
newesis
Be Professional | Have Fun!
DEVOPS SERVICES COMPANY
DevOps Introduction

- Twitter: @RaunoDepa

ABOUT ME

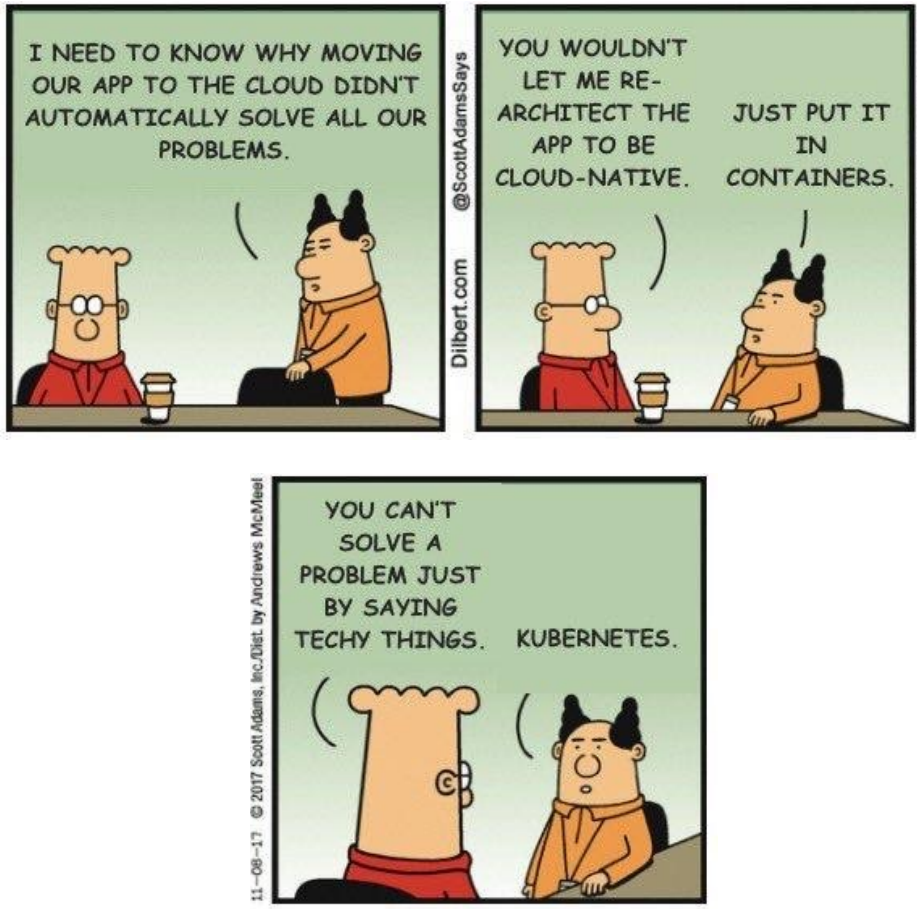
- Rauno De Pasquale, Co-Founder and CTO at Newesis Srl, constantly trying to reconcile his degree in Philosophy with a passion for computer science. After almost 18 year at Deltatre, at the beginning of 2019 he creates Newesis, with the aim of simplifying the use of the most advanced services of Cloud platforms even in fields other than sports.

- Twitter: @newesissrl

- Newesis is a pure technology based, Cloud Native company, having as a target to support and evolve adoption of DevOps culture and practices, with particular focus on Kubernetes as technological platform.

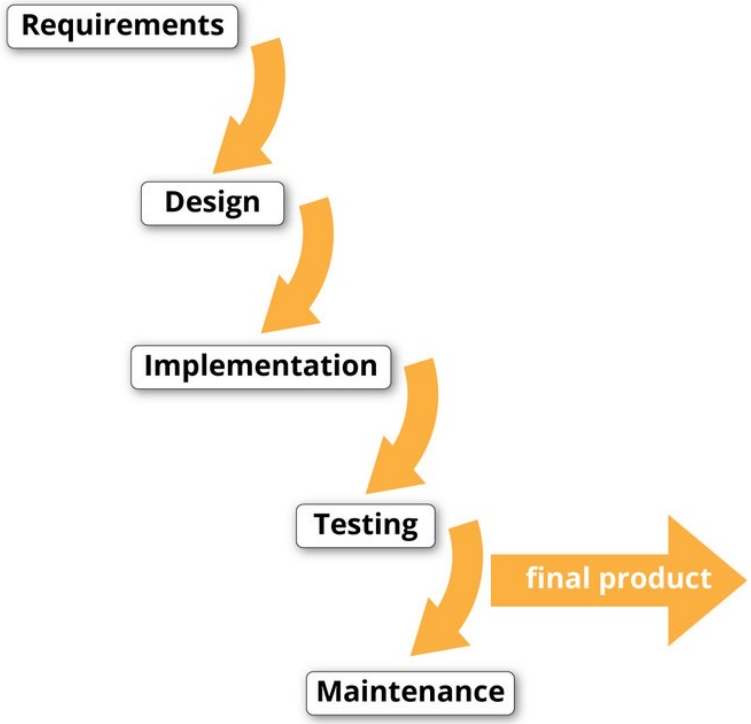
AGENDA

- Agile and DevOps: introduction and concepts
  - Agile – history and concepts
  - DevOps – history and concepts
  - Google SRE and DevOps principles
  - Cloud Native Architectures
  - Infrastructure as code and automation
  - How as a DevOps Service Company we do actually apply Agile Principles



WHAT IS AGILE?

- Snowbird, Utah, is an unlikely place to mount a software revolution. Around 25 miles outside Salt Lake City, Snowbird is certainly no Silicon Valley; it is not known for sunny and temperate climes, for tech-innovation hubs, or for a surplus of ever eager entrepreneurs. But it was here, nestled in the white-capped mountains at a ski resort, that a group of software rebels gathered in 2001 to frame and sign one of the most important documents in its industry's history, a sort of Declaration of Independence for the coding set.
- In the early 1990s, as PC computing began to proliferate in the enterprise, software development faced a crisis. At the time, it was widely referred to as "the application development crisis," or "application delivery lag." Industry experts estimated that the time between a validated business need and an actual application in production was about three years.
- In certain industries, the lag was far greater than three years. In aerospace and defense, it could be 20 or more years before a complex system went into actual use.



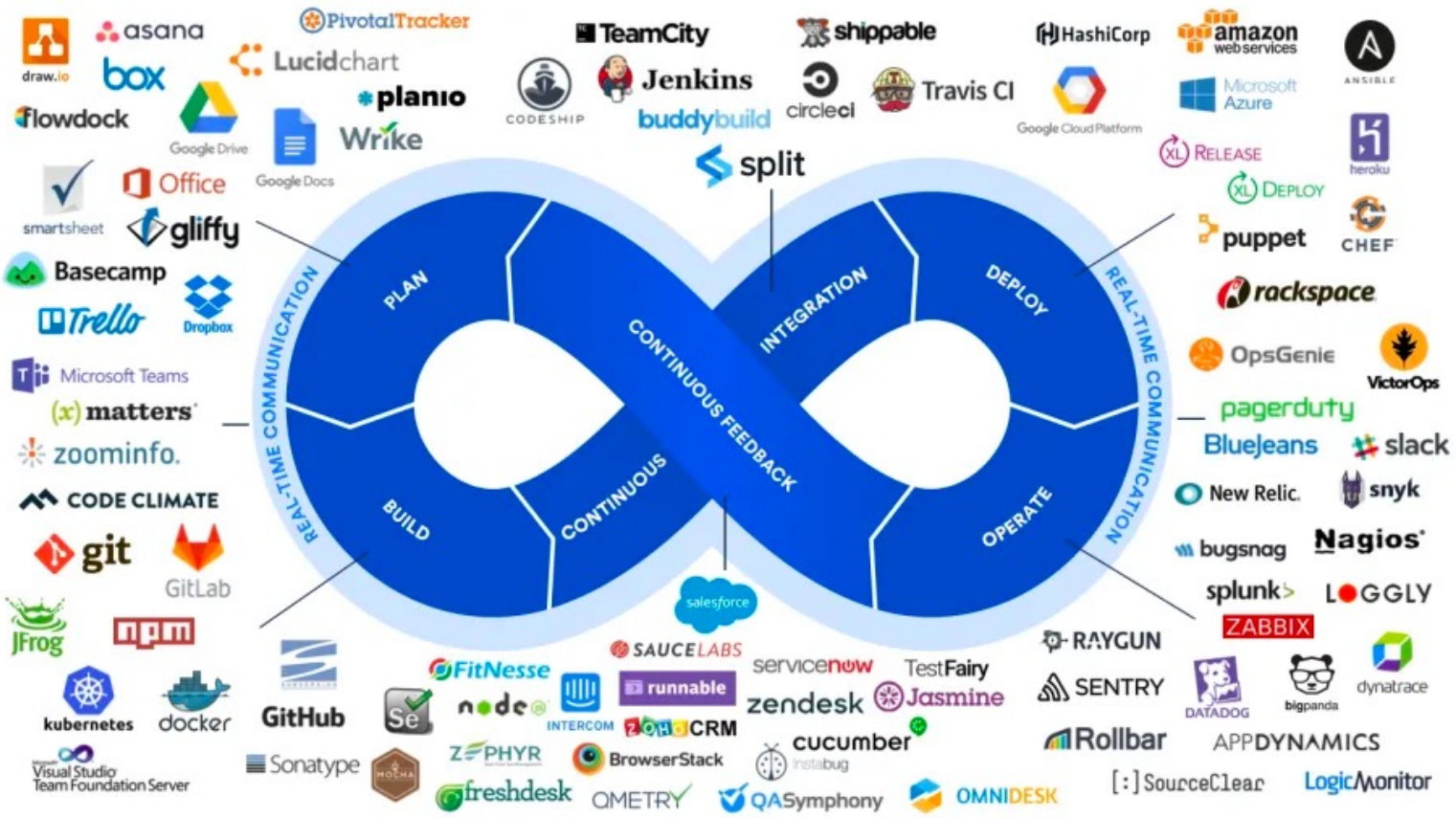
AGILE MANIFESTO

- Software projects rarely have the same kind of stability as traditional engineering projects. Business needs change, seemingly overnight, and certainly faster than the months or years formerly required to complete a software application.
- Software design is both a science and an art, with imperfections and associated human limitations
- The translation from requirements, imperfect as they are, to specifications, and from specifications to implementation, is rife with ambiguities.

12 AGILE PRINCIPLES

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
04 Business people and developers must work together daily throughout the project.	05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
07 Working software is the primary measure of progress.	08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	09 Continuous attention to technical excellence and good design enhances agility.
10 Simplicity – the art of maximizing the amount of work not done – is essential.	11 The best architectures, requirements, and designs emerge from self-organizing teams.	12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

WHAT IS DEVOPS?



Atlassian

DEVOPS – A DEFINITION

DevOps is a set of practices that automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably.

The concept of DevOps is founded on building a culture of collaboration between teams that historically functioned in relative siloes. The promised benefits include increased trust, faster software releases, ability to solve critical issues quickly, and better manage unplanned work.

Amazon

DEVOPS – A DEFINITION

The combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity:

evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

Donovan Brown – Principal DevOps Manager at Microsoft

DEVOPS – A DEFINITION

DevOps is the union of people, process, and products to enable continuous delivery of value to our end users.

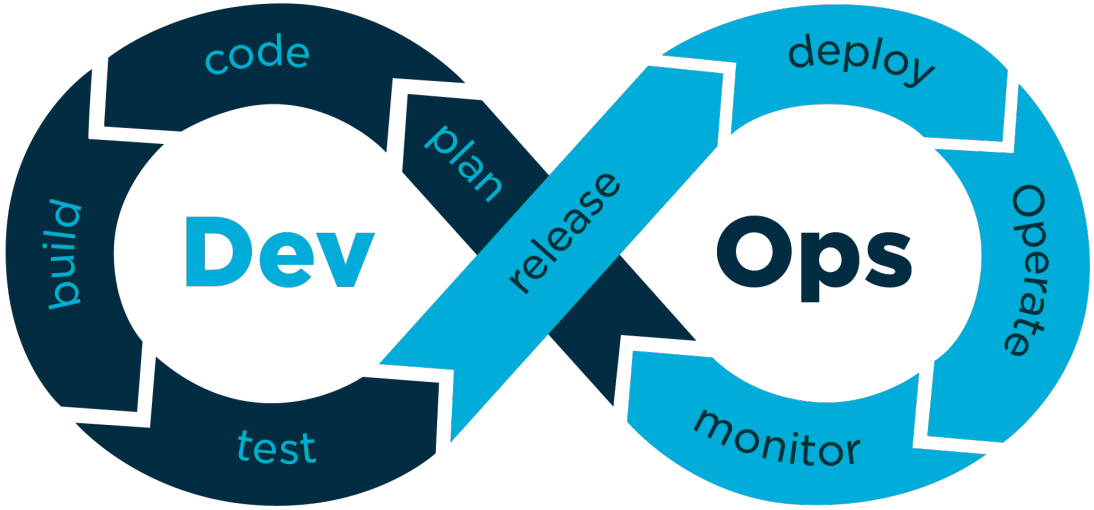
DevOps is not just automating a pipeline so we can quickly deliver software. Our goal is to deliver value.

It is very important to realize that DevOps is not a product.

You cannot buy DevOps and install it.

DevOps is not just automation or infrastructure as code.

DevOps is people following a process enabled by products to deliver value to our end users.



THE RAISE OF DEVOPS

- At the 2008 Agile Toronto conference, Patrick Dubois and Andrew Schafer held a session on applying Agile principles to infrastructure as opposed to application code.
- Velocity 2009 conference, John Allspaw and Paul Hammond gave the seminal “10 Deploys per Day: Dev and Ops Cooperation at Flickr.”
- Patrick Dubois created the first DevOpsDays in Ghent, Belgium, in 2009, coining the word “DevOps.”
- in 2009 at another O’Reilly conference, Velocity, Andrew Clay Shafer gave a presentation on Agile Infrastructure, showing the iconic picture of a wall between developers and operations with a metaphorical depiction of work being thrown over the wall.
- In March of 2011, Cameron Haight of Gartner presented his predictions for the trajectory of DevOps over the next few years, creating interest in DevOps movement by enterprises.
- In 2012, the State of DevOps report was conceived and launched by Alanna Brown at Puppet.

FALSE MYTHS

1. DevOps is Only for Startups
2. DevOps Replaces Agile
3. DevOps is Incompatible with ITIL
4. DevOps is Incompatible with Information Security and Compliance
5. DevOps Means Eliminating IT Operations, or “NoOps”
6. DevOps is Just “Infrastructure as Code” or Automation
7. DevOps is Only for Open Source Software
8. DevOps cross functional teams mean to increase costs because you need more people

DEVOPS PRINCIPLES – CAMS

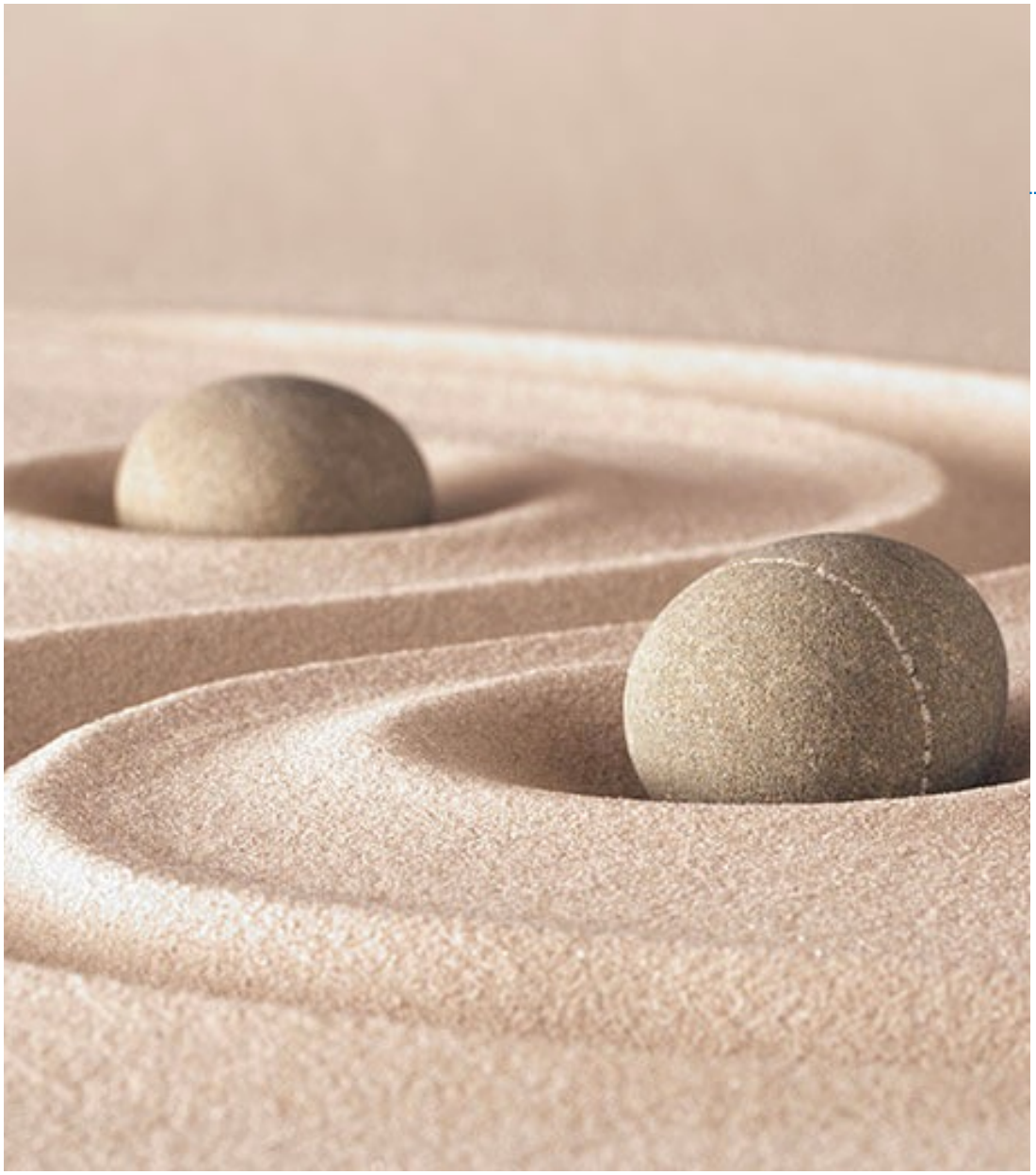
- Culture
  - “When asked what the most difficult thing about adopting DevOps is, I reply — without hesitation — the people” (Donovan Brown). People and process first. If you don’t have culture, all automation attempts will be fruitless.
- Automation
  - The full independency and responsibility of the team for the delivery of value to the end user is a fundamental principle of DevOps. Automation is key not only to allow faster iterations but specifically to make the process repeatable independently from the presence of a certain person or group of people.
- Measurement
  - The continuous improvement process (Kaizen) is at the heart of DevOps. If you can’t measure, you can’t improve. A successful Devops implementation will measure everything it can as often as it can.
- Sharing
  - The transfer of knowledge aims to avoid constraints in the organization, and to promote the collective intelligence. Visibility makes it possible to know if the work of a team can introduce a problem to another team and it also allows for early feedback. Transparency is what allows everyone to work towards a common goal.

THE THREE WAYS

System Thinking: the principles of Flow, which accelerate the delivery of work from Development to Operations to our customers, emphasizes the performance of the entire system, as opposed to the performance of a specific silo of work or department.

Feedback Loops: the principles of Feedback, which enable us to create ever safer systems of work, creating the right to left feedback loops.

Culture of Experimentation: the principles of Continual Learning and Experimentation, which foster a high-trust culture and a scientific approach to organizational improvement risk-taking as part of our daily work



WHY DEVOPS – THE CONTEXT

Agile Software Development

– faster iteration

Microservices Architecture

– increasing complexity

Cloud Platforms

– increasing scale

Containers, Serverless

– increasing abstraction

WHY DEVOPS – DEV VS OPS

"I need a server… now"

"I want access to the production systems"

"The application won't run on our infrastructure because we don't support that version"

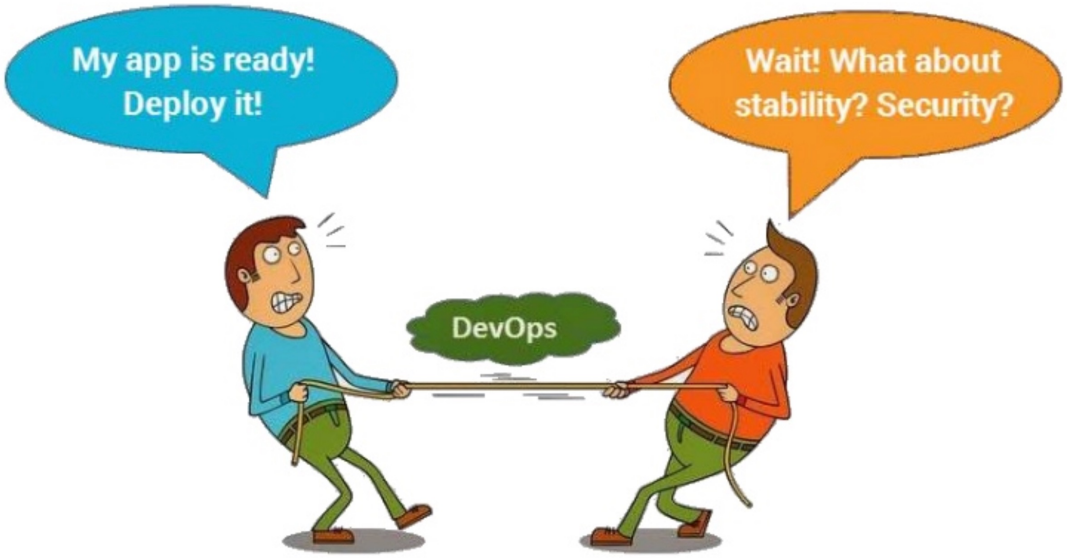
"The architecture of the application doesn't match our security model"

"We can not work on that task because we need xyz who is currently busy working late at night" (the "hero" syndrome)

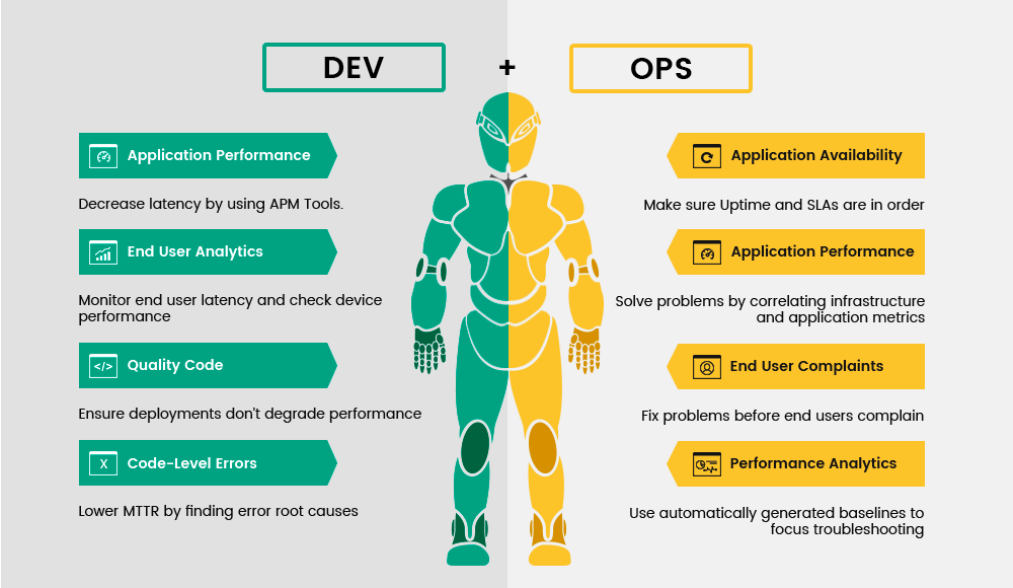
"We weren't consulted about the backup requirement so it can not be move to production"

"It's not our fault, our code is perfect, it works on my machine"

"Why can't they implement that technology? Why are they so backward?"



DEV DOES OPS – OPS DOES DEV



Common mistake is to think DevOps as “Dev does Ops” or as “Ops does Dev”.

Dev and Ops have different skillsets and knowledge that must be combined to get values for the end users.

Sysadmins have the ability to transition to a DevOps team, as long as they are willing to learn current and emerging technologies, and are open to innovative ideas and solutions.

Developers have the ability to transition to a DevOps team, as long as they are willing to consider reliability and operability as part of their duties, and are open to innovative ideas and solutions.

DEVOPS IS DEV + OPS

“DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.” (Jez Humble)

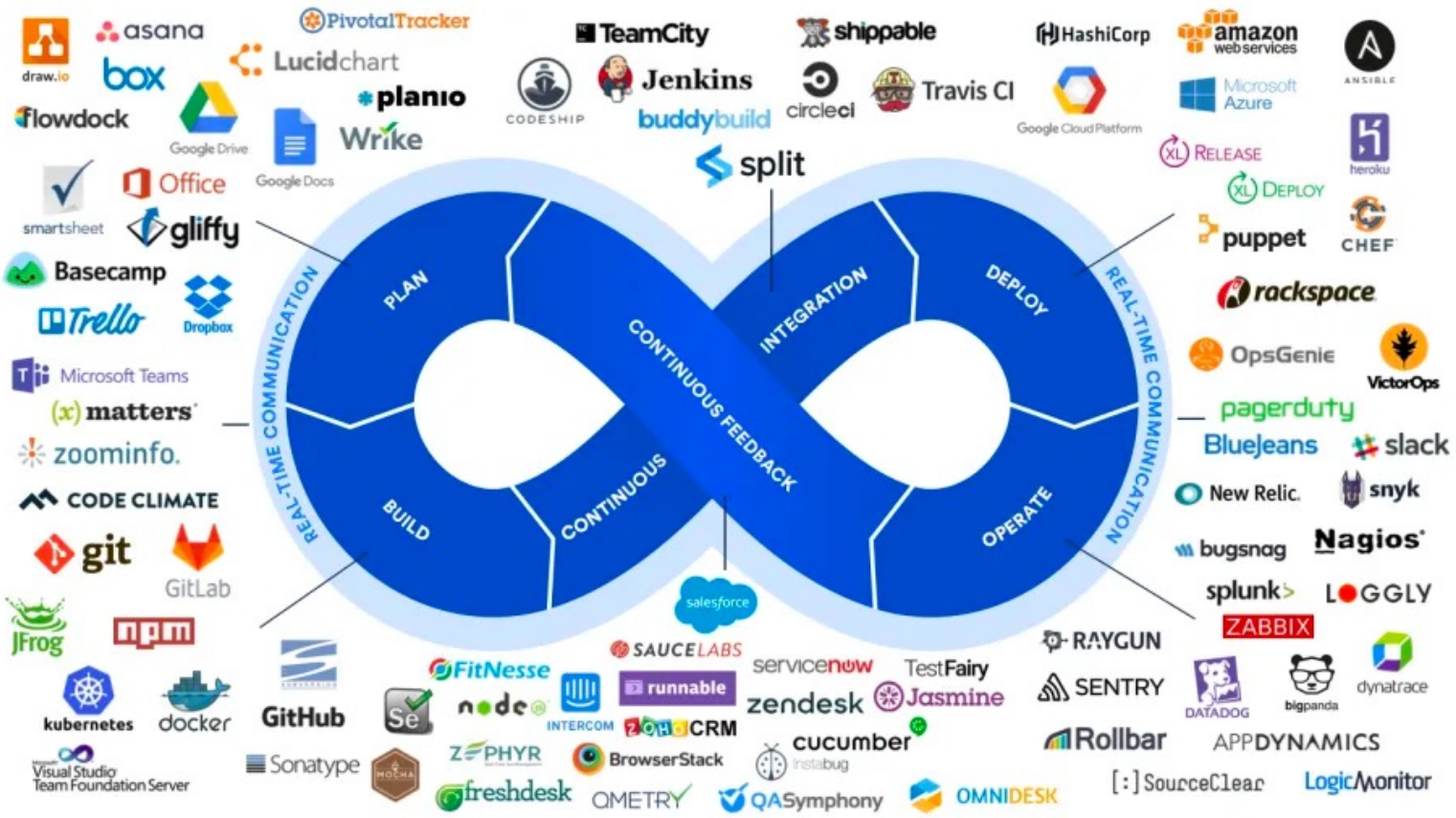
“DevOps is about Ops who think like devs. Devs who think like ops.” (John Allspaw)

Development teams need to embed operations people into their project and development life cycles.

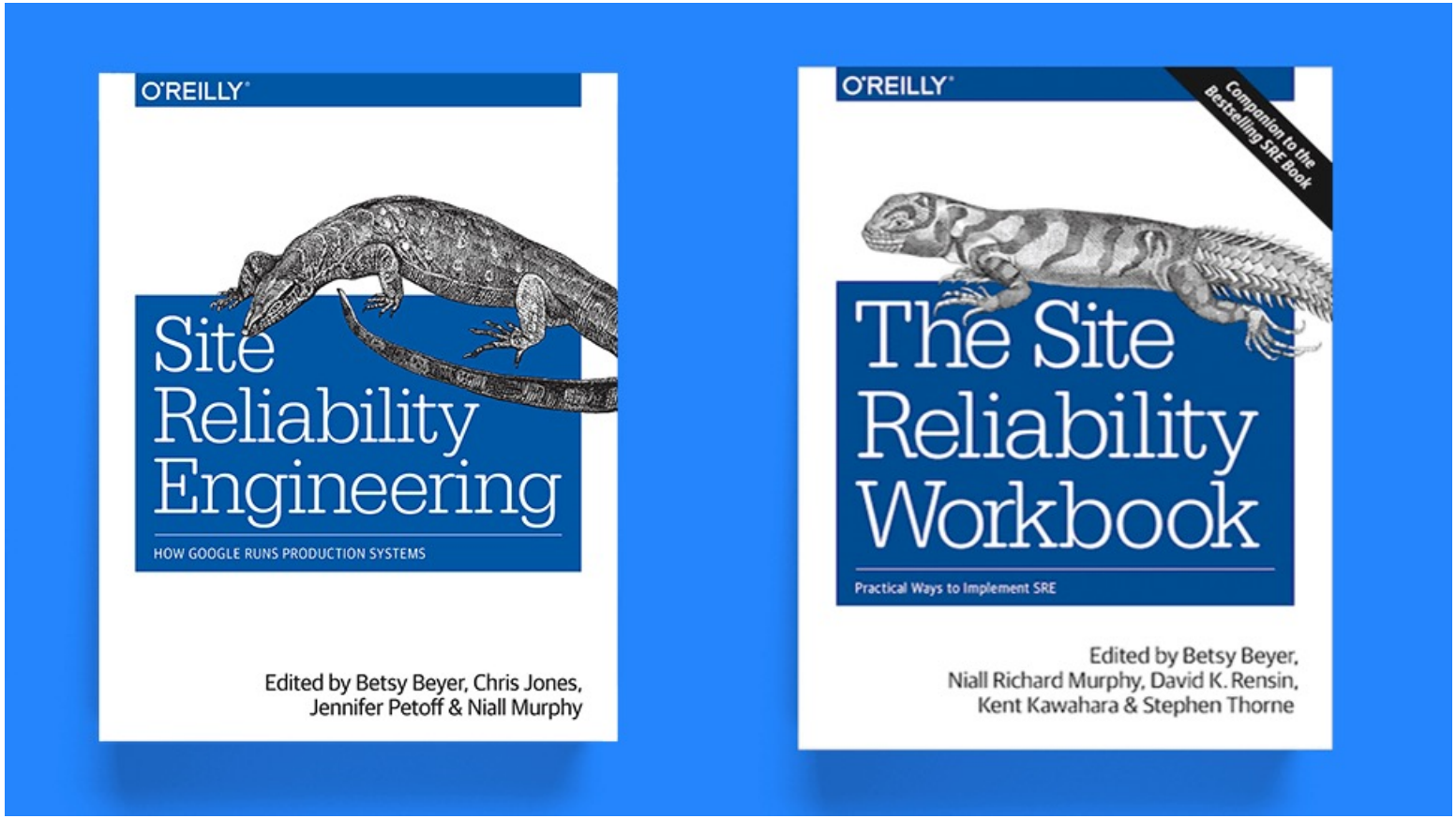
Operations people need to bring development people into the problem and change management space.

