ThoughtWorks®

TECHNOLOGY RADAR VOL. 20

ทัศนคติการวิเคราะห์เทคโนโลยีในมุมมองที่ทันสมัย

thoughtworks.com/radar
#TWTechRadar

ผู้ร่วมสร้างสรรค์

เรดาร์เทคโนโลยีได้รับการสรรค์สร้างโดย
คณะกรรมการที่ปรึกษาด้านเทคโนโลยีของ ThoughtWorks

เรดาร์เทคโนโลยี ThoughtWorks ฉบับนี้ เขียนขึ้นมาจากการประชุม
คณะกรรมการที่ปรึกษาด้านเทคโนโลยี
ในเซินเจิ้น (Shenzhen) ในเดือนมีนาคม พ.ศ. 2562


Rebecca Parsons (CTO)


Martin Fowler (Chief Scientist)


Bharani Subramaniam


Erik Dörnenburg


Evan Bottcher


Fausto de la Torre


Hao Xu


Ian Cartwright


James Lewis


Jonny LeRoy


Ketan Padegaonkar


Lakshminarasimhan Sudarshan


Marco Valtas


Mike Mason


Neal Ford


Ni Wang


Rachel Laycock


Scott Shaw


Shangqi Liu


Zhamak Dehghani

เกี่ยวกับ เรดาร์เทคโนโลยี

พวกเรา Thoughtworkers ล้วนมีความหลงใหลในเทคโนโลยี เราสร้าง วิจัย ทดสอบ โอเพนซอร์ส ผลิตงานเขียน และมีเป้าหมายที่จะปรับปรุงเทคโนโลยีให้ดีขึ้นอย่างต่อเนื่องเพื่อทุกคน

“การผลักดันความเป็นเลิศทางซอฟต์แวร์ และการปฏิวัติวงการไอที” คือภารกิจของพวกเรา เราจัดทำและแบ่งปันเรดาร์เทคโนโลยีของ ThoughtWorks เพื่อสนับสนุนภารกิจดังกล่าว เรามีคณะกรรมการที่ปรึกษาประกอบไปด้วยผู้เชี่ยวชาญทางเทคโนโลยีของ ThoughtWorks พวกเขาประชุมพูดคุยกันต่อเนื่องอย่างสม่ำเสมอเกี่ยวกับยุทธศาสตร์ทางเทคโนโลยีของ ThoughtWorks และแนวทางของเทคโนโลยีที่มีบทบาทสำคัญต่ออุตสาหกรรมของเรา เรดาร์เทคโนโลยี คือ การรวบรวมผลการประชุมดังกล่าวให้อยู่ในรูปแบบที่เป็นประโยชน์กับทุกคนในวงการซอฟต์แวร์ ตั้งแต่นักพัฒนาจนถึง CTO ตัวเนื้อหานั้นเราตั้งใจให้เป็นข้อสรุปที่สั้นกระชับเท่านั้น เราแนะนำให้ทดลองเทคโนโลยีเหล่านี้ด้วยตัวเองสำหรับข้อมูลโดยละเอียด

เรดาร์เทคโนโลยี ใช้แผนภาพวงกลมเป็นแกนหลักในการนำเสนอข้อมูล แผนภาพวงกลมดังกล่าวแบ่งพื้นที่ออกเป็น 4 ส่วน เพื่อใช้จัดสรรเทคโนโลยีออกเป็น 4 กลุ่ม ได้แก่ เทคนิค (Techniques) เครื่องมือ (Tools) แพลตฟอร์ม (Platforms) และ ภาษาและเฟรมเวิร์ก (Language and Frameworks) ในกรณีที่เทคโนโลยีหนึ่งสามารถปรากฏได้ในหลายกลุ่มทางเราจะจัดลงในกลุ่มที่เหมาะสมที่สุด นอกจากนี้ในแต่ละส่วนของวงกลมถูกแบ่งออกเป็นวงแหวน 4 ชั้น เพื่อแสดงสถานะของเทคโนโลยีเหล่านี้ตามความคิดเห็นของเรา

หากสนใจประวัติเพิ่มเติมเกี่ยวกับ เรดาร์เทคโนโลยี สามารถดูเพิ่มเติมได้ที่ thoughtworks.com/radar/faq

ภาพรวม ของเรดาห์

1 นำไปใช้ (ADOPT)

เราสนับสนุนให้อุตสาหกรรมนำเทคโนโลยีเหล่านี้ไปใช้ได้เลย เราได้นำไปใช้แล้วในโปรเจกต์ต่างๆ ตามความเหมาะสม

2 ทดลอง (TRIAL)

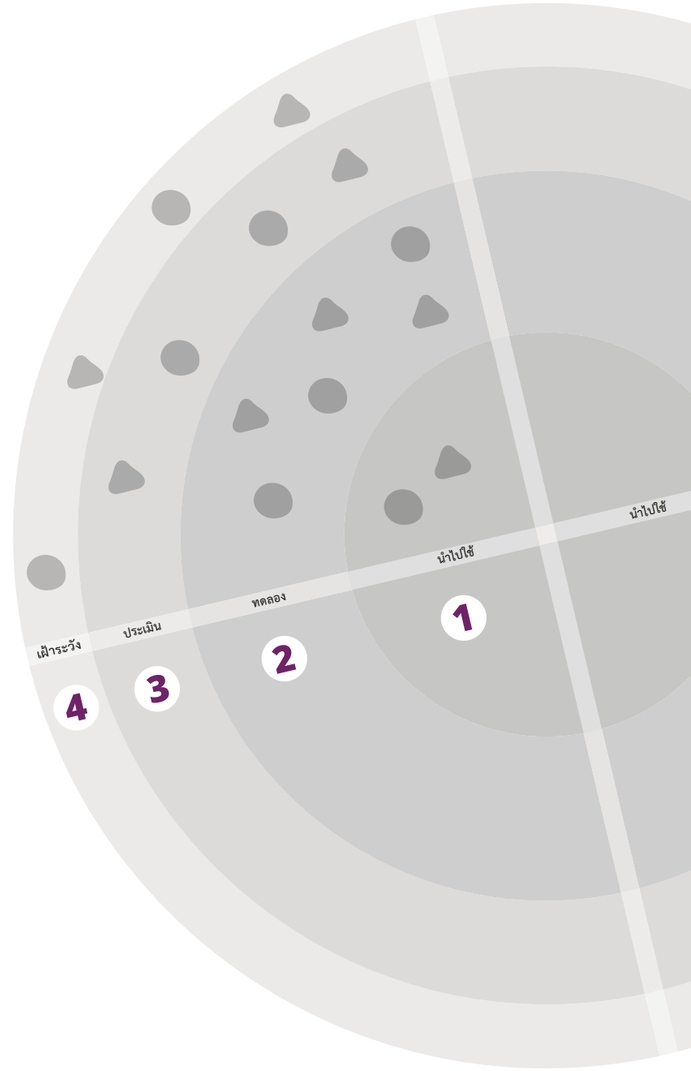
คุ้มค่าที่จะติดตาม ควรทำความเข้าใจว่าจะสร้างความสามารถทางด้านนี้ได้อย่างไร องค์กรขนาดใหญ่ควรทดลองใช้เทคโนโลยีนี้ในโปรเจกต์ที่สามารถรองรับความเสี่ยงได้

3 ประเมิน (ASSESS)

คุ้มค่าที่จะสำรวจเพื่อทำความเข้าใจว่ามีผลกระทบต่อองค์กรอย่างไร

4 เฝ้าระวัง (HOLD)

ควรดำเนินการด้วยความระมัดระวัง



มาใหม่หรือเปลี่ยนแปลง
คงเดิม

เทคโนโลยีที่มาใหม่หรือเทคโนโลยีที่มีการเปลี่ยนแปลงอย่างสำคัญหลังจากเรดาร์ครั้งก่อนหน้าจะถูกแสดงด้วยสามเหลี่ยม ในขณะที่เทคโนโลยีที่คงเดิมไม่เปลี่ยนแปลง ถูกแสดงด้วยวงกลม

เรดาร์ของเรามองไปข้างหน้าเสมอ เพื่อเปิดพื้นที่ของเรดาร์ให้กับเทคโนโลยีใหม่ๆ เทคโนโลยีที่ไม่มีความเคลื่อนไหวจะค่อยๆ จางลง ซึ่งการจางลงไม่เกี่ยวกับคุณค่าของเทคโนโลยีนั้นๆ แต่เนื่องด้วยพื้นที่ของ เรดาร์เทคโนโลยี นั้นมีจำกัด

มีอะไรใหม่?

ประเด็นที่โดดเด่นในฉบับนี้

รูปแบบข้อมูลที่แปรเปลี่ยน

เมื่อทศวรรษที่แล้ว เมื่อพูดถึงข้อมูลทุกคนก็นึกถึงฐานข้อมูลเชิงสัมพันธ์ (relational database) ในตอนนี้ข้อมูลอยู่ในหลากหลายรูปแบบ ได้แก่ แบบ NoSQL แบบเชิงอนุกรมเวลา แบบ SQL store เช่น CockRoachDB, Spanner ที่มี consistency จากฐานข้อมูลที่กระจายตัวอยู่ทั่วโลก และ แบบ event stream ที่อนุญาตให้คิวรี่ (query) จากไฟล์ log ที่รวมกันอยู่ได้โดยตรง ปรากฏการณ์นี้เกิดจากความต้องการทางธุรกิจที่ต้องการการตอบสนองแบบทันทีจากแหล่งข้อมูลที่ขนาดใหญ่ขึ้น หลากหลายขึ้น และรวดเร็วขึ้น ซึ่งสำหรับนักพัฒนาแล้ว ความท้าทายที่เกิดขึ้นคือการทำความเข้าใจข้อดีข้อเสียของการใช้ข้อมูลในแต่ละแบบ นักออกแบบและนักพัฒนาควรมองหาศักยภาพใหม่ๆ ที่เกิดขึ้นที่มาจากเครื่องมือและกระบวนทัศน์ (paradigm) ใหม่ๆ ในขณะเดียวกันก็ต้องระมัดระวังไม่ใช้เครื่องมือใหม่แบบผิดๆ เพราะความคุ้นเคยกับเครื่องมือที่เคยใช้ เราต้องยอมรับว่าเรากำลังอยู่ในช่วงความเปลี่ยนแปลงของเทคโนโลยีข้อมูล และเป็นช่วงเวลาที่แนวทางและเครื่องมือที่เหมาะสมกำลังถูกมองหาอยู่

ระบบนิเวศแห่งเทอร์ราฟอร์ม

นักพัฒนารักระดับนามธรรม (abstraction layer) เพราะเหตุผลคือมันห่อหุ้ม (encapsulate) ความซับซ้อนไว้ ทำให้นักพัฒนามุ่งความสนใจให้กับปัญหาในระดับที่สูงขึ้นไปได้มากกว่า เราเห็นพัฒนาการนี้ในเรดาร์หลายๆ ฉบับ เช่น วิธีที่ทีมพัฒนาจัดการกับการทำงานร่วมกันระหว่างเทคโนโลยีคลาวด์และคอนเทนเนอร์ ในตอนแรกมุ่งเน้นไปที่การสร้าง Docker สำหรับการจัดการคอนเทนเนอร์และระบบนิเวศรอบตัวมันขึ้นมา แล้วจึงสร้าง Kubernetes สำหรับการจัดการกับระบบที่ประกอบไปด้วยคอนเทนเนอร์ในระดับที่ใหญ่ขึ้น ตอนนี้ความเคลื่อนไหวหลักที่เราเห็นคือการสร้างโครงสร้างพื้นฐานในรูปแบบโค้ด (infrastructure as code) โดยทั่วไป และระบบนิเวศของ Terraform โดยเฉพาะ ถึงแม้เราจะแนะนำเครื่องมือที่นอกเหนือจาก Terraform แล้ว แต่มันเป็นที่นิยมมากในชุมชนของผู้ให้บริการเซอร์วิสต่างๆ หัวข้อที่โดดเด่นในเรดาร์ฉบับนี้ ได้แก่ Terratest สำหรับการทดสอบโค้ดโครงสร้างพื้นฐาน และ provider ตัวใหม่ของ GoDC ที่ทำให้คุณสร้างและตั้งค่า GoCD ได้ด้วย Terraform

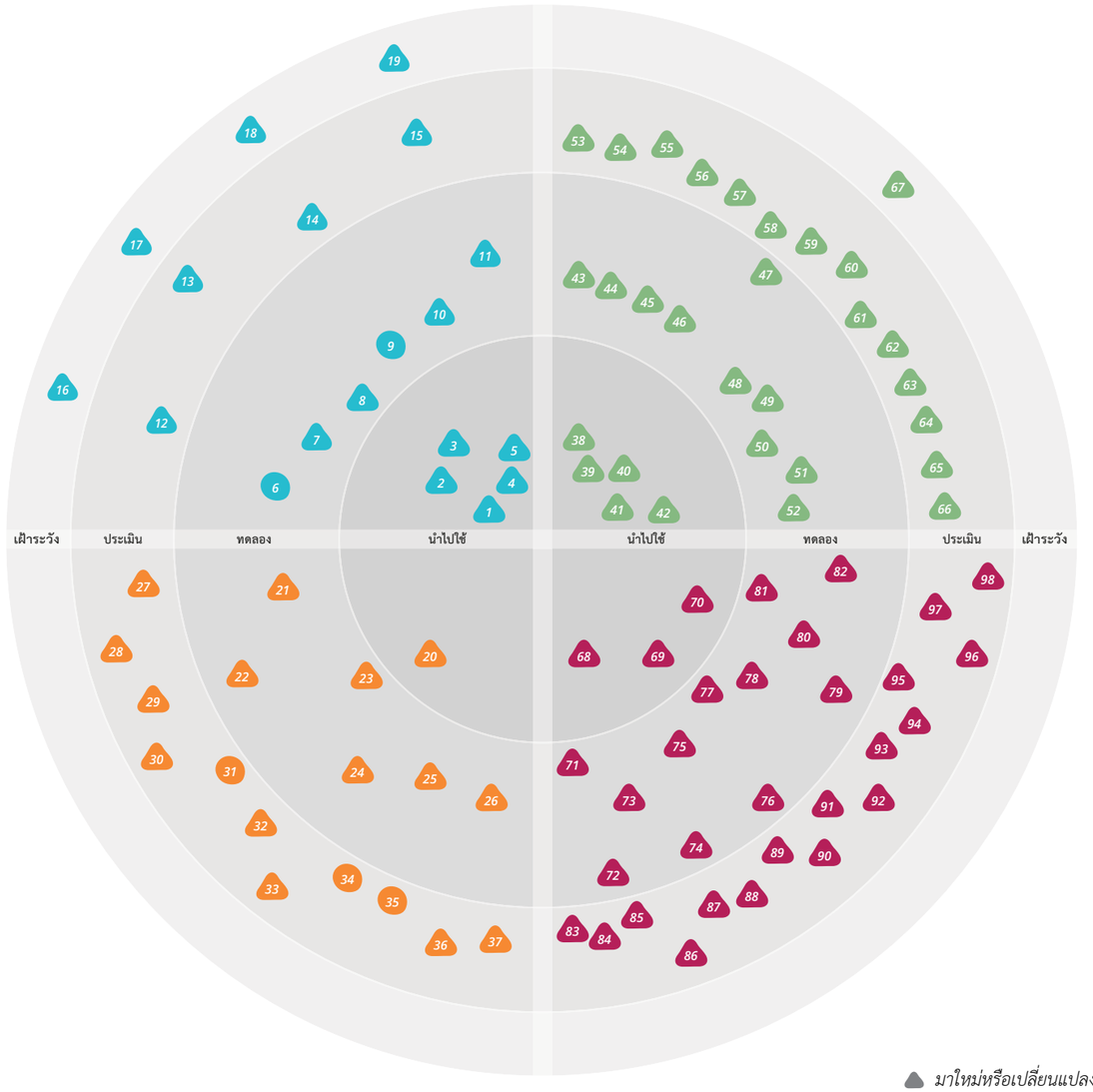
คอทลินตะกายดาว

Kotlin เป็นภาษาโอเพนซอร์สที่ยังคงได้รับความสนใจสูงในเรดาห์ของเราเพราะรูปแบบการใช้งานของมันขยายออกไปไกลเกินกว่าการพัฒนาแอนดรอยด์แต่เพียงอย่างเดียวแล้ว นักพัฒนาที่ JetBrain ไม่ชอบทางเลือกที่มีอยู่ในโลกของภาษาเขียนโปรแกรม พวกเขาจึงสร้าง Kotlin ขึ้นและปล่อยออกมาสู่สาธารณะเป็นโอเพนซอร์สในเวลาต่อมา Kotlin เป็นที่ชื่นชอบของนักพัฒนาหลากหลายรูปแบบ มันถูกใช้ในหลายๆแพล็ตฟอร์ม และถูกใช้สร้างเครื่องมือหลาย ๆ อย่างทั้งในฐานะภาษาสำหรับใช้งานทั่วไปและภาษาที่ใช้ในจุดประสงค์อื่น นอกจากนี้มันยังปรากฏตัวในเรดาร์ของเราบ่อยขึ้นเรื่อยๆ และในหลายๆ ทีมในโปรเจ็คของเรา (เช่น Ktor, MockK, Detekt, HTTP4K) มันน่าตื่นตาตื่นใจที่จะได้เห็นภาษาที่จะประสบความสำเร็จได้เพราะมันถูกออกแบบอยู่บนการใช้งานจริง เป็นเครื่องมือที่สมัยและมีระบบนิเวศที่ขยายตัวอย่างรวดเร็ว

ความซับซ้อนของระบบที่ไหลออก

การมาถึงจุดกำเนินของ “everything as code” ทำให้ทุกอย่างๆ ตั้งแต่ โครงสร้างพื้นฐาน ระบบความปลอดภัย การกำหนดและทำตามข้อยินยอม (compliance) และการดำเนินการของระบบ (operation) ที่ก่อนหน้านี้ยากที่จะเปลี่ยนแปลง ซึ่งทำให้ต้องจัดการได้ด้วยการเขียนโปรแกรม และนั่นทำให้นักพัฒนาสามารถนำข้อปฏิบัติทางวิศวกรรมที่เหมาะสมมาใช้ได้ แต่ถึงอย่างนั้นเรายังเห็นการสร้าง subsystem ที่มีการตั้งค่าที่ซับซ้อน หรือการใช้เครื่องมือจัดการระบบ (orchestration tool) ที่อาศัยการแสดงผลด้วยภาพมากเกินไป โลจิค (logic) ที่แทรกเข้าไปอยู่ในไฟล์ตั้งค่า, ลักษณะการใช้ภาษาที่น่าสยดสยองเพื่อให้เขียนโลจิคใน YAML ได้ และเฟรมเวิร์กสำหรับจัดการหลายๆ ตัวที่ใช้เทคโนโลยีหลากหลายแบบ ในยุคของ การโปรแกรมหลากภาษา (polyglot programming), โครงสร้างพื้นฐานในรูปแบบโค้ด (infrastructure as code) และ *x*-as-a-service ทีมพัฒนาจึงลงเอยโดยการใช้องค์ประกอบหลายๆ ตัวมารวมกันในระบบเดียว ดังนั้นโลจิคที่ควรจะอยู่ภายในขอบเขตของระบบก็แทรกซึมหลุดเข้าไปในเครื่องมือจัดการระบบ ไฟล์ตั้งค่า และส่วนเชื่อมต่ออื่นๆ ถึงแม้ว่าจะจำเป็นต้องทำแบบนี้ในบางครั้ง แต่เราแนะนำให้ทีมพิจารณาอย่างระมัดระวังด้วยการเก็บโค้ดเหล่านั้นไว้ในที่ที่นักพัฒนาสามารถทำการทดสอบ ควบคุมเวอร์ชั่น ทำ continuous integration และนำข้อปฏิบัติทางวิศวกรรมที่เหมาะสมมาใช้ พร้อมทั้งพยายามอย่าใส่โลจิคทางธุรกิจลงไปในไฟล์ตั้งค่า (และหลีกเลี่ยงเครื่องมือที่ทำเช่นนั้น) และพยายามให้เครื่องมือจัดการระบบไม่กลายเป็นโครงหลักหลักของระบบของคุณ

เรดาร์เทคโนโลยี



เทคนิค

นำไปใช้
1. Four key metrics
2. Micro frontends
3. Opinionated and automated code formatting
4. Polyglot programming
5. Secrets as a service

ทดลอง
6. Chaos Engineering
7. Container security scanning
8. Continuous delivery for machine learning (CD4ML) models
9. Crypto shredding
10. Infrastructure configuration scanner
11. Service mesh

ประเมิน
12. Ethical OS
13. Smart contracts
14. Transfer learning for NLP
15. Wardley mapping

เฝ้าระวัง
16. Productionizing Jupyter Notebooks
17. Puncturing encapsulation with change data capture
18. Release train
19. Templating in YAML

แพลตฟอร์ม

นำไปใช้
20. Contentful

ทดลอง
21. AWS Fargate
22. EVM beyond Ethereum
23. InfluxDB
24. Istio
25. Kafka Streams
26. Nomad

ประเมิน
27. CloudEvents
28. Cloudflare Workers
29. Deno
30. Hot Chocolate
31. Knative
32. MinIO
33. Prophet
34. Quorum
35. SPIFFE
36. Tendermint
37. TimescaleDB

เฝ้าระวัง

เครื่องมือ

นำไปใช้
38. Cypress
39. Jupyter
40. LocalStack
41. Terraform
42. UI dev environments

ทดลอง
43. AnyStatus
44. AVA
45. batect
46. Elasticsearch LTR
47. Helm
48. InSpec
49. Lottie
50. Stolon
51. TestCafe
52. Traefik

ประเมิน
53. Anka
54. Cage
55. Cilium
56. Detekt
57. Flagr
58. Gremlin
59. Honeycomb
60. Humio
61. Kubernetes Operators
62. OpenAPM
63. Systems
64. Taurus
65. Terraform provider GoCD
66. Terratest

