Traffic replay from the future:

Forecast latency, throughput and headroom before every deploy

Whitepaper

Nate Lee
Co-Founder & VP of Sales,
Speedscale

SPEEDSCALE

Introduction

While the adoption of cloud infrastructure and platforms continues growing at a rapid pace, delivering highly performant, stable applications in the cloud comes with its own set of challenges. Enterprises are adding new nodes and microservices with every sprint, and more API integrations and interdependencies to keep up with the features customers demand -- lest they abandon them for a nimbler competitor.

The world has seen massive changes in how products are delivered to market, from ecommerce, to SaaS-based applications, and ubiquitous mobility. An omnichannel go-to-market is now par for the course, and customers expect flawless application performance, from any device or location. World events like COVID19 have only driven more employees to work from home while forcing companies to lean even harder on their digital capabilities.

All of these pressures build up to an enormous stress test on the applications we depend on. Simply 'moving to cloud' isn't enough to meet scale and performance demands, when highly complex modern applications are put on a high-speed release train to production.

While distributed, modularized components of an application enable rapid iteration of well-defined functions, the communication and contracts between them have to be considered carefully. A breakdown in communication always has the potential to grind the entire system to a halt.

With entire economies based on digital presence, the stakes for performance and uptime have never been higher.

In this whitepaper, you will learn...

- Current techniques for achieving application quality at scale
- Why ensuring quality at scale is difficult
- Why progressive SRE techniques won't work for some applications and industries
- How Speedscale works
- Benefits and Use Cases for automated service replay and forecasts
- Run traffic replays for gated checks within CI/CD automation

Application Quality at Scale: Current Techniques

Infrastructure automation has grown in a big way, with server images, functions, and containers proliferating throughout IT. Managed workloads and pipelines are needed to streamline application deployment onto newer elastically scaling, ephemeral nodes in new cloud-native architectures.

Layer in a high rate of change between so many nodes, and DevOps teams encounter mind-boggling complexity in assuring functional integrity, secure operations and performance at scale.

The State of the Art in Testing Isn't Always Keeping Up

Software testing has come a long way since test-driven development and early UI-oriented testing tools were introduced. Agile teams now commonly automate unit tests, regression tests and performance tests as part of the software delivery pipeline. That's wonderful.

As the software lifecycle accelerates, complexity starts to break down the reliability of test automation itself, especially at the point of any change -- any next deployment, any spike in the volume of incoming requests, any variability in data returning from downstream services.

It's hard to get all of the resources aligned for a definitive test run for a node that is under constant change, much less all of the components and data needed to validate it. Test suites are vital to each release, but get fractured between releases, causing lots of rework while making test results harder to trust.

So how have we adapted to this shifting technological landscape?
By using the same technology we used to test monoliths,
client-server and SOA systems -- of course.

Unit testing, end-to-end UI tests and manual testing are the standard. Yet we know from the 2019 State of DevOps Report by DORA that "... the types of incidents that bring down production systems are often caused by interactions between components..." when unit tests are meant to catch problems with the logic of a unit of code in isolation.

End-to-end tests can catch defects of an entire application once a UI is completed. However many APIs, databases, and event queues the system under test interacts with don't have a UI available for such a test at all. Lastly, the vast majority of quality automation still requires an author to get things rolling. Someone has the arduous task of "File > Create New Test Case" for all hotpaths and edge cases. Let's not forget maintaining all these tests for regression, performance and exploratory testing uses.

SREs and Canary Deploys

To answer the need for assurance beyond what testing can provide, progressive development shops started releasing new application features to a much smaller set of users with a limited blast radius. Canary releases, feature flags and blue/green deployments are attempts at understanding application quality while affecting as few users as possible with a bug or outage.

Monitoring tools are essential to pick up after such releases as well, discovering small problems in deployment before they become big. Ideally, an impending production outage would trigger the monitor to send threshold alerts, turning these leading indicators over to an SRE (Site Reliability Engineer) for issue resolution before more canary deploys are rolled out to larger and larger audiences.

Once an app is 100% deployed, SRE's and engineers can breathe a sigh of relief -- but unfortunately, continuously delivered software is never really 100% deployed.

