

Deploy a Java application to Tomcat using Octopus and GitHub Actions

In this tutorial, we show you how to build a fully-functional continuous delivery pipeline for a simple Java web application and deploy it to Tomcat. We use GitHub Actions to build the code and run tests, and we use Octopus Deploy to deploy and promote releases.

To get up and running quickly, the [TestDrive VMs](#) provide preconfigured environments demonstrating various continuous delivery pipelines documented in these guides.

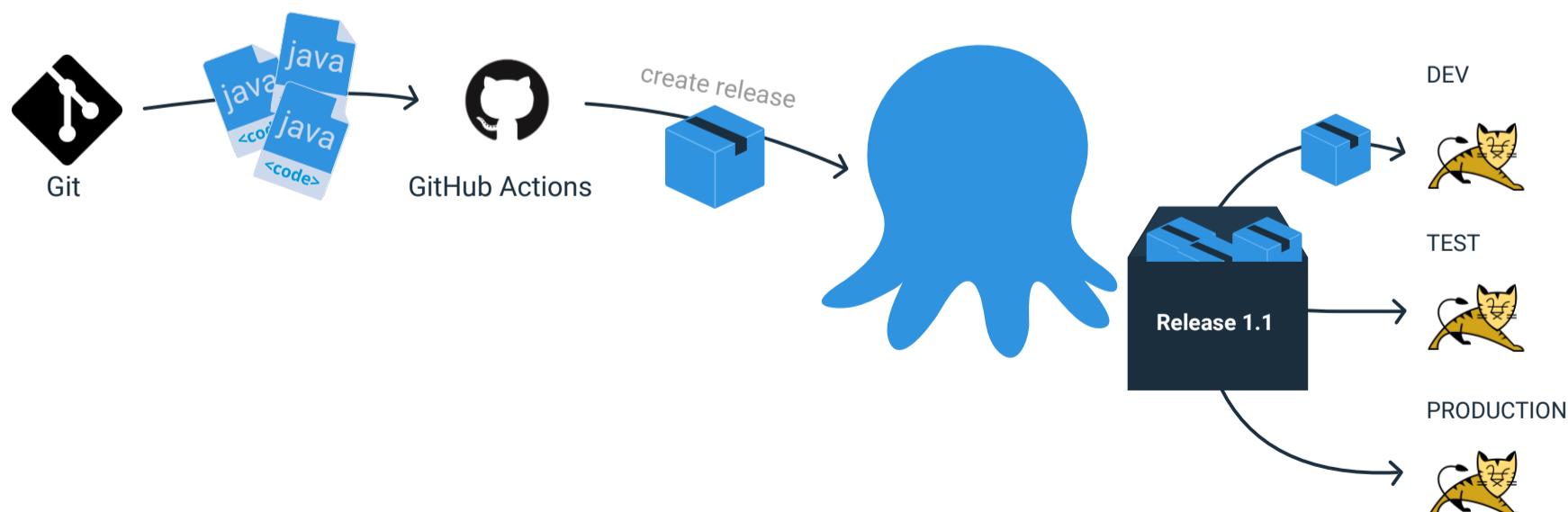
[TestDrive VMs →](#)

Introduction

The application we'll deploy is called **Random Quotes**, which is a simple web application that randomly displays a famous quote each time the page loads. It consists of a web front end and a database that contains the quotes. We'll build a complete Continuous Integration/Continuous Delivery (CI/CD) pipeline with automated builds, deployments to a dev environment, and sign offs for production deployments.

Deployment pipeline

For this tutorial, we assume you use **Git** for version controlling changes to your source code and **GitHub Actions** to compile code and run unit tests. **Octopus Deploy** will take care of the deployment. Here is what the full continuous integration and delivery pipeline will look like when we are finished:



The development team's workflow is:

Octopus Deploy

running unit tests.

- 3 When the GitHub Actions build completes, the change will be deployed to the **Dev** environment.
- 4 When one of your team members (perhaps a tester) wants to see what's in a particular release, they can use Octopus to manually deploy a release to the **Test** environment.
- 5 When the team is satisfied with the quality of the release and they are ready for it to go to production, they use Octopus to promote the release from the **Test** environment to the **Production** environment.

Since Octopus is designed to be used by teams, in this tutorial we also set up some simple rules:

- Anyone can deploy to the dev or test environments.
- Only specific people can deploy to production.
- Production deployments require sign off from someone in our project stakeholders group.
- We'll send an email to the team after any test or production deployment has succeeded or failed.

This tutorial makes use of the following tools:

			
Git Source control Version control of source code	GitHub Actions Build server Compile code, run tests	Octopus Deploy Deployment server Release management & deployments	Tomcat Java application server Hosts the running web application

Octopus is an extremely powerful deployment automation tool, and there are numerous ways to model a development team's workflow in Octopus Deploy, but this tends to be the most common for small teams. If you're not sure how to configure Octopus, we recommend following this guide to learn the basics. You'll then know how to adjust Octopus to suit your team's workflow.

This tutorial takes about an hour to complete. That sounds like a long time, but keep in mind, at the end of the tutorial, you'll have a fully-functional CI/CD environment for your entire team, and you'll be ready to deploy to production at the click of a button. It's worth the effort!

Build vs. deployment

For any non-trivial application, you're going to deploy the software to multiple environments. For this tutorial, we're using the environments **Dev**, **Test**, and **Prod**. This means you need to choose between building your application once or building it before each deployment? To reduce the risk of a failed production deployment, Octopus strongly encourages the practice of building once, and deploying multiple times.

The following activities are a **build time** concern, so they will happen in GitHub Actions after any change to code is committed to Git:

1. Check out the latest changes from Git.
2. Resolve and install any dependencies from Maven.
3. Run unit tests.
4. Package the application by bundling all the files it needs to run into a WAR file.

This results in a green CI build and a file that contains the application and everything it needs to run. Any configuration files will have their default values, but they won't know anything about dev vs. production settings just yet.

Octopus Deploy

An example of a package that is ready to be deployed is:

```
randomquotes.1.0.0.war
```

At this point, we have a single *artifact* that contains all the files our application needs to run, ready to be deployed. We can deploy it over and over, using the same artifact in each environment. If we deploy a bad release, we can go and find the older version of the artifact and re-deploy it.

The following activities happen at **deployment time** by Octopus Deploy:

1. Changing any configuration files to include settings appropriate for the environment, e.g., database connection strings, API keys, etc.
2. Running tasks that need to be performed during the deployment such as database migrations or taking the application temporarily offline.

Octopus Deploy

There are a number of tools you need to install to implement a complete CI/CD workflow. These include the GitHub Actions and Octopus servers, some command-line tools, and Tomcat to host the final deployment.

Git

The source code for the sample application is hosted on GitHub. To access the code, you need the Git client. The [Git documentation](#) has instructions to download and install the Git client.

GitHub Actions

[GitHub Actions](#) are available for free on public GitHub repositories and with paid options for private repositories.

Java

The Java Developer Kit (JDK) is required to be installed to build and test Java applications. [OpenJDK](#) builds are freely available for all platforms.

Maven

The Java application is built using Maven. Maven is available via [Chocolatey](#) for Windows, and it can be installed via the appropriate package manager for Linux.

Octopus Deploy

The installation file used to install Octopus Server on-premises can be downloaded from the [Octopus website](#).

Octopus requires a Microsoft SQL Server to host the database. Microsoft SQL Server Express can be downloaded for free from the [Microsoft website](#).

Complete the following steps to install Octopus:

1. Start the Octopus Installer, click **Next**, accept the Terms in the License Agreement and click **Next**.
2. Accept the default Destination Folder or choose a different location and click **Next**.
3. Click **Install**, and give the app permission to make changes to your device.
4. Click **Finish** to exit the installation wizard and launch the Getting started wizard to configure your Octopus Server.
5. Click **Get started...** and either enter your details to start a free trial of Octopus Deploy or enter your license key and click **Next**.
6. Accept the default Home Directory or enter a location of your choice and click **Next**.
7. Decide whether to use a **Local System Account** or a **Custom Domain Account**.

Learn more about the [permissions required for the Octopus Windows Service](#) or using a [Managed Service Account](#).

8. On the Database page, click the dropdown arrow in the Server Name field to detect the SQL Server database. Octopus will create the database for you which is the recommended process; however, you can also create your own database.
9. Enter a name for the database, and click **Next** and **OK** to create the database.

Be careful not to use the name of an existing database as the setup process will install Octopus into that pre-existing database.

10. Accept the default port and directory or enter your own. This guide assumes Octopus is listening on port 80. Click **Next**.

Octopus Deploy

13. When the installation has completed, click **Finish** to launch the Octopus Manager.

Tomcat

Tomcat is an open source Java application server. It can be downloaded from the [product homepage](#), installed via [Chocolatey](#), or installed via a Linux package manager.

Deployments to Tomcat are done via the Manager. This application is included in some Tomcat distributions, but may need to be installed separately. For example, Ubuntu distributes Tomcat and the Manager application as the **tomcat9** and **tomcat9-admin** packages.

A user account must be added to the Manager application to allow deployments to be performed. The following is an example of a **tomcat-users.xml** file that defines a user called **tomcat** that belongs to the roles **manager-script**, which allows access to the Manager REST API, and **manager-gui**, which allows access to the Manager web interface:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd">
    <role rolename="manager-script"/>
    <role rolename="manager-gui"/>
    <user username="tomcat" password="Password01!" roles="manager-script,manager-gui"/>
</tomcat-users>
```

Clone source code

The source code for the random quotes application is hosted in [GitHub](#). The code can be cloned from <https://github.com/OctopusSamples/RandomQuotes-Java.git> with the command:

```
git clone https://github.com/OctopusSamples/RandomQuotes-Java.git
```

Octopus API key

In order to allow GitHub Actions to communicate with Octopus, we need to generate an API key. This is done in the Octopus web portal. The web portal can be opened from the **Browse** link in the Octopus Manager:

Octopus Deploy

The screenshot shows the Octopus Web Portal interface. At the top, there's a banner with a link to 'Browse: http://localhost/' and a 'Change bindings...' button. Below this, the 'Octopus Windows Service' section shows the service 'OctopusDeploy' is running, with options to Start, Restart, Stop, or Reinstall. The 'Storage' section shows Octopus Deploy data is stored in SQL Server, with links to Export data..., Import data..., and View master key. Under 'Other tasks', there are links to Change proxy server settings... and Delete this Octopus instance... .

From the Octopus Deploy web portal, sign in, and view your profile:

The screenshot shows the Octopus Dashboard. The top navigation bar includes 'Default', 'Dashboard', 'Projects', 'Infrastructure', 'Tenants', 'Insights', 'Library', 'Tasks', 'Configuration', a search bar, a notification bell, and a user dropdown for 'admin'. A context menu is open over the 'Profile' item in the dropdown, showing options like 'Dark Theme', 'Sign Out', and 'Profile'. The main dashboard area has a 'Let's deploy your first software application' card with steps 1 through 3: Tell Octopus where to deploy your software, Define your deployment process, and Deploy your release. There are also 'Quick guide' and 'Video tutorials' links.

Go to the **API keys tab**. This lists any previous API keys that you have created. Click on **New API key**:

Octopus Deploy

The screenshot shows the 'My API Keys' section of the Octopus Deploy interface. A large blue octopus icon is in the top left. Below it, there's a message about changing your picture via Gravatar.com. On the left, a sidebar has 'My API Keys' selected and highlighted with a green box. The main area says 'There aren't currently any API keys to show.' In the bottom right corner of the main area, there's a callout box with the heading '1 Tell Octopus where to deploy your software' and two sub-points: 'Create your first environment' and 'Create your first deployment target (optional)'. A 'NEW API KEY' button is located at the top right of the main area.

Give the API key a name so that you remember what the key is for, and click **Generate New**:

This screenshot shows the 'Generate New API Key' dialog box overlaid on the Octopus Deploy interface. The dialog has a title 'Generate New API Key' and a sub-section 'Record the purpose of this key so that you can revoke it when no longer required.' It includes a dropdown for 'When should this API key expire?' with options 'Set expiry date' and 'Never (default)'. At the bottom are 'CANCEL' and 'GENERATE NEW' buttons. The 'My API Key' input field is highlighted with a green box.

Copy the new API key to your clipboard:

This screenshot shows the 'Generate New API Key' dialog box again, but now it displays a newly generated API key: 'API-J5TH2FSMHAERJCGZRKGRFAAAAV3F4CA'. This key is highlighted with a green box. The dialog also contains a note: 'Your new API key is: [key] API keys cannot be retrieved once they are created. Make sure you save this key in a safe place like a password management tool.' At the bottom are 'CLOSE' and 'COPY TO CLIPBOARD' buttons. The background shows the Octopus Deploy interface with the 'My API Keys' list.

Octopus Deploy

GitHub Actions supports Node.js, Python, Java, Ruby, PHP, Go, Rust, .NET, and more. Build, test, and deploy applications in your language of choice. In this tutorial, we rely on GitHub Actions to do the following:

- Clone the code from Git.
- Resolve and install any dependencies from Maven.
- Run unit tests.
- Package the application by bundling all the files it needs to run into a WAR file.
- Push the package to the Octopus (built-in) package repository.

At a high level, GitHub Actions workflows are YAML documents committed to a GitHub repository that define the [runner](#) that the steps are executed on, the [triggers](#) that execute the workflow, and the steps to be run.

GitHub Actions workflows have access to [encrypted secrets](#). Two secrets are defined called:

- **OCTOPUS_SERVER_URL**, which defines the URL of the Octopus server (for example, <https://myserver.octopus.app> for hosted Octopus instances)
- **OCTOPUS_API_TOKEN**, which defines the API key used to access the Octopus instance.

The YAML document below is saved to a file called **.github/workflows/build.yaml**. It is configured to be triggered on a git push and defines steps to build, test, package, and publish the sample application:

Octopus Deploy