เฝ้าระวัง
67. Handwritten CloudFormation

ภาษาและเฟรมเวิร์ก

นำไปใช้
68. Apollo
69. MockK
70. TypeScript

ทดลอง
71. Apache Beam
72. Formik
73. HiveRunner
74. joi
75. Ktor
76. Laconia
77. Puppeteer
78. Reactor
79. Resilience4j
80. Room
81. Rust
82. WebFlux

ประเมิน
83. Aeron
84. Arrow
85. Chaos Toolkit
86. Dask
87. Embark
88. fastai
89. http4k
90. Immer
91. Karate
92. Micronaut
93. Next.js
94. Pose
95. react-testing-library
96. ReasonML
97. Taiko
98. Vapor

เฝ้าระวัง

เทคนิค

Four key metrics

นำไปใช้

รายงาน State of DevOps ได้มุ่งความสนใจไปที่ข้อมูล (data driven) และ การวิเคราะห์ทางสถิติขององค์กรที่ทำงานแล้วมีประสิทธิภาพสูง ผลจากงานวิจัยซึ่งใช้เวลาหลายปีและตีพิมพ์ใน Accelerate ชิ้นนี้ พบการเชื่อมโยงโดยตรงระหว่าง ประสิทธิภาพขององค์กรและประสิทธิภาพของการส่งมอบซอฟต์แวร์ นักวิจัยพบว่ามีมีตัวชี้วัดสำคัญสี่ตัวที่แสดงถึงความแตกต่างระหว่าง ประสิทธิภาพการทำงานต่ำ กลาง และสูง ซึ่งก็คือ ระยะเวลาที่ต้องรอระหว่างการเปลี่ยนแปลง (lead time) ความถี่ของการดิพลอย (deployment frequency) เวลาเฉลี่ยในการกู้คืน (mean time to restore) และ เปอร์เซ็นต์ของการเปลี่ยนแปลงที่จะทำให้ระบบล้มเหลว (change fail percentage) เราพบว่าสี่ค่านี้ อาจดูเรียบง่าย แต่ทรงพลังสำหรับการเป็นเครื่องมือเพื่อหัวหน้าและทีมจะได้โฟกัสไปที่การวัดค่าและพัฒนาสิ่งที่สำคัญจริงๆ ซึ่งที่แรกๆ ที่ควรจะเริ่มคือการติดตั้ง build pipeline เพื่อคุณจะได้ตัวชี้วัดดังที่กล่าวมา และยังทำให้เห็นถึงสายธารคุณค่าการส่งมอบซอฟต์แวร์ (software delivery value stream) นอกจากนั้นไปปไลน์ GoCD มีความสามารถที่จะวัดตัวชี้วัดสำคัญทั้งสี่ตัวที่กล่าวมาโดยดูได้จาก GoCD analytics

Micro frontends

นำไปใช้

ที่ผ่านมาเราได้เห็นคุณประโยชน์ของการใช้ไมโครเซอร์วิส ซึ่งช่วยในการดูแลและการขยายขนาดของเซอร์วิส สามารถทำให้ส่งมอบแต่ละเซอร์วิสได้เป็นอิสระต่อกัน น่าเสียดายที่เรายังคงเห็นหลายๆทีมสร้างฟรอนท์เอนด์ (frontend) ที่มีขนาดใหญ่ที่โยงใยกับโปรแกรมต่างๆบนบราวเซอร์ และทั้งหมดก็อาศัยอยู่บนแบ็คเอนด์ (backend) ขนาดใหญ่อีกทีหนึ่ง ตั้งแต่เรานิยามไมโครฟรอนท์เอนด์ (micro frontends) ให้เป็นเทคนิคในการออกแบบโครงสร้างเว็บไซต์ เราพบว่าเทคนิคไมโครฟรอนท์เอนด์ให้ประสบการณ์ที่ดีในจากการนำไปใช้และเรายังค้นพบรูปแบบต่างๆ ในการสร้างฟรอนท์เอนด์ ทั้งนี้สังเกตได้ว่าโค้ดถูกย้ายจากฝั่งเซิร์ฟเวอร์ไปยังเว็บบราวเซอร์มากขึ้นเรื่อยๆ อย่างไรก็ตามจนถึงตอนนี้เรายังไม่พบแนวทางที่ชัดเจนในการใช้เว็บคอมโพเนนท์ สำหรับเทคนิคนี้

Opinionated and automated code formatting

นำไปใช้

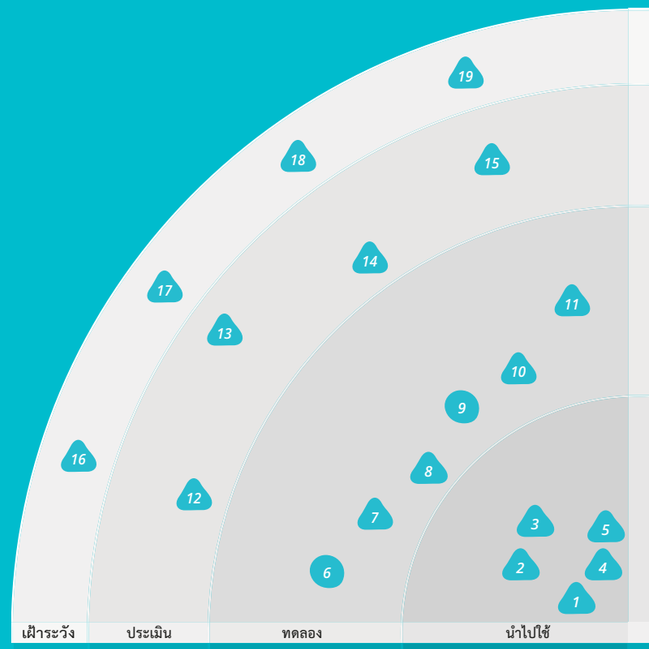
ตั้งแต่เราพอจำได้ การใช้ code format แบบไหนนั้นเป็นเรื่องของรสนิยมส่วนตัว นโยบายบริษัท และจากการโต้เถียงอันดุเดือด ในที่สุดก็เหมือนอุตสาหกรรมนี้ก็เริ่มเหนื่อยสำหรับข้อถกเถียงที่ดูยาวนานไม่สิ้นสุด และทีมก็เหมือนจะใช้เวลามากมายไปกับการพูดคุยถกเถียง และจากนั้นก็หันไปใช้ tools ที่จัดการเรื่อง format แม้คุณจะไม่เห็นด้วย 100% กับ format ที่มีมาให้ของเครื่องมือแต่ละตัว การมุ่งเน้นไปที่การทำให้โค้ดทำงานได้ก่อนแล้วค่อยตามมาด้วยเรื่อง format น่าจะเป็นสิ่งที่ทีมได้ประโยชน์มากกว่าการมุ่งเน้นไปที่เรื่อง format อย่างเดียว Prettier ได้เสียงโหวตส่วนใหญ่จากพวกเราสำหรับ JavaScript ส่วน tools อื่นๆ อย่าง Black สำหรับ Python และดูเหมือนในภาษาส่วนใหญ่ เริ่มที่จะมีการใส่เข้าไปในตัวภาษาตั้งแต่แรกอย่าง Golang และ Elixir จุดประสงค์หลักๆ ในที่นี้ก็คือ ให้เราไม่ต้องไปเสียเวลามากมายเพื่อถกเถียงกันว่าควรตั้งกฎแบบไหน แต่หันไปใช้เครื่องมือที่กำหนดกฎมาให้แล้ว ซึ่งใช้เวลาในการกำหนดค่าน้อยที่สุดและใช้งานแบบอัตโนมัติได้ โดยเฉพาะใช้เป็น pre-commit hook

นำไปใช้
1. Four key metrics
2. Micro frontends
3. Opinionated and automated code formatting
4. Polyglot programming
5. Secrets as a service

ทดลอง
6. Chaos Engineering
7. Container security scanning
8. Continuous delivery for machine learning (CD4ML) models
9. Crypto shredding
10. Infrastructure configuration scanner
11. Service mesh

ประเมิน
12. Ethical OS
13. Smart contracts
14. Transfer learning for NLP
15. Wardley mapping

เฝ้าระวัง
16. Productionizing Jupyter Notebooks
17. Puncturing encapsulation with change data capture
18. Release train
19. Templating in YAML



เทคนิค

การเลือกภาษาที่ถูกต้องและเหมาะสมกับงานสามารถเพิ่มประสิทธิภาพของงานที่ทำได้มากขึ้น การส่งเสริมระบบนิเวศ (ecosystem) ของภาษาที่ใช้มีความสำคัญเพื่อที่จะเร่งกระบวนการส่งมอบซอฟแวร์ที่เร็วขึ้น และทำให้นักพัฒนามีเครื่องมือที่เหมาะสมในการแก้ปัญหา

(Polyglot programming)

การพัฒนาโมเดล Machine Learning แบบดั้งเดิมมักมีระยะเวลานานระหว่าง การเทรนนิ่งโมเดล (training models) และการนำไปใช้งานบนโปรดักชั่น ทำให้โมเดลในโปรดักชั่นล้าช้าจากข้อมูลปัจจุบัน การนำหลักการทำงาน continuous delivery มาใช้ทำ machine learning สามารถแก้ปัญหาเหล่านี้ได้

(Continuous delivery for machine learning (CD4ML) models)

Polyglot programming

นำไปใช้

เราได้ใส่ Polyglot Programming ไว้ในกลุ่มทดลอง ในเทคเรดาร์ครั้งก่อน เราแนะนำว่าการเลือกภาษาที่ถูกต้องและเหมาะสมกับงานสามารถเพิ่มประสิทธิภาพของงานที่ทำได้มากขึ้น (Productivity) อย่างเห็นได้ชัด และได้มีภาษาใหม่ๆ เข้ามาซึ่งมันคุ้มจะลองพิจารณาดู เราต้องการที่จะยกประเด็นขึ้นมาอีกครั้ง เพราะเราได้เห็นการผลักดันที่จะวางมาตรฐานของภาษาโดยเกิดทั้งจากฝั่งนักพัฒนาซอฟแวร์เองและฝั่งขององค์กรเอง ในขณะที่เราทราบดีว่าการไม่มีข้อจำกัดในภาษานั้นสามารถสร้างปัญหามากกว่าการแก้ปัญหา การส่งเสริมภาษาที่สนับสนุน ระบบนิเวศ (ecosystem) ที่แตกต่างหรือ ภาษาที่มีคุณสมบัติ สำคัญสำหรับองค์กรเพื่อที่จะเร่งกระบวนการส่งมอบซอฟแวร์ที่เร็วขึ้น และสำหรับนักพัฒนาซอฟแวร์เพื่อที่จะเป็นเครื่องมือไว้สำหรับแก้ปัญหาได้อย่างตรงจุด

Secrets as a service

นำไปใช้

มนุษย์และเครื่องจักรมีการใช้ซีเคร็ต (secrets) อยู่ตลอดในขั้นตอนของการพัฒนาและการใช้ซอฟต์แวร์ ไปป์ไลน์สำหรับการ build ก็ต้องการใช้ซีเคร็ตเพื่อติดต่อและเข้าถึงโครงสร้างพื้นฐานอย่างปลอดภัย เช่น container registry แอพพลิเคชั่นใช้ API key เป็นซีเคร็ตเพื่อเข้าถึงหน่วยความสามารถหนึ่งๆ ของธุรกิจ และการติดต่อระหว่างเซอร์วิสก็ถูกทำให้ปลอดภัยด้วยการใช้ certificate และ key เป็นซีเคร็ต คุณอาจมีวิธีตั้งค่าและเรียกใช้ซีเคร็ตเหล่านี้อยู่หลายวิธี เราได้เตือนนักพัฒนาให้ระวังการเก็บซีเคร็ตไว้ในระบบจัดการซอร์สโค้ด โดยแนะนำให้แยกการจัดการซีเคร็ตและซอร์สโค้ดออกจากกันและให้ใช้เครื่องมืออย่าง git-secrets และ Talisman ช่วยหลีกเลี่ยงการเก็บซีเคร็ตในซอร์สโค้ด เราได้ใช้เทคนิคการใช้ซีเคร็ตในรูปแบบเซอร์วิส (secrets as a service) เป็นเทคนิคหลักในการจัดเก็บและเรียกใช้ซีเคร็ตมาซักพักแล้ว ด้วยเทคนิคนี้คุณใช้เครื่องมืออย่าง Vault หรือ AWS Key Management Service (KMS) เพื่อควบคุมการอ่านหรือเขียนซีเคร็ตผ่าน HTTPS endpoint ได้โดยมีระดับสิทธิในการเข้าถึงให้เลือกใช้ได้โดยละเอียด เทคนิคซีเคร็ตในรูปแบบเซอร์วิสใช้ระบบจัดสรรอัตลักษณ์จาก(external identity provider) ภายนอกเช่น AWS IAM เพื่อระบุตัวตนผู้ใช้ที่ขอเข้าถึงซีเคร็ตใดๆ ทำให้ผู้ใช้ต้องพิสูจน์ยืนยันตัวตนกับเซอร์วิสก่อนเข้าถึงซีเคร็ต การที่กระบวนการนี้จะทำงานได้ มันสำคัญมากในการทำให้การเพิ่มอัตลักษณ์ของผู้ใช้ เซอร์วิส และแอพลิเคชั่นเป็นไปโดยอัตโนมัติ ซึ่งแพลตฟอร์มอย่าง SPIFFE ช่วยปรับปรุงการให้อัตลักษณ์แก่เซอร์วิสโดยอัตโนมัติให้ทำงานได้ดีขึ้น

Chaos Engineering

ทดลอง

ในปีที่ผ่านมาเราพบว่า Chaos Engineering เป็นเทคนิคที่พูดถึงและได้การยอมรับมากขึ้น ซึ่งหลักสำคัญของเทคนิคนี้เพื่อปรับปรุงและทำให้มั่นใจถึงความยืดหยุ่นของระบบแบบกระจาย (distributed system) สำหรับองค์กรขนาดใหญ่และเล็กเริ่มใช้ Chaos Engineering ในการทำ operational process เราได้ศึกษาการนำไปประยุกต์ใช้สำหรับเทคนิคนี้ในระดับที่ปลอดภัย เทคนิคนี้ไม่ได้เหมาะสำหรับทุกคนอย่างแน่นอนและการที่จะทำให้เทคนิคเกิดประสิทธิภาพและปลอดภัยนั้น เรายังต้องการการสนับสนุนจากองค์กร การยอมรับจากอุตสาหกรรมและความรู้ความชำนาญจะเพิ่มขึ้นจากบริการเชิงพาณิชย์อย่าง Gremlin และเครื่องมือสำหรับพัฒนาอย่างเช่น Spinnaker เป็นเครื่องมือที่ได้นำบางส่วนของเทคนิค Chaos Engineering ไปใช้

Container security scanning

ทดลอง

กระแสเทคโนโลยีคอนเทนเนอร์นำโดย Docker ได้เข้ามาปฏิวัติวงการ เพราะมันลดแรงเสียดทานในการย้ายแอพพลิเคชั่นระหว่าง environment ต่างๆ ซึ่งช่วยผลักดันการทำงานแบบ continuous delivery และ continuous deployment คลื่นความเปลี่ยนแปลงลูกใหม่นี้มาพร้อมกับความเสี่ยงต่อภัยคุกคามใหม่ๆ ซึ่งเทคนิค container security scanning เป็นการตอบสนองที่จำเป็นในด้านนี้ เครื่องมือในการทำ build pipeline จึงมีการตรวจสอบคอนเทนเนอร์โดยอัตโนมัติเพื่อมองหาจุดเสี่ยงที่รู้จัก ตั้งแต่แรกที่เราพูดถึงเทคนิคนี้จนมาถึงตอนนี้ เครื่องมือสำหรับเทคนิคนี้มีพัฒนาการขึ้นมาในระดับนึงแล้ว และเทคนิคนี้ก็ได้พิสูจน์แล้วว่ามีประโยชน์ต่อการทำงานให้กับลูกค้าของเรา

Continuous delivery for machine learning (CD4ML) models

ทดลอง

Continuous delivery for machine learning (CD4ML) models เป็นการนำการทำงานแบบ Continuous delivery มาประยุกต์ใช้ในการพัฒนาและเทรนโมเดลของ Machine learning เพื่อให้เกิดความพร้อมสำหรับการนำไปใช้งานบนโปรดักชั่นอยู่เสมอ เทคนิคนี้ช่วยแก้ปัญหาใหญ่ๆ สองปัญหาในการพัฒนาโมเดล Machine Learning คือ ระยะเวลาระหว่างการเทรนนิ่งโมเดล (training models) และการนำไปใช้งานบนโปรดักชั่น ซึ่งมักใช้ระยะเวลานาน และยังคงต้องอาศัยแรงงานคนในการแปลงโมเดลเพื่อนำไปใช้บนโปรดักชั่นอยู่ อีกปัญหาหนึ่ง คือ การใช้โมเดลบนโปรดักชั่นที่ถูกเทรนนิ่งมากับข้อมูลชุดเก่า

ไปป์ไลน์ Continuous delivery สำหรับ Machine Learning จะกระตุ้นให้เกิด (1) การเปลี่ยนแปลงเชิงโครงสร้างของโมเดล และ (2) การเปลี่ยนแปลงในการเทรนนิ่งและชุดข้อมูลสำหรับการเทรนนิ่ง การทำสิ่งนี้เราจำเป็นต้องระบุเวอร์ชั่นของ the data sets และซอร์สโค้ดของโมเดล ทั้งนี้ไปป์ไลน์มักรวมไปถึงขั้นตอนในการทดสอบโมเดลบนชุดข้อมูล การแปลงโมเดลแบบอัตโนมัติ (หากจำเป็น) ด้วยเครื่องมือต่างๆ เช่น H2O และการนำโมเดลไปใช้งานบนโปรดักชั่นเพื่อใช้เกิดประโยชน์กับผู้ใช้

Crypto shredding

ทดลอง

การดูแลรักษาระดับความปลอดภัยของข้อมูลที่มีความอ่อนไหวเป็นเรื่องยาก โดยเฉพาะเมื่อมันถูกทำสำเนาออกจากระบบหลักเพื่อจุดประสงค์ในการการสำรองข้อมูลหรือการกู้คืน crypto shredding เป็นกระบวนการในการทำให้ข้อมูลอ่อนไหวเหล่านี้ไม่สามารถอ่านค่าได้ รวมถึงการเขียนทับหรือทำลาย

encrption key ที่ใช้เข้าระดับข้อมูลเหล่านี้ เนื่องจากมันมีระบบที่ไม่สามารถหรือไม่ควรลบบันทึกย้อนหลังอยู่ เช่น แอพพลิเคชั่นสำหรับการออดิท (audit) หรือบล็อคเชน (blockchain) เทคนิคนี้จึงมีประโยชน์ในการป้องข้อมูลส่วนตัวหรือเป็นไปตามข้อกำหนดของ GDPR

Infrastructure configuration scanner

ทดลอง

ผ่านมาสักพักใหญ่ๆ แล้วที่เราแนะนำให้ทีมพัฒนารับผิดชอบทุกชั้นของแอพพลิเคชั่นร่วมถึงส่วนโครงสร้างพื้นฐานด้วย นั่นหมายความว่ามันเป็นหน้าที่ของทีมพัฒนาที่จะสร้างโครงสร้างพื้นฐานให้มั่นคงปลอดภัย และเป็นไปตามวัตถุประสงค์ หลายๆ องค์กรเมื่อเริ่มใช้ระบบบนคลาวด์ก็ทำการปิดกั้นสิทธิการจัดการโครงสร้างพื้นฐานให้เป็นอำนาจของส่วนกลางเท่านั้นเพื่อลดความเสี่ยง แต่นั่นก็ทำให้เกิดคอขวดและลดประสิทธิภาพในการทำงานลงอย่างเห็นได้ชัด อย่างไรก็ตามมันยังมีทางออกอื่นๆ อยู่ เช่น ให้ทีมพัฒนาจัดการการตั้งค่าเหล่านี้เอง และให้ใช้โปรแกรมตรวจสอบการตั้งค่าโครงสร้างพื้นฐานเพื่อรับประกันความปลอดภัยแทน ในตอนนี้มีโปรแกรมตรวจสอบแบบโอเพนซอร์สให้เลือกใช้ เช่น prowler สำหรับ AWS และ kube-bench สำหรับการติดตั้ง Kubernetes ส่วน continuous detection นั้นให้ลองสำรวจโปรแกรมบนแพลตฟอร์มเชิงพาณิชย์บนคลาวด์อย่าง AWS Config Rules ดู

Service mesh

ทดลอง

Service mesh เป็นแนวทางในการควบคุมระบบนิเวศ (ecosystem) ของไมโครเซอร์วิสที่ปลอดภัย รวดเร็วและเชื่อถือได้ มันเป็นก้าวสำคัญที่จะทำให้การใช้ไมโครเซอร์วิสในระดับใหญ่ง่ายขึ้น โดยทำให้ความสามารถที่มีการข้ามฟังก์ชั่น (cross-functional capabilities) ในระบบอย่าง การค้นพบ (discovery), ความปลอดภัย (security), การติดตาม (tracing), การตรวจเฝ้าระวัง (monitoring) และการรับมือความล้มเหลว (failure handling) ไม่ต้องอาศัยส่วนใช้ร่วมกันอย่างเกทเวย์ API และไม่ต้องติดตั้งไลบรารี่สำหรับการใช้ความสามารถเหล่านี้ลงในแต่ละเซอร์วิสโดยตรง การนำเทคนิคนี้ไปใช้โดยทั่วไปแล้วจะมี reverse-proxy process ขนาดเล็กหรือที่เรียกว่า sidecar ที่ถูกดิพลอยขึ้นไปพร้อมๆ กับเซอร์วิสในคอนเทนเนอร์ต่างๆ โดย sidecar จะมีหน้าที่ตรวจจับการเชื่อมต่อเข้าและออกจากเซอร์วิส และช่วยทำให้เกิดความสามารถที่มีการข้ามฟังก์ชั่นที่ได้กล่าวถึงข้างต้น แนวทางนี้ช่วยให้ทีมที่ดูแลเซอร์วิสแบบกระจายศูนย์ทำงานง่ายขึ้น เพราะสามารถสร้างและอัพเดทความสามารถดังกล่าวโดยไม่ต้องเขียนเป็นโค้ดฝังอยู่ในเซอร์วิส นี่ทำให้การใช้ polyglot programming ในระบบนิเวศของไมโครเซอร์วิสทำได้ง่ายขึ้นทีมของเราประสบความสำเร็จในการใช้เทคนิคนี้กับโปรเจ็คโอเพนซอร์สอย่าง Istio และ เรากำลังจับตาดูโปรเจ็คสำหรับ open service mesh อย่าง Linkerd อย่างใกล้ชิด

