Compatibility Testing: How to Validate Releases in Customer-representative Environments

A Replicated e-Book
First published: December 2023

Note: this e-Book is a compilation of related blogs previously published on the Replicated website, now presented together here for your convenience. The blogs have been lightly updated and edited for continuity.

Table of Contents

Workloads 3

4 Dimensions of K8s App Compatibility Testing 4

4 Levels of K8s App Compatibility Testing 5

Provisioning Environments for K8s App Compatibility Testing 5

ISV Testing Strategies for Reliability & Compatibility 6

Summary of Common Release Test Stages 7

Smoke Test 7

Policy Test 7

Release Test 8

Canary Test 8

Testing Releases in Customer-representative Environments 9

The Compatibility Testing Maturity Curve 10

The Replicated Compatibility Matrix 10

Provision Resources Quickly for Testing Development, and Support 11

Testing Customer-representative Environments before Releases 11

The Replicated Effect from our Compatibility Matrix 11

How to Add Continuous Release Testing to Continuous Integration 12

Common Types of release tests and how they fit in CI methodologies 12

How Replicated can help automate release compatibility testing into your CI processes 13

Learn More 14

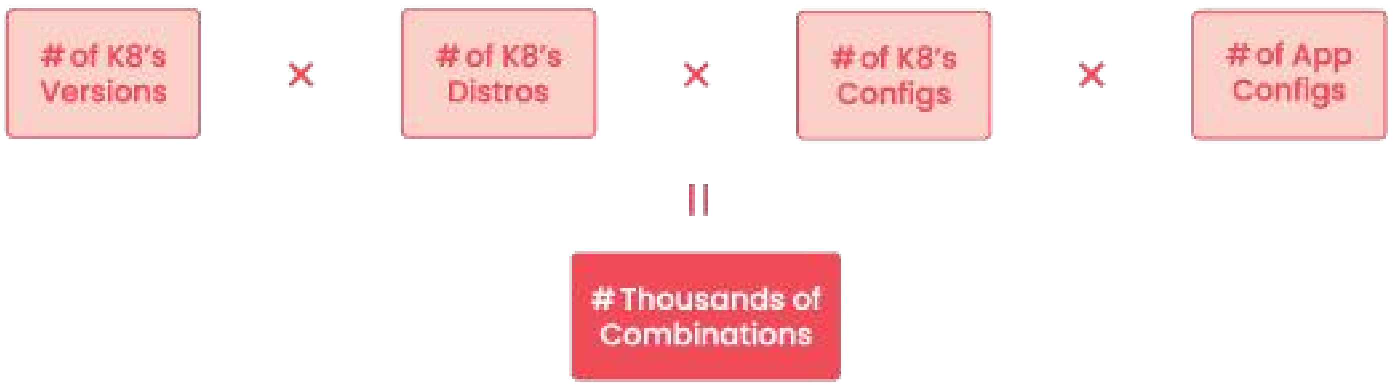
All K8s are not the same: How ISVs test compatibility

When distributing a Helm chart to hundreds of enterprise customers, it's important for ISVs to ensure compatibility with a range of Kubernetes versions, distributions, configurations, add-ons, and entitlements. The ultimate goal is to provide for reliable installation, upgrades and operations of your applications in K8s environments representative of what your customers are actually using. This will improve key metrics like time-to-live, install success rate, number of support calls, and ultimately improve CSAT and NPS too.

But there are some inherent complexities which tend to trip up ISVs when testing compatibility to improve reliability and stability. These often arise when certain combinations of components are running in a particular customer environment. For example:

- OpenShift will not run root or privileged containers (among other things).
- EKS uses a default storageClass named gp2 (but it is default).
- LoadBalancer service controllers are generally only available in cloud providers, and are not always available in bare metal ones.
- K8s 1.28 has a deprecated X, Y and Z that were in 1.23 and earlier

Unfortunately, it's not reasonable to expect that an application tested in GKE will run properly in all configurations of Tanzu. Many successful ISVs implement a comprehensive testing strategy to combine a matrix of variables to validate their updates against a spectrum of possible environments. The most successful ISVs mirror customer environments as closely as possible to create customer representative environments for testing before releasing to those customers.



Compatibility testing challenges

4 Dimensions of K8s App Compatibility Testing

There are 4 dimensions to consider when completing compatibility testing for your K8s app.

