Site Reliability Engineering: An Enterprise Adoption Story

Perry Statham
SRE Squad Leader, Bluemix DevOps Services

Ritchie Schacher
STSM, SRE Architect, Bluemix DevOps Services

Introductions

Perry Statham
SRE Squad Leader
Bluemix DevOps Services

Perry Statham leads the SRE squad for the IBM® Bluemix® Continuous Delivery offering. Since joining IBM in 1998, he has been principle developer and team leader of several application management offerings; deployment architect of the Jazz for Service Management tool set; lead deployment outside-in designer; and technical leader of an IaaS environment hosting more than 2000 virtual and bare metal machines.

Prior to IBM, Perry worked for many years with a healthcare IT startup company where he did both development and operations long before it became known as DevOps.

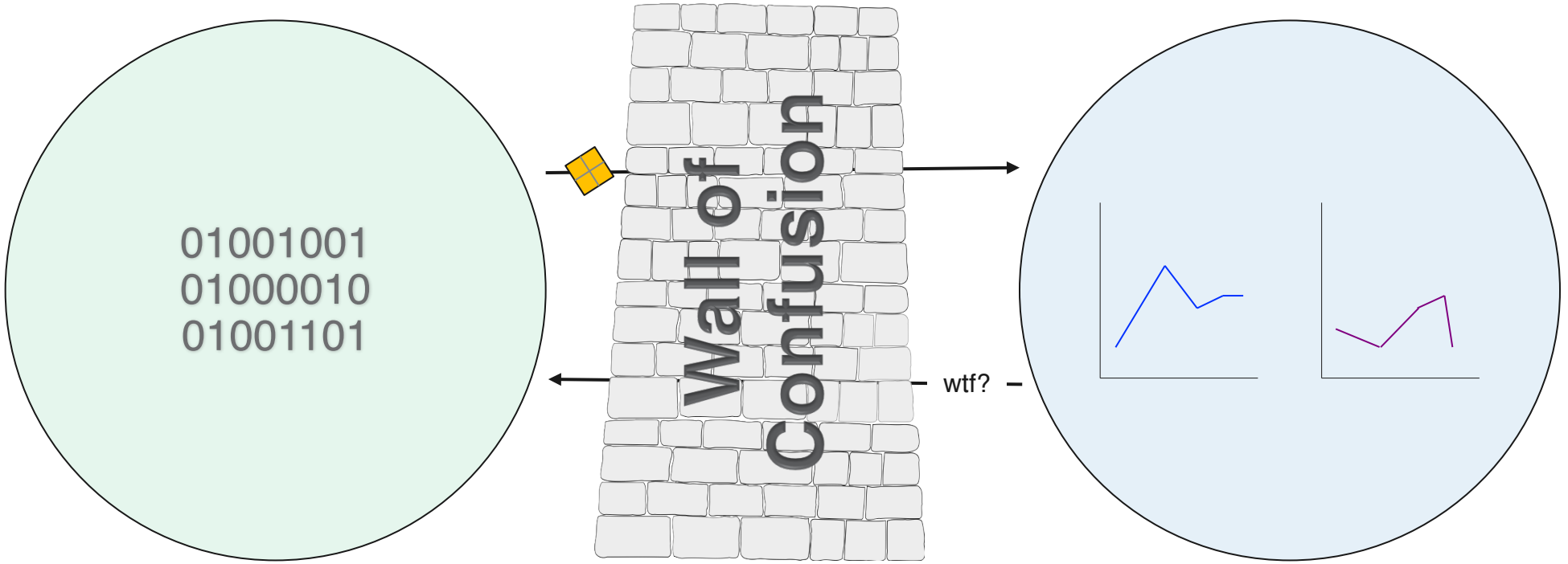
Ritchie Schacher
STSM SRE, Architect
Bluemix DevOps Services

Ritchie Schacher is a Senior Technical Staff Member with IBM® Bluemix® DevOps Services, and lead architect for SRE. He has a background in developer tools and team collaboration. He has broad experience in all aspects of the software development lifecycle, including product conception, planning and managing, design, coding, testing, deploying, hosting, releasing, and supporting products.

For the last 3 years he has been a part of an exciting team developing and supporting cloud-based SaaS offerings for Bluemix®. In his spare time, Ritchie enjoys playing classical guitar.

Once upon a time…

…there were two teams in a large company.

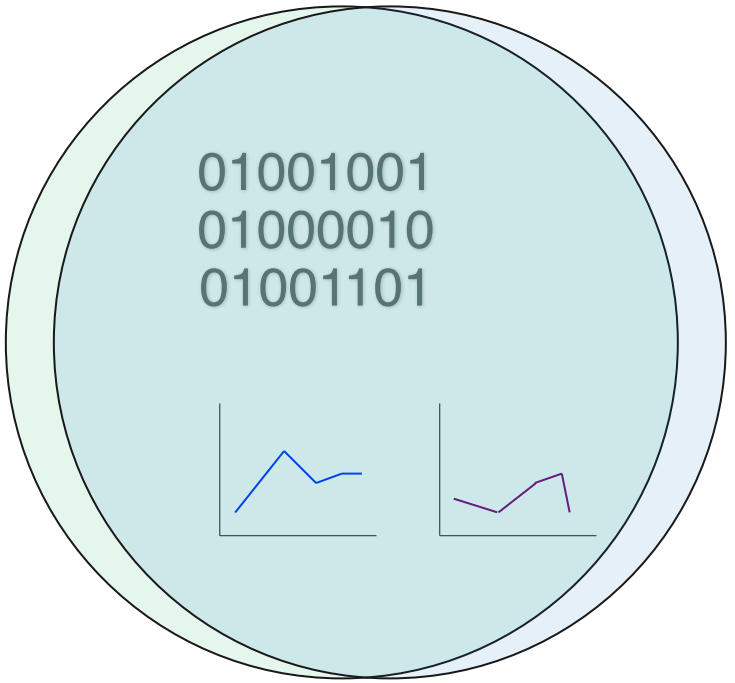


One of the teams developed software products.