While the Googles and Netflixes of the world embrace SRE principles as the way to rapidly deploy at scale, not all companies have hordes of uber-skilled SRE's on call to triage and break-fix emerging problems in production. Many industries can't afford to employ this strategy at all. Fintech, Healthcare, and Government are examples of highly regulated verticals with sensitive data and systems that require a minimum of disruption, and they are known for a low tolerance of failure.

> The current state of the art in testing combined with the number of services, connection points, and rate of change in modern application delivery environments makes it a near-certainty that even the most detail-oriented organization will miss critical flaws.

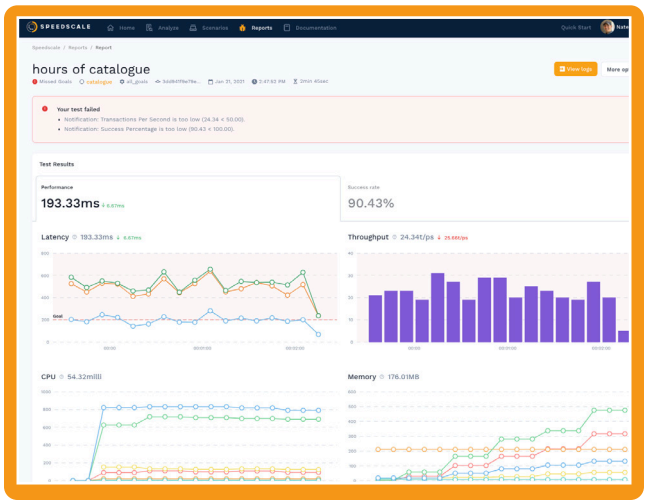
Solution: How Speedscale works

Uncovering and fixing defects in your code well before customers can find them is the ultimate goal. While this sounds obvious, many organizations have fallen into the trap of thinking bugs are OK, so long as we can react to them quickly. Somehow, war rooms, being "on-call" and firefighting issues on the weekends has become the norm -- and so has poor morale among Dev and Ops teams subjected to it. Due to rapid release expectations and application complexity, this problem is not going away anytime soon.

Observe, Analyze, Replay

Replaying traffic unlocks a variety of best practices -- chiefly, validating application quality early and often. With the rise of 2-pizza engineering teams where developers are responsible for their code from start-to-finish, any work that doesn't contribute toward delivering valuable functionality for customers -- or toil -- is the enemy.

By replaying previously observed traffic against a changing system, engineers are able to understand how their changes will likely perform in production. They can receive instant metrics around the SRE's 4 Golden Signals (latency, throughput, saturation and error rate) all before deploying a single line of code to production.



Traffic by its nature contains three ingredients that are critical to validating proper function of new code. These ingredients are: 1) the transaction; 2) its dependencies; and 3) the data itself.

Organizations tend to focus on quality automation of test transactions (1) and soon realize they lack ready environments/dependencies (2) to run the automation regularly. If they have laid the virtual groundwork to reliably automate the provisioning of (1) and (2), then a lack of valid test data (3) becomes the bottleneck.

Speedscale uses traffic to understand how your application is invoked, what dependencies it needs (so it can mock them), with all the proper scenario data in place.

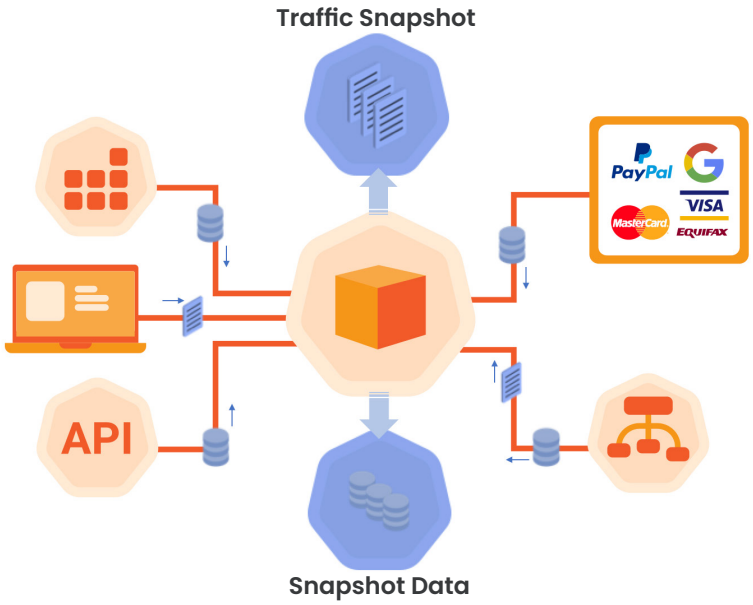


Figure 1. Record traffic and dependencies for replay

Listen and Isolate your Service: The 'Sandwich'

