

Why automating everything adds to your toil

Colin Thorne, SRE, IBM Kubernetes Service

Cam McAllister, SRE, IBM Kubernetes Service

usenix
**SRE
CON** EUROPE
MIDDLE EAST
AFRICA

DUBLIN, IRELAND

2-4 October, 2019

www.usenix.org/srecon19emea



IBM Cloud

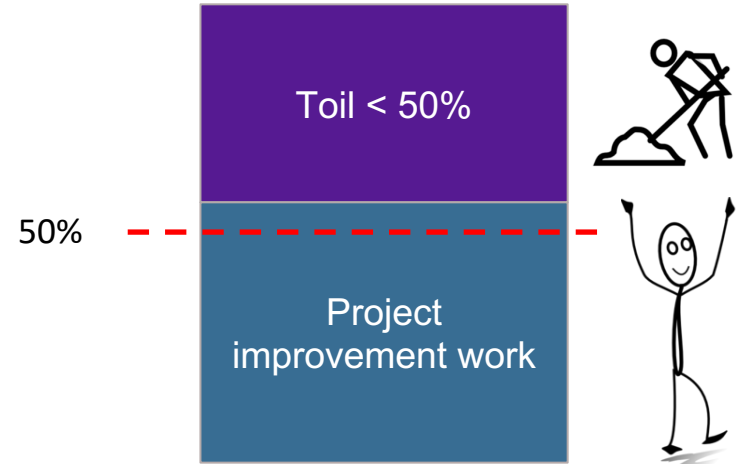
“Just automate it”

What is toil?

“Toil is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows” [Vivek Rau, SRE Book]

What is toil?

- **Gets in the way of making progress**
- **Repetitive manual tasks**
 - Incidents, Tickets
 - Watching dashboards, producing spreadsheets
- **The key is to reduce the amount of toil**
- **Project improvement work adds features or reduce future toil**



What is automation?

- Avoid manual tasks by getting computers to do it for us
- Computers don't get bored and love repetitive tasks
- Humans make mistakes



- Chatbots for ops
- Self healing
- Provisioning/deploying
- Self service



Colin Thorne (IKS SRE) 23:26

10.113.51.224



netmax APP 23:26

@colin:

prod-lon02-infra-vpn-03 (Bare Metal Server) - London 2:01 - Acct531277

Public: 159.122.193.10 159.122.193.0/27 (Prod/Lon02/VPN/F) (1672 (fcr01a.lon02))

Private: 10.113.51.224 10.113.51.192/26 (Prod/Lon02/VPN/B) (1875 (bcr01a.lon02))

Management: 10.113.51.225 10.113.51.192/26 (Prod/Lon02/VPN/B) (1875 (bcr01a.lon02))

Comment: 10.113.51.225

OS: UBUNTU_18_64

Tags: conductors-openvpn

Why automation adds to toil

In the beginning ... automation reduced toil

Reflections → Identified toil → New automation → Happy SRE



Why automation adds to toil

But then automation started to add to our pile of toil

“I tried to use it but the bot doesn't work anymore”

“The automation stopped working ... about a month ago”

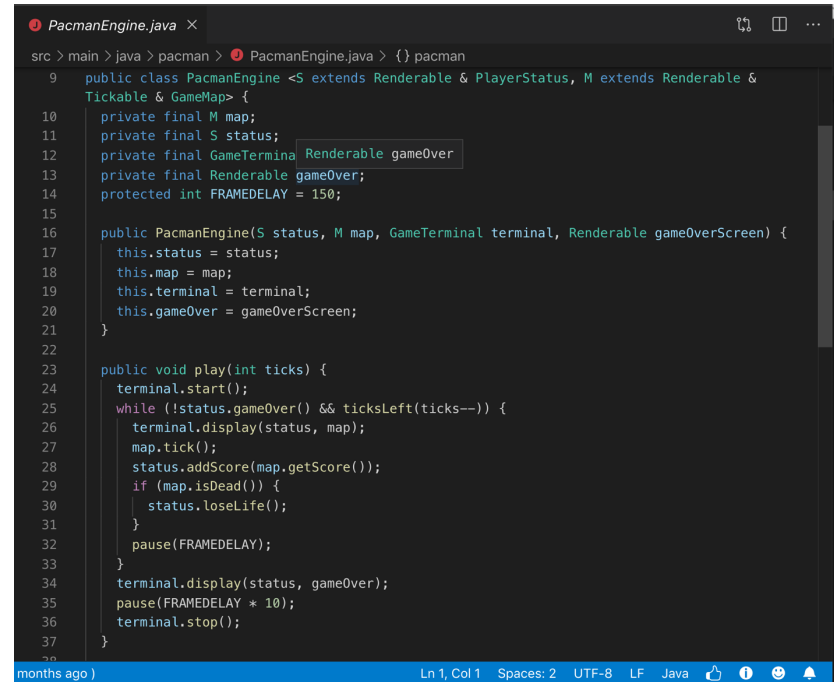
“Oh yes, we changed the api and the automation hasn't caught up”