But there are some inherent complexities which tend to trip up ISVs when testing compatibility to improve reliability and stability. These often arise when certain combinations of components are running in a particular customer environment. For example:

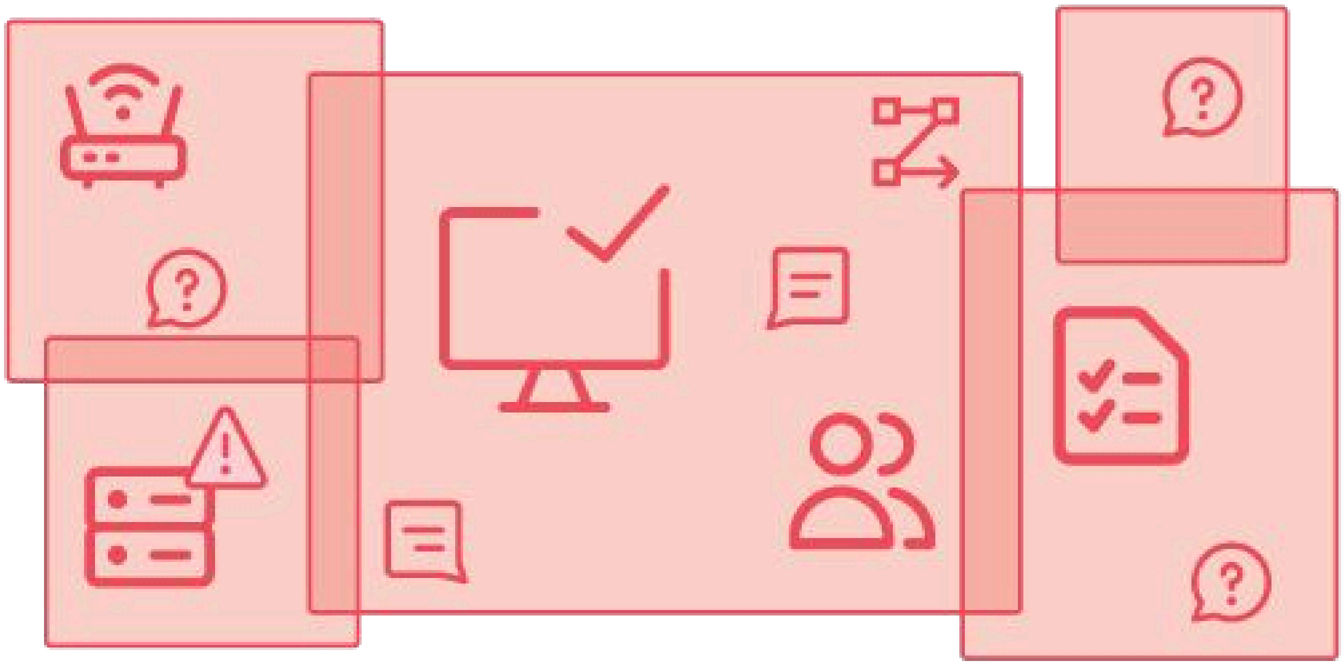
1. **Kubernetes Versions**: Test on the latest patch version of the most recent six minor Kubernetes versions (e.g., 1.25.x, 1.24.x, 1.23.x, 1.22.x, 1.21.x, 1.20.x). This ensures coverage for most users and accounts for the Kubernetes project's support policy and will highlight deprecated APIs that could cause challenges.

2. **Kubernetes Distributions:** Test on popular Kubernetes distributions, as these are widely used in enterprise environments. Some of the most common distributions include:

    - Google Kubernetes Engine (GKE)
    - Amazon Elastic Kubernetes Service (EKS)
    - Microsoft Azure Kubernetes Service (AKS)
    - Red Hat OpenShift
    - VMware Tanzu Kubernetes Grid (TKG)Rancher Kubernetes Engine (RKE)

3. **K8s Configurations:** With a variety of add-ons and common interfaces, it is important to test each distro/version with a set of likely configurations that customers can bring for container runtime, networking (CNI) and ingress, storage (CSI) and types, etc.

4. **App Configuration:** Most ISVs have some high level application configuration that will perform differently and requires compatibility testing with each of these permutations. This might also include individual feature entitlements and different release channels like alpha, beta, stable, and latest.



