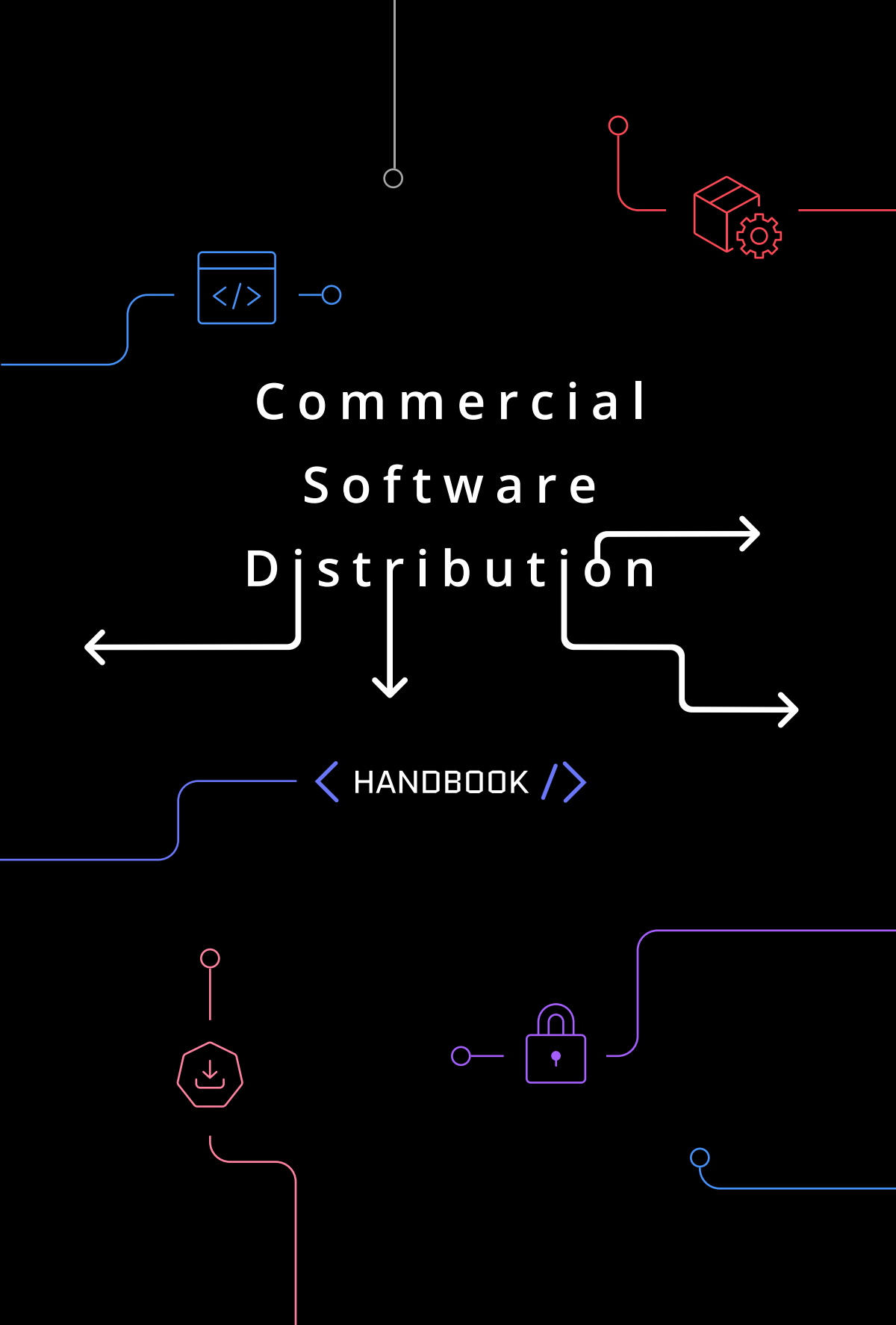

Commercial
Software
Distribution
HANDBOOK

# Table of Contents

# Intro

Commercial software distribution is the business process that independent software vendors (ISVs) use to enable enterprise customers to self-host a fully private instance of the vendor's application in an environment controlled by the customer.

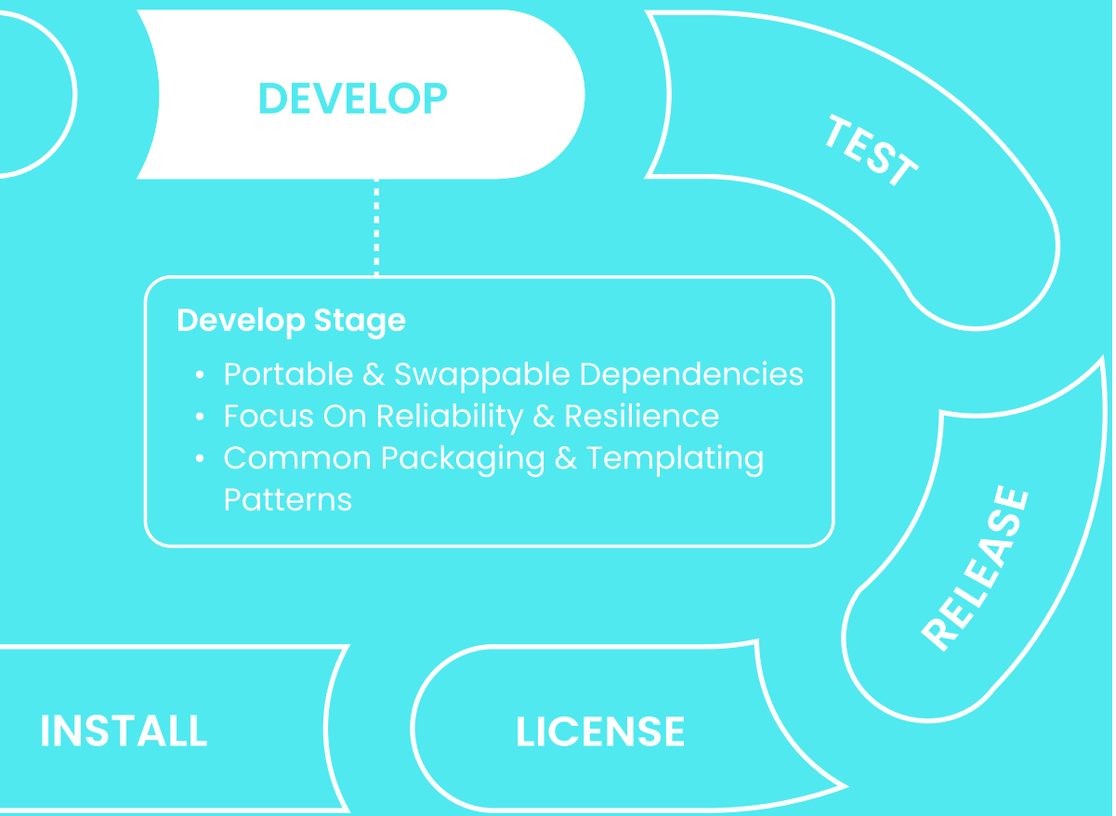
Since its inception, software distribution into self-hosted environments has changed drastically. What was once almost exclusively a process where a Solutions Engineer physically traveled to a customer's office, installed software onto dusty servers in a dark closet, and traveled back to that closet every time an upgrade was needed, is now replaced with a world of VMs, containers and private clouds. Yet, the goals for Software Distribution remain the same: deploy software, quickly and securely, into environments where the customer has complete visibility and control.

