BEST PRACTICE SERIES

Frontend Testing

DATADOG

BEST PRACTICE SERIES

Frontend Testing

Best practices for creating end-to-end tests


Margot Lepizzera


Mallory Mooney

Browser (or UI) tests are a key part of end-to-end (E2E) testing. They are critical for monitoring key application workflows—such as creating a new account or adding items to a cart—and ensuring that customers using your application don't run into broken functionalities. But browser tests can be difficult to create and maintain. They take time to implement, and configurations for executing tests become more complex as your infrastructure grows.

As you develop your application, it's important to create a plan for building test suites that provide appropriate coverage and are easy to adopt and maintain in the long term. In this chapter, we'll walk through some best practices for creating tests, including:

- defining your test coverage
- building efficient, meaningful tests
- designing coherent and easy-to-navigate test suites
- creating notifications that enable teams to respond to issues faster

Along the way, we'll show how Datadog can help you follow these best practices and enable you to easily create and organize your tests.

Define your test coverage

Before you start creating tests, it's important to consider which application workflows you should test. Teams often start with goals to create tests for every feature to reach 100 percent test coverage.

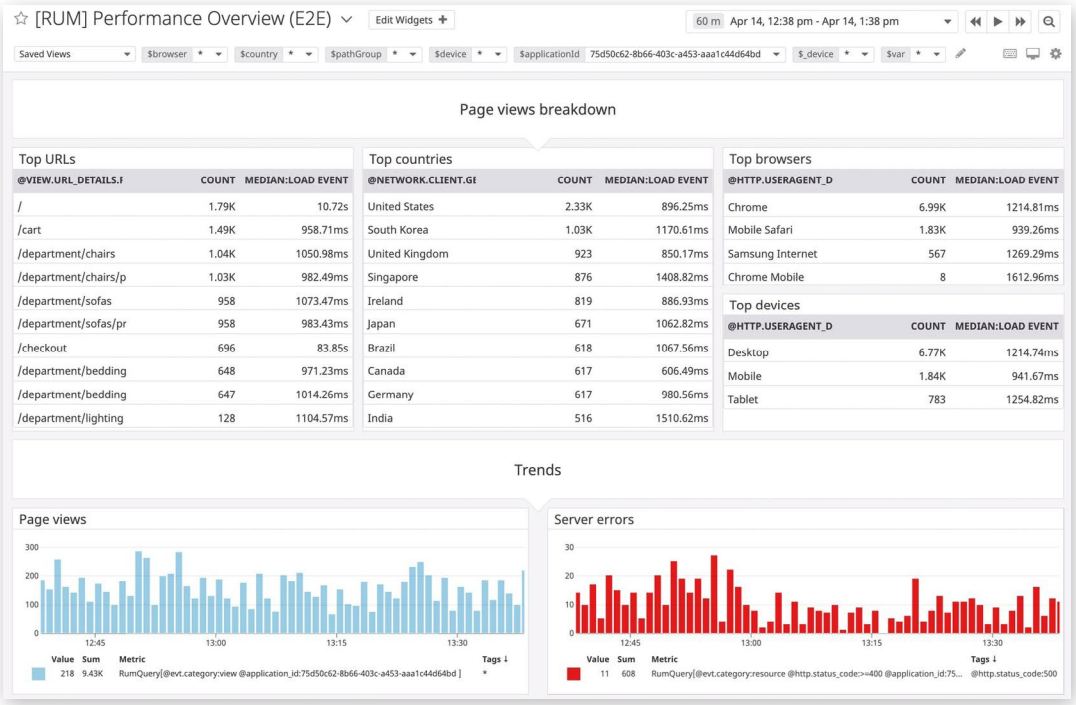
This creates suites that are often too large to maintain efficiently; not every workflow is a good fit for E2E test suites.

Test coverage should be representative of your users and how they interact with your application. This requires **identifying the commonly used application workflows** versus the less popular ones, so you can scope test suites to include only the workflows that bring the most value to your business. This approach will help you strike a better balance between test coverage and maintainability.

Depending on the industry, some of these key application workflows could include:

- creating or logging into an account
- booking a flight
- adding items to a cart and checking out
- viewing a list of employees for your company
- adding direct deposit to an account in an internal payroll system

To capture data about the common workflows for your application, you can use tools such as real user monitoring (RUM). For example, you can use Datadog's RUM product to perform analytics on your application and determine where you should focus testing. This could include the top visited URLs for your application as well as which browsers and devices visitors are using. This gives you a starting point for which user journeys you should include in your E2E test suite.



In addition to testing the more popular workflows, it's important to **create tests for flows that are not used as often** but are still critical to your users' experience. This may include recovering an account password or editing a profile. Since these flows are not used as often, you may not be aware of broken functionality until a customer reports it. Creating tests for these flows enables you to identify broken functionality before a customer does.

Once you know which application workflows need coverage, you can start building efficient test suites. We'll show you how in the next section.

Build meaningful tests for key application workflows

How you approach creating new end-to-end tests will determine if they can adequately test your critical application workflows. This process requires incorporating the right DOM elements (e.g., an input field or button) and commands for executing the necessary workflow steps, as well as assertions that confirm an application's expected behavior. Without a clear plan outlining how to approach a specific workflow, tests can easily become complex, with a large number of unnecessary steps, dependencies, and assertions. This increases test flakiness and execution times, making it more difficult to troubleshoot and respond to issues in a timely manner.

There are a few best practices you can follow that will ensure test suites are efficient, easy to understand for new team members, and will reduce your team's average mean time to repair (MTTR):

- break workflows down with smaller tests
- create meaningful assertions to verify expected behavior
- built tests that adapt to factors out of your control
- create idempotent tests to maintain the state of your applications

We'll walk through each of these best practices in more detail next.

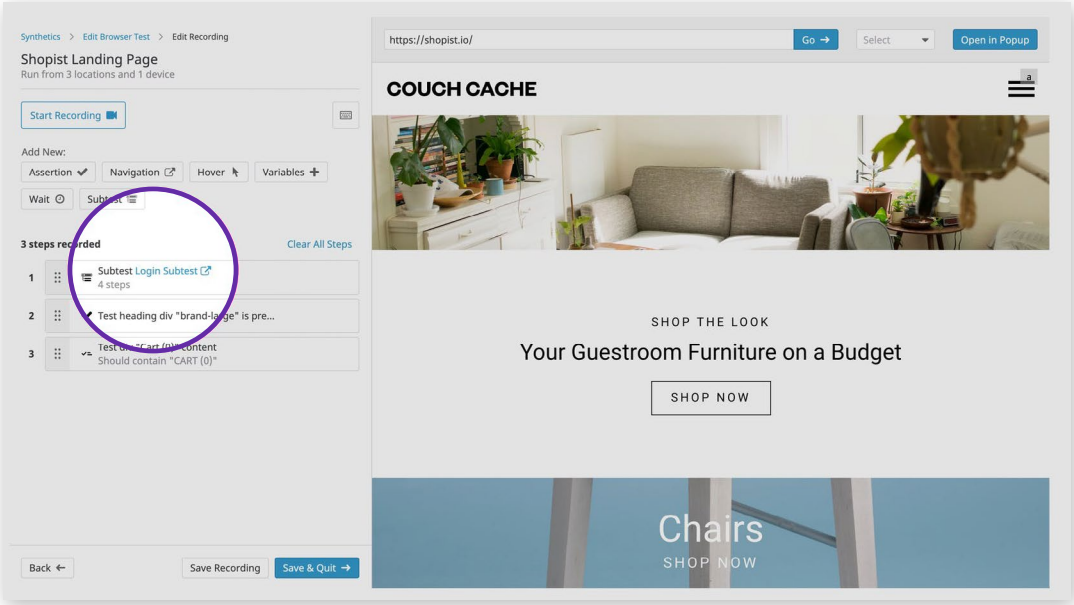
BREAK WORKFLOWS DOWN WITH FOCUSED TESTS

It's important to create tests that do not duplicate steps and stay within the scope of the application workflow that is being tested. **Separating workflows into smaller, focused tests** helps you stay within scope, reduce points of maintenance, and troubleshoot issues faster. For example, a test that is meant to verify functionality for a checkout workflow should not include the steps for creating a new account. If that test fails, you do not want to spend time troubleshooting whether the failure occurred at checkout or at account creation.

Ensuring that every single step of a test case closely follows the steps of the tested workflow, minimizing the number of unrelated steps, and adding the right assertions to verify behavior are key to creating focused tests.

Even simple tests, with a few meaningful steps and assertions, can provide a lot of value for testing your applications.

You can incorporate the DRY principle (i.e., “don’t repeat yourself”) while creating new tests to stay within scope and minimize step duplication. One way you can do this in Datadog is by creating reusable components with subtests. Subtests group reusable steps together so that you can use them across multiple tests. For example, you can create a “login” subtest that includes the steps required to log into an application and use it in your other tests.



The landing page test above uses a login subtest to first log into the site automatically

You could also create subtests to cover other key workflows, such as adding multiple items to a cart, or removing an item from the cart before proceeding to checkout. An added benefit to creating more reusable components is you can incorporate multiple subtests into a single test. With this, you only have to update one subtest instead of multiple tests when a workflow changes. This also drastically cuts down on the time it takes to create new steps for tests, enabling you to focus on capturing the steps for new or updated functionality.

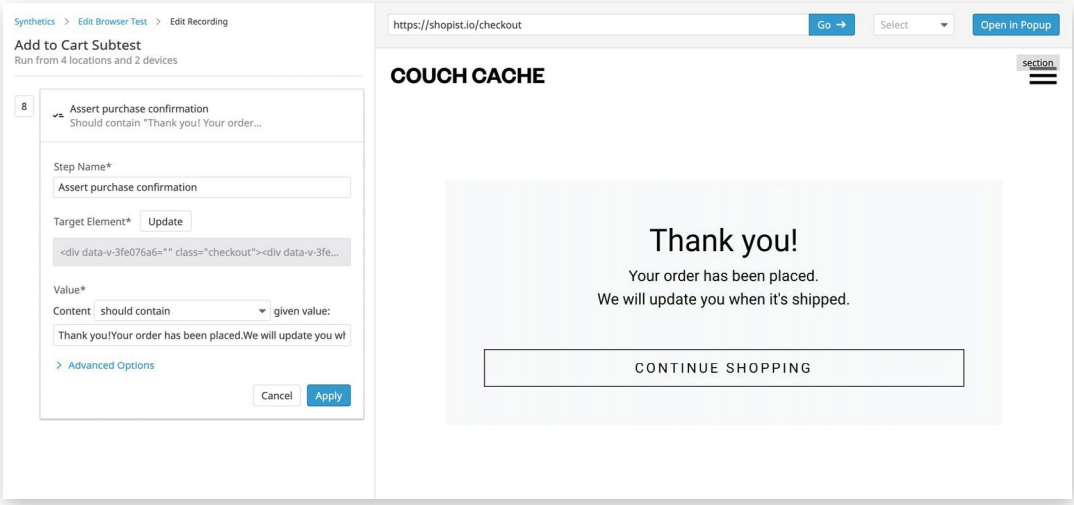
CREATE MEANINGFUL ASSERTIONS TO VERIFY EXPECTED BEHAVIOR

Another aspect of building efficient test suites is ensuring tests can **assert (or verify) important application behavior**. Test assertions are expressions (or steps) that describe workflow logic. They add value to tests by mimicking what users expect to see when they interact with your applications. For example, an assertion can confirm that a user is redirected to the homepage and sees a welcome message after they log into an application.

There are several assertions that are commonly used in tests:

- an element has (or does not have) certain content
- an element or text is present (or not present) on a page
- a URL contains a certain string, number, or regular expression
- a file was downloaded
- an email was sent and includes certain text

It's important to only include assertions that mimic what the user would do or would expect to happen for the tested workflow. A test for checkout functionality, for instance, only needs to include assertions that are relevant to checking out, such as verifying a purchase confirmation.



In the simple example above, we're using one of Datadog's out-of-the-box assertions to verify that users will see a confirmation after purchasing an item from an ecommerce site. With this assertion, the test looks for a specific element on the page and verifies that it contains the expected text. For this example, if Datadog is not able to find the element, or if the text isn't present, the test will fail.

BUILD TESTS THAT AUTOMATICALLY ADAPT TO FACTORS OUT OF YOUR CONTROL

Tests often timeout and fail if an application takes too long to load. Load times can be affected by factors such as an unexpected network blip or an unusual surge in traffic that overloads application servers. These inconsistencies can cause tests to generate false positives, decreasing their ability to reliably notify teams of legitimate issues. And hard coding wait steps for tests, a common method for troubleshooting these types of failures, can make them more unreliable.

A best practice for ensuring tests can adapt to load time fluctuations is to **design them to automatically wait** for a page to be ready to interact with before executing (or retrying) a test step. By doing this, tests will execute or retry a step only when needed (e.g., when the application has loaded). This not only eliminates the need for manually adding wait steps but also allows you to focus on legitimate production issues.

For example, Datadog Synthetic browser tests are designed to regularly check if a page is ready to interact with before trying to complete the step (e.g., locate an element, click on a button, etc.). By default, this verification process continues for 60 seconds, though you can adjust this value. If Datadog can't complete the step within this period, it marks it as failed and triggers an alert.

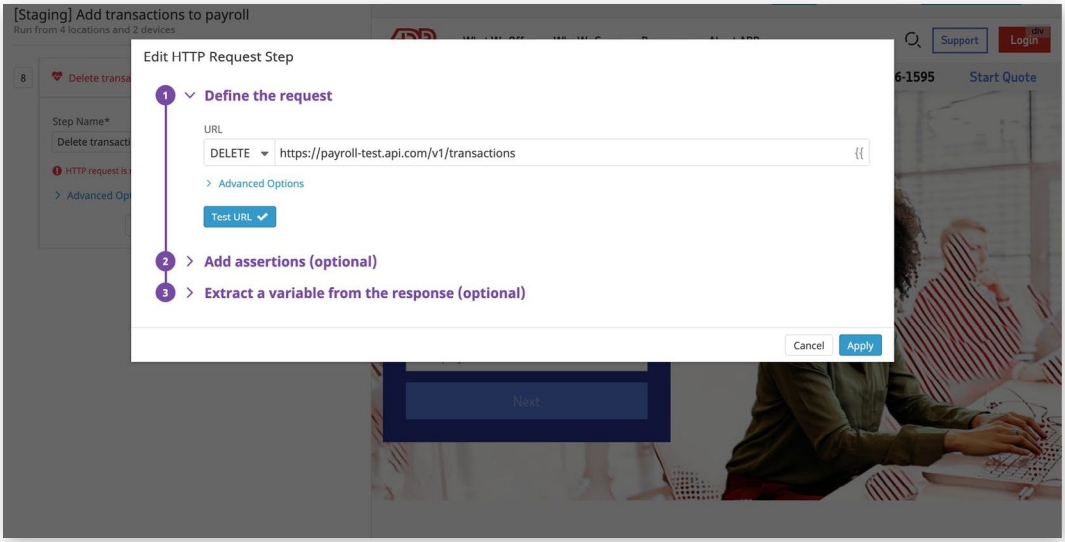
Adding these types of steps to your tests helps minimize failures that are often caused by network blips, but there are other factors that contribute to tests generating false positives. Next, we'll briefly look at how data can affect a test's ability to accurately verify application behavior.

