Spinnaker

Daniel Hinojosa

Table of Contents

Before we begin

Please back up important files, you will be using your personal machine and some of the commands include `rm`. Even the best of us a susceptible to catastrophe.

Continuous Integration and Delivery

Continuous Integration

- Taking code and integrating and testing with every check in!
- Employs Continuous Integration (CI) Server
  - Jenkins
  - Bamboo
  - Cruise Control
  - Travis

Extending CI with Spinnaker

- Spinnaker takes content created in CI and deploys to staging and production
- Jenkins could do that to a point:
  - It does not manage multiple servers too well
  - Jenkins (not JenkinsX) doesn’t work that well with Kubernetes

Continuous Integration

- Tool that monitors version control of changes
- When a change is detected, the tool will automatically
  - compile
  - test
  - report
- Overtime reports on the overall code quality of your project

Why Continuous Integration?

- Confidence that a release will perform
- Fully automated testing
- Ability to have a "one click" deploy
- Repeated and reliable

- Enhances Collaboration Among Developers
  - Builds are no longer secret
  - No longer require the use of deployment expert
- Manual building is fairly stupid
- Process seen in action! You know if it works or doesn't

Purpose of a CI Server

- Configured to watch your version control system
- Check out or update your source code every time a change is made
- Run the automated build process
- Store the binaries where they are accessible to the whole team
- Reducing risk by providing faster feedback
- Identify and fix integration and regression issues faster

How does CI Improve Software Quality

- Everyone has information on the build at every single moment
- Developers are aware of the constant change in their project
- Notifications to all Developers when the build fails

Provide Value to the End User Faster

- Ensures that software is build, tested, and maintained regularly
- There is always a deployment of some kind
- Deployments are no longer magical mystery events that happen every month

Empowering Teams

- Continuous Integration can deploy onto the servers themselves.
- Power to real time view results
- There will be no longer the uber-operations-specialist
- Expand the role of the developer include operations.

Reducing Errors

- Ensures the correct version, configuration, database schema, etc. are applied the same way every time through automation
- Staging areas are equivalent to production areas

- Less surprises.
- No more hidden configuration that we don’t know about at production time

Lowers Stress

- A Release becomes commonplace without the typical stress
- No weekend battles
- Less embarrassing releases
- Rollback versions easily to a well known build

Deployment Flexibility

- Instantiate a new environment or configuration by making a few changes to the automated delivery system.
- Create multiple version of environments

Practice Makes Perfect

- Final deployment into production is being rehearsed every single time the software is deployed to any target environments.
- Overtime deployment becomes standard practice
- Check in frequently
- Don’t check in broken code
- Don’t check in untested code
- Don’t check in when the build is broken
- Don’t go home after checking in until the system builds
- Every Check in is a potential release!

Will it happen overnight?

- It will take time for all those invested
- Testing will have to be commonplace
- Commitments in code will have to be small
- Behaviors and rituals will need to change

How does CI help release on time?

- Automation, Automation, Automation, Automation
- Humans are slow, why not let computers do the work?
- Problems no longer are debt: If anything fails it must be fixed immediately

- Bugs, along with testing, are squashed oftentimes permanently!
- Lessens the fear of "integration hell"
- Lessens the fear of production deployment!

Future is Here

- Spinnaker, Jenkins, and Kubernetes allows for IaC (Infrastructure as Code)
- Continuous Integration is committed as code, using `Jenkinsfile`
- Kuberenetes `replicaSet`, `deployment`, `ingress`, etc are all `yaml` files
- Spinnaker can use these files to deploy fleet of systems

Overview

What is Spinnaker?

- Open Source, Multicloud Platform
- Release Software at high velocity and confidence
- Extends where Jenkins CI leaves off
- Manages Staging and Deployment
- Makes use of containers
- Provides Baking Ability and Canary Ability

Definitions

Applications

- Represents the service which you are going to deploy using Spinnaker
- It is the center of your universe in regards to your application
- Contains all configuration for that service, and all the infrastructure on which it will run
- You will typically create a different application for each service

Includes:

- Firewalls
- Load Balancers
- Clusters of Services or Deployments
- Server Groups
- Canary Configurations

Clusters

- Logical Groups of Server Groups
- Does not map to a Kubernetes Cluster
- It *does* map to a Kubernetes *Deployment*

Source: https://www.spinnaker.io/reference/providers/kubernetes/#cluster

Server Group

- A base resource
  - Identifies the *Deployable Artifact*:
    - VM Image
    - Docker Image
    - Source Location
  - Basic Configuration Settings
    - Number of Instances
    - Autoscaling Policies
    - Metadata
  - Associated with a Kubernetes `Replica Set`
  - Server Group is a collection of instances of the running software
    - VM instances
    - Kubernetes pods

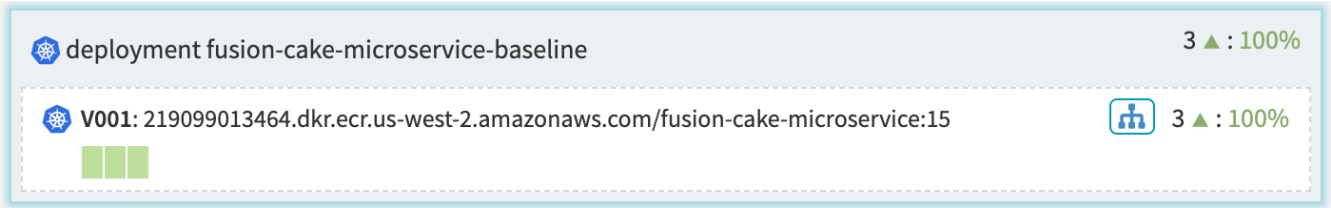


Figure 1. The Deployment is highlighted

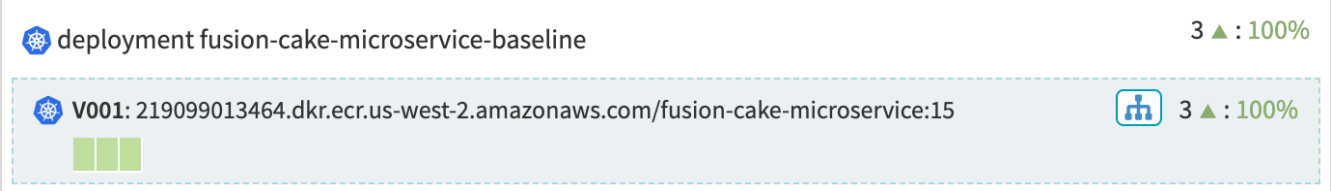
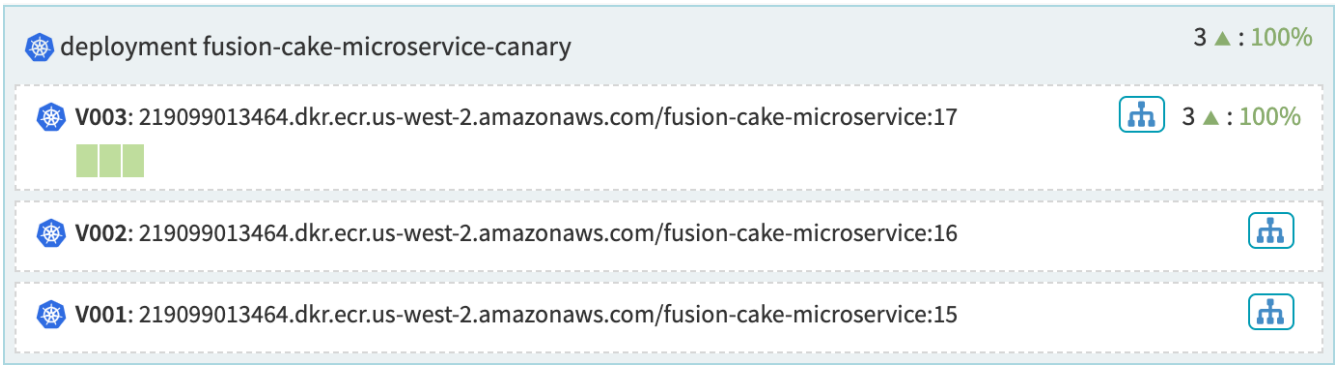


Figure 2. The Replica Set is highlighted

Replica Set History

- With deployments history of the deployments are maintained
- The following shows a history
- Deployments can be:

- *rolled back*
- *deleted*
- *restarted*
- *scaled*



Load Balancers

- Load Balancer is associated with:
  - An ingress protocol
  - Port range
- It corresponds to a Kubernetes *Service*
- It balances traffic among instances in its Server Groups.
- You can enable health checks for a load balancer
- Flexibility to define health criteria and specify the health check endpoint.

Firewalls

- Firewall defines network traffic access
- Firewall rules defined by an IP range (CIDR) along with a communication protocol (e.g., TCP) and port range
- It is associated with Kubernetes Network Policies

Source: https://www.spinnaker.io/reference/providers/kubernetes-v2/#networkpolicies

Spinnaker Process

1. Create an Application
   - One Application per Microservice
2. Define your Infrastructure
   - Define infrastructure for each application
   - Pipelines deploy services to the server groups you define
   - NOTE: Not necessary since you can define infrastructure from pipeline

3. Create a Pipeline
    - Create all the pipelines needed to deploy the service or services covered by the application
4. Run your Pipeline
    - Automatic
    - Triggered

Source: https://www.spinnaker.io/guides/user/get-started/

Pipeline

- Key deployment management construct in Spinnaker
- Consists of a sequence of actions, known as *stages*
- Possibility to pass parameters from stage to stage along the pipeline
- Started manually or triggered by an event:
    - `git push`
    - Docker image upload
    - CRON Schedule
- You can emit notifications

Stage

- Atomic building block for a pipeline
- Sequenced to form a pipeline
- Spinnaker provides multiple stages:
    - Deploy
    - Resize
    - Disable
    - Manual Judgment

Deployment Strategies

Image of the various Deployment Strategies

[deployment strategies] | *https://www.spinnaker.io/concepts/deployment-strategies.png*

- Spinnaker treats cloud-native deployment strategies as first class constructs
- Handling the underlying orchestration such as
    - Verifying health checks
    - Disabling old server groups
    - Enabling new server groups.

Cluster Definition

Source: https://www.spinnaker.io/concepts/clusters/

[clusters] | *https://www.spinnaker.io/concepts/clusters/clusters.png*

- Spinnaker acts as a single pane of glass from which to manage your global deployments across multiple clouds!
- Contains information related to:
  - Health and status of running environments
  - Metadata around deployments and individual instances.
- Also ability to perform ad-hoc actions you can perform on the resources you see such:
  - resize
  - clone
  - disable
  - roll back.

Command and control base where we can increasingly layer on information relevant to deploying your applications.

Cluster Anatomy

[clusters] | *https://www.spinnaker.io/concepts/clusters/clusters.png*

- Left Pane
  - Filters for viewing your deployed services.
  - Search by string match, or narrow by specific attributes of your services,such as:
    - Cloud platform
    - Region
    - Designated environments and stacks.
- Main Paine
  - Lists the deployed services.
    - Here, the green chiclets are individual instances (e.g. an EC2 VM instance, a Kubernetes Pod), grouped into Server Groups (a particular deployment), which are further grouped into Clusters.
- Right Pane
  - Provides details for the item currently selected in the center section
  - For each item, Spinnaker provides details related to both application-level concerns (Jenkins job details) as well as infrastructure-level concerns (machine type, auto-scaling policies).

Pipelines

Pipeline Definition

- Manage deployments in a consistent, repeatable and safe way.
- Sequence of stages ranging from functions that manipulate infrastructure (deploy, resize, disable)
    - Includes *scaffolding functions* (manual judgment, wait, run Jenkins job) that together precisely defines your runbook for managing your deployments

Pipeline Anatomy

[edit pipeline] | *https://www.spinnaker.io/concepts/pipelines/edit-pipeline.png*

- Define your sequence of stages at the top.
- Spinnaker supports parallel paths of stages, as well as the ability to specify whether multiple instances of a pipeline can be run at once.

Specify details for a given stage in the sections below.

Viewing Pipeline Execution History

- Serves as a means to introspect details of each deployment operation
- An effective audit log of enforced processes/policies on how you make changes to your deployed applications landscape.

[pipelines] | *https://www.spinnaker.io/concepts/pipelines/pipelines.png*

One Stage, Multiple Steps; One Step, Multiple Actions

[pipeline tasks] | *https://www.spinnaker.io/concepts/pipelines/pipeline-tasks.png*

- Automation does not end with orchestrating only the high-level steps of your release process
- Each of these operational steps often corresponds to a sequence of calls to the cloud platform
- Each of which needs to be remediated in failure scenarios
- The red/black Deploy stage is an example of how Spinnaker fully supports this notion

In the above image:

- The Red/Black Deploy stage in Spinnaker actually entails a sequence of steps
- Each given step is actually a set of tasks that need polling, remediation to ensure requisite state is reached prior to proceeding
- A given task often entails multiple API calls to the specific cloud platform, cognizant of expected response codes and remediating actions in failure

Pipeline Templates

Managed pipeline templates are a way to scalably manage pipelines across different teams.

Pipeline templates have two main components:

- Template
  - A pipeline template defines a parameterized pipeline, with the set of variables for which your users will provide values
- Configuration
  - A **concrete implementation** of a template.
  - Configurations can inject new stages into the final pipeline graph and inherit or override, or both, triggers, notifications, and parameters.

Pipeline Terminology

- Pipeline template
  - A parameterized pipeline, minus the pipeline configuration found on a pipeline instance. This template helps developers create pipelines that follow a pattern that you establish.
- Pipeline configuration
  - The same as the configuration for a *pipeline not created from a template*, plus variable bindings and a reference to the template.
- Pipeline
  - Whether it's created from a template or not, an executable pipeline that can be visualized in the Deck UI, and can be run by Orca.
- Pipeline template variable
  - A variable defined in a pipeline template, whose value is deteremined when a pipeline is instantiated based on the template. Contrast with pipeline variables, which vary per execution of the pipeline.

Cloud Providers

Overview

- A Cloud Provider is an interface to a set of virtual resources that Spinnaker has control over.
- Typically, this is a IaaS provider, like AWS, or GCP
- It can also be a PaaS, like App Engine, or a container Orchestrator, like Kubernetes.
- The cloud provider is central to everything you do in Spinnaker.
  - It's where you deploy your Server Groups, the source of your deployable artifacts
  - and the subject of automation via Pipelines.

Accounts

- An Account is a named credential Spinnaker uses to authenticate against a cloud provider.
- Each provider has slightly different requirements for what format credentials can be in, and what permissions they need to have afforded to them.
- Each Supported Provider has their own way to create an account and register the credentials with **Halyard**.
- Keep in mind that every Provider can have as many accounts added as desired
    - This will allow you to keep your environments (e.g. staging vs. prod):
        - Separate
        - Restrict access to sets of resources using Spinnaker's Authorization mechanisms

Cloud Providers Supported

- App Engine
- Amazon Web Services
- Azure
- Cloud Foundry
- DC/OS
- Docker v2 Registry (Note: This only acts as a source of images, and does not include support for deploying Docker images)
- Google Compute Engine
- Kubernetes
- Oracle

The Process

- Install Halyard
- Choose a Cloud Provider
- Choose an Environment
- Choose a Storage Service
- Deploy Spinnaker
- Backup your Configuration
- Configure everything else
- Productionalize Spinnaker

Source: https://www.spinnaker.io/setup/install/

Application Deployment

[pipelines] | *https://www.spinnaker.io/concepts/pipelines.png*

Preliminaries

A typical install for administrators of Spinnaker

While not all necessary for this course, an administrator will typically require the following

- `brew`, `pip`, or both
- `kubectl`
- `awscliv2`
- `eksctl`
- `hal`
- `terraform`
- `helm`

Install `kubectl`

Install on MacOSX

```
brew install kubectl
```

Install on Linux

Run the following `curl` download

```
curl -LO https://storage.googleapis.com/kubernetes-
release/release/<code>curl -s https://storage.googleapis.com/kubernetes-
release/release/stable.txt</code>/bin/linux/amd64/kubectl
```

Make `kubectl` binary executable

```
chmod +x ./kubectl
```

Move the binary to the `PATH`

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

Test

```
kubectl version
```

Install `awscli`

Install on MacOSX:

```
curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-macos.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

Install on Linux:

```
curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-linux-x86_64.zip"
-o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

Source: https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html

Install `hal`

Download for MacOSX:

```
curl -O
https://raw.githubusercontent.com/spinnaker/halyard/master/install/macos
/InstallHalyard.sh
```

Download for Linux:

```
curl -O
https://raw.githubusercontent.com/spinnaker/halyard/master/install/debia
n/InstallHalyard.sh
```

Perform Installation:

```
sudo bash InstallHalyard.sh
```

Check version

```
hal -v
```

If this doesn't work re-check configuration

Configure Your AWS CLI Credentials

- Both `eksctl` and the AWS CLI require that you have AWS credentials configured in your environment.
- The aws configure command is the fastest way to set up your AWS CLI installation for general use.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSRODNN2EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

Install `eksctl` for MacOSX

Add Weaveworks Tap to `brew

```
brew tap weaveworks/tap
```

Install

```
brew install weaveworks/tap/eksctl
```

Test

```
eksctl version
```

Install `eksctl` for Linux

To install or upgrade eksctl on Linux using `curl`

Download and extract the latest release of eksctl with the following command.

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/download/latest_release/e
ksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
Move the extracted binary to /usr/local/bin.
```

```
sudo mv /tmp/eksctl /usr/local/bin
```

Test that your installation was successful with the following command.