The other team operated hosted instances of those products.

Then along came DevOps. Yay!

Build it. Deploy it. Run it. Manage it.



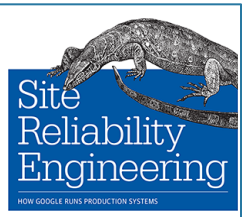
It was a huge improvement, but…

- How do we balance change velocity vs. availability, reliability, security, and other operational attributes? For example:
  - Do we fix a bug in the UI, or do we fix a bug that intermittently requires the backend app to be restarted?
  - Do we add a great new feature, or do we automate backups?
  - Do we improve an app's performance, or do we reduce its mean time to recovery (MTTR)?
  - Do we improve deployment automation, or do we improve standards compliance?
- When each service squad has it's own DevOps process:
  - How do we encourage process consistency and discipline?
  - How do we scale specialized reliability engineering skills?
  - How do we avoid incident response burnout?

Site Reliability Engineering

Note: With some our own additions and interpretations, this section largely comes from the Google SRE Book [Book]. All quotes without a reference are from the same book.

Any misunderstandings are ours and not the fault of the book authors.



- "One could … view SRE as a specific implementation of DevOps with some idiosyncratic extensions."
- "In general, an SRE team is responsible for the *availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning* of their service(s)."

> *"What is Site Reliability Engineering as it has become defined at Google? My explanation is simple: SRE is what happens when you ask a software engineer to design an operations team." – Benjamin Traynor Sloss, Vice President, Engineering, Google*

SRE Principles and Tenets

- Know the Service Level
- Embrace Risk
- Eliminate Toil
- Know What's Broken and Why
- Stuff Happens
- Automate [Almost] Everything
- Reliable Releases
- Keep it Simple

Know the Service Level

- Indicators (SLI): a carefully defined quantitative measure of some aspect of the level of service that is provided
  - Examples include availability, request latency, error rate, and system throughput.
  - Be wary of using the mean (average) of a set of metrics collected over some period of time, since outliers are often hidden.
  - Percentiles usually provide much better metric aggregations.
  - No more than a handful. Not every metric makes a good SLI.
- Objectives (SLO): a target value or range of values for a service level that is measured by an SLI
- Agreements (SLA): an explicit or implicit **contract** with your users that **includes consequences** of meeting (or missing) the SLOs they contain

Embrace Risk

- Goal is to explicitly align the risk taken by a given service with the risk the business is willing to bear.
  - Strive to make a service reliable enough, but no more reliable than it needs to be.
- Which availability metric?
  - Time Based: Uptime / (Uptime + Downtime)
  - Request Based: Successful Requests / (Total Requests)
  - Something else?

Embrace Risk: Error Budgets

- **Error Budgets** balance the goals of
  - Product development teams, which are largely evaluated on product velocity, which creates an incentive to push new code as quickly as possible; and
  - SRE teams, which are evaluated based upon reliability of a service, which implies an incentive to push back against a high rate of change
- A service's error budget is 100% minus the SLO (e.g.100 - 99.95% = .05%)
  - In general, for any software service or system, 100% is not the right reliability target because no user can tell the difference between a system being 100% available and 99.999% available.
    - SRE's goal is no longer "zero outages"; rather, SREs and product developers aim to spend the error budget getting maximum feature velocity.
    - An outage is no longer a "bad" thing – it is an expected part of the process of innovation, and an occurrence that both development and SRE teams manage rather than fear.
- Whenever a service exceeds their error budget, ALL WORK MUST be related to improving availability.

Embrace Risk: Error Budgets

“An error budget stems from this basic observation: 100% is the wrong reliability target for basically everything. The business or the product must establish what the availability target is for the system. If the service natively sits there and throws errors, you know, .01% of the time, you're blowing your entire unavailability budget on something that gets you nothing. So you have an incentive in both the development world and the SRE team to improve the service's native stability so that you'll have budget left to spend on things you do want, like feature launches.” – Ben Traynor [Tra17]

“The other crucial advantage of this is that SRE no longer has to apply any judgment about what the development team is doing. SRE measures and enforces, but we do not assess or judge. Our take is "As long as your availability as we measure it is above your Service Level Objective (SLO), you're clearly doing a good job. You're making accurate decisions about how risky something is, how many experiments you should run, and so on. So knock yourselves out and launch whatever you want. We're not going to interfere." And this continues until you blow the budget.”
– Ben Traynor [Tra17]

Eliminate Toil

"If a human operator needs to touch your system during normal operations, you have a bug. The definition of normal changes as your systems grow. - Carla Geisser, Google SRE

