oduct or feature is in a pre-release state and might change or have limited support. For more information, see the product launch stages lucts/#product-launch-stages).

This tutorial walks you through configuring a deployment of NGINX with Application Delivery. The deployment runs on two environments, `staging` and `prod`. The `prod` environment uses regular configuration, while `staging` uses a slightly modified one.

For more information on Application Delivery, see the Concept document (/kubernetes-engine/docs/concepts/add-on/application-delivery).

To complete this tutorial, you will need the following:

- Git installed locally.
- A GitHub or GitLab account with permissions to create a private repository. Application Delivery supports only GitHub and GitLab repositories.
- A cluster running GKE 1.15 or higher.
- A user with clusterAdmin (/kubernetes-engine/docs/how-to/iam#predefined) privileges.

Before you start, make sure you have performed the following tasks:

- Ensure that you have enabled the Google Kubernetes Engine API.

  Enable Google Kubernetes Engine API (https://console.cloud.google.com/apis/library/container.googleapis.com?q=kubernetes%20engine)
- Ensure that you have installed the Cloud SDK (/sdk/downloads).

Set up default `gcloud` settings using one of the following methods:

- Using `gcloud init`, if you want to be walked through setting defaults.
- Using `gcloud config`, to individually set your project ID, zone, and region.

- Add SSH keys to your GitHub (https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account) or GitLab (https://docs.gitlab.com/ee/gitlab-basics/create-your-ssh-keys.html) account.
- Test your keys with `ssh`:

You might be asked to confirm connection details or your key passphrase. If the connection succeeds, a message is printed on the terminal.

To use Application Delivery, you can create (#create) a new cluster with it enabled, or enable (#enable) it on an existing GKE cluster running version 1.15 and higher. Then, you install `appctl`, the Application Delivery command line tool.

You can create a new cluster with Application Delivery enabled using `gcloud` or the Google Cloud Console.

You can enable Application Delivery on an existing using `gcloud` or the Google Cloud Console.

To check the status of your Application Delivery installation, run the following `kubectl` commands.

When Application Delivery is running, both commands return that there is exactly one pod with a `STATUS` of `Running`.

You install `appctl`, the Application Delivery command line tool, with `gcloud`.

After enabling Application Delivery on a cluster and installing `appctl`, you are ready to deploy your first application.

The following sections describe how to:

1. Create git repositories with `appctl`.
2. Create a base configuration.
3. Create one or more environments for your deployment.

4. Optionally, apply configuration overlays to your environments in your application repository.
5. Create a release candidate in the form of a pull or merge request.
6. Deploy your release.

You create repositories for Application Delivery on GitHub or GitLab or with `appctl`.

1. Change to the directory where you would like to create your application directory.
2. Create your Application Delivery repositories with `appctl`.

`appctl` prompts you to confirm your new private repositories.

Git might prompt you for additional login information.

`appctl` creates two remote private git repositories:

- The application repository `github.com/`***`[USER_NAME]`***`/`***`[APPLICATION_NAME]`***. This repository is cloned to the current directory.
- The deployment repository `github.com/`***`[USER_NAME]`***`/`***`[APPLICATION_NAME]`***`-deployment`.

For more information on the content and structure of these repositories, see the Application delivery (/kubernetes-engine/docs/concepts/add-on/application-delivery) concept guide.

1. Change your working directory to your application repository. For example, if you used the application name `myapp`, run

2. Create the configuration for your Kubernetes workload. This can be any valid Kubernetes deployment.

   The following configuration defines an application named `nginx`, which deploys 3 replicas of the `nginx` container. Copy the configuration into the file `config/base/myapp.yaml` . If you would like to enable a LoadBalancer, uncomment the line `type: LoadBalancer`.

3. Configure Application Delivery to apply this configuration to the base. Paste the following into `config/base/kustomization.yaml`.

1. From your application repository directory, test your configuration with `kubectl apply -k`:

   If your configuration is valid, `kubectl` prints the YAML that will be deployed to your cluster when it is applied.

2. After you have validated your YAML, create and push a commit in your application repository.

Application Delivery deploys your application into environments. You add environments for your releases with `appctl`.

1. Change to your application repository root directory (for example, `cd myapp`).
2. Create your environment with `appctl`

   `appctl` creates a git commit containing a scaffolded Kustomize configuration.

   For example, to add the `staging` and `prod` environments to the cluster `application-cluster`, run the following command:

★ **Note:** `appctl` may prompt you for an access token for your Git repository.

3. Optionally, you can see the changes Application Delivery made in your Git repository with `git log`.

4. Push the configuration to your application repository.

1. Open the GitHub or GitLab page for your deployment repository. For example, if your GitHub username is `octocat` and you created an application named `myapp`, the URL is `https://github.com/octocat/myapp-deployment`. From this page, you can see the branches that were created for each environment.

