

# SPINNAKER

DevelopIntelligence

# ABOUT DEVELOPINTELLIGENCE

## Our purpose...

We help organizations learn and adopt new technologies.



## ...Impacts you daily.

When you talk on the phone, watch a movie, connect with friends on social media, drive a car, fly on a plane, pay with a credit card, shop online, and order a latte with your mobile app, you are interacting with technology developed by one of our customers.

In 2018 alone...



# TECHNOLOGIES WE TEACH

Ruby  Jenkins     

    **AND MANY OTHER TRENDING TECHNOLOGIES**

# OUR PRACTITIONERS





# WELCOME



## Mark Your Attendance:

- Go to the Course Calendar Invite
- Click on the Course Event Link
- Press Check-In

# VIRTUAL TRAINING



What we want



...what we've got

## VIRTUAL TRAINING EXPECTATIONS FOR ME

I pledge to:

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so
- Use an on-screen timer for breaks so you know when to be back

## **VIRTUAL TRAINING EXPECTATIONS FOR YOU**

- Arrive and Return on Time
- Mute unless speaking
- Use chat or ask questions verbally

# AGENDA

# AGENDA

Our *mandatory* goals for this class:

- Become familiar with Spinnaker and it's role
- Understand the UI of Spinnaker
- Understand an Application, Pipeline, and Stage
- Purpose of Spinnaker
- How to get Spinnaker within Salesforce
- Helm and Baking
- Spinnaker Expression Language

## SOME OTHER ITEMS YOU'LL PICK UP

- Canary
- Understanding of Kubernetes
- How it works
- Debugging Tips

# **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
- 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?

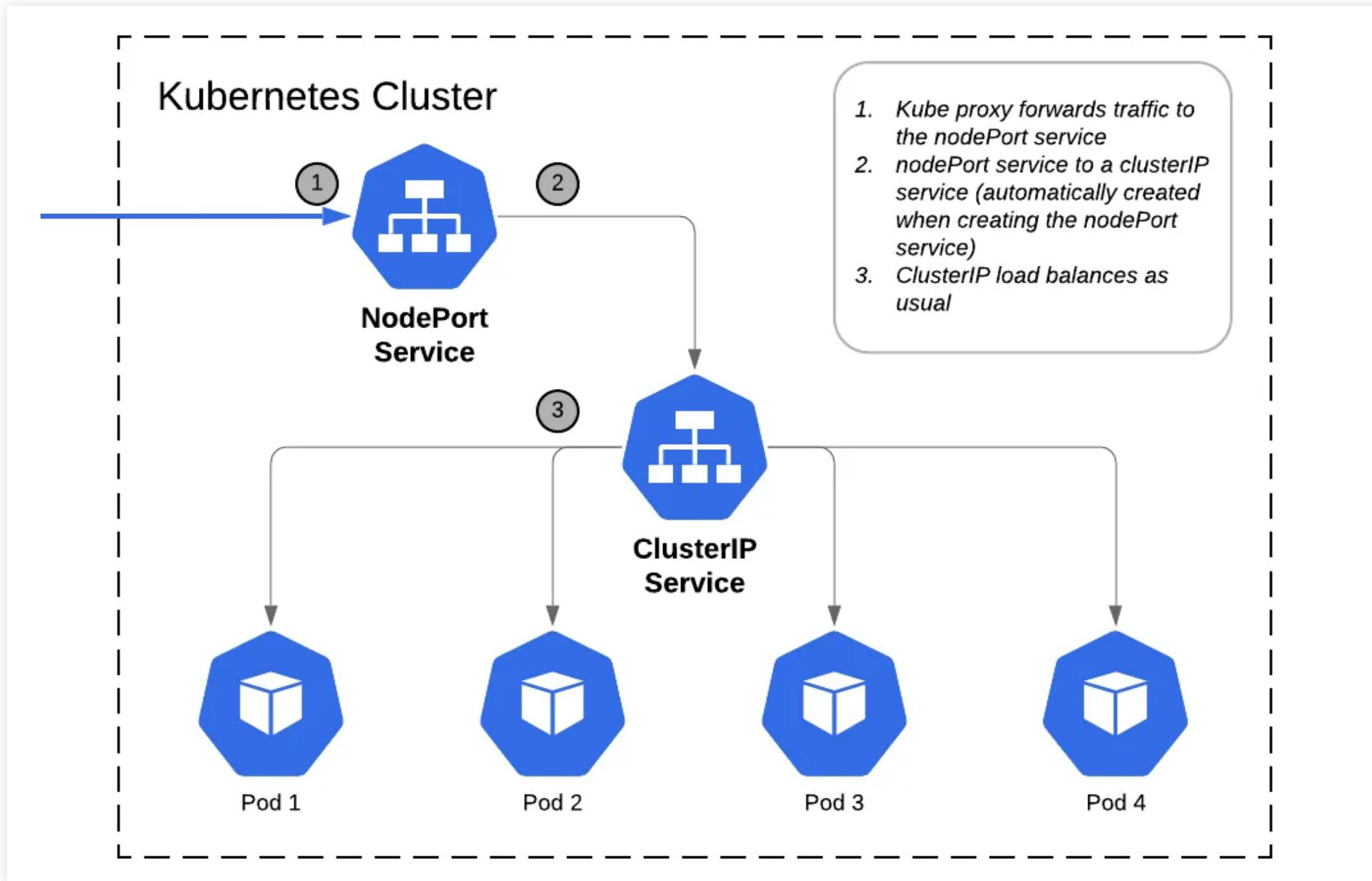
- Automation, Automation, Automation, Automation
- Humans are slow, why not let computers do the work?
- Problems no longer are debt: If anything fails it must be fixed immediately
- Bugs, along with testing, are squashed oftentimes permanently!
- Lessens the fear of "integration hell"
- Lessens the fear of production deployment!

## FUTURE IS HERE

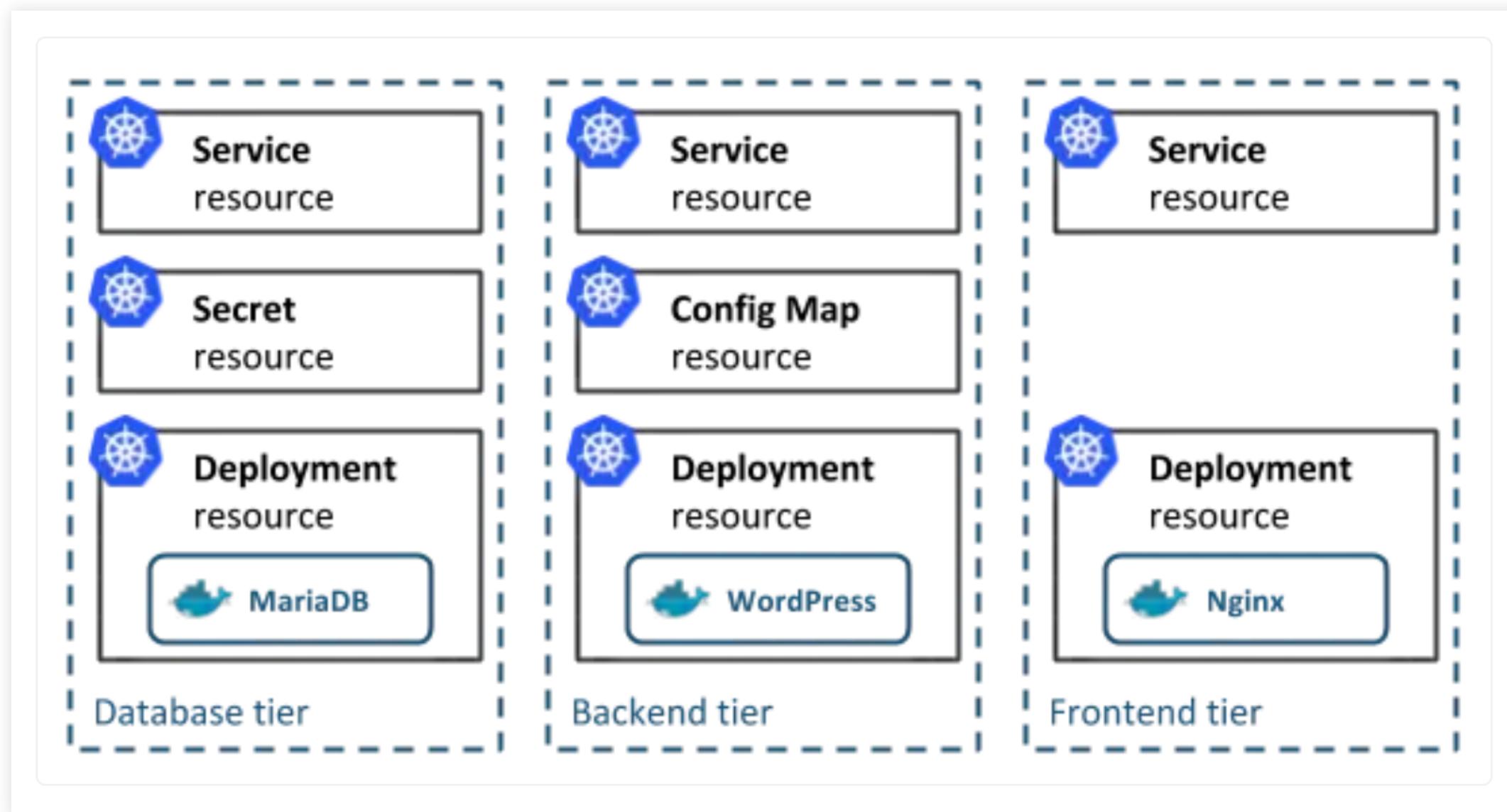
- Spinnaker, Jenkins, and Kubernetes allows for IaC (Infrastructure as Code)
- Continuous Integration is committed as code, using `Jenkinsfile`
- Kuberenetes `replicaSet`, `deployment`, `ingress`, etc are all `yaml files`
- Spinnaker can use these files to deploy fleet of systems

# KUBERNETES

# KUBERNETES NODE PORT AND CLUSTER IP



# 3-TIER ARCHITECTURE



# **REVIEW OF RESOURCES**

# POD

- Group of one or more containers (such as Docker containers)
- Shared storage/network
- Specification for how to run the containers

# SERVICE

- Service is an abstraction which defines a logical set of Pods and a policy by which to access them
- Sometimes this pattern is called a micro-service).
- The set of Pods targeted by a Service is usually determined by a selector

## DEPLOYMENT

- Provides declarative updates for Pods and ReplicaSets.
- Describe a desired state in a Deployment
- Deployment Controller changes the actual state to the desired state at a controlled rate.
- You can define Deployments to create new ReplicaSets,
- Or remove existing Deployments and adopt all their resources with new Deployment

# CONFIGMAP

- Bind to your Pods' containers and system components at runtime
  - Configuration files
  - Command-line arguments
  - Environment variables
  - Port numbers
  - Other configuration artifacts
- Enable you to separate your configurations from your Pods and components, making workloads portable.

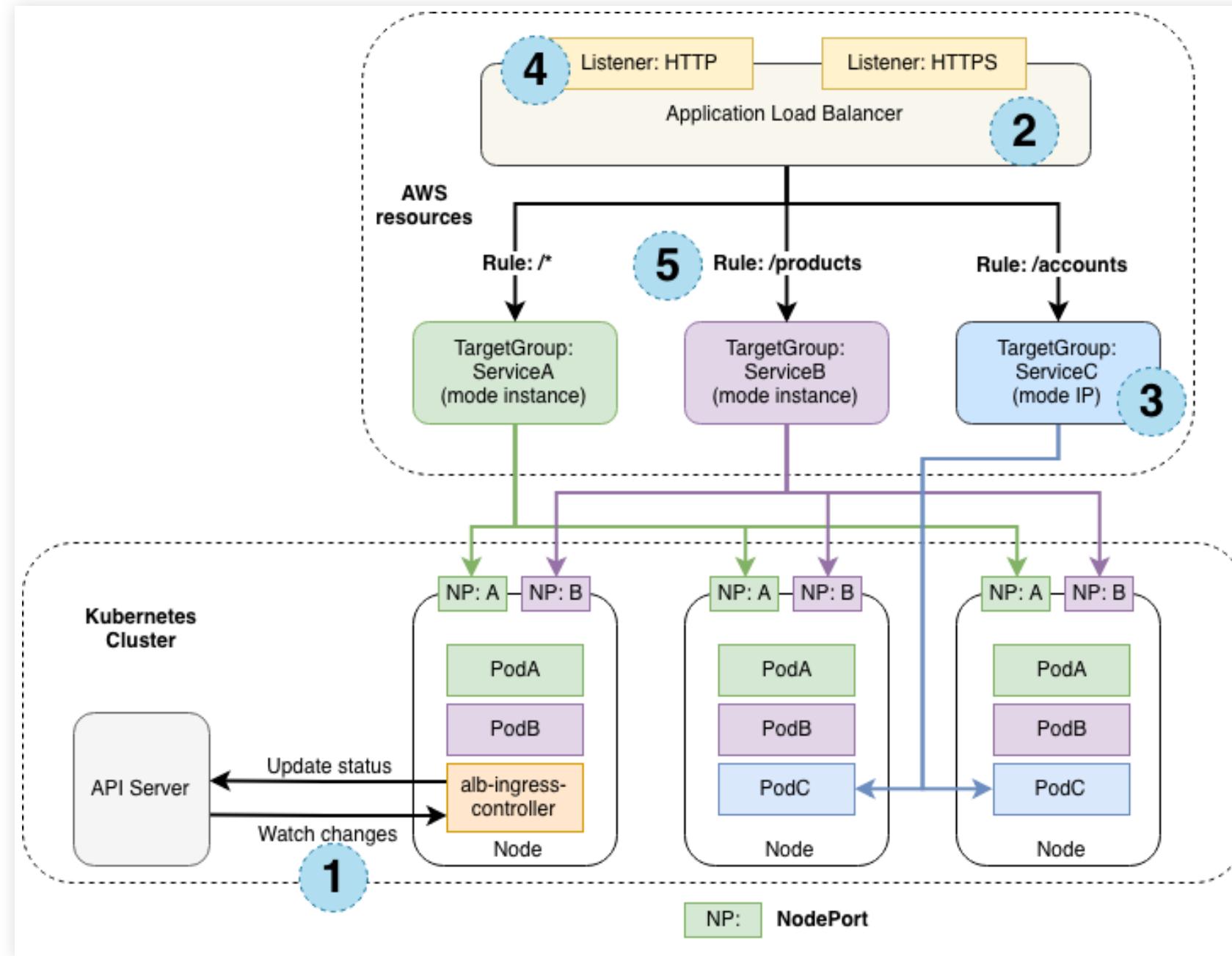
# SERVICEACCOUNT

- Service accounts allow pods to communicate to the Kubernetes API.
- They are bound to specific namespaces, and created automatically by the API server or manually through API calls.
- Service accounts are tied to a set of credentials

# INGRESS

- Object defines rules for routing HTTP(S) traffic to applications running in a cluster.
- An Ingress object is associated with one or more Service objects
- Each Service is associated with a set of Pods

# KUBERNETES AND INGRESS



# 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

# 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

## INSIDE THE APPLICATION

- 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*

The screenshot shows the Spinnaker interface for managing clusters. The top navigation bar has three tabs: CLUSTERS (selected), LOAD BALANCERS, and FIREWALLS. Below the tabs, there are buttons for 'Edit multiple', 'Show Instances' (with checkboxes for 'Instances' and 'with details'), and a 'Create Server Group' button. A cluster named 'EKS-PROD' is selected. Underneath, a specific deployment named 'deployment mild-temper-microservice' is highlighted. This deployment has three instances, indicated by a green triangle icon and the text '3 ▲ : 100%'. Below this, under the 'MICROSERVICES' section, the same deployment name is listed again with three instances at 100% health. There is also a reference to a Docker image: 'V001: 219099013464.dkr.ecr.us-west-2.amazonaws.com/mild-temper-microservice:4'.

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

## **SERVER GROUP**

- Collection of Instances
- With Basic Configuration Settings
  - Number of Instances
  - Autoscaling Policies
  - Their Metadata

# DEPLOYMENT HIGHLIGHTED

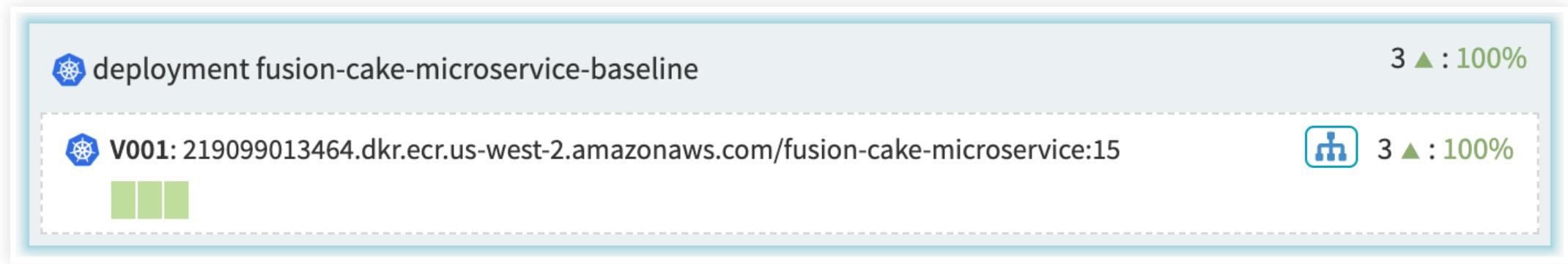


Figure 1. The Deployment is highlighted

## REPLICA SET HIGHLIGHTED

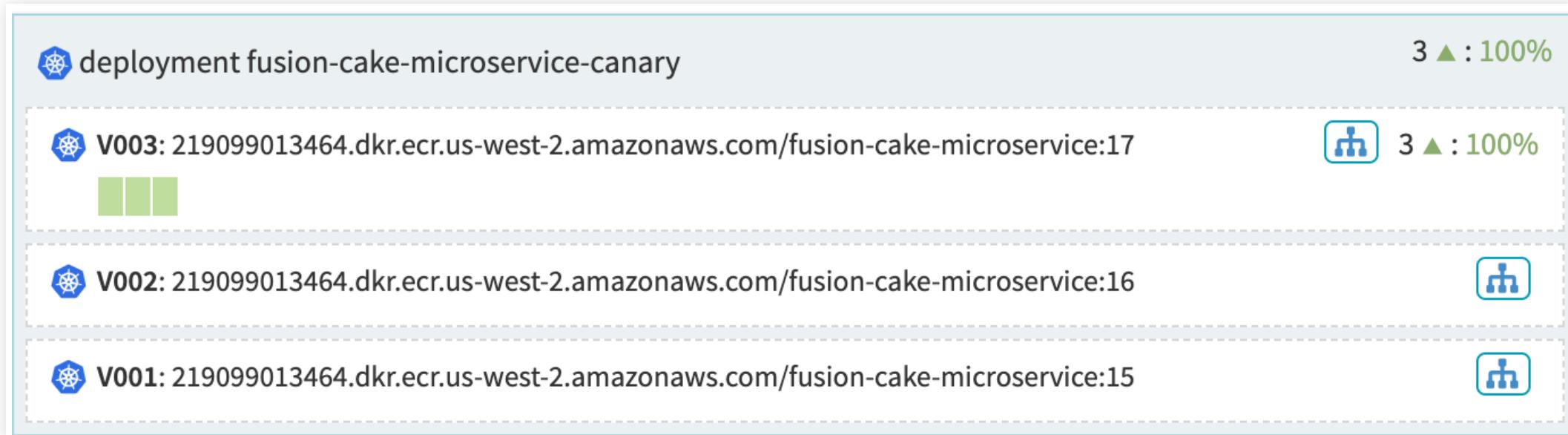


Figure 2. The Replica Set is highlighted

## REPLICA SET HISTORY

- With deployments history of the deployments are maintained
- The following shows a history
- Deployments can be:
  - *rolled back*
  - *deleted*
  - *restarted*
  - *scaled*

# REPLICA SET HISTORY



# LOAD BALANCERS

- Load Balancer is associated with:
  - An ingress protocol
  - 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.