MAINTAIN THE STATE OF YOUR APPLICATIONS WITH IDEMPOTENT TESTS

Tests rely on clean data in order to recreate the same steps with each and every test execution, so it's important to build dedicated configurations (e.g., login credentials, environments, cookies) that are only used by your test suites.

Part of the process for building these configurations is **creating tests that are idempotent**, which means that tests leave the state of your application and test environments the same before and after the test run. A test that focuses on adding multiple items to a cart, for example, should have a mechanism to delete those items from the cart afterward. This cuts down on the number of test orders pending in your system and ensures your application is always available for customers.

Datadog's built-in HTTP request steps can help you maintain idempotency and minimize data dependencies between tests. For example, if you have a test that creates new test transactions for an internal payroll system, you can use the HTTP request step to automatically delete those transactions from the database at the end of a test.



You can also use subtests to clean up data after a test run. These methods provide a quick and easy way to limit the number of times tests fail simply due to environment or data issues.

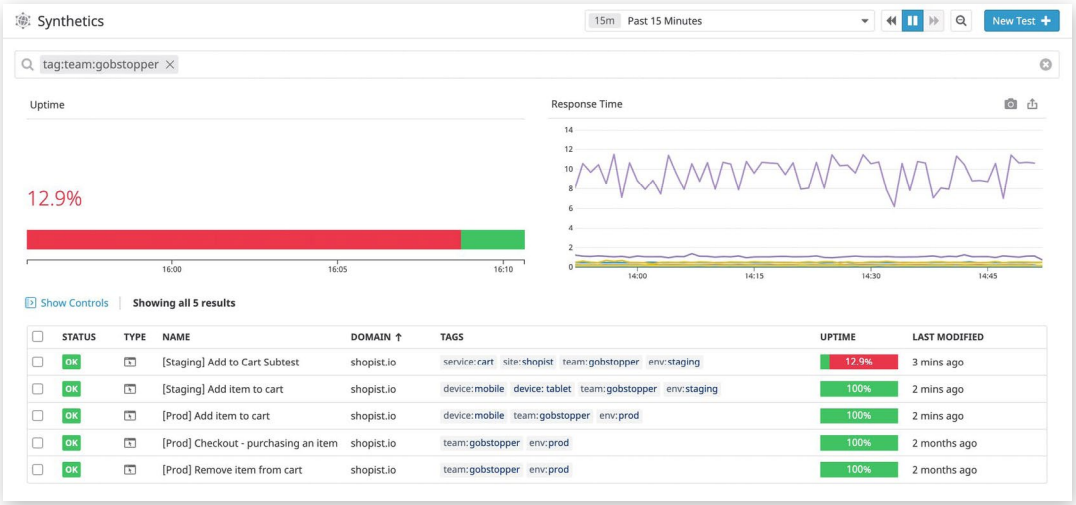
In the last few sections, we looked at best practices for creating focused and valuable test cases. Next, we'll look at test suites as a whole and how you can organize them by testing environments and other key metadata.

Design coherent test suites

- define test configurations based on testing environments
- use metadata to organize and quickly identify your tests

CONFIGURE TESTS BY ENVIRONMENT

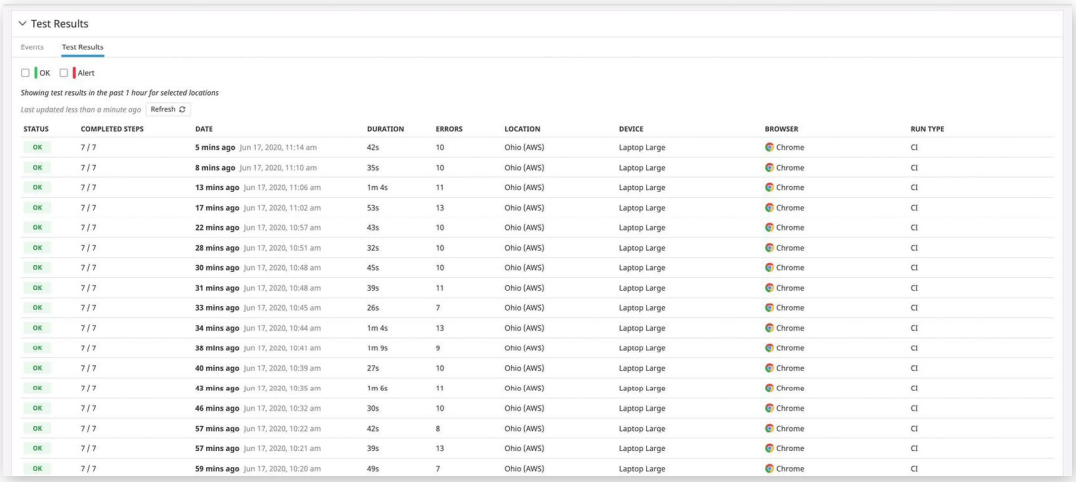
Each environment you use to test functionality during the development process may require different configurations for running tests. For example, tests in a development environment may only need to run on one device, while tests in staging require several different types of devices. You can **configure tests separately by environment** to ensure that tests only run on what is needed for a single environment.



In the example list above, you can see tests for staging and production environments. The "[Staging] Add item to cart" test is configured to automatically run on more devices than the production tests. Note that each test includes the associated environment in the test title and as an environment tag. This helps you quickly search for and identify where tests run.

Adding your end-to-end tests to existing continuous integration (CI) workflows can further streamline the process for testing your code before it makes it to your staging or production environments. Datadog's CI integration gives you more control over test configurations per environment, and enables you to easily override settings such as the starting URL, locations, and retry rules, and run a suite of tests as part of your CI scripts. For example, you can override the start URL to automatically use the environment variables configured for your CI environment. You can easily view the results of your tests directly in your CI tool (e.g., Jenkins, CircleCI, Travis) or in Datadog.

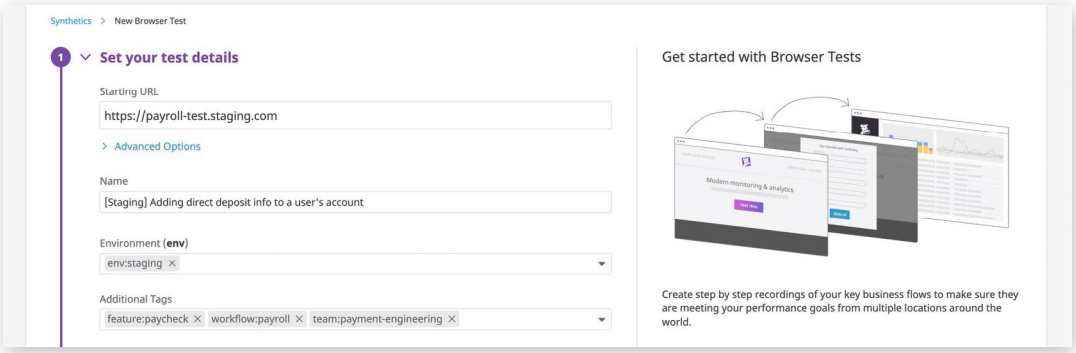
You can learn more about CI/CD testing with Datadog in chapter four of this guide.



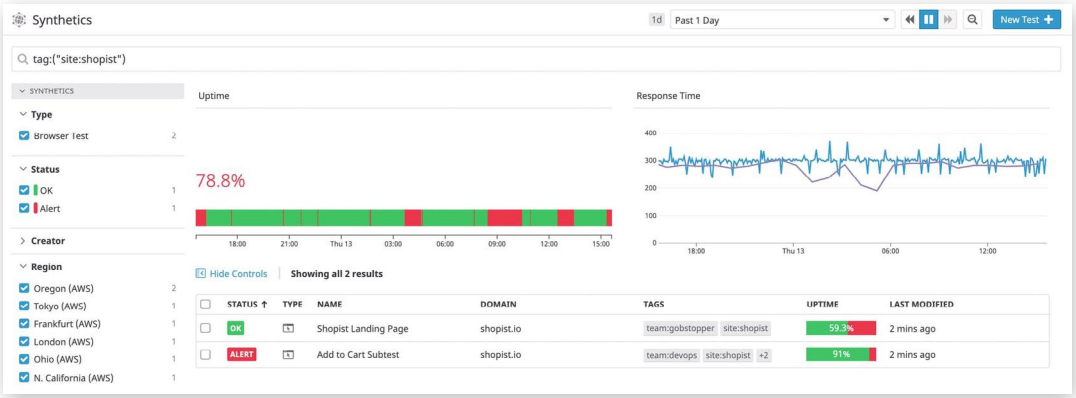
ORGANIZE TESTS BY LOCATION, DEVICE, AND MORE

Finding ways to organize and categorize tests is another key aspect of designing a coherent test suite. By **developing a good organizational structure**, you can easily manage your test suites as they grow, search for specific tests, and get a better picture of where they run. One way you can add structure to test suites is by incorporating metadata or tags such as the test type (e.g., browser, API), owner, and device. Tags like these provide more context for what is being tested, so you can quickly identify which application workflows are broken and reach out to the appropriate teams when a test fails.

Just as adding the environment in the test's title helps you quickly identify where a test runs, tags also enable you to categorize tests with key attributes as soon as you create them.



The example test configuration above uses tags to identify the environment, the feature and workflow to be tested, as well as the team developing the feature. Adding metadata or tags to your tests is especially useful if you manage a large suite of browser tests and only need to review a subset of them. For example, you can use the `team` tag to search for and run tests owned by a team that developed a new feature or a `site` tag to find all of the tests associated with a specific website.



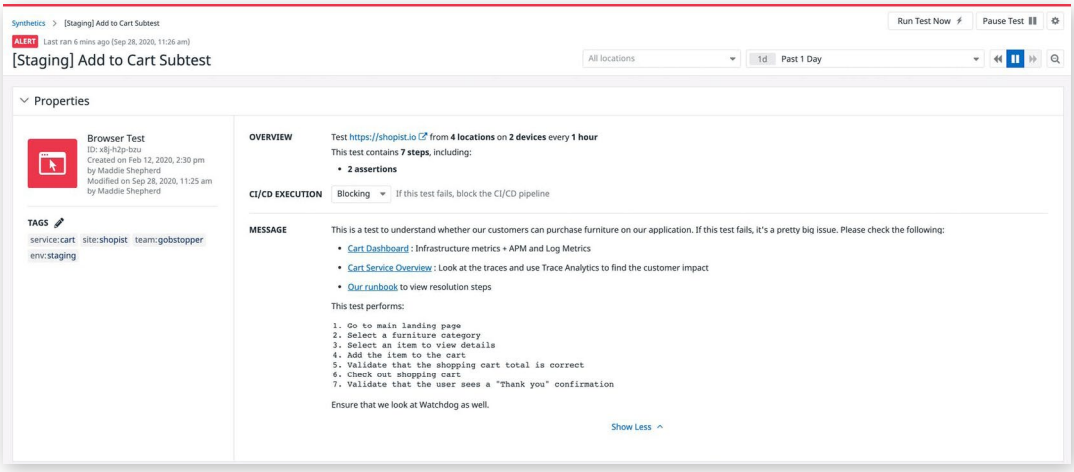
We've walked through a few ways you can streamline and build efficient test suites at a high level. Another important aspect of creating tests involves sending notifications so you can be alerted on their status in a timely manner.

Create notifications that enable teams to respond faster

Regardless of how you structure your tests or where you run them, it's important for the right team to be able to quickly assess test failures as well as know when test suites pass. For example, if you are testing a fix for a critical service in production, then engineers will need to know the state Regardless of how you structure your tests or where you run them, it's important for the right team to be able to quickly assess test failures as well as know when test suites pass. For example, if you are testing a fix for a critical service in production, then engineers will need to know the state of your tests as soon as possible. If all your tests pass, then the team can safely move forward with the deployment. If there are failures, then they need to be notified as soon as possible to quickly troubleshoot.

Test reports are often difficult to analyze, especially as you add more tests to a suite. And if you run tests across multiple devices, you may have to look at several different reports to find the failures for each device. This means you spend more time sifting through test results than finding and fixing the issue.

You can streamline the process for reviewing test results by **building notifications that automatically consolidate** and forward results to the appropriate teams or channels, providing them with the necessary context for what was tested. For example, you can create notifications that include more contextual information about your tests (e.g., steps to take to investi-gate the issue, troubleshooting dashboard links) and send them over email or to a dedicated Slack channel. This ensures that you are notified as soon as tests complete, enabling you to quickly evaluate if you need to take further action.



The example above is a Datadog notification for a test that verifies cart functionality for an e-commerce application. The notification provides detailed information about what is being tested and links to a relevant dashboard, service overview, and runbook for troubleshooting test failures.

Continue reading to learn how to maintain your existing test suite, or check out our documentation for more information on creating E2E tests.

Best practices for maintaining end-to-end tests


Margot Lepizzera


Mallory Mooney

In the previous chapter, we looked at some best practices for getting started with creating effective test suites for critical application workflows. In this chapter, we'll walk through best practices for making test suites easier to maintain over time, including:

- avoiding false positives in tests
- making tests easy to understand
- leveraging a unique test suite across environments

We'll also show how Datadog can help you easily adhere to these best practices to keep test suites maintainable while ensuring a smooth troubleshooting experience for your team.

WHY CAN TESTS BE HARD TO MAINTAIN?

Before we walk through these best practices for maintaining tests over time, it's important to understand why teams often spend more time updating existing tests than creating new ones. As an application's UI evolves, expanding test coverage ensures that new features are validated. But creating new end-to-end tests is only a small part of making certain application workflows or features do not break. For most engineering and QA teams, the bulk of their testing time is spent maintaining existing tests—rather than creating new ones for new functionalities.

There are several different factors that can contribute to the difficulty of maintaining tests including:

- tests that generate false positives during critical releases
- suites that are disconnected from continuous integration workflows, disrupting both application development and other tests when a breaking change is deployed
- inefficient test steps that make failures more difficult to troubleshoot
- tests that do not accurately describe what is being validated, making onboarding more difficult

Next, we'll look at each of these factors—as well as best practices for alleviating these pain points—in more detail.

Reduce test flakiness with reliable locators and alerts

Test flakiness—when tests randomly fail without any code changes—is one of the more common pain points for teams trying to maintain their suites. Tests can quickly become flaky if they do not use the right combination of UI locators to interact with application elements, and if they do not efficiently notify teams of legitimate failures. Interacting with the right UI elements and improving alerts is key to ensuring tests remain easy to maintain and that every single alert your team receives actually matters.