- In SRE, we want to spend time on long-term engineering project work instead of operational work.
  - Because the term operational work maybe misinterpreted, we use a specific word: **toil**.
  - SREs should spend less than X% of their time on toil and the rest on coding
    - Excess toil is redirected to development team
- Toil is mundane, repetitive operational work providing no enduring value, which scales linearly with service growth.
- The work of reducing toil and scaling up services is the "Engineering" in Site Reliability Engineering

Know What's Wrong and Why

> *"…a system that requires a human to read an email and decide whether or not some type of action needs to be taken in response is fundamentally flawed. Monitoring should never require a human to interpret any part of the alerting domain. Instead, software should do the interpreting, and humans should be notified only when they need to take action."*

- If you can't monitor a service, then you don't know what's happening, and if you're blind to what's happening, then you can't be reliable.
- Monitoring a complex application is a significant engineering endeavor in and of itself.
- Monitoring should answer the questions:
  - What's wrong?
  - Why is it wrong?
- Use the right output:
  - **Alerts**: Signify that a human needs to take action immediately in response to something that is either happening or about to happen.
  - **Tickets**: Signify that a human needs to take action, but not immediately.
  - **Logging**: No one needs to look at this information, but it is recorded for diagnostic or forensic purposes.
- Metrics that are useful for overall system operation, but do not influence major user interactions, are not usually SLIs.
  - For example, disk space metrics are not usually web UI application indicators.

Stuff Happens, So Reduce Repair Time

- ***Reliability*** is a function of *mean time to failure (MTTF)* and *mean time to repair (MTTR)* [Sch15]
  - The most relevant metric in evaluating the effectiveness of emergency response is how quickly the response team can bring the system back to health – that is, the MTTR.
- Humans add latency
  - Even if a given system experiences more actual failures, a system that can avoid emergencies that require human intervention will have higher availability than a system that requires hands-on intervention.

Automate [Almost] Everything

> *"If we are engineering processes and solutions that are not automatable, we continue having to staff humans to maintain the system. If we have to staff humans to do the work, we are feeding the machines with the blood, sweat, and tears of human beings. Think The Matrix with less special effects and more pissed off System Administrators." - Joseph Bironas, Google SRE*

- Automation provides
  - Consistency as systems scale
  - A platform for extending to other systems
  - Faster repairs for common problems
  - Faster action than humans
  - Time savings by decoupling operator from operation
- But it's not a panacea
  - It can hide systemic problems.
    - For example, auto-restarting a process periodically can hide memory leaks
  - It can perform the wrong, or even a damaging, operation because of a poor design or implementation
    - Remember the old adage "Garbage-In, Garbage-Out"

Reliable Releases

- Running reliable services requires reliable release processes.
- Continuously build and deploy, including
  - Automating check gates
  - A/B deployments and other methods for checking sanity
- As an SRE, don’t be afraid to roll-back a problem release.
- Use engineering principles to **manage configuration**, including
  - Treating configuration as code, with
    - version control
    - reviews and checks
    - testing
    - change management
  - Automating configuration “deployment”

Keep it Simple

> *The price of reliability is the pursuit of the utmost simplicity. - C.A.R. Hoare, Turing Award lecture*

- Types of complexity:
  - ***Essential*** complexity is the complexity inherent in a given situation that cannot be removed from a problem definition.
  - ***Accidental*** complexity is more fluid and can be resolved with engineering effort
- SREs minimize accidental complexity
  - They push back when it's introduced into systems
  - They constantly strive to eliminate complexity in systems they onboard and for which they assume operational responsibility
- Every new line of code written is a liability.
  - SRE promotes practices that make it more likely that all code has an essential purpose, such as scrutinizing code to make sure that it actually drives business goals, routinely removing dead code, and building bloat detection into all levels of testing.

Our Adoption Story

A version of the rest of our slides was shown at SRECon17 Americas during the session

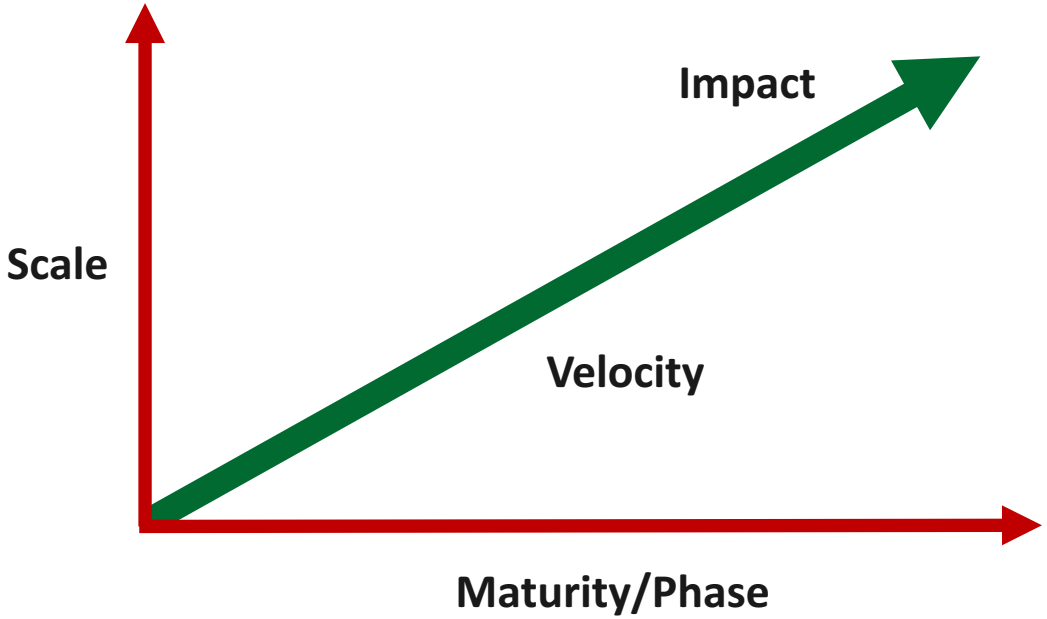
I'm an SRE Lead! Now What?

