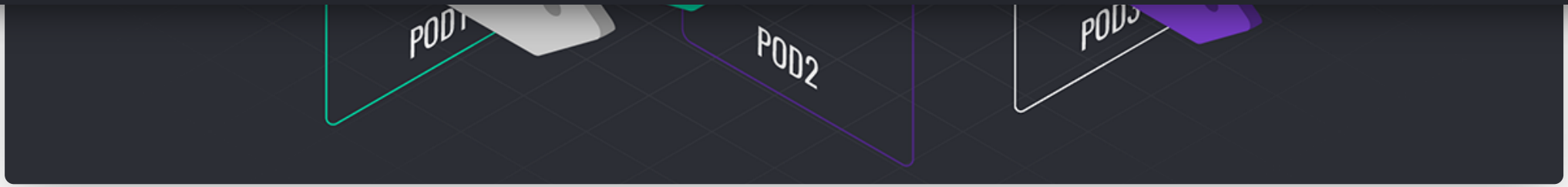
[KUBERNETES]

9 Best Practices and Examples for Working with Kubernetes Labels

Kubernetes labels allow DevOps teams to identify, select and operate on Kubernetes objects. In this blog post we outline 9 best practices for working with Kubernetes Labels and examples for Kubernetes label keys and values

HASHAM HAIDER

September 28, 2018 | 11 minute read



Kubernetes has a lot of nifty built-in features. Kubernetes labels are one of them. Kubernetes labels allow DevOps teams to identify Kubernetes objects and organize them into groups. One good use-case of this is grouping pods based on the application they belong to. Teams can also develop any number of labeling conventions including ones to group pods by environment, customer, team/owner or release version.

But that is just the start; Kubernetes labels pack a lot of other functionality. For example, it is possible to do bulk operations in Kubectl, by filtering a large number of resources, using Kubernetes labels. Kubernetes deployments also use labels to identify the pods that they manage. Similarly, Kubernetes services and replication controllers use labels to refer to a set of pods. Recommended Kubernetes labels also support querying and interoperability between Kubernetes tools.

1. Beware of the Syntax

Make sure that you get the syntax right. Below is an overview of the Kubernetes labeling syntax. There are 3 things you have to consider; label key (label prefix, label name) and label value.

- Label Key
  - Label prefix
    - Label prefix is optional
    - Label prefixes can be no longer than 253 characters
    - Label prefix must be a DNS subdomain
    - Label prefix can also be a series of DNS subdomains, separated by “.”
    - Label prefixes have to end with “/”
  - Label name
    - Label name is required
    - Label names can be up to 63 characters
    - Characters have to be alphanumeric characters
    - Label names can also include “-”, “_” and “.”
    - Label names have to begin and end with an alphanumeric character

- Characters have to be alphanumeric characters
- Label values can also include “-”, “_” and “.”
- Label values have to begin and end with an alphanumeric character

2. Label, Label, Label

The first rule of Kubernetes labeling is to actually do it, Always!

Creating or attaching Kubernetes labels should be second nature whenever new objects are created. Holding regular labeling audits for resources already in operation, to ensure they are labeled, is also good practice.

It might slow you down in whatever you are currently creating, but believe me: Taking the time and labeling everything as you work on the project will pay off later.

Labeling resources and objects will help you avoid Kubernetes production pain down the line. It will also make it easier to do operations in bulk on Kubernetes objects. One example of this

Create a pod with labels “environment=staging” and “team=kube-team” in Kubectl:

```
apiVersion: v1
kind: Pod
metadata:
 name: my-pod
 labels:
   environment: staging
   team: kube-team
spec:
 containers:
   - name: my-container
     image: "k8s.gcr.io/my-app:v0.1"
     resources:
       limits:
        cpu: 1
```

Check that the labels are applied in the newly created pod:

```
kubectl get pod my-pod --show-labels
```

Add a label to a pod using Kubectl:

Remove a label from a pod using Kubectl:

```
kubectl label pod my-pod versionID-
```

Update a label for a pod using Kubectl:

```
kubectl label --overwrite pods my-pod team=ops
```

3. Create company-wide consensus on labeling conventions

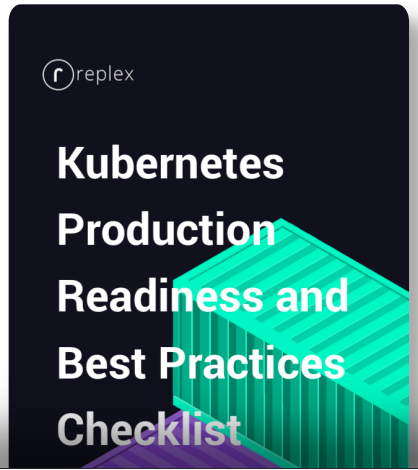
Since labels are meant to be meaningful and relevant for both Kubernetes itself and DevOps teams it is essential to develop consensus on labeling conventions. Sit down with your DevOps team and come-up with standard labeling conventions for Kubernetes resources. Also, ensure that conventions propagate across team and organizational boundaries.

4. Create a list of required labels

Some people just do not get with the times. For them, come up with a list of labels that have to be defined whenever a kubernetes pod is created. Start off small with 3-4 labels per object. You can follow Zalando's example, who require every Kubernetes resource to have application ID, version, owner, stage and release labels.

Below is a pod template with Kubernetes labels of application ID, version, stage, release and owner.

```
apiVersion: v1
kind: Pod
metadata:
 name: my-pod
 labels:
   application-ID: my-app
   version: version-nr
   stage: dev
   release: release-nr
   owner: team-kube
```



Download Checklist

5. Create a more extensive list of Kubernetes labels

Don't stop once you have a list of required labels. While you are on a roll keep it going. Create a more extensive list with labels that can provide more operability and context to your objects. Following is an exhaustive list of Kubernetes labels examples, along with their example values. Take your pick:

Label Example Key	Description	Label Example value
Application-ID/Application-name	The name of the application or its ID	my-awesome-app/app-nr-2345
Version-nr	The version number	ver-0.9

Stage/Phase	The stage or phase	Dev, staging, QA, Canary, Production
Release-nr	The release number	release-nr-2.0.1
Tier	Which tier the app belongs to	front-end/back-end
Customer-facing	Is the resource part of an app that is customer facing?	Yes/No
App-role	What roles does the app have	Cache/Web/Database/Auth
Project-ID	The associated project ID	my-project-276
Customer-ID	The customer ID for the resource	customer-id-29

every object that you create and have a shared prefix; app.kubernetes.io. Prefixes ensure that recommended labels do not get mixed up with custom labels defined by users.

Recommended labels include:

app.kubernetes.io/name

app.kubernetes.io/instance

app.kubernetes.io/version

app.kubernetes.io/component

app.kubernetes.io/part-of

app.kubernetes.io/managed-by

Below is an example pod with example values for recommended Kubernetes labels:

```
apiVersion: v1
kind: Pod
metadata:
 labels:
   app.kubernetes.io/name: my-pod
   app.kubernetes.io/instance: Auth-1a
   app.kubernetes.io/version: “2.0.1”
```

7. Monitor pre-populated Kubernetes labels

Kubernetes also generates a list of pre-populated labels that are attached either exclusively to nodes or both to nodes and persistent volumes. Pre-populated labels can be used in a number of ways e.g. when targeting workloads to specific instance types or spreading pods in a replication controller or service, across multiple cloud provider zones.

