Table of Contents

OpenShift 201 - Pipeline your DevOps

OpenShift Developer Workshop Labs

This application supports the following workshops at the BCDevOps / BC Developers Exchange / BC Gov CSI Lab:

- OpenShift 101 - Developer Operations
- OpenShift 201 - Pipeline your DevOps

Lab Setup

Prior to working on labs, the Platform Services team will have created 4 projects for each attendee:

- tooling
- dev
- production

Each attendee is provided their own dedicated project as a platform "tenant". In this way, attendees are not required to add customized names to their apps, as was the case in OpenShift101.

Lab Requirements

These labs will require access to the pathfinder production environment:

- Pathfinder Web Console

The environment can be accessed from a web browser such as Chrome or Firefox. For users that also leverage the CLI, the `oc` binary is available for download here. The current version of OpenShift is 3.11 and should be installed into your $PATH.

In addition, attendees require GIT locally installed and must have a 2FA device to support access to GitHub.

- To login with the oc utility:
  - Login to the Pathfinder Web Console with your GitHub ID
  - Navigate to the top right corner, select the drop-down from your username, and select `Copy Login Command`
  - Paste the copied command into your terminal of choice

```
oc login https://console.pathfinder.gov.bc.ca:8443 --token=[token]
```

Communication Resources

The lab instructors will open up a dedicated RocketChat channel for sharing of code, content, or discussions for the duration of this class. Please provide the instructors with your RocketChat user id at the same time as your GitHub ID when starting the workshop.

General OpenShift Components and Tasks

In these labs you will explore builds and deployments within the context of the Pathfinder OpenShift Platform. These labs will focus on a set of popular graphing and metrics application (Graphana and Prometheus).

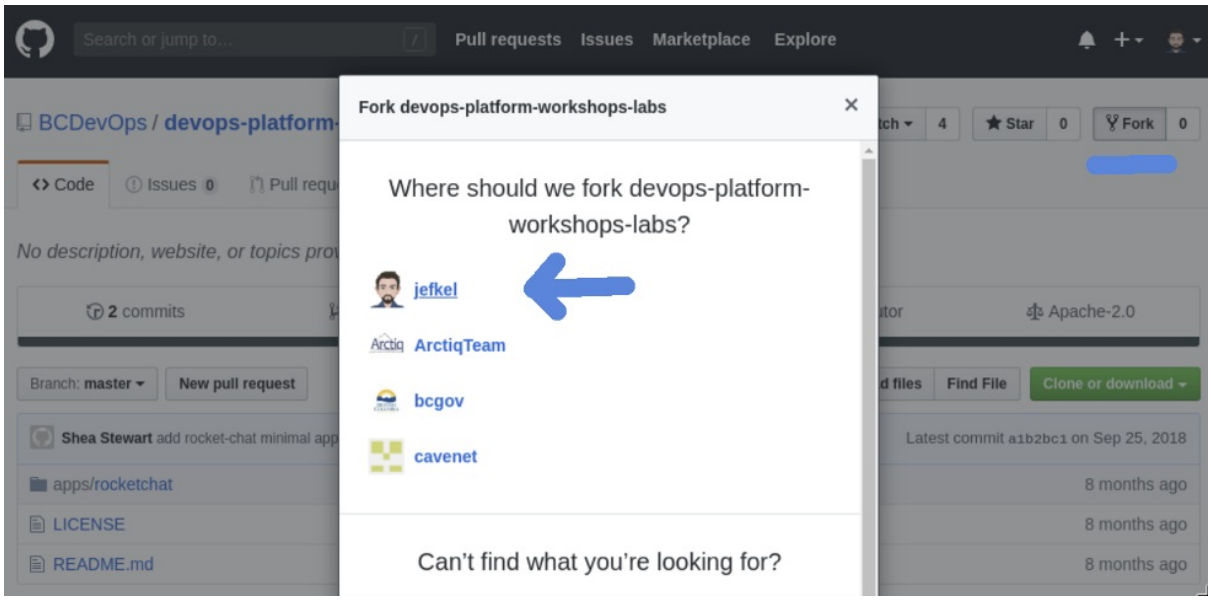
The content in the labs will get `slimmer` as labs progress. The goal here is that previous learnings will help you familiarize yourself with the Web Console or CLI utilities.

Git Setup

In this lab, you will create your own fork of the https://github.com/bcdevops/devops-platform-workshops-labs repository, add a branch for your work, and create a draft PR for your feature work.

Fork the lab repository

Navigate to the https://github.com/bcdevops/devops-platform-workshops-labs repository and click the "fork" button in the upper right corner. Choose your personal account as the location to fork the repository to.



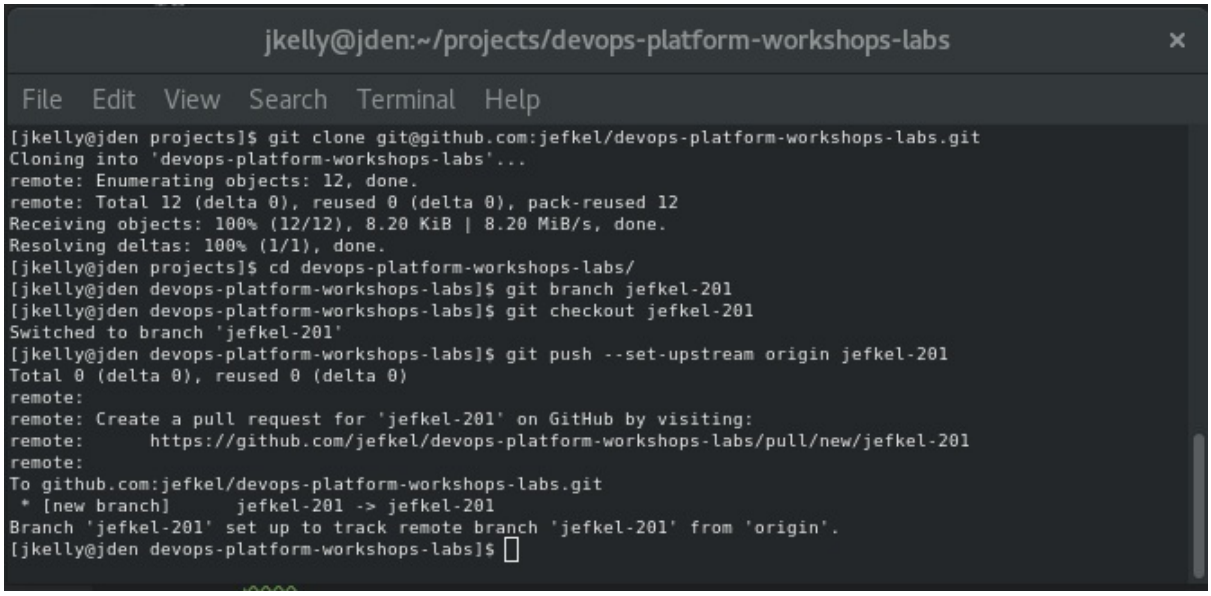
Create your branch with the CLI

Clone the repository and create your branch locally

```
git clone {URL to Repository}
cd devops-platform-workshops-labs
git branch [username]-201
git checkout [username]-201
```

Push your local branch to the upstream project

```
git push --set-upstream origin [username]-201
```



Create your branch with the GUI

Navigate to the public repository (https://github.com/{admin}/devops-platform-workshops-labs)

- Click "Branch:" and add your [username]-201 as the branch name.
- Click "Create branch: [username]-201"

Follow the above instructions to "Create your branch with the CLI" but exclude the `git branch` and `git push` commands to clone a local copy and switch to the already created [username]-201 branch

Create a draft pull request

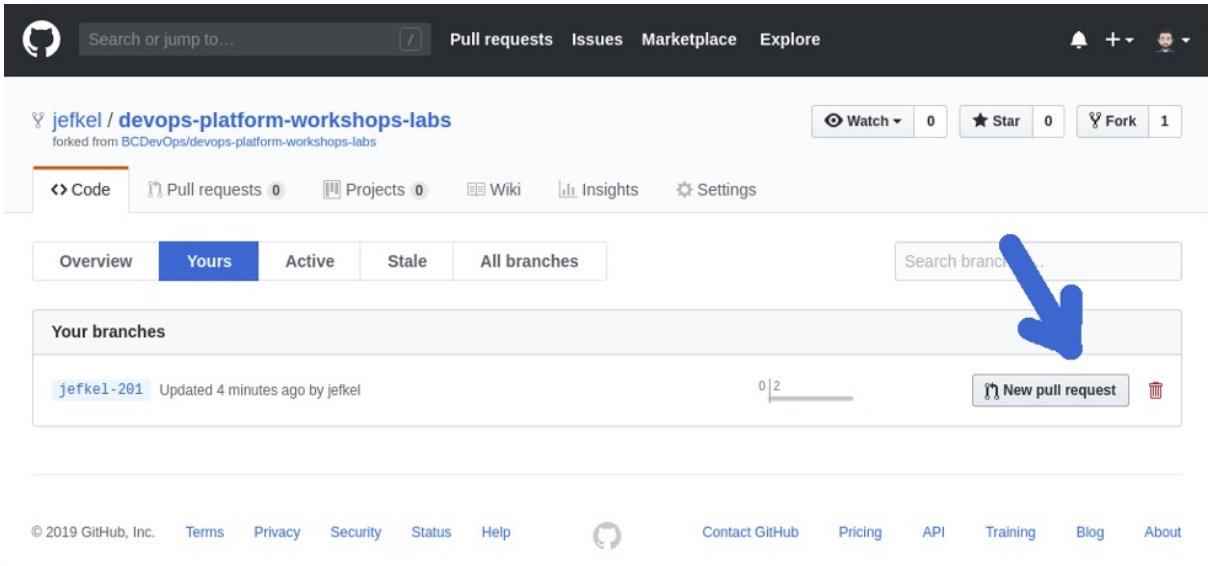
You won't be able to create a pull request until you have committed changes in your branch, so start off by creating a notes.md file and committing it to your branch:

```
echo "# My personal notes message" >> notes.md
git add notes.md
git commit -m "add notes location"
git push
```

- In your browser, navigate to the "branches" tab

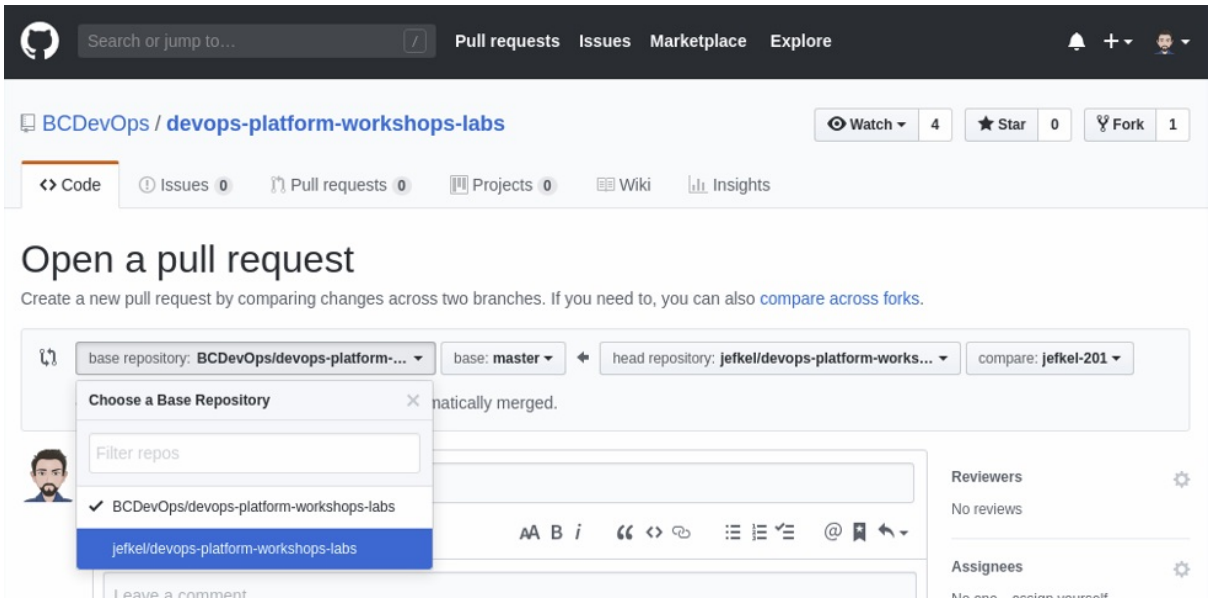
`quick tip: click on the "yours" link to filter out everyone else's branches`

- Click on the "New Pull Request" button beside your branch.



- Update the title and add a message (you can use Markdown, creativity is great!)

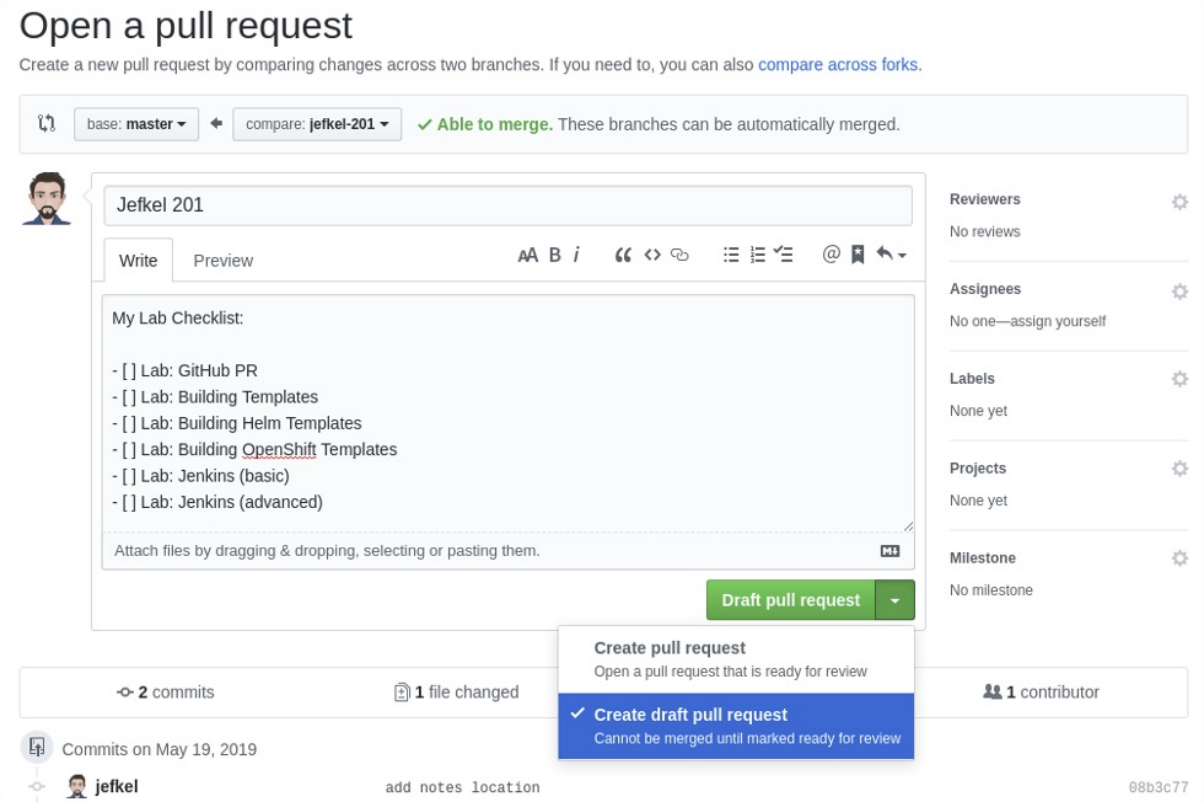
IMPORTANT Change the base repository to be {admin}/devops-platform-workshops-labs from the default of the parent (forked) repository. (We don't need to go creating a bunch of PR's against the main repo just now)



`you can cut and paste the following into the comment section to start a Lab checklist. Add more as you go!`

```
My Lab Checklist:

- [ ] Lab: GitHub PR
- [ ] Lab: Building Templates
- [ ] Lab: Building Helm Templates
- [ ] Lab: Building OpenShift Templates
- [ ] Lab: Jenkins (basic)
- [ ] Lab: Jenkins (advanced)
```

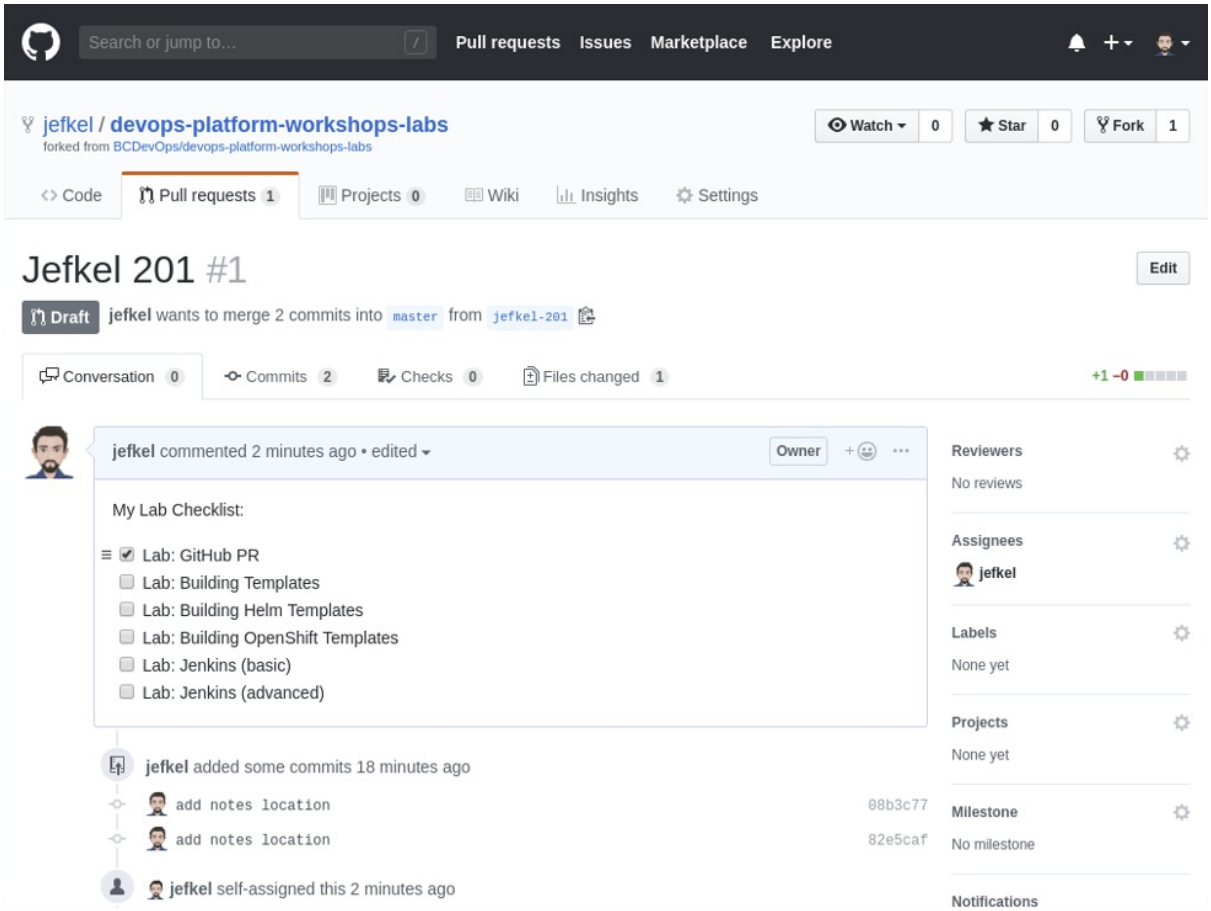
We are not ready for a code review just yet, so let's make sure we open a draft pull request and not the default pull request that's ready for review.



