harpoon

Release 0.1

Dominic Holt

Sep 20, 2022

GETTING STARTED

harpoon is a No Code Kubernetes platform that makes it insanely easy to build, deploy, and maintain software. Deploying software shouldn't just be easy, it should be fun.

CHAPTER

ONE

INTRODUCTION

You shouldn't have to worry about what underlying technologies are deploying your software to the cloud. It should just work. And making it work should be simple. harpoon enables anyone to deploy their software to the cloud without writing code. It's as simple as drag and drop to get your software up and running, and it can be done in seconds. When it comes to monitoring and updating your software, harpoon handles that too. Your software is monitored in real-time to make sure it runs flawlessly. If there's a problem; you'll be notified, and harpoon can automatically restart or roll back your software to ensure a seamless experience for your end users. harpoon does this dynamically for ANY software; not a small, curated list. While simplistic and intuitive, harpoon enables capability even for the most advanced needs.

Want to run your software in the cloud? Just enter your credentials and click the start button. In a few minutes, your production environment will be fully running with security baked in. Adding any software is as simple as searching for it and dragging it onto the screen. Want to add your own software? Connect your GitHub account with only a couple clicks and choose which repository to build and deploy in seconds with no code or complicated configurations.

harpoon enables you to do everything you need, like logging and monitoring, scaling clusters, opening services, and caching data. Except with no code and in seconds. harpoon makes DevOps attainable for anyone, leveling the playing field by delivering your software to your customers at the same speed as the largest and most technologically advanced companies at a fraction of the cost.

Take the first step in simplifying your DevOps workflow; sign up for harpoon and try it today for free!

CHAPTER

TWO

FEATURES

2.1 Elements

Every element depicted below has a visual representation of what state the element is in, excluding the link elements and log elements. Translucent represents the element has no state and is waiting to be activated. Yellow represents the element that is in the process of being deployed. Blue represents a successful deployment and is running successfully. Red represents a failed deployment and error state.

2.1.1 Node Element

The node element is a visual representation of a virtual machine running inside a user's cloud service provider account. Multiple nodes represent a cluster. The cluster can be scaled up by dragging out more available nodes if the user has any available. Starting the cluster with the play button will initiate terraform scripts that will build a given cluster in the users chosen cloud service provider, or on-prem environment. Stopping the cluster with the power button will initiate terraform scripts that will tear down the given cluster and any resources associated with it. Up to 3 elements may be attached to a given node at one time. The frontend calls a backend service which contains the underlying logic required to execute and fulfill the requests input by the user. The node element is mapped to a set of machines that represent a Kubernetes cluster. The backend service manages these machines, and their supporting infrastructure,using Hashicorp Terraform. The backend service dynamically generates the required inputs for terraform based on the requested state from the user. These inputs are injected into the Terraform execution environment using environment variables. The backend service then executes Terraform within this environment by copying the generated terraform files into a temporary directory, injecting the saved state for Terraform, setting up the environment variables, and executing Terraform. The backend service then captures all output from Terraform to watch for errors and to communicate progress to the frontend. Once Terraform completes, the backend service saves any required outputs including secrets and state for use when another set of changes need to be made. The deployed cluster has a set of base services deployed and is configured to dynamically interact with the underlying cloud provider to allow for automated provisioning of volumes, services, and other cloud-native services directly from within the cluster. This also enables features such as autoscaling and other features that require integration between the cloud provider and the Kubernetes cluster managed by harpoon.

2.1.2 Git Element

The git element is a visual representation of a git repository which can be either a public or private repository. A given git repository can be built with a pre-existing docker file or write their own build commands and choose their own operating system to build the repository. Git elements can only be attached to nodes.

2.1.3 Container Element

The container element is a visual representation of a container image that can be deployed to a cluster. Container elements can only be attached to nodes. When a user calls for the deployment of a container on the frontend, a backend service is called to process the request. The frontend passes the image and configuration information to the backend for the creation of the container. harpoon stores the kubeconfig required to connect and interact with a cluster so that requests can be made directly to the cluster. The backend service takes the information provided by the frontend and dynamically generates a set of Kubernetes manifests to allow for the deployment of the container to the running cluster. The exact manifests generated vary based on the exact nature of the request from the frontend but generally, a Kubernetes Deployment object is created to instruct the cluster to deploy the container. The backend service then deploys these manifests to the cluster using its API and then watches for a successful or unsuccessful deployment of the container. The state and status of the deployment is communicated to the frontend to show the user what the state of their request is.