Ethical OS

ประเมิน

นักพัฒนาที่ ThoughtWorks ตระหนักถึงจริยธรรมในการทำงานอย่างจริงจัง เนื่องจากสังคมพึ่งพาเทคโนโลยีมากขึ้นเรื่อยๆ มันสำคัญที่ทีมพัฒนาซอฟต์แวร์ต้องพิจารณาเรื่องจริยธรรมด้วย ทั้งนี้ยังมีเครื่องมือหลายๆ ตัวที่ช่วยให้เราพิจารณาผลกระทบต่ออนาคตของซอฟต์แวร์ที่เราสร้างอยู่ Tarot Cards of Tech และ Ethical OS ก็เป็นเครื่องมือกลุ่มหนึ่งที่เราได้ยินการตอบรับที่ดี ซึ่ง Ethical OS เป็นทั้งเฟรมเวิร์กทางความคิดและชุดเครื่องมือที่ช่วยผลักดันบทสนทนาเกี่ยวกับจริยธรรมในการสร้างซอฟต์แวร์ เฟรมเวิร์กที่ว่าเกิดจากการร่วมมือระหว่าง Institute for the Future และ Tech and Society Solutions Lab โดยอิงจากรายการพื้นที่ที่มีความเสี่ยง เช่น การเสพย์ติด เศรษฐกิจโดพามีน และบทสมมติต่างๆ เพื่อช่วยผลักดันและนำทางการปรึกษาหารือร่วมกัน

Smart contracts

ประเมิน

ยิ่งเราสร้างความคุ้นเคยกับเทคโนโลยี distributed ledger (DTL) มากเท่าไหร่ เรายิ่งพบเหลี่ยมมุมที่ยังขัดเกลาไม่ดีนักของสมาร์ทคอนแทรคท์ (smart contract) การส่งและตรวจสอบคอนแทรคท์ที่ขัดแย้งไม่ได้และเปลี่ยนแปลงไม่ได้ฟังดูดีในทางทฤษฎี แต่เมื่อลองพัฒนาโดยใช้เทคนิคการพัฒนาซอฟต์แวร์แบบปัจจุบันในแบบต่างๆ ดูแล้วถึงได้เจอปัญหาต่างๆ ข้อมูลที่แก้ไขไม่ได้ (immutable data) ก็เป็นเรื่องหนึ่ง แต่ business logic ที่เปลี่ยนแปลงไม่ได้นั่นเป็นเรื่องที่ยุ่งยากขึ้นไปอีก ทำให้มันเป็นสิ่งสำคัญมากๆ ที่จะคำนึงถึง logic ที่เกิดขึ้นว่ามันควรอยู่ในสมาร์ทคอนแทรคท์หรือไม่ ลักษณะการทำงานที่แตกต่างกันในการนำเทคนิคนี้มาใช้ในแพลตฟอร์มต่างๆ เป็นอีกเรื่องหนึ่งที่ต้องคำนึงถึง ตัวอย่างเช่น ตัว contract นั้นจะต้องปรับปรุงค่าของตัวเองเสมอ แต่แพลตฟอร์มต่างๆ ก็รองรับการปรับปรุงนี้ได้มากน้อยต่างกันไป
เราแนะนำให้คุณหยุดแล้วคิด และคิดให้ดีก่อนจะตัดสินใจเลือกใช้ business logic ชุดใดชุดหนึ่งในสมาร์ทคอนแทรคท์ โดยให้คำนึงถึงข้อดีข้อเสียของแพลตฟอร์มต่างๆ ก่อนด้วย

Transfer learning for NLP

ประเมิน

Transfer learning นั้นค่อนข้างมีประสิทธิภาพในการทำ computer vision มันช่วยให้การฝึกฝนโมเดลเป็นไปอย่างรวดเร็วด้วยการนำโมดูลกลับมาใช้ซ้ำ เร็วๆ นี้คนที่อยู่ในวงการ machine learning ตื่นเต้นกันมากเมื่อได้เห็นเทคนิคเดียวกันนี้สามารถใช้กับ natural language processing (NLP) ได้ในงานตีพิมพ์ของ ULMFiT ที่มาพร้อมกับโมเดลโอเพนซอร์สที่ถูกฝึกฝนแล้ว และโค้ดตัวอย่าง เราคิดว่า transfer learning สำหรับ NLP จะช่วยแบ่งเบาภาระในการสร้างระบบ text classification ได้อย่างมีนัยสำคัญ

เทคนิค

เนื่องจากสังคมพึ่งพาเทคโนโลยีมากขึ้นเรื่อยๆ มันสำคัญที่ทีมพัฒนาซอฟต์แวร์ต้องพิจารณาเรื่องจริยธรรมด้วย Ethical OS เป็นทั้งเฟรมเวิร์กทางความคิดและชุดเครื่องมือที่ช่วยผลักดันบทสนทนาเกี่ยวกับจริยธรรมในการสร้างซอฟต์แวร์

(Ethical OS)

ข้อมูลที่แก้ไขไม่ได้ (immutable data) ก็เป็นเรื่องหนึ่ง แต่ business logic ที่เปลี่ยนแปลงไม่ได้นั่นเป็นเรื่องที่ยุ่งยากขึ้นไปอีก - คิดให้ดีก่อนจะตัดสินใจเลือกใช้ business logic ชุดใดชุดหนึ่งใส่ลงไปในสมาร์ทคอนแทรคท์

(Smart contracts)

เทคนิค

เราได้เห็นบางโปรเจ็คใช้ change data capture (CDC) สำหรับการประกาศอีเวนท์ความเปลี่ยนแปลงในระดับแถวของข้อมูล และการดึงอีเวนท์เหล่านี้ไปใช้โดยตรงโดยเซอร์วิสอื่นๆ สามารถทำให้การทำงานร่วมกันของเซอร์วิสในระบบนั้นเปราะบางได้

(Puncturing encapsulation with change data capture)

ถึงแม้เราสนับสนุนหลักการการปล่อยและสาธิตโปรแกรมอย่างสม่ำเสมออย่างแท้จริง แต่เราเจอกับข้อเสียที่รุนแรงในการใช้เทคนิคนี้ ในระยะกลางและระยะยาว มันทำให้เกิดความผูกติดกันระหว่างรอบการส่งมอบซอฟท์แวร์ทำให้การเปลี่ยนแปลงลำดับการส่งมอบซอฟต์แวร์ยากขึ้น และมันยังสามารถลดคุณภาพของซอฟต์แวร์ เพราะทีมพัฒนาต้องเร่งทำงานให้เสร็จในแต่ละรอบด้วย

(Release train)

Wardley mapping

ประเมิน

ปกติทางเราจะมัดระวังที่จะเขียนเกี่ยวกับเทคนิคการทำแผนภาพ (diagrammatic techniques) ลงในเรดาห์ ถึงอย่างนั้นเราเลือกที่จะพูดถึง แผนภาพวาร์ดลีย์ (Wardley mapping) เพราะมันเป็นเทคนิคที่น่าสนใจเมื่อต้องการนำเสนอเกี่ยวกับการพัฒนาทรัพย์สินทางซอฟต์แวร์ขององค์กร ประโยชน์ขั้นพื้นฐานของมันคือการใช้ในการสร้างแผนภาพเพื่อแสดงถึง ห่วงโซ่แห่งคุณค่า (value chain) เริ่มจากความต้องการของลูกค้าและค่อยๆ ต่อยอดออกเป็นแผนภาพที่แสดงถึงศักยภาพในแต่ละด้านและระบบต่างๆ ที่ใช้ในการตอบสนองความต้องการของลูกค้า รวมถึงแผนผังพัฒนาการของศักยภาพและระบบเหล่านี้ด้วย
คุณค่าของเทคนิคนี้อยู่ที่การทำให้เกิดการมีส่วนร่วมและการพูดคุยของคนในองค์กรมากกว่าตัวแผนภาพในตัวมันเอง เราแนะนำให้คนที่เหมาะสมมีส่วนร่วมในการสร้างแผนภาพวาร์ดลีย์ มองมันเป็นสิ่งที่เปลี่ยนแปลงได้ตลอดเวลามากกว่าสิ่งที่เสร็จสิ้นสมบูรณ์

Productionizing Jupyter Notebooks

เฝ้าระวัง

Jupyter Notebooks ได้รับความนิยมในกลุ่ม data scientist ที่ใช้มันสำหรับทำการวิเคราะห์แบบสำรวจ การพัฒนาในขั้นต้น และการแบ่งปันความรู้ และทำให้เกิดกระแสการนำ jupyter notebook มาใช้ในโปรดักชั่น ด้วยการใช้เครื่องมือสนับสนุนให้มันทำงานในระดับใหญ่ได้ ถึงแม้เราไม่อยากจะกีดกันไม่ให้คนใช้เครื่องมือที่ชอบ แต่เราไม่แนะนำให้ใช้ Jupyter Notebooks ในการเขียนโค้ดในระดับโปรดักชั่นที่ต้องสามารถขยายตัวและบำรุงรักษาง่าย เพราะมันขาดระบบ version control ที่มีประสิทธิภาพ ขาดระบบจัดการข้อผิดพลาด นำส่วนเพิ่มเติมหรือขยายมาต่อเติมยาก และคุณสมบัติอื่นๆ ที่จำเป็นสำหรับโค้ดระดับโปรดักชั่นที่ขยายตัวได้ เราแนะนำให้นักพัฒนาและ data scientist ให้ร่วมกันหาทางออกที่ช่วยให้ data scientist สามารถสร้างโมเดล machine learning ได้ด้วยการทำงานแบบ continuous delivery พร้อมกับเฟรมเวิร์กการเขียนโปรแกรมที่เหมาะสม เราเตือนไม่ให้ใช้ Jupyter Notebook ในระดับโปรดักชั่นเพื่อให้ก้าวข้ามอุปสรรคในการทำไปป์ไลน์ continuous delivery สำหรับ machine learning ให้มีประสิทธิภาพ หรือหลีกเลี่ยงการทำการทดสอบแบบอัตโนมัติที่ไม่เพียงพอ

Puncturing encapsulation with change data capture

เฝ้าระวัง

Change data capture (CDC) เป็นเทคนิคที่ทรงพลังในการตรวจจับความเปลี่ยนแปลงในฐานข้อมูล เพื่อทำอะไรบางอย่างกับความเปลี่ยนแปลงนั้น หนึ่งในวิธีที่ใช้การคือ การใช้ transaction log ในการบันทึกและระบุการเปลี่ยนแปลง และปล่อยการเปลี่ยนแปลงนั้นเข้าสู่ event bus ซึ่งถูกใช้โดยเซอร์วิสอื่นๆ วิธีนี้ได้ผลดีในกรณีที่เราต้องการแตกเซอร์วิสขนาดใหญ่ออกเป็นไมโครเซอร์วิสขนาดเล็ก แต่เมื่อเราใช้ในการทำ first-class integration ระหว่างไมโครเซอร์วิส จะก่อให้เกิดการรั่วไหลระหว่างชั้นของข้อมูล และ event contract เราเคยพูดถึงเรื่อง domain scoped events และเทคนิคอื่นๆ ที่ย้ำถึงความสำคัญของการออกแบบโดเมนของเราโดยคำนึงถึง event ที่เกิดขึ้น เราได้เห็นบางโปรเจ็คใช้ CDC ในการตรวจจับการเปลี่ยนแปลงของข้อมูลในระดับแถว และส่งเป็นอีเวนท์ให้เซอร์วิสอื่นๆ ใช้โดยตรง ซึ่งเป็นการทำลาย encapsulation ด้วย change data capture ซึ่งสามารถทำให้ integration ระหว่างไมโครเซอร์วิสต่างๆ เปราะบางลงอย่างรวดเร็ว

Release train

เฝ้าระวัง

เราได้เห็นหลายๆ องค์กรที่พัฒนาความถี่ในการปล่อยเวอร์ชั่นของโปรแกรม ด้วยการนำเทคนิค release train มาใช้ release train เป็นเทคนิคในการจัดการการส่งมอบซอฟต์แวร์ระหว่างทีมพัฒนาๆ หลายๆ ทีม หรือองค์ประกอบของซอฟต์แวร์ที่มี runtime ขึ้นต่อกัน การส่งมอบซอฟต์แวร์ทุกอย่างเกิดขึ้นในเวลาที่กำหนดขึ้น ไม่ว่าฟีเจอร์ที่คาดหวังไว้จะพร้อมทุกตัวหรือไม่ก็ตาม (รถไฟจะไม่รอคุณ ถ้าคุณพลาดรถไฟคันนี้คุณต้องไปขึ้นคันถัดไป) ถึงแม้เราสนับสนุนหลักการการส่งมอบและสาธิตซอฟต์แวร์อย่างสม่ำเสมออย่างแท้จริง แต่เราเจอกับข้อเสียที่รุนแรงในการใช้เทคนิคนี้ในระยะกลางและระยะยาว มันทำให้เกิดความผูกติดกันระหว่างรอบทำให้มีการเปลี่ยนแปลงลำดับการส่งมอบซอฟต์แวร์ยากขึ้น และมันยังทำให้คุณภาพของซอฟต์แวร์ลดลงไปด้วย เพราะทีมพัฒนาเร่งงานให้เสร็จให้ทันในแต่ละรอบด้วย เราสนับสนุนการให้ความสนใจกับการสร้างโครงสร้างพื้นฐานหรือปรับโครงสร้างการจัดการที่สนับสนุนการส่งมอบซอฟต์แวร์ในแต่ละรอบมากกว่า ถึงแม้ release train มีประโยชน์ในการบังคับให้ทีมที่ช้าให้ทำงานเร็วขึ้น แต่มันก็จำกัดความเร็วของทีมที่ทำงานได้รวดเร็วกว่าเหมือนกัน เราเชื่อว่านี่เป็นเทคนิคนี้ไม่ควรนำมาใช้ หรือไม่เช่นนั้นก็ควรใช้อย่างระมัดระวัง

Templating in YAML

เฝ้าระวัง

เมื่อโครงสร้างพื้นฐาน (infrastructure) มีความซับซ้อนมากขึ้น ไฟล์ตั้งค่าของมันก็มีความซับซ้อนเพิ่มตามขึ้นไปด้วย เครื่องมืออย่าง AWS CloudFormation, Kubernetes และ Helm มีไฟล์การตั้งค่าในรูปแบบ JSON หรือ YAML เพราะมันง่ายที่จะเขียนหรือประมวลผล แต่ใช้ไปไม่นานทีมก็มักจะพบปัญหาว่าการตั้งค่าบางส่วนมีความคล้ายคลึงกันแต่ไม่เหมือนกันซะทีเดียว เช่น การตั้งค่าของเซอร์วิสที่จะถูกดิพลอยในพื้นที่ต่างๆ กัน แต่มีการตั้งค่าที่คล้ายคลึงกัน ซึ่งในกรณีแบบนี้ templating in YAML (YAML หรือ JSON) จะสร้างความปวดหัวให้กับผู้ใช้งานมากๆ ปัญหาคือรูปแบบการเขียน JSON และ YAML ต้องถูกดัดแปลงแบบประหลาดๆ เพื่อให้มันใช้ฟีเจอร์อย่างการสร้างลูป (looping) และการสร้างเงื่อนไข (conditioning) ได้ เราแนะนำให้ใช้งานเครื่องมือเหล่านี้ผ่าน API ของมันบนภาษาโปรแกรมแทน หรือหากไม่มีก็ให้ใช้ตัวช่วยในการเขียน JSON หรือ YAML ไม่ว่าจะในภาษาอเนกประสงค์อย่าง Python หรือภาษาเฉพาะกิจอย่าง Jsonnet

แพลตฟอร์ม

Contentful

นำไปใช้

Headless content management systems (headless CMS) กำลังกลายเป็นสิ่งที่ถูกใช้ทั่วไปในแพลตฟอร์มดิจิทัล Contentful คือ headless CMS สมัยใหม่ที่ทีมงานของเราใช้ในกระบวนการพัฒนาเว็บไซต์อย่างประสบความสำเร็จ เราชอบที่ Contentful ให้ความสำคัญกับการเปิดให้ใช้ผ่าน API ทำให้เราใช้งาน CMS ด้วยการเขียนเป็นโค้ดได้ Contentful สนับสนุนให้ผู้ใช้มองการออกแบบโมเดลของเนื้อหาบนเว็บไซต์เป็นชุดคำสั่ง และให้ผู้ใช้พัฒนาโมเดลของเนื้อหาเพิ่มเติมด้วยการเขียนชุดคำสั่งได้
ลักษณะการจัดการเนื้อหาแบบนี้ทำให้เราสามารถมองเนื้อหาเป็นแผนผังของข้อมูลแบบหนึ่ง (data store schema) และทำให้เรานำหลัก การออกแบบฐานข้อมูลแบบวิวัฒนาการ (evolutionary database design) มาใช้กับการพัฒนาเว็บไซต์ด้วย CMS ได้
นอกจากนี้ความสม่ำเสมอในการปล่อยความสามารถใหม่ๆ ออกมา เช่น พื้นที่สำหรับทดลอง (sandbox environment) ทำให้ทีมงามของเราประทับใจขึ้นไปอีก Contentful จึงเป็นตัวเลือกอันดับแรกของเราสำหรับงานประเภทนี้

AWS Fargate

ทดลอง

AWS Fargate คือบริการ docker บน AWS ที่มีให้ใช้ในหลายๆพื้นที่แล้ว หากคุณรู้สึกว่าเซิร์ฟเวอร์ EC2 หรือคลัสเตอร์ Kubernetes นั้นใหญ่ไปสำหรับงานและดูแลจัดการยากหรือฟังก์ชั่นบน AWS Lambda ไม่ทรงพลังพอ การใช้คอนเทนเนอร์ของ docker บน Fargate ก็เป็นทางออกที่ดีทางหนึ่ง ทีมงานของเรามีประสบการณ์ใช้งานที่ดีกับ Fargate อย่างไรก็ตามความสะดวกสบายของเซอร์วิสนี้มาพร้อมกับค่าใช้จ่ายราคาแพงเช่นกัน

EVM beyond Ethereum

ทดลอง

Ethereum Virtual Machine (EVM) ถูกออกแบบพัฒนาไว้สำหรับเครือข่ายของอีเธอเรียม (Ethereum) แต่เนื่องจากไม่มีใครอยากจะมานั่งพัฒนาบล็อคก์เชน (blockchain) ใหม่ตั้งแต่ต้น จึงมีทีมพัฒนา blockchain หลายๆ ทีมเลือกที่ใช้ EVM นอกเหนืออีเธอเรียม (EVM beyond Ethereum) โดยพัฒนาดัดแปลงต่อยอดมันอีกทีหนึ่ง บางทีมฟอร์ก (fork) อีเธอเรียมมาเลย (เช่น Quorum) บางทีมก็นำข้อมูลจำเพาะของ EVM มาใช้ (เช่น Burrow, Pantheon) ประโยชน์ของมันไม่ใช่แค่การนำมาใช้ใหม่ได้เท่านั้น แต่ยังสามารถใช้ประโยชน์จากระบบนิเวศ (ecosystem) ของ EVM และจากชุมชนนักพัฒนาของมันอีกด้วย สำหรับนักพัฒนาหลายๆ คน “แนวคิดสมาร์ทคอนแทรคท์” (smart contract) แทบจะกลายเป็น “สมาร์ทคอนแทรคท์ที่เขียนด้วย Solidity” ไปแล้ว ในตอนนี้ถึงแม้ตัวอีเธอเรียมเองจะมีข้อจำกัดอยู่บ้าง แต่เทคโนโลยีรอบๆ ระบบนิเวศของ EVM กำลังเจริญรุ่งเรือง

InfluxDB

ทดลอง

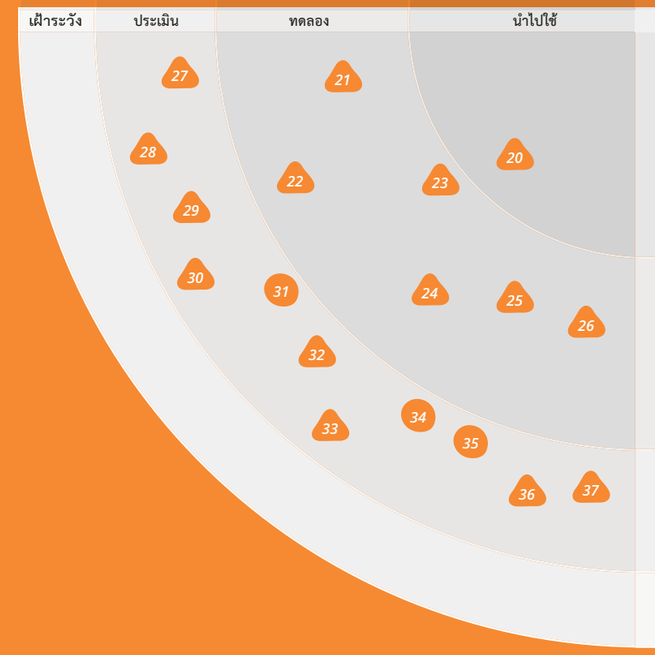
ฐานข้อมูลแบบ Time series (TSDBs) นั้นอยู่กับเรามาสักพักแล้ว แต่ในปัจจุบันกำลังเป็นที่ความนิยมมากขึ้น เนื่องจากรูปแบบการใช้งานที่เหมาะกับการใช้ time series model นั้นมีเพิ่มมากขึ้น รูปแบบการใช้งานที่เป็นที่นิยมในการนำ InfluxDB ไปใช้คือ การนำไปทำการตรวจเฝ้าระวัง (monitoring) TICK Stack ก็เป็นหนึ่งในตัวอย่างของการทำระบบการตรวจเฝ้าระวัง โดยมี InfluxDB เป็นแกนหลัก ไม่นานนี้ Influx 2.0 alpha เพิ่งเปิดตัว Flux — ภาษาสคริปต์สำหรับการ query และประมวลผลข้อมูล time series ถึงแม้ว่า Flux ยัง