Node specific labels include:

`beta.kubernetes.io/arch`

`beta.kubernetes.io/os`

- Populated with the OS type e.g Linux

`kubernetes.io/hostname`

- Populated with the hostname

`beta.kubernetes.io/instance-type`

failure-domain.beta.kubernetes.io/region

- Populated with the cloud provider region e.g. eu-west-1

failure-domain.beta.kubernetes.io/zone

- Populated with the cloud provider zone e.g. eu-west-1b

8. Specify Cloudlabels and NodeLabels in Kops

Kops is a tool for deploying and managing highly available Kubernetes clusters. DevOps teams can specify two types of labels in Kops; CloudLabels and Nodelabels. Both these labels are specified at the InstanceGroup level. Kops InstanceGroups are similar to AWS autoscaling groups.

Once you apply CloudLabels to an InstanceGroup, they are applied to all the instances that are part of the InstanceGroup and show up on your AWS console as AWS tags. These can then be used to do cost allocation and chargeback.

Examples of CloudLabels that are recommended for cost allocation include application name, cost center, stack (test, prod), owner (team), customer and project. NodeLabels are the same

9. Differentiate between Kubernetes labels vs annotations

Kubernetes labels and annotations are both ways of adding metadata to Kubernetes objects. The similarities end there, however. Kubernetes labels allow you to identify, select and operate on Kubernetes objects. Annotations are non-identifying metadata and do none of these things.

Annotations allow you to add non-identifying metadata to Kubernetes objects. Examples include phone numbers of persons responsible for the object or tool information for debugging purposes. In short, annotations can hold any kind of information that is useful and can provide context to DevOps teams.

Conclusion

Kubernetes labels are a great way of identifying and organizing Kubernetes objects and resources. Kubernetes team leads and IT managers are best placed to develop and implement Kubernetes labeling plans. By ensuring labeling conventions are followed across teams, they can tame Kubernetes environments and prevent sprawl. Since Labels make it easier to operate on Kubernetes objects in bulk, adhering to labeling conventions will also make teams more productive in the long run.

To recap here is a list of Kubernetes labels best practices:

- ~~Monitor pre-defined Kubernetes labels~~
- Specify Cloudlabels and NodeLabels
- Differentiate between Kubernetes labels vs annotations
- Label, Label, Label
- Create company-wide consensus on labeling conventions
- Create a list of required Kubernetes labels
- Create a more extensive list of Kubernetes labels

Want to analyze Kubernetes costs and allocate them to individual teams and applications? Download the *Kubernetes Cost Analysis and Allocation Ebook* for AWS

nload Ebook

Hasham Haider

Fan of all things cloud, containers and micro-services!

Introduction to FinOps for Kubernetes: Challenges and Best Practices - Part III

Part 3 of our Introduction to FinOps for Kubernetes: Challenges and Best Practices article series, which outlines a comprehensive list of best practices aimed at implementing FinOps processes for cloud native Kubernetes environments.

July 12, 2021 | 11 min read **READ ARTICLE**

Introduction to FinOps for Kubernetes: Challenges and Best Practices - Part II

Part 2 of our Introduction to FinOps for Kubernetes: Challenges and Best Practices article series, which outlines a comprehensive list of best practices aimed at implementing FinOps processes for cloud native Kubernetes environments.

July 2, 2021 | 11 min read **READ ARTICLE**

The Kubernetes tools ecosystem falls broadly into two categories: tools that complement Kubernetes by extending its feature-set or making it easier to configure already existing native Kubernetes artefacts and ones that make it easier to manage and operate Kubernetes itself. In this blog post we will outline tools in both categories and provide detailed descriptions of each.

June 23, 2021 | 11 min read

READ ARTICLE

Want to Understand How Cloud Infrastructure Drives your Kubernetes Costs and Identify Optimization Opportunities?

Request a quick 20 minute demo to learn how best to leverage cloud VM's for your Kubernetes clusters and make infrastructure cost savings of up to 30% using Replex.io.

Contact Us

Technology

Overview
Book a demo

Solutions

Manage Kubernetes Costs
Govern Kubernetes Environments
Regain Control over Finance

Company

About Us
Contact
Impressum / Imprint
Privacy Policy
Press
Resources

Careers

Open Positions