USE MULTIPLE ELEMENT LOCATORS TO KEEP UP WITH A FAST-CHANGING UI

Test reliability is determined by the locators you use as the basis for test actions and verifications. For example, element IDs are often unique enough and more resilient to development changes, making them a primary choice for tests. If these IDs are not available, or not unique enough, then engineers may try to update frontend code to accommodate test automation. However, this requires substantial effort and risks breaking tests created by other teams. As an alternative, engineers often design tests to rely on CSS selectors or XPaths to interact with an application element.

Element locators, whether IDs, CSS selectors, or XPaths, can all change instantly as teams update and refactor their UIs (e.g., move elements to different areas of a page, add new page elements). As a result, tests that rely on single identifiers often fail, increasing maintenance efforts and reducing confidence in a test suite overall. Additionally, many popular development frameworks (e.g., React, Angular, Vue) dynamically generate (or update) random, complex element IDs and class names with each page render or release cycle. And modern, agile companies can deploy new application code several times a day, making it even harder for teams in charge of test automation to find robust identifiers for tests. They often have to collaborate with frontend developers to adapt automated tests to

application changes and ensure they have reliable ways to locate elements. development frameworks (e.g., React, Angular, Vue) dynamically generate (or update) random, complex element IDs and class names with each page render or release cycle. And modern, agile companies can deploy new application code several times a day, making it even harder for teams in charge of test automation to find robust identifiers for tests. They often have to collaborate with frontend developers to adapt automated tests to application changes and ensure they have reliable ways to locate elements.

A good way to improve the accuracy of your tests is by using a combination of locators, such as an element's class name and its location within a certain div or span, instead of a single locator. This provides more context for an element's location. Datadog's Synthetic browser tests address this pain point by using "multi-locator" algorithms to find UI elements. When a test searches for a specific element to interact with (e.g., a checkout button), Datadog looks at several different points of reference to help find it, including the XPath, text, classes, and what other elements are near it. These points of reference are turned into a set of locators, each uniquely defining the element.

If there is a UI change that modifies an element (e.g., moves it to another location), the test will automatically locate the element again based on the points of reference that were not affected by the change. Once the test completes successfully, Datadog will recompute, or "self-heal," any broken locators with updated values. This ensures that your tests do not break because of a simple UI change and can automatically adapt to the evolution of your application's UI.

In the next section, we'll look at how you can fine-tune your test notifications to ensure that you are only notified of legitimate failures.

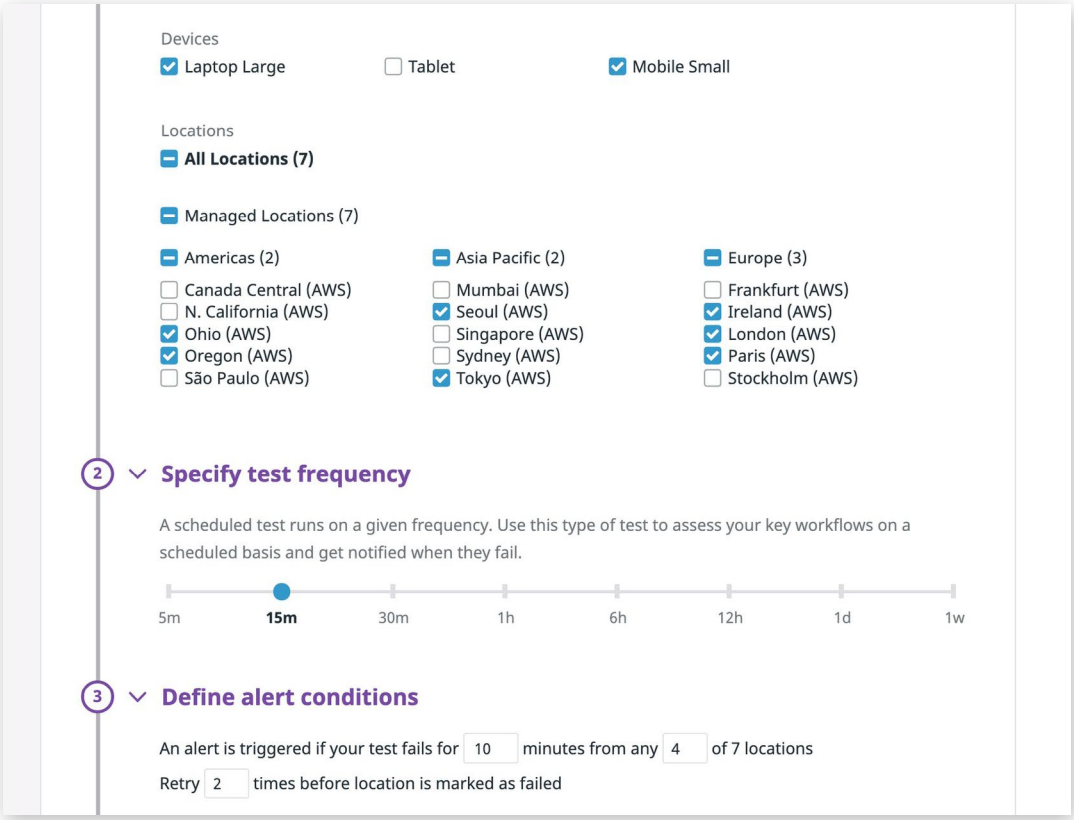
FINE-TUNE ALERTS THAT ARE RESILIENT TO ENVIRONMENT CHANGES

In the previous chapter, we discussed the importance of creating notifications that enable teams to respond faster to issues. Building reliable test suites also means knowing when to trigger alerts. Triggers can indeed be the result of legitimate bugs or they may simply be from changes in the testing environment. By designing alerts that are resilient to factors outside of your control, you can easily reduce false positives—and only be notified of real issues.

You can do this by fine-tuning your tests to have them automatically retry steps before sending a notification of a failure. This helps decrease the number of alerts that are triggered for issues that are not related to the tested workflow such as temporary network blips that do not directly impact your users.

Synthetic browser tests have a built-in wait and retry mechanism, making test steps resilient to potential network slowness. They also include

customizable rules for “fast retries” and alerting on test failures, without the need for modifying any code. Adjusting these rules can help further reduce false positives. For example, increasing the number of locations that are included in the alert condition can improve the odds of triggering an alert for a legitimate failure.



A failure in just one out of the seven configured locations seen in the example above could simply be a result of a network issue. In these cases, configuring Datadog’s fast retry feature to immediately run the test again is often a good way to distinguish a temporary network issue from an actual bug. Similarly, failures in four out of those seven locations is a greater indicator of an issue in the application.

As an application grows, you may need to adjust notifications to accommodate more complex testing environments. For example, you can fine-tune notifications to be a bit more permissive on issues in a staging environment than on those found in production by increasing the number of retries or the amount of time required before triggering the alert. This is useful if your staging environment is running on different hardware than what is used in production, which might mean increased latency.

Tests can fail for reasons outside of your control, and troubleshooting those failures is time consuming. By fine-tuning a test’s alerting conditions, you can ensure that all the alerts you get are reliable and deserve to be looked into. And when a test does fail, Datadog provides deep visibility into the cause by automatically including context—like screenshots, loaded resources,

frontend errors, and backend traces—around the failure for accelerated troubleshooting.

Reducing test flakiness and building alerts that are resilient to unexpected changes in your environments will improve your confidence in your test suite and make it easier to maintain. Next, we'll look at how you can make tests understandable, focused, and more descriptive in order to further simplify maintenance efforts.

Ensure tests are easy to understand for troubleshooting

A key part of test maintainability is ensuring that tests enable everyone on the team (from longtime members to newcomers) to quickly identify, investigate, and resolve errors. Some best practices for building accessible, easy-to-use (and easy-to-understand) tests include:

- eliminating unnecessary steps that complicate tests
- ensuring tests and test steps are descriptive enough to easily troubleshoot issues
- iconsolidating and reusing workflows to reduce points of maintenance

We'll take a look at these best practices in more detail next.

ELIMINATE UNNECESSARY STEPS THAT ADD COMPLEXITY TO TESTS

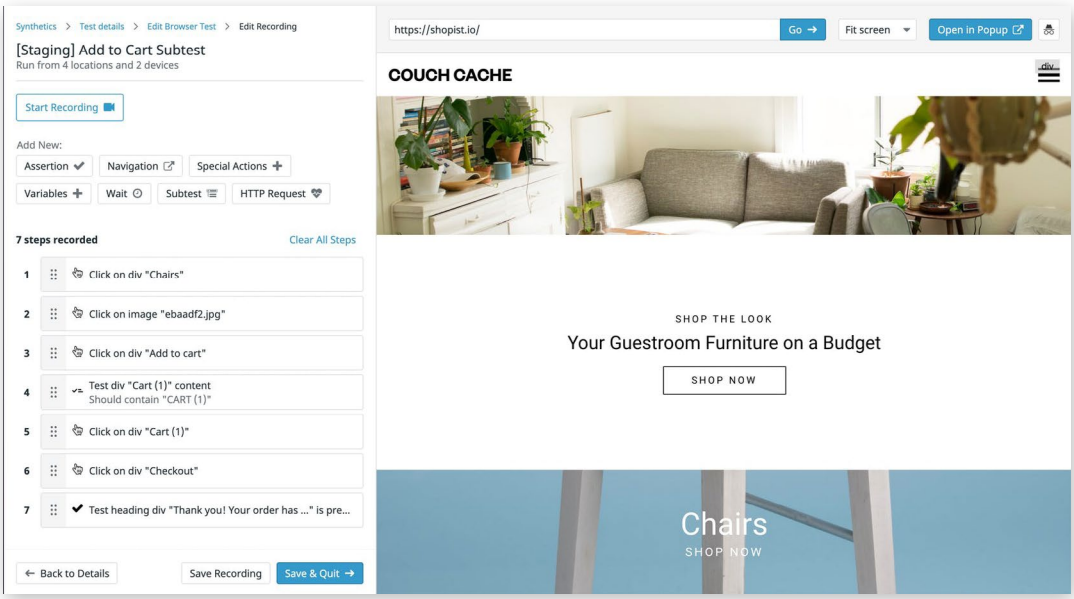
It's important to remove unnecessary steps that make it more difficult to maintain suites and identify legitimate issues in workflows. Two common examples of this are test steps that do not closely mimic how a user interacts with the application, and assertions that do not reflect what the user expects to see.

Tests with too many irrelevant steps and assertions increase execution times and scope, making it more difficult to identify the root cause of an issue in case of real failures. If any step does not mimic user behavior or expectations for the tested workflow, you can safely remove it from your test; you only need to verify actions or behavior relevant to the tested workflow. For example, a test that changes a mailing address in a user's profile only needs to include the specific steps for making the change (e.g., navigating to the profile, filling in the input field, etc.) and a simple assertion to confirm that the profile was successfully updated with the correct address. This helps tests stay focused on one workflow at a time.

By removing unnecessary steps, tests have fewer points of maintenance, allowing you to identify legitimate bugs more efficiently.

ADD DESCRIPTIVE TEST AND STEP TITLES FOR EASIER TROUBLESHOOTING

When a test fails, you need to quickly identify where the failure occurred before you can understand its cause. If tests are too vague, then you may end up spending valuable time manually recreating workflow steps simply to understand what was tested. Additionally, vague test names make it more difficult for new team members to quickly get up to speed on the application they are testing. Spending some time to ensure your tests are sufficiently self-explanatory will aid in uncovering the root cause of a failure and streamlining the onboarding process for your team, so you can resolve issues faster.



As seen in the example test above, step names can show the action a user would take as well as where they interact (e.g., button, input field). Datadog's built-in recording tool automatically names steps based on the step action or assertion. These names are often descriptive enough to understand what is going on in the test, but you can edit them if you need to add more context.

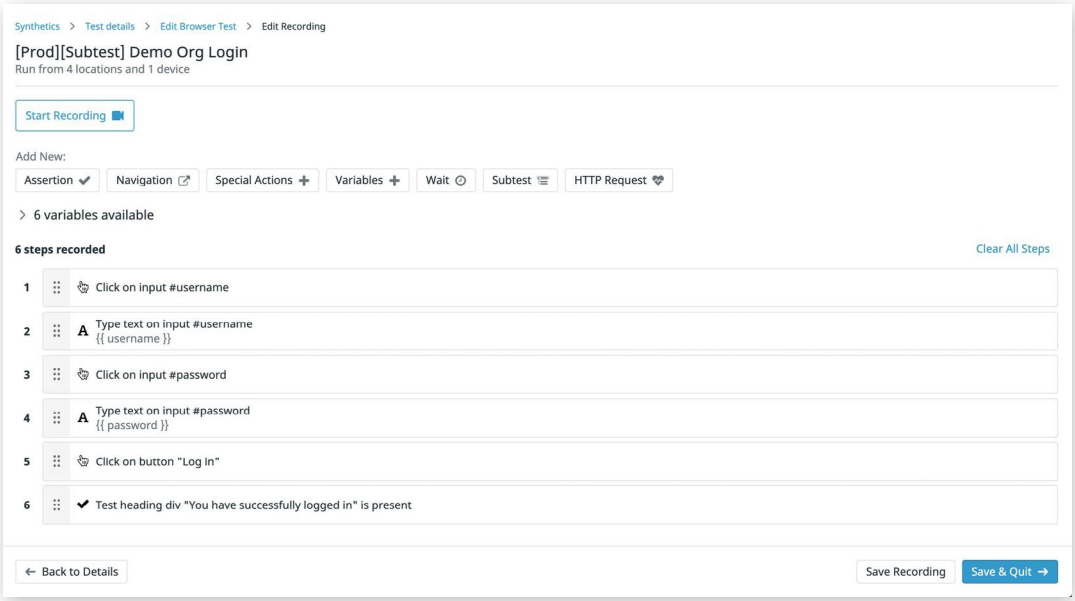
Descriptive test and step names help you understand the workflow that is being tested at a high level as well as the specific interactions in each step, reducing the amount of time your team spends manually recreating test steps. And since Datadog Synthetic tests automatically capture screen-shots of the UI for each test step, you can easily visualize and connect test actions to specific sections of the application UI to troubleshoot failures.

CONSOLIDATE AND REUSE WORKFLOW STEPS TO REDUCE POINTS OF MAINTENANCE

Test duplication is another factor that contributes to tests that are difficult to understand and troubleshoot. Test (and step) duplication can easily happen as team members add to the test suites to quickly support new application features. This complicates the process for creating new tests and adds more points of maintenance when workflows are updated. For example, if you have a group of tests that contain the same four individual steps for logging into an application, when there's a major change to the login process you have to spend time changing those steps for every single test.

A best practice for minimizing these types of issues is to consolidate tests and workflow steps where needed. As with creating new tests, you can use the DRY principle (i.e., "don't repeat yourself") to help you eliminate any redundant tests and test steps. This not only helps you maintain your tests but also ensures that they remain small in scope, focusing on testing one workflow at a time.

