THENEWSTACK

CLOUD NATIVE ECOSYSTEM / SOFTWARE DEVELOPMENT

Our Approach to Config-as-Code at Octopus Deploy

A look at the development process for Octopus Deploy's configuration as code implementation, as well as the design decisions and more.

May 31st, 2022 11:00am by Terence Wong

Featured image via Pixabay.

Octopus Deploy's customers have been asking for a configuration-as-code (CaC) solution for a long time, and after a two-year development period, we released it in March.

Although it is one of our most requested features, it took longer than expected to implement, but we learned some valuable lessons along the way, and we're really proud of this key feature.

CaC captures configuration settings in a source-controlled environment such as Git, making it possible to test configuration changes in branches and approve the changes through pull requests.

Developing a configuration-as-code feature for a deployment tool like Octopus Deploy makes sense. Our customers wanted it, we wanted to use it internally, but it's a feature that cuts across all layers of the application and couldn't be simply "bolted-on."

We believe we've developed an industry-leading CaC implementation that many organizations can learn from. In this post. I'll share our development process, the design decisions we made, the benefits of our CaC feature, our release process and our CaC philosophy.

Why Did We Implement CaC?

Two years ago, it was clear from customer feedback that CaC was the next logical step for our platform, but looking at CaC implementations from other vendors, we wanted to do it differently and better.

Our customers also told us that CaC is usually a step back from a usability standpoint. When CaC is turned on, users drown in YAML, and there are often several compromises in the UI to accommodate CaC features. Despite the compromises, customers put up with them because there's no alternative and there are clear benefits to having CaC functionality. We wanted to build a CaC implementation that has all the CaC benefits but none of the usability compromises that follows.

The Development Process

Our CaC journey started as a throwaway proof of concept built in six weeks. It was our first time building a major throwaway feature. In this ase, there were a lot of unknowns, and we tested and validated many umptions.

When we began shaping CaC, we suddenly had multiple persistence layers to consider. In a CaC implementation, there is no single version of the system — users can switch between different states of the system through branching. A persistence layer is required to capture all the different states of the system. If you would like to know more, we wrote a blog about our thoughts on persistence in CaC.

After validating the proof of concept, we built CaC for real for a year. At the end of the year, the engineering team had learned many lessons and wanted to change some fundamental design choices before shipping. These lessons required development to move one step back before we could take two steps forward. For example, our team wrapped the initial CaC implementation around the command line, but performance issues meant we had to swap this out for the LibGit2Sharp library.

The tension between having a shippable product now and knowing we could do better led to an additional year of development before a general release. We chose to release a version that we were thrilled with, not one that was just good enough for a deadline.

When we look back on the two-year journey and the trade-offs we accepted, we believe the result was well worth it!

Design Decisions

When using existing CaC vendor solutions, users must choose between a UI or a CaC implementation. We didn’t want our users to have to choose. We wanted to harness the power of CaC but maintain all of our UI functionality.

-Based

On one side, users use Git as a sync mechanism, where users only use Git as a database. This approach is not ideal as the true power of Git is not being used.

On the other side is a Git-native implementation with branching and pull requests but no UI. In this case, users drown in excessive YAML and have no easy way to make simple changes without being knee-deep in code.

We wanted to provide a fully-featured UI experience with Git-native powers so users can use the UI to make minor changes or dive deep into the configuration language in Git and make changes there. We chose to use Git concepts such as branches and pull requests rather than abstract those concepts into Octopus-specific terms. Using Git terms allows new users to understand the CaC concepts we are exposing without needing to be an Octopus expert.

Configuration Language

Any CaC implementation needs a configuration language.

Our configuration language had to be easy to read, write and edit complex documents. We knew that YAML, XML and JSON were strong candidates based on their widespread use in other tools.

JSON was the easiest to implement, but it's suitable for serialized objects, not documents.

YAML is readable, but it can be tricky to edit. YAML is also better for trivial documents and becomes harder to use with complex documents.

IL was too verbose compared to the other languages.

Based on the direction HCL was going in future releases, we decided to create our HCL-based language called Octopus Configuration Language (OCL), which is entirely independent of HCL. We've built our own parser/serializer, and there's no obligation on us to follow any direction Hashicorp takes HCL and nothing preventing us from making changes.

