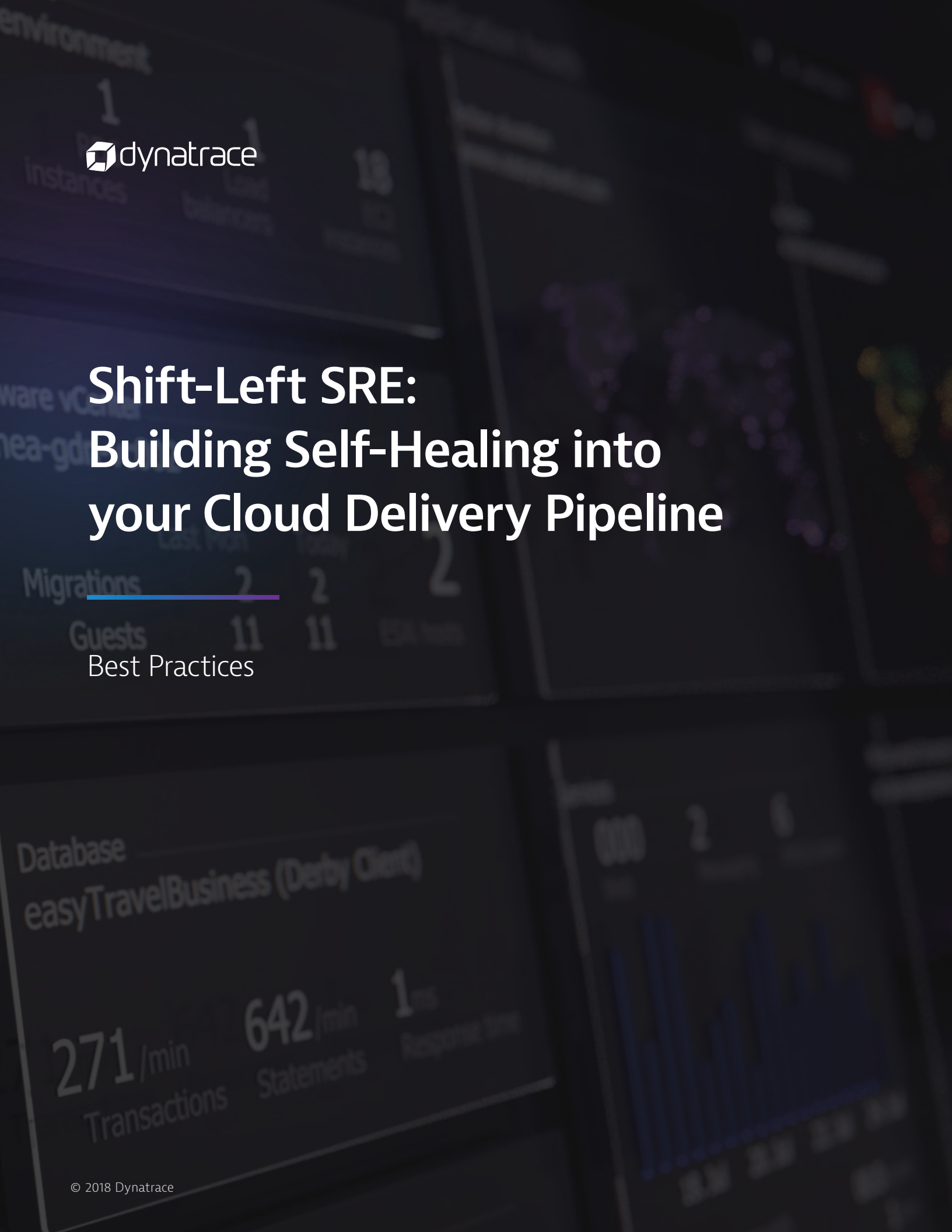

dynatrace
Shift-Left SRE:
Building Self-Healing into
your Cloud Delivery Pipeline
Best Practices
© 2018 Dynatrace

Overview

Site Reliability Engineering is an exciting discipline in our industry. There are many aspects to building reliable and resilient systems which I won't be able to cover in a single blog post. What I learned from many conversations with enterprise organizations is that the term "self-healing" is getting everybody excited. That's also why I put it into the title, to get you "excited"! However—I have at least two major problems with the way self-healing is sold and thought of:

Andreas Grabner has 20+ years of experience as a software developer, tester and architect and is an advocate for high-performing cloud scale applications. He is a regular contributor to the DevOps community, a frequent speaker at technology conferences and regularly publishes articles on blog.dynatrace.com. You can follow him on Twitter: @grabnerandi

1 Self-healing as a term is misleading!

I don't think that there are any real self-healing systems. I think we have gotten better to automate remediation actions—and by leveraging modern monitoring tools we can execute specific remediation actions in a much smarter and efficient way. Hence, I believe we should talk about Smart- or Auto-Remediation and not necessarily Self-Healing as it is misleading!