```
eksctl version
```

Install Terraform

- Find the appropriate package for your system here and download it. Terraform is packaged as a zip archive.
- After downloading Terraform, unzip the package.
- Terraform runs as a single binary named `terraform`.
- Any other files in the package can be safely removed and Terraform will still function.
- Make sure that the terraform binary is available on the PATH.

Install Helm

Install Brew on MacOSX Using Brew

```
brew install helm
```

Install on Linux

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

Installing in AWS EKS

System Requirements

- 18 GB of RAM
- A 4 core CPU
- Ubuntu 14.04, 16.04 or 18.04

Installing EKS on Amazon

- For Production on AWS, using `eksctl`

- The following command will create maximum of 4 node for spinnaker
- Select appropriate node types, in this case *t2.xlarge*, although for this class *t3.medium*
- This will create a node group called `spin-eks-node`

```
eksctl create cluster \
    --name eks-prod \
    --version 1.14 \
    --nodegroup-name eks-prod-nodegroup \
    --node-type t3.medium \
    --nodes 4 \
    --nodes-min 0 \
    --nodes-max 4 \
    --node-ami auto \
    --write-kubeconfig=false

eksctl create cluster \
    --name eks-stage \
    --version 1.14 \
    --nodegroup-name eks-stage-nodegroup \
    --node-type t3.medium \
    --nodes 4 \
    --nodes-min 0 \
    --nodes-max 4 \
    --node-ami auto \
    --write-kubeconfig=false
```

The instructions on the spinnaker website are horrible and the CloudFormation templates are old

This can take a few minutes to get started

Update your `kubeconfig` file

Before running your process please backup your original `kubectl` file

```
aws2 eks update-kubeconfig --name eks-prod --region us-west-2 --alias
eks-prod
aws2 eks update-kubeconfig --name eks-stage --region us-west-2 --alias
eks-stage
```

Change Context to Use Kubernetes on EKS

Find all the current contexts to use in Kubernetes

```
kubectl config get-contexts
```

Select the context that was just updated by `aws eks update-kubeconfig`

```
kubectl config use-context <name>
```

Determine that your nodes are running

```
kubectl get nodes --all-namespaces
```

Halyard

Enabling Kubernetes Cloud Using Haylard

- Halyard is a tool that uploads the kubernetes configuration files for you
- Accounts are just variable names representing external services
- Ensure that you are in the right context in Kubernetes

```
hal config provider kubernetes enable
kubectl config use-context eks-prod
hal config provider kubernetes account add eks-prod --provider-version
v2 --context $(kubectl config current-context)
kubectl config use-context eks-stage
hal config provider kubernetes account add eks-stage --provider-version
v2 --context $(kubectl config current-context)
```

Enable external resources (called artifacts)

```
hal config features edit --artifacts true
```

To view a list of your Halyard Kubernetes accounts *after enabling kubernetes*

```
hal config provider kubernetes account list
```

Setting up Storage Solutions

- Set up persistent storage for your Spinnaker instance by choosing one of the options below
- Ensure that the region supports S3

- You can find your `ACCESS_KEY_ID` and your `AWS_SECRET_ACCESS_KEY` in `~/.aws/credentials`
- Use the same region as your kubernetes cluster, this was displayed by `eksctl`

```
hal config storage s3 edit \
    --access-key-id $YOUR_ACCESS_KEY_ID \
    --secret-access-key \
    --region $REGION
```

Linking Halyard to S3

Finally set up the storage to S3 for Halyard so it knows what to do

```
hal config storage edit --type s3
```

Setting up a Service Account to Prod

```
kubectl config use-context eks-prod
CONTEXT=$(kubectl config current-context)
kubectl apply --context $CONTEXT -f
https://spinnaker.io/downloads/kubernetes/service-account.yml
TOKEN=$(kubectl get secret --context $CONTEXT \
   $(kubectl get serviceaccount spinnaker-service-account \
       --context $CONTEXT \
       -n spinnaker \
       -o jsonpath='{.secrets[0].name}') \
   -n spinnaker \
   -o jsonpath='{.data.token}' | base64 --decode)
kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN
kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user
```

Setting up a Service Account to Stage

```
#Setting up Credentials for Stage

kubectl config use-context eks-stage
CONTEXT=$(kubectl config current-context)
kubectl apply --context $CONTEXT -f
https://spinnaker.io/downloads/kubernetes/service-account.yml
TOKEN=$(kubectl get secret --context $CONTEXT \
   $(kubectl get serviceaccount spinnaker-service-account \
       --context $CONTEXT \
       -n spinnaker \
       -o jsonpath='{.secrets[0].name}') \
   -n spinnaker \
   -o jsonpath='{.data.token}' | base64 --decode)
kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN
kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user
```

Tell `hal` where to deploy

```
hal config provider kubernetes account add eks-stage --provider-version
v2 --context eks-stage
```

Deploy Spinnaker

List the available versions:

```
hal version list
```

You can follow the links to the versions' respective changelogs to see what features each adds.

Set the version you want to use:

```
hal config version edit --version $VERSION
```

Deploy Spinnaker

```
hal deploy apply
```

Undeploy Spinnaker

```
hal deploy clean
```

The information is persisted on S3

Connect to Spinnaker UI

```
hal deploy connect
```

- This command automatically forwards ports 9000 (Deck UI) and 8084 (Gate API service).
- Navigate to https://localhost:9000
- Give it some time, the services need to start
- Determine the status by running the following:

```
kubectl get pods --namespace spinnaker
```

or

```
kubectl get pods --namespace spinnaker
```

If you ever need to expose publicly

- Everything in Spinnaker is a microservice interacting with one another
- As such, we can expose these services

```
export NAMESPACE=spinnaker
kubectl -n ${NAMESPACE} expose service spin-gate --type LoadBalancer
--port 80 --target-port 8084 --name spin-gate-public
kubectl -n ${NAMESPACE} expose service spin-deck --type LoadBalancer
--port 80 --target-port 9000 --name spin-deck-public

export API_URL=$(kubectl -n $NAMESPACE get svc spin-gate-public -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}')
export UI_URL=$(kubectl -n $NAMESPACE get svc spin-deck-public -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}')

hal config security api edit --override-base-url http://$API_URL
hal config security ui edit --override-base-url http://$UI_URL

hal deploy apply

kubectl -n spinnaker get svc
```

Something is wrong?

It is worth while at time when you get started to look at the logs of your pod