```
steps:
  - uses: actions/checkout@v3
    with:
      fetch-depth: '0'
  - name: Install GitVersion
    uses: gittools/actions/gitversion/setup@v0.9.14
    with:
      versionSpec: 5.x
  - id: determine_version
    name: Determine Version
    uses: gittools/actions/gitversion/execute@v0.9.14
    with:
      additionalArguments: /overrideconfig mode=Mainline
  - name: Install Octopus Deploy CLI
    uses: OctopusDeploy/install-octopus-cli-action@v1
    with:
      version: latest
  - name: Set up JDK 1.17
    uses: actions/setup-java@v2
    with:
      java-version: '17'
      distribution: adopt
  - name: Set Version
    run: ./mvnw --batch-mode versions:set -DnewVersion=${{ steps.determine_version.outputs.semVer }}
    shell: bash
  - name: Test
    run: ./mvnw --batch-mode test
    shell: bash
  - if: always()
    name: Report
    uses: dorny/test-reporter@v1
    with:
      name: Maven Tests
      path: target/surefire-reports/*.xml
      reporter: java-junit
      fail-on-error: 'false'
  - name: Package
    run: ./mvnw --batch-mode -DskipTests=true package
    shell: bash
  - id: get_artifact
    name: Get Artifact Path
    run: |-
      # Find the largest WAR or JAR, and assume that was what we intended to build.
      echo "::set-output name=artifact::$(find target -type f \(-iname *.jar -o -iname *.war \) -printf "%p\n" |
sort -n | head -1)"
    shell: bash
  - id: get_artifact_name
    name: Get Artifact Name
    run: |-
      # Get the filename without a path
      path="${{ steps.get_artifact.outputs.artifact }}"
      echo "::set-output name=artifact::${path##*/}"
    shell: bash
  - id: get_octopus_artifact
    name: Create Octopus Artifact
    run: |-
      # Octopus expects artifacts to have a specific file format
      file="${{ steps.get_artifact.outputs.artifact }}"
      extension="${file##*.}"
      octofile="randomquotes.${{ steps.determine_version.outputs.semVer }}.${extension}"
      cp ${file} ${octofile}
      echo "::set-output name=artifact::${octofile}"
      # The version used when creating a release is the package id, colon, and version
```

Octopus Deploy

```
sneii: bash
- name: Push packages to Octopus Deploy 🚀
  uses: OctopusDeploy/push-package-action@v2
  env:
    OCTOPUS_API_KEY: ${{ secrets.OCTOPUS_API_TOKEN }}
    OCTOPUS_HOST: ${{ secrets.OCTOPUS_SERVER_URL }}
  with:
    overwrite_mode: OverwriteExisting
    packages: ${{ steps.get_octopus_artifact.outputs.artifact }}
name: Java Maven Build
'on':
  workflow_dispatch: {}
push: {}
```

There is a lot of work being performed by this workflow, so let's break it down.

The job called **build** is run on an Ubuntu runner:

```
jobs:
  build:
    runs-on: ubuntu-latest
```

The first step checks out the source code from the git repository. Setting the **fetch-depth** option to **0** means all the history for all branches and tags is fetched. This is required by the GitVersion action used in later steps:

```
steps:
- uses: actions/checkout@v3
  with:
    fetch-depth: '0'
```

GitVersion builds meaningful version strings based on the commits to git.

This step installs GitVersion:

```
- name: Install GitVersion
  uses: gittools/actions/gitversion/setup@v0.9.14
  with:
    versionSpec: 5.x
```

This step calls GitVersion which outputs many different variables representing different parts of a SemVer version string generated from the commits to the git repository. Later steps use these output values as the version of the package generated by this workflow:

```
- id: determine_version
  name: Determine Version
  uses: gittools/actions/gitversion/execute@v0.9.14
  with:
    additionalArguments: /overrideconfig mode=Mainline
```

This step installs the Octopus CLI, which is used by the subsequent Octopus GitHub Actions:

Octopus Deploy

```
version: latest
```

This step sets up Java 17:

```
- name: Set up JDK 1.17
uses: actions/setup-java@v2
with:
  java-version: '17'
  distribution: adopt
```

This step sets the project version to the SemVer version determined by GitVersion:

```
- name: Set Version
run: ./mvnw --batch-mode versions:set -DnewVersion=${{ steps.determine_version.outputs.semVer }}
shell: bash
```

This step runs the unit tests:

```
- name: Test
run: ./mvnw --batch-mode test
shell: bash
```

This step publishes the results of the test against the workflow run. The **if** parameter is set to **always()** to ensure that the results are published even if the tests fail:

```
- if: always()
name: Report
uses: dorny/test-reporter@v1
with:
  name: Maven Tests
  path: target/surefire-reports/*.xml
  reporter: java-junit
  fail-on-error: 'false'
```

This step packages the application, skipping tests, as those have already been run:

```
- name: Package
run: ./mvnw --batch-mode -DskipTests=true package
shell: bash
```

This step finds the largest war or jar file in the **target** directory. We assume the largest file is the one we wish to deploy:

```
- id: get_artifact
name: Get Artifact Path
run: |-
  # Find the largest WAR or JAR, and assume that was what we intended to build.
  echo "::set-output name=artifact::$(find target -type f \(
    -iname *.jar -o -iname *.war \
  \) -printf "%p\n" |
  sort -n | head -1)"
  shell: bash
```

Octopus Deploy

```

name: Get Artifact Name
run: |-
  # Get the filename without a path
  path="${{ steps.get_artifact.outputs.artifact }}"
  echo "::set-output name=artifact::${path##*/}"
shell: bash
  
```

Java packages are saved with filenames like **randomquotes-1.0.0.war**, with a dash between the package name and the version.

Octopus requires file names like **randomquotes.1.0.0.war**, with a period between the package name and version.

This step builds two output variables. The first, **artifact**, is the Java package filename reformatted to conform to the format expected by Octopus. The second, **octoversion**, is the package ID and version in a format expected by the create release step (which will be defined later):

```

- id: get_octopus_artifact
  name: Create Octopus Artifact
  run: |-
    # Octopus expects artifacts to have a specific file format
    file="${{ steps.get_artifact.outputs.artifact }}"
    extension="${file##*.}"
    octofile="randomquotes.${{ steps.determine_version.outputs.semVer }}.${extension}"
    cp ${file} ${octofile}
    echo "::set-output name=artifact::${octofile}"
    # The version used when creating a release is the package id, colon, and version
    octoversion="randomquotes:${{ steps.determine_version.outputs.semVer }}"
    echo "::set-output name=octoversion::${octoversion}"
  shell: bash
  
```

The package files are then pushed to Octopus:

```

- name: Push packages to Octopus Deploy 🚀
  uses: OctopusDeploy/push-package-action@v2
  env:
    OCTOPUS_API_KEY: ${{ secrets.OCTOPUS_API_TOKEN }}
    OCTOPUS_HOST: ${{ secrets.OCTOPUS_SERVER_URL }}
  with:
    overwrite_mode: OverwriteExisting
    packages: ${{ steps.get_octopus_artifact.outputs.artifact }}
  
```

The workflow is triggered on each push to the git repository. The workflow can also be triggered manually with the **workflow_dispatch** event:

```

name: Java Maven Build
'on':
  workflow_dispatch: {}
  push: {}
  
```

Commit the file and push it to the GitHub repository with the command:

```
git add .; git commit -m "Added workflow file"; git push
```

GitHub Actions then runs the workflow.

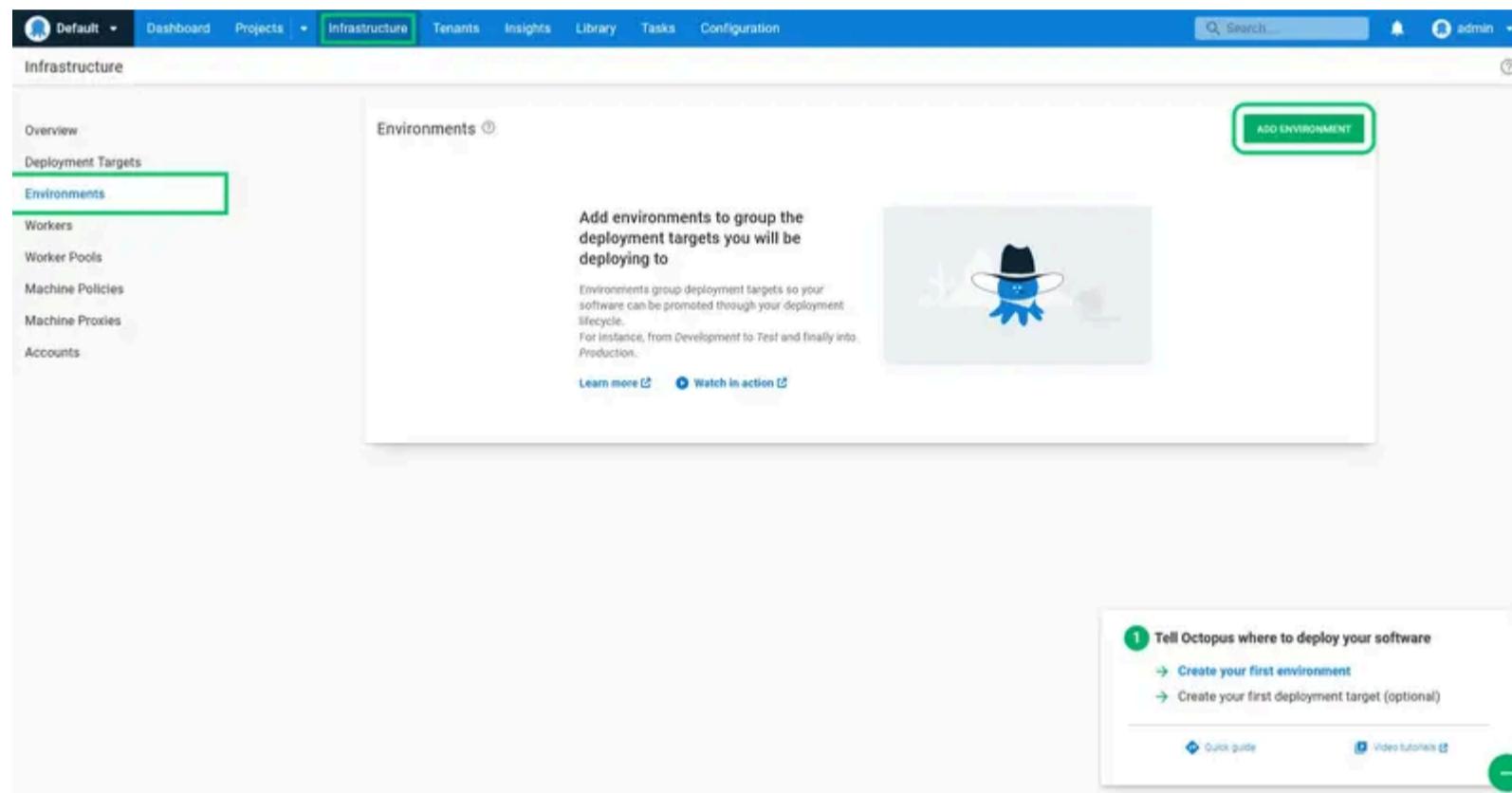
Octopus Deploy

Now that GitHub Actions has successfully built the application, we need to configure Octopus to deploy it into our environments.

Create the environments

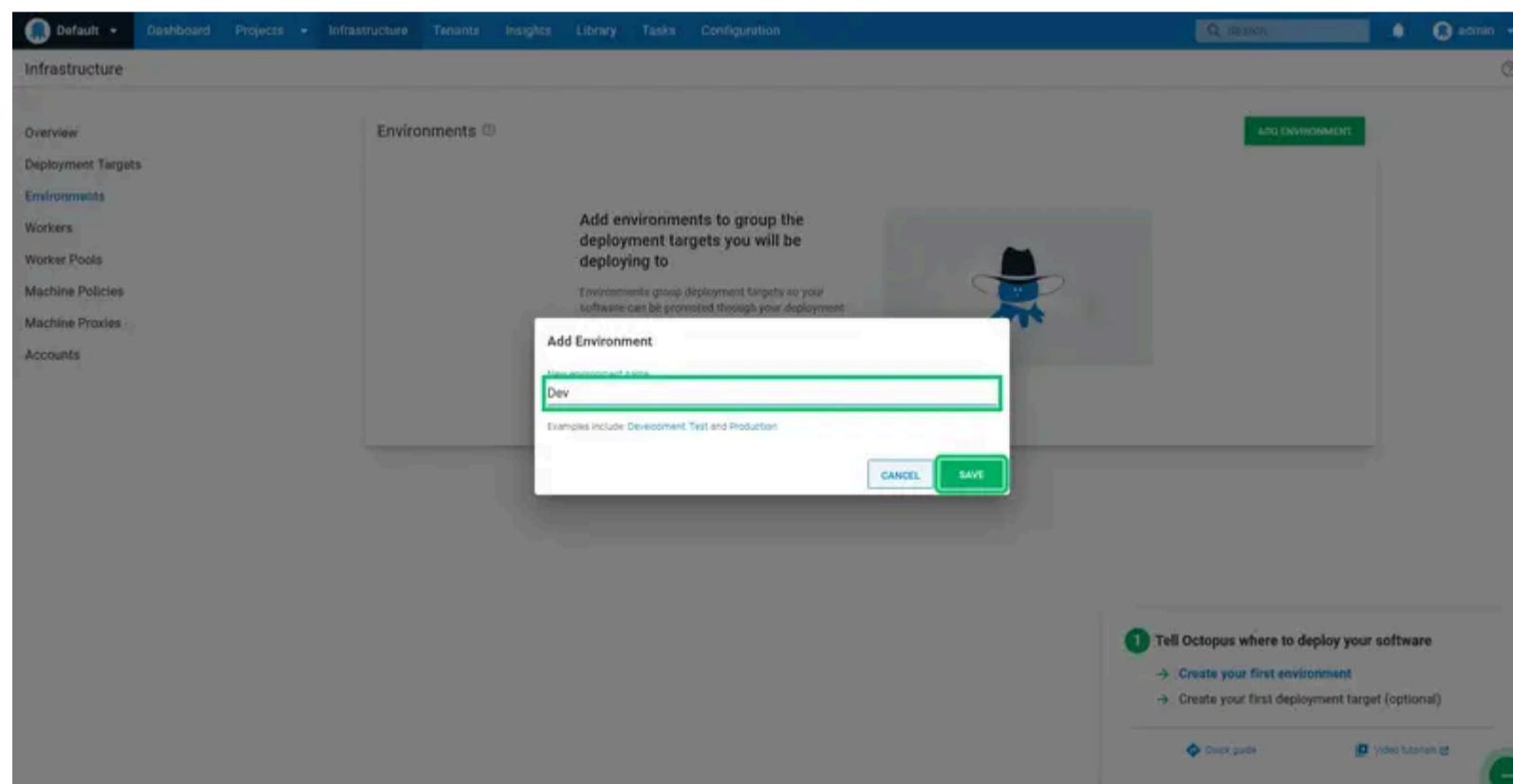
Environments represent the stages that a deployment must move through as part of the deployment pipeline. We'll create three environments: **Dev**, **Test**, and **Prod**.

Log into Octopus, and click the **Infrastructure** link, then the **Environments** link, and click **ADD ENVIRONMENT**:



The screenshot shows the Octopus Deploy interface. The top navigation bar includes 'Default', 'Dashboard', 'Projects', 'Infrastructure' (which is highlighted), 'Tenants', 'Insights', 'Library', 'Tasks', and 'Configuration'. On the far right, there's a search bar, a bell icon, and a user profile 'admin'. Below the navigation is a sidebar with links: 'Overview', 'Deployment Targets', 'Environments' (which is also highlighted), 'Workers', 'Worker Pools', 'Machine Policies', 'Machine Proxies', and 'Accounts'. The main content area is titled 'Environments' and contains a sub-section 'Add environments to group the deployment targets you will be deploying to'. It includes a description about environments grouping deployment targets for promotion through the lifecycle, from Development to Test and finally into Production. There are 'Learn more' and 'Watch in action' buttons. At the top right of this section is a green 'ADD ENVIRONMENT' button. To the right of the main content, there's a sidebar with a step-by-step guide: '1 Tell Octopus where to deploy your software' with links to 'Create your first environment' and 'Create your first deployment target (optional)'. At the bottom of this sidebar are 'Quick guide' and 'Video tutorials' buttons.

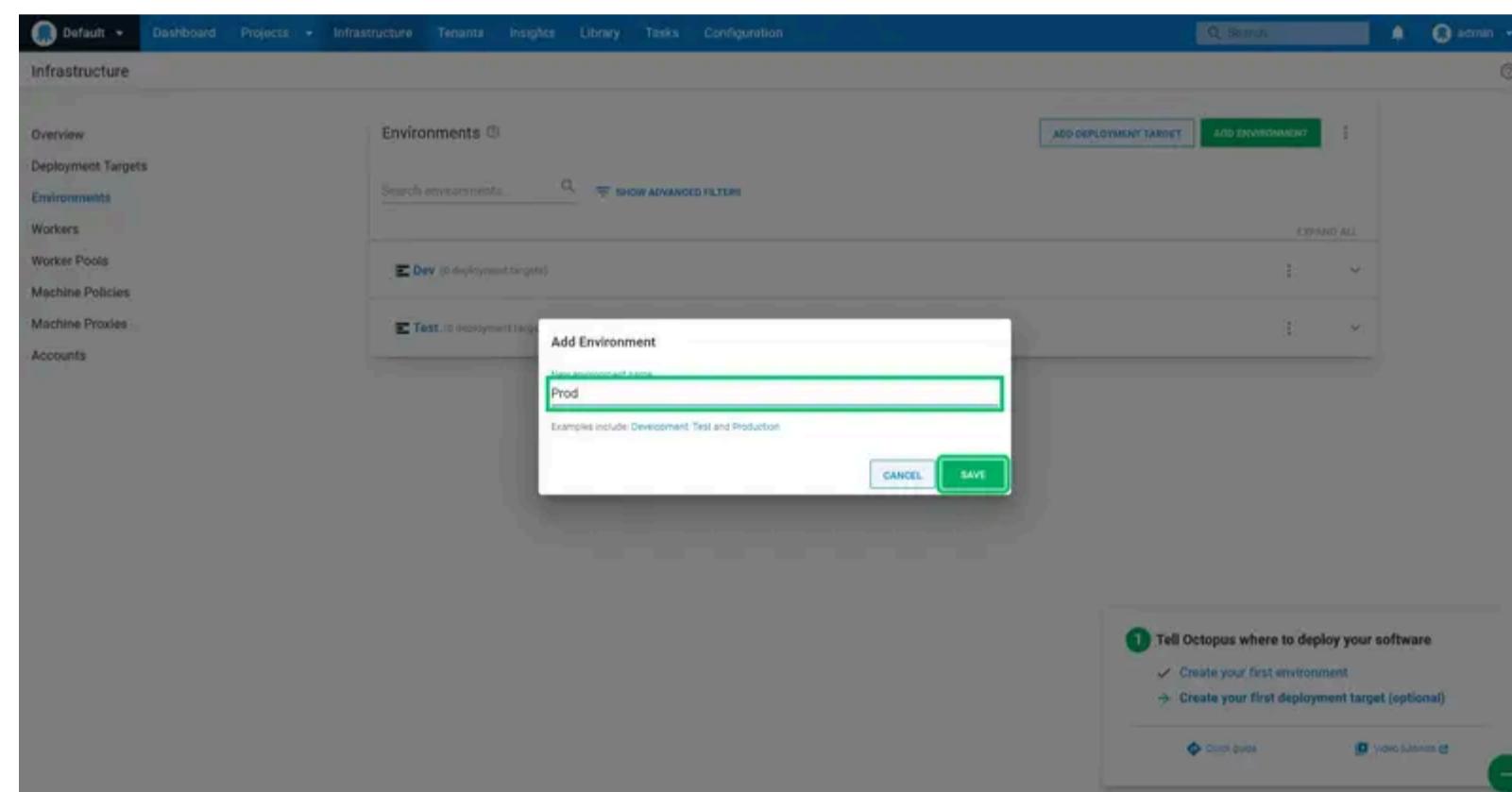
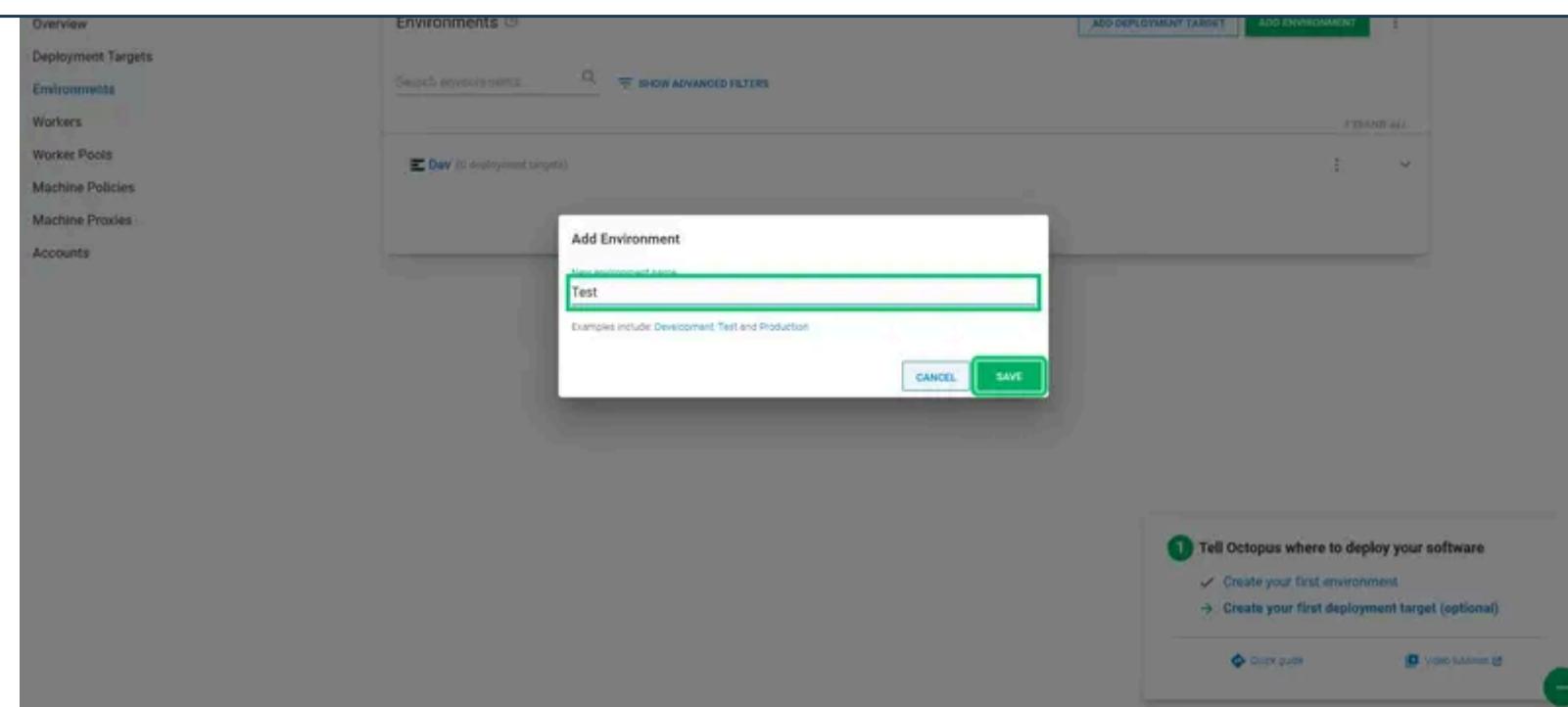
Enter **Dev** as the New environment name, and click **SAVE**:



This screenshot shows the same Octopus Deploy interface as the previous one, but with a modal dialog box in the foreground. The dialog is titled 'Add Environment' and has a single input field labeled 'New environment name' containing the value 'Dev'. Below the input field is a note: 'Examples include: Development, Test and Production.' At the bottom of the dialog are two buttons: 'CANCEL' and 'SAVE', with 'SAVE' being the active button and having a green border. The background of the main interface is dimmed.

Repeat the process for the **Test** and **Prod** environments:

Octopus Deploy

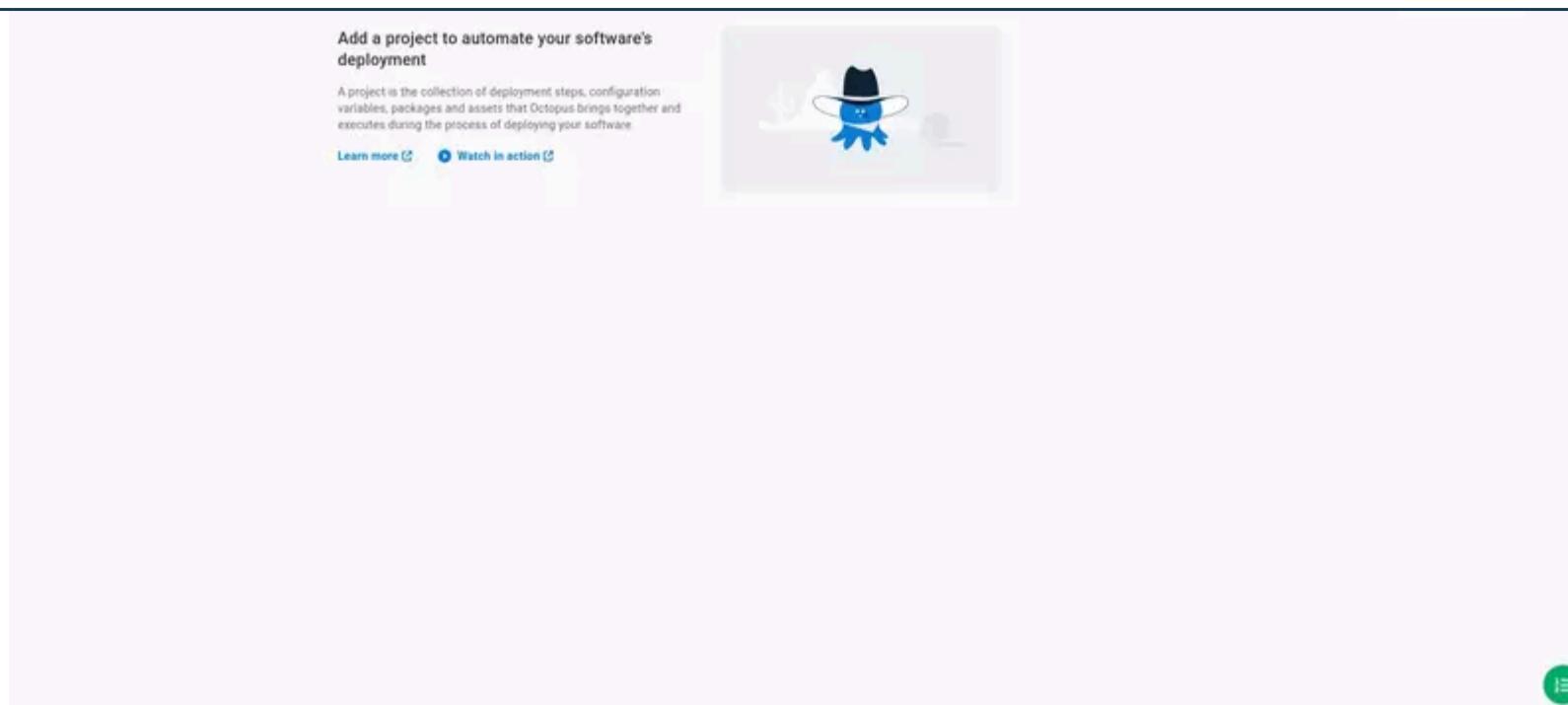


Create the Octopus deployment project

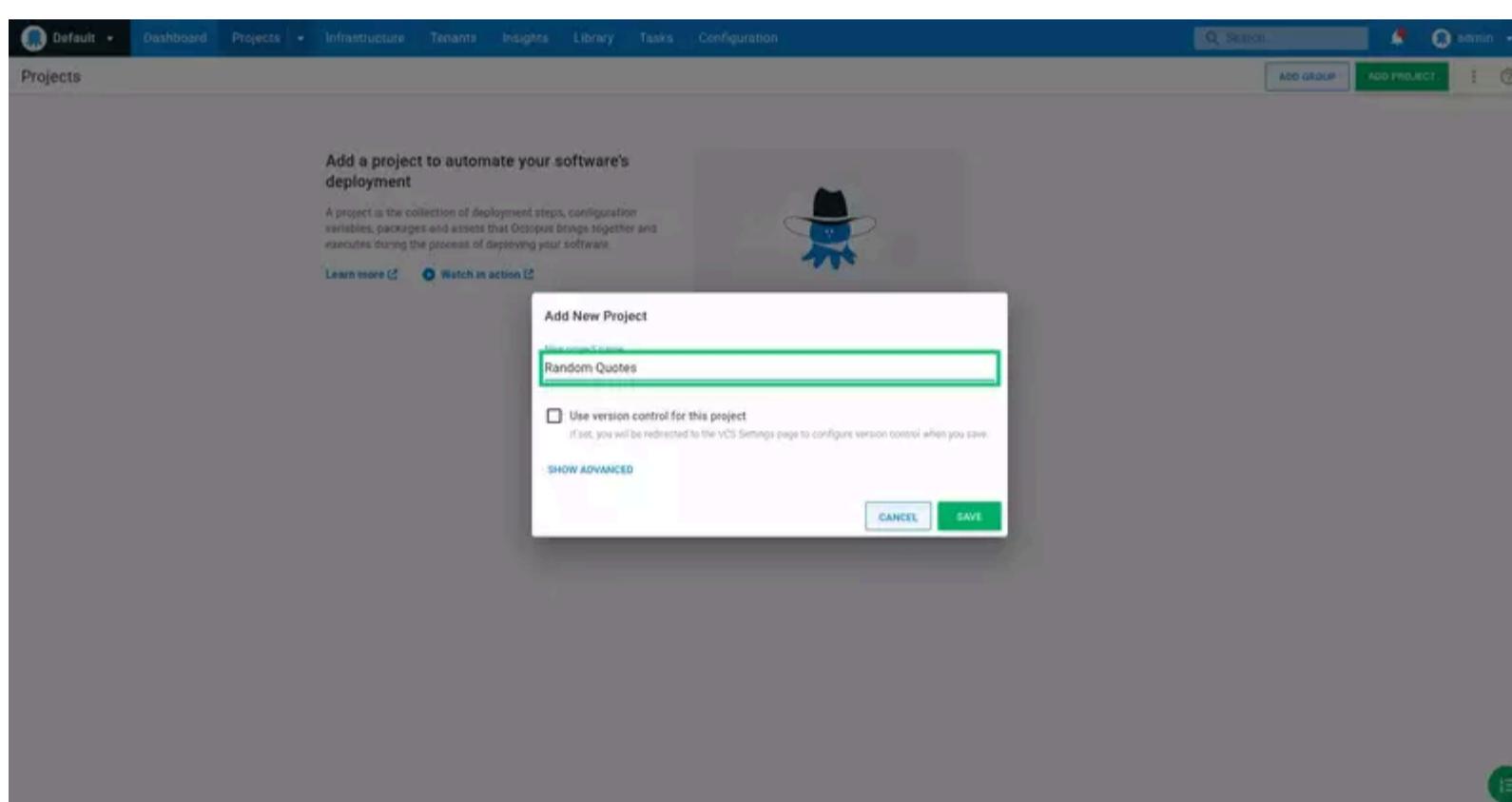
With the environments defined and a target created, we now need to create a deployment project in Octopus.

Log into Octopus, click the **Projects** link, and click **ADD PROJECT**:

Octopus Deploy



Enter **Random Quotes** for the project name, and click **SAVE**:



We start by defining a variable that will be injected into the **application.yml**. Click the **Variables** link:

The screenshot shows the "Project Variables" page for the "Random Quotes" project. On the left sidebar, under the "Variables" section, the "Project" link is highlighted with a green border. The main area displays a table for managing variables, with a single row visible: "Name" (Enter new variable), "Value" (Enter value), and "Scope" (Define scope). Buttons for "SAVE" and "ADD ANOTHER VALUE" are at the bottom right of the table.

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with navigation links: Deployments (Overview, Process, Channels, Releases, Triggers, Settings), Operations (Overview, Runbooks, Triggers), Variables (Project, Library Sets, All, Preview), Tasks, Insights (Overview, Deployment frequency, Deployment lead time, Deployment failure rate, Mean time to recovery), and Settings (General, Version Control). The main area is titled 'Variables' and shows a table with one row. The first column is 'Enter new variable', the second is 'Enter value', and the third is 'Define scope'. The row contains 'spring:profiles:active' in the first column, '#{{Octopus.Environment.Name}}' in the second, and a 'Define scope' button in the third. There are also 'ADD ANOTHER VALUE' and 'ADD TO LIST' buttons. A green circular icon with three horizontal lines is in the bottom right.

We will now define the deployment process. Click the **Deployments** link, click **Overview**, and then click **DEFINE YOUR DEPLOYMENT PROCESS**:

The screenshot shows the Octopus Deploy interface with the 'Deployments' and 'Overview' links highlighted. The main content area has a large 'CREATE PROCESS' button at the top, which is also highlighted with a green box. To its left is a small blue octopus icon wearing a black hat. Below the button, there's a brief description of what a project is. Further down, there are 'Docs' and 'Watch in action' links, and a 'CREATE PROCESS' button. The sidebar on the left is identical to the one in the previous screenshot.

Click **ADD STEP**:

Octopus Deploy

Random Quotes

CREATE RELEASE

Deployments

- Overview
- Process**
- Channels
- Releases
- Triggers
- Settings

Operations

- Overview
- Runbooks
- Triggers

Variables

- Project
- Library Sets
- All
- Preview

Tasks

Insights

Process

Process Editor

Choose Step Template

Filter by name, category or description...

What type of step do you want to add?

Script	Package	Google Cloud	AWS	Azure	Certificate	Docker	Java App Server	Kubernetes	Terraform	Windows Server
Atlassian	...	Built-in Steps	Community Steps							

ADD STEP **SAVE** **⋮**

Enter **Tomcat** into the search box:

Default Project Group

Dashboard **Projects** **Infrastructure** **Tenants** **Insights** **Library** **Tasks** **Configuration**

Search...

admin

Random Quotes

CREATE RELEASE

Deployments

- Overview
- Process**
- Channels
- Releases
- Triggers
- Settings

Operations

- Overview
- Runbooks
- Triggers

Variables

- Project
- Library Sets
- All
- Preview

Tasks

Insights

Process

Process Editor

Choose Step Template

Tomcat

Installed Step Templates (3)

- Start/Stop App in Tomcat
- Deploy a certificate to Tomcat
- Deploy to Tomcat via Manager

by Octopus Deploy •

Community Contributed Step Templates (1)

- Undeploy Tomcat Application via Manager

twerthi •

ADD STEP **SAVE** **⋮**

Click **ADD** on the **Deploy to Tomcat via Manager** tile:

Octopus Deploy

The screenshot shows the 'Step Templates' section of the Octopus Deploy interface. On the left is a sidebar with navigation links for Deployments, Operations, Variables, Tasks, Insights, and Settings. The main area displays 'Installed Step Templates (3)' and 'Community Contributed Step Templates (1)'. The 'Deploy to Tomcat via Manager' template is highlighted with a green border. It has a small icon of a blue cat, a brief description, and an 'ADD' button.