How to Bootstrap and Organize Your SRE Team.

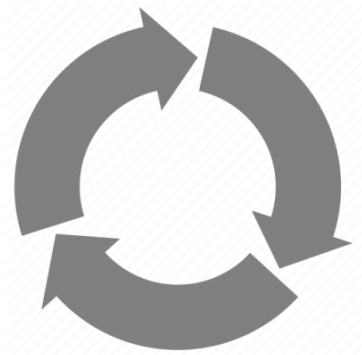
The recording is available at https://www.usenix.org/conference/srecon17americas/program/presentation/schacher

A Staged Approach

1. Define Scope and Obtain Stakeholder Buy-in
2. Define Tools, Processes, and Backlog
3. Build and Organize the SRE Team
4. Implement and Evolve



Iterate along the way



Define Scope and Obtain Stakeholder Buy-in

Build the leadership team and the mission

- SRE Technical leads, Service Managers, Architects
- Subject matter experts
- Define core values

Define the roles and responsibilities (SRE - Dev)

Define the Benchmarks

- Scorecards and checklists
- Promotion gates and approvals

Ensure stakeholder buy-in

- Executives
- Peer service managers
- Service tech leads

Example: SRE Values

We are **obsessed** with availability and reliability

We are an **Engineering** team

- We promote a high-performance culture
- We fill our skills gaps as necessary
- We have an automation mindset
- We use metrics, monitoring and measurement to ensure accountability
- We use Agile to manage our workload

We shape direction and **Positively Influence Outcomes** for our services

- Our RCA actions, checklists, and gap analyses directly feed into development prioritization

We **partner** with the service teams in all respects

- We understand and can use our services and capabilities
- We focus on the customer experience
- We discourage an ”us vs. them” mindset
- We align our priorities

We respect the **autonomy** of the service squads

Example: SRE Responsibilities and Control Points

Responsibilities

- Maintain Stated Availability of Service
- Incident, Problem, Capacity, Security, and Change Management
- Development of features that enhance Site Availability
- Automation and Monitoring Development
- Creation and Publishing of Dashboards and Availability Metrics

Control Points

- Provide Architectural Review and Signoff on a Service based on ability to achieve availability targets
- Accept or Reject services based on their ability to achieve SLA
- Own the Pipeline and Deployments to production
- Gate changes into a production Service contingent on error budget
- Use Error Budget to balance development prioritization between SRE content and new features

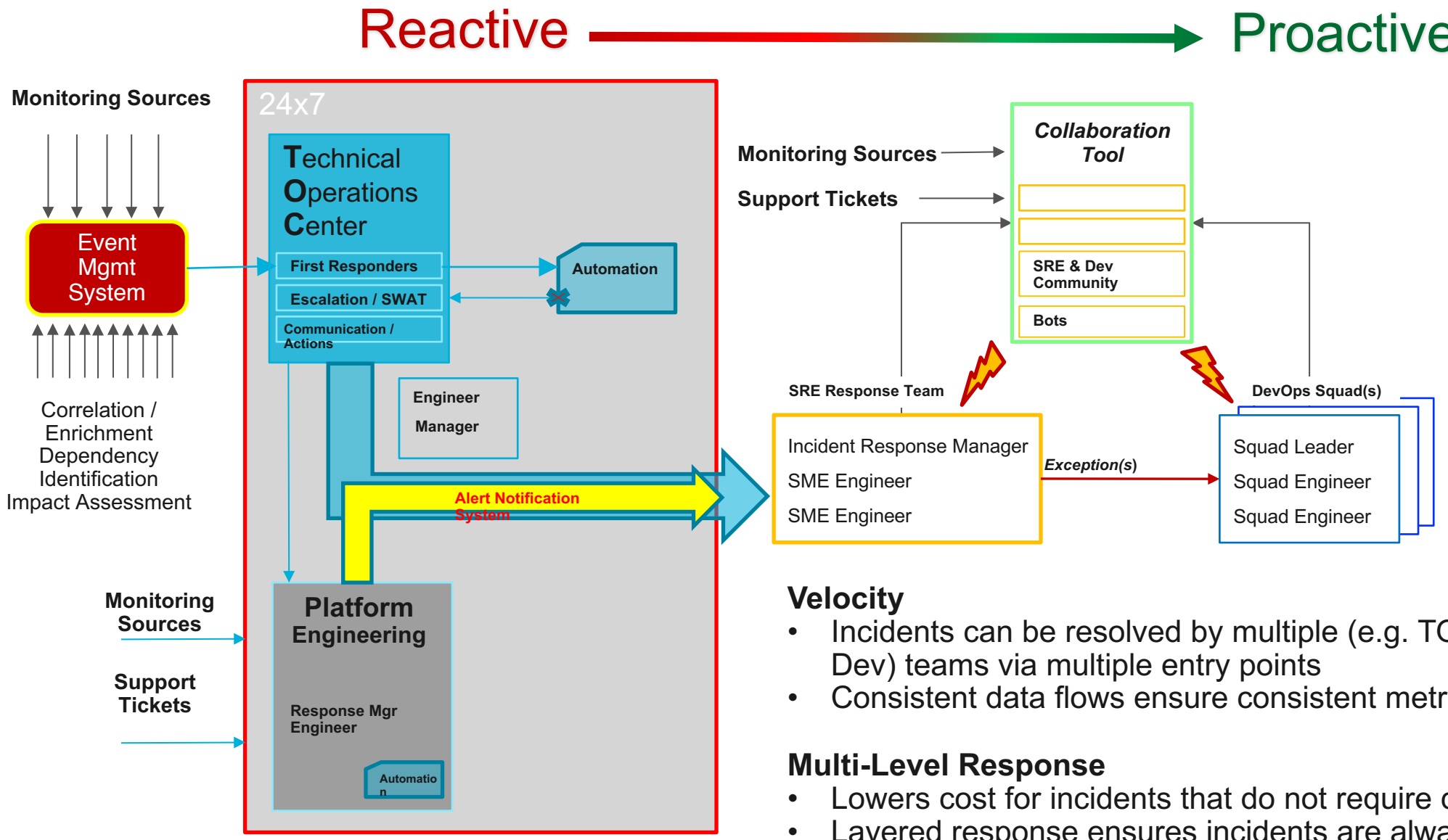
Decisions and recommendations of the SRE team are independent of functional or date commitments