Speedscale's footprint is similar to a monitoring tool. Lightweight speedscale agents in your container or Kubernetes node listen to traffic and use an analyzer to process inbound and outbound data. If agents are not possible in an environment, API gateways, log files, traffic captures (PCAPs) and service meshes offer other possible forms of data ingest.

Once connected, Speedscale parses the data and provides replayable Docker containers that can be provisioned as a traffic generator. If utilizing our agent, Speedscale will automatically recognize and model backend dependencies needed for replay.

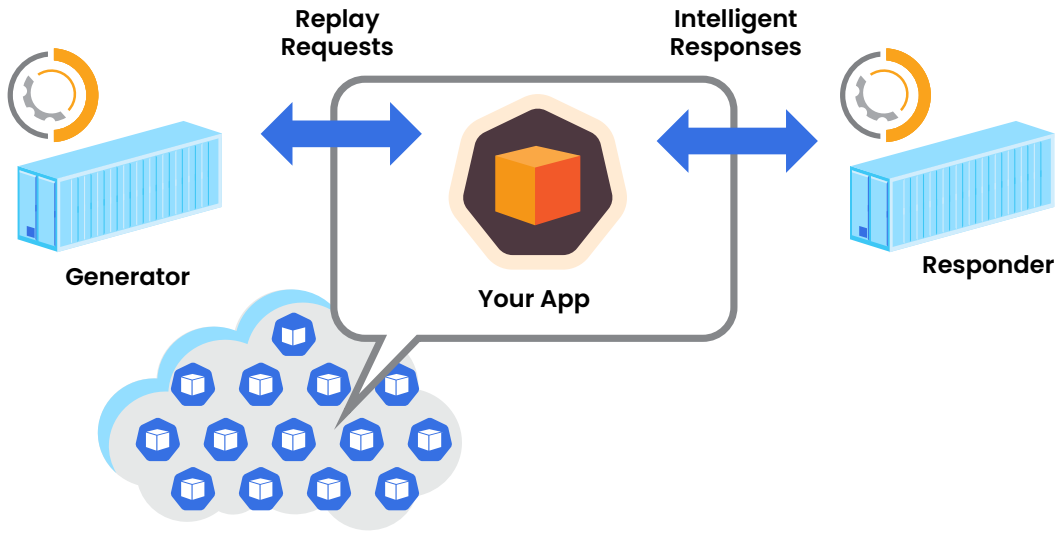


Figure 2. Isolate and performance test any part of your app

By mocking necessary dependencies, and invoking traffic replays, Speedscale is able to 'sandwich' your app with testability, providing isolation for validating the performance of your microservice monolith. Exercising new code with controls on either side (upstream and downstream) eliminates a lot of the uncertainty and noise typically found in staging environments.