2.1.4 Link Element

The link element is a visual representation of a link between elements on the graph. Links are attached by dragging from one element to another until the second element is selected. A link represents a relationship between elements and how they are deployed in Kubernetes. Each element is outlined below.

2.1.5 ConfigMap Element

The config map element is a visual representation of Kubernetes ConfigMaps. A ConfigMap in Kubernetes is a Key/Value pair. ConfigMap elements can only be attached to git or container elements. When a ConfigMap element is attached to a git or container element, it modifies the deployment descriptor for the relevant Kubernetes pod that is already deployed in the Kubernetes cluster and then executes a command in Kubernetes to update the configuration for that deployment using the Kubernetes API. Much like the Container deployments, the frontend makes a call to the backend which generates the required manifests and pushes them to Kubernetes dynamically on behalf of the user. The backend service also modifies the associated container deployment to expose the created ConfigMap, in the user-specified manner, to the running container deployment.

2.1.6 Volume Element

The Volume element is a visual representation of a Kubernetes Persistent Volume Claim (PVC) for a given git or container element. Users can input the volume directory location inside a Kubernetes Pod where the data will be replicated to a distributed volume in the cloud. Volume elements can only be attached to git or container elements. When a Volume element is attached to a git or container element, it modifies the deployment descriptor for the relevant Kubernetes pod that is already deployed in the Kubernetes cluster and then executes a command in Kubernetes to update the configuration for that deployment using the Kubernetes API. The PVC in Kubernetes that is deployed will be dynamically linked to the distributed volume in the cloud. Much like the Container deployments, the frontend makes a call to the backend which generates the required manifests and pushes them to Kubernetes dynamically on behalf of the user. The backend service also modifies the associated container deployment to expose the created Volume, in the user-specified manner, to the running container deployment.

2.1.7 Ingress Element

The ingress element is a visual representation of a Kubernetes Ingress Route for the deployed git or container element. Users can directly input the port number that will be used to open the port for the relevant Pod in Kubernetes. Clicking the lock image on an ingress element will open the lock and open the attached Container/Pod to the internet. Ingress elements can only be attached to git or container elements. When an Ingress element is attached to a git or container element, it modifies the deployment descriptor for the relevant Kubernetes pod that is already deployed in the Kubernetes cluster and then executes a command in Kubernetes to update the configuration for that deployment using the Kubernetes API. Depending on the exact cloud provider and ingress plane configured by the Kubernetes deployment, harpoon will generate the required manifests to configure Ingress at the Kubernetes level. For some Service Mesh based deployments, the harpoon backend services will deploy a loadbalancer using the same Terraform mechanism used for the rest of the cluster. This is then configured to interact with the Service Mesh within the cluster to allow for automated configuration of ingress into the cluster. The backend service will then monitor the standup of the route both internally and externally to inform the user that the route is ready for use. This can include monitoring DNS servers to watch for when names propagate and are ready for use by users.

2.1.8 Secret Element

The secret element is a visual representation of Kubernetes secrets storage for a given git or container element. Secret elements can only be attached to git or container elements. A secret element also takes a key/value pair, much like a ConfigMap, but offers more security/encryption through the Kubernetes secrets storage capability. When the Secret element is attached to a git or container element, it enables the relevant Kubernetes Pod to then use the key associated with the secret as a reference to the value of the secret, thereby obfuscating the true value of the secret in any source code or variables in use by the Pod and giving the option to dynamically modify the secret value without updating the software running in the Pod. Much like the Container deployments, the frontend makes a call to the backend which generates the required manifests and pushes them to Kubernetes dynamically on behalf of the user. The backend service also modifies the associated container deployment to expose the created Secret, in the user-specified manner, to the running container deployment.

2.1.9 Pod Log Element

The log element is a visual representation of logs outputted by the deployed Kubernetes Pods giving users the ability to see what is happening inside their deployed container image. When a user clicks the log button on a specific container or git element that is already deployed (via the Deploy button), a request is made to harpoon's deployment microservice to retrieve the logs. The deployment microservices calls the Kubernetes API to return the logs for the specified pod ID within the relevant namespace. The deployment service waits for Kubernetes to return the response and then forwards that response to the harpoon frontend to display the relevant log data to the user. The harpoon backend services connect directly to the Kubernetes API for the user cluster, using the same dynamic mechanism as the other Kubernetes objects, to pull logs for the user deployments. These are then sent to the frontend for visualization by the user.