“DevOps is also characterized by operations staff making use many of the same techniques as developers for their systems work.” (Jez Humble)

WHAT IS YOUR DEFINITION OF DEVOPS?



GOOGLE SRE – HISTORY



Ben Traynor, VP of engineering at Google

THE ORIGIN

Site reliability engineering (SRE) was **born at Google in 2003**, prior to the DevOps movement.

"**SRE is what happens when you ask a software engineer to design an operations team**." - Ben Traynor, VP of engineering at Google and founder of Google SRE

"SRE is fundamentally doing work that has historically been done by an operations team but using engineers with software expertise and banking on the fact that these engineers are inherently both predisposed to, and have the ability to, substitute automation for human labour. In general, an **SRE team is responsible for availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning**." - Ben Traynor, VP of engineering at Google and founder of Google SRE

Site reliability engineers create **a bridge between development and operations** by **applying a software engineering mindset to system administration** topics.

They split their time between operations/on-call duties and developing systems and software that help increase site reliability and performance. Google puts a lot of emphasis on **SREs not spending more than 50% of their time on operations** and considers any violation of this rule a sign of system ill-health.

The ideal site reliability engineer candidate is either a software engineer with a good administration background or a highly skilled system administrator with knowledge of coding and automation.

"**SRE teams are characterized by both rapid innovation and a large acceptance of Change**" - Ben Traynor, VP of engineering at Google and founder of Google SRE

THE PROBLEM TO SOLVE

"Software engineering has this in common with having children: the labour before the birth is painful and difficult, but the labour after the birth is where you actually spend most of your effort. Yet **software engineering as a discipline spends much more time talking about the first period as opposed to the second** [omissis] **there must be another discipline that focuses on the whole lifecycle of software objects, from inception, through deployment and operation**, refinement, and eventual peaceful decommissioning."

"Running a service with a team that relies on manual intervention for both change management and event handling becomes expensive as the service and/or traffic to the service grows"

"**Traditional operations teams and their counterparts in product development often end up in conflict, most visibly over how quickly software can be released to production**. At their core, the development teams want to launch new features and see them adopted by users. At their core, the ops teams want to make sure the service doesn't break while they are holding the pager. Because most outages are caused by some kind of change a new configuration, a new feature launch, or a new type of user traffic the two teams' goals are fundamentally in tension."

Site Reliability Engineering

THE NAME

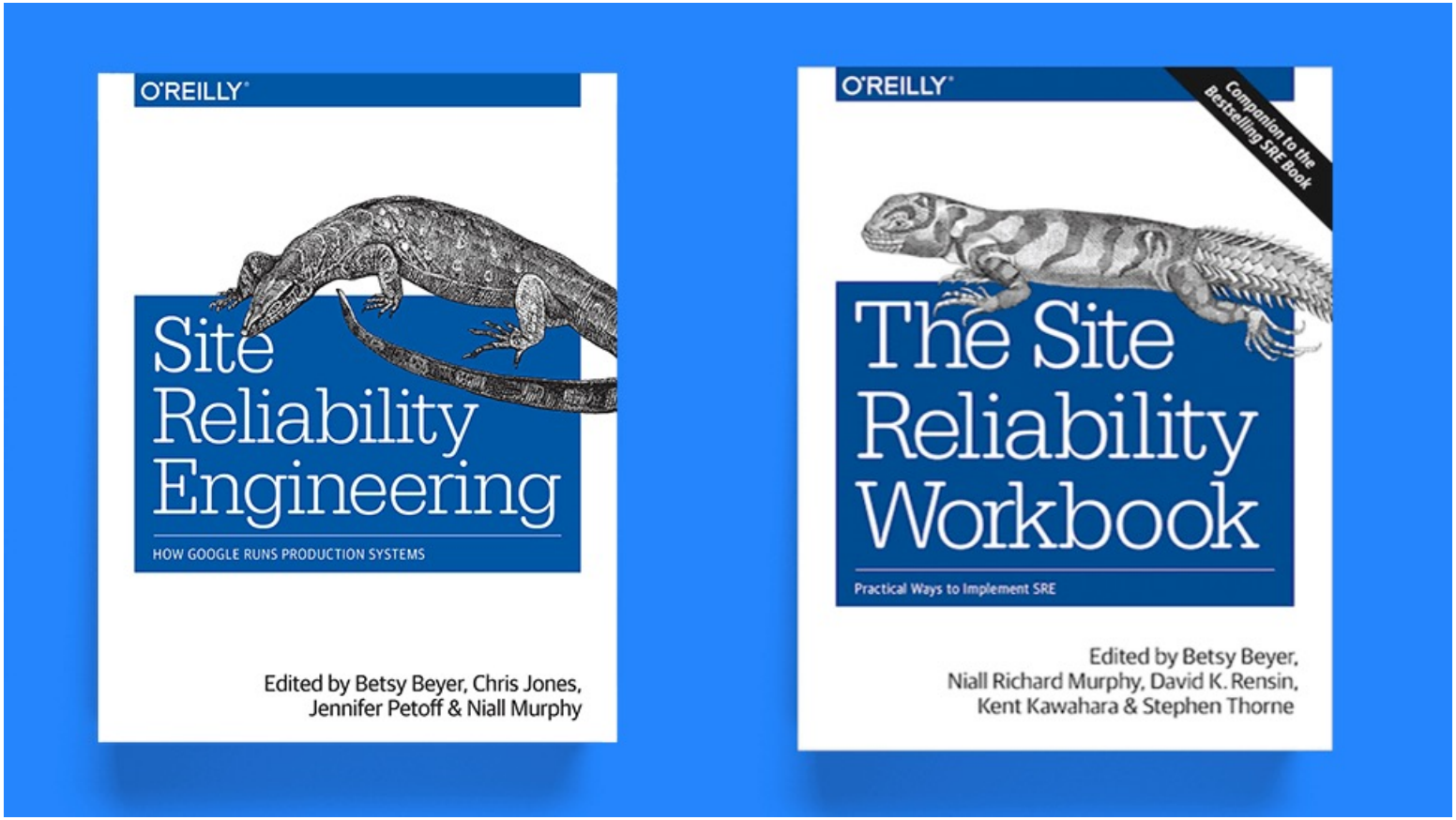
Site : the terms site comes from the original duty of the newly born role, having focus on google.com web site.

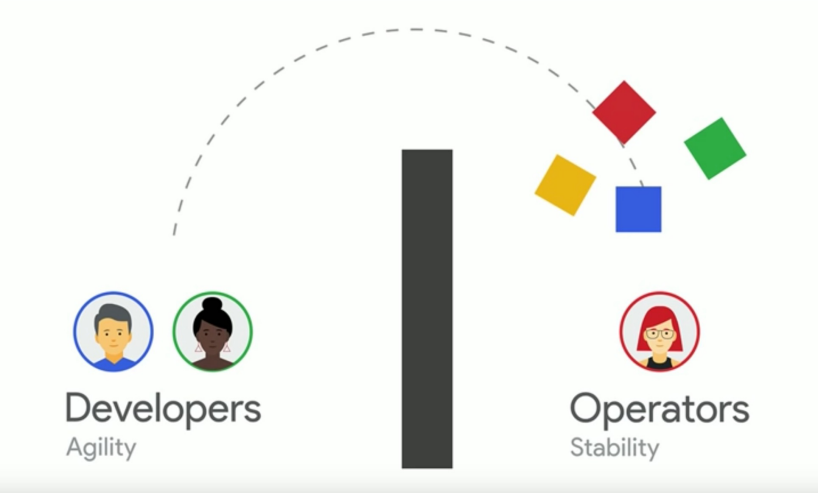
Reliability: “reliability is the most fundamental feature of any product: a system isn’t very useful if nobody can use it!” (Ben Traynor)

Engineering: “we apply the principles of computer science and engineering to the design and development of computing systems” “an SRE team must spend 50% of its time actually doing development.” (Ben Traynor)

.

GOOGLE SRE – CONCEPTS – SRE VS DEVOPS





SRE VS DEVOPS

DevOps movement in the community and SRE initiative in Google started from the same problem, the inefficiency of having Developers and Operators working on the different side of a wall, the first looking for feature and the second for stability.

“One could view DevOps as a generalization of several core SRE principles to a wider range of organizations, management structures, and personnel. One could equivalently view SRE as a specific implementation of DevOps with some idiosyncratic extensions.“(Ben Traynor)

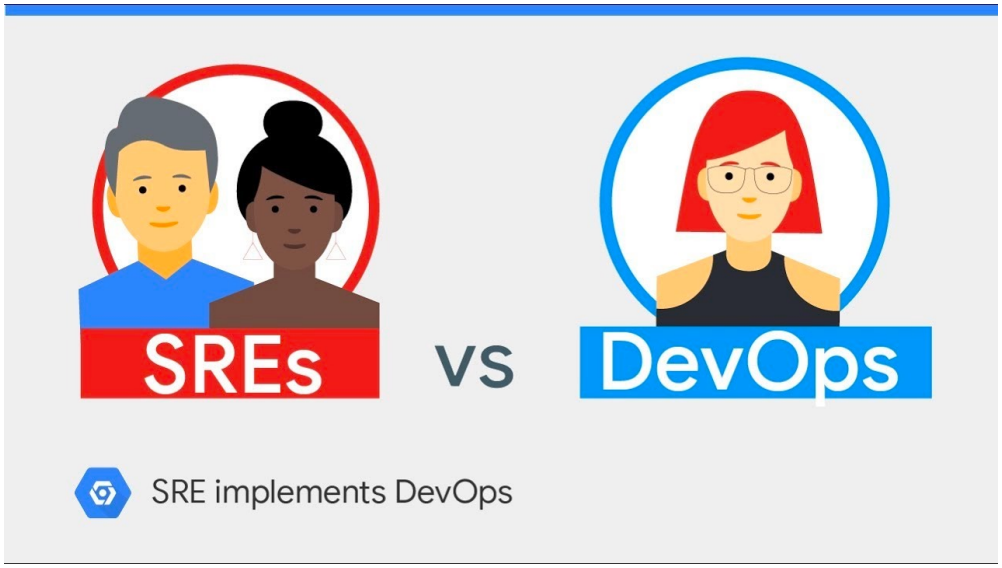
A DevOps Engineer is someone who understands the full SDLC (Software Development Life Cycle)

DevOps focuses more on the automation part

SREs focus is more on the aspects like system availability, observability, and scale

”**The basic tenet of SRE is that doing operations well is a software problem**”

DEVOPS VS SRE (SITE RELIABILITY ENGINEERING)



Is SRE an alternative to DevOps?

Google creates and evolved the concept independently from DevOps movement.

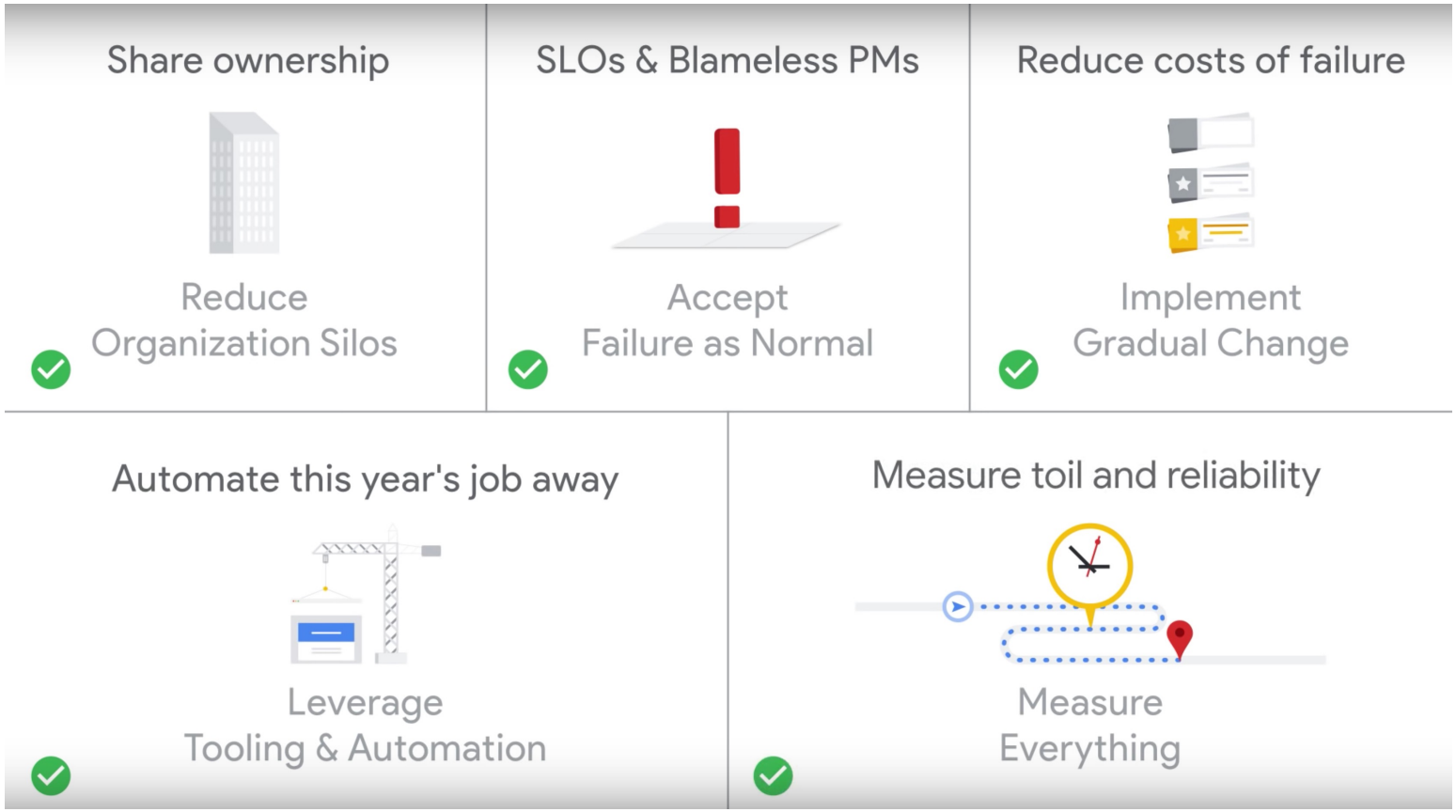
«If you think of DevOps like an interface in a programming language, *class SRE implements DevOps*. SRE includes additional practices and recommendations that are not necessarily part of the DevOps interface. DevOps and SRE are not two competing methods for software development and operations, but rather close friends designed to break down organizational barriers to deliver better software faster.

DevOps emerged as a culture and a set of practices that aims to reduce the gaps between software development and software operation.

The DevOps movement does not explicitly define how to succeed.

SRE prescribes how to succeed in the various DevOps areas.” (Liz Fong-Jones, Seth Vargo)

DEVOPS MANIFESTO — SRE MANIFESTO



SRE VS DEVOPS

Reduce organisation silos: SREs share the ownership of production with development teams and use the same tools

Accept failure as normal: SRE's concept of Error Budget, avoiding 100% SLO

Implement gradual change: SRE is prescriptive about usage of Canary Deployment

Leverage tooling and automation: SRE concept of Toil and the guideline of "automate this year's job"

Measure everything: SRE is prescriptive about measuring Error Budget and Toil

DEVOPS VS. SRE: COMPETING STANDARDS OR FRIENDS? (CLOUD NEXT '19) (@SETHVARGO)

Is SRE "DevOps 2.0"?

No, not trying to be

Can I adopt both DevOps & SRE?

Yes!

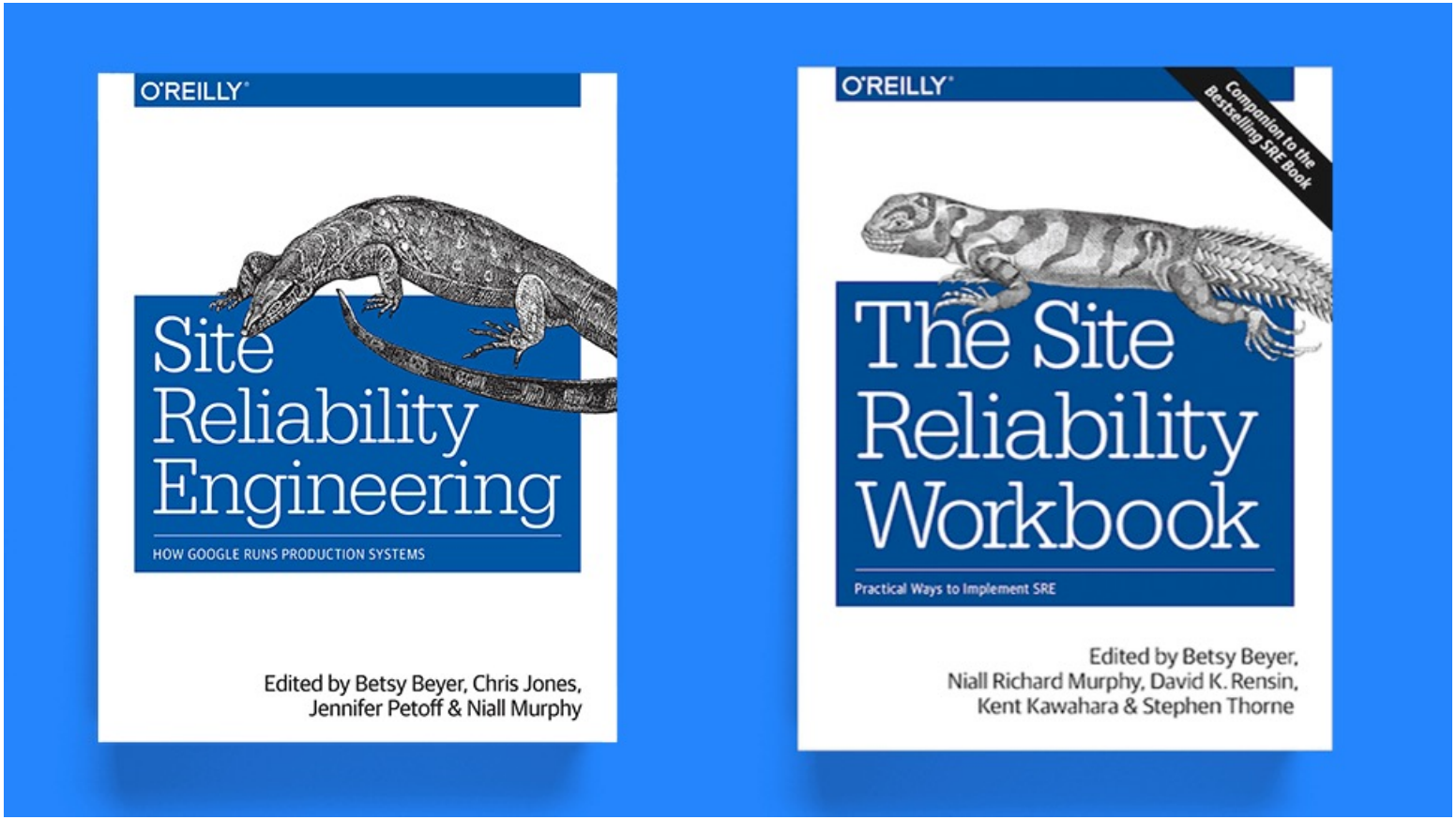
Is SRE trying to overtake DevOps?

No, lol

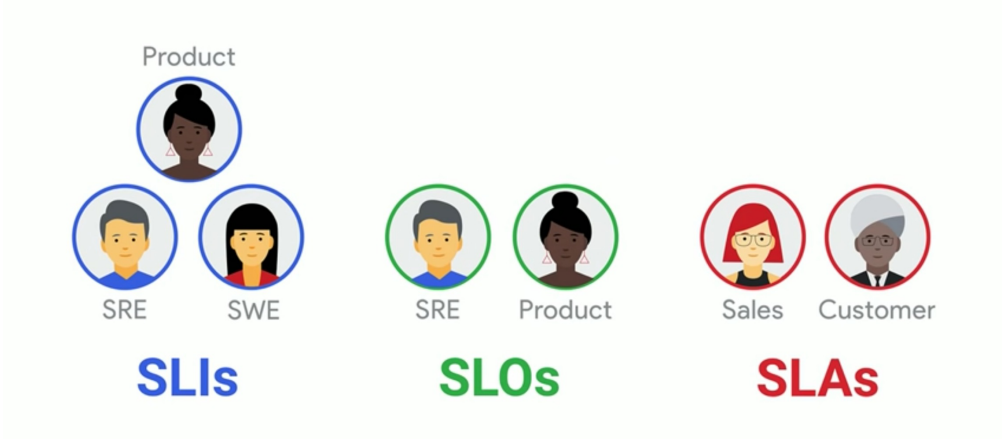
Is DevOps dead?

No (at least I hope not)

GOOGLE SRE — CONCEPTS - SLO



SLI, SLO, SLA – A DEFINITION

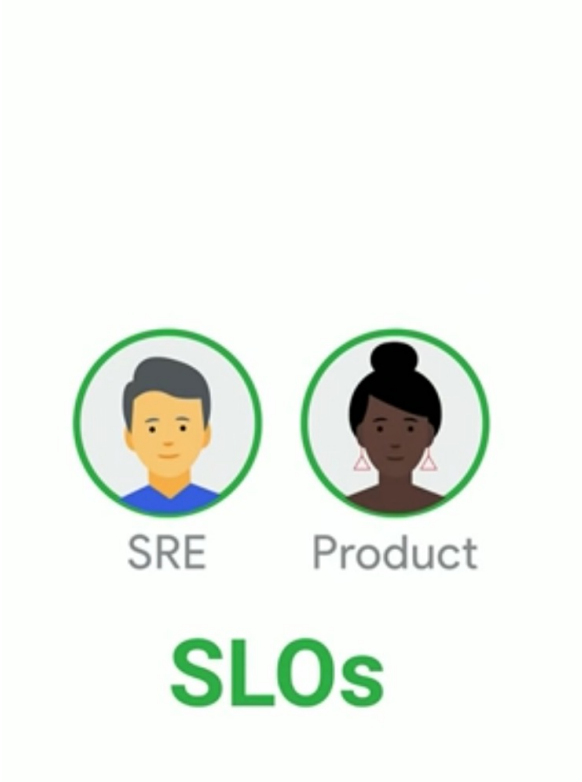


SLIs drive **SLOs** which inform **SLAs**

Service Level Indicators (SLIs): metrics over time such as request latency, throughput of requests per second, or failures per request

Service Level Objectives (SLOs): targets for the cumulative success of SLIs over a window of time

Service Level Agreements (SLAs): a promise by a service provider, to a service consumer, about the availability of a service



SLO – TARGET

"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."

100% is not a viable availability target. Having a 100% availability requirement severely limits a team or developer's ability to deliver updates and improvements to a system.

To set the target isn't a technical question at all, it's a product question, the business lead and the product management must establish the system's availability target:

- What level of availability will the users be happy with, given how they use the product?
- What alternatives are available to users who are dissatisfied with the product's availability?
- What happens to users' usage of the product at different availability levels?

SLI – SELECTION

SLIs must be selected on the base of business value considerations.

Valuable SLIs:

- Request latency
- Batch throughput
- Failures per request (error rate)

Non valuable SLIs:

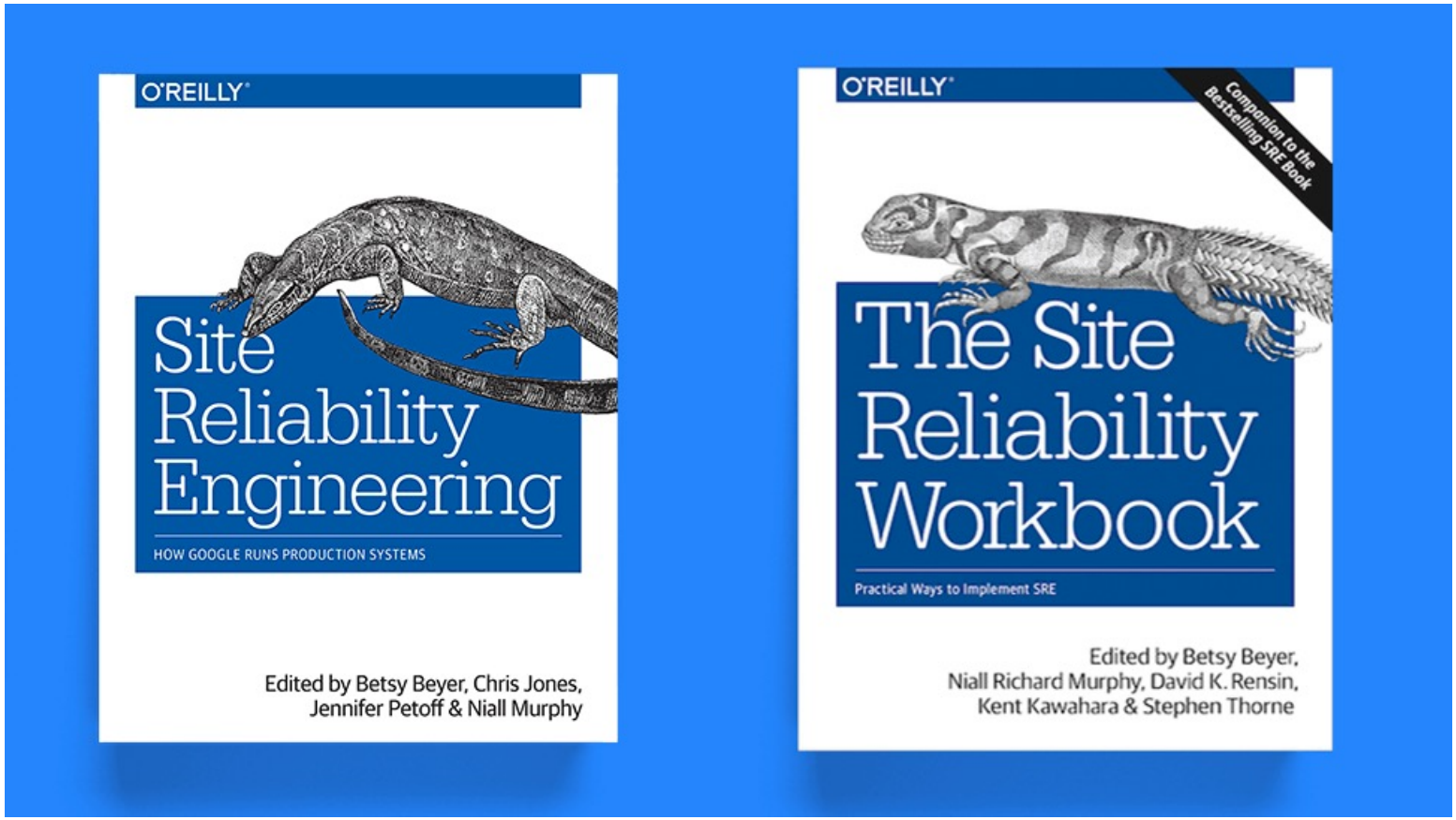
- CPU Time
- Memory Usage
- Operating System Uptime

Metrics must be aggregated overtime (for example “last 5 minutes”) and a function as a percentile to be applied (for example “99 percentile”)

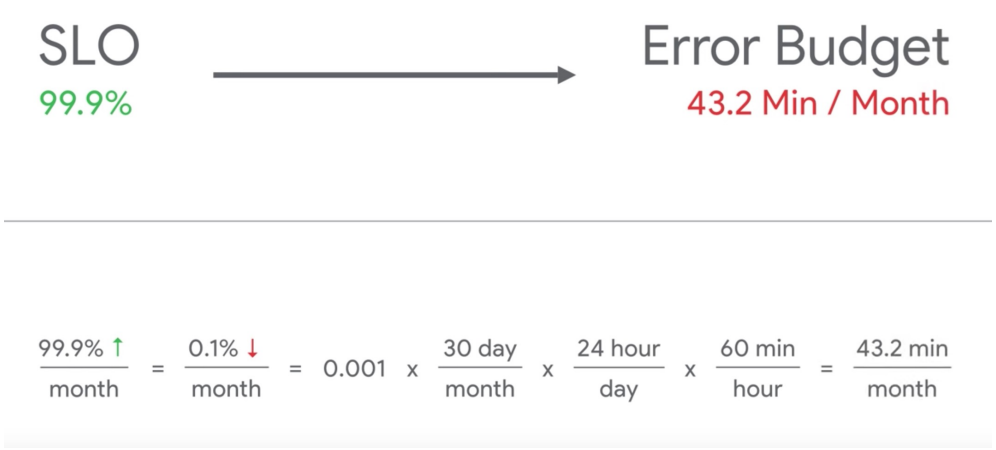
Metrics must be defined in advance and be known and accepted across all organisation

Measurement must be reliable and not only the definition, also the implementation of the measurement must be clearly defined and approved

GOOGLE SRE — CONCEPTS — ERROR BUDGET



ERROR BUDGET AND RISK



Error budget is a quantitative measurement to establish the ratio between work on implementing new features and work on improving stability

“The error budget provides a clear, objective metric that determines how unreliable the service is allowed to be within a single quarter. This metric removes the politics from negotiations between the SREs and the product developers when deciding how much risk to allow”

ERROR BUDGET AND RISK

"If SLO violations occur frequently enough to expend the error budget, releases are temporarily halted while additional resources are invested in system testing and development to make the system more resilient, improve its performance, and so on."

