Toggle Menu

Home / Guides / Tutorials / Codelabs /

Kubernetes Source To Prod (Manifest Based)

Suggest an Edit

In this codelab you will configure:

- A GitHub repo containing both your code to be deployed, and the Kubernetes manifests that will run your code.
- A set of Spinnaker pipelines to deploy changes to your code, manifests, and application configuration from source to production.

0: Prerequisites

Before we begin, we need to do the following:

- Configure GitHub

  You need some source code and manifests to deploy stored in GitHub. We have a repository you can fork to follow along easily.

- Configure DockerHub

  The source code in GitHub will be configured to build automatically on tag pushes.

- Configure Kubernetes

  Two Kubernetes clusters, one for staging and one for prod.

- Configure Spinnaker

  A running Spinnaker instance, able to deploy to Kubernetes and download artifacts from GitHub.

- Configure Webhooks

  GitHub & DockerHub webhooks pointing at Spinnaker, alerting when commits and docker images are pushed respectively.

Configure GitHub

The code we’ll be deploying is stored here. Feel free to fork this into your own account, and make changes/deploy from there.

> ☞ ***Note****: The manifests in this repository point to a specific docker image. If you want to trigger off of changes made to your own docker image, change the image name to reflect that.*

Configure DockerHub

If you're completely unfamiliar with Docker, start here.

This guide covers how to get your GitHub repository (from above) to trigger Docker builds in DockerHub. We'll be relying on this to automatically push code changes into your staging environment. In the end your repository should look something like this.

> ☞ ***Note****: Before continuing, run the created trigger at least once to both push a* `:latest` *image, as well as validate that your configuration is working. This can be done by pushing a commit to your GitHub repo to trigger a Docker build.*

Configure Kubernetes

Create two clusters following one of the guides here. Once you are finished, make sure that you have an up-to-date `~/.kube/config` file that has entries for both clusters you want to deploy to. Details on kubeconfig files here.

Configure Spinnaker

We will be deploying Spinnaker to one of your Kubernetes clusters. To do so, start by installing halyard.

Choose a storage service

Pick a storage service here, and run the required `hal` commands.

Add your Kubernetes accounts

You will need to configure two Kubernetes accounts. See the Kubernetes contexts created in the prior step using:

```
kubectl config get-contexts
```

The output should look like (although the names may vary):

```
CURRENT   NAME                        CLUSTER        AUTHINFO       NAMESPACE
*         staging-demo-us-central1   staging-demo   staging-demo
          prod-demo-us-central1      prod-demo      prod-demo
```

Record the names of the contexts as `$STAGING_CONTEXT` and `$PROD_CONTEXT`.

Now we will register both contexts with Spinnaker.

```
hal config provider kubernetes account add prod-demo \
  --context $PROD_CONTEXT \
  --provider-version v2

hal config provider kubernetes account add staging-demo \
  --context $STAGING_CONTEXT \
  --provider-version v2
```

Configure GitHub artifact credentials

Make sure to add GitHub as an artifact account. This will allow us to fetch the manifests later.

Deploy Spinnaker

Pick a version & specify that you want to deploy Spinnaker inside the staging cluster:

```
hal config version edit --version $(hal version latest -q)

hal config deploy edit --type distributed --account-name staging-demo
```

And finally, deploy Spinnaker.

```
hal deploy apply
```

Configure webhooks

Now that Spinnaker is running, you need to point both Docker and GitHub webhooks at Spinnaker to send events when Docker images and manifest changes happen respectively.

Give Spinnaker an external endpoint

⚠️ ***This is for the codelab only! Do not do this in production****. We're giving an unsecured Spinnaker an external endpoint to easily do this codelab, and are taking limited measures to ensure only GitHub and Docker can trigger pipelines.* ***Tear down Spinnaker once you're done with this codelab, or remove any firewall changes to your Kubernetes cluster****.*

First, edit the Gate service to bind a node port. This means every node in your Kubernetes cluster will forward traffic from that node port to your Spinnaker gate service. **Your nodes should not be accepting requests from external IPs** by default, so making this change doesn't immediately open Spinnaker to public access.

To do this, first run (this will open the service manifest in your text editor):

```
kubectl edit svc spin-gate -n spinnaker --context $STAGING_CONTEXT
```

and then change the field

```
type: ClusterIP
```

to

```
type: NodePort
```

Next, get the port that `spin-gate` has bound to. You can check this with

```
kubectl get svc spin-gate -n spinnaker --context $STAGING_CONTEXT
```

In my case, I see the port is `31355`, which I record into `$NODE_PORT` (`8084` is the port gate is listening on inside the cluster):