```
kubectl logs <pod-name> -n <namespace>
```

or

```
kubectl describe <pod-name> -n <namespace>
```

Deleting your cluster

```
eksctl delete cluster --name my-eks-cluster
```

Spinnaker Architecture

Describe the Services

Here is a list of the services that are deployed with Spinnaker

- Clouddriver - Main integration point for Spinnaker cloud providers like:
  - AWS
  - GCE
  - CloudFoundry
  - Azure etc.
- Deck - Management UI for Spinnaker
- Orca - Orchestration Engine
- Gate - Spinnaker API Gateway
- Rosco - A Bakery for Deployable Engines
- Echo - Eventing Service
- Front 50 - Spinnaker Metadata Repository Service
- Redis - Spinnaker relies on its Redis cache for a number of reasons
  - Caching Infrastructure
  - Storing live executions
  - Returning pipeline definitions faster
  - etc.
- Kayenta - Service that integrates with metric systems like Prometheus to track data

Debugging

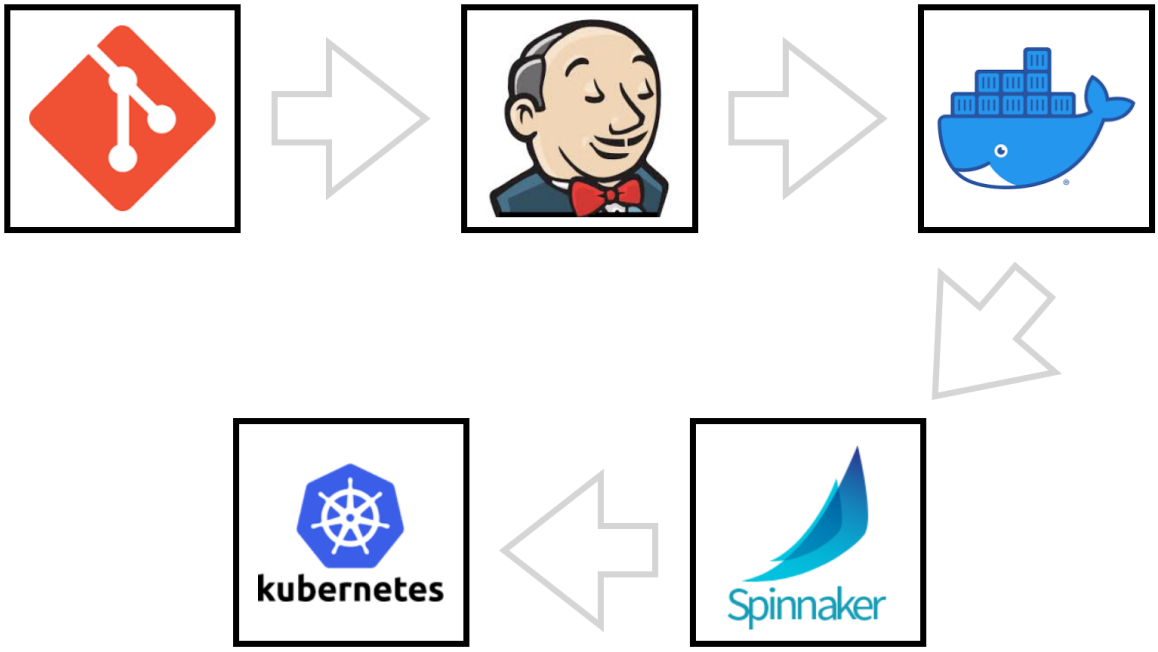
- Each Java service can be configured to listen for a debugger.

- To start the JVM in debug mode, set the Java system property `DEBUG=true`
- The JVM will then listen for a debugger to be attached on a port specific to that service.
- The service-specific debug ports are as follows:

Service	Port
Gate	8184
Orca	8183
Clouddriver	7102
Front50	8180
Rosco	8187
Igor	8188
Echo	8189

Deploying Microservices

The Process



The Steps

1. Setup a repository for a microservice
2. Setup a CI application to run unit integration tests

3. Create an Spinnaker Application
4. Define the Infrastructure
5. Create a Pipeline
6. Run your Pipeline

Source: https://www.spinnaker.io/guides/user/get-started/

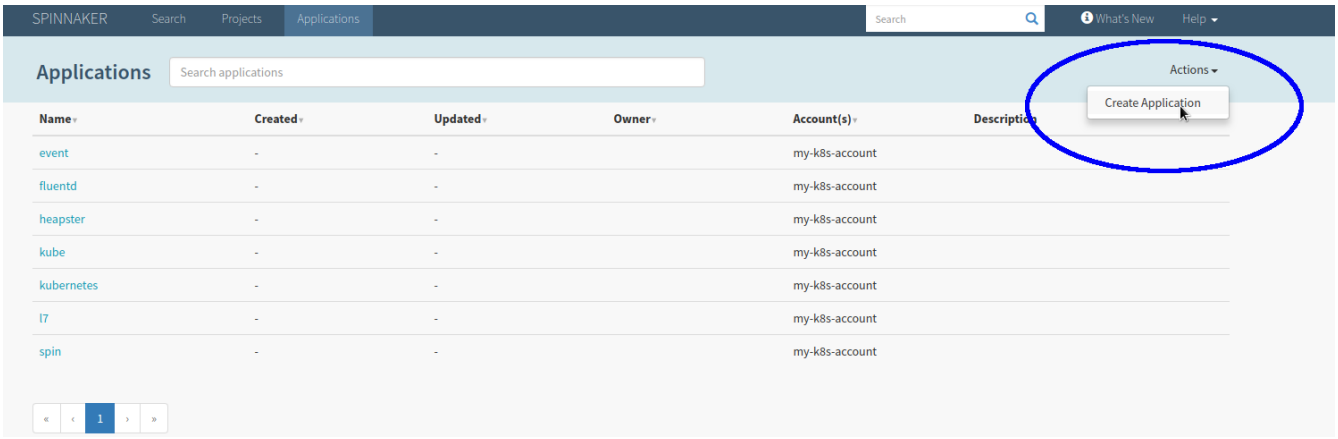
Create an Application

- One Application per Microservice
- This is the root where we will begin
- Note that your application configuration affects what you can do with Spinnaker—you can enable or disable some Spinnaker features.

About an Application

- An application in Spinnaker is a construct that represents some service that you are going to deploy (typically a microservice).
- It includes:
  - The pipelines that process the service through to deployment in production
  - The infrastructure on which the service is run:
    - Clusters
    - Server Groups
    - Firewalls
    - Load Balancers
    - Canary Configs

Creating the Application



- You can't create a deployment pipeline unless you have an application to put it in.
- Click on the **Applications** tab

- Click on Create Application
- Provide the attributes for the new Application
- Click Create

Application Attributes

Field	Required	Description
`Name`	Yes	A unique name to identify this application.
`Owner Email`	Yes	The email address of the owner of this application, within your installation of Spinnaker.
`Repo type`	No	The platform hosting the code repository for this application. Stash, Bitbucket, or GitHub.
`Description`	No	Use this text field to describe the application, if necessary.
`Consider only cloud provider health`	Bool, default=no	If enabled, instance status as reported by the cloud provider is considered sufficient to determine task completion. When disabled, tasks need health status reported by some other health provider (load balancer, discovery service).
`Show health override option`	Bool, default=no	If enabled, users can toggle previous option per task.
`Instance port`	No	This field is used to generate links from Spinnaker instance details to a running instance. The instance port can be used or overridden for specific links configured for your application (via the Config screen).
`Enable restarting running pipelines`	Bool, default=no	If enabled, users can restart pipeline stages while a pipeline is still running. This behavior is not recommended.

Editing the Configuration

If you need to go back and edit your configuration you can hit the config button on the upper right hand side

The Configuration has some of the following sections:

- Edit Notifications
- Features
- Links
- Traffic Guards
- Custom Banners

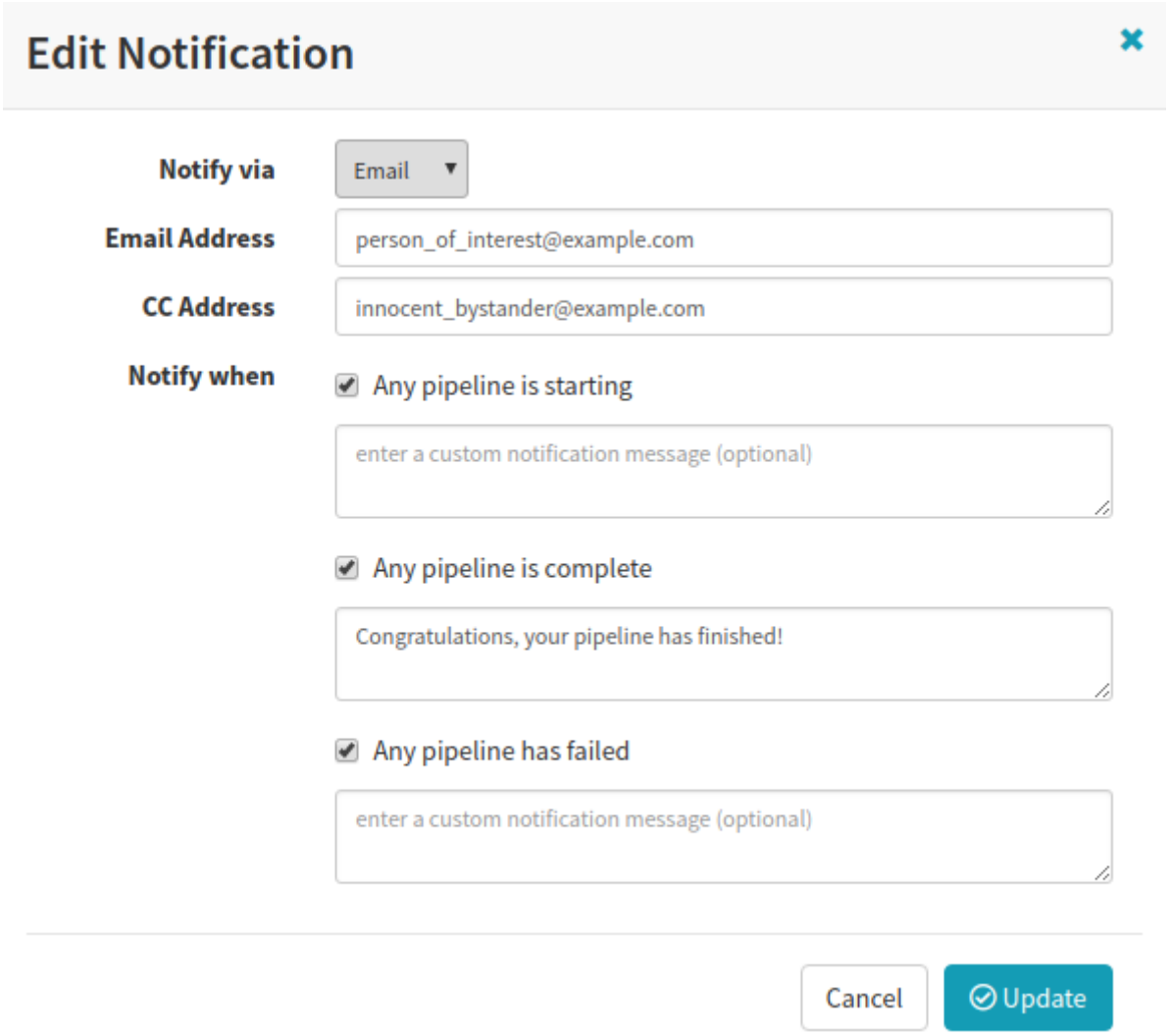
- Delete Application

Editing Notifications

Spinnaker can send notification for any application and are optional

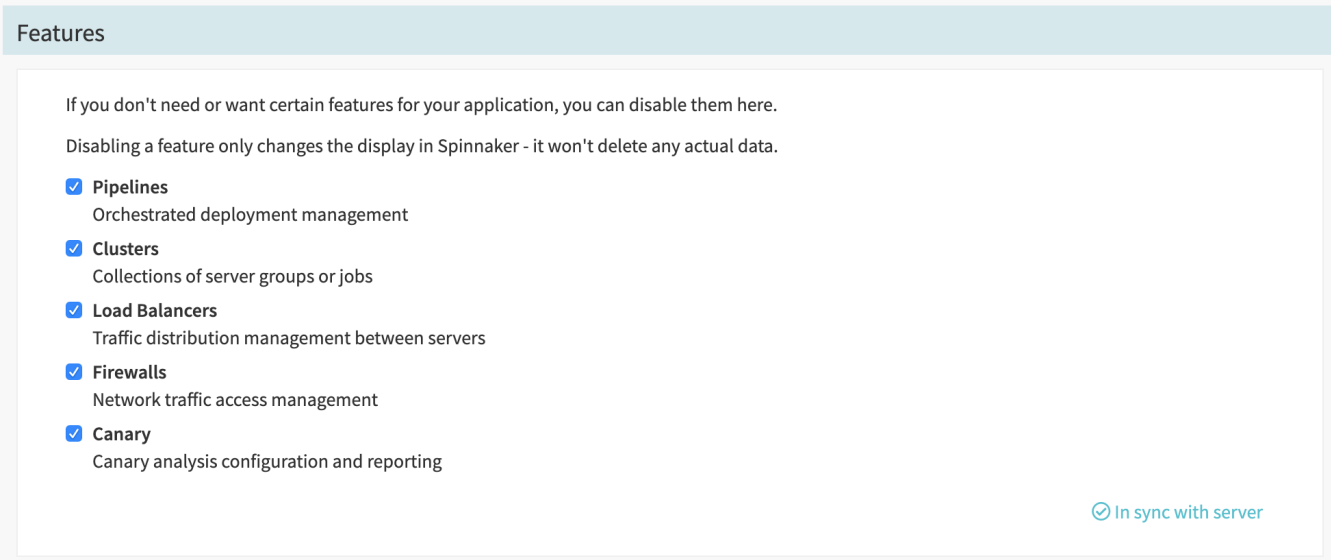
Click **Add Notification Preferences**

1. Select the notification medium:
    - Email
    - SMS
    - Slack
2. Provide the information for the type you chose:
    - Email address
    - Phone number
3. Choose each pipeline event that will trigger this notification:
    - A pipeline started
    - A pipeline finished
    - A pipeline failed
4. For each event, enter any custom text you want included in the notification.
5. Click **Update**
6. Repeat these steps to add more notifications, if you want.
7. Click **Edit** next to any existing notification to change its preferences, or click **Remove** to delete it.



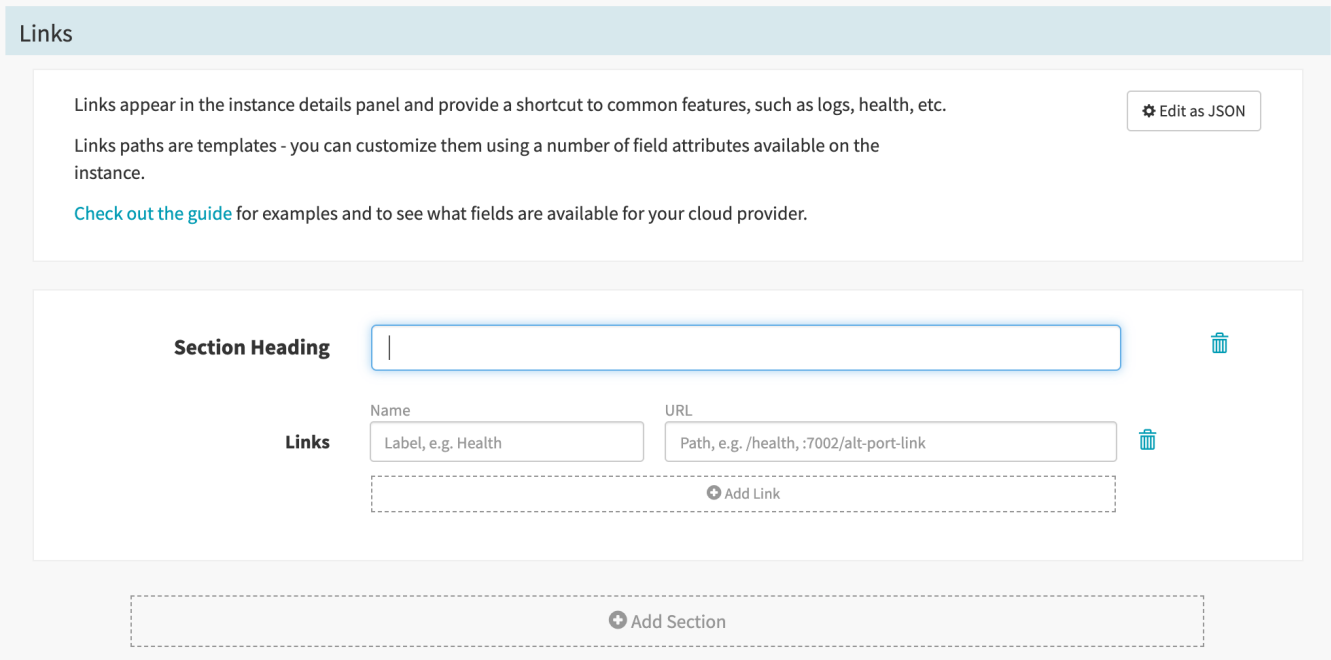
Enabling Features in Spinnaker

In order to not allow Spinnaker from annoying you about resources you don't use you can turn them off



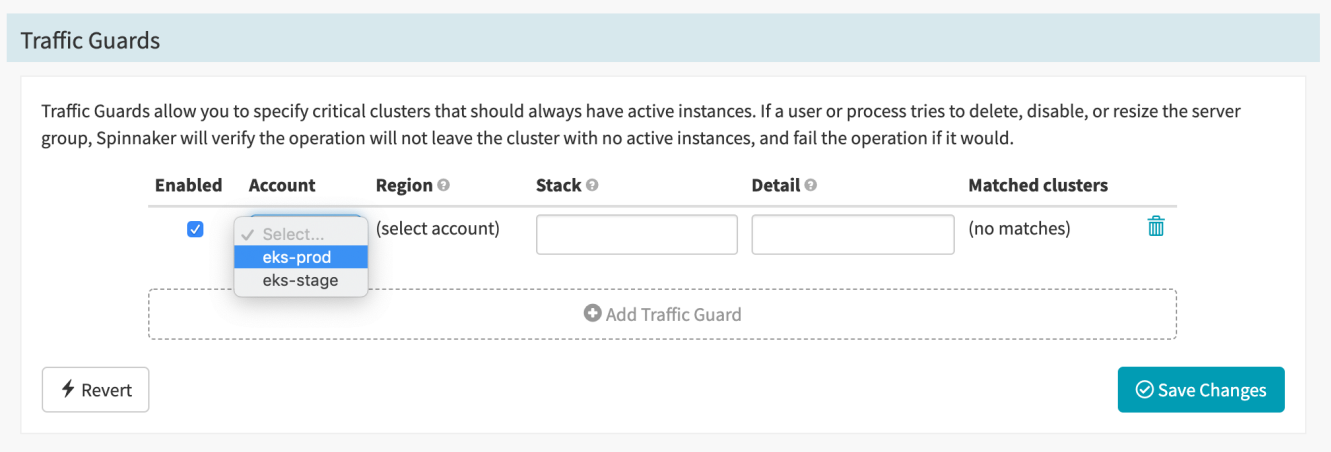
Custom Links

- You can include custom links to the instance details panel, providing shortcuts to information related to the instance, like logs, health, and so on.
- It currently doesn't work with Kubernetes resources unless exposed from cluster



1. In the **Links** section, click **Add Section**
2. Provide text for the new **Section heading**.
3. In the **Links** fields, type the text that you want to display, and the path to the link target.
4. You can reference these specific instance attributes by wrapping them in curly braces. For example, to include the region where the instance lives, for a GCE or AWS instance, use `{region}`. These are for the path only, not the link display text.
5. Click **Add Link** for each additional link you want displayed in this section.
6. Click **Add Section** for any additional sections.
7. Click **Revert** to clear your work in progress.
8. This does not remove any saved custom instance links.
9. Click **Save Changes** when you're finished.

Traffic Guards



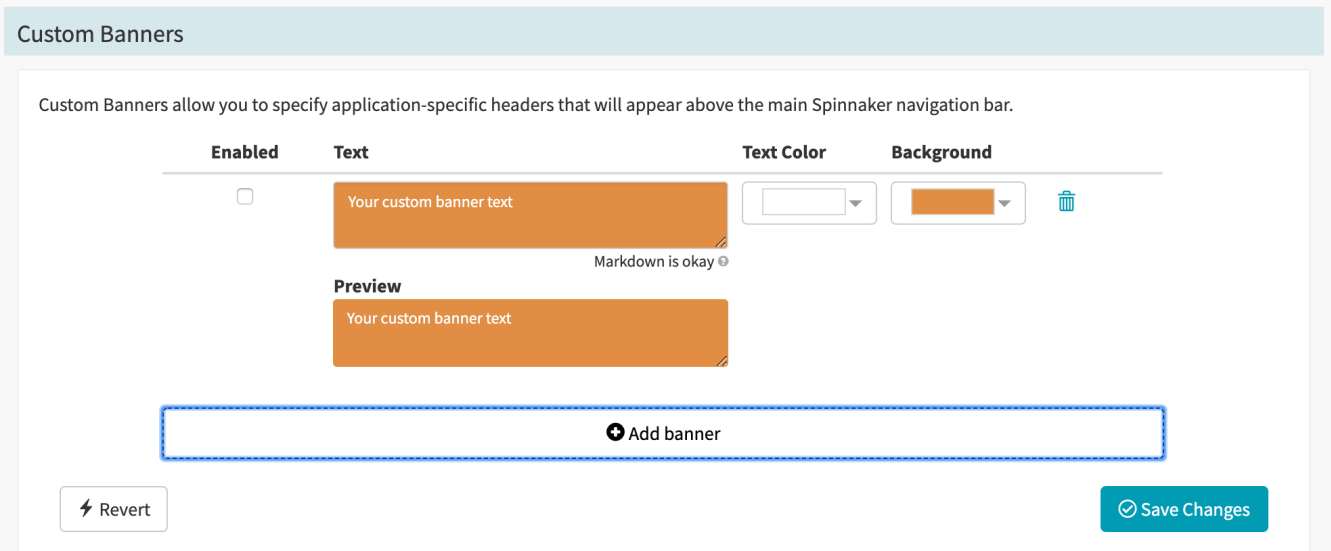
- A traffic guard is a cluster which you have designated as always having at least one active instance.
- With traffic guard enabled, if a user or process tries to delete, disable, or resize the server group, Spinnaker will verify the operation will not leave the cluster with no active instances, and fail the operation if it would.

In the Traffic Guards section:

- Click **Add Traffic Guard**
- Set the following values as seen in table below
- Click **Save Changes** to apply your new traffic guard, or changes to an existing one

Field	Required?	Description
Account	Yes	The account for which you're setting up the traffic guard
Region	Yes	The applicable region from the list, or * for all
Stack	No	The stack to which to apply this traffic guard. Leave blank to apply the guard only to a cluster that has no stack.
Detail	No	The detail string necessary to identify this cluster via the application-stack-detail naming convention

Custom Banners and Deleting the Application



You can create simple text based banners

1. Click **Add Banner**
2. Click **Enable** to enable the banner
3. Select a text color for the foreground color
4. Select a background color for the background color

Define the Infrastructure the Service will run

- You define infrastructure for each application.
- Your pipelines deploys services to the server groups you define.
- This means that production environments can be deployed without a pipeline.
- This will provide you with the initial production environement

this step is not a prerequisite for creating pipelines. In fact, you can use pipelines to create infrastructure.

Pipelines

- How we control and deploy our application
- Series of stages that can be combined in any order
- Can be automatic or manual
- Triggered by a wide arrange of inputs

Create a Pipeline

- After creating the Application, navigate the **Pipelines** tab in your navigation

[create] | *https://www.spinnaker.io/guides/user/pipeline/managing-pipelines/images/create.png*

- Name your pipeline
- Then add stages

Configuration Stage

- The initial stage that you are provided is the configuration stage
- It is where we define:
  - *expected artifacts*
  - *triggers*
  - *parameters*
  - *notifications*

Adding Stages

- The first step in any pipeline is **Configuration**, where you can set up pipeline triggers and parameters.
- Each stage specifies an action the pipeline will take once it's configured.
- The steps include
  - Select Add stage from your pipeline configuration screen.
  - Set the stage type using the drop-down menu.
  - If this isn't the first stage in your pipeline, make sure that this stage depends on the desired upstream stage(s) using the Depends on field.

[add stage] | *https://www.spinnaker.io/guides/user/pipeline/managing-pipelines/images/add-stage.png*

[stage depends on] | *https://www.spinnaker.io/guides/user/pipeline/managing-pipelines/images/stage-*

depends-on.png

Manually Run your Pipeline

You can run a pipeline manually, but most pipelines are triggered automatically. If you so care to, particularly if you don't want to redo the build, you can click on **Start Manual Execution**

[manual execution] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/manual-execution.png

Disabling a Pipeline

- From the Pipelines tab, click **Configure** to modify an existing pipeline
- Click Pipeline actions in the upper right corner, and select **Disable**.

[pipeline actions] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/pipeline-actions.png

Deleting a Pipeline

- From the Pipelines tab, click **Configure** to modify an existing pipeline.
- Click **Pipeline actions** in the upper right corner, and select **Delete**.

[pipeline actions] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/pipeline-actions.png

Editing as JSON

- You can also edit your pipeline in JSON
- This is fraught with danger if you get it wrong
- It can be helpful if things go wrong already
  - Perhaps the UI is not updating the backend correctly
  - You wish to diagnose issues
- When you use the **Edit as JSON** feature, you are directly editing the payload.
- Edit as JSON allows you to set pipeline fields or properties not exposed by the UI.

[pipeline actions] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/pipeline-actions.png

Revision History

- Each time you save your pipeline, the current version is added to revision history
- You can use revision history to diff two versions of a pipeline or to restore an older version of a pipeline.
    1. From the **Pipelines** tab, click **Configure** to modify an existing pipeline
    2. Click **Pipeline Actions** in the upper right corner, and select **View revision history**

[pipeline actions] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/pipeline-actions.png

Changing Versions

- You can compare any version of your pipeline to either the version before it or the current pipeline.
    1. View the version history
    2. Select a version from the Revision drop-down.

[revision history] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/revision-history.png

Comparing Version

You can also compare a review to either the current revision or the previous one

[compare version] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/compare-version.png

Reverting your Pipeline

If you select a different version, you can revert to that same version

[restore revision] | *https://www.spinnaker.io/guides/user/pipeline/managing-pipelines/images/restore-*

revision.png

Rename a Pipeline

1. From the **Pipelines** tab, click **Configure** to modify an existing pipeline.
2. Click **Pipeline actions** in the upper right corner, and select **Rename**

[pipeline actions] | *https://www.spinnaker.io/guides/user/pipeline/managing-*

pipelines/images/pipeline-actions.png

Artifacts

Artifacts in Spinnaker

- Named JSON object that refers to an external resource
  - A Docker image
  - A file stored in GitHub
  - An Amazon Machine Image (AMI)
  - A binary blob in Amazon S3, Google Cloud Storage, etc.

Artifact Frustration

Artifacts are one of the most frustrating aspects to Spinnaker

```
Failed on startup: Unmatched expected artifact ExpectedArtifact
(matchArtifact=Artifact(type=s3/object, customKind=false, name
=s3://XXXXXXXXXXX-alpha.191%2B926bb3b3.tgz, version=null, location=null,
reference=null, metadata={id=01317db7-1e30-4c02-a302-826449d83f94},
artifactAccount=f5cs-spin, provenance=null, uuid=null),
usePriorArtifact=false, useDefaultArtifact=false, defaultArtifact
=Artifact(type=null, customKind=true, name=null, version=null, location
=null, reference=null, metadata={id=ca86ca54-6719-471e-b10e-
fb4a6f20e85d}, artifactAccount=null, provenance=null, uuid=null), id
=0730e728-e19a-43af-b9e5-9856332e939d, boundArtifact=null) could not be
resolved.
```

What is the "address" of our Artifacts?

- URI of the location of our artifact
- Provenance information about the commit that triggered the resource

Provenance *def:* - The place of origin or earliest known history of something

Artifact Decoration

- To allow more provenance information, a JSON is used to describe extras like:
  - name
  - version
  - location
- This is called *artifact decoration*

- Every Spinnaker artifact follows this specification whether:
  - Supplied to a pipeline
  - Accessed within a pipeline
  - Produced by a pipeline–follows

Artifact Format

As an example, an object stored in Google Cloud Storage (GCS) might be accessed using the following Spinnaker artifact:

```
{
  "type": "gcs/object",
  "reference": "gs://bucket/file.json#135028134000",
  "name": "gs://bucket/file.json",
  "version": "135028134000"
  "location": "us-central1"
}
```

Another example

