What to Consider When Building or Buying a Load Testing Solution

Introduction

People are spending an increasing amount of time on services like Google, Amazon, and Instagram. These well-engineered sites have set a high bar when it comes to user experience (UX)- people now expect every application they use to be fast and reliable. A great UX is crucial to the success of your app, which means that accurate and efficient load testing has never been more important. You need a powerful solution that not only handles load testing basics but also complex user scenarios and continuous performance testing.

If you're looking to start doing load testing, then you'll start by evaluating all the implementation options:

- Open-source software (OSS)
- On-premises vendors
- Software as a Service (SaaS)
- A mix of OSS and SaaS

Many load testing tools, like **k6**, are open-source and provide the option to store the testing results to various backends, which is why many software teams start building their own load testing solution.

> Starting a basic load testing solution is easy, but developing a scalable solution for continuous performance testing is not.

In this white paper, we look at:

- The risks and costs of building your own load testing solution.
- Features you should consider when implementing a load testing solution.
- Challenges during the various load testing stages.
- Outgrowing the initial solution.

The risks and costs of building

We've seen companies like Facebook and GitLab create their own load testing solutions. But for most of their software teams, implementing load testing is another internal project that usually ends up with minimal functionality and additional maintenance burdens. Buying a SaaS would seem like a better alternative to building a solution in-house. However, when deciding on building or buying, you should first assess your testing needs and organizational structure, and then ask yourself the following questions:

- Who is going to use the testing solution and how are they going to use it?
- What are the key features we need in our testing solution?
- Do we have the time and budget to build a solution and the resources to maintain it?

In general, we recommend that you start with an open-source load testing tool to build your proof of concept (POC). If you test frequently and your development process requires that you prepare for complex user scenarios, you should consider moving to a robust solution that handles more than basic testing. A robust load testing solution, whether built in-house or purchased, will allow you to go beyond the minimum testing requirements and expand across roles.

Companies often underestimate the domain expertise that goes into a proven load testing solution and how much work it takes to build a reliable platform. Although for engineering managers, the risks of building an internal developer tool are typically well known. For example, they know about the potential technical debt, building a minimum viable solution (MVP), lack of focus, and expected project delays.