Rot has set in

Automation rots over time

Just like any code, automation needs constant care and feeding

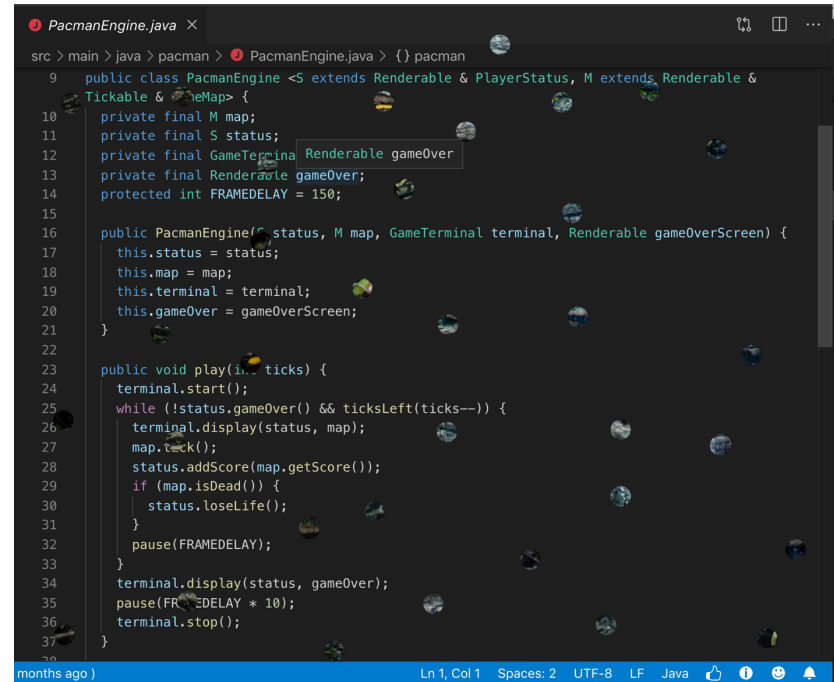


```
PacmanEngine.java
src > main > java > pacman > PacmanEngine.java > {} pacman
9 public class PacmanEngine <S extends Renderable & PlayerStatus, M extends Renderable &
  Tickable & GameMap> {
10     private final M map;
11     private final S status;
12     private final GameTerminal Renderable gameOver
13     private final Renderable gameOver;
14     protected int FRAMEDELAY = 150;
15
16     public PacmanEngine(S status, M map, GameTerminal terminal, Renderable gameOverScreen) {
17         this.status = status;
18         this.map = map;
19         this.terminal = terminal;
20         this.gameOver = gameOverScreen;
21     }
22
23     public void play(int ticks) {
24         terminal.start();
25         while (!status.gameOver() && ticksLeft(ticks--)) {
26             terminal.display(status, map);
27             map.tick();
28             status.addScore(map.getScore());
29             if (map.isDead()) {
30                 status.loseLife();
31             }
32             pause(FRAMEDELAY);
33         }
34         terminal.display(status, gameOver);
35         pause(FRAMEDELAY * 10);
36         terminal.stop();
37     }
38 }
```


Automation rots over time

Just like any code, automation needs constant care and feeding

- Dependencies change



The image shows a code editor window titled 'PacmanEngine.java' with a dark theme. The code is for a Pacman game engine. The background of the editor is a Pacman game visualization with a yellow Pacman character, several blue ghosts, and a maze. The code includes class declarations, private fields for map, status, game terminal, and game over screen, and methods for initialization and game logic.

