



On this page:

- › Overview
- › Quick start
- › Comparison to application platforms and PaaS
- › Features
- › Requirements
- › Auto DevOps base domain
- › Using multiple Kubernetes clusters
- › Enabling Auto DevOps
 - › Deployment strategy
- › Stages of Auto DevOps
 - › Auto Build
 - › Auto Test
 - › Auto Code Quality
 - › Auto SAST
 - › Auto Dependency Scanning
 - › Auto License Management
 - › Auto Container Scanning
 - › Auto Review Apps
 - › Auto DAST
 - › Auto Browser Performance Testing
 - › Auto Deploy
 - › Auto Monitoring
- › Customizing
 - › Custom buildpacks
 - › Custom Dockerfile
 - › Custom Helm Chart
 - › Customizing `.gitlab-ci.yml`
 - › PostgreSQL database support
 - › Environment variables
 - › Advanced replica variables setup
 - › Deploy policy for staging and production environments
 - › Deploy policy for canary environments
 - › Incremental rollout to production
 - › Timed incremental rollout to production
- › Currently supported languages
- › Limitations
 - › Private project support
- › Troubleshooting
 - › Disable the banner instance wide

Auto DevOps

Introduced (<https://gitlab.com/gitlab-org/gitlab-ce/issues/37115>) in GitLab 10.0. Generally available on GitLab 11.0.

Auto DevOps automatically detects, builds, tests, deploys, and monitors your applications.

Overview

🔔 Enabled by default: Starting with GitLab 11.3, the Auto DevOps pipeline will be enabled by default for all projects. If it's not explicitly enabled for the project, Auto DevOps will be automatically disabled on the first pipeline failure. Your project will

continue to use an alternative CI/CD configuration file (`../ci/yaml/README.html`) if one is found. A GitLab administrator can change this setting (`../user/admin_area/settings/continuous_integration.html#auto-devops`) in the admin area.

With Auto DevOps, the software development process becomes easier to set up as every project can have a complete workflow from verification to monitoring without needing to configure anything. Just push your code and GitLab takes care of everything else. This makes it easier to start new projects and brings consistency to how applications are set up throughout a company.

Quick start

If you are using GitLab.com, see the quick start guide ([quick_start_guide.html](#)) for using Auto DevOps with GitLab.com and a Kubernetes cluster on Google Kubernetes Engine.

Comparison to application platforms and PaaS

Auto DevOps provides functionality described by others as an application platform or as a Platform as a Service (PaaS). It takes inspiration from the innovative work done by Heroku [↗](#) and goes beyond it in a couple of ways:

1. Auto DevOps works with any Kubernetes cluster, you're not limited to running on GitLab's infrastructure (note that many features also work without Kubernetes).
2. There is no additional cost (no markup on the infrastructure costs), and you can use a self-hosted Kubernetes cluster or Containers as a Service on any public cloud (for example Google Kubernetes Engine [↗](#)).
3. Auto DevOps has more features including security testing, performance testing, and code quality testing.
4. It offers an incremental graduation path. If you need advanced customizations you can start modifying the templates without having to start over on a completely different platform.

Features

Comprised of a set of stages, Auto DevOps brings these best practices to your project in a simple and automatic way:

1. Auto Build
2. Auto Test
3. Auto Code Quality [?](#)
4. Auto SAST (Static Application Security Testing) [?](#)
5. Auto Dependency Scanning [?](#)
6. Auto License Management [?](#)
7. Auto Container Scanning
8. Auto Review Apps
9. Auto DAST (Dynamic Application Security Testing) [?](#)
10. Auto Deploy
11. Auto Browser Performance Testing [?](#)
12. Auto Monitoring

As Auto DevOps relies on many different components, it's good to have a basic knowledge of the following:

- Kubernetes [↗](#)
- Helm [↗](#)
- Docker [↗](#)
- GitLab Runner (<https://docs.gitlab.com/runner/>)
- Prometheus [↗](#)

Auto DevOps provides great defaults for all the stages; you can, however, customize almost everything to your needs.

For an overview on the creation of Auto DevOps, read the blog post [From 2/3 of the Self-Hosted Git Market, to the Next-Generation CI System, to Auto DevOps](#) (<https://about.gitlab.com/2017/06/29/whats-next-for-gitlab-ci/>).

Requirements

To make full use of Auto DevOps, you will need:

1. **GitLab Runner** (needed for all stages) - Your Runner needs to be configured to be able to run Docker. Generally this means using the Docker (<https://docs.gitlab.com/runner/executors/docker.html>) or Kubernetes executor (<https://docs.gitlab.com/runner/executors/kubernetes.html>), with privileged mode enabled (<https://docs.gitlab.com/runner/executors/docker.html#use-docker-in-docker-with-privileged-mode>). The Runners do not need to be installed in the Kubernetes cluster, but the Kubernetes executor is easy to use and is automatically autoscaling. Docker-based Runners can be configured to autoscale as well, using Docker Machine (<https://docs.gitlab.com/runner/install/autoscaling.html>). Runners should be registered as shared Runners

- ([../ci/runners/README.html#registering-a-shared-runner](#)) for the entire GitLab instance, or specific Runners ([../ci/runners/README.html#registering-a-specific-runner](#)) that are assigned to specific projects.
- 2. Base domain** (needed for Auto Review Apps and Auto Deploy) - You will need a domain configured with wildcard DNS which is going to be used by all of your Auto DevOps applications. Read the specifics.
 - 3. Kubernetes** (needed for Auto Review Apps, Auto Deploy, and Auto Monitoring) - To enable deployments, you will need Kubernetes 1.5+. You need a Kubernetes cluster ([../user/project/clusters/index.html](#)) for the project, or a Kubernetes default service template ([../user/project/integrations/services_templates.html](#)) for the entire GitLab installation.
 - 1. A load balancer** - You can use NGINX ingress by deploying it to your Kubernetes cluster using the [nginx-ingress](#) Helm chart.
 - 4. Prometheus** (needed for Auto Monitoring) - To enable Auto Monitoring, you will need Prometheus installed somewhere (inside or outside your cluster) and configured to scrape your Kubernetes cluster. To get response metrics (in addition to system metrics), you need to configure Prometheus to monitor NGINX ([../user/project/integrations/prometheus_library/nginx_ingress.html#configuring-prometheus-to-monitor-for-nginx-ingress-metrics](#)). The Prometheus service ([../user/project/integrations/prometheus.html](#)) integration needs to be enabled for the project, or enabled as a default service template ([../user/project/integrations/services_templates.html](#)) for the entire GitLab installation.