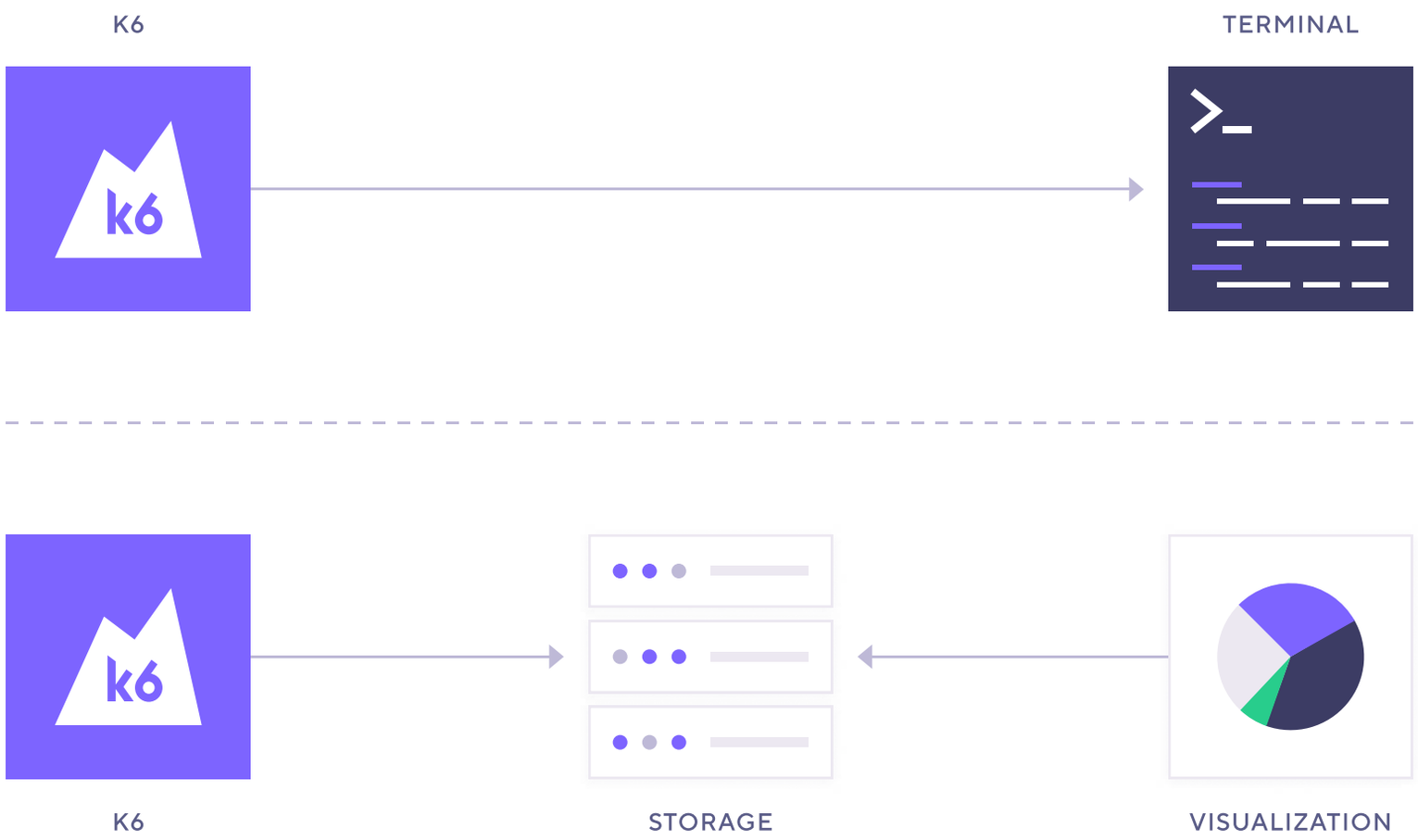
Engineering managers should calculate the Total Cost of Ownership (TCO) of both solutions—in-house and SaaS. However, the TCO and the risks to your organization are directly related to the complexity of your solution. Therefore, you should start your POC by figuring out what your testing needs are now and in the future.

Planning your testing

As we mentioned earlier, you should start with an open-source load testing tool to build your POC. Once your initial solution works well, you could then incorporate capabilities that go beyond basic load testing.

If you decide to start your POC, first, it's very important to identify the teams that will use the tool- QA testers, developers, site reliability engineers (SREs), and others will likely play a role in load testing for your organization.

You should leave the tool decision to the people who will actively develop the tests. They can then choose a load testing tool and envision the architecture of the testing solution. A minimum load testing solution is:



The load testing tool is the core component of your load testing solution- you use it to run a test and visualize the results. Additionally, many open-source tools can send the results to a backend for you to create visualizations later with a third-party analytics tool.