For example, Datadog Synthetic tests enable you to group reusable steps together with **subtests** so that you can use them across multiple tests. By minimizing duplicate test steps, you can both reduce the number of changes required and isolate changes to a handful of specific steps. This keeps tests within the scope of a single workflow and drastically cuts down on the time it takes to update them. Using our previous login example, you can create a "login" subtest that includes the four steps required to log into an application, and then use it in your other tests.



By consolidating workflows into reusable test components, similar to using a Page Object design pattern in test automation, you reduce the risk of creating several, disjointed tests for verifying the same functionality.

And when an application's UI or workflow is updated, you only have to update the shared components instead of individual tests.

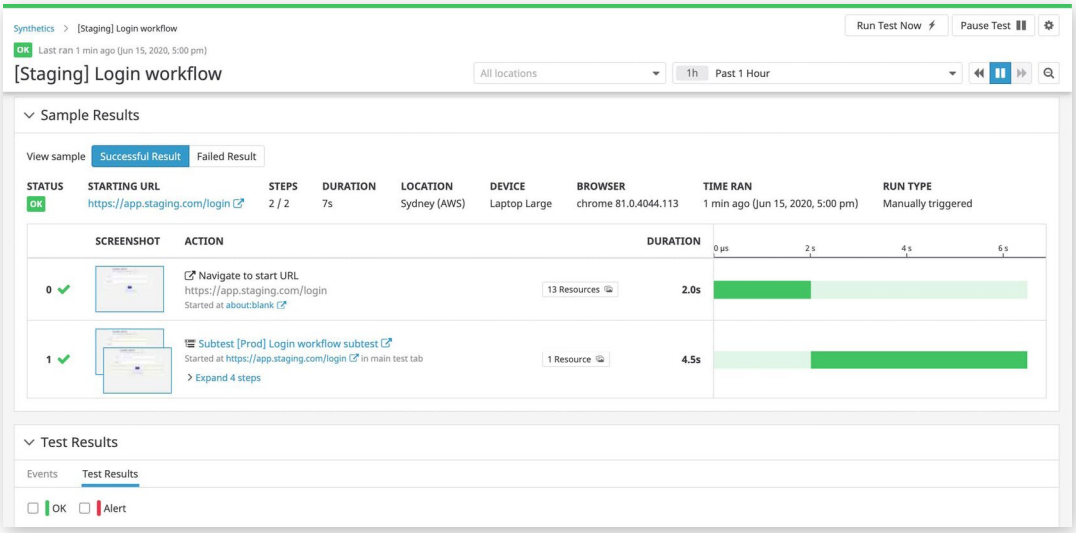
Another benefit of consolidating application workflows is that it makes it easier to leverage your test suite across all your environments. Your teams can use the same efficient suite for their own development, ensuring that core functionality is always tested.

Leverage a unique test suite across environments

As your application grows in complexity, you may create different build environments (e.g., development, staging, production) to develop, stage, and ship features. Testing in all of these environments creates safeguards for your application, thereby ensuring that it continues to function as expected, even as your team deploys code across new environments.

A key aspect of testing in different environments is leveraging a unique test suite, meaning that every environment uses the same tests to verify core functionality. This not only simplifies testing for your team but also further reduces test duplication. Just like you would re-use a Synthetic browser test as a subtest for a specific part of a journey in production (e.g., login), you can use subtests to run a full test scenario across multiple environments, without the need to recreate any individual test steps.

For instance, you can use a test running on production and test the exact same user journey on your staging environment using a subtest. With this setup, you will only need to modify the staging test's starting URL to point to the appropriate environment, which will automatically propagate to the subtest.



You can automatically use the same test steps across multiple environments with subtests.

When changes are deployed to production that require updates to the associated tests, you will only need to update the steps in one subtest—instead of manually updating separate tests for each environment.

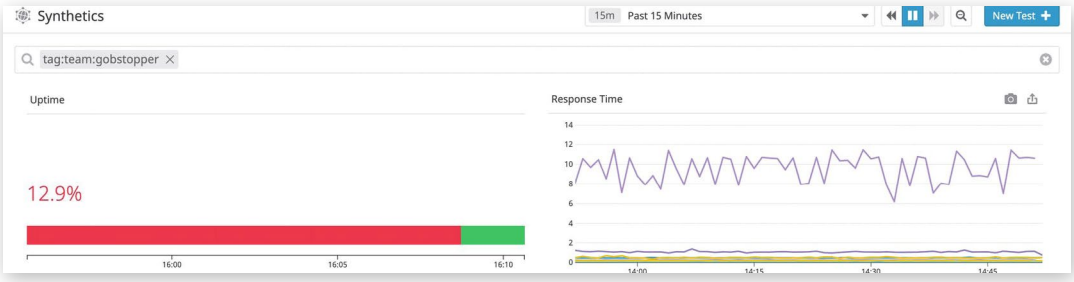
Next, we'll show how your existing continuous integration (CI) workflows can help you go one step further by easily sharing and using the same core suite of tests directly within your CI pipelines.

INTEGRATE YOUR TEST SUITE WITH CI WORKFLOWS

CI automates the processes teams use to rapidly package and deploy new features across environments, from development to production. Test suites that are not tightly integrated with CI workflows can make test maintenance and collaboration more difficult, as well as increase the risk of releasing broken features. Adding existing tests to CI reduces maintenance efforts by encouraging teams to use the same suite throughout development, instead of creating their own tests locally that may fall out of sync with others. Leveraging tests in CI pipelines is also a good way to increase test coverage, as it shifts functional testing to much earlier in the development process.

In addition to Datadog integrations with popular CI tools such as CircleCI, Jenkins, and Travis CI, Synthetic CI/CD Testing now allows you to run Synthetic tests directly within your CI workflows to optionally block deployments in case of degraded user experience. Instead of creating a separate test suite for your CI pipelines, you can simply leverage your existing Synthetic tests by referencing them in a local file located in your code repository. The CLI then auto-discovers the tests that should run with each new PR while giving you flexibility in test parameters to fully match your CI environment's technical specifications.

Though leveraging a unique test suite across all environments can make maintenance easier, it can also quickly make it harder for teams using that test suite to collaborate as they work on different parts of the applications. Datadog Synthetic tests ensure smooth collaboration between teams by allowing developers to quickly modify each test's CI execution rules directly in the UI, without having to make a pull request. If a feature breaks in staging and fails tests, you can decide to directly skip those tests in that environment, allowing developers working on a different feature to continue their efforts while others investigate the bug.



Once the bug is fixed, you can quickly switch the CI execution rules for those tests back to `blocking`. This will automatically block the CI pipeline if the test fails, so you can resolve the problem before you release the feature to your users.

You can configure CI execution rules for each test via the Datadog UI or the dedicated command line interface. This allows your teams to easily collaborate over unique, easily maintainable test suites while ensuring the rest of development is not halted. By adding the process of running test suites within your existing CI workflows, you promote collaboration across your teams and enable them to create a shared understanding of application behavior, use the same suites of tests for their features, and continue to increase test coverage. Integrating end-to-end tests with CI workflows also enables you to consistently verify that tests are up to date with application features and continue providing value for your team, which aids in maintenance efforts.

Test and monitor your application in one platform

With these best practices, you promote test maintainability as well as ensure a consistent, reliable user experience for your customers. To learn more about how you can begin creating efficient browser tests, check out the previous chapter or our documentation for more details.

Test on-premise applications with Datadog Synthetic private locations

Emily Chang

Synthetic monitoring lets you improve end user experience by proactively verifying that they can complete important transactions and access key endpoints. But your applications serve many users, from customers to all the employees who run your business. This makes testing the performance of any internal-facing services within your private network just as critical as monitoring your external-facing applications.

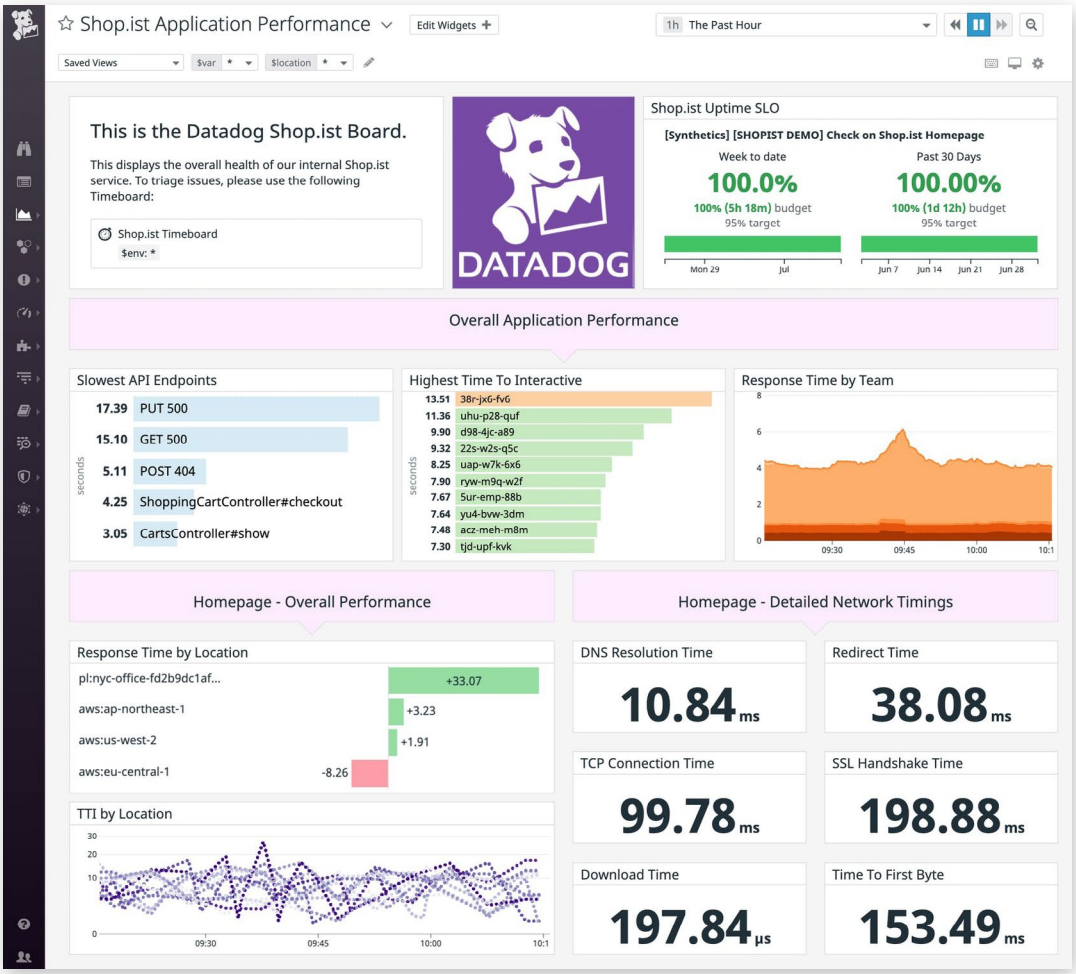
That's why we're happy to announce the availability of private locations in Datadog Synthetic Monitoring. Now, you can run Synthetic tests on all your applications, regardless of whether they're behind a firewall or open to the public. Once you incorporate private locations into your Datadog Synthetic tests, you'll be able to:

- monitor internal applications that your employees depend on
- launch Synthetic tests in your secured CI environment
- add locations that are significant to your business (e.g., factory, call center, or a specific geography)
- compare performance as experienced by users inside and outside your internal network

In this post, we will explore how private locations can expand your Synthetic test coverage—and how to set up your first private location worker in seconds.

Getting more visibility with private locations

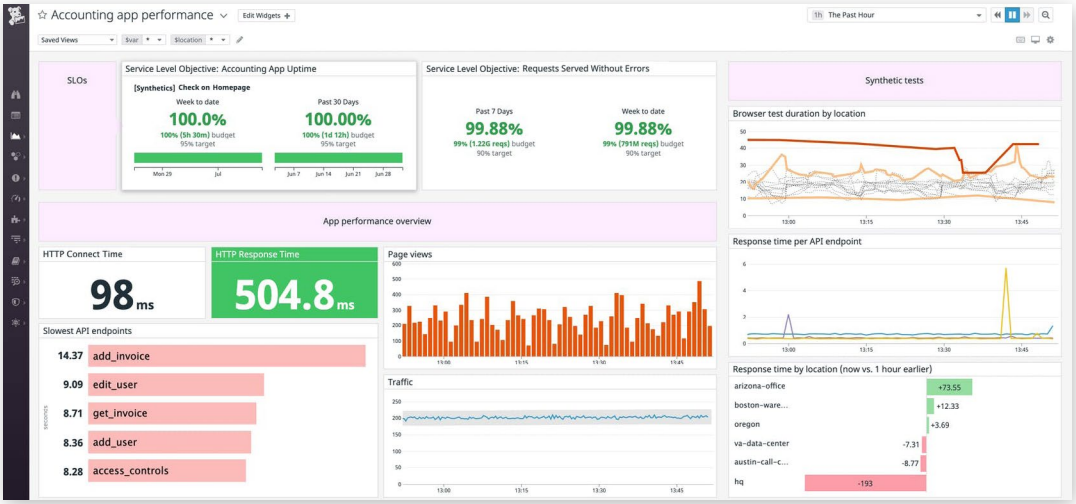
By default, Datadog Synthetic Monitoring provides managed locations around the globe to help you execute API tests and browser tests without setting up your own test infrastructure. With private locations, you can also launch tests from inside your network, allowing you to monitor an even wider range of applications.



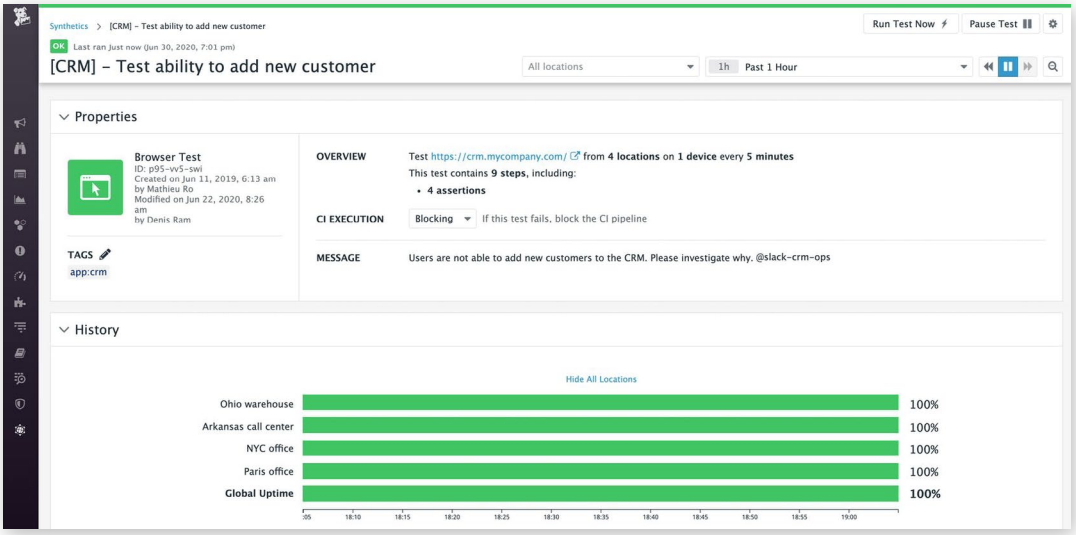
INTERNAL SYNTHETIC MONITORING FROM YOUR PRIVATE NETWORK

From enterprise resource planning (ERP) systems to customer relationship management (CRM) software, your employees depend on a range of applications—many of which are only accessible behind a firewall in your private network. Now you can use private Synthetic Monitoring to test the reliability and performance of your internal applications alongside the rest of your environment.