2 Self-healing is not just an operations activity!

In most of my discussions around self-healing, I talk with Ops teams that try to fix issues that materialized in production. While it is very important to keep our systems running even under "chaotic" conditions in production, I strongly believe we should invest just as much time and effort in preventing these issues ever making it into production. That's where my "Shift-Left SRE" term comes from as I believe we must bake resiliency into our delivery pipelines.

I presented my thoughts on this topic at **"AWS re:Invent 2018. It was called Shift-Left SRE: Self-Healing with Lambda!"**

While my talk was targeted for an AWS crowd, the concept is applicable for any platform, cloud, or technology stack.

I structured my presentation in 4 parts which is the same way I'd like to organize this paper:

1) Remediation Use Cases

2) PREVENT in CI/CD vs REPAIR in Prod

3) "Auto Remediation as Code"

4) The "Unbreakable Delivery Pipeline"

Let's dig into it!

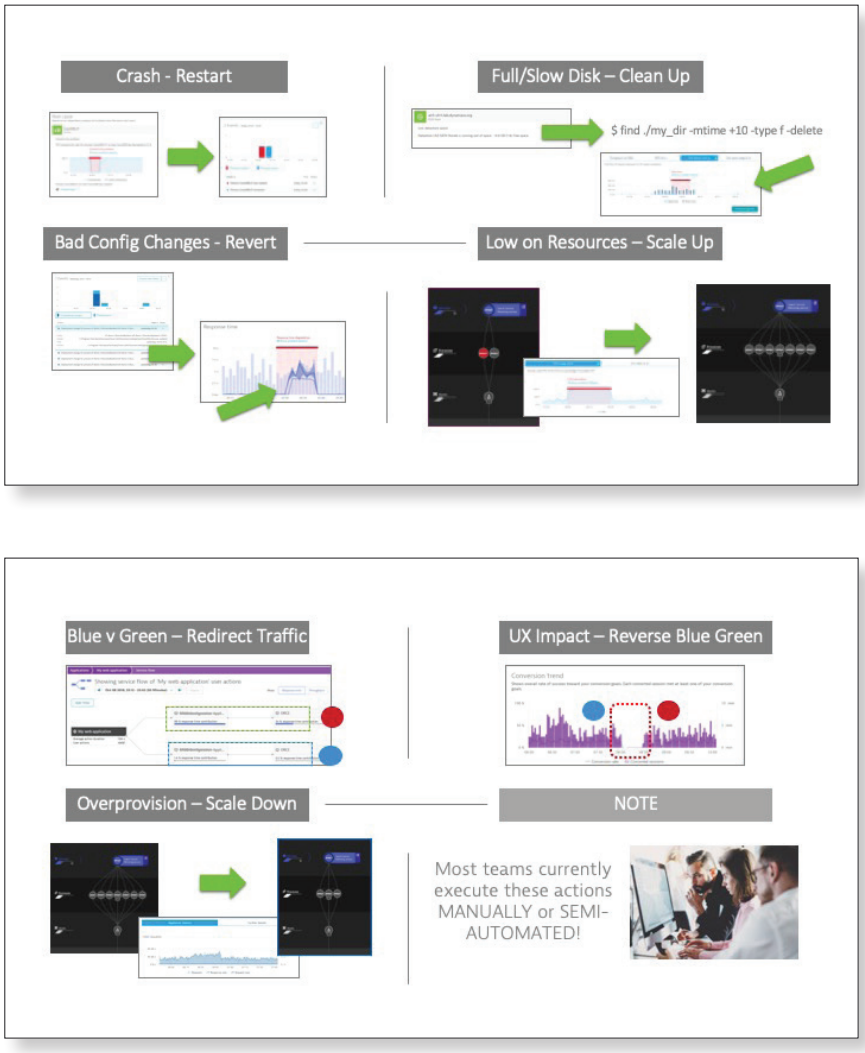
Remediation Use Cases

I have to be honest. I never worked in Operations, nor have I worked as an SRE. I've always worked in Performance Engineering where I ended up breaking a lot of environments for different reasons, e.g: hitting the system with too much load, not cleaning up log directories before a new run or improper configuration of my test or test environment. When talking to my friends, customers and partners that work in Operations, they tell me they see similar root causes in production environments.