Define Tools, Processes, and Backlog

Know, define or redefine tools and processes

- Tracking and Planning
- Monitoring and Availability Reporting
- Recovery Procedures & Automation
- Incident Management
- RCAs (with traceability and reporting)
- Deployments
- Test reporting
- Collaboration

Example: Incident Operational Model



Velocity

- Incidents can be resolved by multiple (e.g. TOC, SRE, Dev) teams via multiple entry points
- Consistent data flows ensure consistent metrics

Multi-Level Response

- Lowers cost for incidents that do not require deep skills
- Layered response ensures incidents are always handled

Define Tools, Processes, and Backlog

Establish **tracking and planning first**; everything else becomes a prioritized work item

Organize SRE **backlog**

Ensure prioritization of SRE work items is **consistent** with feature/function of service

Determine **control points** for adjusting work priorities, e.g.

- 7 day / 30 day availability **metrics**
- Business priorities of the service
- Security compliance

Establish **collaboration** communication and consistent information sharing

- Where do you go to find specific information types?
- Expedite information access and team **communication** during incident recovery

Build and Organize the Team

Build the **engineering** team

- New hires or rotational assignments?
- Evaluate candidate skills and experience
- Ensure broad skill coverage across all aspects of SRE

Organize vertically and horizontally: by **service** and by **discipline**

Designate **SMEs** for each service

- Avoid Human SPoF

Develop domain **expertise** (in the services)

- Architecture and topology, development environments, microservices, deployments
- Acquire skills to develop SRE enhancements to service

Define and **train** call out groups for incident resolution

Cultivate the SRE **mindset**

- Leadership coaching, all hands meetings, reinforce SRE values, conduct retrospectives

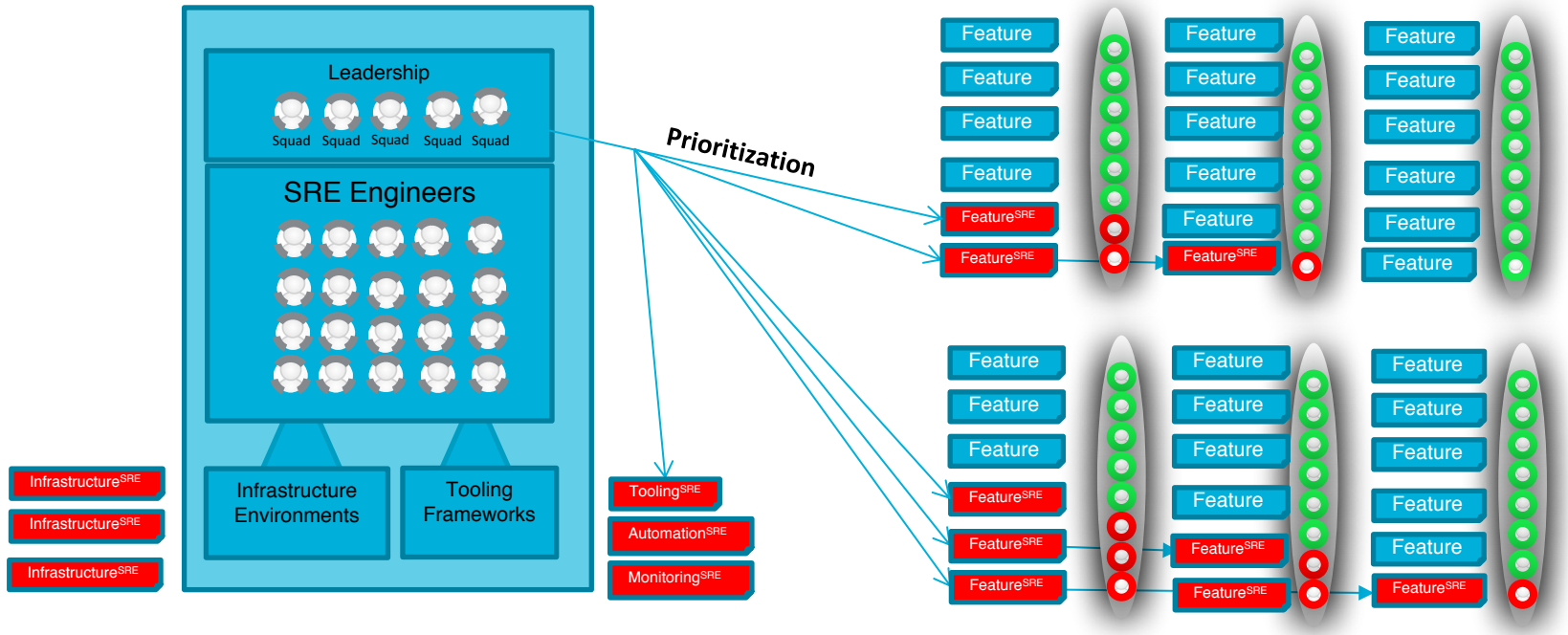
SRE Squad to Service Squad Operational Model

