernetes ทำให้เราได้เห็นวิธีการบังคับใช้ความมั่นคงและการดัดแปลงแพ็กเกตยุคใหม่ ผ่านเทคนิคการใช้สคริปต์ eBPF เพื่อช่วยลดความซับซ้อนของการติดตั้งระบบไมโครเซอร์วิสขนาดใหญ่ลง

Firebase

ทดลอง

Firebase จาก Google ได้รับการพัฒนาขึ้นอย่างมากจากครั้งที่เราได้กล่าวถึงมันเมื่อปี 2016 ผ่านหัวข้อเรื่องสถาปัตยกรรมแบบไร้เซิร์ฟเวอร์ ซึ่ง Firebase เป็นแพลตฟอร์มสำหรับสร้างแอปพลิเคชันบนมือถือและเว็บไซต์ที่มีเครื่องมือมาให้อย่างครบครัน โดยทำงานต่อยอดบนโครงสร้างพื้นฐานที่รองรับการขยายได้ของ Google เรานั้นชอบเครื่องมืออย่าง Firebase App Distribution เป็นพิเศษ ที่ให้เราสามารถออกแอปพลิเคชันเวอร์ชันทดสอบผ่านทางไปป์ไลน์ของการส่งมอบอย่างต่อเนื่องได้ และอีกเครื่องมืออย่าง Firebase Remote Config ที่ให้เราสามารถเปลี่ยนแปลงและส่งออกค่าการปรับแต่งไปหาแอปพลิเคชันที่ติดตั้งไว้อยู่แล้ว โดยที่ไม่ต้องออกแอปพลิเคชันเวอร์ชันใหม่

Hot Chocolate

ทดลอง

ชุมชนนักพัฒนาและโลกของ GraphQL ยังคงเติบโตขึ้นอย่างต่อเนื่อง ทำให้เกิด Hot Chocolate ซึ่งเป็น GraphQL เซิร์ฟเวอร์สำหรับ .NET (ใช้ได้ทั้ง .NET Core และ .NET แบบดั้งเดิม) เราสามารถใช้มันสร้างและโฮสต์โครงสร้างข้อมูลเก็บไว้ เพื่อใช้สืบค้นข้อมูลผ่านคอมโพเนนท์พื้นฐานของ GraphQL ซึ่งได้แก่ ตัวโหลดข้อมูล รีโซล์ฟเวอร์ โครงสร้างข้อมูล โอเปอเรชัน และ ไทป์ ทีมผู้พัฒนา Hot Chocolate ได้เพิ่มความสามารถในการร้อยโครงสร้างข้อมูล (schema stitching) เข้าด้วยกัน ซึ่งช่วยให้เราสามารถประมวลข้อมูลจากโครงสร้างข้อมูลที่มาจากต่างแหล่งที่มาภายในเอนด์พอยน์เดียวได้ แม้ว่าวิธีนี้อาจถูกนำไปใช้ในทางที่ไม่สมควร แต่ทีมของเรายังรู้สึกพอใจกับการใช้ Hot Chocolate เพราะมีเอกสารคู่มือการใช้งานที่ดี และจากที่เราสามารถใช้มันส่งมอบคุณค่าให้กับลูกค้าของเราได้อย่างรวดเร็ว

Hydra

ทดลอง

การติดตั้งระบบ OAuth2 ของตัวเอง อาจไม่ใช่เรื่องจำเป็นสำหรับทุกองค์กร แต่ถ้ามีความจำเป็นเราพบว่า Hydra เป็นเครื่องมือที่น่าสนใจ Hydra เป็นโอเพนซอร์สเซิฟเวอร์ที่ทำตามมาตรฐาน OAuth2 และทำหน้าที่เป็นผู้ให้บริการ OpenID connect มันรองรับการเก็บข้อมูลในหน่วยความจำเพื่อใช้สำหรับการพัฒนาและรองรับฐานข้อมูลเชิงสัมพันธ์อย่าง PostgreSQL สำหรับการใช้งานในโปรดักชัน มันทำงานโดยไม่จดจำสถานะ (stateless) และสามารถขยายประสิทธิภาพในแนวขวางบนแพลตฟอร์มอย่าง Kubernetes ได้โดยง่าย ทั้งนี้คุณอาจจะต้องปรับเพิ่มหรือลดจำนวนหน่วยของฐานข้อมูลในระหว่างการขยายหน่วยของ Hydra ด้วยเช่นกันเพื่อให้ได้ประสิทธิภาพที่คุณ

ต้องการ และเพราะว่า Hydra ไม่มีระบบจัดการเอกลักษณ์ (identity management) มาให้ในตัว คุณสามารถเชื่อมต่อระบบจัดการเอกลักษณ์ใดๆ ก็ตามกับ Hydra ผ่าน API ที่ออกแบบมาอย่างดี การแยกออกจากกันอย่างชัดเจนระหว่างส่วนจัดการเอกลักษณ์ (identity) และ OAuth2 เฟรมเวิร์คที่เหลือ ทำให้เป็นการง่ายที่จะเชื่อม Hydra เข้ากับระบบการยืนยันตัวตนที่มีอยู่แล้ว

OpenTelemetry

ทดลอง

OpenTelemetry เป็นโครงการโอเพนซอร์ส ใช้สำหรับเฝ้าสังเกตการณ์ระบบ ที่รวมเอามาตรฐาน OpenTracing และ OpenCensus เข้าด้วยกัน โดย OpenTelemetry นั้นประกอบไปด้วยข้อกำหนดมาตรฐาน ไลบรารี เอเยนต์ และเครื่องมืออื่นๆ สำหรับใช้จับค่าสัญญาณทางไกลจากเซอร์วิสต่างๆ เพื่อช่วยให้การสังเกตการณ์ การจัดการ และการหาข้อบกพร่องทำได้ดีขึ้น OpenTelemetry ครอบคลุมสามหลักสำคัญของการเฝ้าสังเกตการณ์ ได้แก่ สามารถติดตามร่องรอยการสื่อสารระหว่างกัน มีตัวชี้วัด และบันทึกเหตุการณ์ที่เกิดขึ้น ซึ่งจากข้อกำหนดมาตรฐานแล้ว Correlations API จะเป็นตัวเชื่อมสามสิ่งนี้เข้าด้วยกัน เพราะฉะนั้น เราสามารถใช้ *ตัวชี้วัด* เพื่อตรวจจับปัญหา ใช้ *ร่องรอย* ที่เกี่ยวข้องเพื่อหาว่าผลกระทบเกิดขึ้นบริเวณใดบ้าง และดีที่สุดคือเรียนรู้จาก *บันทึกเหตุการณ์* เพื่อสืบหาว่าต้นตอของปัญหาเกิดจากอะไร นอกจากนั้น ส่วนประกอบต่างๆ ของ OpenTelemetry สามารถเชื่อมต่อกับส่วนแบ็คเอนด์ของระบบเฝ้าสังเกตการณ์ต่างๆ เช่น Prometheus และ Jaeger รวมไปถึงระบบอื่นๆ อีกด้วย แนวทางของ OpenTracing นั้นกำลังดำเนินไปสู่ทิศทางที่ดี ที่มาตรฐานต่างก็เกื้อหนุนกันและการที่เครื่องมือมีความเรียบง่ายขึ้น

Snowflake

ทดลอง

Snowflake ได้พิสูจน์ความสามารถในการเป็นซอฟต์แวร์บนคลาวด์สำหรับเก็บข้อมูลของระบบบิ๊กดาต้า ดาต้าแวร์เฮาส์ หรือดาต้าเลค ให้กับลูกค้าของเราหลายๆ ราย มันมีสถาปัตยกรรมที่เหนือชั้นที่รองรับการขยายพื้นที่จัดเก็บปรับขีดความสามารถในการประมวลผล หรือปรับจำนวนหน่วยบริการเพื่อจัดการข้อมูล นอกจากนี้ Snowflake ยังมีความยืดหยุ่นสูงมากจากการรองรับการเก็บข้อมูลได้ทั้งแบบมีโครงสร้าง แบบกึ่งมีโครงสร้าง และแบบไม่มีโครงสร้าง มันยังมีตัวเชื่อมต่อ (connectors) สำหรับการใช้งานในรูปแบบที่ต่างกัน เช่น ตัวเชื่อมต่อกับ Spark สำหรับใช้ในงานวิทยาศาสตร์ข้อมูล และตัวเชื่อมต่อกับ SQL สำหรับใช้ในงานวิเคราะห์ เป็นต้น อีกทั้งยังรองรับการทำงานร่วมกับผู้ให้บริการคลาวด์หลากหลายราย เราได้ให้คำแนะนำแก่ลูกค้าหลายรายให้ใช้บริการแบบที่ไม่ต้องจัดการดูแลเองสำหรับการใช้เป็นเครื่องมืออำนวยความสะดวก เช่น การเป็นแหล่งเก็บข้อมูลขนาดใหญ่ แต่หากตัวเลือกดังกล่าวมีความเสี่ยงหรือมีกฎข้อบังคับห้ามไว้ เราแนะนำให้ใช้ Snowflake สำหรับงานประมวลผลแทน ซึ่งมันจะเป็นตัวเลือกที่ดีสำหรับองค์กรที่มีข้อมูลปริมาณมหาศาลที่ต้องใช้แรงงานในการประมวลผลอย่างหนัก แม้เราจะประสบความสำเร็จในการใช้ Snowflake กับงานขนาดกลางของเรา เรายังรอที่จะได้ใช้มันในงานที่มีข้อมูลขนาดมหาศาล ที่ข้อมูลต้องถูกรับผิดชอบโดยแต่ละภาคส่วนขององค์กร

Anthos

ประเมิน

เราได้เห็นการเปลี่ยนแปลงในอุตสาหกรรม จากที่แต่ก่อนนั้นการสร้างคลาวด์แบบไฮบริดเกิดจากความไม่ตั้งใจ หรือที่มักคิดกันว่าการย้ายไปคลาวด์ต้องยกไปทั้งแผง สู่การคิดแบบใหม่ที่การสร้างคลาวด์แบบไฮบริดเกิดจากเจตนาและเป็นไปอย่างพอเพียง มีการตัดสินใจที่รอบคอบเพื่อเลือกระหว่างการใช้ผู้ให้บริการคลาวด์หลายเจ้าผสมผสานกัน หรือการใช้เทคนิคการสร้างคลาวด์ที่ไม่ขึ้นกับผู้ให้บริการดี ซึ่งทั้งหมดนี้เกิดจากการที่องค์กรประยุกต์ใช้หลักคิดหลากหลายมิติเข้าด้วยกันในการวางกลยุทธ์และลงมือทำ เช่น การตัดสินใจว่าจะโฮสต์ข้อมูลหรือสินทรัพย์ต่างๆ ไว้ที่ใดโดยพิจารณาจากความเสี่ยง ความสามารถในการควบคุม และคุณลักษณะทางด้านประสิทธิภาพ มีการตั้งคำถามกับตัวเองว่าจะทำอย่างไรถึงจะใช้ระบบภายในที่ลงทุนไปได้เต็มประสิทธิภาพที่สุด ในขณะเดียวกันก็เป็นการช่วยลดค่าใช้จ่ายในการดำเนินการดูแลอีกด้วย หรือจะทำอย่างไรให้ใช้ข้อได้เปรียบของการผสมผสานผู้ให้บริการคลาวด์หลายรายเข้าด้วยกัน โดยเลือกใช้บริการที่เป็นจุดแข็งหรือเป็นความสามารถพิเศษของผู้ให้บริการแต่ละราย โดยไม่สร้างความซับซ้อนหรือความลำบากให้กับทีมพัฒนาในการสร้างและดูแลแอปพลิเคชันของตนเอง

Anthos เป็นคำตอบของ Google ที่จะทำให้คลาวด์แบบไฮบริดหรือกลยุทธ์การใช้คลาวด์หลายรายผสมผสานกันทำได้ง่ายขึ้น โดยมันมีระบบจัดการและตัวบังคับการ เพื่อควบคุมโอเพนซอร์สเทคโนโลยีต่างๆ เช่น GKE, เซอร์วิสเมช และมีระบบจัดการการตั้งค่า โดยใช้ Git เป็นพื้นฐาน โดยมันจะทำให้ชิ้นงานต่างๆ เคลื่อนย้ายไปมาระหว่างสภาพแวดล้อมที่แตกต่างกันได้ ซึ่งรวมทั้งสภาพแวดล้อมของ Google Cloud และฮาร์ดแวร์ที่อยู่ภายในองค์กรส่วนตัว แม้คลาวด์เจ้าอื่นๆ ก็มีบริการในลักษณะเดียวกัน แต่ Anthos มีเจตนาจะไปไกลกว่าการเป็นคลาวด์แบบไฮบริด เพราะต้องการทำตัวเป็นเครื่องมือที่ทำให้เราสามารถโยกงานระหว่างคลาวด์เจ้าต่างๆ ได้ด้วยเช่นกัน โดยใช้ส่วนประกอบจากโครงการโอเพนซอร์สต่างๆ เป็นพื้นฐาน แต่จะไปถึงเป้าหมายนั้นได้เราคงต้องรอดูกันต่อไป เราเห็นถึงความสนใจของ Anthos ที่เพิ่มขึ้น แม้วิธีนี้ของ Google ในการจัดการคลาวด์แบบไฮบริดจะดูมีความหวัง แต่มันก็ไม่ใช่ยาวิเศษ เพราะเราต้องเปลี่ยนแปลงทั้งคลาวด์ที่มีอยู่และระบบภายใน สำหรับคำแนะนำของเราแก่ลูกค้าที่กำลังพิจารณา Anthos อยู่ คือให้สร้างมาตรการเพื่อเปรียบเทียบข้อดีข้อเสียระหว่างการเลือกใช้บริการต่างๆ ของ Google Cloud เปรียบเทียบกับทางเลือกอื่นๆ เพื่อรักษาระดับความเป็นกลางและความสามารถในการควบคุมที่เหมาะสม

Apache Pulsar

ประเมิน

Apache Pulsar เป็นแพลตฟอร์มโอเพนซอร์สสำหรับการประกาศ-รับฟังข้อความ (publish-subscribe messaging/streaming) ซึ่งแข่งขันในตลาดเดียวกันกับ Apache Kafka โดยมันมีความสามารถทั่วไปตามที่เราคาดหวังไว้ ตั้งแต่การรองรับการส่งข้อความทั้งแบบอะซิงโครนัสที่มีความหน่วงต่ำและแบบซิงโครนัสได้ มีรูปแบบการจัดเก็บข้อความแบบถาวรที่รองรับการขยายของข้อมูล รวมถึงมีไลบรารีเพื่อเรียกใช้งานที่หลากหลายให้เลือกใช้ แต่สิ่งที่ทำให้เราสนใจมันคือเรื่องความง่ายต่อการขยายบริการ โดยเฉพาะกับองค์กรขนาดใหญ่ที่กลุ่มผู้ใช้ถูกแบ่งออกเป็นหลาย