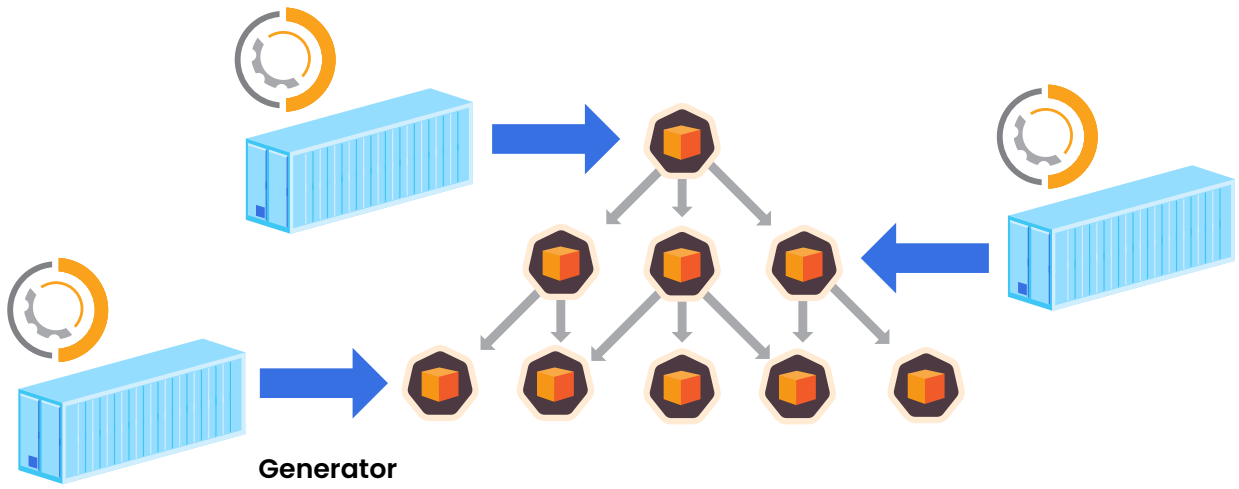


Figure 3. Exercise your app at will with Speedscale with realistic traffic and valid data

Benefits

Replaying traffic is not a brand new concept in general and there are a few open source tools that can capture transactions and data at certain layers, as well as several commercial vendor solutions that combine continuous testing and service virtualization to eliminate constraints.

However, most of these approaches are still too brittle, requiring scripting to set up and maintain test suites and data, which breaks down at the breakneck pace of change required for continuous delivery into cloud environments.

Speedscale was built for the modern age of containerization and microservices, meaning we can capture and replay inbound and outbound traffic at the API layer, or using Kubernetes as a self-contained reference architecture for deployments onto any Hybrid IT infrastructure.

There are several merits to replaying traffic in this way:

- Generating replayable traffic takes only a few minutes with dramatically higher code coverage
- Traffic contains the transaction, dependencies and data, which can be parameterized for different scenarios
- Traffic can check integration, functional integrity, regressions or be multiplied for performance and stability tests
- Traffic can forecast relevant quality metrics before deployment, in line with automated software delivery pipelines
- Repeated replays can allow for experiments, tuning, comparisons and parity checks (eg. different clouds, processors, libraries, vendor software, etc).
- Reduced pre-production lab infrastructure and cloud consumption costs, as Speedscale environments are spun up only when called on by automated pipelines and discarded immediately after each scenario is run.

Pre-production performance environments should actually contribute more value to your organization and its software customers over time, not create an annuity teams have to constantly upkeep.

FAQ

What kind of traffic can you replay?
Currently we can replay HTTP, gRPC, S3, MINIO, and MongoDB with new protocols being added all the time.

How could your mocks possibly simulate our complicated backends?
Speedscale does not try to build a mock of your entire service. We only need to supply the responses your service requires during a set span of time. Remember, we also know what transactions are being sent into your service during that timespan as well.

What if you record sensitive data?
We can integrate with a DLP solution to scan, identify and mask sensitive data as we analyze the traffic for replay. We also have custom rules you can establish for additionally shielding proprietary or sensitive data from observation.

How manual/automated is generating traffic?
Once you select a timespan of traffic according to the replay fidelity you need, depending on whether it is minutes/hours, the traffic replay generation takes seconds or minutes.

Sample use cases for Speedscale

Run in Generator-only mode to predict how your app behaves under peak load	*Run Responder-only mode to simulate unavailable backends (e.g., 3rd parties)*	*Run Speedscale isolation for traffic replay, fix & tune your APIs, then replay*
Compare 1 environment's traffic (e.g., EC2) vs. your new K8s cluster	*Run traffic analysis for high level metrics (with no replay)*	*Run traffic replays for gated checks within CI/CD automation*

About Speedscale

Continuous Resiliency from Speedscale gives you the power of a virtual 'SRE-bot' working inside your automated software release pipeline. Forecast the real-world conditions of every build, and know you'll hit your SLO before you go to production.

Feed Speedscale traffic (or let us listen to your app) and we'll turn it into traffic snapshots and corresponding mock containers. Insert your own service container in between for a robust sanity check every time you commit.

Understand latency, throughput, headroom, and errors--before you release! The best part? You don't have to write any scripts or talk to anyone!

To learn more and request a demo, visit www.speedscale.com.

SPEEDSCALE