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
spin-gate    NodePort    10.7.255.85    <none>         8084:31355/TCP   32m
```

Now pick any node in the cluster and record its IP as `$NODE_IP`; for the purposes of this codelab, we'll be sending external webhooks to `$NODE_PORT` on that node. In order for these webhooks to work, for this codelab only, open your firewall on that node to all addresses for TCP connections on `$NODE_PORT`. If you were running Spinnaker in production with authentication, only webhooks would be allowed, which you can reject by header or payload. See the webhook guide for more details.

Allow Docker to post build events

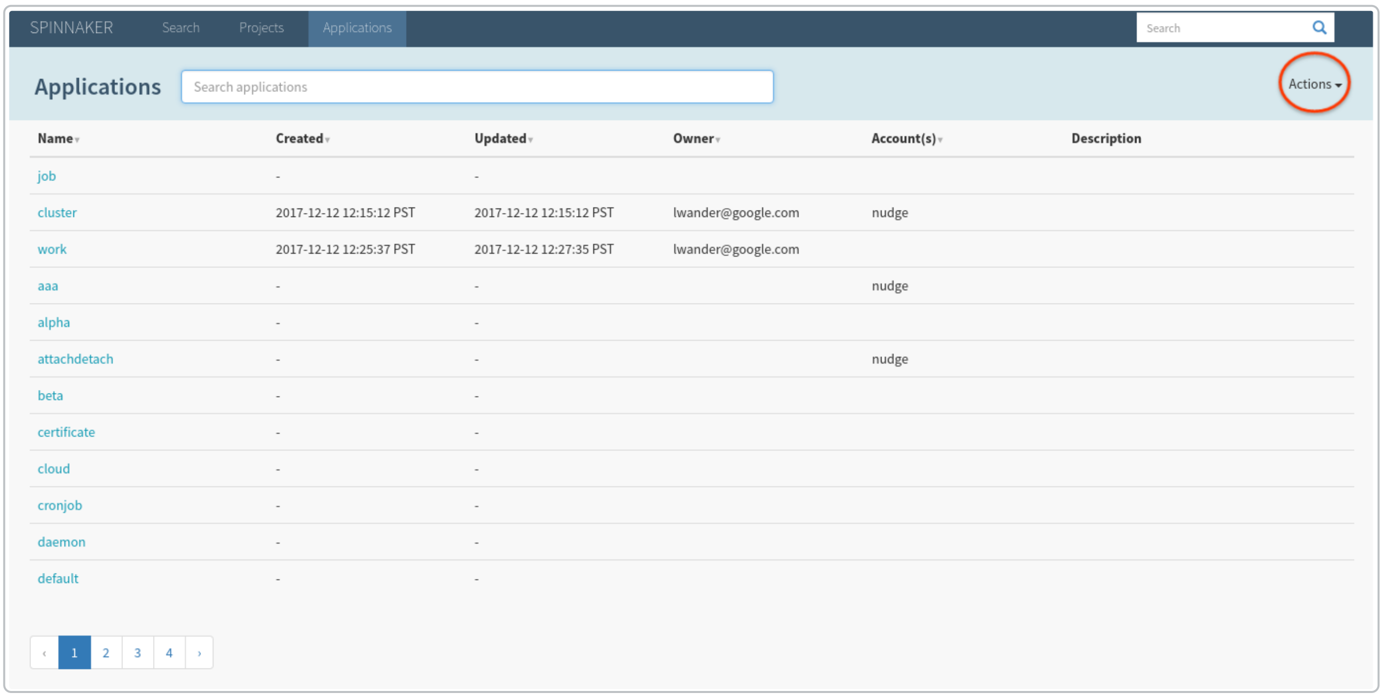
These will be used to trigger pipelines based on new Docker images being published. Follow the steps shown here for your repository. The endpoint you configure must be `http://${NODE_IP}:${NODE_PORT}/webhooks/webhook/dockerhub`.

Allow GitHub to post push events

Follow the steps shown here, where `ENDPOINT=http://${NODE_IP}:${NODE_PORT}`. Keep track of what you pick as the `$SECRET`!

1: Create a Spinnaker application

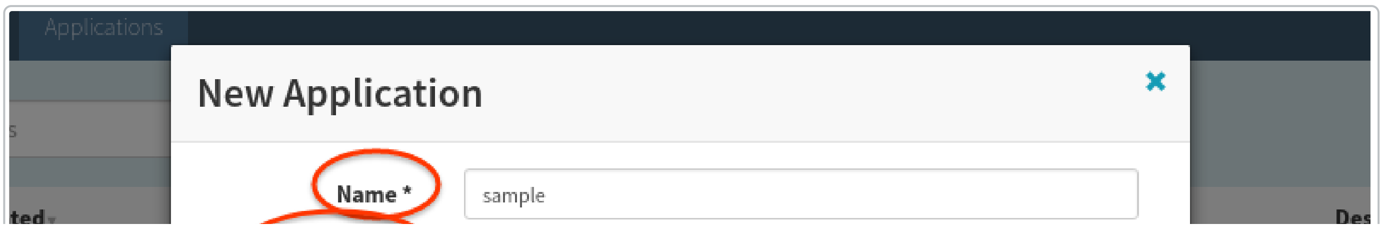
When you first open Spinnaker (if you've followed the above prerequisites) it'll be running on `localhost:9000`) you'll be greeted with the following **Applications** screen.



Name	Created	Updated	Owner	Account(s)	Description
job	-	-			
cluster	2017-12-12 12:15:12 PST	2017-12-12 12:15:12 PST	lwander@google.com	nudge	
work	2017-12-12 12:25:37 PST	2017-12-12 12:27:35 PST	lwander@google.com		
aaa	-	-		nudge	
alpha	-	-			
attachdetach	-	-		nudge	
beta	-	-			
certificate	-	-			
cloud	-	-			
cronjob	-	-			
daemon	-	-			
default	-	-			

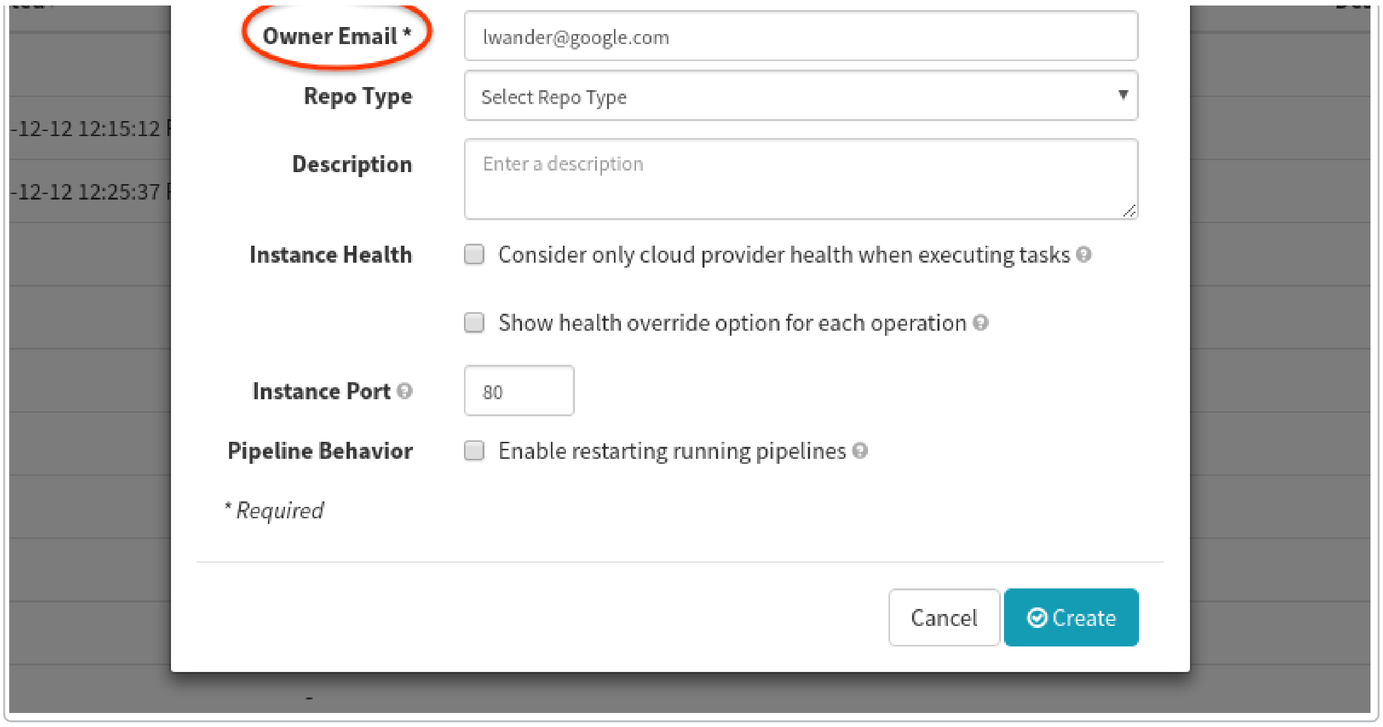
By default, Spinnaker indexes your entire cluster, which explains why the screen is prepopulated with unrelated infrastructure. This can be changed by ommiting namespaces as shown here.

Select **Actions** > **Create Application**, and fill out the form as shown (the owner email will of course be different):



New Application

Name * sample


Owner Email * lwander@google.com

Repo Type: Select Repo Type

Description: Enter a description

Instance Health: ☐ Consider only cloud provider health when executing tasks

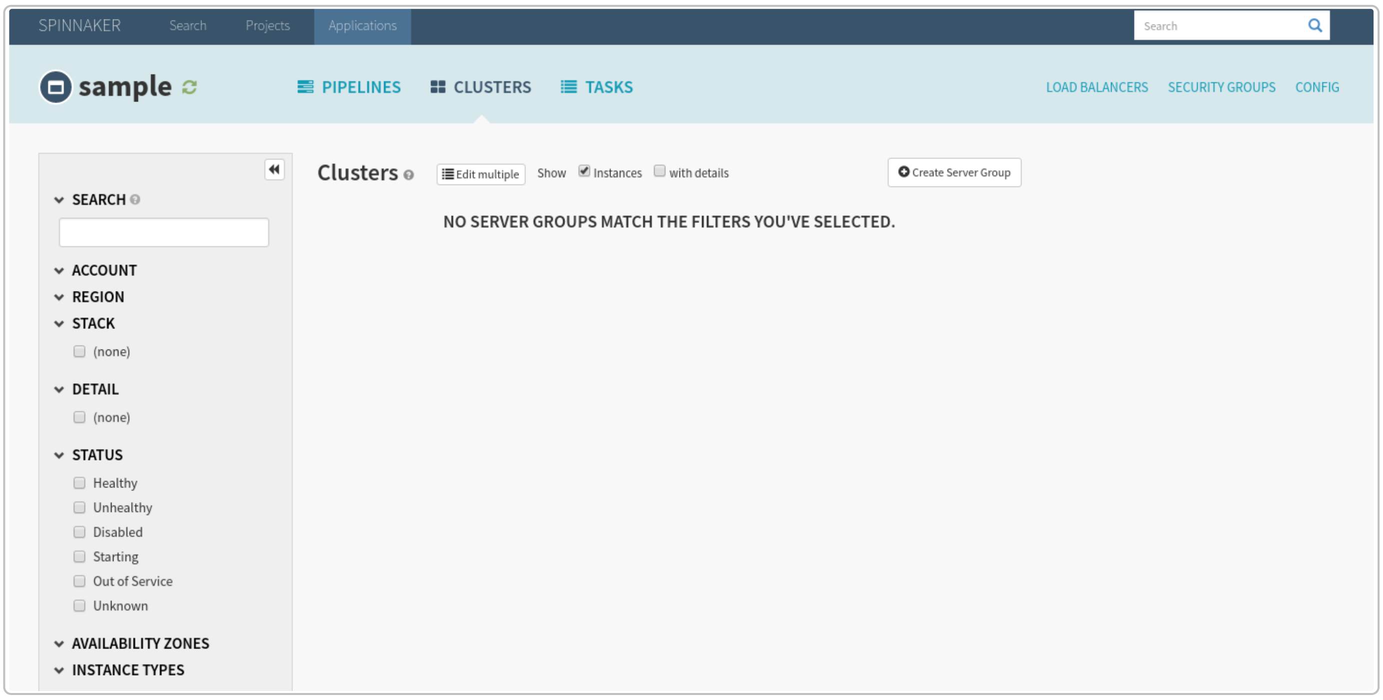
☐ Show health override option for each operation

Instance Port: 80

Pipeline Behavior: ☐ Enable restarting running pipelines

* Required

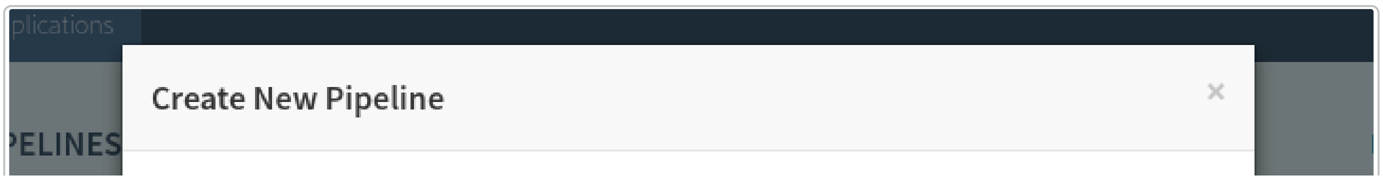
After hitting **Create**, you should be brought to an empty **Clusters** tab:

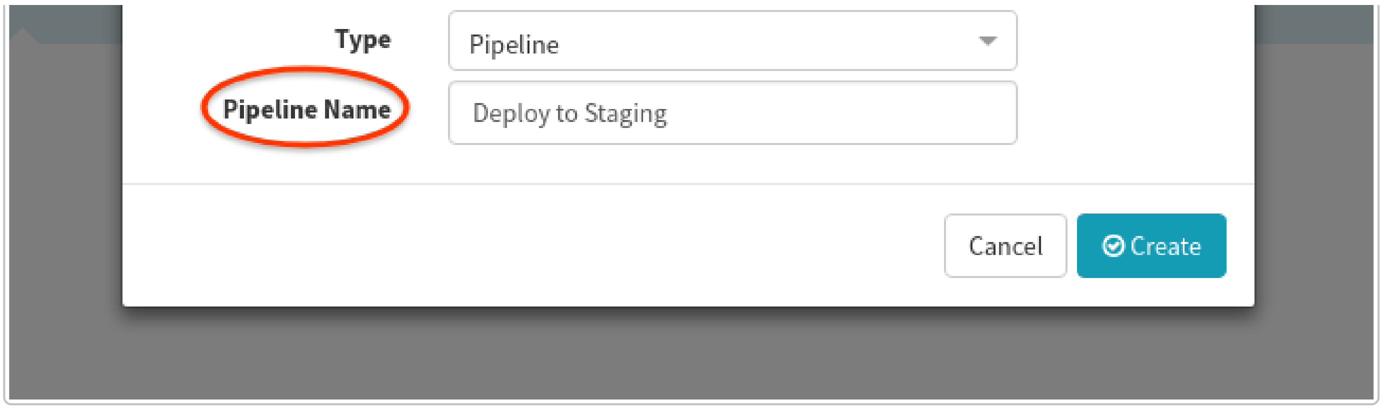


NO SERVER GROUPS MATCH THE FILTERS YOU'VE SELECTED.

2: Create a "Deploy to Staging" pipeline