This means that engineers working on internal-facing applications can have access to the same benefits of Synthetic Monitoring as the other teams in your organization, including deeper insight into their user experience. Teams can also use Synthetic test results to compute the availability of their applications, which ultimately helps them fulfill the service level objectives (SLOs) they committed to their users.



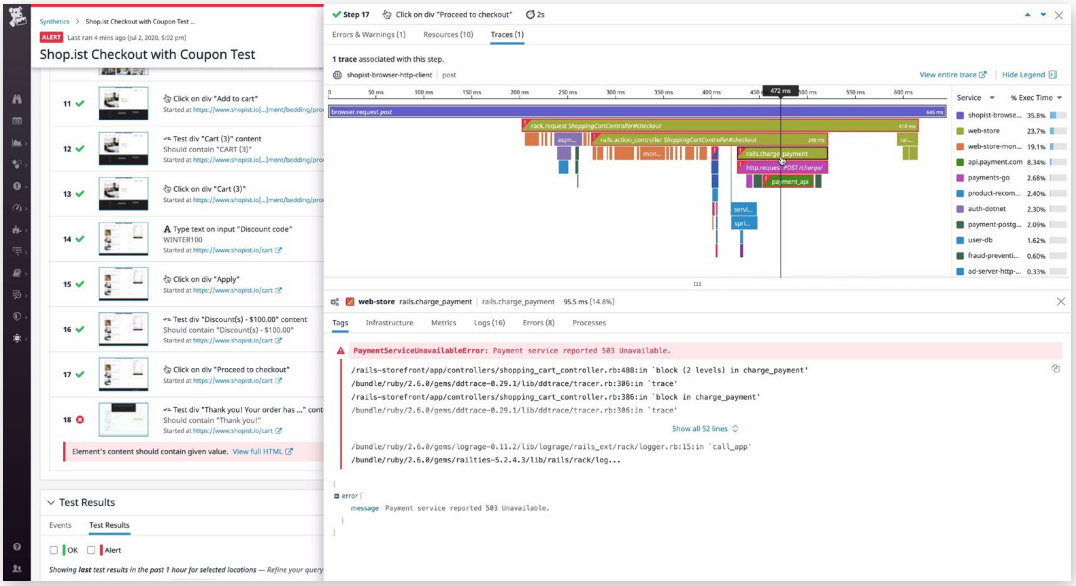
With Datadog Synthetic Monitoring, teams can get alerted to issues with internal applications before their users run into issues. In the example below, a Synthetic browser test helps verify that employees can add customers to an internal-facing CRM app. This test runs from a variety of private locations that are critical to the business.



VERIFY APP PERFORMANCE IN YOUR INTERNAL CI ENVIRONMENT

One of the most valuable aspects of Synthetic Monitoring is being able to verify the performance of features before they're released to your users. Once you've installed a private location on a machine that can access your CI environment, you can verify new features and endpoints from inside your network, and automatically block a deployment if a Synthetic test fails. This means that your teams will be able to debug issues earlier in the development process—before they get deployed to production and degrade the user experience.

The browser test below launched from our private CI environment and surfaced a problem with the latest version of an e-commerce application. Because this failed test automatically blocked the CI pipeline, the team has enough time to troubleshoot the issue before it gets released to the public.

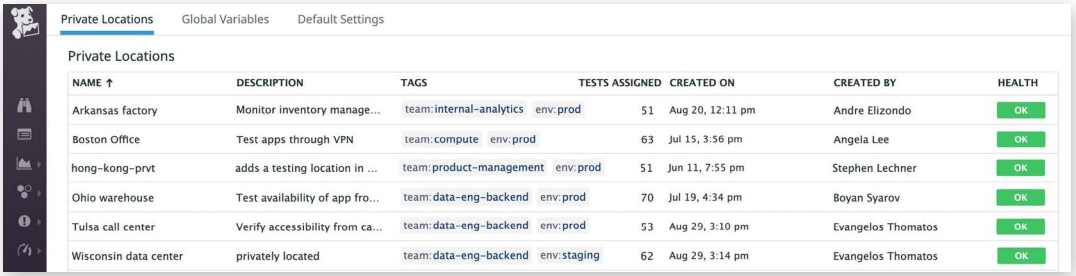


Just like Synthetic tests launched from public locations, any tests launched from private locations are automatically linked to backend data from APM, as well as logs and infrastructure metrics, for faster troubleshooting.

EXPAND THE GEOGRAPHICAL COVERAGE OF YOUR SYNTHETIC TESTS

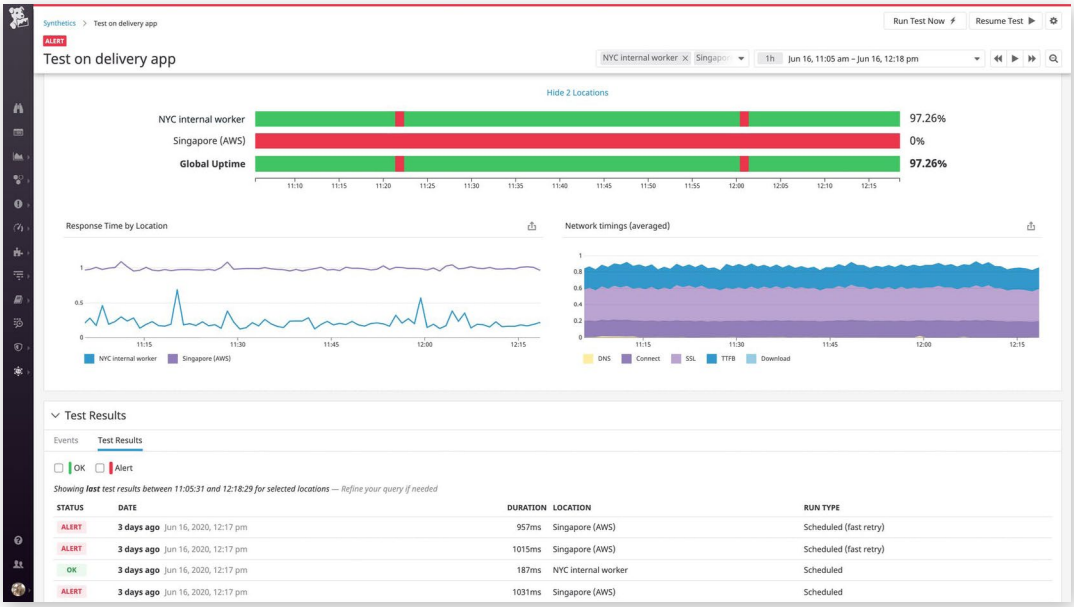
In today's global economy, your applications could be fielding traffic from just about anywhere around the world. You can set up your own private locations to simulate traffic from a broad range of locations. Private locations complement Datadog's out-of-the-box managed locations by letting you run Synthetic tests from regions where you know your key customers are.

You can also use private locations to verify that your applications perform as expected when accessed from mission-critical locations like your factories, offices, call centers, or data centers. In Datadog, you can see an overview of all the private locations that are currently configured, along with the number of tests assigned to each location.

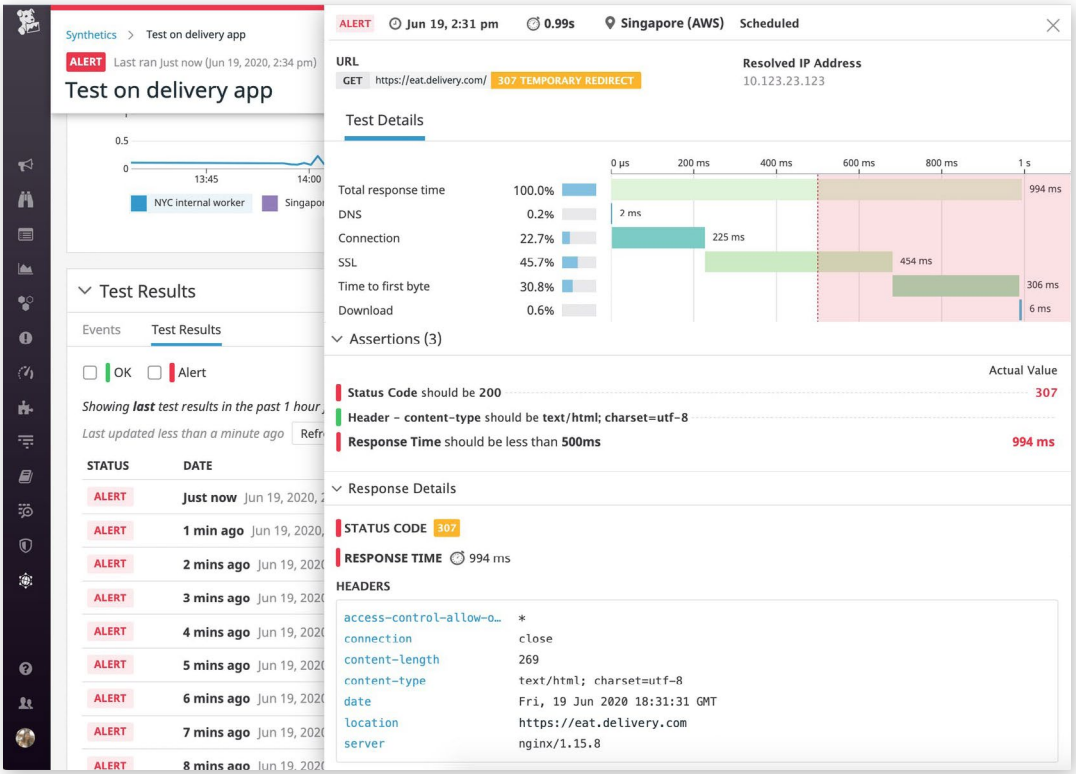


COMPARE APP PERFORMANCE ACROSS INTERNAL AND EXTERNAL USERS

Once you've deployed private location workers, you can use Datadog to compare application performance inside and outside your private network. In Datadog, you'll see out-of-the-box visualizations of the history of each Synthetic test, and you can easily drill down to a specific region or compare performance across locations. Below, you can see that the application endpoint responds much more quickly when accessed over the internal network compared to when it responds to a request from a Datadog-managed location in Singapore.

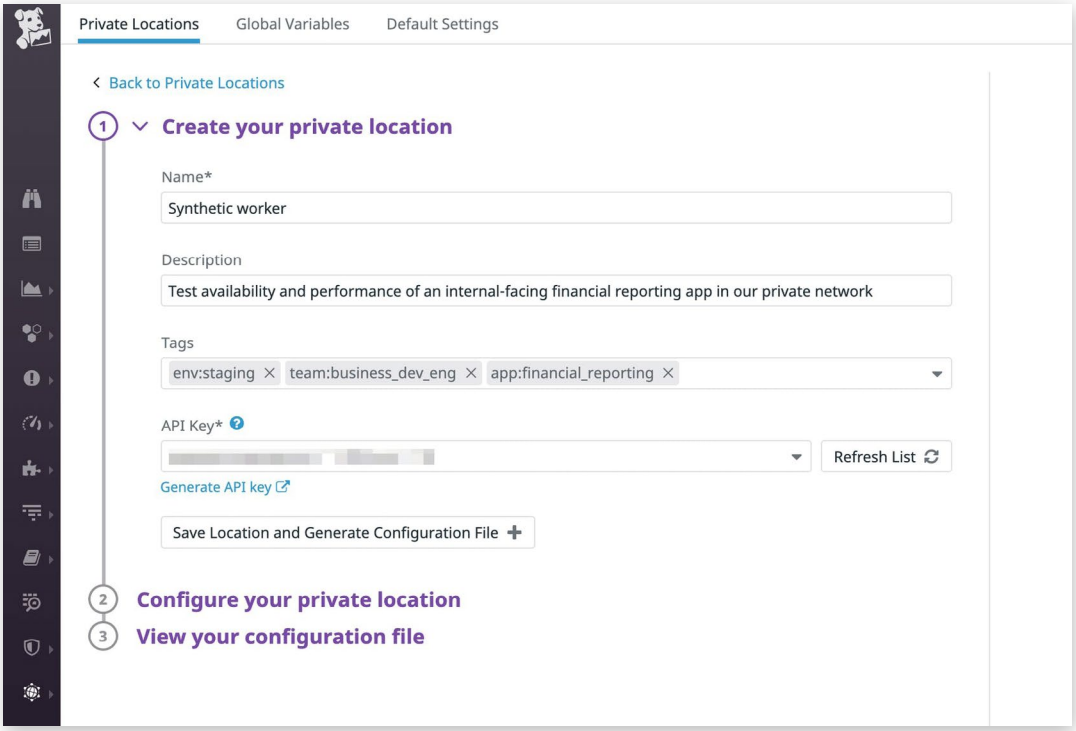


If your tests show that your applications are significantly slower to access over the public internet, you can investigate potential networking issues. Clicking on a slow API test, for example, takes you to a breakdown of network timings, including a waterfall visualization of each step, so you can pinpoint bottlenecks and other issues.