```
{
  "type": "docker/image",
  "reference": "gcr.io/project/image@sha256:29fee8e284",
  "name": "gcr.io/project/image",
  "version": "sha256:29fee8e284"
}
```

Spinnaker Artifact Fields

Field	Explanation	Notes
`type`	How the external resource is classified. (This allows for easy distinction between Docker images and Debian packages, for example).	
`reference`	The URI used to fetch the resource.	
`artifactAccount`	The Spinnaker artifact account that has permission to fetch the resource.	
`version`	The version of the resource. (By convention, version should only be compared between artifacts of the same type and name.)	Optional.
`provenance`	The relevant URI from the system that produced the resource. (This is used for deep-linking into other systems from Spinnaker.)	Optional.

Field	Explanation	Notes
`metadata`	Arbitrary key / value metadata pertaining to the resource. (This can be useful for scripting within pipeline stages.)	Optional.
`location`	The region, zone, or namespace of the resource. (This does not add information to the URI, but makes multi-regional deployments easier to configure.)	Optional.
`uuid`	Used for tracing the artifact within Spinnaker.	Assigned by Spinnaker.

Expected Artifacts

- Expects a particular artifact to be available
- Declarable within a pipeline trigger or stage
- Spinnaker compares an incoming artifact (for example, a manifest file stored in GitHub) to the expected artifact (for example, a manifest with the file path `path/to/my/manifest.yml`)
- If there is a match it is *bound* to that artifact and used

Expected and Bound Artifact

[expected artifact github file] | *https://www.spinnaker.io/reference/artifacts-with-*

artifactsrewrite/expected-artifact-github-file.png

Matched Artifacts

- Used to disambiguate between similar artifacts coming from the same account
  - e.g. Different `yaml` files from the same repository
- Filter by fields in the artifact format against which to compare in the incoming artifact
- Used to specify that the trigger should begin pipeline execution only if the incoming artifact matches the parameters that you provided

Prior execution and Default Artifacts

- **If Missing**
  - Provide fallback behavior for the expected artifact in case the trigger doesn't find the desired artifact.
- **Use prior execution**
  - Spinnaker will fall back to the artifact used in the last execution.
  - If you enable the **Use default artifact** Spinnaker will use a default artifact,
  - Allows you to provide fallback behavior for the first time a trigger is used, when no previous execution exists

[pubsub trigger artifact] | *https://www.spinnaker.io/reference/artifacts/in-pipelines/pubsub-trigger-*

artifact.png

Review of Artifacts

- Expected - An artifact that a trigger or stage expects to have
- Bound - An artifact that has been fulfilled
- Matched - Enter fields to disambiguate similar artifacts

Artifacts in Pipelines

Defined:

- An artifact arrives in a pipeline execution either from an external trigger (for example, a Docker image pushed to a registry)
- Getting fetched by a stage. That artifact is then consumed by downstream stages based on predefined behavior.

Expected Artifacts

- Enable a stage to bind an artifact from:
  - Another pipeline execution
  - Stage output
  - Running environment

Triggers

When configuring a pipeline trigger, you can declare which artifacts the trigger expects to have present before it begins a pipeline execution!

[expected artifact trigger] | *https://www.spinnaker.io/reference/artifacts/in-pipelines/expected-*

artifact-trigger.png

Find Artifact from Execution

- To allow you to promote artifacts between executions, you can make use of the “Find Artifact from Execution” stage.
- All that’s required is the pipeline ID whose execution history to search, and an expected artifact to bind.
- This will be particularly useful for canary attainment of baseline

[find artifact from execution] | *https://www.spinnaker.io/reference/artifacts/in-pipelines/find-artifact-*

from-execution.png

Artifacts Passed between Pipelines

- Artifacts can be passed between pipelines.
- Two cases:
  - Pipeline B is triggered by Pipeline A - Pipeline B is triggered by Pipeline A, therefore Pipeline B will have Pipeline A's artifacts
  - Pipeline A triggers Pipeline B - Pipeline A is triggers Pipeline B, therefore Pipeline B will have Pipeline A's artifacts

Stages that Produce Artifacts

Stages can be configured to 'Produce' artifacts if they expose the following Stage configuration:

[produced artifact] | *https://www.spinnaker.io/reference/artifacts/in-pipelines/produced-artifact.png*

Two ways:

1. To bind artifacts injected into the stage context - If your stage emits artifacts (such as a "Deploy (Manifest)" stage) into the pipeline context, you can match against these artifacts for downstream stages to consume.
2. To artificially inject artifacts into the stage context - If you are running a stage that does not natively emit artifacts (such as the "Run Job" stage), you can use the default artifact, which always binds to the expected artifact, to be injected into the pipeline each time it is run. Keep in mind: If the matching artifact is empty, it will bind any artifact, and your default artifact will not be used.

If you are configuring your stages using JSON, the expected artifacts are placed in a top-level `expectedArtifacts: []` list.

Various Types of Artifacts

There are various types of artifacts or you can define your own:

- Docker Image
- Embedded Base64
- GCS Object
- S3 Object
- Git Repo
- GitHub File
- GitLab File
- Bitbucket File

- HTTP File
- Kubernetes Object
- Oracle Object

Source: https://www.spinnaker.io/reference/artifacts/types/overview

Sources

https://www.spinnaker.io/reference/artifacts-with-artifactsrewrite/ https://www.spinnaker.io/reference/artifacts-with-artifactsrewrite/in-kubernetes-v2/

Linking Spinnaker to Github

- In order to connect to github, we will need to establish a token
- This will build a connection from Spinnaker to Github
- As users you will create the token
- As administration you will put the token in Spinnaker using Halyard

Creating a token

- Visit your github account at: https://github.com/settings/tokens
- Click on the **Generate new token** button

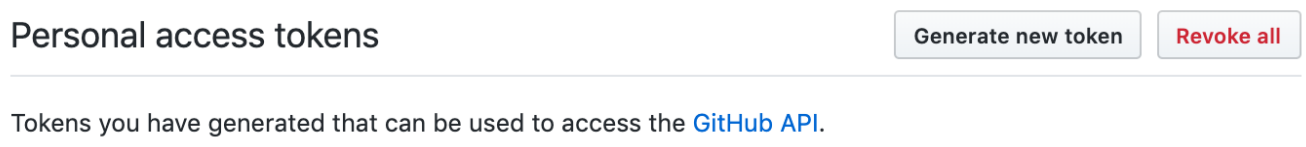


Figure 3. Token Page

- Select the `public_repo` scope

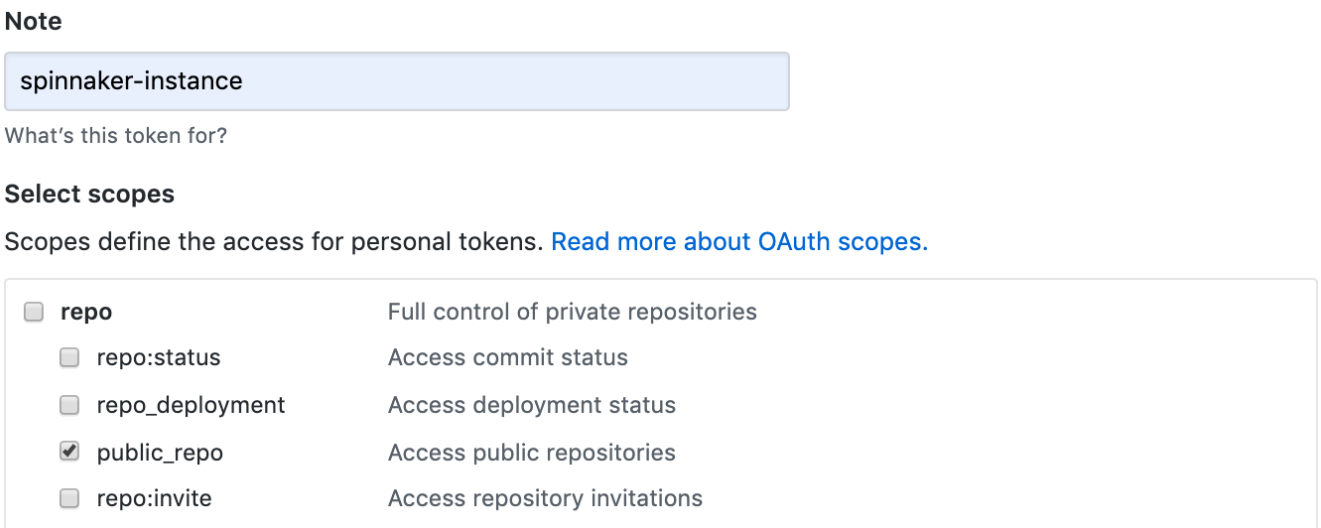
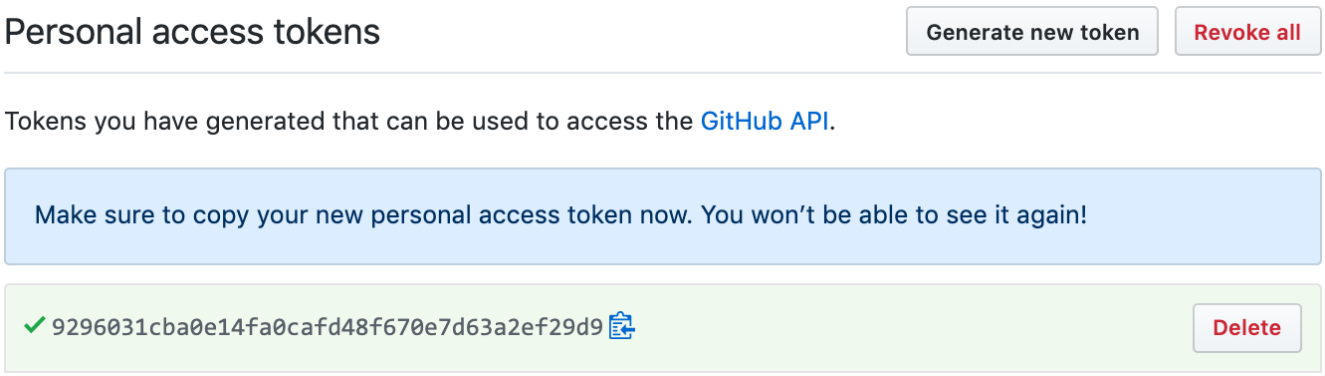


Figure 4. Allowing Use

Copying the token

- You have one shot to copy the token and secure it
- Copy it, by clicking on the clipboard icon next to the hash



Setting up the token in Spinnaker

Paste the token and store it in a temporary file

```
pbpaste > token_file.txt
```

Setup some temporary fields for both the location of token file, and the github account name

```
TOKEN_FILE=~/token_file.txt
ARTIFACT_ACCOUNT_NAME=<github-username>-account
```

Installing your Github connection

```
hal config features edit --artifacts true
hal config artifact github enable
```

Add your account:

```
hal config artifact github account add $ARTIFACT_ACCOUNT_NAME \
    --token-file $TOKEN_FILE
```

Deploy to Spinnaker

