This tutorial guides you through configuring the automated canary analysis feature of Spinnaker on Google Kubernetes Engine (/kubernetes-engine/) (GKE).

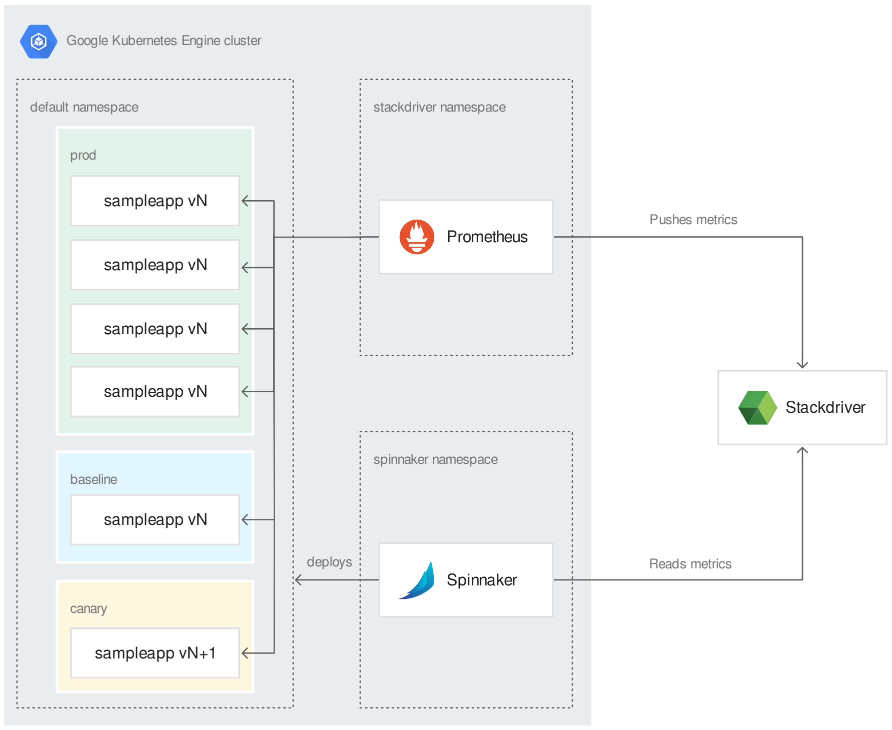
Spinnaker is an open source, continuous delivery system led by Netflix and Google to manage the deployment of apps on different computing platforms, including App Engine, GKE, Compute Engine, AWS, and Azure. Using Spinnaker, you can implement advanced deployment methods, including canary deployments.

In a canary deployment, you expose a new version of your app to a small portion of your production traffic and analyze its behavior before going ahead with the full deployment. This lets you mitigate risks before deploying a new version to all of your users. To use canary deployments, you must accurately compare the behavior of the old and new versions of your app. The differences can be subtle and might take some time to appear. You might also have a lot of different metrics to examine.

To solve those problems, Spinnaker has an automated canary analysis feature: it reads the metrics of both versions from your monitoring system and runs a statistical analysis to automate the comparison. This tutorial shows you how to do an automated canary analysis on an app deployed on GKE and monitored by Stackdriver Monitoring (/monitoring/).

Spinnaker is an advanced app deployment and management platform for organizations with complex deployment scenarios, often with a dedicated release engineering function. You can run this tutorial without prior Spinnaker experience. However, implementing automated canary analysis in production is generally done by teams that already have Spinnaker experience, a strong monitoring system, and that know how to determine if a release is safe.

The app in this tutorial is a simple "Hello World" whose error rate is configured with an environment variable. A pre-built Docker image for this app is provided. As illustrated in the following image, the app exposes metrics in the Prometheus (https://prometheus.io/) format, an open source monitoring system popular in the Kubernetes community, and compatible with Monitoring.



- Install Spinnaker for Google Cloud.
- Deploy an app to GKE without a canary deployment.
- Configure and run a canary deployment of the app.
- Configure the automated canary analysis.
- Test the automated canary analysis.

Important: This tutorial uses the following billable components of Google Cloud:

- GKE
- Monitoring

To generate a cost estimate based on your projected usage, use the pricing calculator (/products/calculator). New Google Cloud users might be eligible for a free trial (/free-trial).

1. Select or create a Google Cloud project.

   GO TO THE MANAGE RESOURCES PAGE (https://console.cloud.google.com/cloud-resource-manager)

2. Enable billing for your project.

ENABLE BILLING (/billing/docs/how-to/modify-project)

3. Create a Workspace.

   GO TO THE Monitoring DOCUMENTATION (/monitoring/accounts/guide#setup-account)

★ **Note:** This tutorial expects the Workspace to have the same name as the Google Cloud project you are using. You can use an existing Workspace in this tutorial only if this is the case.

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. See Cleaning up (#clean-up) for more detail.

In this section, you configure the infrastructure required to complete the tutorial. Run all the terminal commands in this tutorial from Cloud Shell.

Spinnaker for Google Cloud (https://cloud.google.com/docs/ci-cd/spinnaker/spinnaker-for-gcp) gives you a way to set up and manage Spinnaker in a production-ready configuration, optimized for Google Cloud. Spinnaker for Google Cloud sets up many resources (GKE, Memorystore, Cloud Storage buckets and service accounts) required to run Spinnaker in Google Cloud, integrates Spinnaker with related services such as Cloud Build, and provides a Cloud Shell-based management environment for your Spinnaker installations, with helpers and common tools such as `spin` and `hal`.

Note: The installation takes several minutes to complete.

1. In Cloud Shell, open Spinnaker for Google Cloud. This clones the Spinnaker for Google Cloud repository (https://github.com/GoogleCloudPlatform/spinnaker-for-gcp.git) into your Cloud Shell environment and launches the detailed installation instructions.