To deploy an environment with Application Delivery, you:

1. Create a version with `git tag` and push that tag.

★ **Note:** We recommend you use Semantic Versioning (https://semver.org/) for your version numbers. This allows you to sort versions easily on GitHub or GitLab pages.

For example, to push version `v0.1.0`, run the following commands:

2. Use `appctl prepare` to nominate the currently tagged version and generate a pull request in the deployment repository for review.

★ **Note:** `appctl prepare` resets your `kubeconfig`, including the default context and Namespace. If you have changed your default context, you will need to re-apply this change after using `appctl`.

For example, to use the `staging` environment, run the following command:

If `appctl` completed the commit to the deployment repository, it prints a URL to create a pull request.

Open the URL in your browser. The **Comparing Changes** screen (in GitHub) or the **New Merge Request** screen (in GitLab) appears.

3. Use GitHub (https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/merging-a-pull-request) or GitLab (https://docs.gitlab.com/ee/user/project/merge_requests/#creating-merge-requests) to review and approve the pull request.

Caution: When merging pull requests on GitHub or GitLab, use a merge commit. Application Delivery cannot identify "Squash and merge" or "Rebase and merge" commits.

4. After the pull request has been approved, use `appctl apply` to complete the deployment.

   For example, to deploy changes to the `staging` environment, run the following:

5. Confirm that your application is running with `kubectl` or from the Google Cloud Console.

1. To promote a release candidate from one environment to another, run the following command:

   Where `[TARGET_ENVIRONMENT_NAME]` is the name of the environment that you wish to deploy the release candidate currently running on `[SOURCE_ENVIRONMENT_NAME]`.

   For example, to promote `staging` to `prod` run:

2. Use GitHub or GitLab to review and approve the pull request.
3. To deploy the release candidate to the target environment, run the following command:

   For example, to deploy to the `prod` environment, run:

This section assumes you have a `staging` environment configured as in the previous steps (#deploying_an_application). You may need to adapt these instructions for your use.

In this section, you change the parameters for the `staging` environment using a kustomize overlay (https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/#bases-and-overlays). After making the change, you push and tag your changes in git. Application Delivery will update your cluster.

1. Create the file `config/envs/staging/patch-replicas.yaml`, and copy the following text into it. This updates the configuration in the `staging` environment to run one replica instead of three replicas.

★ **Note:** The `apiVersion`, `kind`, and `metadata.name` values are required.

2. Edit `config/envs/staging/kustomization.yaml` and add `patch-replicas.yaml` to a new collection named `patchesStrategicMerge`.

You can also add environment-specific annotations (https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/) in this overlay. The following example adds an annotation named `oncall-team` to add all resources under this environment. For more information, see Kustomize file fields (https://github.com/kubernetes-sigs/kustomize/blob/v2.1.0/docs/fields.md).

3. Test your configuration with `kubectl apply -k`:

4. Add and commit your changes.

   Where `[COMMIT_MESSAGE]` is a message that describes your changes.

5. Create a version with `git tag` and push it.

6. Use `appctl prepare` to generate a pull request in the deployment repository for review.

7. Follow the link to create a GitHub or GitLab pull request.
8. Look over the contents of your pull request. Application Delivery makes a one-line change that sets the value of `replicas` to `1`.
9. Approve the pull request with GitHub or GitLab
10. Use `appctl apply` to apply the changes.

You can use `appctl apply` to roll back to a previous release.

Where `[TARGET_ENVIRONMENT_NAME]` is the name of the environment that you wish to deploy the release tagged with `[GIT_TAG]`.

The `appctl` tool is interactive, expects for user input, by default. If you want to run `appctl` in a script, container, or pipelines, set the environment variable `APPCTL_INTERACTIVE` to `false`.

For example, in the bash shell, run the following command.

Information on specific `appctl` commands is available with `appctl help [COMMAND]`. For example, to get help with `appctl prepare`, run `appctl help prepare` .

To remove the application running in your cluster, you delete all Namespaces created with new environments. For all of your environments and clusters, repeat the following commands:

1. Switch to the cluster for a given environment

2. Delete the namespace where your application for this environment is running

   Where `[APPLICATION_NAME]` is the name of your application repository, and `[ENVIRONMENT_NAME]` is your environment name.

3. From GitHub or GitLab, delete the two git repositories created by `appctl`.

4. Delete your local application directory:

5. You can disable Application Delivery in your cluster from `gcloud` or the Google Cloud Console:

Read more about Kustomize (https://kubectl.docs.kubernetes.io/pages/examples/kustomize.html).