# LOAD BALANCING

# LOAD BALANCING WITH INGRESS

The following shows the ingress protocol

Show  Server Groups  Instances

**EKS-PROD** **ingress** **mild-temper-ingress**

**EKS-PROD** **service** **mild-temper-microservice-service**

**EKS-STAGE** **ingress** **mild-temper-ingress**

**EKS-STAGE** **service** **mild-temper-microservice-service**

**Create Load Balancer**

**microservices**

**microservices**

**replicaSet** **mild-temper-microservice-698848ccf8** 3 ▲ : 100%

**microservices**

**microservices**

**Ingress Actions**

**mild-temper-ingress**

**INFORMATION**

- Created 2020-03-22 17:42:28 PDT
- Account EKS-STAGE
- Namespace microservices
- Kind Ingress
- Service Type
- Sess. Affinity

**STATUS**

No workloads associated with this Ingress.

**Ingress**

a9ace9e2a6aef11ea98780edbc813d86-1215124460.us-west-2.elb.amazonaws.com

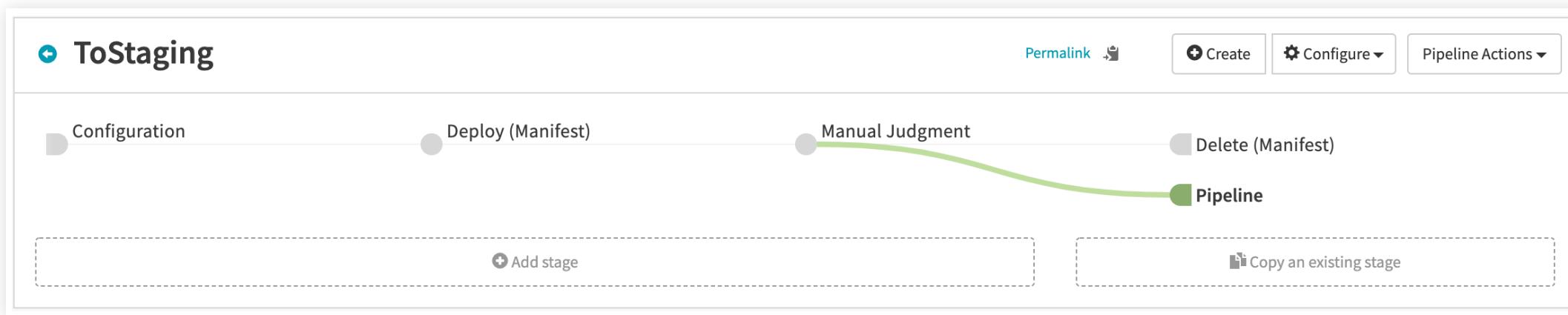
A red arrow points from the EKS-STAGE Ingress details to the external IP address listed under the Ingress section.

## 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>

# PIPELINE



- Key deployment management construct in Spinnaker
- Manage deployments in a consistent, repeatable and safe way.
- Consists of a sequence of actions, known as *stages*
- Possibility to pass parameters from stage to stage along the pipeline using artifacts

# PIPELINES CONTINUED



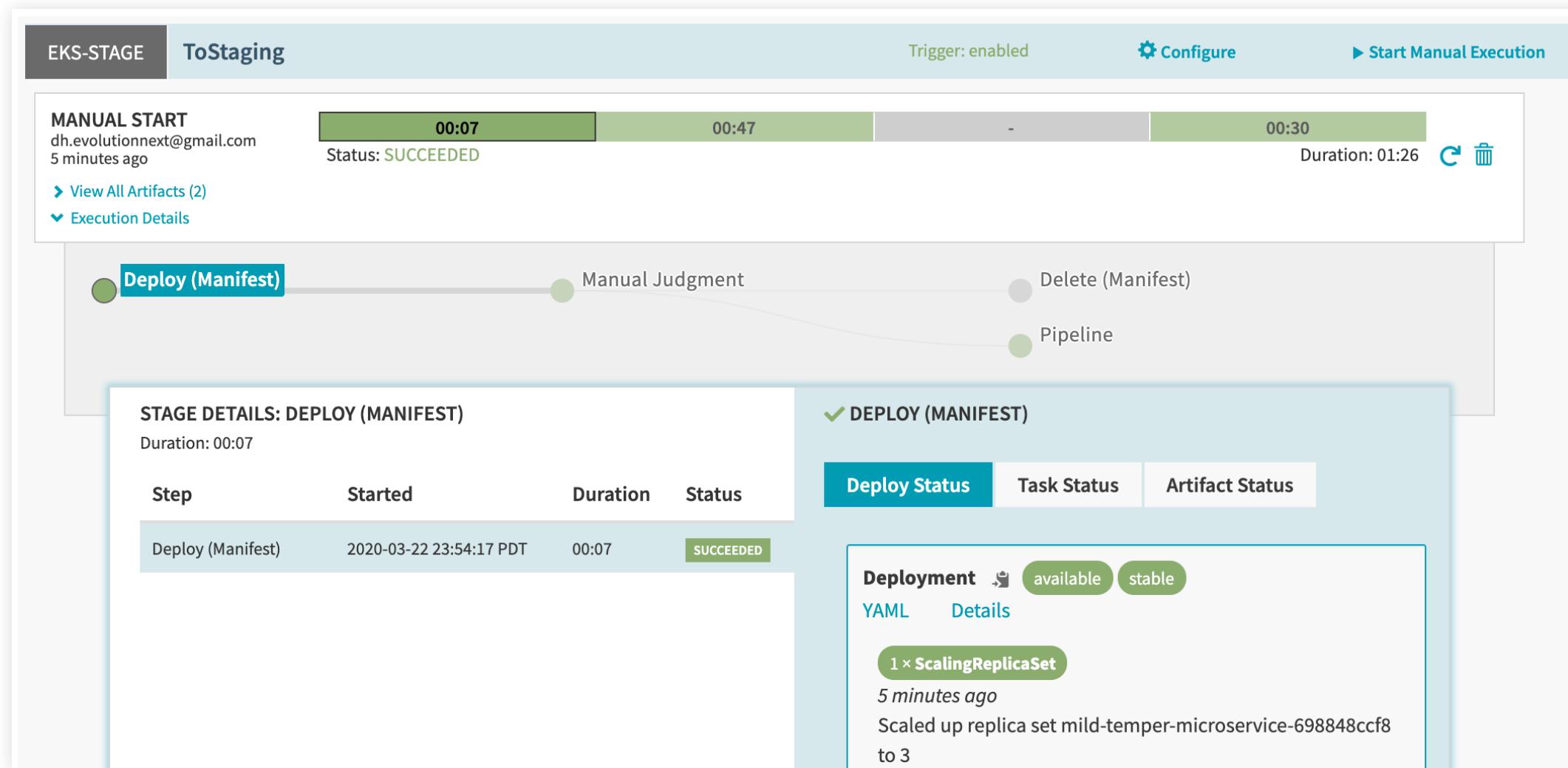
- Includes *scaffolding functions* (manual judgment, wait, run Jenkins job) that together precisely defines your runbook for managing your deployments
- Started manually or triggered by an event:
  - git push
  - Docker image upload
  - CRON Schedule
- You can emit notifications

# PIPELINE ANATOMY

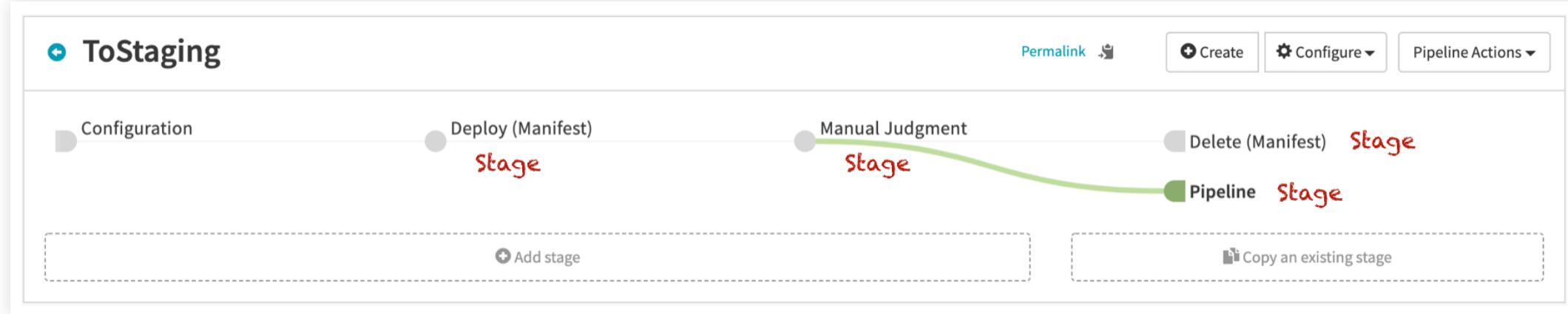
- Define your sequence of stages at the top.
- Spinnaker supports parallel paths of stages, as well as the ability to specify whether multiple instances of a pipeline can be run at once.
- Specify details for a given stage in the sections underneath the pipeline.

# 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.

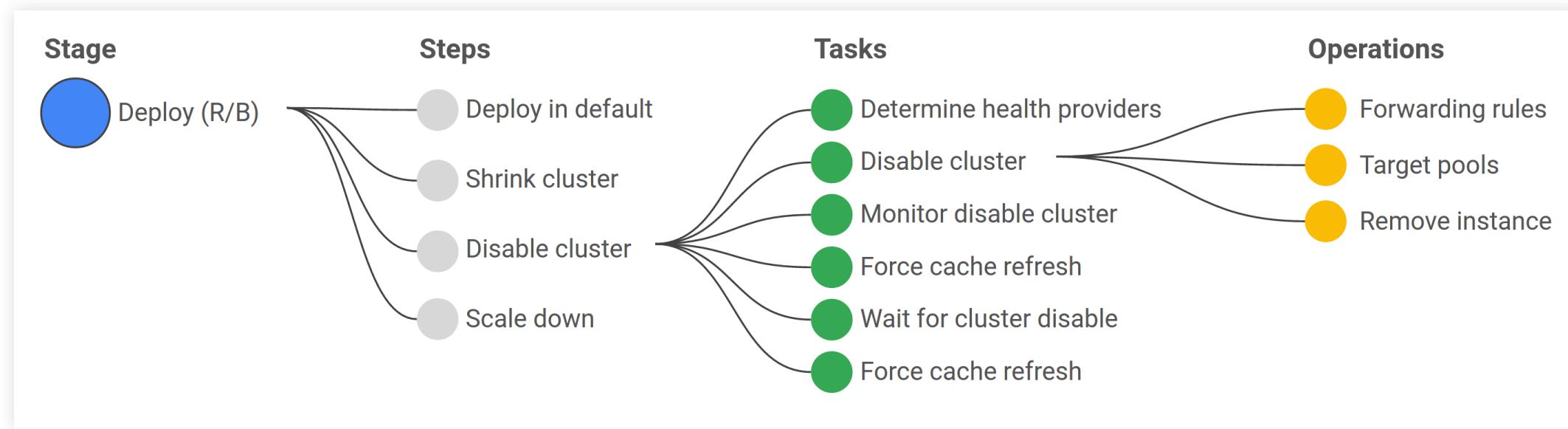


# STAGE



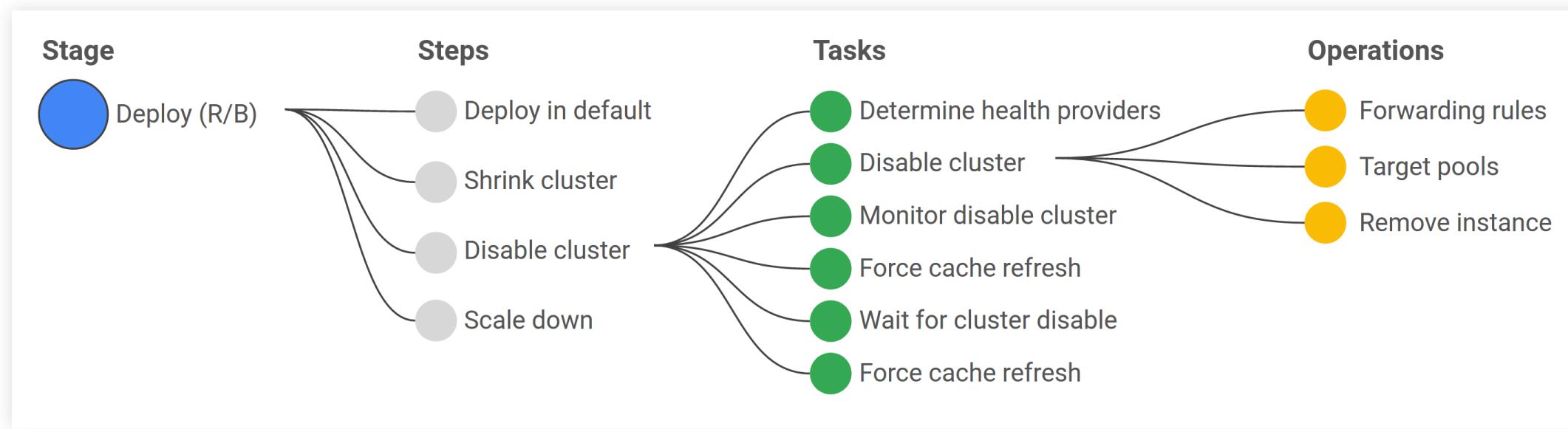
- Atomic building block for a pipeline
- Sequenced to form a pipeline
- Spinnaker provides multiple stages:
  - Deploy
  - Resize
  - Disable
  - Manual Judgment

# ONE STAGE, MULTIPLE STEPS; ONE STEP, MULTIPLE ACTIONS



- 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

# BREAKING DOWN THE STAGE



- 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

# CLUSTERS

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

# CLUSTER DEFINITION

- 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

# LEFT PANE

The screenshot shows the Spinnaker interface with the left pane expanded. The left pane contains filters for viewing deployed services, including:

- SEARCH: A search bar and dropdown for string matching.
- ACCOUNT: Filters for eks-prod and eks-stage.
- REGION: Filter for microservices.
- STACK: Filter for (none).
- DETAIL: Filter for (none).
- STATUS: Filters for Healthy, Unhealthy, Disabled, and Starting.

The main area displays two clusters: EKS-PROD and EKS-STAGE. Each cluster shows a deployment named "mild-temper-microservice" with 3 instances at 100% health. The deployment details include its creation date (2020-03-22), account (EKS-PROD), namespace (microservices), kind (deployment), managing (replicaSet), and name (mild-temper-microservice-698848ccf8). A modal window is open for the deployment in the EKS-PROD cluster, providing more detailed information.

- 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 PANE

The screenshot shows the Spinnaker interface for managing microservices. The top navigation bar includes 'SPINNAKER', 'Search', 'Projects', 'Applications' (selected), 'Pipeline Templates', and a search bar. Below the navigation is a secondary header with tabs for 'PIPELINES', 'INFRASTRUCTURE' (selected), and 'TASKS'. A 'CONFIG' button is also present. The main content area is titled 'mild-temper-microservice' and displays the 'INFRASTRUCTURE' section. Under 'CLUSTERS', there are two entries: 'EKS-PROD' and 'EKS-STAGE'. Each cluster entry shows a deployment named 'mild-temper-microservice' with a status of '3 ▲ : 100%'. The EKS-PROD entry has a 'deployment' icon, while the EKS-STAGE entry has a 'replicaSet' icon. To the right of the clusters, a detailed view of the 'mild-temper-microservice' deployment in 'EKS-PROD' is shown. This view includes sections for 'INFORMATION' (Created: 2020-03-22 21:24:36 PDT, Account: EKS-PROD, Namespace: microservices, Kind: deployment, Managing: replicaSet, Name: mild-temper-microservice-698848ccf8), 'Deployment Actions' (with a dropdown arrow), and 'STATUS' (with a dropdown arrow). On the left side of the main pane, there is a sidebar with filters for 'SEARCH', 'ACCOUNT' (eks-prod, eks-stage), 'REGION' (microservices), 'STACK' ((none)), 'DETAIL' ((none)), and 'STATUS' (Healthy, Unhealthy, Disabled, Starting).

- Lists the deployed services.
- Green chicklets are individual instances (pods)
- All grouped into Server Groups (a particular deployment)

# RIGHT PANE

The screenshot shows the Spinnaker interface with the 'RIGHT PANE' highlighted. The left sidebar contains filters for 'ACCOUNT', 'REGION', 'STACK', 'DETAIL', 'STATUS', and search fields. The main center area displays two clusters: 'EKS-PROD' and 'EKS-STAGE'. Each cluster has a 'deployment mild-temper-microservice' entry with a green progress bar at 100%. The right pane is expanded for the 'mild-temper-microservice' deployment in 'EKS-PROD'. It shows detailed information: Created: 2020-03-22 21:24:36 PDT, Account: EKS-PROD, Namespace: microservices, Kind: deployment, Managing: replicaSet, Name: mild-temper-microservice-69884ccf8. A 'Deployment Actions' dropdown is also visible.

- 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).

