Resiliency(ඔරොත්තු දීමේ හැකියාව) in Distributed Systems

What is Distributed Systems
networked components which communicate and coordinate their actions by passing messages.
යනු පොදු ඉලක්කයක් සඳහා පණිවුඩ හුවමාරු ක්‍රමය මගින් ක්‍රියාපන්ති-ප්‍රතිපල සන්නිවේදනය සහ සම්බන්ධීකරණය කරන්නාවූ ජාල පද්ධතියකි.

විශාල ක්‍රියාවලියක් අනුක්‍රියා රාශියකින් සමන්විතය. අනුක්‍රියා පද්ධතියක් ක්‍රියාපන්තියක්(microservice) ලෙස හඳුන්වන අතර ක්‍රියාපන්ති එකතු වීමෙන් විශාල ක්‍රියාවලියක් සිද්ධවේ.

What is Resiliency ?
capacity to recover from difficulties
අනපේක්ෂිත අවස්ථාවකට ඔරොත්තු දීමේ හැකියාව

why care about Resiliency ?
financial losses
losing customers
affection customers
affection livelihood of drivers

Faults vs Failures

Fault is an incorrect internal state in your system.

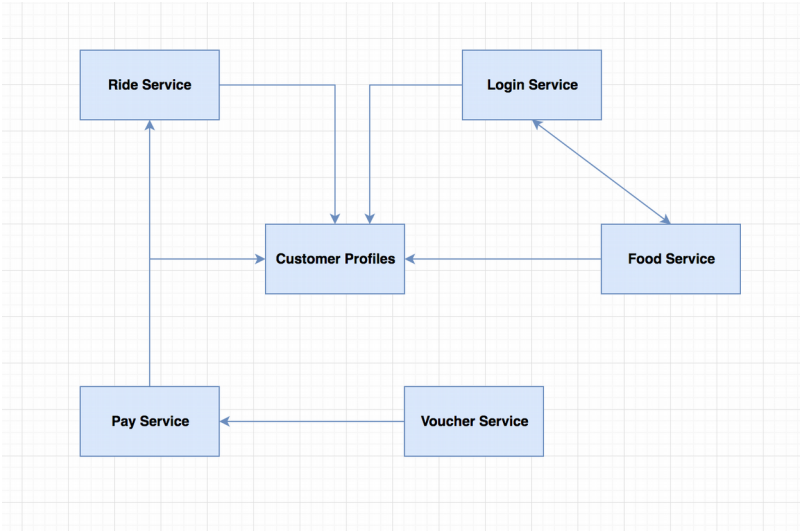
Fault -යනු අභ්‍යන්තර පද්ධතියේ පවතින දෝෂය.

Some common examples of **fault** in systems include:()

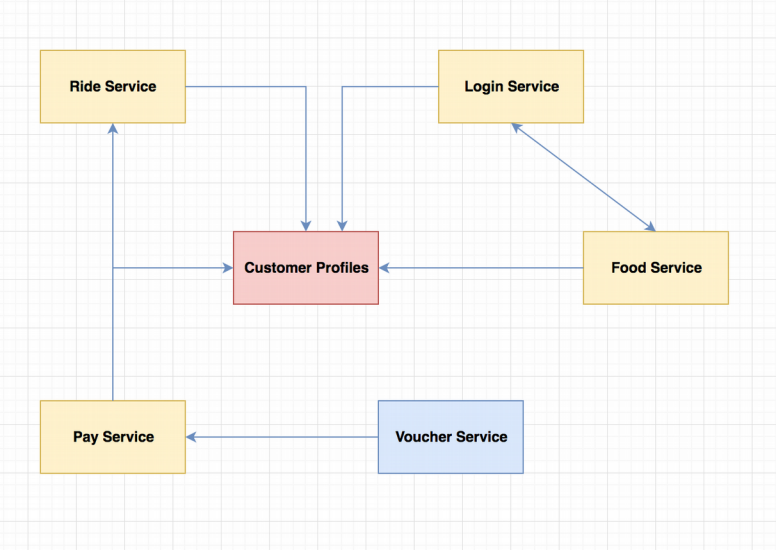
1. Slowing down of storage layer
2. Memory leaks in application
3. Blocked threads
4. Dependency failures
5. Bad data propagating in the system (Most often because there’s not enough validations on input data)

උදාහරණයකට සෞඛ්‍ය සේවා පද්ධතියක Fault වීමේ අවස්ථාවක් සලකා බලමු.