Note: If you do not have Kubernetes or Prometheus installed, then Auto Review Apps, Auto Deploy, and Auto Monitoring will be silently skipped.

Auto DevOps base domain

The Auto DevOps base domain is required if you want to make use of Auto Review Apps and Auto Deploy. It can be defined in three places:

- either under the project's CI/CD settings while enabling Auto DevOps
- or in instance-wide settings in the **admin area > Settings** under the "Continuous Integration and Delivery" section
- or at the project or group level as a variable: `AUTO_DEVOPS_DOMAIN` (required if you want to use multiple clusters)

A wildcard DNS A record matching the base domain(s) is required, for example, given a base domain of `example.com`, you'd need a DNS entry like:

```
*.example.com 3600 A 1.2.3.4
```

In this case, `example.com` is the domain name under which the deployed apps will be served, and `1.2.3.4` is the IP address of your load balancer; generally NGINX (see requirements). How to set up the DNS record is beyond the scope of this document; you should check with your DNS provider.

Alternatively you can use free public services like [nip.io](#) which provide automatic wildcard DNS without any configuration. Just set the Auto DevOps base domain to `1.2.3.4.nip.io`.

Once set up, all requests will hit the load balancer, which in turn will route them to the Kubernetes pods that run your application(s).

Using multiple Kubernetes clusters

PREMIUM SILVER

When using Auto DevOps, you may want to deploy different environments to different Kubernetes clusters. This is possible due to the 1:1 connection that exists between them ([../user/project/clusters/index.html#multiple-kubernetes-clusters](#)).

In the Auto DevOps template (<https://gitlab.com/gitlab-org/gitlab-ce/blob/master/lib/gitlab/ci/templates/Auto-DevOps.gitlab-ci.yml>) (used behind the scenes by Auto DevOps), there are currently 3 defined environment names that you need to know:

- `review/` (every environment starting with `review/`)
- `staging`
- `production`

Those environments are tied to jobs that use Auto Deploy, so except for the environment scope, they would also need to have a different domain they would be deployed to. This is why you need to define a separate `AUTO_DEVOPS_DOMAIN` variable for all the above based on the environment ([../ci/variables/README.html#limiting-environment-scopes-of-variables](#)).

The following table is an example of how the three different clusters would be configured.

Cluster name	Cluster environment scope	AUTO_DEVOPS_DOMAIN variable value	Variable environment scope	Notes
				The review cluster which will run all Review Apps

review Cluster name	Cluster/ environment scope	review.example.com AUTO_DEVOPS_DOMAIN variable value	Variable* environment scope	(../ci/review_apps/index.html). * is a wildcard, which means it will be used by every environment name starting with review/ . Notes
staging	staging	staging.example.com	staging	(Optional) The staging cluster which will run the deployments of the staging environments. You need to enable it first.
production	production	example.com	production	The production cluster which will run the deployments of the production environment. You can use incremental rollouts.

To add a different cluster for each environment:

1. Navigate to your project's **Operations > Kubernetes** and create the Kubernetes clusters with their respective environment scope as described from the table above.

Kubernetes clusters can be used to deploy applications and to provide Review Apps for this project Add Kubernetes cluster

Kubernetes cluster	Environment scope	Project namespace	
production	production	gitlab-docs-1794617	<input checked="" type="checkbox"/>
staging	staging	gitlab-docs-1794617	<input checked="" type="checkbox"/>
review	review/*	gitlab-docs-1794617	<input checked="" type="checkbox"/>

(img/autodevops_multiple_clusters.png)

2. After the clusters are created, navigate to each one and install Helm Tiller and Ingress.
3. Make sure you have configured your DNS with the specified Auto DevOps domains.
4. Navigate to your project's **Settings > CI/CD > Variables** and add the `AUTO_DEVOPS_DOMAIN` variables with their respective environment scope.

AUTO_DEVOPS_DOMAIN	example.com	Protected <input checked="" type="checkbox"/>	production	⊖
AUTO_DEVOPS_DOMAIN	review.example.com	Protected <input checked="" type="checkbox"/>	review/*	⊖
AUTO_DEVOPS_DOMAIN	staging.example.com	Protected <input checked="" type="checkbox"/>	staging	⊖

(img/autodevops_domain_variables.png)

Now that all is configured, you can test your setup by creating a merge request and verifying that your app is deployed as a review app in the Kubernetes cluster with the `review/*` environment scope. Similarly, you can check the other environments.

Enabling Auto DevOps

If you haven't done already, read the requirements to make full use of Auto DevOps. If this is your first time, we recommend you follow the quick start guide (quick_start_guide.html).

To enable Auto DevOps to your project:

1. Check that your project doesn't have a `.gitlab-ci.yml`, or remove it otherwise
2. Go to your project's **Settings > CI/CD > Auto DevOps**
3. Select "Enable Auto DevOps"
4. Optionally, but recommended, add in the base domain that will be used by Kubernetes to deploy your application and choose the deployment strategy
5. Hit **Save changes** for the changes to take effect

Once saved, an Auto DevOps pipeline will be triggered on the default branch.

Note: For GitLab versions 10.0 - 10.2, when enabling Auto DevOps, a pipeline needs to be manually triggered either by pushing a new commit to the repository or by visiting <https://example.gitlab.com/<username>/<project>/pipelines/new> and creating a new pipeline for your default branch, generally `master`.

Note: If you are a GitLab Administrator, you can enable/disable Auto DevOps instance-wide ([../user/admin_area/settings/continuous_integration.html#auto-devops](https://example.gitlab.com/admin/area/settings/continuous_integration.html#auto-devops)), and all projects that haven't explicitly set an option will have Auto DevOps enabled/disabled by default.

Note: There is also a feature flag to enable Auto DevOps to a percentage of projects which can be enabled from the console with `Feature.get(:force_autodevops_on_by_default).enable_percentage_of_actors(10)`.

Deployment strategy

Introduced (<https://gitlab.com/gitlab-org/gitlab-ce/issues/38542>) in GitLab 11.0.

