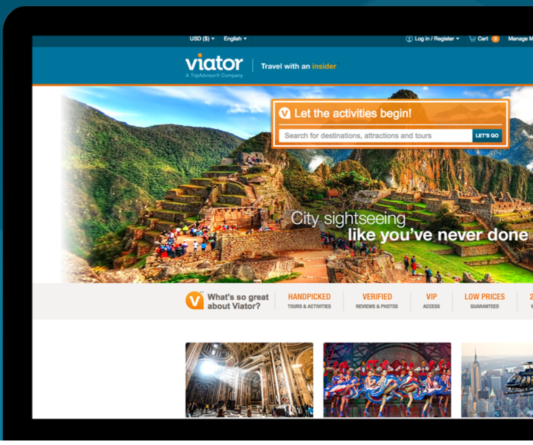
Customer story / e-commerce

How OverOps Allows TripAdvisor to Deliver Innovation in a PCI-Compliant Environment

OverOps helps Viator quickly detect and fix every new error in production, before it impacts the customers



tripadvisor | viator

TripAdvisor is the world's largest online resource and category leader for travel destination activities, reaching over 7 million global travelers per month who are seeking to find, research, and book activities and tours prior to traveling and during their trip.

7M	120M	60	100
Visitors/Mth	API requests/Mth	Developers	AWS Instances

Production Monitoring Ecosystem:

OverOps New Relic catchpoint
SCALYR pagerduty

Highlights

- OverOps helps Viator replicate issues locally in their PCI Level One compliant environment.
- Using OverOps, Viator is able to have a test-driven development approach for legacy code.
- OverOps allows Viator to cut down time spent debugging by 90%.
- OverOps alerts on every error in production, allowing the team to fix issues quickly and easily.

Key challenges and pain points

Working in a PCI Level One compliant environment means that it's nearly impossible to replicate issues locally, and it also limits production access.
On top of that, there's legacy code, when the original developer is long gone and things are being logged just-in-case, that leads to very noisy (and costly) logs. This made identifying, finding and understanding the errors and exceptions a real challenge for us.

Prior to OverOps, once we detected an issue in production, we had to:

1. Roll back the release.
2. Search for the error in the logs and code, where nothing was obvious.
3. Create a new hotfix release with extra logging.
4. release the new version.
5. Wait for replication, which could take days.
6. Get the new verbose logs and roll back the release.
7. Fix the issue.
8. Finally, release the new version.

That process could take days. OverOps reduced that time to minutes, and we have everything we need to reproduce issues straight off the bat.

Short of attaching a debugger in production, OverOps is the next best thing."

Example problem that OverOps helped resolve:

We saw a performance degradation following a release. We tried looking through our APM tools, but couldn't find any obvious elements that might have caused it within the code. Then, we received an email from OverOps that showed many "gets" from a cache failing, so objects were being re-generated.
OverOps then showed us all the variable values across the callstack for that error in production, and enabled us to fix this issue quickly and easily.
Another case was after releasing a major version -
We saw a sudden spike of 500 errors on one of our APIs, and our logs were filling up with exceptions. We had to roll back the release, search the logs and code, create a hotfix with extra logging, release the new version and wait for replication. That took 3 days.

Now when we release a version, OverOps alerts us about errors in real time, shows us the variables and lets us easily reproduce and solve the issue. OverOps turned days of work into minutes. Short of attaching a debugger in production, OverOps is the next best thing.

"OverOps turned days of work into minutes"

How are you integrating OverOps with your daily workflow?

OverOps's email digests alert us on new errors that were detected in our application during the last 24 hours. These emails also notify about specific errors that exceed target volumes that we've set up in advance.

With OverOps, we've cut down our time spent debugging and troubleshooting by 90%. This enables us to speed up development time and deliver a reliable product, while still offering an excellent user experience.

Full code and variable state to immediately reproduce any error.

No need to manually reproduce issues by searching for information in logs. Reduce MTTI by 90%+

<1% overhead in production

OverOps operates between the JVM and processor level enabling it to run in staging and production.

Proactive detection of all new and Critical errors

New issues are detected and routed to the right developer vs. discovered by users. Each error receives a unique code fingerprint unique it across the app.

No change to code or build

New issues are detected and routed to the right developer vs. discovered by users. Each error receives a unique code fingerprint unique it across the app.

Supported Platforms:

JDK 1.6 and above | HotSpot, OpenJDK, IBM JVM | Java, Scala, Clojure, Groovy | Linux, OS X, Windows | Docker, Chef, Puppet, Ansible | Coming soon: .NET

Integrations:

SLF4J, Log4j, Logback, Apache Commons Logging, Java Logger | Splunk, ELK, SumoLogic, and any other log management tool | AppDynamics, New Relic, Dynatrace | Workflow automation: Slack, HipChat, JIRA, Pagerduty | Webhooks | StatsD

Learn how OverOps can help you automate your deployments -
Schedule a demo with an OverOps monitoring engineer