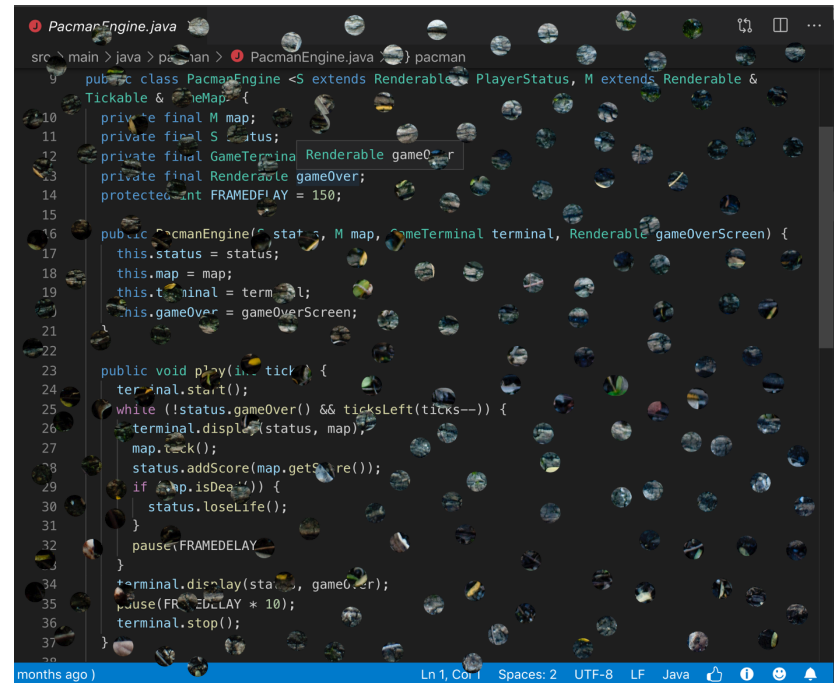
```
src > main > java > pacman > PacmanEngine.java > {} pacman
9 public class PacmanEngine <S extends Renderable & PlayerStatus, M extends Renderable &
  Tickable & GameMap> {
10     private final M map;
11     private final S status;
12     private final GameTerminal Renderable gameOver
13     private final Renderable gameOverScreen;
14     protected int FRAMEDELAY = 150;
15
16     public PacmanEngine(S status, M map, GameTerminal terminal, Renderable gameOverScreen) {
17         this.status = status;
18         this.map = map;
19         this.terminal = terminal;
20         this.gameOver = gameOverScreen;
21     }
22
23     public void play(int ticks) {
24         terminal.start();
25         while (!status.gameOver() && ticksLeft(ticks--)) {
26             terminal.display(status, map);
27             map.tick();
28             status.addScore(map.getScore());
29             if (map.isDead()) {
30                 status.loseLife();
31             }
32             pause(FRAMEDELAY);
33         }
34         terminal.display(status, gameOver);
35         pause(FRAMEDELAY * 10);
36         terminal.stop();
37     }
38 }
```

months ago | Ln 1, Col 1 | Spaces: 2 | UTF-8 | LF | Java | [Icons]