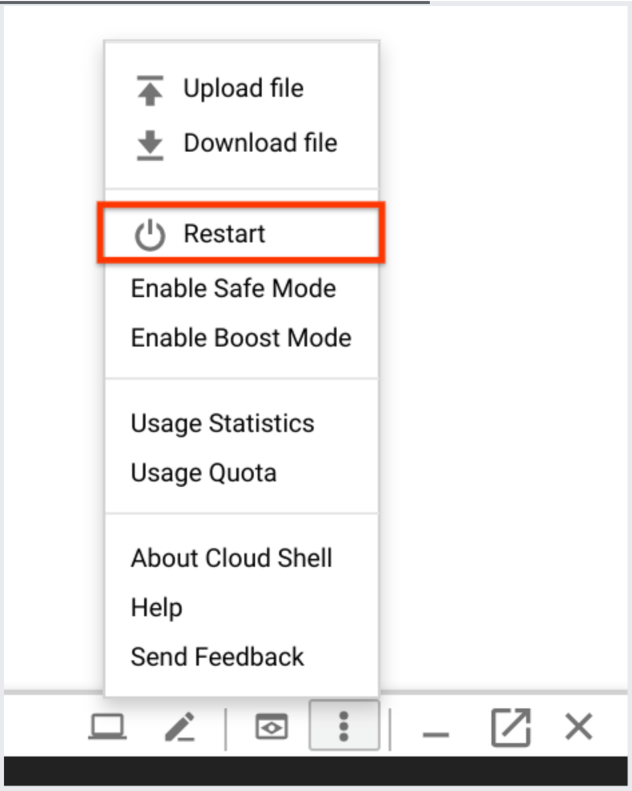
   GO TO CLOUD SHELL (https://console.cloud.google.com/cloudshell/editor?cloudshell_git_repo=https://github.com/GoogleCloudPlatform/spinnaker-for-gc

2. Install Spinnaker for Google Cloud:

★ **Note:** This installation command is only for the basic setup for this tutorial. For a production setup, follow the detailed instructions included with Spinnaker for Google Cloud.

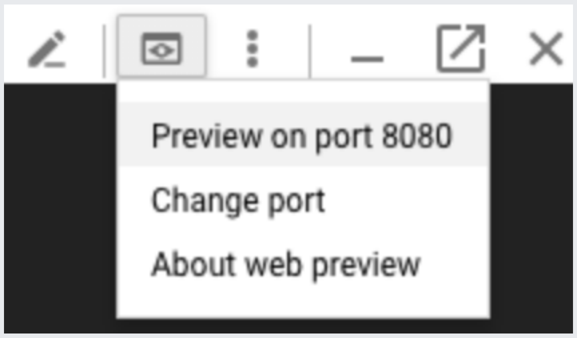
3. Install the Monitoring-Prometheus integration plugin (/monitoring/kubernetes-engine/prometheus):

4. Restart Cloud Shell to load new environment settings.



5. Connect to Spinnaker:

6. In Cloud Shell, select the **Web Preview** icon and select **Preview on port 8080**.



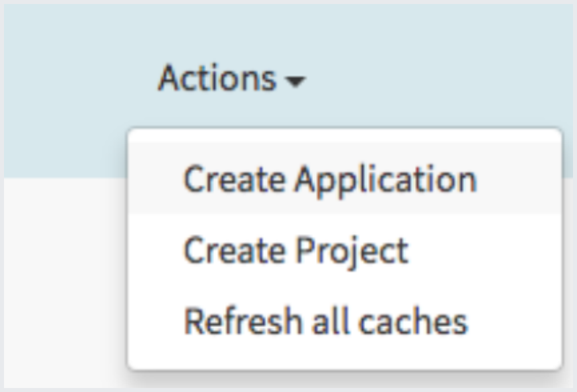
★ **Note:** For now, this Spinnaker instance isn't publicly accessible and only you have access to it, with no authentication. A production Spinnaker instance is a critical component of your infrastructure, so you must properly secure it. Several options are available to you for security and authentication:

- Spinnaker for Google Cloud provides tools (https://github.com/GoogleCloudPlatform/spinnaker-for-gcp/blob/master/scripts/install/provision-spinnaker.md#expose-spinnaker-publicly) to help secure your deployment using IAP with a SSL Certificate.
- Take a look at the security documentation (https://www.spinnaker.io/setup/security/) of Spinnaker.
- Use G Suite as an identity provider (https://www.spinnaker.io/setup/security/authentication/oauth/google/) for Spinnaker authentication.
- Use Google Groups (https://www.spinnaker.io/setup/security/authorization/google-groups/) for Spinnaker authorization.
- Configure a Identity-Aware Proxy (/iap/docs/enabling-kubernetes-howto) in front of Spinnaker to further control who has access to it.

In this section, you configure Spinnaker to deploy an app in the GKE cluster.

Before you deploy, you create the Spinnaker app.

1. In Spinnaker, select **Actions**, and then select **Create Application**.



2. In the **New Application** dialog, enter the following values:
   - **Name**: `sampleapp`
   - **Owner Email**: `[example@example.com]`

Warning: The app *must* be named **`sampleapp`** for the rest of this tutorial to work.

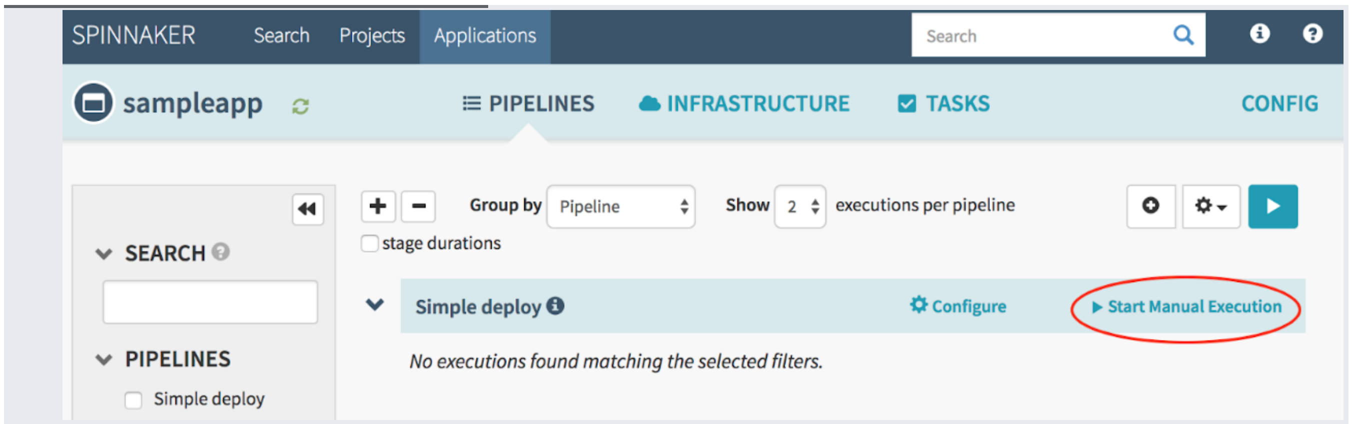
3. Select **Create**.

You are now in the sampleapp of Spinnaker. It isn't configured yet, so most of the tabs are empty.