Click the green "Draft PR" button and you'll soon see your PR details.

Looking to the right, you should see an "Assignees" section. To quickly assign yourself click on the "assign yourself" link.

Now you have a place to get feedback on your work, and track your changes in an easily accessible location!


Search or jump to...
Pull requests
Issues
Marketplace
Explore
jefkel / devops-platform-workshops-labs
forked from BCDevOps/devops-platform-workshops-labs
Watch 0
Star 0
Fork 1
Code
Pull requests 1
Projects 0
Wiki
Insights
Settings
Jefkel 201 #1
Edit
Draft
jefkel wants to merge 2 commits into master from jefkel-201
Conversation 0
Commits 2
Checks 0
Files changed 1
+1 −0
jefkel commented 2 minutes ago • edited
Owner
My Lab Checklist:
Lab: GitHub PR
Lab: Building Templates
Lab: Building Helm Templates
Lab: Building OpenShift Templates
Lab: Jenkins (basic)
Lab: Jenkins (advanced)
jefkel added some commits 18 minutes ago
add notes location 08b3c77
add notes location 82e5caf
jefkel self-assigned this 2 minutes ago
Reviewers
No reviews
Assignees
jefkel
Labels
None yet
Projects
None yet
Milestone
No milestone
Notifications

OpenShift Templates

In this lab, explore existing template structures.

Exploring Templates

Templates can exist within a dedicated project or in a shared project space. The OpenShift catalog us one such example where shared templates are surfaced for use by any platform user.

- Log into the workshop `dev` namespace

```
oc login https://console.pathfinder.gov.bc.ca:8443 --token=[token]
oc project [workshop_project_set_name]-dev
```

- List any templates within your namespace

```
oc get templates
```

- List any templates within the OpenShift namespace, taking note of the number of parameters and objects

```
oc get templates -n openshift
```

- Pick a template and examine it's contents [feel free to choose any template that is interesting]

```
oc describe template postgresql-ephemeral -n openshift
```

Exporting a Template

- Export the template to examine the yaml structure

```
oc get template postgresql-ephemeral -n openshift  --export=true -o yaml > template
_contents.yaml
cat template_contents.yaml
```

Adding a Template

- Create the template inside of the dev namespace

```
oc apply -f template_contents.yaml
```

- Validate that the template exists in your namespace

```
oc get templates
```

Deploying from a Template

- Pass parameters to the template and review the objects

```
oc process postgresql-ephemeral DATABASE_SERVICE_NAME=myawesomedb -o yaml
```

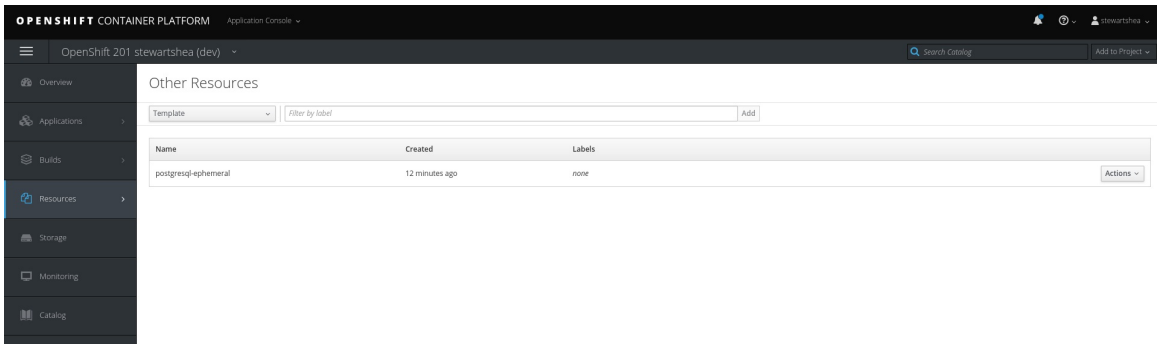
- Using an extended version of that command, create the objects in the dev namespace

```
oc process postgresql-ephemeral DATABASE_SERVICE_NAME=myawesomedb -o yaml | oc appl
y -f -
```

- Monitor the status of the object creation

```
oc status
```

- From the GUI, the template can also be viewed



Cleanup

- Remove all objects, including the template

```
oc get all,templates
oc delete all,templates --all
```

Using Helm Templates

The OpenShift cluster is not running Tiller, but Helm can still be used to help deploy packaged applications to projects/namespaces.

Obtain Helm

- Navigate to GitHub to obtain the appropriate version of Helm for your Operating System:
  - https://github.com/helm/helm/releases/latest
- Initialize helm in client-only mode

```
helm init --client-only
```

Deploy Prometheus into Dev Namespace

- Review the list of chart repositories and validate `https://kubernetes-charts.storage.googleapis.com` is available

```
helm repo  list
```

- Search for prometheus

```
helm search prometheus
```

- Fetch the prometheus chart and verify the tgz package is downloaded

```
helm fetch stable/prometheus
ls -lha  | grep prometheus
```

- Navitage to GitHub and review the available vales for this chart
  - https://github.com/helm/charts/tree/master/stable/prometheus#configuration
  - The prometheus chart has a lot of components enabled; this lab only requires the server component
- Use the helm client to generate the OpenShift artifacts in a file, only installing the `server` component

```
helm template ./prometheus-8.11.1.tgz \
  --name [username]-prometheus \
  --set \
server.persistentVolume.size=1Gi,\
server.name=[username]-prometheus,\
alertmanager.enabled=false,\
pushgateway.enabled=false,\
kubeStateMetrics.enabled=false,\
```

```
nodeExporter.enabled=false,\
serviceAccounts.server.create=false,\
rbac.create=false \
> [username]-prometheus.yaml
```

- Deploy the artifacts with `oc`

```
oc apply -f [username]-prometheus.yaml
```

- Review the logs and notice that prometheus is trying to scrape every OpenShift project/namespace
- Adjust the configMap
  - disable the full cluster scraping configuration by editing the manifest and **Delete lines 30 to 156**
  - Also delete the securityContext section as this is invalid for OpenShift and will cause the deployment to fail
  - edit the `kubernetes-pod` job to add only the current project/namespace; such as

```
- job_name: kubernetes-pods
  kubernetes_sd_configs:
  - role: pod
    namespaces:
        names:
          - [ PROJECT NAME]
  relabel_configs:
  - action: keep
    regex: true
    source_labels:
    - __meta_kubernetes_pod_annotation_prometheus_io_scrape
  - action: replace
    regex: (.+)
    source_labels:
    - __meta_kubernetes_pod_annotation_prometheus_io_path
    target_label: __metrics_path__
  - action: replace
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    source_labels:
    - __address__
    - __meta_kubernetes_pod_annotation_prometheus_io_port
    target_label: __address__
  - action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
  - action: replace
    source_labels:
    - __meta_kubernetes_namespace
    target_label: kubernetes_namespace
  - action: replace
    source_labels:
    - __meta_kubernetes_pod_name
```

```
    target_label: kubernetes_pod_name
```

- Apply the manifest with the changes

```
oc apply -f [username]-prometheus.yaml
```

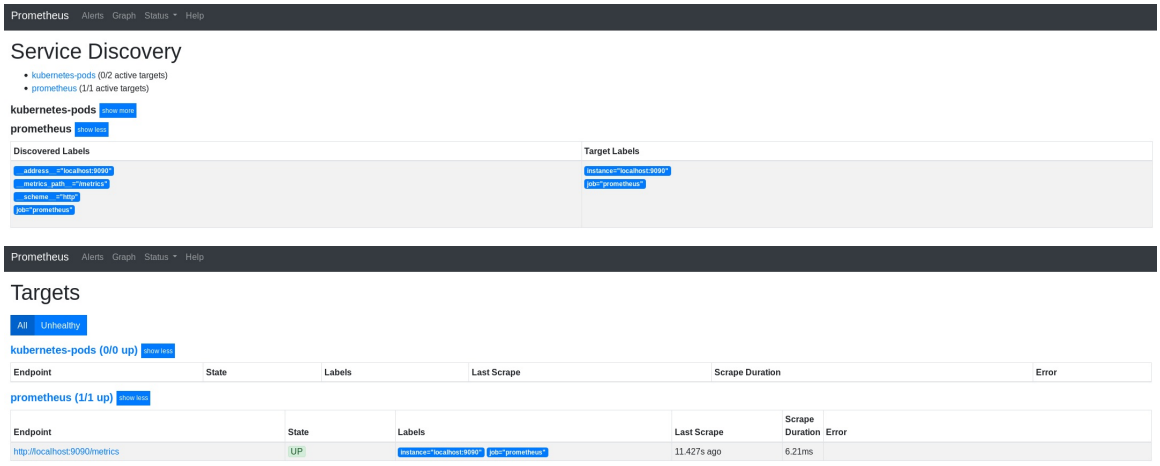
- Add a route to your prometheus service

```
oc get service
oc expose service [service name] --name prometheus -l app=prometheus
```

- Monitor the pod for the config change

```
caller=web.go:416 component=web msg="Start listening for connections" address=0.0.0
.0:9090
caller=main.go:655 msg="TSDB started"
caller=main.go:724 msg="Loading configuration file" filename=/etc/config/prometheus
.yml
caller=kubernetes.go:192 component="discovery manager scrape" discovery=k8s msg="Us
ing pod service account via in-cluster config"
caller=main.go:751 msg="Completed loading of configuration file" filename=/etc/conf
ig/prometheus.yml
caller=main.go:609 msg="Server is ready to receive web requests."
caller=klog.go:86 component=k8s_client_runtime func=Warningf msg="/app/discovery/ku
bernetes/kubernetes.go:283: watch of *v1.Pod ended with: too old resource version:
793031384 (793038695)"
caller=klog.go:86 component=k8s_client_runtime func=Warningf msg="/app/discovery/ku
bernetes/kubernetes.go:283: watch of *v1.Pod ended with: too old resource version:
793044753 (793047516)"
```

- Validate that the prometheus target itself is available



Deploy Another App with Helm

In this section, deploy Loki using the same process as the above. In this deployment, only deploy loki and no other components.

- Using Helm, create the OpenShift artifacts to deploy Loki with a minimal configuration

```
helm repo add loki https://grafana.github.io/loki/charts
helm fetch loki/loki
helm template ./loki-0.8.3.tgz --name=[username]-loki \
--set rbac.create=false,rbac.pspEnabled=false,serviceAccount.create=false \
> loki_template.yaml
oc apply -f loki_template.yaml
```

- Monitor the deployment of loki
- Identify and fix the deployment issue with the current template
- Explore the objects created by the loki helm chart

```
oc get all -l app=loki
```

Building OpenShift Templates

Up to this point you should have 2 apps running inside of your dev namespace, each deployed using the helm client `template` feature. In this lab, deploy additional components with the traditional OpenShift tools and then create the appropriate templates and store them in your `tools` project.

Deploy Grafana Using OpenShift Tools

- Deploy Grafana into the dev namespace using `new-app` and the existing Grafana docker image

```
oc new-app grafana/grafana:6.2.0 --name [username]-grafana
```

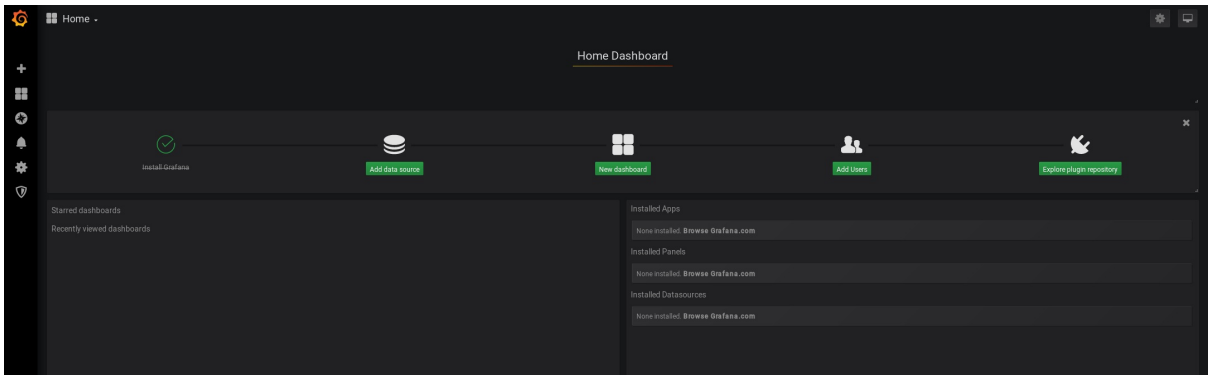
- Explore the output of the `new-app` command
- Explore the objects generated from the utility

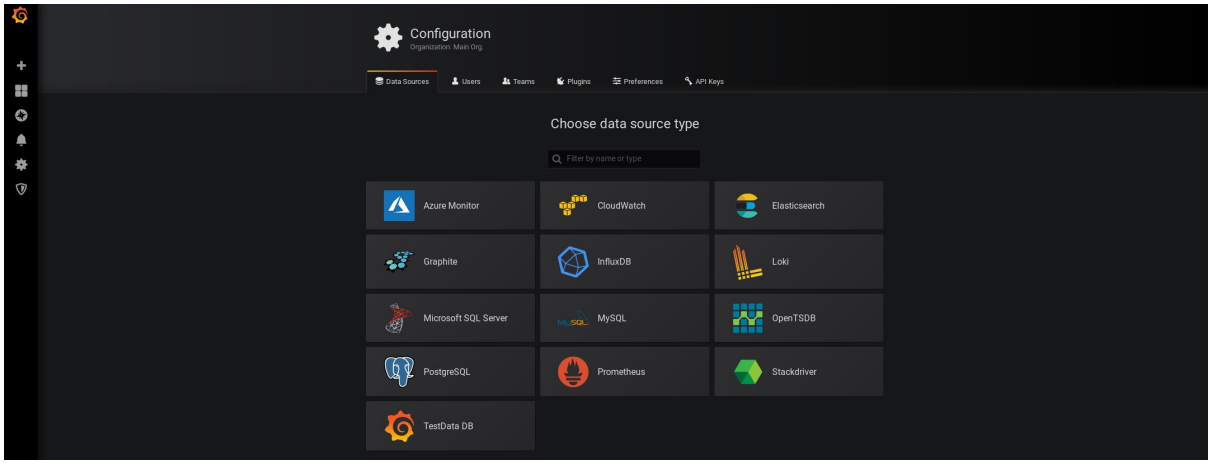
```
oc get all -l app=[username]-grafana
```

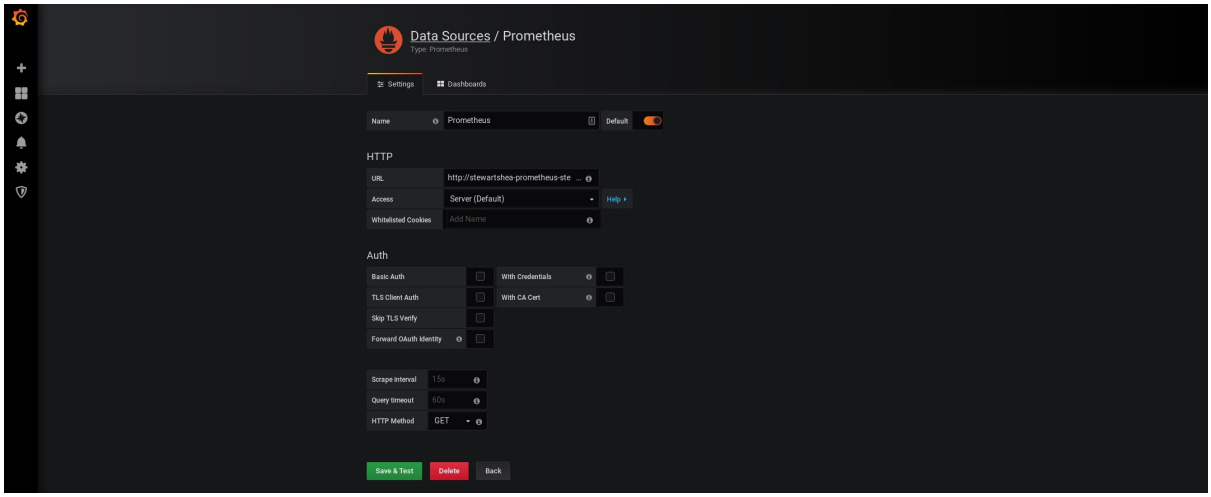
- Expose the Grafana service

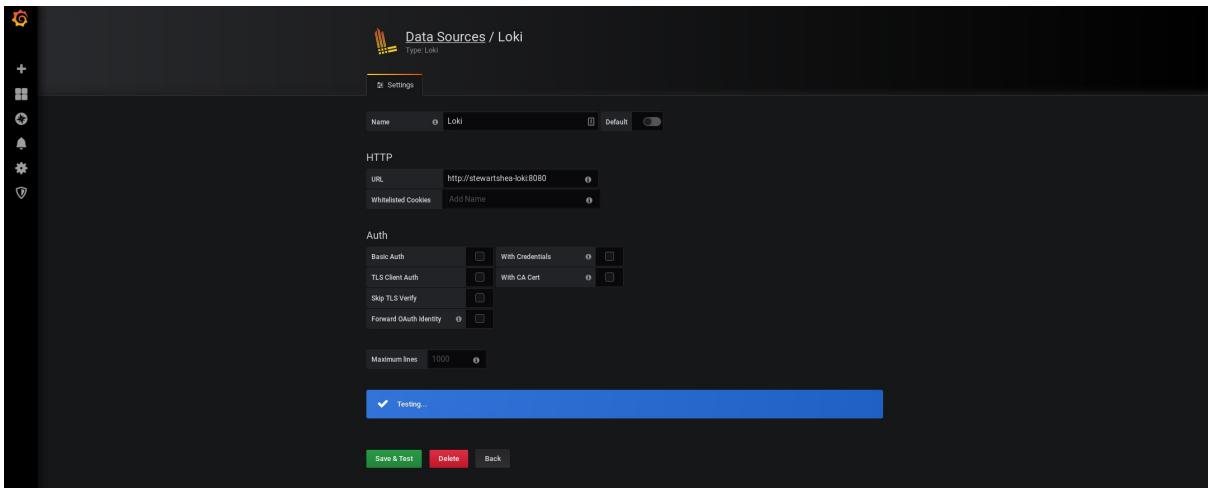
```
oc expose service [username]-grafana --name [username]-grafana
```