Compatibility testing challenges

4 Levels of K8s App Compatibility Testing

This complexity can quickly become unwieldy and requires a strategy for successful compatibility testing. The first challenge is actually making sure these versions are available to your team on demand for testing. From there, ISVs must determine the right level of testing at the different points of the SDLC. This might include:

- Smoke tests
- Policy tests
- Release tests
- Canary tests

There are exponential numbers of combinations possible, but it's usually possible to make educated guesses about common patterns, and validate according to the expected sensitivities of each combination and level of testing.

Provisioning Environments for K8s App Compatibility Testing

Of course, simply deciding on what combinations to test at what levels doesn't solve everything. You still need to provision an environment to actually do the testing. Today this is often a very manual, labor-intensive process involving cloud services, K8s distros, add-ons, platforms, projects, and resources. We often hear vendors say that this becomes too time-consuming to manage, particularly if they have a high frequency release cadence like monthly, weekly, daily, or even more often. While there are test automation suites and tools for the code, there isn't always an easy answer for provisioning the variety of environments needed.

ISV Testing Strategies for Reliability & Compatibility

For independent software vendors (ISVs), validating the reliability and compatibility of a software release is crucial before it is distributed to end-customers. Implementing a comprehensive testing strategy is vital to identify and rectify any potential issues prior to deployment. In this post, we'll explore the stages of release testing and the role they play in improving the end-customer deployment experience.

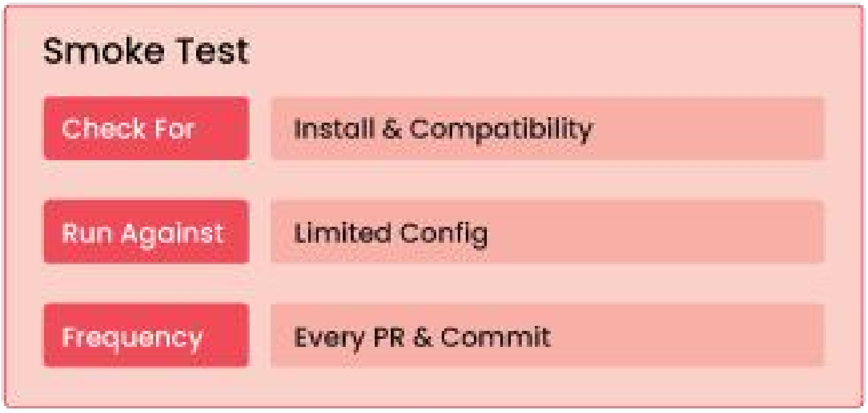
ISVs should always test each release **prior to distributing to end-customers.** Tests cascade from simple and fast checks done at every application code commit, to more time consuming complex tests that are run in preparation for a full customer-facing release.

By breaking down the testing process into different stages, we can optimize what and when to test and reduce the need for executing a full matrix of all possible license values, app config options, Kubernetes config options, Kubernetes versions, and Kubernetes distros.

Summary of Common Release Test Stages

Common test stages include:

Smoke Test

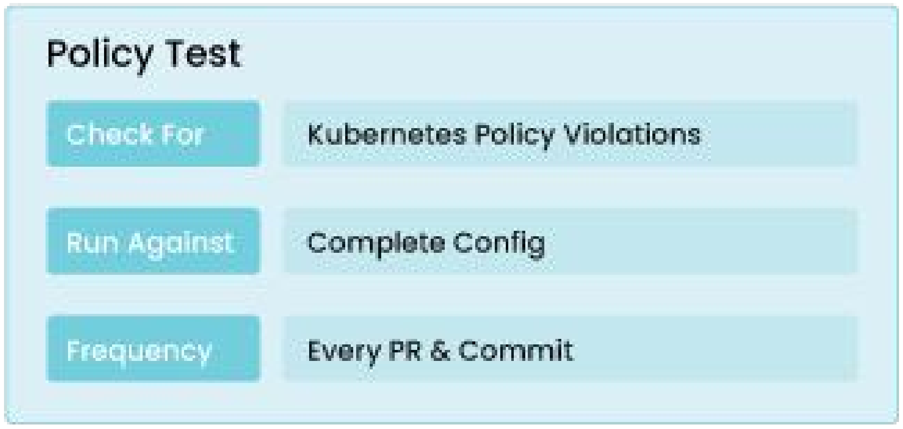


Smoke Test

Check For	Install & Compatibility
Run Against	Limited Config
Frequency	Every PR & Commit

A smoke test is an installation test performed in common environments. The goal here is to ensure that the application can install and start. This stage typically doesn't include end-to-end (E2E) or performance tests for the application. An app release smoke test might be designed to test the Helm Chart rendering, looking for missing or invalid configuration or container images that crashloop on start. Smoke tests should be executed on every PR and commit to a vendor's application repo to validate that the changes don't break compatibility with a basic deployment target and static configurations. A good goal is to complete smoke tests in less than 5 minutes, not counting time to provision an environment.