```
hal deploy apply
```

Lab: Deploying EKS Microservices

Preliminary

1. Please work in a group
   a. We will be running a lot of instances
   b. Dev-ops is maddening
2. We will let a fun service decide your teamname: http://creativityforyou.com/combomaker.html

The instructor is your administrator!

Your instructor will:

- Create an ECR repository
- Create a github connection using a token

Forking A Repository

1. Fork the repository https://github.com/dhinojosa/quarkus-microservice into your own public github account.
2. Notice the `manifests/deploy-microservice.yaml` file, and see the Deployment, and the Service

We will have one of you set up a token in your github repository, which will provide access to the spinnaker server, please delete at the end of class.

Editing your manifest

1. In your personal fork, edit your `manifest/deploy-microservice.yml`
2. Perform a *replace all* in the document and replace `quarkus-microservice` with `<your-team-name>-microservice`
3. Save file, commit, and push your file in your repository

Setting up a Jenkins job

1. Open Jenkins at http://44.230.82.221:8080/ and login with the username **spinnaker** and password **timeisluxury**
2. Click on **New Item**, and a **Pipeline Job** in Jenkins
3. Name your job **<your-team-name>-microservice**, and click **Ok**
4. Under **Build Triggers**, select **Poll SCM**, and in the text area enter `* * * * *`
5. Under **Pipeline**, change **Definition** to **Pipeline Script from SCM**

6. Change **SCM** to `Git`, and enter your repository URL that you cloned for your team

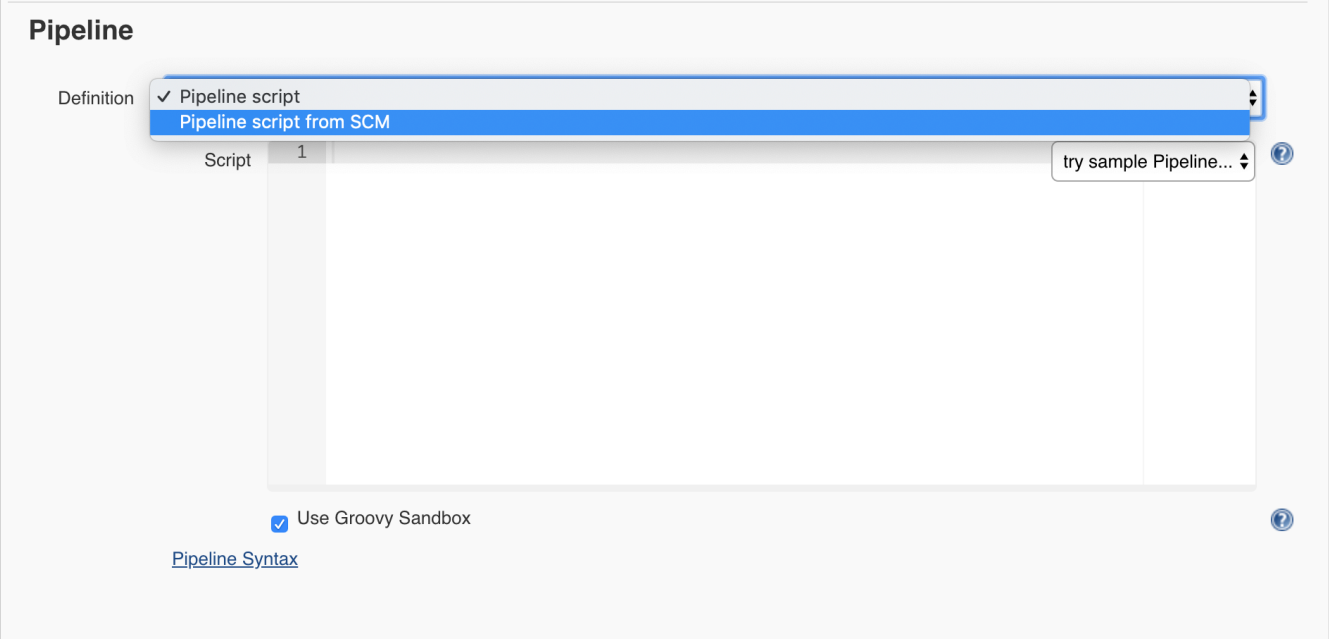


Figure 5. Selecting `Jenkinsfile` that is stored already on the repository

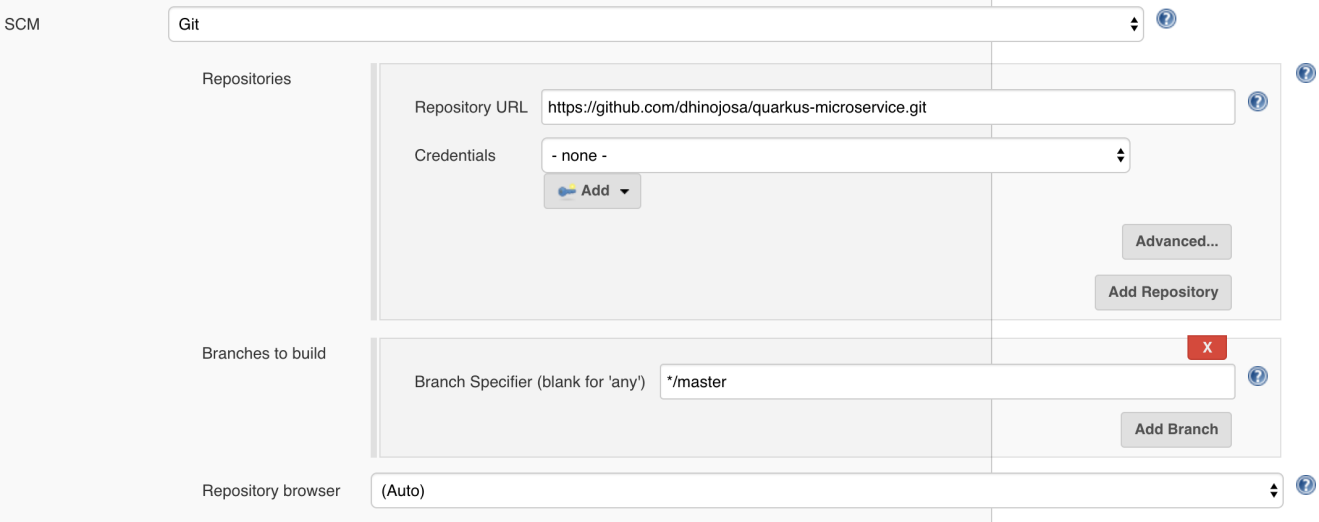


Figure 6. Selecting the Repository itself.

Setting up a Spinnaker Application

1. Open Spinnaker at address given by instructor, this is not secure so don't place any personal information:
2. Once there create an Application with the name: **<your-team-name>-microservice**
3. Add an email
4. In Repo Type add `github`
5. In Description, add a description of your choosing
6. Create an instance port of `80`

Figure 7. Creating a Spinnaker Application

Creating a Pipeline

1. Click on Pipelines, and click on "Configure a new pipeline"
2. Under **Pipeline Name**, name it "ToStaging"

Github Expected Artifact

We will require two artifacts:

- The Deployment `yaml` from Github
- The Docker Container from ECR

Setting up the Github Expected Artifact

1. In **Configuration** under the **Expect Artifacts** section, click on **Add Artifact**
2. In **Account** field, choose your `github` account
3. In **File Path**, enter `manifests/deploy-microservice.yaml`, remember where you saw this entry
4. In **Display Name**, enter 'deploy-microservice.yaml'
5. Check **Use Default Artifact**
6. Select your determined `github` account
7. Enter in your content url and keep in mind this is a different kind of url: `https://api.github.com/repos/$ORG/$REPO/contents/$FILEPATH`

For Example: `https://api.github.com/repos/dhinojosa/quarkus-microservice/contents/manifests/deploy-microservice.yml`

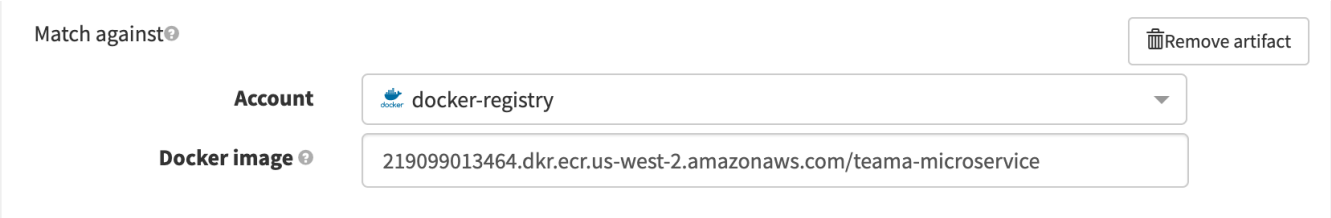
1. In **Commit/Branch** enter `master`

Docker Registry Artifact

1. In **Configuration** under the **Add Artifacts** section, click on **Add Artifact**
2. In **Account**, choose **Docker Registry** Account
3. In **Docker Image**, add `219099013464.dkr.ecr.us-west-2.amazonaws.com/` followed by

your <team-name>-microservice name. This will typically be your project name followed by `microservice`, ask the instructor to show what is on the ECR, or have them show you.

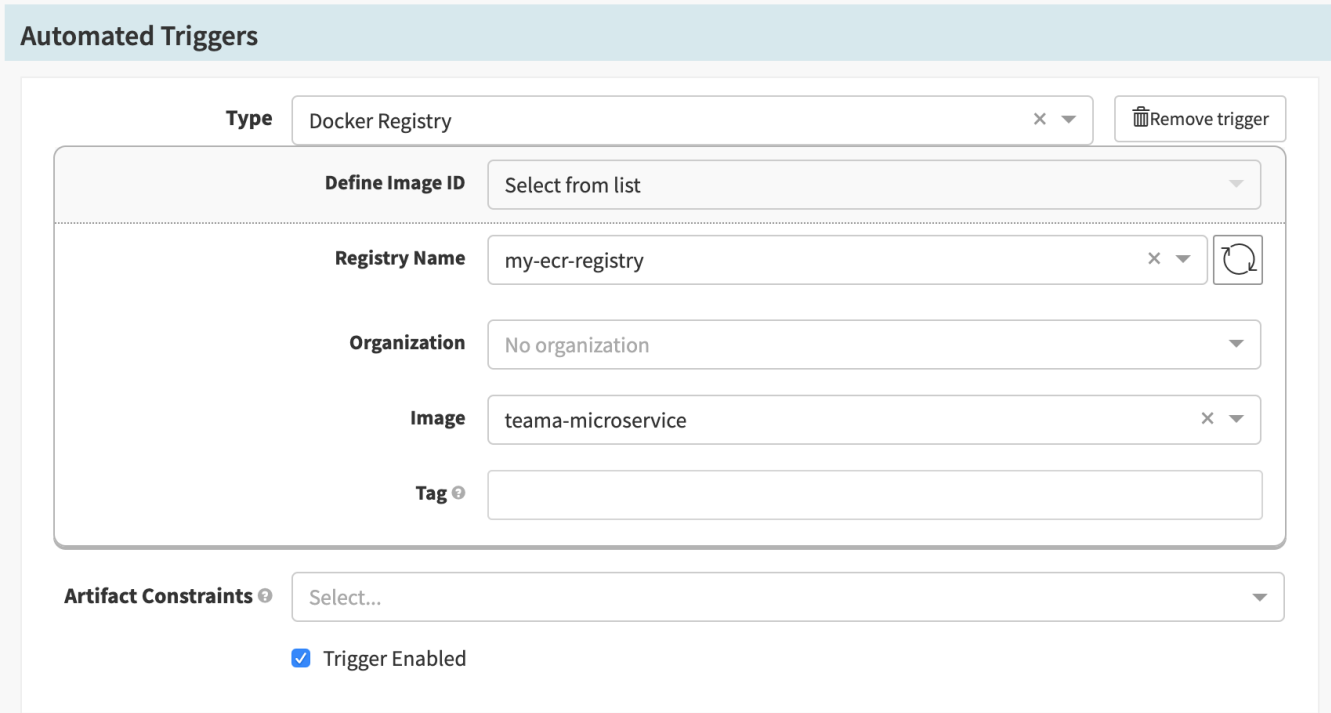
4. In **Display Name**, enter '**<your-team-name>**-microservice'



If this doesn't work, later we can add a default tag to the container

Adding a Docker Registry Container Trigger

1. In **Automated Triggers** section, click **Add Trigger**
2. In the **Type** drop-down field, select **Docker Registry**
3. In the **Registry Name** field, select `my-ecr-registry`
4. In the **Image** field, select your image



If your image name does not show, ask the administrator to add that to ECR

Adding a Notification

1. In the **Notifications** Section, click **Add Notification Preference**, this will produce a dialog box
2. In the **Notify Via**, select `Email`
3. In the **Email Address**, enter your email address

4. Select all the boxes for all notification

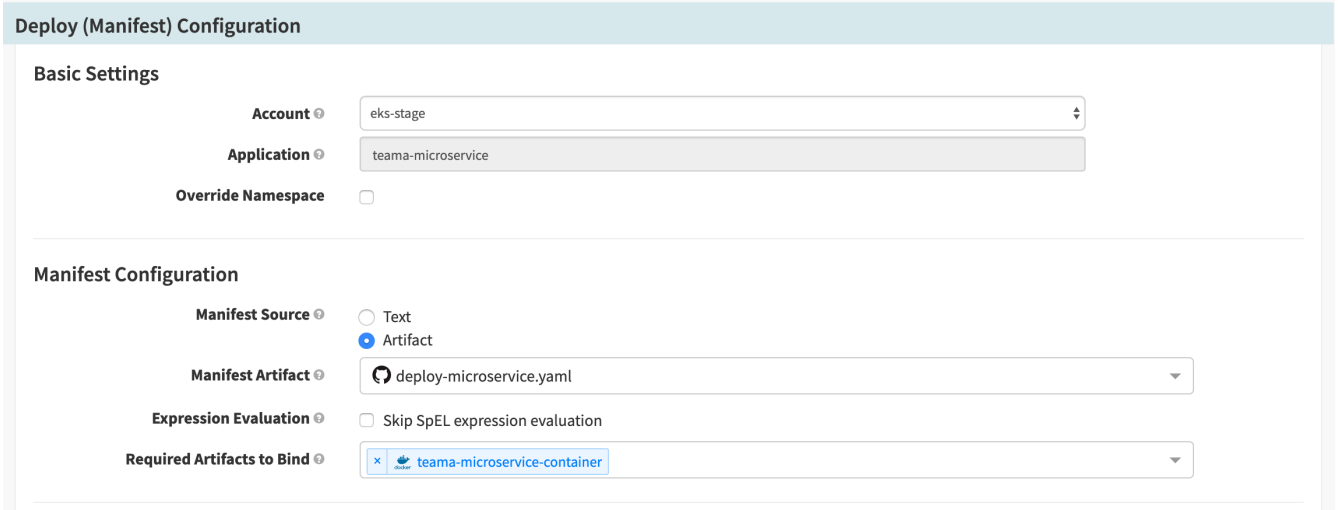
This will get annoying over time, turn it off if needed.

Save Changes

1. Click the Save Changes Button

Adding a Deploy Stage

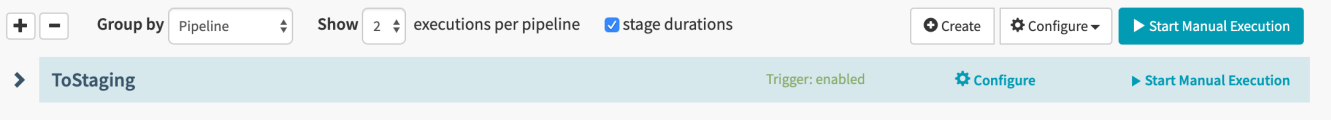
1. In your pipeline, click on the **Add Stage** button
2. In the **Type** field, select **Deploy (Manifest)**
3. This will present some new sections and fields
4. Go to the **Deploy (Manifest) Configuration** and **Basic Settings** section, in **Account** select `eks-stage`
5. In the **Manifest Configuration** section, and for the **Manifest Source**, select **Artifact**
6. In the **Manifest Artifact** select your `deploy-microservice.yaml` file
7. In the **Required Artifacts to Bind** select your container artifact binding



Deploying to Staging

1. Click on the pipelines icon

2. Click on **Start Manual Execution**
3. Check Results
4. If it doesn't work, check configurations



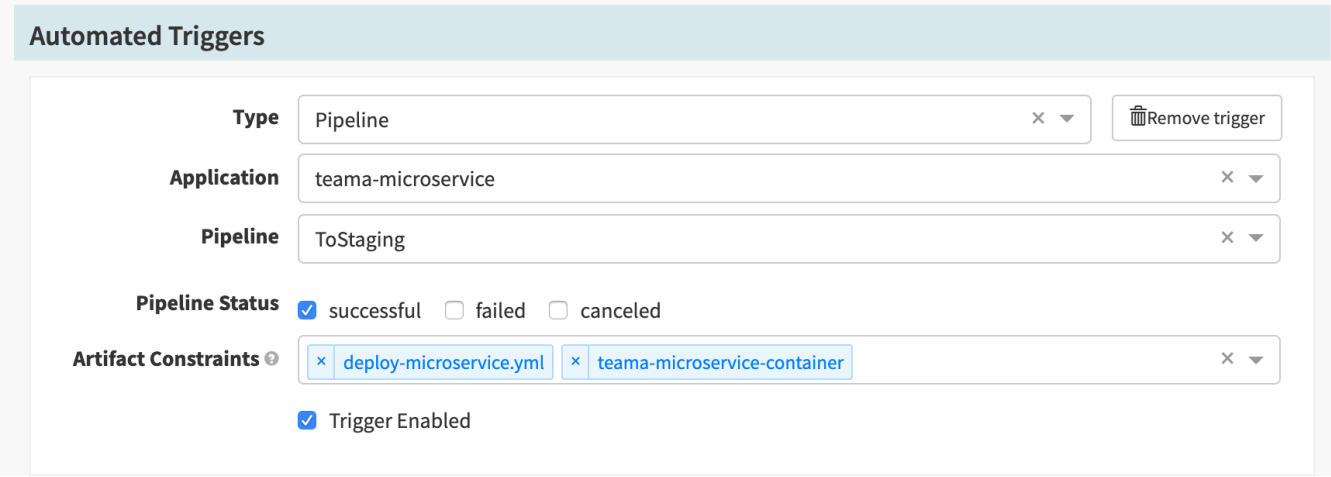
Deploying to Production

Creating a Pipeline

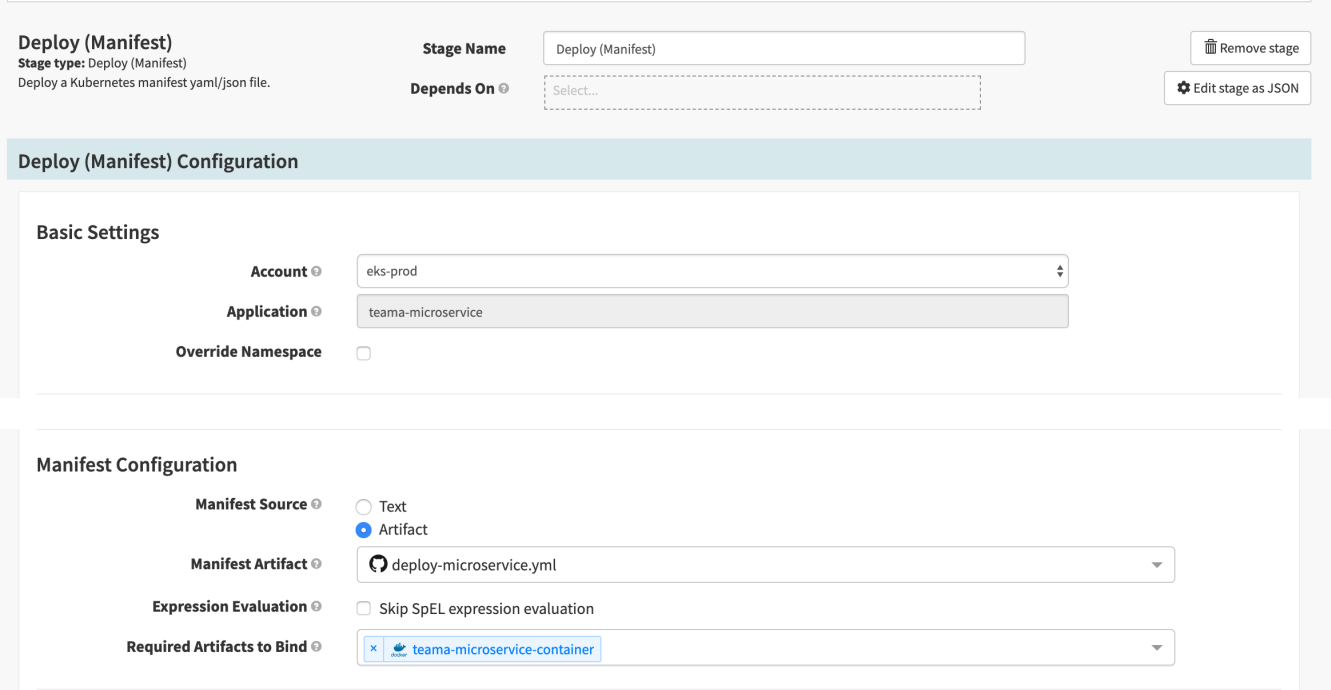
1. Click on Pipelines, and click on "Configure a new pipeline"
2. Under **Pipeline Name**, name it "ToProd"

Production Trigger

1. In the new Pipeline Configuration, go to **Automatic Triggers**
2. In the **Type** field select **Pipeline**
3. In the **Application** field select your Application name (e.g. teama-microservice)
4. In the **Pipeline** field select **ToStaging**, which was the pipeline that was created
5. In the **Pipeline Status** select **Successful**
6. For **Artifact Constraints** add `deploy-microservice.yml` and `<your-team-name>-microservice-container`



Adding a Deploy to Prod Stage



1. In your pipeline, click on the **Add Stage** button
2. In the **Type** field, select **Deploy (Manifest)**
3. This will present some new sections and fields
4. Go to the **Deploy (Manifest) Configuration** and **Basic Settings** section, in **Account** select `eks-prod`
5. In the **Manifest Configuration** section, and for the **Manifest Source**, select **Artifact**
6. In the **Manifest Artifact** select your `deploy-microservice.yml` file
7. In the **Required Artifacts to Bind** select your container artifact binding

Put it all in motion

- Make a change in your repository
- Commit and Push
- Ensure that your pipeline works!

Bonus: Place a manual approve in stage

After you run initially and had time to play around

1. Go back to the "ToStaging" pipeline
2. Select the **Deploy (Manifest)** stage
3. Click the **Add Stage** button
4. Select **Manual Judgement**
5. Add some descriptions in the **Instructions** field

Discover your space

- Ask the instructor for the ip address for your application
- Edit the `index.html` in the `src/main/resources/META-INF.resources/` folder
- Change the color, change the text
- Perform a `git push` and see the results

Things to discuss

- How do the yaml files work
- Ask the instructor to show the results on their `kubectl`
- What bad things happened?

Expressions

- Dynamically set and access variables during pipeline execution
- Any text field can use an expression
- Use them to:
  - Use arbitrary values about the state of your system in the execution of your pipelines.
  - Turn on or off particular stages or branches of your pipeline
  - Dynamically name your stack
  - Check the status of another stage
  - Perform other operations.

What are expressions?

A pipeline expression is made up of $ followed by opening/closing brackets: `${ }`

```
${expression here}
```

String interpolation

```
${expressionA}-randomString-${expressionB}
```

Given "Hello" for `expressionA` and `world` for `expressionB`, this will result in:

```
Hello-randomString-world.
```

Where can Expressions be used?

- Pipeline expressions any place in the UI where you can enter free-form text
- Exception of the pipeline **Configuration** stage with exception of Expected Artifacts and "Use Default Artifact"
- If text field is unavailable, there is a feature called **Edit as JSON**

When are they evaluated

- Expressions are evaluated at the beginning of that stage
- **Configuration** needs to be evaluated *before* expression

Helper Functions

- Helper functions simplify common use cases
  - Access a particular stage by name
  - Strip non-alphanumerical characters
  - Parse JSON

Finding Helper Functions

Adding a pound sign (`#`) within your pipeline expression displays a list of all the helper functions that are available

[helper func autocomplete] |

https://www.spinnaker.io/guides/user/pipeline/expressions/images/helper-func-autocomplete.png

Helper Properties

Helper properties are variables which refer to global information about the current pipeline execution.

- `execution` - refers to the current pipeline execution
- `trigger` - information about the pipeline trigger
  - e.g. Jenkins trigger and want to know which build triggered the current pipeline
  - `${trigger["buildInfo"]["number"]}`

Getting Help with Helper Properties

- List available helper properties and stages enter a question mark (`?`) into your pipeline expression
- Type a little more of what you are looking for to narrow down your choices

[helper properties list] | *https://www.spinnaker.io/guides/user/pipeline/expressions/images/helper-*

properties-list.png

Once a helper property is added to the expression you can use any of the meta keys (Shift, Command, Alt, Control) to bring up a list of all the pipeline context that is relevant to the selected helper property.

Context Values

- *Context Values* are specific to a stage.
- Includes:
  - Stage Name
  - Status
  - Start and End Time

Finding Context Values

[execution json] | *https://www.spinnaker.io/guides/user/pipeline/expressions/images/execution-*

json.png

Figure 8. Click on the source link under the pipeline execution, which opens a new tab containing the JSON details of your pipeline execution.

Testing Pipeline Executions

- Testing is done through the API
- You need a running instance of Spinnaker
- Use **Source** of any stage and identify the pipeline execution

[execution json] | *https://www.spinnaker.io/guides/user/pipeline/expressions/images/execution-*

json.png

- Then use `curl`

```
PIPELINE_ID=[your_pipeline_id]
curl http://api.my.spinnaker/pipelines/$PIPELINE_ID/evaluateExpression \
       -H "Content-Type: text/plain" \
       --data '${ #stage("Deploy").status.toString() }'
```

This then returns if successful

```
{"result": "SUCCEEDED"}
```

Otherwise

```
{
  "detail":
    {
      "{ #stage(\"Deploy\").status.toString() ":
        [
          {
            "description":
              "Failed to evaluate [expression] Expression
              [{ #stage( #root.execution, \"Deploy\").status.toString()
]
              @0: No ending suffix '}' for expression starting at
character 0:
              { #stage( #root.execution, \"Deploy\").status.toString()
",
            "exceptionType"
:"org.springframework.expression.ParseException",
            "level":"ERROR",
            "timestamp":1531254890849
          }
        ]
    },
  "result":"${#stage(\"Deploy\").status.toString() "
}
```

Baking

Baking turns templates into manifests with the help of a templating engine. Helm relies on the `helm template` command.

Reminder on Helm Templates

We can take any `yaml` resource meant for Kubernetes

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mychart-configmap
data:
  myvalue: "Hello World"
```

And templatize it:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
```

`{{ .Release.Name }}` will be replaced by the name of the release when the end user runs

```
helm install --debug --dry-run goodly-guppy ./mychart
```

This will return:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: goodly-guppy-configmap
data:
  myvalue: "Hello World"
```

Helm Charts offers a lot

- Values
- Defaults
- Template Functions and Pipelines
- Flow Control
- Variables
- Embedded Templates

Read more: https://helm.sh/docs/chart_template_guide/getting_started/

Intentions

This stage is intended to help you package and deploy applications that you own, and are actively developing and redeploying frequently. It is not intended to serve as a one-time installation method for third-party packages. If that is your goal, it's arguably better to call helm install once when bootstrapping your Kubernetes cluster.