แพลตฟอร์ม

OpenTelemetry ประกอบไปด้วยข้อกำหนดมาตรฐาน ไลบรารี เอเยนต์ และเครื่องมืออื่นๆ สำหรับใช้จับค่าสัญญาณทางไกลจากเซอร์วิสต่างๆ เพื่อช่วยให้การสังเกตการณ์ การจัดการ และการหาข้อบกพร่องทำได้ดีขึ้น

(OpenTelemetry)

Anthos เป็นคำตอบของ Google ที่จะทำให้คลาวด์แบบไฮบริดหรือกลยุทธ์การใช้คลาวด์หลายรายผสมผสานกันทำได้ง่ายขึ้น โดยมันจะทำให้ชิ้นงานต่างๆ เคลื่อนย้ายไปมาระหว่างสภาพแวดล้อมที่แตกต่างกันได้ ซึ่งรวมทั้งสภาพแวดล้อมของ Google Cloud และฮาร์ดแวร์ที่อยู่ภายในองค์กรส่วนตัว

(Anthos)

แพลตฟอร์ม

BigQuery ML ช่วยให้งานแมชชีนเลิร์นนิงเพื่อการทำนายผลหรือการเสนอแนะทำได้ง่ายขึ้นมาก โดยเฉพาะหากต้องการทำการทดลองอย่างรวดเร็ว

(Google BigQuery ML)

ภาคส่วนเพราะโดยธรรมชาติแล้ว Apache Pulsar รองรับการแบ่งผู้ใช้ออกเป็นกลุ่มๆ สามารถจัดสิทธิ์การเข้าถึงตามบทบาท สามารถแจ้งอัตราการใช้งานแยกตามกลุ่ม และสามารถทำสำเนาข้อมูลข้ามภูมิศาสตร์กันได้ เราได้พิจารณาใช้ Pulsar เพื่อจัดการข้อความที่เกิดจากบันทึกเหตุการณ์ที่คอยเข้ามาไม่รู้จบในระบบข้อมูลขนาดใหญ่ ซึ่งข้อความเหตุการณ์ใดๆ ถูกคาดหวังว่าจะถูกจัดเก็บไปไม่มีที่สิ้นสุด และผู้ที่สนใจรับฟังข้อความสามารถเริ่มดึงข้อความจากอดีตเป็นต้นมาได้เสมอ ซึ่งปัญหานี้ Pulsar รองรับโดยการใช้การจัดการจัดเก็บเป็นชั้น แม้มันจะเป็นแพลตฟอร์มที่น่าจับตาสำหรับองค์กรขนาดใหญ่ มันยังมีสิ่งที่ต้องปรับปรุงอยู่เช่นกัน โดยการติดตั้ง ณ วันนี้ยังต้องจัดการดูแลระบบอย่าง ZooKeeper และ BookKeeper ร่วมด้วย จากที่มันได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ เราหวังว่าจะได้เห็นการสนับสนุนจากชุมชนนักพัฒนามากกว่า

Cosmos

ประเมิน

ประสิทธิภาพของเทคโนโลยีบล็อกเชนนั้นได้พัฒนาไปอย่างมาก หากเทียบกับครั้งที่เราได้ประเมินบล็อกเชนไว้ในเรดาร์ฉบับก่อนๆ อย่างไรก็ตาม ยังไม่มีบล็อกเชนเจ้าใดที่สามารถรองรับการใช้งานในระดับเดียวกันกับ “อินเตอร์เน็ต” ได้ จากการมีแพลตฟอร์มบล็อกเชนเจ้าต่างๆ เกิดขึ้นมากมาย เราจึงเห็นถึงการแยกกันอยู่ของข้อมูลและค่าต่างๆ ด้วยเหตุนี้เทคโนโลยีแบบข้ามเชน (cross-chain tech) จึงเป็นหัวข้อสำคัญในชุมชนนักพัฒนาบล็อกเชนตลอดมา ซึ่งก็คือแนวคิดที่เชื่อว่า “อนาคตของบล็อกเชนคือการสร้างเครือข่ายของบล็อกเชนแบบขนานที่เป็นอิสระต่อกันเข้าด้วยกัน” และนี่ก็เป็นวิสัยทัศน์ของ Cosmos เช่นเดียวกัน Cosmos ได้ปล่อย Tendermint ออกมา และมี CosmosSDK ที่นักพัฒนาสามารถปรับแต่งบล็อกเชนที่อยู่อย่างอิสระใดๆ ได้ ซึ่งบล็อกเชนที่แยกกันอย่างอิสระสามารถแลกเปลี่ยนค่าโดยใช้โปรโตคอล Inter-Blockchain Communication (IBC) และ Peg-Zones ทีมของเรานั้นรู้สึกประทับใจอย่างมากหลังจากใช้งาน CosmosSDK ในขณะเดียวกันโปรโตคอล IBC ก็มีความพร้อมขึ้นไปทุกที สถาปัตยกรรมลักษณะนี้อาจจะแก้ปัญหาการทำงานร่วมกันข้ามบล็อกเชนที่เป็นอยู่ และปัญหาการด้านการขยายของบล็อกเชนก็เป็นได้

Google BigQuery ML

ประเมิน

โดยทั่วไปการฝึกสอนโมเดลแมชชีนเลิร์นนิงและการใช้มันทำนายผลนั้นต้องเขียนโค้ดให้นำข้อมูลไปหาโมเดล **Google BigQuery ML** สลับวิธีคิดนี้ โดยเลือกนำโมเดลไปหาข้อมูลแทน จากที่ Google BigQuery เป็นบริการดาต้าแวร์เฮาส์ที่ออกแบบมาให้รองรับการสืบค้นข้อมูลขนาดใหญ่ด้วยภาษา SQL เพื่อการวิเคราะห์ข้อมูล โดย Google BigQuery ML ได้ต่อยอดมันเพื่อใช้สร้าง ฝึกสอน และวัดผลโมเดลแมชชีนเลิร์นนิงจากชุดข้อมูลที่อยู่ใน Google BigQuery ได้ทันที และยังสามารถนำผลการทำนายของโมเดลไปสร้างเป็นชุดข้อมูลใหม่เก็บไว้ที่ BigQuery ได้อีกด้วย มันรับรองการใช้งานโมเดลบางประเภทมาให้ในตัว เช่น โมเดลการถดถอยเชิงเส้นตรง (linear regression) สำหรับงานพยากรณ์ โมเดลแบบไบนารีหรือมัลติคลาสสำหรับงานจำแนกข้อมูล นอกจากนั้นมันยังรองรับการใช้โมเดลที่ได้รับการฝึกสอนแล้วจาก TensorFlow แต่จะมีความสามารถในการใช้งานบางอย่างที่จำกัด ถึงแม้ว่า BigQuery ML และวิธีการใช้งานมันผ่าน SQL จะช่วยให้งานแมชชีนเลิร์นนิงเพื่อการทำนายผลหรือการเสนอแนะทำได้ง่ายขึ้นมาก โดยเฉพาะหากต้องการทำการทดลองอย่างรวดเร็ว แต่วิธีนี้ก็มีข้อเสียด้วยเช่นกัน นั่นคือการที่ต้องประนีประนอมกับบางแง่มุมของการฝึกสอนโมเดล อย่าง การทดสอบความลำเอียงทางจริยธรรม ความสามารถในการอธิบาย และ การส่งมอบอย่างต่อเนื่องสำหรับแมชชีนเลิร์นนิง

JupyterLab

ประเมิน

JupyterLab เป็นส่วนต่อประสานผู้ใช้ผ่านเว็บไซต์รุ่นถัดไปของโครงการ Jupyter หากคุณกำลังใช้งาน Jupyter Notebook อยู่เราอยากให้พิจารณา JupyterLab ด้วยเช่นกัน เพราะมันสร้างสภาพแวดล้อมเชิงโต้ตอบให้กับโค้ด ข้อมูล และ Jupyter Notebook เรามองว่าสิ่งนี้เป็นวิวัฒนาการของ Jupyter Notebook เพราะมันสร้างประสบการณ์ที่ดีขึ้น จากการขยายความสามารถเดิมที่มีอยู่แล้วของ Jupyter Notebook ที่อนุญาตให้นำโค้ด การแสดงผลเป็นภาพ และเอกสารต่างๆ มาอยู่ร่วมกัน

Marquez

ประเมิน

Marquez เป็นโครงการโอเพนซอร์สน้องใหม่สำหรับจัดเก็บและให้บริการข้อมูลอภิพันธุ์ (metadata) ของระบบนิเวศข้อมูล (data ecosystem) ใดๆ โดยมีโมเดลข้อมูลที่สามารถจัดเก็บค่าต่างๆ เช่น ที่มาที่ไปข้อมูล ตัวประมวลผลข้อมูลต้นทาง ตัวประมวลผลข้อมูลปลายทาง สถานะของงาน สามารถให้คำกำกับ (tag) เพิ่มเติมเพื่อใช้ระบุคุณลักษณะอื่นๆ และมี RESTful API สำหรับจัดการข้อมูลอภิพันธุ์ ซึ่งช่วยให้การเชื่อมต่อ Marquez กับชุดเครื่องมืออื่นๆ ในระบบนิเวศข้อมูลง่ายขึ้น เราเริ่มใช้ Marquez ตรงๆ ในตอนต้น จากนั้นค่อยๆ ปรับแต่งการใช้งานให้เข้ากับความต้องการของเรา เช่น เพิ่มการบังคับใช้นโยบายความมั่นคง และปรับแต่งภาษาให้ตรงกับโดเมน หากคุณกำลังมองหาเครื่องมือขนาดเล็กและใช้งานง่ายสำหรับการเริ่มต้นบันทึกและช่วยแสดงภาพตัวประมวลผลข้อมูลอภิพันธุ์ในระบบอยู่ละก็ Marquez เป็นจุดเริ่มต้นที่ดี

Matomo

ประเมิน

Matomo (ชื่อเดิม Piwik) เป็นแพลตฟอร์มโอเพนซอร์สสำหรับวิเคราะห์สถิติการใช้งานเว็บแอปพลิเคชัน ที่คุณสามารถควบคุมและจัดการข้อมูลด้วยตนเองอย่างเต็มที่ คุณสามารถติดตั้ง Matomo ไว้ที่โฮสต์ส่วนตัวเพื่อรักษาไม่ให้ข้อมูลเชิงวิเคราะห์ของคุณไปอยู่กับระบบภายนอก Matomo สามารถเชื่อมโยงข้อมูลเชิงวิเคราะห์เข้ากับแพลตฟอร์มข้อมูลภายในองค์กรได้ง่าย เพื่อนำข้อมูลไปสร้างรูปแบบโมเดลการใช้งานที่เหมาะสมกับความต้องการของเราโดยเฉพาะ

MeiliSearch

ประเมิน

MeiliSearch คือเครื่องมือสร้างระบบสืบค้นหาคำ (text search engine) ที่ให้ผลลัพธ์การค้นหาที่รวดเร็ว ใช้งานง่ายและไม่ยุ่งยากในการติดตั้ง แม้ตลอดหลายปีที่ผ่านมา Elasticsearch จะเป็นตัวเลือกยอดนิยมสำหรับสร้างระบบสืบค้นหาคำที่รองรับการขยาย แต่หากคุณมีข้อมูลไม่มาก

พอจนต้องใช้ระบบกระจายศูนย์ในการเก็บ และอยากได้ระบบค้นหาคำที่รองรับการค้นด้วยคำที่สะกดผิด เราแนะนำให้ประเมิน MeiliSearch ดู

Stratos
ประเมิน

Ultraleap (เดิมชื่อ Leap Motion) คือผู้นำด้านการสร้างโลกความเป็นจริงขยาย (Extended Reality, XR) และเป็นผู้สร้างอุปกรณ์ฮาร์ดแวร์สุดล้ำที่สามารถติดตามการเคลื่อนไหวของมือ ทำให้มือของผู้ใช้เข้าสู่โลกความเป็นจริงเสมือนได้ Stratos เป็นแพลตฟอร์มเทคโนโลยีที่อยู่เบื้องหลังงานผลงานต่างๆ ของ Ultraleap ซึ่งประกอบไปด้วยเทคโนโลยีระบบสัมผัส เซ็นเซอร์ และแพลตฟอร์มซอฟต์แวร์ ที่มีความสามารถในการใช้เสียงอัลตราซาวด์เพื่อสร้างการตอบสนองด้วยระบบสัมผัส โดยผู้ใช้ไม่จำเป็นต้องถืออุปกรณ์ใดๆ ไว้ ซึ่งสามารถนำไปใช้ในกรณี เช่น รถยนต์ที่สามารถอ่านสัญญาณมือของคนขับเพื่อปรับอุณหภูมิในรถและตอบสนองด้วยการสั่นเพื่อเป็นการรับทราบถึงคำสั่ง เราตื่นเต้นที่จะได้เห็นเทคโนโลยีนี้และเฝ้ารอว่านักคิดสร้างสรรค์ทางเทคโนโลยีทั้งหลายจะนำมันไปใช้แก้ปัญหาอะไร

Trillian
ประเมิน

Trillian เป็นคริปโตชนิดที่ใช้การจัดเก็บข้อมูลแบบรวมศูนย์ตรวจสอบได้ ซึ่งต่างจากการบันทึกธุรกรรมด้วยเทคโนโลยีบล็อกเชนทั่วไป ที่เหมาะสำหรับสภาพแวดล้อมแบบกระจายตัวที่ไม่สามารถเชื่อถือใครได้ สำหรับสภาพแวดล้อมแบบองค์กร ที่ไม่จำเป็นต้องใช้เทคนิคการหาข้อตกลงร่วมกันผ่านการประมวลผลหนักของซีพียู เราอยากแนะนำให้ลองพิจารณา Trillian ดูเป็นทางเลือก