In this section, you first deploy the app with a simple Spinnaker pipeline that takes a `successRate` parameter to create a GKE Deployment with four pods. Those pods throw errors randomly at a rate corresponding to the `successRate` parameter. In this tutorial, they throw 500 errors at a rate of `100 - successRate`.

1. In Cloud Shell, create the pipeline with the provided JSON file. The following command posts the JSON definition of the pipeline directly to the Spinnaker API.

2. In the Pipelines section of Spinnaker, a pipeline called `Simple deploy` appears. If you don't see it, reload the page. Select **Start Manual Execution**.



3. In the **Confirm Execution** window, select a **Success Rate** of **70**, and then select **Run**. After a few seconds, the pipeline successfully deploys the configuration of the app and four pods.
4. In Cloud Shell, create a pod that makes requests to your new app until the end of the tutorial.

1. To see the behavior of the app, check the logs of the injector:

2. A high number of Internal Server Error messages appear in the logs. To stop following the logs of the injector, press **Ctrl+C** .

Now that your app is deployed and serves traffic, see if it's behaving correctly. Of course, in this tutorial, you already know that it isn't because you deployed the app with only a 70% success rate.

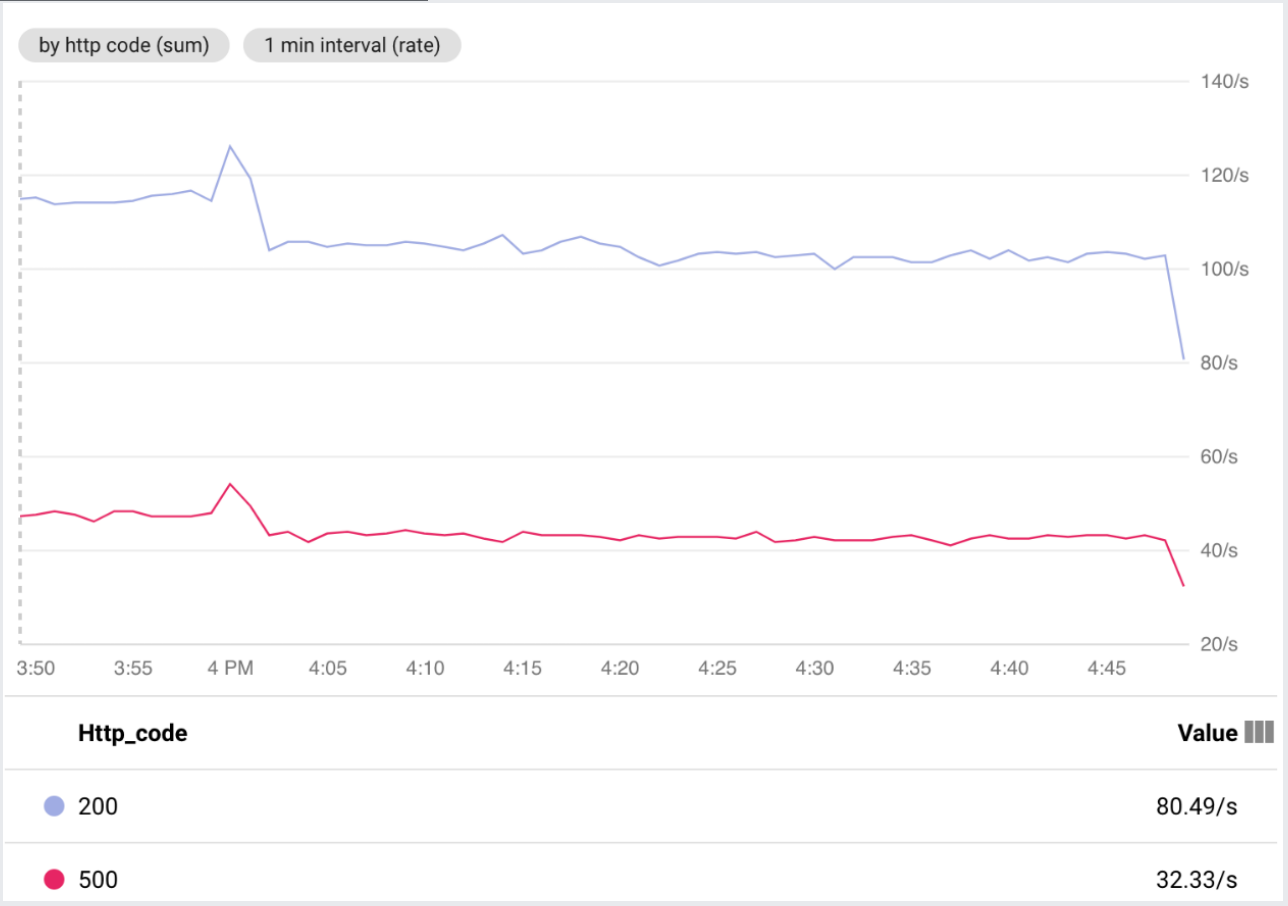
The app exposes a `/metrics` endpoint with metrics in the Prometheus format (/solutions/best-practices-for-operating-containers#metrics_http_endpoint) that are ingested by Monitoring. In this section, you visualize those metrics in Monitoring.

1. In the Google Cloud Console, go to **Monitoring**.

   Go to Monitoring (https://console.cloud.google.com/monitoring)

2. If **Metrics Explorer** is shown in the navigation pane, select
   Metrics Explorer. Otherwise, select **Resources** and then select **Metrics Explorer**.
3. Ensure **Metric** is the selected tab.
4. Select the box labeled **Find resource type and metric**, and enter `external.googleapis.com/prometheus/requests`.
5. To refine the graph, in the **Group By** field, enter `http_code`.

   In the following graph, the rates of HTTP requests answered by the app are grouped by HTTP status code:

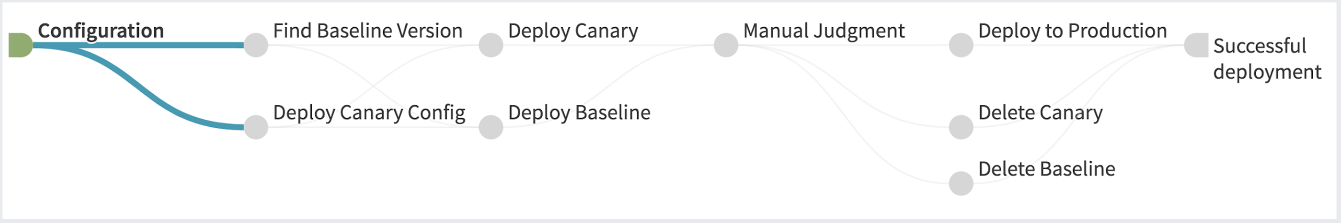


Http_code	Value
● 200	80.49/s
● 500	32.33/s

If you don't have any data in Monitoring, or if you can't find the **external.googleapis.com/prometheus/requests** metric, wait a few minutes for the data to be ingested by Monitoring before reloading Metrics Explorer.