2.2 Search

2.2.1 Search git repositories (public and private)

Users can search for both public and private git repositories. A user links their Github account (a third-party provider) to harpoon using a token from GitHub. When the user searches for a repository by typing in the text of their search term (string), the string is sent to a harpoon microservice where it is combined with the token to make a request to the Github API to find relevant repositories that match the string. When a response is received from the GitHub API, the harpoon microservice sends the response to the harpoon frontend to display with all the relevant data associated in JSON format that can be parsed into the display.

2.2.2 Search for container images

Users can search for container images. A user searches for a container image by typing in the text of their search term (string), the string is sent to a harpoon microservice to make a request to Docker Hub to find relevant container images that match the string. When a response is received from Docker Hub, the harpoon microservice sends the response to the harpoon frontend to display with all the relevant data associated in JSON format that can be parsed into the display.

2.3 Third-party Integration

2.3.1 Link accounts

Users have the ability to link their third-party accounts to harpoon in order to search for software to deploy using harpoon in a drag and drop fashion or connect to multiple cloud providers. The list of third-party providers is currently:

- *Amazon Web Services (AWS)*
- *VMWare*
- *GitHub*
- *Docker Hub*
- *Harbor*

2.4 Other

2.4.1 Projects

Users have the ability to separate deployments into different Projects. Projects are physically on the same cluster but logically isolated. In this way, Project A cannot talk to Project B. Users can create a project from scratch or copy existing projects into a workspace.

CHAPTER

THREE

PRODUCTS

While our products currently share 98% of the same source baseline, we currently have two distinct offerings:

- harpoon SaaS
- harpoon On-Prem

3.1 harpoon SaaS

This is a hosted solution online that deploys any software dynamically to an AWS account. You provide the AWS account and we manage all of the virtual infrastructure and Kubernetes bits through the harpoon interface. You can access harpoon from anywhere and deploy and maintain your software at your leisure. harpoon SaaS is a multi-tenant solution that powers many businesses in a hybrid-SaaS model (e.g. our software, your account).

3.2 harpoon On-Prem

This is exactly what it sounds like. Instead of running a multi-tenant solution in the cloud, we are providing a single-tenant solution that runs only in your datacenter. Currently, harpoon On-Prem supports VMWare-based deployments and comes with software that bootstraps 100% of the installation of harpoon On-Prem itself. It also comes with all the necessary binaries to complete the installation process. Another nifty feature of harpoon On-Prem is that it runs just fine entirely offline, so if you want to run it in a completely air-gapped environment, you can!

CHAPTER

FOUR

USING HARPOON

4.1 Registration

4.2 Linking a Cloud Service Provider Account

See *Amazon Web Services (AWS)*

4.3 Starting a cluster

Once you've linked your cloud service provider account, you just click the "start" button on the cloud/node element in the workspace. That's it. No, really! The cloud/node element will turn yellow and provide a little spinny wheel to entertain you. When the cluster is running, the cloud will return and the element will glow a happy blue color.

4.4 Tearing down a cluster

Don't need your cluster running all the time? No problem! Hit the "shutdown" button on the cloud/node element in the workspace. This will automatically tear down all the infrastructure harpoon was using in your cloud service provider account

WARNING: This will really destroy all of the infrastructure in your cloud service provider account that was provisioned by harpoon. You can't come back from this aside from starting over, so make sure you really do want to make everything go away.

CHAPTER

FIVE

AMAZON WEB SERVICES (AWS)

If you want to deploy software on top of AWS, you will need to provide harpoon with an access key ID and a secret access key. Since harpoon is deploying all of the necessary infrastructure in AWS in addition to the Kubernetes cluster, we require fairly extensive access to the account in order to successfully perform the provisioning of the environment. The following are the specific permissions harpoon needs to successfully deploy a cluster:

- AmazonRDSFullAccess
- IAMFullAccess
- AmazonEC2FullAccess
- AmazonVPCFullAccess
- AmazonS3FullAccess
- AWSKeyManagementServicePowerUser

Also note that harpoon does provision virtual infrastructure in your AWS account, so while harpoon will not interfere with any other virtual infrastructure in your AWS account, you could always potentially run into hitting a resource limit that might require you to request a higher limit for a particular resource.