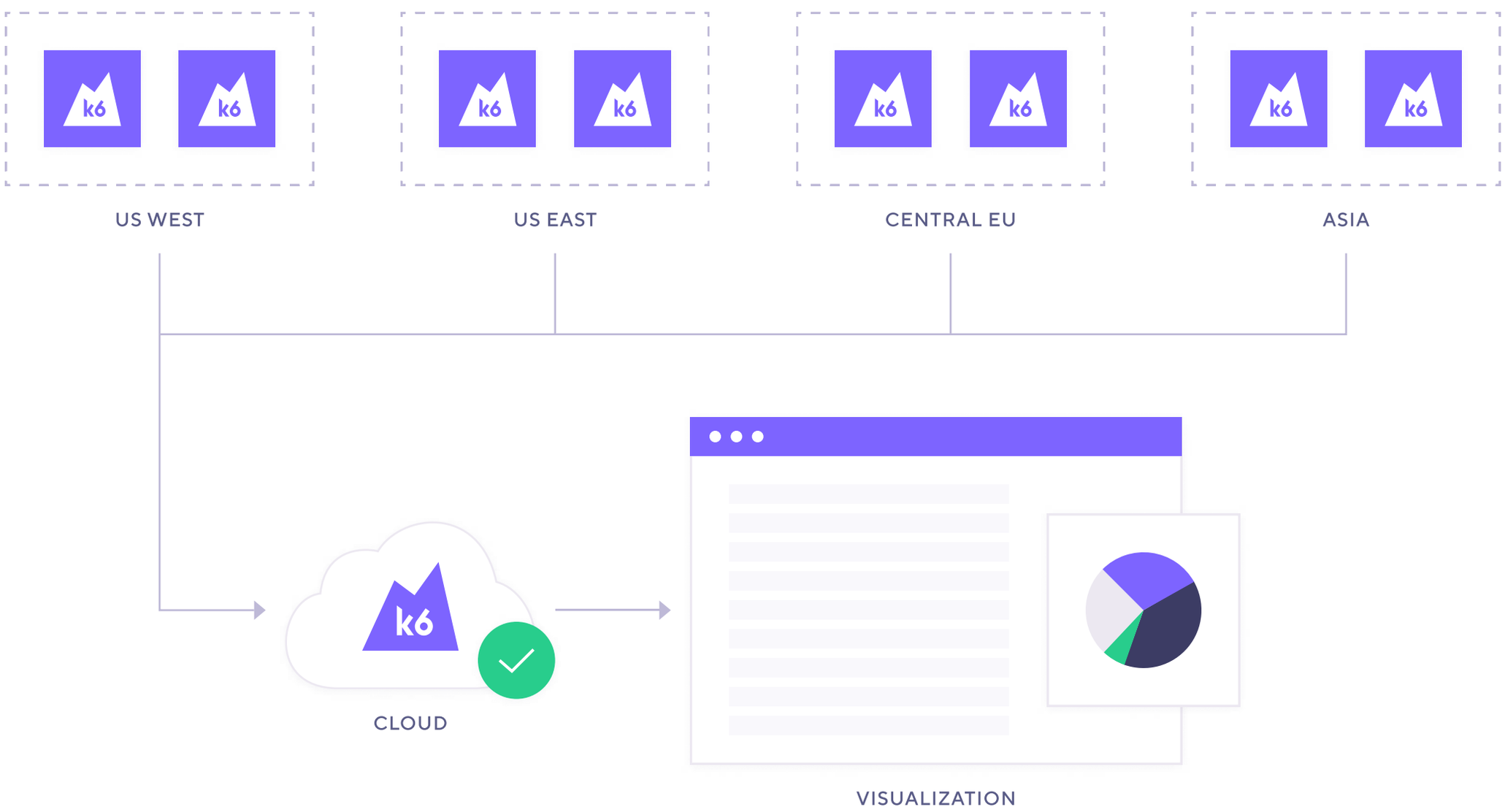
A fully open-source solution is a cost-effective choice and recommended if load testing is not crucial in your development process. Once the testing becomes more frequent or essential however, teams usually outgrow the initial solution, demanding the ability to:

- Run large tests or distribute the test execution across multiple machines.
- Do continuous performance testing.
- Allow different roles to participate in the testing.
- Track and report app performance and reliability over time.
- Improve the solution's reliability and usability.

Scaling your tests horizontally

You could run a large load test using a single machine, either fine-tuning its configuration or increasing the machine resources: CPU, memory, or network - also known as vertical scaling. However, depending on the type of testing, a single machine might not reach your load test's throughput goal. Or it could become overloaded, skewing your testing results making them appear inaccurate.

The solution is to scale the load tests horizontally, distributing the test execution across several machines - the load generators. This is the same strategy as running a test from multiple geographic locations simultaneously. Multiple machines - from different or same locations - will be synchronized to scale the test.