SRE Squad

- SRE STSM drives org-wide priorities
- Engineers rotate on flexible cadence between SRE and Feature squads (some core remain)

Service Development Squads

- Levels of maturity drive amount of resource required for SRE backlog
- 90% of the time these team members are the ones making code changes to the service



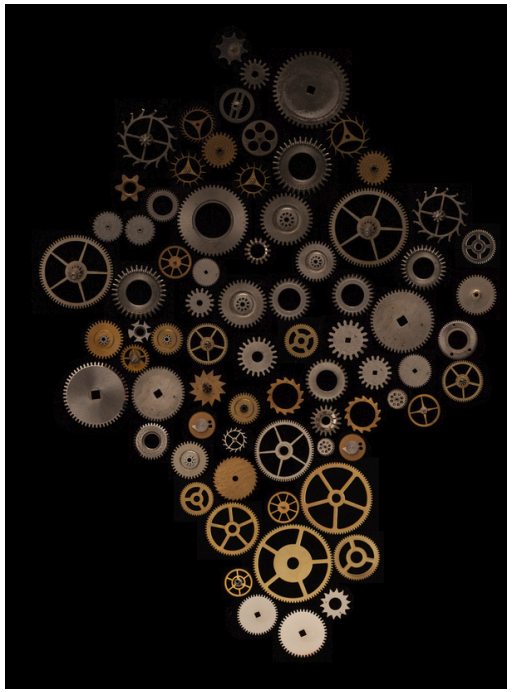
SRE Backlog examples

Automation Framework
Self Service Infrastructure
Logging & Metrics
Monitoring & Reporting
Scaling

Feature Squad SRE Backlog examples

High Availability
Rate Limiting
Zero Downtime Deployment
Resiliency
Serviceability

Implement and Evolve



Used a phased implementation for SRE ownership of a Service

- **Level I:** Monitoring and reporting tool frameworks, first responder service, Incident Manager support, RCA leadership
- **Level II:** Incident Mgmt, Deployment Mgmt, Availability, Security, Compliance and BC/DR

Onboard services to entry level of SRE ownership (Level 1)

Conduct gap analysis of services to:

- Assess the maturity of the service
- Identify gaps gating full SRE take over

Define a roadmap of activities required for SRE Ownership and collaborate with service development to build a plan to address

Define service SLOs and SLIs

Implement SLI monitoring and reporting

SRE Take over of services (Level II Full SRE Ownership)

Avoiding Pitfalls and Lessons Learned

Manage expectations

- Agree on the mission and scope first
- Keep it grounded in reality (e.g. development is NOT off the hook)
- Review and negotiate the roles and responsibilities
- Get **buy-in** from **everyone**

Don’t throw an army of engineers at the problem before you have your mission, scope and backlog defined

- Engineers will appreciate a plan and structure with clearly communicated expectations
- Executives and peer development managers will know what their role is and what to expect

Crawl, Walk, Run

- Tackle the obvious items first (e.g RCAs, monitoring gaps, runbooks, processes, reducing MTTA/MTTR)
- Show early success, incremental progress
- Build on success with improved metrics and automation

Ruthlessly prioritize the backlog

- Better to select a handful of items to focus on and do well, than try to do everything with slow velocity
- Review the priorities monthly with stakeholders
- Practice Agile!