At Replicated, we've been enabling self-hosted software for nearly a decade and have worked with hundreds of software vendors as they implement a modern approach to this problem. Over this time, we've developed a unique expertise in recognizing and implementing the key steps to distributing software well. Now, we are sharing what we've learned with you and empowering every software vendor to transform how their software is distributed.

To this end, we've developed the Commercial Software Distribution Lifecycle. Developed through years of trial and error, thousands of conversations with early-stage startups and Fortune 500 companies, and pulling from the expertise of our staff, the Commercial Software Distribution Lifecycle represents the stages that are essential for every company that wants to deliver their software securely and reliably to customer controlled environments.

This lifecycle was inspired by the DevOps lifecycle and the SDLC, but it focuses on the unique things that must be done to successfully distribute third party, commercial software to tens, hundreds, or thousands of enterprise customers. The phases are:

# Commercial Software Distribution Lifecycle

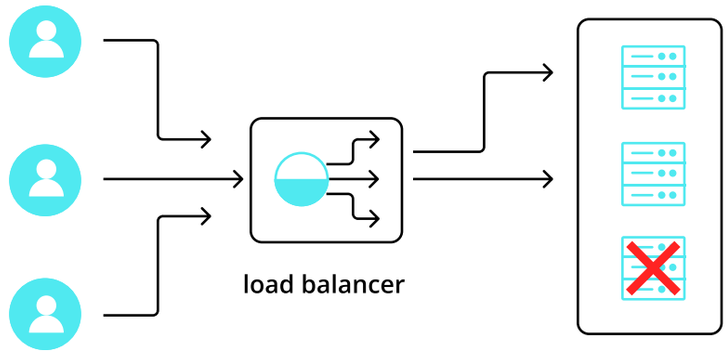



## Develop

Develop refers to the technical decisions made by software vendors to prepare software to be consumed by enterprise customers in on-prem environments. To provide a seamless experience for customers, software vendors must consider how an application will be distributed while developing it—not after the fact. This includes considerations during the design, architecture, and packaging of the application. When done well, customers can install the application when and where they want to without any major changes to the application's architecture.

An application is developed properly for modern on-prem deployments when it addresses the following:

- The application is resilient. It comes back on failures and is also built to avoid externally-facing failures. For example, leveraging cloud-native Kubernetes best practices for packaging will provide you with a dependable solution for resilience and high availability.

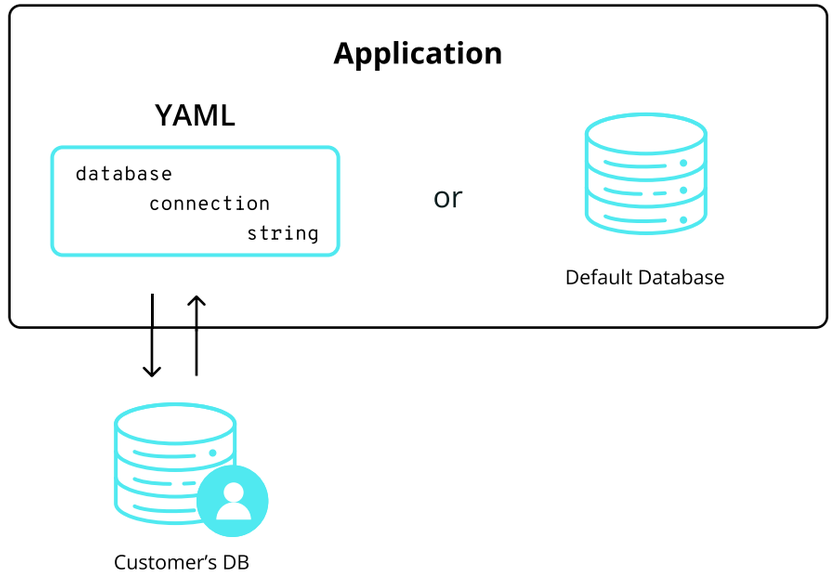


- The application uses components that are portable, so vendors and customers are not locked in to specific cloud providers or services. This makes the application more self-contained and portable. For example, using an open-source queuing solution like RabbitMQL as opposed to AWS's SQS will provide more portability.

- The application is packaged with open standards whenever possible, rather than providing custom scripts or bespoke installers. For example, most enterprise customers are familiar with Helm because it provides a consistent, reusable, and shareable packaging format. For customers who aren't familiar with Helm or Kubernetes, vendors can provide alternative installation options that treat Kubernetes as a dependency without exposing it to the customer.

- The application follows the principle of least privilege to minimize the privileges required from the enterprise customer's cluster, including avoiding tools that require escalated permissions. For example, not expecting to be able to deploy operators into existing clusters as RBAC may not allow it.

- The application allows customers the choice of supplying required services as part of the installation, as opposed to utilizing whatever services are embedded with the application. For example, enterprise customers should be able to provide a connection string to their own database rather than using the embedded database option.