Policy Test

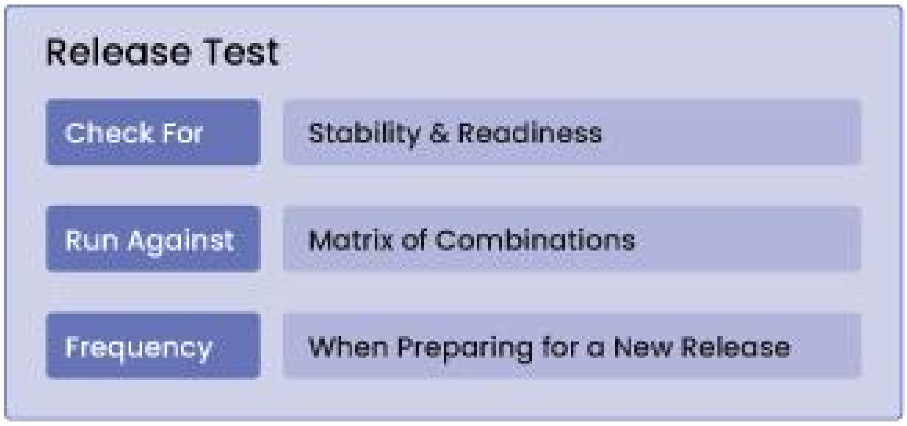


Policy Test

Check For	Kubernetes Policy Violations
Run Against	Complete Config
Frequency	Every PR & Commit

Policy tests should run at the same time as a smoke test against a single K8s distro/version that spins up with default values and will observe the K8s API under a pre-defined number of vendor-supplied integration/end-to-end tests to report on possible K8s policy violations (i.e. app attempts to write to read-only disk). This is where the application E2E test suite should run. Policy violations are detectable only when the application is executing common workloads.

Release Test

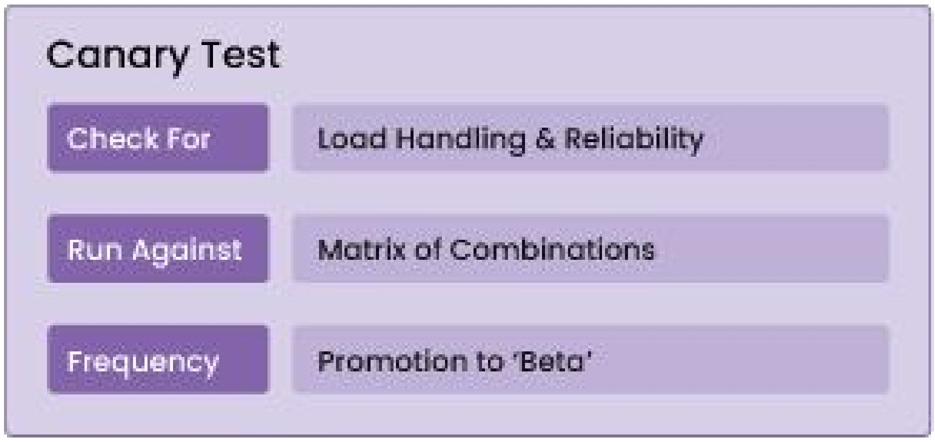


Release Test

Check For	Stability & Readiness
Run Against	Matrix of Combinations
Frequency	When Preparing for a New Release

Release tests should be performed after smoke and policy tests have been passed in preparation for a release. Release tests will consist of a single set of vendor-defined configuration of entitlements/values, and run the app against a matrix of default-configured K8s distros and versions that are currently in support. Vendors can also remove specific distros and versions to reduce the set. The goal is to test all combinations of distros and versions supported by an ISV to set the release up for success in all possible supported environments.