- Login to the interface with the default admin password (admin/admin)
- Create 2 datasources and a sample dashboard make sure your URLs are correct

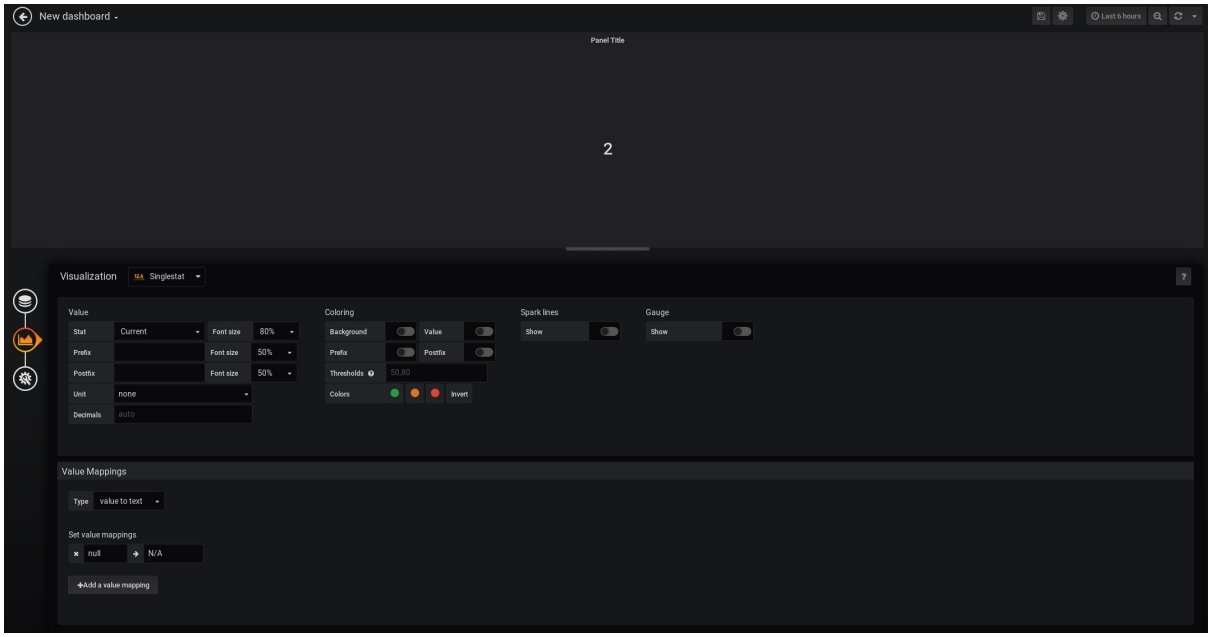


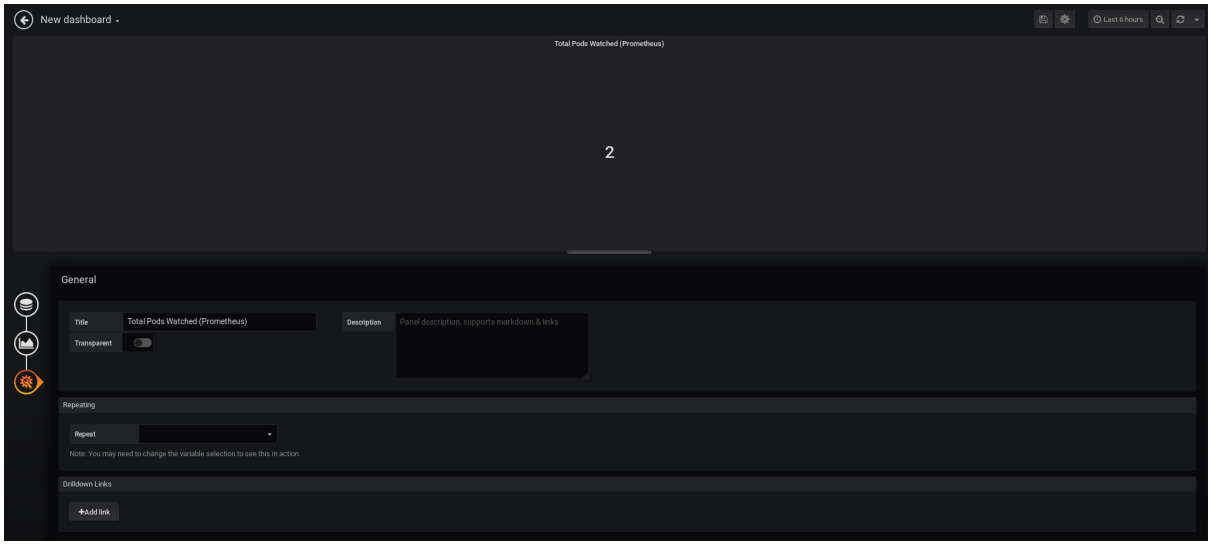

Configuration
Organization: Main Org.
Data Sources
Users
Teams
Plugins
Preferences
API Keys
Choose data source type
Filter by name or type
Azure Monitor
CloudWatch
Elasticsearch
Graphite
InfluxDB
Loki
Microsoft SQL Server
MySQL
OpenTSDB
PostgreSQL
Prometheus
Stackdriver
TestData DB

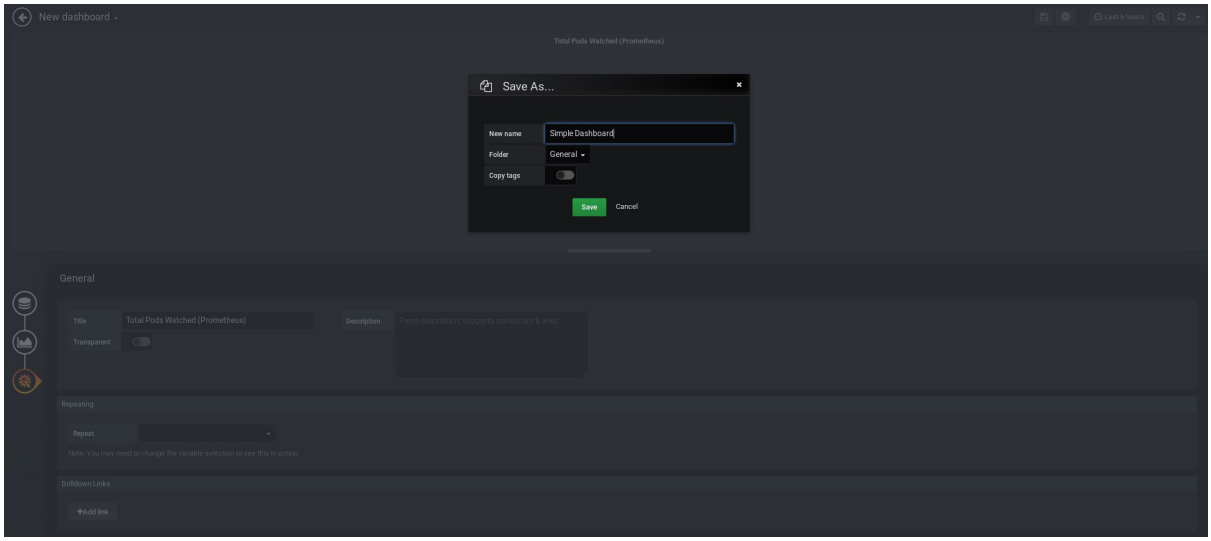

Data Sources / Prometheus
Type: Prometheus
Settings
Dashboards
Name
Prometheus
Default
HTTP
URL
http://stewartshea-prometheus-ste
Access
Server (Default)
Help
Whitelisted Cookies
Add Name
Auth
Basic Auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity
Scrape interval
15s
Query timeout
60s
HTTP Method
GET
Save & Test
Delete
Back


Data Sources / Loki
Type: Loki
Settings
Name
Loki
Default
HTTP
URL
http://stewartshea-loki:8080
Whitelisted Cookies
Add Name
Auth
Basic Auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity
Maximum lines
1000
Testing...
Save & Test
Delete
Back

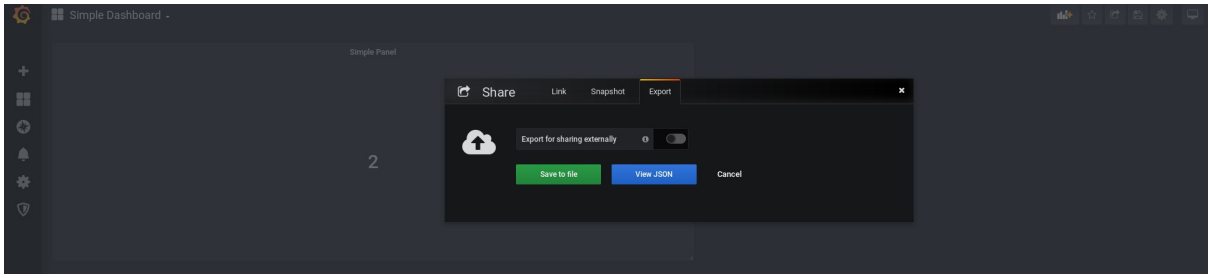

New dashboard
Last 6 hours
Panel Title
sum(up)
Query
default
Add Query
Query Inspector
A
sum(up)
Legend
legend format
Min step
Resolution
1/1
Format
Time series
Instant
Min time interval
Relative time
Time shift


New dashboard
Last 6 hours
Panel Title
2
Visualization
Singlestat
Value
Stat
Current
Font size
80%
Prefix
Font size
50%
Postfix
Font size
50%
Unit
none
Decimals
auto
Coloring
Background
Value
Prefix
Postfix
Thresholds
50,80
Colors
Invert
Spark lines
Show
Gauge
Show
Value Mappings
Type
value to text
Set value mappings
null
N/A
Add a value mapping


New dashboard
Last 6 hours
Total Pods Watched (Prometheus)
2
General
Title
Total Pods Watched (Prometheus)
Transparent
Description
Panel description, supports markdown & links
Repeating
Repeat
Note: You may need to change the variable selection to see this in action.
Drilldown Links
Add link



- Export the dashbaord configuration for later use



- Rollout a new version from the commandline

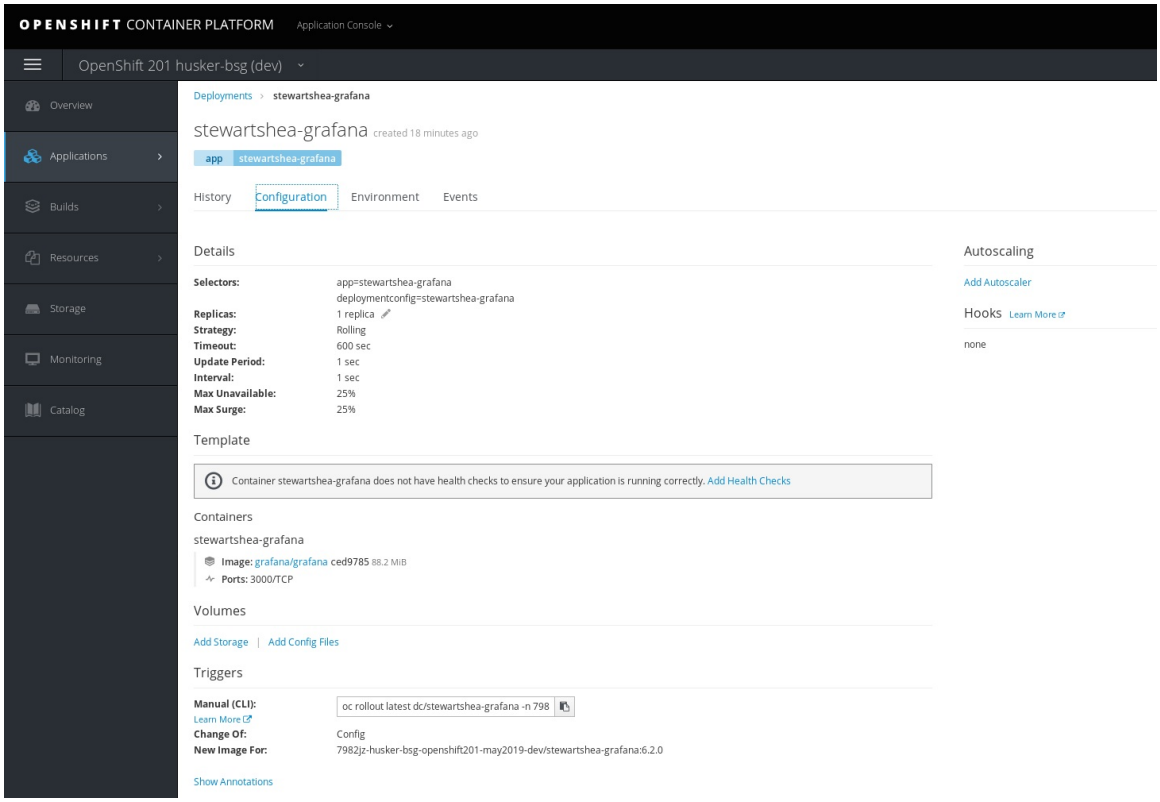
```
oc rollout latest [[username]]-grafana
```

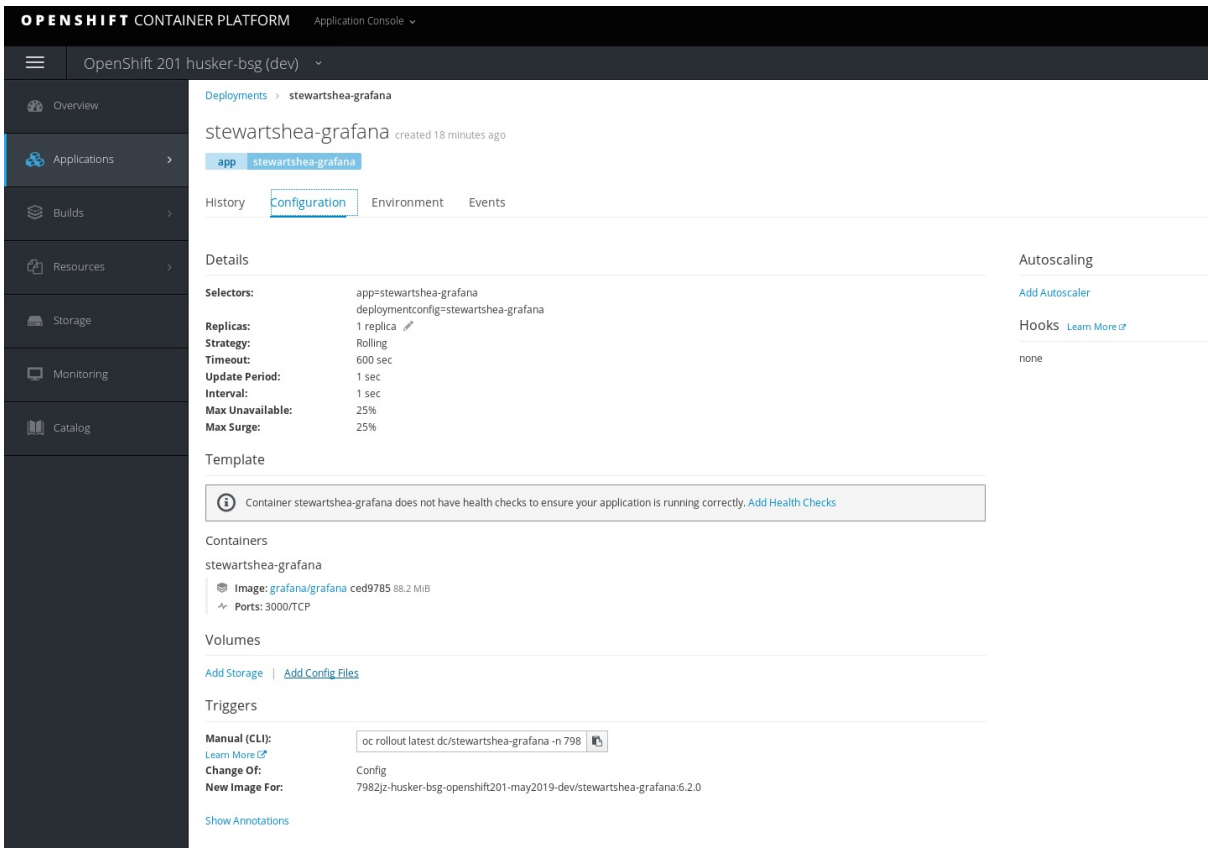
- Explore the UI again, noticing all configuration changes removed
  - The ephemeral state of this pod loses all configuration changes on redeploy
  - Ideally you don't want to use persistent storage for configuration data

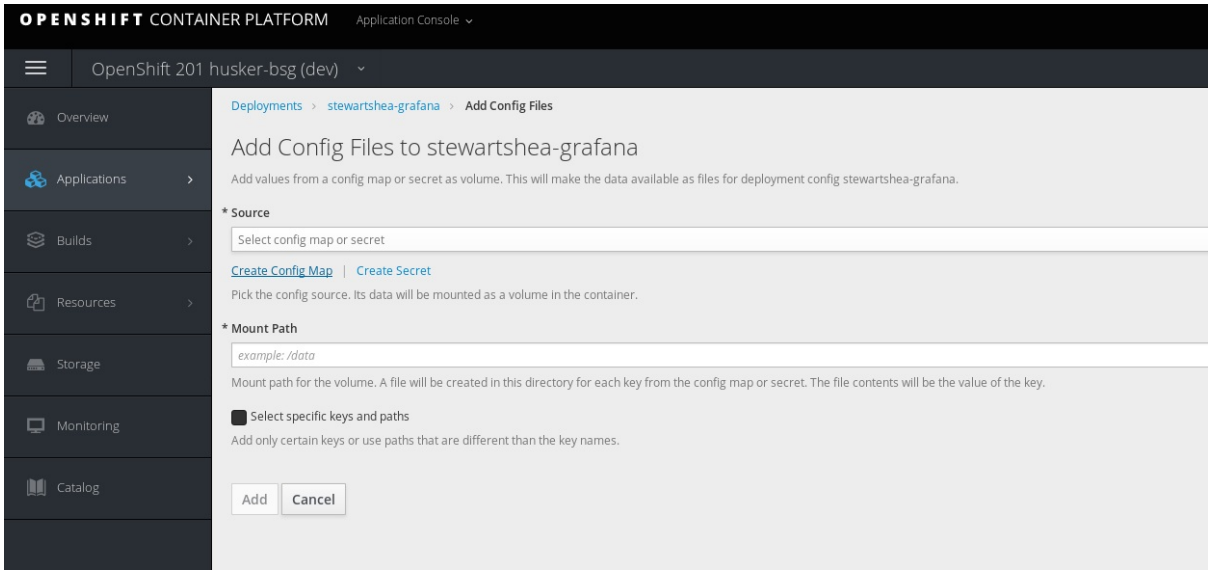
Configure Grafana Without State (ala. configMaps)

This section focuses on using a configMaps to define the configuration of Grafana. Once complete, Grafana instances can scale past 1 pod and will be deemed "stateless".