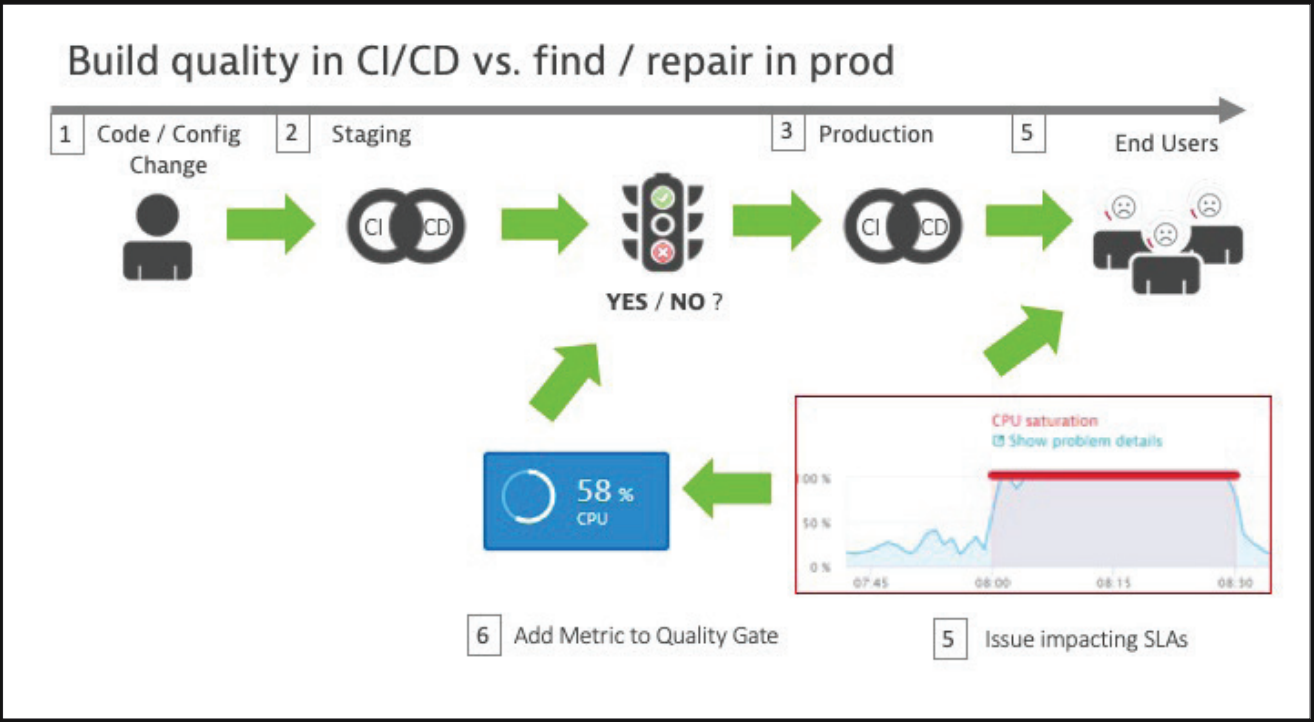
Let me therefore start with a quick overview of common problem scenarios that impact production environments and the remediation actions that can be taken to bring the system back to normal:

This is a very short list of remediation actions—but—a list that addresses very common and basic issues in today's environments: Process Restarts, Resource (e.g: Disk) Cleanup, Revert Bad Configuration Changes, Scale Up or Down, Switch Blue vs Green, for example.

While these use cases have been known for years by many teams I talk with, only a handful have invested in some sort of automation. While many have built scripts—these typically get executed manually when a problem is detected. That means—there is a lot of room for improvement!



Some of the remediation use cases that I have seen in recent months with our users.



Shift-Left SRE: Add CI/CD Quality Gate checks for every production remediation action. This allows us to prevent vs repair!