# 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

# 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

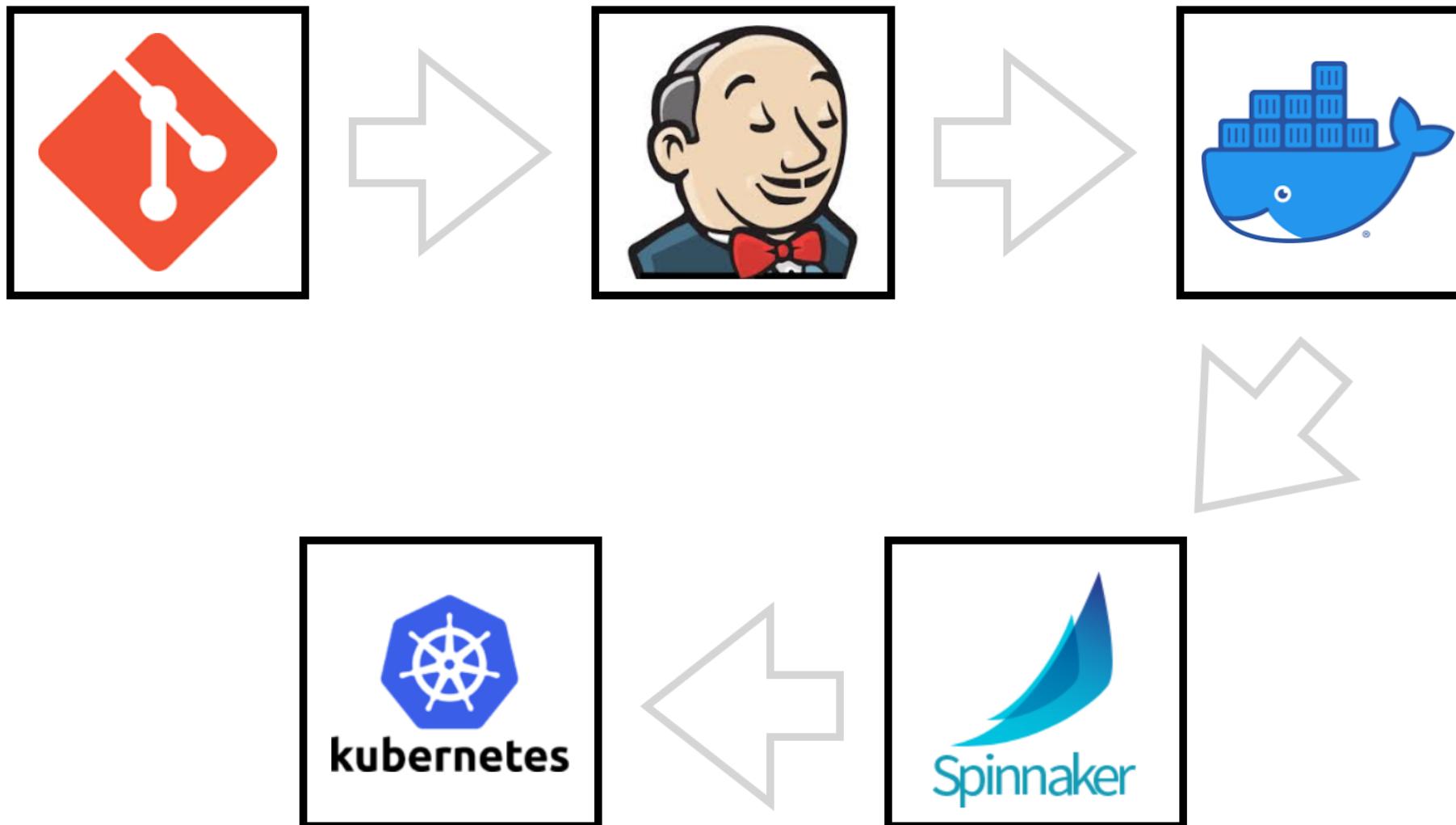
- An Account is a named credential Spinnaker uses to authenticate against a cloud provider.
- Each provider has slightly different requirements for what format credentials can be in, and what permissions they need to have afforded to them.
- Each Supported Provider has their own way to create an account 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

# 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/>

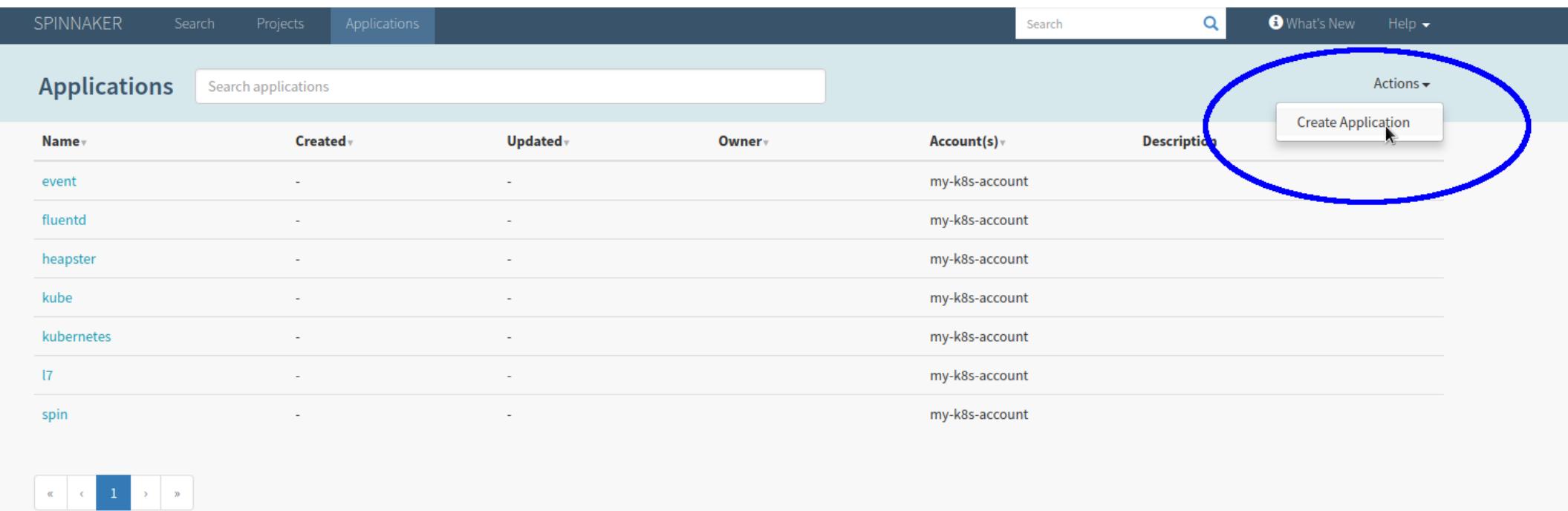
## CREATE AN APPLICATION

- 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



The screenshot shows the Spinnaker UI for managing applications. The top navigation bar includes links for SPINNAKER, Search, Projects, Applications (which is the active tab), What's New, and Help. Below the navigation is a search bar and a 'Create Application' button, which is circled in blue. The main content area is titled 'Applications' and contains a table with columns: Name, Created, Updated, Owner, Account(s), and Description. The table lists several applications: event, fluentd, heapster, kube, kubernetes, l7, and spin, all associated with the account 'my-k8s-account'. At the bottom of the table is a pagination control with a page number '1'.

Name	Created	Updated	Owner	Account(s)	Description
event	-	-		my-k8s-account	
fluentd	-	-		my-k8s-account	
heapster	-	-		my-k8s-account	
kube	-	-		my-k8s-account	
kubernetes	-	-		my-k8s-account	
l7	-	-		my-k8s-account	
spin	-	-		my-k8s-account	

- 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

# ENTERING APPLICATION ATTRIBUTES

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

# MORE 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).

## APPLICATION ATTRIBUTES CONTINUED

Field	Required	Description
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**

**Edit Notification**

**Notify via** Email

**Email Address** person\_of\_interest@example.com

**CC Address** innocent\_bystander@example.com

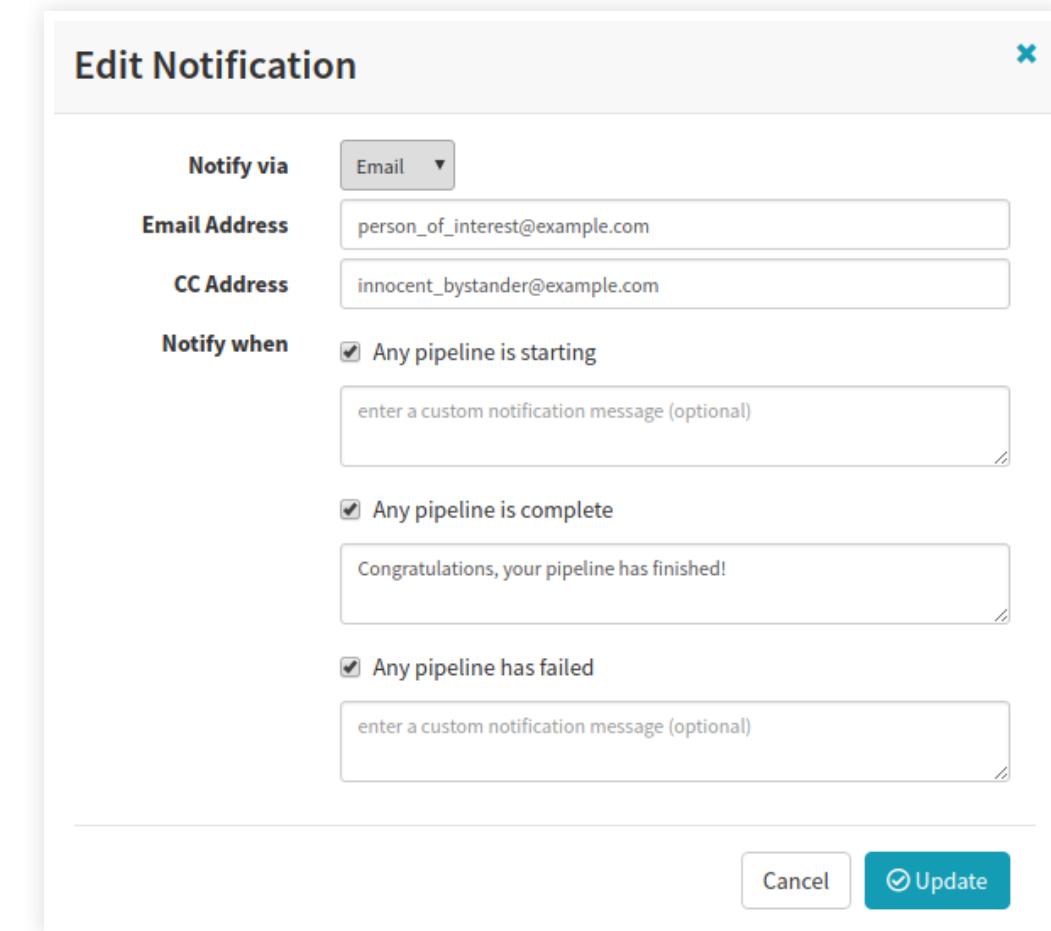
**Notify when**

Any pipeline is starting  
enter a custom notification message (optional)

Any pipeline is complete  
Congratulations, your pipeline has finished!

Any pipeline has failed  
enter a custom notification message (optional)

**Cancel** **Update**



# SELECTING THE NOTIFICATION MEDIUM

- Select the notification medium:
  - Email
  - SMS
  - Slack

**Edit Notification**

Notify via: Email

Email Address: person\_of\_interest@example.com

CC Address: innocent\_bystander@example.com

Notify when:

Any pipeline is starting  
enter a custom notification message (optional)

Any pipeline is complete  
Congratulations, your pipeline has finished!

Any pipeline has failed  
enter a custom notification message (optional)

# EDITING NOTIFICATIONS

- Provide the information for the type you chose:
  - Email address
  - Phone number
  - Slack Members

**Edit Notification**

Notify via: Email

Email Address: person\_of\_interest@example.com

CC Address: innocent\_bystander@example.com

Notify when:

- Any pipeline is starting  
enter a custom notification message (optional)
- Any pipeline is complete  
Congratulations, your pipeline has finished!
- Any pipeline has failed  
enter a custom notification message (optional)

# PIPELINE EVENTS

For each event, enter any custom text you want included in the notification.

- A pipeline started
- A pipeline finished
- A pipeline failed

**Edit Notification**

Notify via: Email

Email Address: person\_of\_interest@example.com

CC Address: innocent\_bystander@example.com

Notify when:

Any pipeline is starting  
enter a custom notification message (optional)

Any pipeline is complete  
Congratulations, your pipeline has finished!

Any pipeline has failed  
enter a custom notification message (optional)

# UPDATE AND ADDING NOTIFICATIONS

1. Click **Update**
2. Repeat these steps to add more notifications, if you want.
3. Click **Edit** next to any existing notification to change its preferences, or click **Remove** to delete it.

**Edit Notification**

Notify via: Email

Email Address: person\_of\_interest@example.com

CC Address: innocent\_bystander@example.com

Notify when:

- Any pipeline is starting  
enter a custom notification message (optional)
- Any pipeline is complete  
Congratulations, your pipeline has finished!
- Any pipeline has failed  
enter a custom notification message (optional)

# ENABLING FEATURES IN SPINNAKER

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

**Features**

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

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

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

 In sync with server

# 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

## Links

Links appear in the instance details panel and provide a shortcut to common features, such as logs, health, etc. [Edit as JSON](#)

Links paths are templates - you can customize them using a number of field attributes available on the instance.

[Check out the guide](#) for examples and to see what fields are available for your cloud provider.

Section Heading	Name	URL	Actions
Links	Label, e.g. Health	Path, e.g. /health, :7002/alt-port-link	
<a href="#">+ Add Link</a>			
<a href="#">+ Add Section</a>			

# CUSTOM LINK PROCESS

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

## Traffic Guards

Traffic Guards allow you to specify critical clusters that should always have active instances. 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.

Enabled	Account	Region <small>?</small>	Stack <small>?</small>	Detail <small>?</small>	Matched clusters
<input checked="" type="checkbox"/>	<div><input checked="" type="button"/> Select... eks-prod eks-stage</div>	(select account)			(no matches) <span>trash</span>
<a href="#">+ Add Traffic Guard</a>					

[⚡ Revert](#) [Save Changes](#)

- 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.

# TRAFFIC GUARD PROCESS

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

## Custom Banners

Custom Banners allow you to specify application-specific headers that will appear above the main Spinnaker navigation bar.

Enabled	Text	Text Color	Background	
<input type="checkbox"/>	Your custom banner text	<input type="color"/> ▾	<input type="color"/> ▾	<span>trash</span>
Markdown is okay ⓘ				
<b>Preview</b>				
Your custom banner text				

**+ Add banner**

⚡ Revert ✓ Save Changes

## CUSTOM BANNER PROCESS

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 environment

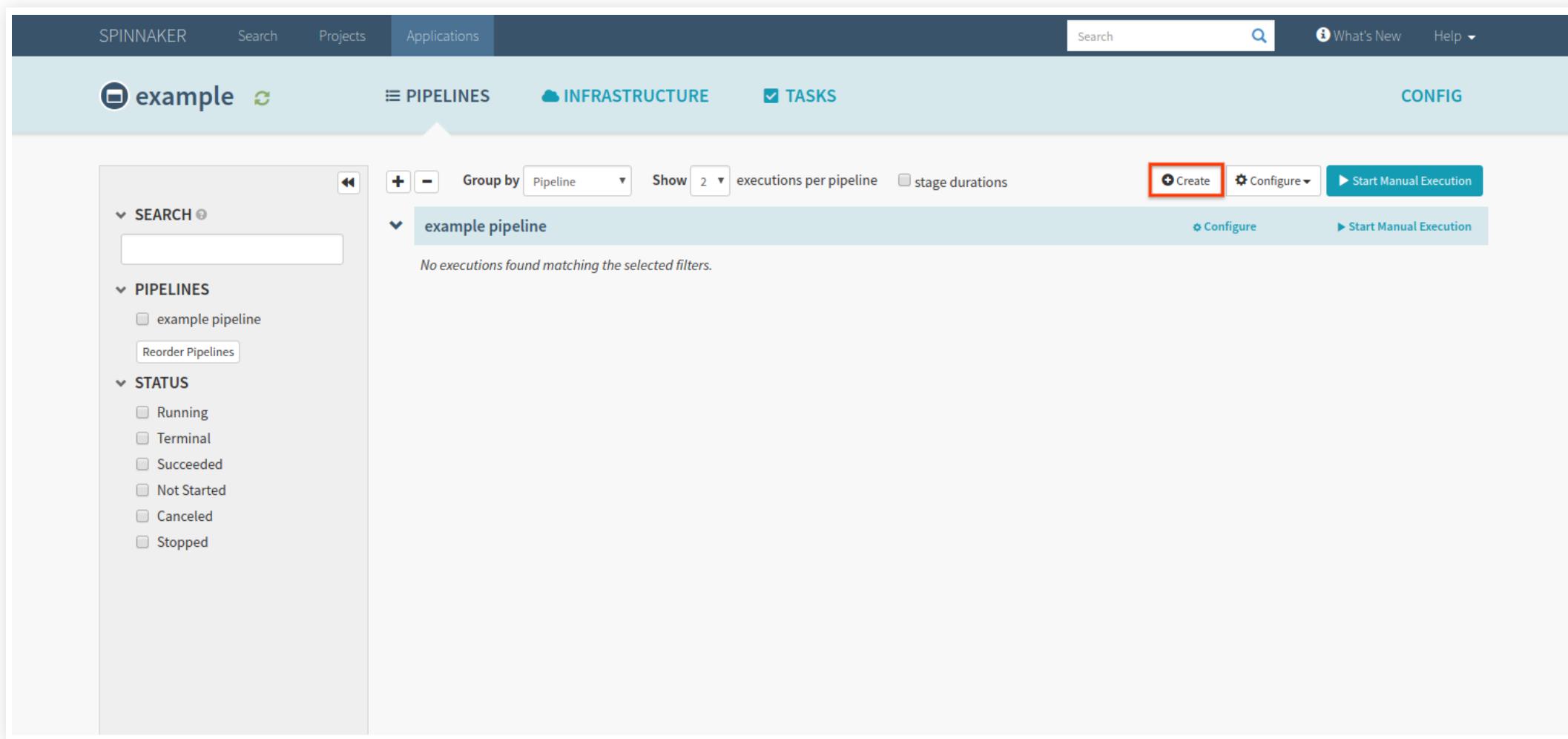
**i** this step is not a prerequisite for creating pipelines. In fact, you can use pipelines to create infrastructure.

# 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
  - 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 screenshot shows the Spinnaker interface for managing pipelines. At the top, there's a navigation bar with 'SPINNAKER', 'Search', 'Projects', and 'Applications'. Below that is a secondary navigation bar with tabs for 'example' (selected), 'PIPELINES' (selected), 'INFRASTRUCTURE', 'TASKS', and 'CONFIG'. The main content area is titled 'example pipeline'. It has a 'Configuration' section with a red box around the 'Add stage' button. To the right is a dashed box with a 'Copy an existing stage' button. On the left, there are tabs for 'CONCURRENT EXECUTIONS' (selected), 'AUTOMATED TRIGGERS', 'PARAMETERS', 'NOTIFICATIONS', and 'DESCRIPTION'. The 'CONCURRENT EXECUTIONS' tab contains sections for 'Concurrent Executions' (with checkboxes for 'Disable concurrent pipeline executions (only run one at a time)' and 'Do not automatically cancel pipelines waiting in queue.'), 'Automated Triggers' (with a message: 'You don't have any triggers configured for example pipeline.' and an 'Add Trigger' button), and a 'Revert' button. At the bottom right is a 'Save Changes' button.

## ADDING STAGE PROCESS

- 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.