นำไปใช้
20. Contentful

ทดลอง
21. AWS Fargate
22. EVM beyond Ethereum
23. InfluxDB
24. Istio
25. Kafka Streams
26. Nomad

ประเมิน
27. CloudEvents
28. Cloudflare Workers
29. Deno
30. Hot Chocolate
31. Knative
32. MinIO
33. Prophet
34. Quorum
35. SPIFFE
36. Tendermint
37. TimescaleDB

เฝ้าระวัง



แพลตฟอร์ม

ฐานข้อมูลแบบ Time series(TSDBs) นั้นอยู่กับเรามาสักพักแล้ว แต่ในปัจจุบันกำลังเป็นที่ความนิยมมากขึ้น เนื่องจากรูปแบบการใช้งานที่เหมาะกับการใช้ time series model นั้นมีเพิ่มมากขึ้น InfluxDB ยังคงเป็นทางเลือกที่ดีสำหรับงานลักษณะนี้

(InfluxDB)

อีเวนท์เป็นระบบสำหรับกระตุ้นแบบให้แอพพลิเคชั่นแบบ FaaS ทำงาน ที่ทุกผู้ให้บริการคลาวด์มีให้ แต่มี specification การใช้งานต่างกัน ทำให้แอพพลิเคชั่นไม่สามารถทำงานข้ามคลาวด์ของแต่ละผู้ให้บริการได้ CloudEvents เป็นมาตรฐานที่กำลังเติบโตเพื่อจัดการกับปัญหานี้

(CloudEvents)

ออกมาได้ไม่นาน และชุมชนนักพัฒนาส่วนใหญ่ยังไม่เห็นภาพของการนำ flux ไปใช้แบบชัดเจน แต่มันถูกตั้งความคาดหวังไว้ว่าจะสามารถมีประสิทธิภาพสูงกว่าและใช้งานง่ายกว่า InfluxQL รวมถึงมันจะสามารถผลักภาระการประมวลผลที่เกี่ยวข้องกับการวิเคราะห์ time series ออกไปให้ฐานข้อมูลได้อีกด้วยอย่างไรก็ตาม การทำคลัสเตอร์ของ InfluxDB สามารถทำได้เฉพาะในเวอร์ชั่นสำหรับองค์กรเท่านั้น และก็เป็นสาเหตุที่ทำให้เราไม่ได้ใช้มันอย่างแพร่หลายในโปรเจกต์ของเรา

Istio

ทดลอง

Istio กำลังกลายเป็นส่วนจำเป็นของการดำเนินการบนระบบนิเวศของไมโครเซอร์วิส (microservices ecosystem) มันมีความสามารถในการจัดการส่วนของระบบที่ส่งผลกับส่วนอื่นหลายๆ ส่วน เช่น การค้นพบเซอร์วิส (service discovery), ความปลอดภัยระหว่างเซอร์วิสและออริจินกับเซอร์วิส (service-to-service and origin-to-service security), การสังเกตการณ์ได้ (observability - รวมถึงการเข้าถึงข้อมูลจากนอกระบบและการติดตามข้อมูลระหว่างเซอร์วิส) และการออกเวอร์ชั่น (rolling release) และ ความทนทาน (resiliency) ความสามารถเหล่านี้ทำให้เราเริ่มสร้างระบบไมโครเซอร์วิสได้อย่างรวดเร็ว และเป็นทางเลือกหลักในการนำเทคนิค service mesh มาใช้ Istio ปล่อยเวอร์ชั่นใหม่ออกมาทุกเดือน และมีพัฒนาขึ้นเรื่อยๆอย่างต่อเนื่อง เราใช้ Istio ทำบูตแสตรป (bootstrap) โปรเจ็คของเรา เริ่มจากการสังเกตการณ์ได้ (การติดตามข้อมูลและการเข้าถึงข้อมูล) และความปลอดภัยระหว่างเซอร์วิส เรากำลังเฝ้าดูการพัฒนาระบบพิสูจน์ตัวตนระหว่างเซอร์วิส (service-to-service authentication) ทั้งภายในและภายนอก service mesh เราคาดหวังว่ามันจะสร้างแนวทางทำงานที่ดีในการตั้งค่า (configuration file) เพื่อสร้างสมดุลย์ระหว่างการให้อิสระในการพัฒนากับนักพัฒนาเซอร์วิสและความสามารถในการควบคุมให้แก่ service mesh operator

Kafka Streams

ทดลอง

Kafka Streams เป็นไลบรารีขนาดเล็กที่ใช้ในการสร้างสตรีมมิ่งแอพพลิเคชั่น มันรองรับสตรีมมิ่ง API พื้นฐานต่างๆ เช่น join, filter, map และ aggregate รวมถึง local storage สำหรับการใช้งานทั่วไปเช่น windowing และ sessions
เทียบกับแพลตฟอร์มที่ใช้ในการประมวลผลสตรีมอื่นๆ เช่น Apache Spark และ Alpakka Kafka แล้ว Kafka Streams นั้นเหมาะสมกับงานเล็กๆ ที่ไม่กระจายตัวและไม่ต้องการ การประมวลผลแบบขนานมากกว่า เราสามารถใช้งานมันได้ทันทีโดยไม่จำเป็นต้องเพิ่มชิ้นส่วนต่างๆ เช่นระบบจัดการคลัสเตอร์เข้าไปในโครงสร้างพื้นฐาน โดยพื้นฐานแล้ว Kafka Streams นั้นเป็นตัวเลือกที่ดีเมื่อใช้งานร่วมกับระบบนิเวศของ Kafka โดยเฉพาะอย่างยิ่งเมื่อเราต้องการประมวลผลข้อมูลที่ต้องการความถูกต้องในเรื่องลำดับและไม่ซ้ำ ตัวอย่างการใช้งาน Kafka Streams อย่างหนึ่งก็คือการสร้างแพลตฟอร์มสำหรับ change data capture (CDC)

Nomad

ทดลอง

HashiCorp ยังคงปล่อยซอฟต์แวร์ที่น่าสนใจออกมาเรื่อยๆ เราได้แนะนำ HashiCorp Vault ไปเมื่อเดือนมีนาคม 2017 และบรรดาเครื่องมือที่เกี่ยวข้องกับ Terraform ก็มีอยู่เต็มไปหมดในเรดาห์ฉบับนี้ เราย้าย Nomad มาที่ ทดลอง เพราะเรามีประสบการณ์ที่ดีจากการใช้งาน ในขณะที่ Kubernetes ได้รับความนิยมเพิ่มขึ้นอย่างต่อเนื่อง เรากลับชอบการใช้งานทั่วไปของ Nomad ที่ไม่ใช่แค่รันงานในคอนเทนเนอร์แต่สามารถใช้ในการกำหนดรูทีนประจำในการรันแอพลิเคชั่น นอกจากนี้ยังสนับสนุนจาวาและ Golang เช่นเดียวกันกับ batch และ distributed cron job เราชอบที่ Nomad โฟกัสไปที่การใช้งานคลาวแบบหลายแห่ง หรือผสมกัน มากกว่าการยึดติดกับผู้ให้บริการรายเดียวและข้อเท็จจริงที่ว่ามันจัดการได้ดีมาก

CloudEvents

ประเมิน

โค้ดของแอพพลิเคชั่นบน serverless functions มักยึดติดกับแพล็ตฟอร์มคลาวด์ที่มันตั้งอยู่อย่างเหนียวแน่น อย่างเช่น อีเวนท์ (event) ที่เป็นระบบสำหรับกระตุ้นให้แอพพลิเคชั่นทำงาน ถึงแม้มันจะเป็นคุณสมบัติที่ทุกผู้ให้บริการคลาวด์มีให้ แต่มี specification การใช้งานต่างกัน ทำให้แอพพลิเคชั่นไม่สามารถทำงานข้ามคลาวด์ของแต่ละผู้ให้บริการได้ specification ของ CloudEvents นั้นเติบโตจากมาตรฐานที่เป็นส่วนหนึ่งของ CNCF Sandbox มาตรฐานดังกล่าวกำลังอยู่ในขั้นพัฒนา แต่ก็รองรับในหลายๆ ภาษาแล้ว รวมถึง Microsoft ก็ประกาศให้การสนับสนุนอย่างเต็มตัวใน Azure และพวกเราหวังว่าผู้ให้บริการคลาวด์เจ้าอื่นๆ จะค่อยๆ ตามมา

Cloudflare Workers

ประเมิน

Server-side ที่ทันสมัยหรือโค้ด serverless ที่ถูกรันมักจะอยู่ในคอนเทนเนอร์หรือ Virtual Machine(VM) Cloudflare Workers นั้นได้ใช้วิธีการที่แตกต่างในการโฮส serverless computing ของตัวเอง ตัว Cloudflare Worker เองใช้ V8 Isolates ซึ่งเป็นโอเพนซอร์ส JavaScript engine ที่พัฒนาขึ้นสำหรับ Chrome การที่จะรัน functions as a service (FaaS) บน extensive CDN network นั้นเราสามารถเขียนโค้ดด้วยภาษา JavaScript หรือภาษาอะไรก็ตามที่คอมไพล์เป็น WebAssembly และข้อมูลสามารถถูกเข้าถึงได้จากแคชของ Cloudflare หรือ key-value store ประโยชน์ที่นักพัฒนาซอฟแวร์จะได้รับคือประสิทธิภาพในการเข้าถึง cold-starts ซึ่งการที่เราอยู่ที่ edge network ซึ่งใกล้เคียงกับผู้ใช้งานจริง จะใช้เวลาเพียงแค่ 5 มิลลิวินาที สำหรับผู้ให้บริการจะได้ประโยชน์ในความสามารถในการรันโปรเซสต่อเครื่องได้มากขึ้นเพราะการลดหน่วยความจำที่เกินความจำเป็นและ เพิ่มความเร็วโดยการลด process context switching นั้นเป็นวิธีการที่น่าสนใจสำหรับการติดตามผลและการประเมิน

Deno

ประเมิน

พวกเรามักรู้สึกขัดแย้งในตัวเองเมื่อเราตัดสินใจสร้างแอป JavaScript สำหรับฝั่ง server โดยเฉพาะเมื่อเหตุผลเดียวของการตัดสินใจนี้คือ เพื่อหลีกเลี่ยง polyglot programming อย่างไรก็ดีหากคุณตัดสินใจใช้ JavaScript หรือ TypeScript บน server แล้ว Deno ก็เป็นสิ่งที่น่าศึกษา Deno ถูกเขียนขึ้นโดย Ryan Dahl ที่เป็นผู้สร้าง Node.js โดยมีเป้าหมายเพื่อหลีกเลี่ยงความผิดพลาดที่เคยทำไว้ใน Node.js ตัว Deno มีระบบ sandbox ที่เข้มงวด มีระบบจัดการ dependency และแพ็คเกจมาพร้อมด้วยในตัว นอกจากนี้มันยังสนับสนุน TypeScript ตั้งแต่ต้นอีกด้วย ตัว Deno ถูกสร้างขึ้นด้วย Rust และ V8

Hot Chocolate

ประเมิน

ในระบบนิเวศ (ecosystem) และชุมชนของ GraphQL ที่กำลังเติบโตขึ้นเรื่อยๆ Hot Chocolate เป็นเซิร์ฟเวอร์ GraphQL สำหรับ .NET (core และ classic) มันอนุญาตให้คุณสร้างและโฮสท์ schema และเรียกคิวรี่ (query) จาก schema เหล่านั้น ทีมพัฒนา Hot Chocolate เพิ่งเพิ่มฟีเจอร์ schema stitching เข้ามา ซึ่งอนุญาตให้ใช้ entry point เพียงตัวเดียวก็สามารถเข้าถึง schema หลายๆ ตัวที่รวบรวมจากหลายๆแหล่งได้ ถึงแม้ว่ามันเปิดโอกาสให้ใช้อย่างผิดๆ ได้หลายๆ วิธี แต่มันก็คุ้มค่าที่จะลองประเมินดูว่ามันเหมาะที่จะนำมาใช้หรือไม่

Knative

ประเมิน

โครงสร้างระบบคอมพิวเตอร์แบบ serverless ทำให้ FaaS เป็นที่นิยมในชุมชนนักพัฒนา มันช่วยให้นักพัฒนาใช้เวลากับการแก้ปัญหาที่เป็น core business ด้วยฟังก์ชั่นที่ดิพลอยตัวเองขึ้นมาตอบสนองเหตุการณ์หนึ่งๆ ดำเนินกระบวนการทางธุรกิจ กระตุ้นเหตุการณ์ในขั้นตอนต่อไป และปิดทำลายตัวเองลงไป (scale down to zero) ในตอนแรกแพลตฟอร์ม serverless ที่มีกรรมสิทธิ์อย่างเช่น AWS Lambda และ Microsoft Azure Functions ทำให้เกิดกระบวนทัศน์การเขียนโปรแกรม (programming paradigm) นี้ขึ้นมา ตอนนี้จึงมี Knative ที่เป็นแพล็ตฟอร์มโอเพนซอร์สที่อาศัย Kubernetes ในการทำงานแบบ FaaS Knative มีจุดเด่นอยู่คือ มันไม่เข้าข้างผู้ให้บริการเจ้าใดเจ้าหนึ่งเป็นพิเศษ การทำงานแบบ serverless ของ Knative เป็นไปตาม whitepaper ของ CNCF Serverless Working Group มันรับรองความสามารถในการทำงานข้ามเซอร์วิสด้วยการสร้าง eventing interface ให้เป็นไปตามข้อกำหนดของ CNCF CloudEvents และที่สำคัญที่สุดคือมันสามารถแก้ปัญหาการออกแบบสถาปัตยกรรมของระบบที่รองรับการทำงานร่วมกันระหว่าง FaaS กับคอนเทนเนอร์อายุยืน (ทำงานเป็นเวลานาน) ด้วยการรวม Istio และ Kubernetes เข้าด้วยกัน ตัวอย่างเช่น นักพัฒนาสามารถใช้ประโยชน์จากเทคนิคการ roll-out ของ Istio ที่มีการจำแนก revision ของแต่ละฟังก์ชั่น ซึ่งทำให้เราสามารถตรวจสอบการทำงานของทั้งคอนเทนเนอร์อายุยืน และโปรแกรม FaaS ในคลัสเตอร์ Kubernetes ตัวเดียวกันได้ เราคาดว่า ต่อไป eventing interface ของ Knative จะยังคงช่วยให้เกิดบูรณาการแบบใหม่ระหว่างเหตุการณ์ต้นทางและปลายทาง (source and destination event)

MinIO

ประเมิน

Object storage เป็นทางเลือกที่ได้รับความนิยมสูงสำหรับการจัดเก็บข้อมูลบนคลาวด์ ทั้งข้อมูลแบบไร้โครงสร้าง (unstructured data) หรือแม้แต่ข้อมูลแบบมีโครงสร้าง (structured data) ในบางกรณี เราไม่แนะนำให้ใช้แหล่งเก็บข้อมูลบนคลาวด์แบบมาตรฐาน แต่ถ้าคุณต้องการลดความเสี่ยงที่จะถูกผูกขาดโดยผู้ให้บริการคลาวด์เจ้าใดเจ้าหนึ่ง เราพบว่า MinIO ค่อนข้างมีประโยชน์ MinIO ครอบเซอร์วิส object storage ของ AWS, Azure, และ Google Cloud Platform ไว้ และสร้าง API ที่เป็นมาตรฐานระหว่างผู้ให้บริการขึ้นมา เราเคยใช้มันอย่างประสบความสำเร็จในการพัฒนาโปรแกรมที่มีการส่งข้อมูลระหว่างดาต้าเซ็นเตอร์และผู้ให้บริการคลาวด์หลายๆ เจ้า

Prophet

ประเมิน

แม้แต่ในยุคของ deep learning โมเดลทางสถิติก็ยังมีบทบาทสำคัญในการสนับสนุนการตัดสินใจทางธุรกิจ โมเดลอนุกรมเวลายังคงถูกใช้อย่างกว้างขวางในการทำนายปริมาณสินค้าคงคลัง, อุปสงค์, ปริมาณลูกค้า และอื่นๆ อีกมากมาย เป็นเวลานานแล้วที่การสร้างโมเดลที่ทันสมัยและยืดหยุ่นเป็นหน้าที่ของผู้เชี่ยวชาญทางสถิติหรือตัวแทนจำหน่ายซอฟต์แวร์เจ้าใหญ่ ตอนนี้จึงมี Prophet เป็นทางเลือกโอเพนซอร์สนอกเหนือจากโปรแกรมพาณิชย์สำหรับพยากรณ์ทางการค้าทั่วไป โดยที่คุณสามารถใช้ Prophet ด้วยภาษา R หรือ Python ก็ได้ Facebook อ้างว่าได้ใช้ Prophet ภายในบริษัทสำหรับพยากรณ์ทางธุรกิจในสเกลใหญ่ และได้เปิดให้ทุกคนใช้เป็นแพ็คเกจโอเพนซอร์ส เราชอบที่ Prophet ช่วยกำจัดส่วนที่น่าเบื่อหน่ายในส่วนของการสร้างโมเดล การบำรุงรักษา และการดัดแปลงข้อมูลออกไป เพื่อให้นักวิเคราะห์และผู้เชี่ยวชาญเฉพาะด้านสามารถไปทำงานด้านอื่นที่สำคัญกว่าได้

แพลตฟอร์ม

หากคุณตัดสินใจใช้ JavaScript หรือ TypeScript บน server แล้ว Deno ก็เป็นทางเลือกที่น่าสนใจ มันมีระบบ sandbox ที่เข้มงวด มีระบบจัดการ dependency และแพ็คเกจมาพร้อมด้วยในตัว นอกจากนี้มันยังสนับสนุน TypeScript ตั้งแต่ต้นอีกด้วย

(Deno)

โครงสร้างระบบคอมพิวเตอร์แบบ serverless ทำให้ FaaS เป็นที่นิยมในชุมชนนักพัฒนา Knative เป็นแพล็ตฟอร์มโอเพนซอร์สที่ไม่เข้าข้างผู้ให้บริการเจ้าใดเจ้าหนึ่ง ที่อาศัย Kubernetes ในการทำงานแบบ FaaS

(Knative)

แพลตฟอร์ม

Tendermint เป็นเอนจิ้นสำหรับผลิตซ้ำ BFT state machine ที่ช่วยให้คุณสร้างระบบ blockchain ของคุณเอง

(Tendermint)

Quorum

ประเมิน

Quorum เป็น “อีเธอเรียมสำหรับองค์กร” ที่ความตั้งใจคือการเพิ่มการจัดการสิทธิการเข้าถึงเครือข่าย สร้างความเป็นส่วนตัวในการทำธุรกรรม และเพิ่มสมรรถนะของระบบ ทีมของเราทีมหนึ่งใช้งาน Quorum อย่างละเอียดแต่มีประสบการณ์ที่ไม่ค่อยดีนัก ปัญหาบางอย่างเกิดจากความซับซ้อนของโปรแกรม smart contract และบางอย่างก็เกิดจากตัว Quorum เองตัวอย่างเช่น มันทำงานร่วมกับ load balancer ได้ไม่ดี และสนับสนุนการใช้งานฐานข้อมูลเพียงบางส่วนเท่านั้น ซึ่งทำให้เป็นภาระใหญ่ของการดิพลอย (deployment) เราเจอปัญหาด้านความเสถียรและการใช้งานร่วมกับโปรแกรมอื่น โดยเฉพาะในการทำธุรกรรมส่วนตัว Quorum พึ่งได้รับความสนใจไม่นานนี้เพราะ JPM Coin อย่างไรก็ตามหากมองในแง่เทคโนโลยีแล้ว เราแนะนำให้ระมัดระวังการนำ Quorum มาใช้ และจับตาดูว่ามันจะพัฒนาต่อไปในทางไหน

SPIFFE

ประเมิน

มาตรฐานทางอัตลักษณ์ของเซอร์วิส, SPIFFE, เป็นก้าวที่สำคัญในการแก้ปัญหาแบบเบ็ดเสร็จในเรื่องการเข้ารหัสแบบ end-to-end และการยืนยันตัวตนระหว่างเซอร์วิส มาตรฐาน SPIFFE อาศัย OSS SPIFFE Runtime Environment (SPIRE) ซึ่งช่วยสร้างอัตลักษณ์ที่ยืนยันตัวตนด้วยการเข้าและถอดรหัสได้ให้กับเซอร์วิสต่างๆ ตัว SPIRE นี้ถูกใช้ในเทคโนโลยีอย่าง Istio ด้วย SPIFFE มีประโยชน์ใช้งานมากมาย ได้แก่ การแปลอัตลักษณ์ (identity translation), การยืนยันตัวตน client ของ OAuth, mTLS “encryption everywhere”, และการสังเกตการณ์การระดับการทำงาน (workload observability) ThoughtWorks กำลังทำงานรวมกับชุมชนของ Istio และ SPIFFE เพื่อเชื่อมช่องว่างระหว่างผู้ให้บริการระบุตัวตนของเซอร์วิสแบบเก่าเข้ากับ ระบบระบุตัวตนที่ใช้ SPIFFE เพื่อให้ mTLS สามารถถูกใช้งานได้ทุกที่ระหว่างเซอร์วิส ทั้งในและนอก service mesh

Tendermint

ประเมิน

Byzantine fault tolerance (BFT) เป็นปัญหาพื้นฐานในระบบ cryptocurrency และ blockchain ที่ทั้งระบบจะต้องเห็นชอบกับค่าของข้อมูลที่ถูกต้องเพียงค่าเดียว ในขณะที่มีข้อมูลที่ผิดปรากฏอยู่ รวมถึงข้อมูลจากผู้ประสงค์ร้ายด้วย Tendermint เป็นเอนจิ้น (engine) สำหรับผลิตซ้ำ BFT state machine ที่ช่วยให้คุณสร้างระบบ blockchain ของคุณเอง เอนจิ้นมติเอกฉันท์ (consensus engine) ที่เรียกว่า Tendermint Core ทำหน้าที่ดูแลส่วนการติดต่อแบบ peer-to-peer และดูแลระบบยืนยันมติเอกฉันท์ให้ คุณเพียงต้องสร้างส่วนที่เหลือ (เช่น การทำธุรกรรม และการยืนยันลายเซ็นเข้ารหัส) และติดต่อกับ Tendermint Core ผ่าน ABCI เอง ในขณะนี้ระบบ blockchain บางตัวเลือกใช้ Tendermint เป็นระบบมติเอกฉันท์แล้ว