As you can see in the graph, the app currently has an unacceptable error rate—around 30%, as expected. The rest of the tutorial guides you through the setup of a canary deployment pipeline and an automatic analysis to prevent future deployments of an app with such a high error rate.

In this section, you create a canary deployment pipeline, without automated analysis, to test the new version of the app before deploying it fully to production. In the following image, different stages of this pipeline are outlined:



- **Step 0**: Like in the `Simple Deploy` pipeline, the pipeline takes a *Success Rate* parameter as input. This new pipeline uses this parameter to simulate different success rates. This is the *Configuration* of the pipeline.
- **Step 1**: The *Find Baseline Version* stage retrieves the current version of the app running in production from the latest execution of the `Simple Deploy` pipeline. In this tutorial, it retrieves the success rate of the currently deployed app.

  In parallel with the *Find Baseline Version* stage, the *Deploy Canary Config* stage deploys the new success rate configuration for the canary version of the app.

- **Step 2**: The *Deploy Canary* and *Deploy Baseline* stages deploy the two versions for comparison, the new canary version and a baseline version. The canary version uses the configuration created in *Deploy Canary Config* whereas the baseline version uses the configuration used by the production version.

★ **Note:** Read Canary best practices (https://www.spinnaker.io/guides/user/canary/best-practices/#compare-canary-against-baseline-not-against-production) to find out why it's best to compare a canary to a baseline version and not directly to the instances running in production.

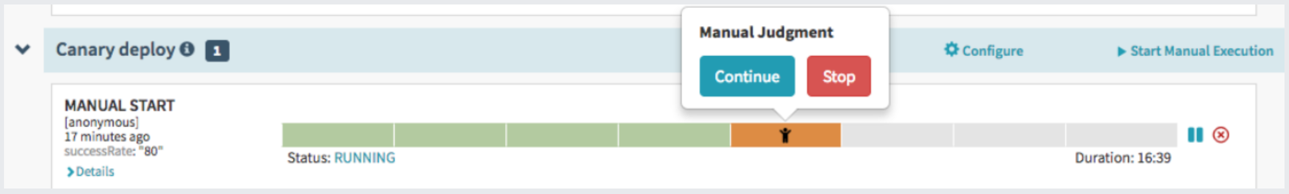
- **Step 3**: The *Manual Judgment* stage stops the pipeline until you continue. During this stage, you can check if the canary version behaves correctly.
- **Step 4**: Once you continue past the *Manual Judgment* stage, both the *Delete Canary* and *Delete Baseline* stages clean up the infrastructure.

  In parallel with the cleanup, the *Deploy to Production* stage is launched and triggers the `Simple Deploy` pipeline with the same *Success Rate* parameter that you gave initially. The same version of the app you tested in a canary is deployed in production.

  The *Deploy to Production stage* is triggered only if you chose to *continue* during the *Manual Judgment* stage.
- **Step 5**: Finally, the *Successful Deployment* stage validates that the whole pipeline is successful. It checks that you gave the go-ahead in the *Manual Judgment* stage and only executes if the *Deploy to Production*, *Delete Canary*, and *Delete Baseline* stages executed successfully.

Now, you can create and run the `Canary Deploy` pipeline.

1. To create the `Canary Deploy` pipeline, run the following command to fetch the ID of the `Simple deploy` pipeline and inject it into the `Canary Deploy` pipeline:
2. If you don't see the `Canary Deploy` pipeline in Spinnaker, reload the **sampleapp** page, and select **Pipelines**.
3. To launch the `Canary Deploy` pipeline:
   a. Select **Start Manual Execution**.
   b. Select a **Success Rate** of **80**.
   c. Select **Run**.
4. When the pipeline reaches the **Manual Judgment** stage, don't select **Continue** yet because you need to compare the canary version with the baseline version.



5. In Cloud Shell, run the `kubectl -n default get pods` command to see the new pods labeled `canary` and `baseline`:

6. In the Google Cloud Console, go to **Monitoring**.

   Go to Monitoring (https://console.cloud.google.com/monitoring)

7. If **Metrics Explorer** is shown in the navigation pane, select **Metrics Explorer**. Otherwise, select **Resources** and then select **Metrics Explorer**.

8. Ensure **Metric** is the selected tab.

9. To display the error rate for both the baseline and the canary, specifying the following parameters:

   a. **Metric**: `external.googleapis.com/prometheus/requests`

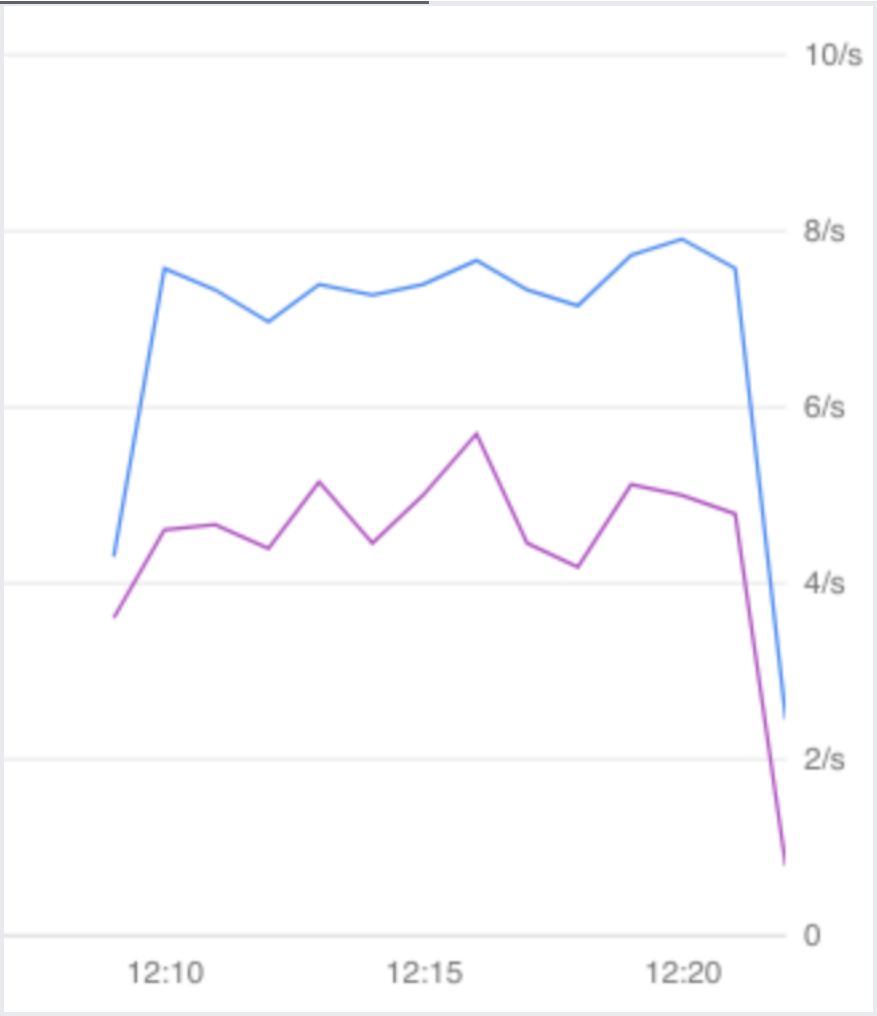
   b. **Filters**:

      i. `http_code` equals `500`

      ii. `version` different (`!=`) from `prod`

   If Monitoring is missing some data, wait a few minutes for it to appear.

10. Compare the canary version (purple in the following graph) with the baseline version (blue in the following graph). Colors might differ in your graph. In this tutorial, the canary version has a lower error rate than the baseline version. Therefore, it is safe to fully deploy the canary version to production. If the canary version didn't have a lower error rate, you might want to stop the deployment at this stage and make some corrections to your app.



11. In Spinnaker, in the **Manual Judgement** dialog, select **Continue**.
12. When the deployment is finished, go back to **Monitoring**.

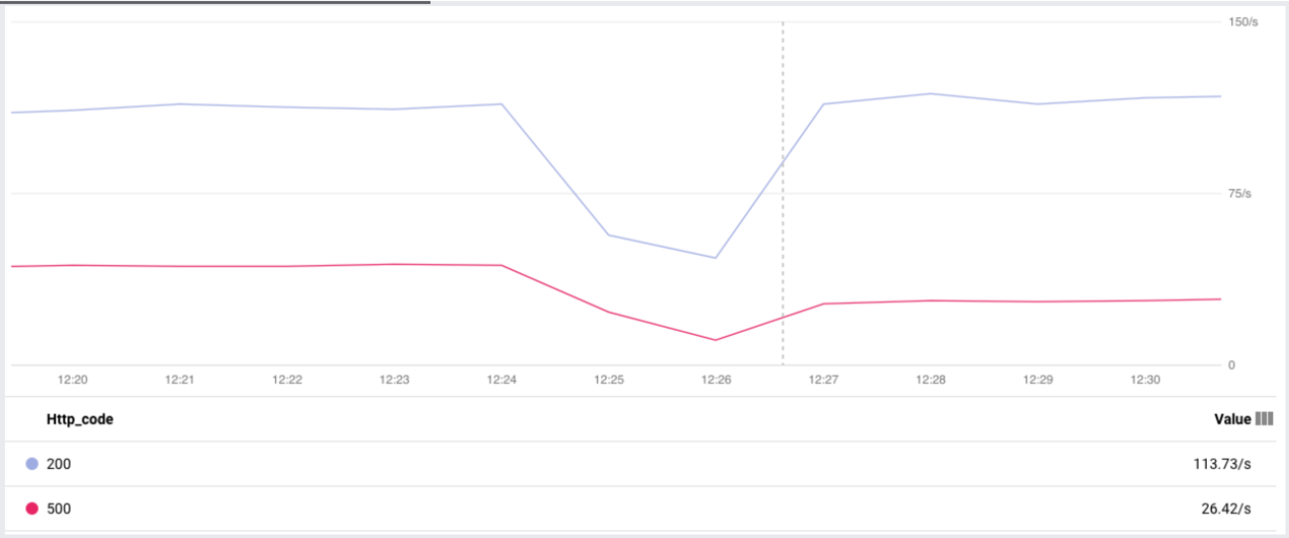
   Go to Monitoring (https://console.cloud.google.com/monitoring)

13. If **Metrics Explorer** is shown in the navigation pane, select **Metrics Explorer**. Otherwise, select **Resources** and then select **Metrics Explorer**.
14. Ensure **Metric** is the selected tab.
15. Select the box labeled **Find resource type and metric**, and then select from the menu or enter the name for the resource and metric. Use the following information to complete the fields for this text box:

   a. For the **Metric**, select or enter `external.googleapis.com/prometheus/requests`.

   b. In the **Group By** field, enter `http_code`.

   In the following graph, the rate of HTTP requests answered by the app is split by HTTP status code:



This graph shows the rate of HTTP codes, 200 and 500, for all pods: production, baseline and canary. Because the canary version had a lower error rate, you deployed it in production. After a short period of time during the deployment, where the total number of requests is slightly lower, you can see that the overall error rate is lowered: the canary version has correctly been deployed in production.

A canary deployment is useful, but in its current implementation, it's a manual process. You have to manually check that the canary behaves as you want before doing a full deployment, and the difference between canary and baseline isn't always clear.

Automating the canary analysis is a good idea: you don't have to do it yourself, and an automated statistical analysis is better suited than humans to detect problems in a set of metrics. In this section, the *Manual Judgement* stage is replaced by an automated canary analysis.

Note: Canary analysis isn't designed to detect errors or failures. It detects the *deviation* of given metrics and makes sure that the behavior of the new version is similar to the old version. Because you might be fine with a specific metric increasing (or decreasing), you can say the *increase* or *decrease* of a metric is OK.

First, in Spinnaker you configure the automated canary analysis feature, called Kayenta (https://cloudplatform.googleblog.com/2018/04/introducing-Kayenta-an-open-automated-canary-analysis-tool-from-Google-and-Netflix.html). To configure Kayenta, use Halyard, the same tool used to configure and deploy Spinnaker.