Configure Artifact support in Hal

- Make sure that you have configured artifact support in Spinnaker first in Halyard
- All Helm charts are fetched/stored as artifacts in Spinnaker

Create a Bake Manifest Stage

- Create a "Bake (Manifest)" stage

Configure the Bake Manifest Stage

- Enter the release name (required) - The Helm release name for this chart. This determines the name of the artifact produced by this stage
- The template artifact (required) - The Helm chart that you will be deploying, stored remotely as a `.tar.gz` archive
- The release namespace - The Kubernetes namespace to install release into. If parameter is not specified default namespace will be used
- Zero or more override artifacts (optional) - The files passed to `--values` parameter in the helm template command. Each is a remotely stored artifact representing a Helm Value File.
- Statically specified overrides - The set of static of `key/value` pairs that are passed as `--set` parameters to the `helm template` command.

Helm Charts and Namespace

Make sure you have `namespace` declaration in your helm chart if you are overriding the `namespace`

```
metadata:
  namespace: {{ .Release.Namespace }}
```

Producing Baked Artifacts

In the “Produces Artifacts” section:

- Spinnaker has automatically created an `embedded/base64` artifact
- The artifact is bound when the stage completes
- Represents the fully baked manifest set to be deployed downstream

[produces] | *https://www.spinnaker.io/guides/user/kubernetes-v2/deploy-helm/produces.png*

Configure Downstream Deployment

Now that the manifest set has been baked by Helm

- Configure a downstream stage (in the same pipeline or in one triggered by this pipeline)
- Deploy the artifact produced by the “Bake (Manifest)” stage

[expected artifact] | *https://www.spinnaker.io/guides/user/kubernetes-v2/deploy-helm/expected-*

artifact.png

Note: Make sure to select "embedded-artifact" as the artifact account for your `base64` manifest set. This is required to translate the manifest set into the format required by the deploy stage.

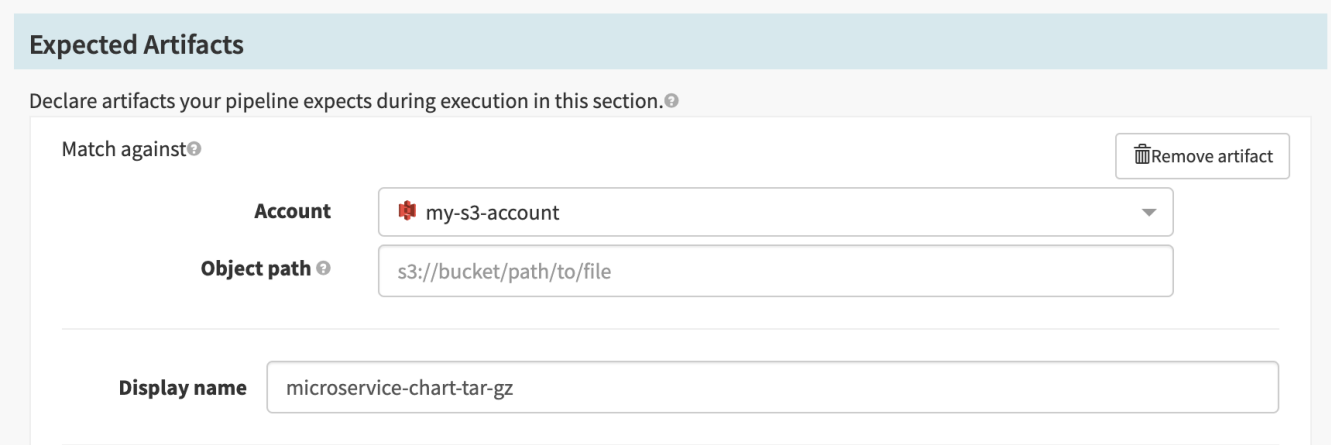
Lab: Deploying Helm Microservices

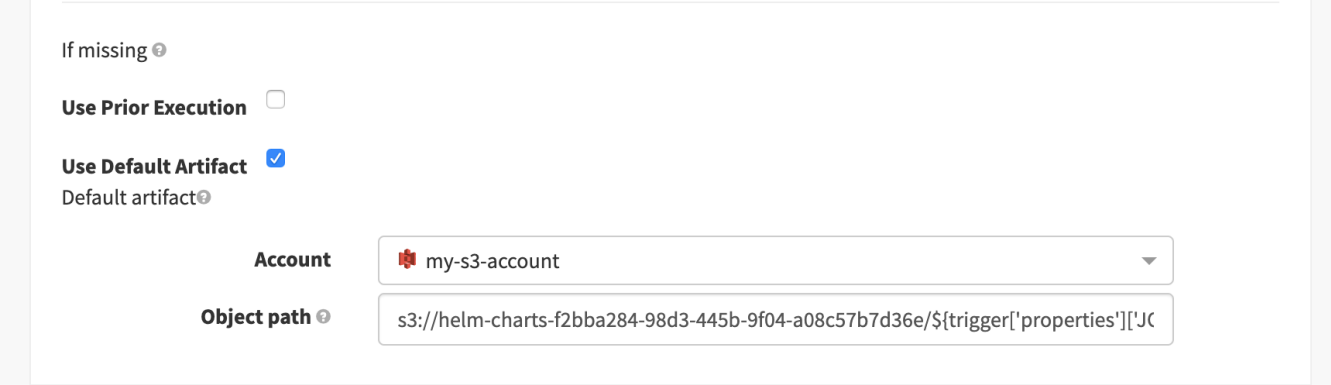
1. Create a new Application, call it "<your-team-name>-baked"
2. Use the same repository
3. Create a new Pipeline called "ToStaging"
4. Disable your first project

Configuration

In the **Configuration** Stage:

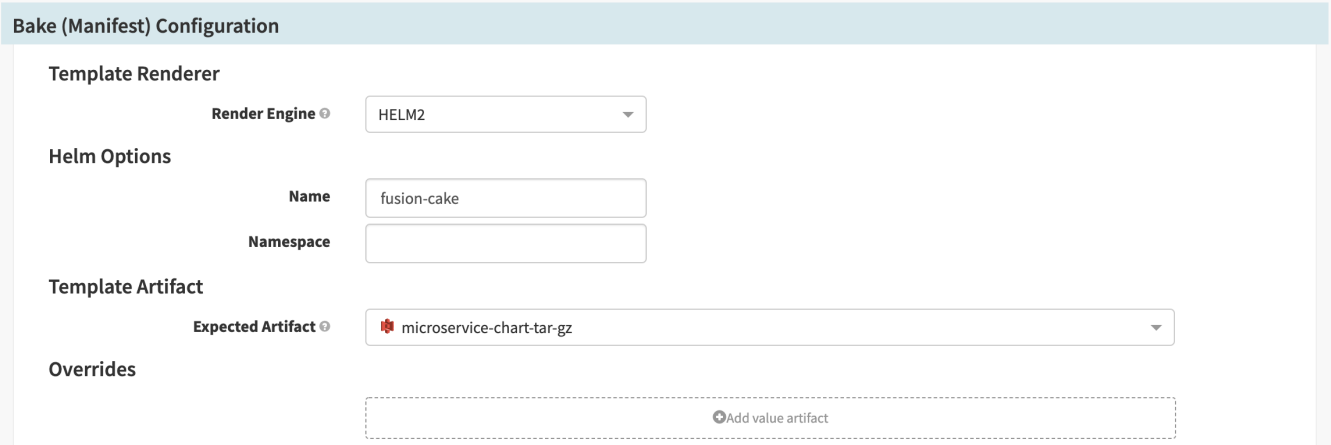
1. In **Expected Artifacts** click on **Add Artifact**
2. Under **Match Against**, select **my-s3-account**
3. Leave **Object Path** blank
4. For **Display Name**, enter `microservice-chart-tar-gz`
5. Check **Use Default Artifact**
6. In the **Account** field for the default artifact select **my-s3-account**
7. In the **Object Path** enter: `s3://helm-charts-f2bba284-98d3-445b-9f04-a08c57b7d36e/${trigger['properties']['JOB_NAME']}/${trigger['properties']['BUILD_ID']}/quarkus-microservice-chart.tar.gz`
8. Save

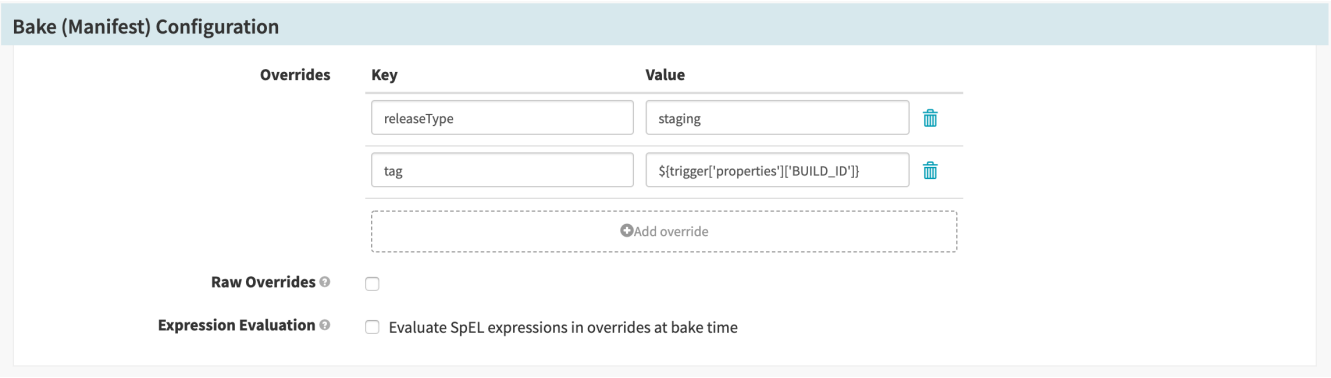




Bake a Manifest

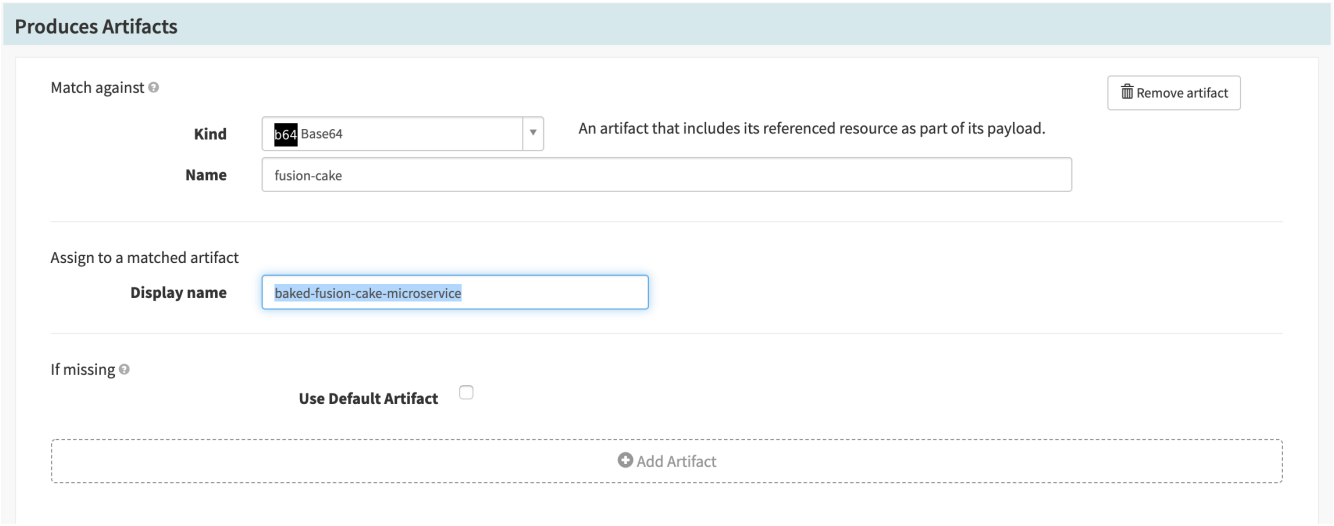
1. While **Configuration** is highlighted, click **Add Stage**
2. Select **Bake Manifest**
3. In your new set of fields, for **Render Engine**, select **HELM2**
4. For the name, enter your team name, do not put the word `microservice`
5. For **Expected Artifact**, select the `microservice-chart-tar-gz` you created in the last step
6. In **Overrides**, enter new **Key** and **Value**
   a. `releaseType` key with a value of `staging`
   b. `tag` key with a value of `${trigger['properties']['BUILD_ID']}`





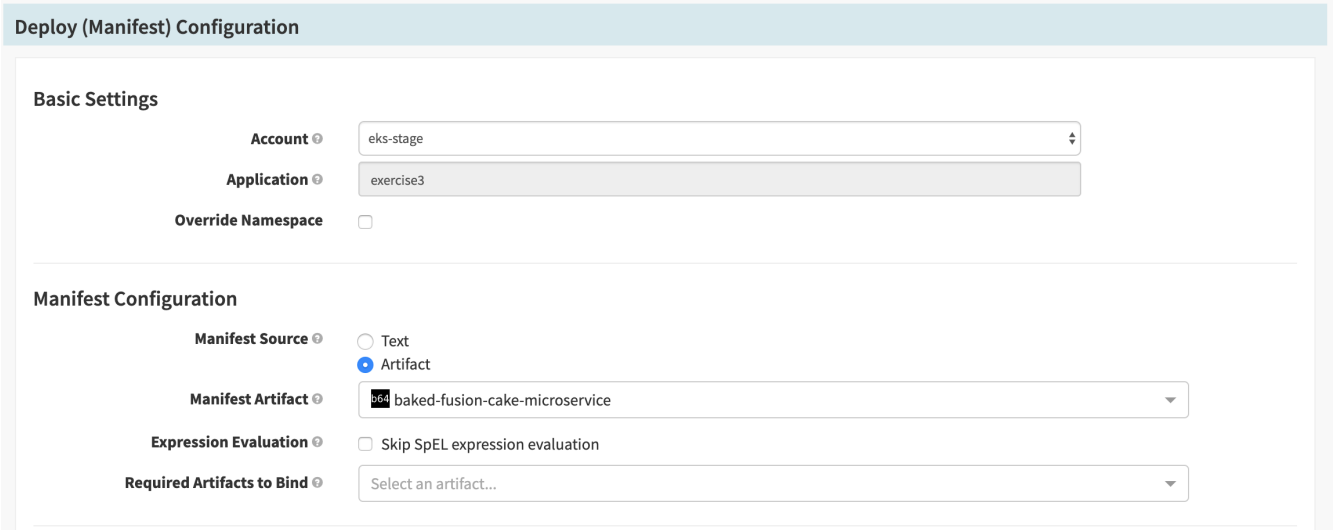
Producing Artifact

1. Staying on the same stage, navigate down to **Produces Artifacts**
2. In **Match Against** select kind as **Base 64**
3. In the **name** field, enter your team name
4. In the **Display Name** in the next section enter `baked-<teamname>-microservice`
5. Save



Create a Deploy Stage

1. While the "Bake Manifest" Phase is selected, click **Add Stage**
2. Select **Deploy Manifest**
3. In **Basic Settings**, and in **Account**, select `eks-stage`, since we will deploy to stage
4. In **Manifest Configuration**, and in **Manifest Source**, select **Artifact**
5. In **Manifest Artifact** select the Base64 object you created in the last step `baked-<teamname>-microservice`
6. Save



Put it all in motion

- Make a change in your repository
- Commit and Push
- Ensure that your pipeline works!

Things to discuss

- Where did **my-s3-account** come from?
- What was that funk that we typed into the **Object Path**?
- What is the significance of to tar file?
  - How is it being stored?
  - Where do we get it?
  - How was it created?
- Base 64 why?

Rollbacks

Automatic Rollbacks

- Having manifests is optimal, IaC
- Rollbacks though are necessary when things go wrong

Viewing the Rollbacks

- When using a Kubernetes `Deployment`, history of rollouts is in the `Clusters` tab of your Application
- Spinnaker exposes “Undo Rollout” functionality in two places:
  - Clusters tab
  - Pipeline stage

[revisions] | *https://www.spinnaker.io/guides/user/kubernetes-v2/automated-rollbacks/revisions.png*

Ad Hoc Rollbacks

- In cases where you see something is immediately wrong and isolated to a resource in the “Clusters” tab
- You can select “Undo Rollout” from the “Actions” dropdown

- This provides an opportunity to select a healthier alternative

[adhoc] | *https://www.spinnaker.io/guides/user/kubernetes-v2/automated-rollbacks/adhoc.png*

Automated Rollbacks

- You can also configure automated rollbacks inside of Spinnaker pipelines.
- These stages are best configured to run when other stages or branches fail, indicating that something has gone wrong in your rollout.
- Number of Revisions Back:
  - Counts how many revisions the current active revision should be rolled back by. If you have the following state:

```
nginx-deployment-2d8178b77 (Revision 5) # active
nginx-deployment-7bdd110f7 (Revision 4)
nginx-deployment-0b13cc8c1 (Revision 1)
```

And roll back by `"1"` revision, `(Revision 4)` will be active again. Roll back by `"2"` revisions and `(Revision 1)` will be active again.