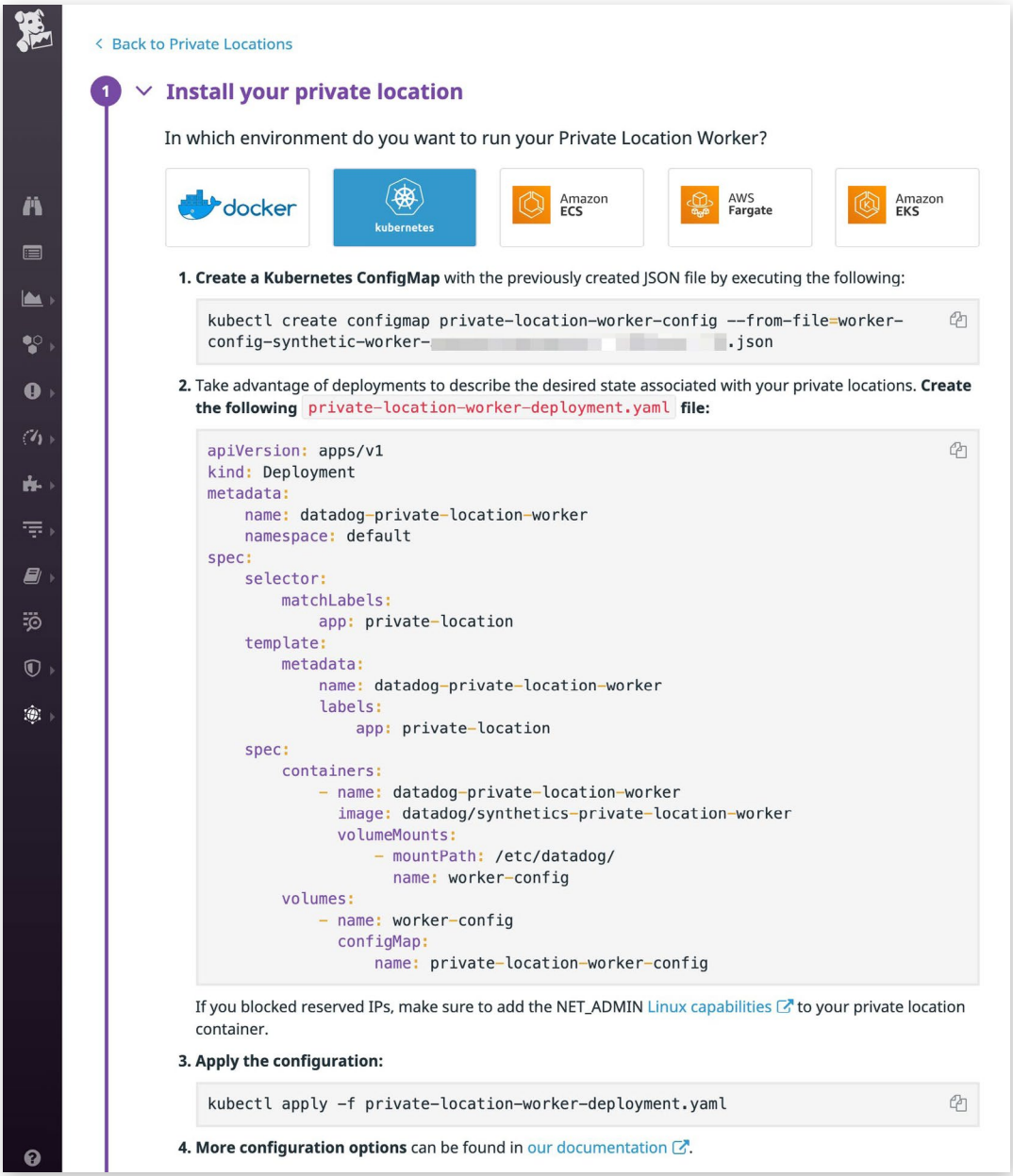


How to set up private Synthetic Monitoring locations

Setting up your first private location takes just a few seconds. In Datadog, navigate to the Private Locations view and click to create a new private location. Give your private location a name, description, and tag it with the name of your app or any other relevant information.

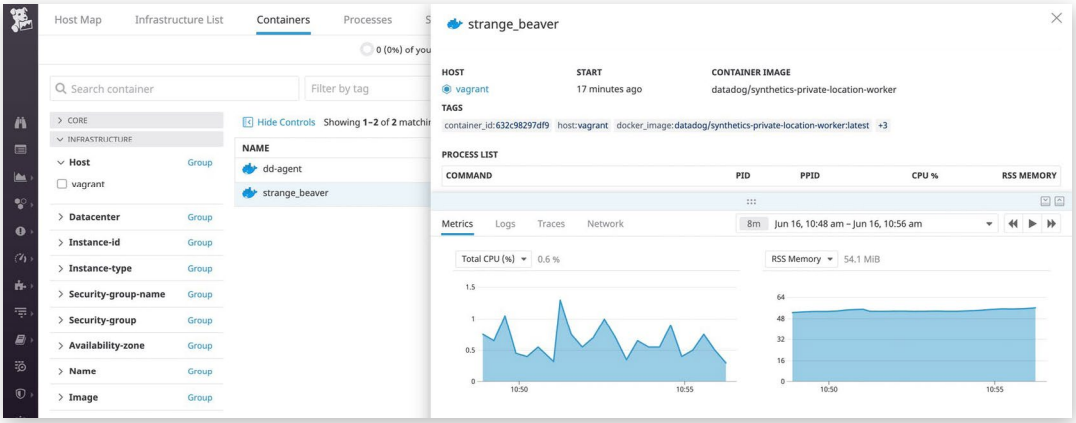


Datadog's built-in integrations with Docker Compose, Kubernetes, Amazon ECS, AWS Fargate, and Amazon EKS make it easy to deploy private Synthetic Monitoring on your existing environment.



We recommend launching one private location in each region of interest, and provisioning several containers for each location (though you can scale this as needed to reflect the frequency and number of tests you want to run). The concurrency parameter allows you to adjust the number of tests to run in parallel.

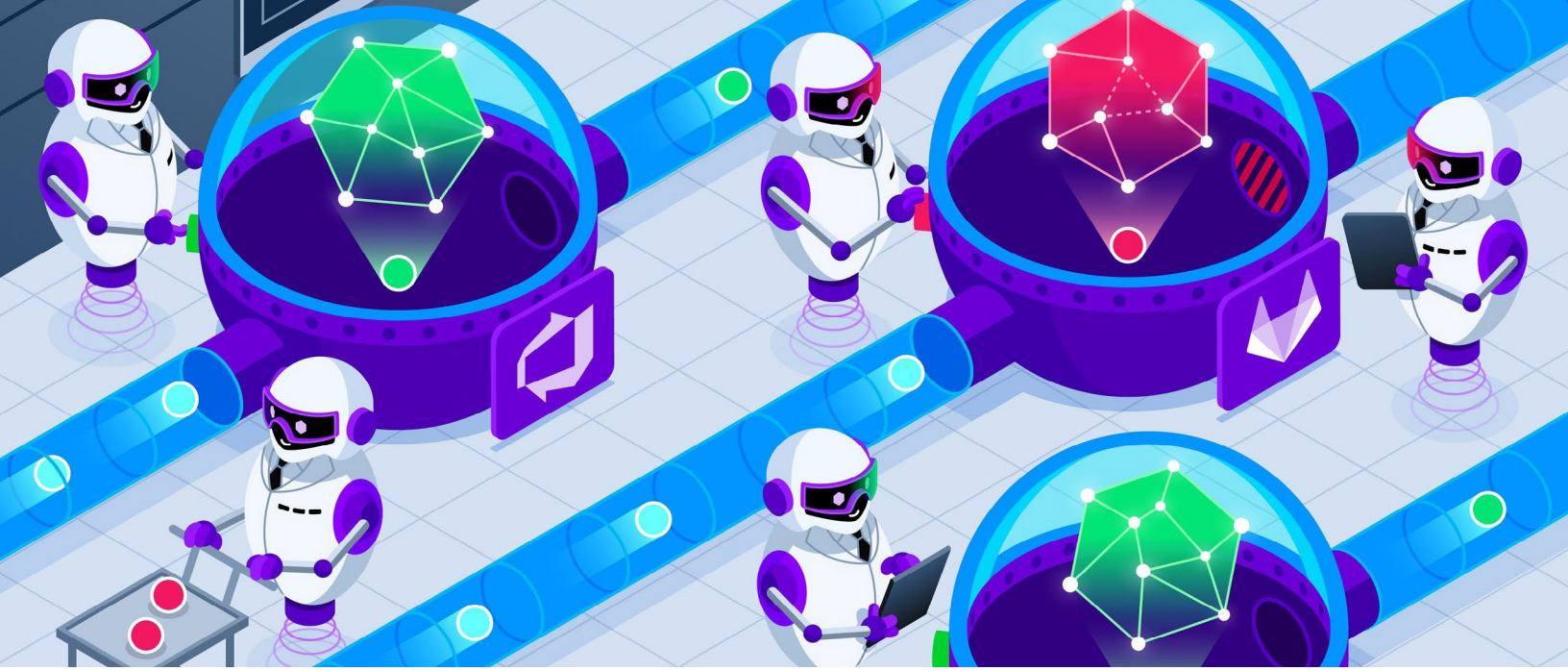
You can manage private locations in the Datadog app and even use the Datadog Agent to monitor your private location containers via our Docker integration.



Private locations work just like any Datadog-managed location—simply check them off when adding or editing an API or browser test. That means you can easily consolidate and manage all your Synthetic tests in one place, whether they're monitoring internal or external applications, or testing your applications in production or as part of your CI pipelines.

Launch Synthetic Monitoring from your private network

Private locations are generally available in Datadog Synthetic Monitoring. Check out our documentation to set up your first private location in minutes.



Incorporate Datadog Synthetic tests into your CI/CD pipeline

Betsy Sallee

Margot Lepizzera

Testing within the CI/CD pipeline, also known as shift-left testing, is a devops best practice that enables agile teams to continually assess the viability of new features at every stage of the development process. Running tests early and often makes it easier to catch issues before they impact your users, reduce technical debt, and foster efficient, cross-team collaboration.

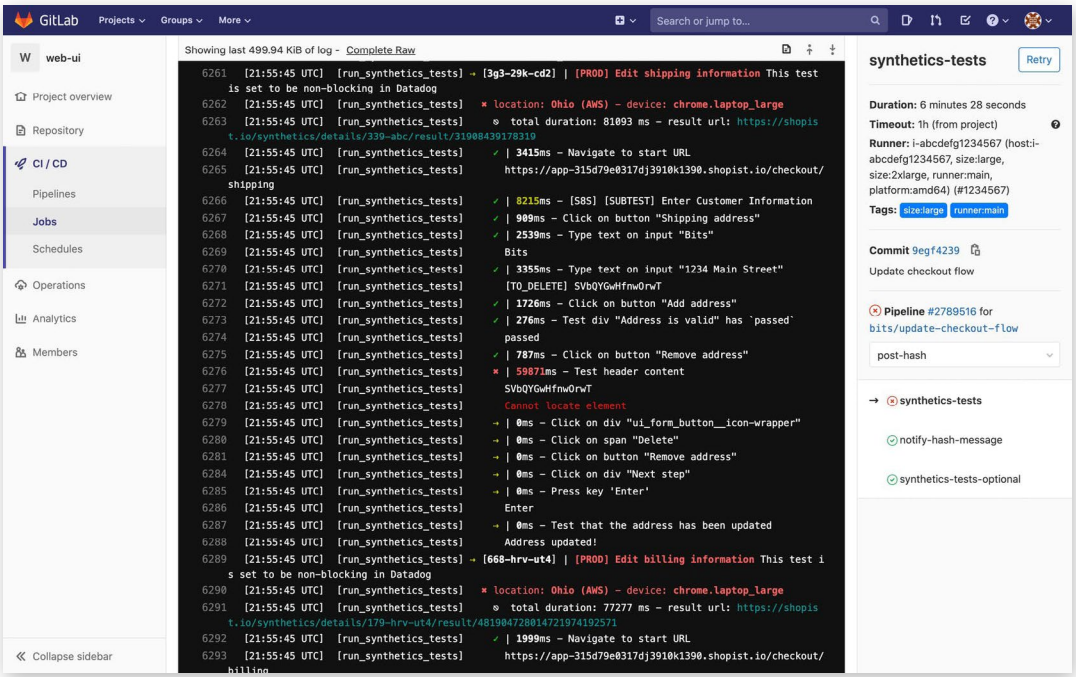
Datadog Synthetic CI/CD Testing offers a flexible approach to shift-left testing that allows you to incorporate your existing Datadog Synthetic tests at any point in your CI/CD pipeline, so you can maintain a positive user experience and minimize downtime. You can use a single Synthetic test suite for a range of testing scenarios—including canary testing and testing in your CI environment—which eliminates the need for environment-specific tests. Your scheduled tests will still run against your production environment as expected, but you'll be able to run those same tests—with whatever adjustments you see fit—on demand and at every stage of the development process.

Run Synthetic tests in your CI pipeline for earlier issue detection

The rise in popularity of the agile development methodology has made continuous integration testing crucial to the preservation of a healthy production environment. Agile teams frequently deploy new features and application improvements, so it's critical for companies to ensure that these rapid deployments do not introduce regressions. Tests that are run within the CI pipeline must therefore reliably verify that code changes have not degraded the user experience—without impeding the team's ability to ship changes quickly.

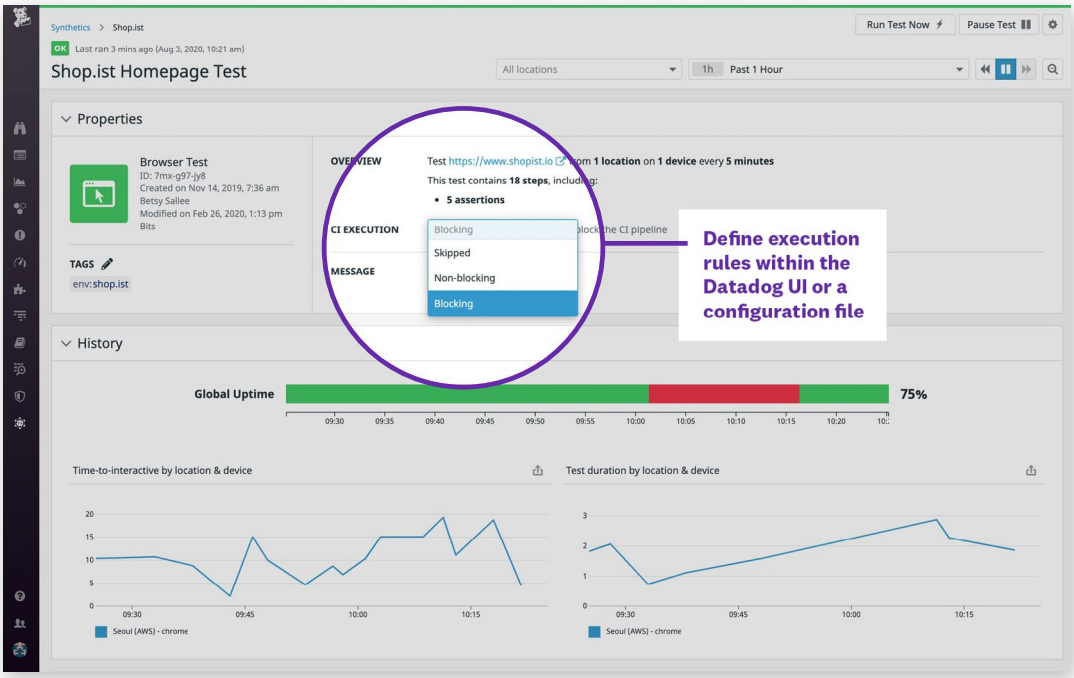
Traditionally, end-to-end tests have been too brittle and slow to be run in the CI pipelines of teams that ship code on a daily basis. Datadog Synthetic tests, however, are designed with reliability and efficiency in mind. Synthetic tests automatically update in response to UI changes and employ a wait and retry mechanism to ensure that failures are legitimate. And because Synthetic CI/CD Testing runs tests simultaneously rather than sequentially, it won't slow down your development process.