Automation rots over time

Just like any code, automation needs constant care and feeding

- Dependencies change
- Requirements change



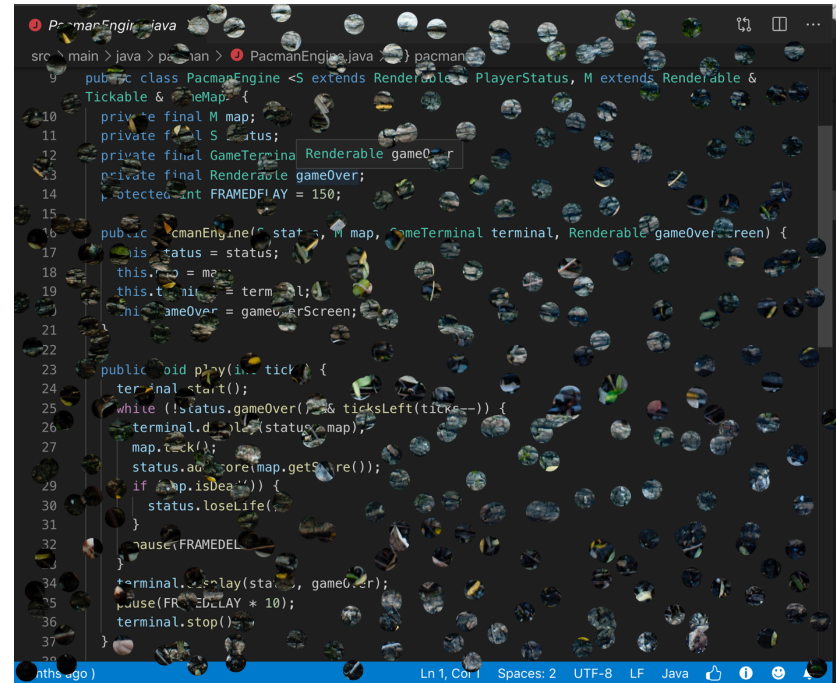
The image shows a code editor window titled 'PacmanEngine.java'. The code is in Java and defines a 'PacmanEngine' class. The code is partially obscured by a Pacman game overlay. The code includes class declarations, private fields for 'map', 'status', 'gameOver', and 'gameOverScreen', a constant 'FRAMEDELAY', and methods for 'play' and 'pause'. The game overlay shows a Pacman character on a map with various obstacles and power pellets.

```
src \main > java > pacman > PacmanEngine.java > pacman
9 public class PacmanEngine <S extends Renderable, M extends Renderable &
  Tickable & GameMap> {
10     private final M map;
11     private final S status;
12     private final GameTerminal Renderable gameOver;
13     private final Renderable gameOverScreen;
14     protected int FRAMEDELAY = 150;
15
16     public PacmanEngine(S status, M map, GameTerminal terminal, Renderable gameOverScreen) {
17         this.status = status;
18         this.map = map;
19         this.terminal = terminal;
20         this.gameOver = gameOverScreen;
21
22
23     public void play(int ticks) {
24         terminal.start();
25         while (!status.gameOver() && ticksLeft(ticks--)) {
26             terminal.display(status, map);
27             map.tick();
28             status.addScore(map.getScore());
29             if (map.isDead()) {
30                 status.loseLife();
31             }
32             pause(FRAMEDELAY);
33         }
34         terminal.display(status, gameOverScreen);
35         pause(FRAMEDELAY * 10);
36         terminal.stop();
37     }
38 }
```

Automation rots over time

Just like any code, automation needs constant care and feeding

- Dependencies change
- Requirements change
- SREs change



The image shows a terminal window with a Pacman game running in the background. The game is a classic Pacman-style game with a black background, a maze, and various colored dots. The terminal window is titled "PacmanEngine.java" and shows the following code:

```
src \main > java > pacman > PacmanEngine.java > pacman
9 public class PacmanEngine <S extends Renderable, P extends PlayerStatus, M extends Renderable &
  Tickable & GameMap> {
10     private final M map;
11     private final S status;
12     private final GameTerminal gameOver;
13     private final Renderable gameOverScreen;
14     protected int FRAMEDELAY = 150;
15
16     public PacmanEngine(S status, M map, GameTerminal terminal, Renderable gameOverScreen) {
17         this.status = status;
18         this.map = map;
19         this.terminal = terminal;
20         this.gameOver = gameOverScreen;
21
22
23     public void play(int ticks) {
24         terminal.start();
25         while (!status.gameOver() && ticksLeft(ticks--)) {
26             terminal.draw(status, map);
27             map.tick();
28             status.advance(map.getScore());
29             if (map.isDead()) {
30                 status.loseLife();
31             }
32             pause(FRAMEDELAY);
33         }
34         terminal.draw(status, gameOverScreen);
35         pause(FRAMEDELAY * 10);
36         terminal.stop();
37     }
38 }
```