- Add a configMap to the grafana app for configuring the dashboard provider

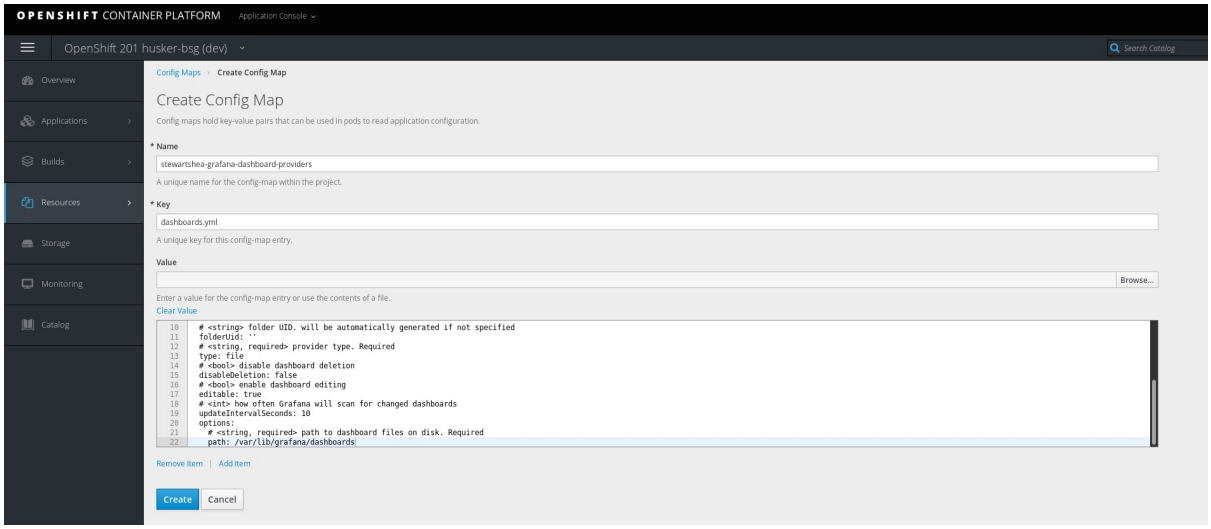

OPENSHIFT CONTAINER PLATFORM
Application Console
OpenShift 201 husker-bsg (dev)
Overview
Applications
Builds
Resources
Storage
Monitoring
Catalog
Deployments › stewartshea-grafana
stewartshea-grafana created 18 minutes ago
app stewartshea-grafana
History Configuration Environment Events
Details
Selectors: app=stewartshea-grafana
deploymentconfig=stewartshea-grafana
Replicas: 1 replica
Strategy: Rolling
Timeout: 600 sec
Update Period: 1 sec
Interval: 1 sec
Max Unavailable: 25%
Max Surge: 25%
Autoscaling
Add Autoscaler
Hooks Learn More
none
Template
Container stewartshea-grafana does not have health checks to ensure your application is running correctly. Add Health Checks
Containers
stewartshea-grafana
Image: grafana/grafana ced9785 88.2 MiB
Ports: 3000/TCP
Volumes
Add Storage | Add Config Files
Triggers
Manual (CLI):
Learn More
oc rollout latest dc/stewartshea-grafana -n 798
Change Of: Config
New Image For: 7982jz-husker-bsg-openshift201-may2019-dev/stewartshea-grafana:6.2.0
Show Annotations


OPENSHIFT CONTAINER PLATFORM
Application Console
OpenShift 201 husker-bsg (dev)
Overview
Applications
Builds
Resources
Storage
Monitoring
Catalog
Deployments › stewartshea-grafana
stewartshea-grafana created 18 minutes ago
app stewartshea-grafana
History Configuration Environment Events
Details
Selectors: app=stewartshea-grafana
deploymentconfig=stewartshea-grafana
Replicas: 1 replica
Strategy: Rolling
Timeout: 600 sec
Update Period: 1 sec
Interval: 1 sec
Max Unavailable: 25%
Max Surge: 25%
Autoscaling
Add Autoscaler
Hooks Learn More
none
Template
Container stewartshea-grafana does not have health checks to ensure your application is running correctly. Add Health Checks
Containers
stewartshea-grafana
Image: grafana/grafana ced9785 88.2 MiB
Ports: 3000/TCP
Volumes
Add Storage | Add Config Files
Triggers
Manual (CLI):
Learn More
oc rollout latest dc/stewartshea-grafana -n 798
Change Of: Config
New Image For: 7982jz-husker-bsg-openshift201-may2019-dev/stewartshea-grafana:6.2.0
Show Annotations

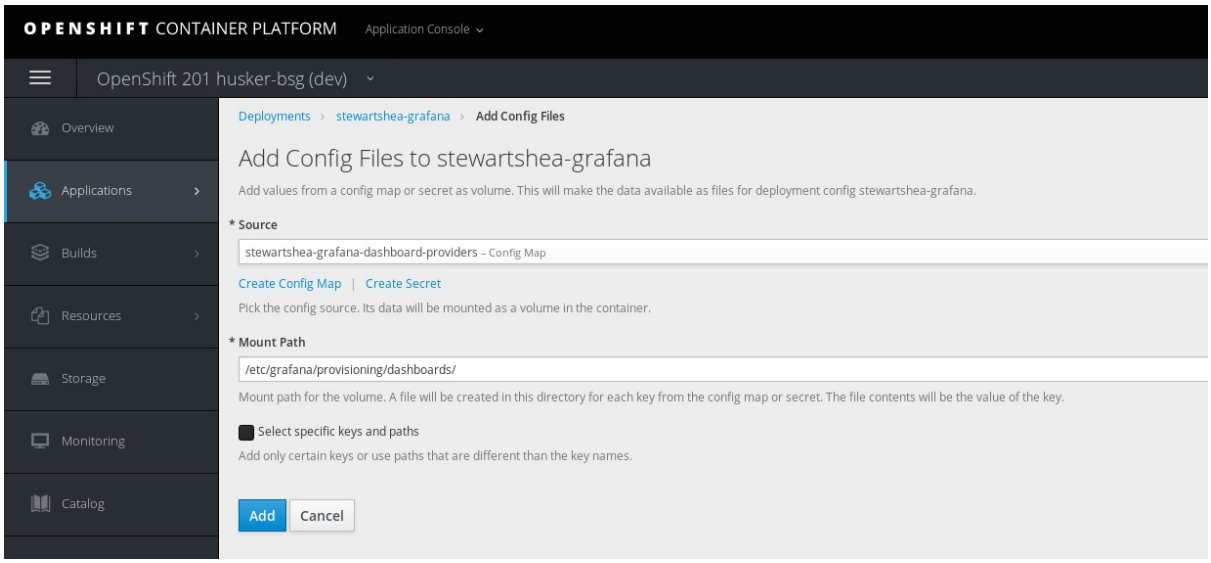


- Use the following content for the configMap

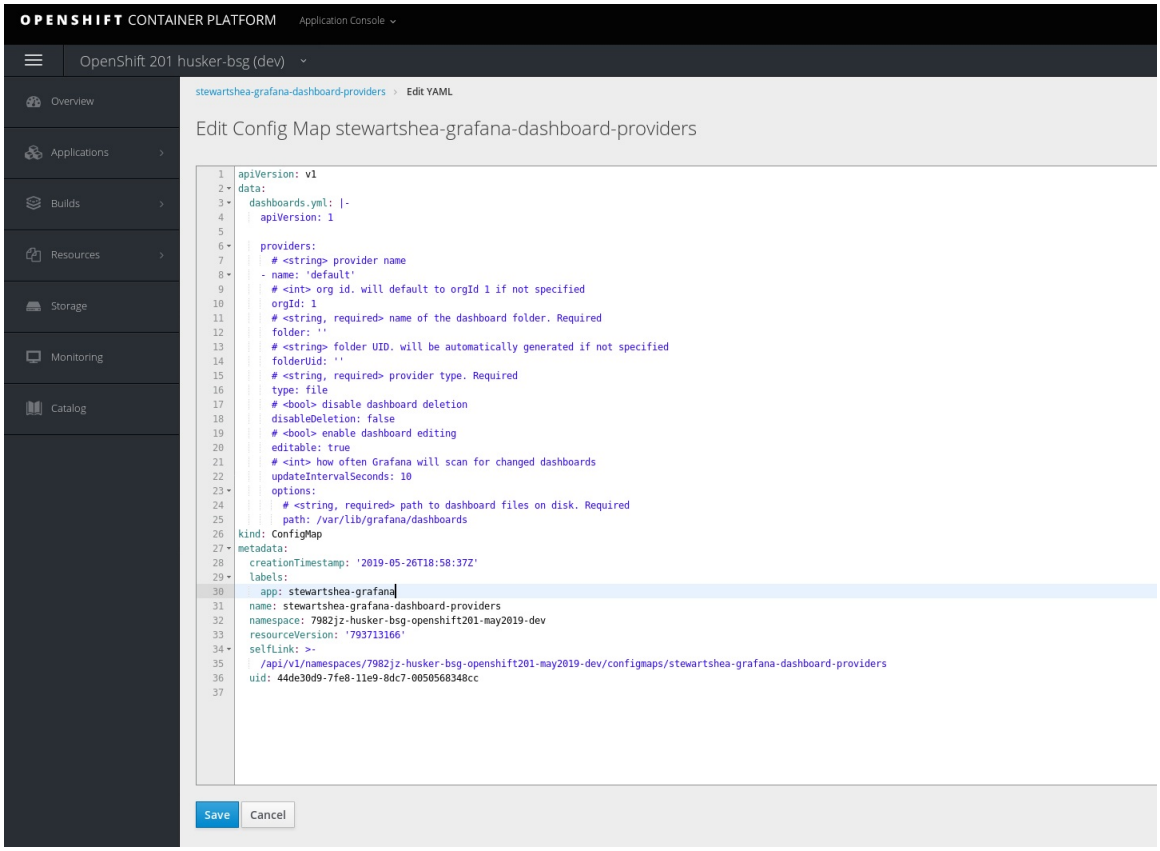
```
apiVersion: 1

providers:
  # <string> provider name
- name: 'default'
  # <int> org id. will default to orgId 1 if not specified
  orgId: 1
  # <string, required> name of the dashboard folder. Required
  folder: ''
  # <string> folder UID. will be automatically generated if not specified
  folderUid: ''
  # <string, required> provider type. Required
  type: file
  # <bool> disable dashboard deletion
  disableDeletion: false
  # <bool> enable dashboard editing
  editable: true
  # <int> how often Grafana will scan for changed dashboards
  updateIntervalSeconds: 10
  options:
    # <string, required> path to dashboard files on disk. Required
    path: /var/lib/grafana/dashboards
```



- Mount the configMap at `/etc/grafana/provisioning/dashboards`



- Add a label to the configMap to easily associate this with the app



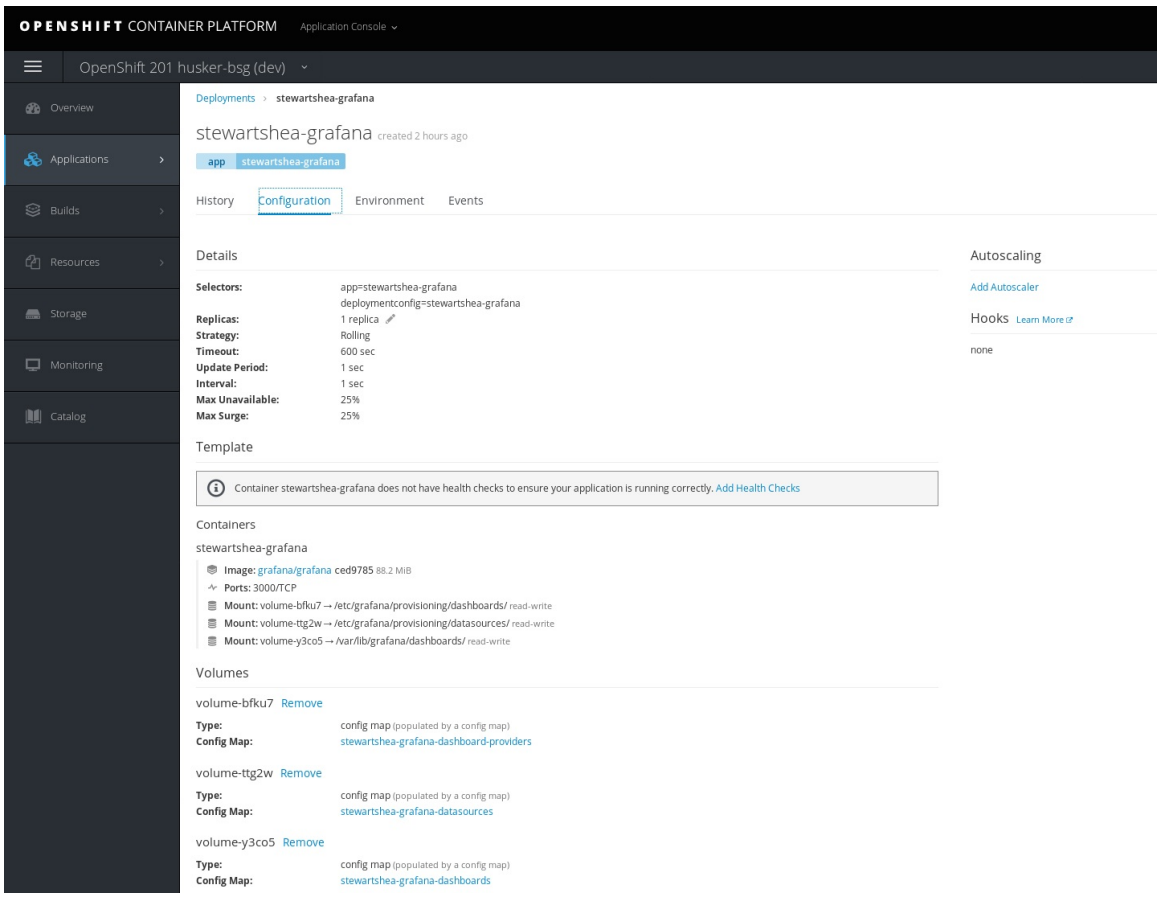
- Repeat the previous tasks for the **datasources** and **dashboards** configMaps
  - **datasources** mount path: `/etc/grafana/provisioning/datasources/`
  - **datasources** key: `datasources.yml`
  - **datasources content**:

```
# config file version
apiVersion: 1

datasources:
- name: Prometheus
  type: prometheus
  access: proxy
  orgId: 1
  url: http://prometheus:80
  isDefault: true
  version: 1
  editable: true
- name: Loki
  type: loki
  orgId: 1
  access: proxy
  url: http://loki:3100
  jsonData:
      maxLines: 1000
```

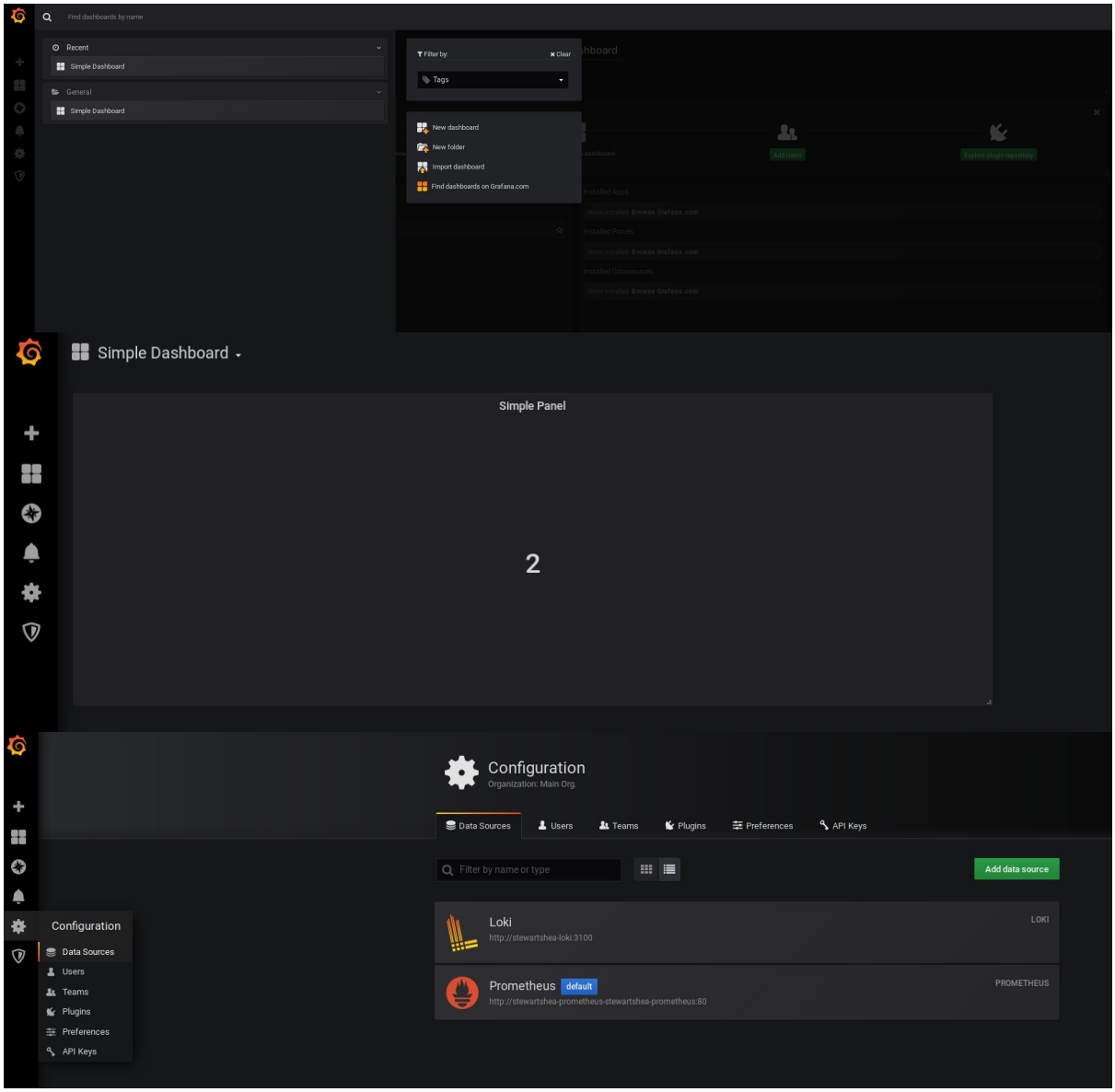
- **dashboards** mount path: `/var/lib/grafana/dashboards/`
- **dashboard** key: `simple_dashboard.json`
- **dashboard content**: json content from the dashboard export

Note Remember to add the appropriate labels to all configMaps

- When complete, the grafana deployment config should have 3 configmaps similar to the following:



- Redeploy the grafana pod, log in with default admin credentials, and validate that the dashboard and datasources are present



Create an OpenShift Deployment Artifact for Grafana

While the `new-app` command was quick and helpful to get an app deployed, it doesn't help in the management of the associated application components. In this section, create an OpenShift artifact (similar to the Helm lab), which can be used for easier management of the components.