PREVENT in CI/CD vs REPAIR in Prod (Shift Left)

While we must invest in automating these remediation scripts in production we should learn from every problem we detect in production, figure out what the root cause is, how to replicate, measure or detect that root cause in a lower level environment and automate that into the CI/CD Delivery Pipeline as a Quality Gate:

This approach of adding these checks in a pre-production environment requires the teams (SRE Teams, Ops) that currently work on auto-remediation scripts for production to also think about how these situations can be detected in a lower level environment as part of the automated delivery pipeline. Here are the tasks that from my perspective define Shifting-Left SRE:

- Take the lessons learned from production
- Reproduce them in Pre-Production through e.g: Chaos Monkey Tests
- Automate the checks into CI/CD Quality Gates

In my presentation at re:Invent, I listed examples on how to detect certain use cases, what the metrics are to watch out for and how to capture these metrics:

Check 1
- Bad coding leading to higher costs?

- ❑ **Metrics**
- ❑ Memory usage
- ❑ Bytes sent / received
- ❑ Overall CPU
- ❑ CPU per transaction type

Check 2
- New dependencies? On Purpose?
- Services connecting accurately?
- Number of container instances needed?

- ❑ **Metrics**
- ❑ Number of incoming / outgoing dependencies
- ❑ Number of instances running on containers

Check 3
- Are we jeopardizing our SLAs?
- Does load balancing work?
- Difference between Canaries?

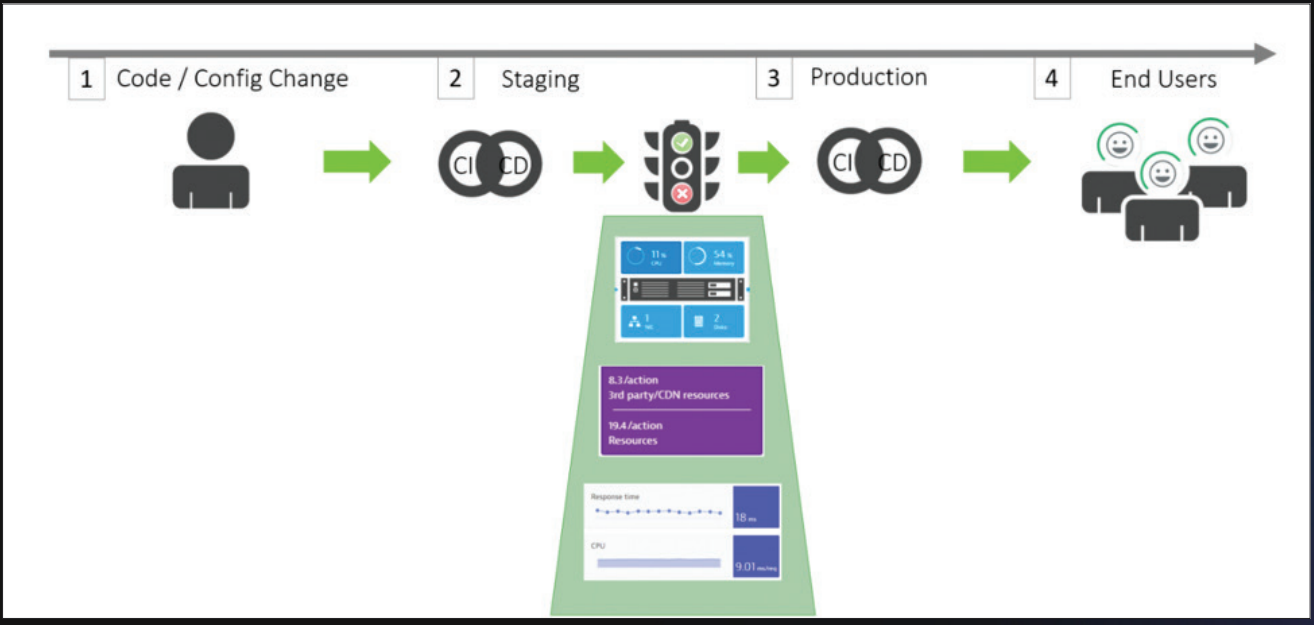
- ❑ **Metrics**
- ❑ Response Time (Percentiles)
- ❑ Throughput & Perf per Instance / Canary

Check 4
- Did we introduce new "hidden" exceptions?

- ❑ **Metrics**
- ❑ Total Exceptions
- ❑ Exceptions by Class & Service

List of Use Cases, Metrics and how to Query these metrics in your CI/CD Pipeline.

Once we have the metrics and the tests to replicate these scenarios in a lower level environment we can add it to the pipeline as quality gate checks:



Simple add these tests and metric checks into your pipeline. This will prevent issues from ever entering production!

If you are interested in integrating Dynatrace captured metrics into your Jenkins, Azure, Concourse, and Bamboo pipelines, check out my recent blog posts and YouTube tutorials: Jenkins Performance Signature, Azure DevOps for Dynatrace, AWS DevOps Tutorial!