You can change the deployment strategy used by Auto DevOps by going to your project's **Settings > CI/CD > Auto DevOps**.

The available options are:

- **Continuous deployment to production:** Enables Auto Deploy with `master` branch directly deployed to production.
- **Continuous deployment to production using timed incremental rollout:** Sets the `INCREMENTAL_ROLLOUT_MODE` variable to `timed`, and production deployment will be executed with a 5 minute delay between each increment in rollout.
- **Automatic deployment to staging, manual deployment to production:** Sets the `STAGING_ENABLED` and `INCREMENTAL_ROLLOUT_MODE` variables to `1` and `manual`. This means:
 - `master` branch is directly deployed to staging.
 - Manual actions are provided for incremental rollout to production.

Stages of Auto DevOps

The following sections describe the stages of Auto DevOps. Read them carefully to understand how each one works.

Auto Build

Auto Build creates a build of the application in one of two ways:

- If there is a `Dockerfile`, it will use `docker build` to create a Docker image.
- Otherwise, it will use Herokuish [↗](#) and Heroku buildpacks [↗](#) to automatically detect and build the application into a Docker image.

Either way, the resulting Docker image is automatically pushed to the Container Registry ([../user/project/container_registry.html](https://example.gitlab.com/admin/area/settings/project/container_registry.html)) and tagged with the commit SHA.

Important: If you are also using Auto Review Apps and Auto Deploy and choose to provide your own `Dockerfile`, make sure you expose your application to port `5000` as this is the port assumed by the default Helm chart.

Auto Test

Auto Test automatically runs the appropriate tests for your application using Herokuish [↗](#) and Heroku buildpacks [↗](#) by analyzing your project to detect the language and framework. Several languages and frameworks are detected automatically, but if your language is not detected, you may succeed with a custom buildpack. Check the currently supported languages.

Note: Auto Test uses tests you already have in your application. If there are no tests, it's up to you to add them.

Auto Code Quality

STARTER BRONZE

Auto Code Quality uses the Code Quality image (<https://gitlab.com/gitlab-org/security-products/codequality>) to run static analysis and other code checks on the current code. The report is created, and is uploaded as an artifact which you can later download and check out.

In GitLab Starter, differences between the source and target branches are also shown in the merge request widget ([../user/project/merge_requests/code_quality.html](https://gitlab.com/user/project/merge_requests/code_quality.html)).

Auto SAST

ULTIMATE GOLD

Introduced in GitLab Ultimate (<https://about.gitlab.com/pricing/>) 10.3.

Static Application Security Testing (SAST) uses the SAST Docker image (<https://gitlab.com/gitlab-org/security-products/sast>) to run static analysis on the current code and checks for potential security issues. Once the report is created, it's uploaded as an artifact which you can later download and check out.

In GitLab Ultimate, any security warnings are also shown in the merge request widget ([../user/project/merge_requests/sast.html](https://gitlab.com/user/project/merge_requests/sast.html)).

Auto Dependency Scanning

ULTIMATE GOLD

Introduced in GitLab Ultimate (<https://about.gitlab.com/pricing/>) 10.7.

Dependency Scanning uses the Dependency Scanning Docker image (<https://gitlab.com/gitlab-org/security-products/dependency-scanning>) to run analysis on the project dependencies and checks for potential security issues. Once the report is created, it's uploaded as an artifact which you can later download and check out.

Any security warnings are also shown in the merge request widget ([../user/project/merge_requests/dependency_scanning.html](https://gitlab.com/user/project/merge_requests/dependency_scanning.html)).

Auto License Management

ULTIMATE GOLD


Introduced in GitLab Ultimate (<https://about.gitlab.com/pricing/>) 11.0.

License Management uses the License Management Docker image (<https://gitlab.com/gitlab-org/security-products/license-management>) to search the project dependencies for their license. Once the report is created, it's uploaded as an artifact which you can later download and check out.

Any licenses are also shown in the merge request widget ([../user/project/merge_requests/license_management.html](https://gitlab.com/user/project/merge_requests/license_management.html)).

Auto Container Scanning

Introduced in GitLab 10.4.

Vulnerability Static Analysis for containers uses Clair  to run static analysis on a Docker image and checks for potential security issues. Once the report is created, it's uploaded as an artifact which you can later download and check out.

In GitLab Ultimate, any security warnings are also shown in the merge request widget ([../user/project/merge_requests/container_scanning.html](https://gitlab.com/user/project/merge_requests/container_scanning.html)).

Auto Review Apps

Note: This is an optional step, since many projects do not have a Kubernetes cluster available. If the requirements are not met, the job will silently be skipped.

Caution: Your apps should *not* be manipulated outside of Helm (using Kubernetes directly.) This can cause confusion with Helm not detecting the change, and subsequent deploys with Auto DevOps can undo your changes. Also, if you change

something and want to undo it by deploying again, Helm may not detect that anything changed in the first place, and thus not realize that it needs to re-apply the old config.

Review Apps ([../ci/review_apps/index.html](https://ci/review_apps/index.html)) are temporary application environments based on the branch's code so developers, designers, QA, product managers, and other reviewers can actually see and interact with code changes as part of the review process. Auto Review Apps create a Review App for each branch.

The Review App will have a unique URL based on the project name, the branch name, and a unique number, combined with the Auto DevOps base domain. For example, `user-project-branch-1234.example.com`. A link to the Review App shows up in the merge request widget for easy discovery. When the branch is deleted, for example after the merge request is merged, the Review App will automatically be deleted.

Auto DAST

ULTIMATE GOLD

Introduced in GitLab Ultimate (<https://about.gitlab.com/pricing/>) 10.4.

Dynamic Application Security Testing (DAST) uses the popular open source tool OWASP ZAPProxy [↗](#) to perform an analysis on the current code and checks for potential security issues. Once the report is created, it's uploaded as an artifact which you can later download and check out.

In GitLab Ultimate, any security warnings are also shown in the merge request widget ([../user/project/merge_requests/dast.html](https://user/project/merge_requests/dast.html)).

Auto Browser Performance Testing

PREMIUM SILVER

Introduced in GitLab Premium (<https://about.gitlab.com/pricing/>) 10.4.