การใช้งาน Node สำหรับทุกสิ่ง
เฝ้าระวัง

เทคโนโลยีที่ได้รับความนิยมสูงๆ มักมีแนวโน้มที่จะถูกนำไปใช้อย่างพร่ำเพรื่อ ที่เห็นได้ชัดเจนในขณะนี้ คือ**การใช้งาน Node สำหรับทุกสิ่ง** ซึ่งมีโอกาสให้เกิดการเลือกใช้งาน Node โดยไม่พิจารณาให้ถี่ถ้วนเสียก่อน หรือใช้มันด้วยเหตุผลผิดๆ ซึ่งมีอยู่สองเหตุผลที่เรามักได้ยินกันเป็นประจำ เหตุผลแรกที่คนถูกชักชวนให้ใช้ Node เพราะมันจะได้กลายเป็นภาษาโปรแกรมมิ่งเดียวที่ใช้กันทั้งระบบ ซึ่งเราเห็นต่างว่าการผสมใช้หลายภาษาโปรแกรมมิ่ง (Polyglot programming) เข้าด้วยกันเป็นแนวทางที่ดีกว่าเสมอ ทั้งนี้ JavaScript เองก็เป็นภาษาที่ดีเช่นกัน เหตุผลที่สองที่คนมักจะอ้างในการเลือก Node.js คือเหตุผลด้าน“ประสิทธิภาพ” แม้จะมีผลการทดสอบประสิทธิภาพที่ดีออกมามากมายให้เปรียบเทียบ แต่ความคิดนี้ก็ยังฝังติดมาจากช่วงแรกที่ Node.js ได้รับกระแสความนิยม เพราะขณะนั้นมันเป็นเฟรมเวิร์คแรกที่นำเทคนิคการเขียนโปรแกรมแบบไม่มีการบล็อค (Nonblocking Programming Model) มาใช้ ซึ่งวิธีนี้ให้ประสิทธิภาพที่ดีมากกับงานประเภทที่ใช้ I/O หนักๆ (ซึ่งเราได้กล่าวถึงในงานเขียนเรื่อง Node.js ตั้งแต่ปี 2012) แต่ก็ด้วยธรรมชาติการทำงานที่อาศัยหน่วยการประมวลผลย่อยเพียงหน่วยเดียว (Single-threaded) เท่านั้น มันจึงไม่เหมาะกับการใช้ในงานประมวลผลหนักๆ ปัจจุบันแพลตฟอร์มอื่นๆ ต่างก็มีความสามารถการเขียนโปรแกรมแบบไม่มีการบล็อคออกมาให้ใช้งานแล้ว ซึ่งบางตัวยังมี API ที่เรียบง่ายและทันสมัยกว่า การเลือกใช้งาน Node.js เพื่อเหตุผลด้าน “ประสิทธิภาพ” จึงไม่ใช่เหตุผลที่ดีอีกต่อไป

แพลตฟอร์ม

MeiliSearch คือเครื่องมือสร้างระบบสืบค้นหาคำที่ให้ผลลัพธ์การค้นหาที่รวดเร็ว ใช้งานง่าย และไม่ยุ่งยากในการติดตั้ง มันเหมาะอย่างมากหากคุณมีข้อมูลไม่มากพอจนต้องใช้ระบบกระจายศูนย์ในการเก็บ และอยากได้ระบบค้นหาคำที่รองรับการค้นด้วยคำที่สะกดผิด

(MeiliSearch)

Stratos คือ เทคโนโลยีระบบสัมผัส เซ็นเซอร์ และแพลตฟอร์มซอฟต์แวร์ที่อยู่เบื้องหลังงาน XR ที่ Ultraleap (ชื่อเดิม Leap Motion) เป็นผู้บุกเบิก

(Stratos)

TECHNOLOGY RADAR Vol. 22

เครื่องมือ

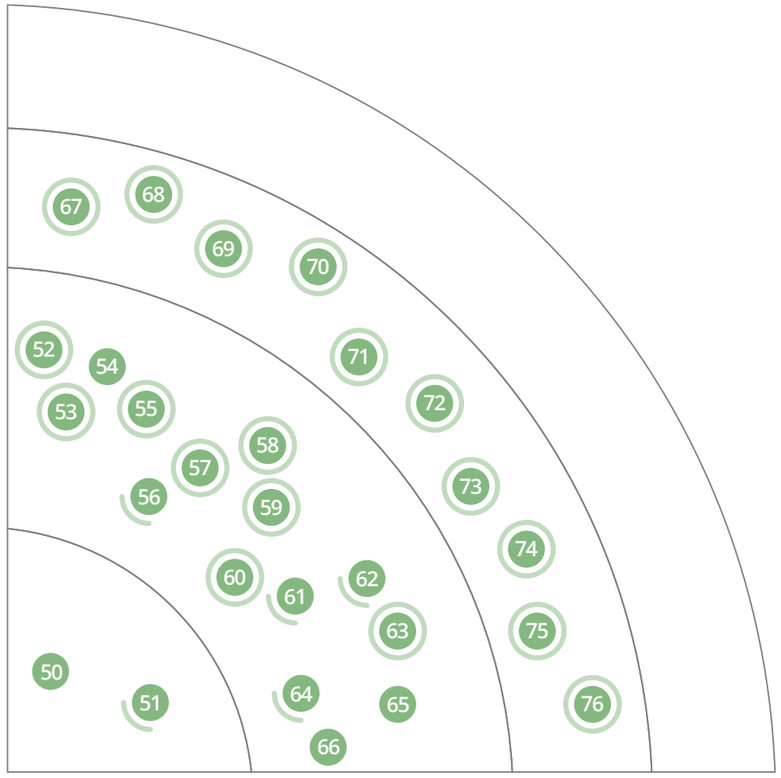
เครื่องมือ

Cypress
นำไปใช้

Cypress ยังเป็นเครื่องมือโปรดสำหรับเรา เพราะช่วยให้นักพัฒนาสามารถจัดการแบบทดสอบการทำงานตั้งแต่ต้นจนจบได้ด้วยตัวเอง ตามแนวคิดการวางชุดทดสอบเป็นสามเหลี่ยมพีระมิด (test pyramid) เราเลือกที่จะหยิบยกมันขึ้นมาอีกครั้งในเรดาร์ฉบับนี้ เพราะในเวอร์ชันล่าสุด Cypress ได้เพิ่มความสามารถการทำงานร่วมกับเบราว์เซอร์ Firefox เข้ามา ซึ่งเราก็เห็นด้วยอย่างยิ่งกับการทดสอบงานบนเบราว์เซอร์ที่แตกต่างกัน ที่ผ่านมา การครองตลาดของเบราว์เซอร์ Chrome หรือเบราว์เซอร์ในตระกูล Chromium ได้นำไปสู่เทรนด์ที่น่ากังวล ว่าทีมจะให้ความสำคัญกับการทดสอบแค่บนเบราว์เซอร์เดียวเท่านั้น ซึ่งอาจก่อให้เกิดเรื่องประหลาดใจที่ร้ายแรงขึ้นมาได้

Figma
นำไปใช้

Figma ได้พิสูจน์ตัวเองแล้วว่าเป็นเครื่องมือไว้ร่วมกันออกแบบที่ดี โดยไม่ใช่แค่เฉพาะกับทีมนักออกแบบด้วยกันเอง แต่รวมถึงทีมที่มีสมาชิกต่างบทบาทรวมอยู่ด้วย นักพัฒนาหรือคนอื่นๆ สามารถที่จะดูการออกแบบที่ทำไว้และให้ความคิดเห็นผ่านเว็บไซต์โดยไม่จำเป็นต้องติดตั้งแอปพลิเคชันเพิ่มเติม เมื่อเทียบกับคู่แข่ง (เช่น Invision หรือ Sketch) ที่คุณจำเป็นต้องใช้เครื่องมือหลายตัวร่วมกันเพื่อจัดการเวอร์ชัน ประสานงาน และแชร์ผลการออกแบบ แต่ Figma ได้รวมความสามารถทั้งหมดนี้ไว้ในเครื่องมือตัวเดียวกัน ทำให้ง่ายสำหรับทีมในการร่วมมือและสร้างสรรค์ไอเดียใหม่ๆ ด้วยกัน ทีมของเราพบว่า Figma มีประโยชน์มาก โดยเฉพาะกับการทำงานแบบทางไกล หรือกับทีมที่กระจายสถานที่กันทำงาน นอกจากความสามารถในการทำงานร่วมกันและการออกแบบได้อย่างเรียลไทม์แล้ว Figma ยังมี API ไว้ให้เราพัฒนากระบวนการ DesignOps ให้ดีขึ้นอีกด้วย



Dojo
ทดลอง

เมื่อไม่กี่ปีที่ผ่านมา Docker และแนวคิดคอนเทเนอร์ได้เปลี่ยนวิธีคิดของเราไปอย่างสิ้นเชิง ถึงวิธีการสร้างแพ็คเกจ การติดตั้ง และการใช้งานแอพลิเคชัน แม้สภาพแวดล้อมโปรดักชันจะถูกปรับปรุงไปมากขึ้นแค่ไหนก็ตาม แต่สำหรับสภาพแวดล้อมสำหรับการพัฒนายังมีการเปลี่ยนแปลงไม่มากนัก นักพัฒนายังเสียเวลาอย่างมากให้กับการเตรียมความพร้อมก่อนลงมือพัฒนาและยังคงประสบปัญหา “แต่มันทำงานได้ที่เครื่องฉันนะ” อยู่เป็นประจำ ซึ่ง Dojo พยายามเข้ามาช่วยแก้ปัญหานี้ โดยการทำให้สภาพแวดล้อมในการพัฒนามีมาตรฐาน ถูกจัดเก็บเป็นเวอร์ชัน สามารถแจกจ่ายผ่านอิมเมจของ Docker ได้มีหลายทีมของเราใช้ Dojo เพื่อปรับปรุงให้การพัฒนา การทดสอบและการบิลด์โค้ดเป็นหนึ่งเดียวกันทั้งหมด ตั้งแต่บนเครื่องส่วนตัวไปจนถึงในไปป์ไลน์ของโปรดักชัน

DVC
ทดลอง

ในปี 2018 เราได้กล่าวถึง DVC ร่วมกับการกำหนดเวอร์ชันของข้อมูลเพื่อการวิเคราะห์ที่ทำซ้ำได้ นับตั้งแต่นั้นเป็นต้นมา DVC ได้กลายเป็นเครื่องมือยอดนิยมเพื่อใช้จัดการการทดลองในโครงการแมชชีนเลิร์นนิง DVC ใช้ git เป็นพื้น

เครื่องมือ

Jaeger คือระบบโอเพนซอร์สสำหรับติดตามร่องรอยแบบกระจายตัว เราเองประสบความสำเร็จในการใช้มันร่วมกับ Istio และ Envoy บน Kubernetes และเรายังชอบส่วนต่อประสานผู้ใช้ที่มีให้อีกด้วย

(Jaeger)

k9s เป็นเครื่องมือประเภทโต้ตอบกับผู้ใช้ที่สามารถทำงานทุกอย่างที่ kubectl ทำได้ ผ่านส่วนต่อประสานผู้ใช้ที่อยู่ในรูปแบบแอปพลิเคชันบนเทอมินอล

(k9s)

ฐานจึงเป็นสภาพแวดล้อมที่คุ้นเคยสำหรับนักพัฒนาให้สามารถประยุกต์ใช้หลักปฏิบัติทางวิศวกรรมกับแนวทางของแมชชีนเลิร์นนิงได้สะดวกขึ้น เนื่องจากมันกำหนดเวอร์ชันของโค้ดที่ประมวลผลข้อมูลไว้กับตัวข้อมูลเอง และช่วยติดตามสถานะขั้นตอนของไปป์ไลน์ มันจึงช่วยให้กิจกรรมต่างๆ ในการสร้างโมเดลข้อมูลเป็นระเบียบมากขึ้น โดยไม่ส่งผลรบกวนต่อกระบวนการการวิเคราะห์ข้อมูลแต่อย่างใด

เครื่องมือติดตามการทดลองสำหรับแมชชีนเลิร์นนิง

ทดลอง

งานในแต่ละวันของแมชชีนเลิร์นนิงมักจะวนเวียนอยู่กับการทดลองปรับแต่งตัวแปรต่างๆ เช่น ปรับวิธีเลือกรูปแบบโมเดลและการเชื่อมต่อของเน็ตเวิร์ก ปรับข้อมูลที่จะนำมาใช้ฝึกสอน ปรับปรุงประสิทธิภาพการทำงาน หรือปรับแต่งโมเดลเล็กน้อยให้สมบูรณ์ นักวิทยาศาสตร์ข้อมูลจำเป็นต้องใช้ประสบการณ์และสัญชาตญาณในการตั้งสมมุติฐานเพื่อปรับแต่งค่า แล้วจึงตรวจสอบว่าการเปลี่ยนแปลงค่าเหล่านั้นส่งผลต่อประสิทธิภาพของโมเดลโดยรวมอย่างไร หลังจากที่วิธีปฏิบัตินี้มีความพร้อมและเป็นที่รู้จักมากขึ้น เราพบว่ามีหลายทีมมีความต้องการใช้ **เครื่องมือติดตามการทดลองสำหรับแมชชีนเลิร์นนิง** มากขึ้นเรื่อยๆ เพราะมันช่วยให้ผู้ทำการทดลองสามารถติดตามผลและทำงานอย่างมีระบบแบบแผนยิ่งขึ้น แม้ยังไม่มีผู้ชนะที่ชัดเจน แต่เครื่องมืออย่าง MLflow และแพลตฟอร์มอย่าง Comet หรือ Neptune ก็ช่วยเพิ่มความเคร่งครัดและความสามารถในการทำซ้ำให้กับกระบวนการทำงานแมชชีนเลิร์นนิง

Goss

ทดลอง

เราได้กล่าวถึง **Goss** แบบคร่าวๆ ในเรดาร์ของเรามาบ้างว่าเป็นเครื่องมือสำหรับ ทดสอบการสร้างสภาพแวดล้อม (provisioning testing) เมื่อครั้งที่เราพูดถึงเทคนิค การพัฒนาคอนเทเนอร์ที่ขับเคลื่อนด้วยชุดทดสอบ Goss อาจจะแทน Serverspec ไม่ได้โดยตรงเพราะมันไม่ได้มีฟีเจอร์ที่มากเท่า แต่คุณก็อาจพิจารณาใช้มันได้หากว่ามันเพียงพอต่อความต้องการ ด้วยขนาดที่เล็กและอยู่ในรูปแบบไบนารีที่ไม่พึ่งพิงกับใคร (เมื่อเทียบกับการที่ ServerSpec ต้องมี Ruby ติดตั้งไว้ก่อน) รูปแบบที่ไม่ควรแต่พบเห็นได้บ่อยเมื่อใช้เครื่องมืออย่าง Goss คือ การต้องบริหารจัดการเป็นคู่เสมอ กล่าวคือ เมื่อการเปลี่ยนแปลงใดๆ เกิดขึ้นที่ไฟล์โครงสร้างพื้นฐานด้วยโค้ดเราจะต้องตามไปแก้ไฟล์ที่เกี่ยวข้องในชุดทดสอบด้วยเสมอ เนื่องจากทั้งคู่มีความสัมพันธ์ที่ผูกกันอย่างใกล้ชิดเกินไปจึงต้องใช้พลังงานอย่างมากในการดูแลทั้งสองฝั่ง โดยความผิดพลาดที่เกิดขึ้นส่วนใหญ่มักมาจากนักพัฒนาเองที่ไปแก้ไขที่ฝั่งหนึ่งแล้วลืมแก้ไขอีกฝั่งด้วย ซึ่งการสร้างชุดทดสอบในลักษณะนี้น้อยครั้งมากที่จะตรวจจับเจอปัญหาได้จริง