RISK MANAGEMENT AND PRIORITISATION

Calculated Expected Cost

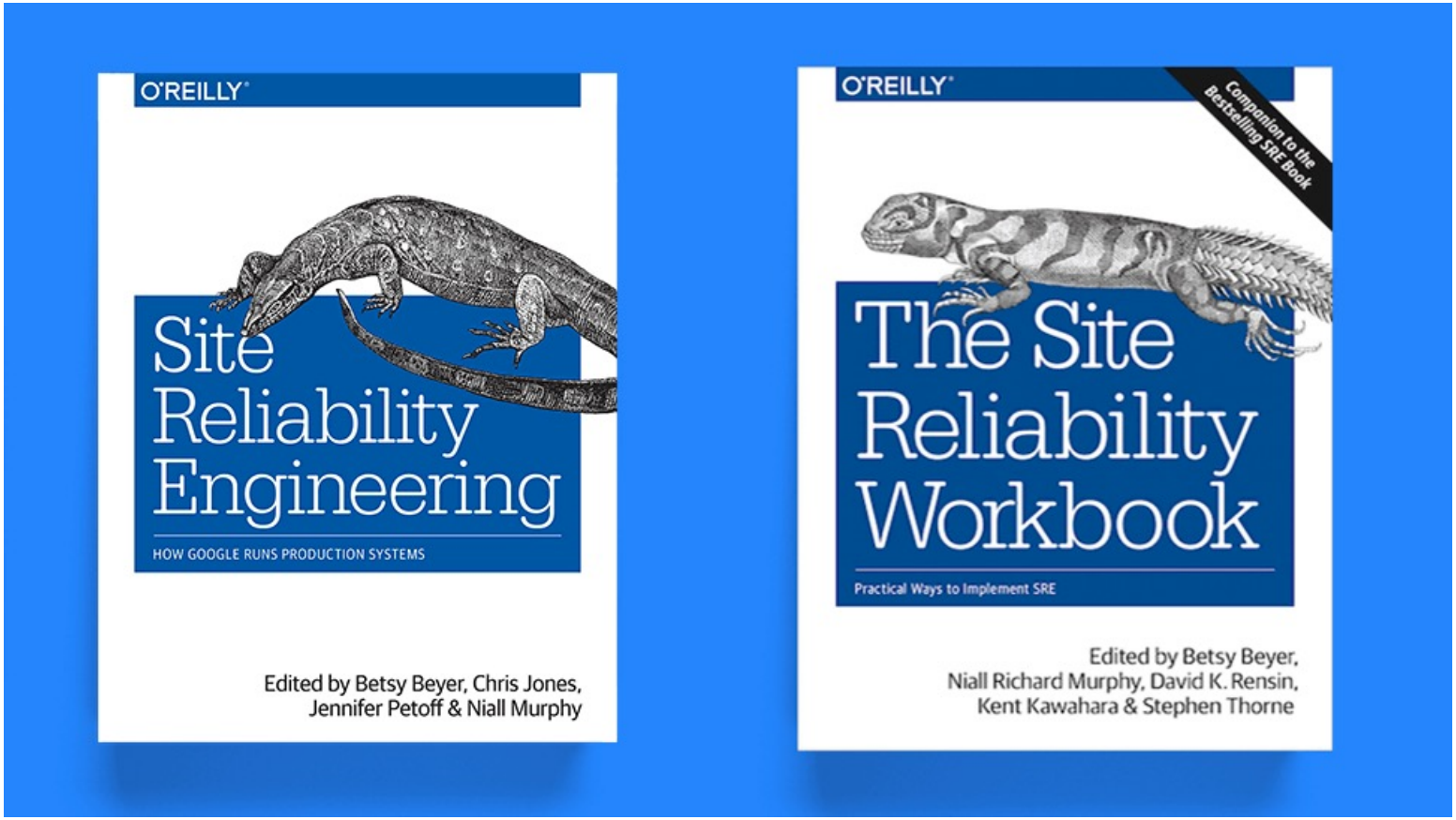
Risk	TTD	TTR	Freq / Yr	Users	Bad / Yr
Production database backup	0	2h	12	100%	**1440m**
Degraded QoS during code push	30m	30m	26	50%	**780m**
Datacenter failure	15m	8h	1	100%	**495m**
DDoS by IoT botnet	2h	5h	6	80%	**2016m**
Bad code deploy	45m	15m	150	25%	**2250m**
Upstream provider failure	5m	30m	6	75%	**157m**

Use measurement to calculate expected cost in terms of error budget of each risk

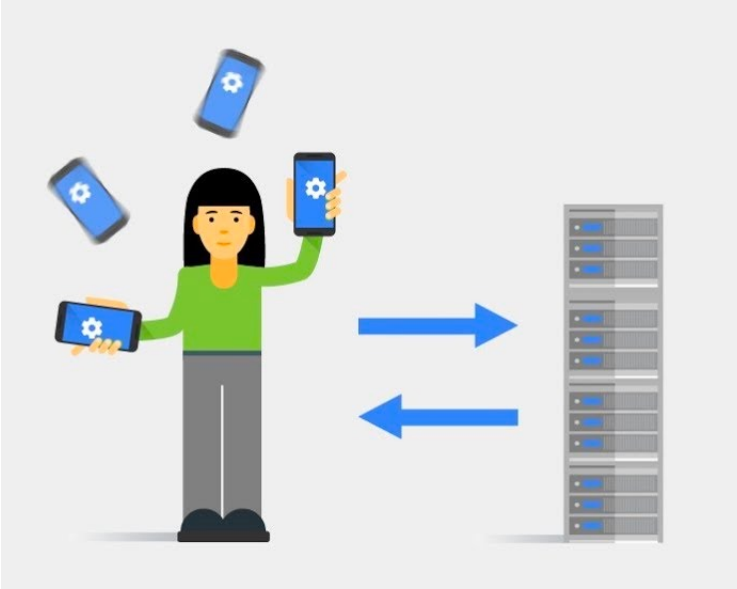
Prioritise work on improvement on the base of the computed impact of each risk in terms of error budget

Cost for mitigating the risks could require a review of the SLO due to business value considerations

GOOGLE SRE — CONCEPTS — TOIL



TOIL AND TOIL BUDGET



Toil is not simply "work I don't like to do."

Toil is not overhead (commuting, expenses reports, meetings, …)

Toil is specifically tied to the running of a production service.

It is work that tends to be **manual**, **repetitive**, **automatable**, **tactical** and devoid of long-term value.

Toil is also reactive, as intervention done due to an alert.

When SREs find tasks that can be automated, they work to engineer a solution to prevent that toil in the future.

Toil is not always bad. Predictable, repetitive tasks are great ways to onboard a new team member and often produce an immediate sense of accomplishment and satisfaction with low risk and low stress.

ON CALL SUPPORT

"We ensure that the teams consistently spending less than 50% of their time on development work change their practices. Often this means shifting some of the operations burden back to the development team, or adding staff to the team without assigning that team additional operational responsibilities."

SREs must not be spending more than 50% of their time on support\operation activities

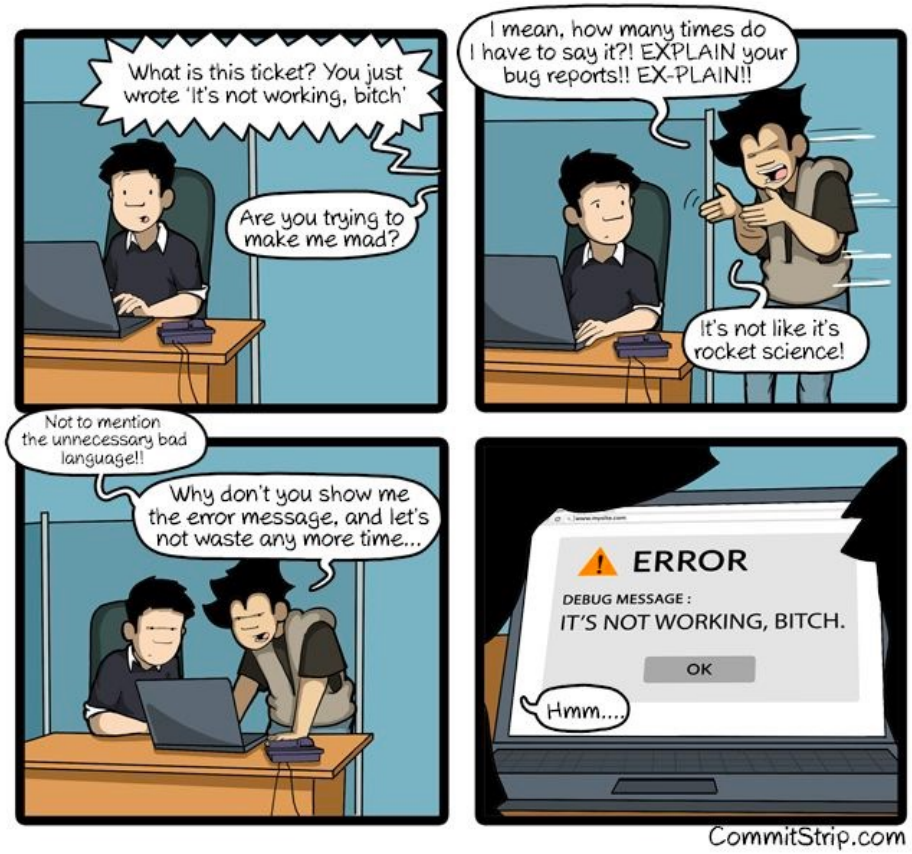
Developers should be involved in support regularly, but their **involvement** become **mandatory if** a product requires **SREs** to **spend more than 50%** of time on operations

At least eight people need to be part of the on-call team to correctly balance the load

Each on call person must handle no more than two events per on-call shift to assure right quality

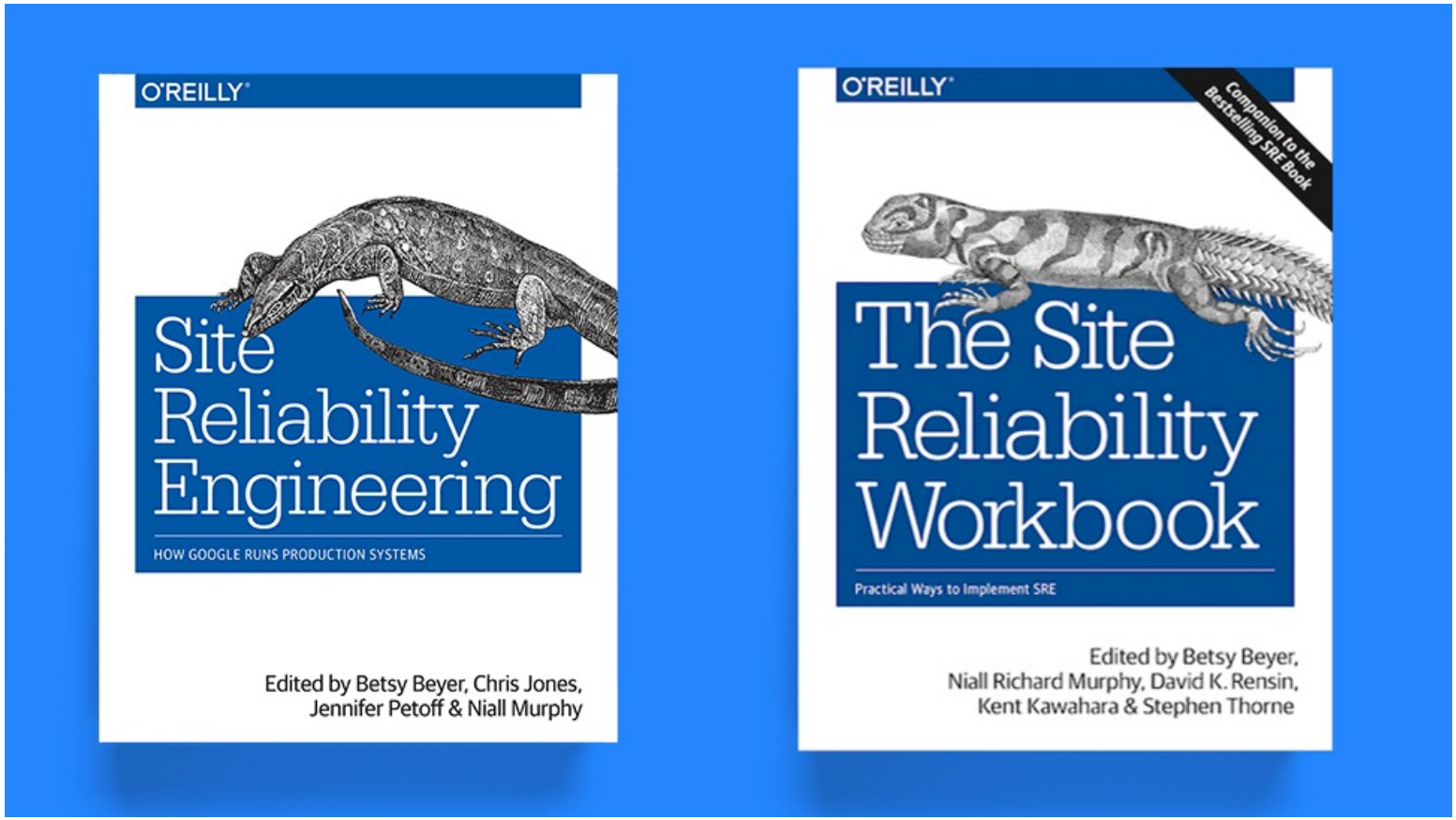
To preserve readiness for an effective on call support **practice handling hypothetical outages**

SUPPORT TICKET



"If you currently assign tickets randomly to victims on your team, stop . Doing so is extremely disrespectful of your team's time, and works completely counter to the principle of not being interruptible as much as possible. Tickets should be a full-time role, for an amount of time that's manageable for a person. If you happen to be in the unenviable position of having more tickets than can be closed by the primary and secondary on-call engineers combined, then structure your ticket rotation to have two people handling tickets at any given time. Don't spread the load across the entire team."

GOOGLE SRE – CONCEPTS – OBSERVABILITY AND MONITORING



WHITE-BOX AND BLACK-BOX MONITORING

Each monitoring system should address two questions: what's broken (symptom) and why (cause)

Two types of monitoring can be defined:

- White-box monitoring: it inspects the internal state of the target service (application components metrics, traces, logs). Focus on causes.
- Black-box monitoring: it accesses the systems from external, as a real user (http\tcp probes, dns resolution, network ping). Symptom-oriented. Active recognition of error condition.

MONITORING AND ALERTING

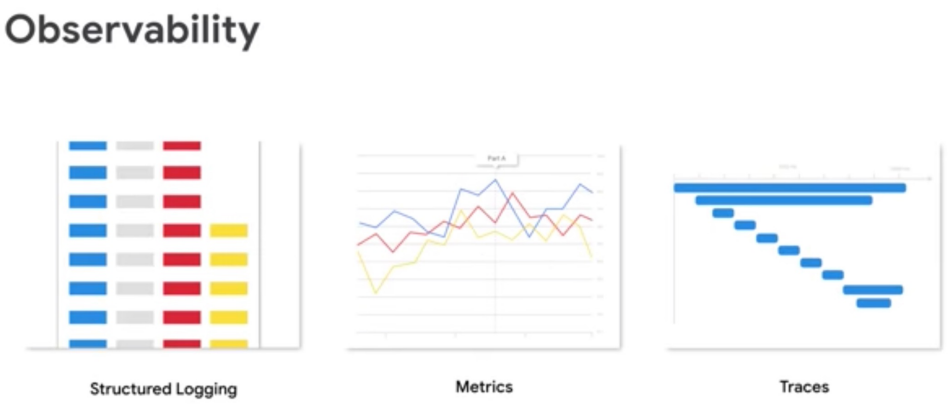
Monitoring may have only three output types:

- Pages - A human must do something now
- Tickets - A human must do something within a few days
- Logging - No one need look at this output immediately, but it's available for later analysis if needed

"Putting alerts into email and hoping that someone will read all of them and notice the important ones is the moral equivalent of piping them to /dev/null : they will eventually be ignored."

An alert must be analysed, if an alert is ignored, remove the alerting rule.

OBSERVABILITY



Collecting metrics and alerting are business driven activities.

Collecting metrics and manage alerting are different thing. You need to collect metrics to have visibility on your systems, but you do not have to use all metrics to generate alerts.

Measurement, monitoring and alerting must be related to SLOs.

Do not alert on anything, alerts must report a service impact and be actionable.

THE FOUR GOLDEN SIGNALS (1/2)

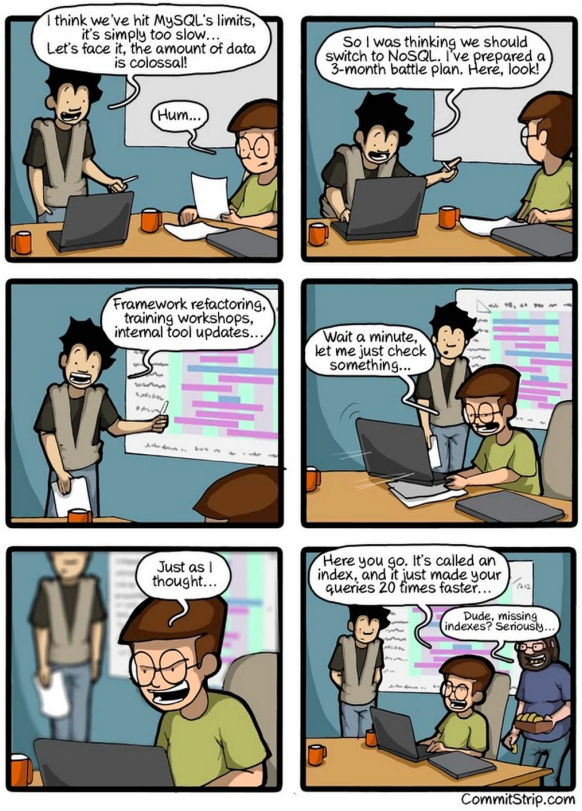
Latency

The time it takes to service a request. It's important to distinguish between the latency of successful requests and the latency of failed requests. For example, an HTTP 500 error triggered due to loss of connection to a database or other critical backend might be served very quickly; however, as an HTTP 500 error indicates a failed request, factoring 500s into your overall latency might result in misleading calculations. On the other hand, a slow error is even worse than a fast error! Therefore, it's important to track error latency, as opposed to just filtering out errors.

Traffic

A measure of how much demand is being placed on your system, measured in a high-level system-specific metric. For a web service, this measurement is usually HTTP requests per second, perhaps broken out by the nature of the requests (e.g., static versus dynamic content). For an audio streaming system, this measurement might focus on network I/O rate or concurrent sessions. For a key-value storage system, this measurement might be transactions and retrievals per second.

THE FOUR GOLDEN SIGNALS (2/2)



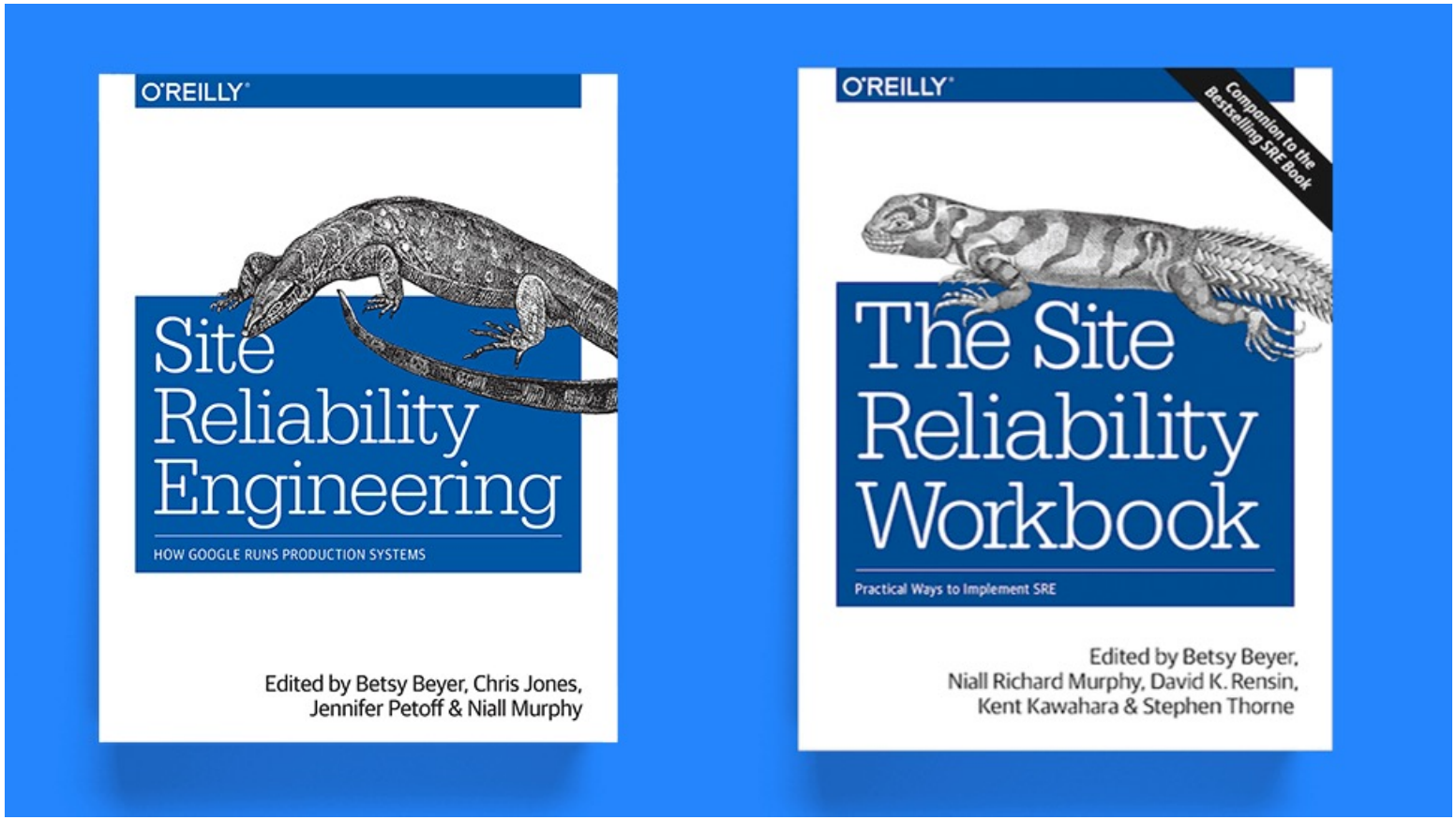
Errors

The rate of requests that fail, either explicitly (e.g., HTTP 500s), implicitly (for example, an HTTP 200 success response, but coupled with the wrong content), or by policy (for example, "If you committed to one-second response times, any request over one second is an error"). Where protocol response codes are insufficient to express all failure conditions, secondary (internal) protocols may be necessary to track partial failure modes. Monitoring these cases can be drastically different: catching HTTP 500s at your load balancer can do a decent job of catching all completely failed requests, while only end-to-end system tests can detect that you're serving the wrong content.

Saturation

How "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained (e.g., in a memory-constrained system, show memory; in an I/O-constrained system, show I/O). Note that many systems degrade in performance before they achieve 100% utilization, so having a utilization target is essential.

GOOGLE SRE — CONCEPTS — INCIDENT MANAGEMENT



INCIDENT MANAGEMENT

Effective incident management is key to limiting the disruption caused by an incident and restoring normal business operations as quickly as possible.

A well-designed incident management process has the following features:

- Recursive separation of responsibilities
  - Incident command
  - Operational work
  - Communication
  - Planning
- A recognised command post
- Live incident state document
- Clear handoff

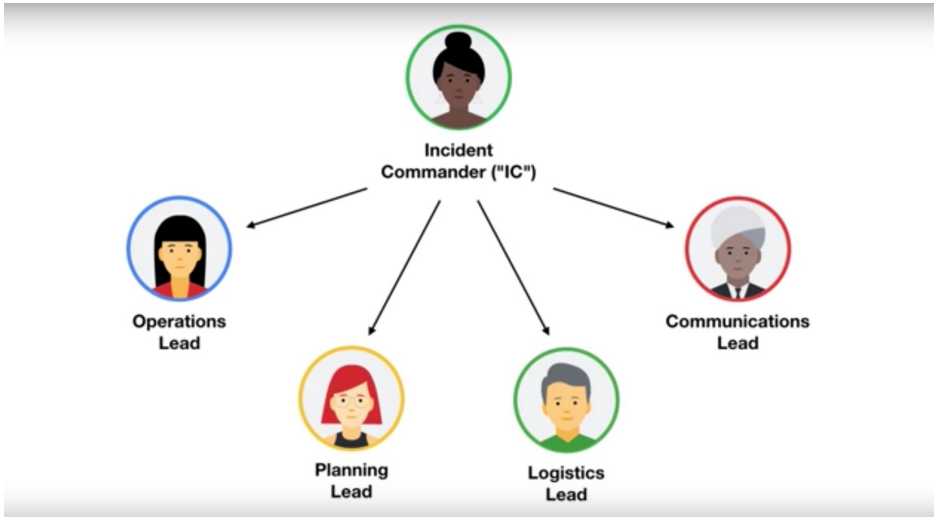
WHEN TO DECLARE AN INCIDENT

”It is better to declare an incident early and then find a simple fix and close out the incident than to have to spin up the incident management framework hours into a burgeoning problem.”

If any of the following is true, the event is an incident:

- Do you need to involve a second team in fixing the problem?
- Is the outage visible to customers?
- Is the issue unsolved even after an hour’s concentrated analysis?.

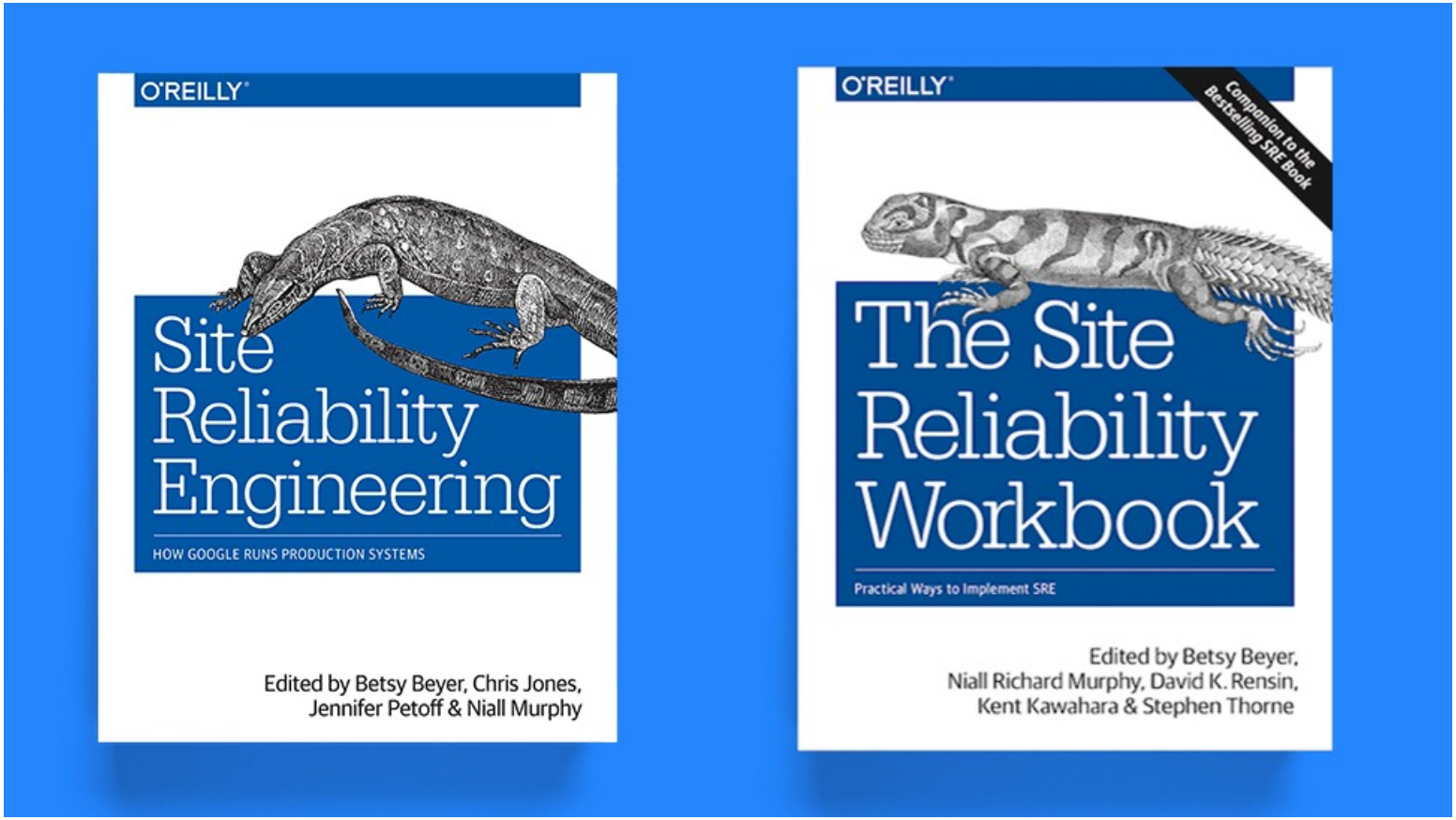
INCIDENT MANAGEMENT



Incident Management Roles:

- Incident Commander
    - The incident commander holds the high-level state about the incident. They structure the incident response task force, assigning responsibilities according to need and priority.
- Operations lead
    - The Ops lead works with the incident commander to respond to the incident by applying operational tools to the task at hand. The operations team should be the only group modifying the system during an incident.
- Communication lead
    - This person is the public face of the incident response task force.
- Planning lead
    - The planning role supports Ops by dealing with longer-term issues, such as filing bugs, arranging handoffs, and tracking how the system has diverged from the norm so it can be reverted once the incident is resolved.
- Logistics lead
    - The logistic role supports Ops by dealing with things as ordering dinner or dealing with vendors for spare parts.

GOOGLE SRE — CONCEPTS — POSTMORTEMS



POSTMORTEMS AND RETROSPECTIVES

”Postmortems should be blameless and focus on process and technology, not people. Assume the people involved in an incident are intelligent, are well intentioned, and were making the best choices they could given the information they had available at the time.”

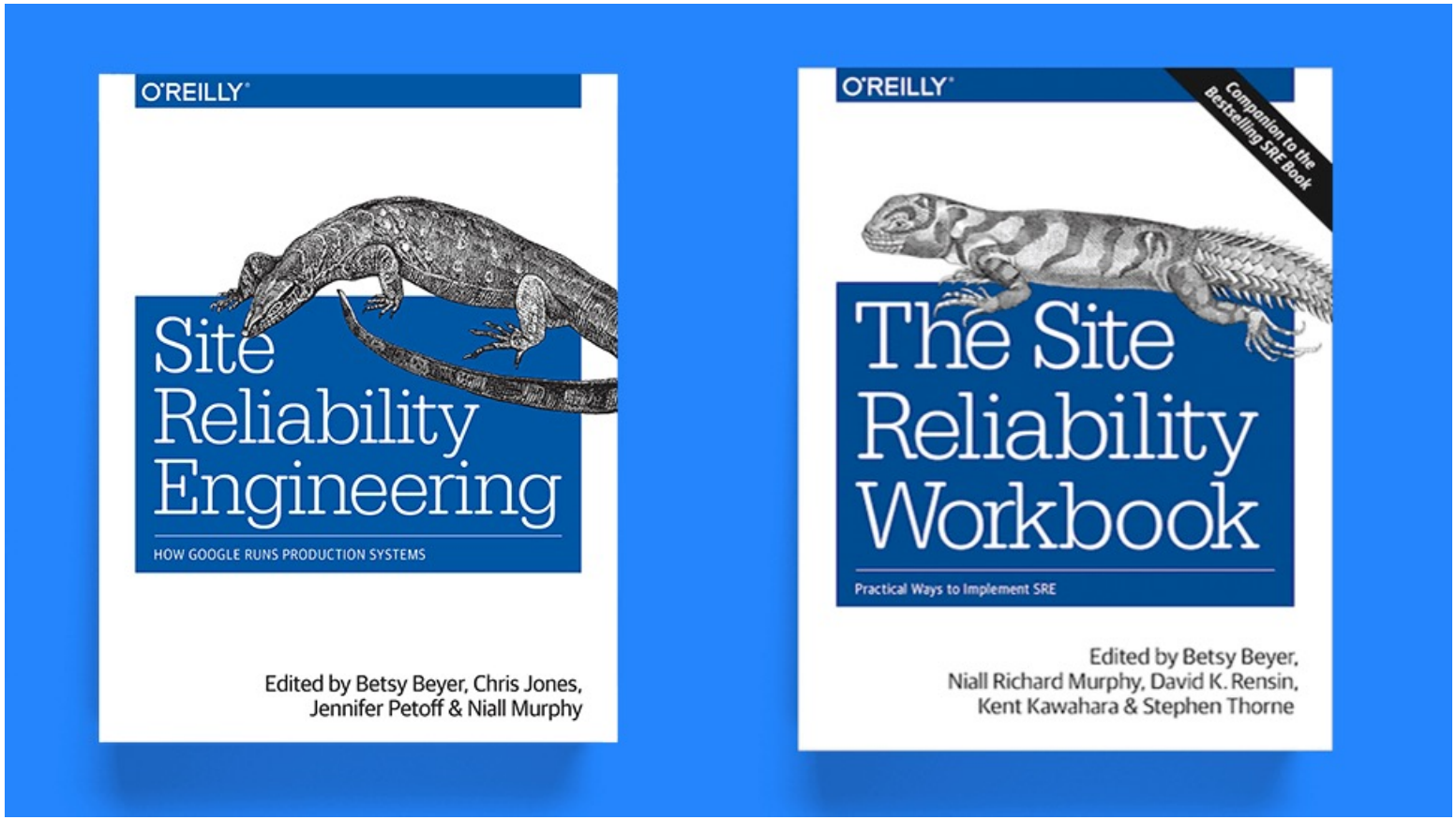
Questions about which data where available, how the system was behaving, which actions done and their effect. Avoid questions about why an action has been done or why not.