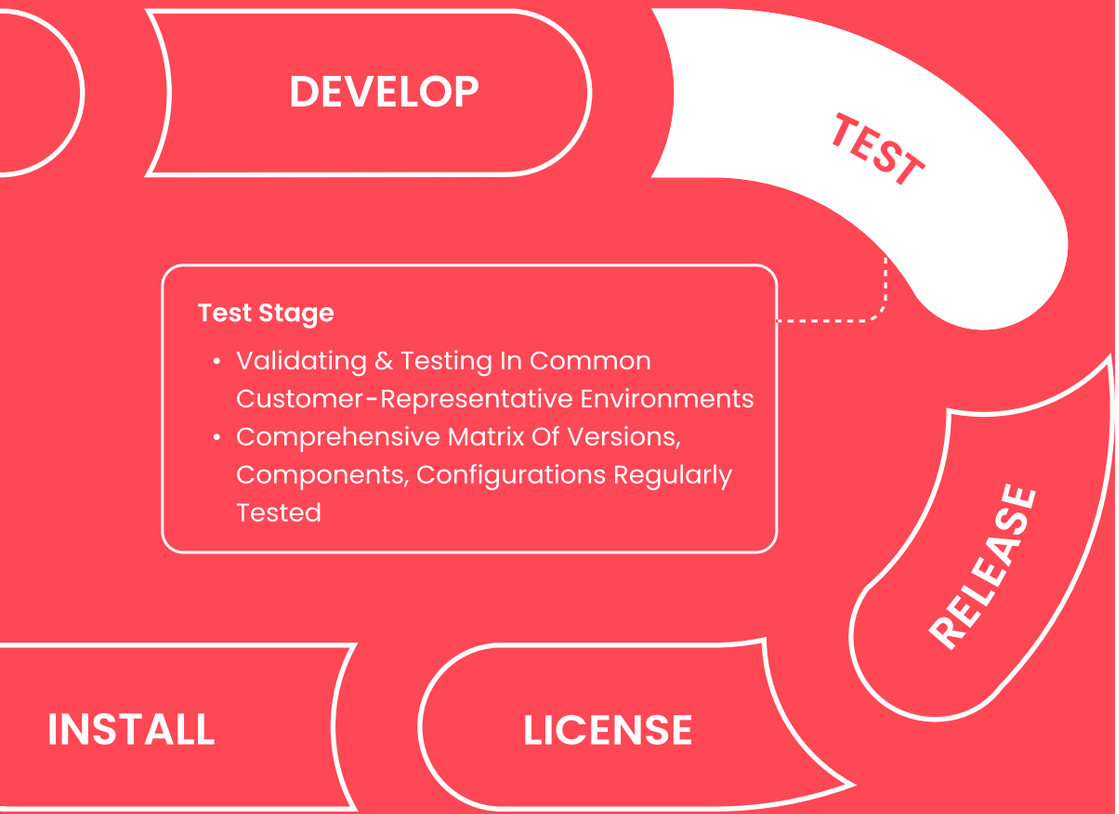



- The application is developed and tested with an understanding of the different network settings that enterprises will require. For most applications, the vast majority of enterprise customers are not going to give inbound access to their instance of the application. Others will allow outbound access, either directly or through a proxy. Finally, the most security conscious customers will not provide any outbound access to the internet, also known as an air gap environment. These network settings can complicate installation, updates, reporting, and application operations if not thoroughly tested.

When developed well, these considerations allow the enterprise customer to install the software where and how they want to. They aren't limited by the application's architecture and are able to bring their own requirements and tooling when needed. In this way, the application provides the right amount of structure and flexibility for the enterprise customer to be successful, whether this is the first self-hosted application they are deploying or the 100th.

## Test

# Commercial Software Distribution Lifecycle




Testing refers to ensuring that enterprise software can be successfully and reliably distributed to current and future customer-managed environments. Ideally, testing happens before the application ever gets into the hands of customers. Different customer environments act differently, and the same application successfully installed in one customer environment could present challenges in the next. For this reason, testing self-hosted software needs to go beyond unit and integration tests and also focus on the environments where the application will be deployed.

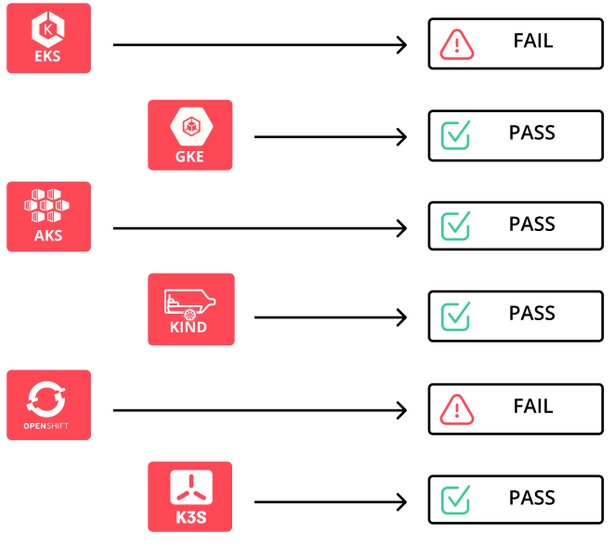
Software vendors should consider the following when creating a testing stack for modern on-prem software distribution:

- Expect different customer environments to act differently. For example, don't anticipate a deployment into an OpenShift environment to require the same steps as a deployment into AKS. Each customer-representative environment might require different elements to be tested.
- Test that different configurations of the application deploy successfully in the same environment. For example, test different database configurations or different core services.

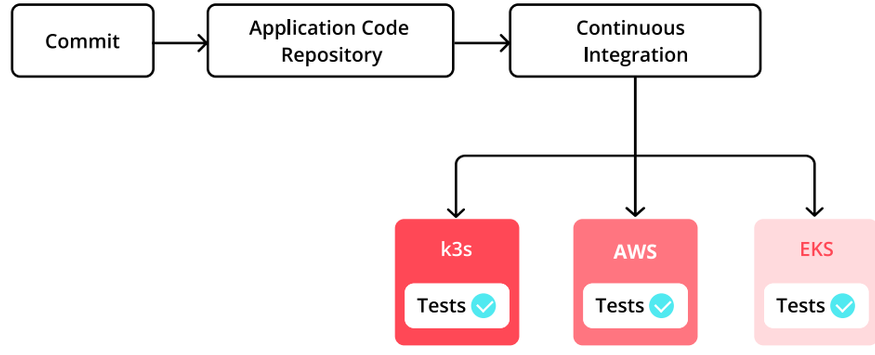
- Ensure that testing environments are representative of current customer environments. For example, if different customers are using AKS and EKS, then tests should validate both those distributions.

- Test for environments that are likely to be supported in the future, especially if those environments have additional security requirements or complexity. For example, test highly complex environments such as OpenShift before ever signing a customer that uses OpenShift.

- Add tests to continuous integration (CI) pipelines. This ensures that tests run automatically before each new release without requiring manual intervention from the team. Logic can also be added to CI pipelines to prevent releases from being shipped unless all tests pass.




Self-hosted software presents a unique challenge, in that both the application and the customers' environments could be the cause for a failed deployment. Tests should verify that the software functions correctly, regardless of the environment where it's going to be deployed. In doing so, vendors will be on the path to satisfy internal teams and create a positive customer experience where every deployment is successful on the very first attempt.



# Commercial Software Distribution Lifecycle


## Release

Releasing refers to the process of delivering software to licensed users, ensuring that new features, improvements, and bug fixes get into the hands of the right customers at the right frequency.

Key considerations for vendors when releasing modern enterprise software include:

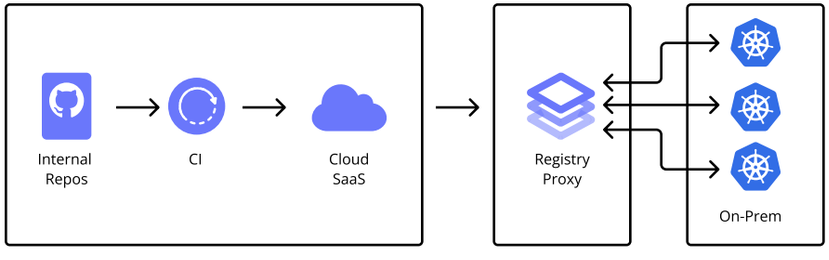
- Making application images available for customers to access securely
- Packaging and publishing release artifacts in multiple formats to support different installation methods and customer environments
- Managing release streams for different customers, including automating workflows for production (GA) and pre-release (alpha, beta) versions
- Versioning releases with a consistent pattern so that it is easy for customers to understand backward compatibility

The following sections explain each of these considerations in greater detail.