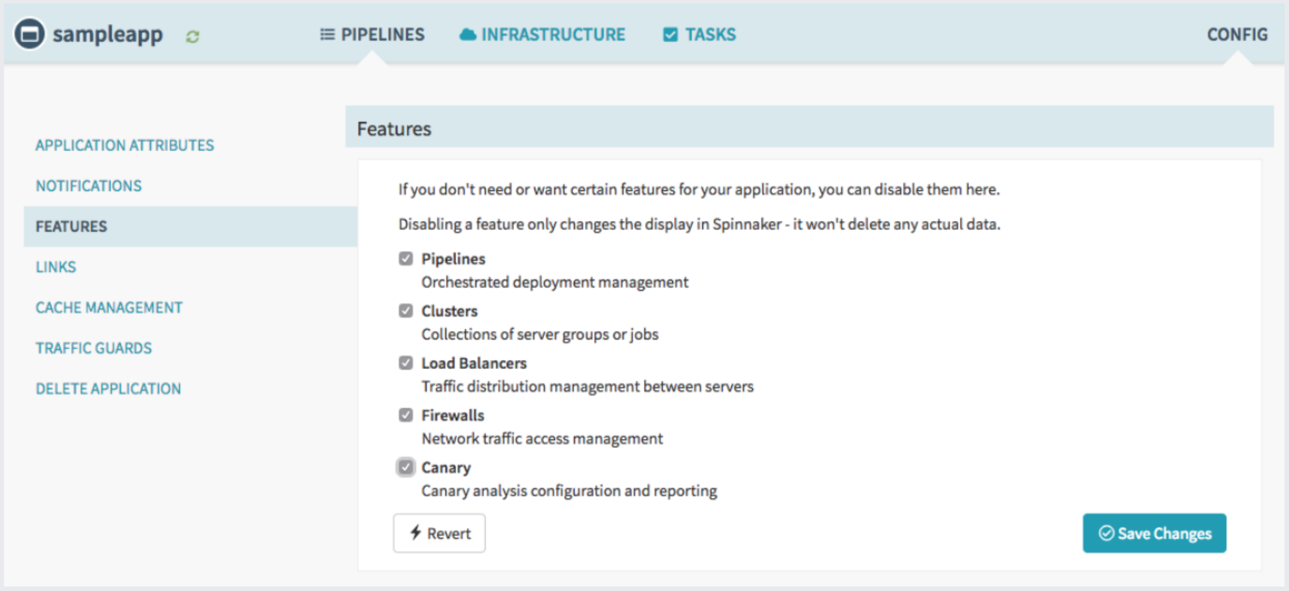
1. Configure Kayenta to use Monitoring as a backend:

2. Apply the new configuration:

   The deployment takes a few minutes to complete.

Now that Kayenta is enabled, configure it for `sampleapp`.

1. In Spinnaker, select **Config**.
2. In the **Features** section, select **Canary**, and then select **Save Changes**.



Features

If you don't need or want certain features for your application, you can disable them here.

Disabling a feature only changes the display in Spinnaker - it won't delete any actual data.

- [✓] **Pipelines** Orchestrated deployment management
- [✓] **Clusters** Collections of server groups or jobs
- [✓] **Load Balancers** Traffic distribution management between servers
- [✓] **Firewalls** Network traffic access management
- [✓] **Canary** Canary analysis configuration and reporting

In Spinnaker, an automated canary analysis runs a statistical test on different metrics and outputs a score. This score can range from 0 to 100 and represents the number of metrics that pass or fail the comparison between the baseline and the canary. You can influence the score, by placing metrics in different groups, with different weights for each group. Depending on the score of the analysis, you might want to go ahead with the deployment or not. If you use a single metric—like in this tutorial—the score can only be 0 (fail) or 100 (pass).

An app can have several canary configurations that can be shared across several apps. A canary configuration has two main elements:

- A set of metrics to analyze (possibly in different groups).
- Marginal and pass thresholds for the score.

In a deployment pipeline, a canary configuration is used during the *Canary Analysis* stage. This stage can include several canary runs. If the score of any run is below the marginal threshold, the stage is stopped and the other runs are not executed. The last run's score needs to be above the pass threshold for the whole stage to be considered successful.

To create a canary configuration, follow these steps:

1. Now that canary is enabled, the **Pipelines** section is replaced with **Delivery** (if you don't see the **Delivery** section, reload Spinnaker). In the **Delivery** section, go to **Canary Configs**.
2. Select **Add Configuration**.
3. For **Configuration Name**, enter `kayenta-test`.
4. In the **Metrics** section, select **Add Metric**.
5. In the **Add Metric** dialog, enter the following values, and then select **OK**:
   - **Name**: `error_rate`
   - **Fail on**: `increase`
   - **Resource Type**: `k8s_container`
   - **Metric type**: `external.googleapis.com/prometheus/requests`
   - **Aligner**: `ALIGN_RATE`

- **Filter Template**: Choose **Create new**.
  - For the **Name** of the new Filter Template, enter: `http_code`
  - For the **Template** of the new Filter Template, enter: `metric.labels.http_code = "500" AND resource.label.pod_name = starts_with("${scope}")`
  - Select **Save**

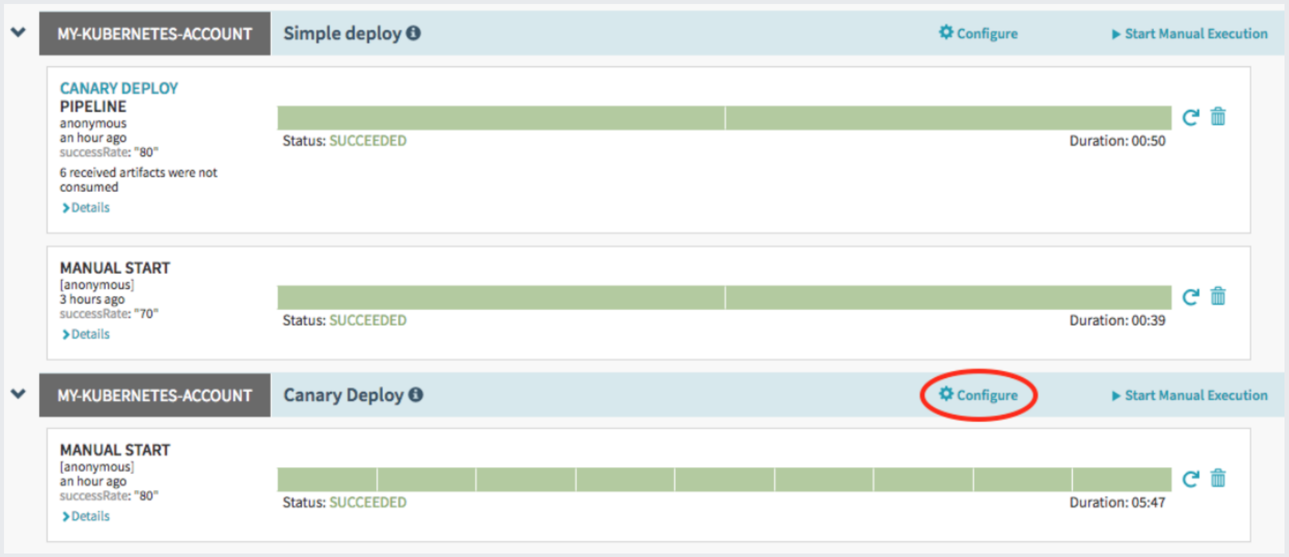
★ **Note:** The **Aligner** is a parameter of the Stackdriver Monitoring API that compares the rate of change of the metrics rather than their absolute value. The app in this tutorial exposes the total number of errors since it started as a metric. Kayenta needs to compare if the rate of those errors changes between two versions of the app.