Canary Test



Canary Test

Check For	Load Handling & Reliability
Run Against	Matrix of Combinations
Frequency	Promotion to 'Beta'

Canary tests should be made when a release is promoted to the Beta channel and will be tested against N environments where N = the sum of unique environments that represent an ISV's paying customers' full configuration (entitlements, values, k8s config, distro, K8s version) as reported through telemetry. This could also include starting with the version that a customer is currently using and then performing an upgrade of the application.

Using Replicated's Compatibility Matrix will allow the canary test to change automatically over time an ISV's customers change. If a new customer adds a new untested environment, the test will update to accommodate the new environment. We also recommend that the vendor supply their standard E2E test suite to be executed in addition to other tests you run, like load tests for high load customer-representative environments.

By investing in these testing strategies, ISVs demonstrate their commitment to delivering high-quality, compatible software that meets the needs and expectations of their customers in various environments. Ultimately, a robust testing approach enhances the customer experience by providing them with a stable and reliable software solution that they can trust.

Testing Releases in Customer-representative Environments

A common challenge for independent software vendors (ISVs) is that the wild diversity of customer-managed operating environments makes testing their application very complex and time consuming. They often are faced with an exponential number of combinations of:

- Kubernetes (K8s) distributions
- K8s platforms and cloud managed services
- K8s versions
- Underlying compute and storage infrastructure and instances
- Major and minor application releases and versions

The unfortunate truth is that K8s distros, versions, and platforms are NOT all the same. They can perform differently in different contexts. The differences may be subtle but can cause unexpected issues.

Yet due to the sheer number of possible combinations, the de facto result is that most ISVs don't test nearly enough different customer-representative environments.

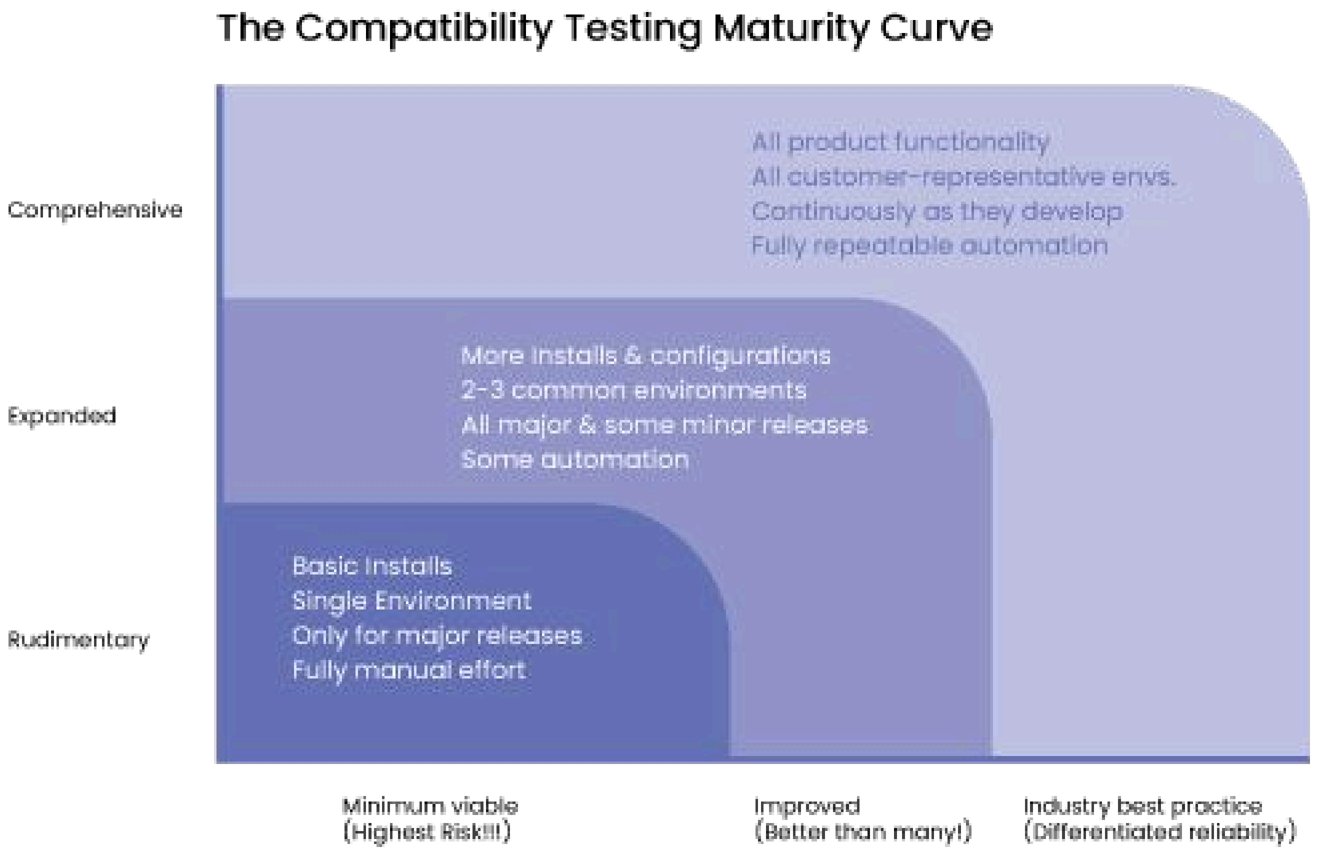
Further complicating matters are the realities of how testing usually gets done, and the common barriers to more testing, which often include:

- The cost of infrastructure for testing
- The time to provision all those environments
- A lack of time to build processes and tests
- A lack of automation for repeatability
- And a lack of understanding of which combinations are the most common patterns or significant anti-patterns in real world customer environments

Due to these challenges, we again note that most ISVs will only test a few combinations, maybe the last three versions of one distro running in one managed service. And they only do those tests around a major version, not for the vast majority of smaller releases nor continuously as they are developing new features.

The Compatibility Testing Maturity Curve

The (not unexpected) impact of this lack of testing is that problems are usually discovered not in the lab, but in live customer accounts. For obvious reasons, this leads to customer frustration, growing dissatisfaction, and all too often, churn. At the same time the vendor will experience extra overhead in managing relationships, while facing an extra burden on support teams and core engineers to try to recreate and resolve unforeseen issues. It can become ugly, but what can vendors do about this situation?


The Compatibility Testing Maturity Curve
Comprehensive
Expanded
Rudimentary
All product functionality
All customer-representative envs.
Continuously as they develop
Fully repeatable automation
More installs & configurations
2-3 common environments
All major & some minor releases
Some automation
Basic installs
Single Environment
Only for major releases
Fully manual effort
Minimum viable (Highest Risk!!!)
Improved (Better than many!)
Industry best practice (Differentiated reliability)

The Compatibility Testing Maturity Curve

To help vendors rate their own testing effectiveness against their peers in the industry, we have created a compatibility testing maturity curve (see diagram above.) This shows that the best vendors are comprehensively testing all their functionality in customer-representative environments, continuously for every new release and multiple recent versions, and that testing is fully automated, scalable, and repeatable. Mid-pack companies test common combinations, with 2-3 distros and versions supported, doing periodic testing with some automation. The bare minimum, where many vendors are today, is testing basic installation with a single distro and version, just before big releases, with a very manual effort. Not good.

The Replicated Compatibility Matrix

We are delighted to announce a new service from Replicated that provides programmatic access to test clusters of various Kubernetes distributions and versions, available in seconds (a few minutes at most) to help vendors provision many combinations of customer-representative environments on-demand.situation?

Provision Resources Quickly for Testing, Development, and Support

With the Compatibility Matrix, your engineers can create test clusters in under 2 minutes using our warm pool of common cloud machine types on Amazon EKS, GKE, AKS and more with per-minute billing. Or they can choose their preferred vCPUS, memory, and storage from VM-based clusters. The provisioned resources will have a default time-to-live (TTL) so clusters expire automatically if not deleted sooner, keeping costs down. Your team can leverage kind and K3s Kubernetes distributions. Red Hat OpenShift is available today, and we will continue to rapidly expand to more clouds and distros, based on our telemetry of what is in highest demand.

Now your team can spin up representative combinations programmatically for testing as they develop, before every release, and when needed to recreate customer environments for support. Then they can run any variety of tests you want to build and automate. We don't explicitly provide for different types of testing, that's up to the vendor to suit their needs, but anyone can define their own smoke tests, policy tests, release tests, and canary tests. There is a wide variety of testing strategies that might be desirable, which are also covered in our blog about ISV testing strategies for reliability and compatibility.