In truth, any of these language choices could have worked as they are popular in the industry. We went with what we felt worked best for us.

Work in the UI or Code

In our CaC implementation, users can use either the UI or the source-controlled implementation. Both modes are fully featured. If one user prefers to use the UI only, they can accomplish as much as another user that chooses to use the source-controlled version. These two users can also work together using their preferred mode.

Benefits of CaC

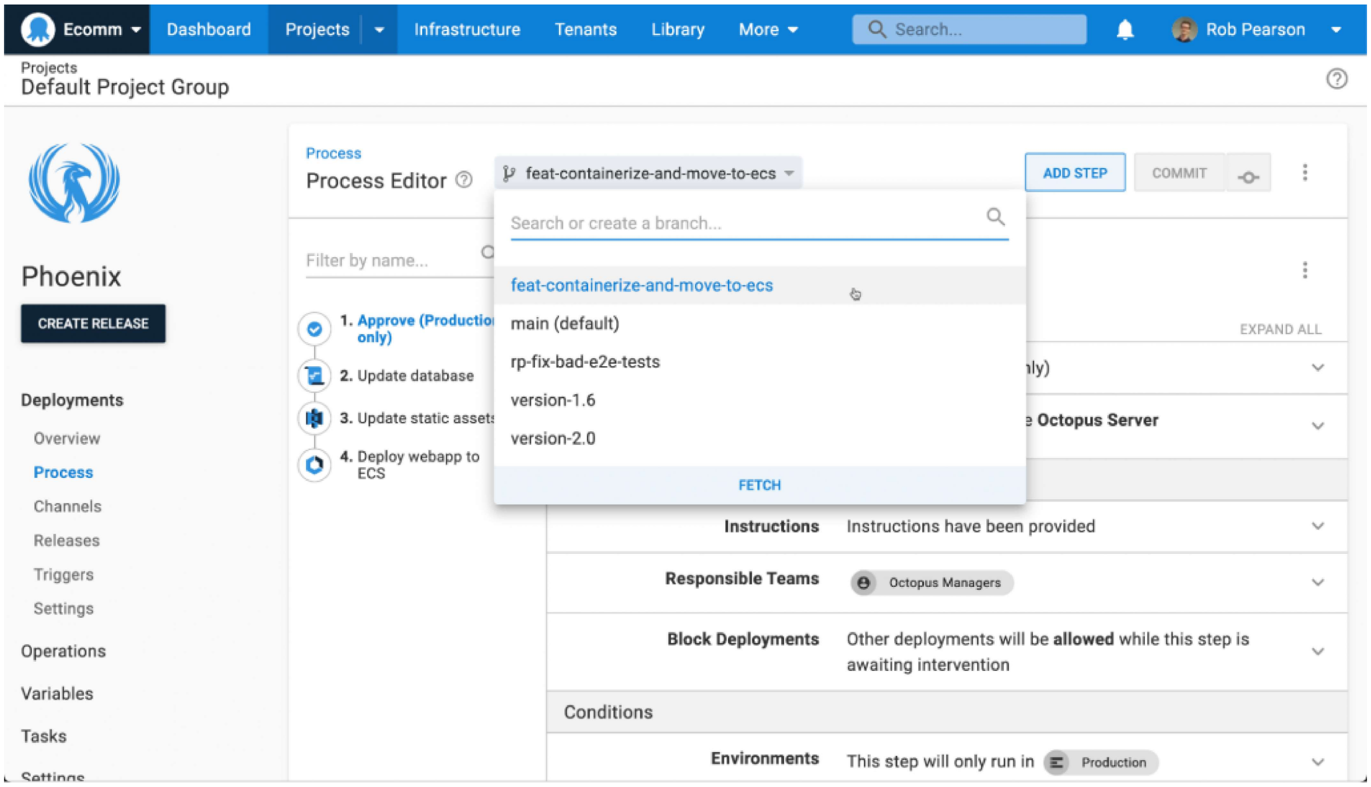
As I mentioned earlier, CaC in Octopus Deploy allows users to evolve their Octopus configuration alongside their application code. Here are some of the benefits that we believe our CaC implementation provides:

Branches as a Core Concept

Octopus Deploy fully exposes the power of Git branches, and you can:

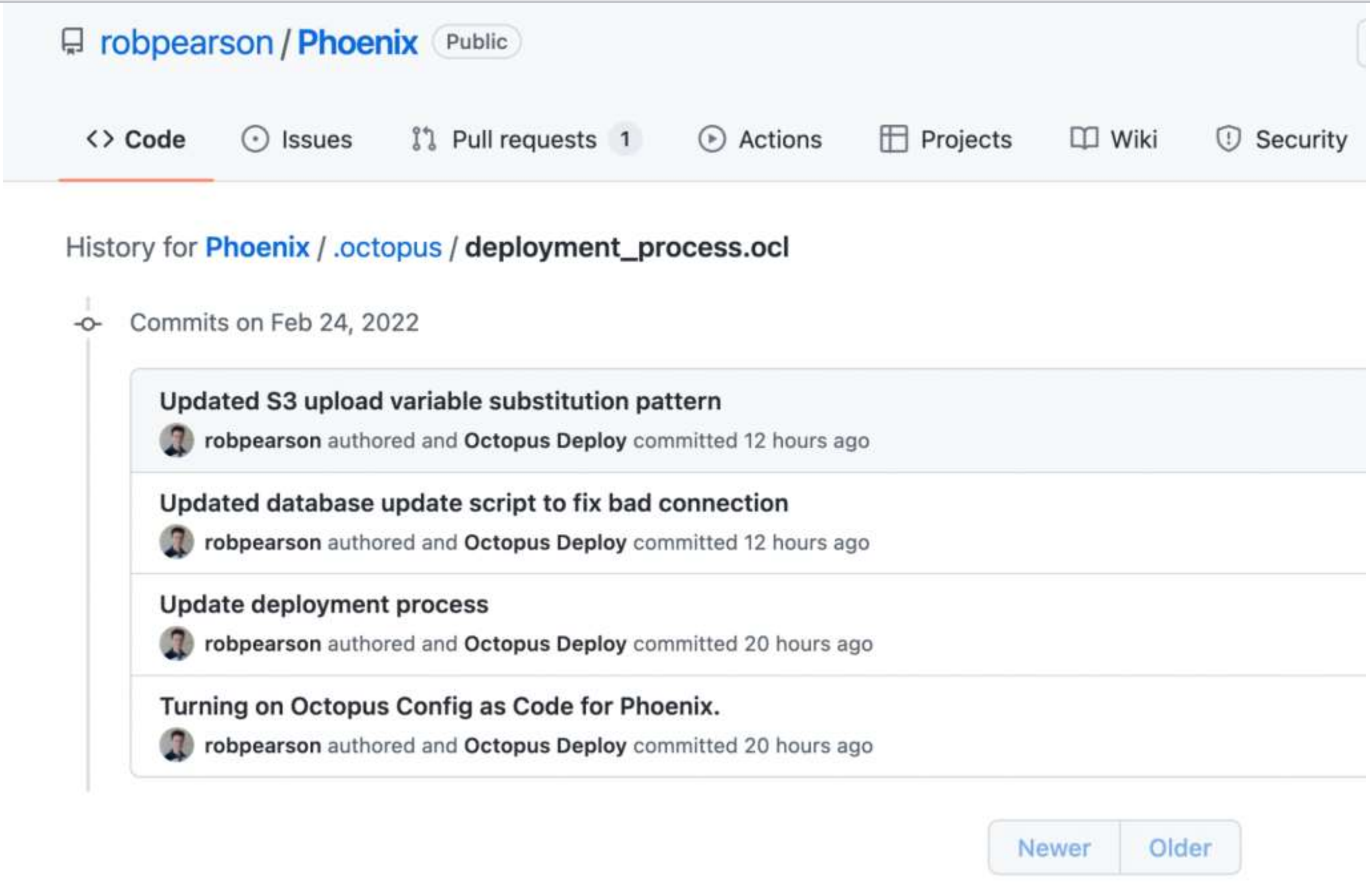
- Switch branches or create new ones in the Octopus UI or your favorite Git client.
- Commit changes to your deployment process on a branch.
- Add (optional) commit messages when you change your deployment process so others understand why it changed.
- Create releases from your branch and deploy to test your changes, to support many versions of your deployment process.