Keep in mind that Kubernetes will implicitly roll-forward the old configuration, creating `(Revision 6)` in both cases.

Parameterized Rollbacks

- Parameterize the target resource to roll back
- Reference anything via pipeline expressions
- Point to something specified using pipeline parameters, for example:
  - Upstream deploy stage
  - Another stage's outputs
  - Using pipeline expressions.

Versioning Artifacts

- Rolling Back will not work unless artifacts are versioned, this includes:
  - Docker images
  - `ConfigMap`
  - Kubernetes Secrets

Source: https://blog.questionable.services/article/kubernetes-deployments-configmap-change/

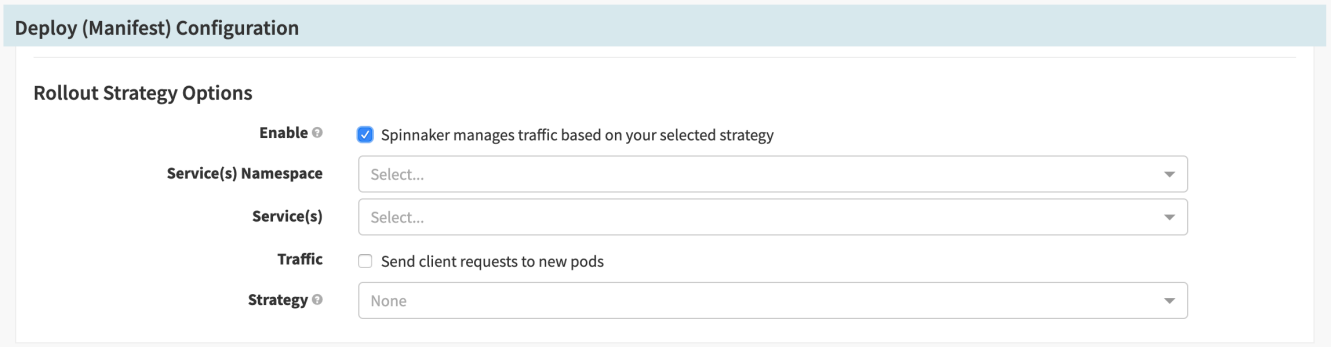
Scaling

Rollout Stratagies

- Deploys your containers as either:
  - Dark Rollouts
  - Highlander
  - Red/Black (also known as Blue/Green)
- Is only valid for Kubernetes `ReplicaSet`

Configuration Options

- Configuration options allow you to associate a workload with one or more services
- Decide whether the workload should receive traffic, and determine how Spinnaker should handle any previous versions of the workload in the same cluster and `namespace`



- Service namespace - Select the namespace containing the service(s) you would like to associate with the workload.
- Service(s) - Select one or more services you would like to associate with the workload. Spinnaker will add a traffic.spinnaker.io/load-balancers annotation listing the selected services as described here.
- Traffic - Check this box if you would like the workload to begin receiving traffic from the selected services as soon as it is ready. If you do not check this box, you can add a subsequent Enable (Manifest) stage to begin sending traffic to the workload.
- Strategy - Select a strategy if you would like Spinnaker to handle previous versions of the workload currently deployed to the same cluster and namespace. Select None if you do not want Spinnaker to take any action regarding existing workloads.

Strategies

Dark Rollouts

- Use a dark rollout to deploy a new version of your application alongside the existing version(s)
- Does not immediately route any traffic to the new version.
- Optionally, add subsequent Enable (Manifest) and Disable (Manifest) stage to begin sending traffic to the new version and stop sending traffic to the old version(s).

[dark] | *https://www.spinnaker.io/guides/user/kubernetes-v2/rollout-strategies/dark.png*

Highlander

> "There can be only one" — Highlander (1986)

- Deploy a new version of your application alongside the existing version(s)
- Send client traffic to the new version
- Then disable and destroy existing versions in the cluster

[highlander] | *https://www.spinnaker.io/guides/user/kubernetes-v2/rollout-strategies/highlander.png*

Red/Black (Blue/Green)

- Red/black rollout to deploy a new version of your application alongside the existing version(s)
- Send client traffic to the new version, and then disable existing versions in the cluster
- Optionally, add subsequent Destroy (Manifest) stages to clean up any unwanted workloads in the cluster
- Alternately, easily roll back to a previous version by:
  - Configuring an **Enable (Manifest)** stage
  - Or using an ad-hoc Enable operation from the Clusters tab.

[redblack] | *https://www.spinnaker.io/guides/user/kubernetes-v2/rollout-strategies/redblack.png*

Restrict Execution

Execution Windows

- Execution windows allow you to restrict the times of the day or week when deployments can happen.
- By using execution windows, you can ensure that deployments don’t interfere with times where your service is at peak demand.
- You can also use execution windows to make sure that there is always someone in the office ready to manually intervene or rollback your pipeline.

Manual Judgement

Manual Judgments serve as a gate for your pipeline You can add manual judgement stages to interrupt the pipeline execution to run a manual check This is great for teams that have a manual or out-of-band QA process

1. Click on Configuration and select **Add New Stage**
2. Enter “Please approve this pipeline” in the instructions

[manual1] | *https://www.spinnaker.io/guides/tutorials/codelabs/safe-*

deployments/images/manual1.png

Notifications with Manual Judgment

1. Click on **Add Notification Preference**
2. In the popup dialog:
    a. Select type Email
    b. Enter your email address
    c. Select to notify when **This stage is awaiting judgment**.
    d. Click **Update**

[manual2] | *https://www.spinnaker.io/guides/tutorials/codelabs/safe-*

deployments/images/manual2.png

Waiting for Approval

[manual6] | *https://www.spinnaker.io/guides/tutorials/codelabs/safe-*

deployments/images/manual6.png

Changing Pipeline on Selected Judgment

You can enter in your own choices and use SpEL to respond to the answers

[manual7] | *https://www.spinnaker.io/guides/tutorials/codelabs/safe-*

deployments/images/manual7.png

Once applied you can create a subsequent stage that will conditionally trigger on your choice

[manual8] | *https://www.spinnaker.io/guides/tutorials/codelabs/safe-*

deployments/images/manual8.png

Canary Deployment

What is it?

- Canary is a deployment process in which a change is partially rolled out then evaluated against the current deployment (baseline)
- Ensure that the new deployment is operating at least as well as the old.
- This evaluation is done using key metrics that are chosen when the canary is configured.
- Microservices would need to expose some of their metrics
- Canaries are usually run against deployments containing changes to code, but they can also be used for operational changes, including changes to configuration.

Prerequisites

- Have metrics to evaluate
  - Provide metrics in your microservice application
  - Either deploy metrics or have it scraped
  - Support is built in for:
    - StackDriver
    - Datadog
    - Prometheus
    - SignalFx
    - NewRelic

Process

- Enable Canary
- Create one or more canary configurations
- Add one or more canary stages

Enable Canary

- Use Halyard
- Setup *one* Metrics Service
- Setup *one* a Storage Service

Setting up Canary Analysis

```
hal config canary enable
```

Setting up the Scope

- Each configuration is visible to all canary pipeline stages
- It can be configured so that it is only visible to the application where it was created

```
hal config canary edit --show-all-configs-enabled false
```

Setting up the canary judge

- The current default judge is `NetflixACAJudge-v1.0`
- A judge accesses the quality of your deployment against the baseline
- Each is compared to determine any degradation
- Other judges are available and are pluggable

```
hal config canary edit --default-judge JUDGE
```

Identify your metrics provider

```
hal config canary edit --default-metrics-store STORE
```

`STORE` can be:

- atlas
- datadog
- stackdriver
- prometheus
- newrelic

Provide a default metrics account

- Add the account name to use for your metrics provider
- Default can be overridden

```
hal config canary edit --default-metrics-account ACCOUNT
```

Provide the default storage account

- Add the account name to use for your storage provider
- Default can be overridden

```
hal config canary edit --default-storage-account ACCOUNT
```

Setting up Prometheus

```
hal config canary prometheus enable
```

Manage or View Prometheus Account for Canary

Add a Halyard/Spinnaker account to Prometheus

```
hal config canary prometheus account add ACCOUNT --base-url
```

Edit a Halyard/Spinnaker account to Prometheus

```
hal config canary prometheus account edit ACCOUNT --base-url
```

Operations for `delete` and `list` are also available

Create a Canary Configuration

- Canary configuration is done per *application*
- Each Application will have one or more configurations
- Stages are defined separately

What you will be adding into a Canary Configuration

You will be adding the following to your canary configuration:

- A name by which a canary stage can choose this configuration
- The specific metrics to evaluate, and a logical grouping of those metrics
- Default scoring thresholds (which can be overridden in the canary stage)
- Optionally, one or more filter templates

Enabling Canary at the User Level

- In the Application config, activate the Canary option
- Do this separately for all applications that will use automated canary analysis

[enable canary] | *https://www.spinnaker.io/guides/user/canary/config/enable_canary.png*

Create your Canary Configuration

- Configurations you create within an application are available to all pipelines in that application
- Unless it is setup that all configurations are available to all applications

Steps for Canary Configuration Support

Selecting the Canary Configuration

- Hover over the **Delivery** Tab, select Canary Configs

[delivery menu canary] |

https://www.spinnaker.io/guides/user/canary/config/delivery_menu_canary.png

Select **Add Configuration**

Provide a name and a description. The name is displayed when you show the canary stage on pipeline

Select your telemetry provider

Create Metrics Group

1. Create groups to determine weights
2. Click Group to create each group you'll use. Select the group and click the edit icon
3. An example, would be "cpu" group to add a set of cpu metrics

Create Metrics

1. In the **Metrics** section, add **Metric**
2. Select the group to add this metric
3. Give the metric a name
4. Specify whether this metric fails when the value deviates too high or too low, or both compared to the baseline
5. Optionally, choose a filter template

```
resource.type = "gce_instance" AND
resource.labels.zone = starts_with("${zone}")
```

1. Identify the metric you wish to include in this configuration

[metric type list cpu] | *https://www.spinnaker.io/guides/user/canary/config/metric_type_list_cpu.png*

1. Optionally, if your providers, supports aggregation, you can click **Group by** and enter the metric metadata For example, when you create a metric you can group its time series by resource or metric label. You can group a time series by zone, for example (`resource.zone`)

When you create a canary configuration, you create metric groups, and scoring thresholds and weights are applied to groups (rather than to specific metrics). But the grouping described in this step is for aggregating metrics before they're returned to Kayenta.

1. Click OK to save metric

Add Filter Template

If you are using *StackDriver* or *Prometheus* you can add filter templates then assign it to a metric

1. Click **Add Template**
2. Provide a Name - This is the name by which you can select it when configuring the specific metric.
3. In the Template field, enter an expression using the *FreeMarker* template language[https://freemarker.apache.org/docs/dgui_quickstart_template.html]
4. The expression is expanded using the variable bindings specified via the Extended Params in any canary stage that uses this configuration.
5. These variable bindings are also implicitly available: `project`, `resourceType`, `scope`, `location`

Editing a Configuration

- In **Delivery** tab select **Canary Configs**
- Select the configuration you would like to edit

Adding a Canary Stage

About Canary Stages

- Canary analysis can be performed over data points collected beginning from the moment of execution and into the future, or it can be performed over a specified time interval.

Real-time Analysis

A real-time analysis means that the canary analysis is performed over a time interval beginning at the moment of execution. The analysis happens for a specified time period, beginning when the stage executes (or after a specified Delay). For Real Time, also specify the number of hours to run (Lifetime).

Retrospective Analysis

In a retrospective analysis the canary analysis is performed over an explicitly specified time interval (likely in the past). Analysis occurs over some specified period. Typically, this is done for a time period in the past, against a baseline deployment and a canary deployment which have already been running before this canary analysis stage starts. Note that this analysis might analyze data for resources which no longer exist, for which there are still published time series.

Metric Scope

Metric scope defines:

- Where, when, and on what the canary analysis occurs.
- Specific baseline and canary server groups
- Start and end times and interval
- Cloud resource on which the baseline and canary are running

Defining the Canary Stage

- In your pipeline click on **Add Stage**
- For type, select **Canary**
- Give it the stage name and depends on field
- In the analysis type select **Real Time** or **Retrospective**
- Choose a configuration name for the configuration you created before adding this stage
- Set a delay: How many minutes to wait until warmed up
- Set an interval: This is how frequently (in minutes) to capture and score the metrics.
- Lookback Type: Growing or Slowing
  - In a growing analysis, a judgment is taken every [interval] minutes, but each judgment goes all the way back to the beginning of the Lifetime.
  - A sliding Lookback also makes a judgment every [interval], but each judgment only looks at the data from the most recent lookback duration. It would not be unusual for the Interval and the look-back duration to be the same, but they don't have to be.

[stage config analysis] | *https://www.spinnaker.io/guides/user/canary/stage/stage_config_analysis.png*

- Describe the metrics scope, Metric Scope has a magic wand next to the name that automatically resolve to available resources
- Those resources are a starting point, which you can edit to match the specific resources you need.
  - Baseline - The server group to treat as the "control" in the canary analysis—that is, the deployment against which to compare the canary deployment.
  - Baseline Region - The region in which the baseline server group is deployed.
  - Canary - The server group to treat as the experiment in the analysis.
  - Canary Region - The region in which that canary server group is deployed.
  - Step - The interval, in seconds, for the metric time series.
  - Start Time and End Time (for retrospective) For a retrospective analysis, the specific time span over which to conduct the analysis.
  - Extended Params - Add any additional parameters, which are specific to the metric sources and which can be used to refine the scope of the analysis. These parameters can provide variable bindings for use in the expansion of custom filter templates specified in the canary config.

[metric scope] | *https://www.spinnaker.io/guides/user/canary/stage/metric_scope.png*

- Adjust Scoring Thresholds is needed - The thresholds are pre-populated based on those configured in the main canary config, but you can override them here.
- Specify the account that you are using for metrics and storage

Judgement

Judgement goes through two phases:

- Data collection
  - Collect data from both baseline and canary deployments
  - Stored in a timeseries database
  - Include a set of tags or annotations that identify deployment
- Judgement
  - Compare the metrics from the data collection
  - Renders a pass or fail
  - Can be configured with a "marginal" decision

Judgement Phases

Step 1: Data Validation

- Ensures that there is data
- If no data is found in either baseline or canary, it moves onto the next metric
- Judge does not fail if there is no data, since some data it’s ok to have no data

Step 2: Data Cleaning

- Handling Missing Values
- Impute values for missing values
- Optionally, outliers can be removed

Step 3: Metric Comparison

- This is the step that compares the canary and baseline data for each included metric.
- Classification indicating there is a significant difference between baseline and canary
- Each metric is either *Pass*, *High*, or *Low*
- Uses nonparametric statistical test to check for a significant difference between the canary and baseline metrics

Step 4: Check the score

- If 9 out of 10 is "Pass" it is classified as pass if specified in the root configuration
- Default judge is biased for simplicity- easy to interpret and understand

[metric classifications] |

https://www.spinnaker.io/guides/user/canary/judge/metric_classifications.png

Prometheus

- Prometheus is just one of the available metrics packages that analyzes data from your microservice
- There are various metrics packages to include in your microservice in a variety of languages
- The metrics are "scraped" by prometheus and aggregated into reports

[architecture] | *https://prometheus.io/assets/architecture.png*

Prometheus setup in Kubernetes

- There are various utilities to get Prometheus up and running, and documentation is terrible
- One of the easier ones, despite horrible documentation, is the Prometheus Operator on Helm(https://coreos.com/operators/prometheus/docs/latest/user-guides/getting-started.html)

[1*6KI8wlyWwLwPYgt SP1CCA] |

*https://miro.medium.com/max/3592/1*6KI8wlyWwLwPYgt_SP1CCA.png*

Setup for Prometheus

```
helm install <name> stable/prometheus-operator
```

Ensure that you are in the right context. For canary testing you would need to be in `staging`

Create a Service Monitor

- By default, Prometheus is install in the `default` namespace
- `namespaceSelector` will look for metrics in other `namespaces`
- This `ServiceMonitor` needs to be deployed in the `default` namespace

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    release: lasertag
  name: teama-microservice-service-monitor
spec:
  endpoints:
    - interval: 30s
      path: /metrics
      port: web
  namespaceSelector:
    matchNames:
      - microservices
  selector:
    matchLabels:
      app: teama-microservice-service
```

Create a `NodePort` or `LoadBalancer`

Once established, deploy a `NodePort` or `LoadBalancer`

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus
spec:
  type: (NodePort | LoadBalancer)
  ports:
  - name: web
    nodePort: 30900
    port: 9090
    protocol: TCP
    targetPort: web
  selector:
    app: prometheus
```

Choose `NodePort` or `LoadBalancer`, `(NodePort | LoadBalancer)` is not a valid option

Lab: Advanced Features

Discovering Prometheus

1. Visit http://aba3fd2983dac11eab3c20a9ecbc5f37-1991264767.us-west-2.elb.amazonaws.com:9090/graph
2. In the dropdown that contains states `-insert metric at cursor` select `application_org_acme_metrics_PrimeNumberChecker_checksTimer_mean_seconds`
3. Go to the source code and locate the *src/main/java/org/acme/metrics/PrimeNumberChecker.java* file

Notice the annotations:

```
@GET
@Path("/{number}")
@Produces("text/plain")
@Counted(name = "performedChecks",
         description = "How many primality checks have been performed.")
@Timed(name = "checksTimer",
       description = "A measure of how long it takes to perform the
primality test.",
       unit = MetricUnits.MILLISECONDS)
```