- Review all artifacts associated with Grafana

```
oc get all -l app=[username]-grafana
```

- Expand the search to additional resources

```
oc get all,configmap -l app=[username]-grafana
```

- Export the components into a yaml file that works with `oc apply`

```
oc get all,configmap -l app=[username]-grafana --export -o yaml > grafana_template.
yaml
```

- Verify that the file works by removing all objects and deploying from the manifest

```
oc delete all,configmap -l app=[username]-grafana
```

- Attempt to reapply the object list

```
oc apply -f grafana_template.yaml
```

- Identify the types of objects that shouldn't belong in the template
- Remove the undesired objects from the template and reapply

```
oc apply -f grafana_template.yaml
```

- Continue to reapply the same file and review the output; it's clear there is still too much "specific" detail in the manfest
- Edit the manifest to make it more "generic"
    - Remove any references to:
        - last applied configuration annotations (just to keep the file clean)
        - creationTimestamp
        - generation
        - clusterIP
        - namespace
        - resourceVersion
        - selflink
        - uid
        - status
- Remove the objects again and apply the manifest more than once (this should succeeed with `unchanged` showing up when no changes are made)

```
oc delete all,configmap -l app=[username]-grafana
oc apply -f grafana_template.yaml
oc apply -f grafana_template.yaml
```

Create a Template for Grafana

With the above content added to a manifest with all items in a list, it can be parameterized into a template object. Using an existing template for reference (such as the one explored in previous labs), build a template for Grafana.

The following code can be used as an example:

```
apiVersion: template.openshift.io/v1
kind: Template
labels:
  template: grafana-template
message: |-
  The following service(s) have been created in your project: ${GRAFANA_SERVICE_NAM
E}.
  For more information about using this template, including OpenShift consideration
s, contact the rocketchat community.
metadata:
  annotations:
    description: |-
      Grafana Template for use in OpenShift 201 lab without persistent storage.

      WARNING: Any configuration stored will be lost upon pod destruction. ConfigMa
ps should be used for codified configuration.
    iconClass: icon-other-unknown
    openshift.io/display-name: grafana-openshift201
    openshift.io/documentation-url: https://github.com/bcdevops/devops-platform-wor
kshops
    openshift.io/long-description: This template provides a sample Grafana configur
ation for lab purposes.
    openshift.io/provider-display-name: BCDevOps
    openshift.io/support-url: https://github.com/bcdevops/
    tags: monitoring,grafana
  name: grafana-template
objects:
- apiVersion: v1
  kind: Service
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}
  spec:
    ports:
    - name: 3000-tcp
      port: 3000
      protocol: TCP
      targetPort: 3000
    selector:
      app: ${GRAFANA_SERVICE_NAME}
      deploymentconfig: ${GRAFANA_SERVICE_NAME}
    sessionAffinity: None
    type: ClusterIP
- apiVersion: apps.openshift.io/v1
  kind: DeploymentConfig
```

```
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}
  spec:
    replicas: 1
    revisionHistoryLimit: 10
    selector:
      app: ${GRAFANA_SERVICE_NAME}
      deploymentconfig: ${GRAFANA_SERVICE_NAME}
    strategy:
      activeDeadlineSeconds: 21600
      resources: {}
      rollingParams:
        intervalSeconds: 1
        maxSurge: 25%
        maxUnavailable: 25%
        timeoutSeconds: 600
        updatePeriodSeconds: 1
      type: Rolling
    template:
      metadata:
        annotations:
          openshift.io/generated-by: OpenShiftNewApp
        labels:
          app: ${GRAFANA_SERVICE_NAME}
          deploymentconfig: ${GRAFANA_SERVICE_NAME}
      spec:
        containers:
        - image: grafana/grafana:v6.2.0
          imagePullPolicy: IfNotPresent
          name: ${GRAFANA_SERVICE_NAME}
          ports:
          - containerPort: 3000
            protocol: TCP
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
          - mountPath: /etc/grafana/provisioning/dashboards/
            name: volume-bfku7
          - mountPath: /etc/grafana/provisioning/datasources/
            name: volume-ttg2w
          - mountPath: /var/lib/grafana/dashboards/
            name: volume-y3co5
        dnsPolicy: ClusterFirst
        restartPolicy: Always
        schedulerName: default-scheduler
        securityContext: {}
```

```
        terminationGracePeriodSeconds: 30
        volumes:
        - configMap:
            defaultMode: 420
            name: ${GRAFANA_SERVICE_NAME}-dashboard-providers
          name: volume-bfku7
        - configMap:
            defaultMode: 420
            name: ${GRAFANA_SERVICE_NAME}-datasources
          name: volume-ttg2w
        - configMap:
            defaultMode: 420
            name: ${GRAFANA_SERVICE_NAME}-dashboards
          name: volume-y3co5
    test: false
    triggers:
    - type: ConfigChange
    - imageChangeParams:
        automatic: true
        containerNames:
        - ${GRAFANA_SERVICE_NAME}
        from:
          kind: ImageStreamTag
          name: ${GRAFANA_SERVICE_NAME}:6.2.0
          namespace: ${NAMESPACE}
      type: ImageChange
- apiVersion: image.openshift.io/v1
  kind: ImageStream
  metadata:
    annotations:
      openshift.io/generated-by: OpenShiftNewApp
      openshift.io/image.dockerRepositoryCheck: 2019-05-26T17:34:32Z
    generation: 2
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}
  spec:
    lookupPolicy:
      local: false
    tags:
    - annotations:
        openshift.io/imported-from: grafana/grafana:6.2.0
      from:
        kind: DockerImage
        name: grafana/grafana:6.2.0
      generation: 2
      importPolicy: {}
      name: 6.2.0
      referencePolicy:
        type: Source
- apiVersion: route.openshift.io/v1
```

```
  kind: Route
  metadata:
    annotations:
      openshift.io/host.generated: "true"
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}
  spec:
    host: ${GRAFANA_SERVICE_NAME}-${NAMESPACE}.${ROUTE_SUBDOMAIN}
    port:
      targetPort: 3000-tcp
    to:
      kind: Service
      name: ${GRAFANA_SERVICE_NAME}
      weight: 100
    wildcardPolicy: None
- apiVersion: v1
  data:
    dashboards.yml: "apiVersion: 1\n\nproviders:\n  # <string> provider name\n- nam
e:
      'default'\n  # <int> org id. will default to orgId 1 if not specified\n  orgI
d:
      1\n  # <string, required> name of the dashboard folder. Required\n  folder:
      ''\n  # <string> folder UID. will be automatically generated if not specified
\n
      \ folderUid: ''\n  # <string, required> provider type. Required\n  type: file
\n
      \ # <bool> disable dashboard deletion\n  disableDeletion: false\n  # <bool>
      enable dashboard editing\n  editable: true\n  # <int> how often Grafana will
      scan for changed dashboards\n  updateIntervalSeconds: 10  \n  options:\n    #
      <string, required> path to dashboard files on disk. Required\n    path: /var/
lib/grafana/dashboards"
  kind: ConfigMap
  metadata:
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}-dashboard-providers
- apiVersion: v1
  data:
    simple_dashboard.json: |-
      {
        "annotations": {
          "list": [
            {
              "builtIn": 1,
              "datasource": "-- Grafana --",
              "enable": true,
              "hide": true,
              "iconColor": "rgba(0, 211, 255, 1)",
              "name": "Annotations & Alerts",
              "type": "dashboard"
```

```
            }
          ]
        },
        "editable": true,
        "gnetId": null,
        "graphTooltip": 0,
        "id": 1,
        "links": [],
        "panels": [
          {
            "cacheTimeout": null,
            "colorBackground": false,
            "colorValue": false,
            "colors": [
              "#299c46",
              "rgba(237, 129, 40, 0.89)",
              "#d44a3a"
            ],
            "format": "none",
            "gauge": {
              "maxValue": 100,
              "minValue": 0,
              "show": false,
              "thresholdLabels": false,
              "thresholdMarkers": true
            },
            "gridPos": {
              "h": 9,
              "w": 12,
              "x": 0,
              "y": 0
            },
            "id": 2,
            "interval": null,
            "links": [],
            "mappingType": 1,
            "mappingTypes": [
              {
                "name": "value to text",
                "value": 1
              },
              {
                "name": "range to text",
                "value": 2
              }
            ],
            "maxDataPoints": 100,
            "nullPointMode": "connected",
            "nullText": null,
            "options": {},
            "postfix": "",
```

```
            "postfixFontSize": "50%",
            "prefix": "",
            "prefixFontSize": "50%",
            "rangeMaps": [
              {
                "from": "null",
                "text": "N/A",
                "to": "null"
              }
            ],
            "sparkline": {
              "fillColor": "rgba(31, 118, 189, 0.18)",
              "full": false,
              "lineColor": "rgb(31, 120, 193)",
              "show": false
            },
            "tableColumn": "",
            "targets": [
              {
                "expr": "sum(up)\n",
                "format": "time_series",
                "intervalFactor": 1,
                "refId": "A"
              }
            ],
            "thresholds": "",
            "timeFrom": null,
            "timeShift": null,
            "title": "Simple Panel",
            "type": "singlestat",
            "valueFontSize": "80%",
            "valueMaps": [
              {
                "op": "=",
                "text": "N/A",
                "value": "null"
              }
            ],
            "valueName": "current"
          }
        ],
        "schemaVersion": 18,
        "style": "dark",
        "tags": [],
        "templating": {
          "list": []
        },
        "time": {
          "from": "now-6h",
          "to": "now"
        },
```

```
        "timepicker": {
          "refresh_intervals": [
            "5s",
            "10s",
            "30s",
            "1m",
            "5m",
            "15m",
            "30m",
            "1h",
            "2h",
            "1d"
          ],
          "time_options": [
            "5m",
            "15m",
            "1h",
            "6h",
            "12h",
            "24h",
            "2d",
            "7d",
            "30d"
          ]
        },
        "timezone": "",
        "title": "Simple Dashboard",
        "uid": "ybk9TjWZk",
        "version": 1
      }
  kind: ConfigMap
  metadata:
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}-dashboards
- apiVersion: v1
  data:
    datasource.yml: "# config file version\napiVersion: 1\n\ndatasources:\n- name:
      Prometheus\n  type: prometheus\n  access: proxy\n  orgId: 1\n  url: http://${
PROMETHEUS_SERVICE_NAME}:80
      \n  isDefault: true\n  version: 1\n  editable: true\n- name: Loki\n  type: lo
ki\n
      \ orgId: 1\n  access: proxy\n  url: http://${LOKI_SERVICE_NAME}:3100 \n  json
Data:\n
      \     maxLines: 1000"
  kind: ConfigMap
  metadata:
    labels:
      app: ${GRAFANA_SERVICE_NAME}
    name: ${GRAFANA_SERVICE_NAME}-datasources
parameters:
```

```
- description: Maximum amount of memory the container can use.
  displayName: Memory Limit
  name: MEMORY_LIMIT
  required: true
  value: 512Mi
- description: The namespace this templated is deployed into.
  displayName: Namespace
  name: NAMESPACE
  value: openshift
- description: The name of the OpenShift Service exposed for the database.
  displayName: Grafana Service Name
  name: GRAFANA_SERVICE_NAME
  required: true
  value: grafana
- description: The name of the Loki service to connect to.
  displayName: Loki Service Name
  name: LOKI_SERVICE_NAME
  required: true
  value: loki
- description: The name of the Prometheus service to connect to.
  displayName: Prometheus Service Name
  name: PROMETHEUS_SERVICE_NAME
  required: true
  value: prometheus
- description: Default route subdomain
  displayName: Route subdomain
  name: ROUTE_SUBDOMAIN
  required: true
  value: pathfinder.gov.bc.ca
```

- Install the template in the `dev` namespace

```
oc create -f [openshift_grafana_template_filename].yaml
```

- Deploy another version of Grafana with a different service name to validate functionality

```
oc process grafana-template \
    -p GRAFANA_SERVICE_NAME=[grafana app name] \
    -p LOKI_SERVICE_NAME=[loki-service] \
    -p PROMETHEUS_SERVICE_NAME=[prometheus-service] \
    -p ROUTE_SUBDOMAIN=pathfinder.gov.bc.ca \
    -p NAMESPACE=[dev namespace] \
    | oc apply -f -
```

Create a Template for Prometheus and Loki

Using the experience gained above, create 2 additional templates for Prometheus and Loki. This should not require any GUI/UI driven changes since the you already have the configuration artifacts from the `Helm` lab.

- Test each template in the dev namespace by passing a separate name for the service and ensuring the resources come up properly

```
oc process -f openshift_template_prometheus.yaml -p PROMETHEUS_SERVICE_NAME=prom2 -
p PROMETHEUS_PVC_SIZE=2Gi -p NAMESPACE=[namespace] | oc apply -f -
```

Commiting The Templates to SCM

Copy all template files into a `templates` directory in your forked repo and commit the changes to your branch.

Jenkins and Basic Pipeline Integration

In this lab you will create a basic Jenkins pipeline that is integrated with OpenShift.

Preparation

- Clean up all artifacts that were created in the dev namespace *Note* Repeat this command as necessary for each `app` name that was deployed

```
oc delete all,configmaps,templates,pvc,secrets -l [labels]
```

Creating the Initial Pipeline and Folder Structure

Create a simple folder structure in your forked repo for the openshift201 material, similar to the following:

```
├── apps
├── LICENSE
├── notes.md
├── openshift201
└── README.md
```

- Create a templates folder and add in the previously created templates, similar to the following:

```
├── openshift201
│   └── templates
│       ├── template_grafana_sample.yaml
│       ├── template_loki_sample.yaml
│       ├── template_prometheus_sample.yaml
│       └── template_sample_postgres_ephemeral.yaml
└── README.md
```

- Create a `.pipeline` folder under the openshift201 folder and add the `Jenkinsfile` into the `.pipline` folder.
- Place the following contents into the Jenkinsfile, changing the definitions to your specific environment names

```
#!/usr/bin/env groovy

//ENV Vars
def TOOLS_NAMESPACE = "[project_name]-tools"
def DEV_NAMESPACE = "[project_name]-dev"
def PROD_NAMESPACE = "[project_name]-prod"
```

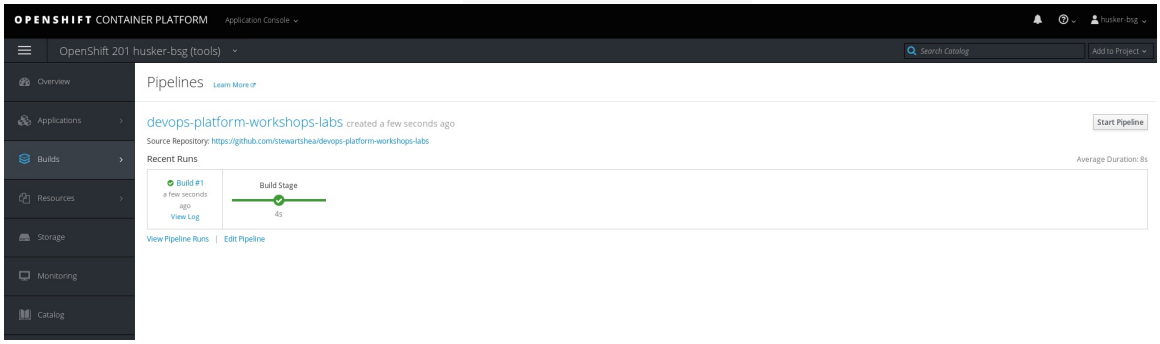
```
//Pipeline
node {
            stage ('Build Stage'){
                dir ('simple_pipeline') {
                    checkout scm
                    sh "echo HELLO WORLD! "
                    sh "ls -lha openshift201/templates"
                }
            }
    }
```

Building and Integrating the Initial Pipeline

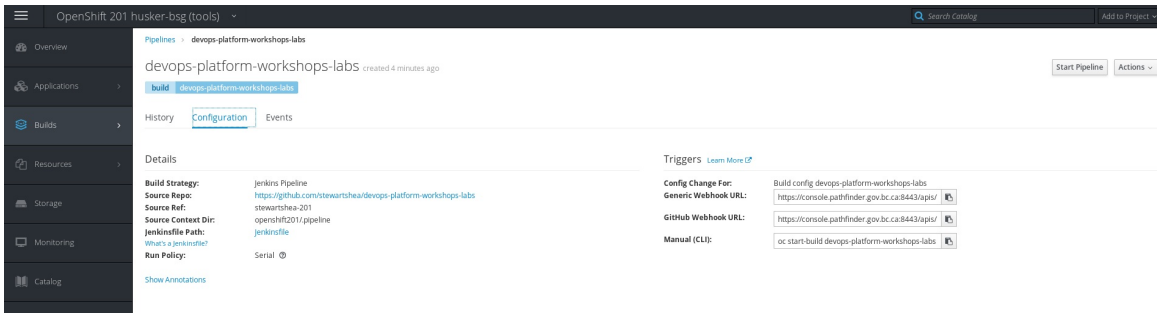
- From the `tools` project, create the Pipeline style buildConfig