POSTMORTEMS AND RETROSPECTIVES

Content of a postmortem:

- Incident summary
- Detailed timeline
- Detection
- Impact
- Root Causes
- Triggers
- Mitigation and Resolution
- Lesson learned
  - What went well
  - What went wrong
  - Where we got lucky
- Action Items (with clear owner)

GOOGLE SRE — CONCEPTS — ENGAGEMENT MODEL



SRE ENGAGEMENT MODEL

Not all Google services receive close SRE engagement.

Google defines three different engagement models

- Simple PRR (Product Readiness Reviews) Model
- Early Engagement Model
- Frameworks and SRE Platform

SIMPLE PRR MODE

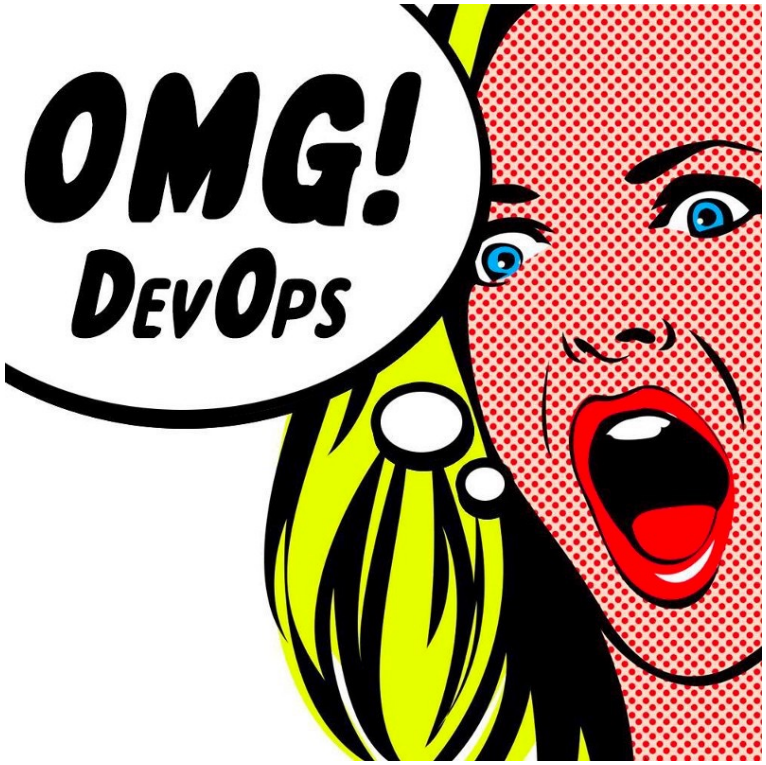
Development team requests that SRE take over production management of a service, one to three SREs are selected to conduct the PRR process.

The discussion covers matters such as:

- Establishing an SLO/SLA for the service
- Planning for potentially disruptive design changes required to improve reliability
- Planning and training schedules

The Training phase unblocks onboarding of the service by the SRE team. It involves a progressive transfer of responsibilities and ownership of various production aspects of the service, including parts of operations, the change management process, access rights, and so forth. To complete the transition, the development team must be available to back up and advise the SRE team for a period of time as it settles in managing production for the service. This relationship becomes the basis for the ongoing work between the teams.

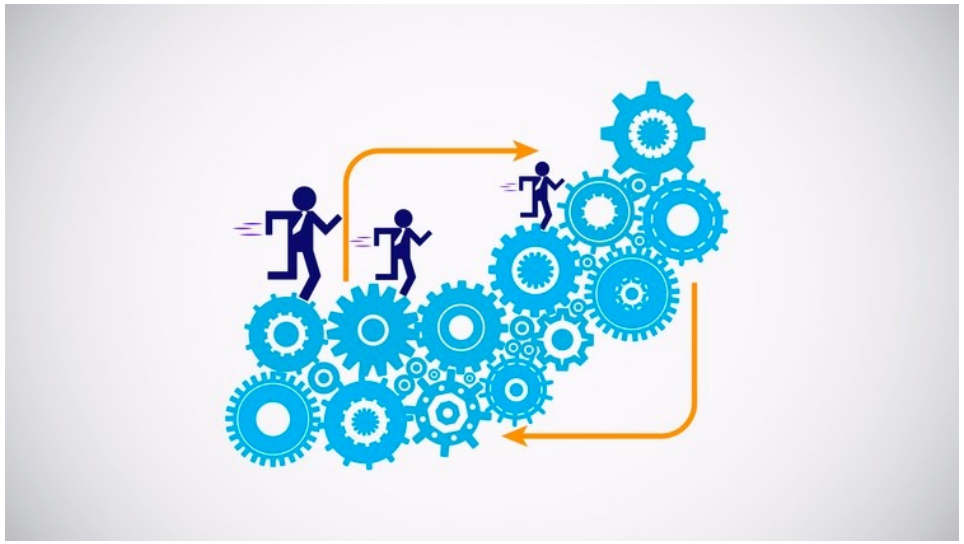
EARLY ENGAGEMENT MODEL



The Early Engagement Model introduces SRE earlier in the development lifecycle.

SRE participates in Design and later phases, eventually taking over the service any time during or after the Build phase.

This model is based on active collaboration between the development and SRE teams.



FRAMEWORKS AND SRE PLATFORM

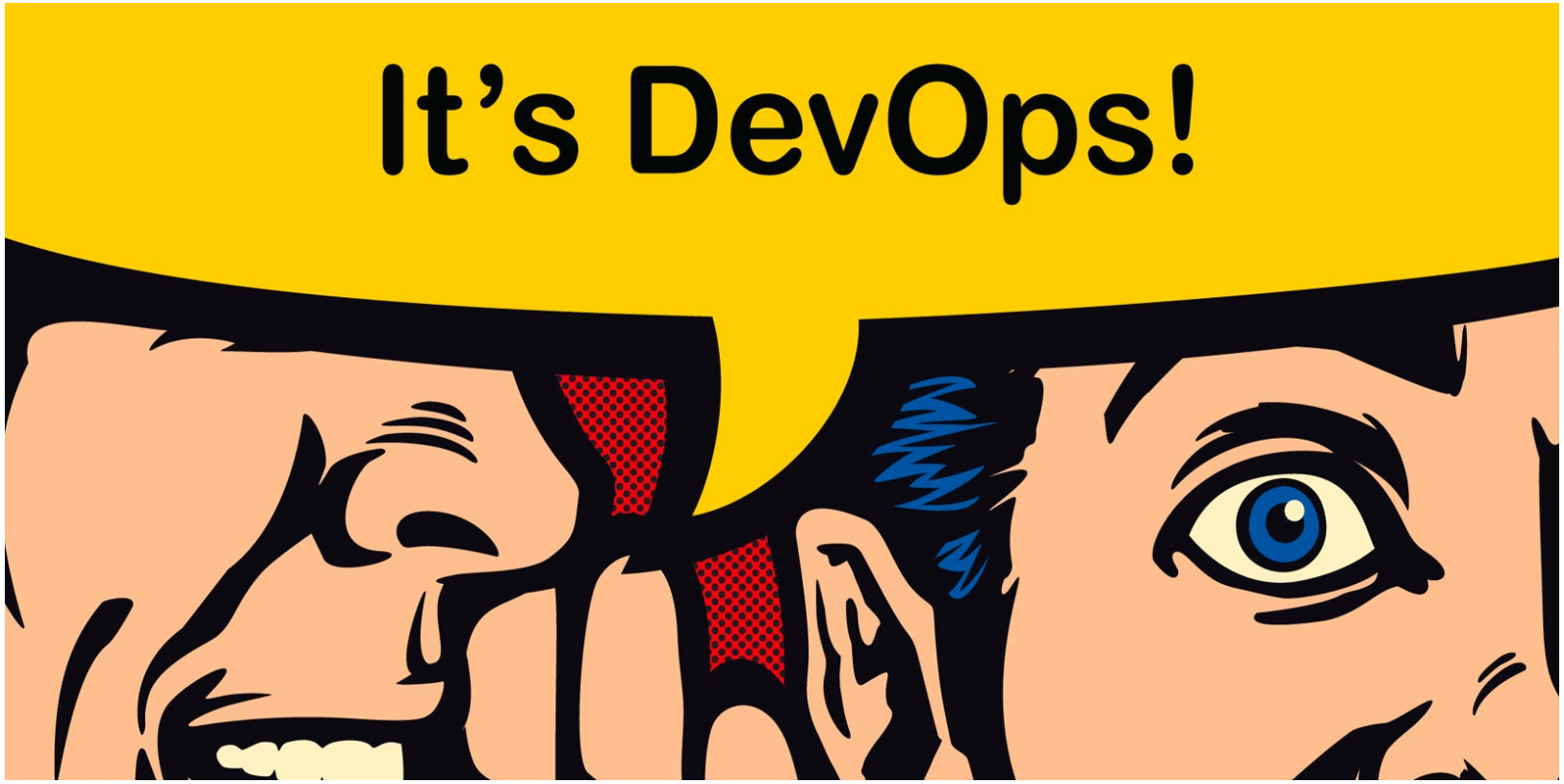
SRE builds framework modules to implement canonical solutions for the concerned production area. As a result, development teams can focus on the business logic, because the framework already takes care of correct infrastructure use. A framework essentially is a prescriptive implementation for using a set of software components and a canonical way of combining these components.

The service frameworks implement infrastructure code in a standardized fashion and address various production concerns. Each concern is encapsulated in one or more framework modules, each of which provides a cohesive solution for a problem domain or infrastructure dependency.

Framework modules address the various SRE concerns enumerated earlier, such as:

- Instrumentation and metrics
- Request logging
- Control systems involving traffic and load management

AM I DOING SRE OR DEVOPS?



AM I DOING SRE OR DEVOPS?

How can I know if I'm doing DevOps?

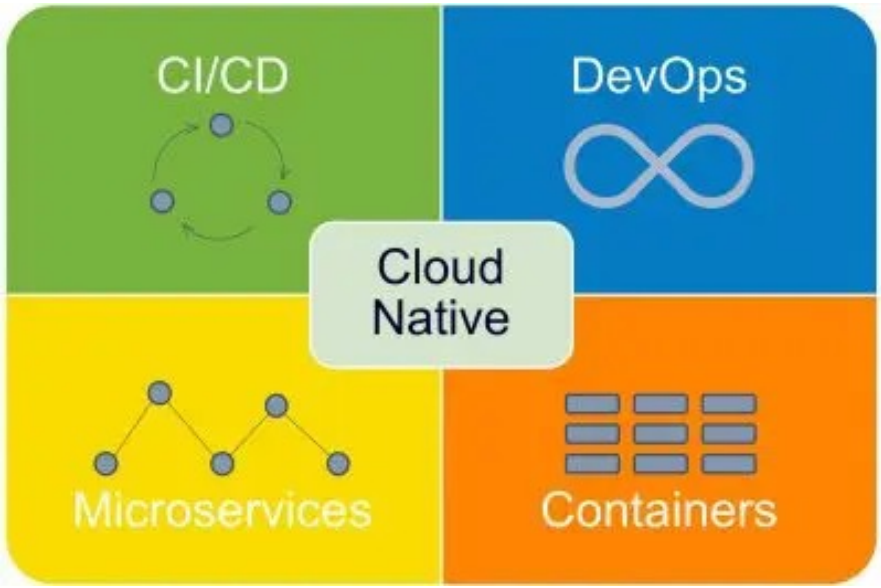
- Do I consider Developers, Test Engineers and Sys Admins as members of the same team having one common goal?
- Do I know and care about end users features and functionalities (business value)?
- Do I automate infrastructure and code deployment? Do I understand and manage SDLC?
- Do I focus on collecting as much metrics as possible from build process to production runtime?

How can I know if I'm doing SRE?

- Do I protect my 50% of the time for engineering activities?
- Do I define SLO and manage Error Budget?
- Do I continuously work to improve self-healing of production solutions? Do I target "autonomous" solutions and not only "automated"?
- Do I use a structured Incident Management process with clearly defined roles?

CLOUD NATIVE ARCHITECTURE

The principle of architecting for the cloud, a.k.a. cloud-native architecture, focuses on how to optimize system architectures for the unique capabilities of the cloud. Traditional architecture tends to optimize for a fixed, high-cost infrastructure, which requires considerable manual effort to modify. Traditional architecture therefore focuses on the resilience and performance of a relatively small fixed number of components. In the cloud however, such a fixed infrastructure makes much less sense because cloud is charged based on usage (so you save money when you can reduce your footprint) and it's also much easier to automate (so automatically scaling-up and down is much easier). Therefore, cloud-native architecture focuses on achieving resilience and scale though horizontal scaling, distributed processing, and automating the replacement of failed components.



DESIGN FOR AUTOMATION

Infrastructure: Automate the creation of the infrastructure, together with updates to it, using tools Terraform

Continuous Integration/Continuous Delivery: Automate the build, testing, and deployment of the packages that make up the system by using tools like Jenkins or Spinnaker.

Scale up and scale down: Unless your system load almost never changes, you should automate the scale up of the system in response to increases in load, and scale down in response to sustained drops in load

Monitoring and automated recovery: You should bake monitoring and logging into your cloud-native systems from inception.

BE SMART WITH STATE

Storing of 'state', be that user data (e.g., the items in the users shopping cart, or their employee number) or system state (e.g., how many instances of a job are running, what version of code is running in production), is the hardest aspect of architecting a distributed, cloud-native architecture. You should therefore architect your system to be intentional about when, and how, you store state, and design components to be stateless wherever you can.

Stateless components are easy to:

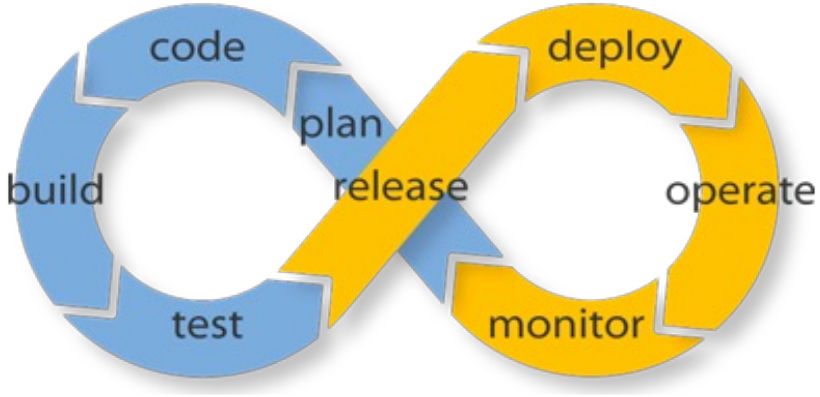
Scale: To scale up, just add more copies. To scale down, instruct instances to terminate once they have completed their current task.

Repair: To 'repair' a failed instance of a component, simply terminate it as gracefully as possible and spin up a replacement.

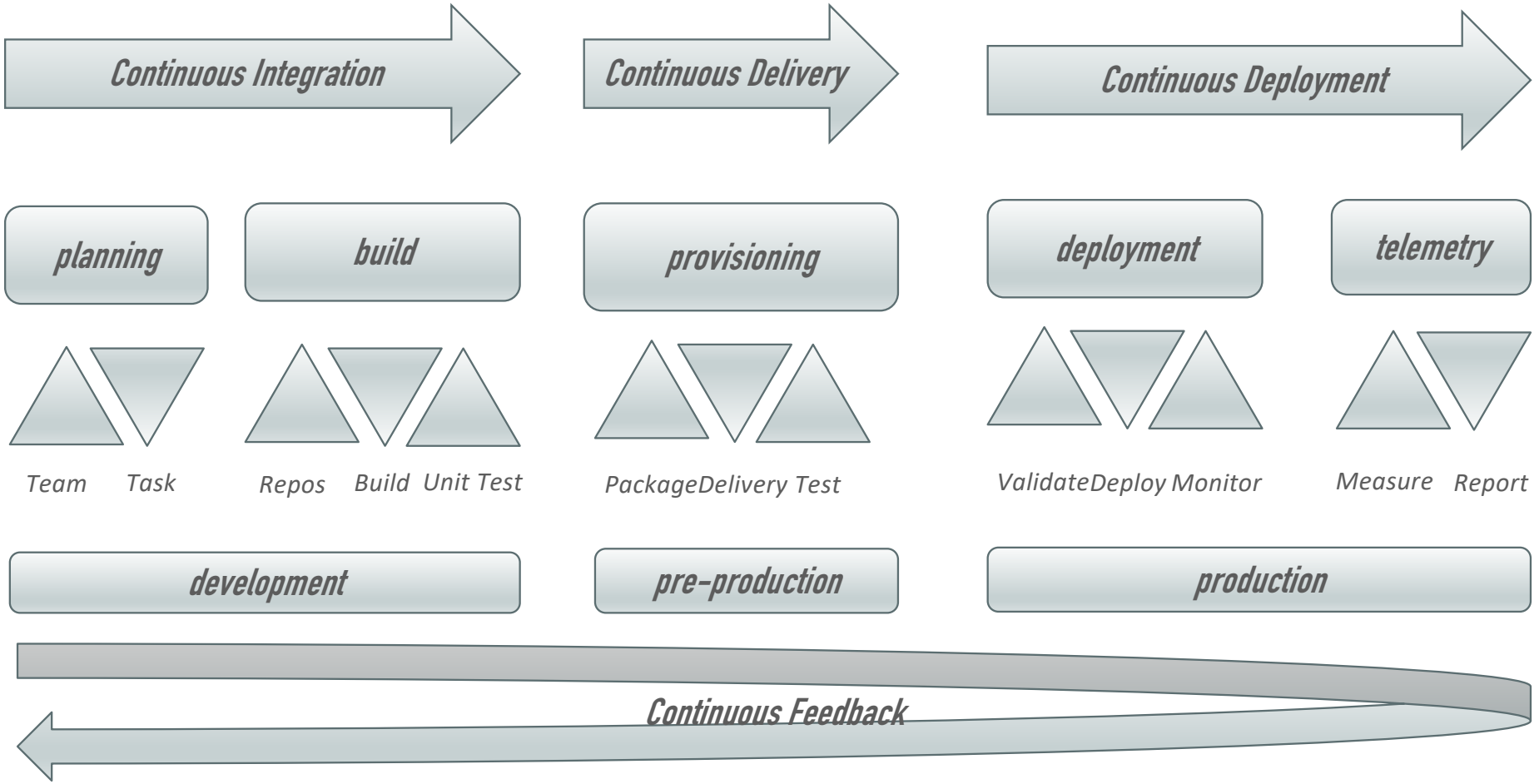
Roll-back: If you have a bad deployment, stateless components are much easier to roll back

Load-Balance across: When components are stateless, load balancing is much simpler since any instance can handle any request.

CONTINUOUS INTEGRATION,DELIVERY



CONTINUOUS DELIVERY VS CONTINUOUS DEPLOYMENT



AUTOMATED – REPEATABLE - RELIABLE

- One artifact for all environments
  - Don’t create different builds for different environments
- One process for all environments
  - Don’t create different pipelines for different environments
- Fix and don't get around
  - If a step fails it must be corrected and never circumvented
- Nothing outiside of the repository
  - The repository is the only complete source of truth
- The more complex it is, the more frequently it needs to be addressed
  - Complex tasks become complicated when done infrequently

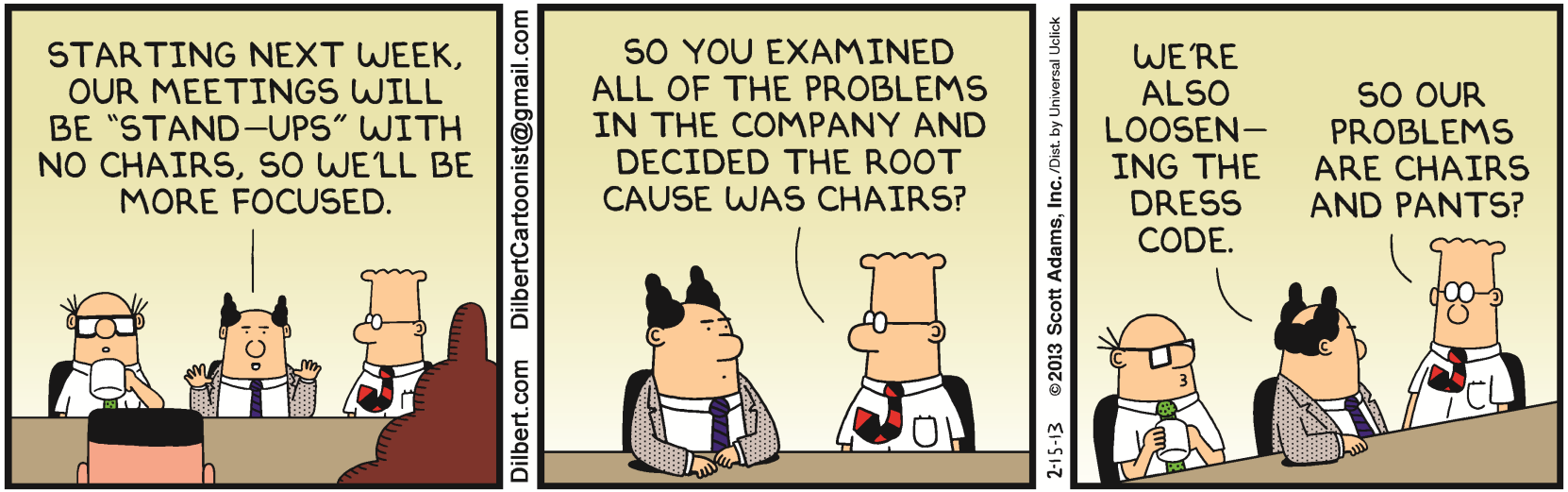
INFRASTRUCTURE AS CODE

- From Wikipedia: “Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.”
- Imperative vs Declarative approach: for declarative you specify a list of resources you would like, whereas for imperative you specify a list of commands to run to create the resources that you want
- Why Declarative:
  - Idempotency — idempotency is the ability to run the same command and achieve the same result.
  - State Management - a declarative tool needs to manage the current state. It’s entirely possible that the declarative infrastructure as code’s picture of the current state actually can differ from the reality of our state.
  - Dealing with “Configuration Drift” — Configuration drift is when infrastructure changes slowly over time. Declarative infrastructure as code will be able to adapt more easily to changes, reporting the differences and leaving it up to you to decide how to proceed.

PET VS CATTLE

- PET model
  - Management of each individual element of your infrastructure
  - Goals: each server part of your infrastructure should be available 24/7
- CATTLE model
  - Manager of the overall infrastructure considering each element as disposable
  - Goals: solution\service uptime, considering that single elements will go down and will be destroyed and replaced

HOW DO WE MANAGE OUR WORK



PROJECTS – SCRUM METHODOLOGY

- Project Specific Backlog and Sprint Board
- User Stories with tags and status
- Explicit Acceptance Criteria
- Explicit and transparent evaluation
- Evaluation by complexity
- Scheduled ceremonies with customer:
  - Backlog Refinement
  - Sprint Planning
  - Sprint Review

SOME REAL EXAMPLE OF USER STORIES

Description

Come tecnico del progetto desidero rimuovere la necessità di mantenere file cmd all'interno della soluzione, sostituendoli, ove possibile, con comandi nativi in pipeline o rimuovendoli se non più necessari dopo il passaggio alle pipeline Gitlab, in modo da ridurre le attività e le complessità di manutenzione.

Rimozione utilizzo SetDevOpsEnvironment.cmd (in primis), valutare gli altri

Acceptance Criteria

L'esecuzione di SetDevOpsEnvironment.cmd non è più presente nella pipeline
L'esecuzione della pipeline porta allo stesso risultato precedentemente ottenuto
quando SetDevOpsEnvironment.cmd era presente
Esiste una lista di file cmd rimossi e sostituiti in pipeline mantenendo il corretto funzionamento.

Planning

Story Points
5

Priority
2

Risk

Classification

Value area
Business

SOME REAL EXAMPLE OF USER STORIES

Description

Come tecnico voglio che all'esecuzione di una pipeline di release venga realizzato un documento, in formato markdown, di release note, in modo che il team di QA possa partire da un modello per la realizzazione delle release notes finali

Acceptance Criteria

- esiste un job di pipeline che utilizzando i riferimenti a Redmine, raccoglie i ticket che fanno parte di una specifica release
- il job viene eseguito solo quando viene rilasciata una nuova versione
- lo stato del job (successo o fallimento) ed eventuali log di esecuzione sono disponibili internamente alla storia della pipeline
- all'esecuzione del job viene realizzato un file in formato markdown che riporta la versione e la lista dei ticket che ne fanno parte
- il markdown è disponibile come artefatto scaricabile anche in momenti successivi all'esecuzione della pipeline

Planning

Story Points
3

Priority
2

Risk

Classification

Value area
Business

SOME REAL EXAMPLE OF USER STORIES

Description

Come tecnico del progetto desidero che il modulo CARF sia attivato nel sistema per poter verificare il flusso di dati.

Acceptance Criteria

- Esiste un test ripetibile che verifica il corretto passaggio di dati in entrata e in uscita dal modulo CARF
- il CARF dovrà essere in grado di ricevere messaggi in input e correttamente inviarli sulle code del bus
- i messaggi inviati saranno letti dai servizi interessati

Dato uno scenario di dati in ingresso (fisso), il test dovrà essere in grado di scatenare i flussi necessari a SIM Handler e MEC Electronic Horizon (salvataggio fleet vehicles e abilitazione servizio re-routing) per poter avere un output predicibile.

Details

Priority
2

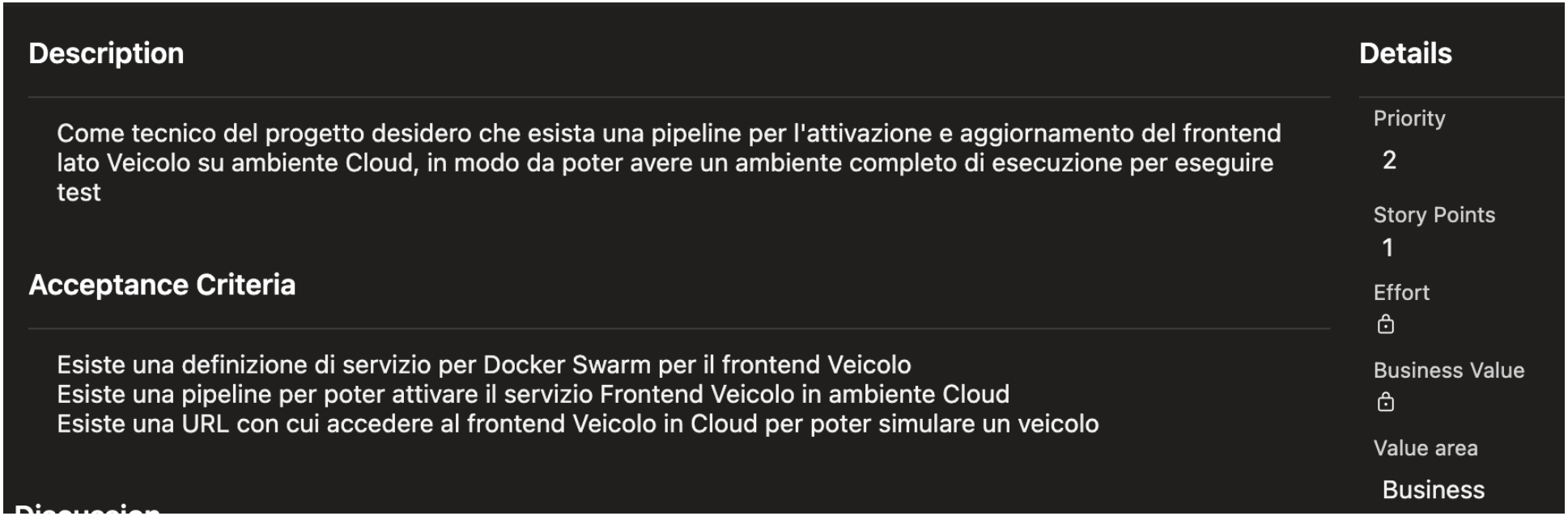
Story Points
5

Effort

Business Value

Value area
Business

SOME REAL EXAMPLE OF USER STORIES



Description

Come tecnico del progetto desidero che esista una pipeline per l'attivazione e aggiornamento del frontend lato Veicolo su ambiente Cloud, in modo da poter avere un ambiente completo di esecuzione per eseguire test

Acceptance Criteria

Esiste una definizione di servizio per Docker Swarm per il frontend Veicolo
Esiste una pipeline per poter attivare il servizio Frontend Veicolo in ambiente Cloud
Esiste una URL con cui accedere al frontend Veicolo in Cloud per poter simulare un veicolo

Details

Priority
2

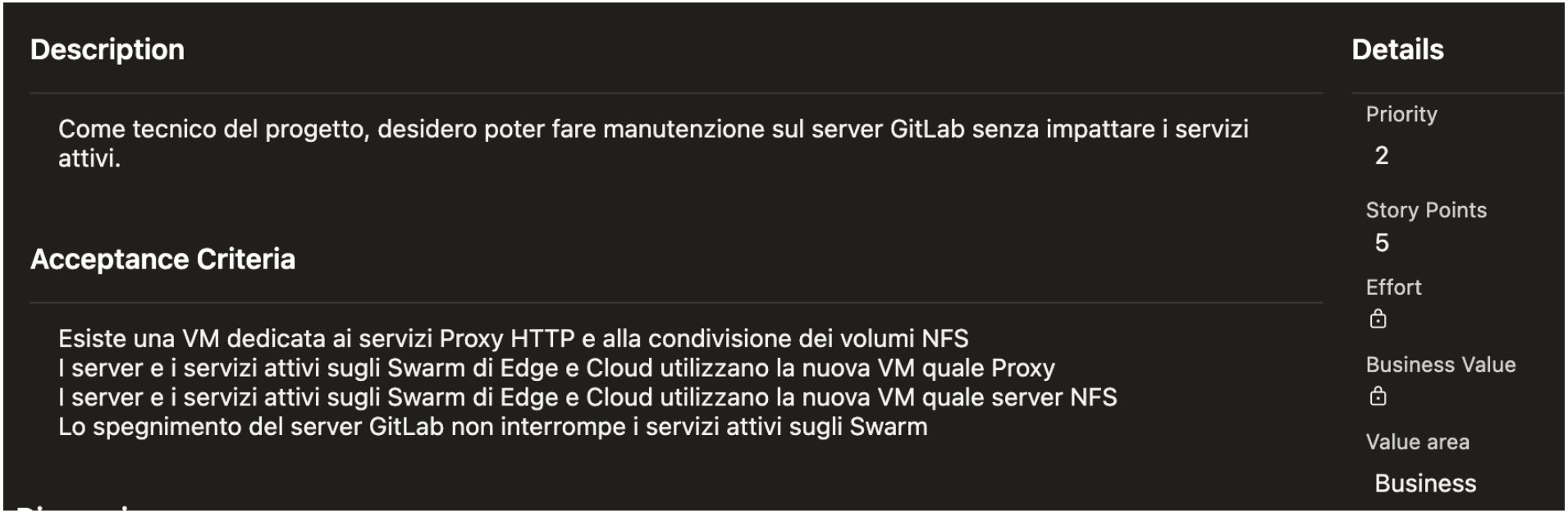
Story Points
1

Effort

Business Value

Value area
Business

SOME REAL EXAMPLE OF USER STORIES



Description

Come tecnico del progetto, desidero poter fare manutenzione sul server GitLab senza impattare i servizi attivi.