Automation rots over time

Just like any code, automation needs constant care and feeding

- Dependencies change
- Requirements change
- SREs change
- Production systems change



```
PacmanEngine.java
src \main > java > pacman > PacmanEngine.java } pacman
public class PacmanEngine <S extends Renderable, PlayerStatus, M extends Renderable &
Tickable & TMap {
    private final M map;
    private final S status;
    private final GameTerminal renderable gameOver;
    private final Renderable gameOver;
    protected int FRAMEDLAY = 150;

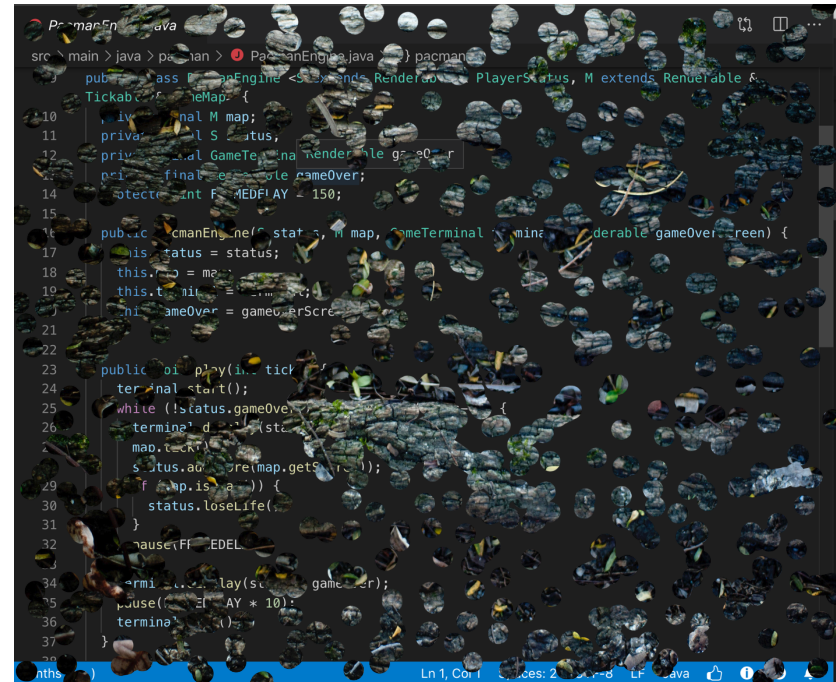
    public PacmanEngine(S status, M map, GameTerminal renderable, Renderable gameOver, Screen) {
        this.status = status;
        this.map = map;
        this.renderable = renderable;
        this.gameOver = gameOver;
    }

    public void play(int ticks) {
        terminal.start();
        while (!status.gameOver) {
            terminal.render(status, map);
            map.tick();
            status.action(map.getScore());
            if (map.isGameOver()) {
                status.loseLife();
            }
            pause(FRAMEDLAY);
        }
        terminal.play(status, gameOver);
        pause(FRAMEDLAY * 10);
        terminal.render();
    }
}
```

Automation rots over time

Just like any code, automation needs constant care and feeding

- Dependencies change
- Requirements change
- SREs change
- Production systems change
- Languages change
(Python2 -> Python 3 anyone?)



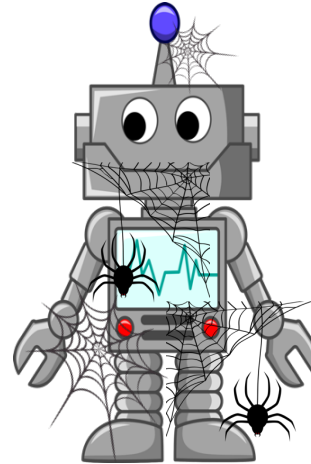
```
src \main > java > pacman > PacmanEngine.java } pacman
public class PacmanEngine <S, E> extends Renderable &
Tickable {
    private final M map;
    private final S status;
    private final GameTerminal gameTerminal;
    private final boolean gameOver;
    private final int FLY_SPEED = 150;

    public PacmanEngine(S status, M map, GameTerminal gameTerminal, boolean gameOver, Screen screen) {
        this.status = status;
        this.map = map;
        this.gameTerminal = gameTerminal;
        this.gameOver = gameOver;
    }

    public void play(int tick) {
        terminal.start();
        while (!status.gameOver) {
            terminal.draw(status);
            map.tick();
            status.action(map.getScreen());
            if (map.isGameOver()) {
                status.loseLife();
            }
            pause(FLY_SPEED);
        }
        terminal.play(status.gameOver);
        pause(FLY_SPEED * 10);
        terminal.reset();
    }
}
```

Unused automation

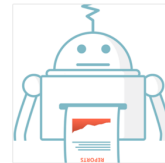
- Automation written once by one team, but no one uses it
- Not publicized
- So we've spent effort to create it, but no one (or very few) people use it
- Yet we still maintain it



Duplicate automation

- How many different bots can you have to do the same thing?
- Not invented here leads to duplicate automation

“Of course that other team’s automation is good, but it doesn’t quite fit what I need. I’ll write my own”



Adds to system complexity

- How much can SREs keep in their brain?