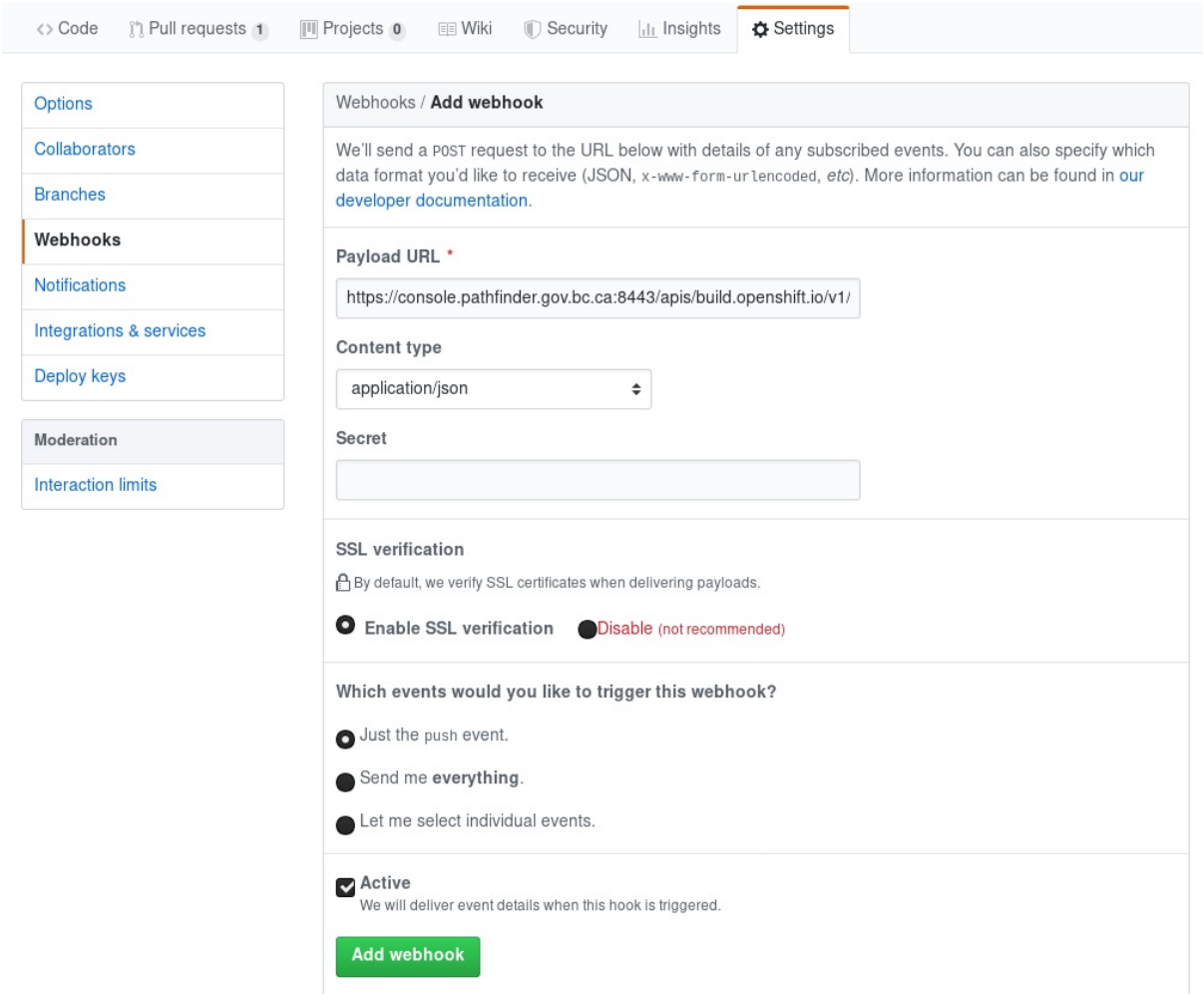
```
oc new-build https://github.com/[github_id]/devops-platform-workshops-labs#[branch-name] --context-dir=openshift201/.pipeline
```

- Notice that Jenkins will automatically start up if it wasn't previously available
- In the OpenShift web console, navigate to `Builds -> Pipelines`

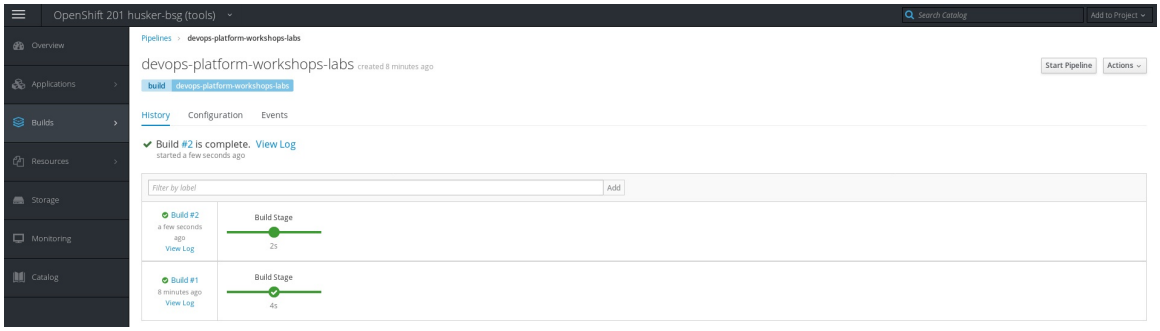


- Obtain the OpenShift supplied webhook and configure it on your forked repository





- Test the webhook by pushing adding any file and commit/push to the repo



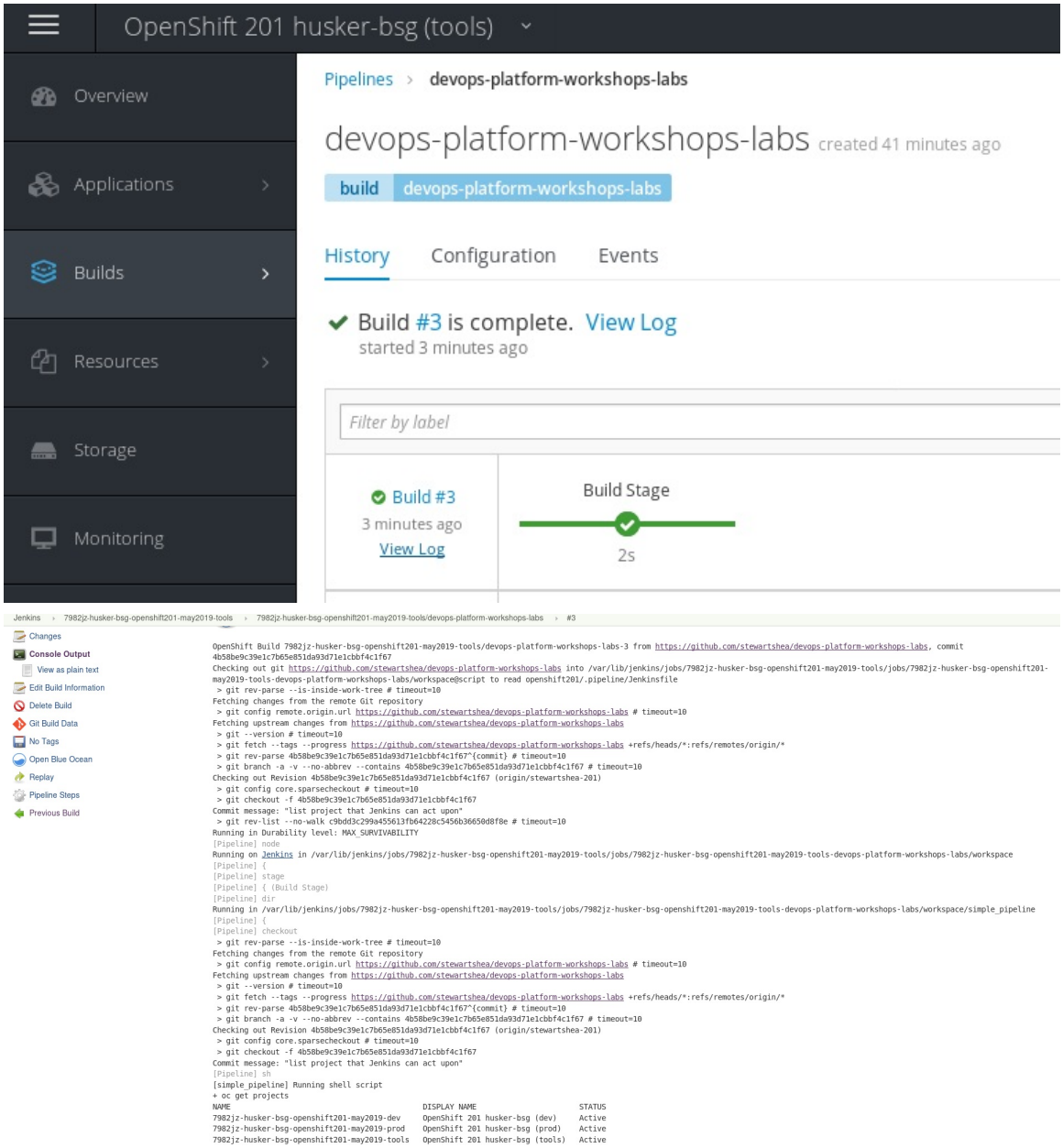
Exploring Jenkins Integration Capabilities

- Modify the pipeline in a `shell` step to explore the capabilities wtih `oc`
  - List all projects that Jenkins can operate upon with `oc get projects`

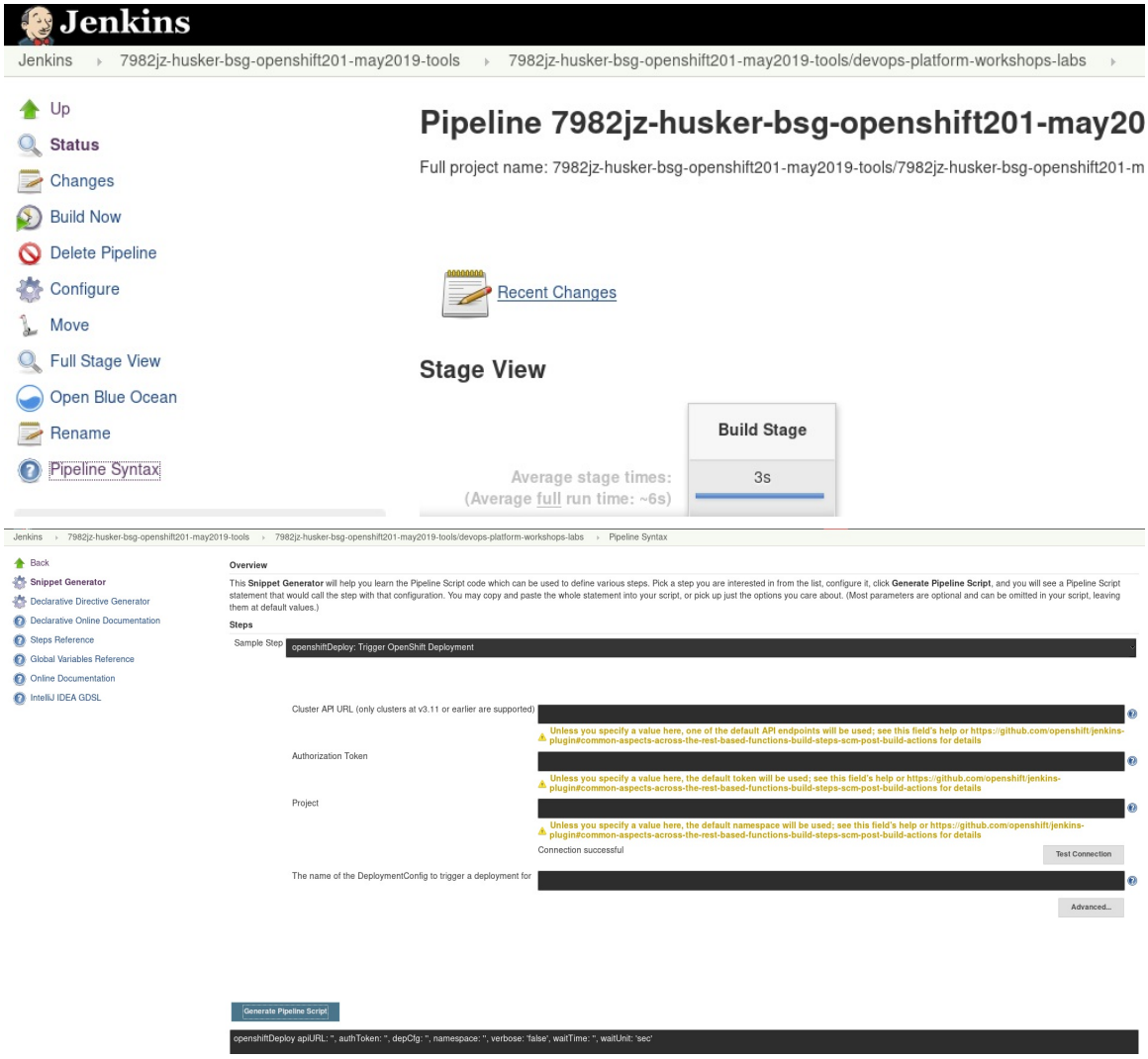
```
//Pipeline
node {
            stage ('Build Stage'){
                dir ('simple_pipeline') {
                    checkout scm
```

```
                    sh "oc get projects"
                }
            }
        }
```

- Review the log output from the job



- Jenkins also provides a `Pipeline Syntax` tool with a Snippet Generator to assist in generating pipeline code



Deploy the Loki Template to Dev

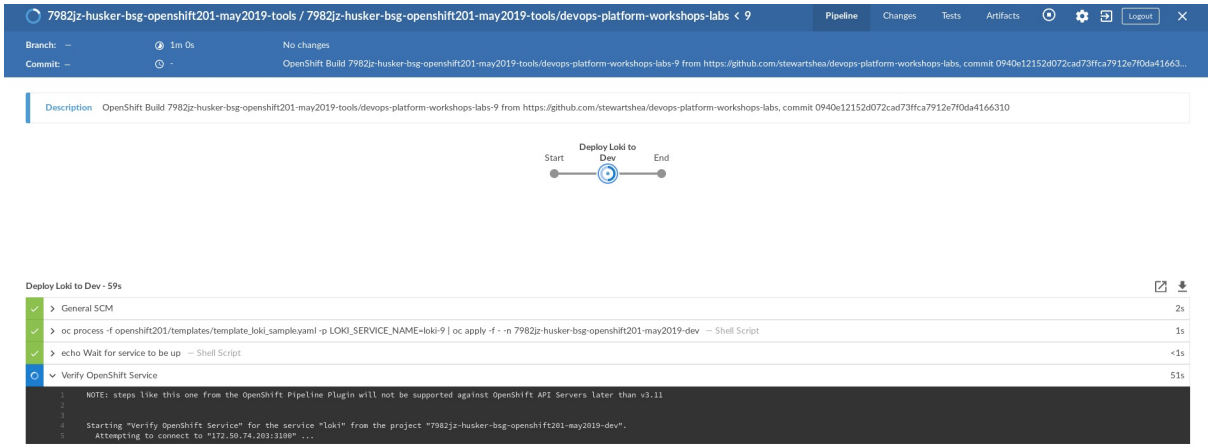
- Modify the Jenkinsfile to deploy the Loki template to dev
  - Use environment variables as parameters into the templates

```
#!/usr/bin/env groovy

//ENV Vars
def TOOLS_NAMESPACE = "[namespace]-tools"
def DEV_NAMESPACE = "[namespace]-dev"
def PROD_NAMESPACE = "[namespace]-prod"
def LOKI_SERVICE  = "loki"


//Pipeline
node {
            stage ('Deploy Loki to Dev'){
                dir ('simple_pipeline') {
                    checkout scm
```

```
                    sh "oc process -f openshift201/templates/template_loki_sample.y
aml -p  LOKI_SERVICE_NAME=${LOKI_SERVICE} | oc apply -f - -n ${DEV_NAMESPACE}"

                }
            }
    }
```

- Explore the `-tools` project and notice that no other jenkins agent pods are leveraged
- Add in functionality to verify the service in the deployment

```
                    sh "echo Wait for service to be up"
                    openshiftVerifyService apiURL: '', authToken: '', namespace: "$
{DEV_NAMESPACE}", svcName: "${LOKI_SERVICE}", verbose: 'false'
```



- Pause and discuss this function with the class
- Kill the Jenkins Pod
- Wrap that function with a timeout and run the job again

```
                    sh "echo Wait for service to be up"
                    timeout (time: 60, unit: "SECONDS"){
                    openshiftVerifyService apiURL: '', authToken: '', namespace: "$
{DEV_NAMESPACE}", svcName: "${LOKI_SERVICE}", verbose: 'false'
                    }
```

- The openshiftVerifyService won't work in our case, use one of the more advanced OpenShift DSL functions to verify status

```
#!/usr/bin/env groovy

//ENV Vars
def TOOLS_NAMESPACE = "[namespace]-tools"
def DEV_NAMESPACE = "[namespace]-dev"
def PROD_NAMESPACE = "[namespace]-prod"
def LOKI_SERVICE  = "loki"
```

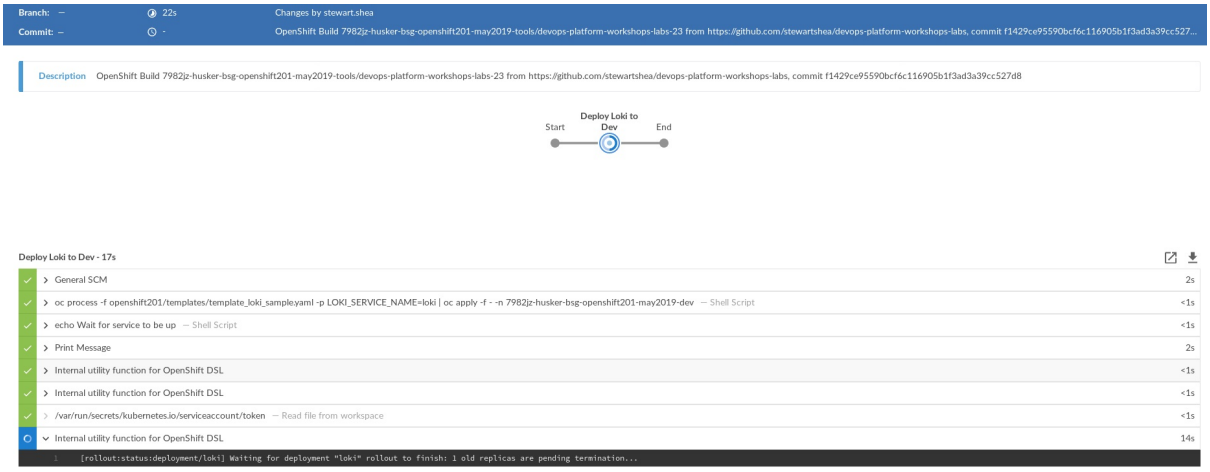
```
//Pipeline
node {
            stage ('Deploy Loki to Dev'){
                dir ('simple_pipeline') {
                    checkout scm
                    sh "oc process -f openshift201/templates/template_loki_sample.y
aml -p  LOKI_SERVICE_NAME=${LOKI_SERVICE} | oc apply -f - -n ${DEV_NAMESPACE}"
                    sh "echo Wait for service to be up"
                    timeout (time: 180, unit: 'SECONDS'){
                        openshift.withCluster() {
                        openshift.withProject("${DEV_NAMESPACE}") {
                            def dc = openshift.selector('deployment', "${LOKI_SERVI
CE}")
                            // this will wait until the desired replicas are availa
ble
                            dc.rollout().status()
                        }
                        }
                    }

             }
        }
    }
```