Acceptance Criteria

Esiste una VM dedicata ai servizi Proxy HTTP e alla condivisione dei volumi NFS
I server e i servizi attivi sugli Swarm di Edge e Cloud utilizzano la nuova VM quale Proxy
I server e i servizi attivi sugli Swarm di Edge e Cloud utilizzano la nuova VM quale server NFS
Lo spegnimento del server GitLab non interrompe i servizi attivi sugli Swarm

Details

Priority
2

Story Points
5

Effort

Business Value

Value area
Business

SOME REAL EXAMPLE OF USER STORIES

Description

Come release manager desidero poter eseguire la pipeline di generazione setup avendo sia un setup predefinito che la possibilità di definire gruppi di setup diversi per la specifica esecuzione.
Esistono una o più variabili per la definizione dei setup da realizzare.
Partiamo dall'integrazione del POC dei "child job" (
2349 **[POC] Pipeline downstream per selezione configurazioni** | Closed)

Acceptance Criteria

La pipeline può essere esportata da ambiente POC e importata in ambiente produzione senza richiedere modifiche per la sua esecuzione inclusiva di step di setup (al netto di eventuali valori da modificare in variabili di progetto).
I test di leapwork partono dopo il setup di default e non prima (qui vedere per "convenzione" accesso file di setup per il tool di test)
Aggancio alla creazione file per redmine (vedasi PBI relativo)
Variabili utilizzate sono documentate nella pagina del wiki relativa alle variabili
La pipeline viene eseguita in automatico nelle condizioni predefinite (merge su branch di destinazione definito)
Esiste la possibilità di lanciare la pipeline (in automatico) con generazione dei setup predefiniti
Esiste la possibilità di lanciare la pipeline (manuale) indicando esplicitamente quali setup opzionali eseguire
Esiste la possibilità di lanciare la sola generazione dei setup a partire da una build specifica, con criteri di setup diversi, più volte.
Il risultato dei setup eseguiti è salvato come artefatto
Far partire leapwork per ogni singolo setup aggiuntivo è una opzione non obbligatoria (nice to have).

Planning

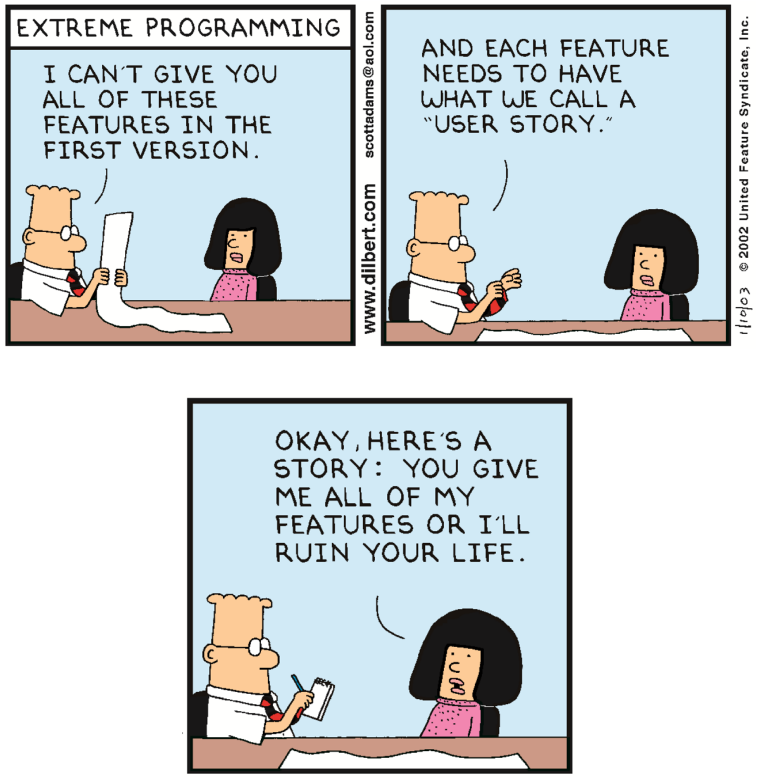
Story Points
5
Priority
2
Risk

Classification

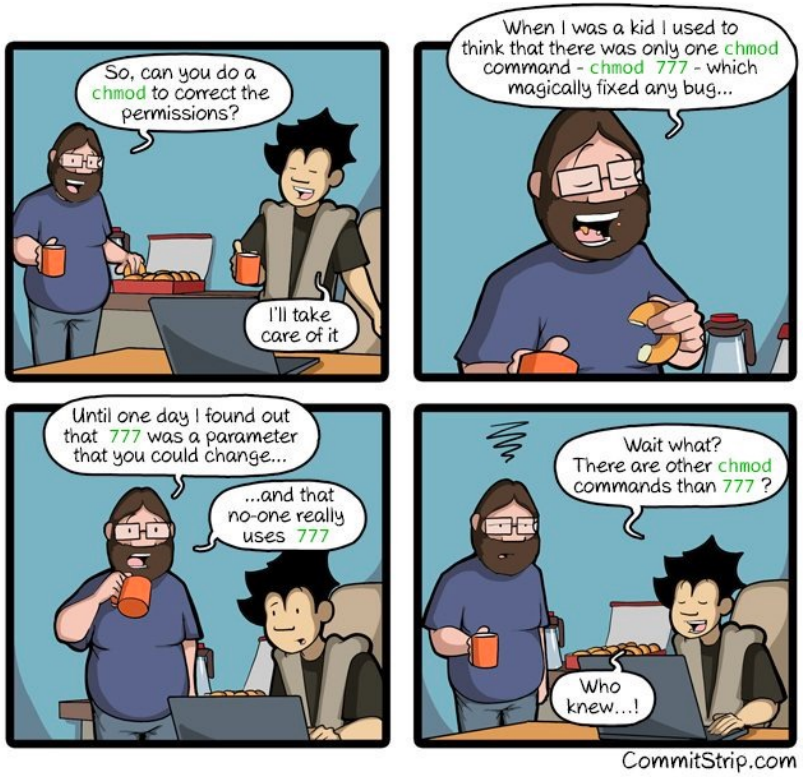
Value area
Business

COMPLEXITIES

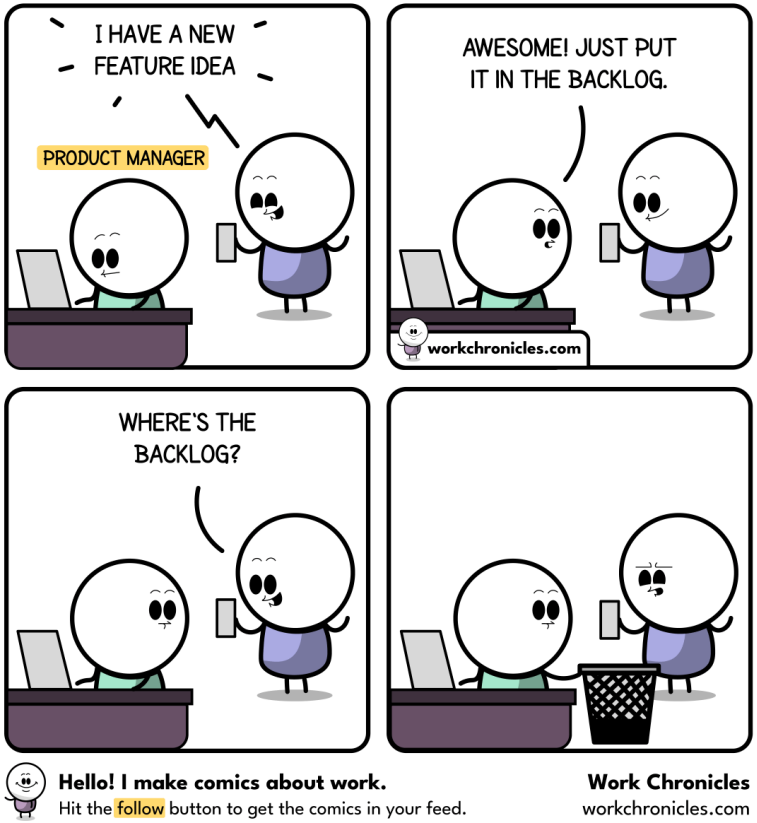
- Same Team - Multiple Projects and Customers
- Scrum Projects vs Kanban Maintenance
- People Availability vs Skills Availability

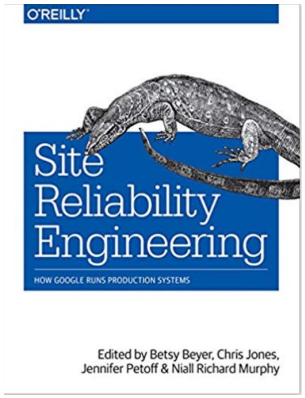


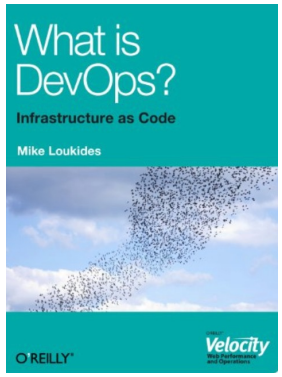
THE END – Q&A ?

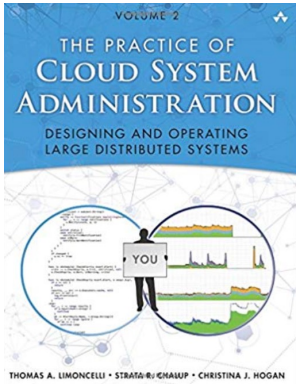

So, can you do a chmod to correct the permissions?
I'll take care of it
When I was a kid I used to think that there was only one chmod command - chmod 777 - which magically fixed any bug...
Until one day I found out that 777 was a parameter that you could change...
...and that no-one really uses 777
Wait what? There are other chmod commands than 777 ?
Who knew...!
CommitStrip.com

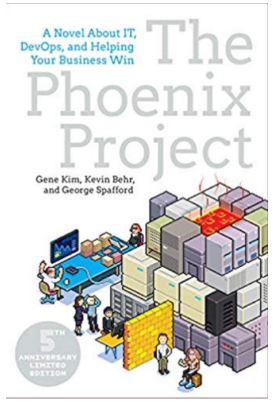
SOME ADDITIONAL LINK

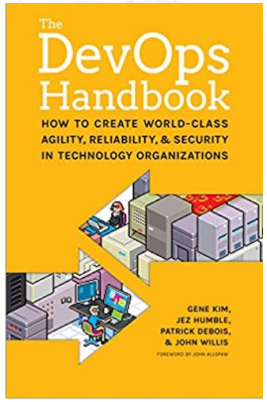


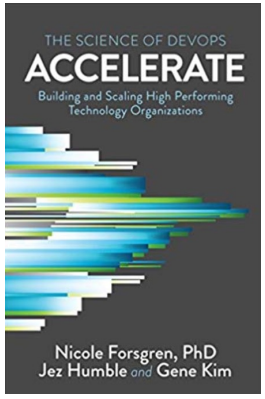


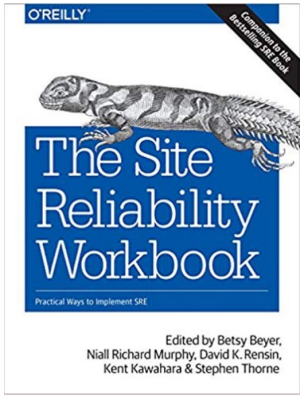


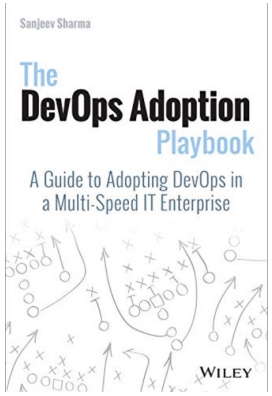






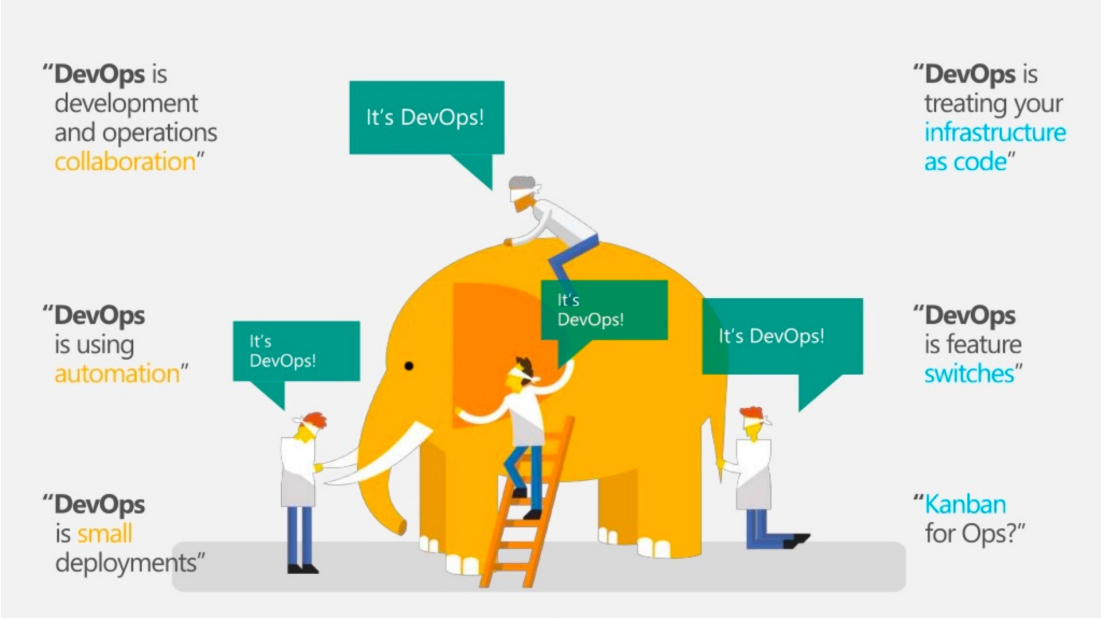






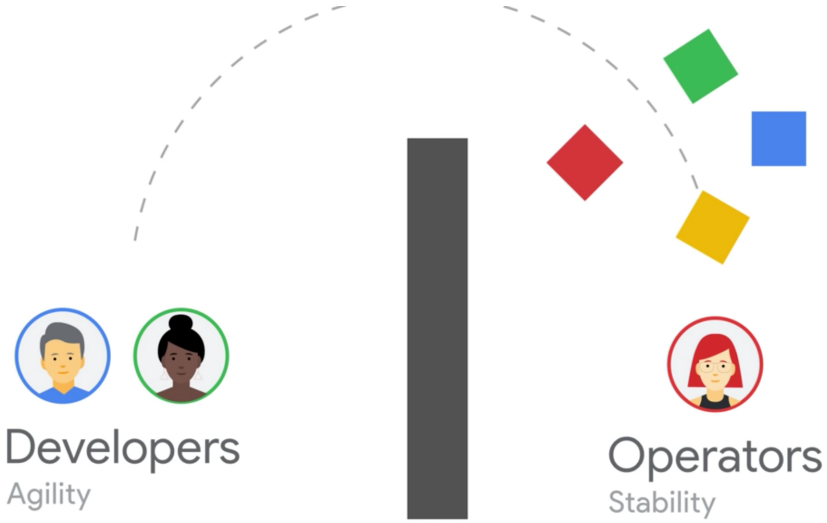
REFERENCES – BOOKS

- The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win (Gene Kim , Kevin Behr, George Spafford)
- The Devops Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations (Gene Kim, Jez Humble, Patrick Debois, John Willis, John Allspaw)
- Site Reliability Engineering: How Google Runs Production Systems (Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy)
- The Site Reliability: Practical Ways to Implement SRE (Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara, Stephen Thorne)
- The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems, Volume 2 (Thomas A. Limoncelli, Strata R. Chalup, Christina J. Hogan)
- Release It! Design and Deploy Production-Ready Software (Michael T. Nygard)
- Accelerate: The Science Behind Devops: Building and Scaling High Performing Technology Organizations (Nicole Forsgren, Jez Humble, Gene Kim)
- What is DevOps? (Mike Loukides)
- The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise (Sanjeev Sharma)



REFERENCES – WEB

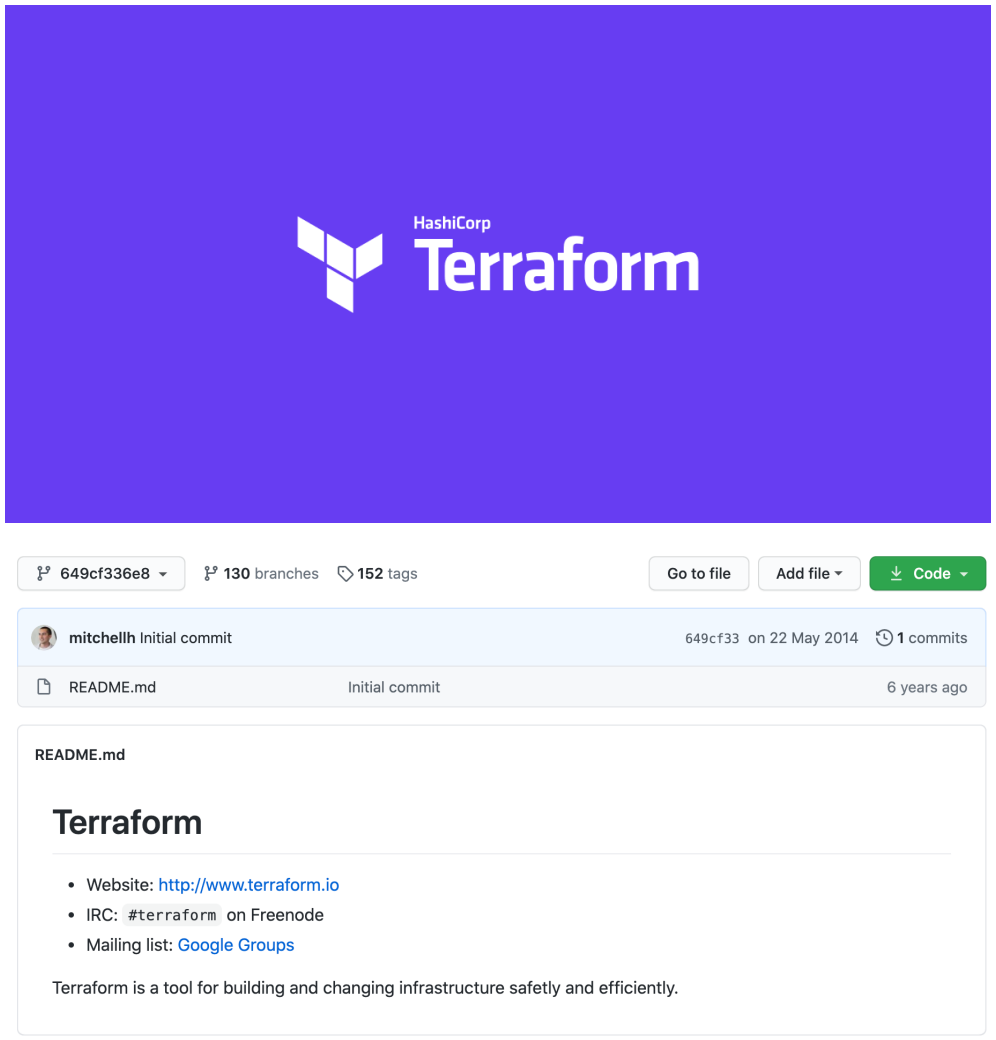
- State of DevOps Puppet - https://puppet.com/resources/report/state-of-devops-report/
- DevOps Team Topologies - https://web.devopstopologies.com/
- Microsoft DevOps Resource Center - https://docs.microsoft.com/en-us/azure/devops/learn/
- DevOps Fundamentals - https://medium.com/faun/all-about-devops-fundamentalsyou-ever-wanted-to-know-2333328a2b40
- Donovan Brown Definition of DevOps - https://medium.com/@DonovanBrown_41367/dissecting-the-definition-69151da0435f
- Agile and DevOps - https://www.atlassian.com/agile/devops
- Agile and DevOps – https://theagileadmin.com/what-is-devops/
- CAMS - https://blog.chef.io/what-devops-means-to-me/
- What is DevOps - https://www.kartar.net/2010/02/what-devops-means-to-me/
- AWS definition of DevOps - https://aws.amazon.com/devops/what-is-devops/
- Four Principles of DevOps - http://radify.io/blog/four-principles-of-devops/
- DevOps vs SysAdmin - https://opensource.com/article/19/7/devops-vs-sysadmin
- What DevOps is not - https://agileweboperations.com/2010/12/02/what-devops-is-not/



REFERENCES – VIDEO

- How Netflix Thinks of DevOps - https://www.youtube.com/watch?v=UTKIT6STSVM
- What's the Difference Between DevOps and SRE? (class SRE implements DevOps) - https://www.youtube.com/watch?v=uTEL8Ff1Zvk
- What is DevOps? In Simple English - https://www.youtube.com/watch?v=_I94-tJlovg
- DevOps Culture at Amazon - https://www.youtube.com/watch?v=mBU3AJ3j1rg
- What is DevOps (DevOps 101 from IBM)- https://www.youtube.com/watch?v=UbtB4sMaaNM&list=PLOspHqNVtKAAm1dmyiR9WMmw1UBoOwZVj
- #ChefConf 2014: Jez Humble, "DevOps Culture and Practices To Create Flow" - https://www.youtube.com/watch?v=oX8af9kLhlk
- Spotify Engineering Culture (by Henrik Kniberg) - https://www.youtube.com/watch?v=4GK1NDTWbkY
- DevOps Vs. SRE: Competing Standards or Friends? (Cloud Next '19) - https://www.youtube.com/watch?v=0UyrVqBoCAU&t=1s

EXTRA CONTENT – TERRAFORM IAC SOLUTION



TERRAFORM – HASHICORP

- Infrastructure as Code tool based on a declarative approach
- From GitHub: “Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.”
- 22 May 2014 - Mitchell Hashimoto first commit to create Terraform Project
- 19 July 2020 – 152 releases, 26357 commits, 130 branches, nearly 6k forks, nearly 23k stars, 1468 contributors
- Written in GO
- From HashiCorp introduction to Terraform:“Terraform is the infrastructure as code offering from HashiCorp. It is a tool for building, changing, and managing infrastructure in a safe, repeatable way. Operators and Infrastructure teams can use Terraform to manage environments with a configuration language called the HashiCorp Configuration Language (HCL) for human-readable, automated deployments.”

- ACME
- Akamai
- Alibaba Cloud
- Archive
- Arukas
- Auth0
- Avi Vantage
- Aviatrix
- AWS
- Azure
- Azure Active Directory
- Azure DevOps
- Azure Stack
- A10 Networks
- BaiduCloud
- Genymotion
- GitHub
- GitLab
- Google Cloud Platform
- Grafana
- Gridscale
- Hedvig
- Helm
- Heroku
- Hetzner Cloud
- HTTP
- HuaweiCloud
- HuaweiCloudStack
- Icinga2
- Ignition
- Oracle Public Cloud
- OVH
- Packet
- PagerDuty
- Palo Alto Networks PANOS
- Palo Alto Networks PrismaCloud
- PostgreSQL
- PowerDNS
- ProfitBricks
- Pureport
- RabbitMQ
- Rancher
- Rancher2
- Random
- Cobbler
- Cohesity
- Constellix
- Consul
- Datadog
- DigitalOcean
- DNS
- DNSimple
- DNSMadeEasy
- Docker
- Dome9
- Dyn
- EnterpriseCloud
- Exoscale
- External
- F5 BIG-IP
- MongoDB Atlas
- MySQL
- Naver Cloud
- Netlify
- New Relic
- Nomad
- NS1
- Null
- Nutanix
- 1&1
- Okta
- Okta Advanced Server Access
- OpenNebula
- OpenStack
- OpenTelekomCloud
- TencentCloud
- Terraform
- Terraform Cloud
- Time
- TLS
- Triton
- Turbot
- UCloud
- UltraDNS
- Vault
- Venafi
- VMware Cloud
- VMware NSX-T
- VMware vCloud Director
- VMware vRA7
- VMware vSphere

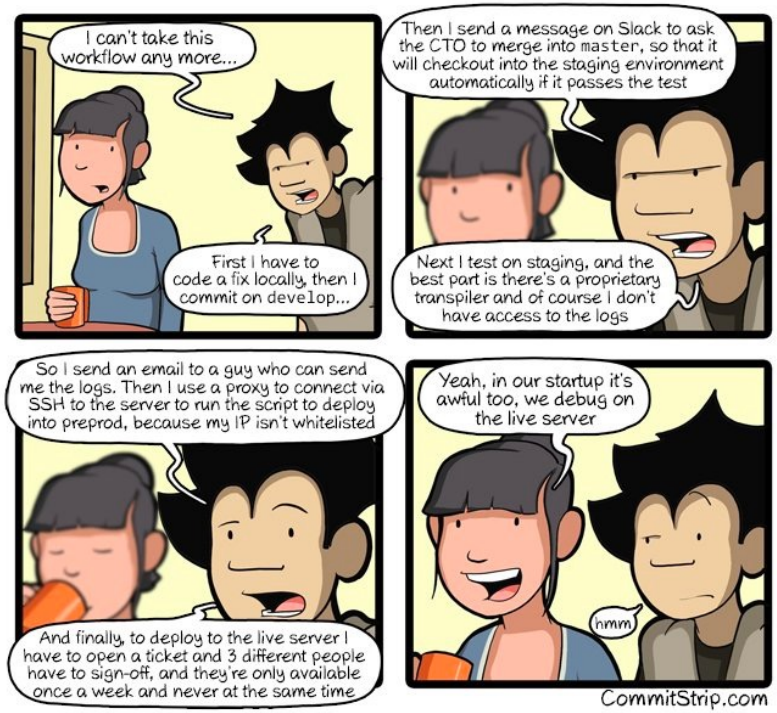
- Bitbucket
- Brightbox
- CenturyLinkCloud
- Check Point
- Chef
- CherryServers
- Circonus
- Cisco ASA
- Cisco ACI
- Cisco MSO
- CloudAMQP
- Cloudflare
- Cloud-init
- CloudScale.ch
- CloudStack
- Incapsula
- InfluxDB
- Infoblox
- JDCloud
- KingsoftCloud
- Kubernetes
- Lacework
- LaunchDarkly
- Librato
- Linode
- Local
- Logentries
- LogicMonitor
- Mailgun
- MetalCloud
- RightScale
- Rubrik
- Rundeck
- RunScope
- Scaleway
- Selectel
- SignalFx
- Skytap
- SoftLayer
- Spotinst
- StackPath
- StatusCake
- Sumo Logic
- TelefonicaOpenCloud
- Template
- Fastly
- FlexibleEngine
- FortiOS
- OpsGenie
- Oracle Cloud Infrastructure
- Oracle Cloud Platform
- Vultr
- Wavefront
- Yandex

HashiCorp Terraform

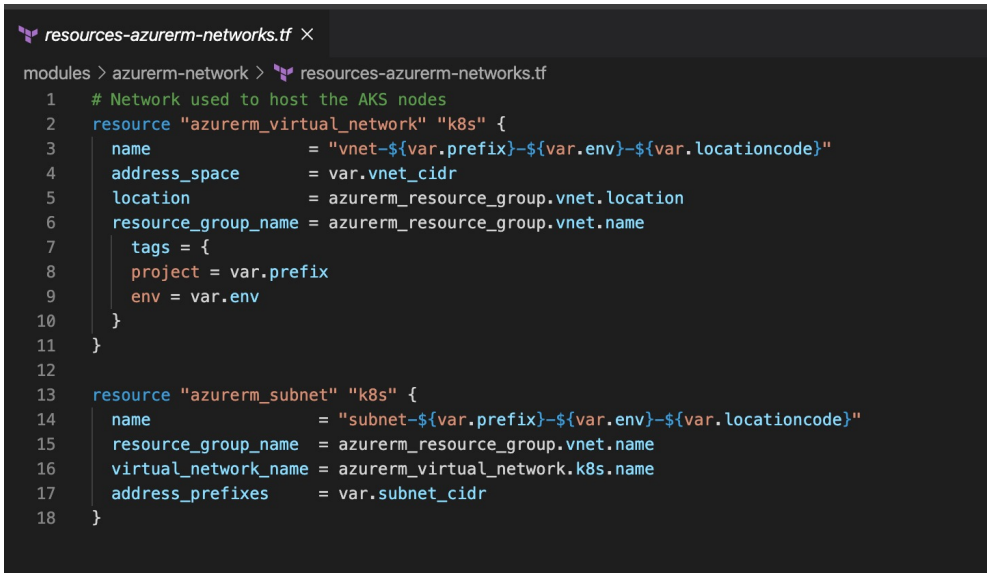
WHY A TOOL AS TERRAFORM

- Automate Infrastructure Provisioning
- One single tool and language (HCL) instead of multiple cloud specific solutions (AWS CloudFormation; Azure Resource Manager, Google Cloud Deployment Manager, etc..
- Omni comprehensive list of providers
  - 148 Officially supported
  - 200 Community based
- Definition is documentation
- Easy distribution:
  - Single binary files
  - Plain text files
- Open Source with a large community

TERRAFORM CONCEPTS

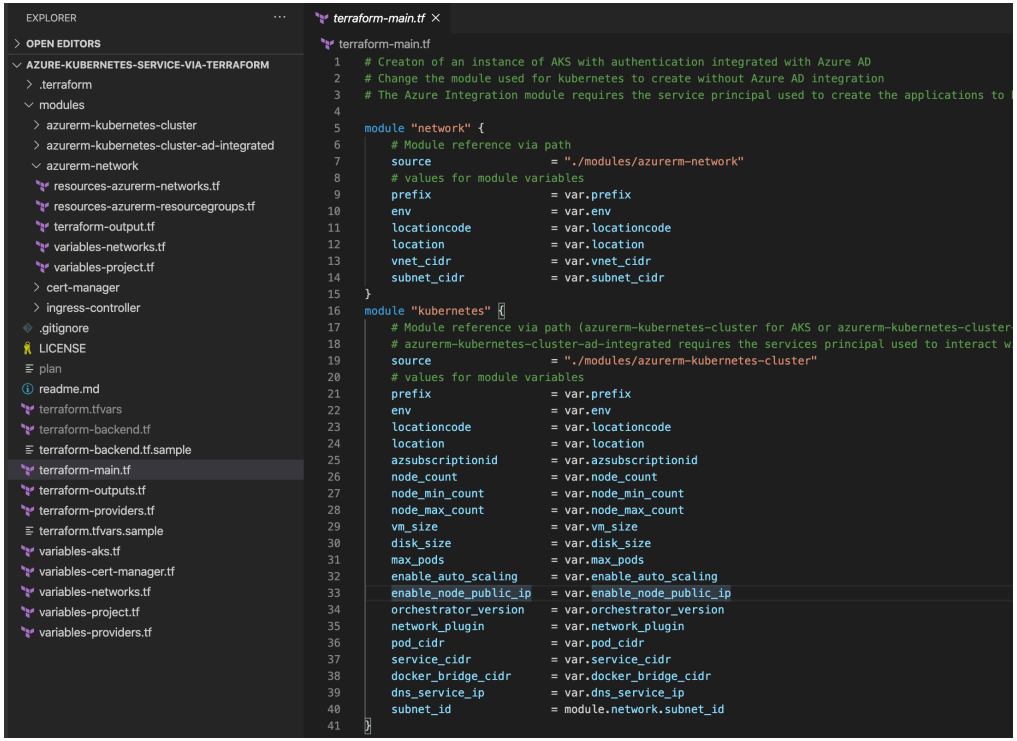


HASHICORP CONFIGURATION LANGUAGE (HCL)



- “Terraform uses its own configuration language, designed to allow concise descriptions of infrastructure. The Terraform language is declarative, describing an intended goal rather than the steps to reach that goal.”
- HCL syntax is composed of stanzas or blocks that define a variety of configurations available to Terraform. Provider plugins expand on the available base Terraform configurations.
- Stanzas or blocks are comprised of key = value pairs. Terraform accepts values of type string, number, boolean, map, list, tuple, object.
- Single line comments start with #, while multi-line comments use an opening /* and a closing */.
- Interpolation syntax can be used to reference values stored outside of a configuration block, like in an input variable, or from a Terraform module’s output.
- You can include multi-line strings by using an opening <<EOF, followed by a closing EOF on its own line.
- Strings are wrapped in double quotes.
- Lists of primitive types (string, number, and boolean) are wrapped in square brackets: [”value-one", ”value-two", ”value-three"].
- Maps use curly braces {} and colons :, as follows: { "password" : "my_password", "db_name" : "wordpress" }.