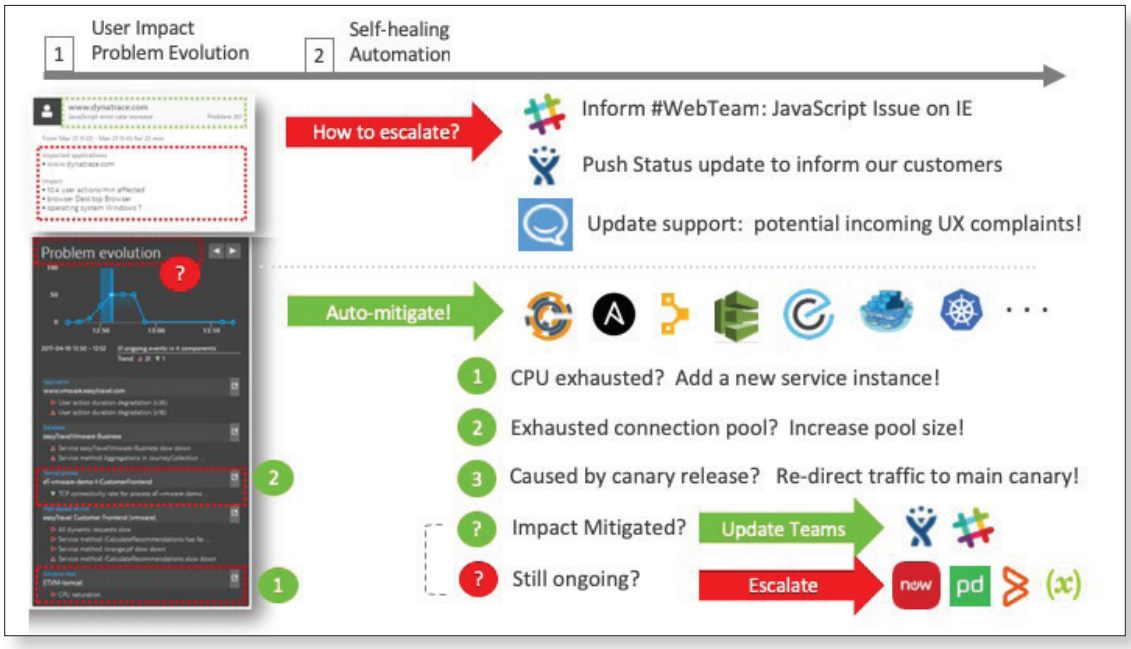
Auto-Remediation As Code

When using Dynatrace for full-stack monitoring of platforms we do not just receive alerts based on individual metrics that violated a static threshold or baseline, e.g: CPU is high or Disk is Full. The Smartscape Model, the PurePaths and the high-fidelity monitoring data we ingest and forward to our AI allows Dynatrace to detect problems and correlate all relevant events and data points to a single problem instance. All correlated information on that problem is made available to the auto-remediation as code scripts that SRE engineers wrote to handle specific problem root causes. The auto-remediation scripts do not have to do any correlation work but simply focus on fixing the root cause of a single problem vs chasing many loose ends of unrelated events you get from other monitoring or alerting tools.