Let's deploy the manifests and code in our staging cluster by setting up automated pipelines to do so. Start by navigating to **Pipelines** > **Configure a new Pipeline**. Name the pipeline as shown and hit create:

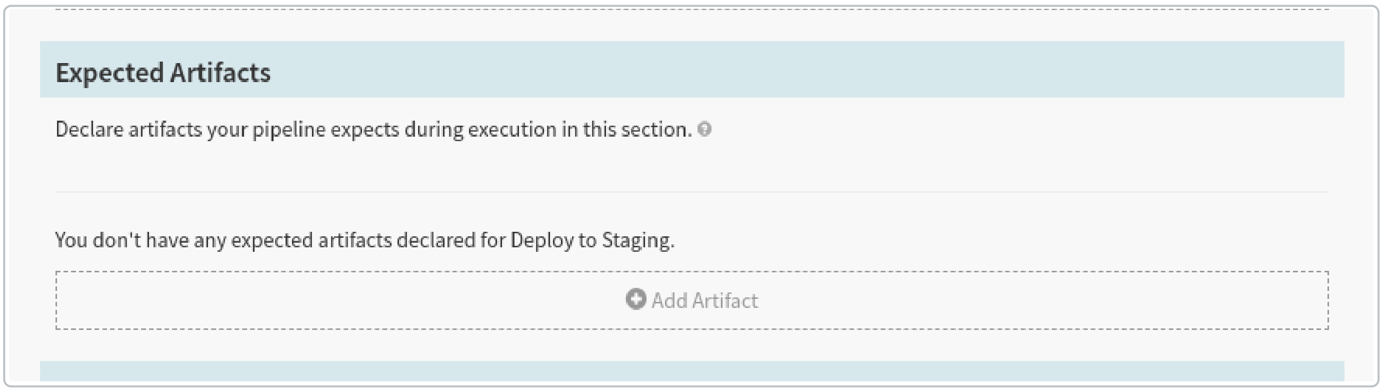




Type	Pipeline
Pipeline Name	Deploy to Staging

At this point we want to add the manifest from GitHub as an expected artifact in this pipeline, meaning we expect each time that this pipeline executes, either a GitHub event will supply us with a new manifest to deploy, or we will use some default or prior manifest.

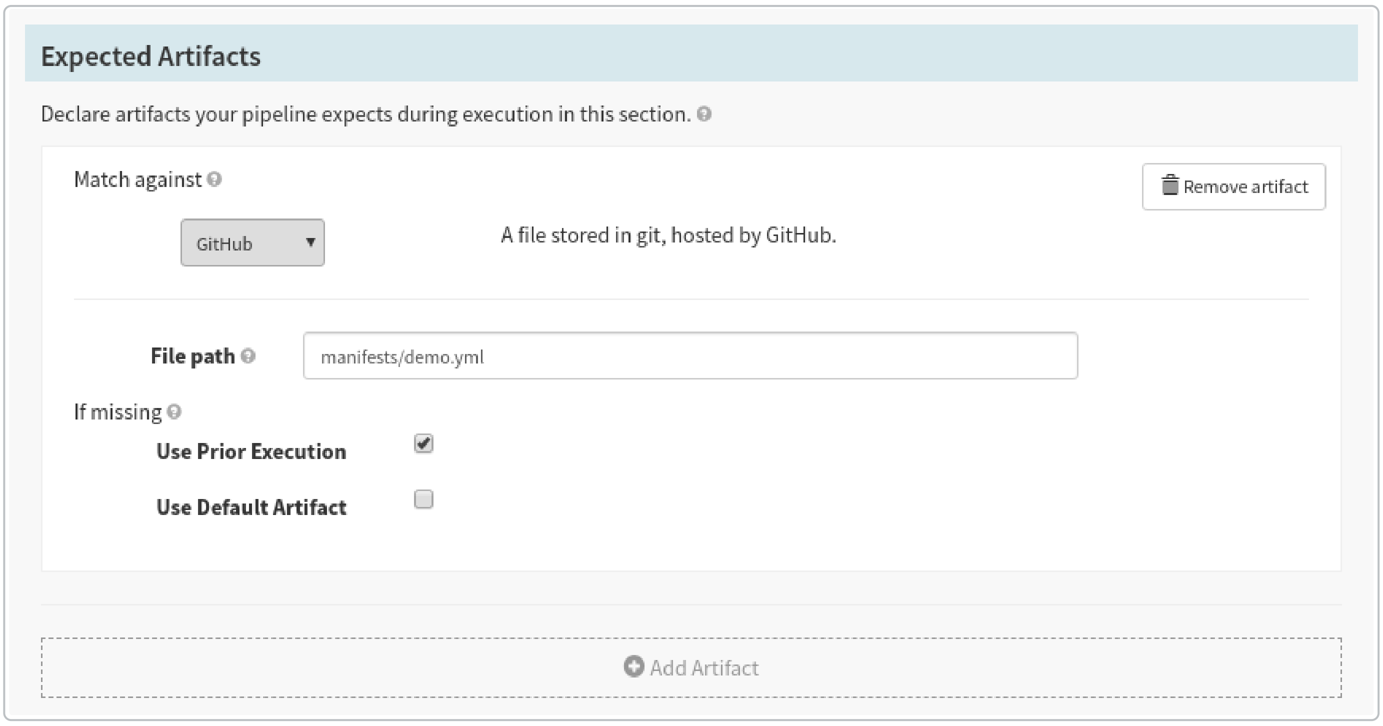
Select **Add Artifact**:


Expected Artifacts

Declare artifacts your pipeline expects during execution in this section.

You don't have any expected artifacts declared for Deploy to Staging.

Select GitHub as the artifact type, and set the **File Path** to `manifests/demo.yml`, and select **Use Prior Execution**, to tell Spinnaker that if no matching artifact is found, to use the last execution's value. (This will be useful later).


Expected Artifacts

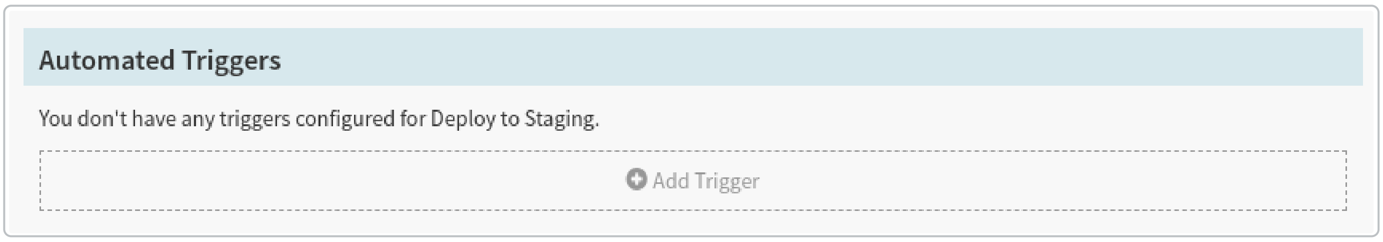
Declare artifacts your pipeline expects during execution in this section.

Match against: GitHub — A file stored in git, hosted by GitHub.

File path: manifests/demo.yml

If missing
- Use Prior Execution ✓
- Use Default Artifact ☐

Next, let's add a GitHub trigger:

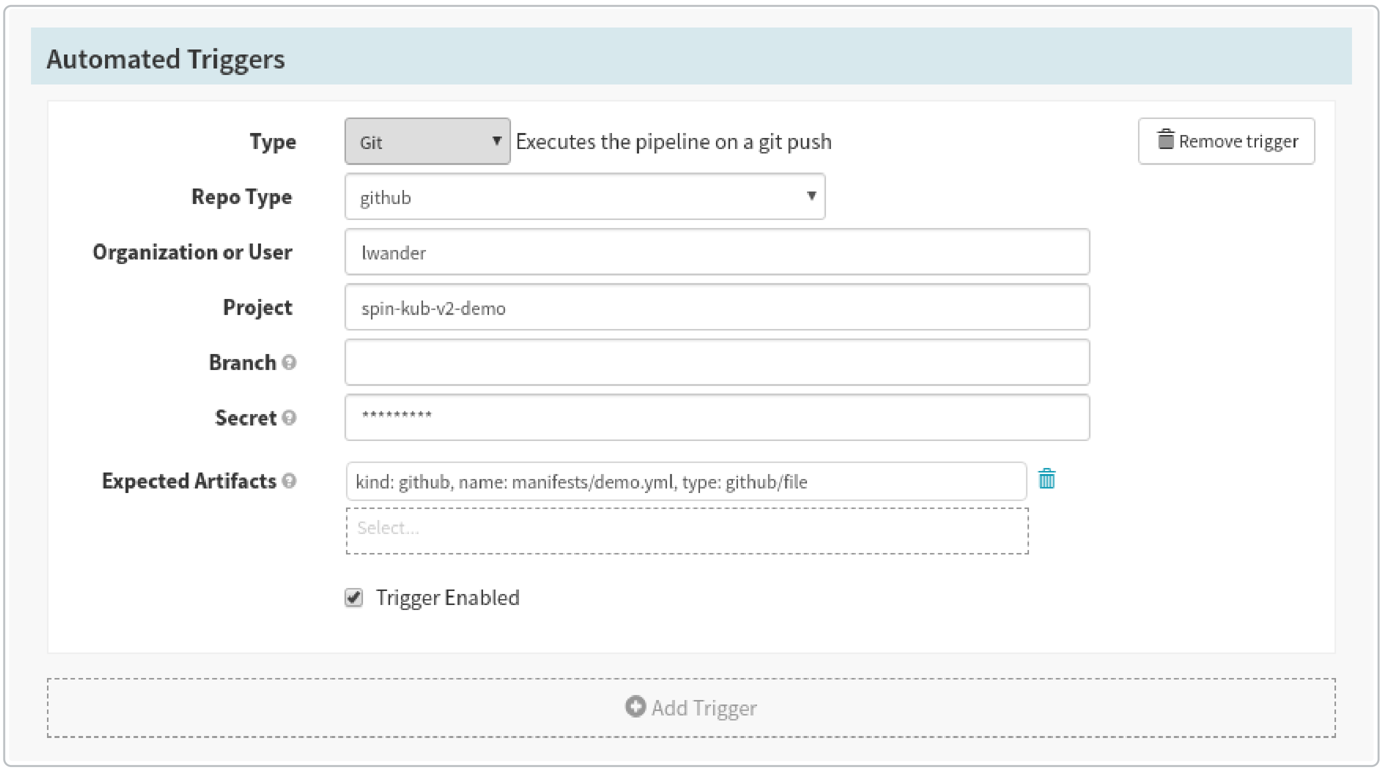


Automated Triggers

You don't have any triggers configured for Deploy to Staging.

Supply the following configuration values:

Field	Value
Type	"Git"
Repo Type	"GitHub"
Organization or User	The user you forked the above code into.
Project	"spin-kub-v2-demo"
Secret	The `$SECRET` chosen above.
Expected Artifacts	Must reference the `manifests/demo.yml` artifact.



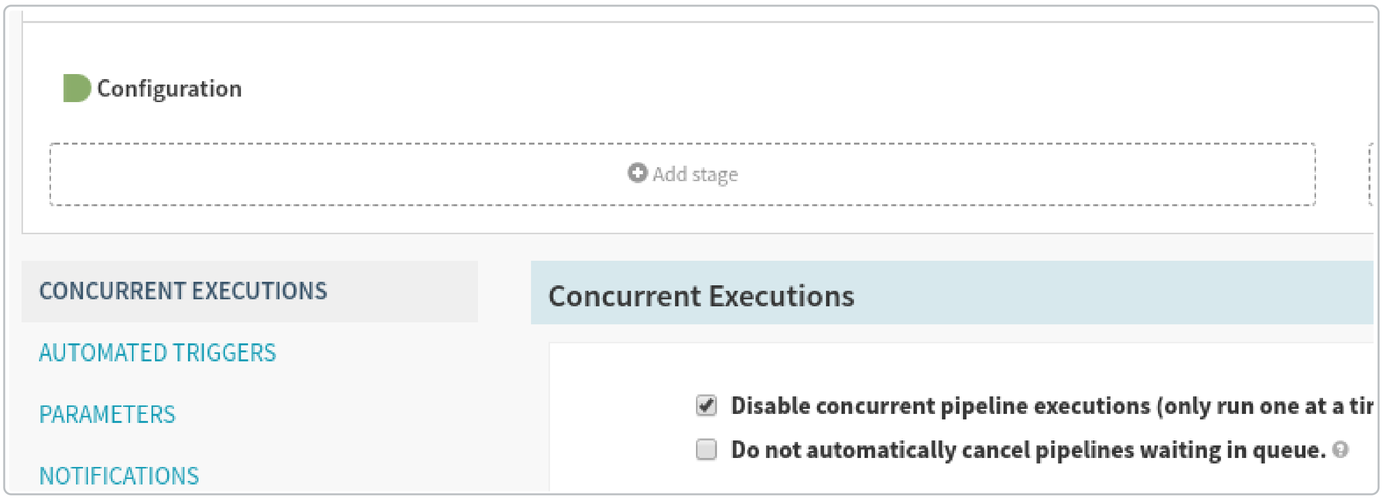
Automated Triggers

Type	Git — Executes the pipeline on a git push
Repo Type	github
Organization or User	lwander
Project	spin-kub-v2-demo
Branch	
Secret	*********
Expected Artifacts	kind: github, name: manifests/demo.yml, type: github/file

We supply the expected artifact to be sure that we only trigger the pipeline when that file changes.

With the trigger configuration in place, let's configure a "Deploy manifest" stage.

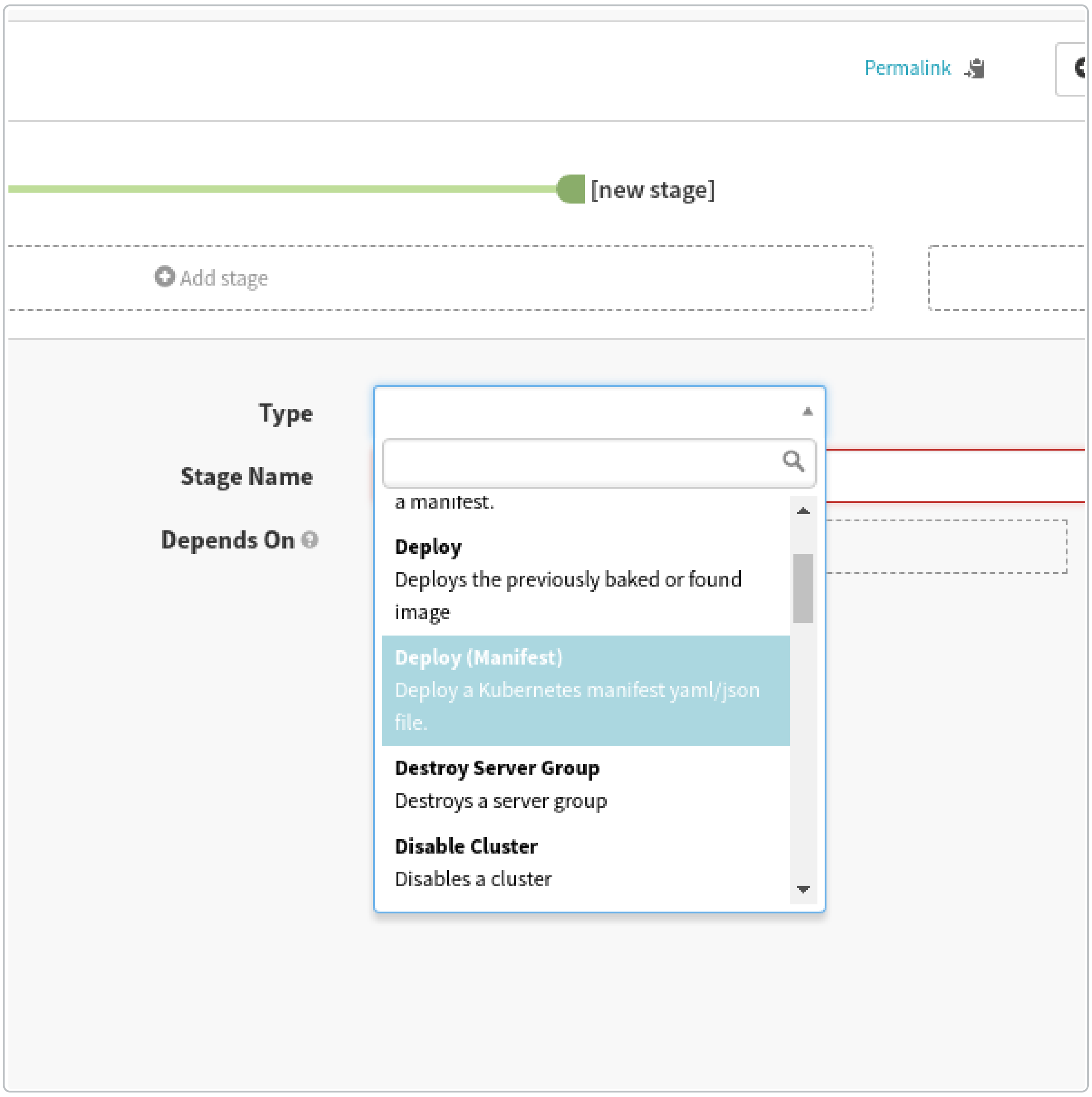
First add a stage:



Concurrent Executions

- [✓] Disable concurrent pipeline executions (only run one at a tir
- [ ] Do not automatically cancel pipelines waiting in queue.

Then select the “Deploy (Manifest)” stage type:



a manifest.

Deploy
Deploys the previously baked or found image

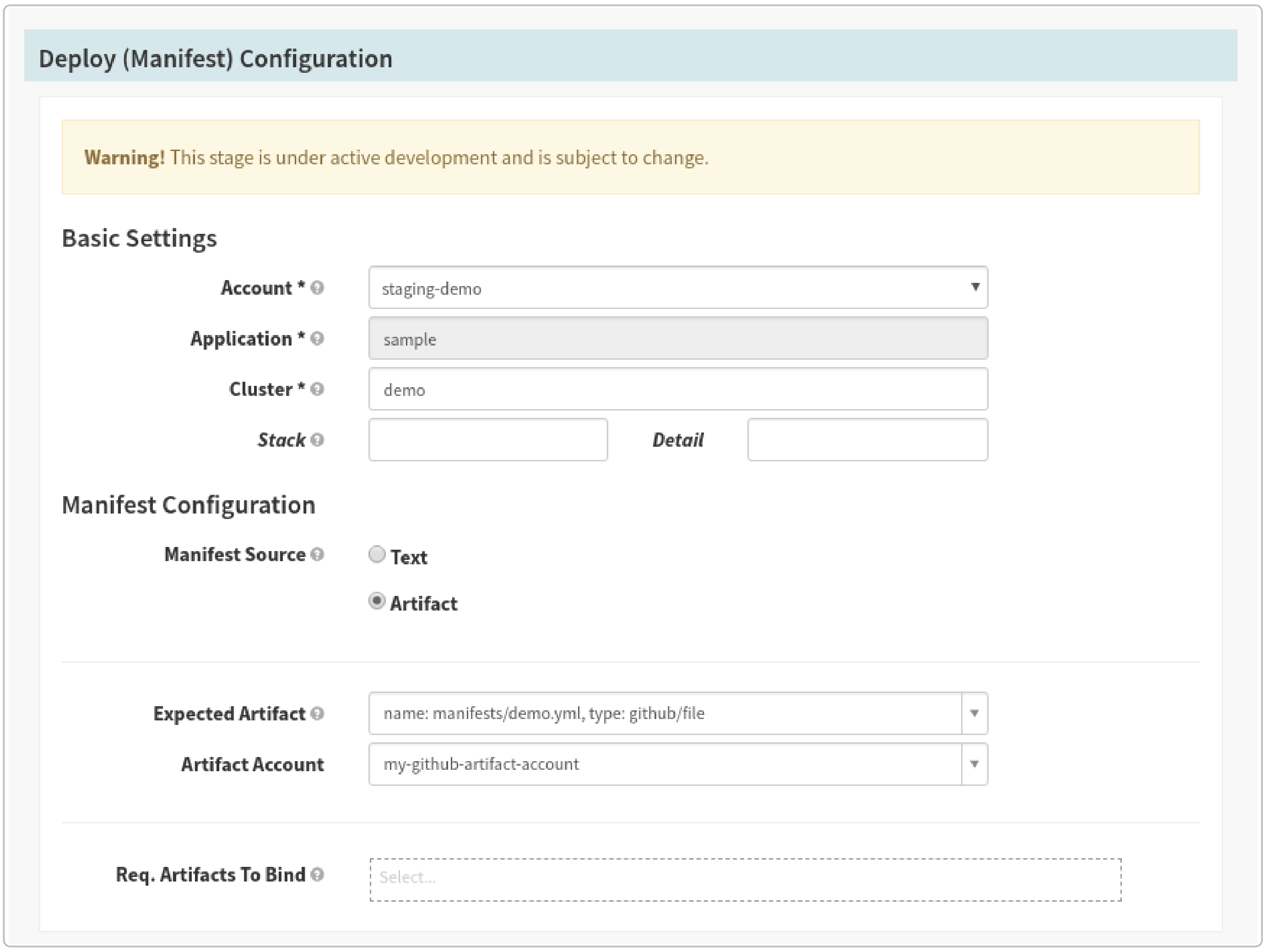
Deploy (Manifest)
Deploy a Kubernetes manifest yaml/json file.

Destroy Server Group
Destroys a server group

Disable Cluster
Disables a cluster

Finally, configure the stage with the following values:

Field	Value