Avoiding Pitfalls and Lessons Learned

DevOps and SRE are more than just rebranding of Dev and Operations

- Push the engineers out of an ops mindset, comfort zone
- Watch for an ”us vs. them” culture
- Be pragmatic vs. purist, without sacrificing principles
- Push for automation

SRE doesn’t come for free

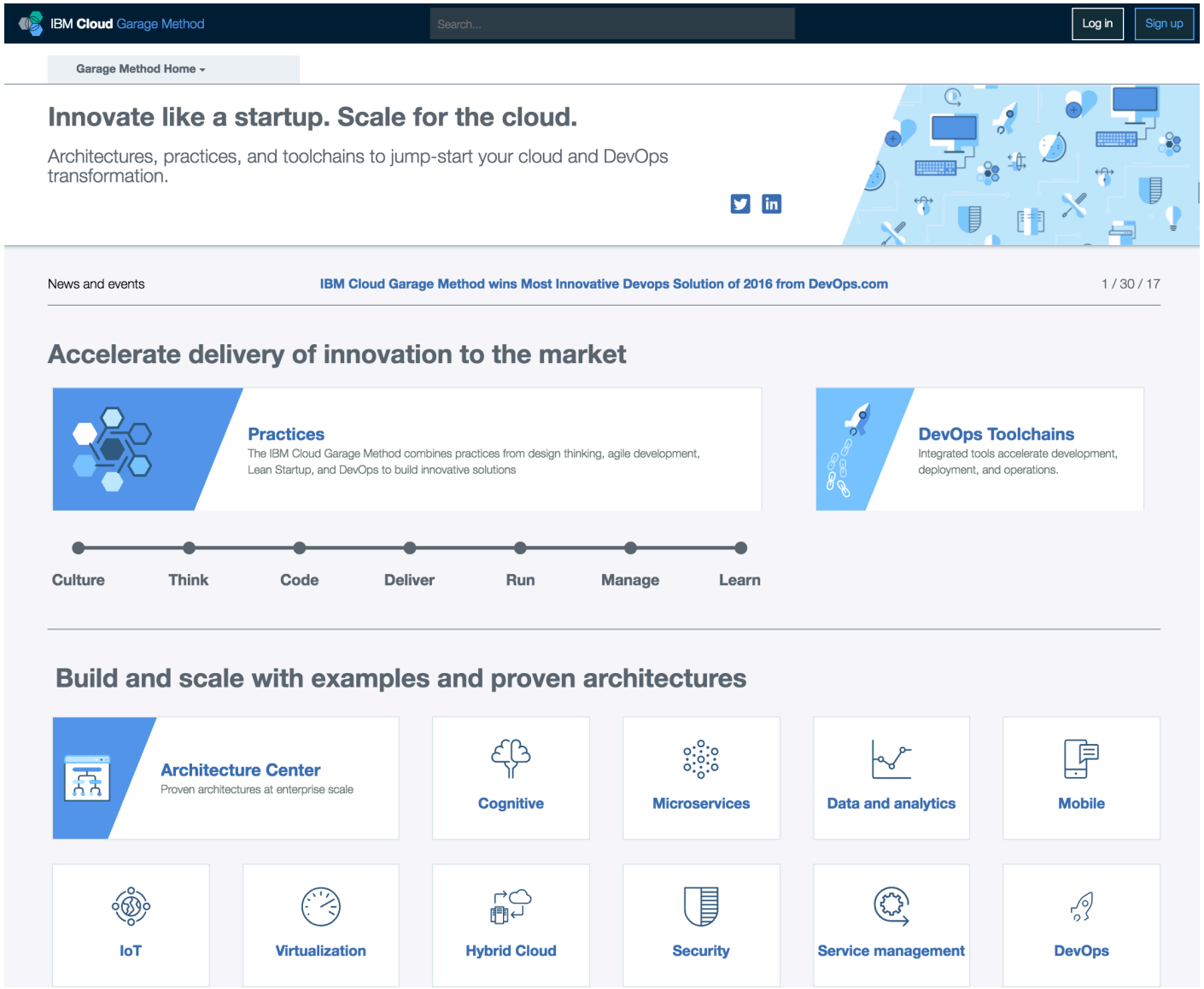
- This is a focus that did not exist before; either pull resources from services or hire new
- Short term velocity will be slow but will improve
- Good thing is that once established, this approach scales

Not enough engineers, not right skills

- SRE must be seeded with engineers experienced in the services
- Establish a resource rotation model with development to balance out the team
- Negotiate with the stakeholders; know who they are sending over