Auto Browser Performance Testing utilizes the Sitespeed.io container [↗](#) to measure the performance of a web page. A JSON report is created and uploaded as an artifact, which includes the overall performance score for each page. By default, the root page of Review and Production environments will be tested. If you would like to add additional URL's to test, simply add the paths to a file named `.gitlab-urls.txt` in the root directory, one per line. For example:

```
/
/features
/direction
```

In GitLab Premium, performance differences between the source and target branches are shown in the merge request widget ([../user/project/merge_requests/browser_performance_testing.html](https://user/project/merge_requests/browser_performance_testing.html)).

Auto Deploy

Note: This is an optional step, since many projects do not have a Kubernetes cluster available. If the requirements are not met, the job will silently be skipped.

Caution: Your apps should *not* be manipulated outside of Helm (using Kubernetes directly.) This can cause confusion with Helm not detecting the change, and subsequent deploys with Auto DevOps can undo your changes. Also, if you change something and want to undo it by deploying again, Helm may not detect that anything changed in the first place, and thus not realize that it needs to re-apply the old config.

After a branch or merge request is merged into the project's default branch (usually `master`), Auto Deploy deploys the application to a `production` environment in the Kubernetes cluster, with a namespace based on the project name and unique project ID, for example `project-4321`.

Auto Deploy doesn't include deployments to staging or canary by default, but the Auto DevOps template (<https://gitlab.com/gitlab-org/gitlab-ce/blob/master/lib/gitlab/ci/templates/Auto-DevOps.gitlab-ci.yml>) contains job definitions for these tasks if you want to enable them.

You can make use of environment variables to automatically scale your pod replicas.

It's important to note that when a project is deployed to a Kubernetes cluster, it relies on a Docker image that has been pushed to the GitLab Container Registry ([../user/project/container_registry.html](https://gitlab.com/gitlab-org/project/container_registry.html)). Kubernetes fetches this image and uses it to run the application. If the project is public, the image can be accessed by Kubernetes without any authentication, allowing us to have deployments more usable. If the project is private/internal, the Registry requires credentials to pull the image. Currently, this is addressed by providing `CI_JOB_TOKEN` as the password that can be used, but this token will no longer be valid as soon as the deployment job finishes. This means that Kubernetes can run the application, but in case it should be restarted or executed somewhere else, it cannot be accessed again.

Introduced (https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/21955) in GitLab 11.4

Database initialization and migrations for PostgreSQL can be configured to run within the application pod by setting the project variables `DB_INITIALIZE` and `DB_MIGRATE` respectively.

If present, `DB_INITIALIZE` will be run as a shell command within an application pod as a helm post-install hook. Note that this means that if any deploy succeeds, `DB_INITIALIZE` will not be processed thereafter.

If present, `DB_MIGRATE` will be run as a shell command within an application pod as a helm pre-upgrade hook.

For example, in a Rails application:

- `DB_INITIALIZE` can be set to `cd /app && RAILS_ENV=production bin/setup`
- `DB_MIGRATE` can be set to `cd /app && RAILS_ENV=production bin/update`

Note: The `/app` path is the directory of your project inside the docker image as configured by Herokuish [↗](#)

Introduced (https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/19507) in GitLab 11.0.

For internal and private projects a GitLab Deploy Token ([../user/project/deploy_tokens/index.html###gitlab-deploy-token](https://gitlab.com/gitlab-org/project/deploy_tokens/index.html###gitlab-deploy-token)) will be automatically created, when Auto DevOps is enabled and the Auto DevOps settings are saved. This Deploy Token can be used for permanent access to the registry.

Note: **Note** When the GitLab Deploy Token has been manually revoked, it won't be automatically created.

Auto Monitoring

Note: Check the requirements for Auto Monitoring to make this stage work.

Once your application is deployed, Auto Monitoring makes it possible to monitor your application's server and response metrics right out of the box. Auto Monitoring uses Prometheus ([../user/project/integrations/prometheus.html](https://gitlab.com/gitlab-org/project/integrations/prometheus.html)) to get system metrics such as CPU and memory usage directly from Kubernetes ([../user/project/integrations/prometheus_library/kubernetes.html](https://gitlab.com/gitlab-org/project/integrations/prometheus_library/kubernetes.html)), and response metrics such as HTTP error rates, latency, and throughput from the NGINX server ([../user/project/integrations/prometheus_library/nginx_ingress.html](https://gitlab.com/gitlab-org/project/integrations/prometheus_library/nginx_ingress.html)).

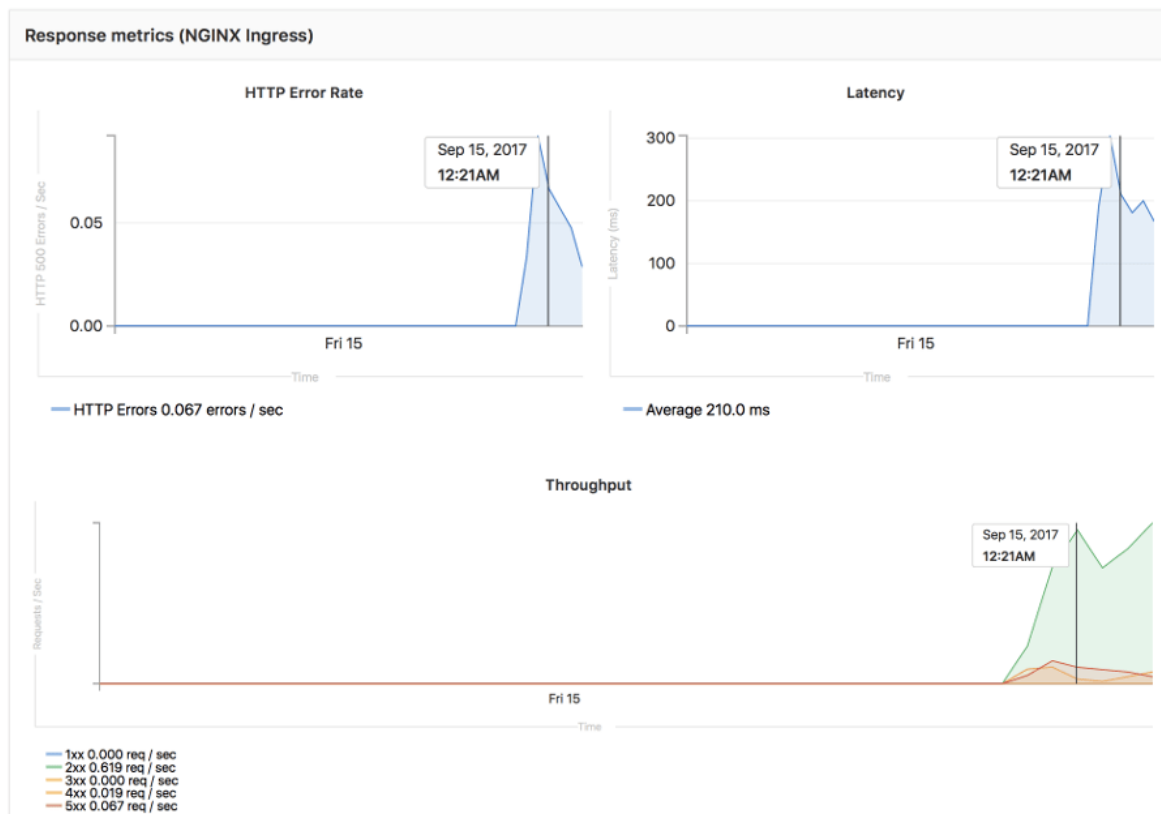
The metrics include:

- **Response Metrics:** latency, throughput, error rate
- **System Metrics:** CPU utilization, memory utilization

In order to make use of monitoring you need to:

1. Deploy Prometheus ([../user/project/integrations/prometheus.html#configuring-your-own-prometheus-server-within-kubernetes](https://gitlab.com/gitlab-org/project/integrations/prometheus.html#configuring-your-own-prometheus-server-within-kubernetes)) into your Kubernetes cluster
2. If you would like response metrics, ensure you are running at least version 0.9.0 of NGINX Ingress and enable Prometheus metrics [↗](#).
3. Finally, annotate [↗](#) the NGINX Ingress deployment to be scraped by Prometheus using `prometheus.io/scrape: "true"` and `prometheus.io/port: "10254"`.

To view the metrics, open the Monitoring dashboard for a deployed environment ([../ci/environments.html#monitoring-environments](https://gitlab.com/gitlab-org/ci/environments.html#monitoring-environments)).

Environment: **production**

(img/auto_monitoring.png)

Customizing

While Auto DevOps provides great defaults to get you started, you can customize almost everything to fit your needs; from custom buildpacks, to Dockerfile s, Helm charts, or even copying the complete CI/CD configuration into your project to enable staging and canary deployments, and more.

Custom buildpacks

If the automatic buildpack detection fails for your project, or if you want to use a custom buildpack, you can override the buildpack(s) using a project variable or a `.buildpacks` file in your project:

- **Project variable** - Create a project variable `BUILDPACK_URL` with the URL of the buildpack to use.
- **.buildpacks file** - Add a file in your project's repo called `.buildpacks` and add the URL of the buildpack to use on a line in the file. If you want to use multiple buildpacks, you can enter them in, one on each line.

⚠ Caution: Using multiple buildpacks isn't yet supported by Auto DevOps.

Custom Dockerfile

If your project has a `Dockerfile` in the root of the project repo, Auto DevOps will build a Docker image based on the Dockerfile rather than using buildpacks. This can be much faster and result in smaller images, especially if your Dockerfile is based on Alpine [↗](#).

Custom Helm Chart

Auto DevOps uses Helm [↗](#) to deploy your application to Kubernetes. You can override the Helm chart used by bundling up a chart into your project repo or by specifying a project variable:

- **Bundled chart** - If your project has a `./chart` directory with a `Chart.yaml` file in it, Auto DevOps will detect the chart and use it instead of the default one (<https://gitlab.com/charts/auto-deploy-app>). This can be a great way to control exactly how your application is deployed.


- **Project variable** - Create a project variable ([../ci/variables/README.html#secret-variables](#)) `AUTO_DEVOPS_CHART` with the URL of a custom chart to use.

Customizing `.gitlab-ci.yml`


If you want to modify the CI/CD pipeline used by Auto DevOps, you can copy the Auto DevOps template (<https://gitlab.com/gitlab-org/gitlab-ce/blob/master/lib/gitlab/ci/templates/Auto-DevOps.gitlab-ci.yml>) into your project's repo and edit as you see fit.

Assuming that your project is new or it doesn't have a `.gitlab-ci.yml` file present:

1. From your project home page, either click on the "Set up CI/CD" button, or click on the plus button and (+), then "New file"
2. Pick `.gitlab-ci.yml` as the template type
3. Select "Auto-DevOps" from the template dropdown
4. Edit the template or add any jobs needed
5. Give an appropriate commit message and hit "Commit changes"

 **Tip:** The Auto DevOps template includes useful comments to help you customize it. For example, if you want deployments to go to a staging environment instead of directly to a production one, you can enable the `staging` job by renaming `.staging` to `staging`. Then make sure to uncomment the `when` key of the `production` job to turn it into a manual action instead of deploying automatically.

PostgreSQL database support

In order to support applications that require a database, PostgreSQL  is provisioned by default. The credentials to access the database are preconfigured, but can be customized by setting the associated variables. These credentials can be used for defining a `DATABASE_URL` of the format:

```
postgres://user:password@postgres-host:postgres-port/postgres-database
```


Environment variables


The following variables can be used for setting up the Auto DevOps domain, providing a custom Helm chart, or scaling your application. PostgreSQL can also be customized, and you can easily use a custom buildpack.