TimescaleDB

ประเมิน

ในเรดาร์ครั้งก่อนเราได้พูดถึง PostgreSQL for NoSQL การเติบโตและการขยายตัวของ PostgreSQL นำไปสู่กระแสหลักของนวัตกรรมการเก็บข้อมูลแบบถาวร(persistence stores) ที่สร้างอยู่บน Postgres engine สิ่งหนึ่งที่เราได้ให้ความสนใจนั่นคือ TimescaleDB ฐานข้อมูลที่ปล่อยให้เราเขียนและ optimized queries ผ่านข้อมูล time-series แม้ว่าความสามารถยังไม่ครบถ้วนเท่า InfluxDB TimescaleDB ได้เสนอทั้งทางเลือกใหม่สำหรับรูปแบบการจัดเก็บข้อมูลและสมรรถนะของการคิวรี่ (query) คุณควรลองศึกษา TimescaleDB ถ้าคุณต้องการความสามารถในการขยายตัวแบบพอประมาณ ต้องการจะใช้ SQL ชอบความเสถียรและคุ้นเคยกับอินเตอร์เฟสการจัดการที่ PostgreSQL ได้นำเสนอไว้

เครื่องมือ

Cypress

นำไปใช้

เราได้รับการตอบรับที่ดีสำหรับ เครื่องมือทดสอบ Web UI ยุคหลังจาก Selenium อย่าง Cypress, TestCafe และ Puppeteer การทดสอบระบบทั้งระบบ (end-to-end) ทำให้เห็นความท้าท้ายในการทดสอบ ไม่ว่าจะเป็นระยะเวลาในการทดสอบที่นาน ความไม่เสถียรของบางกรณีทดสอบ และความท้าท้ายในการแก้ไขข้อผิดพลาดใน CI เมื่อรันชุดทดสอบในโหมด headless ทีมของเรามีประสบการณ์ที่ดีกับ Cypress ในการแก้ไขปัญหาเรื่องประสิทธิภาพ, การใช้เวลานานในการตอบสนอง และการโหลดทรัพยากร Cypress เลยกลายมาเป็นตัวเลือกสำหรับ การทดสอบระบบทั้งระบบภายในทีมเรา

Jupyter

นำไปใช้

ในไม่กี่ปีมานี้ เราได้เห็นการเติบโตในความนิยมของ analytics notebooks ซึ่งนี่คือ แอพพลิเคชันที่ได้แรงบันดาลใจมาจาก Mathematica ที่มาพร้อมกับตัวหนังสือ การนำเสนอภาพข้อมูล (visualization) และโค้ดบนเอกสารที่ดูมีชีวิตชีวาและคำนวนได้ Jupyter Notebook ถูกใช้อย่างกว้างขวางโดยทีมของพวกเราสำหรับการสร้างต้นแบบ (prototyping) และการสำรวจสำหรับการวิเคราะห์และ machine learning เราได้ย้าย Jupyter ไปในเกณฑ์ นำไปใช้สำหรับ tech radar เล่มนี้ เพื่อบอกว่ามันเป็นตัวพื้นฐานของ Python notebooks ไปแล้ว อย่างไรก็ตามเราแนะนำว่าให้ใช้ Jupyter Notebooks ในproduction ด้วยความระมัดระวัง

LocalStack

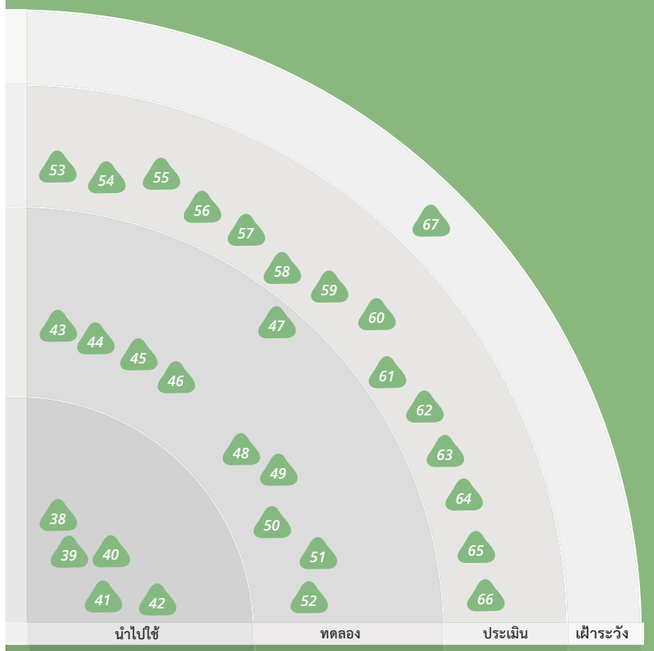
นำไปใช้

ความท้าทายอย่างหนึ่งในการใช้บริการคลาวด์ คือการพัฒนาและทดสอบบนเครื่องของตัวเอง LocalStack ช่วยแก้ไขปัญหานี้บน AWS ด้วยการทำส่วนจำลอง (test double) ของบริการต่างๆของ AWS ได้แก่ S3, Kinesis, DynamoDB, และ Lambda LocalStack ทำงานอยู่บนเครื่องมือชั้นดีอื่นๆ เช่น Kinesalite, dynalite และ Moto แถมยังเพิ่มฟีเจอร์อย่างการทำ isolated process และ error injection
LocalStack นั้นใช้ง่าย มาพร้อมกับ JUnit runner และ JUnit 5 และยังทำงานใน คอนเทนเนอร์ของ docker ได้ด้วย LocalStack กลายเป็นทางเลือกแรกสำหรับใช้ทดสอบเซอร์วิสที่ทำงานอยู่บน AWS

Terraform

นำไปใช้

Terraform ได้กลายมาเป็นตัวเลือกอันดับต้นๆ อย่างรวดเร็วสำหรับเอาไว้สร้างและจัดการ โครงสร้างพื้นฐานบนคลาวด์ ด้วยการเขียนจัดการเชิงประกาศ (declarative) ตัวปรับแต่งของเซิฟเวอร์ที่ instantiated จาก terraform มักจะถูกเหลือไว้ให้จัดการโดย Puppet, chef หรือ Ansible เราชอบ Terraform เพราะ syntax ของไฟล์ดูแล้วสามารถอ่านง่ายและตัว terraform ก็ถูกซัพพอร์ทสำหรับผู้ให้บริการคลาวด์จำนวนหนึ่งโดยที่เราไม่จำเป็นต้องเข้าใจความหมายจากคำสั่งของผู้ให้บริการแต่ละเจ้า นอกจากนั้นชุมชนที่กระตือรือร้นก็เป็นคนซัพพอร์ทให้กับฟีเจอร์ใหม่ๆที่มาจากผู้บริการคลาวด์หลายๆเจ้า
จากที่พวกเราเคยกังวลถึง terraform ที่พูดไว้เมื่อสองปีก่อน ดูเหมือนว่ามีการพัฒนาและได้กลายเป็นผลิตภัณฑ์ที่เริ่มคงตัวพร้อมด้วยระบบนิเวศ (ecosystem) ที่ดี ซึ่งพิสูจน์แล้วว่ามันมีคุณค่าต่อโปรเจกต์ ปัญหาเกี่ยวกับการจัดการสถานะ ตอนนี้สามารถใช้สิ่งที่เรียกว่า “remote state backend” ได้ พวกเราได้ใช้มันกับ AWS S3 อย่างประสบความสำเร็จ



นำไปใช้
38. Cypress
39. Jupyter
40. LocalStack
41. Terraform
42. UI dev environments

ทดลอง
43. AnyStatus
44. AVA
45. batect
46. Elasticsearch LTR
47. Helm
48. InSpec
49. Lottie
50. Stolon
51. TestCafe
52. Traefik

ประเมิน
53. Anka
54. Cage
55. Cilium
56. Detekt
57. Flagr
58. Gremlin
59. Honeycomb
60. Humio
61. Kubernetes Operators
62. OpenAPM
63. Systems
64. Taurus
65. Terraform provider GoCD
66. Terratest

เฝ้าระวัง
67. Handwritten CloudFormation

เครื่องมือ

Storybook, React Styleguidist, Compositor และ MDX สร้าง environment ที่ครอบคลุมสำหรับการทำ UI component ซ้ำๆอย่างรวดเร็ว โดยโฟกัสไปที่ความร่วมมือระหว่างนักออกแบบประสบการณ์ผู้ใช้ (user experience) และนักพัฒนา

(UI dev environments)

เราเสียเวลาและแรงกายไปมากกับการพยายามทำให้ development environment ของโปรเจกต์หนึ่งๆ ทำงานบนคอมพิวเตอร์ของเราได้ batect ทำให้การติดตั้งและแจกจ่าย development environment เป็นไปอย่างง่ายดายด้วยการใช้ Docker เพื่อใช้สร้างระบบพัฒนาที่ทำงานบนคอนเทนเนอร์ (container)

(batect)

UI dev environments

นำไปใช้

เทคนิค DesignOps นั้นเป็นที่ยอมรับมากขึ้นเรื่อยๆ สำหรับหลายหลายทีม วิธีปฏิบัติและเครื่องมือในกลุ่มนี้จึงพัฒนาขึ้น UI dev environments ให้สภาพแวดล้อมที่ครอบคลุมสำหรับการทำ UI component ซ้ำๆ อย่างรวดเร็ว โดยโฟกัสไปที่ความร่วมมือระหว่างนักออกแบบประสบการณ์ผู้ใช้ (user experience) และนักพัฒนา ตอนนี้เรามีตัวเลือกในกลุ่มนี้อยู่เล็กน้อย คือ Storybook, React Styleguidist, Compositor และ MDX คุณสามารถใช้เครื่องมือเหล่านี้แยกกันในไลบรารี่ของคอมโพเนนท์ หรือในการพัฒนาระบบการออกแบบเช่นเดียวกับการเพิ่มเข้าไปในโปรเจกต์เว็บแอพพลิเคชั่นหลายๆ ทีมสามารถลดระยะเวลา UI feedback ของตัวเองลงได้และช่วยพัฒนาเวลาทำงานด้าน UI เพื่อเตรียมพร้อมสำหรับงานพัฒนา ซึ่งทำให้การใช้ UI dev environments เป็นตัวเลือกเริ่มต้นนั้นดูสมเหตุสมผลสำหรับเรา

AnyStatus

ทดลอง

นักพัฒนาซอฟแวร์มักจะ push คอมมิทเล็กๆ หลายๆ ครั้งอยู่ทุกวัน เรามักจะพึ่งพาระบบตรวจเฝ้าระวัง (monitors) ที่คอยบอกเราแน่ใจว่าสถานะ build ยังคงเป็นสีเขียว AnyStatus เป็นเดสก์ท็อปแอพพลิเคชั่นที่มีขนาดเล็กสำหรับ Windows ที่แสดงตัวชี้วัด (metrics) และจากเหตุการณ์ที่เกิดขึ้นจากแหล่งที่มาต่างๆ รวมกันไว้ที่เดียว ตัวอย่างเช่นผลของการ build และ releases หรือจะเป็น health checks สำหรับเซอร์วิสที่แตกต่างและ ตัวชี้วัดของระบบปฏิบัติการ เปรียบได้เหมือนกับ CCTray ที่โด้ปมาแล้ว AnyStatus ยังมีปลั๊กอินใน Visual Studio ให้ใช้อีกด้วย

AVA

ทดลอง

AVA เป็นเครื่องมือที่ใช้รันชุดทดสอบสำหรับ Node.js แม้ว่าภาษา JavaScript นั้นจะทำงานแบบ single-threaded, IO ใน Node.js สามารถทำงานแบบขนานได้เพราะลักษณะการทำงานของมันเป็นแบบอะซิงโคนัส (asynchronous) AVA ใช้ประโยชน์จากสิ่งนี้ในการรันชุดทดสอบแบบพร้อมกัน ซึ่งเป็นประโยชน์มากสำหรับการทดสอบที่เกี่ยว IO นอกจากนี้ AVA สามารถรันชุดทดสอบแต่ละไฟล์แยกโปรเซส (process) กันได้ ข้อดีก็คือจะทำให้รันชุดทดสอบทั้งหมดได้เร็วขึ้น และแต่ละชุดทดสอบสามารถรันบน สภาพแวดล้อม (environment) ของตัวเองได้ ทั้งนี้ AVA จะมีฟีเจอร์ (feature) ค่อนข้างน้อยกว่าหากเปรียบเทียบกับ framework อื่นที่มีฟีเจอร์ ครบถ้วนอย่างเช่น Jest ซึ่ง AVA มีแนวคิดที่เจาะจงและบังคับให้เขียนกรณีทดสอบให้เป็นหน่วยเล็กๆ

batect

ทดลอง

เราเสียเวลาและแรงกายไปมากกับการพยายามทำให้ development environment ของโปรเจกต์หนึ่งๆ ทำงานบนคอมพิวเตอร์ของเราได้ เป็นเวลาหลายปีที่เรานำวิธีการการทำงานแบบ “นำลงมาแล้วใช้ได้เลย” (check out and go) มาใช้ ด้วยการเขียนเป็นสคริปท์ที่สามารถเรียกใช้เพื่อสร้าง development environment ที่เริ่มต้นทำงานได้ทันที batect เป็นเครื่องมือโอเพนซอร์ส (open source) ที่พัฒนาโดย Thoughtworks เอง โดยมีจุดประสงค์เพื่อทำให้การติดตั้งและแจกจ่าย development environment เป็นไปอย่างง่ายดายด้วยการใช้ Docker คุณสามารถใช้ batect เป็นจุดเริ่มต้นในการเขียนสคริปท์เพื่อใช้สร้างระบบพัฒนาที่ทำงานบนคอนเทนเนอร์ (container) ที่คุณสามารถสร้างขึ้นมาแล้วทำงานได้เลยโดยที่ไม่ต้องเสียเวลากับการตั้งค่าบนคอมพิวเตอร์ของคุณเอง หากคุณเปลี่ยนแปลงการตั้งค่าหรือ dependency ของ development environment ใดๆ คุณก็สามารถแจกจ่ายออกไปโดยไม่ต้องเสียเวลาการนำไปติดตั้งบนคอมพิวเตอร์เครื่องอื่นๆ หรือบน CI agents เลย ถึงแม้เราชอบเครื่องมืออื่นๆ ที่มีลักษณะทำงานคล้ายๆ กันอยู่ เช่น Cage เราพบว่า batect กำลังเป็นที่นิยมมากขึ้นเรื่อยๆ ในทีมงานของเรา

Elasticsearch LTR

ทดลอง

ความท้าทายอย่างหนึ่งในการทำระบบค้นหาคือการทำให้ผลลัพธ์ที่ผู้ใช้สนใจที่สุดอยู่บนอันดับต้นๆของรายการผลลัพธ์ นี่เป็นสิ่งที่นำ การเรียนรู้เพื่อจัดลำดับ (learn to rank - LTR) มาช่วยได้ LTR คือกระบวนการที่ระบบค้นหาที่นำ machine learning มาใช้จัดลำดับข้อมูลที่ได้มา ถ้าคุณใช้งาน Elasticsearch อยู่ คุณสามารถจัดลำดับผลลัพธ์ตามความเกี่ยวข้องกับผู้ใช้ได้ด้วยโปรแกรมเสริม (plug-ins) ที่เรียกว่า Elasticsearch LTR ซึ่งใช้ RankLib ในการสร้างโมเดล machine learning เมื่อคุณค้นหาด้วย Elasticsearch คุณสามารถใช้โปรแกรมเสริมตัวนี้เพื่อประเมินคะแนนความเกี่ยวข้องของผลลัพธ์ในรายการค้นหา เราได้ใช้ Elasticsearch LTR ในสองสามโปรเจกต์แล้วมีผลลัพธ์ออกมาเป็นที่น่าประทับใจ สำหรับผู้ใช้ Solr นั้นก็มีโปรแกรมเสริมที่ใช้ LTR ไว้ให้ใช้เช่นกัน

Helm

ทดลอง

Helm เป็นตัวโปรแกรมจัดการชุดโปรแกรมสำหรับ Kubernetes โดยมี Charts เป็นชุดคำสั่ง เพื่อกำหนดการติดตั้งชุดโปรแกรมหนึ่งๆบน Kubernetes ทาง Helm มี repository ที่ดูแลเองอยู่อย่างเป็นทางการ เพื่อให้มันใจได้ว่า มันประกอบไปด้วย Charts ที่ติดตั้งชุดโปรแกรมที่เชื่อถือได้ Helm ประกอบไปด้วยสองส่วน ส่วนแรกคือ client สำหรับส่งชุดคำสั่งจัดการเรียกว่า Helm ส่วนที่สองคือเซิร์ฟเวอร์ที่ใช้จัดการคลัสเตอร์ (cluster) ที่เรียกว่า Tiller การติดตั้งเซิร์ฟเวอร์ Tiller ในคลัสเตอร์อย่างปลอดภัยเป็นหัวข้อที่กว้างและละเอียดอ่อน เราแนะนำให้ติดตั้งในคลัสเตอร์ที่มีการควบคุมสิทธิการเข้าถึงด้วยการแต่งตั้งบทบาท (role-based access control - RBAC) เราได้ใช้ Helm ในโปรเจกต์ของลูกค้าจำนวนหนึ่งแล้ว พบว่าด้วยศักยภาพในการช่วยบริหารจัดการโปรแกรมที่จำเป็น (dependency) การสร้างแบบแผนการติดตั้งโปรแกรมในคลัสเตอร์ และการจัดการคำร้องด้วย hook mechanism ทำให้การจัดการวงจรชีวิตของโปรแกรมบน Kubernetes เป็นไปด้วยความง่ายดาย อย่างไร้ก็ตาม เราแนะนำให้ใช้ Helm อย่างระมัดระวัง

เพราะรูปแบบการใช้ YAML ของ Helm ในการนิยามโปรแกรมที่จะถูกติดตั้งนั้นมีความซับซ้อนเข้าใจได้ยาก ส่วนตัว Tiller เองก็ยังไม่ถูกขัดเกลาดีนัก คาดว่า Helm 3 จะแก้ปัญหาเหล่านี้ในอนาคต

InSpec

ทดลอง

เราต้องทำอย่างไร องค์กรจึงจะให้อิสระในการทำงานแก่ทีม แต่ในขณะเดียวกันยังสามารถที่จะมั่นใจได้ว่าสิ่งที่ทีมทำนั้นปลอดภัยและตรงตามมาตรฐาน เราจะมั่นใจได้อย่างไรว่าเซิร์ฟเวอร์ของเรานั้น หลังจากถูกเปิดใช้งานแล้วจะไม่ถูกเปลี่ยนแปลงการตั้งค่าในภายหลัง. InSpec เกิดมาเพื่อเป็นคำตอบในการตรวจสอบมาตรฐาน และความปลอดภัยอย่างต่อเนื่อง อีกทั้งยังสามารถใช้ทดสอบโครงสร้างพื้นฐานในเรื่องทั่วๆ ไปได้อีกด้วย. Inspec ช่วยให้เราสามารถสร้างบททดสอบเชิงประกาศ (declarative tests) สำหรับโครงสร้างพื้นฐาน และนำไปทดสอบบนสภาพแวดล้อมต่างๆ ที่เราสร้างขึ้นมา ได้อย่างสม่ำเสมอแม้แต่บนสภาพแวดล้อมสำหรับใช้งานจริง. ทีมของเรายังชื่นชอบในการออกแบบที่รองรับส่วนต่อขยายในเรื่องของ resources และ matchers สำหรับแพลตฟอร์มต่างๆ. เราแนะนำให้ลองใช้ InSpec หากคุณต้องการความมั่นใจในเรื่องมาตรฐาน และความปลอดภัย

Lottie

ทดลอง

UI แอนิเมชันที่ดีจะสามารถช่วยพัฒนาประสบการณ์ที่ดีขึ้นของผู้ใช้งาน อย่างไรก็ตามการทำแอนิเมชันที่ปราณีตซ้ำ ๆ ในแอปนั้น เป็นเรื่องที่ท้าทายสำหรับนักพัฒนาซอฟแวร์ Lottie เป็นไลบรารีสำหรับ Android, iOS, เว็บ และ Windows ซึ่งสามารถนำไฟล์ json ที่นำออกจาก Adobe After Effects โดยใช้ Bodymovin เข้ามาใช้งานได้เลย พร้อมกับการแสดงผลแบบเนทีฟ (native) บนมือถือและเว็บ ทั้งนักออกแบบและนักพัฒนาซอฟแวร์จะสามารถใช้เครื่องมือที่พวกเขาถนัดและร่วมงานกันได้อย่างราบรื่น

Stolon

ทดลอง

High Availability (HA) คือ ความสามารถของระบบหนึ่งๆ ที่สามารถทำให้พร้อมใช้งานตลอดเวลา แต่การทำให้เกิด HA ใน instance ของ PostgreSQL นั้นเป็นสิ่งที่ยุ่งยาก เราจึงชอบที่จะใช้ Patroni ช่วยในการติดตั้งคลัสเตอร์ฐานข้อมูล PostgreSQL (PostgreSQL clusters) ให้เป็นไปอย่างรวดเร็ว Stolon เป็นอีกเครื่องมือหนึ่งที่เราใช้ในระดับโปรดักชั่นอย่างประสบความสำเร็จร่วมกับคลัสเตอร์ PostgreSQL ที่มี HA และจัดการด้วย Kubernetes แม้ว่า PostgreSQL นั้นมาพร้อมกับความสามารถในการทำ streaming replication ซึ่งช่วยให้เกิด HA ในตัวคลัสเตอร์เองอยู่แล้ว ก็ยังมีความท้าทายอย่างอื่นในการทำให้เกิด HA อีก เช่น การรับประกันการเชื่อมต่อกันระหว่างเครื่อง client กับเครื่อง master เราจึงชอบที่ Stolon ช่วยแก้ปัญหานี้ด้วยการบังคับให้ client เชื่อมต่อและส่งคำขอ (request) ไปยังเครื่อง master ที่ถูกต้องอยู่เสมอ