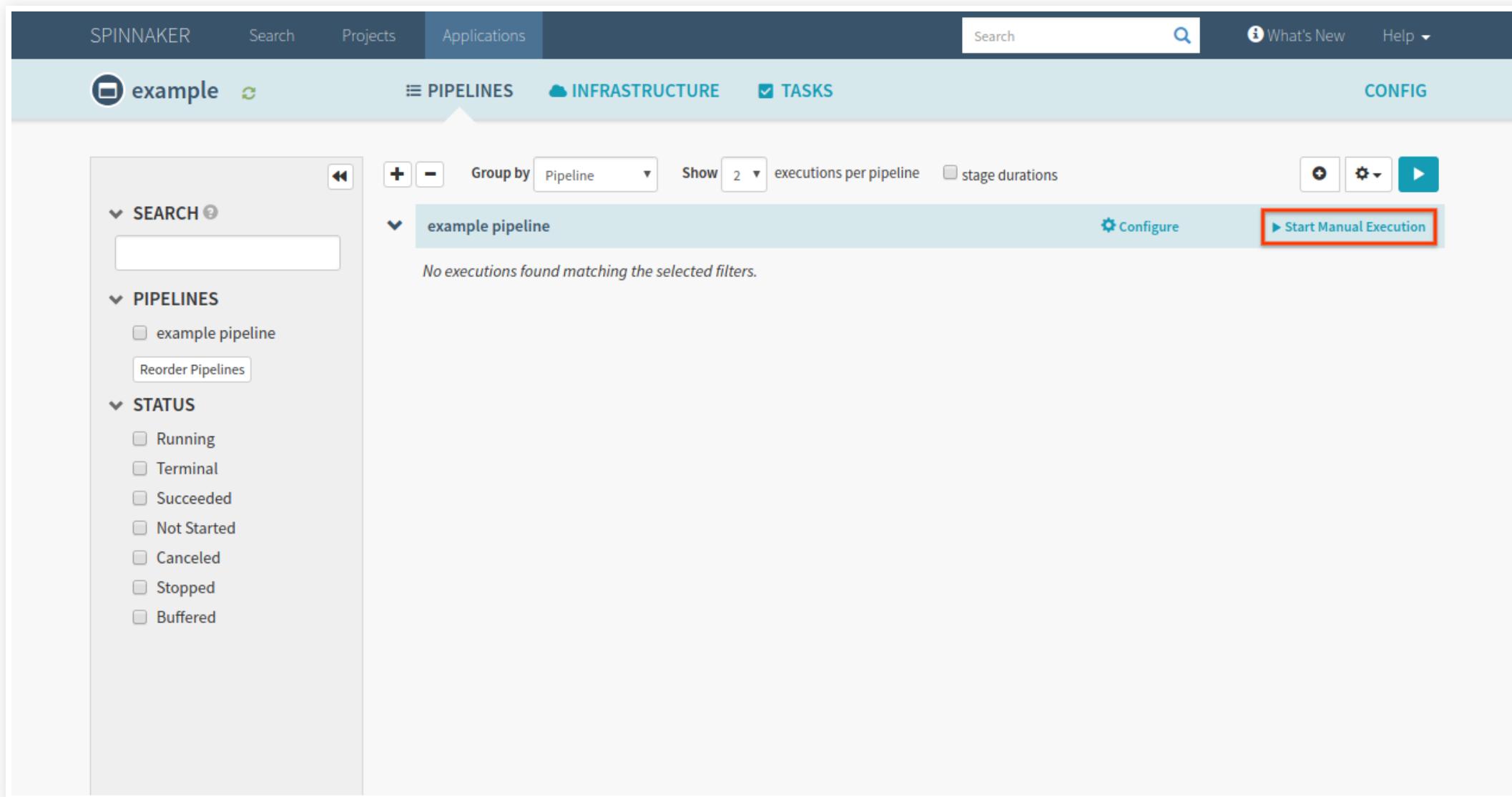
# STAGE DEPENDS ON

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.

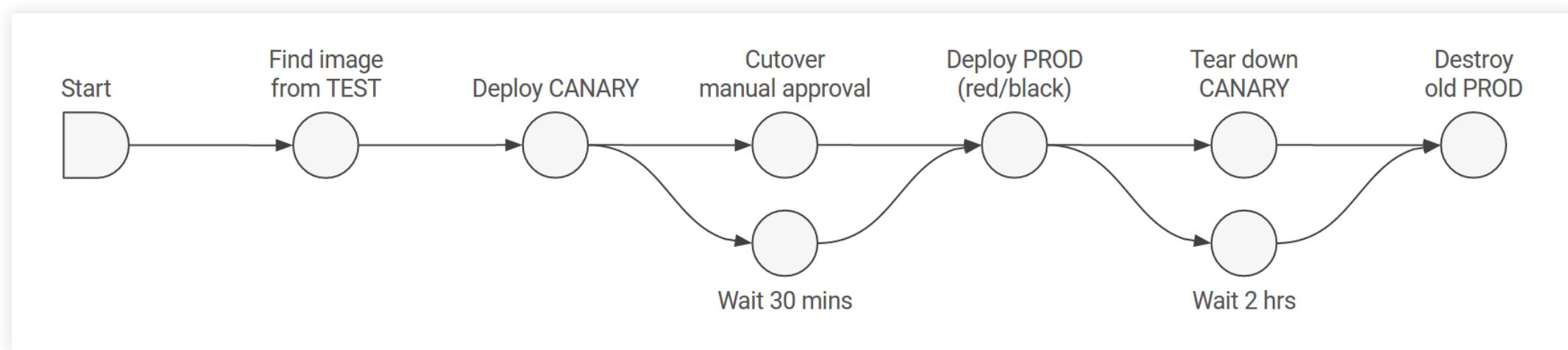
The screenshot shows the Spinnaker UI interface for managing pipelines. At the top, there's a navigation bar with tabs for SPINNAKER, Search, Projects, Applications, PIPELINES (which is selected), INFRASTRUCTURE, TASKS, and CONFIG. Below the navigation is a breadcrumb trail showing 'example > PIPELINES > example pipeline'. The main area displays the 'example pipeline' configuration. The pipeline consists of three stages: Configuration, Bake, and Deploy. The 'Deploy' stage is currently selected. On the left, there are buttons for 'Add stage' and 'Copy an existing stage'. On the right, there are buttons for 'Create', 'Configure', 'Pipeline Actions', 'Remove stage', and 'Edit stage as JSON'. The 'Depends On' field for the Deploy stage is highlighted with a red box, containing the value 'Bake'. Below the pipeline configuration, there are sections for 'DEPLOY CONFIGURATION' (with tabs for EXECUTION OPTIONS, NOTIFICATIONS, and COMMENTS), 'Deploy Configuration' (with a table for Account, Cluster, Region, Strategy, Capacity, and Actions), and buttons for 'Revert' and 'Save Changes'.

# 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**

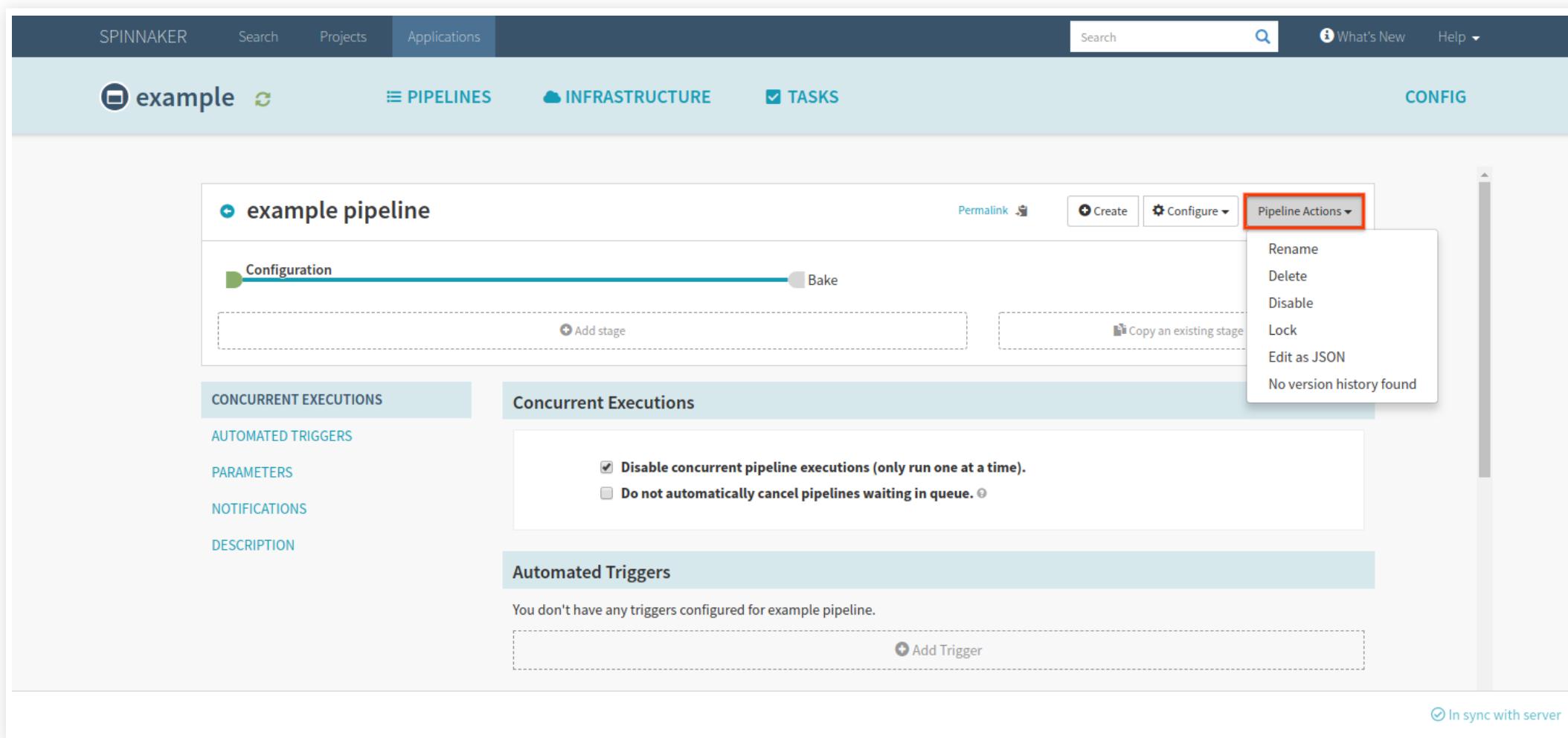


# APPLICATION DEPLOYMENT



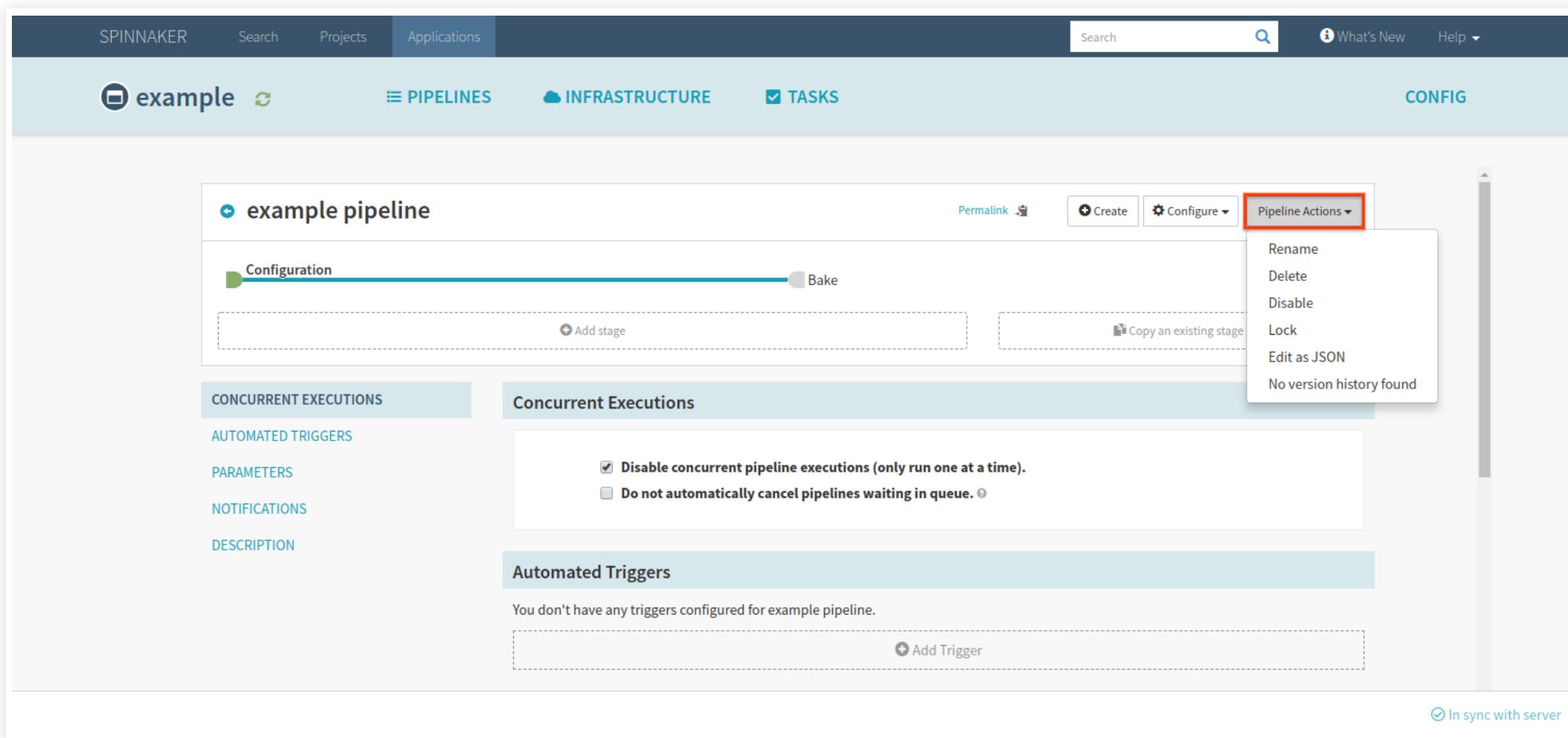
# DISABLING A PIPELINE

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



# 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**.



# EDITING AS JSON

The screenshot shows the Spinnaker interface for managing pipelines. At the top, there's a navigation bar with 'SPINNAKER', 'Search', 'Projects', 'Applications', a search bar, 'What's New', and 'Help'. Below the navigation is a header with tabs: 'example' (selected), 'PIPLINES' (with a count of 1), 'INFRASTRUCTURE', 'TASKS', and 'CONFIG'.

The main content area displays the 'example pipeline'. It consists of two stages: 'Configuration' (green) and 'Bake' (grey). There are buttons for 'Add stage' and 'Copy an existing stage'. To the right of the stages is a 'Pipeline Actions' dropdown menu, which is highlighted with a red box. The menu contains the following options: 'Rename', 'Delete', 'Disable', 'Lock', 'Edit as JSON', and 'No version history found'. The 'Edit as JSON' option is the last item in the list.

On the left side of the pipeline view, there are several sections: 'CONCURRENT EXECUTIONS' (selected), 'AUTOMATED TRIGGERS', 'PARAMETERS', 'NOTIFICATIONS', and 'DESCRIPTION'. The 'CONCURRENT EXECUTIONS' section contains two checkboxes: 'Disable concurrent pipeline executions (only run one at a time)' (checked) and 'Do not automatically cancel pipelines waiting in queue'.

The 'AUTOMATED TRIGGERS' section has a message: 'You don't have any triggers configured for example pipeline.' and a 'Add Trigger' button.

At the bottom right, there's a status indicator 'In sync with server' with a checkmark.

## EDITING AS JSON PROCESS

- 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.

# REVISION HISTORY

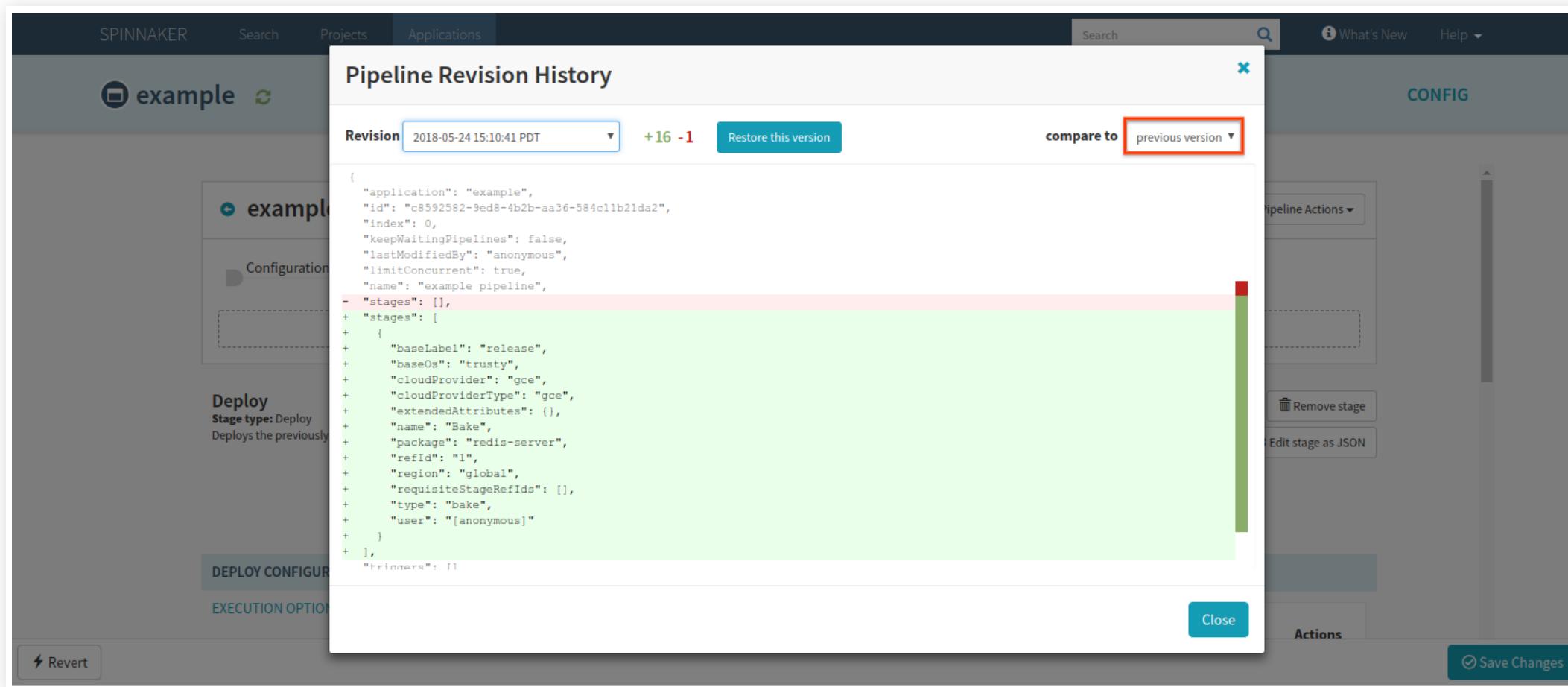
- Each time you save your pipeline, the current version is added to revision history
- You can use revision history to diff two versions of a pipeline or to restore an older version of a pipeline.
  1. From the **Pipelines** tab, click **Configure** to modify an existing pipeline
  2. Click **Pipeline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