Variable	Description
<code>AUTO_DEVOPS_DOMAIN</code>	The Auto DevOps domain; by default set automatically by the Auto DevOps setting.
<code>AUTO_DEVOPS_CHART</code>	The Helm Chart used to deploy your apps; defaults to the one provided by GitLab (https://gitlab.com/charts/auto-deploy-app).
<code>REPLICAS</code>	The number of replicas to deploy; defaults to 1.
<code>PRODUCTION_REPLICAS</code>	The number of replicas to deploy in the production environment. This takes precedence over <code>REPLICAS</code> ; defaults to 1.
<code>CANARY_REPLICAS</code>	The number of canary replicas to deploy for Canary Deployments (https://docs.gitlab.com/ee/user/project/canary_deployments.html); defaults to 1
<code>CANARY_PRODUCTION_REPLICAS</code>	The number of canary replicas to deploy for Canary Deployments (https://docs.gitlab.com/ee/user/project/canary_deployments.html) in the production environment. This takes precedence over <code>CANARY_REPLICAS</code> ; defaults to 1
<code>POSTGRES_ENABLED</code>	Whether PostgreSQL is enabled; defaults to <code>"true"</code> . Set to <code>false</code> to disable the automatic deployment of PostgreSQL.
<code>POSTGRES_USER</code>	The PostgreSQL user; defaults to <code>user</code> . Set it to use a custom username.
<code>POSTGRES_PASSWORD</code>	The PostgreSQL password; defaults to <code>testing-password</code> . Set it to use a custom password.
<code>POSTGRES_DB</code>	The PostgreSQL database name; defaults to the value of <code>\$CI_ENVIRONMENT_SLUG</code> (../ci/variables/README.html#predefined-variables-environment-variables). Set it to use a custom database name.
<code>BUILDPACK_URL</code>	The buildpack's full URL. It can point to either Git repositories or a tarball URL. For Git repositories, it is possible to point to a specific <code>ref</code> , for example

Variable	Description
SAST_CONFIDENCE_LEVEL	The minimum confidence level of security issues you want to be reported; 1 for Low, 2 for Medium, 3 for High; defaults to 3.
DEP_SCAN_DISABLE_REMOTE_CHECKS	Whether remote Dependency Scanning checks are disabled; defaults to "false". Set to "true" to disable checks that send data to GitLab central servers. Read more about remote checks (https://gitlab.com/gitlab-org/security-products/dependency-scanning#remote-checks).
DB_INITIALIZE	From GitLab 11.4, this variable can be used to specify the command to run to initialize the application's PostgreSQL database. It runs inside the application pod.
DB_MIGRATE	From GitLab 11.4, this variable can be used to specify the command to run to migrate the application's PostgreSQL database. It runs inside the application pod.
STAGING_ENABLED	From GitLab 10.8, this variable can be used to define a deploy policy for staging and production environments.

CANARY_ENABLED	From GitLab 11.0, this variable can be used to define a deploy policy for canary environments.
INCREMENTAL_ROLLOUT_MODE	From GitLab 11.4, this variable, if present, can be used to enable an incremental rollout of your application for the production environment. Set to: <ul style="list-style-type: none"> manual, for manual deployment jobs. timed, for automatic rollout deployments with a 5 minute delay each one.
TEST_DISABLED	From GitLab 11.0, this variable can be used to disable the test job. If the variable is present, the job will not be created.
CODE_QUALITY_DISABLED	From GitLab 11.0, this variable can be used to disable the codequality job. If the variable is present, the job will not be created.
SAST_DISABLED	From GitLab 11.0, this variable can be used to disable the sast job. If the variable is present, the job will not be created.
DEPENDENCY_SCANNING_DISABLED	From GitLab 11.0, this variable can be used to disable the dependency_scanning job. If the variable is present, the job will not be created.
CONTAINER_SCANNING_DISABLED	From GitLab 11.0, this variable can be used to disable the sast:container job. If the variable is present, the job will not be created.
REVIEW_DISABLED	From GitLab 11.0, this variable can be used to disable the review and the manual review:stop job. If the variable is present, these jobs will not be created.
DAST_DISABLED	From GitLab 11.0, this variable can be used to disable the dast job. If the variable is present, the job will not be created.
PERFORMANCE_DISABLED	From GitLab 11.0, this variable can be used to disable the performance job. If the variable is present, the job will not be created.


 **Tip:** Set up the replica variables using a project variable ([../ci/variables/README.html#secret-variables](https://docs.gitlab.com/ee/topics/autodevops/index.html#secret-variables)) and scale your application by just redeploying it!

 **Caution:** You should *not* scale your application using Kubernetes directly. This can cause confusion with Helm not detecting the change, and subsequent deploys with Auto DevOps can undo your changes.

Advanced replica variables setup

Apart from the two replica-related variables for production mentioned above, you can also use others for different environments.

There's a very specific mapping between Kubernetes' label named `track`, GitLab CI/CD environment names, and the replicas environment variable. The general rule is: `TRACK_ENV_REPLICAS`. Where:

- `TRACK`: The capitalized value of the `track` Kubernetes label  in the Helm Chart app definition. If not set, it will not be taken into account to the variable name.
- `ENV`: The capitalized environment name of the deploy job that is set in `.gitlab-ci.yml`.

That way, you can define your own `TRACK_ENV_REPLICAS` variables with which you will be able to scale the pod's replicas easily.

In the example below, the environment's name is `qa` and it deploys the track `foo` which would result in looking for the `FOO_QA_REPLICAS` environment variable:


```
QA testing:
  stage: deploy
  environment:
    name: qa
  script:
    - deploy foo
```

The track `foo` being referenced would also need to be defined in the application's Helm chart, like:

```
replicaCount: 1
image:
  repository: gitlab.example.com/group/project
  tag: stable
  pullPolicy: Always
  secrets:
    - name: gitlab-registry
application:
  track: foo
  tier: web
service:
  enabled: true
  name: web
  type: ClusterIP
  url: http://my.host.com/
  externalPort: 5000
  internalPort: 5000
```

Deploy policy for staging and production environments

Introduced (https://gitlab.com/gitlab-org/gitlab-ci-yml/merge_requests/160) in GitLab 10.8.

 **Tip:** You can also set this inside your project's settings.

The normal behavior of Auto DevOps is to use Continuous Deployment, pushing automatically to the `production` environment every time a new pipeline is run on the default branch. However, there are cases where you might want to use a staging environment and deploy to production manually. For this scenario, the `STAGING_ENABLED` environment variable was introduced.

If `STAGING_ENABLED` is defined in your project (e.g., set `STAGING_ENABLED` to `1` as a secret variable), then the application will be automatically deployed to a `staging` environment, and a `production_manual` job will be created for you when you're ready to manually deploy to production.

Deploy policy for canary environments

PREMIUM SILVER

Introduced (https://gitlab.com/gitlab-org/gitlab-ci-yml/merge_requests/171) in GitLab 11.0.