Jaeger

ทดลอง

Jaeger คือระบบโอเพนซอร์สสำหรับติดตามร่องรอยแบบกระจายตัว ซึ่งรองรับมาตรฐาน OpenTelemetry โดย Jaeger ได้รับแรงบันดาลใจจากงานวิจัยเรื่อง Dapper ของ Google เช่นเดียวกับ Zipkin เราเองประสบความสำเร็จในการใช้ Jaeger ร่วมกับ Istio และ Envoy บน Kubernetes และเรายังชอบส่วนต่อประสานผู้ใช้งานที่มีให้อีกด้วย มันเปิดให้ดึงข้อมูลตัวชี้วัดที่ได้จากการทิ้งร่องรอยมาเก็บไว้ในรูปแบบที่ใช้กับ Prometheus ได้ ซึ่งทำให้เครื่องมืออื่นสามารถนำข้อมูลไปใช้ต่อ อย่างไรก็ตาม เครื่องมือยุคใหม่อย่าง Honeycomb จะรวมข้อมูลการทิ้งร่องรอยและข้อมูลตัวชี้วัดการทำงานของระบบเข้าด้วยกันไว้ให้ในตัว เพื่อให้งานวิเคราะห์ข้อมูลที่มาใช้ต่อทำได้ง่ายขึ้น Jaeger เข้าร่วมกับโครงการ CNCF ตั้งแต่ปี 2017 และถูกยกระดับจาก CNCF ให้อยู่ในระดับความพร้อมสูงที่สุดไปแล้ว ซึ่งบ่งบอกถึงความแพร่หลายของการใช้งานมันบนโปรดักชัน

k9s

ทดลอง

เรายังคงเป็นผู้สนับสนุนให้ใช้การกำหนดโครงสร้างพื้นฐานด้วยโค้ดอย่างหนักแน่น และเราเชื่อเสมอว่าการจะบริหารจัดการแอปพลิเคชันแบบกระจายศูนย์ได้ดีจำเป็นต้องมีวิธีการตรวจสอบดูแลระบบที่แข็งแรงเสียก่อน ถึงอย่างนั้นการมีเครื่องมือประเภทโต้ตอบกับผู้ใช้ได้ อย่างเช่น AWS web console ก็มีประโยชน์เสริมอยู่บ้าง เพราะเครื่องมือประเภทนี้ช่วยให้เราสามารถสำรวจดูทรัพยากรต่างๆ เป็นกรณีเฉพาะกิจไปโดยไม่จำเป็นต้องจำคำสั่งที่คลุมเครือได้ขึ้นใจ อย่างไรก็ตามการแก้ไขระบบด้วยเครื่องมือประเภทนี้โดยตรงยังเป็นวิธีการปฏิบัติที่ไม่เหมาะสม สำหรับ Kubernetes นั้นมี **k9s** เป็นเครื่องมือประเภทโต้ตอบกับผู้ใช้ที่สามารถทำงานทุกอย่างที่ kubectl ทำได้ ผ่านส่วนต่อประสานผู้ใช้ที่อยู่ในรูปแบบแอปพลิเคชันบนเทอมินอล ซึ่งอาจทำให้บางคนหวนนึกถึงความทรงจำเก่าๆ ที่เคยมีกับ Midnight Commander

kind

ทดลอง

kind เป็นเครื่องมือสร้างคลัสเตอร์ของ Kubernetes ในเครื่องส่วนตัว มันสามารถจำลองสภาพแวดล้อมขึ้นมาได้โดยอาศัยการสร้างคอนเทเนอร์ของ Docker ขึ้นมา พอนำ kubetest มาใช้ร่วมกับ kind แล้ว จะสามารถทดสอบการทำงานของระบบตั้งแต่ต้นจนจบภายใต้สภาพแวดล้อมที่จำลองขึ้นมาได้อย่างง่ายดาย ส่วนเรานั้นใช้ kind ในการสร้างคลัสเตอร์ขึ้นมาชั่วคราวเพื่อทดสอบว่าทรัพยากรของ Kubernetes เช่น โอเปอเรเตอร์และ CRD ทำงานถูกต้องหรือไม่ โดยทำการทดสอบเป็นส่วนหนึ่งของไปป์ไลน์

mkcert

ทดลอง

mkcert เป็นเครื่องมืออำนวยความสะดวกในการสร้างใบรับรองอิเล็กทรอนิกส์ เพื่อใช้ระหว่างการพัฒนาซอฟต์แวร์ การใช้ใบรับรองอิเล็กทรอนิกส์ตัวจริงที่ออกโดยผู้ให้บริการรับรองสากล (CA) บนเครื่องส่วนตัวนักพัฒนาเป็นเรื่องที่ยากจนแทบเป็นไปไม่ได้ (เช่น บนโฮสต์อย่าง example.test, localhost หรือ 127.0.0.1) ในสถานการณ์เช่นนี้ การเลือกใช้ใบรับรองอิเล็กทรอนิกส์ชนิดที่ออกด้วยตนเอง (self-signed) อาจเป็นเพียงตัวเลือกเดียวที่เป็นไปได้ mkcert สามารถทำให้เรื่องนี้สะดวกขึ้น โดยจะสร้างใบรับรองชนิดที่ออกด้วยตนเองขึ้นมาที่ผ่านการรับรองจาก CA ปลอมที่มันสร้างขึ้นมาคู่กัน และทำการติดตั้ง CA ปลอมในเครื่องนักพัฒนาให้อย่างอัตโนมัติ อย่างไรก็ตามสำหรับการใช้งานเพื่อจุดประสงค์อื่นที่ไม่ใช่เพื่อการพัฒนาบนเครื่องส่วนตัวหรือเพื่อการทดสอบแล้ว เราขอแนะนำเป็นอย่างยิ่งให้ใช้ใบรับรองอิเล็กทรอนิกส์ที่ออกโดยผู้ให้บริการสากลตัวจริงเพื่อหลีกเลี่ยงปัญหาความน่าเชื่อถือของใบรับรองที่จะตามมา

MURAL

ทดลอง

MURAL ให้คำนิยามตัวเองว่าคือ “พื้นที่ทำงานดิจิทัลเพื่อประสานงานกันแบบเห็นภาพ” โดยเป็นเว็บไซต์ที่จำลองพื้นที่กระดานไวท์บอร์ดและกระดาษโน้ตเพื่อให้ทีมมาสร้างปฏิสัมพันธ์ร่วมกันออนไลน์ มันมีความสามารถในการโหวต การให้ข้อคิดเห็น การเขียนโน้ตและการ“ติดตามผู้นำเสนอ” มาให้ในตัว ซึ่งหนึ่งในความสามารถที่เราชอบเป็นพิเศษคือ ผู้ใช้สามารถสร้างกระดานแม่แบบเตรียมไว้ก่อน แล้วใช้มันซ้ำกับการประชุมแต่ละครั้ง ผู้ให้บริการผลิตภัณฑ์ชุดสำหรับทำงานร่วมกันหลายเจ้าต่างก็มีเครื่องมือในลักษณะนี้เหมือนกัน (เช่น Google Jamboard และ Microsoft Whiteboard) ซึ่งต่างก็น่าศึกษาเพิ่มเติม แต่เราพบว่า MURAL นั้นลื่นไหล ให้ประสิทธิผลที่ดี และมีความยืดหยุ่นกับการใช้งาน

Open Policy Agent (OPA)

ทดลอง

Open Policy Agent (OPA) ได้กลายเป็นเครื่องมือที่เราโปรดปรานอย่างรวดเร็วสำหรับใช้เป็นส่วนประกอบหนึ่งในระบบแบบกระจายศูนย์ที่เราสร้างให้กับลูกค้า โดยที่ OPA เป็นเฟรมเวิร์คและภาษาที่ช่วยสร้างความเป็นหนึ่งเดียวให้กับการประกาศ การบังคับใช้ และการควบคุมนโยบายสำหรับคอมโพเนนท์ใดๆ บนระบบคลาวด์เนทีฟ OPA เป็นตัวอย่างที่ดีเยี่ยมของเครื่องมือที่ใช้แนวคิดกำหนดนโยบายความมั่นคงด้วยโค้ด พวกเราได้รับประสบการณ์ที่ลื่นไหลจากการใช้ OPA ในหลากหลายสถานการณ์ ตั้งแต่ ใช้ในการติดตั้งทรัพยากรให้กับคลัสเตอร์ของ Kubernetes บังคับใช้ควบคุมสิทธิ์เข้าถึงบริการต่างๆ ในเซอร์วิสเมช (service mesh) และวางกฎเพื่อควบคุมการเข้าถึงทรัพยากรของแอปพลิเคชันอย่างละเอียด ล่าสุด Styra มีบริการเชิงพาณิชย์ออกมาในชื่อ Declarative Authorization Service (DAS) ที่ช่วยให้องค์กรขนาดใหญ่สามารถนำ OPA มาใช้ได้สะดวกยิ่งขึ้น ด้วยการเพิ่มเครื่องมือควบคุมจัดการ OPA สำหรับ Kubernetes มีไลบรารีที่เตรียมชุดนโยบายต่างๆ ไว้ล่วงหน้ามาให้ในตัว มีเครื่องมือวิเคราะห์ผลกระทบของการประยุกต์นโยบาย และมาพร้อมกับความสามารถในการบันทึกเหตุการณ์สำคัญที่เกิดขึ้น เราตั้งตารอที่จะเห็น OPA มีความพร้อมมากขึ้น และถูกขยายการใช้งานไปนอกเหนือการให้บริการในระดับปฏิบัติการ ไปสู่โลกของข้อมูลขนาดใหญ่

Optimal Workshop

ทดลอง

งานวิจัยพฤติกรรมและประสบการณ์ของผู้ใช้ต้องเก็บและวิเคราะห์ข้อมูล เพื่อใช้ตัดสินใจทิศทางของผลิตภัณฑ์ ทีมเราพบว่า Optimal Workshop มีประโยชน์เพราะมันช่วยให้การพิสูจน์งานต้นแบบและการปรับแต่งการทดสอบเพื่อเก็บข้อมูลทำได้ง่าย ส่งผลให้เราตัดสินใจได้ดีกว่า ความสามารถต่างๆ เช่น การรู้ว่าผู้ใช้คลิกตรงไหนเป็นที่แรก การจัดเรียงการ์ด หรือการรายงานบริเวณใดที่ผู้ใช้ให้ความสนใจบ่อย นอกจากช่วยพิสูจน์งานต้นแบบแล้ว ยังช่วยปรับปรุงการออกแบบการนำทางในเว็บไซต์ และการจัดลำดับข้อมูลในการแสดงผลอีกด้วย เครื่องมือนี้เหมาะสมอย่างยิ่งกับทีมที่ทำงานต่างสถานที่กันเพราะสามารถทำการวิจัยแบบทางไกลได้

Phrase

ทดลอง

เราเคยกล่าวไว้ในหัวข้อ Crowdin ว่าปัจจุบันการจะบริหารงานแปลเพื่อให้ผลิตภัณฑ์รองรับผู้ใช้หลายภาษา คุณมีแพลตฟอร์มต่างๆ เป็นทางเลือกแทนการใช้วิธีแลกเปลี่ยนสเปรตชีตผ่านอีเมลกันไปมา ทีมของเราได้รับประสบการณ์ที่ดีเมื่อใช้ Phrase เพราะความง่ายต่อการใช้งานสำหรับผู้ใช้ทุกบทบาท นักแปลสามารถใช้เครื่องมือนี้จากบราวเซอร์โดยตรง ผู้จัดการสามารถเพิ่มรายการใหม่ที่ต้องการแปลเข้าไป และสามารถเชื่อมโยงงานแปลให้กับทีมต่างๆ ผ่านทางหน้าจอเดียวกัน ส่วนนักพัฒนาก็สามารถใช้งาน Phrase บนเครื่องของนักพัฒนาได้โดยตรง หรือเรียกใช้งานบนไปป์ไลน์ได้ด้วย อีกฟีเจอร์ที่คู่ควรแก่การกล่าวถึง คือความสามารถในการกำหนดเวอร์ชั่นของงานแปลผ่านการแท็ก ซึ่งส่งผลให้เราสามารถมองเห็นข้อแตกต่างระหว่างงานแปลที่ต่างเวอร์ชั่นกันได้ภายในผลิตภัณฑ์ของเราโดยตรง

ScoutSuite

ทดลอง

ScoutSuite เป็นเครื่องมือที่ถูกพัฒนาต่อยอดมาจาก Scout2 อีกที (ซึ่งเราเคยกล่าวถึงในเรดาร์ปี 2018) เราสามารถใช้มันเพื่อประเมินสถานภาพความมั่นคงขององค์กรที่อยู่ภายใต้ผู้ให้บริการคลาวด์รายต่างๆ ครอบคลุมทั้ง AWS, Azure, GCP และผู้ให้บริการรายอื่นๆ โดยมันจะรวบรวมข้อมูลการตั้งค่าต่างๆ บนสภาพแวดล้อมใดๆ ไว้อย่างอัตโนมัติ และทำการประเมินโดยการนำข้อมูลไปประยุกต์กับกฎที่ตั้งไว้ จากหลายๆ โครงการของเรา พบว่ามันมีประโยชน์อย่างมากหากต้องการประเมินความมั่นคงที่ช่วงเวลาใดเวลาหนึ่ง

เครื่องมือสำหรับทดสอบการเปลี่ยนแปลงด้านภาพแสดงผล

ทดลอง

นับตั้งแต่ครั้งแรกที่เรากล่าวถึง **เครื่องมือสำหรับทดสอบการเปลี่ยนแปลงด้านภาพแสดงผล (visual regression testing tools)** ในปี 2014 การใช้งานเทคนิคนี้ก็เป็นที่แพร่หลายมากขึ้น เครื่องมือที่เกี่ยวข้องก็มีพัฒนาการที่เปลี่ยนไปจากนั้นอย่างมาก BackstopJS ยังคงเป็นเครื่องมือที่ยอดเยี่ยมและมีฟีเจอร์ใหม่ออกมาอย่างสม่ำเสมอ หนึ่งในนั้นคือความสามารถในการทำงานภายใต้ Docker คอนเทเนอร์ Loki เป็นอีกเครื่องมือที่เราได้กล่าวถึงในเรดาร์ฉบับที่แล้ว ส่วนเครื่องมืออย่าง Applitools, CrossBrowserTesting และ Percy ก็อยู่ในรูปแบบคลาวด์โซลูชัน ส่วนเครื่องมือที่น่ากล่าวถึงอีกตัวคือ Resemble.js ซึ่งเป็นไลบรารีในการหาความแตกต่างของรูปภาพ แม้ทีมส่วนมากจะได้ใช้มันในทางอ้อมผ่าน BackstopJS เท่านั้น แต่ก็มีบางทีมนำ Resemble.js ไปใช้วิเคราะห์และเปรียบเทียบรูปภาพหน้าเว็บเพจโดยตรง โดยรวมแล้ว จากประสบการณ์ของเราพบว่าในช่วงแรกของการพัฒนาเครื่องมือทดสอบประเภทนี้จะมีประโยชน์ไม่มากนัก เพราะส่วนต่อประสานผู้ใช้มีการเปลี่ยนแปลงขนานใหญ่อยู่ตลอด แต่จะเริ่มคุ้มค่าเมื่อผลิตภัณฑ์มีความพร้อมมากขึ้น มีส่วนต่อประสานผู้ใช้ที่นิ่งแล้ว