TERRAFORM PROJECTS



- The simplest Terraform configuration is a single root module containing only a single
- Terraform language uses configuration files that are named with the .tf extension
- Configuration files must always use UTF-8 encoding
- The root module is built from the configuration files in the current working directory when Terraform is run
- Any module may reference child modules in other directories
- The ordering of blocks is generally not significant, Terraform automatically processes resources in the correct order based on relationships defined between them in configuration
- The Terraform command line interface (CLI) is a general engine for evaluating and applying Terraform configurations
- This general engine has no knowledge about specific types of infrastructure objects. Instead, Terraform uses plugins called providers that each define and manage a set of resource types

META-ARGUMENTS

```
variable "groups" {
  description = "Security Groups to be assigned as cluster admin, grouname and id"
  type = map
  default = {
    TeamA = "00-01-02"
    TeamB = "00-21-32"
    TeamC = "00-14-42"
  }
}

resource "kubernetes_cluster_role_binding" "aad_integration" {
  for_each = var.zones
  metadata {
    name = format("aks-admin-%s", each.key)
  }
  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "ClusterRole"
    name      = "cluster-admin"
  }
  subject {
    kind = "Group"
    name = each.value
  }
  depends_on = [
    azurerm_kubernetes_cluster.aks
  ]
}
```

- Terraform CLI defines the following meta-arguments, which can be used with any resource type to change the behavior of resources:
  - depends_on, for specifying hidden dependencies
  - count, for creating multiple resource instances according to a count
  - for_each, to create multiple instances according to a map, or set of strings
  - provider, for selecting a non-default provider configuration
  - lifecycle, for lifecycle customizations
  - provisioner and connection, for taking extra actions after resource creation

FUNCTIONS

```
locals {
  prefixMongoName = "l${var.tenant}-${var.env}-node"
  mongoInstances = "${lower(var.hasReplicaSet) != "true" ? 1 : 3 }"
  isCluster = "${local.mongoInstances != 1 ? "true" : "false"}"
  nodes = "${join("," ,formatlist("%s-dns.${lower(replace(var.location, " ", ""))}.cloudapp.azure.com:27017", azurerm_virtual_machine.MongoVirtualMachine.*.name))}"
  mongoNoReplica = "mongodb://${var.username}:${var.mongopassword}@${local.nodes}"
  mongoReplicaset = "mongodb://${var.username}:${var.mongopassword}@${local.nodes}/${var.tenant}-#collectionName#?authSource=admin&replicaSet=rs0"
  mongoConnectionString = "${local.isCluster != "true" ?  local.mongoNoReplica : local.mongoReplicaset }"

}
```

- The Terraform language includes a number of built-in functions that you can call from within expressions to transform and combine value

- String functions:
  - Trim, Regex, Lower, Join, Split, Substr; etc..
- Collection functions:
  - Concat, Distinct, Index, Reverse, Lookup, etc..

```
locals {
  prefixMongoName = "l${var.tenant}-${var.env}-node"
  mongoInstances = "${lower(var.hasReplicaSet) != "true" ? 1 : 3 }"
  isCluster = "${local.mongoInstances != 1 ? "true" : "false"}"
  nodes = "${join("," ,formatlist("%s-dns.${lower(replace(var.location, " ", ""))}.cloudapp.azure.com:27017", azurerm_virtual_machine.MongoVirtualMachine.*.name))}"
  mongoNoReplica = "mongodb://${var.username}:${var.mongopassword}@${local.nodes}"
  mongoReplicaset = "mongodb://${var.username}:${var.mongopassword}@${local.nodes}/${var.tenant}-#collectionName#?authSource=admin&replicaSet=rs0"
  mongoConnectionString = "${local.isCluster != "true" ?  local.mongoNoReplica : local.mongoReplicaset }"

}
```

FUNCTIONS

- Encoding functions:
  - Urlencode, Base64Encode, Base64Decode, Csvdecode, etc..
- Filesystem functions:
  - File, Fileexists, Templatefile,
- Data and Time functions:
  - Timestamp, Timeadd, Formatdate
- Hash and Crypto functions:
  - Sha1, Sha256, Md5, Uuid, Base64sha256, etc..
- IP Network functions:
  - Cidrhost, Cidrnetmask, Cidrsubnet
- Type Conversion functions:
  - Try, Tostring, Tonumber, Can, etc..

INPUT VARIABLES

modules > cert-manager > variables-cert-manager.tf

```
1   variable "namespace" {
2     description = "The name of the namespace used"
3     default = "cert-manager"
4   }
5
6   variable "emailproduction" {
7     description = "The email to be used to configure the production cluster-issuer"
8   }
9
10  variable "emailstaging" {
11    description = "The email to be used to configure the staging cluster-issuer"
12  }
```

- Input variables serve as parameters for a Terraform module
- Each input variable accepted by a module must be declared using a variable block
- Variables can be declared on the root module or on child modules
- The variable declaration can optionally include a type argument
- The variable declaration can also include a default argument
- The name of a variable can be any valid identifier except the following reserved names:
    - Source, Version, Providers, Count, For_each, Lifecycle, Depends_on, L ocals

LOCAL VALUES

```
locals {
  tags = {
    source       = "terraform"
    division     = var.division
    customer     = var.tenant
    project      = var.tenant
    env          = var.env
    product      = var.product
    supported-by = "b2b"
    host-role    = "Mongo"
  }
}

module "VirtualNetwork" {
  source = "../Vnet"

  location = var.location
  resource_group_name = var.resourceGroupName
  address_space_size = 16
  tags = local.tags
}
```

- A local value assigns a name to an expression, allowing it to be used multiple times within a module without repeating it
- A set of related local values can be declared together in a single locals block
- The expressions assigned to local value names can either be simple constants
- The expressions assigned to a local values can be more complex expressions that transform or combine values from elsewhere in the module

OUTPUT VALUES

modules > azurerm-kubernetes-cluster-ad-integrated > terraform-outputs.tf

```
output "id" {
  value = azurerm_kubernetes_cluster.aks.id
}

output "kube_config" {
  value = azurerm_kubernetes_cluster.aks.kube_config_raw
}

output "client_key" {
  value = azurerm_kubernetes_cluster.aks.kube_config.0.client_key
}

output "client_certificate" {
  value = azurerm_kubernetes_cluster.aks.kube_config.0.client_certificate
}

output "cluster_ca_certificate" {
  value = azurerm_kubernetes_cluster.aks.kube_config.0.cluster_ca_certificate
}

output "host" {
  value = azurerm_kubernetes_cluster.aks.kube_config.0.host
}

output "server-app-id" {
  value = "${azuread_application.aks-aad-srv.application_id}"
}
output "client-app-id" {
  value = "${azuread_application.aks-aad-client.application_id}"
}
```

- Output values are like the return values of a Terraform module
- Output must explicitly be declared with the dedicated blocks
- Any valid expression is allowed as an output value
- Outputs are only rendered when Terraform applies your plan
  - A child module can use outputs to expose a subset of its resource attributes to a parent module.
  - A root module can use outputs to print certain values in the CLI output after running terraform apply.
  - When using remote state, root module outputs can be accessed by other configurations via a terraform_remote_state data source.
- output blocks can optionally include description, sensitive, and depends_on arguments

PROVIDERS

terraform-providers.tf

```
provider "azurerm" {
    version = "=2.17.0"
    features {}
    subscription_id = var.azsubscriptionid
    client_id = var.azclientid
    client_secret = var.azclientsecret
    tenant_id = var.aztenantid
}

provider "azuread" {
  version = "=0.9.0"
  subscription_id = var.azsubscriptionid
  client_id = var.azclientid
  client_secret = var.azclientsecret
  tenant_id = var.aztenantid
}

provider "kubernetes" {
  version = "1.11.3"
  host                   = module.kubernetes.host
  client_certificate     = base64decode(module.kubernetes.client_certificate)
  client_key             = base64decode(module.kubernetes.client_key)
  cluster_ca_certificate = base64decode(module.kubernetes.cluster_ca_certificate)
}

provider "helm" {
  version = "1.2.3"
  kubernetes {
    host     = module.kubernetes.host

    client_certificate     = base64decode(module.kubernetes.client_certificate)
    client_key             = base64decode(module.kubernetes.client_key)
    cluster_ca_certificate = base64decode(module.kubernetes.cluster_ca_certificate)
  }
}
```

- A provider works pretty much as an operating system's device driver.
- Providers usually require some configuration of their own to specify endpoint URLs, regions, authentication settings, and so on
- Terraform must initialize the provider before it can be used
- Each provider offers a set of named resource types, and defines for each resource type which arguments it accepts, which attributes it exports, and how changes to resources of that type are actually applied to remote APIs
- There are also two "meta-arguments" that are defined by Terraform itself and available for all provider blocks:
  - version, for constraining the allowed provider versions
  - alias, for using the same provider with different configurations for different resources
    - To select an aliased provider for a module, resource or data source, set its provider meta-argument to a <PROVIDER NAME>.<ALIAS>

```
# AAD K8s cluster admin group / AAD

resource "azuread_group" "aks-aad-clusteradmins" {
  name = "aks-${var.prefix}-${var.env}-${var.locationcode}-clusteradmin"
}

# Service Principal for AKS

resource "azuread_application" "aks_sp" {
  name                       = "aks-${var.prefix}-${var.env}-${var.locationcode}"
  homepage                   = "https://aks-${var.prefix}-${var.env}-${var.locationcode}"
  identifier_uris            = ["https://aks-${var.prefix}-${var.env}-${var.locationcode}"]
  reply_urls                 = ["https://aks-${var.prefix}-${var.env}-${var.locationcode}"]
  available_to_other_tenants = false
  oauth2_allow_implicit_flow = false
}

resource "azuread_service_principal" "aks_sp" {
  application_id = azuread_application.aks_sp.application_id
}

resource "random_password" "aks_sp_pwd" {
  length  = 16
  special = true
}

resource "azuread_service_principal_password" "aks_sp_pwd" {
  service_principal_id = azuread_service_principal.aks_sp.id
  value                = random_password.aks_sp_pwd.result
  end_date             = "2024-01-01T01:02:03Z"
}

resource "azurerm_role_assignment" "aks_sp_role_assignment" {
  scope                = data.azurerm_subscription.current.id
  role_definition_name = "Contributor"
  principal_id         = azuread_service_principal.aks_sp.id

  depends_on = [
    azuread_service_principal_password.aks_sp_pwd
  ]
}
```

RESOURCES

- Resources are the most important element in the Terraform language
- Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records
- A resource block declares a resource of a given type with a given local name, the combination of type and name must be unique in the module
- The name is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside of the scope of a module
- Each resource is associated with a single resource type, which determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports
- Each resource type in turn belongs to a provider,
- Within the block body (between { and }) are the configuration arguments for the resource itself, specific for each resource defined by the provider
- When Terraform creates a new infrastructure object represented by a resource block, the identifier for that real object is saved in Terraform's state, allowing it to be updated and destroyed in response to future changes
- On a second or subsequent execution, Terraform compares the actual configuration (from the state) of the object with the arguments given in the configuration and, if necessary, updates the object to match the configuration

DATA SOURCES

```
data "azurerm_subscription" "current" {
    subscription_id = var.azsubscriptionid
}

resource "azurerm_role_assignment" "aks_sp_role_assignment" {
  scope                = data.azurerm_subscription.current.id
  role_definition_name = "Contributor"
  principal_id         = azuread_service_principal.aks_sp.id

  depends_on = [
    azuread_service_principal_password.aks_sp_pwd
  ]
}
```

- Data sources allows a Terraform configuration to make use of information defined outside of Terraform
- Each data resource is associated with a single data source, which determines the kind of object (or objects) it reads and what query constraint arguments are available
- Each data source belongs to a provider
- A data block declares a resource of a given type with a given local name, the combination of type and name must be unique in the module
- The name is used to refer to this data from elsewhere in the same Terraform module, but has no significance outside of the scope of a module
- Within the block body (between { and }) are query constraints defined by the data source, specific for each data source defined by the provider
- Each data instance will export one or more attributes, which can be used in other resources as reference expressions of the form data.<TYPE>.<NAME>.<ATTRIBUTE>

```
module "network" {
    # Module reference via path
    source                  = "./modules/azurerm-network"
    # values for module variables
    prefix                  = var.prefix
    env                     = var.env
    locationcode            = var.locationcode
    location                = var.location
    vnet_cidr               = var.vnet_cidr
    subnet_cidr             = var.subnet_cidr
}
module "kubernetes" {
    # Module reference via path (azurerm-kubernetes-cluster for AKS or azurerm-kubernetes-cluster-ad-integrated to have
    # azurerm-kubernetes-cluster-ad-integrated requires the services principal used to interact with Azure to have Globa
    source                  = "./modules/azurerm-kubernetes-cluster"
    # values for module variables
    prefix                  = var.prefix
    env                     = var.env
    locationcode            = var.locationcode
    location                = var.location
    azsubscriptionid        = var.azsubscriptionid
    node_count              = var.node_count
    node_min_count          = var.node_min_count
    node_max_count          = var.node_max_count
    vm_size                 = var.vm_size
    disk_size               = var.disk_size
    max_pods                = var.max_pods
    enable_auto_scaling     = var.enable_auto_scaling
    enable_node_public_ip   = var.enable_node_public_ip
    orchestrator_version    = var.orchestrator_version
    network_plugin          = var.network_plugin
    pod_cidr                = var.pod_cidr
    service_cidr            = var.service_cidr
    docker_bridge_cidr      = var.docker_bridge_cidr
    dns_service_ip          = var.dns_service_ip
    subnet_id               = module.network.subnet_id
}

module "ingress" {
    # Module reference via path
    source                  = "./modules/ingress-controller"
    # values for module variables
    prefix                  = var.prefix
    env                     = var.env
    locationcode            = var.locationcode
    location                = var.location
}
```

MODULES

- A module is a container for multiple resources that are used together
- Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory
- A module can call other modules, which lets you include the child module's resources into the configuration
- Modules can also be called multiple times, either within the same configuration or in separate configurations, allowing resource configurations to be packaged and re-used
- Modules are called from within other modules using module blocks
- The label immediately after the module keyword is a local name, which the calling module can use to refer to this instance of the module
- Within the block body (between { and }) are the arguments for the module, corresponding to input variables defined by the module

STATE

- State is a necessary requirement for Terraform to function
- Terraform must store state about your managed infrastructure and configuration
- This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment
- Terraform uses this local state to create plans and make changes to your infrastructure
- While the format of the state files are just JSON, direct file editing of the state is discouraged!
- Terraform provides the terraform state command to perform basic modifications of the state using the CLI
- Terraform is able to import existing infrastructure updating the state with the information about it
- Terraform state can contain sensitive data (e.g. initial password), reat the state itself as sensitive data!

BACKEND

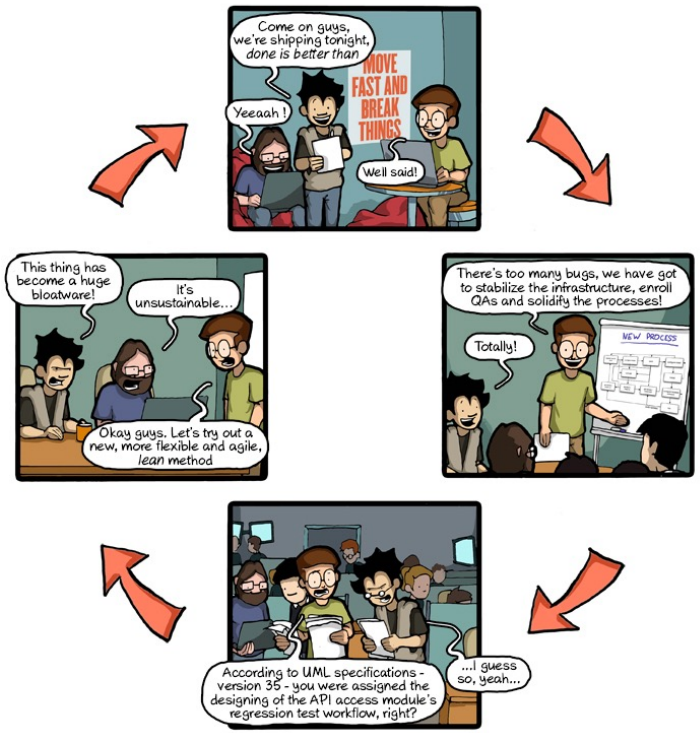
```
terraform {
  backend "azurerm" {
    resource_group_name  = "resource-group-name-of-the-storage-used-for-the-state-file"
    storage_account_name = "name-of-the-storage-used-for-the-state-file"
    container_name       = "name-of-the-container-on-the-blob-storage-used-for-the-state-file"
    key                  = "state-file-name-key"
    subscription_id = "00000000-0000-0000-0000-00000000000"
    client_id = "00000000-0000-0000-0000-00000000000"
    client_secret = "AAAAAA.AAAAAAAAAAAAAAAAAAAA-AAAA"
    tenant_id = "00000000-0000-0000-0000-00000000000"
  }
}
```

- A "backend" in Terraform determines how state is loaded and how an operation such as apply is executed
- By default, Terraform uses the "local" backend, which is the normal behaviour of Terraform
- Backends are configured directly in Terraform files in the terraform section
- After configuring a backend, it has to be initialized
- You can change your backend configuration at any time; Terraform will automatically detect any changes in your configuration and request a reinitialization and as part of the reinitialization process, it will ask if you'd like to migrate your existing state to the new configuration
- Remote backend are used for:
  - Team work (sharing the same state file)
  - Secure state file (encrypted and monitoring storage)
  - Execution of Terraform from pipelines

WORKSPACE

- Named workspaces allow conveniently switching between multiple instances of a single configuration within its single backend
- Terraform starts with a single workspace named "default”
- Workspaces are managed with the terraform workspace set of commands
  - terraform workspace new *name*
  - terraform workspace select *name*
- A common use for multiple workspaces is to create a parallel, distinct copy of a set of infrastructure in order to test a set of changes before modifying the main production infrastructure
- Non-default workspaces are often related to feature branches in version control
- Workspaces alone are not a suitable tool for system decomposition
- Named workspaces are not a suitable isolation mechanism if you want to create a strong separation between multiple deployments of the same infrastructure serving different development stages

TERRAFORM WORKFLOW



INIT

- The terraform init command is used to initialize a working directory containing Terraform configuration files
- This command is always safe to run multiple times
- During init, the configuration is searched for module blocks, and the source code for referenced modules is retrieved from the locations given in their source arguments
- During init, Terraform searches the configuration for both direct and indirect references to providers and attempts to load the required plugins

VALIDATE

- The terraform validate command validates the configuration files in a directory
- Validate runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state
- Validation requires an initialized working directory with any referenced plugins and modules installed

PLAN

- The terraform plan command is used to create an execution plan
- Terraform determines what actions are necessary to achieve the desired state specified in the configuration files
- This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state
- The optional -out argument can be used to save the generated plan to a file for later execution with terraform apply

APPLY

- The command actually applying the infrastructure creation, update and configuration as defined in the modules
- The terraform apply command is used to apply the changes required to reach the desired state of the configuration
- It will define the plan refreshing the state from the current configuration and defining the changes to be applied in the form of create, change, destroy actions on each individual resource
- It can use a pre-defined plan execution file saved by the plan command
- It will update the state with the new actual configuration at the end of the execution

DESTROY

- The terraform destroy command is used to destroy the Terraform-managed infrastructure
- This command accepts all the arguments and flags that the apply command accepts, with the exception of a plan file argument
- This command will use the state and configuration files to delete each and every defined resource
- This command will update the state file once operations completed
- The behavior of any terraform destroy command can be previewed at any time with an equivalent terraform plan -destroy command

EXTRA CONTENT – OBSERVABILITY

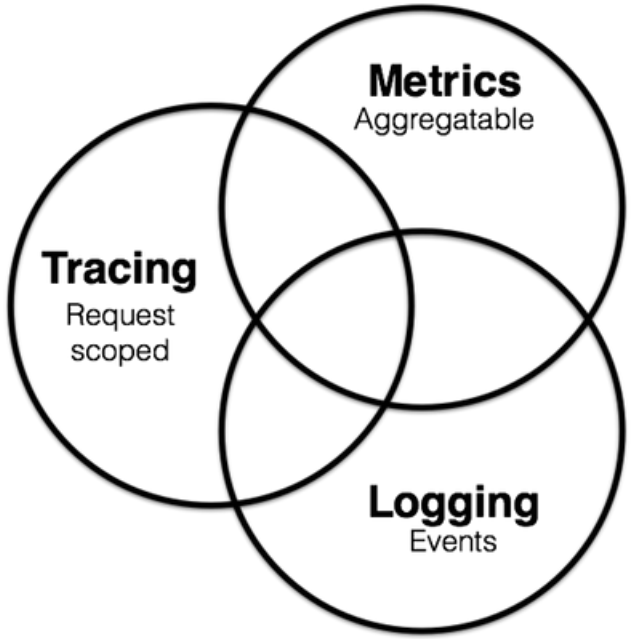
OBSERVABILITY – THE BASIC CONCEPTS

- Enterprise IT and software-driven consumer product development are increasingly complex.
- Both the system observability and monitoring play critical roles in achieving system dependability—but they're not the same thing.
- "Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. In control theory, the observability and controllability of a linear system are mathematical duals. The concept of observability was introduced by Hungarian-American engineer Rudolf E. Kálmán for linear dynamic systems. A dynamical system designed to estimate the state of a system from measurements of the outputs is called a state observer or simply an observer for that system."
- In enterprise IT, monitoring refers specifically to the process of translating infrastructure log metrics data into meaningful and actionable insights.

OBSERVABILITY – THE BASIC CONCEPTS

Monitoring tells you whether the system works.
Observability lets you ask why it's not working.

- @xarpb, 2017



- The shift of mentality between traditional monitoring and observability can be summarised as follow:
  - Monitor Services/Applications not Machines
  - Apply Black Box Monitoring + White Box Monitoring
  - Collects Data
  - Collects More Data
  - Correlate Data
  - Look for anomalies

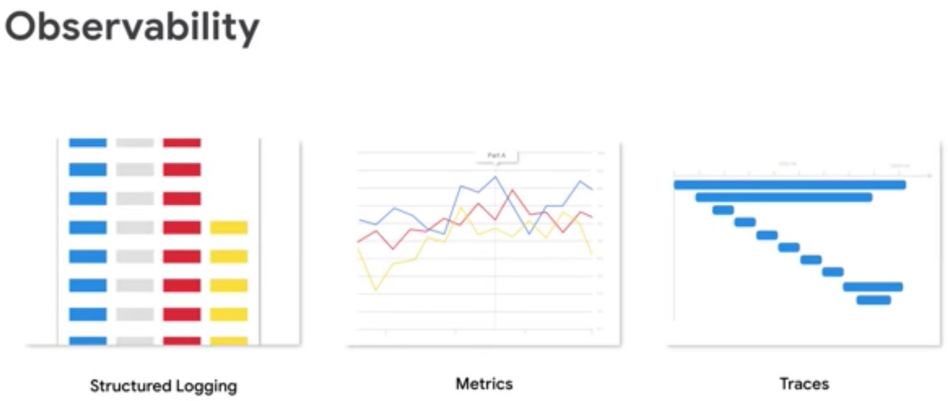
WHITE-BOX AND BLACK-BOX MONITORING

Each monitoring system should address two questions: what's broken (symptom) and why (cause)

Two types of monitoring can be defined:

- White-box monitoring: it inspects the internal state of the target service (application components metrics, traces, logs). Focus on causes.
- Black-box monitoring: it accesses the systems from external, as a real user (http\tcp probes, dns resolution, network ping). Symptom-oriented. Active recognition of error condition.

OBSERVABILITY – THE BASIC CONCEPTS



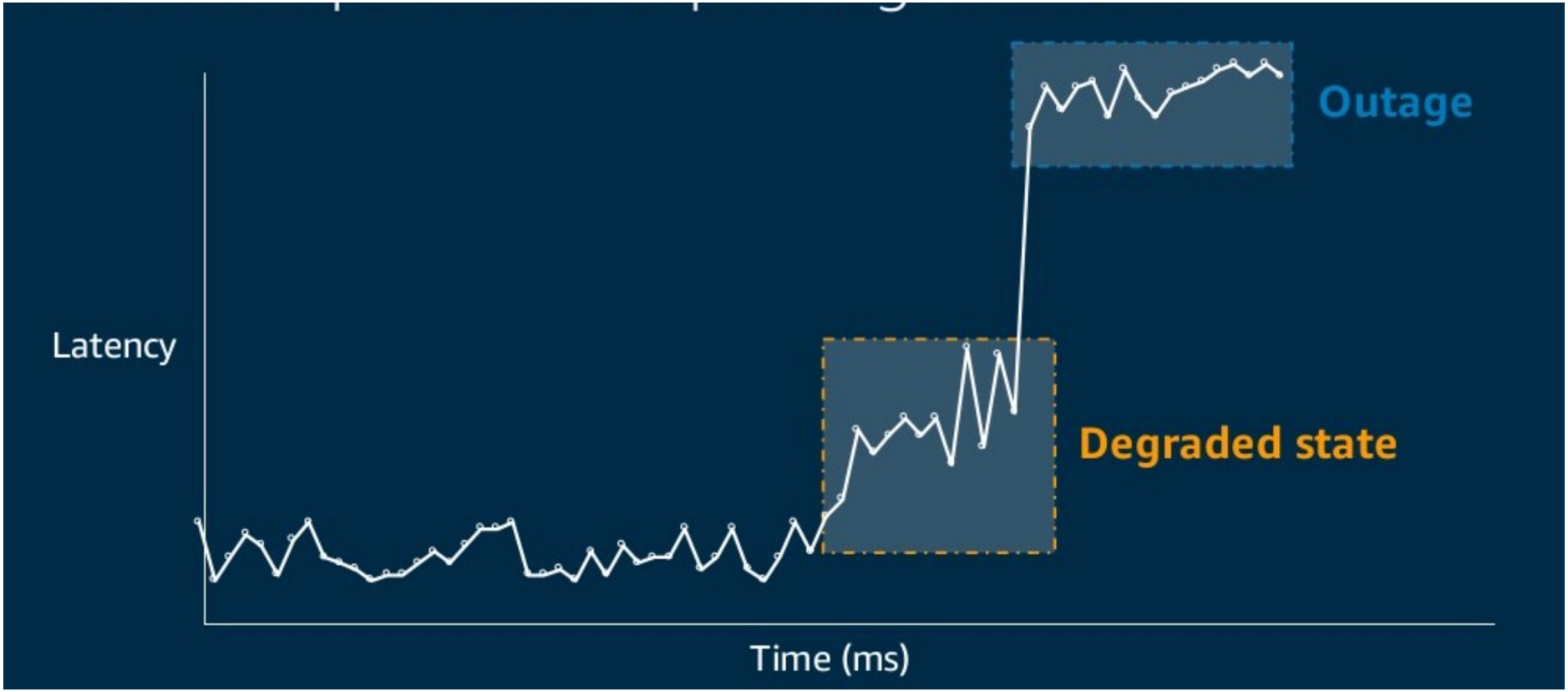
Collecting metrics and alerting are business driven activities.

Collecting metrics and manage alerting are different thing. You need to collect metrics to have visibility on your systems, but you do not have to use all metrics to generate alerts.

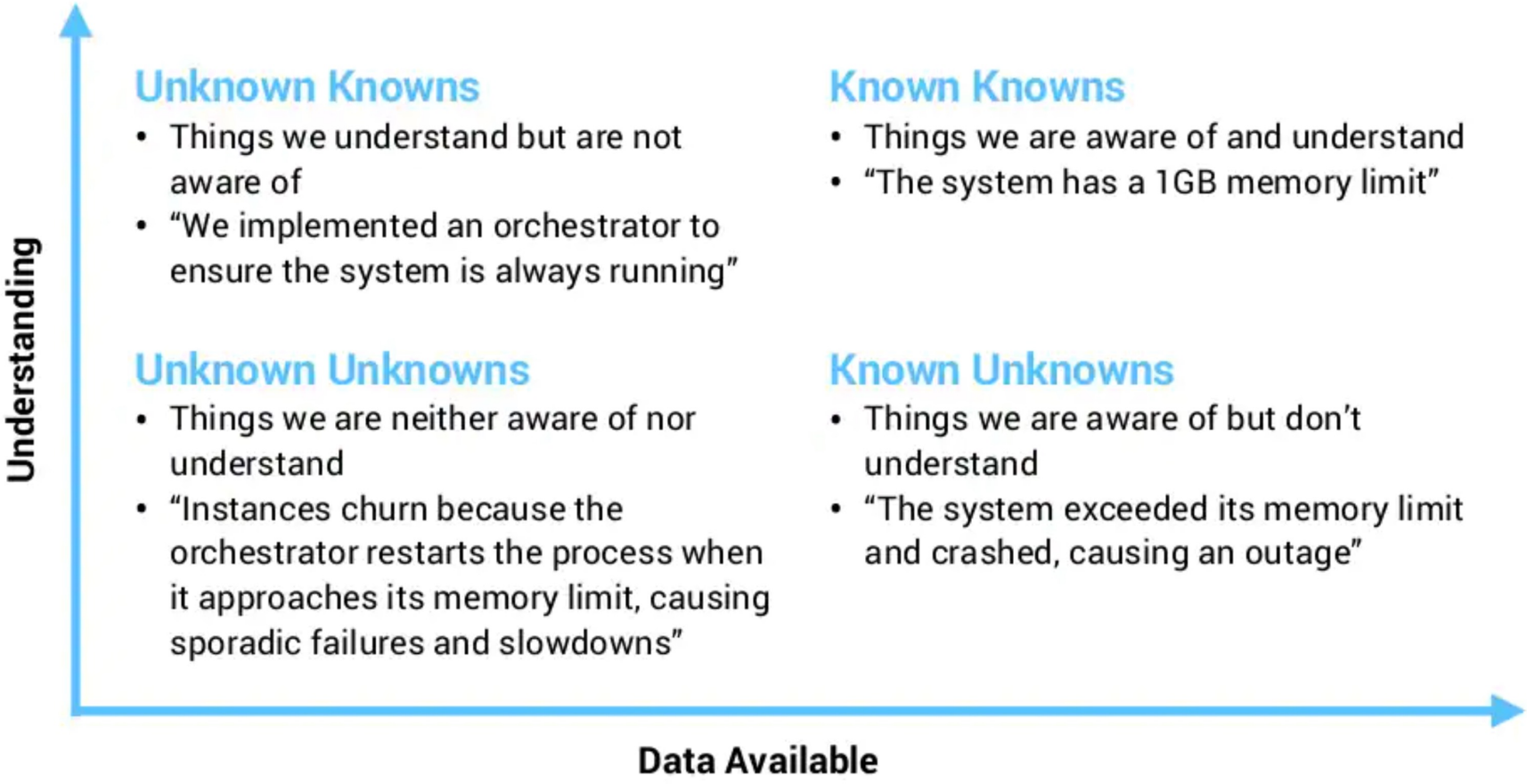
Measurement, monitoring and alerting must be related to SLOs.