The main challenges of a scalable load testing architecture are server synchronization and database bottlenecks. Running a test on a single machine may be relatively easy but coordinating several distributed servers to execute the load test is not a trivial task.

Optionally, your team might need to run various tests concurrently, which requires additional implementation. Even today - when cloud providers have improved their services and APIs significantly - there are many challenges when implementing an architecture for running concurrent distributed tests.

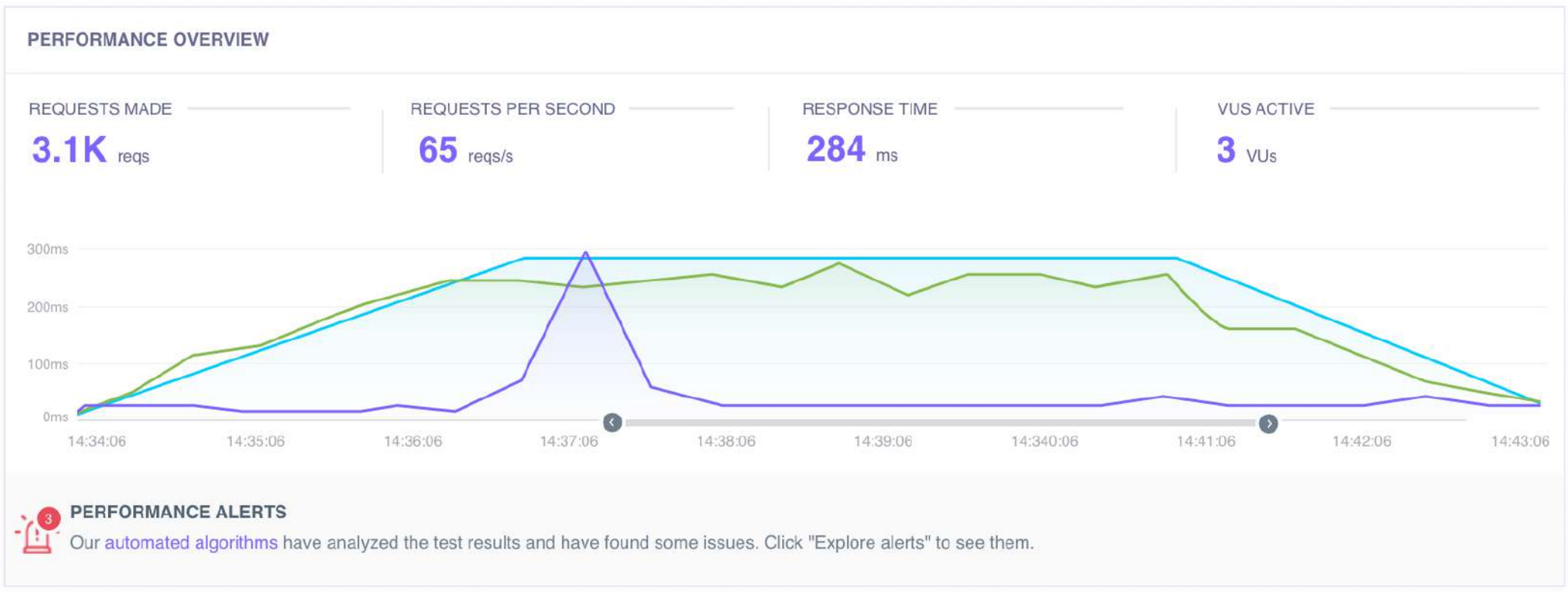
Also, a large load test will likely produce a massive amount of data points. On the storage side, the database will likely become another bottleneck- even the best-performing time-series database can struggle to store the results of a large load test. In this case, your team must choose a high-performing database, optimizing for this use case, and typically aggregating the data before storing it.

Testing continuously

While some businesses might need testing for a particular event as a one-time activity, product teams want to ensure reliability as the application evolves.

