DevOps and SRE

7.2.1

Introduction to DevOps and SRE

For full-stack automation to be truly effective, it requires changes to organizational culture, including breaking down the historical divides between Development (Dev) and Operations (Ops).

This topic discusses the history of the DevOps approach and the core principles of successful DevOps organizations.

7.2.2

DevOps Divide

Historically, creating applications was the job of software developers (Dev), and ensuring that apps work for users and the business has been the specialized province of IT operations (Ops).

Characteristic	Dev	Ops
Cares about:	Bespoke applications and how they work	Applications and how they ru Infrastructure (historically, ha network, commodity services security, scaling, and mainter
Business treats as:	Profit center: demands resources	Cost center: provides and ac resources
Participates in on-call rotation:	Occasionally (and is disturbed only when issues are escalated to dev)	Regularly (point of spear)
Performance measured:	Abstractly (including bad metrics, like lines of code/day)	Concretely (SLA compliance, resolved)

Characteristic	Dev	Ops
Skills required:	More deep than broad: Languages, APIs, architecture, tools, process, "frontend," "backend," etc.	More broad than deep: Confi administration, OS, manufact admin CLIs/APIs, shell, auton security, etc.
Agility required:	Much! Move fast, innovate, break things, fix later.	Not so much! Investments mu extensively justified, expectat Longish timescales are norma no than yes.

In the traditional, pre-virtualization, enterprise IT ecosystem, separating Dev from Ops seemed sensible. As long as infrastructure was based on investment in capital equipment (manually installed, provisioned, configured, and managed hardware), it required gatekeepers. The Ops gatekeepers worked on a different timescale from users and developers, and had different incentives and concerns.

Legacy bottlenecks

Traditionally, project resourcing or system scaling would be plan-driven, rather than demand-driven. Requisitioning, purchasing, installing, provisioning, and configuring servers or network capacity and services for a project could take months. With limited physical resources, resource-sharing was common.

The lack of simple ways to set up and tear down isolated environments and connectivity meant that organizations tended to create long-running systems that became single points of failure. Mixed-use networks were difficult to secure and required meticulous ongoing management.

Colloquially, such long-running, elaborate systems were referred to as "pets" because people would name them and care for that one system. The alternative is "cattle", which are stripped-down, ephemeral workloads and virtual infrastructure built and torn down by automation. This method ensures that there is a new system (or “cow”) available to take over the work.

Fusing Dev and Ops

Sophisticated tech organizations have been migrating away from these historic extremes for generations. The process accelerated with widespread adoption of server virtualization, cloud, and Agile software development. In the early 2000s, there began a movement to treat Dev and Ops as a single entity:

- Make coders responsible for deployment and maintenance
- Treat virtualized infrastructure as code

7.2.3

Evolution of DevOps

DevOps evolved and continues to evolve in many places in parallel. Some key events have shaped the

discipline as we know it today.

Defining Moments 1: Site Reliability Engineering (SRE)

By 2003, the world's biggest and most-advanced internet companies had significantly adopted virtualization. They were dealing with large data centers and applications operated on a massive scale. There were failures that resulted in Dev vs. Ops finger-pointing, fast-growing and perpetually insufficient organizational headcounts, and on-call stress.

Google was among the first companies to understand and institutionalize a new kind of hybrid Dev+Ops job description. This was the Site Reliability Engineer (SRE). The role of the SRE is intended to fuse the disciplines and skills of Dev and Ops, creating a new specialty and best-practices playbook for doing Ops with software methods.

The SRE approach was adopted by many other companies. This approach is based on:

- Shared responsibility
- Embracing risk
- Acknowledgment of failure as normal
- Commitment to use automation to reduce or eliminate "toil"
- Measurement of everything
- Qualifying success in terms of meeting quantitative service-level objectives

Defining Moments 2: Debois and “Agile Infrastructure”

At Agile 2008, Belgian developer Patrick Debois gave a presentation called Agile Infrastructure and Operations. His presentation discussed how to apply Developer methods to Ops while maintaining that Dev and Ops should remain separate. Nevertheless, the following year, Debois went on to found the DevOpsDays event series.

Debois's presentation was influential in advancing discussions around automating virtual (and physical) infrastructure, using version-control (such as Git) to store infrastructure deployment code (procedural or declarative), and applying Agile methods to the development and maintenance of infrastructure-level solutions.

Defining Moments 3: Allspaw and Hammond

By the late 2000s, DevOps was increasing in popularity. Among the first compelling illustrations of its value was a presentation by John Allspaw and Paul Hammond at VelocityConf in 2009. In this presentation, the authors outline a simple set of DevOps best practices founded on the idea that both Dev and Ops cooperatively *enable the business*. These best practices include:

- Automated infrastructure
- Shared version control
- Single-step builds and deployments

The presentation also describes automation, teamwork, responsibility-sharing, transparency, trust, mutual accountability, and communications practices that have since become commonplace among DevOps/SRE practitioners. These practices include CI/CD, automated testing, metrics, SLOs/SLAs, and blame-free embrace of risk and inevitable failure.