- Roll back to a previous version of your deployment process if things go wrong.



Auditability/Traceability

Users can roll configuration back to previous settings, and users can view an entire audit history. The commit history improves the traceability of changes to a deployment process. Users know what changed, when, by whom and, most importantly, why. Users don't have to search through audit logs to determine why their configuration changed. Git history and diffs provide clear traceability, telling the complete story. Octopus even records the committer details in the Git history.



History for Phoenix / .octopus / deployment_process.ocl

Commits on Feb 24, 2022

Updated S3 upload variable substitution pattern
robpearson authored and Octopus Deploy committed 12 hours ago

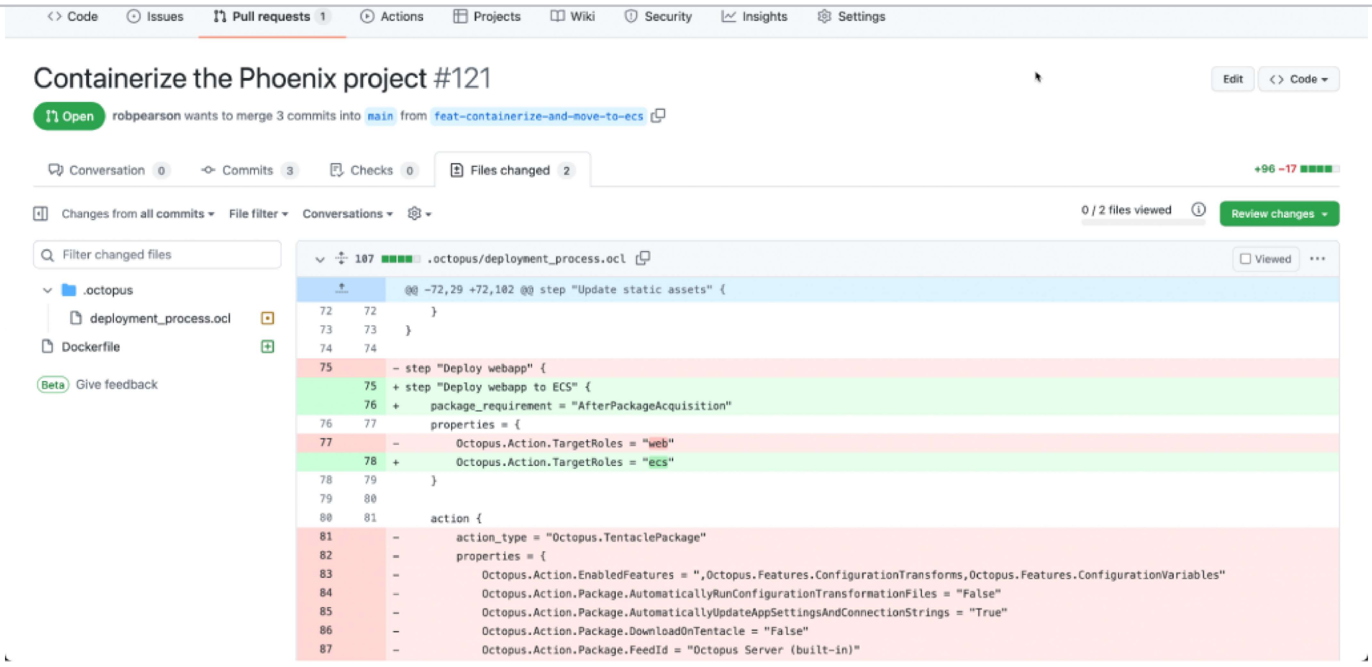
Updated database update script to fix bad connection
robpearson authored and Octopus Deploy committed 12 hours ago

Update deployment process
robpearson authored and Octopus Deploy committed 20 hours ago

Turning on Octopus Config as Code for Phoenix.
robpearson authored and Octopus Deploy committed 20 hours ago

Pull Requests as Approvals

Pull requests are built into Git and can act as an approval process in the deployment process. Pull requests, protected branches and code owners enable a new set of workflows to improve the quality and safety of releases. By incorporating pull requests, we can help reduce downtime from bad deployments and improve the quality of releases.