We want to take advantage of a feature in Octopus that will replace values in the application.yml file, which will set the Spring profile to the name of the environment that is being deployed to. To do this, we enable a feature on the deployment.

Click **CONFIGURATION FEATURES:**

The screenshot shows the 'Process Editor' for a deployment step. The sidebar on the left is identical to the previous one. The main area shows a step named 'Deploy to Tomcat via Manager'. It includes sections for 'Execution Location' (set to 'Where will Octopus run this step?'), 'On Targets in Roles' (with a note about target roles), and 'Package Details' (with a note about packages). The 'ADD STEP' button at the top right is highlighted with a green border.

Enable the **Structured Configuration Variables** option, and click **OK**:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with 'Random Quotes', 'CREATE RELEASE', 'Deployments' (selected), 'Process' (highlighted with a green border), 'Channels', 'Releases', 'Triggers', 'Settings', 'Operations', 'Variables', 'Project', 'Library Sets', 'All', 'Preview', 'Tasks', and 'Insights'. The main area is titled 'Process Editor' and shows a step named '1. Deploy to Tomcat via Manager'. A modal window titled 'Enabled Features' is open, listing three options: 'Custom Deployment Scripts', 'Structured Configuration Variables' (which is checked and highlighted with a green border), and 'Substitute Variables in Templates'. At the bottom of the modal are 'CANCEL' and 'OK' buttons. Below the modal, there's a section for 'On Targets in Roles' with a note about target roles and a 'CONFIGURE A ROLLING DEPLOYMENT' button.

Enter Deploy to Tomcat as the Step Name:

This screenshot shows the same Octopus Deploy interface as the previous one, but the 'Step Name' field in the '1. Deploy to Tomcat' step is now filled with 'Deploy to Tomcat' and has a green border around it. The rest of the interface remains the same, including the sidebar and the configuration details for the deployment step.

Enter web as the Role:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with navigation links for Random Quotes, Deployments (with 'Overview' selected), Variables, Operations, Tasks, and Insights. The main area is titled 'Process Editor' and shows a step named '1. Deploy to Tomcat'. The 'Step Name' is 'Deploy to Tomcat'. Under 'Execution Location', it says 'This step will run on each deployment target' and shows a diagram of a central target with five arrows pointing to five separate targets. Under 'On Targets in Roles', it says 'Target roles are used as tags to select the deployment targets this step will execute on. Deployment targets are configured in Infrastructure.' A dropdown menu shows 'Runs on targets in roles (type to add new)' with 'web' selected. At the bottom, there's a 'Package Details' section with a 'Package' field containing 'Octopus Server (built-in)' and a 'Package ID' field containing 'randomquotes'.

Select the **randomquotes** package:

This screenshot shows the same Octopus Deploy interface as above, but the 'Process' tab is selected in the sidebar. The 'Deployment' section has 'Overview' selected. The main process editor shows a step with 'Execution Location' set to 'This step will run on each deployment target' and 'On Targets in Roles' set to 'Runs on targets in roles (type to add new)'. In the 'Package Details' section, the 'Package' field is set to 'Octopus Server (built-in)' and the 'Package ID' field is set to 'randomquotes'.

Enter the path to the Tomcat Manager application. The URL will be something like <http://localhost:9091/manager>.

Provide the credentials for the user account that has been assigned the **manager-script** role in the `tomcat-users.xml` file. See the [Prerequisites](#) for more details.

Octopus Deploy

The screenshot shows the Octopus Deploy interface with a deployment step configuration. On the left, there's a sidebar with 'Library Sets' (All, Preview), 'Tasks', 'Insights' (Overview, Deployment frequency, Deployment lead time, Deployment failure rate, Mean time to recovery), and 'Settings' (General, Version Control). The main area shows a 'Run on target instances (type to add new)' section with a note: 'This step will run on all deployment targets with these roles.' Below it is a 'CONFIGURE A ROLLING DEPLOYMENT' section. Under 'Package Details', there's a 'Package' section with a note: 'This step is used to deploy a package to one or more machines which may be sourced from an external feed or the Octopus built-in feed. You can configure the remote machines to deploy to in the Infrastructure tab.' It shows 'Octopus Server (built-in)' selected. Under 'Tomcat deployment feature', there's a 'Tomcat Details' section with fields for 'Tomcat Manager URL' (set to 'http://localhost:9091/manager'), 'Management user' (set to 'tomcat'), and 'Management password' (set to '*****'). A green box highlights the 'Tomcat Manager URL' field.

Enter **randomquotes-#{Octopus.Environment.Name | ToLower}** as the **Context path**. This ensures that the application is deployed to a unique path for every environment. This is how we can use a single Tomcat server to host multiple environments:

The screenshot shows the Octopus Deploy interface with a deployment step configuration. The sidebar is identical to the previous screenshot. The main area shows a 'Process Editor' section with a 'Process' tab. Under 'Package', there's a note: 'This step is used to deploy a package to one or more machines which may be sourced from an external feed or the Octopus built-in feed. You can configure the remote machines to deploy to in the Infrastructure tab.' It shows 'Octopus Server (built-in)' selected. Under 'Tomcat deployment feature', there's a 'Tomcat Details' section with fields for 'Tomcat Manager URL' (set to 'http://localhost:9091/manager'), 'Management user' (set to 'tomcat'), and 'Management password' (set to '*****'). A green box highlights the 'Tomcat Manager URL' field. Below it is a 'Tomcat Deployment' section with a 'Context path' field containing 'randomquotes-#{Octopus.Environment.Name | ToLower}'. A green box highlights the 'Context path' field.

In the **Structured Configuration Variables** section, enter ****/application.yml** as the **Target files** value. Click **Save**.

The slash in the filename is operating system specific. The pattern ****/application.yml** (with a forward slash) is for Linux targets, while ****\application.yml** (with a backslash) is for Windows targets.

The contents of the application.yml file is shown below. The **spring:profiles:active** variable we defined earlier will overwrite the Spring active profile in this YAML file:

Octopus Deploy