- Clean out the existing deployment and run the pipeline again with the new verification

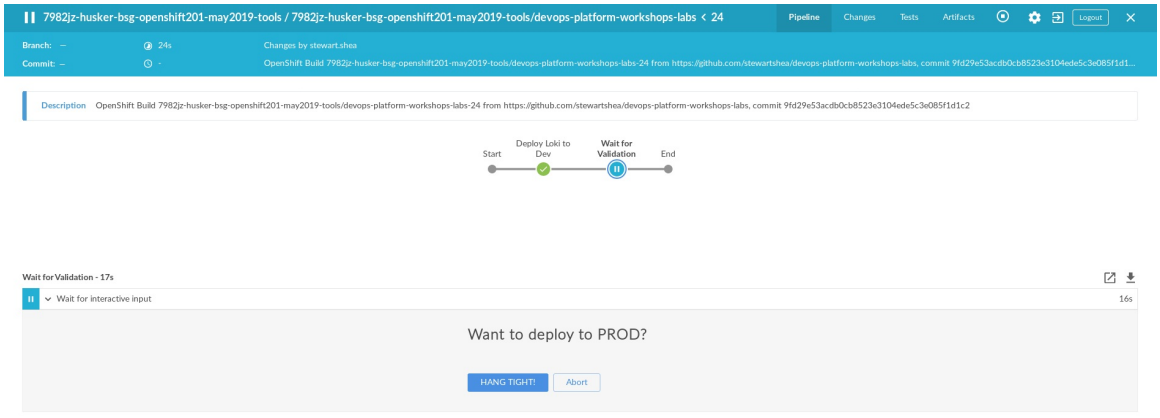
- Modify the Loki deployment template with **arbitrary metadata on the container** which will trigger a new deployment

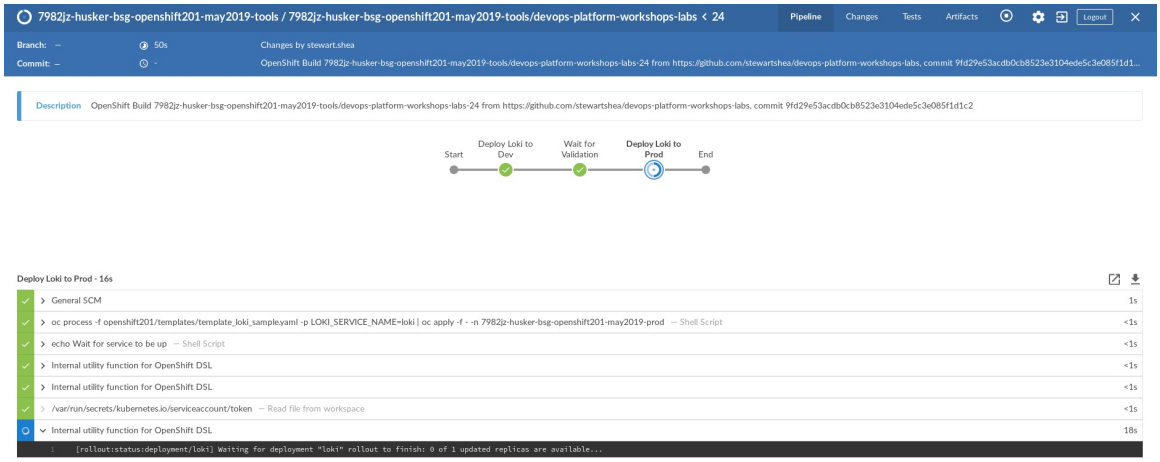
```
- apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: ${LOKI_SERVICE_NAME}
    labels:
      app: ${LOKI_SERVICE_NAME}
    annotations:
      {}
```

```
  spec:
    replicas: 1
    minReadySeconds: 0
    selector:
      matchLabels:
        app: ${LOKI_SERVICE_NAME}
    strategy:
      type: RollingUpdate
    template:
      metadata:
        labels:
          app: ${LOKI_SERVICE_NAME}
        annotations:
          prometheus.io/port: http-metrics
          prometheus.io/scrape: "true"
          our-awesome-metadata: "nothing to see here"
```



- Deploy Loki across all namespaces with an input stage for approval; it should resemble the following




7982jz-husker-bsg-openshift201-may2019-tools / 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs < 24
Pipeline
Changes
Tests
Artifacts
Logout
Branch: –
50s
Changes by stewart.shea
Commit: –
OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-24 from https://github.com/stewartshea/devops-platform-workshops-labs, commit 9fd29e53acdb0cb8523e3104ede5c3e085f1d1...
Description
OpenShift Build 7982jz-husker-bsg-openshift201-may2019-tools/devops-platform-workshops-labs-24 from https://github.com/stewartshea/devops-platform-workshops-labs, commit 9fd29e53acdb0cb8523e3104ede5c3e085f1d1c2
Start
Deploy Loki to Dev
Wait for Validation
Deploy Loki to Prod
End
Deploy Loki to Prod - 16s
General SCM
1s
oc process -f openshift201/templates/template_loki_sample.yaml -p LOKI_SERVICE_NAME=loki | oc apply -f - -n 7982jz-husker-bsg-openshift201-may2019-prod — Shell Script
<1s
echo Wait for service to be up — Shell Script
<1s
Internal utility function for OpenShift DSL
<1s
Internal utility function for OpenShift DSL
<1s
/var/run/secrets/kubernetes.io/serviceaccount/token — Read file from workspace
<1s
Internal utility function for OpenShift DSL
18s
[rollout:status:deployment/loki] Waiting for deployment "loki" rollout to finish: 0 of 1 updated replicas are available...

Jenkins Build Pipeline

These labs continue from the previous set, building on the previous effort & learnings. The previous lab focused on deploying items across projects, while this phase focuses on building and image and promoting that image across projects.

Create a New Pipeline for BlackBox Exporter Build

In this lab, create a simple pipeline that will build the blackblox exporter app within the tools namespace, and then deploy it in dev, and then in production.

- Create the new pipeline under `blackbox_exporter/.pipeline/Jenkinsfile`

```
#!/usr/bin/env groovy

//ENV Vars
def TOOLS_NAMESPACE = "[project]-tools"
def DEV_NAMESPACE = "[project]-dev"
def PROD_NAMESPACE = "[project]-prod"


//Pipeline
node {
            stage ('Buld in Tools Namespace'){
                dir ('simple_pipeline') {
                    checkout scm
                    timeout (time: 600, unit: 'SECONDS'){
                        openshift.withCluster() {
                        openshift.withProject("${TOOLS_NAMESPACE}") {
                                def blackboxSelector = openshift.selector("bc", "bl
ackboxexporter")
                                try {
                                    blackboxSelector.object()
                                    builds = blackboxSelector.related( "builds" )
                                } catch (Throwable t) {
                                    nb = openshift.newBuild( "https://github.com/[u
sername]/devops-platform-workshops-labs.git#[username]-201", "--context-dir=blackbo
x_exporter", "--name=blackboxexporter" )

                                    // Print out information about the objects crea
ted by newBuild
                                    echo "newBuild created: ${nb.count()} objects :
 ${nb.names()}"

                                    // Filter non-BuildConfig objects and create se
lector which will find builds related to the BuildConfig
                                    builds = nb.narrow("bc").related( "builds" )
```

```
                                }
                                    openshift.selector("bc", "blackboxexporter").st
artBuild("--wait")
                                    builds.watch {
                                    // 'it' is bound to the builds selector.
                                    // Continue to watch until at least one build i
s detected
                                    if ( it.count() == 0 ) {
                                        return false
                                    }
                                    // Print out the build's name and terminate the
 watch
                                    echo "Detected new builds created by buildconfi
g: ${it.names()}"
                                    return true
                                    }

                                    echo "Waiting for builds to complete..."

                                    // Like a watch, but only terminate when at lea
st one selected object meets condition
                                    builds.untilEach {
                                        return it.object().status.phase == "Complet
e"
                                    }
                            }
                            }
                        }

                }
            }
        }
```

- Create the buildConfig inside of the `tools` namespace

```
oc new-build https://github.com/[username]/devops-platform-workshops-labs.git#[user
name]-201 --context-dir=blackbox_exporter/.pipeline --name blackbox
```

- Either link up the the webhook to this build config as well, or manually start the pipeline as required
- Add additional stage to create the app in the `dev` namespace

```
#!/usr/bin/env groovy

//ENV Vars
def TOOLS_NAMESPACE = "[project_name]-tools"
def DEV_NAMESPACE = "[project_name]-dev"
def PROD_NAMESPACE = "[project_name]-prod"
```

```
//Pipeline
node {
            stage ('Buld in Tools Namespace'){
                dir ('simple_pipeline') {
                    checkout scm
                    timeout (time: 600, unit: 'SECONDS'){
                        openshift.withCluster() {
                        openshift.withProject("${TOOLS_NAMESPACE}") {
                                def blackboxSelector = openshift.selector("bc", "bl
ackboxexporter")
                                try {
                                    blackboxSelector.object()
                                    builds = blackboxSelector.related( "builds" )
                                } catch (Throwable t) {
                                    nb = openshift.newBuild( "https://github.com/[u
sername]/devops-platform-workshops-labs.git#[username]-201", "--context-dir=blackbo
x_exporter", "--name=blackboxexporter" )

                                    // Print out information about the objects crea
ted by newBuild
                                    echo "newBuild created: ${nb.count()} objects :
 ${nb.names()}"

                                    // Filter non-BuildConfig objects and create se
lector which will find builds related to the BuildConfig
                                    builds = nb.narrow("bc").related( "builds" )
                                }
                                    openshift.selector("bc", "blackboxexporter").st
artBuild("--wait")
                                    builds.watch {
                                    // 'it' is bound to the builds selector.
                                    // Continue to watch until at least one build i
s detected
                                    if ( it.count() == 0 ) {
                                        return false
                                    }
                                    // Print out the build's name and terminate the
 watch
                                    echo "Detected new builds created by buildconfi
g: ${it.names()}"
                                    return true
                                    }

                                    echo "Waiting for builds to complete..."

                                    // Like a watch, but only terminate when at lea
st one selected object meets condition
                                    builds.untilEach {
                                        return it.object().status.phase == "Complet
e"
```

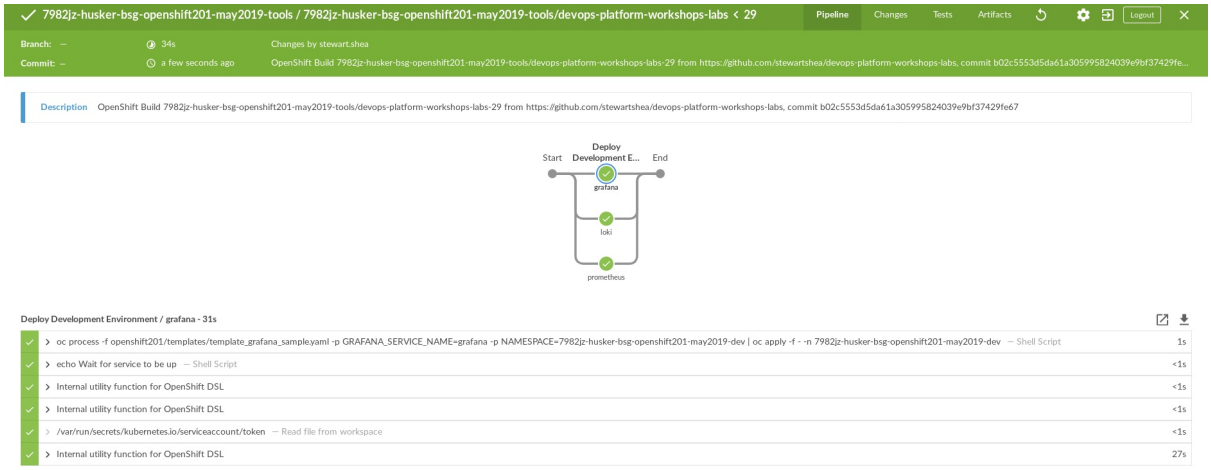
```
                                    }
                        }
                        }
                    }

             }
        }
        stage('Promote to DEV') {
                openshift.withCluster() {
                    openshift.tag("blackboxexporter:latest", "blackboxexporter:dev"
)
                }
            }
        stage('Create DEV') {
            openshift.withCluster() {
                openshift.withProject("${DEV_NAMESPACE}") {
                                def blackboxdcSelector = openshift.selector("dc", "
blackboxexporter")
                                try {
                                    blackboxdcSelector.object()
                                    deploymentconfig = blackboxdcSelector.related(
"deploymentconfig" )
                                } catch (Throwable t) {
                                    na = openshift.newApp("${TOOLS_NAMESPACE}/black
boxexporter:dev", "--name=blackboxexporter").narrow('svc').expose()

                                    // Print out information about the objects crea
ted by newBuild
                                    // echo "newApp created: ${na.count()} objects
: ${na.names()}"

                                }
                }
            }
            }
```

- Extend the pipeline to deploy to the `prod` namespace as well

Parallelize the Deployment of Prometheus/Loki/Grafana

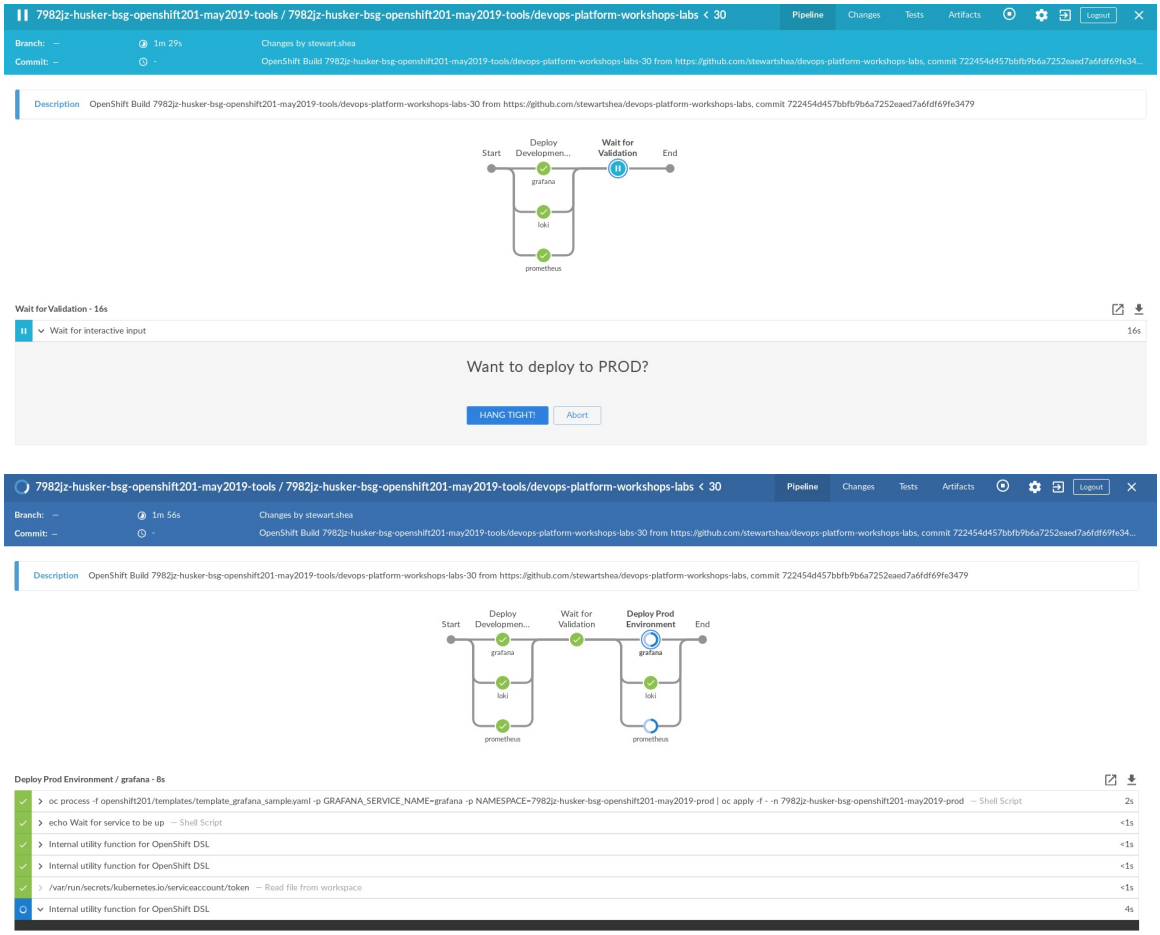
In this next stage, deploy all *three* components to the `dev` project with parallelization. This following code snippet is an example of how parallelization is achieved:

```
            stage('Testing Stage'){

                parallel (
                    'load test': {
                        sh "echo load test"
                    },
                    'security test': {
                        sh "echo security test"
                    },
                    'e2e test': {
                        echo "e2e test"
                    }
                )

            }
```



Note All troubleshooting of the templates should be done in code and commited to your repository, automatically starting a new pipeline run.