TestCafe

ทดลอง

เราได้รับการตอบรับที่ดีสำหรับ เครื่องมือทดสอบ Web UI ยุคหลังจาก Selenium อย่าง Cypress, TestCafe และ Puppeteer ตัว TestCafe ก็ปล่อยให้คุณเขียนชุดทดสอบโดยใช้ JavaScript หรือ TypeScript และรันบนเบราว์เซอร์ในตอนทดสอบได้ TestCafe มีคุณสมบัติมากมายรวมถึง การรันเทส out-of-the-box แบบขนาน และการจำลอง HTTP request TestCafe ใช้รูปแบบการดำเนินการแบบอะซิงโคนัส (asynchronous) โดยที่ไม่มีเวลารอที่ชัดเจน ซึ่งผลลัพธ์ออกมาคือชุดทดสอบมีความเสถียรมากขึ้น และตัว selector API ทำให้ง่ายในการเขียนในรูปแบบ PageObject ซึ่งตัว TestCafe เพิ่งทำการปล่อยเวอร์ชั่น 1.0.x ซึ่งพัฒนาเสถียรภาพและฟังก์ชันต่าง ๆ

Traefik

ทดลอง

Traefik เป็นเครื่องมือโอเพนซอร์สสำหรับทำ reverse proxy และ load balancing ถ้าคุณมองหา edge proxy ที่มี routing แบบง่ายๆ โดยไม่ต้องมีฟีเจอร์เต็มตัวแบบ NGINX และ HAProxy แล้วละก็ Traefik ก็เป็นทางเลือกที่ดีทางหนึ่ง โดยเป็นเราเตอร์ ที่อนุญาตให้เปลี่ยนการตั้งค่าได้โดยไม่ต้องรีโหลด, สนับสนุนการเก็บค่าตัวชี้วัด (metrics), ตรวจเฝ้าระวัง (monitoring) และ การตัดการเชื่อมต่อ (circuit breakers) ซึ่งเป็นฟีเจอร์ที่จำเป็นสำหรับไมโครเซอร์วิส (microservice) นอกจากนี้ยังสนับสนุนการใช้ Let's Encrypt สำหรับ SSL termination และสามารถทำงานร่วมกับเซอร์วิสในชั้นโครงสร้างพื้นฐานได้ดี ไม่ว่าจะเป็น Kubernetes, Docker Swarm, หรือ Amazon ECS โดยที่มันสามารถตรวจพบเซอร์วิสหรืออินสแตนท์ (Instance) ใหม่ได้โดยอัตโนมัติ เพื่อนำเข้ามาเป็นส่วนหนึ่งในการทำ load balancing

Anka

ประเมิน

Anka เป็นชุดเครื่องมือสำหรับการสร้าง, จัดการ และเผยแพร่ environment สำหรับการพัฒนาและทดสอบซอฟต์แวร์ใน iOS และ macOS มันสร้าง environment ของระบบปฏิบัติการ macOS ที่มีคุณลักษณะเดียวกับ Docker คือ เริ่มต้นได้ทันที มีชุดคำสั่งสำหรับจัดการ virtual machine (VM) และ registry เช่น การแบ่งเวอร์ชั่น และการแท็ก VM เพื่อเผยแพร่ต่อไป เราค้นพบ Anka เมื่อเราเสนอโปรเจ็คคลาวด์ส่วนตัวบนระบบปฏิบัติการ macOS ให้แก่ลูกค้า มันเป็นเครื่องมือที่คุ้มค่าที่จะลองศึกษาดูเมื่ออยากทำ DevOps บนระบบปฏิบัติการ iOS และ macOS

เครื่องมือ

เชื่อมั่น แต่ตรวจสอบเสมอ Inspec ช่วยให้เรามั่นใจว่าหลังจากเซิร์ฟเวอร์ถูกเปิดใช้งานแล้ว จะปลอดภัยและตรงตามมาตรฐานตลอดอายุการใช้งาน

(InSpec)

ใช้ Linux eBPF สร้างเครือข่ายและระบบความปลอดภัยที่รู้จัก API ต่างๆ ขึ้นมา โดยอ้างอิงจากอัตลักษณ์ของเซอร์วิส, pod หรือคอนเทนเนอร์นั้นๆ นอกจากนี้มันทำงานกับไมโครเซอร์วิสได้แล้ว มันยังรับมือไมโครเซอร์วิสที่เปลี่ยนแปลงเสมอได้ด้วย

(Cilium)

เครื่องมือ

Detekt เป็นเครื่องมือวิเคราะห์ซอร์สโค้ดสำหรับภาษา Kotlin สำหรับตรวจสอบโค้ดเพื่อหาส่วนที่ซับซ้อนและอาจมีปัญหาในอนาคต เครื่องมือนี้มีชุดคำสั่งให้ใช้ หรือใช้มันร่วมกับเครื่องมือสำหรับการพัฒนาอื่นๆ ก็ได้

(Detekt)

Humio เป็นผู้เล่นที่ค่อนข้างใหม่ในวงการการจัดการบันทึกข้อมูลของระบบ (log management) มันถูกสร้างขึ้นโดยคำนึงถึงความรวดเร็วเป็นหลักมาตั้งแต่ต้นทั้งในด้านการบันทึกและการเรียกดู ด้วยการใช้ภาษาคิวรี (query) ของ Humio เอง ที่ทำงานอยู่บนฐานข้อมูลที่ถูกออกแบบมาสำหรับข้อมูลเชิงอนุกรมเวลา

(Humio)

Cage

ประเมิน

Cage เป็นโอเพนซอร์สที่ครอบ Docker Compose ไว้ โดยให้คุณปรับแต่งและรันคอมโพเนนท์หลายตัวเป็นแอพพลิเคชั่นขนาดใหญ่ คุณสามารถจัดการการดำเนินการของคอมโพเนนท์ เหมือนกับ Docker images เซอร์วิสที่โค้ดมาจาก repo สคริปต์ที่โหลด datastores และ pods ซึ่งเป็นคอนเทนเนอร์ที่รันด้วยกันรวมเป็นหนึ่งหน่วย Cage ใช้การปรับแต่งรูปแบบไฟล์เป็นแบบ Docker Compose v2 ที่ได้เข้าถึงช่องว่างบางอย่างของ Docker Compose เช่นการรองรับ environments หลายอย่าง รวมไปถึง dev environment สำหรับรัน distributed application บนเครื่องของนักพัฒนาซอฟแวร์ และบน environment สำหรับการทดสอบแบบ integration และบนโปรดักชั่น

Cilium

ประเมิน

Linux มีส่วนจัดการความปลอดภัยของเครือข่ายโดยพื้นฐานอย่าง iptables ที่ทำงานด้วยการคัดกรองหมายเลขไอพีแอดเดรสและหมายเลขพอร์ต TCP/UDP แต่ในระบบแบบไมโครเซอร์วิส (Microservice) ที่มีการเปลี่ยนแปลงเสมอ ไอพีแอดเดรสของเซอร์วิสต่างๆมีการเปลี่ยนแปลงบ่อยๆทำให้การตั้งค่าความปลอดภัยด้วย iptables ทำได้ยาก
Cilium ใช้ Linux eBPF สร้างเครือข่ายและระบบความปลอดภัยที่รู้จัก API ต่างๆ ขึ้นมา ด้วยการทำให้ความปลอดภัยขึ้นอยู่กับการรู้จักอัตลักษณ์ของเซอร์วิส pod หรือคอนเทนเนอร์นั้นๆ แทนการรู้จักไอพีแอดเดรส ด้วยการแยกระบบความปลอดภัยออกจากไอพีแอดเดรส มีความเป็นไปได้ที่ Cilium จะสร้างเลเยอร์ความปลอดภัยของเครือข่ายชั้นใหม่ขึ้นมาในวงการด้านความปลอดภัย ซึ่งเราแนะนำให้คุณลองดู

Detekt

ประเมิน

Detekt เป็นเครื่องมือวิเคราะห์ซอร์สโค้ดสำหรับภาษา Kotlin สำหรับตรวจสอบโค้ดเพื่อหาส่วนที่ซับซ้อนและอาจมีปัญหาในอนาคต เครื่องมือนี้มีชุดคำสั่งให้ใช้ แต่คุณก็สามารถใช้งานร่วมกันกับเครื่องมืออย่าง Gradle (เพื่อวิเคราะห์โค้ดขณะสร้างแพ็คเกจ) หรือ SonarQube (เพื่อเพิ่มความสามารถการทำ code coverage ที่นอกเหนือจากการวิเคราะห์ static code) และ IntelliJ ก็ได้ Delekt เป็นเครื่องมือที่ดีสำหรับทำไปป์ไลน์การสร้างแพ็คเกจของแอพพลิเคชั่น Kotlin

Flagr

ประเมิน

ความสามารถในการเปิด-ปิดฟีเจอร์ (Feature toggles) เป็นสิ่งสำคัญในการพัฒนาแบบ continuous deployment ถึงแม้เราจะได้ค้นพบแนวทางดีๆ ในแบบของเราเองในการทำการเปิด-ปิดฟีเจอร์จำนวนหนึ่งแล้ว แต่เราก็ชอบแนวทางที่ Flagr เลือกใช้: การเปิด-ปิดฟีเจอร์เหมือนกับเป็นเซอร์วิสตัวหนึ่งที่อาศัยอยู่ในคอนเทนเนอร์ของมันเอง Flagr มี SDK ไว้ให้ใช้ในภาษาหลักๆ หลายภาษา มีคู่มือ REST API อย่างดีเข้าใจง่าย และมาพร้อมกับฟรอนท์เอนด์ที่ใช้ง่าย

Gremlin

ประเมิน

Gremlin เป็นเครื่องมือทดสอบความสามารถในการคงอยู่หรือการฟื้นตัวคืนสภาพกลับมาเป็นปกติของระบบเมื่อเกิดความเสียหายเกิดขึ้น เช่น เครื่องหยุดการทำงานเน็ตเวิร์คมีปัญหา หรือที่รู้จักในการทดสอบแบบโกลาหล (Chaos Experiments) โดย Gremlin นั้นทำหน้าที่เป็นเครื่องมือสำหรับจำลองสถานการณ์ต่างๆ ให้เกิดขึ้น เพื่อตรวจดูว่าระบบของเรายังสามารถทำงานได้ตามสถานการณ์ที่เกิดขึ้นหรือไม่ นอกจากนี้ Gremlin ยังมีส่วนติดต่อกับผู้ใช้(user interface) ที่ง่ายต่อการใช้งานอยู่บนเว็บไซต์ ทำให้การบริหารจัดการทดสอบแบบ Chaos Experiments สามารถกระทำได้โดยง่าย โดยเฉพาะผู้ใช้ Kubernetes สามารถติดตั้งได้โดยง่าย ผ่าน Helm เพื่อติดตั้ง Gremlin และยังรันตามความต้องการหรือ กำหนดเป็นช่วงเวลาทดสอบเป็นประจำได้ด้วย

Honeycomb

ประเมิน

Honeycomb คือเครื่องมือสังเกตการณ์ที่สุ่มตัวอย่างแบบไดนามิกจากข้อมูล (log) ปริมาณมหาศาลบนระบบโปรดักชั่น นักพัฒนาสามารถบันทึกข้อมูลเหล่านี้ไว้เพื่อวิเคราะห์ในภายหลังได้ ระบบในทุกวันนี้กระจายตัวอยู่บนเซิร์ฟเวอร์หลายๆตัว และมีความซับซ้อนสูงทำให้ตัดสินใจได้ยากว่าข้อมูลชุดไหนที่จำเป็นในภายหลัง แนวทางการจัดการกับ log แบบสุ่มข้อมูลทั้งหมดเพื่อจัดเก็บของ Honeycomb จึงมีประโยชน์มากๆ

Humio

ประเมิน

Humio เป็นผู้เล่นที่ค่อนข้างใหม่ในวงการการจัดการบันทึกข้อมูลของระบบ (log management) มันถูกสร้างขึ้นโดยคำนึงถึงความรวดเร็วเป็นหลักมาตั้งแต่ต้นทั้งในด้านการบันทึกและการเรียกดู ด้วยการใช้ภาษาคิวรี (query) ของ Humio เอง ที่ทำงานอยู่บนฐานข้อมูลที่ถูกออกแบบมาสำหรับข้อมูลเชิงอนุกรมเวลา โดยเฉพาะ Humio ทำงานได้กับโปรแกรมอื่นที่มีอยู่เกือบทุกอย่าง ไม่ว่าจะเป็นเครื่องมือในการนำเข้าข้อมูล (ingestion), การแสดงผลข้อมูลด้วยภาพ (visualization) หรือ การแจ้งเตือน (alerting)
ด้านแวดวง log management นั้นมักจะนิยมใช้ Splunk กับ ELK Stack เป็นเวลานานแล้ว ทำให้เราสนใจอยู่ไม่น้อยในการจับตาดู Humio เป็นพิเศษในฐานะนักพัฒนาหน้าใหม่ในวงการ

Kubernetes Operators

ประเมิน

เราตื่นเต้นกับสิ่งที่ Kubernetes นำมาสู่อุตสาหกรรม แต่ก็กังวลกับความซับซ้อนที่เพิ่มขึ้นในการทำงานด้วยการตั้งคลัสเตอร์(cluster) ของ Kubernetes ขึ้นมาและดูแลจัดการ package ในคลัสเตอร์เป็นสิ่งที่ใช้เวลาและอาศัยความสามารถเฉพาะทาง

กระบวนการเบื้องหลังของคลัสเตอร์ เช่น อัพเกรด (upgrade), การขนย้าย (migration), การสำรอง (backup), และอื่นๆ นับเป็นงานเต็มเวลาได้เลย เราคิดว่า Kubernetes Operators จะเป็นตำแหน่งที่มีบทบาทสำคัญในการลดความซับซ้อนในส่วนนี้ลง และจะทำให้เกิดกลไกที่เป็นมาตรฐานที่ช่วยนิยามกระบวนการเบื้องหลังอัตโนมัติต่างๆ ของเกี่ยวกับแพ็คเกจใน Kubernetes ตอนนี้ RedHat กำลังเป็นกำลังหลักที่ส่งเสริมการสร้างตำแหน่ง Kubernetes Operators ขึ้นมา ในขณะเดียวกันตำแหน่งนี้ก็เริ่มเกิดขึ้นในชุมชนนักพัฒนาแพ็คเกจที่เป็นโอเพนซอร์สต่างๆ เช่น Jaeger, MongoDB และ Redis

OpenAPM

ประเมิน

ความท้าทายหนึ่งของการนำโปรแกรมโอเพนซอร์สมาใช้แทนโปรแกรมเชิงพาณิชย์ที่เป็นที่นิยมคือการมองหาโปรแกรมที่คุณต้องการในบรรดาโปรเจกต์ต่างๆ มีมุมมองหลายๆ อย่างที่ต้องพิจารณา เช่น โปรแกรมตัวไหนทำงานกับตัวไหนได้ดี หรือตัวไหนครอบคลุมปัญหาส่วนไหนบ้าง นี่เป็นเรื่องที่ยากเป็นพิเศษในการเลือกใช้เครื่องมือสังเกตการณ์ (observability tool) สิ่งที่ทำกันทั่วไปคือซื้อเครื่องมือราคาแพงที่ทำได้ทุกอย่างมาใช้ OpenARM ช่วยให้การเลือกเครื่องมือสังเกตการณ์แบบโอเพนซอร์สเป็นไปโดยง่าย โดยโปรแกรมได้คัดกรองแพ็คเกจชั้นนำมาให้เลือกใช้ และแบ่งออกเป็นประเภทต่างๆ ตามความสามารถ ตราบใดที่คุณอัพเดท OpenARM ให้ล่าสุดอยู่เสมอ มันสามารถช่วยนำทางคุณไปเจอเครื่องมือที่เหมาะสม

Systems

ประเมิน

มันคงง่ายที่จะมองกระบวนการทำงานของเราเป็นเส้นเวลาเป็นลำดับขบวนของเหตุการณ์และผลลัพธ์เชื่อมต่อกันเป็นเส้นเดียว แต่ในความเป็นจริงมันคือระบบอันซับซ้อนที่ประกอบไปด้วยสัญญานทางบวกและลบที่ถูกส่งกลับเข้าระบบเพื่อประมวลผลลัพธ์ออกมา Systems คือชุดเครื่องมือสำหรับอธิบาย สั่งงาน และแสดงแผนภาพ ของขั้นตอนกระบวนการทำงาน มีภาษาเฉพาะสำหรับใช้สั่งงาน และสามารถใช้งานโดยตรงหรือผ่าน jupyter notebook ก็ได้
Systems สามารถช่วยให้กระบวนการทำงานที่ค่อนข้างซับซ้อนดูเข้าใจง่าย และแสดงความเคลื่อนไหวของข้อมูลในกระบวนการให้เห็นอย่างง่ายดาย มันเป็นเครื่องมือที่เหมาะกับการใช้งานเฉพาะกลุ่มแต่ก็น่าสนใจและใช้งานสนุกอยู่ไม่น้อย

Taurus

ประเมิน

Taurus เป็นแอพพลิเคชั่นที่มีประโยชน์ในการทดสอบประสิทธิภาพ (performance) ของเซอร์วิส ถูกเขียนโดยใช้ Python ซึ่ง Taurus ได้รวบรวมชุดปฏิบัติการทดสอบประสิทธิภาพ อย่าง Gatling และ Locust คุณสามารถรันผ่าน command line และ สามารถใช้งานร่วมกับไปป์ไลน์ได้อย่างง่ายดาย ในการรันการทดสอบประสิทธิภาพในขั้นตอนต่างๆ ของไปป์ไลน์ Taurus ยังมีการรายงานผลที่ยอดเยี่ยมไม่ว่าจะเป็นแบบแสดงในรูปแบบตัวอักษร หรือ แสดงผ่านเว็บ UI ทีมของเราค้นพบว่าการปรับแต่งไฟล์ YAML ของ Taurus นั้นง่ายเพราะคุณสามารถใช้ไฟล์ ในการอธิบายการทดสอบแต่ละอย่างและ อ้างถึงชุดปฏิบัติการสำหรับสถานการณ์ต่าง ๆ

Terraform provider GoCD

ประเมิน

Terraform provider GoCD ช่วยให้คุณสามารถสร้างไปป์ไลน์ (pipeline) ด้วย Terraform ซึ่งเป็นเครื่องมือที่นิยมใช้อย่างกว้างขวางใน infrastructure as code ด้วยเครื่องมือนี้คุณจะสามารถเขียนไปป์ไลน์ได้ด้วยภาษา HashiCorp Configuration Language (HCL) ซึ่งจะสามารถใช้งานทุกฟังก์ชั่นที่มีใน Terraform ทั้ง workspaces, modules และ remote state วิธีนี้เป็นอีกทางเลือกที่ดีมากของ Gomatic ที่เราได้เคยแนะนำในส่วน Pipelines as code มาแล้ว นอกจากนี้ Golang SDK ที่ใช้ในบริการนี้ยังมีขั้นตอนการทำ regression test สำหรับ GoCD API อัตโนมัติ ซึ่งช่วยลดภาระในระหว่างการอัพเกรดได้

Terratest

ประเมิน

Terraform เป็นชุดคำสั่งสำหรับการจัดสรรและกำหนดค่าโครงสร้างพื้นฐานบนคลาวด์ที่เรานำมาใช้อย่างกว้างขวาง Terratest เป็นไลบรารีของ Golang ที่ทำให้เราเขียนระบบทดสอบอัตโนมัติได้ง่ายขึ้น เมื่อเราสั่งเริ่มการทำงาน มันจะสร้างโครงสร้างพื้นฐานบน AWS ขึ้นมาจริงๆ เช่น เซิร์ฟเวอร์, ไฟร์วอลล์ (firewall), หรือโหลดบาลานเซอร์ (load balancers) หลังจากนั้นมันจะติพลอยแอปแอพพลิเคชั่นขึ้นไป และตรวจสอบพฤติกรรมที่ควรจะเป็นด้วย Terratest หลังจากการทดสอบจบแล้ว Terratest สามารถทำลายแอพพลิเคชั่นและเซอร์วิสที่ถูกสร้างขึ้นบนคลาวด์ให้ได้ ทำให้มันมีประโยชน์ในการทำการทดสอบโครงสร้างพื้นฐานแบบ end-to-end ในสภาพแวดล้อมการทำงานจริง

Handwritten CloudFormation

เฝ้าระวัง

AWS CloudFormation เป็นภาษา declarative เชิงพาณิชย์สำหรับสร้างโครงสร้างพื้นฐานใน AWS ด้วยการเขียนโค้ด Handwritten CloudFormation มักเป็นทางเลือกหลักในการบูตแสตรปโครงสร้างพื้นฐานใน AWS แบบอัตโนมัติ ถึงแม้มันดูจะเป็นทางเลือกที่สมเหตุสมผลในการเริ่มโปรเจ็คขนาดเล็ก ทีมของเราและอุตสาหกรรมโดยรวมพบว่า Handwritten CloudFormation ไม่สามารถขยายตัวตามขนาดโครงสร้างพื้นฐานได้ ข้อเสียที่เห็นได้ชัดในโปรเจ็คขนาดใหญ่คือ อ่านยาก ขาดการสร้างกฎเกณฑ์ที่สำคัญ มีคำสั่งให้ใช้น้อย และขาด type checking ความพยายามในการแก้ปัญหาเหล่านี้ทำให้เกิดระบบนิเวศที่หลากหลายอันประกอบด้วยเครื่องมือโอเพนซอร์สหลายๆตัว เราพบว่า Terraform คือทางเลือกหลักที่เหมาะสมที่นอกจากจะแก้จุดอ่อนเหล่านี้ของ CloudFormation แล้ว มันยังมีชุมชนที่คอยเพิ่มและสนับสนุนฟีเจอร์ใหม่ของ AWS และคอยแก้ไขบั๊กที่เกิดขึ้น นอกเหนือจาก Terraform แล้ว คุณสามารถเลือกเครื่องมือและภาษาอื่นๆ สำหรับงานนี้ เช่น troposphere, sceptre, Stack Deployment Tool และ Pulumi