เครื่องมือ

OPA เป็นเฟรมเวิร์คและภาษาที่ช่วยสร้างความเป็นหนึ่งเดียวให้กับการประกาศ การบังคับใช้ และการควบคุมนโยบาย สำหรับคอมโพเนนท์ใดๆ บนระบบคลาวด์เนทีฟ

(Open Policy Agent (OPA))

ทีมเราพบว่า Optimal Workshop มีประโยชน์เพราะมันช่วยให้การพิสูจน์งานต้นแบบและการปรับแต่งการทดสอบเพื่อเก็บข้อมูลทำได้ง่าย ส่งผลให้เราตัดสินใจได้ดีกว่า

(Optimal Workshop)

เครื่องมือ

โครงการโอเพนซอร์ส AsyncAPI เป็นความพยายามที่จะสร้างมาตรฐานสำหรับ API แบบอะซิงโครนัสขึ้นมาเพื่อใช้ในงานที่ขับเคลื่อนพฤติกรรมด้วยเหตุการณ์ (event-driven) รวมทั้งมีชุดเครื่องมือสำหรับการใช้งานมาตรฐานนี้

(AsyncAPI)

ConfigCat รองรับการเปิด/ปิดฟีเจอร์แบบเรียบง่าย การแบ่งกลุ่มประเภทผู้ใช้งานและการทดสอบแบบ A/B พร้อมทั้งมีแผนแบบฟรีที่เป็นมิตรสำหรับการใช้งานในปริมาณไม่มากหรือสำหรับผู้เริ่มต้น

(ConfigCat)

Visual Studio Live Share

ทดลอง

Visual Studio Live Share เป็นชุดส่วนต่อขยายของ Visual Studio Code และ Visual Studioในช่วงสถานการณ์นี้ที่หลายทีมกำลังมองหาตัวเลือกสำหรับการทำงานร่วมกันระยะไกล เราอยากให้คนให้ความสนใจกับเครื่องมืออันเยี่ยมยอดนี้ Live Share ให้ประสบการณ์ที่ดีในการแพร์กันระยะไกล ใช้เวลาแฝงน้อย โดยแบนด์วิธที่ใช้ต่ำกว่ามากเมื่อเทียบกับการใช้เครื่องมือสำหรับแสดงหน้าจอทั่วไป ที่สำคัญคือ ตลอดเวลาที่ทำงานร่วมกันนั้น นักพัฒนาสามารถใช้การตั้งค่า ส่วนต่อขยาย และคีย์ลัดของตัวเองได้ นอกเหนือการทำงานร่วมกันแบบเรียลไทม์เพื่อแก้ไข หรือตรวจหาข้อบกพร่องแล้ว Live Share ยังอนุญาตให้สามารถใช้การสนทนาเสียงและใช้เทอร์มินัลและเซิร์ฟเวอร์ร่วมกันได้ด้วย

Apache Superset

ประเมิน

Apache Superset เป็นเครื่องมือ Business Intelligence (BI) ที่ดีเยี่ยมสำหรับการสำรวจและการแสดงแผนภาพข้อมูลจากแหล่งข้อมูลขนาดใหญ่อย่างดาต้าเลคหรือดาต้าแวร์เฮาส์ มันทำงานร่วมกับ Presto, Amazon Athena และ Amazon Redshift ได้เป็นอย่างดี อีกทั้งยังสามารถเชื่อมต่อเข้ากับระบบยืนยันตัวตนขององค์กรได้ด้วย มันถูกออกแบบมาให้เหมาะกับนักพัฒนาทั่วไปใช้สำรวจหาข้อมูลในการทำงานในแต่ละวัน คุณไม่จำเป็นต้องเป็นวิศวกรข้อมูลก็สามารถใช้เครื่องมือนี้ได้ อย่างไรก็ตาม Apache Superset ยังเป็นโครงการใหม่ที่อยู่ในช่วงการบ่มเพาะภายใต้ Apache Software Foundation (ASF) ซึ่งหมายความว่ามันยังไม่ได้รับการรับรองอย่างเป็นทางการโดย ASF

AsyncAPI

ประเมิน

มาตรฐานเปิดเป็นหนึ่งในรากฐานสำคัญของการสร้างระบบแบบกระจายตัว ยกตัวอย่างเช่น OpenAPI (ชื่อเดิม Swagger) ซึ่งเป็นข้อกำหนดที่ได้กลายเป็นมาตรฐานทางอุตสาหกรรมเพื่อใช้นิยาม API แบบ RESTful มันถือเป็นเครื่องมือสำคัญในความสำเร็จของไมโครเซอร์วิสเลยทีเดียว และมันก็ได้ก่อให้เกิดเครื่องมือจำนวนมากออกมาเพื่อช่วยในการจัดการ API ตั้งแต่กระบวนการบิลด์ การทดสอบ และการติดตามการใช้งาน อย่างไรก็ตามมันยังขาดมาตรฐานที่สำคัญอีกประเภทหนึ่งที่ระบบแบบกระจายตัวต้องการ นั่นคือมาตรฐานเพื่อใช้กำหนด API ที่ถูกขับเคลื่อนจากเหตุการณ์ที่เกิดขึ้น (event-driven)

โครงการโอเพนซอร์ส **AsyncAPI** เป็นความพยายามที่จะสร้างมาตรฐานสำหรับ API แบบอะซิงโครนัสขึ้นมา เพื่อใช้ในงานที่ขับเคลื่อนพฤติกรรมด้วยเหตุการณ์ รวมทั้งมีชุดเครื่องมือสำหรับการใช้งานมาตรฐานนี้ ข้อกำหนดของ AsyncAPI นั้นได้รับแรงบันดาลใจมาจากข้อกำหนดของ OpenAPI นั่นเอง ซึ่งมันพยายามจะอธิบาย API ที่ถูกขับเคลื่อนจากเหตุการณ์ในรูปแบบเอกสารที่เครื่องสามารถอ่านออก จากข้อกำหนดไม่ได้จำกัดการใช้งานไว้กับโปรโตคอลใดเป็นพิเศษ ฉะนั้นมันจึงเข้ากันได้กับหลายโปรโตคอล ตั้งแต่ MQTT, WebSocket และ Kafka เราตั้งตาคอยที่จะได้เห็นพัฒนาการของ AsyncAPI และความพร้อมที่มากขึ้นของชุดเครื่องมือที่ไว้คอยสนับสนุน

ConfigCat

ประเมิน

ConfigCat เป็นตัวเลือกที่น่าสนใจสำหรับใครที่กำลังมองหาบริการควบคุมการเปิด/ปิดฟีเจอร์ (feature toggle) ที่รองรับการปรับแต่งได้อย่างยืดหยุ่น (โปรดระลึกอีกครั้งว่าการใช้วิธีเปิด/ปิดฟีเจอร์แบบพื้นๆ ก็เพียงพอสำหรับงานส่วนมากแล้ว) เราให้คำจำกัดความมันว่า “คล้ายๆ กับบริการของ LaunchDarkly ด้วยราคาที่เป็นมิตรแต่ไม่หรูหราเท่า” ซึ่งเราพบว่าความสามารถที่มีให้ก็เพียงพอกับความต้องการของเราแล้ว ConfigCat รองรับการเปิด/ปิดฟีเจอร์แบบเรียบง่าย การแบ่งกลุ่มประเภทผู้ใช้งานและการทดสอบแบบ A/B พร้อมทั้งมีแผนแบบฟรีที่เป็นมิตรสำหรับการใช้งานในปริมาณไม่มากหรือสำหรับผู้เริ่มต้น

Gitpod

ประเมิน

คุณสามารถเริ่มพัฒนาซอฟต์แวร์ส่วนใหญ่ด้วยสองขั้นตอนง่ายๆ โดยเริ่มจากการเอาโค้ดลงมาจากที่จัดเก็บซอร์สโค้ดแล้วก็รันสคริปต์คำสั่งเดียวเพื่อบิลด์ แต่การจะจัดเตรียมสภาพแวดล้อมของเครื่องให้พร้อมต่อการพัฒนาบางครั้งยังเป็นเรื่องที่ยุ่งยากอยู่ **Gitpod** ช่วยแก้ปัญหานี้โดยการจัดเตรียมสภาพแวดล้อมบนคลาวด์ไว้ล่วงหน้า ทำให้เราพร้อมเขียนโค้ดได้ทันที และรองรับโครงการที่อยู่บน Github หรือ Gitlab ซึ่ง Gitpod มาพร้อมกับเว็บ IDE ที่มีพื้นฐานจาก Visual Studio Code โดยค่าตั้งต้นสภาพแวดล้อมเหล่านี้จะถูกสร้างขึ้นบนแพลตฟอร์ม Google Cloud ให้อัตโนมัติ อย่างไรก็ตามคุณสามารถนำไปติดตั้งเพื่อใช้งานภายในองค์กรได้ด้วย เรายังเห็นว่าเครื่องมือนี้จะช่วยอำนวยความสะดวกให้นักพัฒนาที่อยากสนับสนุนโครงการโอเพนซอร์สให้เริ่มต้นได้ง่ายขึ้น ทั้งนี้ยังคงต้องดูต่อไปว่า แนวทางนี้จะเหมาะกับการใช้ในสภาพแวดล้อมองค์กรมากน้อยเพียงใด

Gloo

ประเมิน

จากความนิยมที่เพิ่มขึ้นของ Kubernetes และ เซอร์วิสเมช ทำให้เครื่องมือประเภทเอพีไอเกตเวย์ (API gateway) กำลังประสบปัญหาไม่มีที่ยืนในระบบกระจายศูนย์แบบคลาวด์เนทีฟ เนื่องจากหลายๆ ความสามารถของเอพีไอเกตเวย์ (เช่น การควบคุมการจราจรของ API การจัดการความมั่นคง การจัดเส้นทาง และการเฝ้าสังเกตการณ์) นั้นก็มีในตัวควบคุมทางเข้า (ingress controller) ของคลัสเตอร์และเกตเวย์ เมชอยู่แล้ว **Gloo** เป็นเอพีไอเกตเวย์ขนาดเล็กที่ปรับตัวรับการเปลี่ยนแปลงนี้ โดยมันใช้ Envoy เป็นเทคโนโลยีจัดการเกตเวย์ โดยเสริมด้วยคุณค่าอื่นๆ เช่น สามารถแสดงภาพความสัมพันธ์ของ API เทียบกับผู้ใช้งานและแอปพลิเคชันที่เกี่ยวข้อง มีหน้าจอสำหรับนักดูแลระบบเพื่อใช้ควบคุมเกตเวย์ของ Envoy และทำหน้าที่เป็นตัวเชื่อมต่อการทำงานระหว่างเซอร์วิสเมชต่างๆ เช่น Linkerd, Istio และ AWS App Mesh แม้ว่าเวอร์ชันโอเพนซอร์สของ Gloo จะมีความสามารถพื้นฐานทั้งหมดที่เอพีไอเกตเวย์ควรจะมี แต่เวอร์ชันเพื่อการพาณิชย์นั้นสามารถควบคุมความมั่นคงได้สมบูรณ์กว่า เช่น มีระบบจัดการ API คีย์ หรือสามารถเชื่อมต่อกับ OPA ได้ Gloo เป็นเอพีไอเกตเวย์ขนาดเล็กที่น่าจับตามอง เนื่องจากสามารถเข้ากันได้ดีกับเทคโนโลยีและสถาปัตยกรรมแบบคลาวด์เนทีฟ ในขณะเดียวกันก็ช่วยหลีกเลี่ยงหลุมพรางของการใช้เอพีไอเกตเวย์แบบผิดๆ ที่มักจะถูกใช้เพื่อทำหน้าที่ร้อยเรียง API ต่างๆ เข้าด้วยกันให้กับผู้ใช้

Lens

ประเมิน

หนึ่งในจุดแข็งของ Kubernetes คือความยืดหยุ่น สามารถปรับแต่งได้แทบทุกอย่าง รวมทั้งมีกลวิธีการปรับแต่งที่ทำได้ผ่าน API ทั้งยังสามารถเขียนโปรแกรมไปกำกับและมีชุดคำสั่งคอมมานด์ไลน์ที่สามารถมองเห็นและควบคุมระบบผ่านการใช้ไฟล์ตั้งค่า อย่างไรก็ตาม จุดแข็งนี้อาจกลายเป็นจุดอ่อนได้เหมือนกัน เช่น เมื่อการติดตั้งมีความซับซ้อนสูง หรือตอนที่ต้องบริหารหลายๆ คลัสเตอร์ด้วยกัน ซึ่งมันเป็นเรื่องยากที่จะเห็นภาพของสถานะโดยรวมทั้งหมดผ่านการใช้ชุดคำสั่งคอมมานด์ไลน์และการใช้ไฟล์ตั้งค่าเท่านั้น

Lens จึงพยายามจะแก้ปัญหานี้ด้วยการสร้างเป็นเครื่องมือ IDE ที่สามารถแสดงสถานะปัจจุบันของคลัสเตอร์และปริมาณงานที่เกิดขึ้น สามารถแสดงค่าสถิติผลการชี้วัดออกเป็นกราฟ และสามารถปรับแต่งค่าต่างๆ ผ่านเครื่องมือแก้ไขข้อความที่มีมาให้ในตัว แทนที่จะสร้างเป็นแอปพลิเคชันที่มีหน้าจอกราฟิก Lens ถูกสร้างเป็นเครื่องมือที่นักดูแลระบบจะสามารถใช้งานได้ด้วย โดยเป็นเครื่องมือที่สามารถถูกเรียกจากคำสั่งคอมมานด์ไลน์เพื่อเปิดออกเป็นหน้าต่างที่ผู้ใช้สามารถเปลี่ยนหน้าไปมาได้ เครื่องมือนี้เป็นหนึ่งในหลายๆ วิธีที่พยายามลดความซับซ้อนในการจัดการระบบ Kubernetes แม้เรายังไม่เห็นผู้ชนะอย่างชัดเจน แต่ Lens ก็จัดสมดุลได้ดีระหว่างการเป็นเครื่องมือที่มีหน้าจอกราฟิก หรือเป็นแค่ชุดคำสั่งคอมมานด์ไลน์