CHAPTER

SIX

VMWARE

CHAPTER

SEVEN

DOCKER HUB

As soon as you register an account with harpoon, you can immediately start searching for container images on Docker Hub; there are no credentials required to perform the search, and you can drag and drop whatever container image you want into the workspace for deployment. harpoon populates all the possible "tags" or versions of a container image and you can choose to deploy whichever tag you'd like by selecting it. You can tell a container image is from Docker Hub because it will have a small Docker logo in the bottom right corner.

Once you've dragged a container image into the workspace, all you have to do is click "Deploy" and harpoon will handle the rest. If the container image turns blue it has deployed successfully while red means a problem has occurred during deployment. You can click the "i" button to see the logs for the container image at any time.

CHAPTER

EIGHT

GITHUB

harpoon can connect to your GitHub account with as little as the click of a button. Once your GitHub account is attached to harpoon, you can search for any public or private repository you have access to. To make it easier to find your own or your organizations' private repositories, we provide a toggle button so that you can switch back and forth between public and private repository search. When toggled to "private" search, only the private repositories you have access to will be returned in the search.

Once you find the repository you're looking for, you can drag and drop it onto the workspace. You'll know that an element in the workspace is a GitHub repository because it will have the Octocat logo in the bottom right corner of the element.

In order for harpoon to successfully build a GitHub repository, we currently require the repository to have a top-level Dockerfile, which is industry best practice. If the Dockerfile is there, once you click the "Build" button, harpoon will automatically find it and build a container image that gets pushed to a private container registry only harpoon has access to. After a successful build, the "Deploy" button will become enabled, and you can deploy the software directly.

If you're running into issues building or deploying any particular GitHub repository, remember you can always click on the log button to see what is going on under the covers.

CHAPTER

NINE

HARBOR

CHAPTER

TEN

TROUBLESHOOTING

harpoon is made to be as intuitive as possible, but sometimes there are complexities with deploying and manipulating software in general, and these complexities can sometimes be confusing. While we endeavor to resolve this confusion and eliminate complexity from harpoon with great resolve, the following are some examples of common scenarios that may cause some cognitive dissonance.

10.1 Starting a cluster

After providing your cloud service provider keys and hitting the "start" button on the cloud node element, the node will turn yellow and provide a status indicator that harpoon is working. Unfortunately, this operation is quite complex and is configuring your cloud service provider account with numerous infrastructure components which are being provisioned and configured in real-time. Once the infrastructure layer has been provisioned, harpoon then deploys a Kubernetes cluster on top and dynamically configures it to work specifically with your cloud service provider account. It may seem like harpoon is being "slow" or "unresponsive" during this time. The truth is, it is just delivering a massive amount of value to the end user, which can take a bit of time. We find the average time to complete a cluster deployment is about 3 minutes.

10.2 Deploying third-party container images from Docker Hub

harpoon provides dynamic deployment of over a million container images from Docker Hub, but we don't verify or validate anything about the container images you are choosing to deploy in your Kubernetes cluster. Your mileage may vary when it comes to successfully deploying third-party container images that you are not personally familiar with.

10.3 Deploying third-party git repositories from GitHub

There are over 120 million public repositories at your fingertips to search GitHub using harpoon, but much like third-party container images from Docker Hub, many of these repositories require special configuration to work correctly, some of them may not compile, or have other unknown issues. Additionally, at this time, it is a requirement for a repository to have only one Dockerfile at the top level of the repository (best practice) in order for harpoon to be able to deploy it dynamically.

10.4 Deployment times

Many container images and git repositories deploy in only a matter of seconds using harpoon, but some can take longer. This could be because the repository or container image is especially large, or it could be related to an issue with the container/software itself. If some software is taking a long time to deploy and then eventually turns red, it could be timing out after a number of failed attempts to deploy the software.

CHAPTER

ELEVEN

OPEN A SUPPORT TICKET

If you're having trouble using harpoon or accomplishing a specific task, you can always file a support ticket with our helpdesk, which you can find here: https://help.harpooncorp.io/