1 රූපය නිරූපනය කරන්නේ පද්ධතිය ක්‍රියාපන්ති සමග සම්බන්ධව ඇති ආකාරයත් ඒවා සාමාන්‍ය පරිදි ක්‍රියා කරන ආකාරයත්ය.

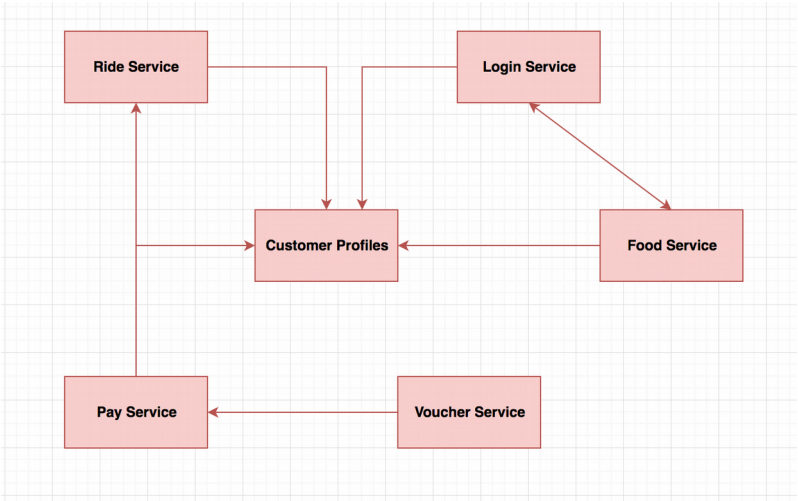


2 රූපය නිරූපනය කරන්නේ පද්ධතියේ ක්‍රියාපන්තියක දෝෂයක්(faults) (ඉහත දක්වා ඇති කරුණු 5 එකක් හෝ කිපයක් හේතුවෙන්) හටගැනීම සහ එය සමස්ත පද්ධතියෙන බලපෑම් කරන ආකාරයයි.

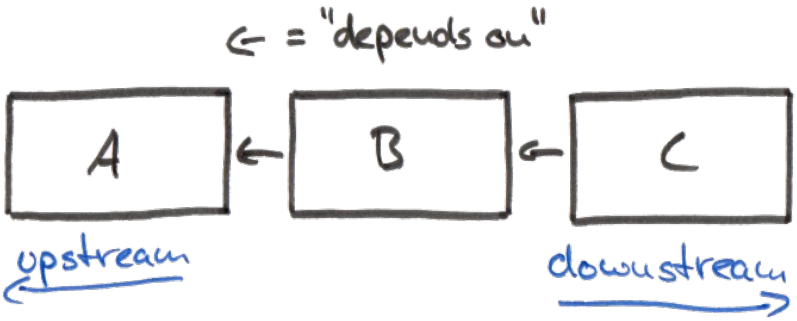


පද්ධතියේ ක්‍රියාපන්තියක හටගත් දෝෂය අනෙකුත් ක්‍රියාපන්ති සමග සුලභව ක්‍රියා කරයි නම් එය සමස්ත පද්ධතියේ දෝෂයක් එනම් පද්ධතිය බිඳවැටීමක්(failures) දුරට පරිවර්තනය විය හැකිය.

3 රූපය නිරූපනය කරන්නේ ක්‍රියා පන්තියක දෝෂය හේතුකොටගෙන පද්ධතිය බිඳවැටීමට භාජනය වීමයි.



මෙහි Resilience යනුවෙන් අදහස් කරන්නේ සිදුවිය හැකි දෝෂ(faults) සමස්ථ පද්ධතියේ බිඳවැටීමක් (failures)දක්වා පරිවර්තනය වීම වලක්වා ගැනීමයි.



Resiliency in distributed systems is hard

දෝෂ සඳහා ඔරොත්තු දෙන එකිනෙකාට සන්නිවේදනය කරගන්නා වූ සංකීර්ණ ක්‍රියාපංති(micro-services) ආකෘතියක් සහිත distributed systems එකක් ගොඩනැගීම පහසු නොවේ. මෙහෙදී බලපාන දුෂ්කරතා කීපයක් පහත දැක්වේ.