## Securely Accessing Images

A single release for an application contains all the artifacts required to install and run the application, such as container images or executables. When publishing a release for distribution to on-prem environments, software vendors need to make images available to customers securely.

For online (internet-connected) environments, proxy servers can be used to grant proxy, or pull-through, access to images. Proxies work as an intermediary between the software vendor's private image registry and the enterprise customer, allowing users to access images on the vendor's private registry without exposing registry credentials. With a proxy, customers can access images using credentials determined by the software vendor, such as providing their unique license or customer ID.




For air gap environments, customers must have access to downloadable archives that contain the release images so they can push images to their own registry.

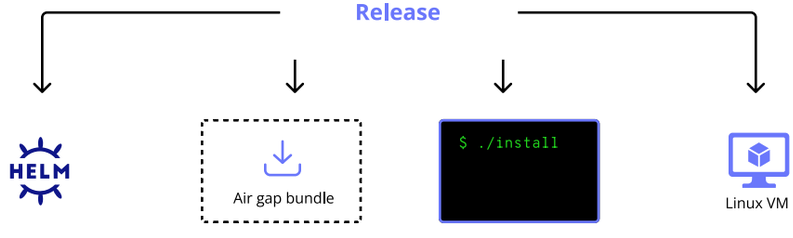
As discussed later in the *License* and *Report* phases, all of this activity can be tracked for auditing and reporting.

## Package and Publish Release Artifacts

Releases should be made available in multiple formats to support different installation methods and customer environments. For example, while some enterprise customers with Kubernetes expertise will prefer to install in their own cluster, others will prefer to install on a virtual machine (VM) or bare metal server. Additionally, enterprises with an emphasis on security might need to deploy software in air gap environments with no outbound internet access.

Allow enterprise customers to choose the release asset they need based on their unique preferences and requirements. For example, a vendor might need to publish all of the following for a single release:

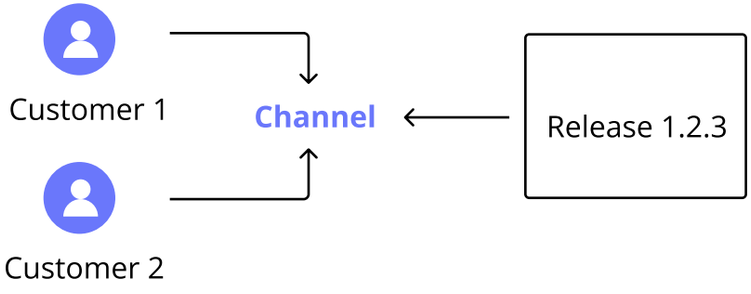
- Raw artifacts, such as Helm charts or containers
- Downloadable archives that contain the release images for air gap installations
- Installation scripts, such as scripts that install the application in Kubernetes clusters or on VMs
- Release assets that are specific to the operating system, such as unique assets for installations in Linux or Windows environments



## Release Management

Release management is also important for ensuring that each release is made available to the right subset of users (including internal teams and customers), and that the vendor has control over the frequency that new releases are published. A successful release management practice achieves these goals with minimal maintenance burden for the vendor.

One common release management strategy is publishing releases to different channels or lanes. For example, vendors might keep separate channels for internal-only, experimental, beta, and generally available (GA) releases. Enterprise customers and internal users can then access the releases published to the channel where they are subscribed.




Channels can be useful as a release management tool because they allow vendors to create a logical separation between different types of releases, including those releases intended only for internal development and testing, without having to manually grant and restrict access to features or risk prematurely releasing new code. Channels also provide flexibility in release frequency, allowing vendors to more quickly publish updates to internal or pre-release channels while maintaining a different pace for GA releases.

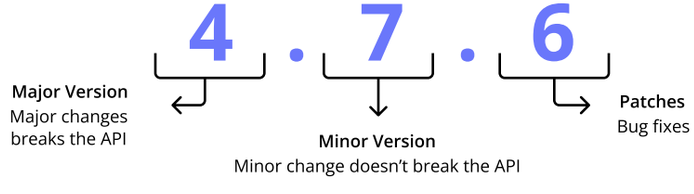
To minimize the need for manual intervention when releasing new versions, continuous integration and continuous delivery (CI/CD) pipelines should include workflows that automate release management and publishing. For example, vendors could create Github workflows that run tests, publish releases to the right channel, and notify customers subscribed to the channel that a new version is available.

## Release Versioning

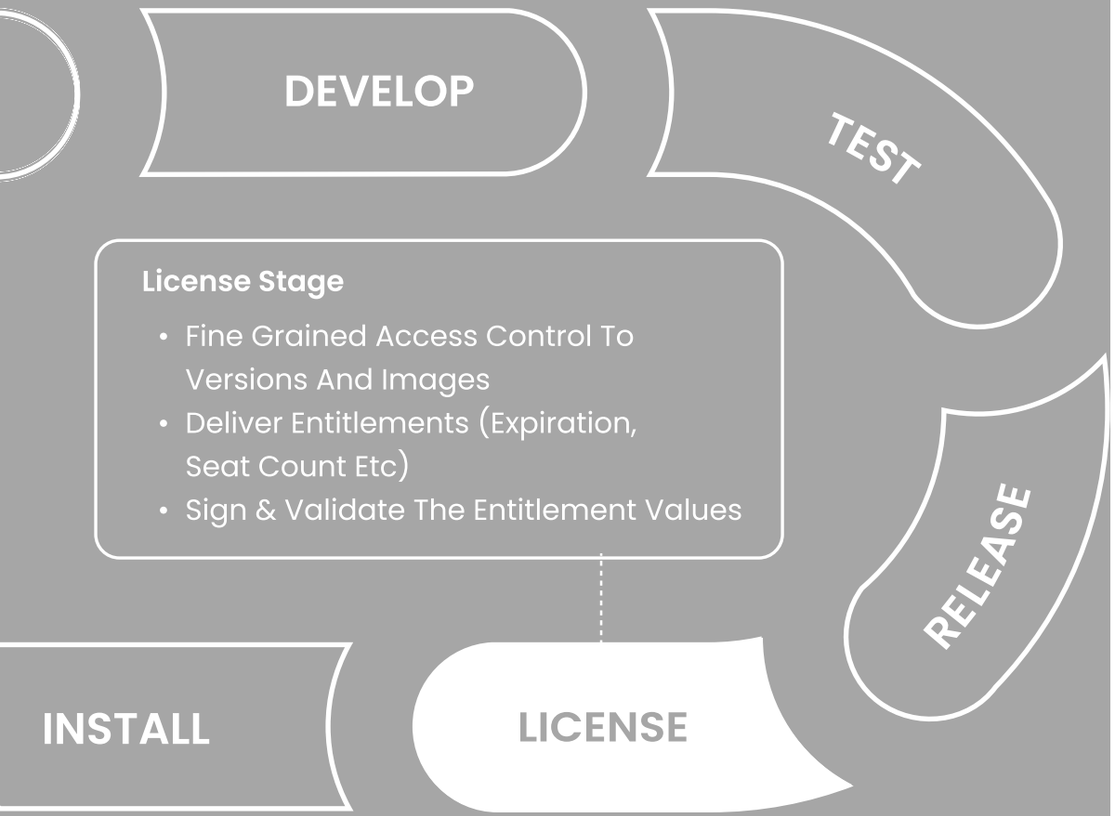
Software vendors should assign and increment version numbers for releases using a consistent pattern, such as Semantic Versioning (SemVer). SemVer is a commonly-used and recommended versioning strategy that provides implicit information about the backwards compatibility of each version, using the format MAJOR.MINOR.PATCH.