```

spring:
  profiles:
    active: Dev
  h2:
    console:
      enabled: true
  jpa:
    database-platform: org.hibernate.dialect.H2Dialect
  datasource:
    url: jdbc:h2:mem:testdb
    dbcp2:
      driver-class-name: org.h2.Driver
  flyway:
    locations: classpath:db/migration/{vendor}

```

The screenshot shows the Octopus Deploy interface for creating a deployment process. The top navigation bar includes 'Default', 'Dashboard', 'Projects', 'Infrastructure', 'Tenants', 'Insights', 'Library', 'Tasks', and 'Configuration'. A search bar and user 'admin' are also present.

The main area is titled 'Process' and 'Process Editor'. It contains sections for 'Tomcat Deployment' and 'Advanced Options' under 'Tomcat Deployment' settings, and 'Structured Configuration Variables'.

- Tomcat Deployment:** Set 'Context path' to `randomquotes-#{Octopus.Environment.Name | ToLower}`. A tooltip explains: "This field defines the context path of the deployed artifact. For example, setting this field to `myapp` will result in the deployment being having the context path `/myapp` in Tomcat. Set the value to `/` to deploy to the root context."
- Advanced Options:** Set 'Deployment version' to `#()`. A tooltip states: "Tomcat versions are used with parallel deployments." It also notes: "Leave this field blank to leave the Tomcat version undefined, in which case you will overwrite any existing deployment with the same context path, and parallel deployments will not be enabled." Another note says: "Alternatively you can define a custom Tomcat version. These versions are compared as plain strings, meaning traditional version strings like `1.0.1` may not be suitable as this format is not guaranteed to correctly identify the latest version using string comparisons. A date string like `#{} | NowDateUtc "yyMMddHHmmss"` is a good option, as this format produces the correct result when compared as strings."
- Structured Configuration Variables:** Set 'Target file' to `**/application.yml`.

Deploy!

Now we have a deployment project in Octopus ready to deploy our Java application to our **Dev**, **Test**, and **Prod** environments. The next step is to create and deploy a release.

Click **CREATE RELEASE**.

The release creation screen provides an opportunity to review the packages that will be included and to add any release notes. By default, the latest package is selected automatically. Click **SAVE**:

Octopus Deploy

This screen allows you to select the environment that will be deployed into. Lifecycles can be used to customize the progression of deployments through environments (this is [demonstrated later in the guide](#)), however, we will accept the default option to deploy to the **Dev** environment by clicking **DEPLOY TO DEV...**:

Click **DEPLOY** to deploy the application into the **Dev** environment:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with sections like Random Quotes, Deployments (with 'Overview' highlighted), Releases, Operations, Variables, Tasks, and Insights. The main area is titled 'Release 0.0.1' and shows a deployment configuration for 'Deploy release 0.0.1'. It includes settings for environments (set to 'Dev'), deployment time ('Now (default)'), excluded steps, failure mode, package download, and preview customization. A large green 'DEPLOY' button is at the top right.

The application is then deployed:

The screenshot shows the task log for the deployment of 'Release 0.0.1'. The task summary indicates a successful deployment to the 'Dev' environment. The task log details the steps taken: 'Deploy Random Quotes release 0.0.1 to Dev', 'Acquire packages', and 'Step 1: Deploy to Tomcat'. The log entries show the process of acquiring packages from cache and deploying the application to Tomcat. A success message at the end congratulates the user on completing their first deployment, accompanied by a cartoon character icon.

Congratulations! You have now built and deployed your first application. Visit <http://localhost:9091/randomquotes-dev> in a browser (replacing **localhost:9091** with the IP address and port of your Tomcat instance) to view a random quote. Note the context path matches the templated value of the context path that we deployed the application to. Ensuring each environment deploys to a unique context path is how we can use a single Tomcat instance to host multiple environments:

Octopus Deploy

Simplicity is not the goal. It is the by-product of a good idea and modest expectations.

— Paul Rand

Refresh

© 2018 - Random Quotes | Version 1.0.1 running in Dev
Quote count: 2

Octopus Deploy

The process of deploying a successful build to the **Dev** environment is currently a manual one; GitHub Actions pushes the file to Octopus, and we must trigger the initial deployment to the **Dev** environment from within Octopus. Typically though, deployments to the **Dev** environment will be performed automatically if a build and all of its tests succeed.

To trigger the initial deployment to the **Dev** environment after a successful build, we will go back to the project we created in GitHub Actions and add an additional step to create an Octopus release and then deploy it to the **Dev** environment.

When added to the workflow file, the YAML below creates a new release in Octopus with each build and deploys it to the development environment.

Note that the package versions are explicitly defined via the output variable called **octoversion** generated in previous steps:

```
- name: Create Octopus Release
  uses: OctopusDeploy/create-release-action@v1.1.1
  with:
    api_key: ${{ secrets.OCTOPUS_API_TOKEN }}
    project: Random Quotes
    server: ${{ secrets.OCTOPUS_SERVER_URL }}
    deploy_to: Dev
    packages: ${{ steps.get_octopus_artifact.outputs.octoversion }}
```

Octopus Deploy

Now we will explore some of the more advanced features of Octopus that allow us to customize the deployment progression through environments, secure deployments to production environments, add deployment sign offs, view the audit logs, and add notifications.

Lifecycles

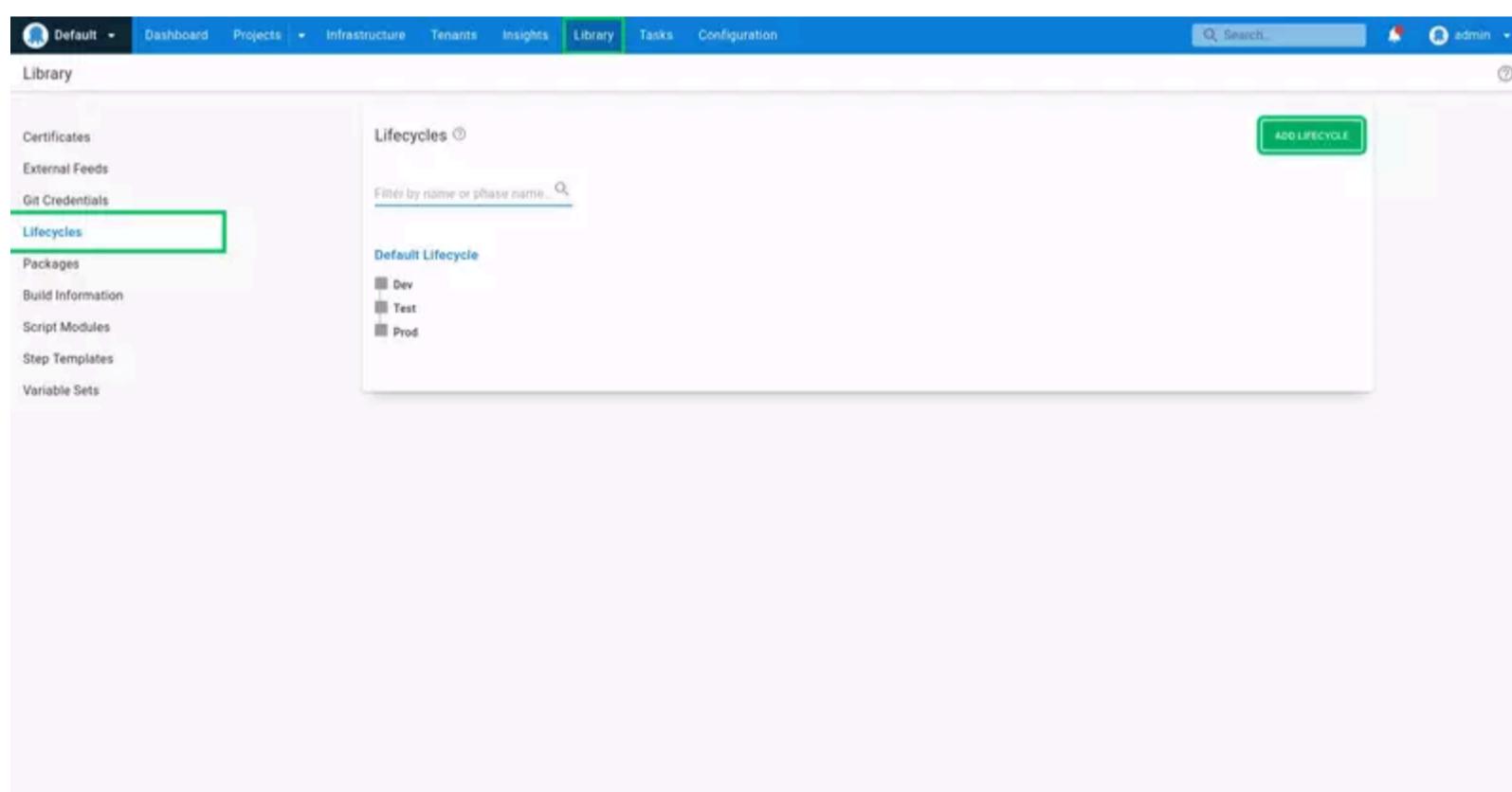
Our project currently uses the default lifecycle, which defines a progression through all the environments in the order they were created.

A custom lifecycle allows the progression of a deployment to be further refined, by defining only a subset of environments that can be deployed to, allowing some environments to be skipped entirely, or requiring that a minimum number of environments are successfully deployed to before moving onto the next environment.

Here we will create a custom lifecycle that makes deployments to the **Dev** environment optional. This means that initial deployments can be made to the **Dev or Test** environments, but a successful deployment *must* be made to the **Test** environment before it can be progressed to the **Prod** environment.

Skipping the **Dev** environment like this may be useful for promoting a release candidate build directly to the **Test** environment for product owners or testers to review.

Click the **Library** link, click the **Lifecycles** link, and click **ADD LIFECYCLE**:



Set the lifecycle name to **Dev, Test, and Prod**, and the description to **Progression from the Dev to the Prod environments**:

Octopus Deploy

DETAILS

Name Enter a name for your lifecycle.

Description Enter a description for your lifecycle.

Retention Policy Change the retention policy.

How long should we keep releases?
 Keep all (default)
 Keep a limited number

How long should we keep extracted packages and files on disk on Tentacles?
 Keep all (default)
 Keep a limited number

Retention policies dictate how long releases and deployments are kept for. For more information please see [retention policies documentation](#).

Phases

This lifecycle will use the default conventions. They allow deployment to any environment, so long as environments are deployed in the order that they are defined on the [environments](#) page. Use the Add phase button to explicitly define your own phases or to restrict the lifecycle to specific environments.

Phases are used to group environments that can accept a deployment. Simple lifecycles, such as the lifecycle we are creating, have a one-to-one relationship between phases and environments.

Click **ADD PHASE**:

Phases

This lifecycle will use the default conventions. They allow deployment to any environment, so long as environments are deployed in the order that they are defined on the [environments](#) page. Use the Add phase button to explicitly define your own phases or to restrict the lifecycle to specific environments.

ADD PHASE

Enter **Dev** as the phase name, and select the **Optional phase** option. This means that deployments can skip this phase and any environments defined in it, and deploy directly to the next phase.

Because we are mapping each environment to its own phase, the name of the phase matches the name of the environment:

Octopus Deploy

The screenshot shows the 'Create Lifecycle' page. In the 'Phases' section, there is a table with one row. The first column is labeled 'Phase 1' and contains the placeholder text 'Please enter phase details'. The second column is labeled 'Phase name' and contains the value 'Dev'. A green rectangular box highlights the 'Phase name' input field.

Click ADD ENVIRONMENT:

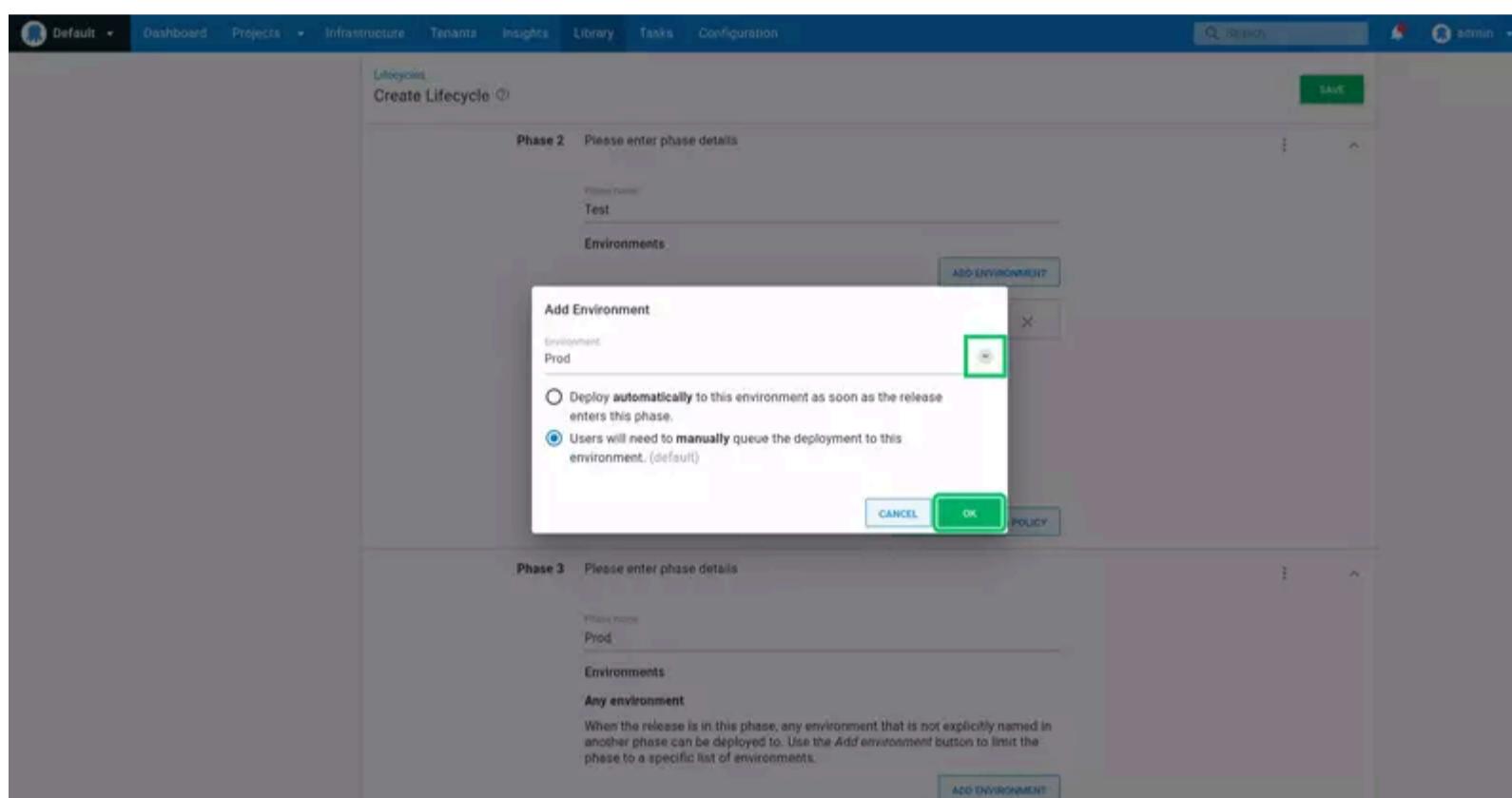
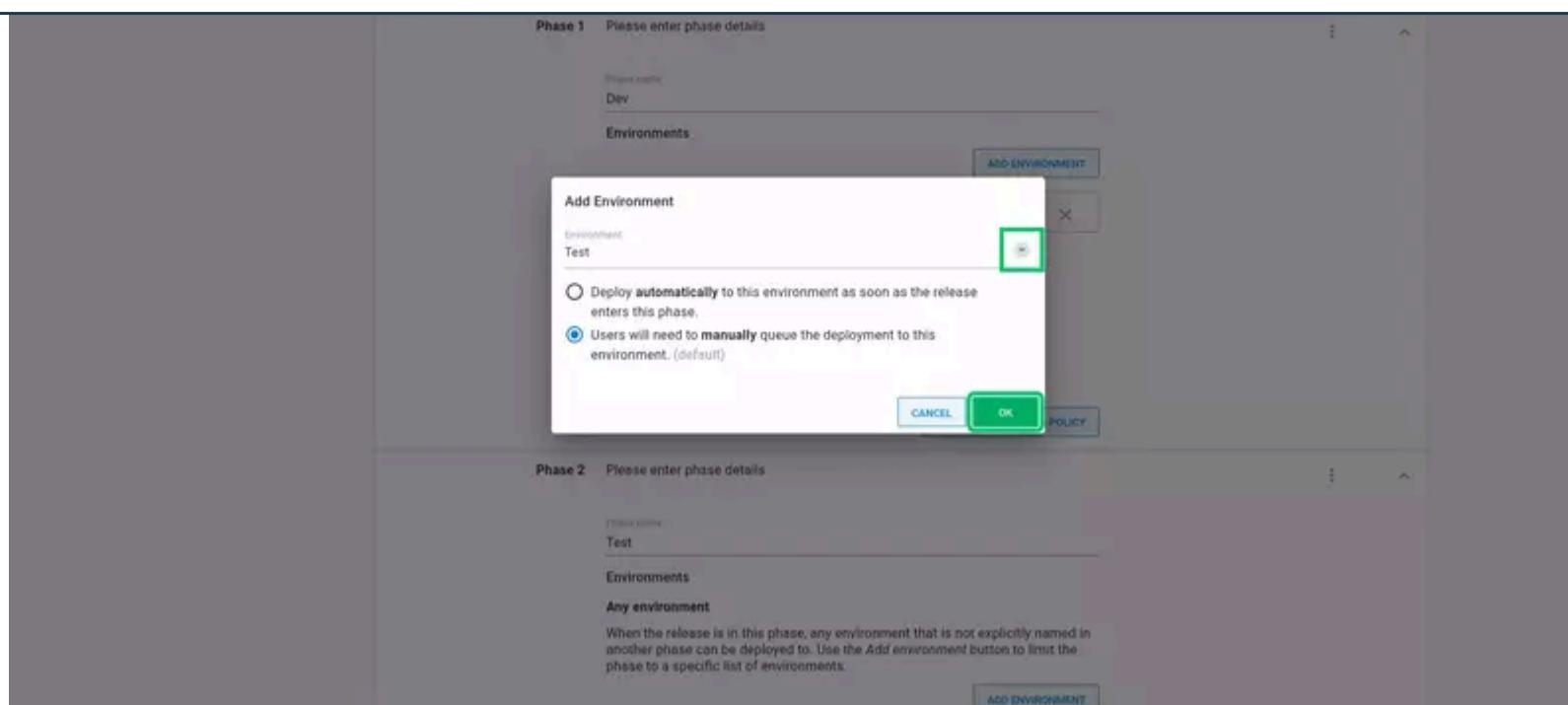
A modal window titled 'Add Environment' is displayed over the main form. It has two options: 'Deploy automatically to this environment as soon as the release enters this phase.' (unchecked) and 'Users will need to manually queue the deployment to this environment. (default)' (checked). A green rectangular box highlights the 'OK' button at the bottom right of the modal.

Click the dropdown arrow, select the **Dev environment, and click **OK**:**

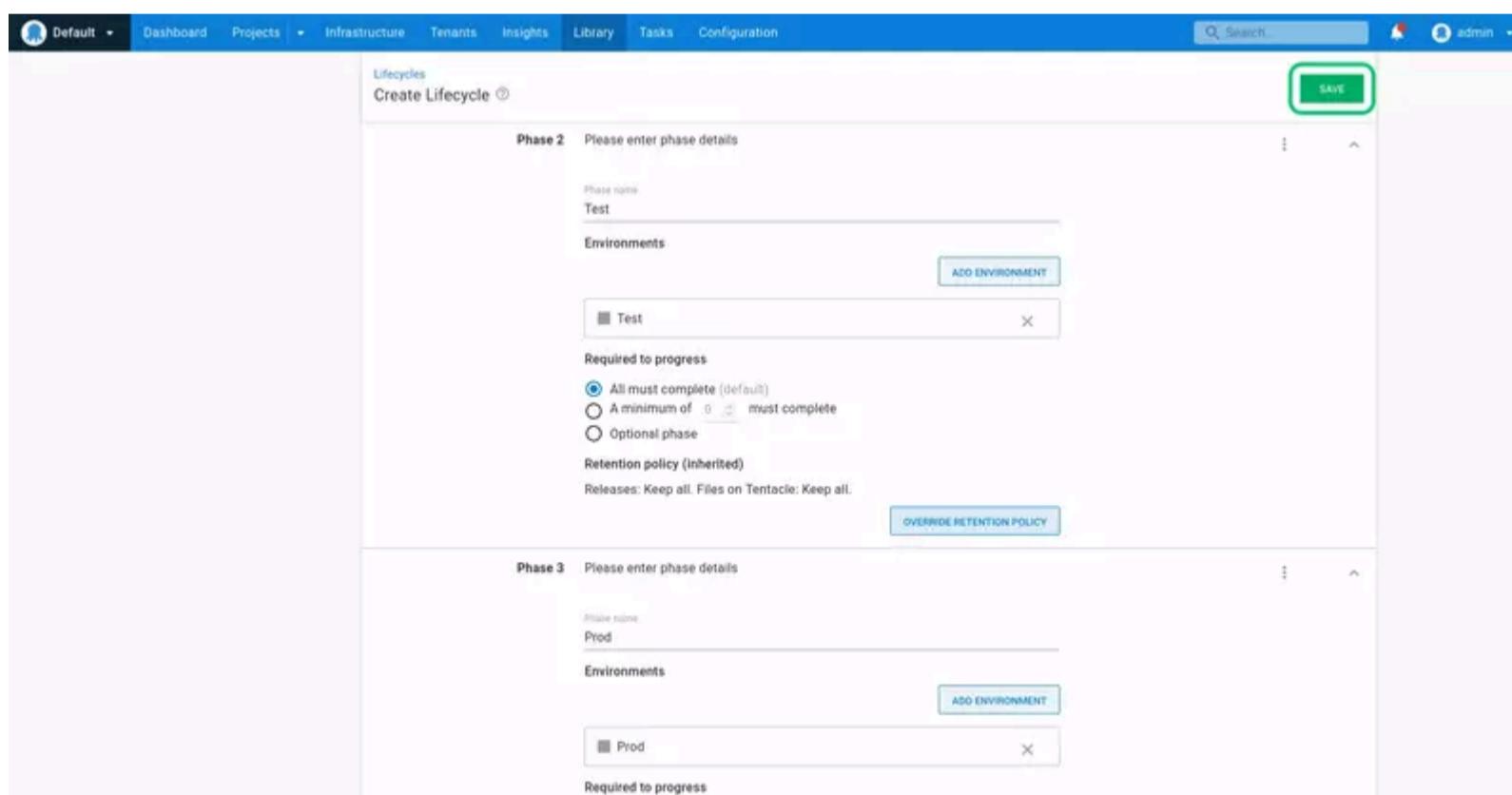
The modal window is still open, showing the 'Environment' dropdown with 'Dev' selected. A green rectangular box highlights the 'OK' button at the bottom right of the modal.

Repeat the process to add a new phase for the **Test and **Prod** environments, leaving the default **All must complete** option selected:**

Octopus Deploy



Click SAVE:



Now, we need to switch the deployment project from the **Default Lifecycle** to the newly created lifecycle.

Click the **Projects** link, and click the **Random Quotes** project tile:

Octopus Deploy



Click the **Process** link, and click **CHANGE**:

Select the **Dev**, **Test**, and **Prod** lifecycle. Notice the **Dev** environment is shown as optional.

Click **SAVE**:

Click **CREATE RELEASE**, and click **SAVE** to save the new release:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with navigation links for Random Quotes (CREATE RELEASE, DEPLOYMENTS, OPERATIONS, VARIABLES, TASKS, INSIGHTS), Infrastructure (PROJECTS, TENANTS, LIBRARY, TASKS, CONFIGURATION), and a search bar. The main area is titled 'Create release for Random Quotes' and shows details for Version 0.0.3, which includes 1 package at version 1.0.2. There are sections for 'Release Notes' (No release notes provided) and a large 'SAVE' button.

Because the **Dev** environment has been configured as optional, the initial release can be made to either the **Dev** or **Test** environments. We'll skip the **Dev** environment and deploy straight to **Test**.

Click **DEPLOY TO...**, and select the **Test** environment:

This screenshot shows the deployment configuration for 'Release 0.0.3'. It lists environments: Dev (optional), Test, and Prod. The 'Test' environment is selected. A dropdown menu for 'DEPLOY TO...' is open, showing options like 'Deploy to both environments...', 'Filter...', 'Dev', and 'Test', with 'Test' highlighted. Other sections visible include Progression, Packages (Deploy to Tomcat randomquotes version 1.0.2), Variable Snapshot, and Artifacts.

Click **DEPLOY** to deploy the application to the **Test** environment:

This screenshot shows the deployment preview for 'Release 0.0.3'. It includes fields for 'Environments' (Test), 'When' (Now (default)), 'Excluded steps' (No steps excluded (default)), 'Failure mode' (Use the default setting from the target environment (default)), and 'Package download' (Use cached packages (if available) (default)). The 'Preview and customize' section shows a table with columns: Environment, Current Version, Deployment Process, Targets, and Target Status. The 'Targets' column shows 'All included' and '1 HEALTHY'.

Octopus Deploy

Opening <http://localhost:9091/randomquotes-test> displays the copy of the Random Quotes application deployed to the **Test** environment. Note the context path **randomquotes-test** is the templated value of the context path that we deployed the application to.

Also see the footer text that says **running in Test**. This is the result of the contents of the `deployed-application.yml` file being processed during deployment to define the current Spring Boot profile to be the name of the environment that the application was deployed into:

Approvals

It's a common business requirement to have testers or product owners manually verify that a particular build meets the requirements before a deployment can be considered successful.

Octopus supports this workflow through the use of manual intervention steps. We'll add a manual intervention step to the end of the deployment process, which requires a responsible party to verify the build meets the requirements.

Open the Random Quotes project, click the **Process** link, and click the **ADD STEP** button:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with navigation links like Random Quotes, Deployments, Process, Operations, Variables, and Tasks. The main area is titled 'Process' and shows a step named '1. Deploy to Tomcat'. This step is described as 'Deploy a Java application to Tomcat 7+ using package randomquotes from Octopus Server (built-in) to deployment targets in role web'. To the right of the step, there's a 'Lifecycle' section set to 'Dev, Test and Prod' with options for 'Dev (optional)', 'Test', and 'Prod'. Below that is a 'Script Modules' section which is currently empty.

Search for the **Manual Intervention Required** step, and add it to the process:

This screenshot shows the 'Process Editor' dialog with the 'Choose Step Template' tab selected. In the 'Installed Step Templates' section, the 'Manual Intervention Required' step is highlighted with a green border. Below it, there are sections for 'Community Contributed Step Templates' and other installed templates like 'SQL - Deploy DACPAC with AAD Auth support' and 'Automate Manual Intervention Response'.

Enter **Deployment Sign Off** for the **Step Name**:

The screenshot shows the 'Process Editor' with the 'Deployment Sign Off' step selected. The 'Step Name' field is filled with 'Deployment Sign Off'. Under the 'Execution Location' section, it says 'This step will run on the Octopus Server'. The 'Manual Intervention' section contains an 'Instructions' field with the placeholder 'These instructions will be presented to the user to follow.' At the bottom, there's a 'Responsible Teams' section where users can select teams.

Enter the following for the **Instructions**:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with navigation links like Dashboard, Projects, Infrastructure, Tenants, Insights, Library, Tasks, Configuration, and a search bar. The main area is titled "Process Editor" and shows a workflow step titled "2. Deployment Sign Off". This step has a "Step Name" of "Deployment Sign Off", is set to run on the "Octopus Server", and includes instructions: "Open the application and confirm it meets all the requirements.". A "SAVE" button is visible at the top right of the editor.

Because every build is automatically deployed to the **Dev** environment, it doesn't make sense to force someone to manually approve all those deployments. To accommodate this, we do not enable the manual intervention step for deployments to the **Dev** environment.

Expand the **Environments** section under the **Conditions** heading, select the **Skip specific environments** option, and select the **Dev** environment.

Click **SAVE** to save the step:

This screenshot shows the same Octopus Deploy interface as above, but with the "Conditions" section expanded. Under the "Environments" tab, the "Skip specific environments" option is selected, and the "Dev" environment is listed. The "SAVE" button is highlighted with a green box.

When this application is deployed to the **Test** or **Prod** environments, a prompt will be displayed requiring manual sign off. Click **ASSIGN TO ME** to assign the task to yourself:

Octopus Deploy

TASK SUMMARY

This task is waiting for manual intervention and **must be assigned** before proceeding.

Assigned to: no one **ASSIGN TO ME**

Ran on: OctopusServerNodes-45a697243b37

Changes: Release 0.0.4

Artifacts: No artifacts have been added. Learn more about collecting artifacts.

Task History:

When	Who	What
a few seconds ago	system	Deploy to Test started for Random Quotes release 0.0.4 to Test
a few seconds ago	admin	Scheduled to deploy Random Quotes release 0.0.4 to Test at Wednesday, 04 January 2023 1:15 AM +00:00
a few seconds ago	admin	Ran task Deploy Random Quotes release 0.0.4 to Test

Previous Deployments:

- 0.0.3 → Deploy to Test Jan 4, 2023 1:10 AM

Add a note in the provided text box, and click **PROCEED** to complete the deployment:

TASK SUMMARY

This task is waiting for manual intervention.

Assigned to: you

Ran on: OctopusServerNodes-45a697243b37

Changes: Release 0.0.4

Artifacts: No artifacts have been added. Learn more about collecting artifacts.

Task History:

When	Who	What
a minute ago	system	Deploy to Test started for Random Quotes release 0.0.4 to Test
a minute ago	admin	Scheduled to deploy Random Quotes release 0.0.4 to Test at Wednesday, 04 January 2023 1:15 AM +00:00
a minute ago	admin	Ran task Deploy Random Quotes release 0.0.4 to Test

Previous Deployments:

- 0.0.3 → Deploy to Test Jan 4, 2023 1:10 AM

The deployment will then complete successfully:

TASK SUMMARY

This task has not yet started. This task has not started yet, but is next in the queue and will be executed as soon as possible.

Ran on: OctopusServerNodes-45a697243b37

Changes: Release 0.0.4

Artifacts: No artifacts have been added. Learn more about collecting artifacts.

Task History:

When	Who	What
a minute ago	system	Deploy to Test started for Random Quotes release 0.0.4 to Test
a minute ago	admin	Scheduled to deploy Random Quotes release 0.0.4 to Test at Wednesday, 04 January 2023 1:15 AM +00:00
a minute ago	admin	Ran task Deploy Random Quotes release 0.0.4 to Test

Previous Deployments:

- 0.0.3 → Deploy to Test Jan 4, 2023 1:10 AM

Octopus Deploy

process to let people know when a release has been deployed to an environment.

To start, we need to configure an SMTP server to send our emails. For this guide, we'll use the free SMTP server provided by Google.

Click the **Configuration** link:

Click the **SMTP** link:

- Enter **smtp.gmail.com** as the **SMTP Host**.
- Enter **587** as the **SMTP Port**.
- Enable the **Use SSL/TLS** option.
- Enter your Gmail address as the **From Address**.
- Enter your Gmail address and password in the **credentials**.

You will enable the [Less secure apps](#) option on your Google account for Octopus to send emails via the Google SMTP server.

Octopus Deploy

The screenshot shows the SMTP Configuration page in Octopus Deploy. On the left, a sidebar lists various settings like Audit, Backup, Diagnostics, Features, Git, License, Maintenance, Nodes, Performance, Settings, Extensions, SMTP (which is selected and highlighted in green), Subscriptions, Spaces, Teams, Telemetry, Test Permissions, Thumbprint, Users, User Invites, and User Roles. The main content area is titled "SMTP Configuration" and contains fields for "SMTP Host" (smtp.gmail.com), "SMTP Port" (587), "Timeout" (12 seconds), "Use SSL/TLS" (Yes), and "From Address" (octopusguides@gmail.com). A note at the bottom states: "Credentials have been entered; username is octopusguides@gmail.com". There are "SAVE AND TEST" and "SAVE" buttons at the top right.

Open the Random Quotes project, click the **Process** link, and click **ADD STEP**:

The screenshot shows the Process Editor for the "Random Quotes" project. The left sidebar shows "Deployments" selected. The main area displays a process flow with two steps: "1. Deploy to Tomcat" and "2. Deployment Sign Off". The "Deploy to Tomcat" step is currently selected. On the right, there's a "Lifecycle" section set to "Dev, Test and Prod" and a "Script Modules" section stating "No script modules have been included". A green "ADD STEP" button is located in the top right corner of the main process area.

Search for the **Send an Email** step, and add it to the process:

The screenshot shows the "Process Editor" screen for the "Random Quotes" project. The left sidebar has "Process" selected. In the center, the "Choose Step Template" section is open, showing "Installed Step Templates (1)" with a single item: "Send an Email" (described as "Sends an email using a configured SMTP server"). Below it, "Community Contributed Step Templates (2)" are listed: "Venafi TPP - Create and Provision Certificate" and "SQL - Query Octopus Database for Fragmentation". The "Send an Email" template is highlighted with a green border.

Enter **Random quotes deployment status** for the **Step Name**:

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with navigation links like Random Quotes, Deployments, Operations, Variables, and Insights. The main area is titled "Process Editor" and shows a deployment process with three steps: "1. Deploy to Tomcat", "2. Deployment Sign Off", and "3. Random quotes deployment status". The third step is currently selected. In the configuration pane, under "Step Name", the value "Random quotes deployment status" is highlighted with a green border. Under "Execution Location", it says "This step will run on the Octopus Server". Below that is a "Send an Email" section where the "To" field contains "Random quotes deployment status".

Enter the email address to receive the notification in the **To** field:

This screenshot shows the same Octopus Deploy interface as the previous one, but with a different email address entered in the "To" field. The "To" field now contains "octopsguides@gmail.com", which is highlighted with a green border. The rest of the configuration remains the same, with the "Step Name" set to "Random quotes deployment status" and the execution location set to the Octopus Server.

Enter **Random quotes deployment status** as the **Subject**:

This screenshot shows the final configuration of the email step. The "Subject" field in the "Send an Email" section is filled with "Random quotes deployment status", which is highlighted with a green border. The other fields in the "Send an Email" section ("To", "To Teams", "CC", "CC Teams", "BCC", "BCC Teams") are empty. Below the email section, there are sections for "Priority" (with "Normal" selected) and "Body" (with a placeholder "Provide the subject line of the email").

Enter the following as the **Body**:

Octopus Deploy