Manifold

ประเมิน

Manifold เป็นเครื่องมือช่วยพินิจหาข้อบกพร่องในแมชชีนเลิร์นนิง โดยไม่ขึ้นกับโมเดลใดๆ เนื่องจากนักพัฒนาโมเดลมักจะใช้เวลาส่วนใหญ่ไปกับการพัฒนาและปรับปรุงโมเดลที่มีอยู่ มากกว่าการสร้างโมเดลใหม่ขึ้นมาอยู่แล้ว Manifold ช่วยปรับมุมมองไปอยู่ที่เรื่องของกลุ่มข้อมูลที่ใช้ แทนที่จะสนใจเรื่องของโมเดลแต่อย่างเดียว พอนำมาใช้ร่วมกับตัวชี้วัดประสิทธิภาพอื่นๆ ที่มีอยู่ ก็เป็นการส่งเสริมกันและกันได้ดี โดยมันช่วยแสดงให้เห็นว่า บุคลิกของข้อมูลนั้นส่งอิทธิพลอย่างไรต่อประสิทธิภาพของโมเดลได้บ้าง เราคิดว่า Manifold จะเป็นเครื่องมือที่มีประโยชน์ในการประเมินสภาพแวดล้อมของแมชชีนเลิร์นนิง

Sizzy

ประเมิน

การสร้างเว็บแอปพลิเคชันให้หน้าตาเหมือนกับที่ออกแบบไว้สำหรับทุกอุปกรณ์และทุกขนาดหน้าจอเป็นเรื่องที่ยุ่งยาก **Sizzy** เป็นซอฟต์แวร์บนคลาวด์ที่ช่วยรวบรวมหน้าจอหลายๆ ขนาดมาไว้บนหน้าต่างเดียวกัน การแสดงผลของแอปพิเคชันก็จะเกิดขึ้นบนหลายหน้าจอพร้อมกัน รวมทั้งการโต้ตอบกับแอปพลิเคชันใดๆ จะส่งผลไปที่หน้าจอต่างๆ ทันที จากประสบการณ์ของเราพบว่า การทดสอบแอปพลิเคชันด้วยวิธีการแบบนี้ช่วยให้จับข้อผิดพลาดได้ตั้งแต่เนิ่นๆ ก่อนที่เครื่องมือทดสอบการแสดงผล (visual regression tool) จะจับได้ในไปป์ไลน์เสียอีก อย่างไรก็ตาม ทีมพัฒนาของเราที่ได้ใช้เครื่องมือนี้มาสักพักบางส่วนยังชอบการใช้เครื่องมือนักพัฒนาที่มากับ Chrome มากกว่า

Snowpack

ประเมิน

Snowpack เป็นสมาชิกใหม่ที่น่าสนใจในกลุ่มของเครื่องมือบิลด์สำหรับ JavaScript พัฒนาการที่สำคัญที่ทำให้มันแตกต่างจากเครื่องมืออื่น คือความสามารถที่ทำให้การบิลด์แอปพลิเคชันที่สร้างจากเฟรมเวิร์คสมัยใหม่ เช่น React.js, Vue.js หรือ Angular ได้โดยไม่จำเป็นต้องใช้เครื่องมือสร้างแพกเก็จเป็นบันเดิล การตัดขั้นตอนการรวมไฟล์เพื่อสร้างแพ็คเกจออกไป ช่วยลดเวลาที่ใช้ในการรอผลและเพิ่มประสิทธิภาพการพัฒนาไปอย่างมาก เพราะเราสามารถเห็นผลลัพธ์ของการเปลี่ยนแปลงในเบราว์เซอร์แทบจะทันที เพื่อให้เวทย์มนต์นี้เป็นจริงได้ Snowpack ใช้วิธีการแปลงโมดูลที่พึ่งพิงใน *node_modules* ให้กลายเป็นไฟล์ JavaScript เพียงไฟล์เดียว แล้วเก็บไว้ที่ *web_modules* ซึ่งสามารถถูกนำเข้าเป็นโมดูลของ ECMAScript ได้ต่อไป สำหรับเบราว์เซอร์ IE11 และเบราว์เซอร์อื่นๆ ที่ไม่สนับสนุนการใช้โมดูลของ ECMAScript นั้นมีวิธีการเลี่ยงปัญหานี้อยู่เช่นกัน แต่เป็นที่น่าเสียดายที่ตอนนี้ยังไม่มีเบราว์เซอร์ใดที่สามารถนำเข้า CSS โดยใช้ JavaScript โดยตรงได้ เนื่องจากการใช้ CSS module ไม่ใช่สิ่งที่ตรงไปตรงมาเท่าไหร่

tfsec

ประเมิน

ความมั่นคงเป็นหน้าที่ของทุกคนไม่ใช่ของใครคนใดคนหนึ่ง และการตรวจพบความเสี่ยงตั้งแต่เนิ่นๆ ย่อมดีกว่าการพบปัญหาในภายหลังเสมอ เราได้กล่าวถึงเรื่อง การกำหนดโครงสร้างพื้นฐานด้วยโค้ด ซึ่ง Terraform เป็นตัวเลือกที่ชัดเจนมากสำหรับการจัดการสภาพแวดล้อมบนคลาวด์ ซึ่งตอนนี้เราก็ใช้ **tfsec** ร่วมด้วย มันเป็นเครื่องมือเชิงพิเคราะห์ที่ช่วยตรวจสอบไฟล์ของ Terraform เพื่อเตือนถึงความเสี่ยงทางความมั่นคงที่เป็นไปได้ โดยมันได้เตรียมกฎการตรวจสอบสำหรับการใช้งานร่วมกับผู้ให้บริการคลาวด์เจ้าต่างๆ ทั้ง AWS และ Azure ไว้ให้ เราชอบเครื่องมือใดก็ตามที่สามารถแบ่งเบาภาระในการตรวจสอบความเสี่ยงทางด้านความมั่นคงอยู่เสมอ และ tfsec ไม่ได้แค่เยี่ยมยอดในเรื่องการตรวจหาความเสี่ยงเหล่านี้เท่านั้น มันยังมีดีที่สามารถติดตั้งและใช้งานสะดวกอีกด้วย

เครื่องมือ

Lens เป็นหนึ่งในหลายๆ วิธีที่พยายามลดความซับซ้อนในการจัดการระบบ Kubernetes

(Lens)

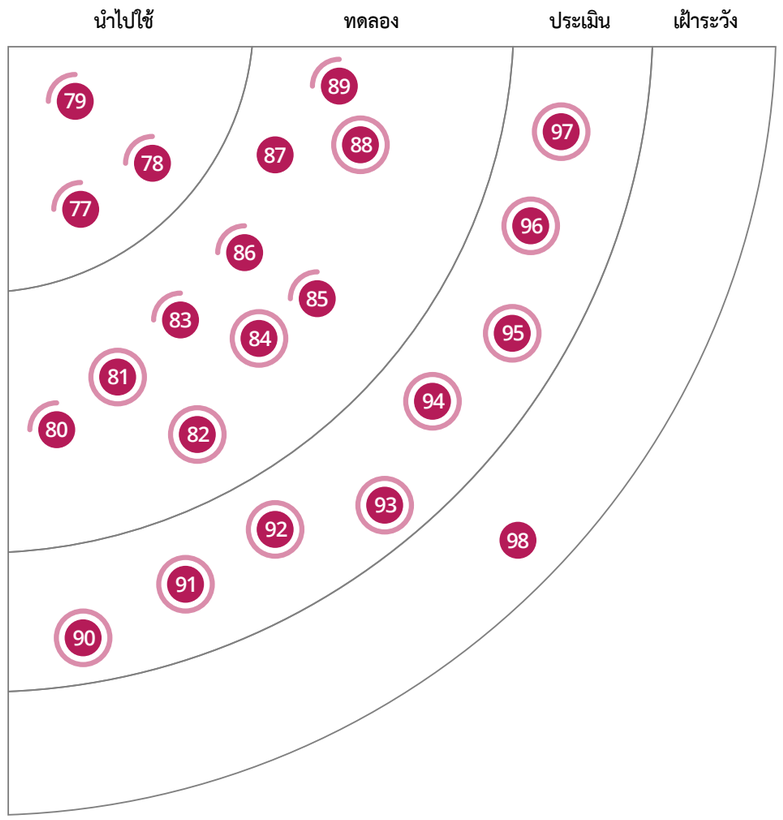
tfsec เป็นเครื่องมือเชิงพิเคราะห์ที่ช่วยตรวจสอบไฟล์ของ Terraform เพื่อเตือนถึงความเสี่ยงทางความมั่นคงที่เป็นไปได้

(tfsec)

TECHNOLOGY RADAR Vol. 22

ภาษาและเฟรมเวิร์ค

ภาษาและเฟรมเวิร์ค



React Hooks
นำไปใช้

React Hooks ได้นำเสนอวิธีใหม่ในการจัดการกลไกการจดจำสถานะ (Managing stateful logic) จากที่ผ่านมา React คอมโพเนนท์เหมาะกับการเขียนในรูปแบบฟังก์ชันมากกว่าการเขียนในรูปแบบคลาสอยู่แล้ว Hook จึงนำหลักการนี้มาปรับใช้ ทำให้เราสามารถจัดการค่าสถานะในรูปแบบฟังก์ชัน แทนการเขียนเป็นเมธอดของคลาส จากประสบการณ์ของเรา Hook ทำให้โค้ดเข้าใจง่าย และทำให้ความสามารถต่างๆ ถูกใช้ซ้ำระหว่างคอมโพเนนท์มากกว่าเดิม จากที่เราสามารถทดสอบการทำงานร่วมกับ Hook ได้ง่ายขึ้นแล้ว ผ่านการใช้ React Test Renderer และ React Testing Library ร่วมกับการมีชุมชนนักพัฒนาที่เติบโตขึ้นเรื่อยๆ มันจึงกลายเป็นทางเลือกเราชอบมากกว่าวิธีเดิม

React Testing Library
นำไปใช้

ด้วยประสบการณ์ของเราที่มากขึ้นและการเปลี่ยนแปลงที่รวดเร็วในของโลก JavaScript ทำให้คำแนะนำของเราก็เปลี่ยนตามไปด้วย **React Testing Library** เป็นไลบรารีสำหรับใช้ทดสอบฟรอนท์เอนด์ที่พัฒนาด้วย React มันเป็นตัวอย่างที่ดีของเฟรมเวิร์คที่ยิ่งใช้งานมากขึ้นเท่าไร ก็ยิ่งชัดเจนว่ามันเป็นทางเลือกที่ดีกว่า ทีมของเราพบความจริงที่ว่าชุดทดสอบที่ถูกเขียนด้วยเฟรมเวิร์คนี้มีความเปราะบางน้อยกว่าเฟรมเวิร์คอื่นๆ อย่าง Enzyme เพราะมันจะสนับสนุนให้ทดสอบความสัมพันธ์ระหว่างคอมโพเนนท์ใดๆ มากกว่าที่จะทดสอบวิธีการทำงานภายใน ซึ่งแนวคิดนี้ถูกคิดค้นโดย Testing Library ซึ่ง React Testing Library เป็นเพียงส่วนหนึ่งเท่านั้น มันยังมีไลบรารีอื่นๆ ในตระกูลนี้สำหรับใช้ทดสอบ Angular หรือ Vue.js อีกด้วย

Vue.js
นำไปใช้

Vue.js ได้กลายเป็นหนึ่งใน JavaScript เฟรมเวิร์คที่ประสบความสำเร็จด้านการถูกประยุกต์ใช้ ได้รับเสียงชื่นชมและความน่าเชื่อถือจากชุมชนนักพัฒนาฟรอนท์เอนด์ แม้จะมีเฟรมเวิร์คอื่นที่มีผู้ใช้งานจำนวนมากให้เลือกใช้ เช่น React.js แต่ด้วย API ที่ถูกออกแบบมาอย่างเรียบง่ายของ Vue.js ซึ่งแยกส่วนอย่างชัดเจนระหว่างไดเร็กทีฟกับคอมโพเนนท์ (แยกหนึ่งไฟล์ต่อหนึ่งคอมโพเนนท์โดยธรรมเนียม) พร้อมทั้งมีวิธีจัดการสถานะ (state management) ที่ง่ายกว่า จึงทำให้มันเป็นตัวเลือกที่น่าดึงดูดท่ามกลางบรรดาตัวเลือกอื่นๆ

CSS-in-JS
ทดลอง

เมื่อปี 2017 เราได้กล่าวถึงเทคนิคการเขียน **CSS ด้วย JavaScript** ว่าเป็นเทคนิคเกิดใหม่ ในปัจจุบันเทคนิคนี้ได้รับความนิยมแพร่หลายขึ้น เราเองก็ได้ใช้มันด้วยเช่นกัน ซึ่งจากบทพิสูจน์ในงานโปรดักชันที่เราดูแลเอง เราจึงมั่นใจที่

นำไปใช้
77. React Hooks
78. React Testing Library
79. Vue.js

ทดลอง
80. CSS-in-JS
81. Exposed
82. GraphQL Inspector
83. Karate
84. Koin
85. NestJS
86. PyTorch
87. Rust
88. Sarama
89. SwiftUI

ประเมิน
90. Clinic.js Bubbleprof
91. Deequ
92. ERNIE
93. MediaPipe
94. Tailwind CSS
95. Tamer
96. Wire
97. XState

เฝ้าระวัง
98. Enzyme

ภาษาและเฟรมเวิร์ค

Koin เป็นเฟรมเวิร์คในภาษา Kotlin ที่เข้ามาช่วยแก้ปัญหาหนึ่งที่พบเป็นประจำในการพัฒนาซอฟต์แวร์ ซึ่งก็คือการฉีดสิ่งพึ่งพา (dependency injection, DI)

(Koin)

NestJS เป็นเฟรมเวิร์คสำหรับ Node.js ที่ช่วยให้การพัฒนาปลอดภัยขึ้น ช่วยลดโอกาสการสร้างข้อผิดพลาดโดยไม่ได้ตั้งใจลง และรองรับภาษา TypeScript เป็นสำคัญ

(NestJS)