The release versioning pattern used should also dictate how build metadata and pre-release versions are indicated. For example, with SemVer, alpha or beta versions are denoted by appending a hyphen followed by the pre-release label or version number, such as 1.0.0-alpha or 1.2.3-0.0.2.

A consistent versioning pattern such as SemVer is important for modern commercial software distribution because it is common for vendors to support (and continue to release patches on) multiple different versions of their software concurrently. SemVer enables this because enterprise customers can easily understand that a new patch release is backwards compatible with the corresponding minor version, without needing to worry about breaking changes.



# Commercial Software Distribution Lifecycle

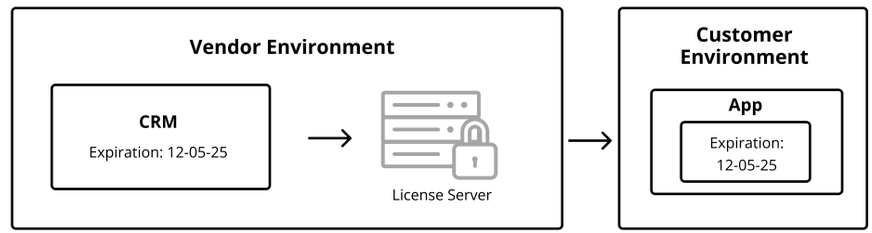



## License

Licensing refers to securely granting access to software. Licensing codifies the agreements defined in the software contract between the vendor and the enterprise customer, and makes those agreements available to the underlying application through a license server during startup or runtime.

Licensing is a cross-functional concern as it is important to many different teams that licenses are easy to create, update and sync to customer instances:

- For Sales teams, the license server that keeps track of users and entitlements should be integrated with internal CRM tools, such as Salesforce. This allows customer entitlements to be easily turned on and off based on changes to the software contract.




- Support teams should be able to use the license as a unique customer identifier to get visibility into insights such as the customer's entitlements and product usage.

- For Engineering teams, it is important that application logic can be used to control access to code and images so that engineers do not need to update code each time a license agreement changes.

## License Entitlements

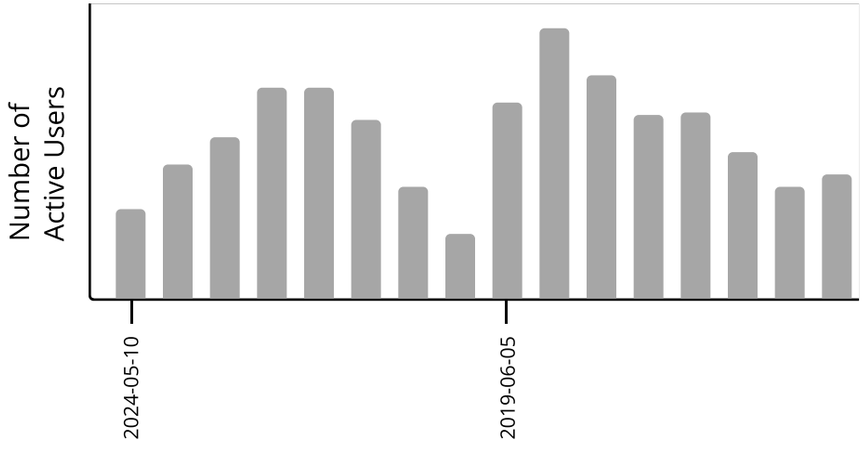
License agreements for enterprise software often include entitlements that address the following common concerns:

- Enforcing expirations of licenses, such as trial or Proof-of-Concept licenses
- Controlling feature-based and usage-based entitlements to facilitate product assortment. For example, license entitlements can determine a customer's access to a feature that is available only under a certain product plan
- Limiting the number of instances of an application that can be run by a single customer
- Other application- or customer-specific entitlements. For example, many AI applications require granular restrictions for model images to control access to sensitive data, and so it is necessary to define which images users can access. Other entitlements might include the number of users permitted, the number of nodes permitted, and so on.

## Measuring and Reporting Usage

In most cases, software vendors can confidently rely on the license agreement or contract to enforce entitlements, as enterprise customers will be wary of violating a software contract. Because of this, it is likely unnecessary to write application code that prevents certain actions or blocking usage. Instead, most software vendors will track or communicate usage that exceeds the contract and "true up" customers at renewal. One exception to this strategy is enforcing expiration dates, which can be easily extended by the vendor as needed to ensure that the enterprise customer can continue using the software.

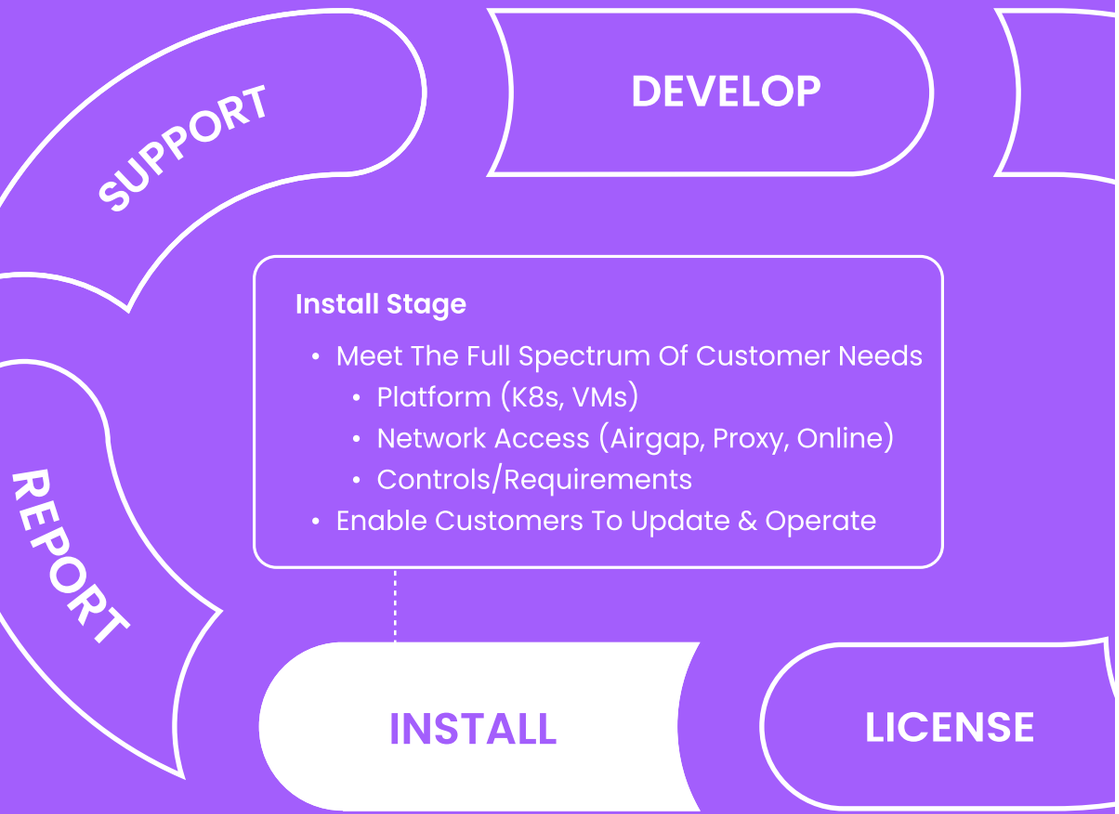
Measuring usage surfaces relevant data to both the vendor and the customer without the negative consequence of reducing or preventing usage of the software. For vendors, it is helpful to know how customers are using the software to identify opportunities to extend or expand the agreement. For customers, understanding their own usage is valuable for avoiding violations of the contract.