7.2.4

Core Principles of DevOps

DevOps/SRE have many core principles and best practices:

- A focus on automation
- The idea that "failure is normal"
- A reframing of "availability" in terms of what a business can tolerate

Just as Agile Development can be seen as a method for defining and controlling management expectations for software projects, DevOps can be viewed as a way of structuring a healthy working culture for the technology-based parts of businesses. It can also be viewed as a way of reducing costs. By making failure normal and automating mitigation, work can move away from expensive and stressful on-call hours and into planned workday schedules.

Automation, avoiding toil, and retaining talent

Automation delivers speed and repeatability while eliminating manual, repetitive labor. This enables technical talent to spend time solving new problems, increasing the value to the business in the time spent.

DevOps/SRE practitioners are often expected to devote a significant fraction of working hours (50% or more in some scenarios) to delivering new operational capabilities and engineering reliable scaling, including development of automation tooling. This reduces hours spent on-call and intervening manually to understand and fix issues.

Acquisition and retention of technical talent requires organizations to cooperate with their innovators to minimize the boredom and stress of low-value labor and on-call work, and the risk of confronting inevitable technology failures in "fire drill" mode.

rather than as a cost-center.

Failure is normal

The assumption that failures will occur does influence software design methodology. DevOps must build products and platforms with greater resiliency, higher latency tolerance where possible, better self-monitoring, logging, end-user error messaging, and self-healing.

When failures do occur and DevOps must intervene, the resulting activities should be viewed not simply as repair work, but as research to identify and rank procedural candidates for new rounds of automation.

SLOs, SLIs, and error budgets

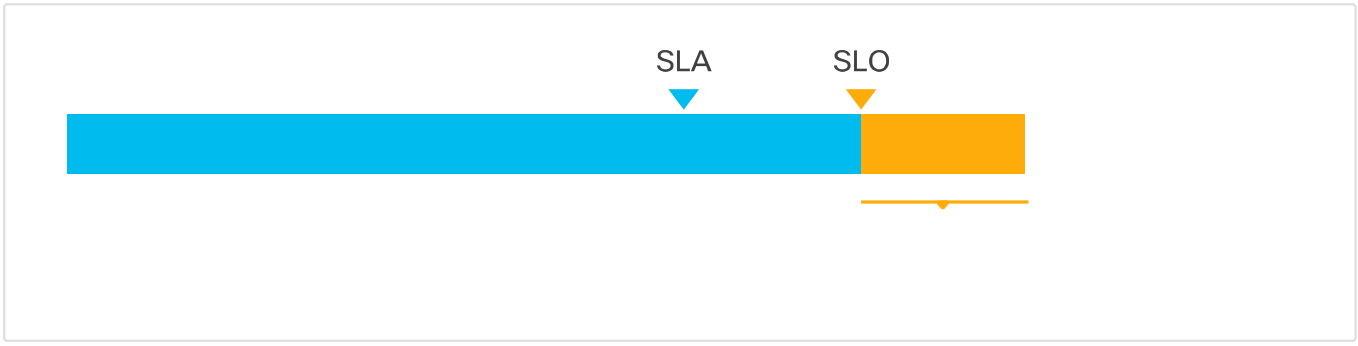
Critical to DevOps/SRE culture are two linked ideas: 1. DevOps must deliver measurable, agreed-upon business value, AND 2. the statistical reality of doing so perfectly is impossible. These ideas are codified in a Service Level Objective (SLO) that is defined in terms of real metrics called Service Level Indicators (SLIs).

SLIs are engineered to map to the practical reality of delivering a service to customers: they may represent a single threshold or provide more sophisticated bracketing to further classify outlier results. For example, an SLI might state that 99% of requests will be served within 50 milliseconds, and may also require capturing information such as whether a single >50msec request completes at all, or whether a particular request has failed for your biggest customer.

SLO/SLI methodology permits cheaper, more rapid delivery of business value by removing the obligation to seek perfection in favor of building what is "good enough". It can also influence the pace, scope, and other aspects of development to ensure and improve adequacy.

One way of modeling SLO/SLI results requires establishing a so-called "error budget" for a service for a given period of time (day, week, month, quarter, etc.), and then subtracting failures to achieve SLO from this value. If error budgets are exceeded, reasonable decisions can be made, such as slowing the pace of releases until sources of error are determined, and specific fixes are made and tested.

SLA vs. SLO and Error Budget



For a given service, it makes sense to commit to delivering well within capacity, but then overdeliver. SLAs (external agreements) are best set to where they will be easy to fulfill. SLOs (targets for actual performance) can be set higher. The error budget is the difference between SLO and 100% availability.

7.2.5

DevOps and SRE Summary

You have seen how DevOps/SRE is co-evolving with technologies like virtualization and containerization, enabling a unified approach and unified tool set to support coordinated application and infrastructure engineering.

Next, you will learn about some of the mechanics of infrastructure automation.

7.1
Automating Infrastructure with Cisco

7.3
Basic Automation Scripting