จะแนะนำต่อ styled components ซึ่งเป็นเฟรมเวิร์คที่เราได้กล่าวถึงในเรดาร์ฉบับที่แล้ว เป็นจุดเริ่มต้นศึกษาที่ดี นอกเหนือจากข้อดีต่างๆ แล้วเทคนิคนี้ก็มีข้อเสียอยู่บ้าง คือ การประมวลผลสไตล์ในช่วงรันไทม์บางครั้งสามารถทำให้เกิดอาการหน่วงที่สังเกตได้จากมุมผู้ใช้งาน Linaria เป็นเฟรมเวิร์คประเภทใหม่ออกมาเพื่อปิดข้อเสียดังกล่าว โดย Linaria ประยุกต์ใช้เทคนิคต่างๆ เข้าด้วยกันเพื่อย้ายความหน่วงช่วงรันไทม์ไปอยู่ในช่วงบิลด์ไทม์แทน อย่างไรก็ตามวิธีนี้ก็มีข้อเสียของตัวเองเช่นกัน โดยมันไม่รองรับการเปลี่ยนสไตล์ในช่วงรันไทม์บนเบราว์เซอร์รุ่นเก่าอย่าง IE11

Exposed

ทดลอง

จากการใช้งาน Kotlin อย่างหนัก ทีมพัฒนาของเรามีทักษะและประสบการณ์มากขึ้นกับเฟรมเวิร์คที่ออกแบบมาเพื่อ Kotlin โดยเฉพาะ แม้จะอยู่มานานแล้ว แต่ **Exposed** ก็ทำให้เราสนใจมันในฐานะการเป็นไลบรารีสำหรับแปลงออบเจกต์กับฐานข้อมูลเชิงสัมพันธ์ (object-relational mapper, ORM) ที่มีขนาดเล็ก โดยมันมีวิธีใช้สองวิธีด้วยกัน คือใช้ผ่านภาษาเฉพาะทางที่ครอบไวยกรณ์ภาษา SQL เอาไว้เพื่อให้สะดวกและปลอดภัย หรือใช้ผ่านรูปแบบออบเจกต์เพื่อการเข้าถึงข้อมูล (data access object, DAO) ก็ได้เช่นกัน มันรองรับความสามารถหลักที่ ORM ที่ดีควรจะมี เช่น การรองรับการโมเดลความสัมพันธ์แบบกลุ่มต่อกลุ่ม การดึงข้อมูลไว้ล่วงหน้า การเชื่อมความสัมพันธ์ระหว่างเอนติตี้ต่างๆ เข้าด้วยกัน เราชอบวิธีการทำงานภายในของมันด้วย โดยมันไม่ใช้เทคนิคการสร้างพร็อกซี่หรือใช้รีเฟลกชันเลย ซึ่งย่อมดีกับประสิทธิภาพมากกว่า

GraphQL Inspector

ทดลอง

GraphQL Inspector ช่วยให้เราสามารถเปรียบเทียบระหว่างโครงสร้างข้อมูลของ GraphQL สองตัวได้ จากที่เราเคยให้ข้อควรระวังเมื่อต้องเลือกใช้ GraphQL ไปนั้น เป็นเรื่องน่ายินดีที่จากนั้นเป็นต้นมา พัฒนาการด้านเครื่องมือสำหรับ GraphQL ได้เพิ่มสูงขึ้น ทีมที่ใช้ GraphQL สำหรับทำหน้าที่รวบรวมข้อมูลทางฝั่งเซิร์ฟเวอร์ ส่วนใหญ่ยังคงใช้เทคนิคนี้อย่างต่อเนื่อง และเมื่อนำ GraphQL Inspector มาใช้ร่วมกันบนไปป์ไลน์ CI แล้วเราพบว่ามันช่วยตรวจจับการเปลี่ยนแปลงที่มีโอกาสก่อให้เกิดปัญหา อันเนื่องจากโครงสร้างข้อมูลของ GraphQL ที่เปลี่ยนไปได้เป็นอย่างดี

Karate

ทดลอง

จากประสบการณ์ของเราพบว่าชุดทดสอบคือเอกสารยืนยันวิธีใช้งาน API ที่สำคัญที่สุด ฉะนั้นเราจึงมองหาเครื่องมือใหม่ๆ ที่อาจช่วยให้การทดสอบดีขึ้น **Karate** เป็นเฟรมเวิร์คสำหรับทดสอบ API ที่มีเอกลักษณ์อยู่ที่ความสามารถในการเขียนชุดทดสอบด้วยภาษา Gherkin โดยไม่จำเป็นต้องสร้างตัวแปลงภาษาเพื่อสร้างพฤติกรรมแต่อย่างใด โดยมันกำหนดภาษาเฉพาะทางขึ้นมาสำหรับใช้ทดสอบกับ API ที่อยู่ในรูปแบบโปรโตคอล HTTP โดยเฉพาะ ทีมของเราชอบเครื่องมือนี้เพราะโค้ดที่ได้ กลายเป็นเอกสารที่อ่านเข้าใจในทันที ซึ่งเราขอแนะนำให้ใช้ Karate เป็นชุดทดสอบในระดับชั้นบนๆ ตามวิธีการวางชุดทดสอบเป็นสามเหลี่ยมพีระมิด โดยทดสอบเฉพาะภาพรวมเท่านั้นไม่จำเป็นต้องลงรายละเอียดจนลึกเกินไป

Koin

ทดลอง

จากการที่ภาษา Kotlin ถูกใช้งานอย่างแพร่หลายเพื่อพัฒนาแอปพลิเคชันบนมือถือหรือใช้งานในฝั่งเซิร์ฟเวอร์ เครื่องมือที่เกี่ยวข้องกับทั้งคู่จึงมีพัฒนาการเปลี่ยนไปด้วย **Koin** เป็นเฟรมเวิร์คในภาษา Kotlin ที่เข้ามาช่วยแก้ปัญหาหนึ่งที่พบเป็นประจำในการพัฒนาซอฟต์แวร์ ซึ่งก็คือการฉีดสิ่งพึ่งพา (dependency injection, DI) เข้าสู่ออบเจกต์ แม้ใน Kotlin จะมีเฟรมเวิร์คทางเลือกอื่นให้เลือกอีกหลากหลาย แต่ทีมของเราก็ชอบความเรียบง่ายของ Koin ซึ่งมันหลีกเลี่ยงวิธีประกาศแบบใช้แอนโนเทชัน (annotation) และใช้การฉีดผ่านคอนสตรัคเตอร์ (constructor) หรือฉีดผ่านวิธีที่เลียนแบบการสร้างออบเจกต์แบบล่าช้า (lazy initialization) ของภาษา Kotlin เอง (ซึ่งทำให้การฉีดเกิดขึ้น ณ เวลาต้องการ) เราจะเห็นได้ว่าแนวทางที่ Koin ใช้นั้นตรงกันข้ามกับ Dagger ที่เป็นเฟรมเวิร์คเพื่อทำ DI สำหรับ Android ที่การฉีดเกิดขึ้น ณ ตอนคอมไพล์ไทม์เลย ทีมนักพัฒนาของเราชื่นชอบเฟรมเวิร์คนี้เพราะมีขนาดเล็กโดยธรรมชาติ และการที่มันมีความสามารถที่รองรับการทดสอบมาให้ในตัว

NestJS

ทดลอง

ความนิยมที่เพิ่มขึ้นของ Node.js มาพร้อมกับเทรนด์อย่าง การใช้ Node สำหรับทุกสิ่ง ที่ได้นำพาผู้คนให้เลือกพัฒนาแอปพลิเคชันทางธุรกิจด้วย Node.js จึงมีบ่อยครั้งที่เราพบปัญหาด้านการดูแลรักษาและการรองรับการขยายกับแอปพลิเคชันขนาดใหญ่ที่เขียนด้วย JavaScript **NestJS** เป็นเฟรมเวิร์คสำหรับ Node.js ที่ช่วยให้การพัฒนาปลอดภัยขึ้น ช่วยลดโอกาสการสร้างข้อผิดพลาดโดยไม่ได้ตั้งใจลง และรองรับภาษา TypeScript เป็นสำคัญ ดังนั้นมันจึงค่อนข้างชี้นำวิธีการเขียนให้เป็นในแนวทางที่มันคิดว่าดี รองรับการจัดการโค้ดโดยอิงหลักการ SOLID และมีสถาปัตยกรรมที่ได้รับอิทธิพลจาก Angular มาให้ในตัว เมื่อใดที่เราต้องการสร้างไมโครเซอร์วิสด้วย Node.js หนึ่งในเฟรมเวิร์คที่เรามักเลือกใช้เสมอ คือ NestJS เพราะมันสนับสนุนให้นักพัฒนาสามารถสร้างแอปพลิเคชันที่สามารถทดสอบได้ รองรับการขยาย จัดวางโค้ดให้อยู่กันแบบหลวมๆ และง่ายต่อการดูแล

PyTorch

ทดลอง

ตลอดมาทีมของพวกเรายังคงใช้และชื่นชอบเฟรมเวิร์คสำหรับแมชชีนเลิร์นนิงอย่าง **PyTorch** สำหรับหลายๆ ทีมแล้ว Pytorch เป็นตัวเลือกที่ดีกว่า TensorFlow เสียอีก ด้วยความที่ PyTorch เปิดให้เห็นถึงรายละเอียดการทำงานภายในของกระบวนการแมชชีนเลิร์นนิง ซึ่งต่างจาก TensorFlow ที่เลือกปิดไว้ โปรแกรมเมอร์จึงมีความคุ้นเคยกับโครงสร้างไวยากรณ์มากกว่า และการตรวจสอบหาข้อผิดพลาดของโปรแกรมก็ทำได้ง่าย ยิ่งในเวอร์ชันล่าสุดของ PyTorch มีการปรับปรุงประสิทธิภาพการทำงานของมันให้เร็วขึ้น และพวกเราก็ประสบความสำเร็จด้วยดีในการใช้มันในระดับโปรดักชัน

Rust
ทดลอง

ภาษา **Rust** ยังคงได้รับความนิยมอย่างต่อเนื่อง โดยที่ผ่านมาเราได้ถกเถียงอย่างร้อนแรงว่าภาษาไหนดีกว่ากันสำหรับงานระบบ ระหว่างการใช้ Rust เทียบกับ C++ ผสมกับ Go แต่เราก็ยังไม่ได้มีผู้ชนะที่ชัดเจน ถึงอย่างไรก็ตาม เราดีใจที่ได้เห็น Rust มีพัฒนาการสูงขึ้นอย่างมีนัยยะสำคัญเมื่อเทียบกับตอนที่เราได้กล่าวถึงมันในเรดาร์ครั้งก่อน มี API มาให้ในตัวที่นิ่งขึ้นและมากขึ้น ซึ่งรวมทั้ง API ที่สนับสนุนการเขียนโปรแกรมแบบอะซิงโครนัสขั้นสูงด้วย ยิ่งกว่านั้น Rust ยังเป็นแรงบันดาลใจในการออกแบบภาษาใหม่ๆ อีกด้วย เช่น ภาษา Move บน Libra ที่ได้หยิบยืมวิธีการบริหารจัดการทรัพยากรหน่วยความจำของ Rust ไปใช้ เพื่อทำให้มั่นใจว่าทรัพย์สินในรูปแบบดิจิทัลไม่สามารถถูกคัดลอกหรือละทิ้งโดยไม่ได้ตั้งใจ

Sarama
ทดลอง

Sarama เป็นไคลเอนต์ไลบรารีสำหรับเชื่อมต่อกับ Apache Kafka ที่พัฒนาด้วยภาษา Go หากคุณกำลังพัฒนา API ด้วยภาษา Go อยู่แล้ว จะพบว่าเราสามารถตั้งค่าและจัดการ Sarama ได้ค่อนข้างง่าย เนื่องจากมันไม่ขึ้นกับเนทีฟไลบรารีใดๆ Sarama ประกอบด้วย API สองแบบด้วยกัน ได้แก่ API ระดับสูงที่ใช้รับส่งข้อความสำเร็จรูป และ API ระดับล่างที่ใช้ควบคุมการรับส่งข้อมูลระดับไบต์

SwiftUI
ทดลอง

Apple เลือกเปลี่ยนแปลงครั้งใหญ่ด้วยการนำเสนอเฟรมเวิร์ค **SwiftUI** ออกมา มันเป็นเฟรมเวิร์คสำหรับใช้สร้างส่วนต่อประสานผู้ใช้ให้กับ macOS และ iOS เราชอบที่ SwiftUI ตัดสินใจทิ้งการจัดการส่วนต่อประสานผู้ใช้แบบเดิมที่ใช้ Interface Builder ของ Xcode ไปใช้วิธีการใหม่ที่ใช้การประกาศ โดยให้โค้ดเป็นศูนย์กลาง ที่การแก้ไขโค้ดจะเห็นผลการเปลี่ยนแปลงในทันที ทำให้ประสบการณ์ในการพัฒนาดีขึ้นกว่าเดิมมาก SwiftUI เฟรมเวิร์คได้รับแรงบันดาลใจหลายอย่างจาก React.js ที่เป็นผู้นำด้านการพัฒนาเว็บไซต์ในช่วงหลายปีมานี้ จากการออกแบบโดยสนับสนุนให้ใช้ตัวแปรที่ค่าเปลี่ยนแปลงไม่ได้ภายหลัง (Immutable values) ในโมเดลการแสดงผล และการรองรับกลไกการเปลี่ยนแปลงข้อมูลแบบอะซิงโครนัส ก็เข้ากันได้ดีกับแนวคิดการเขียนโปรแกรมเชิงรีแอกทีฟ (reactive programming) มันจึงกลายเป็นอีกทางเลือกแบบเนทีฟให้นักพัฒนานอกเหนือจากเฟรมเวิร์คในเชิงเดียวกันอย่าง React Native หรือ Flutter SwiftUI จะเป็นตัวแทนการพัฒนาส่วนต่อประสานผู้ใช้แห่งอนาคตของ Apple อย่างแน่นอน แม้มันยังใหม่มากแต่ก็ได้แสดงให้เห็นถึงประโยชน์และศักยภาพของตัวเอง จากการใช้งานของเราพบว่ามันมอบประสบการณ์ที่ดี และไม่ต้องใช้การเรียนรู้มากนัก อย่างไรก็ตามหากต้องการใช้มัน คุณควรคำนึงถึงลักษณะการใช้งานของผู้ใช้ก่อนเสมอเนื่องจากมันไม่รองรับ iOS เวอร์ชัน 12 และต่ำกว่า