License

# Commercial Software Distribution Lifecycle

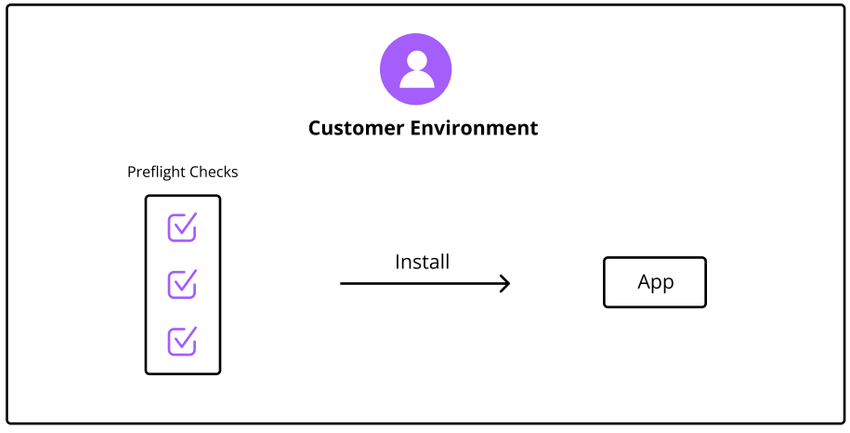



## Install

Installing refers to the steps that enterprise customers need to take to securely deploy software in their environment. For modern on-prem software, the installation process varies depending on the release delivery method and the installation environment (such as online versus air gap installations, or Kubernetes clusters versus VMs or bare metal servers).

To ensure a good installation experience for all customers, vendors should provide detailed installation instructions that explain how each component of the software is configured and installed. This should include information about upgrading or downgrading, proxy installations, advanced configuration options, and any other support installation path. Ultimately, the challenge with installation is that the vendor has to be prepared to meet a spectrum of customer requirements and sophistication. This increases the complexity for vendors who need to consider each installation path in all future releases, testing, updates, support, and so on.

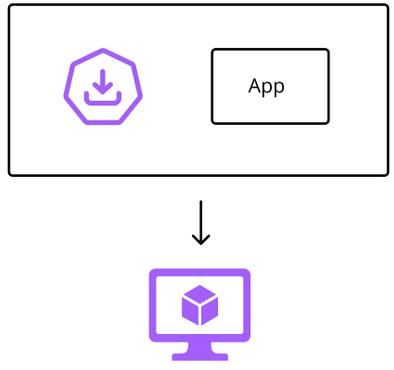

In addition to thorough installation instructions, the best vendors also provide preflight checks that customers can run to validate if the resources provided meet the hardware, network, and environment requirements for the software before proceeding with installation. These types of checks can help increase the success rate of installations and upgrades, reducing customer frustration and speeding up the time-to-value for the application.




Providing an installation GUI can also make it easier for less advanced customers to complete installation tasks, such as providing their license or configuring the deployment, without needing to interact with the command line or edit complex YAML files. This can improve the customer experience and cut down on installation errors, helping to reduce the number of support issues related to installation.

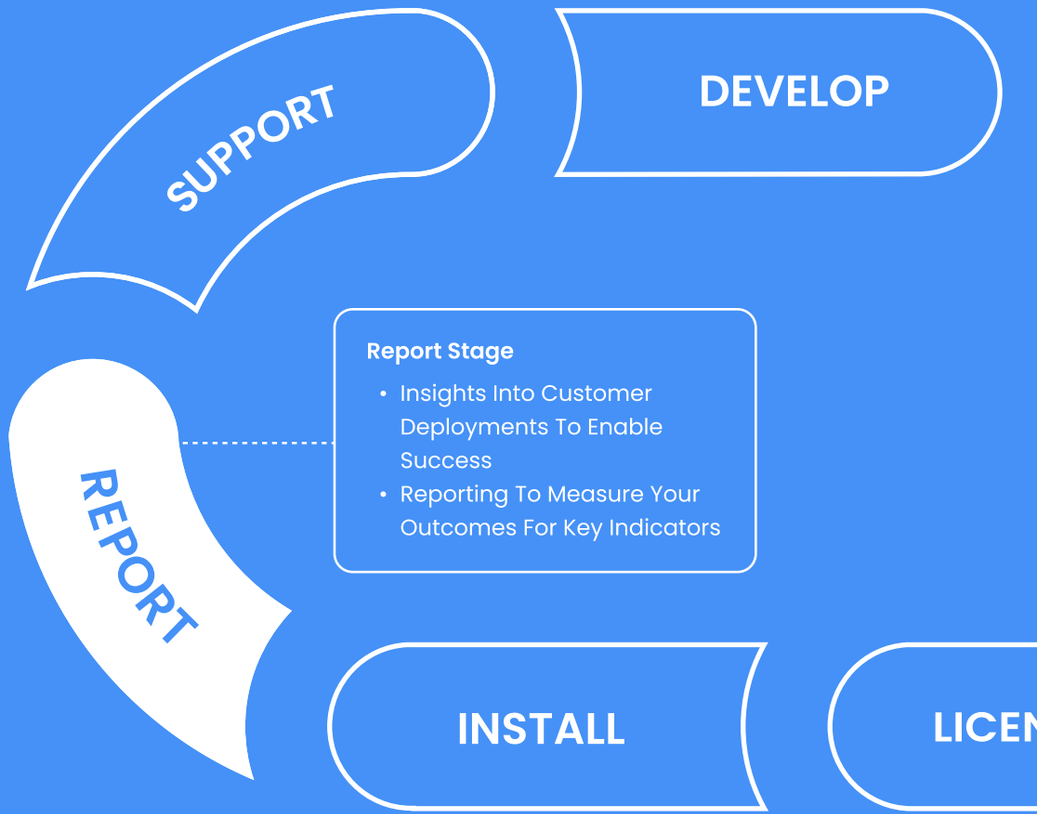
Finally, whenever possible, vendors should utilize existing packaging and installation tooling that is already widely adopted across the industry. For example, Helm is a popular package manager and installer for Kubernetes applications used by many modern enterprises and software vendors. Taking advantage of contemporary industry standards like Helm avoids the overhead of maintaining your own installer, and also ensures that many enterprise customers will already be familiar with the tooling.