A canary environment (https://docs.gitlab.com/ee/user/project/canary_deployments.html) can be used before any changes are deployed to production.


If `CANARY_ENABLED` is defined in your project (e.g., set `CANARY_ENABLED` to `1` as a secret variable) then two manual jobs will be created:

- `canary` which will deploy the application to the canary environment
- `production_manual` which is to be used by you when you're ready to manually deploy to production.

Incremental rollout to production

PREMIUM SILVER

Introduced (<https://gitlab.com/gitlab-org/gitlab-ee/issues/5415>) in GitLab 10.8.

 **Tip:** You can also set this inside your project's settings.

When you have a new version of your app to deploy in production, you may want to use an incremental rollout to replace just a few pods with the latest code. This will allow you to first check how the app is behaving, and later manually increasing the rollout up to 100%.

If `INCREMENTAL_ROLLOUT_MODE` is set to `manual` in your project, then instead of the standard `production` job, 4 different manual jobs (`./ci/pipelines.html#manual-actions-from-the-pipeline-graph`) will be created:

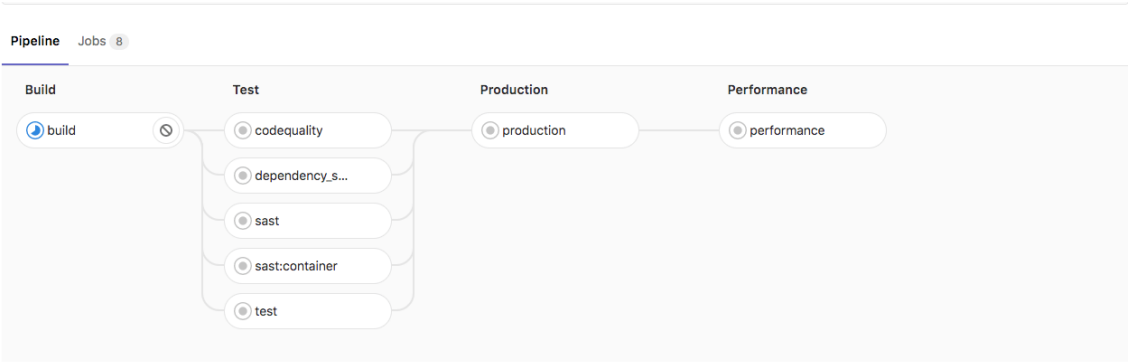
1. `rollout 10%`
2. `rollout 25%`
3. `rollout 50%`
4. `rollout 100%`

The percentage is based on the `REPLICAS` variable and defines the number of pods you want to have for your deployment. If you say `10`, and then you run the `10%` rollout job, there will be `1` new pod + `9` old ones.

To start a job, click on the play icon next to the job's name. You are not required to go from `10%` to `100%`, you can jump to whatever job you want. You can also scale down by running a lower percentage job, just before hitting `100%`. Once you get to `100%`, you cannot scale down, and you'd have to roll back by redeploying the old version using the rollback button (`./ci/environments.html#rolling-back-changes`) in the environment page.

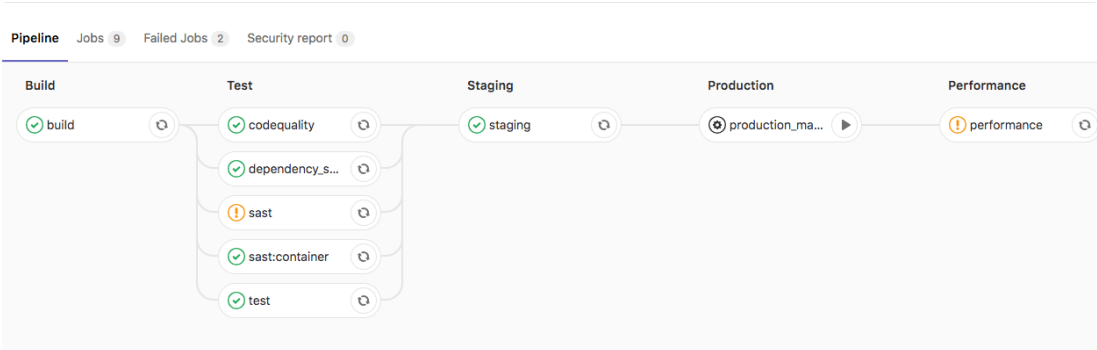
Below, you can see how the pipeline will look if the rollout or staging variables are defined.

Without `INCREMENTAL_ROLLOUT_MODE` and without `STAGING_ENABLED` :



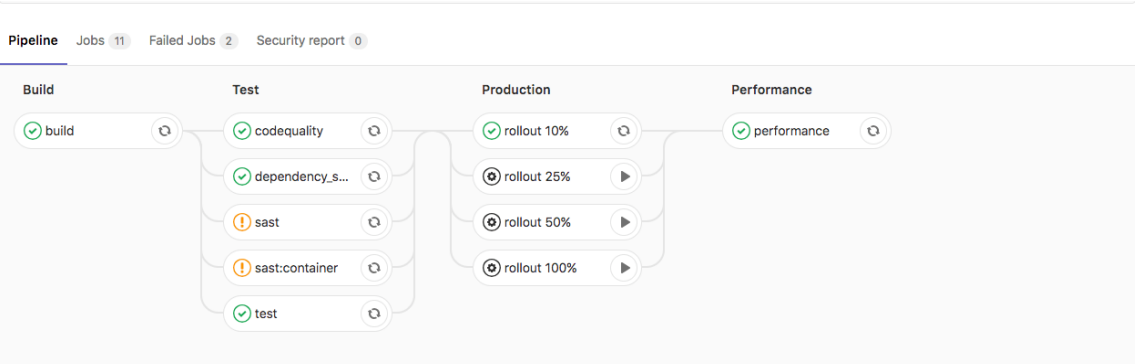
(img/rollout_staging_disabled.png)

Without `INCREMENTAL_ROLLOUT_MODE` and with `STAGING_ENABLED` :