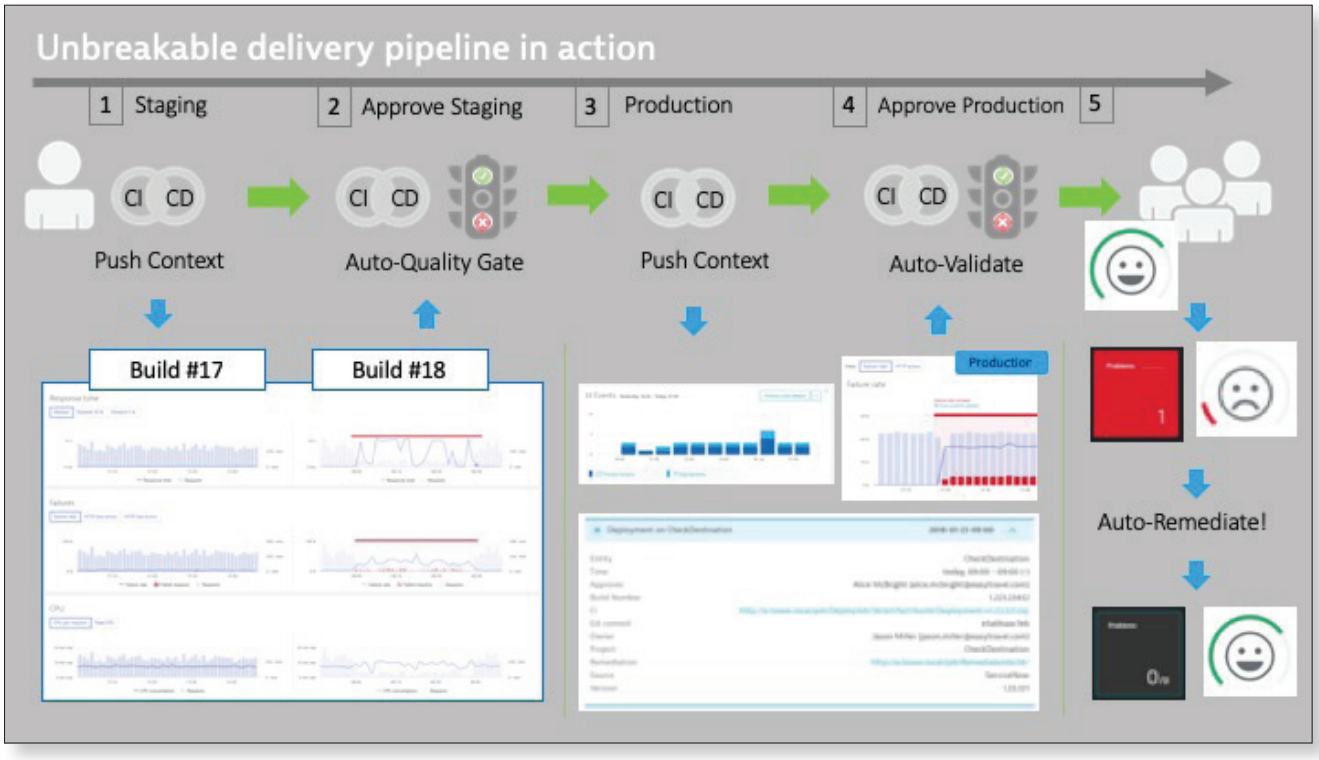
The following visualizes the full problem/incident response workflow. Once Dynatrace detects a problem we set the correct incident response actions in motion to inform the impacted teams as well as call auto-remediation as code scripts that address the root causes rather than just symptoms:

In my demo at re:Invent, I showed the self-healing part of my AWS DevOps Tutorial where I deploy a faulty version of a Service that results in a high failure rate impacting end users. During the deployment I pass the link to an AWS Lambda function that the developer (in this case me) wrote to be called in case of an issue in this service (Auto-Remediation As Code). Once Dynatrace AI detects the issue, it calls the Lambda function that is associated with the latest deployment of the faulty service!

I encourage Operations or SRE Teams to collaborate with the engineering teams so that they can start writing their own remediation code. This code should be battle tested in a pre-prod environment when those problems get simulated that have been seen in production before. This is a great way to build resilience into the code package that comes from engineering and as it has been battle tested in CI/CD the chances of a real problem in production are very small.



Self-Healing with Dynatrace AIOps and Auto-Remediation as Code enables the Path to Autonomous Ops



The Unbreakable Delivery Pipeline with Shift-Left SRE Quality Gates and Self-Healing in Production

The "Unbreakable Delivery Pipeline"

Now—if we put all the discussed concepts together in an automated delivery pipeline we can speak of what I coined "The Unbreakable Delivery Pipeline":

- We stop potential negative impacting changes early
- We auto-validate "Auto-Remediation as Code" via a Quality Gate
- We use "Auto-Remediation as Code" to "self-heal" production

We "Shift-Left" every problem we haven't yet auto-remediated.

I hope you got a couple of new ideas on how to Shift-Left SRE (Site Reliability Engineering). Working closely with engineers, providing them the frameworks to develop and extend the delivery pipeline to automatically validate and deploy your auto-remediation scripts allows you to build more and better resilient systems.

Learn more at dynatrace.com

Dynatrace provides software intelligence to simplify enterprise cloud complexity and accelerate digital transformation. With AI and complete automation, our all-in-one platform provides answers, not just data, about the performance of applications, the underlying infrastructure and the experience of all users. That's why many of the world's largest enterprises, including 72 of the Fortune 100, trust Dynatrace to modernize and automate enterprise cloud operations, release better software faster, and deliver unrivaled digital experiences.

4.01.19 6311_WP_cs