เครื่องมือ

Systems คือชุดเครื่องมือสำหรับอธิบาย สั่งงาน และแสดงแผนภาพของระบบอันซับซ้อนที่ประกอบไปด้วยสัญญานทางบวกและลบที่ถูกส่งกลับเข้าระบบเพื่อประมวลผลลัพธ์ออกมา

(Systems)

Terraform provider GoCD ช่วยให้คุณสามารถสร้างไปป์ไลน์ (pipeline) ด้วย Terraform ซึ่งเป็นเครื่องมือที่นิยมใช้อย่างกว้างขวางใน infrastructure as code นอกจากนี้มันยังมีการทำ regression test สำหรับ GoCD API อัตโนมัติ ซึ่งช่วยลดภาระในระหว่างการอัพเกรดได้

(Terraform provider GoCD)

ภาษาและเฟรมเวิร์ก

Apollo

นำไปใช้

ทีมของเราพบว่า Apollo กลายเป็นไลบรารีที่เราเลือกใช้ เมื่อเราต้องการสร้างแอพพลิเคชั่นด้วย React ที่ใช้ GraphQL เพื่อเข้าถึงข้อมูลจากแบ็คเอนด์ แม้ว่า Apollo ได้เตรียมเฟรมเวิร์กฝั่งเซิร์ฟเวอร์และเกทเวย์ของ GraphQL ไว้ให้ แต่ Apollo client เป็นสิ่งที่ทำให้เราสนใจ เพราะมันทำให้การผูกการเชื่อมต่อ UI component เข้ากับข้อมูลจาก GraphQL เรียบง่ายขึ้น นั่นหมายความว่ามันทำให้เราเขียนโค้ดน้อยกว่าการใช้ REST และ redux

MockK

นำไปใช้

MockK เป็นเครื่องมือหลักที่เราใช้ทำ mocking สำหรับเขียนทดสอบแอพพลิเคชั่นภาษา Kotlin เราชอบไลบรารีนี้เพราะมันสนับสนุนการใช้ฟีเจอร์ของ Kotlin เป็นอย่างดี เช่น coroutines หรือ lambda blocks ด้วยความที่มันเป็นไลบรารีที่เขียนขึ้นสำหรับ Kotlin โดยเฉพาะมันทำให้เราเขียนเทสต์ได้สั้นและกระชับกว่าการใช้ wrapper ของ Mockito หรือ PowerMock

TypeScript

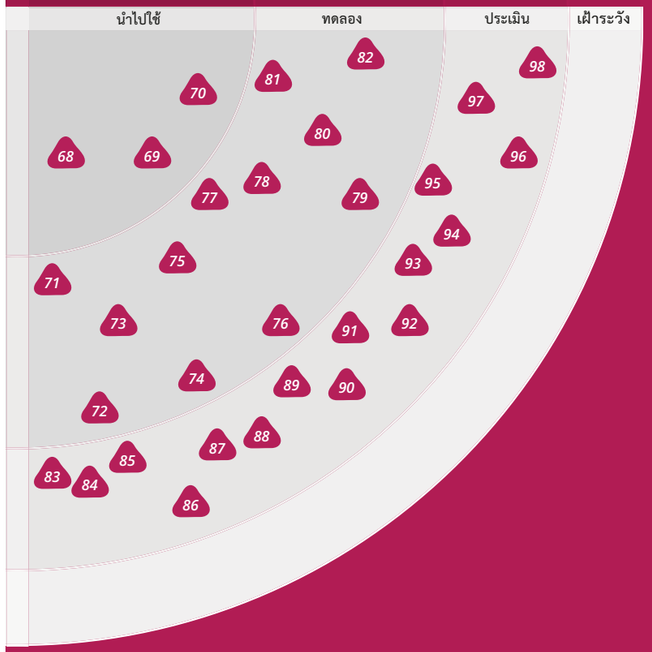
นำไปใช้

TypeScript เป็นภาษาที่กำหนดชนิด (ข้อมูล) ตั้งแต่ตอนคอมไพล์ (statically typed language) ที่เป็นซูเปอร์เซต (superset) ของ JavaScript ได้กลายมาเป็นตัวเลือกแรกๆของเรา โดยโปรเจ็คใหญ่ๆ ได้ประโยชน์อย่างมากจากการที่มี type safety นักพัฒนาของเราชอบที่มันใช้งานได้โดยไม่ต้องปรับแต่งมากนัก ทำงานได้ดีกับ IDE และ การที่สามารถปรับปรุงโค้ดให้ดีขึ้น (refactor code) ได้อย่างปลอดภัยและค่อยๆ ใส่ type เข้าไปได้ การที่มีการนิยามชนิดข้อมูลที่ดีของ TypeScript-type ทำให้เราสามารถนำไปใช้ร่วมกับทุกไลบรารีของ JavaScript ที่มีอยู่มากมาย ในขณะที่ยังได้ความสามารถของ type safety อีกด้วย

Apache Beam

ทดลอง

Apache Beam เป็นโมเดลโอเพนซอร์สสำหรับการเขียนโปรแกรมเพื่อเป็นมาตรฐาน (unified programming model) เพื่อใช้ออกแบบและสั่งงานไปป์ไลน์สำหรับประมวลผลข้อมูลทั้งแบบ batch และสตรีมมิ่ง ซึ่ง Beam เป็นโมเดลที่อ้างอิงจากโมเดล Dataflow อีกทีหนึ่ง มันช่วยทำให้เราสามารถออกแบบ logic ที่อนุญาตให้สลับประมวลผลระหว่างข้อมูลแบบ batch, windowed batch และสตรีมมิ่งได้ เนื่องจากระบบนิเวศ (ecosystem) ของการประมวลผล big data นั้นได้พัฒนามาไกลแล้วและมีเครื่องมือให้เลือกใช้มากมาย ทำให้การเลือกใช้เอนจิ้นสำหรับประมวลผลข้อมูลที่เหมาะสมเป็นสิ่งที่ยาก สาเหตุสำคัญที่ทำให้เลือกใช้ Beam คือมันช่วยให้เราสลับใช้ระหว่างตัวประมวลต่างๆ ได้ เช่น Apache Spark, Apache Flink, Google Cloud Dataflow หรือ แม้แต่ Apache Samza ที่เป็นเครื่องมือใหม่ที่ออกมาเมื่อสองสามเดือนก่อนก็ได้รับการสนับสนุนจาก Beam แล้ว เครื่องมือแต่ละตัวมีศักยภาพที่แตกต่างกันและการทำให้ API ใช้งานร่วมกันได้นั้นเป็นเรื่องยาก Beam พยายามแก้ปัญหานี้ด้วยการพยายามสร้างสมดุลย์ระหว่างการดึงนวัตกรรมของเครื่องมือเหล่านี้มาใช้ กับการผลักดันชุมชนนักพัฒนาต่างๆให้ออกแบบเครื่องมือที่เหมาะสมกับโมเดลของ Beam ตัว Beam เองมี SDK ให้ใช้ในหลายๆ ภาษา ได้แก่ Java, Python และ Golang นอกจากนี้เราก็เคยใช้ Scio ซึ่งเป็น wrapper ของ Beam ที่ทำให้ใช้กับภาษา Scala ได้



นำไปใช้
68. Apollo
69. MockK
70. TypeScript

ทดลอง
71. Apache Beam
72. Formik
73. HiveRunner
74. joi
75. Ktor
76. Laconia
77. Puppeteer
78. Reactor
79. Resilience4j
80. Room
81. Rust
82. WebFlux

ประเมิน
83. Aeron
84. Arrow
85. Chaos Toolkit
86. Dask
87. Embark
88. fastai
89. http4k
90. Immer
91. Karate
92. Micronaut
93. Next.js
94. Pose
95. react-testing-library
96. ReasonML
97. Taiko
98. Vapor

เฝ้าระวัง

ภาษาและเฟรมเวิร์ก

joi เป็นภาษาที่ใช้อธิบายโครงสร้างของข้อมูล (schema description language) และเป็นเครื่องมือตรวจสอบความถูกต้องของ JavaScript object ที่ไม่ขึ้นอยู่กับเฟรมเวิร์กของเว็บแอปพลิเคชันใดๆ

(joi)

Reactive มาพร้อมกับความสามารถในการขยายตัว (scalability) และการฟื้นตัว (resilience)ของระบบ ซึ่งทำให้การใช้งานยากขึ้นและมีส่วนที่ต้องตรวจสอบแก้ไขเพิ่มขึ้นด้วย สำหรับโปรเจกต์ของเราที่นำ Reactor และ Reactive มาใช้ เราสังเกตว่าความสามารถในการขยายตัวเพิ่มขึ้นอย่างเห็นได้ชัด

(Reactor)

Formik

ทดลอง

Formik เป็น higer-order component ที่มีประโยชน์ที่สามารถจัดการการเขียนฟอร์ม (form) ยุ่งยากและซับซ้อนใน React ให้ง่ายขึ้นมาก มันมีการจัดการ state ของแต่ละฟอร์มแยกภายในเอง, มันยังช่วยในการซับมิตฟอร์ม (form submit) และสามารถใช้ Yup ช่วยการให้ตรวจสอบ (validation) ข้อมูลเรียบง่ายขึ้นได้

HiveRunner

ทดลอง

HiveRunner เป็นเฟรมเวิร์กโอเพนซอร์สสำหรับทำ unit test กับคำสั่งคิวรี่ (query) สำหรับ Apache Hadoop Hive โดยทำงานอยู่บน JUnit4 สำหรับการเขียนเพื่อวิเคราะห์ข้อมูลเล็กๆ น้อยๆ หรือ ทำไปป์ไลน์ (pipeline) ของข้อมูลด้วย Hive SQL แล้ว เราพบว่า HiveRunner ช่วยทำให้เราเขียนการทดสอบคำสั่ง SQL ที่มีความซับซ้อนปานกลางได้ดี ซึ่งทำให้เราพัฒนา Hive SQL ในลักษณะ TDD ได้

joi

ทดลอง

joi เป็นภาษาที่ใช้อธิบายโครงสร้างของข้อมูล (schema description language) และเป็นเครื่องมือตรวจสอบความถูกต้องของ JavaScript object เราชอบที่ joi นั้นไม่ขึ้นอยู่กับเฟรมเวิร์กของเว็บแอปพลิเคชันใดๆ ทีมของเราเลยสามารถใช้โครงสร้างเดียวกัน กับชุดเทคโนโลยีที่ใช้พัฒนาที่แตกต่างกันได้ นอกจากนี้คุณยังสามารถใช้ไลบรารี่ที่มาด้วยกันในการสร้าง Swagger ของ API ที่ใช้ joi ตรวจสอบข้อมูลได้อีกด้วย

Ktor

ทดลอง

Kotlin ได้แสดงให้เห็นว่ามีคุณค่าเกินว่าที่จะเป็นแค่ไว้ใช้พัฒนาแอปมือถือ เมื่อสร้างไมโครเซอร์วิสและส่งมอบซอฟต์แวร์สู่มือลูกค้า ทีมของเรามีประสบการณ์ที่ดีกับ Ktor

Ktor เป็นเฟรมเวิร์กที่ไม่เหมือนเว็บเฟรมเวิร์กทั่วไปที่สนับสนุน Kotlin เพราะ Ktor ได้ถูกเขียนโดยใช้ Kotlin และใช้คุณสมบัติของภาษาอย่าง coroutines ซึ่งสามารถทำให้ประยุกต์ใช้ asynchronous nonblocking ได้ ความยืดหยุ่นในการใช้เครื่องมือต่างๆ สำหรับ logging, DI และ template engine นอกจากนี้ Ktor ยังเป็นสถาปัตยกรรมที่มีขนาดเล็ก ทำให้ Ktor เป็นทางเลือกที่น่าสนใจสำหรับการสร้างเซอร์วิสแบบ RESTful

Laconia

ทดลอง

Laconia เป็นเฟรมเวิร์กสำหรับพัฒนาฟังก์ชั่นของ AWS Lambda ในภาษา JavaScript เมื่อเทคโนโลยีเซิร์ฟเวอร์เลส (serverless) เติบโต ความซับซ้อนในการสร้างแอพพลิเคชั่นก็ซับซ้อนตามขึ้นไปด้วย Laconia เป็นเฟรมเวิร์กขนาดเล็กที่ช่วยเราจัดการปัญหาในการพัฒนาบน Lambda มันใช้ dependency injection ในการแยกโค้ดของแอพพลิเคชั่นของคุณออกจากส่วน API ระดับล่างของ AWS และมี adapter ให้แอพพลิเคชั่นของคุณทำงานตาม event ต่างๆ ให้เลือกใช้ นอกจากนี้ Laconia ยังทำงานได้ดีกับเฟรมเวิร์กเซอร์เลส (Serverless Framework) อื่นๆ ในขั้นตอนการดิพลอย (deploy) เราชอบเฟรมเวิร์กที่เล็กและง่าย และ Laconia ก็เป็นอย่างที่เราชอบเลย

Puppeteer

ทดลอง

คล้ายกับ Cypress และ TestCafe ตัว Puppeteer เองคือหนึ่งในเครื่องมือทดสอบ web UI ที่ได้รับการชื่นชมจากทีมของเรา Puppeteer มีการควบคุม headless เบราว์เซอร์ที่ละเอียด สามารถดึงค่าเวลามาทำการวินิจฉัยประสิทธิภาพและอื่นๆ ทีมของเราพบว่า Puppeteer มีความเสถียร รวดเร็ว และยืดหยุ่นกว่า WebDriver ทางเลือกตัวอื่นๆ

Reactor

ทดลอง

เราได้พูดถึง Reactor มาแล้วในเรดาร์ฉบับก่อนๆ และมันก็ยังเป็นที่นิยมมากขึ้นเรื่อยๆ ในโปรเจกต์ต่างๆ ของเรา ด้วยความที่ Reactor ทำงานร่วมกับระบบนิเวศ (ecosystem) ของ Spring ได้ดี ทำให้มันเป็นที่นิยมอันดับต้นๆในการทำ Reactive Streams ในขณะที่ระบบของ Reactive มาพร้อมกับความสามารถในการขยายตัว (scalability) และการฟื้นตัว (resilience) ของระบบ ซึ่งทำให้การใช้งานยากขึ้นและมีส่วนที่ต้องตรวจสอบแก้ไขเพิ่มขึ้นด้วย สำหรับโปรเจกต์ที่รับข้อเสียตรงนี้ได้ Reactor ก็เป็นทางเลือกที่ดี สำหรับโปรเจกต์ของเราที่นำ Reactor และ Reactive มาใช้ เราสังเกตว่าความสามารถในการขยายตัวเพิ่มขึ้นอย่างเห็นได้ชัด การเชื่อมต่อกับ RDBMS เป็นจุดอ่อนอย่างหนึ่งของเซอร์วิส Reactive ซึ่งก็มีการแก้ไขให้เราได้เห็นอย่าง R2DBC ที่เป็นไดรฟ์เวอร์ของ RDBMS ที่สนับสนุน Reactive

Resilience4j

ทดลอง

Resilience4j เป็นไลบรารีที่ช่วยป้องกันความผิดพลาดของระบบ (fault tolerance library) ซึ่งมีขนาดเล็กและได้แรงบันดาลใจจาก Hystrix ของ Netflix เราชอบที่ Resilience4j นั้นมีขนาดเล็กและมีการจำแนกการใช้งานออกเป็นโมดูลที่มีความสามารถเฉพาะตัว เช่น เทคนิคตัดการเชื่อมต่อกับเซอร์วิสที่ทำงานผิดปกติ (circuit-breaking), เทคนิคตัดการเชื่อมต่อที่มีความถี่สูงเกิน (rate-limiting), เทคนิคการพยายามเชื่อมต่ออีกครั้งหากล้มเหลว (retry), และเทคนิคควบคุมปริมาณการเรียกใช้งานพร้อมๆกัน (bulkhead) ขณะที่ในระดับ service meshes ของระบบกำลังเริ่มเพิ่มศักยภาพในการป้องกันความผิดพลาด แต่ไลบรารี่เหล่านี้ยังคงเป็นส่วนสำคัญในระบบอยู่ เพราะสำหรับเซอร์วิสที่ไม่ได้ติดตั้งอยู่ในคอนเทนเนอร์แล้ว ไลบรารี่นี้ สามารถรองรับความผิดพลาดแบบต่างๆ อย่างเฉพาะเจาะจงได้ดีกว่า เนื่องจาก Hystrix กำลังเข้าสู่ช่วงหยุดพัฒนาแล้ว Resilience4j จึงกลายเป็นทางเลือกอันดับแรกในระบบนิเวศ (ecosystem) ของ Java อีกทั้งยังทำงานได้ดีกับ

ทั้ง synchronous API และ reactive API นอกจากนี้ยังสามารถใช้โมดูลเสริมที่ใช้ส่งข้อมูลการทำงานของโมดูลเข้า dropwizard metrics หรือ Prometheus ได้อีกด้วย

Room

ทดลอง

Room เป็นไลบรารีสำหรับเรียกใช้ SQLite บนแอนดรอยด์ มันสามารถตรวจสอบความถูกต้องของ SQL ในเวลาคอมไพล์ทำให้โค้ดสำหรับเรียกใช้ฐานข้อมูลเรียบง่ายขึ้น มีส่วนซ้ำน้อยลง และตรงไปตรงมา นักพัฒนาของเราชอบที่มันทำงานรวมกับ LiveData ได้อย่างดีซึ่งทำให้เราสังเกตการทำงานของคิวรี่ได้ Room เป็นส่วนหนึ่งของ Android Jetpack ที่ถูกพัฒนาขึ้นเพื่อช่วยให้การพัฒนาแอพพลิเคชั่นสำหรับแอนดรอยด์ง่ายดายขึ้น

Rust

ทดลอง

ตั้งแต่ที่เราพูดถึง Rust ไว้ในเรดาร์ฉบับมกราคมปี 2015 เราก็เห็นความสนใจใน Rust เพิ่มมากขึ้นเรื่อยๆ เราพบว่าลูกค้าของเราบางแห่งได้นำ Rust มาใช้แล้วในส่วนของเครื่องมือการจัดการโครงสร้างพื้นฐานและอุปกรณ์ฝังตัว (embedded device) ระดับสูง ความสนใจที่เพิ่มมากขึ้นเกิดจากการเติบโตของระบบนิเวศ (ecosystem) และพัฒนาการของตัวภาษา โดยด้านภาษาประกอบด้วยการพัฒนาประสิทธิภาพการทำงานและด้านการใช้งานที่ง่ายขึ้น เช่น การเปลี่ยนไปใช้ non-lexical scoping และการเปลี่ยนแปลงครั้งสำคัญถูกรวมอยู่ใน Rust เวอร์ชั่นปี 2018 ที่ปล่อยสู่สาธารณะเมื่อเดือนธันวาคมที่ผ่านมา

WebFlux

ทดลอง

WebFlux เป็นการนำ Reactive Streams มาใช้ใน Spring Framework เราเห็นโมเดลการเขียนโปรแกรมแบบ reactive ที่ใช้กันทั่วไปในหลายๆทีมของเรา และทีมที่ทำงานในระบบนิเวศของ Spring อยู่ก็เลือกใช้ WebFlux ซึ่งใช้ได้ดีในระบบนิเวศของไมโครเซอร์วิสขนาดใหญ่ที่สมรรถนะในการรับมือ requests ที่เป็นสิ่งสำคัญ มันอนุญาตให้ดำเนินการ request หลายๆ ตัวพร้อมกันแบบอะซิงโคนัส (asynchronous) ได้โดยไม่ต้องใช้ thread หลายตัว WebFlux มี Reactor เป็นไลบรารีสำหรับ reactive แต่สามารถทำงานร่วมกับไลบรารี reactive แบบอื่นๆ ผ่าน Reactive Streams ได้อีกด้วย มันใช้ Netty เป็นเอนจิ้นสำหรับการติดต่อสื่อสารที่มีสมรรถนะสูง เราแนะนำการใช้ Reactive Streams รวมถึงการนำโมเดลการเขียนโปรแกรมแบบ reactive มาใช้ แต่การทำเช่นนั้นอาศัยการปรับเปลี่ยนแนวคิดในการเขียนโปรแกรมขนานใหญ่

Aeron

ประเมิน

Aeron เป็นระบบส่งข้อความแบบ peer-to-peer มันมีระบบบันทึก persistent log ของข้อความด้วย media driver จำนวนนึง ได้แก่ HTTP, UDP, และ TCP มันยังมีระบบเก็บข้อมูลแบบ persistent สำหรับสตรีมของข้อความ (message stream) เพื่อนำไปใช้ในภายหลัง ได้ Aeron อาจจะเกินความจำเป็นสำหรับแอพพลิเคชั่นหลายๆ ตัวเพราะว่ามันทำงานในระดับต่ำ (OSI Layer 4 ในทางทฤษฎี) แต่ด้วยการทำงานแบบ peer-to-peer และ มีเวลาแฝง (latency) ที่ต่ำและคาดเดาได้ ทำให้มันมีประโยชน์ในแอพพลิเคชั่น machine learning รวมถึงในสถาปัตยกรรมของระบบในลักษณะ event-driven เราคิดว่ามันคุ้มค่าที่จะชี้ให้เห็นว่ามันยังมีโปรโตคอลสำหรับส่งข้อความ (messaging protocol) อื่นๆ โดยที่ไม่ต้องใช้เซอร์วิสอื่นอย่าง Apache Kafka มาช่วยให้ทำงานได้