That said, many customers might not be proficient enough in contemporary tools (like Kubernetes and Helm) to successfully install. There might also be customers who would prefer to install on a VM or a dedicated Kubernetes cluster rather than attempting to install to an existing shared cluster. To address these use cases, many vendors include a Kubernetes installer that delivers Kubernetes alongside the application so that customers can install on a VM or bare metal server.




In this case, vendors must ensure that any installer artifacts are either packaged with the corresponding application release or are published separately where they can be accessed by customers. This also puts the burden of Kubernetes management on the vendor, as their customer will consider it a dependency of the application as opposed to a core system they manage themselves.

# Commercial Software Distribution Lifecycle



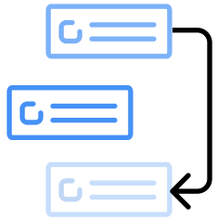

## Report

Reporting refers to gaining visibility into performance and usage metadata for software instances running in customer-controlled environments. For example, many vendors will collect:

- Metadata about the environment where the application is running, such as the Kubernetes distribution, Kubernetes version, or cloud provider
- Application uptime and service status
- Adoption data such as the current application version
- Usage data such as daily or weekly active users of the application


In contrast to traditional observability, which often includes a firehose of logs or key-value pairs from a database, the goal of reporting for modern enterprise software is to provide insight on application usage and functionality. This type of insight-driven reporting is often described with terminology such as telemetry, phone home, or heartbeat.

For vendors, access to reporting data empowers the team to take more informed action:



- Product and Engineering teams can use adoption and usage data to inform prioritization decisions about feature development. For example, low feature usage can indicate the need to invest in usability, discoverability, documentation, or in-product onboarding.

- For Customer Success and Sales teams, decreased or plateaued usage for a customer can indicate a potential churn risk, while increased usage for a customer can indicate the opportunity to invest in growth, co-marketing, and upsell efforts.

- For Support teams, performance data, such as simple uptime data, helps more quickly troubleshoot and resolve issues.

Enterprise customers also often expect access to this reporting data through formats such as dashboards, data exports, reports, or notifications. For example, customers like to see their usage data to make sure they're within their contractual limits. This is especially useful for air gap customers, who will typically self-report when they are over usage limits in order to get back into compliance with the contract.

When reporting on instances of enterprise software, vendors should adhere to principles of transparency, privacy, and customer choice:

- Be transparent with customers about the types of data that will be collected by providing clear documentation

- Redact sensitive data (such as database connection strings, passwords, or other API tokens) from being sent back to the vendor environment

- Give customers the opportunity to opt in or out of sending different types of data. For example, some customers might opt to send nothing, or to send only diagnostic data used for support purposes

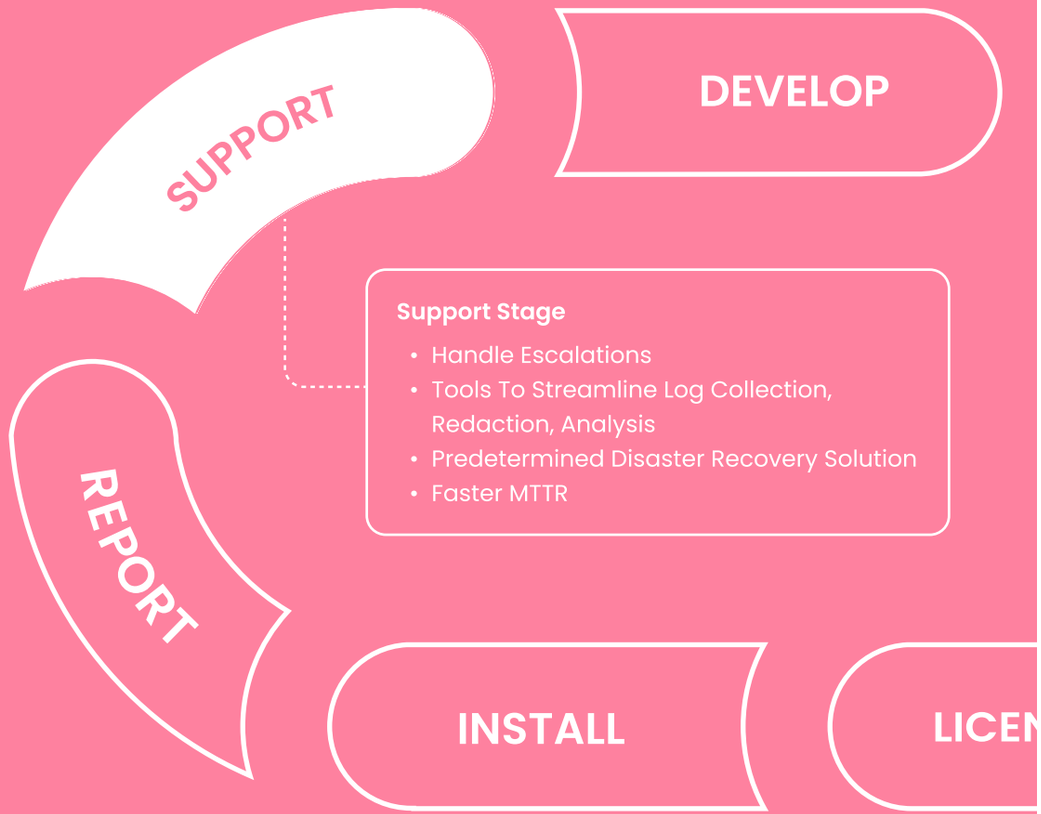
This helps to build trust between the vendor and customer in that security-minded enterprise customers can be assured that the vendor will not collect more data than was agreed to, nor expose them to potential security or privacy concerns.

Transparency, privacy, and customer choice are also critical for reporting on air gap instances, which present unique challenges due to the lack of outbound internet access. Software vendors can collect reporting data for air gap instances by providing opportunities for customers to send redacted data when opening a support request, or through regular surveys. Collecting reporting data from air gap environments on a regular basis (such a monthly or quarterly) can help give software vendors a more complete picture of how customers are using their software, while providing air gap customers valuable insight into their usage.


Report

# Support

## Commercial Software Distribution Lifecycle




Support refers to the services, tools, and documentation offered by a software vendor that help customers troubleshoot and resolve issues with their instance of the vendor's software. Providing robust support ensures that issues are resolved quickly, reducing interruptions to usage. This improves the customer experience and builds customer confidence in the software.