- When automation doesn't work, where to start looking?

“Ironically, although intended to relieve SREs of work, automation adds to systems’ complexity and can easily make that work even more difficult” [Seeking SRE, John Allspaw and Richard Cook]

Too many tools

- **The more tools you have, the more you have to maintain**
- **Some tools many not get used for weeks or months**
- **Danger is that when you come to use a tool it doesn't work**
 - more wasted time either fixing or falling back to the manual way
- **Not always obvious which tool should be used to solve a problem**





Automation is Good!





Automation is Good!



Toil is Bad

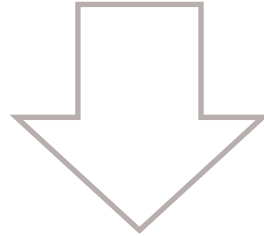




Automation is Good!



Toil is Bad



Reduce the toil caused by automation

Minimising Rot Potential



IBM Cloud

Build as a Developer

Automation is development, so treat it as such

- **Designed**
- **Clean architecture**
- **Properly tested**
- **Full managed lifecycle**

- **Deployed to a production system (not just running on my laptop)**

- **Properly maintained (yuck.. Toil)**

Build Self-Service tools

- **We don't need Full autonomy**
- **Create self-service tools that anyone can use**
 - people AND services (microservice architecture)
- **Call other self-service tools provided by developers**
 - Tied to product function & new changes – preventing rot

Igor

- **Faithful assistant (see Discworld / Pratchett)**
- **Self service bot**
 - Used by SRE and Developers over Slack
- **Doer not a thinker**



Igor bot (SRE) APP 10:34 AM

Lightning master! we need more lightning!

"If all you have is a hammer, everything looks like a nail"

- **We want our tools to be used a lot**
 - Design our hammer so that many errors behave like nails.

How big a hammer?



- Restart the container
- Reboot the machine
- Reload the operating system and reinstall it
- Re-provision the machine
- ?
- ?
- ?
- Delete the universe and recreate it

Oh look.. a nail!



Cameron McAllister (IKS SRE) ♥ SRE 5:39 PM

reload 10.171.78.70 outage:0s



Igorina bot (SRE) APP 5:39 PM

Raising a new change request.

Change request raised: `CHG0376706` waiting for approval

I am starting the OS reload of `prod-dal10-carrier2-worker-1035` (10.171.78.70) now.

Maximising Usage

To minimize Rot



IBM Cloud

Usage Prevents Rot

- **Place it somewhere prominent**
 - Eg in a slackbot
 - Don't hide it
- **Make it as easy to use as breathing**
 - Minimise barrier to entry

Usage Prevents Rot

- **Promote the Automation**

- Playbacks & Education
- Runbooks should point to the automation first

- **Success is people raising Issues on your automation (as long as you fix them)**

Dealing with Rot

You cannot avoid it forever



IBM Cloud

Minimise Effort invested

- **Have a very strict MVP to see if it gets used**
- **Don't deal with all corner conditions**
- **Defer to SRE if something unexpected happens**

Survival of the Fittest

- **Don't assume that the current approach is the best one**
- **Encourage Innovation**
 - Can we do this better a different way?
- **Compare effort to fix rot with benefit of the tool running**

When Rot has set in...

... It's often most humane to put it down.



Cameron McAllister (IKS SRE) 12:18 PM

quiesce TERMINATE



Igor bot (SRE) APP 12:18 PM

Prod: But i served you so well.. I hope my replacement displeases you less



- **Self service bot**

- Used by SRE and Developers over Slack
- Doer not a thinker

- **Enhancements**

- Used by other microservices over a REST API (for self healing)
- Auditable – via change management tooling
- Sharded

In Conclusion



IBM Cloud

Summary

- Automation Good
- Toil Bad
- Reduce toil produced by automation
 - Build as a developer
 - Maximise use of your automation
 - Treat your automation as evolutionary steps

Contacts

colin_thorne@uk.ibm.com

 [linkedin.com/in/cjthorne](https://www.linkedin.com/in/cjthorne) @ColinJThorne

C.McAllister@uk.ibm.com

 [linkedin.com/in/cam-mcallister](https://www.linkedin.com/in/cam-mcallister)

Questions?



IBM Cloud