Arrow

ประเมิน

Arrow เป็นไลบรารีบน Kotlin สำหรับการเขียนโปรแกรมแบบ functional มันเกิดจากการรวมไลบรารีชื่อดังสองตัวเข้าด้วยกัน (kategory และ funKTionale) ในขณะที่ Kotlin มีฟีเจอร์พื้นฐานสำหรับการเขียนโปรแกรมแบบ functional ให้ใช้ Arrow เป็นแพ็คเกจสำเร็จรูปที่ช่วยให้เขียนง่ายขึ้น มันมี data type, type class, effect, topics และ รูปแบบการเขียนโปรแกรมแบบ functional แบบอื่นๆ ให้ใช้ รวมถึงส่วนเชื่อมต่อกับไลบรารีอื่นๆ ที่เป็นที่นิยมด้วย Arrow ช่วยทำให้ไลบรารีต่างๆ ทำงานด้วยกันได้ ซึ่งมีประโยชน์มากๆ ในการทำให้ชุมชนพัฒนาทำงานไปในทิศทางเดียวกันได้

Chaos Toolkit

ประเมิน

Chaos Toolkit เป็นหนึ่งในชุดเครื่องมือสำหรับ Chaos Engineering ที่ได้ลงในเรดาร์ฉบับนี้ คุณใช้เครื่องมือนี้ในการออกแบบและดำเนินการทดสอบที่ทำซ้ำได้กับโครงสร้างพื้นฐาน (infrastructure) ของคุณ เพื่อทำความเข้าใจความคงทนและสามารถในการฟื้นตัวในกรณีที่มีความล้มเหลวเกิดขึ้น ทีมของเราหลายๆทีมใช้เครื่องมือที่สร้างขึ้นเองในการทำสิ่งนี้อยู่แล้ว เราจึงดีใจที่ได้เห็นโปรเจ็คโอเพนซอร์สที่ทำด้านนี้โดยเฉพาะ Chaos Toolkit มีไดรฟ์เวอร์สำหรับ AWS, Azure Service Fabric และ GCE และทำงานได้ดีกับเครื่องมือสร้างแพ็คเกจซอฟต์แวร์ซึ่งทำให้คุณสามารถทดลองทำ automation ได้ แต่มีข้อควรระวังอยู่คือ Chaos Engineering เป็นเทคนิคที่ทรงพลังที่ควรใช้กับระบบถูกสร้างขึ้นโดยคำนึงถึงความคงทนในการฟื้นตัวไว้แล้ว ด้วยเหตุผลนี้เราแนะนำให้ลองใช้ Chaos Toolkit กับ environment ที่ไม่ใช่โปรดักชั่นก่อน

Dask

ประเมิน

Data scientists และวิศวกรส่วนใหญ่จะใช้ไลบรารีอย่างเช่น pandas เพื่อช่วยในการทำงานเกี่ยวกับการวิเคราะห์ข้อมูล ถึงแม้ว่ามันจะทำความเข้าใจได้ง่ายและมีประสิทธิภาพ แต่ไลบรารีเหล่านั้นก็มีข้อจำกัดตรงที่มันทำงานบน CPU เครื่องเดียวและไม่สามารถขยายระบบแบบ horizontal เพื่อรับมือกับข้อมูลขนาดใหญ่ได้ ในทางกลับกัน Dask มี scheduler ที่ขนาดเล็กและประสิทธิภาพสูง รองรับการขยายระบบได้ตั้งแต่เครื่องเดียวไปจนถึงคลัสเตอร์และเนื่องจากมันสามารถทำงาน

ภาษาและเฟรมเวิร์ก

Room ทำให้การใช้ฐานข้อมูลบนแอนดรอยด์เรียบง่ายขึ้น มีส่วนซ้ำน้อยลง และตรงไปตรงมา ด้วยความสามารถในการตรวจสอบความถูกต้องของ SQL ในเวลาคอมไพล์

(Room)

Dask ช่วยให้เราเพิ่มขนาดของงานที่ประมวลผลด้วย Pandas, Scikit-Learn และ Numpy โดยแทบไม่ต้องเขียนโค้ดใหม่ นี่เป็นทางเลือกที่ดีสำหรับ data scientist ที่ต้องการความสามารถในการขยายงานออกในแนวนอนเพื่อให้กับชุดข้อมูลขนาดใหญ่

(Dask)

ภาษาและเฟรมเวิร์ก

fastai เป็นไลบรารีแบบโอเพนซอร์สสำหรับ Python ที่ทำให้การฝึกโมเดล neural network ให้แม่นยำและรวดเร็วได้ง่ายดายขึ้น มันสนับสนุนการทำ computer vision, natural language processing (NLP) และอื่นๆ อีกมากมาย

(fastai)

http4k เป็นชุดเครื่องมือ HTTP ที่ถูกเขียนขึ้นด้วยภาษา Kotlin ล้วน สำหรับใช้เซอร์วิส HTTP มันเรียบง่าย สวยงาม และ ให้ความสำคัญกับการทดสอบได้

(http4k)

กับ NumPy pandas และ Scikit-learn ได้ Dask จึงเป็นอีกสิ่งหนึ่งที่น่าสนใจสำหรับการศึกษาเพิ่มเติม

Embark

ประเมิน

เราเคยแนะนำให้ใช้ Truffle มาก่อนสำหรับการพัฒนาแอพพลิเคชั่นที่ทำงานแบบกระจายศูนย์ (decentralized application - dapp) ตัว Embark เป็นเฟรมเวิร์กอีกตัวหนึ่งที่สามารถทำให้ชีวิตของคุณง่ายขึ้น Embark มาพร้อมกับฟีเจอร์ต่างๆ ได้แก่ การขึ้นโครงสร้าง (scaffolding), การสร้าง (building), การทดสอบ (testing), และ การตรวจสอบแก้ไขข้อบกพร่อง (debugging) แล้วมันสามารถทำงานร่วมกับที่เก็บข้อมูลแบบกระจายศูนย์ (decentralized storage) อย่างเช่น IPFS ได้ Embark มีการกำหนดค่าแบบ declarative ซึ่งอนุญาตให้คุณจัดการสมาร์ทคอนแทรคท์ (smart contract) configuration, dependencies, artifact และ deployment ได้อย่างง่ายดาย อีกทั้งแดชบอร์ดปฏิสัมพันธ์ประเภท CLI ของ Embark ก็น่าประทับใจ เราเห็น Remix ถูกใช้เขียนสมาร์ทคอนแทรคท์ และ ดิพลอยแอปด้วยมือโดยไม่มีส่วนทดสอบแบบอัตโนมัติ, ส่วนควบคุมจัดการซอร์สโค้ด, หรือส่วนจัดการ artifact เราจึงอยากชักชวนผู้คนให้ทำ dapp ด้วยการแนะนำเครื่องมืออย่าง Truffle และ Embark

fastai

ประเมิน

fastai เป็นไลบรารีแบบโอเพนซอร์สสำหรับ Python ที่ทำให้การฝึกโมเดล neural network ให้แม่นยำและรวดเร็วได้ง่ายดายขึ้น มันถูกต่อยอดมากจาก PyTorch และกลายเป็นเครื่องมือที่เป็นนิยมมากในกลุ่ม data scientist มันช่วยทำให้ขั้นตอนที่น่าเบื่อหน่ายอย่างการจัดเตรียมและการนำเข้าข้อมูลลดลงเหลือเพียงโค้ดไม่กี่บรรทัด มันถูกสร้างขึ้นโดยอ้างอิงจากแบบปฏิบัติที่ดีที่สุดของ deep learning อีกทั้งสนับสนุนการทำ computer vision, natural language processing (NLP) และอื่นๆ อีกมากมาย ผู้สร้าง fastai ตั้งใจให้มันเป็นไลบรารีสำหรับ deep-learning ที่ใช้งานง่าย และปรับปรุงให้ดีขึ้นต่อยาก Keras อีกทั้งผู้ให้บริการคลาวด์อย่าง GCP, AWS และ Azure ก็พร้อมใจกันเปิดใจรับ fastai เข้าอยู่ใน image ของ VM อย่างรวดเร็ว ผู้สร้าง fastai ตระหนักถึงข้อจำกัดทางความเร็วและความปลอดภัยของ Python และได้ประกาศความตั้งใจที่จะใช้ภาษา Swift เป็นอีกทางเลือกหนึ่งในการทำ deep learning ทั้งนี้เราจะจับตาดูวงการนี้อย่างใกล้ชิด

http4k

ประเมิน

http4k เป็นชุดเครื่องมือ HTTP ที่ถูกเขียนขึ้นด้วยภาษา Kotlin ล้วน สำหรับใช้เซอร์วิส HTTP แนวคิดหลักของ http4k คือ โมเดลของแอป HTTP ประกอบด้วย 2 ฟังก์ชั่นพื้นฐาน – HttpHandler และ Filter ซึ่งได้รับแรงบันดาลใจจากงานเขียนของ Twitter ที่ชื่อว่า “Your Server as a Function” มันเป็นเครื่องมือที่มีขนาดเล็กที่มี dependency เพียง StdLib ของ Kotlin เท่านั้น นอกจากความเรียบง่ายสวยงามแล้ว เรายังชอบที่มันให้ความสำคัญกับการทำให้ทดสอบได้ (testability) เนื่องจากวัตถุที่ถูกสร้างขึ้นจากไลบรารีนั้นไม่สามารถเปลี่ยนแปลงได้ (immutable entity) และ เส้นทางของแอป รวมถึงตัวแอปเองเป็นเพียงฟังก์ชั่นรูปแบบหนึ่ง มันเลยใช้ทำการทดสอบง่ายมากๆ ซึ่งหนึ่งสิ่งที่ต้องพึ่งระวังคือ nonblocking หรือ coroutine ยังไม่ได้รับการสนับสนุนจาก http4k

Immer

ประเมิน

เนื่องจากแอพพลิเคชั่น JavaScript แบบ single-page มีความซับซ้อนมากขึ้นเรื่อยๆ การจัดการ state predictability จึงมีความสำคัญมากขึ้นเช่นกัน Immutability ช่วยให้แอพพลิเคชั่นมีพฤติกรรมที่คาดเดาได้ แต่โชคร้ายที่ JavaScript ไม่สนับสนุนการสร้าง immutable object ไลบรารีอย่าง Immutable.js ช่วยเติมเต็มช่องว่างตรงนั้นแต่ก็สร้างปัญหาใหม่ขึ้นมา เพราะมันทำให้มี object และ array อย่างละสองแบบในแอพพลิเคชั่นเดียว คือแบบของไลบรารีและแบบพื้นฐานของ JavaScript คำว่า Immer เป็นภาษาเยอรมันแปลว่า เสมอ มันเป็นแพ็คเกจขนาดเล็กที่ให้คุณใช้งาน immutable object ด้วยวิธีที่สะดวกสบายกว่า มันอาศัยกลไก copy-on-write มันมี API ปริมาณน้อย และมันทำงานร่วมกับ object และ array พื้นฐานของ JavaScript เท่ากับว่าเราสามารถเข้าถึงข้อมูลได้อย่างเรียบเนียน และรีเฟคเตอร์ (refactor) สะดวกขึ้นเมื่อต้องการใช้ immutability กับโค้ดเดิมที่มีอยู่แล้ว

Karate

ประเมิน

จากประสบการณ์ของเราที่ว่า การทดสอบ API specification นั่นมีความสำคัญ เราจึงมองหาเครื่องมือใหม่ๆ ที่สามารถช่วยในส่วนนี้ได้อยู่เสมอ Karate เป็นเฟรมเวิร์กสำหรับการทดสอบ API ที่มีฟีเจอร์พิเศษคือคำสั่งทดสอบถูกเขียนใน Gherkin โดยตรงโดยไม่ต้องอาศัยภาษาโปรแกรมอื่นๆ Karate มีภาษาของมันเองสำหรับเขียนการทดสอบ API แบบ HTTP
ถึงแม้แนวทางนี้จะน่าสนใจและตั้งใจทำให้ specification เข้าใจง่าย แต่การใช้ภาษาเฉพาะทางในการตรวจสอบความถูกต้องของ payload นั้น ทำให้ต้องเขียนเยอะและทำให้อ่านทำความเข้าใจยากขึ้น คงต้องดูกันต่อไปว่าสำหรับการทดสอบที่ซับซ้อนแล้ว การเขียนการทดสอบด้วยวิธีนี้จะยังอ่านง่ายและดูแลง่ายอยู่ไหม

Micronaut

ประเมิน

Micronaut เป็นเฟรมเวิร์ก JVM ตัวใหม่สำหรับพัฒนาไมโครเซอร์วิสด้วย Java, Kotlin หรือ Groovy มันมีจุดเด่นคือ มันใช้งานหน่วยความจำน้อยและเริ่มงานเร็วโดยการหลีกเลี่ยงการทำ dependency injection (DI) และ การสร้าง proxy ตอน runtime และใช้ DI/AOP container เพื่อทำ DI ในขั้นตอนคอมไพล์ ด้วยศักยภาพนี้ทำให้มันเป็นที่สนใจสำหรับการทำไมโครเซอร์วิสบนเซิร์ฟเวอร์ และมีประโยชน์กับการใช้งานอื่นๆ ด้วย เช่น การทำ Internet of Things, แอพพลิเคชั่นสำหรับแอนดรอยด์ และ serverless function ตัว Micronaut ใช้ Netty และสนับสนุนการเขียนโปรแกรม reactive เป็นอย่างดี มันยังรวมฟีเจอร์มากมายสำหรับการใช้งานร่วมกับคลาวด์ด้วย เช่น service discovery และ circuit breaking

Micronaut เป็นผู้เล่นใหม่ในวงการเฟรมเวิร์กฟูลสแต็ค (full stack) สำหรับ JVM และเรากำลังจับตาดูมันอยู่อย่างใกล้ชิด

Next.js

ประเมิน

React.js ได้ปฏิวัติวิธีการเขียน single-page แอพพลิเคชั่นของ JavaScript โดยปกติแล้ว เราแนะนำให้ใช้ React ในการสร้างแอพพลิเคชั่น โดยที่คุณไม่ต้องปรับแต่งการติดตั้ง หรือสร้างแพคเกจด้วยตัวเอง แต่นักพัฒนาซอฟแวร์บางส่วนต้องการเครื่องมือที่ ตั้งค่าแรกเริ่มที่สะท้อนให้เห็นถึงความคิดเห็นของพวกเขา Next.js เป็นเฟรมเวิร์กที่ใช้ความเห็นของตนเองเป็นหลักและเป็นการรวบรวมความสนใจระหว่างผู้ที่ชื่นชอบ front-end ของเรา Next.js ลดความยุ่งยากในการทำ routing, เรนเดอร์เพจบนฝั่ง server side และ streamlines dependencies พร้อมทั้งการ build ให้ง่ายขึ้น เรากระตือรือร้นที่จะเห็นว่า Next.js ทำได้ตามความคาดหวังในโปรเจ็คของเรา

Pose

ประเมิน

Pose เป็นไลบรารีแอนิเมชันเรียบง่ายคล้าย CSS สำหรับเฟรมเวิร์ก React.js, React Native และ Vue.js ซึ่งเป็นระบบ declarative motion ที่ผสมผสานความเรียบง่ายของ CSS syntax และความยืดหยุ่นของ JavaScript แอนิเมชันและการปฏิสัมพันธ์กัน

react-testing-library

ประเมิน

เนื่องจากการเปลี่ยนแปลงของเฟรมเวิร์ก JavaScript นั้นเริ่มช้าลง ทีมของพวกเราก็เลยมีเวลาที่ทำงานกับเฟรมเวิร์กบางตัวเป็นพิเศษ และได้ผลลัพธ์เป็นข้อมูลเชิงลึก สำหรับ React และ Enzyme ที่เป็นเฟรมเวิร์กที่ใช้ในการทดสอบ ที่ครองตลาดตอนนี้ เราได้สังเกตเห็นถึงความกังวลต่อ unit test ที่ผูกติดกับรายละเอียดของ implementation มากเกินไป (เนื่องจากมุ่งความสำคัญไปที่การ shallow component เพื่อทดสอบเท่านั้น) นั่นไม่ได้ทำให้เรามั่นใจต่อฟีเจอร์ (features) ที่เราทดสอบ ซึ่ง unit test พวกนี้ทำให้การออกแบบต่อยอดเป็นไปได้ยากขึ้น และทำให้ต้องย้ายความรับผิดชอบในการทดสอบในมุมมองของ test pyramid เลื่อนขึ้นไปยัง functional testing ที่มากขึ้น ด้วยเหตุนี้ทำให้เราย้อนกลับไปยังไอเดียของ subcutaneous testing นอกจากนี้ ปัญหาของ Enzyme คือการออกแบบที่พยายามตามการพัฒนาของ React ให้ทัน จากทั้งหมดนี้ทำเราผลักดัน react-testing-library ในการเป็นเฟรมเวิร์กตัวใหม่สำหรับการทดสอบแอพพลิเคชั่น React

ReasonML

ประเมิน

ReasonML เป็นภาษาที่น่าสนใจตัวใหม่ที่ได้ต้นแบบมาจาก OCaml พร้อมกับ syntax ส่วนนึงที่คล้ายกับภาษา C และคอมไพล์ไปเป็นภาษา JavaScript ReasonML เป็นภาษาที่สร้างโดย Facebook สามารถใส่ javascript snippets ได้ และยังสามารถใช้ jsx ร่วมกับ React ได้อีกด้วย ตัวภาษามุ่งเป้าหมายเพื่อให้สามารถเข้าถึงได้โดยผู้พัฒนา JavaScript และเพื่อยกระดับระบบนิเวศ (ecosystem) ที่มีอยู่ ในขณะที่ตัวภาษาเป็น functional ที่มาพร้อมกับ type safety

Taiko

ประเมิน

Taiko เป็นไลบรารี่ Node.js ซึ่งมี API สำหรับช่วยเหลือการทำ automation บนเว็บเบราว์เซอร์ chrome และ chromium คุณสามารถใช้ประโยชน์จาก selector อัจฉริยะของ Taiko ในการเขียนเทสต์ที่น่าเชื่อถือไปพร้อมๆ กับการพัฒนาโครงสร้างของเว็บแอปพลิเคชันโดยไม่จำเป็นต้องมี selector จำพวก id, CSS หรือ XPath หรือการเพิ่มคำสั่งรอโดยตรง (สำหรับ XHR request) ในเทสต์สคริปต์ Taiko มีเครื่องมือสำหรับบันทึกการโต้ตอบกับ REPL ซึ่งมีประโยชน์เมื่อคุณต้องการพัฒนาการทดสอบควบคู่ไปกับการตรวจฟังก์ชันการทำงานของระบบ และแม้ว่า Taiko นั้นสามารถทำงานได้อย่างอิสระ แต่เราก็มีเรื่องราวดีๆจากการใช้งานร่วมกับ Gauge

Vapor

ประเมิน

เราได้นำเสนอ polyglot programming แต่ตระหนักว่าในบางกรณี ก็เหมาะสมที่จะโฟกัสภาษาโปรแกรมภาษาใดภาษาหนึ่ง ถ้าคุณใช้เวลาส่วนใหญ่ไปกับภาษา Swift เพราะว่าคุณต้องพัฒนาของบน iOS และคุณก็ค้นพบว่าตัวเองกำลังมองหาเทคโนโลยี ไว้สำหรับเขียนเซอร์วิส server-side คุณควรลองดู Vapor เพราะเป็นเว็บเฟรมเวิร์กสำหรับ Swift ที่ได้รับความนิยมอยู่พอสมควร

ภาษาและเฟรมเวิร์ก

Micronaut เป็นเฟรมเวิร์ก JVM ตัวใหม่สำหรับพัฒนาไมโครเซอร์วิสด้วย Java, Kotlin หรือ Groovy ที่ใช้งานหน่วยความจำน้อยและเริ่มงานได้เร็ว

(Micronaut)

Taiko เป็นไลบรารี่ Node.js ซึ่งมี API สำหรับช่วยเหลือการทำ automation บนเว็บเบราว์เซอร์ chrome และ chromium การทดสอบที่เขียนด้วย Taiko มักอ่านและดูแลรักษาง่ายมากๆ

(Taiko)

อยากติดตามข่าวสารและข้อมูลเชิงลึกล่าสุดเกี่ยวกับเรดาร์ใช่ไหม?

ติดตามเราบนช่องทางโซเชียลที่คุณถนัด
หรือ สมัครรับข่าวสารผ่านทางอีเมล์

กดติดตามเลย

ThoughtWorks®

เราคือบริษัทให้คำปรึกษาทางด้านเทคโนโลยี
เราเป็นชุมชนที่เป็นแหล่งรวมของนักสร้างสรรค์
ผู้ที่มีเป้าหมายและความหลงไหลในสิ่งที่ทำเป็นแรงผลักดัน
เราช่วยลูกค้าของเราประยุกต์ใช้เทคโนโลยี
เข้าไปเป็นแกนสำคัญอีกแกนหนึ่งที่ช่วยขับเคลื่อนธุรกิจ
พร้อมทั้งสร้างสรรค์ซอฟต์แวร์ที่มีความสำคัญกับพวกเขา
ไปด้วยกันพวกเราทุ่มเทให้กับภารกิจอันสำคัญของเรา คือ
การใช้ซอฟต์แวร์ผลักดันสังคมและมนุษยชาติ นอกจากนี้
เราร่วมมือกับหลากหลายองค์กรที่มีวัตถุประสงค์เดียวกัน
เพื่อการสร้างความเปลี่ยนแปลงที่ดีให้เกิดขึ้นกับสังคม

กว่า 25 ปี ThoughtWorks ได้เติบโตเป็นบริษัท
ที่มีผู้คนเข้าร่วมด้วยกว่า 6,000 คน และเรายังมี
แผนกที่มีหน้าที่คิดค้นเครื่องมือใหม่ๆ ให้ทีมพัฒนา
ซอฟต์แวร์ของเราใช้โดยเฉพาะอีกด้วย

ThoughtWorks มีสำนักงานทั้งหมด 40 แห่ง กระจายอยู่
14 ประเทศทั่วโลก ได้แก่ ออสเตรเลีย, บราซิล, แคนาดา,
ชิลี, จีน, เอกวาดอร์, เยอรมัน, อินเดีย, อิตาลี, สิงคโปร์,
สเปน, ไทย, สหราชอาณาจักร และ สหรัฐอเมริกา

thoughtworks.com

ThoughtWorks®

thoughtworks.com/radar

#TWTechRadar