Vendors can now efficiently check releases for compatibility, readiness, stability, whatever they need. They can explore a simple configuration, end-to-end configurations, or a broad matrix of combinations.

Testing Customer-representative Environments before Releases

An added benefit here is that with our Instance Insights, the Replicated vendor portal reports and graphs can actually show which combinations are in real world use by customers. For each of individual customer, Replicated can show the:

- Kubernetes distribution and version
- Cloud provider and region
- Application version
- Installation method and version

Even better, vendors can use this information to programmatically provision truly customer-representative environments, configure them, and set up a series of tests for each release. If the test passes, a script that can even automatically promote the release to your stable or latest release channels and notify customers that is available for upgrade. Maybe even take advantage of semantic versioning and automatic upgrades here!

The Replicated Effect from our Compatibility Matrix

There are a number of business benefits that vendors will enjoy with the Replicated Compatibility Matrix. Comprehensive testing will build confidence with customers, reduce costs and the effort to test, increase install success rates, reduce support incidents in the field, and reduce risk in diverse environments.

There are technical benefits too, including the ability to spin up test environments on-demand (useful for recreating support issues too), identify version incompatibilities, test more comprehensively, automate common tests to reduce effort, and ultimately catch issues before customers find them.

How to Add Continuous Release Testing to Continuous Integration

Delivering high-quality software often requires testing every release for compatibility in varied customer environments. Diverse operating systems, hardware configurations, and software dependencies among customers can lead to unforeseen issues, such as crashes or feature inconsistencies, that might hinder adoption and satisfaction; thorough compatibility testing as part of your continuous integration (CI) processes minimizes these risks, fostering user trust and loyalty while reducing post-release support burdens.

Testing is a key technique in the efficient delivery of high-quality software and tools. From unit testing to end-to-end testing and everything in-between, teams of all sizes rely on different testing techniques to ship often and with high confidence. When building apps that will run in customer-hosted environments, however, testing every software release against every environment it might run in becomes quite difficult to do regularly or in any sort of automated fashion. For this reason, many teams settle for "best effort" testing of their customer hosted software, perhaps testing against only the two most recent editions of Kubernetes, or only regularly testing against their known most-popular K8s distribution (e.g. EKS).

If not tested, customers' diverse operating systems, hardware configurations, and software dependencies can lead to unforeseen issues, crashes or feature inconsistencies. True compatibility testing for customer-hosted Kubernetes applications means embracing completeness (testing all relevant combinations) and frequency (shifting left and testing as early in the development process as possible).

Common types of release tests and how they fit in CI methodologies

Developers often combine continuous integration processes with release testing methodologies like compatibility testing, canary testing, smoke tests, and load handling to ensure the quality and reliability of software releases. In the CI pipeline, code changes are automatically integrated into a shared repository several times a day. As part of this process, automated unit tests and smaller-scale integration tests are executed to catch early-stage bugs and prevent integration issues.

