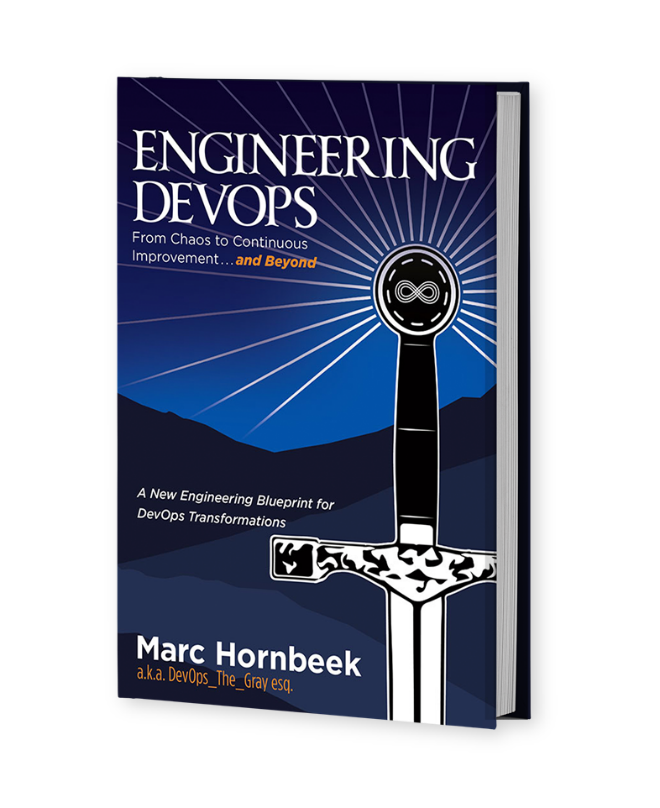
Site Reliability Engineering

This paper is derived from selections from my book.



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from [**https://engineeringdevops.com/documents**](https://engineeringdevops.com/documents)

About the Author:

**Marc Hornbeek a.k.a.** DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

August 9, 2020

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**How do you Engineer Site Reliability for DevOps?**

**Site Reliability Engineering (SRE)**

According to the book Site Reliability Engineering: “How Google Runs Production Systems”, “SRE teams are responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of services. SREs codify rules of engagement and principles for how they interact with their environment.”

Site Reliability Engineering (SRE) compliments DevOps by measuring and achieving reliability of applications and services working on production and DevOps infrastructure in a prescribed manner using error budgets, team relationships brokered by an error budget, Ops-as-code, and the use of reliability to control deployments.

**Why Is SRE Important to DevOps Engineering?**

SRE emphasizes the following objectives that are important to DevOps engineering:

• Assure availability of applications and services

• Assure error rate and latency meet Service Level Agreements (SLAs)