Reliability metrics are defined in service level agreements and testing continuously is the most effective process to avoid breaching service level objectives (SLOs). The solution should help you to test regularly and adapt to your industry's best practices.

The testing solution needs to integrate with CI/CD vendors as well as automation and scheduler tools. This is so you can easily configure the testing frequency and conditions defined in your testing strategy. When a meaningful reliability issue happens, the solution should alert your team to act quickly before the problem impacts your users.

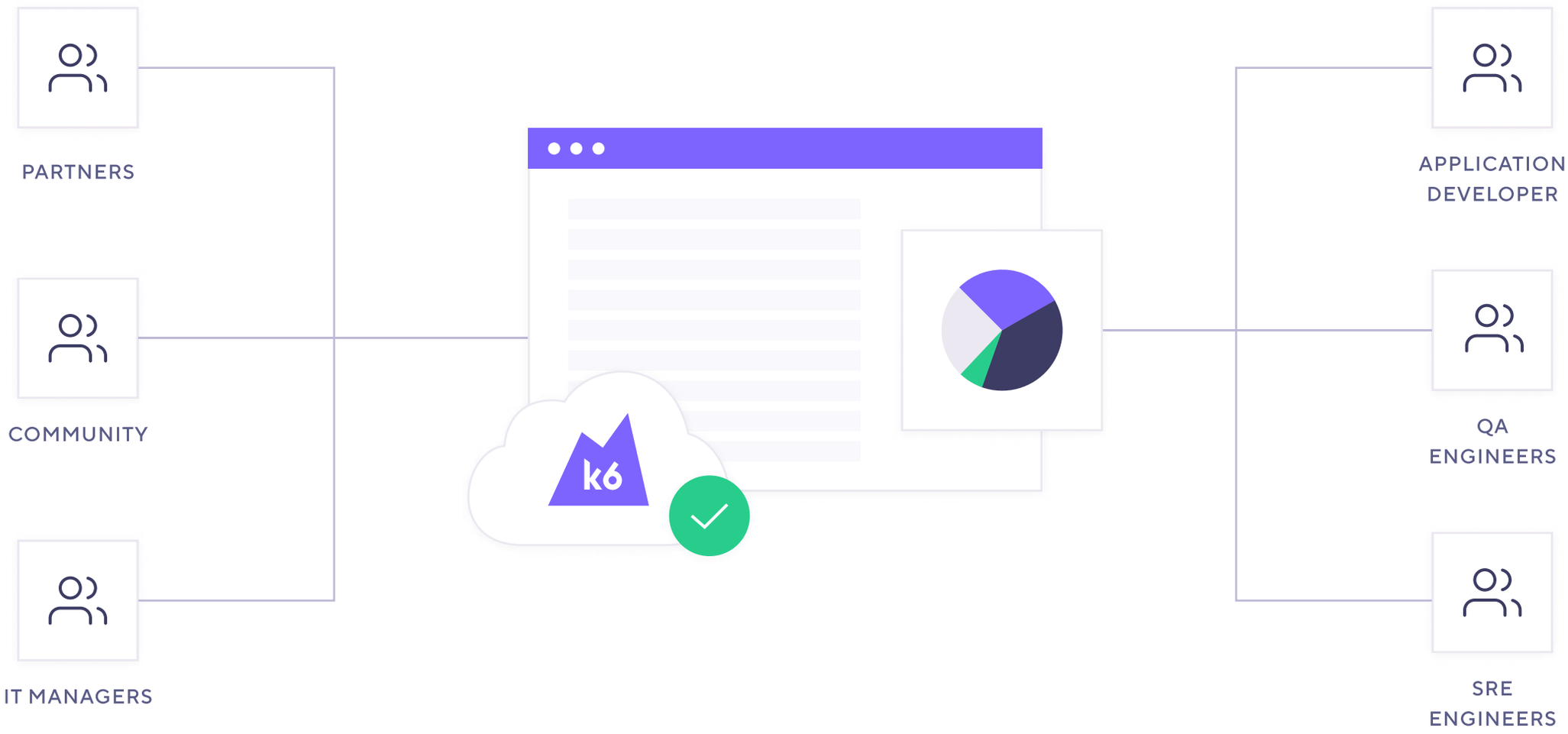


When you start testing frequently, having the ability to compare changes or interpret any trend is also important in helping your team make informed decisions faster. When building your own load testing solution, these abilities are often forgotten. Most open-source load testing tools cannot compare different tests or analyze trends over time.