Clinic.js Bubbleprof
ประเมิน

เมื่อต้องการปรับปรุงประสิทธิภาพของโค้ดเราให้ดียิ่งขึ้น การใช้เครื่องมือวิเคราะห์การใช้ทรัพยากร (profiling tool) เข้าช่วยจึงมีประโยชน์อย่างมาก เพราะมันช่วยระบุหาคอขวดของปัญหาหรือหาว่าความล่าช้าใดๆ ในระบบเกิดจากส่วนไหนของโค้ด ซึ่งปกติแล้วจะระบุได้ยากโดยเฉพาะอย่างยิ่งกับระบบที่ทำงานเป็นอะซิงโครนัส **Clinic.js Bubbleprof** แสดงให้เห็นเป็นภาพถึงงานอะซิงโครนัสต่างๆ ที่อยู่ในโปรเซสของ Node.js และนำความล่าช้าในแอพลิเคชันออกมาสร้างเป็นแผนที่ เราชอบเครื่องมือนี้เพราะมันช่วยให้นักพัฒนาสามารถระบุปัญหาและจัดลำดับความสำคัญของสิ่งที่ต้องปรับปรุงในโค้ดได้อย่างง่ายได้

Deequ
ประเมิน

ในโลกของวิศวกรรมข้อมูลยังมีช่องว่างของการนำเครื่องมือที่ส่งเสริมหลักปฏิบัติทางวิศวกรรมที่ดีมาใช้เมื่อเทียบกับโลกของการพัฒนาซอฟต์แวร์ทั่วไป ทีมของเราต้องประหลาดใจเมื่อพบว่ามีเครื่องมือเพียงไม่กี่ตัวในท้องตลาดที่สามารถช่วยให้การตรวจสอบคุณภาพของข้อมูลในขั้นตอนต่างๆ ของไปป์ไลน์ข้อมูลนั้นเป็นไปอย่างอัตโนมัติ สุดท้ายแล้วพวกเราลงเอยกับ **Deequ** ซึ่งเป็นไลบรารีสำหรับสร้างชุดทดสอบที่คล้ายกับการเขียนชุดทดสอบระดับยูนิตให้กับข้อมูล Deequ นั้นใช้ Apache Spark เป็นพื้นฐานการทำงานอยู่เบื้องล่าง แม้จะถูกเผยแพร่โดย AWS Labs แต่เราสามารถใช้มันได้กับทุกสภาพแวดล้อมนอกเหนือจากที่ AWS

ERNIE
ประเมิน

ในเรดาร์ฉบับก่อนเราได้พูดถึง BERT ซึ่งเป็นก้าวที่ยิ่งใหญ่ของวงการการประมวลผลภาษาธรรมชาติ ปีที่แล้ว Baidu ได้ปล่อย **ERNIE** 2.0 (Enhanced Representation through kNowledge IntEgration) ซึ่งทำคะแนนดีกว่า BERT ในเจ็ดงานจากมาตรฐานความเข้าใจภาษาของ GLUE และทั้งเก้างานในงานประมวลผลภาษาจีน (Chinese NLP) ERNIE คล้ายกับ BERT ที่มีโมเดลที่ผ่านการฝึกสอนมาล่วงหน้ามาให้ ซึ่งสามารถปรับแต่งความสามารถโดยการเพิ่มเลเยอร์ขาออก (output layers) ของนิวรัลเน็ตเวิร์กเพื่อสร้างเป็นโมเดลที่เป็นที่สุดสำหรับงานประมวลผลภาษาธรรมชาติ ERNIE แตกต่างจากโมเดลการฝึกสอนล่วงหน้าแบบดั้งเดิมตรงที่มันเป็นเฟรมเวิร์คการฝึกสอนแบบต่อเนื่อง (continual pretraining framework) กล่าวคือแทนที่จะฝึกสอนด้วยข้อมูลจำนวนน้อยๆ มันสามารถใช้งานฝึกสอน (pretraining tasks) ที่หลากหลายชนิดมาฝึกสอนอย่างต่อเนื่องเพื่อให้การเรียนรู้ภาษาของโมเดลเที่ยงตรงมากขึ้น เราตื่นเต้นมากกับความก้าวหน้าของการประมวลผลภาษาธรรมชาติในปัจจุบัน และคาดหวังว่าจะสามารถเอา ERNIE ไปใช้กับโครงการของเราได้ในอนาคต

MediaPipe
ประเมิน

MediaPipe คือเฟรมเวิร์คสำหรับการสร้าง MultiModal (เช่น วิดีโอ เสียง ข้อมูลจำแนกตามเวลา ฯลฯ) ที่ทำงานได้ข้ามแพลตฟอร์ม (ไม่ว่าจะเป็น Android, iOS, เว็บไซต์และอุปกรณ์ IoT) และเป็นไปป์ไลน์ทางแมชชีนเลิร์นนิงแบบประยุกต์ ซึ่ง MediaPipe ก็มาพร้อมกับความสามารถที่

ภาษาและเฟรมเวิร์ค

Deequ เป็นไลบรารีสำหรับสร้างชุดทดสอบที่คล้ายกับการเขียนชุดทดสอบระดับยูนิตให้กับข้อมูล ที่สามารถนำไปวางไว้ในแต่ละขั้นตอนของไปป์ไลน์ข้อมูลเพื่อให้การตรวจสอบคุณภาพเป็นไปอย่างอัตโนมัติ

(Deequ)

ERNIE มีโมเดลภาษาธรรมชาติที่ผ่านการฝึกสอนมาล่วงหน้ามาให้ ซึ่งสามารถปรับแต่งความสามารถโดยการเพิ่มเลเยอร์ขาออก (output layers) ของนิวรัลเน็ตเวิร์กเพื่อสร้างเป็นโมเดลใหม่ที่เป็นที่สุดสำหรับงานประมวลผลภาษาธรรมชาติ

(ERNIE)

ภาษาและเฟรมเวิร์ค

Tailwind ได้นำเสนอวิธีการที่น่าสนใจ โดยมันมีชุดคลาสพื้นฐานที่ทำหน้าที่ อำนวยความสะดวกไว้ให้ ซึ่งคลาสเหล่านี้ จะไม่ยุ่งกับการแสดงผลแต่จะมุ่งประสงค์ เพื่อให้ง่ายต่อการปรับแต่งเท่านั้น

(Tailwind CSS)

Wire ซึ่งเป็นเครื่องมือช่วยฉีดสิ่งที่พึ่งพาให้ โดยอาศัยการสร้างโค้ดขึ้นมาอย่าง อัตโนมัติ และผูกมันเข้าด้วยกันกับ คอมโพเนนท์ต่างๆ ตอนคอมไพล์ไทม์

(Wire)

หลากหลายไม่ว่าจะเป็น การตรวจจับใบหน้า ตรวจจับการเคลื่อนไหวของมือ การตรวจจับท่าทาง และการตรวจจับวัตถุ แม้ว่ามันมักจะถูกใช้งานบนแพลตฟอร์มมือถือเป็นหลักแต่ก็เริ่มเห็นการใช้งานได้บนเบราว์เซอร์แล้วเช่นกัน ซึ่งทำผ่านเทคโนโลยี WebAssembly และไลบรารี XNNPack ML Inference เข้าช่วยเหลือ เรากำลังทดลองใช้ MediaPipe เพื่อใช้กับงานความเป็นจริงเสริม (Augmented Reality) ซึ่งเท่าที่ได้ใช้งานมามันก็ทำงานได้อย่างดี

Tailwind CSS

ประเมิน

เครื่องมือและเฟรมเวิร์คสำหรับ CSS มักนำเสนอคอมโพเนนท์สำเร็จรูปให้ใช้เพื่อความสะดวกรวดเร็วในการพัฒนา อย่างไรก็ตามเมื่อเวลาผ่านไป การปรับแต่งคอมโพเนนท์เหล่านั้นกลับให้ความยุ่งยากกับเราแทน **Tailwind CSS** ได้นำเสนอวิธีการที่น่าสนใจไว้ โดยมันกำหนดชุดคลาสพื้นฐานที่ทำหน้าที่อำนวยความสะดวกต่างๆ ไว้ให้ ซึ่งคลาสเหล่านี้จะไม่ยุ่งกับการแสดงผลแต่จะมุ่งประสงค์เพื่อให้ง่ายต่อการปรับแต่งเท่านั้น จากชุดคลาสพื้นฐานที่ให้มาอย่างครอบคลุมการใช้งานทั่วไป ทำให้เราแทบไม่ต้องเขียนคลาสขึ้นมาเองเลย ซึ่งนำไปสู่โค้ดที่ง่ายต่อการดูแลในระยะยาว จากทั้งหมดที่กล่าวมานั้นดูเหมือนว่า Tailwind CSS พยายามเป็นเฟรมเวิร์คสำหรับสร้างคอมโพเนนท์การแสดงผลที่แบ่งสมดุลได้ดีทีเดียว เพื่อให้ปรับแต่งได้ง่ายและสามารถนำไปใช้ใหม่ที่อื่น

Tamer

ประเมิน

หากคุณต้องการเชื่อมข้อมูลจากฐานข้อมูลเชิงสัมพันธ์ (relational databases) ไปต่อกับทอปปิกของ Kafka เราอยากให้คุณลองพิจารณา **Tamer** มันให้นิยามตนเองว่าเป็น “ตัวเชื่อมต่อฐานข้อมูลผ่าน JDBC สำหรับใช้กับ Kafka ที่เชื่องมือ” แม้ว่าเฟรมเวิร์คนี้จะค่อนข้างใหม่มาก แต่เราพบว่า Tamer นั้นมีประสิทธิภาพมากกว่าตัวเชื่อมต่อฐานข้อมูลผ่าน JDBC ของ Kafka เองเสียอีก โดยเฉพาะเมื่อต้องทำงานกับข้อมูลที่มีปริมาณมากๆ

Wire

ประเมิน

ระหว่างคนในชุมชนนักพัฒนาของ Golang มีการแบ่งความเห็นออกเป็นสองฝ่ายที่เท่าๆ กันเมื่อพูดถึงการใช้เทคนิคการฉีดสิ่งที่พึ่งพา (dependency injection, DI) ซึ่งความไม่เห็นด้วยส่วนหนึ่งมาจากความสับสนระหว่างความต่างของรูปแบบกับเฟรมเวิร์คประเภทนี้ และจากที่นักพัฒนาส่วนใหญ่ที่มีพื้นฐานเป็นนักเขียนโปรแกรมระบบมาก่อนจะไม่ชอบการเสียค่าสิ้นเปลือง ณ รันไทม์ ที่จะเกิดจากรีเฟลคชันจนกระทั่งการมาถึงของ **Wire** ซึ่งเป็นเครื่องมือช่วยทำ DI โดยอาศัยการสร้างโค้ดขึ้นมาอย่างอัตโนมัติ และผูกมันเข้าด้วยกันกับคอมโพเนนท์ต่างๆ ตอนคอมไพล์ไทม์ Wire จึงไม่ต้องเสียค่าสิ้นเปลืองใด ณ รันไทม์เลย อีกทั้งการใช้กราฟแสดงการพึ่งพา (dependency graph) แบบคงที่ก็ยังง่ายต่อการทำความเข้าใจอีกด้วย ไม่ว่าคุณจะร้อยสิ่งที่พึ่งพาด้วยตัวเองหรือว่าจะใช้เฟรมเวิร์คเข้าช่วย เรายังคงแนะนำให้ใช้เทคนิคการฉีดสิ่งที่พึ่งพาอยู่เสมอ เพื่อส่งเสริมการออกแบบโค้ดที่แยกเป็นสัดส่วนและรองรับการทดสอบ

XState

ประเมิน

ในเรดาร์ฉบับก่อนๆ เราได้แนะนำไลบรารีสำหรับจัดการสถานะ (state management) ไปหลายตัว แต่ **XState** มีแนวคิดบางอย่างที่น่าสนใจต่างจากตัวอื่น มันเป็นเฟรมเวิร์คที่เรียบง่ายสำหรับสร้างเครื่องสถานะจำกัด (finite state machine) ด้วยภาษา JavaScript และ TypeScript โดยรองรับกับมาตรฐานเครื่องสถานะจำกัดของ W3C และมีความสามารถในการแสดงผลค่าสถานะในรูปแบบแผนภูมิได้อีกด้วย โดยเราสามารถใช้มันร่วมกับเฟรมเวิร์คยอดนิยม เช่น Vue.js, Ember.js, React.js และ RxJS หรือเราสามารถแปลงโมเดลของมันให้อยู่ในรูปแบบฟอร์แมตต่างๆ สิ่งหนึ่งที่เราพบว่ามีประโยชน์เมื่อใช้เครื่องสถานะจำกัดในบริบทอื่น (โดยเฉพาะในบริบทการเขียนตรรกะให้เกมส์) คือการได้เห็นค่าสถานะและเส้นทางการเปลี่ยนแปลงระหว่างสถานะเป็นภาพ ซึ่งเราชอบที่ XState ทำสิ่งนี้ได้เช่นกันผ่านตัวแสดงสถานะ ที่มีมาให้

Enzyme

เฝ้าระวัง

ไม่บ่อยนักที่เราจะย้ายเครื่องมือที่ถูกแทนที่แล้วมายังกลุ่มเฝ้าระวังในเรดาร์ แต่ทีมของเรามีความเชื่ออย่างยิ่งว่า React Testing Library ได้เข้าแทนที่ **Enzyme** ในการเป็นเครื่องมือสำหรับทดสอบ React คอมโพเนนท์ในระดับยูนิตแล้ว สืบเนื่องจากหลายทีมพบว่า Enzyme มุ่งเน้นการทดสอบโดยอ้างอิงข้อมูลภายในคอมโพเนนท์จนเกินไป ทำให้ชุดทดสอบมีความเปราะบางและยากต่อการจัดการ

ThoughtWorks®

ThoughtWorks คือบริษัทให้คำปรึกษาและวางแผนทางด้านเทคโนโลยีระดับโลก เราคือชุมชนที่รวมปัจเจกชนผู้หลงใหลในการทำในสิ่งที่เราเชื่อมากกว่า 7,000 คน จากสำนักงาน 43 แห่งที่กระจายอยู่ 14 ประเทศทั่วโลก ด้วยประสบการณ์มากกว่า 25 ปีที่เราใช้เทคโนโลยีสร้างความแตกต่างเพื่อช่วยแก้ปัญหาทางธุรกิจที่ซับซ้อนให้กับลูกค้า เมื่อการเปลี่ยนแปลงคือความแน่นอน เราเตรียมคุณให้พร้อมรับกับสิ่งไม่คาดฝัน

อยากติดตามข่าวสารและข้อมูลเชิงลึกล่าสุดเกี่ยวกับเรดาร์ใช่ไหม? ติดตามเราบนช่องทางโซเชียลที่คุณถนัด หรือ สมัครรับข่าวสารทางอีเมล์จากเรา

อยากติดตามข่าวสารและข้อมูลเชิงลึกล่าสุดเกี่ยวกับเรดาร์ใช่ไหม?

ติดตามเราบนช่องทางโซเชียลที่คุณถนัด
หรือ สมัครรับข่าวสารทางอีเมล์จากเรา

สมัครรับข่าวสารทันที!

ThoughtWorks®

thoughtworks.com/radar

#TWTechRadar