Containerize the Phoenix project #121

Open robpearson wants to merge 3 commits into `main` from `feat-containerize-and-move-to-ecs`

107 .octopus/deployment_process.ocl

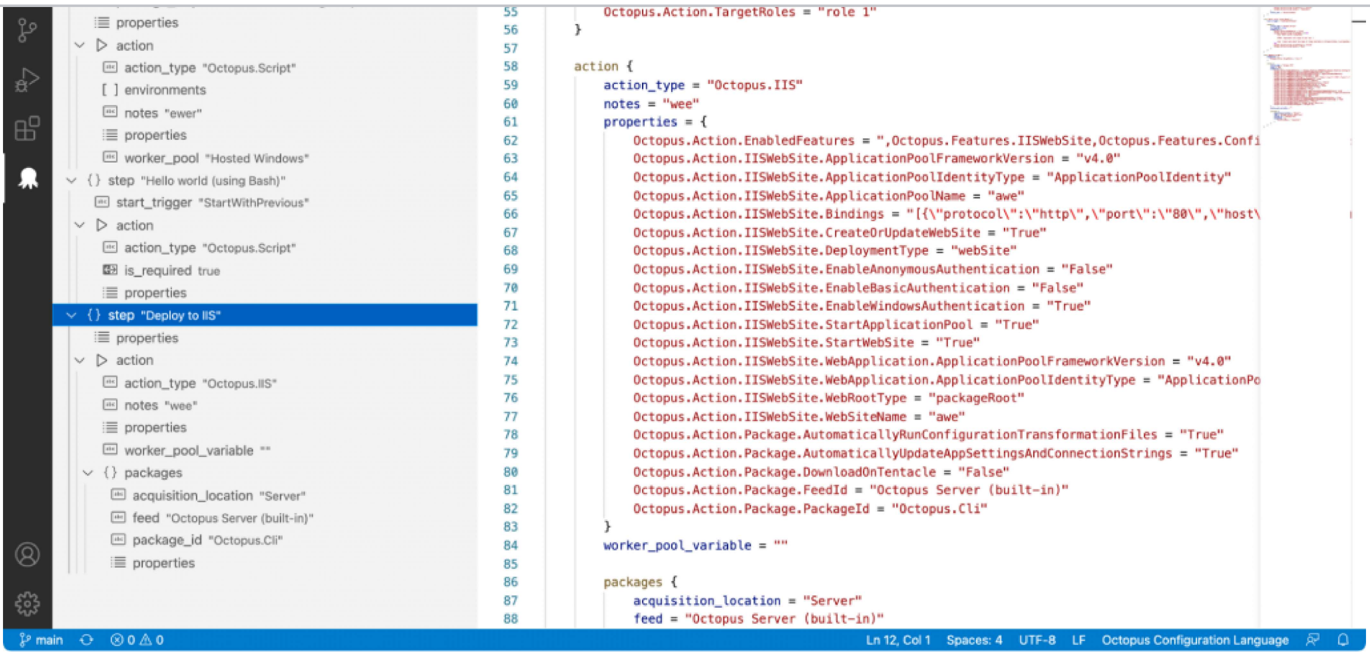
```
@@ -72,29 +72,102 @@ step "Update static assets" {
        }
    }

- step "Deploy webapp" {
+ step "Deploy webapp to ECS" {
+     package_requirement = "AfterPackageAcquisition"
      properties = {
-         Octopus.Action.TargetRoles = "web"
+         Octopus.Action.TargetRoles = "ecs"
      }

      action {
-         action_type = "Octopus.TentaclePackage"
-         properties = {
-             Octopus.Action.EnabledFeatures = ",Octopus.Features.ConfigurationTransforms,Octopus.Features.ConfigurationVariables"
-             Octopus.Action.Package.AutomaticallyRunConfigurationTransformationFiles = "False"
-             Octopus.Action.Package.AutomaticallyUpdateAppSettingsAndConnectionStrings = "True"
-             Octopus.Action.Package.DownloadOnTentacle = "False"
-             Octopus.Action.Package.FeedId = "Octopus Server (built-in)"
```

Text Editor and Visual Studio Extension

The Octopus UI has a text editor to make changes to configuration code, which can be easier than making changes directly to the code in Git. To make working with OCL easier, we built an extension for Visual Studio Code to complement the Octopus UI.