Collaborating across teams

Today, testing is no longer an isolated QA activity. Today, delivering a first-class user experience is a team effort across the entire engineering organization. Your testing solution is the central point for everyone participating in the testing. And to build confidence in the testing, the solution must accommodate the different roles in your organization.

For example, your management team likely requires reports and a holistic view of your testing, and engineers will need to drill into the data. Developers usually demand a powerful scripting language, while non-coders want to launch faster tests using intuitive UI.



Organizations often underestimate how a solution needs to grow as more roles start participating in testing processes.

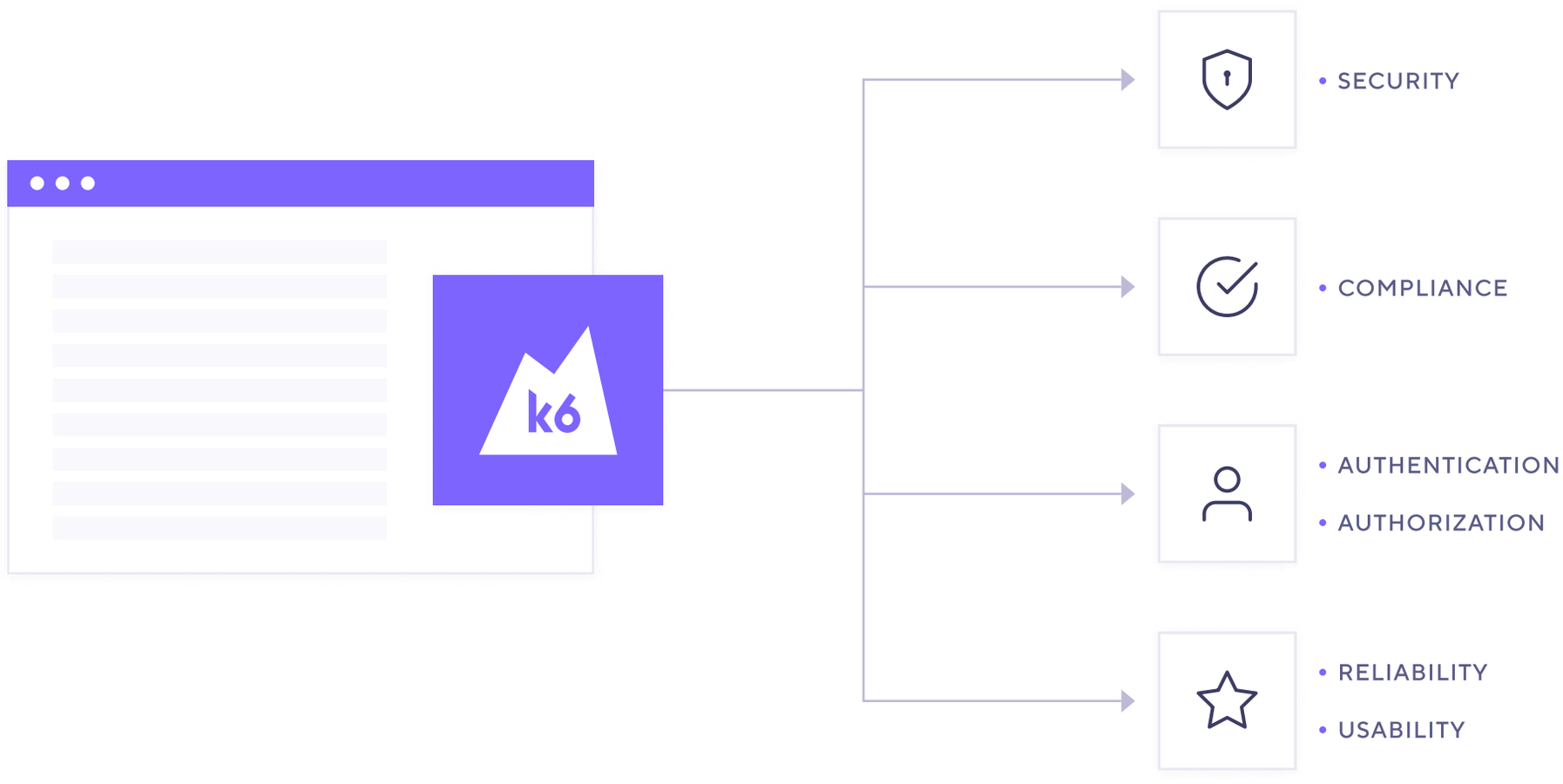
Making your load testing solution reliable