Do not alert on anything, alerts must report a service impact and be actionable.

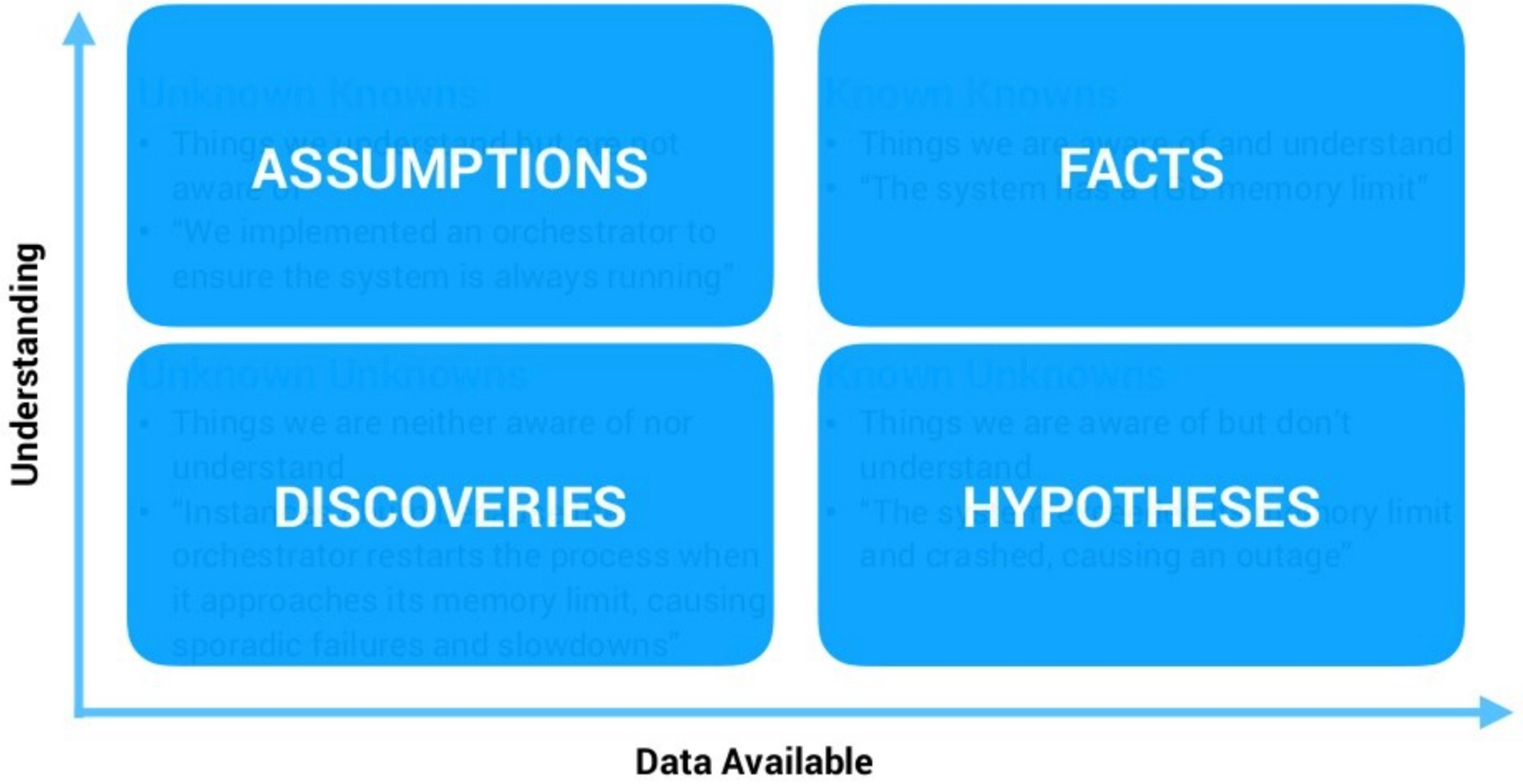
OBSERVABILITY – THE BASIC CONCEPTS



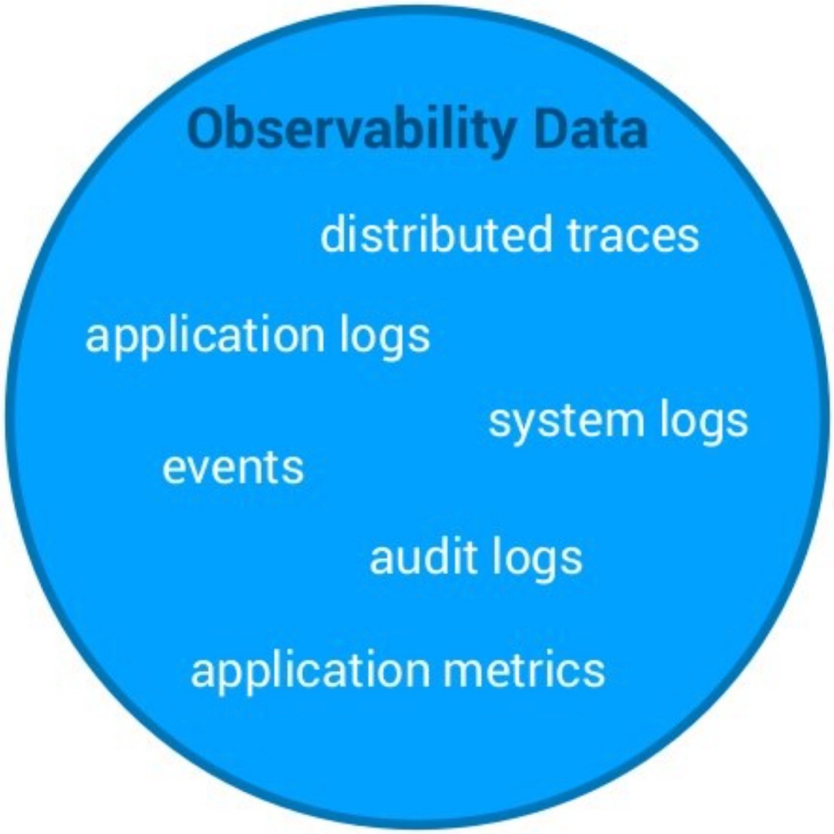
OBSERVABILITY – THE BASIC CONCEPTS



OBSERVABILITY – THE BASIC CONCEPTS



OBSERVABILITY – THE BASIC CONCEPTS



- Distributed systems are difficult to monitor
- Each system and application can provide a large amount of information about its status
- Different information from the same system can be correlated
  - Traces reporting higher response time + Metrics reporting high CPU usage
- Different information from different systems can be correlated
  - Traces from web application reporting high error rate + Logs from database system reporting high occurrence of deadlocks

OBSERVABILITY – THE BASIC CONCEPTS

Data Sources

- VMs
- Containers
- Load balancers
- Service meshes
- Audit logs
- VPC flow logs
- Firewall logs
- ...

What data to send?
Where to send it?
How to send it?

Data Sinks

- Centralized logging
- SIEM
- Monitoring
- APM
- Alerting
- Cold storage
- BI
- ...

Evolving to an Observability Pipeline

- Adopt structured logging
- Move log/data collection out of process
- Use a centralized logging system
- Introduce a streaming data solution
- Start adding data consumers

MONITORING AND ALERTING

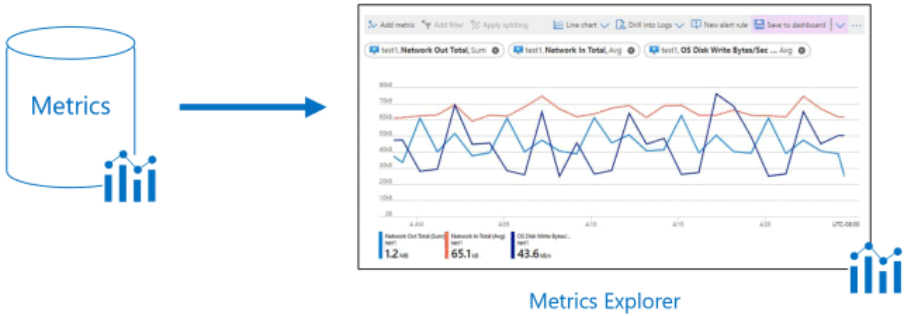
Monitoring may have only three output types:

- Pages - A human must do something now
- Tickets - A human must do something within a few days
- Logging - No one need look at this output immediately, but it's available for later analysis if needed

"Putting alerts into email and hoping that someone will read all of them and notice the important ones is the moral equivalent of piping them to /dev/null : they will eventually be ignored."

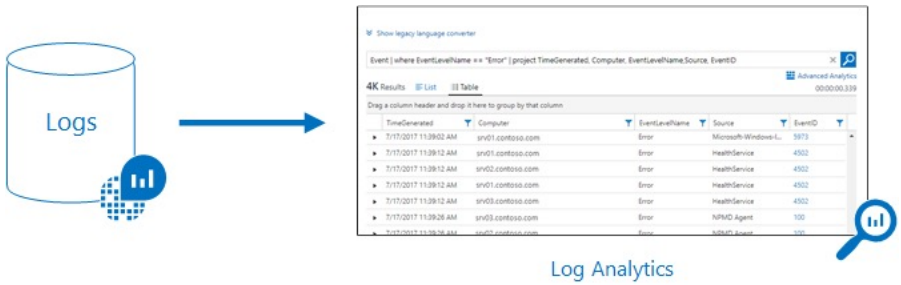
An alert must be analysed, if an alert is ignored, remove the alerting rule.

OBSERVABILITY – METRICS



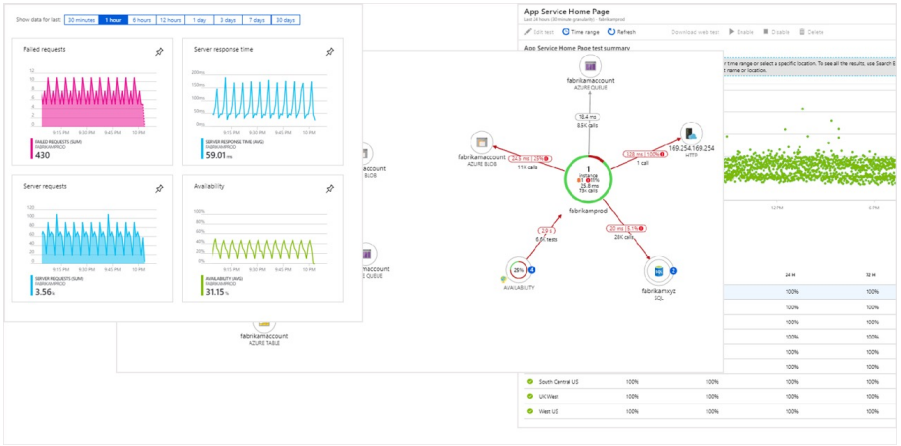
- Metrics are a central part of any monitoring process, but even when you have the right ones, you're necessarily limited by the constraints of linear time. People decide on metrics based on failures they've already found and fixed in the past. But there may be unknown unknowns: failures you haven't seen before, and therefore can't anticipate. Preemptively checking your metrics to find patterns is an option, but this isn't a replacement for being able to come back quickly from a failure. In short, metrics are necessary, but not sufficient.

OBSERVABILITY – LOGS



- While metrics should be constantly tracked, you only look at logs when your metrics are showing something strange you'd like to investigate. They're more specific and detailed than metrics, and they exist to show you what happened in each event. Having understandable, queryable, comprehensive logs is a significant component of what separates the observable from the non-observable system.

OBSERVABILITY – TRACES



- Tracing is really just a type of logging that's designed to record the flow of a program's execution. Typically, tracing is more granular than standard logging: while logs may say that a program installation failed, a trace will show you the specific exception that was thrown and when during the runtime it happened. Tracing is frequently used to detect latency issues or find out which of many microservices is not working. It's especially useful for error detection in distributed systems, to such an extent that this use case has its own name: distributed tracing.

THE FOUR GOLDEN SIGNALS (1/2)

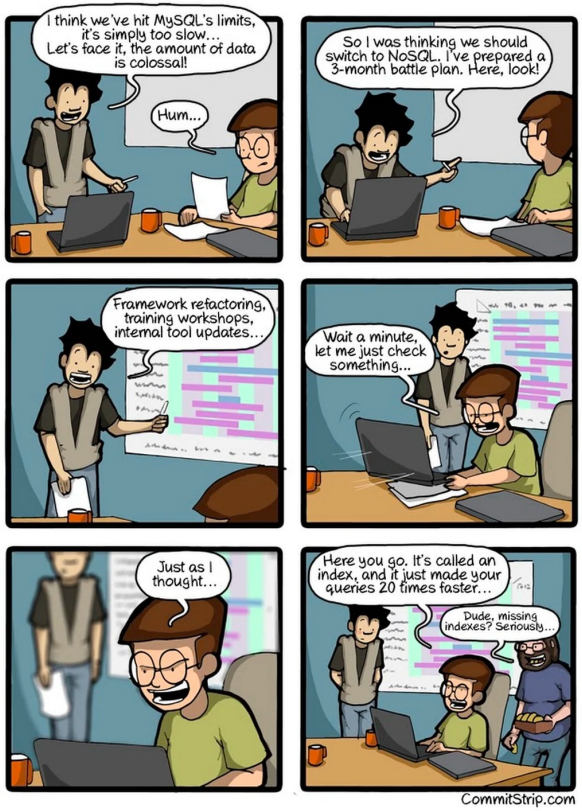
Latency

The time it takes to service a request. It's important to distinguish between the latency of successful requests and the latency of failed requests. For example, an HTTP 500 error triggered due to loss of connection to a database or other critical backend might be served very quickly; however, as an HTTP 500 error indicates a failed request, factoring 500s into your overall latency might result in misleading calculations. On the other hand, a slow error is even worse than a fast error! Therefore, it's important to track error latency, as opposed to just filtering out errors.

Traffic

A measure of how much demand is being placed on your system, measured in a high-level system-specific metric. For a web service, this measurement is usually HTTP requests per second, perhaps broken out by the nature of the requests (e.g., static versus dynamic content). For an audio streaming system, this measurement might focus on network I/O rate or concurrent sessions. For a key-value storage system, this measurement might be transactions and retrievals per second.

THE FOUR GOLDEN SIGNALS (2/2)



Errors

The rate of requests that fail, either explicitly (e.g., HTTP 500s), implicitly (for example, an HTTP 200 success response, but coupled with the wrong content), or by policy (for example, "If you committed to one-second response times, any request over one second is an error"). Where protocol response codes are insufficient to express all failure conditions, secondary (internal) protocols may be necessary to track partial failure modes. Monitoring these cases can be drastically different: catching HTTP 500s at your load balancer can do a decent job of catching all completely failed requests, while only end-to-end system tests can detect that you're serving the wrong content.

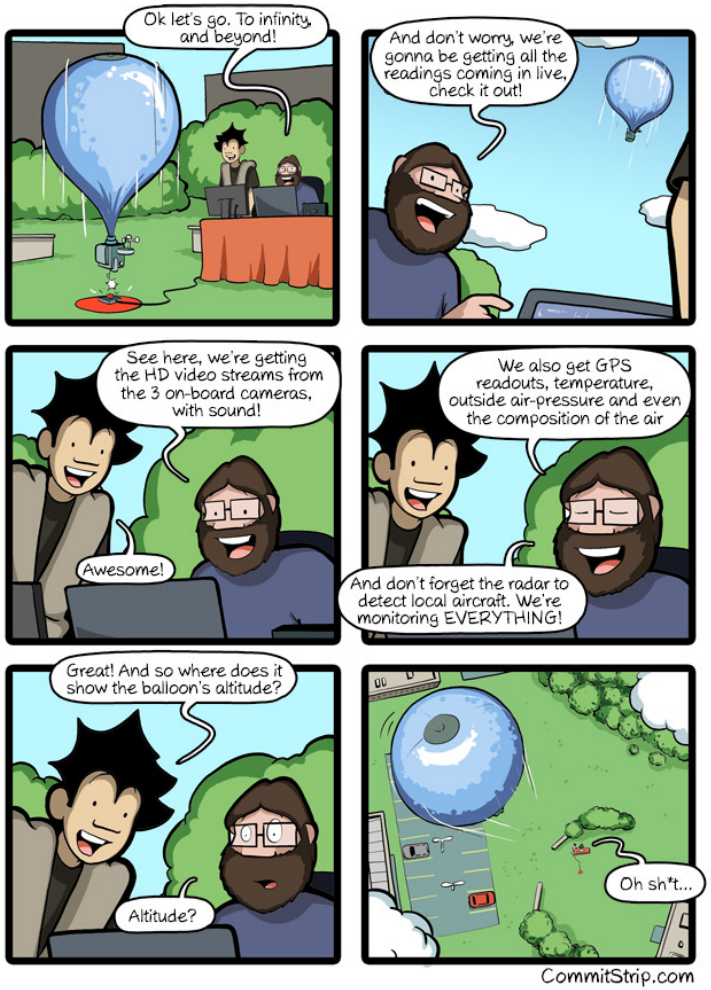
Saturation

How "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained (e.g., in a memory-constrained system, show memory; in an I/O-constrained system, show I/O). Note that many systems degrade in performance before they achieve 100% utilization, so having a utilization target is essential.

OBSERVABILITY – PATTERNS

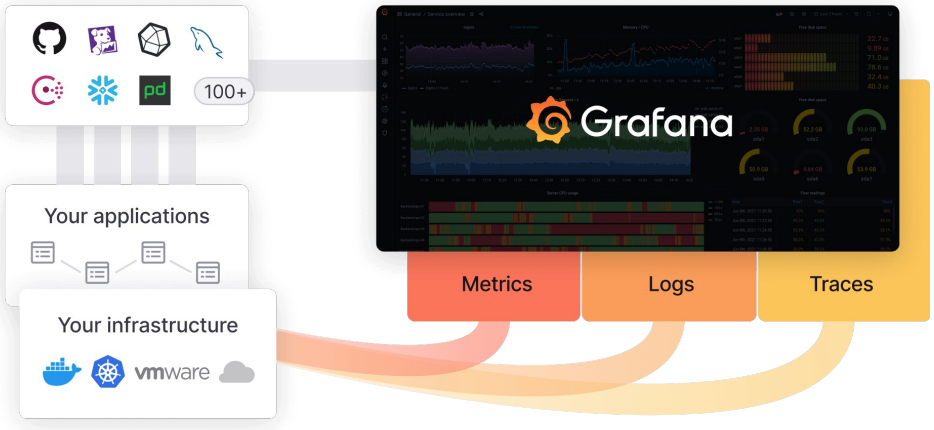
- R.E.D. Pattern for services
  - Requests / Errors / Duration
  - Services are usually request-driven systems
  - Requests: requests received per second
  - Errors: percentage of requests that returned an error

OBSERVABILITY – PATTERNS

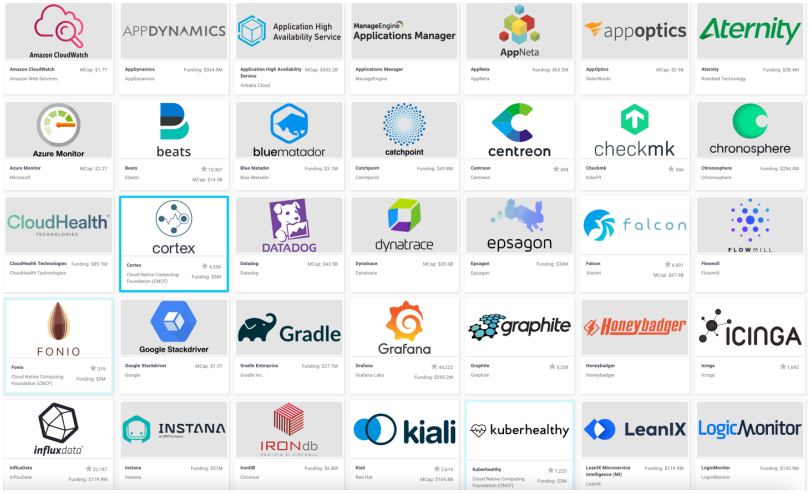


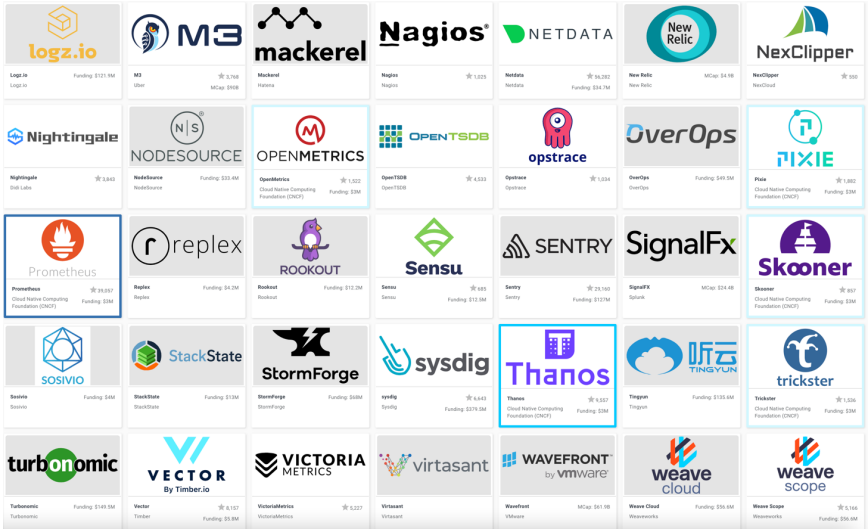
- U.S.E. Pattern for resources
  - Utilisation / Saturation / Errors
  - Infrastructure components
  - CPU, Memory, DIsks
  - Network

OBSERVABILITY – OPEN SOURCE SOLUTIONS



- Prometheus
- Grafana
- Loki
- Elastic Search
- Kibana

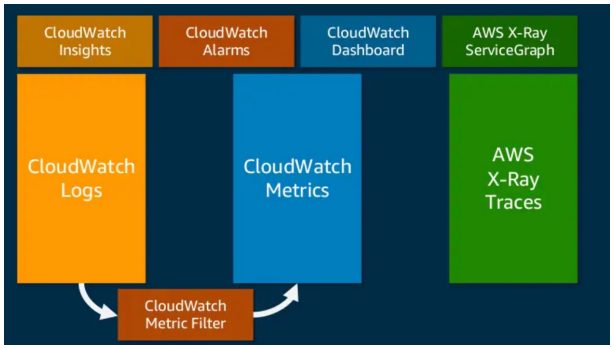


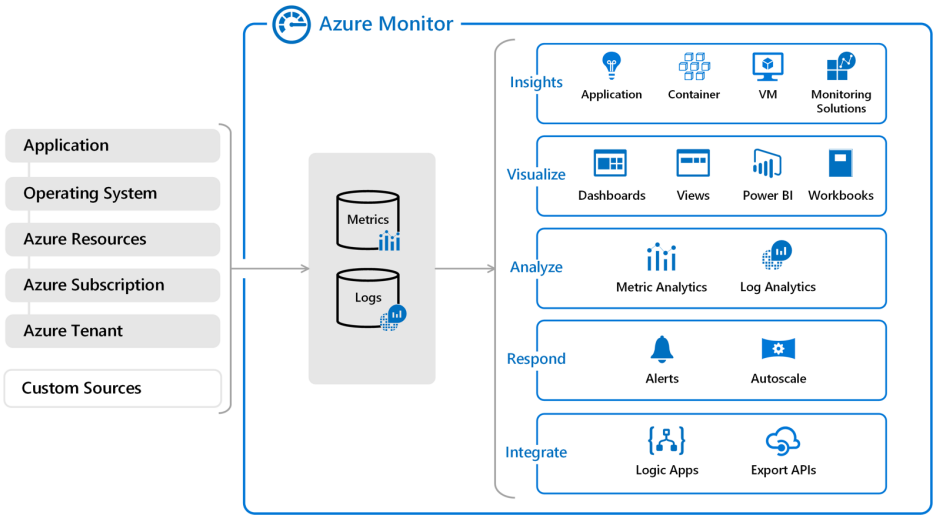


OBSERVABILITY – COMMERCIAL SOLUTIONS

- Many providers are offering Monitoring and Observability solutions
  - New Relic
  - AppDynamics
  - Dynatrace
  - Splunk
  - SolarWind
  - DataDog
  - Logz.io
  - Sumo Logic
- Offering can be based on
  - On Premises / SelfManaged licenses software
  - SaaS

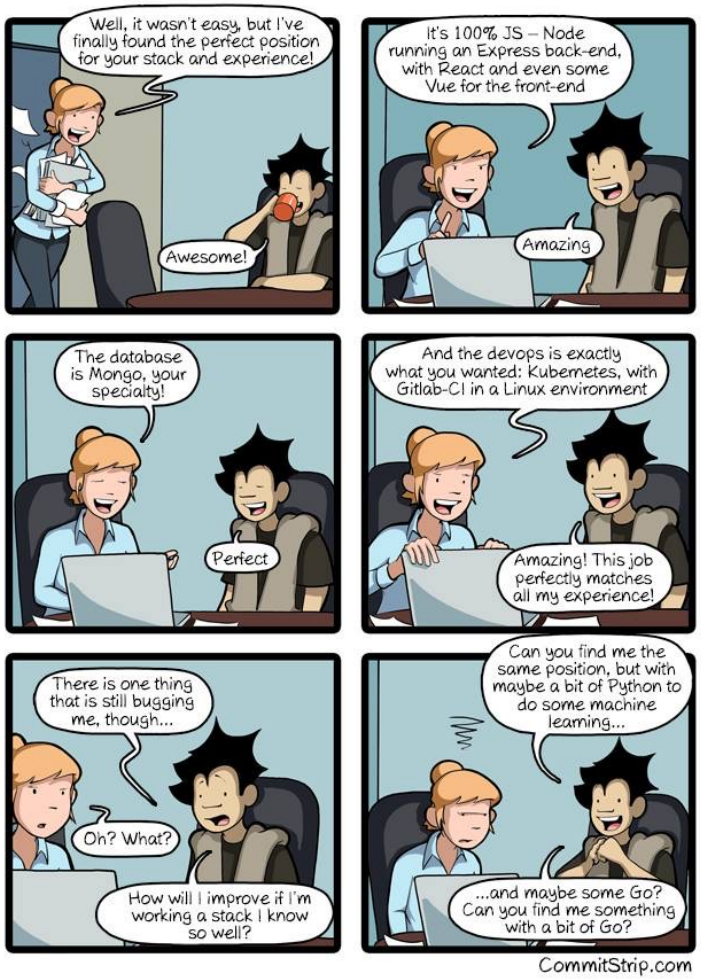
OBSERVABILITY – CLOUD SOLUTIONS





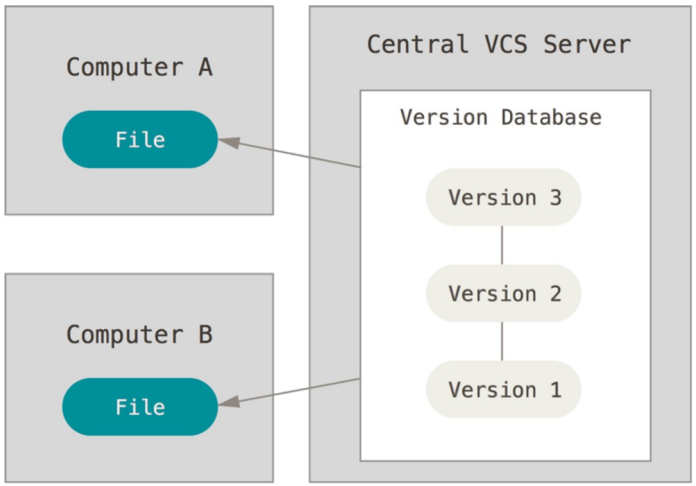
- Major Cloud Providers have an offering for integrated monitoring and observability solutions:
  - Azure Monitoring
  - AWS CloudWatch
  - Google Cloud StackDriver
  - Alibaba Cloud Monitor
- Advantage of these solutions are:
  - Out of the box integration with Cloud Services
  - SaaS Model
  - Costs management internal to the Cloud

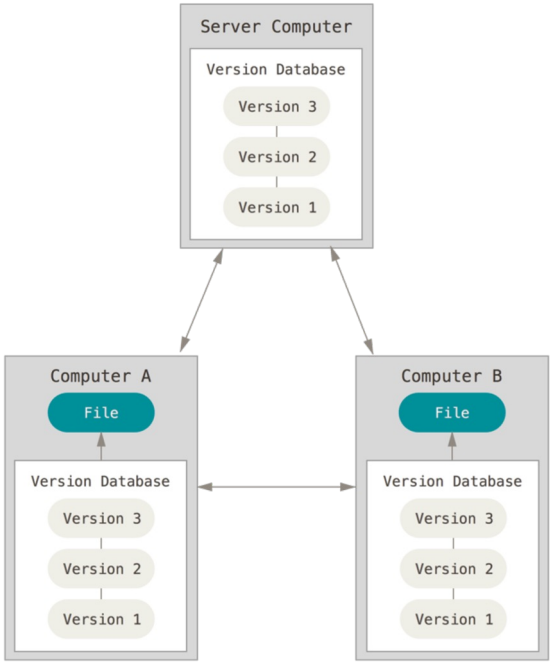
EXTRA CONTENT – GITOPS



GITFLOW TOOLS – REPO

- A version control system
- Created in 2005 by Linus Torvalds
- Strong support for non-linear development
- Distributed
- Mainly offline

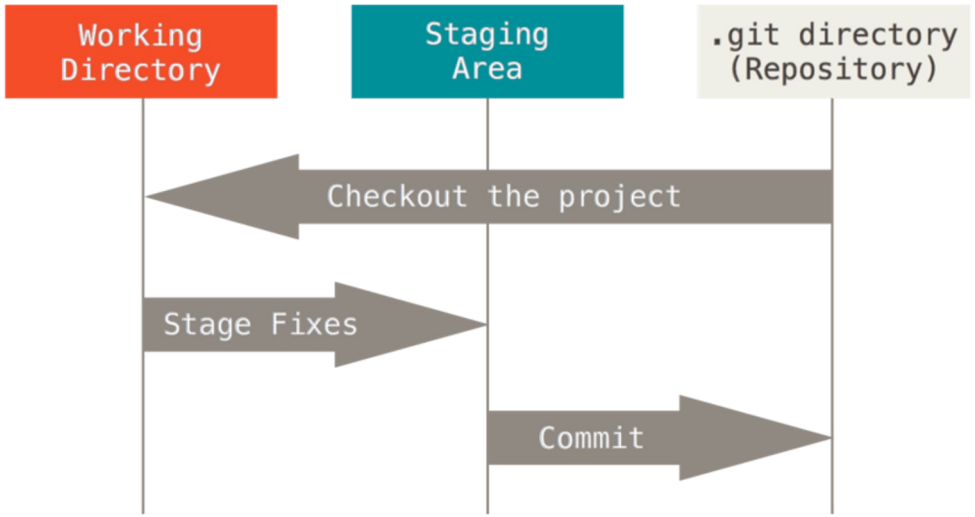




GITFLOW TOOLS – REPO

- VCS - centralized vs distributed
  - Centralized VCS relies on versioned database stored on a central server
  - Distributed VCS host the versioned database locally on each computer