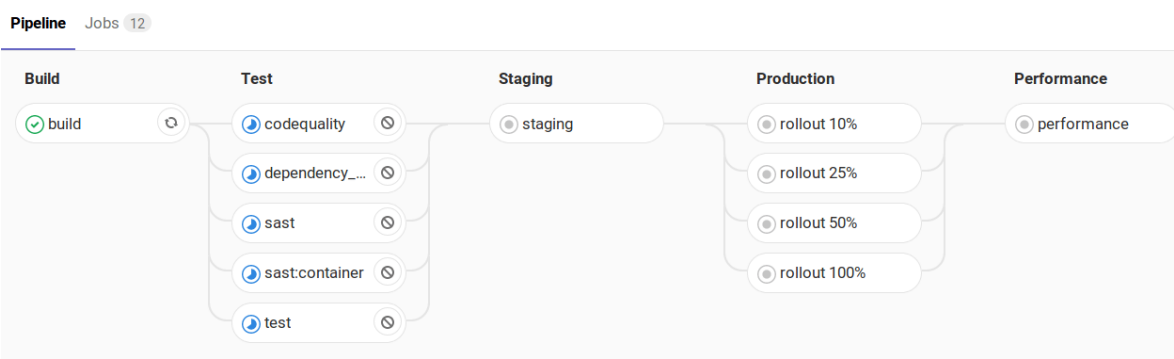
(img/staging_enabled.png)

With INCREMENTAL_ROLLOUT_MODE set to manual and without STAGING_ENABLED :



(img/rollout_enabled.png)

With INCREMENTAL_ROLLOUT_MODE set to manual and with STAGING_ENABLED




(img/rollout_staging_enabled.png)

⚠ Caution: Before GitLab 11.4 this feature was enabled by the presence of the `INCREMENTAL_ROLLOUT_ENABLED` environment variable. This configuration is deprecated and will be removed in the future.

Timed incremental rollout to production

PREMIUM SILVER

Introduced (<https://gitlab.com/gitlab-org/gitlab-ee/issues/7545>) in GitLab 11.4.



 **Tip:** You can also set this inside your project's settings.

This configuration based on incremental rollout to production.

Everything behaves the same way, except:

- It's enabled by setting the `INCREMENTAL_ROLLOUT_MODE` variable to `timed`.
- Instead of the standard `production` job, the following jobs with a 5 minute delay between each are created:
 1. `timed rollout 10%`
 2. `timed rollout 25%`
 3. `timed rollout 50%`
 4. `timed rollout 100%`

Currently supported languages

 **Note:** Not all buildpacks support Auto Test yet, as it's a relatively new enhancement. All of Heroku's officially supported languages  support it, and some third-party buildpacks as well e.g., Go, Node, Java, PHP, Python, Ruby, Gradle, Scala, and Elixir all support Auto Test, but notably the multi-buildpack does not.


As of GitLab 10.0, the supported buildpacks are:

- heroku-buildpack-multi	v1.0.0
- heroku-buildpack-ruby	v168
- heroku-buildpack-nodejs	v99
- heroku-buildpack-clojure	v77
- heroku-buildpack-python	v99
- heroku-buildpack-java	v53
- heroku-buildpack-gradle	v23
- heroku-buildpack-scala	v78
- heroku-buildpack-play	v26
- heroku-buildpack-php	v122
- heroku-buildpack-go	v72
- heroku-buildpack-erlang	fa17af9
- buildpack-nginx	v8

Limitations

The following restrictions apply.

Private project support

 **Caution:** Private project support in Auto DevOps is experimental.

When a project has been marked as private, GitLab's Container Registry (`../user/project/container_registry.html`) requires authentication when downloading containers. Auto DevOps will automatically provide the required authentication information to Kubernetes, allowing temporary access to the registry. Authentication credentials will be valid while the pipeline is running, allowing for a successful initial deployment.

After the pipeline completes, Kubernetes will no longer be able to access the Container Registry. **Restarting a pod, scaling a service, or other actions which require on-going access to the registry may fail.** On-going secure access is planned for a subsequent release.

Troubleshooting

- Auto Build and Auto Test may fail in detecting your language/framework. There may be no buildpack for your application, or your application may be missing the key files the buildpack is looking for. For example, for ruby apps, you must have a `Gemfile` to be properly detected, even though it is possible to write a Ruby app without a `Gemfile`. Try specifying a custom buildpack.
- Auto Test may fail because of a mismatch between testing frameworks. In this case, you may need to customize your `.gitlab-ci.yml` with your test commands.

Disable the banner instance wide

If an administrator would like to disable the banners on an instance level, this feature can be disabled either through the console:

```
sudo gitlab-rails console
```

Then run:

```
Feature.get(:auto_devops_banner_disabled).enable
```

Or through the HTTP API with an admin access token:

```
curl --data "value=true" --header "PRIVATE-TOKEN: personal_access_token" https://gitlab.example.com/api/v4/features/auto_
```

★ Help and feedback

[📄 DOCS FEEDBACK](#)[🚀 PRODUCT FEEDBACK](#)[🔗 GET HELP](#)[📈 GET MORE FEATURES](#)

If you **spot an error or a need for improvement** and would like to **fix it yourself** in a merge request →

[✎ EDIT THIS PAGE \(https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/topics/autodevops/index.md\)](https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/topics/autodevops/index.md)

If you would like to **suggest an improvement to this doc** →

[🔗 CREATE AN ISSUE \(https://gitlab.com/gitlab-org/gitlab-ce/issues/new?issue\[title\]=Docs%20feedback:%20Write%20your%20title&issue\[description\]=Link%20the%20doc%20and%20de](https://gitlab.com/gitlab-org/gitlab-ce/issues/new?issue[title]=Docs%20feedback:%20Write%20your%20title&issue[description]=Link%20the%20doc%20and%20de)

If you want to give **quick and simple feedback** on this doc →

[🗨 SHOW AND POST COMMENTS](#)

Products

Features
Installation
GitLab.com
Pricing
Releases

Services

Resellers
Services


Community

Resources
Events

- Core Team
- Contributors
- Find a Speaker
- Documentation
- Getting Help
- Contributing
- Applications
- Hall of Fame

Company

- Source Code
- Blog
- Customers
- Press and Logos
- Shop
- About Us
- Team
- Direction
- Handbook
- Jobs
- Terms
- Privacy Policy
- Contact Us

Created with Nanoc, hosted on GitLab Pages  Cookies Policy
[🔗 Edit this page](#)