6. In the **Scoring** section set **Group 1** to **100**.
7. Select **Save Changes**.

Now that you have a canary configuration, modify your existing deployment pipeline to replace the *Manual Judgment* stage with a *Canary Analysis* stage that uses this configuration.

Note: If you make a mistake in this section and you can't recover from it, use the commands listed in the tutorial's repository (https://github.com/spinnaker/spinnaker/tree/master/solutions/kayenta#create-the-automated-canary-deploy-pipeline).

1. Go to **Delivery** > **Pipelines**, and for the `Canary Deploy` pipeline, select **Configure**.



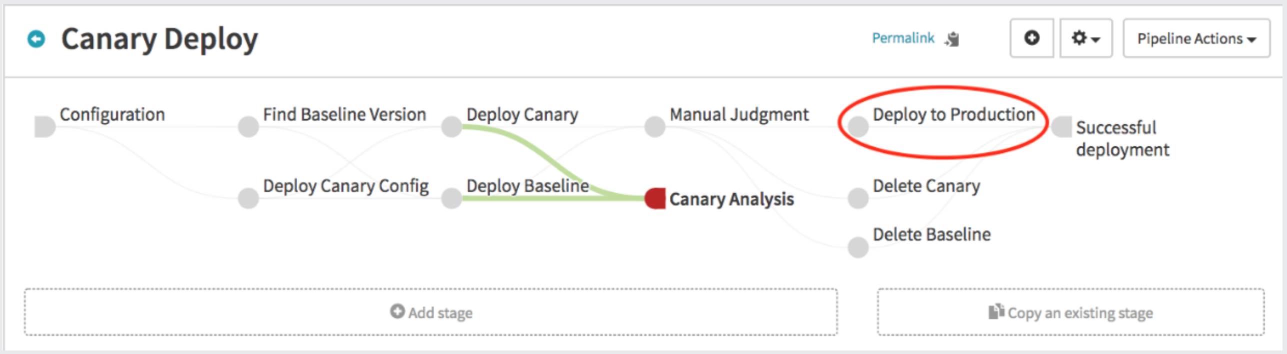
2. Select **Add Stage**.
3. For **Type**, select **Canary Analysis**.
4. In the **Depends On** section, modify your new stage to depend on the following selections:
   - **Deploy Canary**
   - **Deploy Baseline**
5. Fill in the **Canary Analysis Configuration** section with the following values:

Parameter name	Value	Definition
Analysis Type	Real Time (Manual)	The automatic mode, where canary and baseline are created for you, is not yet available for Kubernetes.

Parameter name	Value	Definition
Config Name	kayenta-test	The name of the canary configuration you created earlier.
Lifetime	0 hours 5 minutes	How long the canary analysis should last.
Delay	0	The time we give to the app to warm up before doing the analysis.
Interval	5	The time window Kayenta should use to run a single statistical analysis.
Baseline	sampleapp-baseline	The GKE Deployment Kayenta should use as baseline.
Baseline Location	default	The GKE namespace in which the baseline lives.
Canary	sampleapp-canary	The GKE Deployment Kayenta should use as canary.
Canary Location	default	The GKE namespace in which the canary lives.
Marginal	75	The threshold score for a marignal canary pass.
Pass	95	The threshold score for an overall canary pass.

★ **Note:** A canary analysis is composed of several *canary runs*. A canary run has two parts: a time window—defined by the *interval* parameter—during which metrics are gathered, and an actual statistical analysis of those metrics. Kayenta makes as many runs as possible in the *lifetime* you provide. For example, if you configure an interval of five minutes and a lifetime of one hour, Kayenta makes 12 canary runs. For more information on the right settings for those parameters, see Canary best practices (https://www.spinnaker.io/guides/user/canary/best-practices/#some-configuration-values-to-start-with).

6. In the **Execution Options** section, select **Ignore the failure**. You ignore the failure so you can destroy the baseline and the canary even if the canary analysis failed. Later in the tutorial, you modify the stages to take a potential canary failure into account.
7. In the pipeline's schema, select **Deploy to Production**.

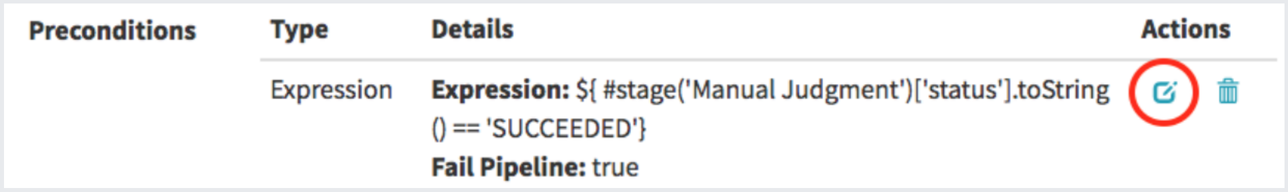


8. Change the **Depends On** section, to the following parameters:
   a. Add **Canary Analysis**.
   b. Remove **Manual Judgment**.
9. To ensure that you deploy to production only if the canary analysis succeeds, change the **Conditional on Expression** parameter.

10. In the pipeline's schema, select **Delete Canary**, and change the **Depends On** section to the following parameters:
    a. Add **Canary Analysis**.
    b. Remove **Manual Judgment**.
11. In the pipeline's schema, select **Delete Baseline**, and change the **Depends On** section.
    a. Add **Canary Analysis**.

b. Remove **Manual Judgment**.

12. To ensure that the whole pipeline fails if the canary analysis fails, in the pipeline's schema, select **Successful deployment**, and then for the existing precondition select the **Edit** icon.