GITFLOW TOOLS – REPO

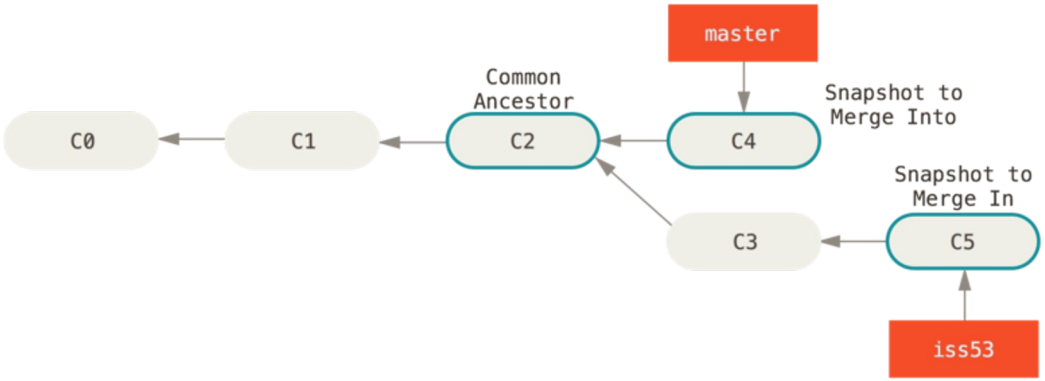


- Git - stages
  - Modified
  - Staged
  - Committed

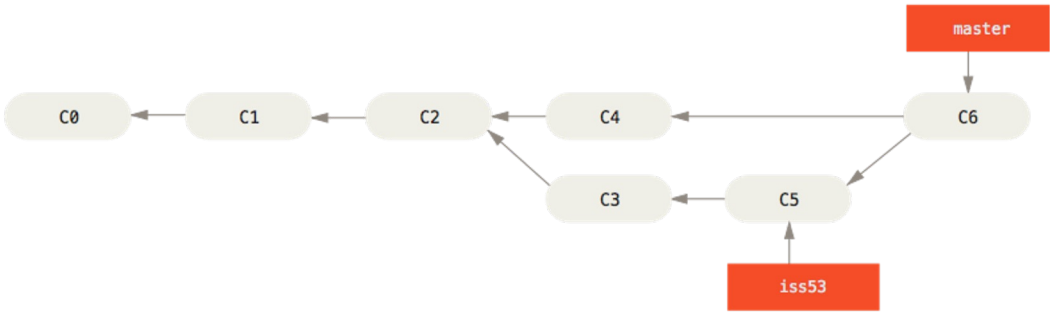
GITFLOW TOOLS – REPO

- Git - Branches
  - Pointers
  - Allows to move back in repo's history
  - Created to diverge
  - Allows to create new histories

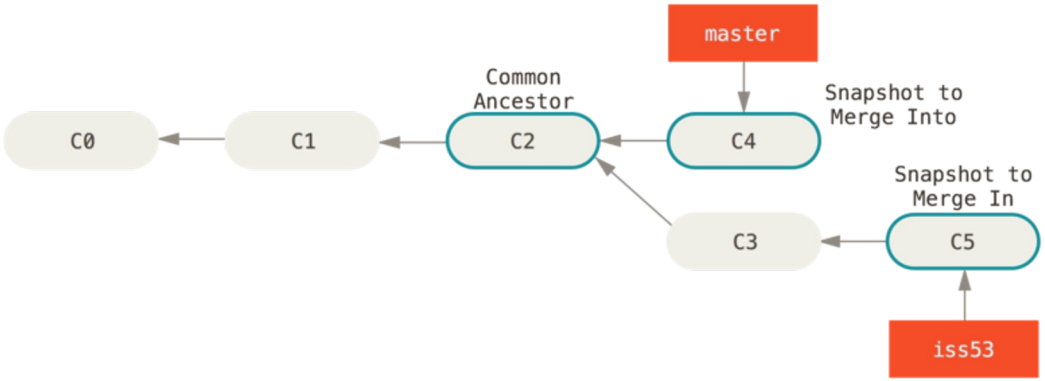
GITFLOW TOOLS – REPO



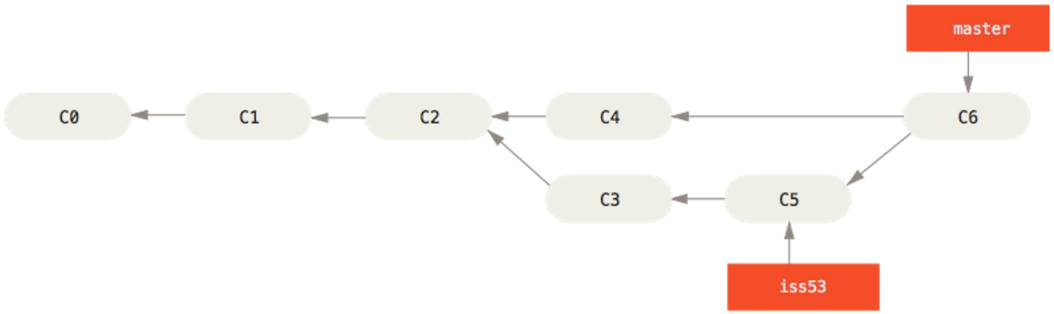
- Git - Merging
  - Relies on Common Ancestor shared by the branches
  - Create new “merge commit” that contains both branches histories



GITFLOW TOOLS – REPO



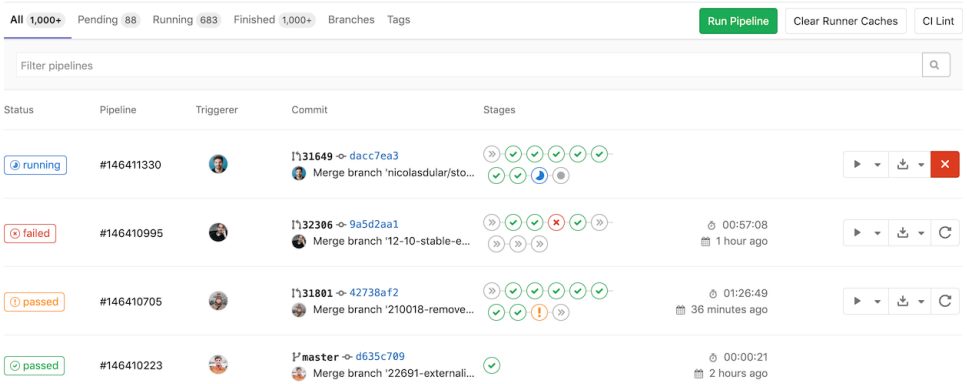
- Git - Conflicts
  - Same part(s) of the same file(s)
  - Require manual intervention
  - Special markers in the file (identified by <<<<<<<, =======, and >>>>>>>)



GITFLOW TOOLS – REPO

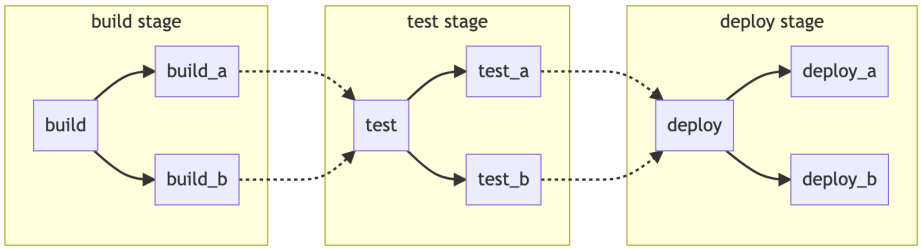
- Git - Commands
  - $ git init / $ git clone [remote URL]
  - $ git status
  - $ git stash / $ git stash pop
  - $ git add [path] / $ git rm [path]
  - $ git commit -m “[message]”
  - $ git push
  - $ git reset
  - $ git checkout / $ git merge

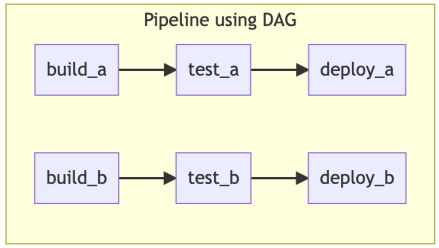
GITFLOW TOOLS – PIPELINE

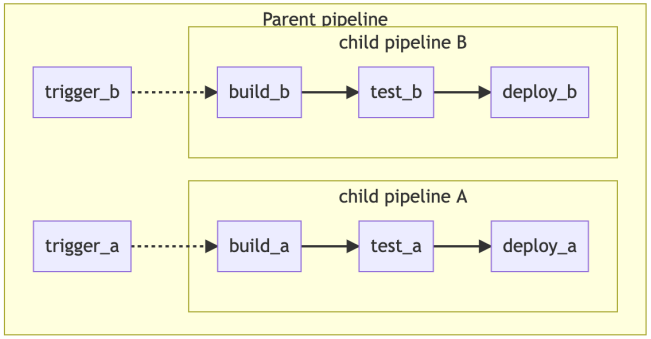


- Pipelines are the top-level component of continuous integration, delivery, and deployment.
- Pipelines comprise:
  - Jobs, which define what to do. For example, jobs that compile or test code.
  - Stages, which define when to run the jobs. For example, stages that run tests after stages that compile the code.
- Jobs are executed by runners. Multiple jobs in the same stage are executed in parallel, if there are enough concurrent runners.
- If all jobs in a stage succeed, the pipeline moves on to the next stage.
- If any job in a stage fails, the next stage is not (usually) executed and the pipeline ends early.
- In general, pipelines are executed automatically and require no intervention once created. However, there are also times when you can manually interact with a pipeline.
- A typical pipeline might consist of four stages, executed in the following order:
  - A build stage, with a job called compile.
  - A test stage, with two jobs called test1 and test2.
  - A staging stage, with a job called deploy-to-stage.
  - A production stage, with a job called deploy-to-prod.

GITFLOW TOOLS – PIPELINE







- There are three main ways to structure your pipelines, each with their own advantages. These methods can be mixed and matched if needed:
  - Basic: Good for straightforward projects where all the configuration is in one easy to find place.
  - Directed Acyclic Graph: Good for large, complex projects that need efficient execution.
  - Child/Parent Pipelines: Good for monorepos and projects with lots of independently defined components.

GITFLOW TOOLS – PIPELINE

```
stages:
  - build
  - test
  - deploy

image: alpine

build_a:
  stage: build
  script:
    - echo "This job builds something."

build_b:
  stage: build
  script:
    - echo "This job builds something else."

test_a:
  stage: test
  script:
    - echo "This job tests something. It will only run when all jobs in the"
    - echo "build stage are complete."

test_b:
  stage: test
  script:
    - echo "This job tests something else. It will only run when all jobs in the"
    - echo "build stage are complete too. It will start at about the same time as test_a."

deploy_a:
  stage: deploy
  script:
    - echo "This job deploys something. It will only run when all jobs in the"
    - echo "test stage complete."
```

GITFLOW TOOLS – PIPELINE

```yaml
stages:
  - build
  - test
  - deploy

image: alpine

build_a:
  stage: build
  script:
    - echo "This job builds something quickly."

build_b:
  stage: build
  script:
    - echo "This job builds something else slowly."

test_a:
  stage: test
  needs: [build_a]
  script:
    - echo "This test job will start as soon as build_a finishes."
    - echo "It will not wait for build_b, or other jobs in the build stage, to finish."

test_b:
  stage: test
  needs: [build_b]
  script:
    - echo "This test job will start as soon as build_b finishes."
    - echo "It will not wait for other jobs in the build stage to finish."

deploy_a:
  stage: deploy
  needs: [test_a]
  script:
    - echo "Since build_a and test_a run quickly, this deploy job can run much earlier."
    - echo "It does not need to wait for build_b or test_b."
```

GITFLOW TOOLS – PIPELINE

```yaml
stages:
  - triggers

trigger_a:
  stage: triggers
  trigger:
    include: a/.gitlab-ci.yml
  rules:
    - changes:
        - a/*

trigger_b:
  stage: triggers
  trigger:
    include: b/.gitlab-ci.yml
  rules:
    - changes:
        - b/*
```

```yaml
stages:
  - build
  - test
  - deploy

image: alpine

build_a:
  stage: build
  script:
    - echo "This job builds something."

test_a:
  stage: test
  needs: [build_a]
  script:
    - echo "This job tests something."

deploy_a:
  stage: deploy
  needs: [test_a]
  script:
    - echo "This job deploys something."
```

```yaml
stages:
  - build
  - test
  - deploy

image: alpine

build_b:
  stage: build
  script:
    - echo "This job builds something else."

test_b:
  stage: test
  needs: [build_b]
  script:
    - echo "This job tests something else."

deploy_b:
  stage: deploy
  needs: [test_b]
  script:
    - echo "This job deploys something else."
```

```yaml
build:
  stage: build
  script: ./build
  only:
    - main

test:
  stage: test
  script: ./test
  only:
    - merge_requests

deploy:
  stage: deploy
  script: ./deploy
  only:
    - main
```

```yaml
workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH && $CI_OPEN_MERGE_REQUESTS'
      when: never
    - if: '$CI_COMMIT_BRANCH'
```

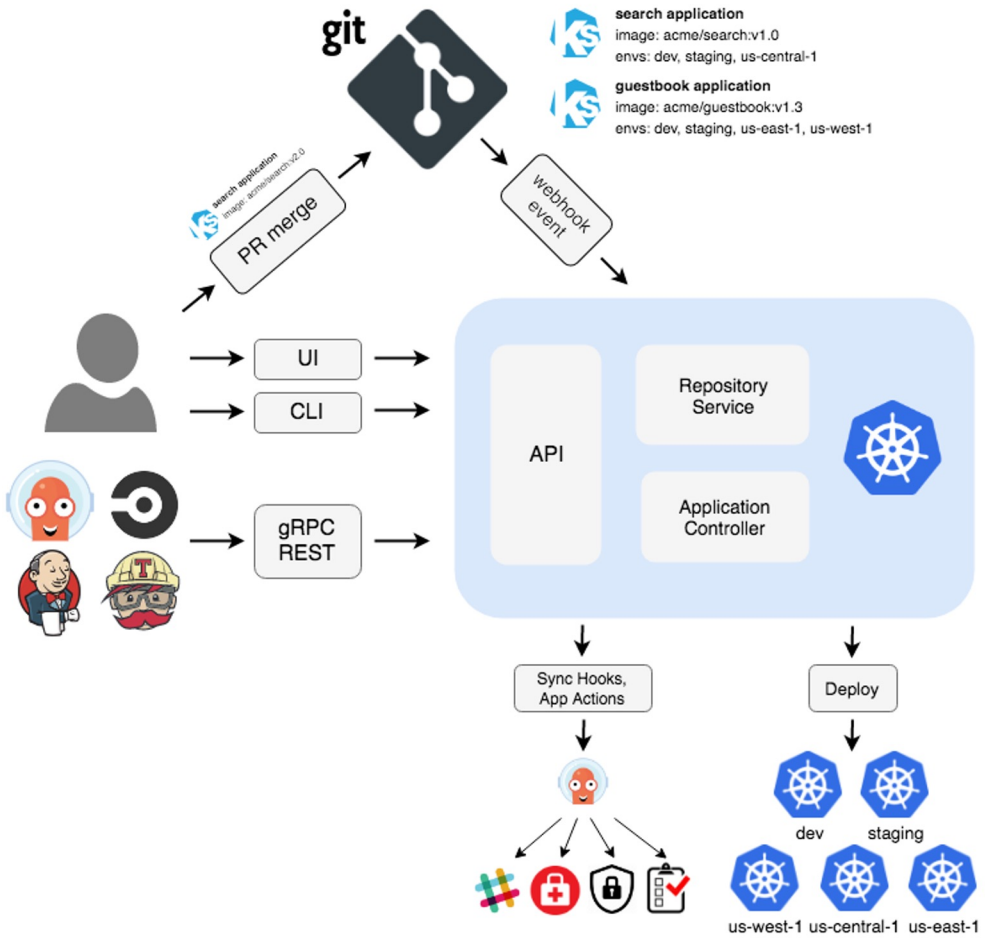
GITFLOW TOOLS – PIPELINE

- In a basic configuration, GitLab runs a pipeline each time changes are pushed to a branch.
- If you want the pipeline to run jobs only on commits associated with a merge request, you can use pipelines for merge requests.
- These pipelines are labeled as detached in the UI, and they do not have access to protected variables. Otherwise, these pipelines are the same as other pipelines.
- Pipelines for merge requests can run when you:
  - Create a new merge request.
  - Commit changes to the source branch for the merge request.
  - Select the Run pipeline button from the Pipelines tab in the merge request.
- If you use this feature with merge when pipeline succeeds, pipelines for merge requests take precedence over other pipelines.

GITFLOW TOOLS – IAC

- From Wikipedia: “Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.”
- Imperative vs Declarative approach: for declarative you specify a list of resources you would like, whereas for imperative you specify a list of commands to run to create the resources that you want
- Why Declarative:
  - Idempotency — idempotency is the ability to run the same command and achieve the same result.
  - State Management - a declarative tool needs to manage the current state. It’s entirely possible that the declarative infrastructure as code’s picture of the current state actually can differ from the reality of our state.
  - Dealing with “Configuration Drift” — Configuration drift is when infrastructure changes slowly over time. Declarative infrastructure as code will be able to adapt more easily to changes, reporting the differences and leaving it up to you to decide how to proceed.

GITFLOW TOOLS – ARGOCD



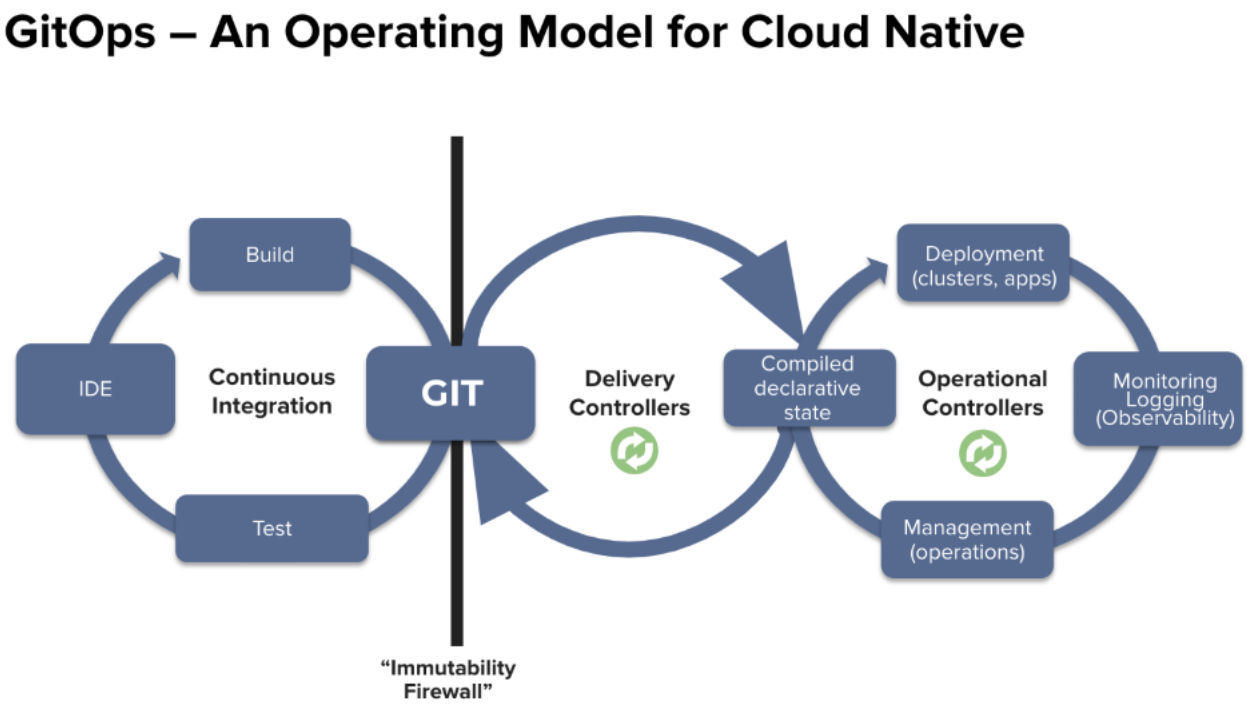
- Argo CD is implemented as a kubernetes controller which continuously monitors running applications and compares the current, live state against the desired target state (as specified in the Git repo).

- Supported manifests:
- Kustomize applications
- Helm charts
- Ksonnet application (project is terminates)
- Jsonnet files
- Plain directory of YAML/json manifests
- Any custom config management tool configured as a config management plugin

ArgoCD - Features

- *Automated deployment of applications to specified target environments*
- *Support for multiple config management/templating tools (Kustomize, Helm, Ksonnet, Jsonnet, plain-YAML)*
- *Ability to manage and deploy to multiple clusters*
- *SSO Integration (OIDC, OAuth2, LDAP, SAML 2.0, GitHub, GitLab, Microsoft, LinkedIn)*
- *Multi-tenancy and RBAC policies for authorization*
- *Rollback/Roll-anywhere to any application configuration committed in Git repository*
- *Health status analysis of application resources*
- *Automated configuration drift detection and visualization*
- *Automated or manual syncing of applications to its desired state*
- *Web UI which provides real-time view of application activity*
- *CLI for automation and CI integration*
- *Webhook integration (GitHub, BitBucket, GitLab)*
- *Access tokens for automation*
- *PreSync, Sync, PostSync hooks to support complex application rollouts (e.g.blue/green & canary upgrades)*
- *Audit trails for application events and API calls*
- *Prometheus metrics*
- *Parameter overrides for overriding ksonnet/helm parameters in Git*

Putting all together - GitOps



"DevOps is all about the cultural change in an organization to make people work better together.

GitOps is a technique to implement Continuous Delivery."

GitOps – Some definitions

Kelsey Hightower

“GitOps: versioned CI/CD on top of declarative infrastructure.
Stop scripting and start shipping.”

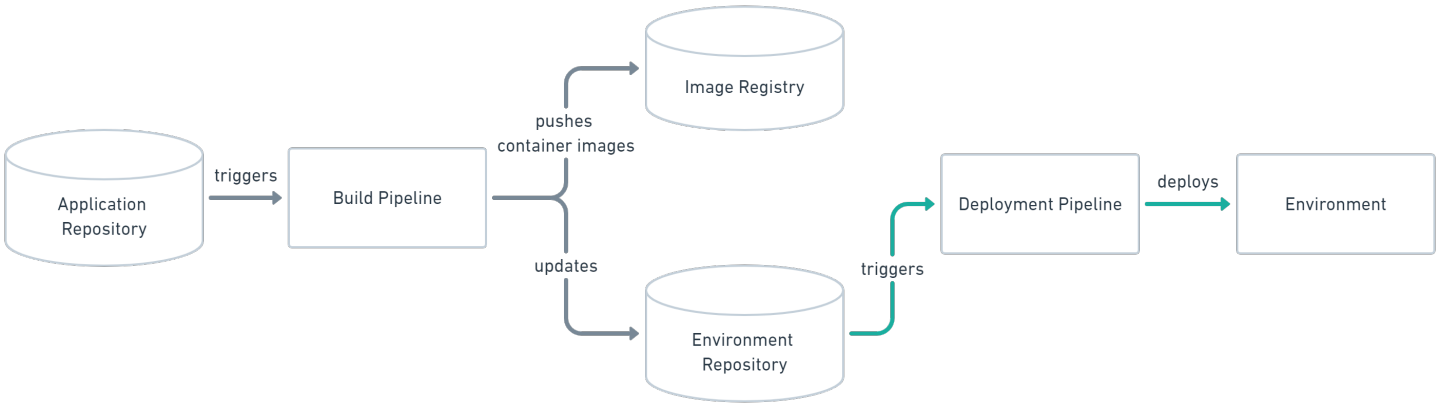
“GitOps is the best thing since configuration as code.
Git changed how we collaborate, but declarative configuration is the key to dealing with infrastructure at scale, and sets the stage for the next generation of management tools.”

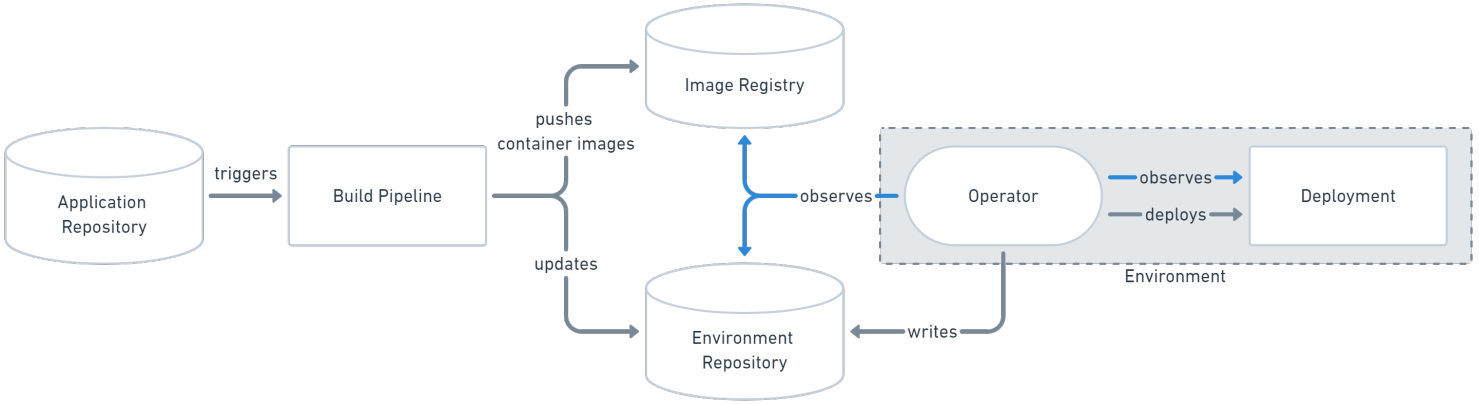
So… Why Git Ops?

- *Easy and Fast Error Recovery*
- *Easier Credential Management*
- *Self-documenting Deployments*
- *Shared Knowledge in Teams*

GitOps organizes the deployment process around code repositories as the central element. There are at least two repositories: the application repository and the environment configuration repository. The application repository contains the source code of the application and the deployment manifests to deploy the application. The environment configuration repository contains all deployment manifests of the currently desired infrastructure of an deployment environment.

Push vs Pull


Application Repository
triggers
Build Pipeline
pushes container images
Image Registry
updates
Environment Repository
triggers
Deployment Pipeline
deploys
Environment


Application Repository
triggers
Build Pipeline
pushes container images
Image Registry
updates
Environment Repository
observes
Operator
observes
deploys
Deployment
Environment
writes

The context of GitOps

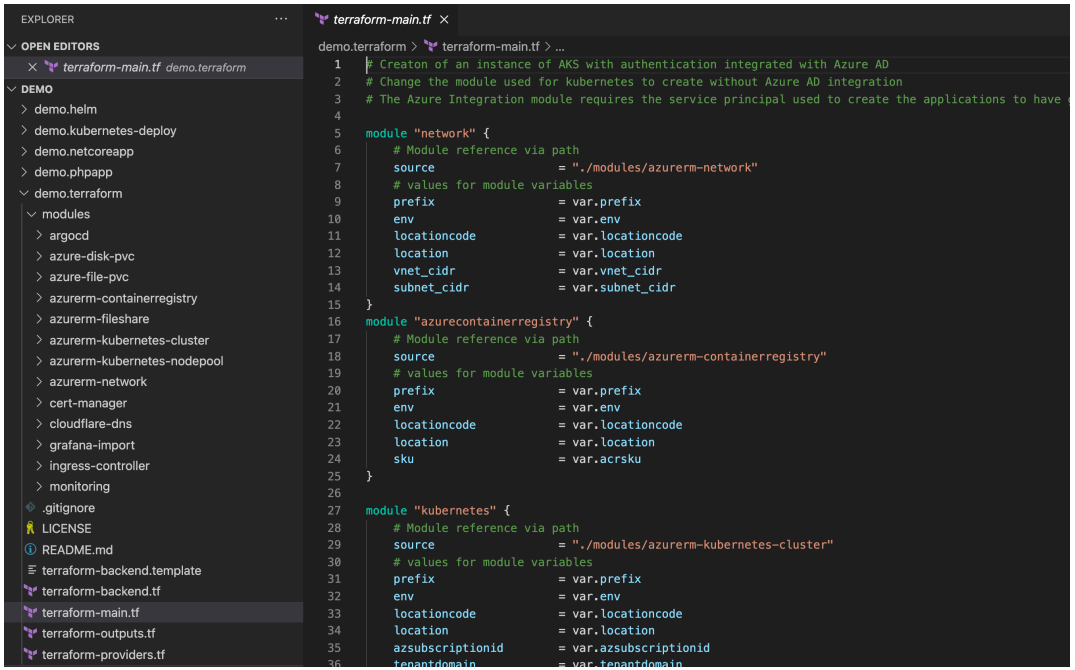
The entire system described declaratively.

The canonical desired system state versioned in Git.

Approved changes that can be automatically applied to the system.

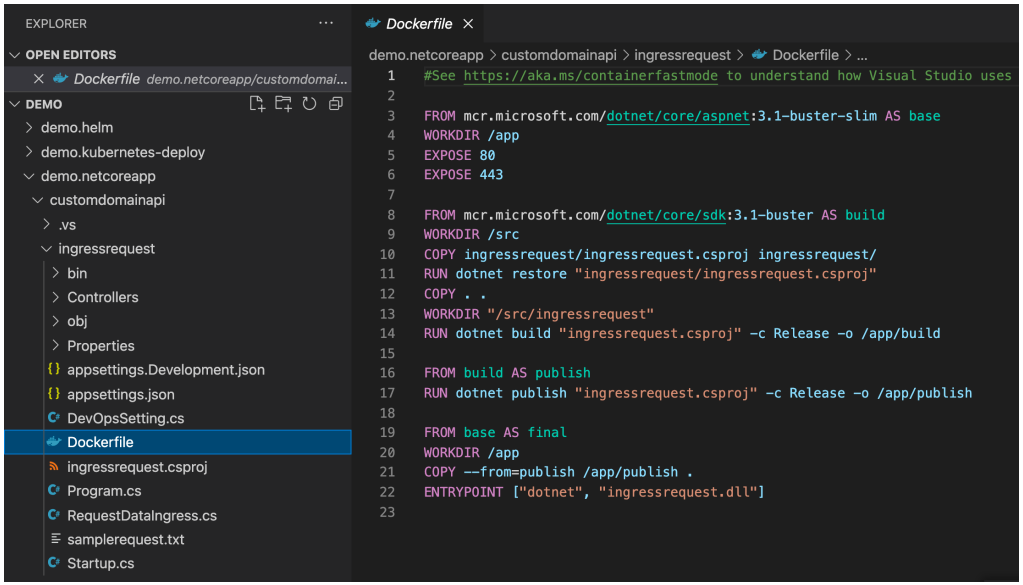
Software agents to ensure correctness and alert on divergence.

DEMO



- Using IaC tools (Terraform in our case):
  - Create a terraform project
  - Structure for resources and modules
  - State management and backend
  - The “main.tf” file
  - Input Variables
  - How to validate the project

DEMO



- Container build and application definition
  - Dockerfile structure
  - Docker build commands
  - Helm Chart
    - Templates
    - Variables

DEMO

demo.terraform > ! .git-lab-ci.yml > YAML > {} variables > PLAN

```yaml
image:
  name: registry.gitlab.com/gitlab-org/gitlab-build-images:terraform
  entrypoint:
    - '/usr/bin/env'
    - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'

# Default output file for Terraform plan
variables:
  PLAN: plan.tfplan
  JSON_PLAN_FILE: tfplan.json

cache:
  paths:
    - .terraform
    - .terraform.lock.hcl

before_script:
  - alias convert_report="jq -r '([.resource_changes[]?.change.actions?]|flatten)|{\"create\":(map(select(.==\"create\"
  - terraform --version
  - terraform init

stages:
  - validate
  - build
  - test
  - deploy

validate:
  stage: validate
  script:
    - terraform validate

plan:
  stage: build
  script:
    - terraform plan -out=$PLAN
    - "terraform show --json $PLAN | convert_report > $JSON_PLAN_FILE"
  artifacts:
```

- Automatation
  - Runners and pipelines
  - ArgoCD
    - Application
    - Triggers
    - State management

REAL LIFE CI/CD