For enterprise software, the scope and breadth of support services provided by the vendor are often defined in a Service Level Agreement (SLA). For self-hosted software, SLAs typically define service agreements for support, such as maximum wait time for addressing issues, maximum response time for support requests, standard support hours, and emergency support. For example, 24/7 support hours and a response time of less than three hours for the most critical support issues are both common expectations for enterprise customers.

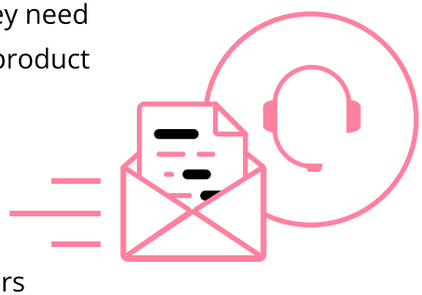
Support teams for modern enterprise software aim to reduce the mean time to resolution (MTTR) for support issues while also meeting the agreements defined in the SLA. To be successful, support teams need:

- Global coverage that enables 24/7 standard support hours
- The necessary training and expertise to address customer issues

- Access to the right diagnostic information from the customer environment, such as support logs, the Kubernetes distribution and version, and usage data

Accessing diagnostic information from the customer environment is challenging for on-prem software because the environments are often disconnected. This means that support engineers cannot simply SSH into machines or view a stream of observability data. Instead, software vendors can provide tools that collect redacted information from the customer environment and run diagnostics. This type of tooling can not only be used to generate troubleshooting suggestions for the customer, but can also provide the option for the customer to send the diagnostic information back to the vendor for additional support.

Apart from support tools and services, high-quality documentation and community-based help articles are also critical for providing customers with the information they need to self-resolve issues. Keeping the product documentation and help articles up-to-date with troubleshooting information for common support issues helps to avoid multiple different customers running into the same issue, saving time and frustration.



# Conclusion

Distributing software into self-hosted environments poses unique challenges that require unique solutions and guidelines. By following the steps outlined in the Commercial Software Distribution Lifecycle, software vendors can ensure that enterprise customers get the best possible experience, product teams lead the way towards the most impactful new features, and development teams prioritize fixing the right problems.

**Develop** with the enterprise customer in mind.

**Test** across as many unique environments as possible.

**License** software with simplicity and customization.

**Release** the right software to the right people using a consistent approach.

**Install** by meeting the customer where they are.

**Report** to surface valuable insights to internal teams and customers.

**Support** complex problems with clarity and speed.

**Create world-class self-hosted software.**

# Assessment

Want to see where your team stands in developing self-hosted software against the Commercial Software Distribution Lifecycle?

Go through this assessment together to analyze your software maturity.

## Develop

| | |
|---|---|
| **0** | App Contains Thousands Of Services, All Manually Deployed, No Automation |
| **1** | App Is Portable |
| **2** | App Is Portable And Customers Can Swap In Databases, Statefulsets, Infra Components They Manage |
| **3** | App Is Resilient (Comes Back On Failure) |
| **4** | App Is Highly Available (Avoids Failure) |

## Test

| | |
|---|---|
| **0** | Minimal Testing Outside Of Unit And Functional Tests Of The Application |
| **1** | Test Installation Into Customer Distros And Versions |
| **2** | Add CNI, CSI, CRI (Addons) Into Matrix |
| **3** | Add Upgrades For Customer Tests |
| **4** | Add Customer Representative Data Into Test Matrix |

## Release

| | |
|---|---|
| **0** | Release A Version, Customers Find It |
| **1** | Notify Customers On Release |
| **2** | Assign Customers To Channels, Work With Various Release Cadences |
| **3** | Required And Skippable Versions |
| **4** | Support For Collecting Telemetry Data From Airgap Installations |


Assessment

## License

| | |
|---|---|
| **0** | No Licensing |
| **1** | Unique Credentials For Registry |
| **2** | License Key That The App Validates To Run |
| **3** | Signed Values For Entitlements, Set Expiration Dates, Update |
| **4** | Full License Management System With Advanced RBAC For Images, Easy To Update Entitlements And Sync To Customer Instances |

## Install

| | |
|---|---|
| **0** | Customers Don't Have Choice (Need Specific K8s, Or OS) |
| **1** | Customers Can Bring Any K8s, Installs With Helm |
| **2** | Helm Plus An Embedded Cluster Option |
| **3** | Add Airgap Installation Support |
| **4** | Support For Existing Environments That Aren't K8s (Docker, Nomad, ECS) |

## Report

| | |
|---|---|
| **0** | No Visibility Into Customer Environments |
| **1** | Basic Telemetry Collected For Online Installations (Installed Or Not, Version, Ip Address, Cloud Provider) |
| **2** | Specific Operations Telemetry (Pod Status, Restarting, Deployment Rollout Status, Etc) |
| **3** | Add In Custom Metrics & Telemetry For Usage Reporting |
| **4** | Support For Collecting Telemetry Data From Airgap Installations |

## Support

| | |
|---|---|
| **0** | Customers Grab Logs And Send Them Upon Request |
| **1** | Predefined Script To Generate Common Logs |
| **2** | Log Gather Tool That Provides Redaction Capabilities |
| **3** | Log Gather Tool That Redacts And Analyzes To Give Customers Insight |
| **4** | Regular Snapshots Stored And Team Of 24/7 Experts Available To Solve Complex Environmental Issues |


Assessment

Take the number you selected from each section and add them together. Where your number falls in the below ranges indicates the quality of your software distribution architecture and how you can use this information to improve or exemplify your process.

| | |
|---|---|
| **0-7** | There's a lot to be desired in your software distribution architecture. Make sure to spend time reading through each section of this book to learn how to improve. |
| **8-16** | You have some of the basic building blocks, but there's still a lot of work to do to ensure you're providing a successful installation experience for all your customers. |
| **17-23** | You're on the right path, and have put a lot of great work into ensuring that you can distribute to many different types of customers. Still, you probably hit customer installation blockers relatively often and your self-hosted deployments could be improved. |
| **24-28** | You have set up an amazing software distribution architecture, and you're most likely successfully handling many different installation methods for your customers. |

## Great job! You've completed the assessment!

Want help improving how you distribute your application to self-hosted customers? Check out Replicated.com



# About the Authors

**Grant Miller**

Grant Miller is the CEO and Co-Founder of Replicated. He has 20+ years of experience in the technology industry, and is also the creator of EnterpriseReady.io. When not creating products, podcasts, and articles about enterprise software, he's traveling and exploring with his wife and two kids. He currently lives in Austin, TX.

**Paige Calvert**

Paige Calvert is the Documentation Manager at Replicated. She has 10+ years of experience as a technical writer in the enterprise software space. Outside of tech comm, she enjoys playing volleyball, testing out new recipes, and watching live theater. She currently lives in Denver, CO.

**Kaylee McHugh**

Kaylee McHugh is the Director of Marketing at Replicated. She has 10+ years of experience in the technology industry with roles ranging from software engineer to CEO and everything in between. In her spare time she enjoys hiking, snowshoeing, and cozying up with a good book. She currently lives in Seattle, WA.

REPLICATED

replicated.com