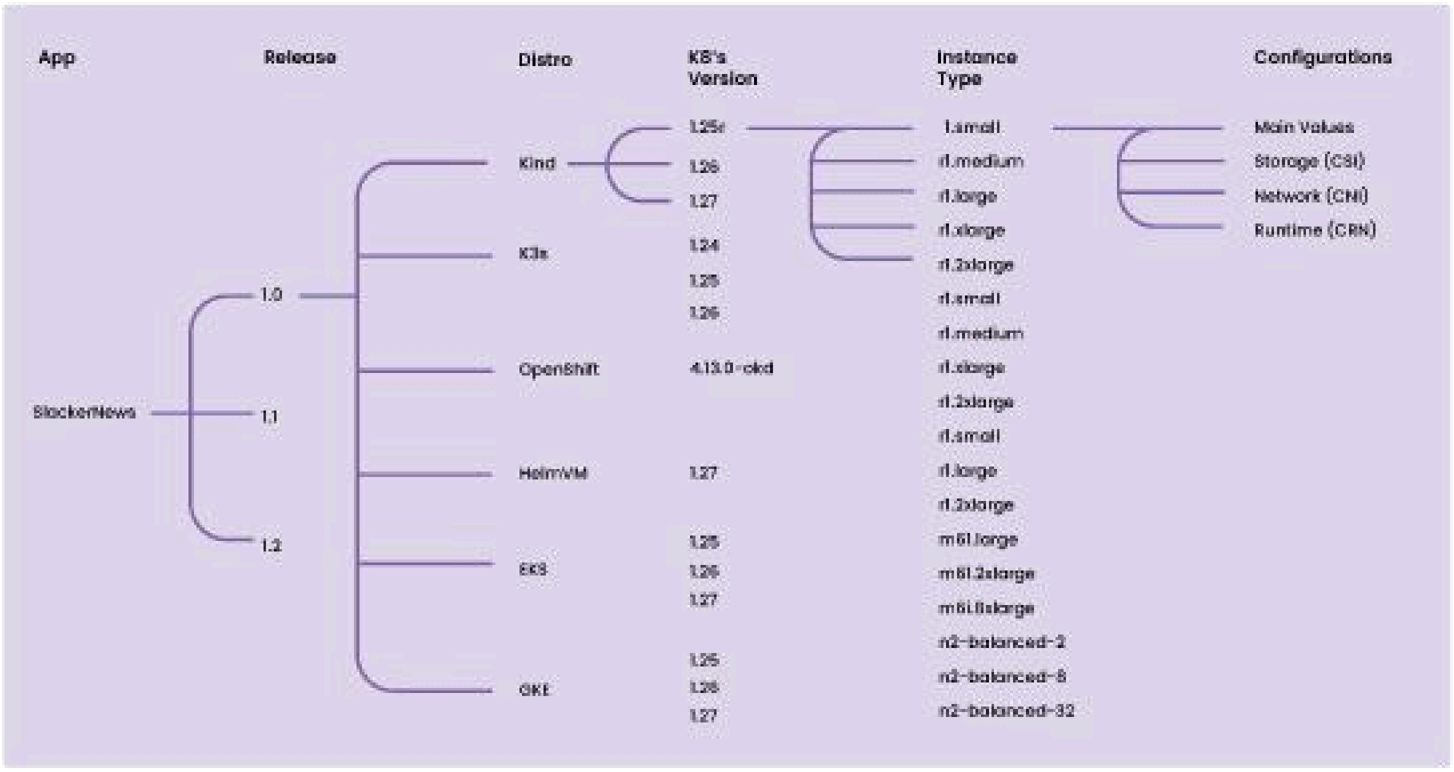
By incorporating these testing methodologies into the CI process, developers can catch bugs and issues early in the development cycle, leading to more stable and reliable software releases. Automated testing at each stage ensures that code changes are continuously validated, reducing the risk of regressions and providing a solid foundation for more extensive manual testing and user acceptance testing before the final release. We've written about this before in the blog on ISV Testing Strategies for Reliability and Compatibility, but now let's take a closer look at HOW we can help specifically around CI processes.

How Replicated can help automate release compatibility testing into your CI processes

The issue is not that developers are uninterested in testing, it's the complexity and effort around identifying customer environments, provisioning exponential combinations, and automating testing that causes people to conduct only limited release testing.

The secret power here is how our new Compatibility Matrix can be… ahem… continuously integrated with continuous integration. The primary tool here will be the replicated CLI, which in this case enables developers to programmatically request customer-representative combinations of Kubernetes and operating environments for testing with every release. It's worth noting here that we can authoritatively identify the customer-representative combinations from our Instance Insights and telemetry notifications.

So once you've identified common combinations and provisioned relevant resources, then it's a very small leap to run the tests you've already configured against every possible combination, and do so on every single commit and release (if you so choose.) Our approach is generally adaptable to a wide range of CI tools such as GitHub Actions, Jenkins, CircleCI, TravisCI, and more.



As noted elsewhere, the Compatibility Matrix enables you to create test clusters (often under 2 minutes) for Amazon EKS, GKE, Red Hat OpenShift, kind, K3s, and more K8s environments coming soon. You can spin up representative combinations programmatically (with consolidated per-minute billing!) and a time-to-live (TTL) to shut down resources after testing to minimize your costs.

Note that we don’t provide all the different types of release tests specific to your product, but we empower you to automate them widely for all your releases.

For more information on the specific steps to take, please visit our documentation About Integrating with CI/CD.

Learn more

Interested in getting access to the Compatibility Matrix and giving us feedback? Sign up now to learn more.

You can also check out the Compatibility Matrix docs or watch our short Compatibility Matrix intro video from RepliCon Q2.

Not a Replicated customer, but interested? Schedule a demo with us!