1. The network is unreliable (ජාලයෙන් සන්නිවේදන විකුර්ති වීම)

2. Dependencies can always fail (ක්‍රියාපන්ති වල භාවිතා කරඇති Dependencies වල දෝෂ)

3. User behavior is unpredictable (පරිශීලකයාගේ අනපේක්ෂිත හැසිරීම)

දෝෂ සඳහා ඔරොත්තු දෙන distributed systems ගොඩනැගීමේ ඉහත දුෂ්කරතා මගහැරීම සඳහා ඉදිරියේදී දැක්වෙන ක්‍රම(Pattern) භාවිතා කල හැකිය.

Pattern[0] = nocode

The best way to write reliable and secure applications is write no code at all—Write Nothing and Deploy nowhere—Kelsey Hightower.

Pattern[1] = Timeouts
ක්‍රියාපන්තියකින් ප්‍රතිඵලය ලැබෙන තෙක් බල සිටීම යම් කාලයකට පමණක් සීමා කිරීම.
ස්වභාවයෙන්ම client HTTP rq සඳහා කාලය සීමා කිරීමක් නොමැත.එමනිසා පහත රූපයේ පරිදි යම් ක්‍රියාපන්තියක දෝෂයක් ඇති වුවහොත් A විසින් යවන ලද HTTP rq එකෙහි ප්‍රතිඵලය ලැබෙන තෙක් A ට වගකියන සියලු ක්‍රියාපන්ති(downstream services) ක්‍රියා කිරීම නැවතිමයි. උදාහරණයක් ලෙස 1ms තුල rq ප්‍රතිඵලය නොලැබුන හොත් එය fail එකක් ලෙස downstream services එක දැනගෙන ඉදිරියට ක්‍රියා කිරීම.

Preventing cascading failures.
Timeouts කිරීම මගින් දෝෂය අනෙකුත් කොටස් වලට කරන පැතිරයාමෙන් ඇතිවන බලපෑම අවම කල හැක.

Providing failure isolation (දෝෂය හුදෙකලා කිරීම)
Timeouts කර දෝෂය හුදෙකලා කිරීම මගින් දෝෂය පද්ධතියේ යම් කොටසකට පමණක් සීමා කිරීම.

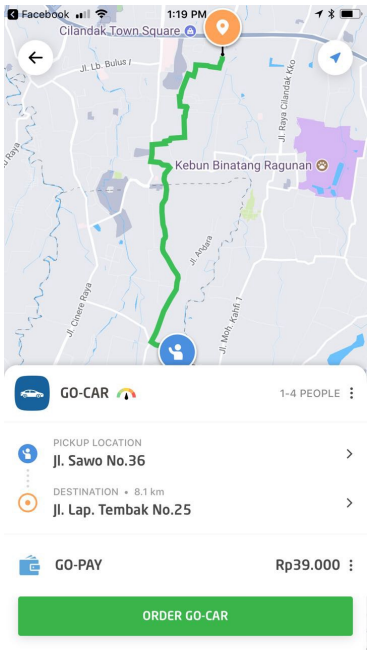
Pattern[2] = Retries
If you fail once, try again

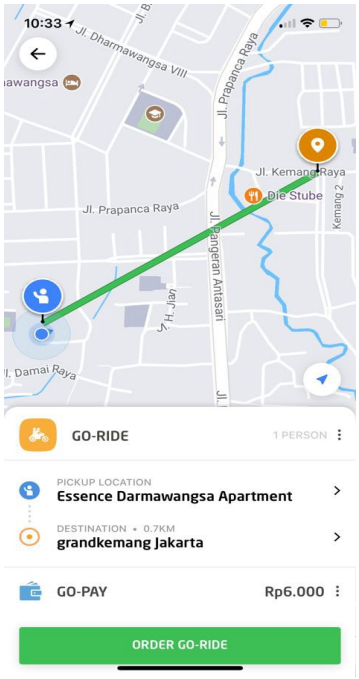
ප්‍රති rq එක නොලැබෙන විට එය ලැබෙන තෙක් තවදුරටත් බලා නොසිට නැවත rq එකක් යැවීම.
මෙය වඩා ප්‍රයෝජනවත් වන්නේ intermittent failures අවස්ථා වලදීය.
intermittent failures යනු පද්ධතියක් හෝ උපකරණයක් ක්‍රමවත්ව ක්‍රියා නොකරන(malfunction) අවස්ථා වලදී එය විරාම ගැනීම ආදී කටයුතු නිසි අවස්ථාවේදී ප්‍රතිචාර නොලැබීමයි.
Retrying immediately might not always be useful
Timeouts එකක් සමග Retries කිරීම වැඩ ප්‍රයෝජනවත් වේ. මන්ද යත් යම් ක්‍රියා පන්තියක් නැවත යථාතත්වයට පත්වීමට යම් කාලයක් ගතවන බැවිනි.