# 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.

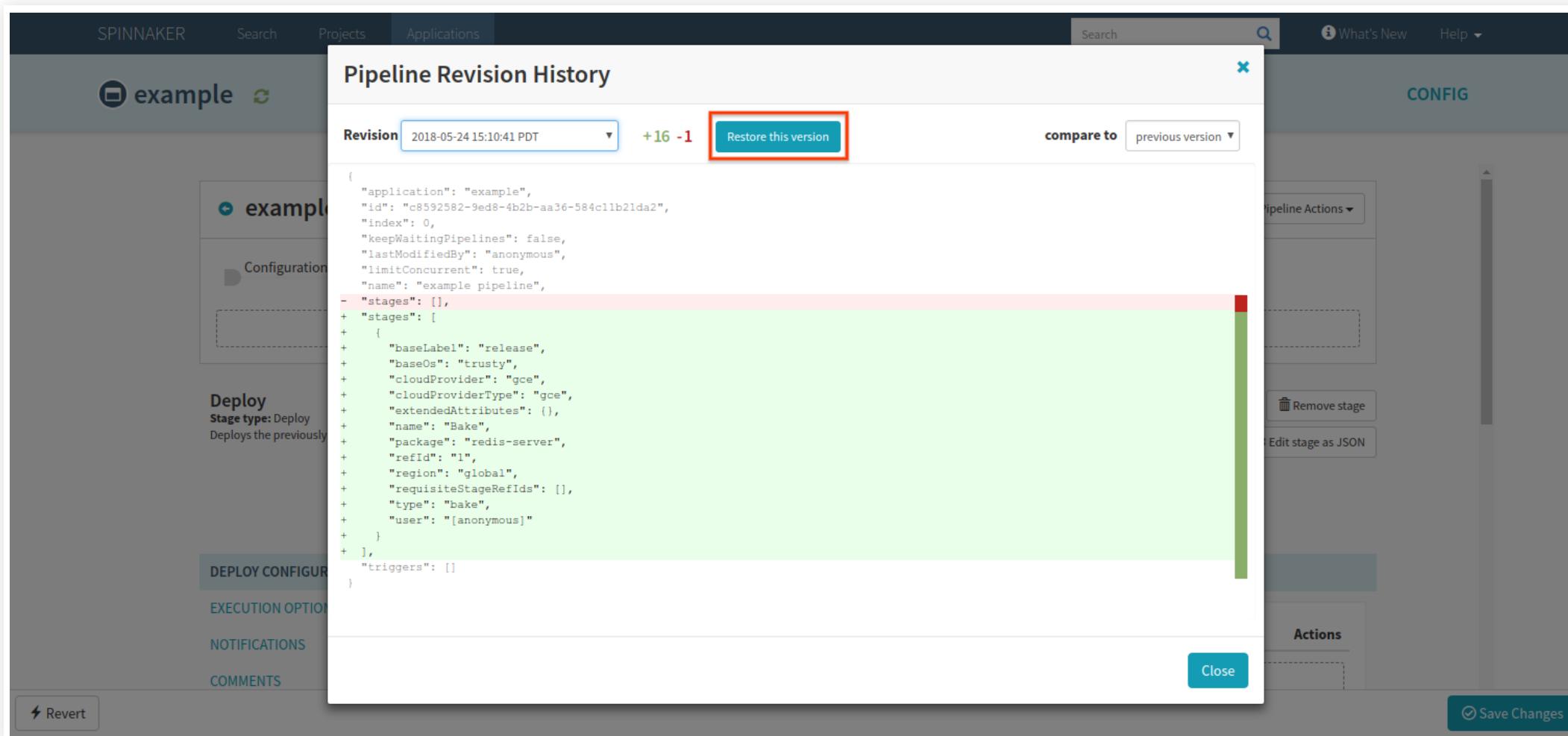
# COMPARING VERSION

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



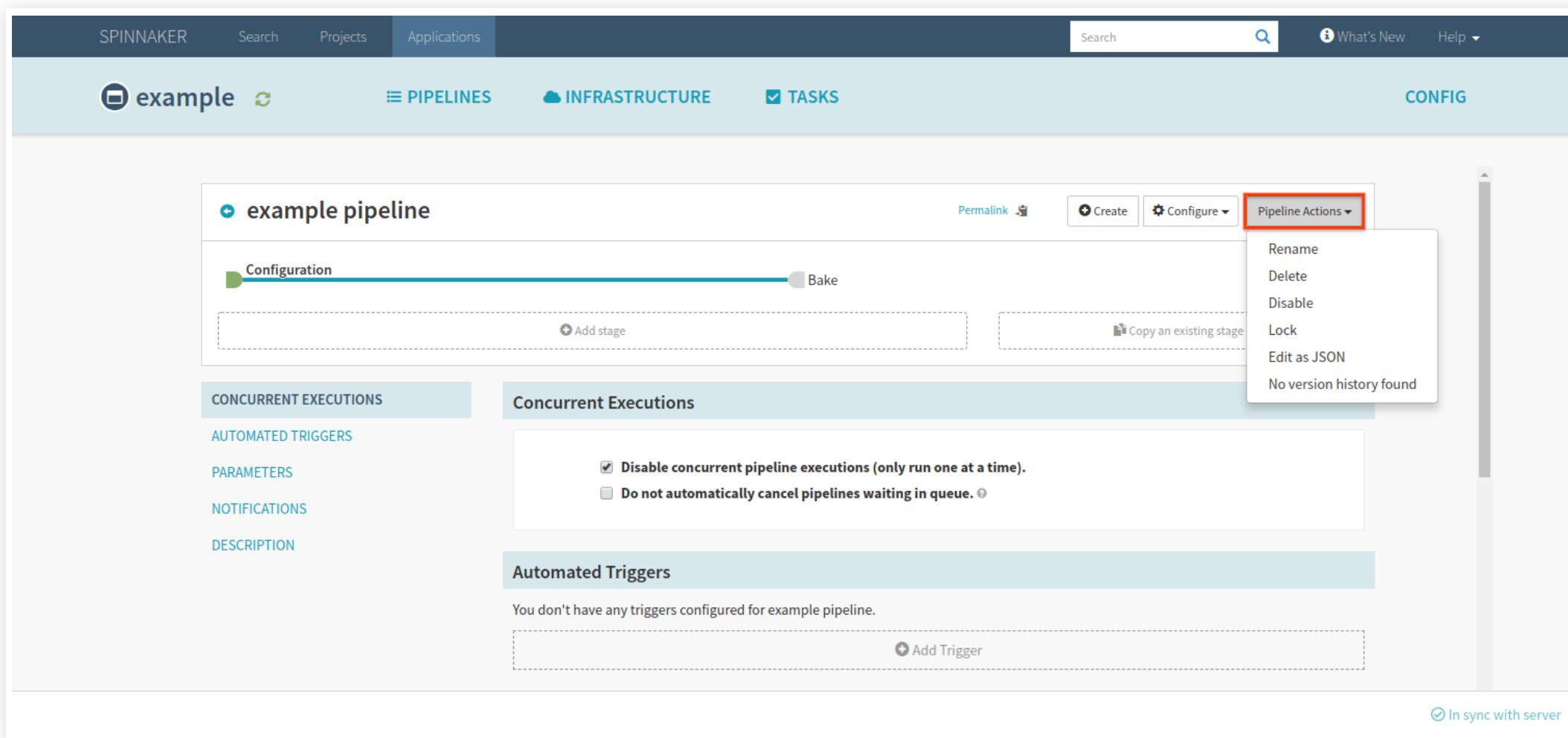
# REVERTING YOUR PIPELINE

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



# RENAME A PIPELINE

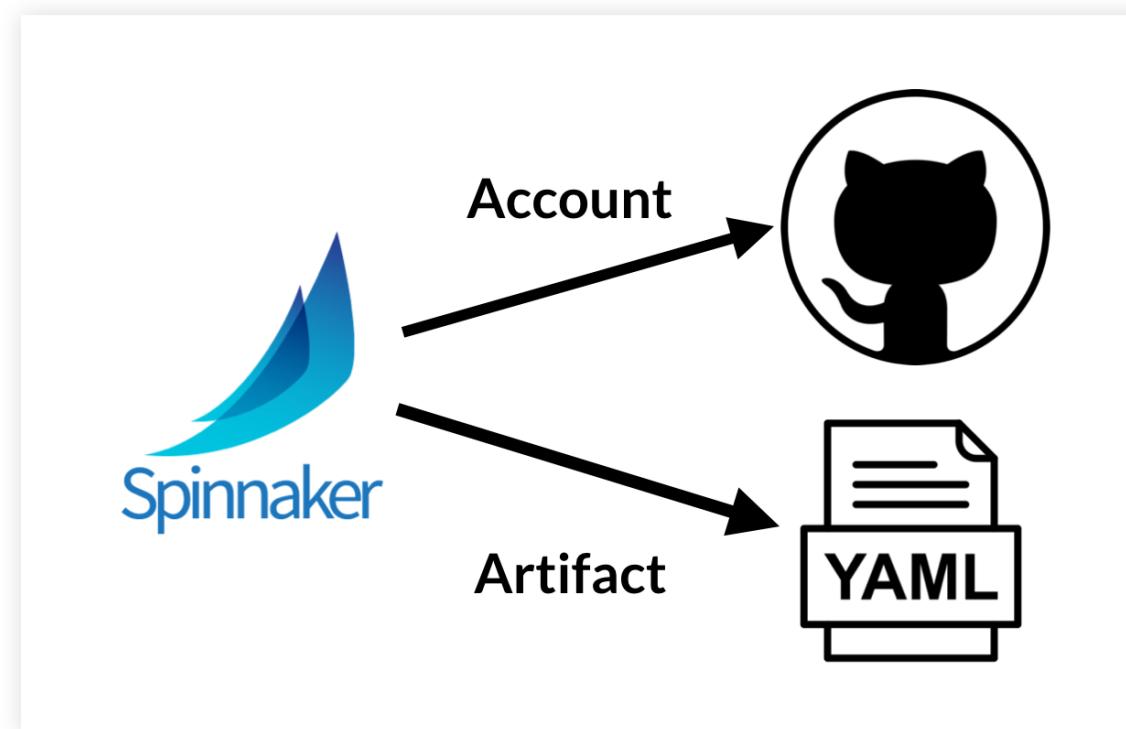
1. From the **Pipelines** tab, click **Configure** to modify an existing pipeline.
2. Click **Pipeline actions** in the upper right corner, and select **Rename**



# 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://XXXXXXXXXXXX-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.

## SPINNAKER ARTIFACT FIELDS CONTINUED

metadata	Arbitrary key / value metadata pertaining to the resource. <i>(This can be useful for scripting within pipeline stages.)</i>	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 Artifacts

Declare artifacts your pipeline expects during execution in this section. [?](#)

Match against [?](#)

**Account** abollapr-account [Remove artifact](#)

**File path** manifests/deploy-microservice.yml

---

**Display name** soul-source-deploy-microservice.yml

---

If missing [?](#)

**Use Prior Execution**

**Use Default Artifact**

## 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 ⓘ

**Use Prior Execution**

**Use Default Artifact**

Default artifact ⓘ

<b>Account</b>	 abollapr-account
<b>Content URL</b> ⓘ	<a href="https://api.github.com/repos/abollapr/quarkus-microservice/contents/manifests/de">https://api.github.com/repos/abollapr/quarkus-microservice/contents/manifests/de</a>
<b>Commit/Branch</b> ⓘ	master

# PRIOR EXECUTION AND DEFAULT ARTIFACTS FIELDS

- **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

## 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!

**Automated Triggers**

Type Docker Registry x ▾ Remove trigger

Define Image ID Select from list

Registry Name my-ecr-registry x ▾ ⟳

Organization No organization ▼

Image soul-source-microservice x ▾

Tag ?

Artifact Constraints ? Select...

Trigger Enabled

The screenshot shows a configuration form for an 'Automated Triggers' entry. At the top, the 'Type' is set to 'Docker Registry'. Below this, there's a section for defining the image ID, which includes a dropdown menu labeled 'Select from list'. Underneath, the 'Registry Name' is specified as 'my-ecr-registry', and the 'Image' name is 'soul-source-microservice'. There are also fields for 'Organization' (set to 'No organization') and 'Tag' (empty). At the bottom of the main configuration area, there's a section for 'Artifact Constraints' with a dropdown menu labeled 'Select...', and a checkbox labeled 'Trigger Enabled' which is checked.

## 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

## 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:

**Produces Artifacts**

Match against  GCS A GCS object.

**Object path**  `gs://manifest/baked-manifest.yml`

If missing   Use Default Artifact

  Add Artifact

 Remove artifact

## WAYS TO PRODUCE ARTIFACTS

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.

**i** 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

## SOURCES

<https://www.spinnaker.io/reference/artifacts-with-artifactsrewrite/>

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

# **LINKING SPINNAKER TO GITHUB**

# OVERVIEW

- 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

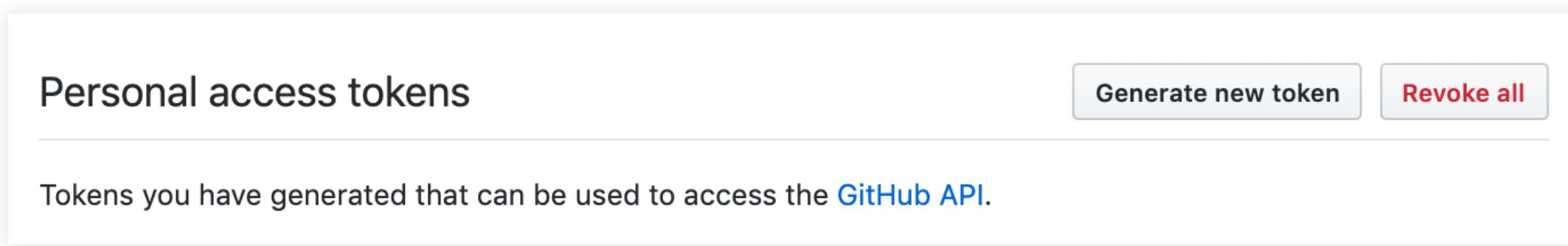


Figure 3. Token Page

# SELECT THE public\_repo SCOPE

Select the `public_repo` scope

## Note

spinnaker-instance

What's this token for?

## Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

- |  |                                      |
|--|--------------------------------------|
| <input type="checkbox"/> <b>repo</b>                         | Full control of private repositories |
| <input type="checkbox"/> <code>repo:status</code>            | Access commit status                 |
| <input type="checkbox"/> <code>repo_deployment</code>        | Access deployment status             |
| <input checked="" type="checkbox"/> <code>public_repo</code> | Access public repositories           |
| <input type="checkbox"/> <code>repo:invite</code>            | Access repository invitations        |

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

Personal access tokens

Generate new token    Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ 9296031cba0e14fa0caf48f670e7d63a2ef29d9 	Delete
---	--------

## 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 1: DEPLOYING EKS MICROSERVICES**

# **EXPRESSIONS**

# EXPRESSIONS DEFINED

- 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 anywhere 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

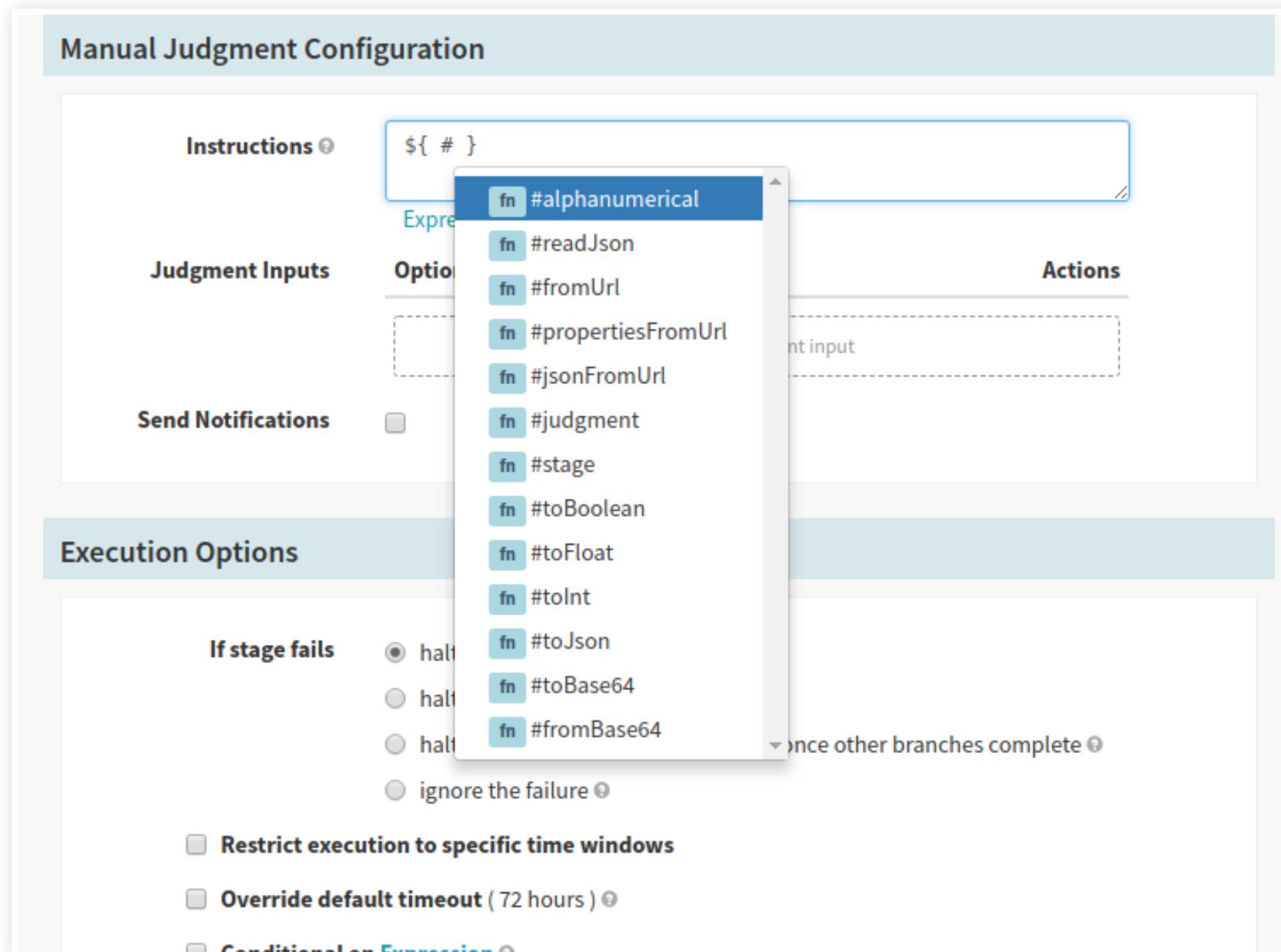
- Expressions are evaluated at the beginning of that stage
- Configuration needs to be evaluated *before* expression