The OCL editing experience includes:

- Syntax highlighting
- Code snippets for steps and actions
- An integrated tree view to navigate nodes in the file



```
properties
action
    action_type "Octopus.Script"
    [ ] environments
    notes "ewer"
    properties
    worker_pool "Hosted Windows"
{} step "Hello world (using Bash)"
    start_trigger "StartWithPrevious"
    action
        action_type "Octopus.Script"
        is_required true
        properties
{} step "Deploy to IIS"
    properties
    action
        action_type "Octopus.IIS"
        notes "wee"
        properties
        worker_pool_variable ""
        {} packages
            acquisition_location "Server"
            feed "Octopus Server (built-in)"
            package_id "Octopus.Cli"
            properties
```

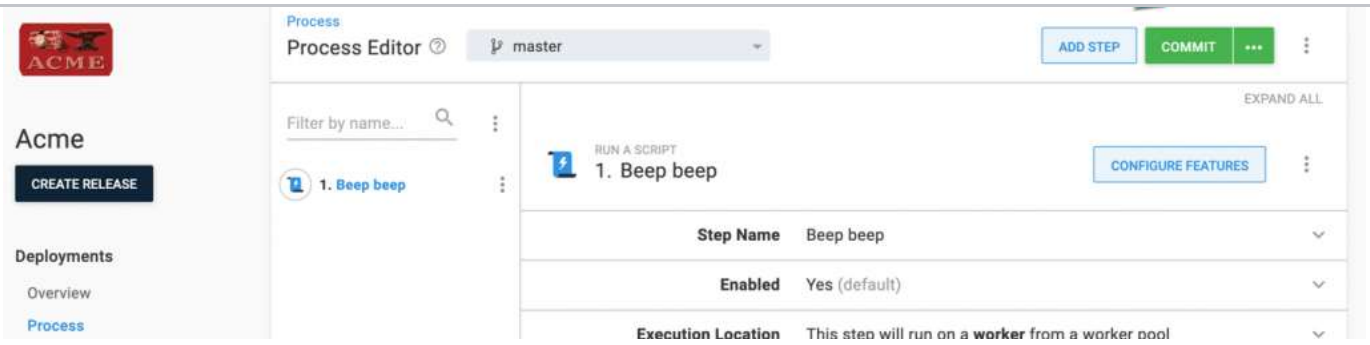
```
        Octopus.Action.TargetRoles = "role 1"
    }

    action {
        action_type = "Octopus.IIS"
        notes = "wee"
        properties = {
            Octopus.Action.EnabledFeatures = ",Octopus.Features.IISWebSite,Octopus.Features.Confi
            Octopus.Action.IISWebSite.ApplicationPoolFrameworkVersion = "v4.0"
            Octopus.Action.IISWebSite.ApplicationPoolIdentityType = "ApplicationPoolIdentity"
            Octopus.Action.IISWebSite.ApplicationPoolName = "awe"
            Octopus.Action.IISWebSite.Bindings = "[{\"protocol\":\"http\",\"port\":\"80\",\"host\
            Octopus.Action.IISWebSite.CreateOrUpdateWebSite = "True"
            Octopus.Action.IISWebSite.DeploymentType = "webSite"
            Octopus.Action.IISWebSite.EnableAnonymousAuthentication = "False"
            Octopus.Action.IISWebSite.EnableBasicAuthentication = "False"
            Octopus.Action.IISWebSite.EnableWindowsAuthentication = "True"
            Octopus.Action.IISWebSite.StartApplicationPool = "True"
            Octopus.Action.IISWebSite.StartWebSite = "True"
            Octopus.Action.IISWebSite.WebApplication.ApplicationPoolFrameworkVersion = "v4.0"
            Octopus.Action.IISWebSite.WebApplication.ApplicationPoolIdentityType = "ApplicationPo
            Octopus.Action.IISWebSite.WebRootType = "packageRoot"
            Octopus.Action.IISWebSite.WebSiteName = "awe"
            Octopus.Action.Package.AutomaticallyRunConfigurationTransformationFiles = "True"
            Octopus.Action.Package.AutomaticallyUpdateAppSettingsAndConnectionStrings = "True"
            Octopus.Action.Package.DownloadOnTentacle = "False"
            Octopus.Action.Package.FeedId = "Octopus Server (built-in)"
            Octopus.Action.Package.PackageId = "Octopus.Cli"
        }
        worker_pool_variable = ""

        packages {
            acquisition_location = "Server"
            feed = "Octopus Server (built-in)"
```