Our Services

Garage Method



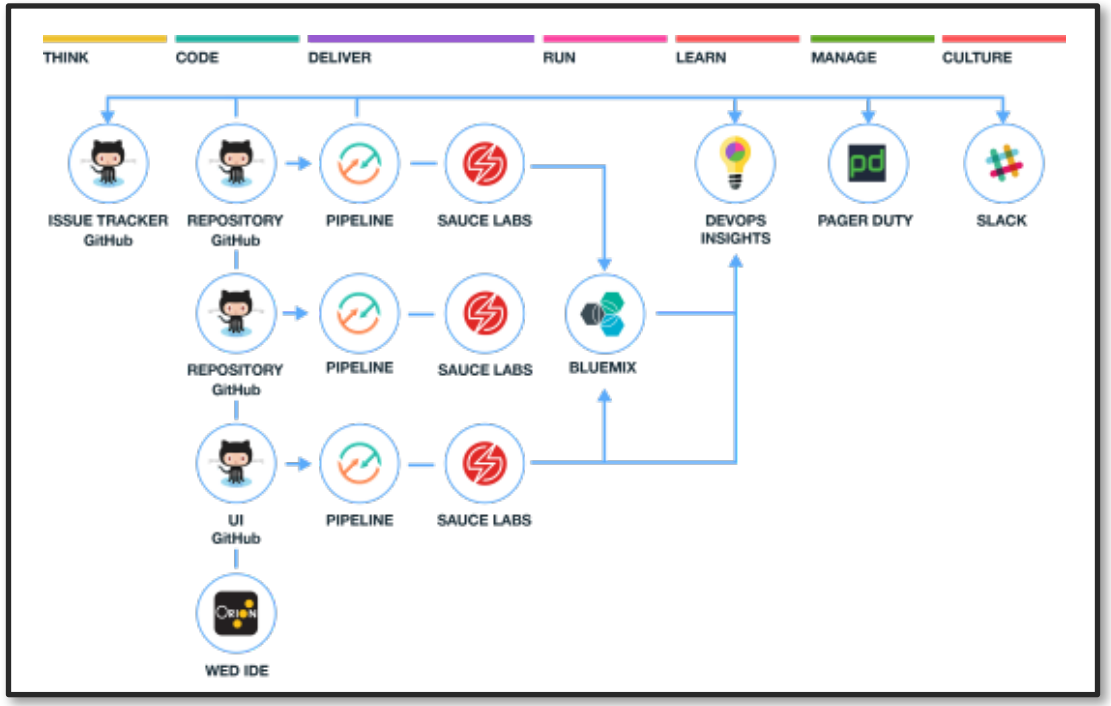
Combine industry practices, including **IBM Design Thinking, Lean Startup, agile development, and continuous delivery**, to build **innovative solutions**

ibm.com/devops/method

Open Toolchain

Create and manage toolchains of best-of-breed industry tools

A sample open toolchain for building, and deploying and managing three microservices



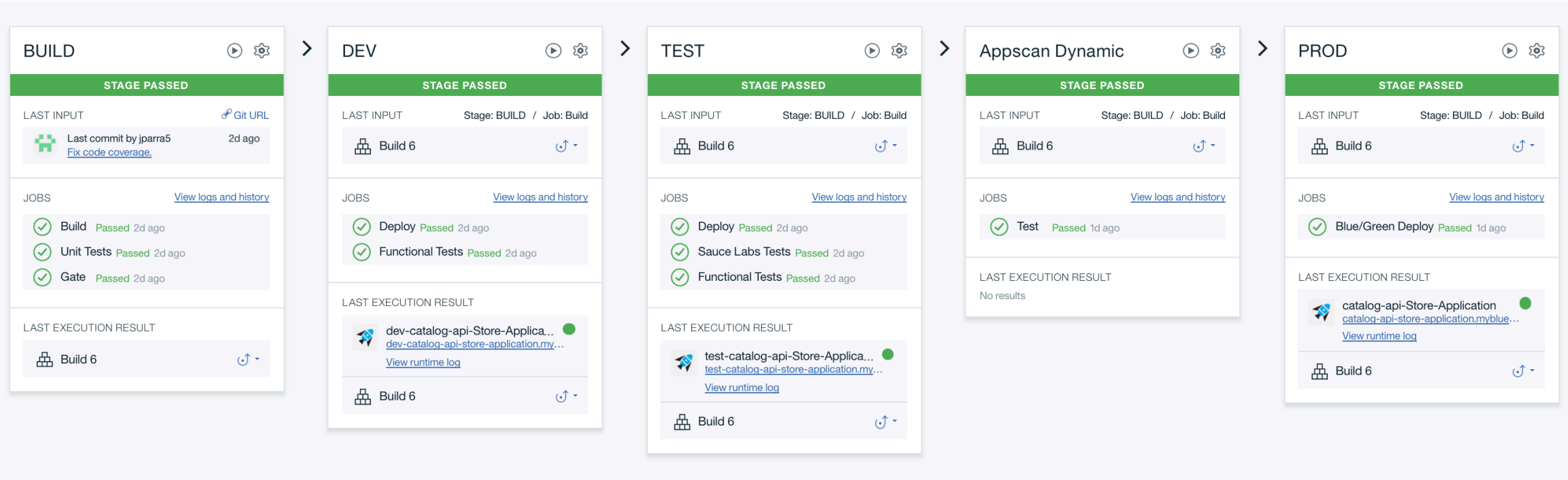
Toolchains provide an integrated set of tools that support the best practices to build, deploy and manage your apps.

You can create toolchains that include Bluemix services, open source tools, and third-party tools that make development and operations repeatable and easier to manage.

Rapidly instantiate new toolchains from templates to on-board new teams quickly.

bluemix.net/devops

Integrated Delivery Pipeline



Easy Setup

- Deploy an application from a Git repository in a few clicks.

Continuous Integration

- Automate builds and deployments for many types of code, running builds automatically when code changes.

Continuous Testing

- Integrate automated unit tests as part of your builds.

Continuous Deliver to Multiple Cloud Platforms

- Deploy applications to one or many Cloud Foundry or IBM Containers on Bluemix environments.

Thank you!

References

- [Book] B. Beyer, et. al., "Site Reliability Engineering: How Google Runs Production Systems", ISBN: 9781491929124
- [Try17] B. Traynor, "What is 'Site Reliability Engineering'?", blog post, captured 15 April 2017.
- [Sch15] B. Schwartz, "The Factors That Impact Availability, Visualized", blog post, 21 December 2015.