Pattern[3] = Fallbacks

Degrade gracefully

අපගේ පද්ධතියේ දෝෂයක් ඇතිවිට එය හඳුනාගෙන ඊට අදාල rs එකක් යැවීමට කටයුතු කිරීමෙන් සම්පූර්ණ පද්ධතිය ඇනහිටීම වලක්වාගත හැක.





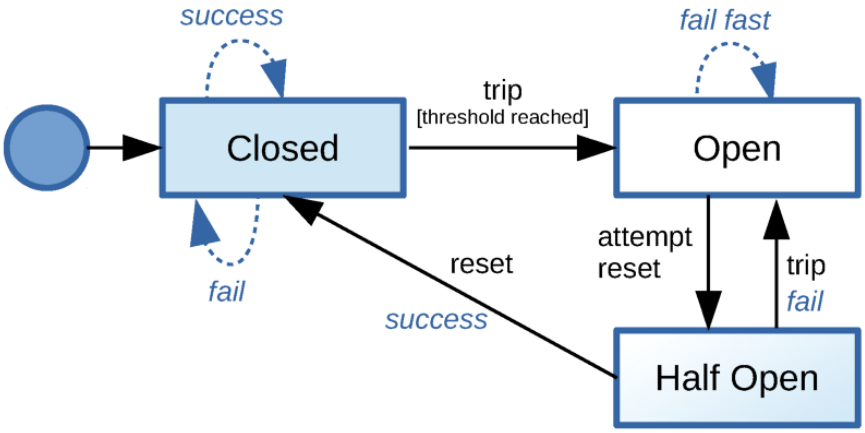
උදාහරණයකට google සිතියමේ api එකෙන් මාර්ගසටහන ලබාගන්නා අවස්ථාවක් සලකා බලමු. අප යවන ස්ථාන වල බණ්ඩාංක වලට යා හැකි ආසන්නතම මාර්ගය බලාපොරොත්තුවෙන් සිටින අවස්ථාවක එම දත්තය පමා වූ හෝ නොලැබෙන අවස්ථාවකදී දෙවැනි රූපසටහනේ පරිදි එම බණ්ඩාංක දෙක සරල රේඛාවකින් යා කර පද්ධතිය නොබිඳී පවත්වා ගත හැකිය.

එකිනෙකට රදාපවතින ක්‍රියාපන්ති අතර fallback යෙදීම යෝග්‍ය වේ.

Pattern [4] = Circuit Breakers

Trip the circuit to protect your dependencies

මෙහිදී කරනුයේ යවන ලද rq කිපයකටම යම් ක්‍රියාපන්තියකින් ප්‍රතිචාරයක් නොදක්වයි නම් එම ක්‍රියාපංතිය හි දෝෂයක් ඇතිබව හඳුනාගෙන තවදුරටත් rq යැවීම වලකා එම ක්‍රියාපංතිය නිවැරදි කිරීමට(recover) අවස්ථාව ලබාදීමයි.



ඉහත රූපයේ දක්වා ඇත්තේ circuit breaker(CB) පරිපථයකි.

ක්‍රියාපංතියේ දෝෂ නොමැති අවස්ථාවේදී CB closed අවස්ථාවේ පවතින අතර ක්‍රියාපන්ති සමග සන්නිවේදනය පවත්වාගනී. යවන ලද rq කීපයකටම ක්‍රියාපන්තිය ප්‍රතිචාරයක් නොදක්වයි නම් CB open අවස්ථාවට පැමිණෙන අතර නව requests දෝෂ සහිත downstream service යොමු කිරීම නවතයි. දෝෂ සහිත ක්‍රියාපන්තියේ නිවැරදි කිරීම් (sleep threshold) කීපයකට පසුව CB half open අවස්ථාවට පත් වේ. පසුව යනවා requests එක සාර්ථක වුවහොත් නැවත CB closed අවස්ථාවට පැමිණ සාමාන්‍ය ආකාරයට ක්‍රියා කිරීමට පටන් ගනී.

Pattern[5] = Resiliency Testing

Test to Break

මෙහිදී කරනුයේ එක එක් ක්‍රියාපන්ති වල දෝෂ ඇද්දැයි වෙන වෙනම පරීක්ෂා කර බැලීමයි.