• Enable large-scale systems while controlling risk

• Create operational cost savings by automation and policies

• Improve skills of developers and SREs

• Resolve conflicts between Dev and SRE

• Improve capacity planning and provisioning

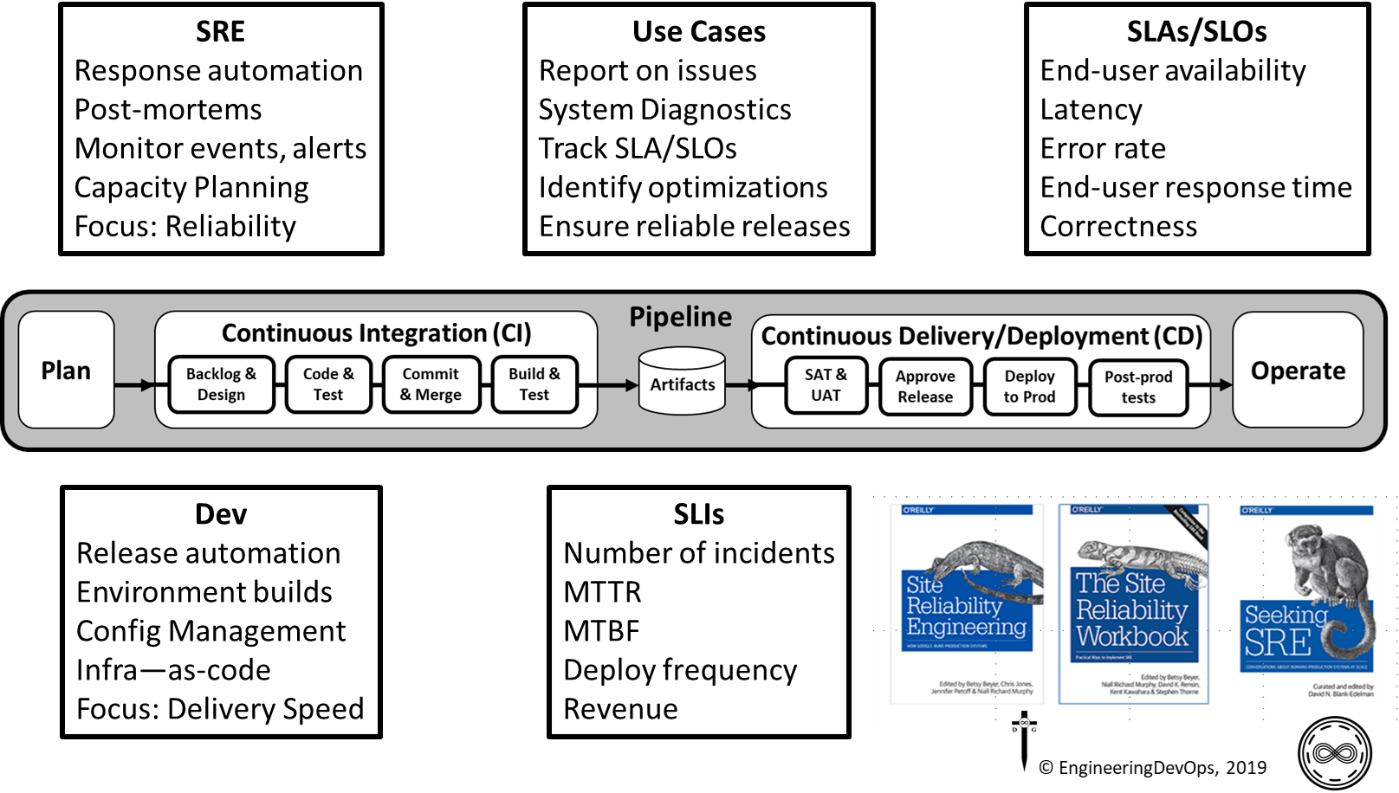
**How Does SRE Work with DevOps?**

Automation improves problems caused by human fatigue, familiarity/contempt, and inattention to highly repetitive tasks. As a result, both release velocity and safety increase. An outage is an expected part of the process of innovation and an occurrence that Dev and SRE teams manage rather than fear. The SRE role is defined as vital part of the organization.

Real-time collaboration and communication solutions like Slack are used extensively in organizations that implement SRE.

As indicated in **Figure 1 —Site Reliability Engineering (SRE) Blueprint**, a key tenet of SRE is an absolute focus on **Service Level Indicators (SLIs)**, **Service Level Objectives (SLOs),** and **Service Level Agreements (SLAs)** as a contract between Dev and SRE teams.

Service metrics such as those shown in **Figure 1—Site Reliability Engineering (SRE) Blueprint** are selected for continuous monitoring and action. An error budget (= 1-availability) for internal Service Level Objectives (SLO)s are established for each service. Products are required to work within an error budget. Tactics such as phased rollouts and 1% experiments are used to keep within error budgets. Automation is used to implementing progressive rollouts, detect problems, and implement roll-back changes safely when problems arise, because outages are primarily caused by changes in a live system. The SRE team is responsible for capacity planning and provisioning. Regular load testing of the system is used to correlate raw capacity (servers, disks, and so on) to service capacity.



**Figure 1—Site Reliability Engineering (SRE) Blueprint**

The following recommended engineering practices for Site Reliability Engineering (SRE) are presented in the following categories:

* Culture
* Toil Reduction
* SLAs/SLOs/SLIs
* Measurements
* Anti-Fragility
* Work Sharing
* Deployments
* Performance Management
* Incident Management

**Culture**

Successful organizations implementing SRE practices have a culture of continuously improving services and systems that support production to ensure services continue to be reliable as they scale. SREs deliberately and pro-actively work to Shift-left “Wisdom of Production”, to educate and facilitate operations viewpoints and requirements into services development team’s knowledge and activities. The "wisdom of production" informs better system design and behavior.

To accomplish this, SRE's spend 50% of their time doing "ops" related work such as issue resolution, and 50% of their time on development tasks such as new features, scaling or automation.

**Toil Reduction**

SREs work to reduce non-value-added production work by employing tools and automating toilsome tasks. Automation improves problems and reduces Toil, caused by human fatigue, familiarity/contempt, and inattention to highly repetitive tasks. SREs are careful to decide what to automate, and how to automate it using an engineering-based approach to problems rather than just toiling at them over and over.

**SLAs/SLOs/SLIs**

A key tenet of SRE ensures that each **Service Level Agreements (SLA)** contract with customers are met. To accomplish this, **Service Level Indicators (SLIs)** are constantly monitored, to ensure internal contracts with application development teams, called **Service Level Objectives (SLOs),** including metrics such as service availability, latency, and response times, are met. An error budget (= 1-availability) for internal Service Level Objectives (SLO)s are established for each service. Products are required to work within an error budget.

**Measurements**

Monitoring, alerting and automation of responses to events is a large part of successful SRE. SREs determine where in application and infrastructure architectures that instrumentation is needed to obtain useful data needed for service monitoring. Telemetry is used to communicate monitoring data to a core monitoring system that generates alerts, dashboard displays and analysis. Best practices involve resolving monitoring and alert data into higher level observability information that correlates to user requirements and to ensure SLOs are met.

**Anti-Fragility**

Successful SRE includes continuously working to ensure services and infrastructures that services depend on are resilient to failures. Tactics and strategies involve using Fire Drills, Chaos Monkey, continuous security measures, operations-as-code and automation to proactively determine failure modes and implement defenses against the most likely, vulnerable and the highest risks areas.