Reliability and compliance are also aspects to consider for your load testing solution. Depending on the type of testing you need to do or company requirements, you may need an enterprise-grade solution that provides:

- 99.9% availability.
- Fast response times, in the milliseconds range.
- Comprehensive and delightful user experience.
- N+1 infrastructure redundancy.

And don’t forget about compliance requirements:

- Auditing who and when someone triggers a change.
- Policies to grant access to project members.
- Security policies for data encryption, retention, etc.



Assessing the Total Cost of Ownership

At k6, we've dealt with all the challenges highlighted in this white paper- from finding approaches for scalable and reliable testing to automating testing processes and meeting compliance requirements.

When software teams discover new and exciting open-source technologies, they often decide to build an open-source stack. However, we frequently find these software teams later switching to our SaaS solution after underestimating the requirements of their custom load testing solution and the total costs of managing it.

When planning the infrastructure and tools to support your solution, we suggest that you clearly define your medium-term requirements first. Next, you should consider the TCO and all the organizational impacts surrounding your choices:

- Building engineering efforts.
- Software and infrastructure maintenance.
- Infrastructure expense.
- Business disruption and lost productivity.
- Expertise and support.

In general, we recommend an in-house solution for organizations with basic and occasional load testing needs or large engineering resources. You should consider an SaaS option when you need a scalable team solution for frequent testing.

Go beyond basic load testing with k6 Cloud

For more than 20 years we've consulted businesses about load testing and we've spent the past 12 years developing cutting-edge load and performance testing tools. We provide an **open-source load testing tool** for developers. We also provide an enterprise-grade performance testing solution in the form of **k6 Cloud**.

k6 Cloud brings industry best practices to the forefront allowing you to get load and performance testing on the right track. Our easy-to-use platform accelerates the testing process, so you spend less time managing your testing infrastructure and more time building reliable and well-performing applications.

Our k6 Cloud platform offers:

- **Scalability** - We've made k6 Cloud scalable by peak running hundreds of machines to process the metric data volume.
- **Automation** - We've built k6 with native support to automate your performance testing. When testing continuously, k6 adds valuable features like trending, test comparison, and notifications on threshold failures.
- **Collaboration** - Your teams can collaborate more efficiently with k6 Cloud. Our platform includes key collaboration features, such as role-based access control (RBAC), test concurrency, advanced sharing capabilities, and dashboards.
- **Rich insights** - Our analytics UIs help you better understand your testing and run automatic analysis to quickly detect performance issues.
- **Integrations** - k6 Cloud provides additional integrations with Application Performance Monitoring (APM) solutions for further test visualization, data correlation, and incident response.

k6 Cloud is used by companies large and small- among our 6,000+ customers are Amazon, Carvana, Grafana, Microsoft, Sephora, and Olo. And our tools are used by engineering teams, developers, QA and SRE engineers.