## HELPER FUNCTIONS

- Helper functions simplify common use cases
  - Access a particular stage by name
  - Strip non-alphanumeric 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 PROPERTIES

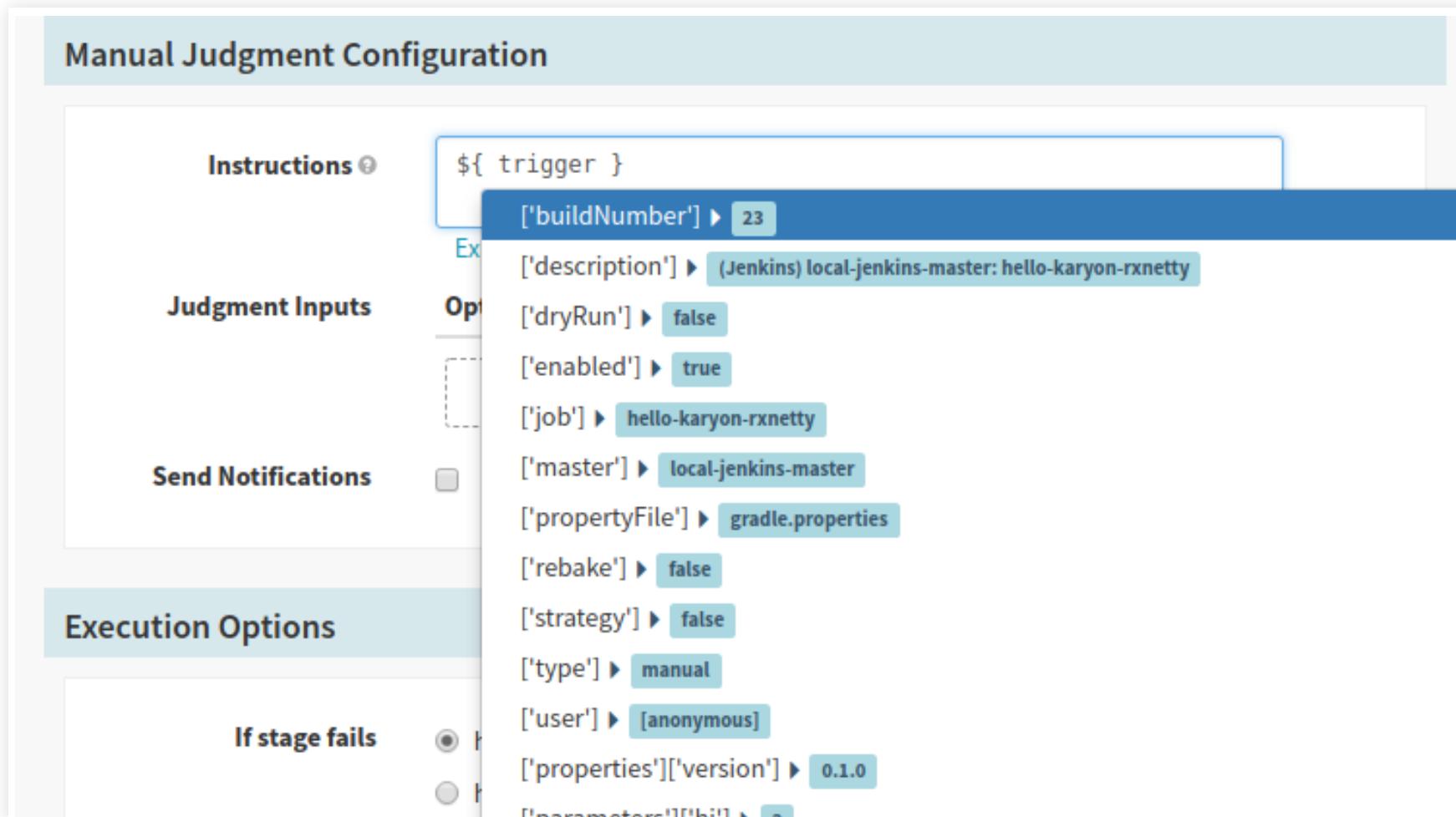
Helper properties are variables which refer to global information about the current pipeline execution.

- `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

! 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

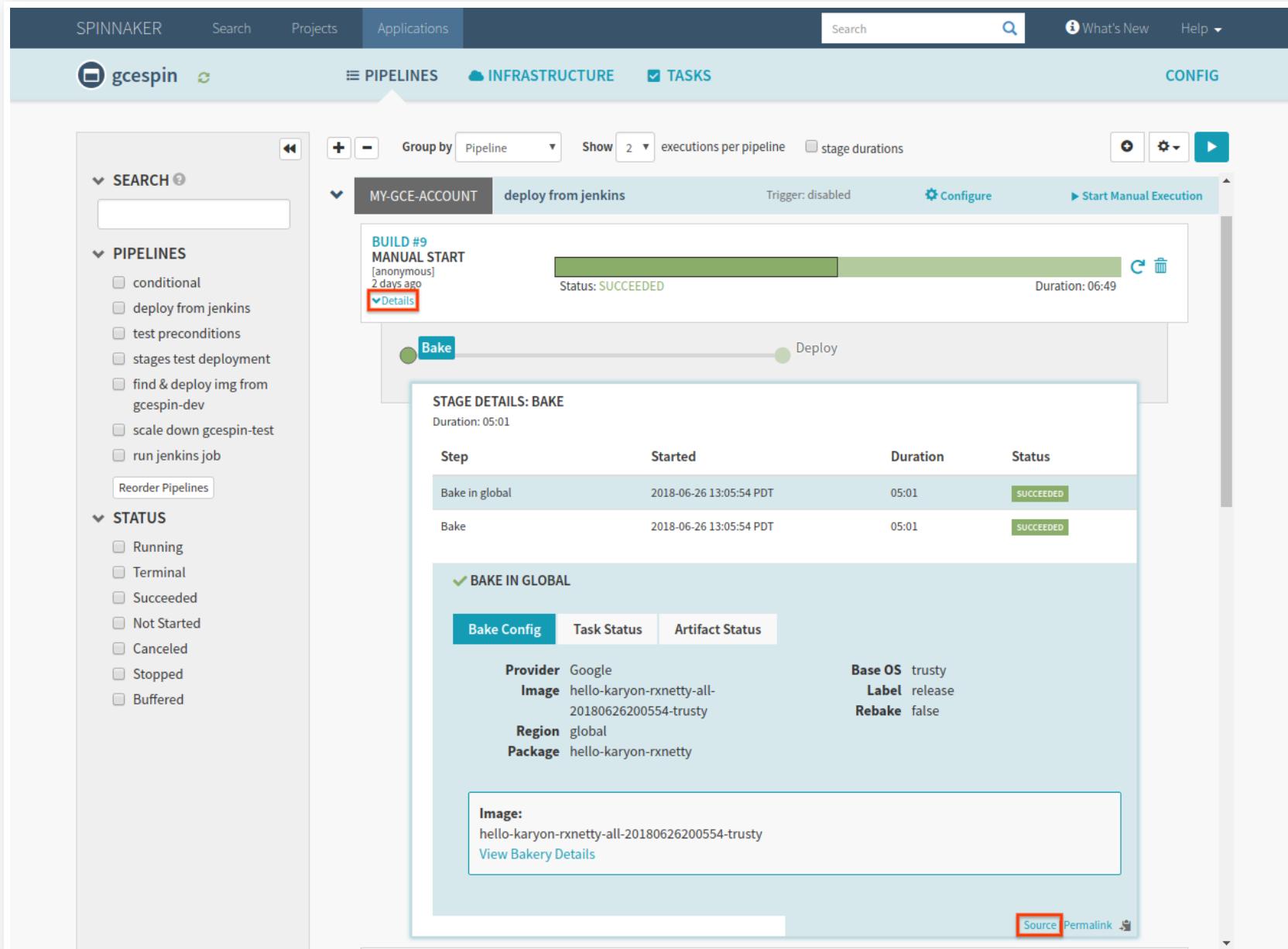
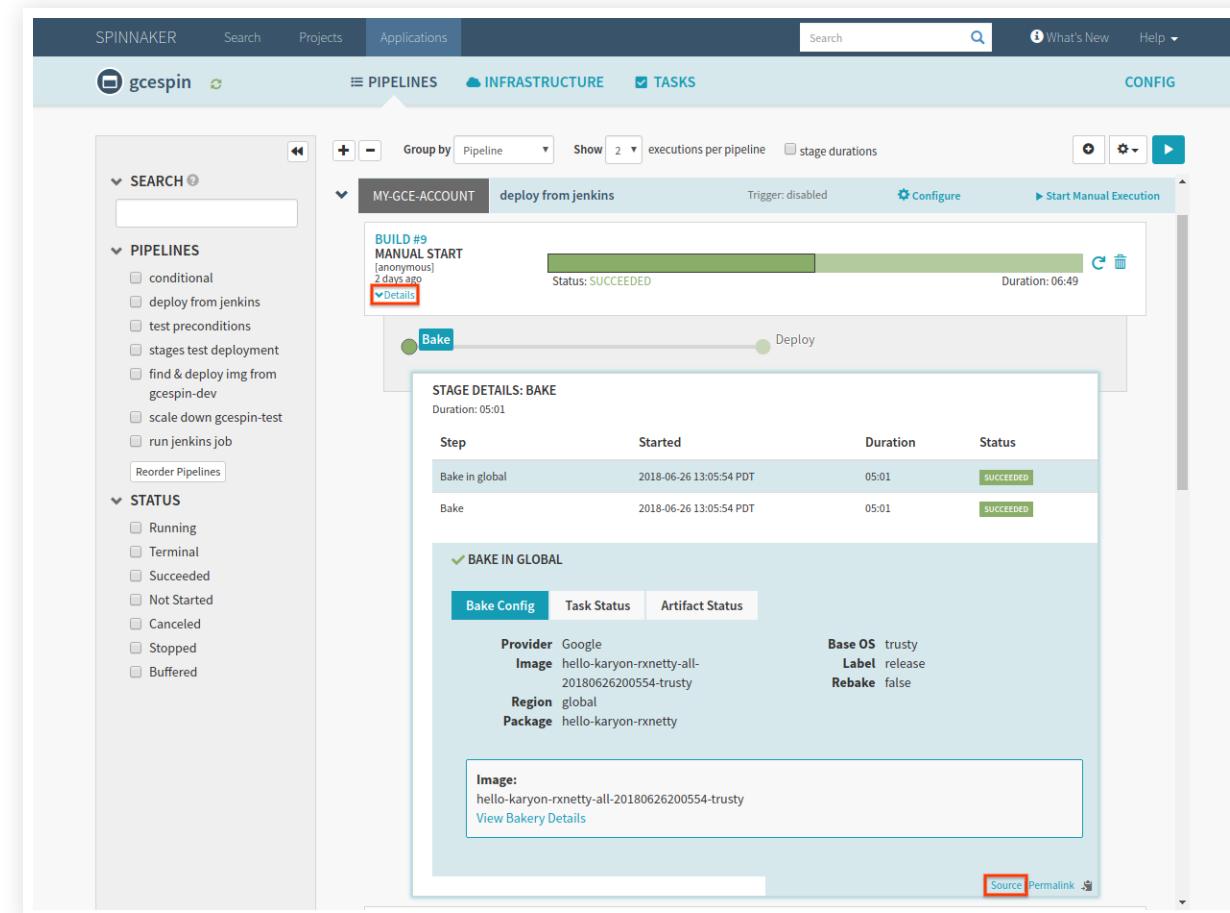


Figure 5. Click on the source link under the pipeline execution, which opens a new tab containing the JSON details of your pipeline execution.

# 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



## USING curl

You can then use `curl` to test your expression

```
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"}
```

# PIPELINE UNSUCCESSFUL

```
{  
  "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()  
",  
        ]  
    }  
}
```

## **LAB 2: USING EXPRESSIONS**

## BAKING

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

## 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"
```

## TEMPLATIZE HELM TEMPLATES:

We can then templatize it with a release, called *release name*

```
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

## DEBUGGING OUR CHART

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

# RESULT CHART

```
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/](https://helm.sh/docs/chart_template_guide/getting_started/)

# INTENTIONS

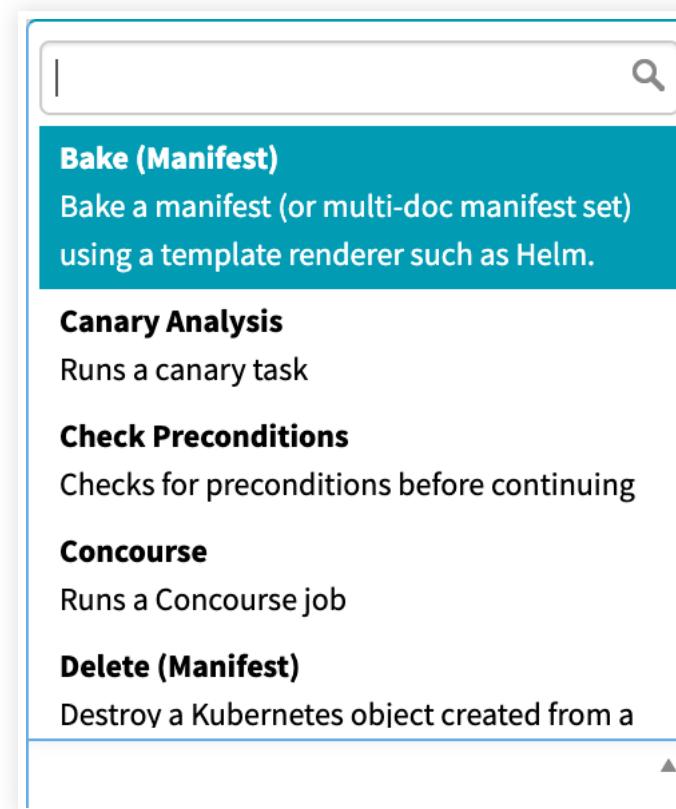
This stage is intended to help you package and deploy applications that you own, and are actively developing and redeploying frequently. It is not intended to serve as a one-time installation method for third-party packages. If that is your goal, it's arguably better to call `helm install` once when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

**Template Renderer**

**Render Engine** HELM2

**Helm Options**

**Name** canary-microservice

**Namespace** microservices

**Template Artifact**

**Expected Artifact**

**Overrides**

**Expected Artifact**

**Add value artifact**

Overrides	Key	Value
		<b>Add override</b>

**Raw Overrides**

**Expression Evaluation**  Evaluate SpEL expressions in overrides at bake time

## CONFIGURE THE BAKE MANIFEST STAGE DETAIL

- 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 key/value pairs that are passed as `--set` parameters to the `helm template` command.

## HELM CHARTS AND NAMESPACE

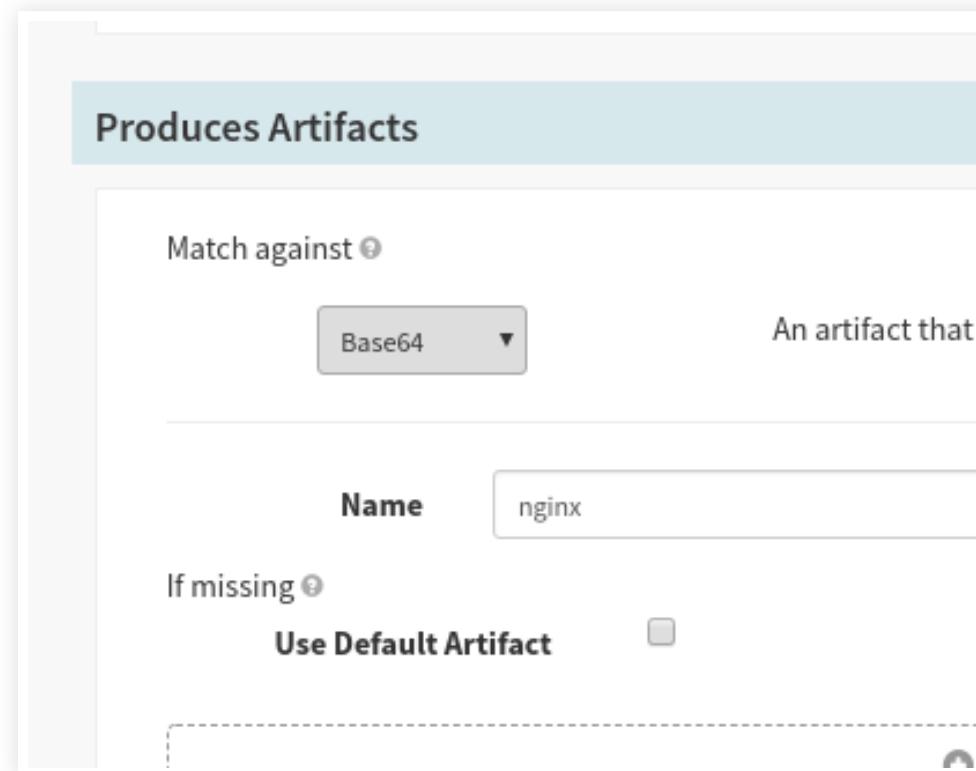
Make sure you have `namespace` declaration in your helm chart if you are overriding the  
namespace

```
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



# CONFIGURE DOWNSTREAM DEPLOYMENT

Now that the manifest set has been baked by Helm

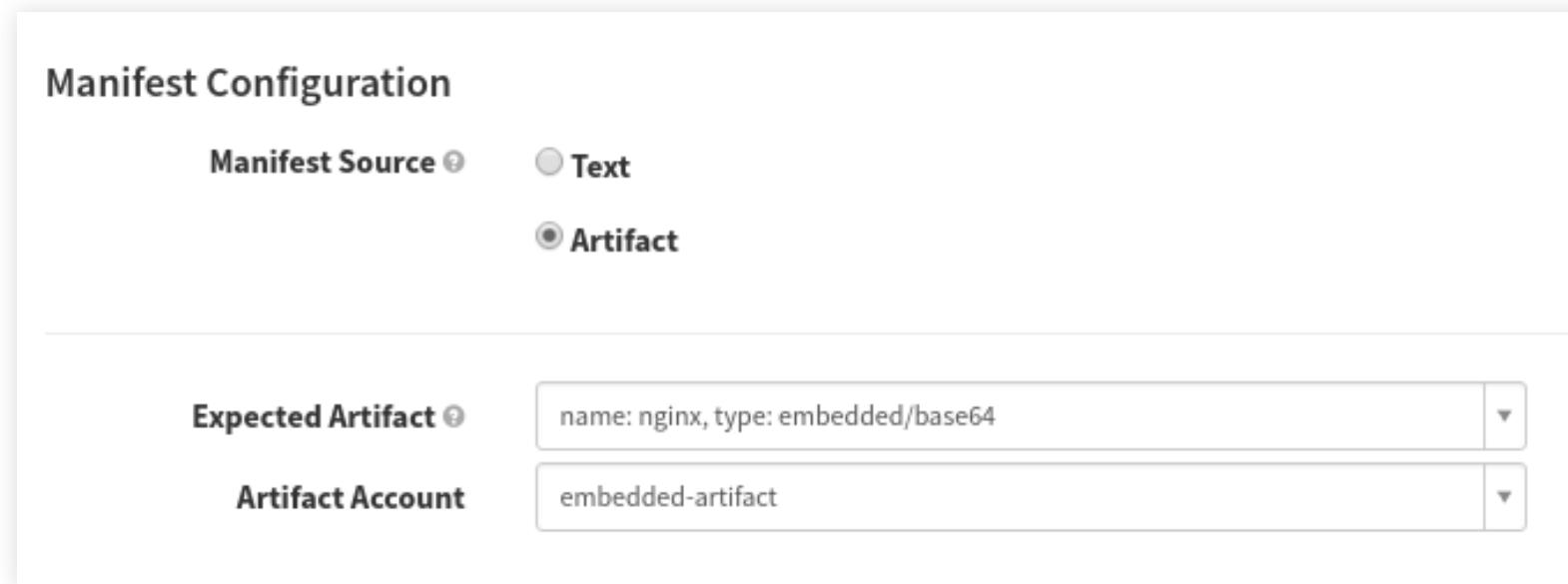
- Configure a downstream stage (in the same pipeline or in one triggered by this pipeline)
- Deploy the artifact produced by the “Bake (Manifest)” stage