Datadog's integrations with popular CI tools such as GitLab, Jenkins, and Azure DevOps already allow you to collect events from your pipelines within the Datadog platform. Synthetic CI/CD Testing takes it a step further by enabling you to incorporate Synthetic tests in your CI pipelines for earlier issue detection. You can even view the results of Synthetic CI test runs directly within your CI, as shown in the following screenshot.



Visualize your test results in your CI/CD platform.

Adding Synthetic tests to your existing workflow is seamless. You can either send requests to a trigger endpoint and a polling endpoint or install an npm package and use the associated CLI as part of your CI build process. The CLI will autodiscover the tests to run in your repositories, execute the tests, and retrieve their results, blocking the deployment pipeline if key Synthetic tests fail in the CI environment. This helps prevent breaking changes from reaching staging or production—and safeguards the core functionality of your application by creating a safety net that only lets viable code progress through the pipeline.



Configure execution rules to block breaking changes from reaching production.

This same CLI tool can be used to upload your frontend JavaScript source maps, which makes the frontend errors you collect with Datadog RUM more meaningful. For example, you will be able to use Error Tracking to investigate when an error first occurred, which line of code fired it, and whether the latest release fixed it.

Run Synthetic tests on deployment to ensure safe releases

In addition to running Synthetic tests throughout the development cycle, you can also use Datadog Synthetic CI/CD Testing to execute tests as part of your CD process. Evaluating the state of your production application immediately after a deployment finishes enables you to detect potential regressions that could impact your users—and automatically trigger a rollback whenever a critical test fails. This capability is particularly useful for highly agile development teams that want to both own features from development to production and deploy frequently without risking extended downtime.

Synthetic CI/CD Testing can be utilized as an automatic safety measure in your canary deployments. If, for instance, you've deployed a new feature to a subset of users in a particular location, you can run your Synthetic tests against production in that location immediately after your changes go live. You can then use the API polling endpoint to send the results of your tests to a custom webhook, which, in the event of a failure, will trigger an automated rollback in order to revert to the most recent stable state.

Use a single, customizable test suite for every environment

Datadog Synthetic CI/CD Testing abstracts tests from their parameters, enabling you to leverage your existing, production-level test suite within your CI/CD pipeline, as well as within any environment that requires authentication or direct access to a network. The flexible nature of these Synthetic tests breaks down the silos between development, operations, QA, and product teams by eliminating the need to maintain separate testing scenarios for each stage of development. Simply create a test suite once, and tailor it for every use case.

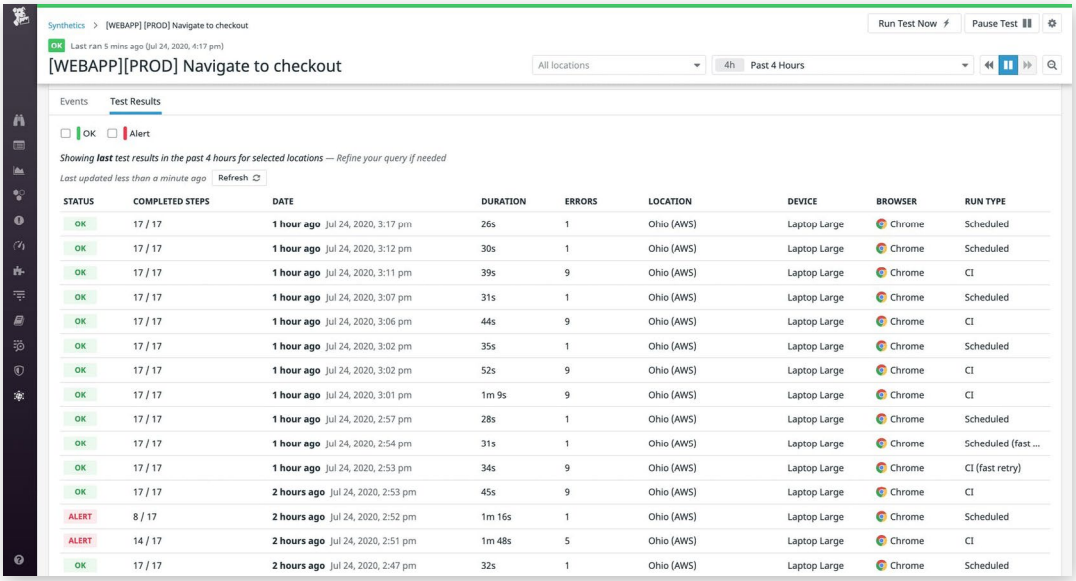
By default, Synthetic tests that are triggered through Synthetic CI/CD Testing will run according to the same configuration that they have on production. You can, however, specify overrides in order to meet the particular needs of each environment. This can be done for your entire test suite within a global configuration file (as demonstrated in the `global` section of the following code snippet) or at the individual test level.

```
{
   "apiKey": "<DATADOG_API_KEY>",
   "appKey": "<DATADOG_APPLICATION_KEY>",
   "global": {
       "basicAuth": { "username": "bits", "password": "labrador" },
       "startUrl": "{{URL}}?static_hash={{STATIC_HASH}}",
   },
   "proxy": {
     "host": "127.0.0.1",
     "port": 3128,
     "protocol": "http"
   }
}
```

For example, if your testing environment is behind a proxy or basic auth system, you can use overrides to add that specificity to your CI tests without overcomplicating your production-level tests. You can also use environment variables to dynamically configure an override value for your tests' `startUrl`. If, for instance, your pipeline generates a static hash for staging purposes whenever an engineer opens a PR, that static hash can be dynamically incorporated into the `startUrl` parameter, as shown above. For a complete list of customizable parameters, review the documentation.

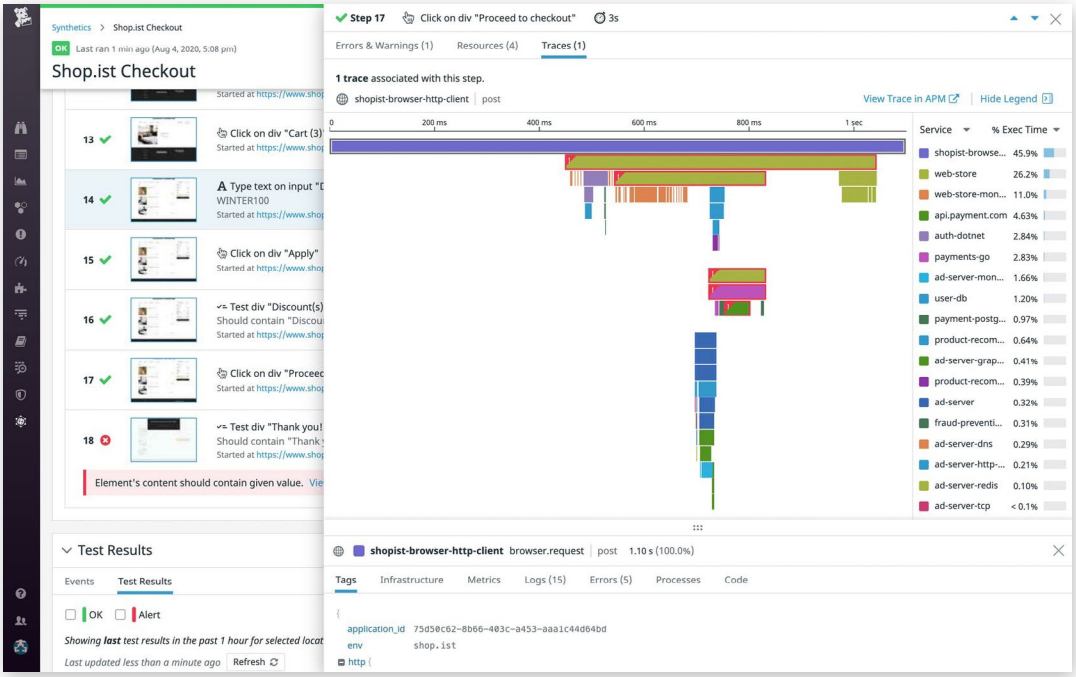
Shift system-wide troubleshooting to the left

Whenever an issue is surfaced by a Synthetic test, it's important for users to be able to easily troubleshoot it, regardless of the environment in which the test was run or where the issue originates within the stack. Datadog Synthetic Monitoring enables you to view test results from every environment—along with a wealth of contextual data—in the Datadog UI.



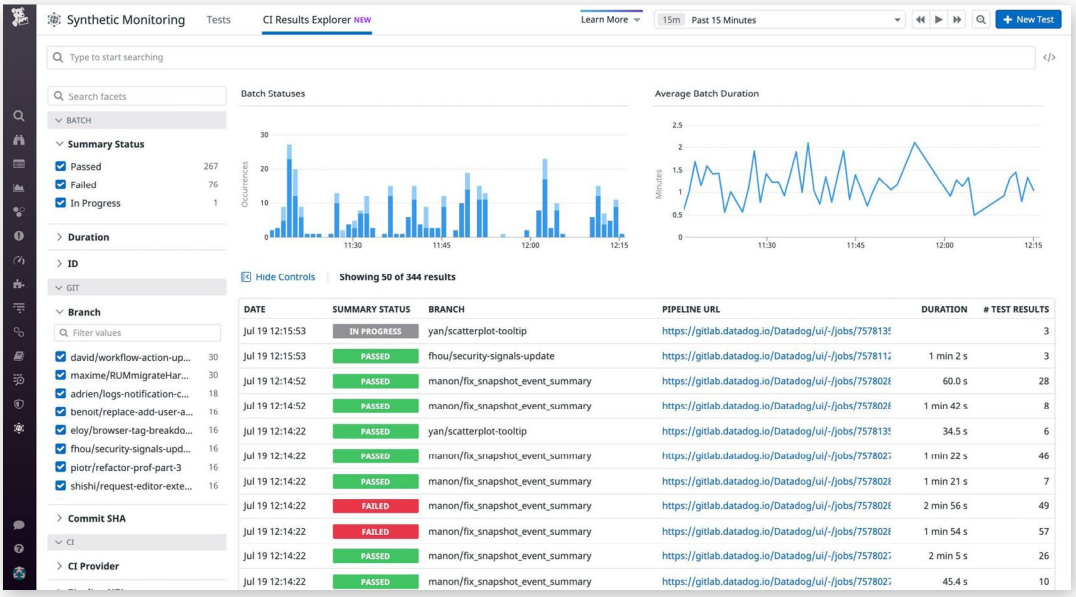
View test results from every environment in the Datadog UI.

Tests that are run in the CI/CD pipeline have the same level of context as your production-level test runs, so you can use relevant error messages, resources, and screenshots to pinpoint the precise point of failure at any stage of the development cycle. And because Synthetic Monitoring is tightly coupled with APM, each test run is automatically correlated with metrics, traces, and logs from your application and infrastructure, enabling you to conduct efficient investigations without switching contexts.



View metrics, traces, and logs that are relevant to each test

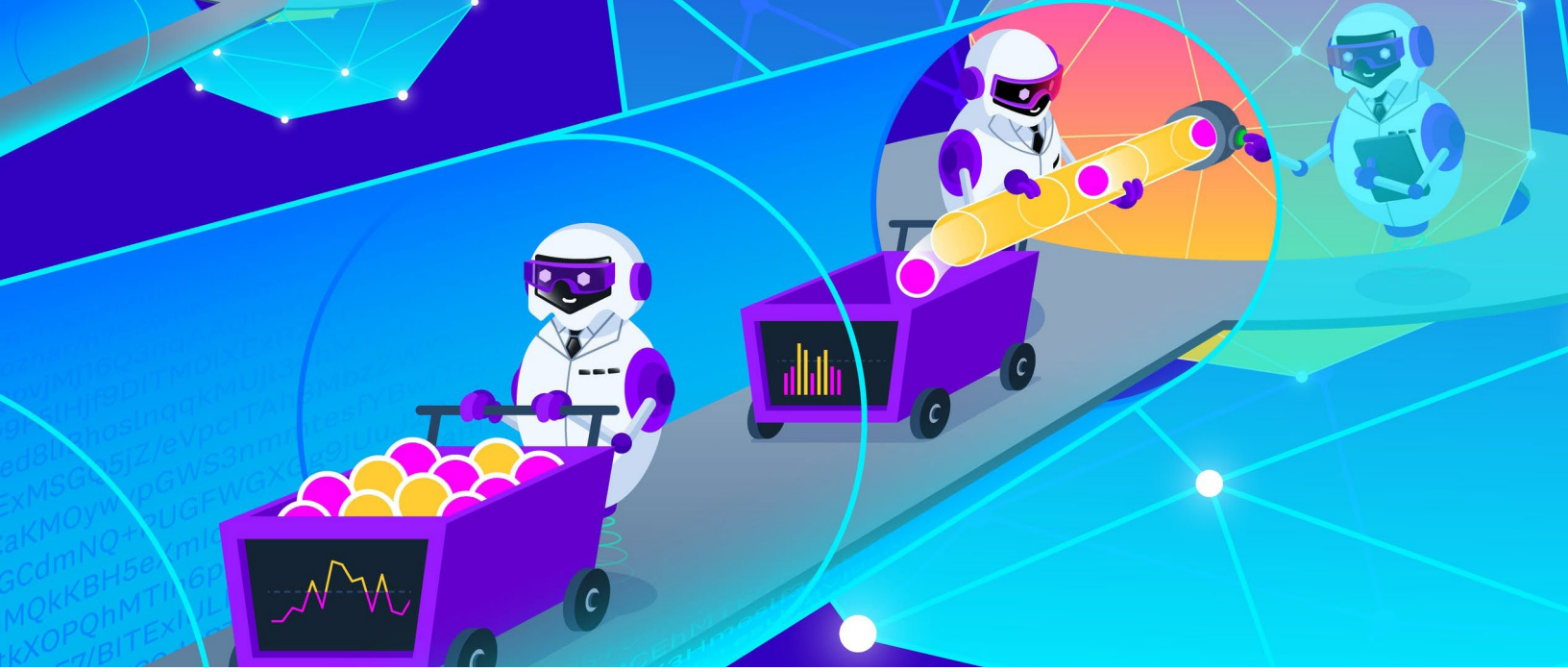
You can also visualize CI/CD test results in the context of their job executions in the CI Results Explorer, along with test batch statuses and average batch duration over time. This view can be filtered by facets that are relevant to the CI environment, such as `branch` and `Commit sha`, which makes it easy to focus your investigation on the job executions you care about. The CI Results Explorer also allows you to dive into any test batch in a single click in order to monitor the progress of individual test runs, identify failures, and compare results across different browsers, devices, and locations.



The CI Results Explorer displays all test batches that are executed in the CI/CD pipeline.