**Work Sharing**

SRE's share skill sets and tasks with product development teams to help remove boundaries between “application development” and “production”. Technical debt is worked in small increments. Goals for work-load % for Ops, Dev and On-call work are explicitly managed. Real-time collaboration and communication solutions like Slack are used extensively in organizations that implement SRE.

**Deployments**

SREs look for ways to avoid late problem discovery and improve MTTR (mean time to repair) by deploying changes incrementally using Canary, A/B, and Green Blue deployment methods, automation scripts, testing and monitoring. Tactics such as phased rollouts and 1% experiments are used to keep within error budgets. Automation is used to implement progressive rollouts, detect problems, and implement roll-back changes safely when problems arise, because outages are primarily caused by changes in a live system. Both release velocity and safety increase with successful SRE.

**Performance Management**

The SRE team is responsible for capacity planning and provisioning. Regular load testing of the system is used to correlate raw capacity (servers, disks, and so on) to service capacity. Application Performance Monitoring (APM) together with testing are used to determine the capability to scale an application and its infrastructure. Tactics such as automation and auto-scaling are used to prepare an applications infrastructure for anticipated variable load demands.

**Incident Management**

An outage is an expected part of the process of innovation and an occurrence that Dev and SRE teams manage rather than fear. This is accomplished using tactics such as incident response protocols, automated as much as possible. Ops and Dev teams help each other by sharing workloads an on-call rotations. Typically, not more than 25% of individual workloads. Blameless retrospectives conducted after incidents that are meet certain criterion help to prepare the organization to eliminate the re-occurrence or reduce the impact of similar incidents.

**What Is Needed to Engineer SRE with DevOps?**

The following steps are recommended for implementing SRE:

• SRE and Site Reliability Champion roles are defined. “Site Reliability Engineer = Software Engineer + Systems Enthusiast.” “I’m going to take the time to automate this right now and stop anyone else from having to do this painful thing.”

• SRE teams include software developers with strong operations knowledge or IT operations people with strong software development skills.

• Codify rules of engagement and principles for SRE teams.

• Cap the aggregate Ops work for all SREs.

• Standardization of both tools and processes enable relatively small teams of SREs to support larger product teams.

• Monitor solutions with software interpreting, only notifying humans when they need to act.

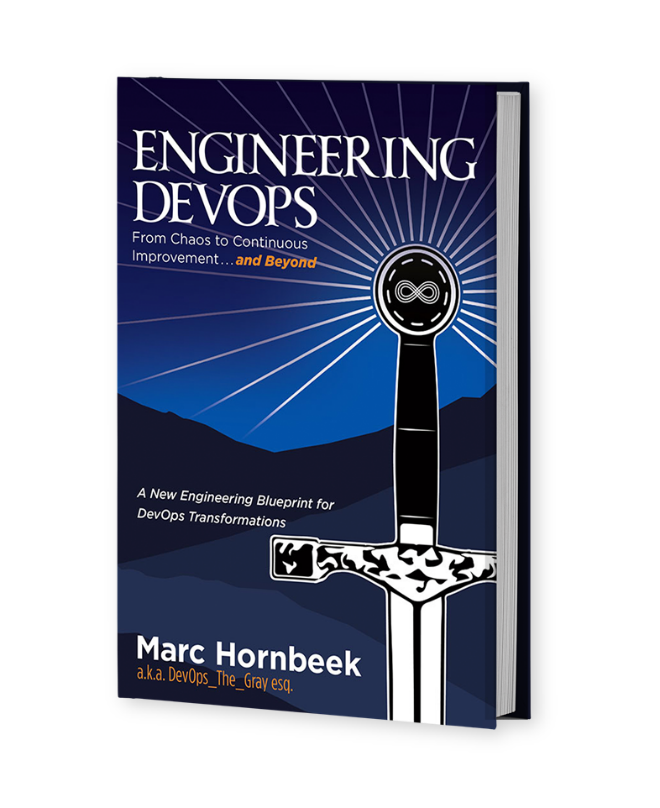
• Playbooks record the recommended engineering practices for human response actions.

• Automate progressive rollouts, problem detection, and roll-backs.

**Summary**

In summary, the answer to the question: “**How do you engineer Site Reliability for DevOps?”** has been explained in this white paper. Site Reliability Engineers (SREs) and Site Reliability Engineering (SRE) practices provide visibility and control over functioning DevOps environments. Site Reliability Engineering (SRE) compliments DevOps by measuring and achieving reliability of applications and services working on production and DevOps infrastructure in a prescribed manner using error budgets, team relationships brokered by an error budget, Ops-as-code, and the use of reliability to control deployments. Pratice categories listed in this chapter – namely Culture, Toil Reduction, SLAs/SLOs/SLIs, Measurements, Anti-Fragility, Work Sharing, Deployments, Performance Management, and Incident Management, are typically covered well in successful SRE environments.

**Learn More**



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from <https://engineeringdevops.com/documents>

About the Author:

**Marc Hornbeek a.k.a.** DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevOps 2019, 2020

All rights reserved