Account	“staging-demo”
Cluster	“demo”
Manifest Source	“Artifact”
Expected Artifact	Must reference the `manifests/demo.yml` artifact.
Artifact Account	The GitHub artifact account configured above.



Deploy (Manifest) Configuration

Warning! This stage is under active development and is subject to change.

Basic Settings

Account *	staging-demo
Application *	sample
Cluster *	demo
Stack	
Detail	

Manifest Configuration

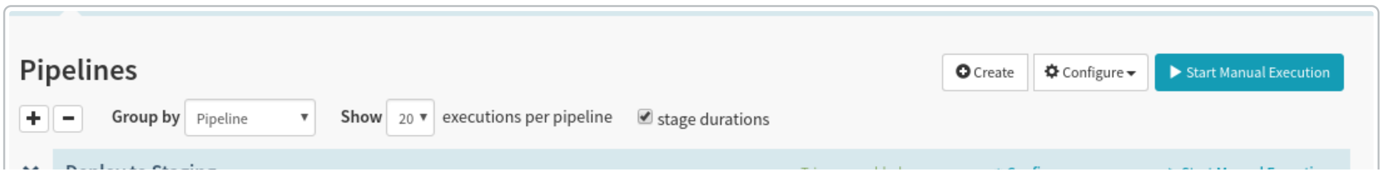
Manifest Source: ( ) Text [✓ Artifact]

Expected Artifact	name: manifests/demo.yml, type: github/file
Artifact Account	my-github-artifact-account
Req. Artifacts To Bind	Select...

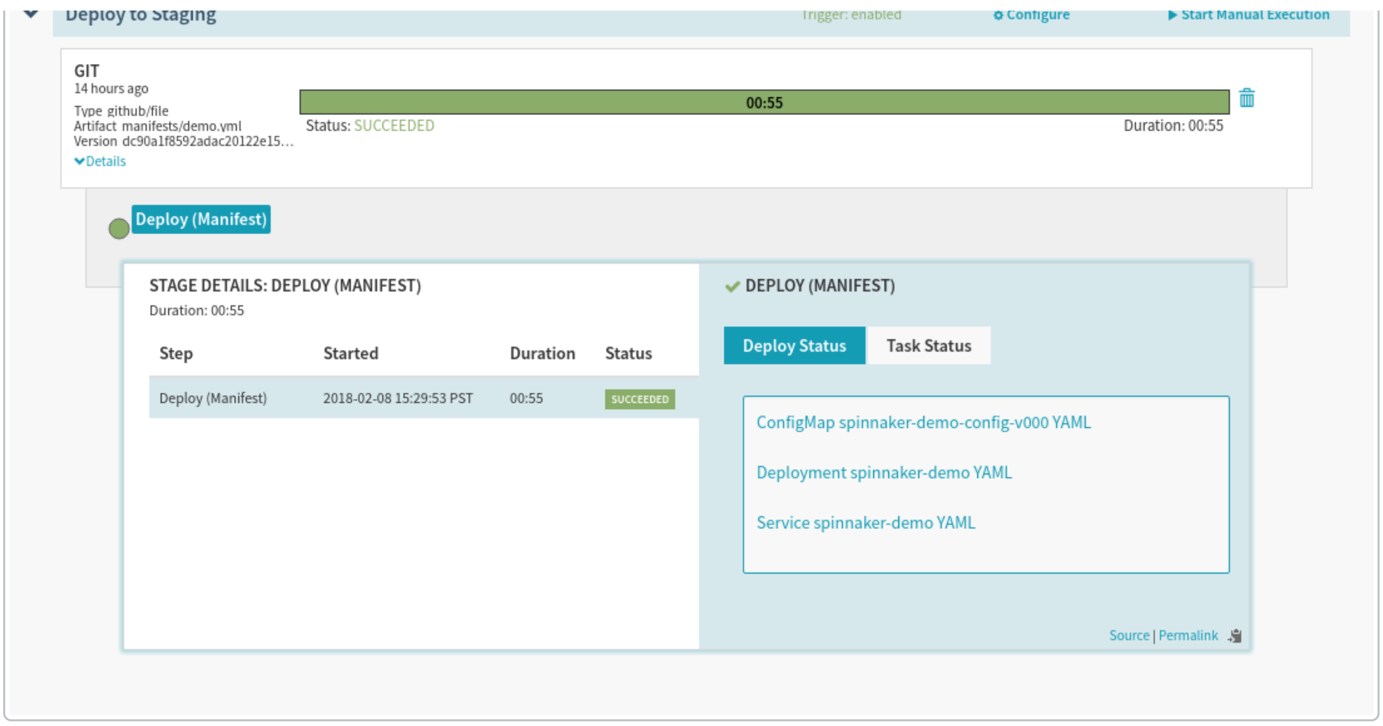
Save the pipeline.

3. Deploy manifests to staging

Trigger the pipeline by pushing a commit to the `manifests/demo.yml` file in your repository. The pipeline should start in a few seconds. When it completes, click **Details** to see information about the execution:



Pipelines



Deploy to Staging
Trigger: enabled ⚙ Configure ▶ Start Manual Execution

GIT
14 hours ago
Type github/file
Artifact manifests/demo.yml
Version dc90a1f8592adac20122e15…
Details

00:55
Status: SUCCEEDED
Duration: 00:55

Deploy (Manifest)

STAGE DETAILS: DEPLOY (MANIFEST)
Duration: 00:55

Step	Started	Duration	Status
Deploy (Manifest)	2018-02-08 15:29:53 PST	00:55	SUCCEEDED

DEPLOY (MANIFEST)
Deploy Status | Task Status

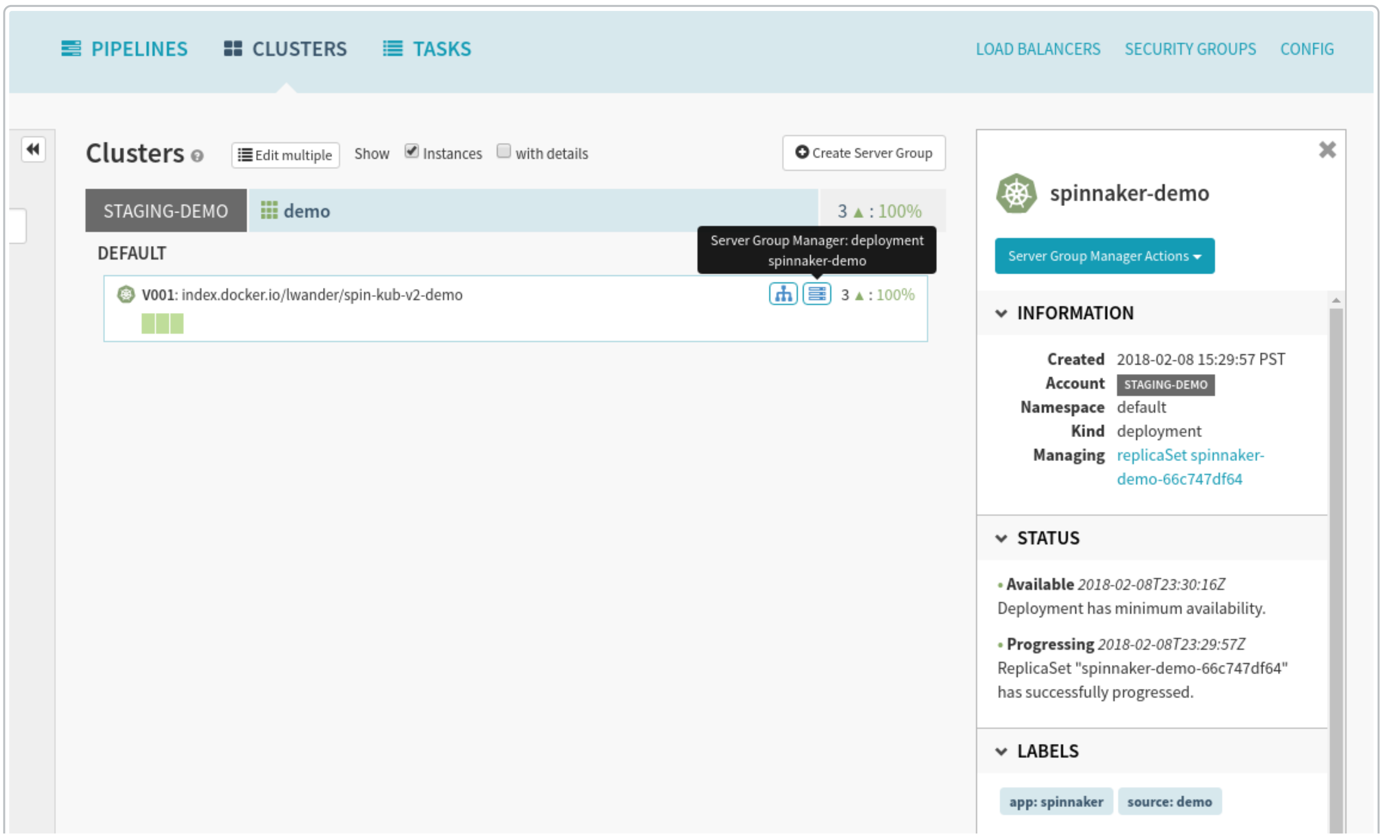
ConfigMap spinnaker-demo-config-v000 YAML
Deployment spinnaker-demo YAML
Service spinnaker-demo YAML

Source | Permalink

There are a couple of things to notice here:

- In the top left we get details about the commit that triggered this pipeline.
- In the **Deploy Status** we can see what the YAML was that Spinnaker deployed.
- We see that the ConfigMap that we deployed was assigned version `-v000`. This was done to ensure that you can statically reference this ConfigMap, insulating any Pod that references it from accidental changes.

Next, let's see what this infrastructure looks like in Spinnaker. Navigate to the **Clusters** tab, and select the blue Deployment object attached to the Replica Set shown below:



INFORMATION

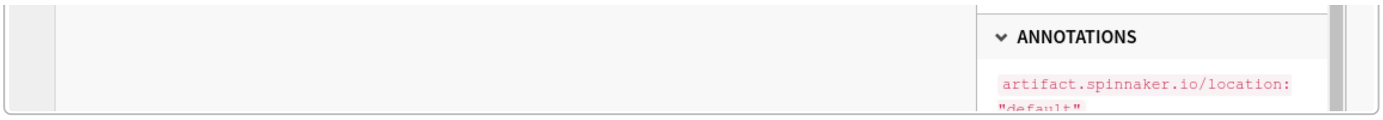
Created	2018-02-08 15:29:57 PST
Account	STAGING-DEMO
Namespace	default
Kind	deployment
Managing	replicaSet spinnaker-demo-66c747df64

STATUS

- **Available** *2018-02-08T23:30:16Z* Deployment has minimum availability.
- **Progressing** *2018-02-08T23:29:57Z* ReplicaSet "spinnaker-demo-66c747df64" has successfully progressed.

LABELS

app: spinnaker source: demo



We can see in the **Artifact** section on the right that we have bound our Docker image as well as our ConfigMap.

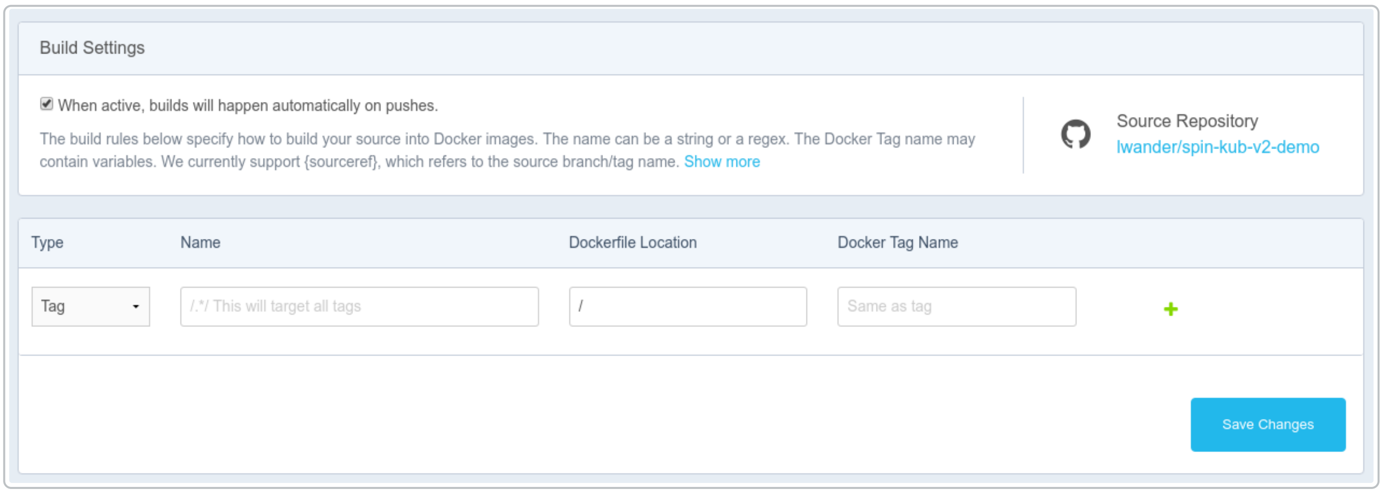
Let's see what our application is serving. Run:

```
kubectl proxy --context $STAGING_CONTEXT
```

And then visit the sample service in your browser. Let's make a change to this service, and configure Spinnaker to listen to Docker builds.

4. Configure Docker triggers

Important: We need to configure DockerHub to build on **Tag** events only, if we build on every commit, this particular setup will trigger both when manifests & code are changed at once. This can be configured under your Docker repository's **Build Settings** tab as shown here:



Build Settings

✓ When active, builds will happen automatically on pushes.

The build rules below specify how to build your source into Docker images. The name can be a string or a regex. The Docker Tag name may contain variables. We currently support {sourceref}, which refers to the source branch/tag name. Show more

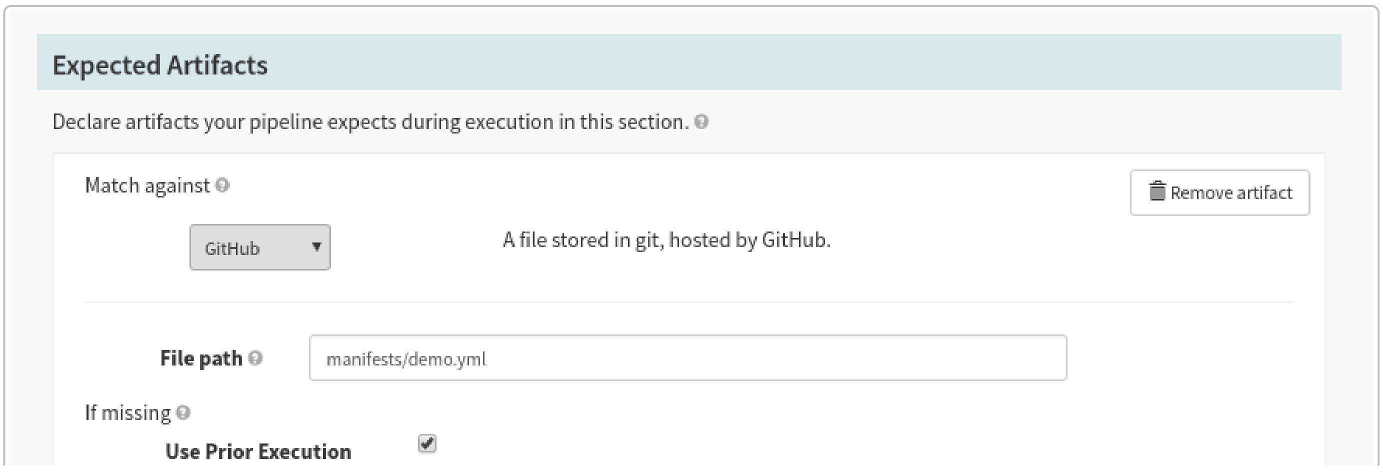
Source Repository
lwander/spin-kub-v2-demo

Type	Name	Dockerfile Location	Docker Tag Name
Tag	/.*/ This will target all tags	/	Same as tag

This build rule will create a matching image tag each time you push a git tag.

Next, in Spinnaker, let's edit our Pipeline to allow Docker images to trigger a deployment:

First, add a Docker expected artifact next to our Git expected artifact:



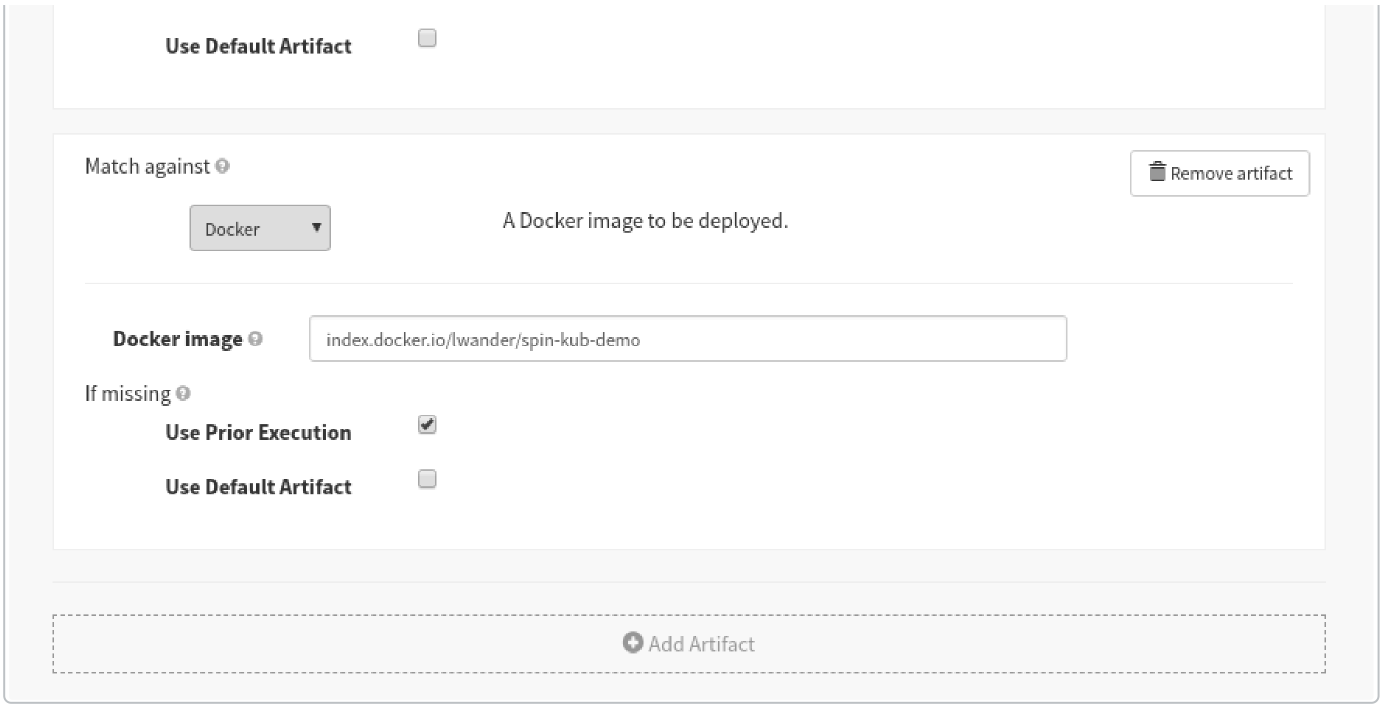
Expected Artifacts

Declare artifacts your pipeline expects during execution in this section.

Match against: GitHub — A file stored in git, hosted by GitHub.

File path: manifests/demo.yml

If missing
Use Prior Execution ✓


Use Default Artifact ☐

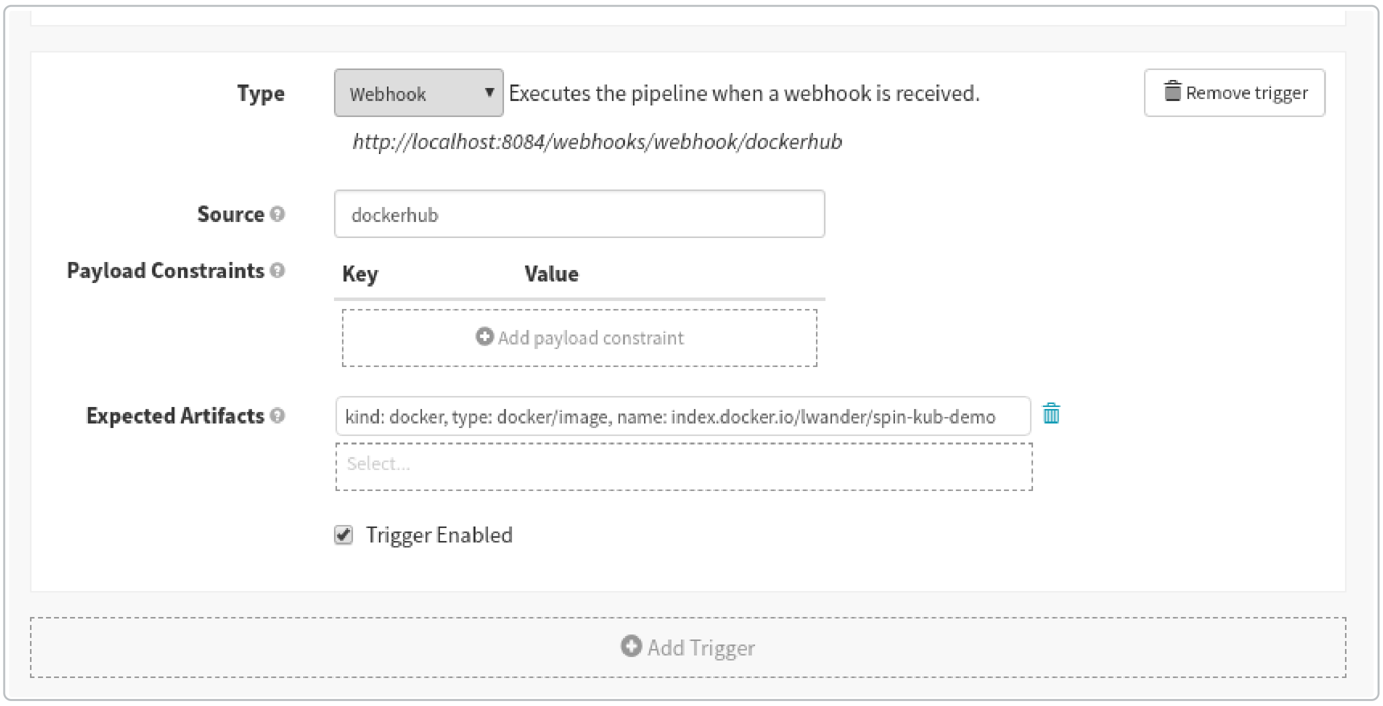
Match against
Docker
A Docker image to be deployed.
Remove artifact

Docker image: index.docker.io/lwander/spin-kub-demo

If missing
Use Prior Execution ✓
Use Default Artifact ☐

Add Artifact

Next, add a *Webhook* trigger to listen to build events from DockerHub. The *Docker* trigger alone won't provide us with provenance information.


Type: Webhook — Executes the pipeline when a webhook is received.
Remove trigger

http://localhost:8084/webhooks/webhook/dockerhub

Source: dockerhub

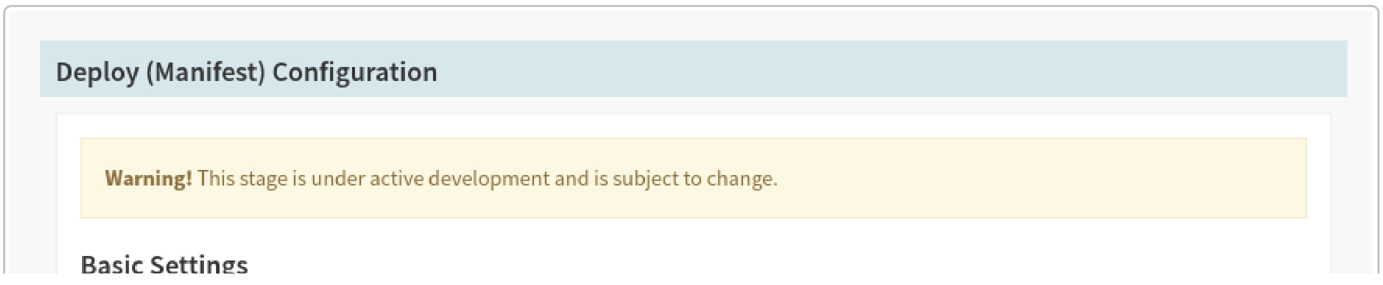
Payload Constraints
Key Value
Add payload constraint

Expected Artifacts: kind: docker, type: docker/image, name: index.docker.io/lwander/spin-kub-demo
Select...

✓ Trigger Enabled

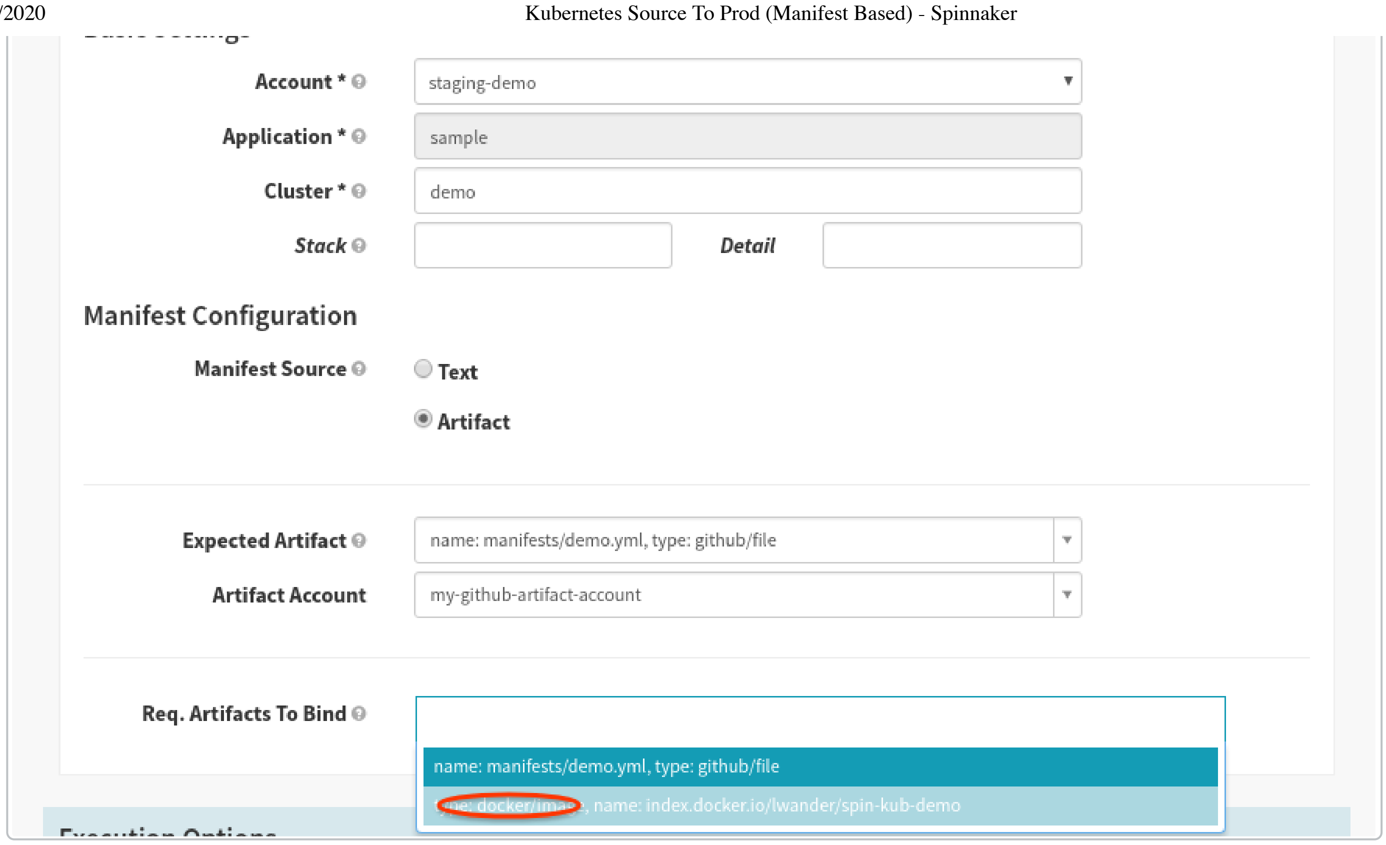
Add Trigger

Finally, back in the "Deploy (Manifest)" stage configuration, select the Docker artifact to bind in this deployment:


Deploy (Manifest) Configuration

Warning! This stage is under active development and is subject to change.

Basic Settings



Account *	staging-demo
Application *	sample
Cluster *	demo
Stack	
Detail	

Manifest Configuration

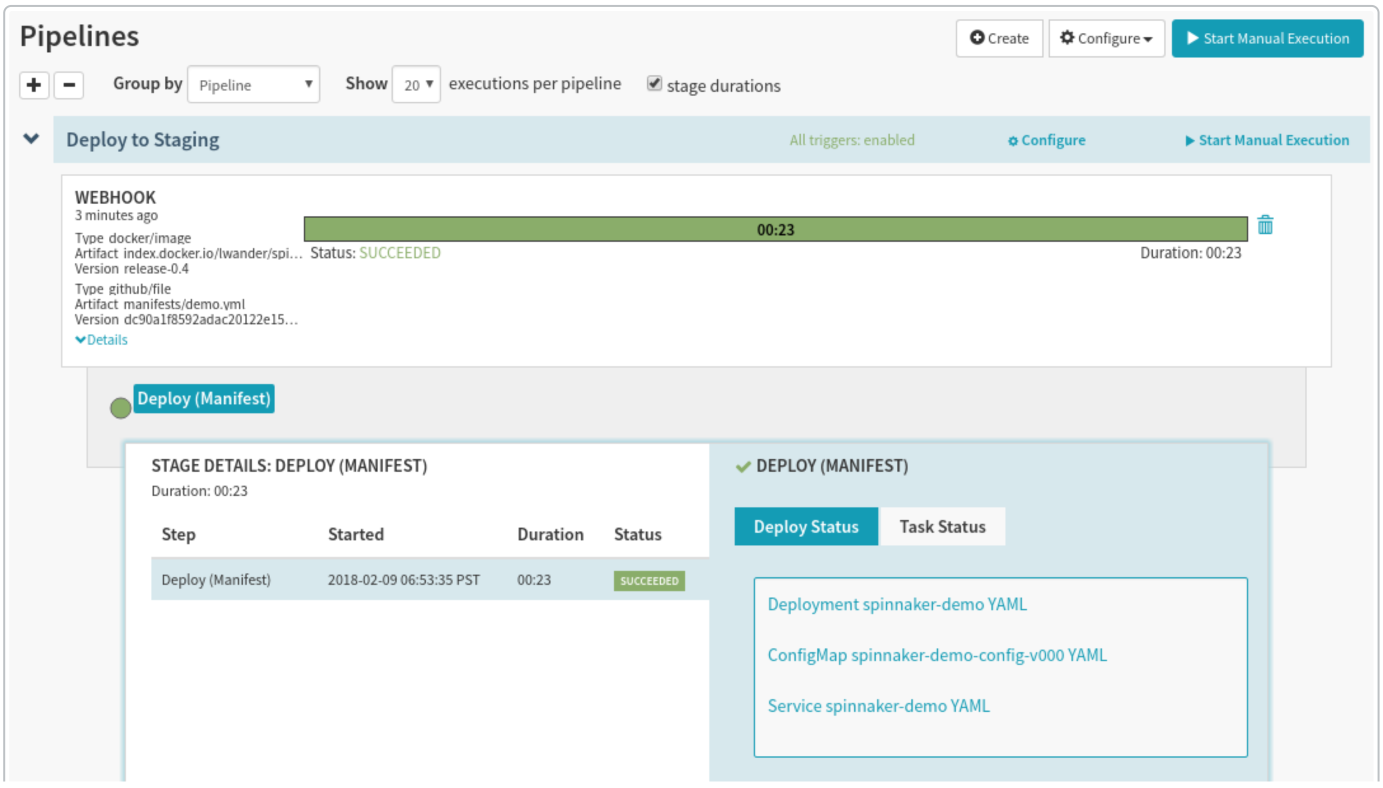
Manifest Source	Text [✓ Artifact]
Expected Artifact	name: manifests/demo.yml, type: github/file
Artifact Account	my-github-artifact-account
Req. Artifacts To Bind	name: manifests/demo.yml, type: github/file; type: docker/image, name: index.docker.io/lwander/spin-kub-demo

Save the pipeline.

5. Deploy Docker to staging

You can push a tag to your repository by running:

```
git tag release-1.0
git push origin release-1.0
```



Pipelines

Group by Pipeline Show 20 executions per pipeline ✓ stage durations

Deploy to Staging All triggers: enabled

WEBHOOK
3 minutes ago
Type docker/image
Artifact index.docker.io/lwander/spi…
Version release-0.4
Type github/file
Artifact manifests/demo.yml
Version dc90a1f8592adac20122e15…

00:23
Status: SUCCEEDED
Duration: 00:23

Deploy (Manifest)

STAGE DETAILS: DEPLOY (MANIFEST)
Duration: 00:23

Step	Started	Duration	Status
Deploy (Manifest)	2018-02-09 06:53:35 PST	00:23	SUCCEEDED

✓ DEPLOY (MANIFEST)

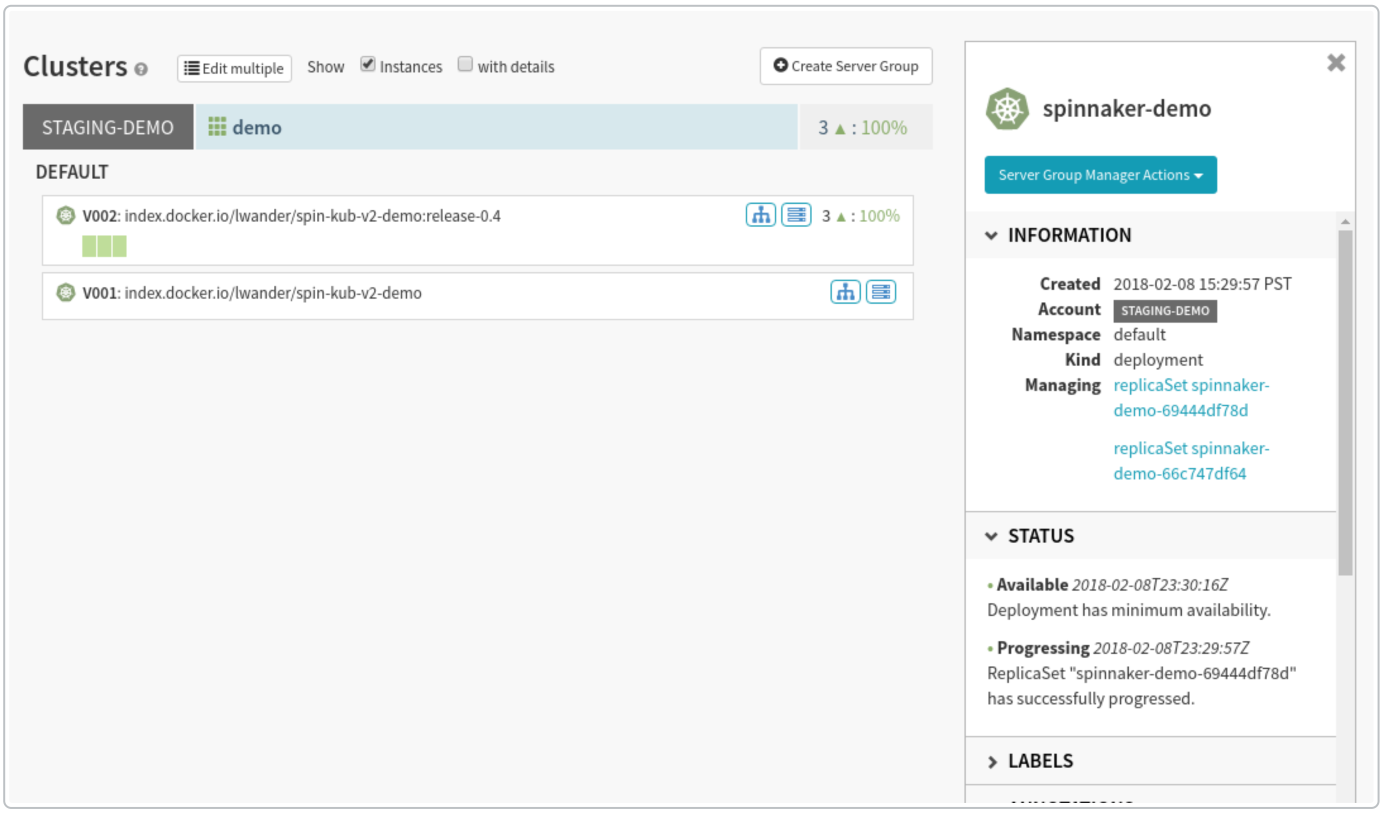
Deploy Status | Task Status

Deployment spinnaker-demo YAML
ConfigMap spinnaker-demo-config-v000 YAML
Service spinnaker-demo YAML

Notice that this time the trigger was a Webhook trigger, and we see details about both types of artifacts that we deployed. Since the GitHub file artifact was configured to **Use Prior Execution**, we redeployed the same manifests as last time, but with a new Docker image. Because of this, we did not deploy a new ConfigMap, and kept the version at `-v000`.

This deployment is a lot faster than the last one, since the docker image was already pulled into our cluster, meaning it took less time for the images to start running and appear as "Healthy".

Back on the **Clusters** tab we can see the deployment has rolled out our new image:



V002: index.docker.io/lwander/spin-kub-v2-demo:release-0.4

V001: index.docker.io/lwander/spin-kub-v2-demo

Created	2018-02-08 15:29:57 PST
Account	STAGING-DEMO
Namespace	default
Kind	deployment
Managing	replicaSet spinnaker-demo-69444df78d replicaSet spinnaker-demo-66c747df64

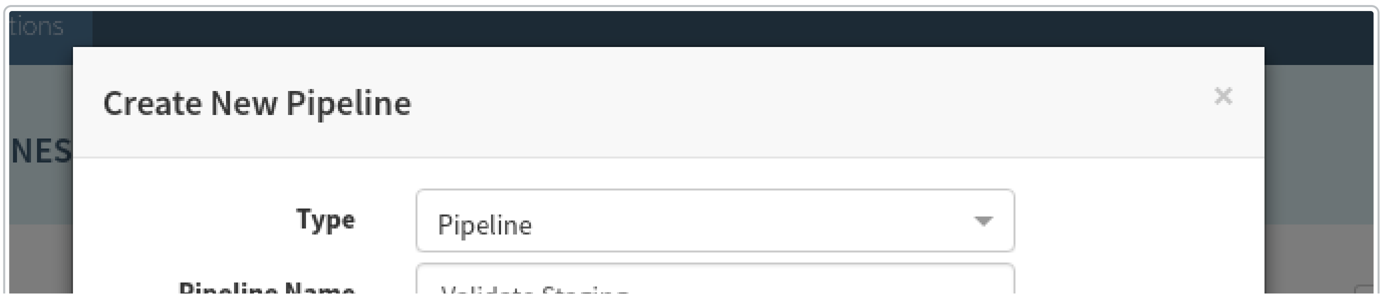
• **Available** *2018-02-08T23:30:16Z*
Deployment has minimum availability.

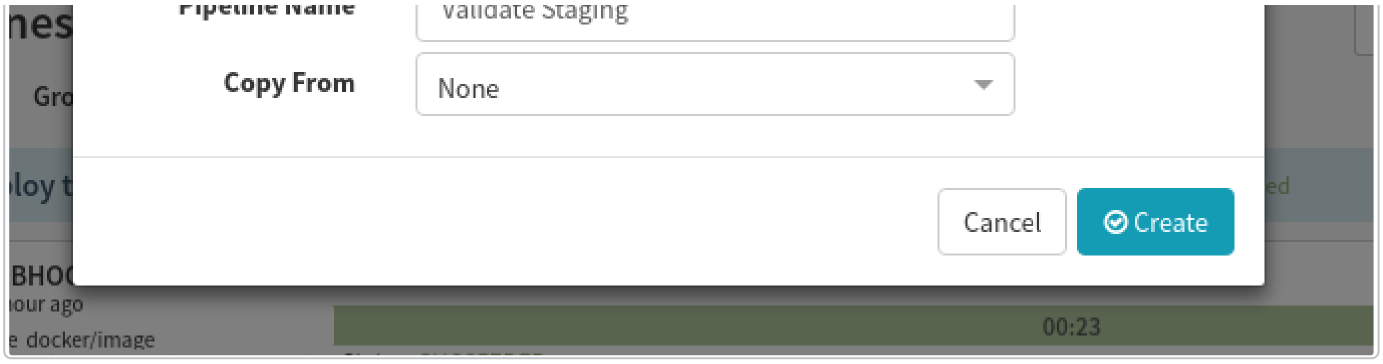
• **Progressing** *2018-02-08T23:29:57Z*
ReplicaSet "spinnaker-demo-69444df78d" has successfully progressed.

6. Configure a validation pipeline

For the sake of a simple codelab, we will control which deployments make it to production by adding a "Manual Judgement" pipeline. In practice, this can be replaced by a canary, integration test suite, or other mechanism for validating staging; however, keeping the manual judgement stage is fine too.

Start by creating a new pipeline, and call it "Validate Staging":

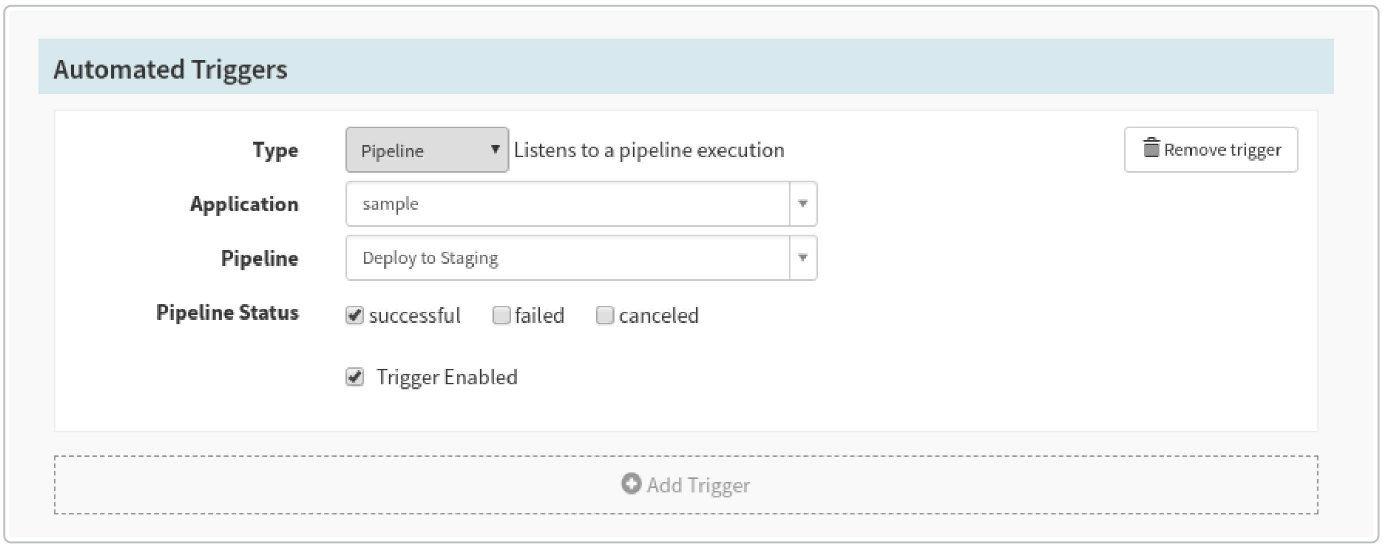




Pipeline Name: Validate Staging

Copy From: None

We only want this pipeline to run when we successfully deploy to our staging environment, so create a Pipeline trigger in this new pipeline like shown:



Automated Triggers