```
Status: #{step.Status.Code}
#{/each}
```

Here we use the **#{Octopus.Environment.Name}** variable provided by Octopus to add the name of the environment that was deployed to, and then loop over the status codes in the **#{Octopus.Step}** variable to return the status of each individual step.

The complete list of system variables can be found in the [Octopus documentation](#).

The screenshot shows the Octopus Process Editor interface. On the left, there's a sidebar with 'Tasks' and 'Insights' sections. The main area is titled 'Process' and 'Process Editor'. It shows an email step configuration with fields for CC, BCC, Priority (set to Normal), Subject (Random quotes deployment status), and Body. The Body field contains a script block:

```
Deployment to #{Octopus.Environment.Name}
#{foreach step in Octopus.Step}
StepName: #{step}
Status: #{step.Status.Code}
#{/each}
```

We want to be notified of the status of this deployment regardless of whether the deployment succeeded or failed. Click the **Run Conditions** section to expand it.

Select the **Always run** option, which ensures the notification email is sent even when the deployment or the manual intervention fail:

The screenshot shows the Octopus Process Editor with the 'Run Conditions' section expanded. Under 'Run Condition', the 'Always run' option is selected. Other options include 'Success: only run when previous steps succeed', 'Failure: only run when previous steps fail', and 'Variable: only run when the variable expression is true'.

Given every change to the source code will result in a deployment to the **Dev** environment, we do not want to generate reports for deployments to this environment.

Click the **Environments** section to expand it. Select the **Skip specific environments** option, and select the **Dev** environment to skip it.

This is the last piece of configuration for the step, so click **SAVE** to save the changes:

Octopus Deploy

The screenshot shows the Octopus Deploy configuration interface for an email step. The step is titled "Random quotes deployment status". The "Body" section contains a template with variables like {{Octopus.Environment.Name}} and {{StepName}}. The "Conditions" section is set to "Skip specific environments" and includes a dropdown menu with "Dev" selected. Other options include "Run for any environment (default)" and "Run only for specific environments".

Deploy the project to the Test or Prod environments. When the deployment succeeds, the notification email will be sent:

The screenshot shows a Gmail inbox with an email from "octopusguides@gmail.com" titled "Random quotes deployment status". The email was sent at 1:48 PM (2 minutes ago) and contains the following deployment details:

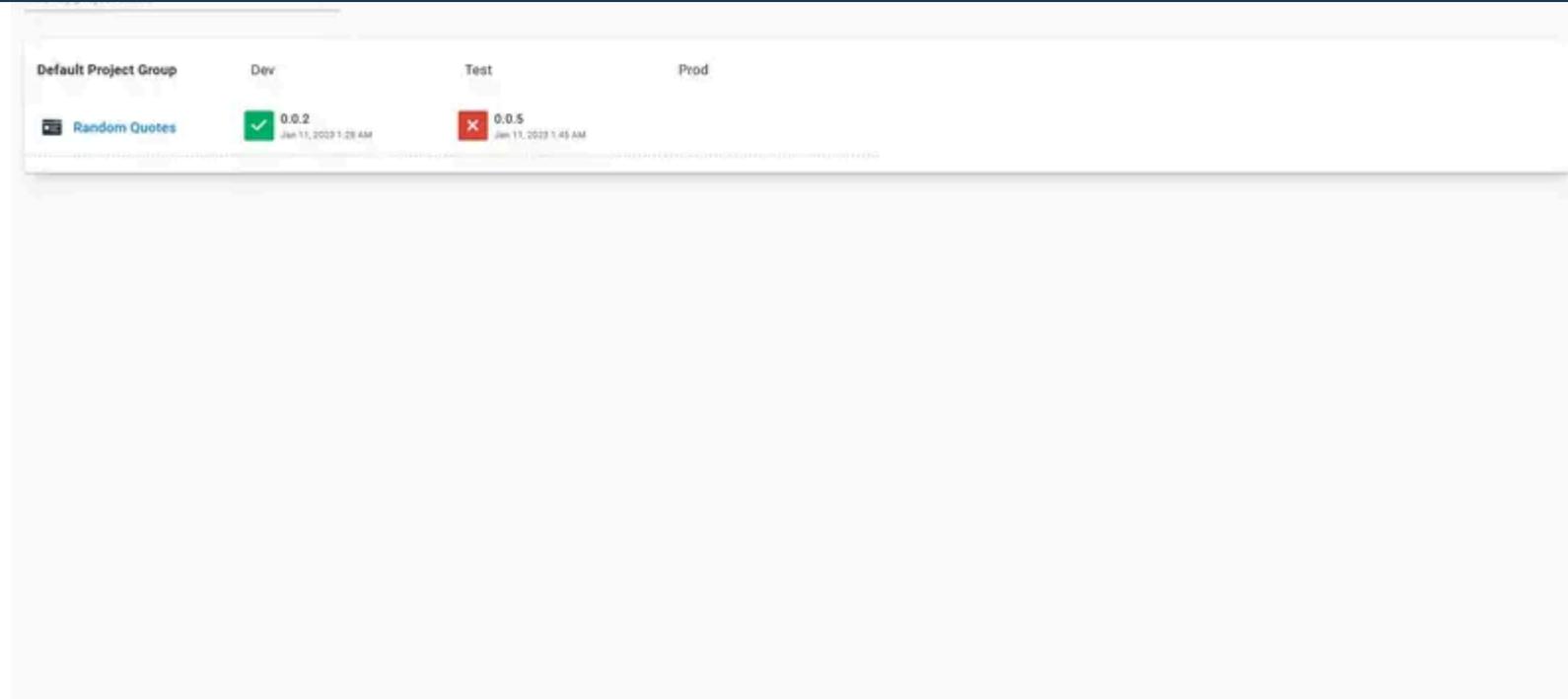
- Deployment to Dev
- StepName: Deploy web app to IIS
- Status: Succeeded
- StepName: Deployment Sign Off
- Status: Skipped
- StepName: Success Notification
- Status: Running
- StepName: Acquire Packages
- Status: Succeeded
- StepName: Apply Tentacle Retention Policy
- Status: Succeeded
- "

Permissions

One of the strengths of Octopus is that it models environments as first-class entities. This means the security layer can apply rules granting access only to specific environments. We'll take advantage of this ability to model and secure environments to create two users, an internal deployer who can deploy to the **Dev** and **Test** environments, and a production deployer who can deploy to the **Prod** environment.

We start by creating the users. Click the **Configuration** link:

Octopus Deploy



Click the **Users** link:

Click **ADD USER**:

Enter **internaldeployer** as the **Username**:

Octopus Deploy

Audit
Backup
Diagnostics
Features
Git
Let's Encrypt
License
Maintenance
Nodes
Performance
Settings
Extensions
SMTP
Subscriptions
Spaces
Teams
Telemetry
Test Permissions
Thumbprint
Users
User Invites
Clear Oracle

New User

Username: Enter a username the user authenticates with.
internaldeployer

Display Name: Enter a display name for the user. This does not need to be unique.
Internal Deployer

Service Account: A service account can log in using API keys only. After creating the user you'll need to add some API keys before the account can be used.
 The user is a service account.

Email Address: Enter an email address.
internaldeployer@example.org

Logins

Password: Optional. Set a password for this user.
internaldeployer

Confirm password: Confirm the password.

SAVE

Enter Internal Deployer as the Display Name:

Default ▾ Dashboard Projects Infrastructure Tenants Insights Library Tasks Configuration Search... admin ▾

Configuration

Audit
Backup
Diagnostics
Features
Git
Let's Encrypt
License
Maintenance
Nodes
Performance
Settings
Extensions
SMTP
Subscriptions
Spaces
Teams
Telemetry
Test Permissions
Thumbprint
Users
User Invites
Clear Oracle

New User

Username: Enter a username the user authenticates with.
internaldeployer

Display Name: Enter a display name for the user. This does not need to be unique.
Internal Deployer

Service Account: A service account can log in using API keys only. After creating the user you'll need to add some API keys before the account can be used.
 The user is a service account.

Email Address: Enter an email address.
internaldeployer@example.org

Logins

Password: Optional. Set a password for this user.
internaldeployer

Confirm password: Confirm the password.

SAVE

Enter the user's **email address**. We have used a dummy address of **internaldeployer@example.org** here:

Default ▾ Dashboard Projects Infrastructure Tenants Insights Library Tasks Configuration Search... admin ▾

Configuration

Audit
Backup
Diagnostics
Features
Git
Let's Encrypt
License
Maintenance
Nodes
Performance
Settings
Extensions
SMTP
Subscriptions
Spaces
Teams
Telemetry
Test Permissions
Thumbprint
Users
User Invites
Clear Oracle

New User

Username: Enter a username the user authenticates with.
internaldeployer

Display Name: Enter a display name for the user. This does not need to be unique.
Internal Deployer

Service Account: A service account can log in using API keys only. After creating the user you'll need to add some API keys before the account can be used.
 The user is a service account.

Email Address: The user's email address.
internaldeployer@example.org

Logins

Password: Optional. Set a password for this user.
internaldeployer

Confirm password: Confirm the password.

SAVE

Enter a password and confirm it. Then click **SAVE** to create the user:

Octopus Deploy

The screenshot shows the 'New User' creation form in Octopus Deploy. The 'Username' field is set to 'internaldeployer'. The 'Display Name' field is set to 'Internal Deployer'. The 'Email Address' field is set to 'internaldeployer@example.org'. In the 'Logins' section, two password fields are shown, both containing '*****'. A 'Service Account' checkbox is unchecked. The 'SAVE' button is visible at the top right.

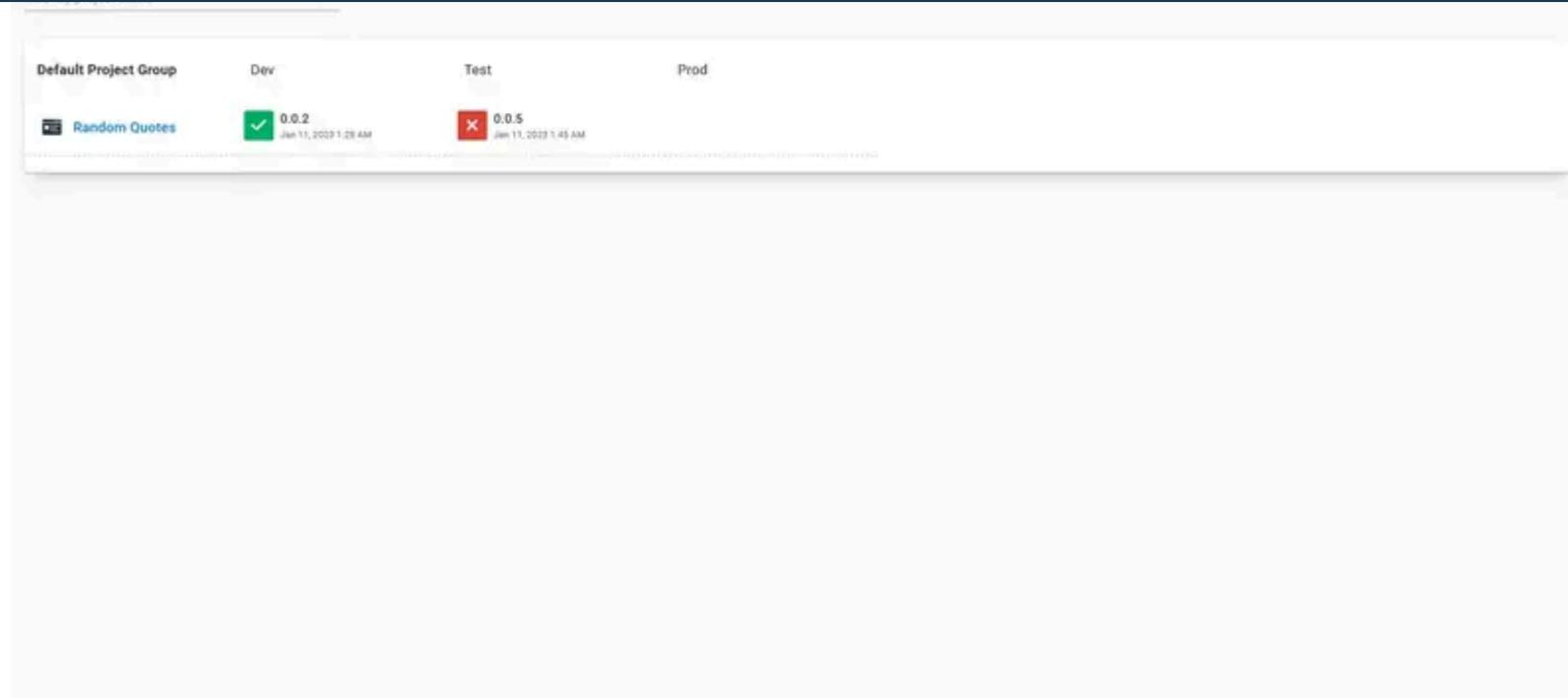
Repeat the process for a user called **productiondeployer**. The summary for the **productiondeployer** user is shown below:

The screenshot shows the 'Production Deployer' user summary page in Octopus Deploy. The 'Username' field is 'productiondeployer'. The 'Display Name' field is 'Production Deployer'. The 'Email Address' field is 'productiondeployer@example.org'. The 'Is Active' status is checked. In the 'Teams' section, it shows 'Assigned to teams' with 'Everyone' selected. The 'CHANGE PASSWORD' and 'SAVE' buttons are visible at the top right.

The newly created users are not assigned to any teams and have no permissions to do anything. To grant them permissions, we must first create two teams. The internal deployment team will grant access to deploy to the **Dev** and **Test** environments, while the production deployment team will grant access to deploy to the **Prod** environment.

Click the **Configuration** link:

Octopus Deploy



Click the **Teams** link:

Click **ADD TEAM**:

Enter **Internal Deployers** for the **New team name**, and **Grants access to perform a deployment to the internal environments** for the **Team description**. Click **SAVE** to create the team:

Octopus Deploy

The screenshot shows the Octopus Deploy interface with the 'Teams' section selected. A modal window titled 'Add New Team' is open, showing a team named 'Internal Deployers' with a description: 'Grants access to perform a deployment to the internal environments'. There are two radio button options for accessibility: 'Accessible in the default space only' (selected) and 'Accessible in all spaces (system team)'. A 'SAVE' button is at the bottom right of the modal.

We need to add the **internaldeployer** user to this team. Click **ADD MEMBER**:

The screenshot shows the 'Internal Deployers' team configuration page. The 'MEMBERS' tab is selected. A 'ADD MEMBER' button is highlighted with a green box.

Select the **Internal Deployer** user from the dropdown list, and click **ADD**:

The screenshot shows the 'Internal Deployers' team configuration page. A modal window titled 'Add Member' is open, showing a dropdown menu with 'Internal Deployer (internaldept...)'. A 'Select users' dropdown is also visible. A 'ADD' button is highlighted with a green box.

The team does not grant any permissions yet. To add permissions click the **USER ROLES** tab, and click **INCLUDE USER ROLE**:

Octopus Deploy

The screenshot shows the Octopus Deploy configuration interface. On the left, there's a sidebar with various navigation options like Audit, Backup, Diagnostics, Features, Git, Let's Encrypt, License, Maintenance, Nodes, Performance, Settings, Extensions, SMTP, Subscriptions, Spaces, Teams (which is selected), Telemetry, Test Permissions, Thumbprint, Users, and User Invites. The main area is titled 'Internal Deployers' under the 'Space Team' section. It says 'Grants access to perform a deployment to the internal environments'. Below this are tabs for 'MEMBERS', 'USER ROLES' (which is highlighted with a green box), and 'SETTINGS'. A note below the tabs says 'User roles grant teams permissions. Learn more'. At the bottom right of the main area is a button labeled 'INCLUDE USER ROLE'.

Select the **Deployment creator** role from the dropdown list. As the name suggests, this role allows a user to create a deployment, which results in the deployment process being executed.

Click **DEFINE SCOPE**:

The screenshot shows the 'INCLUDE USER ROLE' dialog box. It has a title 'Select a user role' with a dropdown menu showing 'Deployment creator'. Below it is a note: 'A user role specifies what actions users in this team will be able to perform.' and 'In order to view/assign a user role to a space-scoped team, that user role must contain at least one space-level permission.' There is a radio button next to 'Deployment creator' with the text 'Deployment creators can create new deployments and runbook runs.' Below the dialog are 'SHOW PERMISSIONS', 'DEFINE SCOPE' (which is highlighted with a green box), 'CANCEL', and 'APPLY' buttons.

We only want to allow the internal deployer to deploy to the internal environments. Select the **Dev** and **Test** environments, and click **APPLY**:

Octopus Deploy

The permissions are then applied. We need a second permission to allow the internal deployer to view the projects dashboard. Click **INCLUDE USER ROLE** again:

Select the **Project viewer** role. This role does not need to be scoped, so click the **APPLY** button:

Here are the final set of roles applied to the team:

Octopus Deploy

The screenshot shows the 'Teams' section of the Octopus Deploy configuration. A new team named 'Space Team' is being created. The 'USER ROLES' tab is selected, showing two roles: 'Deployment creator' and 'Project viewer'. The 'Deployment creator' role is described as granting permissions for all project groups and projects, and environments like Dev, Test, and Prod. The 'Project viewer' role is described as granting permissions for all project groups, projects, environments, and tenants. Buttons for 'INCLUDE USER ROLE' are visible next to each role entry.

Repeat the process to create a team called **Production Deployers** that includes the **productiondeployer** user, and grants the **Deployment creator** role scoped to the **Prod** environment:

This screenshot shows the same 'Teams' configuration screen as the previous one, but for a different team named 'Production Deployers'. The 'USER ROLES' tab is selected, and it shows the same two roles: 'Deployment creator' and 'Project viewer'. The 'Deployment creator' role is described as granting permissions for all project groups and projects, and environments like Prod. The 'Project viewer' role is described as granting permissions for all project groups, projects, environments, and tenants. Buttons for 'INCLUDE USER ROLE' are visible next to each role entry.

When we log in as the **internaldeployer** user, we see that the Random Quotes project dashboard shows **DEPLOY...** buttons for the **Dev** and **Test** environments. Any deployments in the production environment will be visible, but they cannot be created by this user:

The screenshot shows the 'Random Quotes' project dashboard. On the left, there's a sidebar with navigation links for 'Random Quotes' (Deployment, Overview, Process, Channels, Releases, Triggers, Settings), 'Operations' (Overview, Runbooks, Triggers), 'Tasks', 'Insights' (Overview, Deployment frequency, Deployment lead time, Deployment failure rate, Mean time to recovery), and 'Settings' (General, Version Control). The main area displays deployment history for releases 0.0.1, 0.0.2, 0.0.4, and 0.0.5 across environments Dev, Test, and Prod. For each release, there are 'DEPLOY...' buttons for Dev and Test, and a status indicator for Prod. Release 0.0.5 has a red 'X' icon next to its Prod status, indicating a failed deployment. Other releases show green checkmarks for both environments.

Octopus Deploy

The screenshot shows the Octopus Deploy interface. On the left, there's a sidebar with links for Projects, Infrastructure, Tenants, Insights, Library, Tasks, and Configuration. The main area is titled "Random Quotes Overview". It shows deployment history for four versions: 0.0.5 (failed), 0.0.4 (pending), 0.0.2 (successful), and 0.0.1 (successful). Each deployment row includes a "SHOW ADVANCED FILTERS" link.

Audit Log

Important interactions in Octopus are tracked in the audit log. This is useful for teams that have security or legislative requirements to track the changes and deployments made to their infrastructure.

To view the audit log, click the **Configuration** link:

The screenshot shows the Octopus Configuration page. At the top, there's a navigation bar with links for Dashboard, Projects, Infrastructure, Tenants, Insights, Library, Tasks, Configuration (which is highlighted in green), Search, Notifications, and User admin. Below the navigation is a "Dashboard" section with a "Filter by project name" input field. The main content area shows a deployment log for the "Default Project Group". It lists a deployment from "Random Quotes" to the "Prod" environment on "Jan 11, 2023 1:28 AM".

Click the **Audit** link:

Octopus Deploy

The screenshot shows the Octopus Deploy configuration interface. On the left, there's a sidebar with various settings like Audit, Backup, Diagnostics, Features, Git, etc. The 'Features' tab is selected and highlighted with a green border. The main content area displays the 'Features' configuration page. At the top right of this area are 'SAVE' and 'EXPAND ALL' buttons. The configuration is organized into sections: 'Early Access' (Kubernetes Cloud Target Discovery, Insights), 'Feeds' (Composite DockerHub Client), 'Steps' (Step Template Updates, Community Step Templates, Run steps on Octopus Server), 'Help Sidebar' (Customize the Help Sidebar, Toggle the Help Sidebar), and 'Dynamic Extensions' (Net Promoter Score, Additional Data). Each section has a collapse arrow icon.

A complete list of records are shown, with filtering available to help find specific events:

The screenshot shows the Octopus Deploy audit log interface. The top navigation bar includes 'Default', 'Dashboard', 'Projects', 'Infrastructure', 'Tenants', 'Insights', 'Library', 'Tasks', 'Configuration', a search bar, and a user dropdown. The 'Audit' section is selected in the sidebar. The main area shows a table of audit events. The first event is a warning: 'Better performance coming soon. To improve performance, archived audit logs older than 90 days will only be available as downloadable files. These audit log files can be accessed through the overflow menu (three dots) on this page or through the API. Learn more about this change here.' Below this is a table header with columns: Action, Date, User, and Type. The table lists several events from January 11, 2023, involving users 'admin', 'system', and 'internaldeployer'. Examples include 'Login succeeded for admin from 127.0.0.1' and 'Team Internal Deployers was modified'.

Octopus Deploy

In this guide we ran through the process of building a complete CI/CD pipeline with:

- GitHub Actions building and testing the source code and pushing the package to Octopus.
- Octopus deploying the package to the **Dev**, **Test**, and **Prod** environments.
- Email notifications generated when deployments succeed or fail.
- Manual sign off for deployments to the **Test** and **Prod** environments.
- Users with limited access to create releases in a subset of environments.

This is a solid starting point for any development team, but Octopus offers so much more! Below are more resources you can access to learn about the advanced functionality provided by Octopus:

- [The Octopus documentation](#)
- [The Octopus blog](#)
- [Upcoming events and webinars](#)
- [The Octopus community slack channel](#)