Get started with Synthetic CI/CD Testing

Datadog Synthetic CI/CD Testing allows you to leverage the power of your existing Synthetic test suite at any point in your CI/CD pipeline, so you can catch issues early, run safe deployments, and provide the best possible experience to your users. If you're already a Datadog customer and you would like more information about how to extend the power of Datadog Synthetic tests to your CI/CD pipeline, check out the documentation.



Test internal applications with Datadog's testing tunnel and private locations


Mallory Mooney


Margot Lepizzera

As part of your monitoring and testing strategy, you may run tests on different types of applications that are not publicly available—from local versions of production-level websites to internal applications that directly support your employees. Testing each one requires leveraging tools that allow you to verify functionality across a wide range of devices, browsers, and workflows while maintaining a secure environment. Datadog Synthetic Monitoring already lets you create your own custom probes (on-premise test runners) with private locations to routinely test and monitor all of your internal-facing applications. Now, for on-demand testing, you can also use Datadog's testing tunnel, a secure tunnel connection that requires little setup.

Private locations and the testing tunnel give you more flexibility over how you test applications in your internal environments, but each tool offers some unique benefits to support different testing goals.

In this guide, we'll cover:

- using the testing tunnel for on-demand testing in local and continuous integration (CI) environments
- creating private locations for durable testing and monitoring

CI and local testing with the testing tunnel

The testing tunnel leverages Datadog's command line interface (CLI) to create an end-to-end encrypted HTTP proxy between your infrastructure and Datadog. The CLI is an NPM package that enables you to launch Datadog Synthetic tests as part of your CI/CD pipelines, so you can identify and fix regressions in your applications before they impact your users. When used in conjunction with the testing tunnel feature, any test requests you send using the CLI are automatically routed through the `datadog-ci` client, allowing Datadog to access and test your internal applications.

Datadog's testing tunnel is designed to support CI pipelines and local development, so you can use it for:

- verifying hotfixes or new features locally before committing code
- running tests in environments reserved for CI pipelines (e.g., staging, user acceptance testing, etc.) or in ephemeral cloud environments

We'll look at how the tunnel's unique features and benefits can support these particular testing goals next.

AN EASY-TO-USE TOOL FOR TESTING ON DEMAND

A key benefit of the testing tunnel is its ease of use within existing infrastructure; it enables you to incorporate API and end-to-end tests into all of your workflows. For example, your teams (e.g., developers, testers) can use this tool out of the box to quickly verify that a hotfix for a time-sensitive issue, such as a service outage, works as expected locally before deploying it to end users. You can also use the tunnel service to run test suites as part of your CI pipelines without launching multiple browsers directly on CI servers, where processing power may be limited.

You can instantly create a tunnel connection to run tests using a simple command:

```
datadog-ci synthetics run-tests --config synthetics.global.json --tunnel
```

The example command above will open a WebSocket Secure tunnel connection and launch the suite of tests defined in your local machine's or CI server's test configuration files. These files include the public IDs of the tests that you want to run, along with other configuration attributes, such as endpoint URLs, device IDs, and locations.

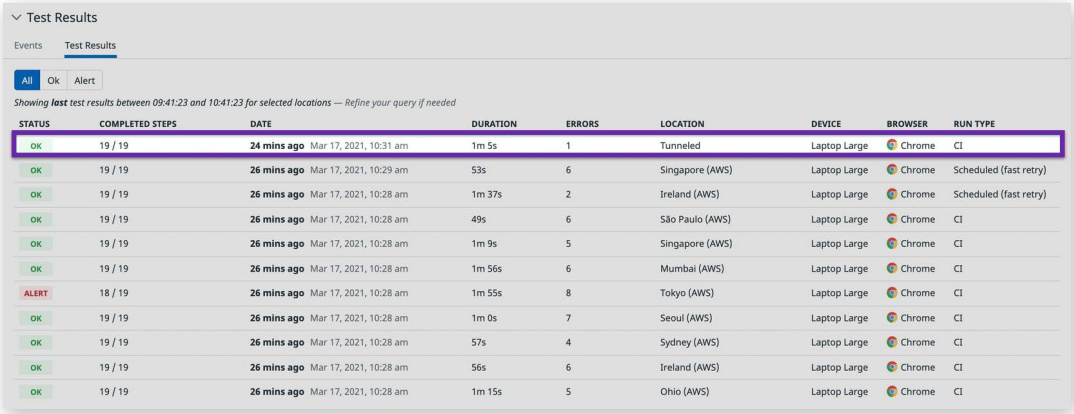
Since the tunnel is built into Datadog's CLI, it enables you to quickly start testing your internal applications at any time.

LAUNCH TESTS WITH MINIMAL OVERHEAD

The tunnel is independent of existing infrastructure, so you can use it without deploying, maintaining, or monitoring additional services. Tests launched via the tunnel are executed from Datadog-managed locations. This means that as long as the host running Datadog's CI client can create the connections needed to run multiple tests, Datadog will automatically scale to support the increased load as needed. Tunnel connections then end when the Datadog CI client receives all necessary results, so you do not need to track long-running connections to your network.

The tunnel also makes it easy to dynamically override where your tests run with Datadog's built-in environment variables, so you can continue testing your applications without interruption, even as the environment you are testing changes. This includes environments that rely on ephemeral cloud instances and containers. For example, you can automatically pass the URL of a newly deployed application instance as the starting URL for any tests launched with the tunnel, instead of hard coding that data into your tests.

Datadog shows which tests were launched through the tunnel service so you can monitor them alongside the rest of your synthetic tests. For any test failures, Datadog provides end-to-end visibility for troubleshooting and resolving issues, including details such as screenshots of the UI, JavaScript and network errors, load times for page resources, and APM traces if your test is hitting an instrumented service endpoint.



Now that we've covered the benefits of using the testing tunnel for straightforward, on-demand testing, we'll look at how Datadog's private locations support your long-term testing and monitoring goals.

Durable testing and monitoring using private locations

As we've seen, the testing tunnel offers a turn-key solution for secure, rapid testing in short-lived environments. For organizations who need to regularly test and monitor applications hosted on permanent environments, Datadog provides private locations: Docker containers that you can deploy as custom points of presence (e.g., data centers, geographic locations) inside of your infrastructure using orchestration tools like Docker Compose, Kubernetes, AWS Fargate, and Amazon ECS.

Because private locations are deployed as a durable probing service for launching your tests, they can be useful for:

- customizing and managing a centralized testing tool that is readily available for teams across your organization
- triggering tests on long-running environments (e.g., staging, pre-production) as part of your CI/CD pipelines
- regularly running tests on internal applications that are hosted on private networks to ensure you can maintain your availability SLOs

We'll look at how you can use private locations to create a customizable, scalable, and easily accessible service in more detail next.

A FULLY-FLEDGED AND CUSTOMIZABLE TESTING SERVICE FOR INTERNAL APPLICATIONS

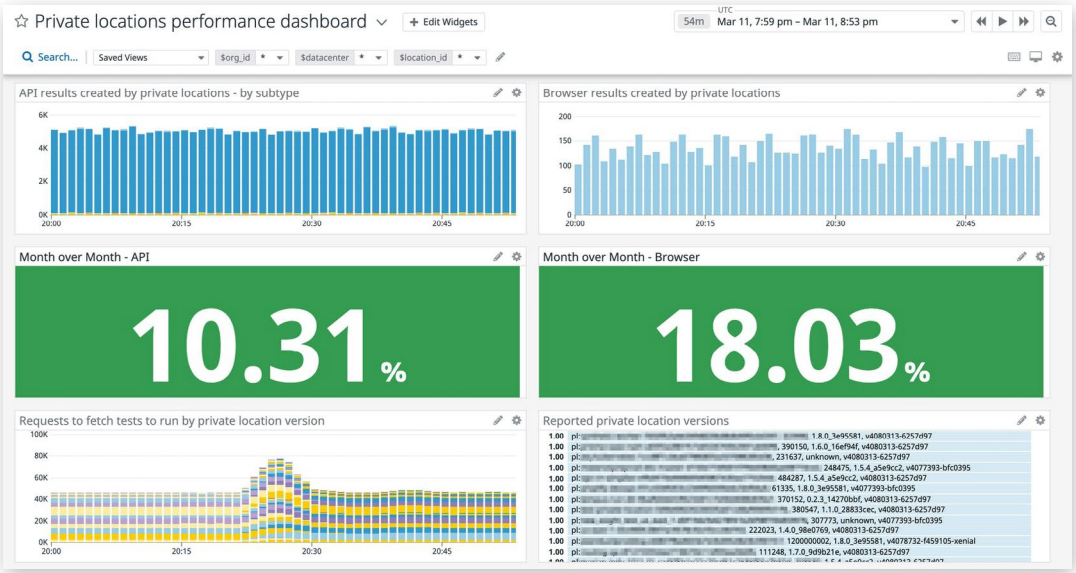
Since testing is a crucial part of building resilient applications, you need a system that can support testing a growing network of services as your organization scales. Using private locations, your SRE teams have greater flexibility in not only customizing a probing service for every use case—via their preferred orchestration tool—but also ensuring it can scale to continually verify functionality and monitor application performance.

Private locations come with a number of parameters you can use to match your infrastructure and private network configurations, such as built-in controls to block IPs in order to prevent users from creating synthetic tests on potentially sensitive endpoints in reserved IP ranges. And, as your applications grow, you can horizontally or vertically scale your locations in order to run more synthetic tests concurrently, enabling you to seamlessly test newly added features alongside existing functionality. Leveraging these measures ensure your applications—and your test infrastructure—remain secure and continue supporting your users.

Monitoring private locations

Private locations are designed to regularly test and monitor your applications long term. Because of their longevity—and since tests run on the servers where you've deployed private locations—you need to ensure that every location is working as expected. Datadog provides visibility into your entire infrastructure, so you can monitor the performance of your custom

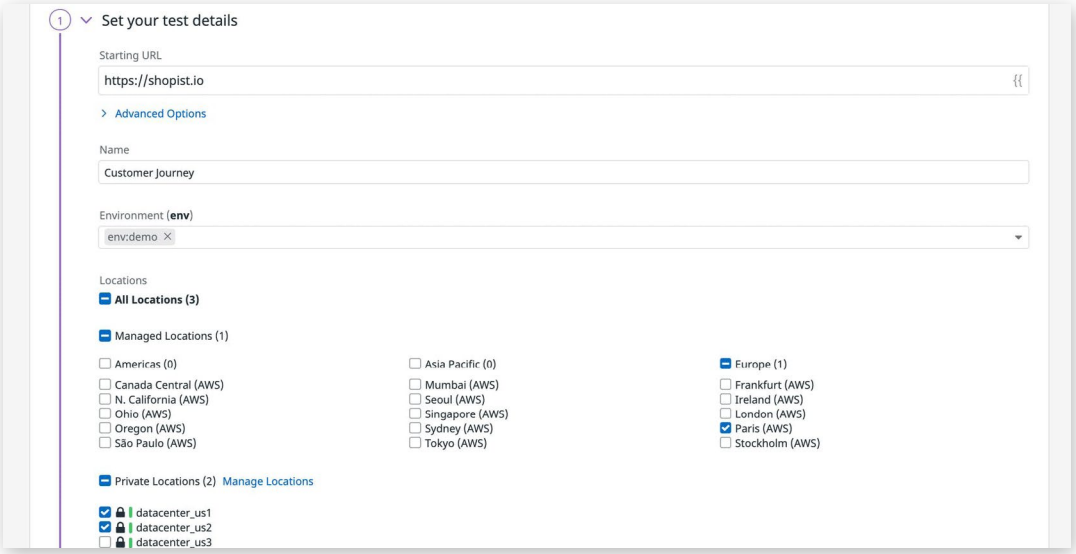
locations in one place. For example, you can create custom dashboards to get a high-level overview of all of your private locations and easily monitor usage, as seen below.



You can also use the Datadog Agent to get deeper visibility into the state of your private locations' underlying containers and confirm that they are performing optimally. If you notice unusual changes in the tests executed by your private location, such as a significant increase in response time, you can then drill down to the affected container in order to troubleshoot further.

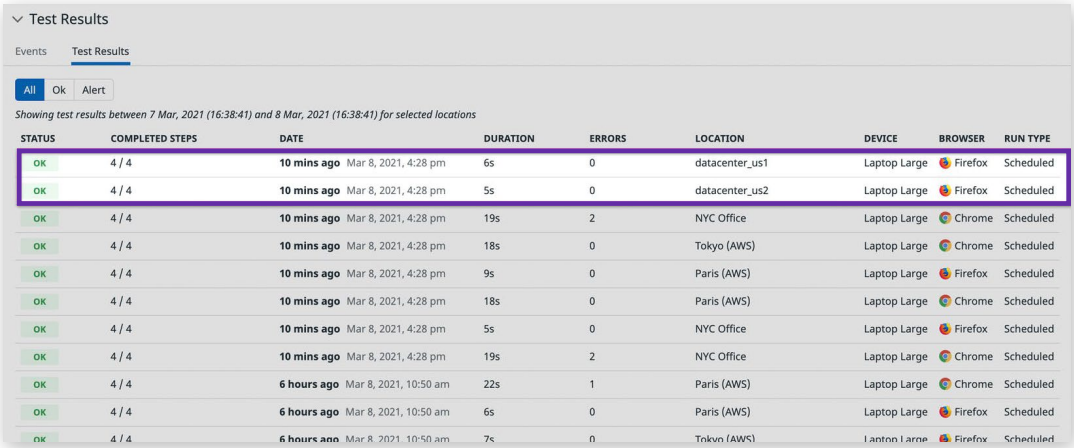
SELF-SERVICE TESTING FOR EVERY TEAM

Once deployed, private locations provide a centralized and readily available service for testing, so your teams can create their own tests and assign them to specific locations in one click.



All of your teams can easily add available private locations when creating new tests.

This enables your teams to routinely test applications under a wide variety of conditions. For example, your corporate IT team can launch tests on private locations deployed to multiple data centers to ensure that your company intranet or a key SaaS provider is performing optimally for a growing team of distributed employees, regardless of their location.



Or, your QA team can leverage the same tests and private locations as part of their CI/CD pipelines to verify that key workflows are still accessible to users after a canary deployment of new intranet features.

Your map for comprehensive internal application testing

With private locations and the testing tunnel, you have more options for testing and monitoring your internal-facing applications. Each service offers unique features to help you accomplish your testing goals, whether they require long-running probing services or the ability to quickly launch tests on demand and with little setup. Check out the documentation for private locations and the tunnel service (currently in public beta) to learn how to get started with both.

Get started with Datadog today

In this guide, we reviewed several best practices for proactively monitoring your application's frontend performance with end-to-end tests. Datadog Synthetic Monitoring makes it easy to implement these strategies at your organization, so you can identify and resolve user-facing issues as quickly as possible. If you're not yet a Datadog customer, get started today with a free, 14-day trial.


DATADOG