Commit Messages

In line with maintaining Git concepts, we replaced the "Save" button with the "Commit" button in the Octopus UI. In a release scenario, there are two types of commits:

- One with a significant change to the deployment process that requires a commit message.
- One that is part of a series of commits related to fixing the same issue. This scenario is useful when unique commit messages for every commit provides little value.

Octopus caters to both scenarios and allows users to accept a default commit message or provide their own.



Step Name: Beep beep
Enabled: Yes (default)
Execution Location: This step will run on a **worker** from a worker pool

Release Process

We released an early access preview to allow users to provide feedback that would inform the general release.

Config as code is now available for deployments, but it also makes sense to provide CaC for variables and runbooks, and this is something we're working on. We shared a public roadmap to inform users of upcoming work and planned features.

Our Configuration as Code Philosophy

Configuration as code sounded great from a customer and developer perspective. Our customers would get to use it, we'd get to build and use it internally, and everyone wins!

The natural extension of this is to have everything as code (EaC) eventually. Every aspect of the deployment process could be as code, but the added value for concepts like environments and tenants could be low. There's also concerns about secrets to be as code as they shouldn't be stored in version control. Once deployments, runbooks and variables are as code in Octopus, there are diminishing returns for progressing to an EaC system compared to other features we could develop.

Ideally, users could use the OCL language to provision infrastructure. A more mature scenario would be to use EaC to replicate everything in the production environment in the local development environment for testing. This would validate any changes to the entire production environment before a release. In EaC, everything becomes throwaway as everything is as code.

The big question to answer is what part of the DevOps pipeline is throwaway?

Do we need to abstract every aspect of the pipeline in a version-controlled system, or are some parts of the pipeline better suited to traditional storage methods? The team will continue to tackle these problems and more as we move forward with CaC in Octopus.

TNS

Terence Wong is a software engineer who loves problem-solving and technology. Terence enjoys understanding new technologies and communicating complex information to different audiences. As a technical content creator at Octopus, Terence works with DevOps and cloud technologies to understand and...

Read more from Terence Wong →

ATED STORIES

e Case for Environment-Specific Docker Images

How to Enable and Use the Minikube Dashboard

I Need to Talk to You about Kubernetes GitOps

INSIGHTS FROM OUR SPONSOR

Octopus Deploy

Founded in 2012, Octopus Deploy helps DevOps teams at over 25,000 companies accelerate reliable, repeatable, and traceable deployments across clouds and on-prem infrastructure. Octopus also integrates with hundreds of technologies, including Azure, AWS, GCP, and Kubernetes. Octopus Deploy and TNS are under common control.

Learn More →

The new DevOps performance clusters
18 January 2023

DevOps and platform engineering
11 January 2023

Getting started with Bamboo
4 January 2023

What is GitOps?
21 December 2022

An Octopus Christmas Carol
19 December 2022

New in GitHub Actions for Octopus Deploy v3
14 December 2022

THENEWSTACK

A newsletter digest of the week's most important stories & analyses.

EMAIL ADDRESS

SUBSCRIBE

THENEWSTACK

WebAssembly
Cloud Services
Data
Machine Learning
Security

OPERATIONS

Platform Engineering
Operations
CI/CD
Tech Life
DevOps
Kubernetes
Observability
Service Mesh

CHANNELS

Podcasts
Ebooks
Events
Newsletter
TNS RSS Feed

THE NEW STACK

About / Contact
Sponsors
Sponsorship
Contributions

roadmap.sh

THENEWSTACK

Backend Developer Roadmap

Devops Roadmap