Type: Pipeline — Listens to a pipeline execution

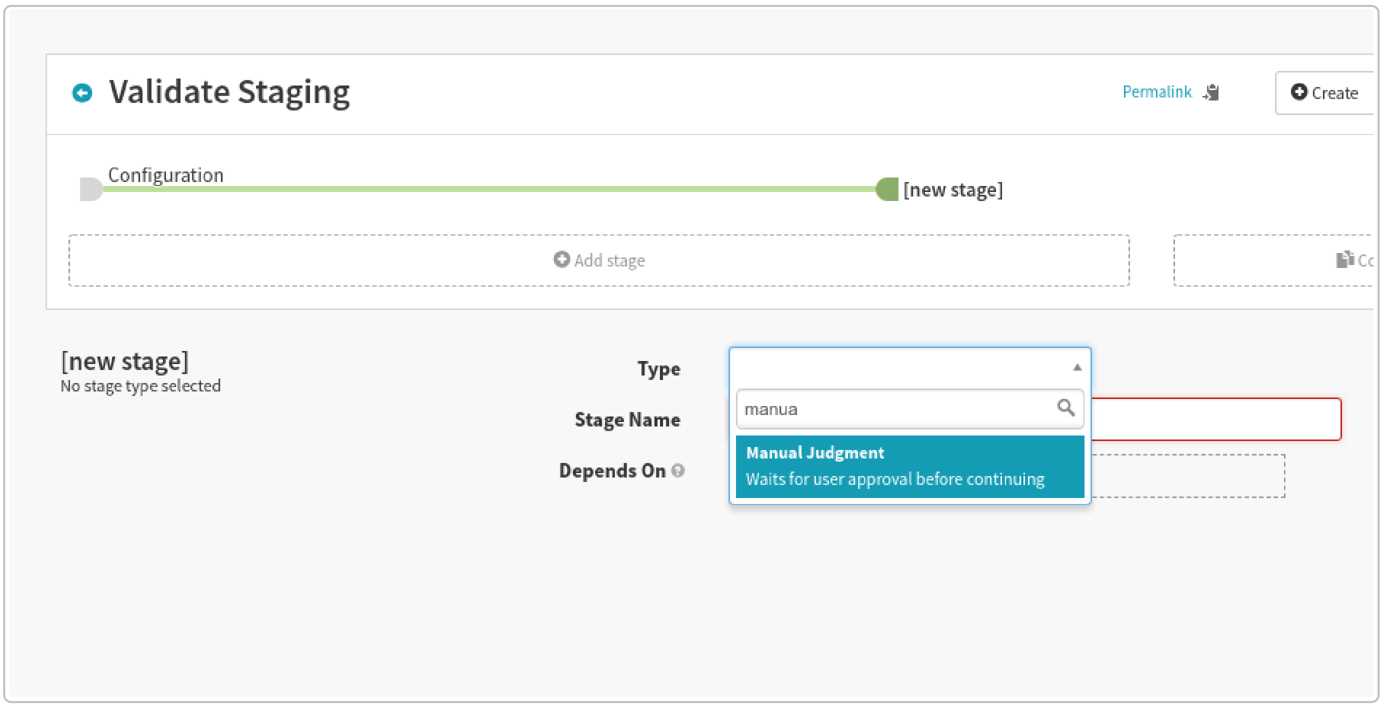
Application: sample

Pipeline: Deploy to Staging

Pipeline Status: [✓ successful] failed canceled

[✓ Trigger Enabled]

Add a single stage with type “Manual judgement”:



Validate Staging

[new stage]
No stage type selected

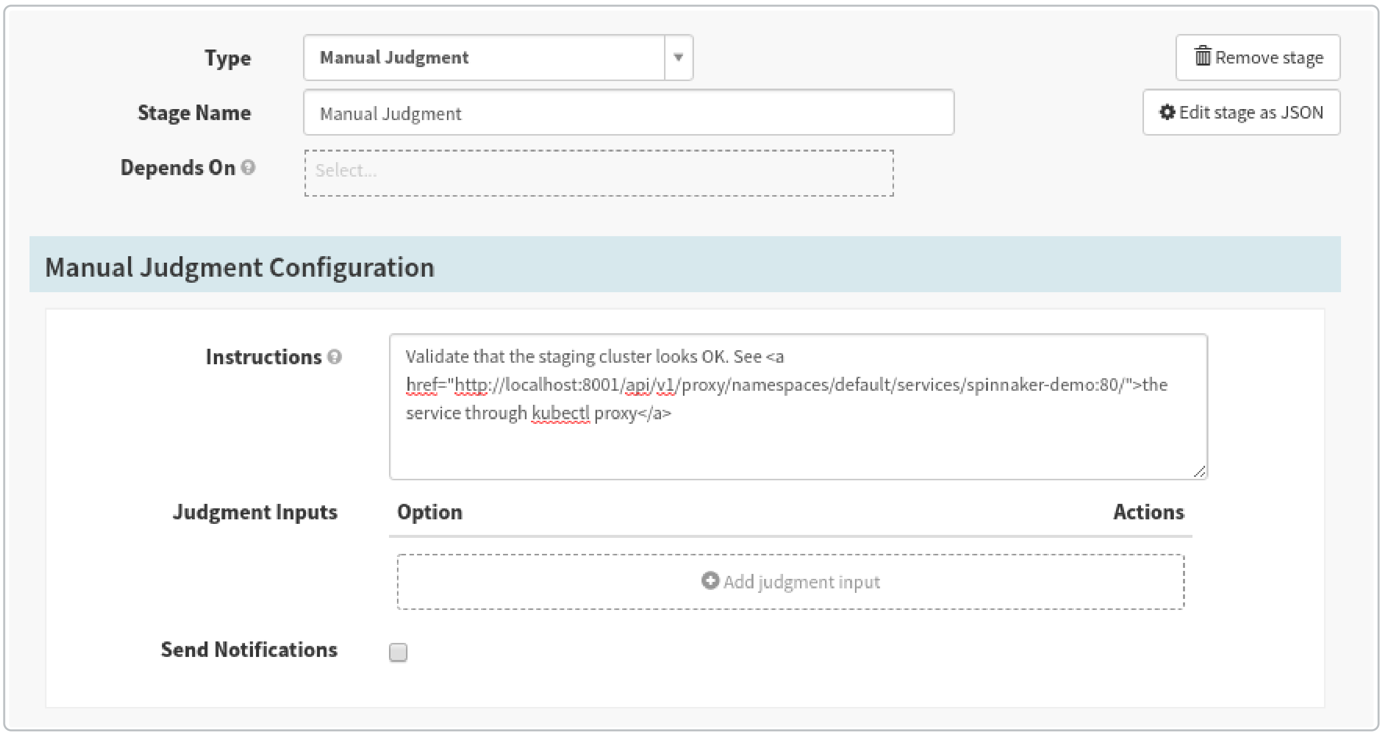
Type: manua

Manual Judgment
Waits for user approval before continuing

Stage Name

Depends On

If desired, you can add additional "Instructions" for how to validate the cluster:

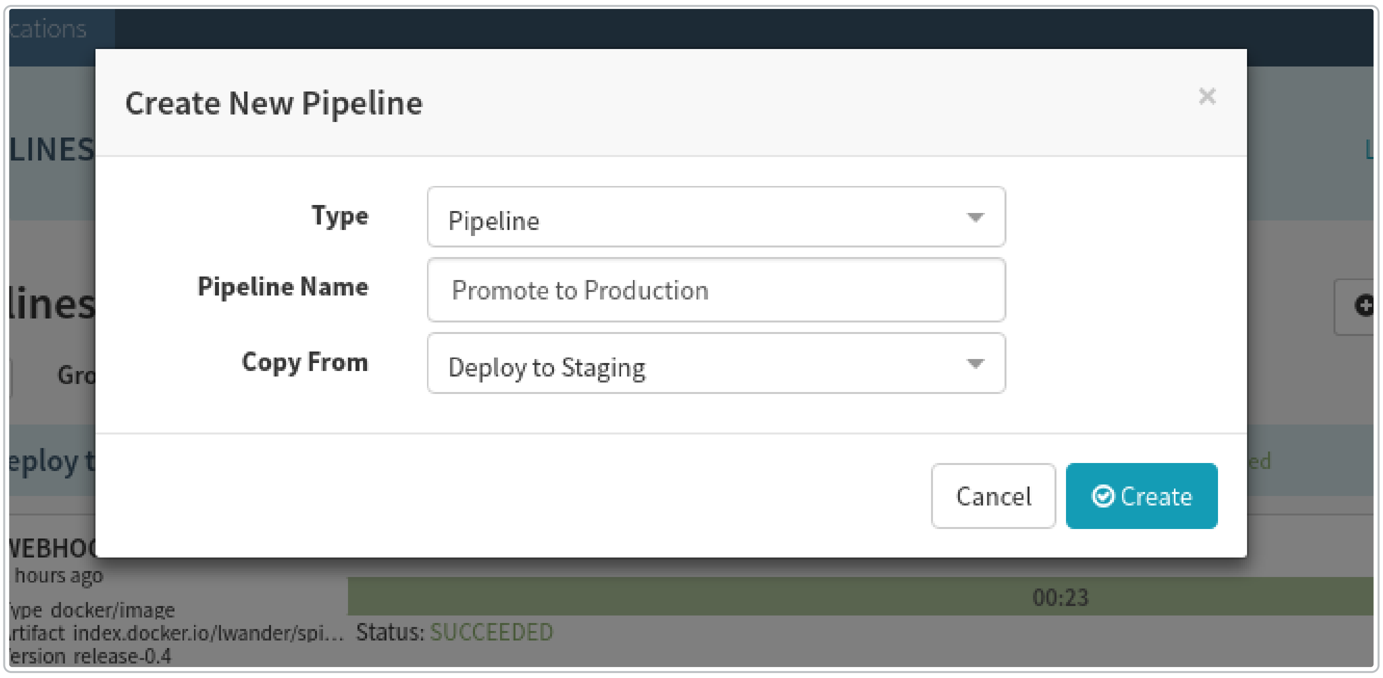


Type	Manual Judgment
Stage Name	Manual Judgment
Depends On	Select...
Instructions	Validate that the staging cluster looks OK. See <a href="http://localhost:8001/api/v1/proxy/namespaces/default/services/spinnaker-demo:80/">the service through kubectl proxy</a>
Judgment Inputs	Option / Actions
Send Notifications	☐

Save the Pipeline.

7. Promote to production

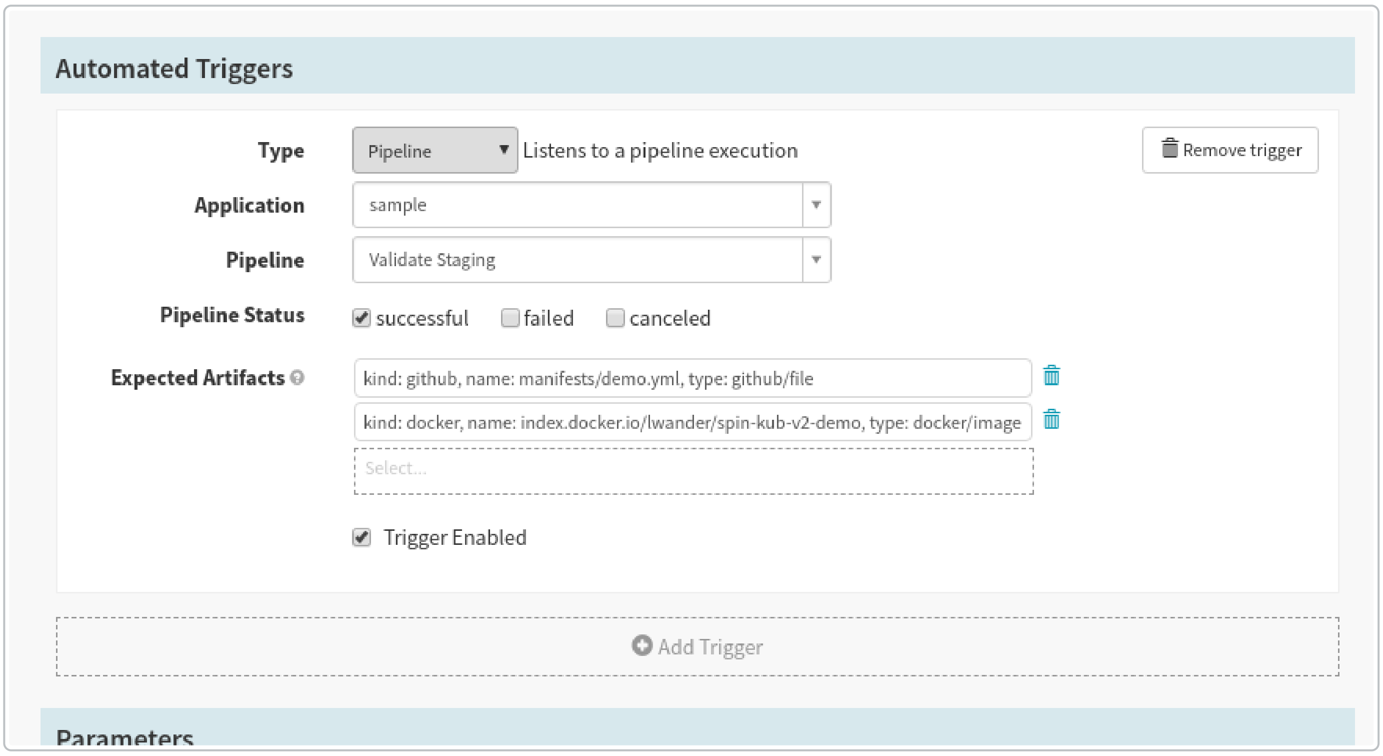
Let's promote these artifacts into our production cluster. Create a new pipeline, but instead of creating it from scratch, let's copy the "Deploy to Staging" pipeline like this:



Type	Pipeline
Pipeline Name	Promote to Production
Copy From	Deploy to Staging

We need to make two changes to this pipeline:

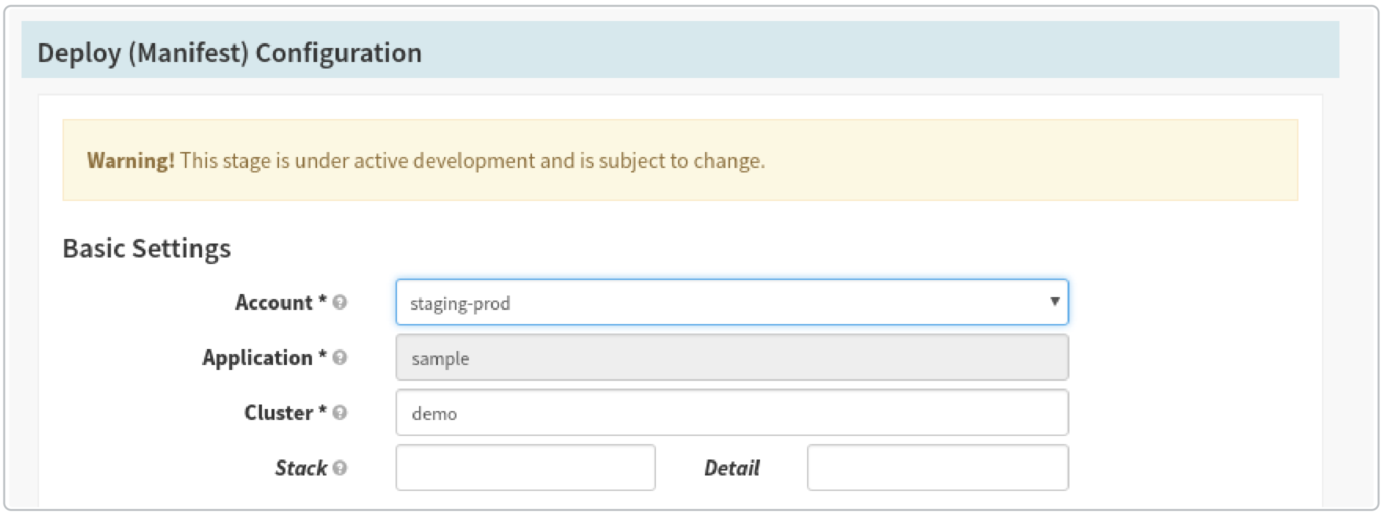
First, delete the webhook and Git triggers, and replace it with a pipeline trigger that depends on the "Validate Staging" pipeline:



Automated Triggers

Type	Pipeline ▾ Listens to a pipeline execution
Application	sample
Pipeline	Validate Staging
Pipeline Status	☑ successful ☐ failed ☐ canceled
Expected Artifacts	kind: github, name: manifests/demo.yml, type: github/file kind: docker, name: index.docker.io/lwander/spin-kub-v2-demo, type: docker/image Select...
	☑ Trigger Enabled

Next, change the **Account** the "Deploy (Manifest)" stage deploys to point at **prod-demo**:



Deploy (Manifest) Configuration

Warning! This stage is under active development and is subject to change.

Basic Settings

Account *	staging-prod
Application *	sample
Cluster *	demo
Stack	
Detail	

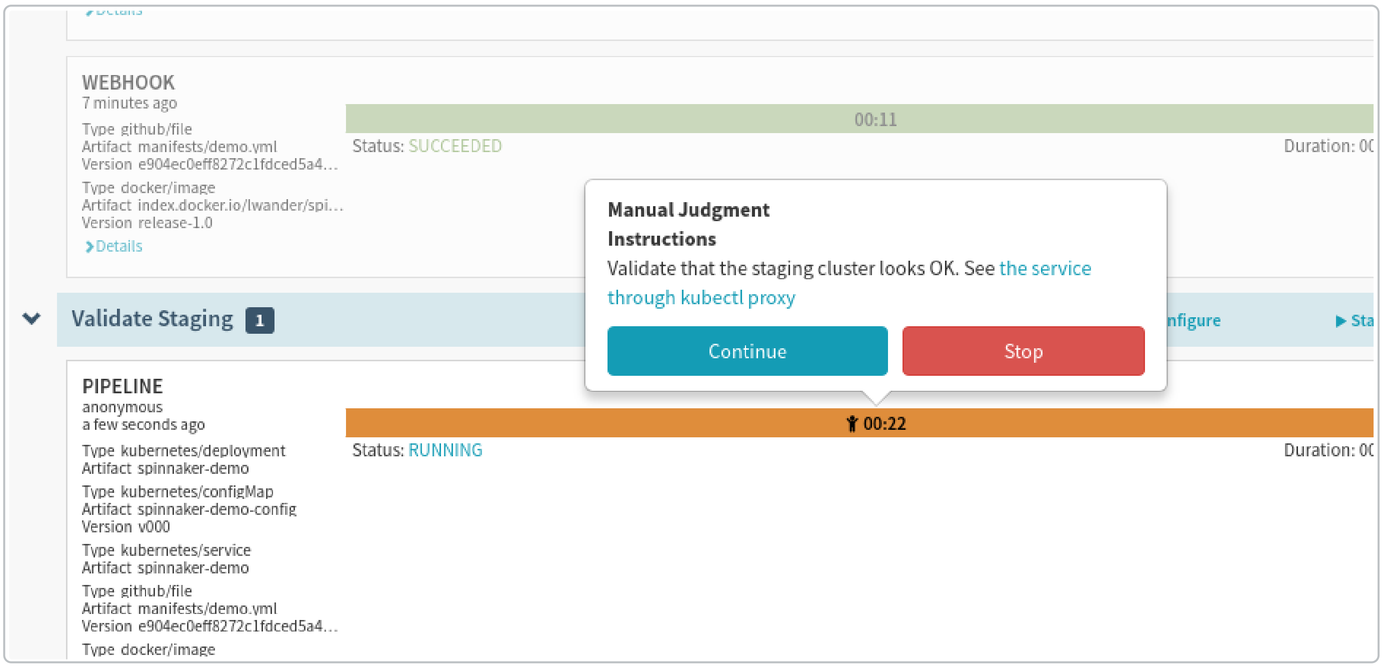
8. Run the full flow

Now our full flow is ready to go - let's kick it off by changing the background color of our application.

Open `content/index.html` in your text editor, and change the background color attribute, and generate a new commit. We can safely push this commit to GitHub without running our pipeline because we are only listening to change to the `manifests/demo.yml` file in our "Deploy to Staging" trigger. Tag and push this commit to generate a new docker build:

```
git tag release-1.1
git push origin release-1.1
```

When Spinnaker prompts you, accept (or reject) the manual judgement:

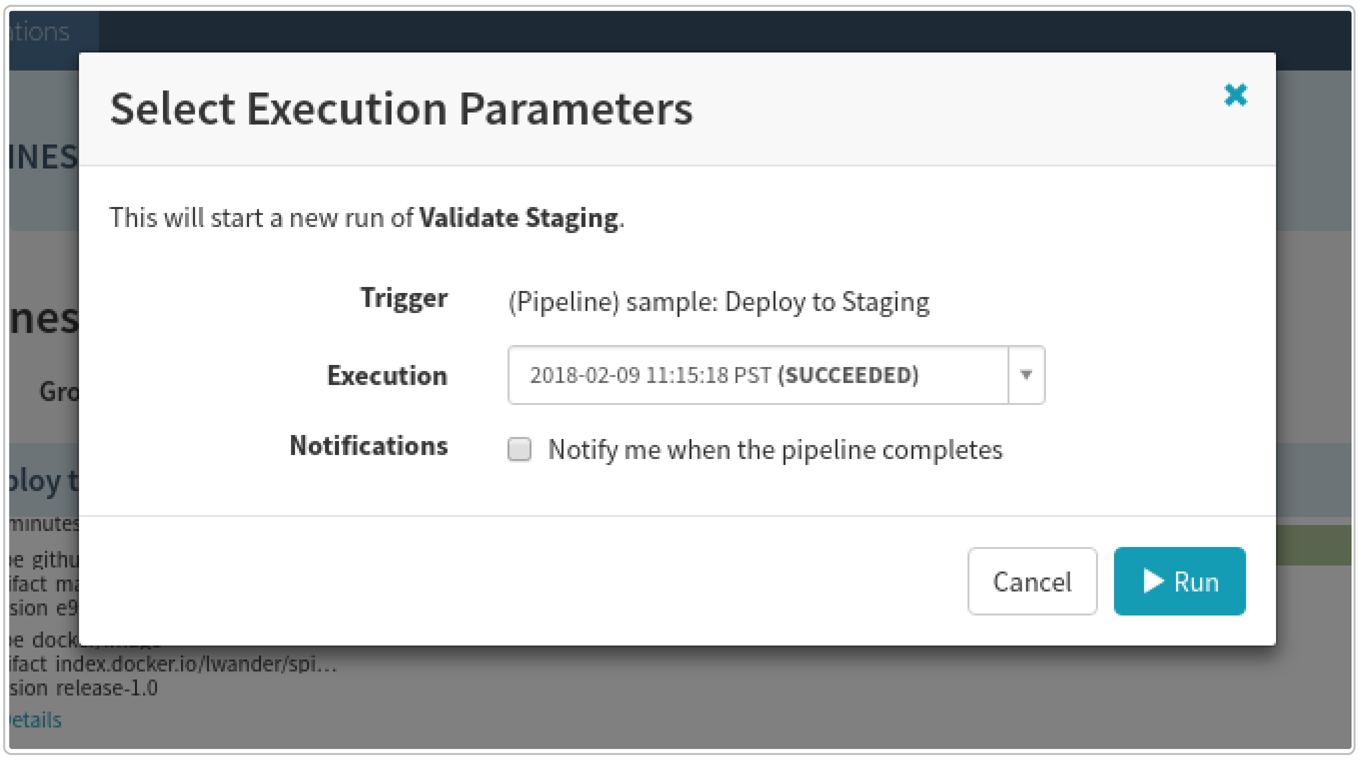

WEBHOOK
7 minutes ago
Type github/file
Artifact manifests/demo.yml
Version e904ec0eff8272c1fdced5a4…
Type docker/image
Artifact index.docker.io/lwander/spi…
Version release-1.0
Status: SUCCEEDED
Duration: 00

Manual Judgment
Instructions
Validate that the staging cluster looks OK. See the service through kubectl proxy

Validate Staging 1

PIPELINE
anonymous
a few seconds ago
Type kubernetes/deployment
Artifact spinnaker-demo
Type kubernetes/configMap
Artifact spinnaker-demo-config
Version v000
Type kubernetes/service
Artifact spinnaker-demo
Type github/file
Artifact manifests/demo.yml
Version e904ec0eff8272c1fdced5a4…
Type docker/image
Status: RUNNING
Duration: 00

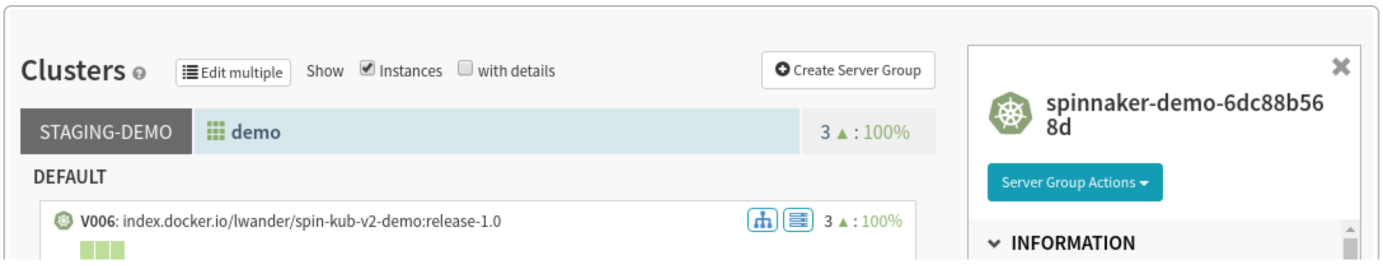
Keep in mind, if you reject the manual judgement, but later change your mind, you can always trigger this pipeline again using the same context by selecting **Start manual execution**, and picking the latest parent execution:

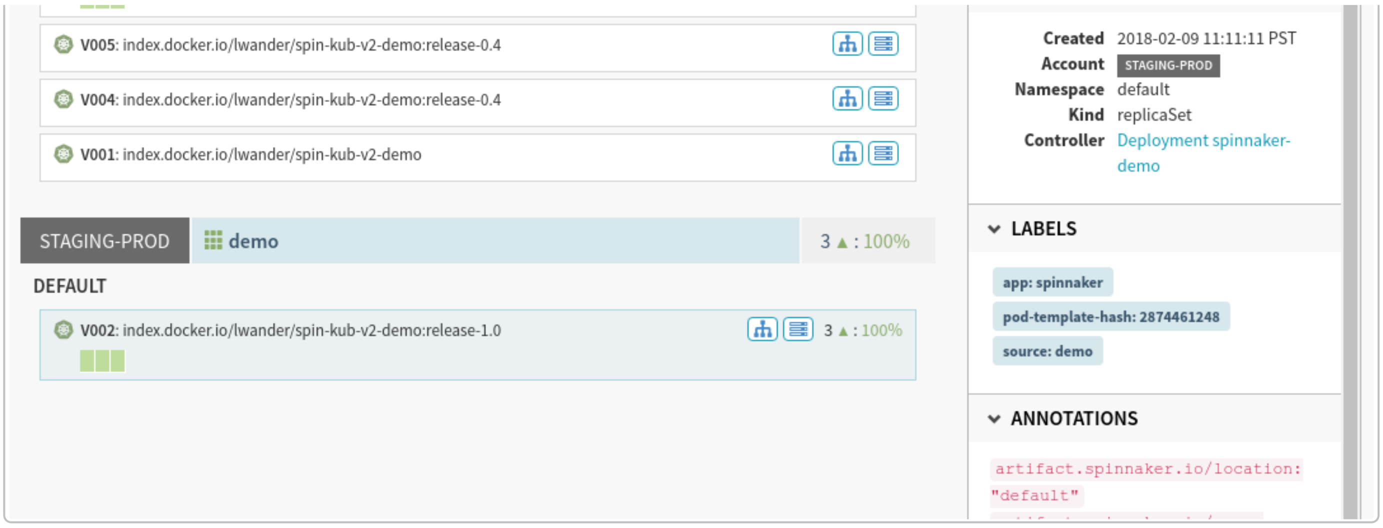

Select Execution Parameters

This will start a new run of **Validate Staging**.

Trigger (Pipeline) sample: Deploy to Staging
Execution 2018-02-09 11:15:18 PST **(SUCCEEDED)**
Notifications ☐ Notify me when the pipeline completes

Once all three pipelines complete, you should have your docker image running in both environments:


Clusters
STAGING-DEMO demo 3 ▲ : 100%
DEFAULT
V006: index.docker.io/lwander/spin-kub-v2-demo:release-1.0 3 ▲ : 100%
spinnaker-demo-6dc88b56 8d


V005: index.docker.io/lwander/spin-kub-v2-demo:release-0.4

V004: index.docker.io/lwander/spin-kub-v2-demo:release-0.4

V001: index.docker.io/lwander/spin-kub-v2-demo

STAGING-PROD demo 3 ▲ : 100%

DEFAULT

V002: index.docker.io/lwander/spin-kub-v2-demo:release-1.0 3 ▲ : 100%

Created	2018-02-09 11:11:11 PST
Account	STAGING-PROD
Namespace	default
Kind	replicaSet
Controller	Deployment spinnaker-demo

LABELS

app: spinnaker

pod-template-hash: 2874461248

source: demo

ANNOTATIONS

```
artifact.spinnaker.io/location:
"default"
```

9. Extra credit

At this point there are few things you can play with:

- Change the ConfigMap definition in the manifest file. In particular, flip the single flag from “false” to “true”. See what happens to the ConfigMap version.
- Roll back a “broken” change either in prod or staging using the “Undo Rollout” stage.
- Insert pipeline expressions into your manifest files.

10. Teardown

As referenced above, please teardown Spinnaker once you are done:

```
hal deploy clean
```

File a bug

Comment on this document