Manifest Configuration

Manifest Source  Text  Artifact

Expected Artifact  name: nginx, type: embedded/base64  name: nginx, type: embedded/artifact

Artifact Account  embedded-artifact  artifact-account



- i** 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 3: DEPLOYING HELM MICROSERVICES**

# ROLLBACKS

## AUTOMATIC ROLLBACKS

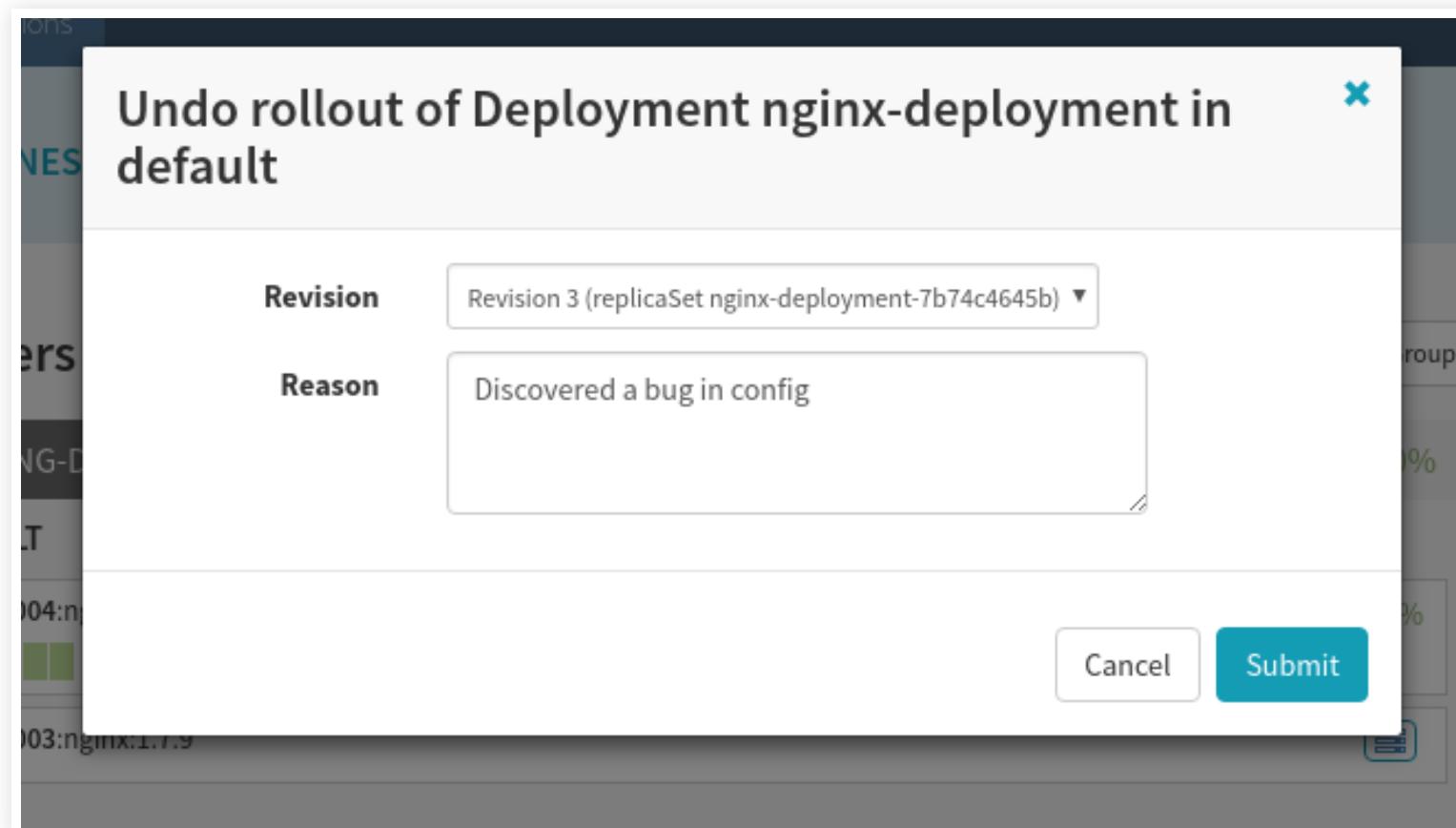
- Having manifests is optimal, IaC
- Rollbacks though are necessary when things go wrong

# 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

# 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



# 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

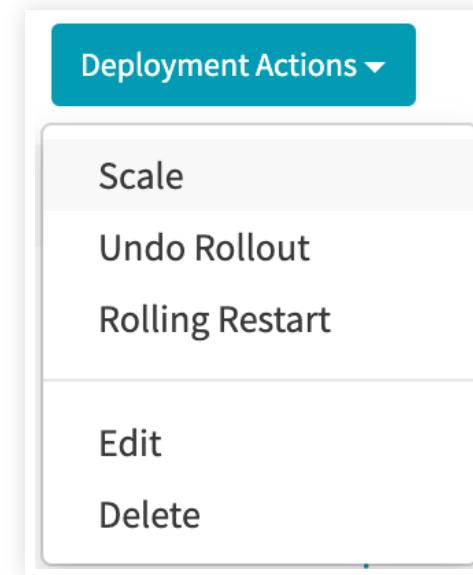
- Rolling Back will not work unless artifacts are versioned, this includes:
  - Docker images
  - ConfigMap
  - Kubernetes Secrets

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

# SCALING

# SCALING RESOURCES

From the **Clusters** Page and selecting the Deployment you can scale the Replica Sets supports



## SELECT THE REPLICAS

You can then select the number of replicas you wish to either increase or decrease

Scale Deployment large-breed-microservice in microservices ×

<b>Replicas</b>	3
<b>Reason</b>	(Optional) anything that might be helpful to explain the reason for this change; HTML is okay

Cancel Submit

- ⓘ If you want the number of replica sets permanent, be sure to update your yaml file, changes will not be persisted

# ROLLOUT STRATEGIES

- Deploys your containers as either:
  - Dark Rollouts
  - Highlander
  - Red/Black (also known as Blue/Green)
- Is only valid for Kubernetes `ReplicaSet`
- Verifies health checks
- Disables old server groups
- Enabling new server groups

# 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

Deploy (Manifest) Configuration

Rollout Strategy Options

<b>Enable</b> ⓘ	<input checked="" type="checkbox"/> Spinnaker manages traffic based on your selected strategy
<b>Service(s) Namespace</b>	Select...
<b>Service(s)</b>	Select...
<b>Traffic</b>	<input type="checkbox"/> Send client requests to new pods
<b>Strategy</b> ⓘ	None

# CONFIGURATION DETAILS

- 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.
- 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).

Rollout Strategy Options

<b>Enable</b> ⓘ	<input checked="" type="checkbox"/> Spinnaker manages traffic based on your selected strategy
<b>Service(s) Namespace</b>	default
<b>Service(s)</b>	x maggie-k8s-demo
<b>Traffic</b>	<input type="checkbox"/> Send client requests to new pods
<b>Strategy</b> ⓘ	None

# 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

**Rollout Strategy Options**

<b>Enable</b> <small>?</small>	<input checked="" type="checkbox"/> Spinnaker manages traffic based on your selected strategy
<b>Service(s) Namespace</b>	default
<b>Service(s)</b>	<input type="text" value="maggie-k8s-demo"/> <small>x</small>
<b>Traffic</b>	<input checked="" type="checkbox"/> Send client requests to new pods
<b>Strategy</b> <small>?</small>	Highlander

# 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.

Rollout Strategy Options

<b>Enable</b> <small>?</small>	<input checked="" type="checkbox"/> Spinnaker manages traffic based on your selected strategy
<b>Service(s) Namespace</b>	default
<b>Service(s)</b>	x maggie-k8s-demo
<b>Traffic</b>	<input checked="" type="checkbox"/> Send client requests to new pods
<b>Strategy</b> <small>?</small>	Red/Black

# 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.

**Execution Options**

**Restrict execution to specific time windows**

**Days of the Week** (No days selected implies execution on any day if triggered)

Sun	Mon	Tue	Wed	Thu	Fri	Sat
-----	-----	-----	-----	-----	-----	-----

All    None    Weekdays    Weekend

**Time of Day**

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

This stage will only run within the following windows (all times in PST):

From  :  to  :

**MANUAL JUDGMENT**

## MANUAL JUDGMENT DEFINED

Manual Judgments serve as a gate for your pipeline. You can add manual judgment 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

# NOTIFICATIONS WITH MANUAL JUDGMENT

**Stage Name** Manual Judgment Remove stage

**Depends On** Select...

## Manual Judgment Configuration

**Instructions** please approve this pipeline

**Propagate Authentication**

Judgment Inputs	Option	Actions
	<input type="button" value="Add judgment input"/>	

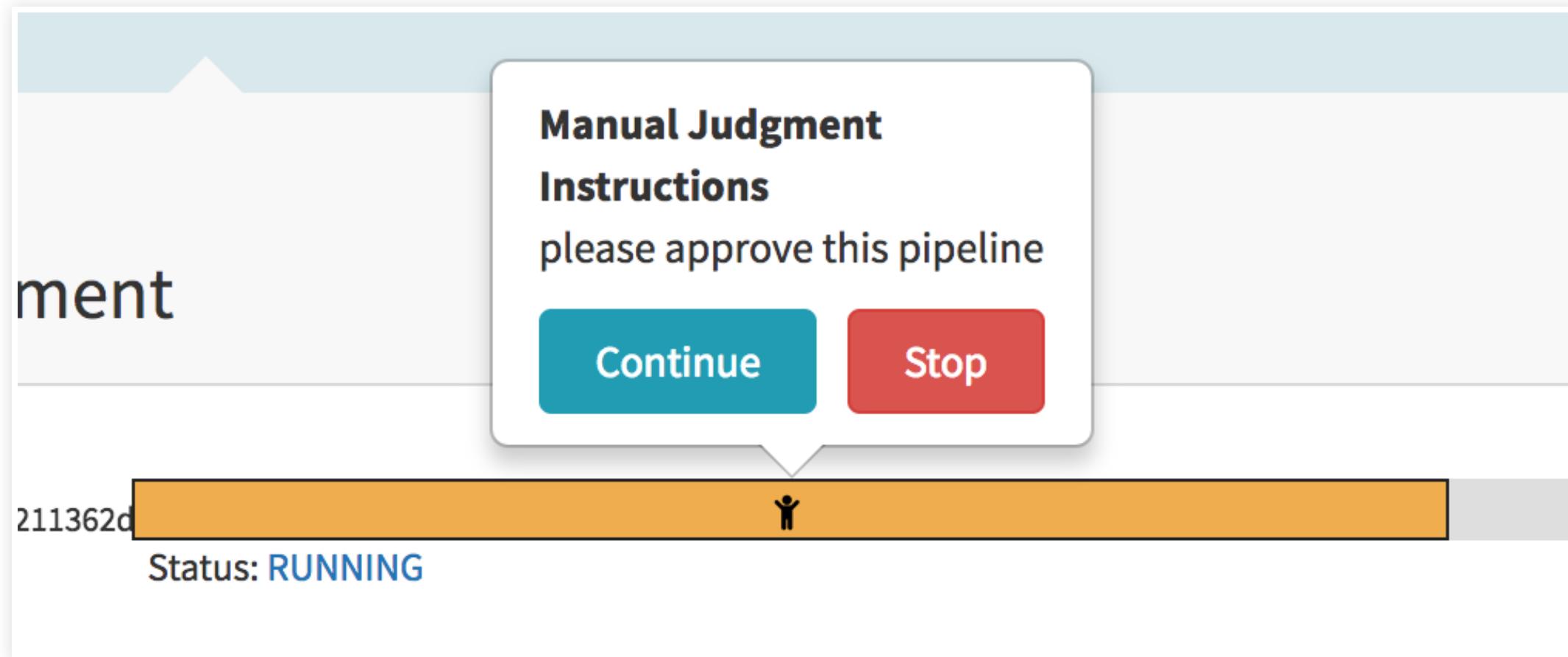
**Send Notifications**

Type	Details	Notify When	Actions
Email	clin@netflix.com	This stage is awaiting judgment	Edit Remove

## NOTIFICATIONS WITH MANUAL JUDGMENT

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

# WAITING FOR APPROVAL



# CHANGING PIPELINE ON SELECTED JUDGMENT

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

## TRIGGERING ON CHOICE

Once applied you can create a subsequent stage that will conditionally trigger on your choice, using Expressions (SpEL)



**Conditional on Expression** 

```
judgment("Manual Judgment").equals("rollback")}
```

# 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

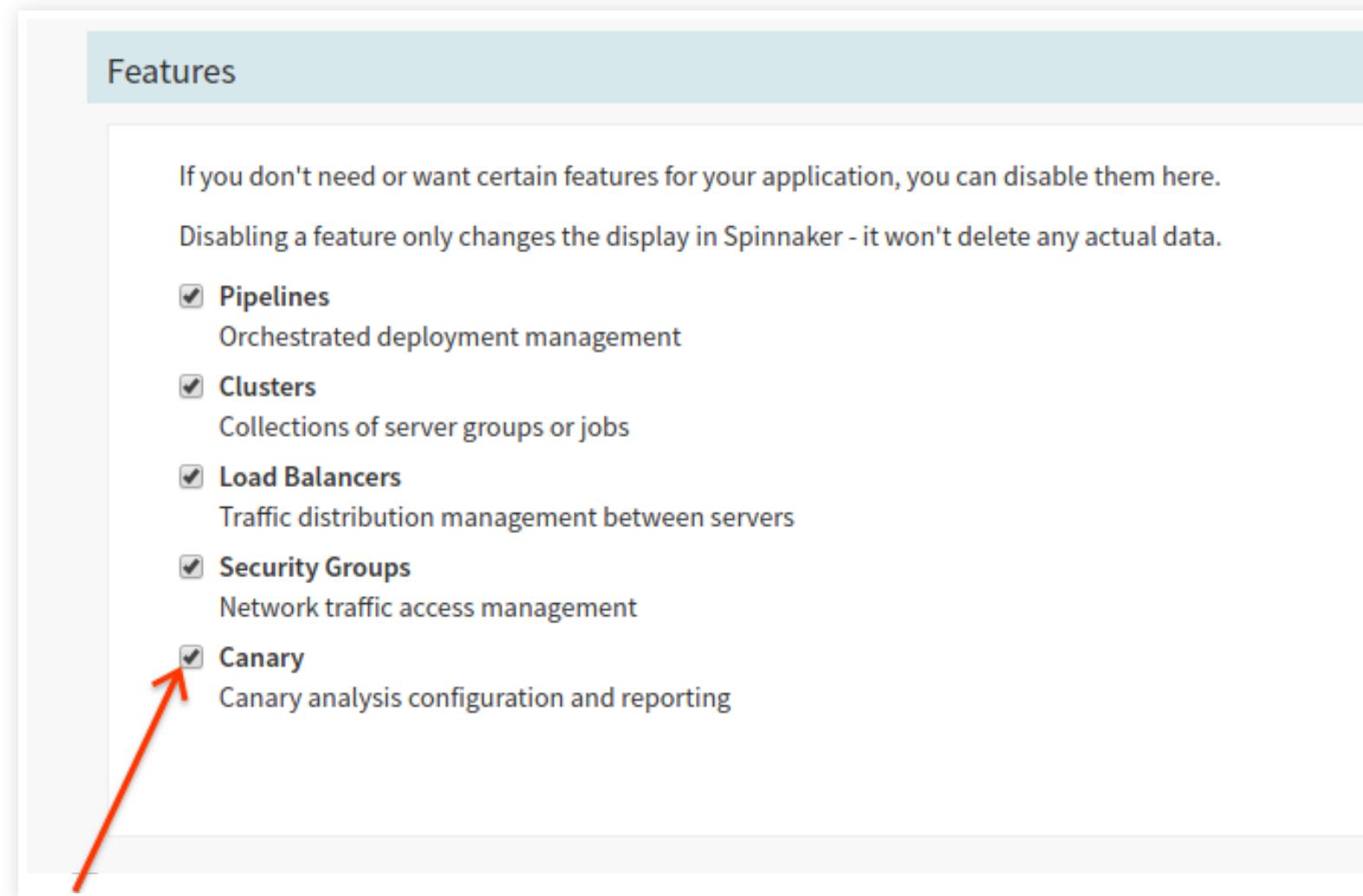
## WHAT YOU WILL BE ADDING INTO A CANARY CONFIGURATION

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



## 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

The screenshot shows the Spinnaker UI for managing an application named "myapp". The top navigation bar includes "SPINNAKER", "Search", "Projects", and "Applications". Below the application name, there are tabs for "DELIVERY", "INFRASTRUCTURE", and "TASKS". A dropdown menu is open over the "DELIVERY" tab, showing options: "PIPELINES", "CANARY CONFIGS", and "CANARY REPORTS". The "CANARY CONFIGS" option is highlighted with a cursor. On the left, a sidebar lists "APPLICATION ATTRIBUTES", "NOTIFICATIONS", "FEATURES", "LINKS", "CACHE MANAGEMENT", "TRAFFIC GUARDS", and "DELETE APPLICATION". On the right, the "myapp" application attributes are displayed:

Owner	duftler@google.com
Description	
Account(s)	my-gce-account
Cloud Provider(s)	aws, gce, appengine
Instance health	This application considers only cloud provider health when executing tasks.
Instance Port	79
User Data Format	This application requires legacy user data format.

At the bottom right of the attribute list, there is a link: "Edit Application Attributes".