Preconditions	Type	Details	Actions
	Expression	**Expression:** ${ #stage('Manual Judgment')['status'].toString () == 'SUCCEEDED'} **Fail Pipeline:** true	

a. Change the **Expression** to the following:

b. Select **Update**.

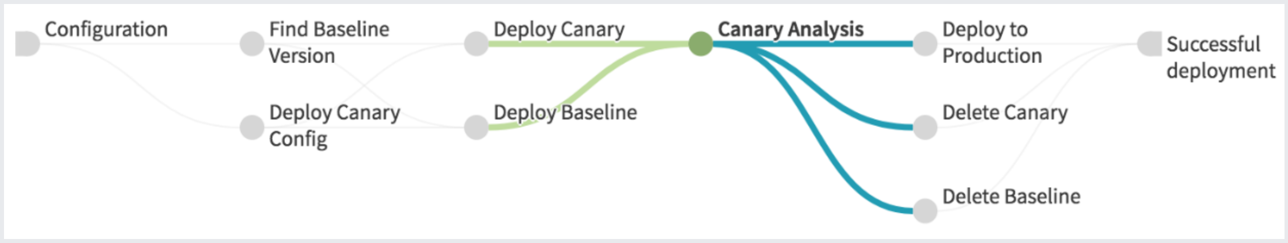
13. Finish replacing the *Manual Judgement* stage with the newly created *Canary Analysis* stage.

a. In the pipeline's schema, select **Manual Judgment**.

b. Select **Remove stage**.

14. Select **Save Changes**.

Your pipeline now looks like the following image:



Now that the automated canary analysis is configured, test the pipeline to ensure it behaves as expected.

1. Go to **Delivery** > **Pipelines**, and for the `Canary Deploy` pipeline, or **Automated Canary Deploy** if you used the CLI, select **Start Manual Execution**.

2. Select a **Success Rate** of **60** and then select **Run**.

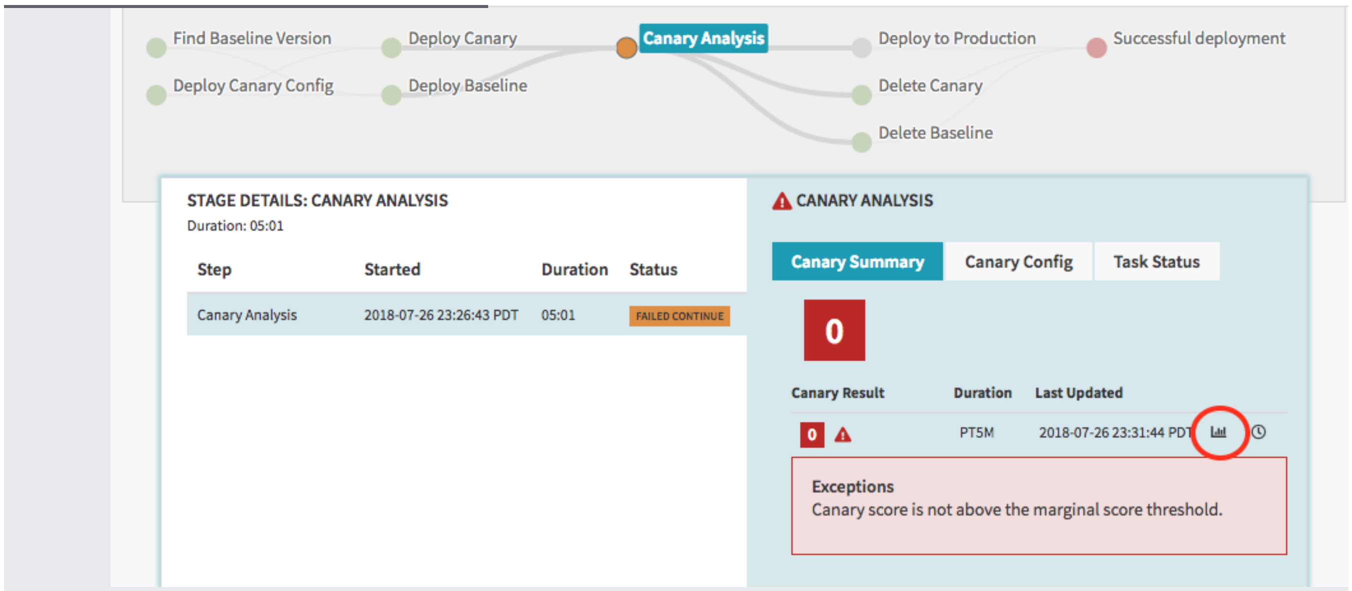
3. To check the current progress of the canary analysis, select **Canary Analysis**, and then select **Task Status**. After a few minutes, the **Canary Analysis** stage fails, because the current success rate in production is `80`. When the **Canary Analysis** stage fails, go to the report for this canary analysis.

a. Select **Canary Analysis**.

b. Select **Canary Summary**.

c. Select the **Report** icon.

On the report page, the error rate is higher for the canary version than it is for the baseline version.



STAGE DETAILS: CANARY ANALYSIS
Duration: 05:01

Step	Started	Duration	Status
Canary Analysis	2018-07-26 23:26:43 PDT	05:01	FAILED CONTINUE

CANARY ANALYSIS

Canary Summary | Canary Config | Task Status

0

Canary Result	Duration	Last Updated
0	PT5M	2018-07-26 23:31:44 PDT

Exceptions
Canary score is not above the marginal score threshold.

4. Repeat the steps in this section, but select a **Success Rate** of **90** for a successful canary analysis.

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

   Go to the Manage resources page (https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

If you want to keep the Google Cloud project you used in this tutorial, delete the individual resources:

1. Delete the GKE cluster.

2. When prompted for confirmation, type `Y`.

- Read more about Spinnaker (https://www.spinnaker.io/).
- Watch how Waze and Netflix use Spinnaker and Kayenta (https://www.youtube.com/watch?v=PLNheBiWOGI).
- See all resources about Continuous Delivery (/solutions/continuous-delivery/).
- Learn how to use Spinnaker with Cloud Build and Cloud Source Repositories (/solutions/continuous-delivery-spinnaker-kubernetes-engine).
- Go through a codelab that covers Continuous Delivery to GKE Using Spinnaker (https://codelabs.developers.google.com/codelabs/cloud-spinnaker-kubernetes-cd/).
- Take a look at spin (https://blog.spinnaker.io/spin-and-roer-managed-pipeline-templates-4fde2951c648), a CLI tool for managing Spinnaker apps and pipelines.
- Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).