View the metrics of your microservice

1. In the URL of your application, go to the `metrics` folder of your microservice

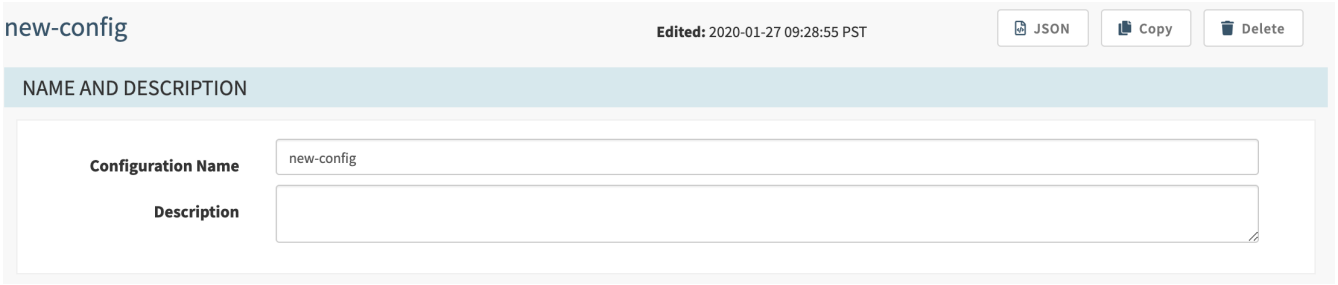
2. For example if your team name was teama, then go to: `http://<url>/teama/metrics`
3. Notice the results

Create an Application

1. Create a new Application, call it "<your-team-name>-canary"
2. Use the same repository
3. Create a new Pipeline called "ToStaging"
4. Disable your second project

Create Canary Configs

1. Click on **Delivery**, and **Canary Configs**
2. On the left, click on **Add Configuration**
3. For the **Configuration Name**, call it `new-config`



Add a Metric

1. In the **Metrics** section, select **Group 1**, and click on **Add Metric**
2. In your dialog box,

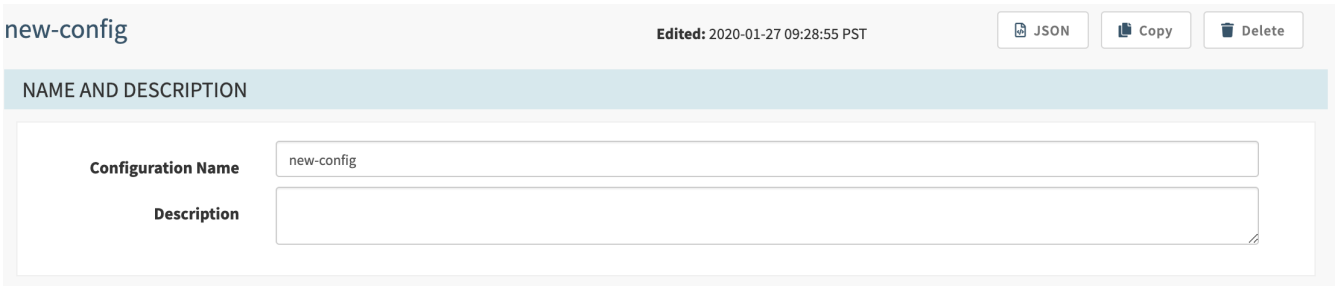
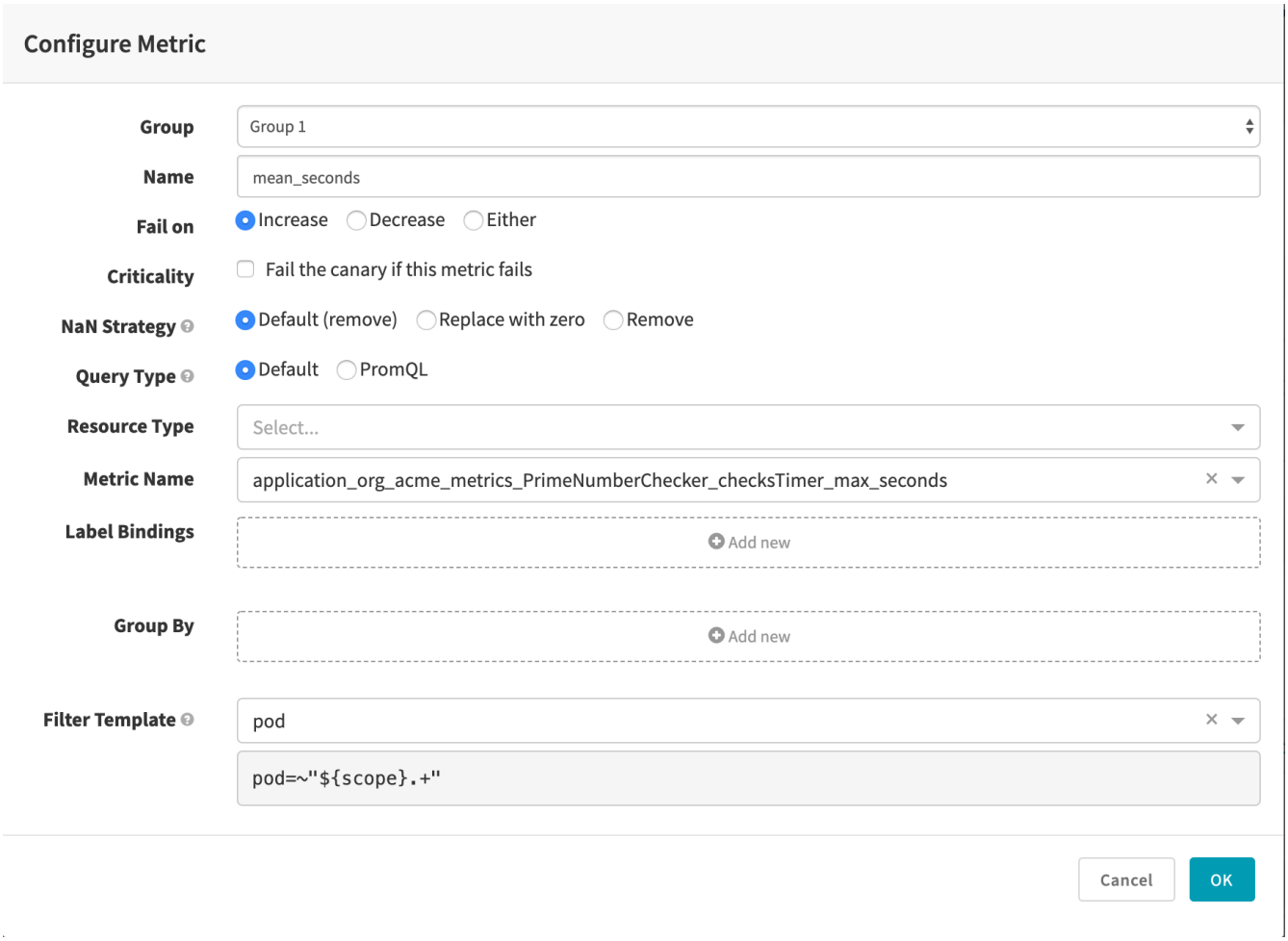


Figure 9. Initial Canary Config

Figure 10. Adding a Metric

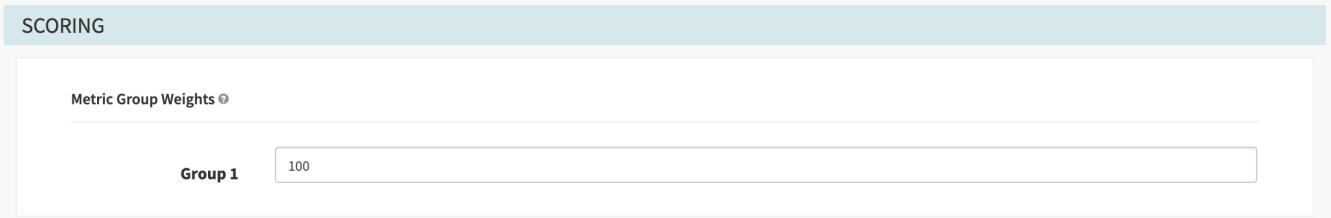
Configuration of the Metric

- In the dialog box, in the **Name** field, enter `mean_seconds`
- In the **Fail On** field, select **Increase**
- For the **NaN Strategy**, leave the **Default (remove)**
- For the **Query** type, select **Default**
- For **Metric Name**, select `application_org_acme_metrics_PrimeNumberChecker_checksTimer_max_seconds`
- In **Filter Template**, select **Create New**
- For the new **Filter Template**
  - Under the **Name** field enter `pod`
  - For the **Template** enter `pod=~"${scope}.+"`



Scoring

For Scoring, since we have one group, we will make that value `100`



Creating a Pipeline

- Create a new pipeline called "ToStaging"
- Create one Trigger, for Jenkins
  - For **Type** select "Jenkins"
  - For **Controller** select "jenkins-master"
  - For **Job** select your job from Jenkins
  - In **Property File**, enter `spinnaker.properties`
  - Ensure that the Trigger is enabled

Create Deploy Manifest Stage

- Building off of **Configuration**, create a `Deploy (Manifest)` stage
- For **Stage Name**, enter `Deploy Canary`
- In **Basic Settings**, for **Account**, enter `eks-stage`
- In the **Manifest Source**
  - Select **Text**
  - Enter the following manifest, replacing `your-team` with your team name

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: your-team-microservice-canary
    release: canary
    source: demo
    stack: frontend
  name: your-team-microservice-canary
  namespace: microservices
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      run: your-team-microservice-canary
  template:
    metadata:
      labels:
        app: your-team-microservice-canary
        release: canary
        run: your-team-microservice-canary
        source: demo
    spec:
      containers:
        - image: >-
            219099013464.dkr.ecr.us-west-2.amazonaws.com/your-team-
microservice:${trigger['properties']['BUILD_ID']}
          name: your-team-microservice-canary
          ports:
            - containerPort: 8080
              name: web
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: your-team-microservice-service-canary
  name: your-team-microservice-service-canary
  namespace: microservices
spec:
  ports:
    - name: web
      port: 30201
      protocol: TCP
      targetPort: 8080
  selector:
    app: your-team-microservice-canary
    source: demo
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    release: lasertag
  name: your-team-microservice-service-monitor-canary
spec:
  endpoints:
    - interval: 30s
      path: /metrics
      port: web
  namespaceSelector:
    matchNames:
      - microservices
  selector:
    matchLabels:
```

```
      app: your-team-microservice-service-canary
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
  name: your-team-ingress-canary
  namespace: microservices
spec:
  rules:
    - http:
        paths:
          - backend:
              serviceName: your-team-microservice-service-canary
              servicePort: 30201
            path: /your-team/canary(/|$)(.*)
```

Find Baseline

- While Selecting **Deploy Canary**, click on **Add Stage**
- Select `Find Artifacts From Resource (Manifest)`
- For **Stage Name** enter `FindBaseline`
- In the **Find Artifacts from Resource (Manifest) Configuration**
  - In **Account** enter `eks-prod`
  - In **Namespace** enter `microservices`
  - In **Kind** enter `deployment`
  - For Selector select "Choose a static target"
  - For the **Name**, enter `<team-name>-microservice`

Deploy Baseline

- While Selecting **FindBaseline**, click on **Add Stage**
- Select `Deploy (Manifest)`

- For **Stage Name** enter `DeployBaseline`
- In **Basic Settings**, for **Account**, enter `eks-stage`
- In the **Manifest Source**
  - Select **Text**
  - Enter the following manifest, replacing `your-team` with your team name

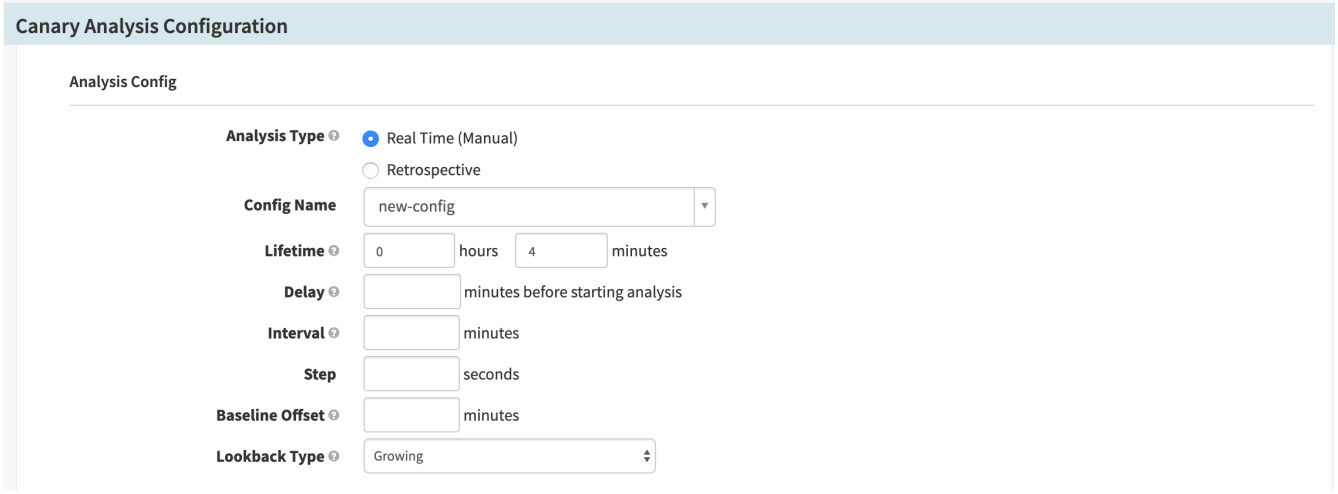
```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: your-team-microservice-baseline
    release: baseline
    source: demo
    stack: frontend
  name: your-team-microservice-baseline
  namespace: microservices
spec:
  replicas: 3
  selector:
    matchLabels:
      run: your-team-microservice-baseline
  template:
    metadata:
      labels:
        app: your-team-microservice-baseline
        release: baseline
        run: your-team-microservice-baseline
        source: demo
    spec:
      containers:
        - image: '${#stage(''FindBaseline
'').context["artifacts"][0]["reference"]}'
          name: your-team-microservice-baseline
          ports:
            - containerPort: 8080
              name: web
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: your-team-microservice-service-baseline
  name: your-team-microservice-service-baseline
  namespace: microservices
spec:
  ports:
    - name: web
      port: 30201
      protocol: TCP
```

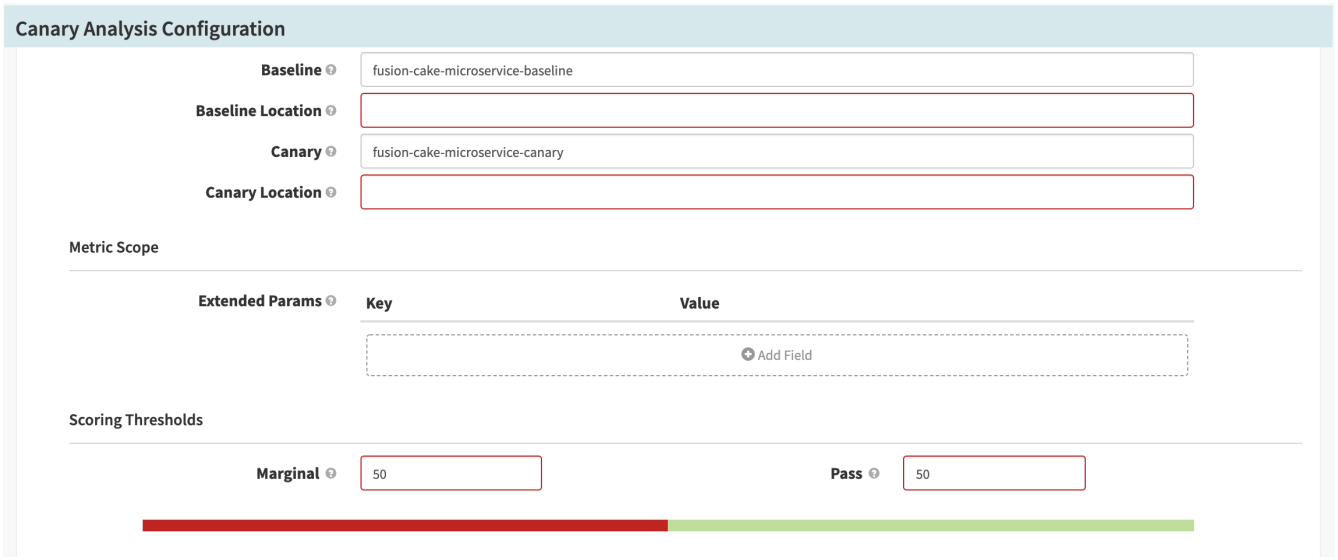
```
      targetPort: 8080
  selector:
    app: your-team-microservice-baseline
    source: demo
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    release: lasertag
  name: your-team-microservice-service-monitor-baseline
spec:
  endpoints:
    - interval: 30s
      path: /metrics
      port: web
  namespaceSelector:
    matchNames:
      - microservices
  selector:
    matchLabels:
      app: your-team-microservice-service-baseline
  targetLabels:
    - release app source
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
  name: your-team-ingress-baseline
  namespace: microservices
spec:
  rules:
    - http:
        paths:
          - backend:
              serviceName: your-team-microservice-service-baseline
              servicePort: 30201
            path: /your-team/baseline(/|$)(.*)
```

Perform a Canary Analysis

- While Selecting **DeployBaseline**, click on **Add Stage**
- Select `Canary Analysis`
- For **Stage Name** enter or leave it as `Canary Analysis`
- In the **Canary Analysis Configuration** and **Analysis Config** section:
  - In **Analysis Type**, select **Real Time (Manual)**

- In **Config Name**, select the config you just created `new-config`
- In **Lifetime**, select 4 minutes





Delete tokens from Github

If you were the member of your team that was in charge of the github account, be sure to delete your token

Things to discuss

- How was the information scraped?
- How did spinnaker talk with prometheus
- Is this too much work? What's the payoff?

Conclusion

- Dev-ops is hard, requires a lot of knowledge in:

- Command line programming
- Kubernetes
- Prometheus
- Multiple Languages

You will need to be creative