## SELECTING YOUR CONFIGURATION

- 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

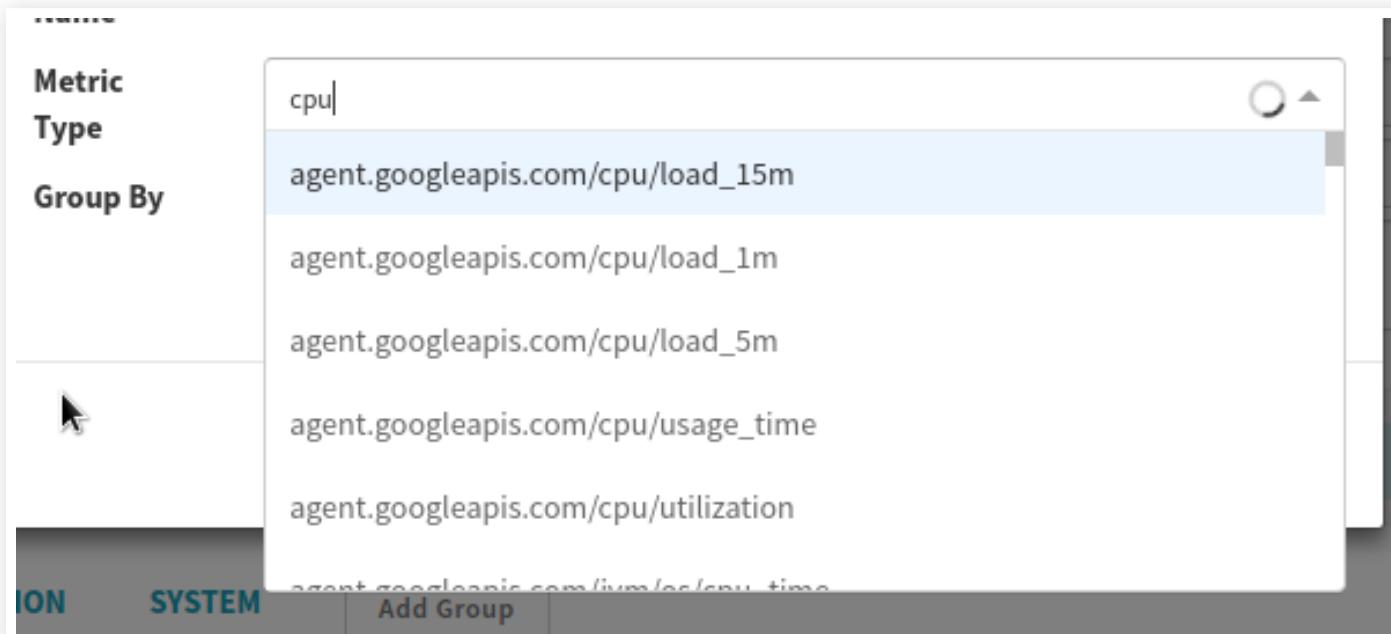
1. In the **Metrics** section, add **Metric**
2. Select the group to add this metric
3. Give the metric a name
4. Specify whether this metric fails when the value deviates too high or too low, or both compared to the baseline
5. Optionally, choose a filter template

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

# IDENTIFYING METRICS

- Identify the metric you wish to include in this configuration
- Optionally, if your provider 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`)

**i** 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.



## 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](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

**Canary Config**

---

**Analysis Type** Retrospective

**Analysis Config**

---

**Config Name** MySampleStackdriverCanaryConfig

**Interval** 180 minutes

**Analysis Type** Growing

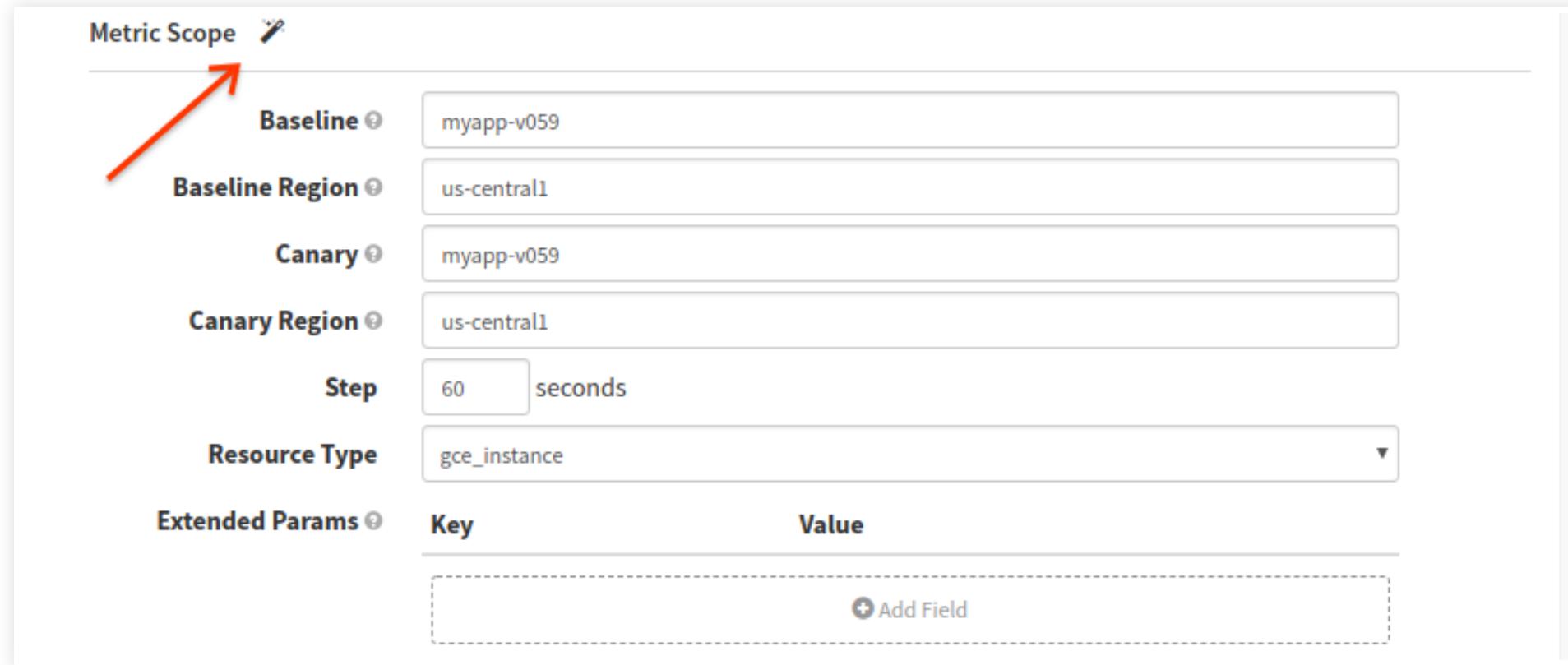
## DEFINING THE CANARY STAGE DETAILS

- 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

## DEFINING THE CANARY STAGE DETAILS CONTINUED

- 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.

# THE METRIC SCOPE



The screenshot shows a configuration interface for a 'Metric Scope'. At the top left, there is a section labeled 'Metric Scope' with a small icon. Below this, there are several configuration fields:

- Baseline**: myapp-v059
- Baseline Region**: us-central1
- Canary**: myapp-v059
- Canary Region**: us-central1
- Step**: 60 seconds
- Resource Type**: gce\_instance
- Extended Params**: A table with one row and two columns. The first column is 'Key' and the second is 'Value'. There is a button labeled '+ Add Field'.

- 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
- Metric Scope has a magic wand next to the name that automatically resolve to available resources

# DEFINING THE CANARY STAGE DETAILS CONTINUED

- 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.

# JUDGMENT

Judgment 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
- Judgment
  - Compare the metrics from the data collection
  - Renders a pass or fail
  - Can be configured with a "marginal" decision

# JUDGMENT PHASES

## 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

## DATA CLEANING

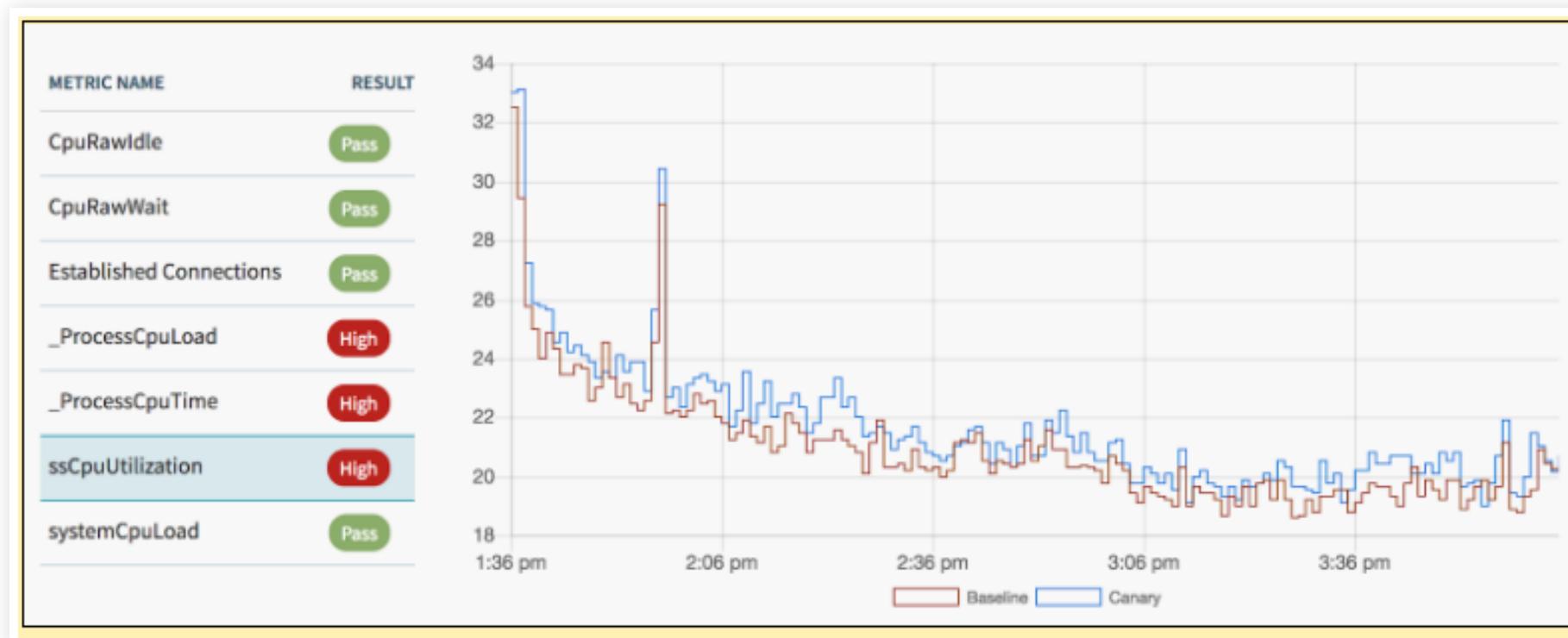
- Handling Missing Values
- Impute values for missing values
- Optionally, outliers can be removed

## 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

# 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

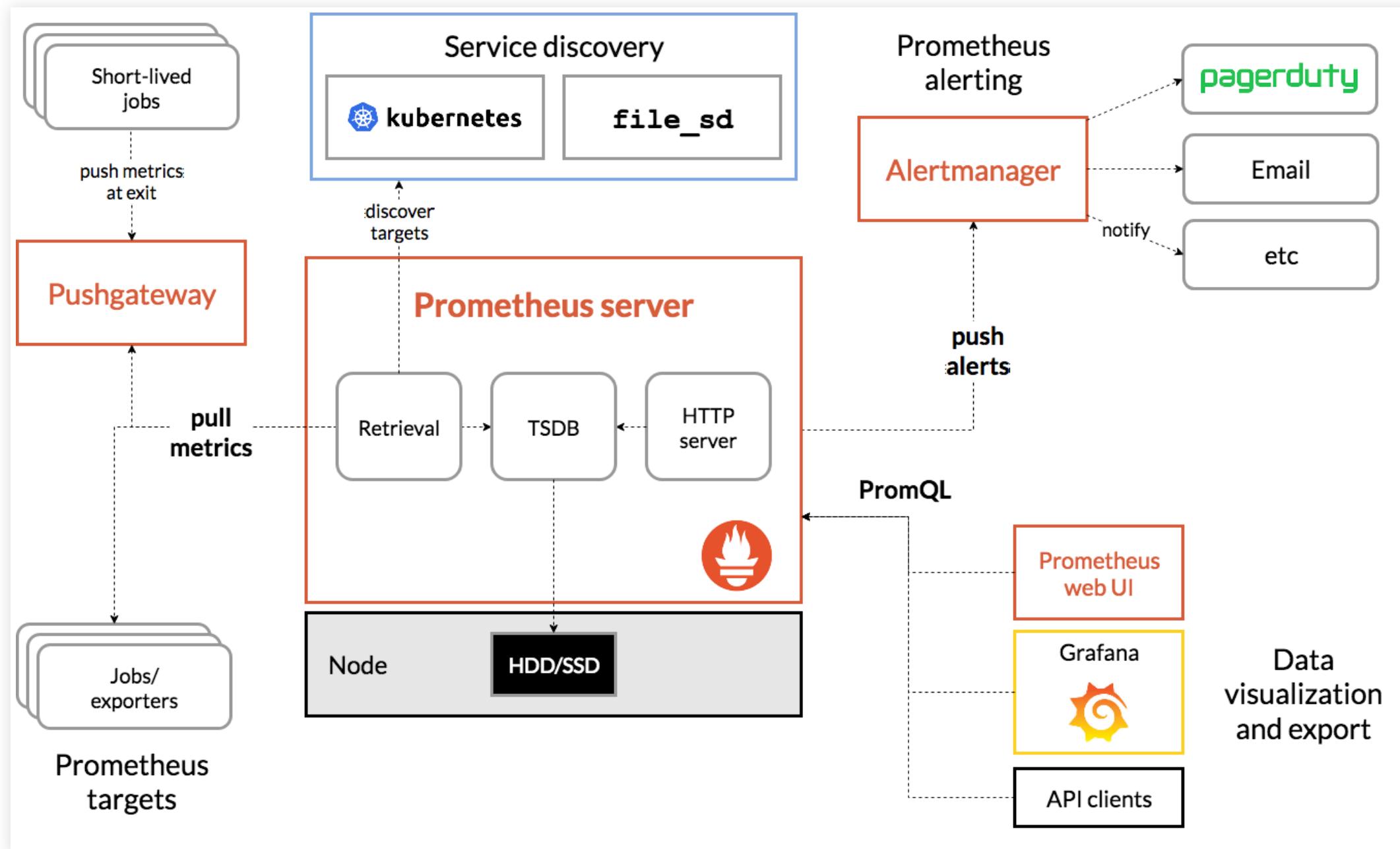


PROMETHEUS

# PROMETHEUS OVERVIEW

- 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

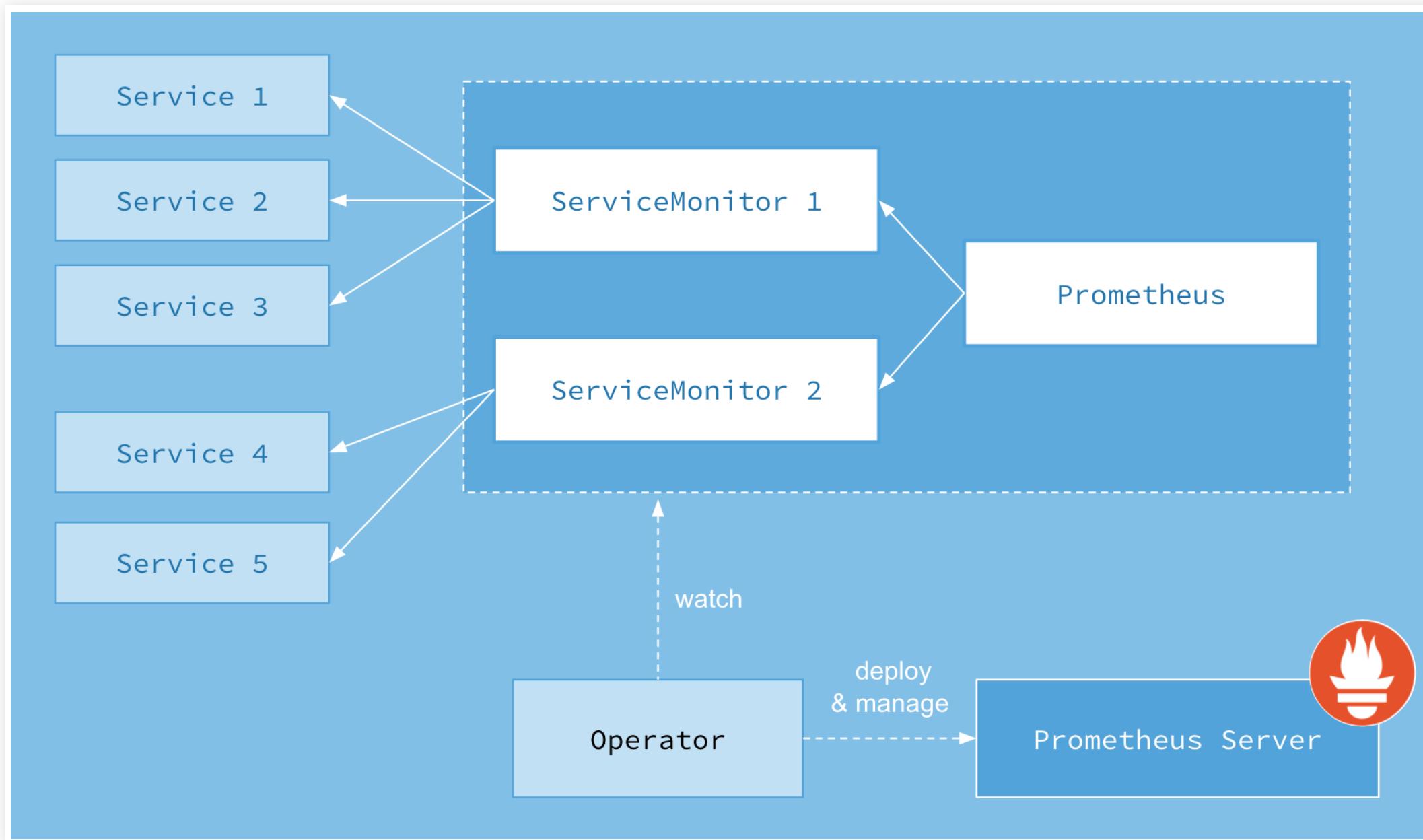
# PROMETHEUS ARCHITECTURE



## 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>)

# PROMETHEUS OPERATOR



# 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
```

# 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 4: ADVANCED FEATURES**

# AGENDA

# AGENDA

Our *mandatory* goals for this class:

- Become familiar with Spinnaker and it's role
- Understand the UI of Spinnaker
- Understand an Application, Pipeline, and Stage
- Purpose of Spinnaker
- How to get Spinnaker within Salesforce
- Helm and Baking
- Spinnaker Expression Language

## SOME OTHER ITEMS YOU'LL PICK UP

- Canary
- Understanding of Kubernetes
- How it works
- Debugging Tips

## **FINAL POLL**

Please take time to fill in this final poll: <https://forms.gle/JRCtiducAFnE9vko8>

# CONCLUSION

- Dev-ops is hard, requires a lot of knowledge in:
  - Command line programming
  - Kubernetes
  - Prometheus
  - Multiple Languages

You will need to be creative