- Similar to the previous lab, extend the pipeline to deploy to prod



Modify Grafana and with Automated Deployment

With the pipeline fully configured, make changes to Grafana to perform the following tasks:

- Set ENV variables to configure logging
  - output to `/logs/`
  - mode set to `file`

```
## deployment
          env:
            - name: GF_PATHS_LOGS
              value: /logs
            - name: GF_LOG_MODE
              value: file
```

- Configure an `emptyDir` mount for `/logs`

```
## deployment

          - mountPath: /logs
            name: logs
```

- Add a promtail container to read from `/logs`

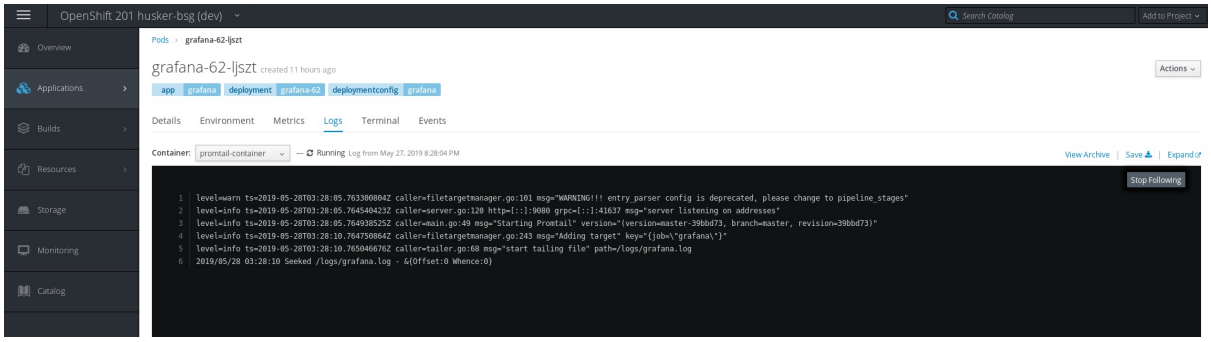
```
## deployment
        - name: promtail-container
          image: grafana/promtail:latest
          args:
          - -config.file=/etc/promtail/promtail.yaml
          volumeMounts:
          - name: logs
            mountPath: /logs
          - name: promtail-config
            mountPath: /etc/promtail
```

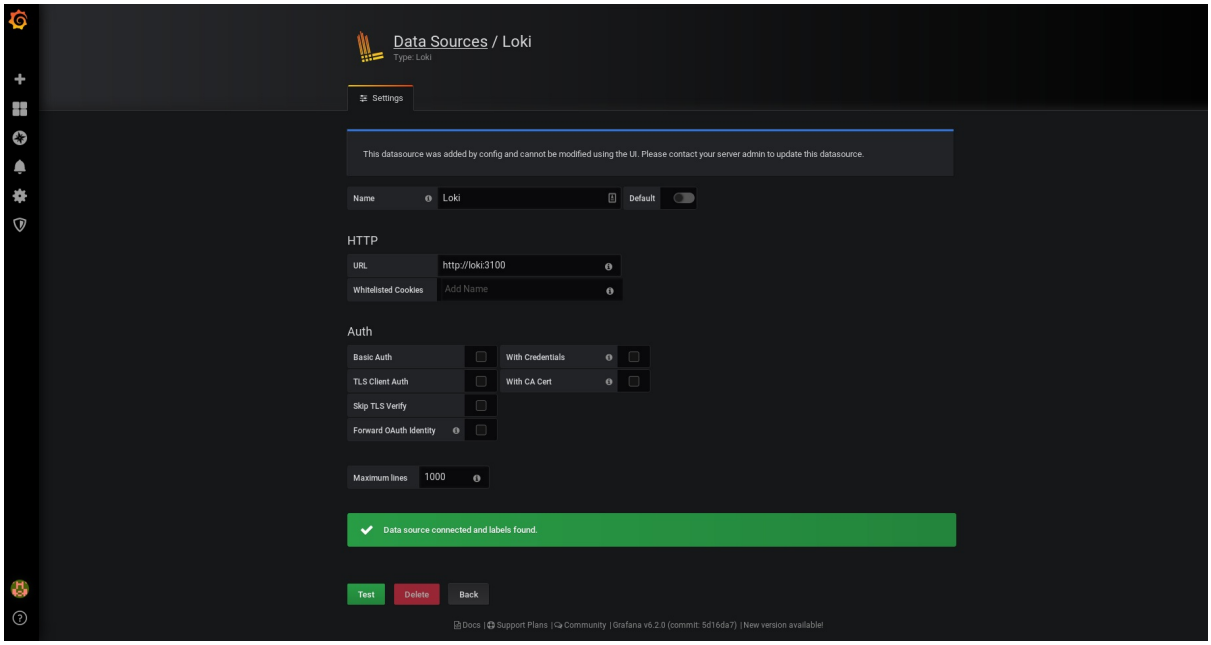
- Add a config map for promtail

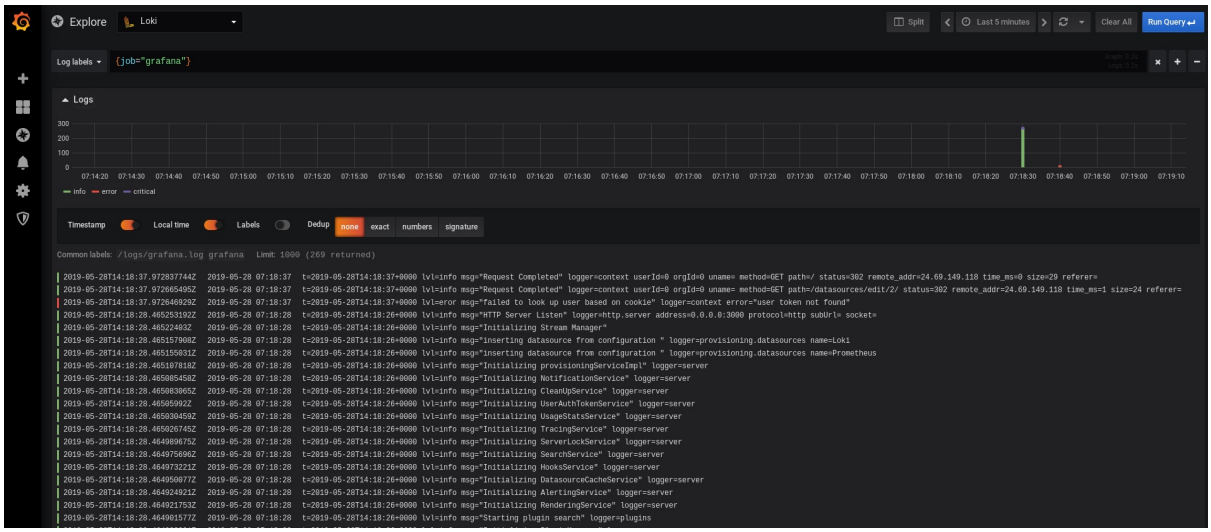
```
## deployment
        volumes:
        - configMap:
            defaultMode: 420
            name: ${GRAFANA_SERVICE_NAME}-promtail-config
          name: promtail-config

## configmap
- apiVersion: v1
  kind: ConfigMap
  metadata:
    name: ${GRAFANA_SERVICE_NAME}-promtail-config
    labels:
      app: ${GRAFANA_SERVICE_NAME}
  data:
    promtail.yaml: |
      server:
        http_listen_port: 9080
        grpc_listen_port: 0
      positions:
        filename: /tmp/positions.yaml
      clients:
        - url: http://${LOKI_SERVICE_NAME}:3100/api/prom/push
      scrape_configs:
        - job_name: system
          entry_parser: raw
          static_configs:
            - targets:
                - localhost
              labels:
                job: grafana
                __path__: /logs/grafana.log
```

- Push the changes and monitor the deployment

- Validate the datasource and explore the Loki interface in Grafana







Patroni Setup

In this lab we are building a back-end database for our grafana application. Grafana will use this DB for managing any configuration not covered by the configmaps, as well as user session state. The database tool we will be using is postgresql with patroni operating as a cluster manager.

Create your image

Generate an image in your project space to use for the lab.

- Check that you are logged into the workshop -tools namespace

```
oc login https://console.pathfinder.gov.bc.ca:8443 --token=[token]
oc project [workshop_project_set_name]-tools
```

- clone and examine the https://github.com/bcdevops/platform-services repository and look at the patroni templates (/apps/pgsql/patroni/)
- process the build template into your -tools environment with appropriate parameters

Shortcut: instead of cloning this repository locally, you can process a raw github link directly:

```
oc process -f \
  https://raw.githubusercontent.com/BCDevOps/platform-services/master/apps/pgsql/patroni/openshift/build.yaml \
  --param-file=./patroni-build.env | oc apply -f -
```

- tag your latest successful build with a tag for the environment you want to deploy to: "v10-dev" for example.

```
oc tag [your-patroni-imageStream]:v10-latest patroni:v10-dev \
-n [workshop_project_set_name]-tools
```

Create the deployment pre-requisites

The patroni template uses a pre-created secret.

- Examine parameters in the included patroni pre-requisite template (secret object)
- Create a parameter file for your deployment environment with settings needed for both the pre-requisite secret and the deployment.

```
cat << EOT > dev.env
NAME=[your-patroni]
IMAGE_STREAM_NAMESPACE=[workshop_project_set_name]-tools
```

```
IMAGE_STREAM_TAG=[your-patroni-imageStream]:v10-dev
PVC_SIZE=1Gi
APP_DB_NAME=grafana
APP_DB_USERNAME=grafana
EOT
```

- process the prereq file

```
oc project [workshop_project_set_name]-dev
oc process -f openshift/deployment-prereq.yaml --param-file=./dev.env
```

Tip: add --ignore-unknown-parameters=true to skip this error message and combine the application templates into a single env file

- Re-process and apply the pre-req to your dev environment

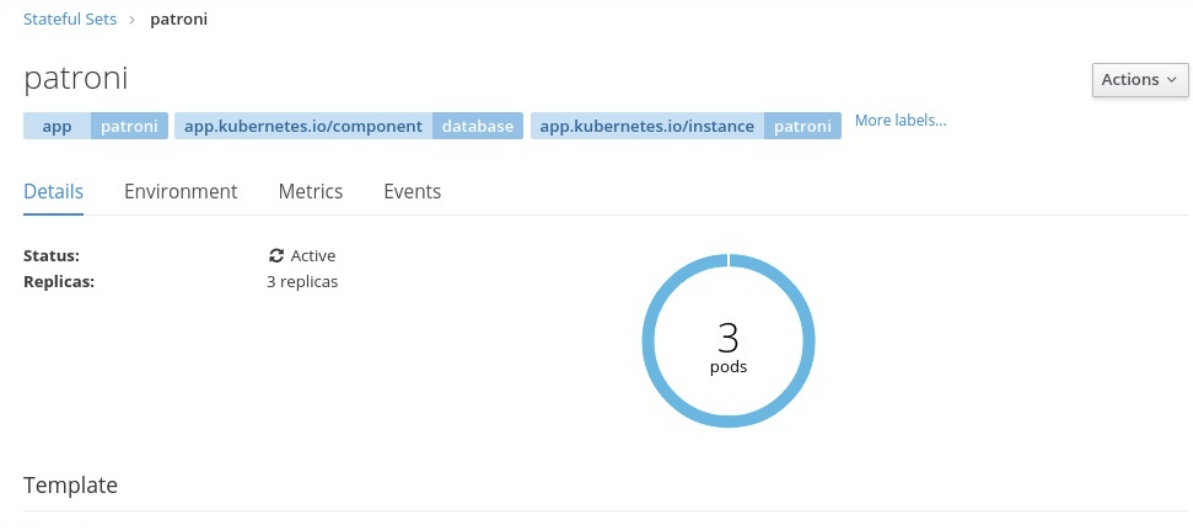
```
oc process -f openshift/deployment-prereq.yaml \
  --param-file=./dev.env --ignore-unknown-parameters=true \
  | oc apply -f -
```

Create the stateful set

Have a look at the objects created by the deployment.yaml as this OpenShift Template will be creating a few things that you'll want to keep track of, and the statefulSet itself will be creating additional objects afterwards.

```
oc process -f openshift/deployment.yaml \
  --param-file=./dev.env --ignore-unknown-parameters=true \
  | oc apply -f - -n [workshop_project_set_name]-dev
```

- Watch the deployment spin up the stateful set

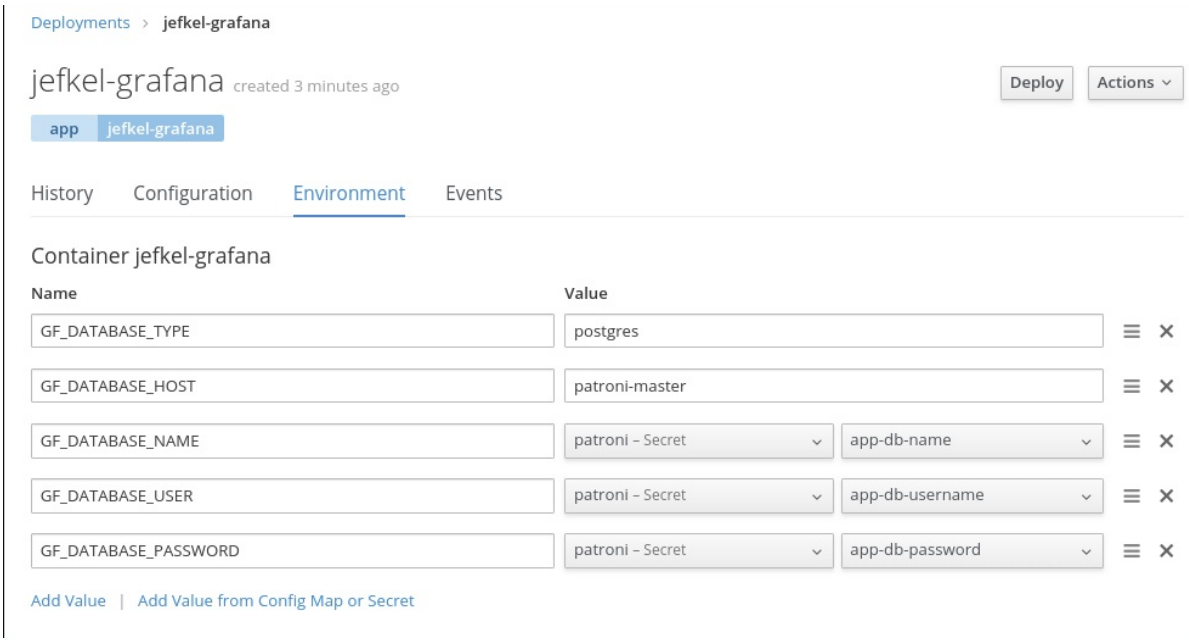


Configure grafana

Configure the grafana app to use patroni postgres DB instead of it's sqlite DB for session caching and anything not in a configMap.

- Update the grafana deploymentConfig with the following environment variables

```
GF_DATABASE_TYPE=postgres
GF_DATABASE_HOST=[your-patroni]-master
GF_DATABASE_NAME=(secret:your-patroni/app-db-name)
GF_DATABASE_USER=(secret:your-patroni/app-db-username)
GF_DATABASE_PASSWORD=(secret:your-patroni/app-db-password)
```



Now you can scale up the grafana deployment and both state configuration as well as user sessions will be highly available via the database.

Don't forget to export your changes

Postgres configuration

In this lab we will look at a simple change to the postgres configuration through the different Patroni methods. For additional reading and reference: https://patroni.readthedocs.io/en/latest/

Change number of connections

Modifying the postgres configuration will require changing the DCS (Distributed Configuration Store) which is held in a generated configMap [clustername]-config.

Option 1: export/modify/re-apply

- Export the patroni-config configMap
- Change config by stripping out non-essential entries (resourceVersion, selfLink, etc)
  - The annotations that you don't care about can be safely removed from your export
- Apply new configmap

Option 2: Use Patroni API from within a pod

The configuration section has examples on how to access the patroni API using the curl command. Once you make the change using the curl command, have a look at the configMap to see the changes added there.

- example curl command:

```
curl -s -XPATCH -d \
        '{"postgresql":{"parameters":{"max_connections":"101"}}}' \
        http://localhost:8008/config | jq .
```

Check configuration status

- rsh to one of the patroni pods
- use the patroni api to check the configuration

```
curl -s http://localhost:8008/config | jq .

curl -s http://localhost:8008/patroni | jq .
```

notice the "Pending restart"

- Alternatively, run the following to get a quick status of all members

```
patronictl list
```

notice the "Pending restart" column

Restart to apply the new configuration

Patroni has only flagged the pods for a restart, but will allow you to coordinate any operational activities for pushing that change. The StatefulSet update strategy is not triggered by this configMap change.

- Using the `patronictl` tool within one of the containers, restart one (or both) of the replicas.

- once your replicas are available and sync'd, rsh to a restarted replica and check it's configuration to confirm the settings have been applied.

```
curl -s http://localhost:8008/patroni | jq .
```

You can check the active postgres settings with the following:

```
psql
select  * from
(select count(*) used from pg_stat_activity) q1,
(select setting::int res_for_super from pg_settings where name=$$superuser_reserved
_connections$$) q2,
(select setting::int max_conn from pg_settings where name=$$max_connections$$) q3;
\q
```

- switch the master to the replica with the updated configuration either by using the patronictl tool (`switchover` or `failover` are both valid options) or through another method of your choice.
- Confirm confguration has changed on all pods

```
$ patronictl list
+---------+-----------+----------------+--------+---------+----+-----------+
| Cluster |   Member  |      Host      |  Role  |  State  | TL | Lag in MB |
+---------+-----------+----------------+--------+---------+----+-----------+
| patroni | patroni-0 | 172.51.121.73  |        | running |  2 |         0 |
| patroni | patroni-1 | 172.51.77.144  | Leader | running |  2 |         0 |
| patroni | patroni-2 | 172.51.112.237 |        | running |  1 |         0 |
+---------+-----------+----------------+--------+---------+----+-----------+
$ psql
select  * from
(select count(*) used from pg_stat_activity) q1,
(select setting::int res_for_super from pg_settings where name=$$superuser_reserved_connections$$) q2,
(select setting::int max_conn from pg_settings where name=$$max_connections$$) q3;
\q
psql (10.8 (Debian 10.8-1.pgdg90+1))
Type "help" for help.

postgres=# select  * from
postgres-# (select count(*) used from pg_stat_activity) q1,
postgres-# (select setting::int res_for_super from pg_settings where name=$$superuser_reserved_connections$$) q2,
postgres-# (select setting::int max_conn from pg_settings where name=$$max_connections$$) q3;
 used | res_for_super | max_conn
------+---------------+----------
    7 |             3 |      105
(1 row)

postgres=# \q
$
```

Update base image

The `/home/postgres/patroni.yml` file is the default base configuration for your cluster when it is first initialized. There are some settings that can be managed in this file with environment variables in the buildConfig. Not all options have been exposed yet, so for more complex changes you may need to replace the `entrypoint.sh` with your own customizations.

- Add the following to the buildConfig's environment variables and rebuild your image.

```
POSTGRESQL_MAX_CONNECTIONS=200
```

- To see if your pods are using the new image, check the patroni.yml file found in the container.
- Kill a replica pod and see which image it uses on a restart.
- Create a new tag, and update the statefulSet to use the new image tag.

These steps illustrate when the rollingUpdate is triggered, as well as when it's not.

Notice that the active postgres configuration did not change even after the patroni.yml was changed

Postgres backup and recovery

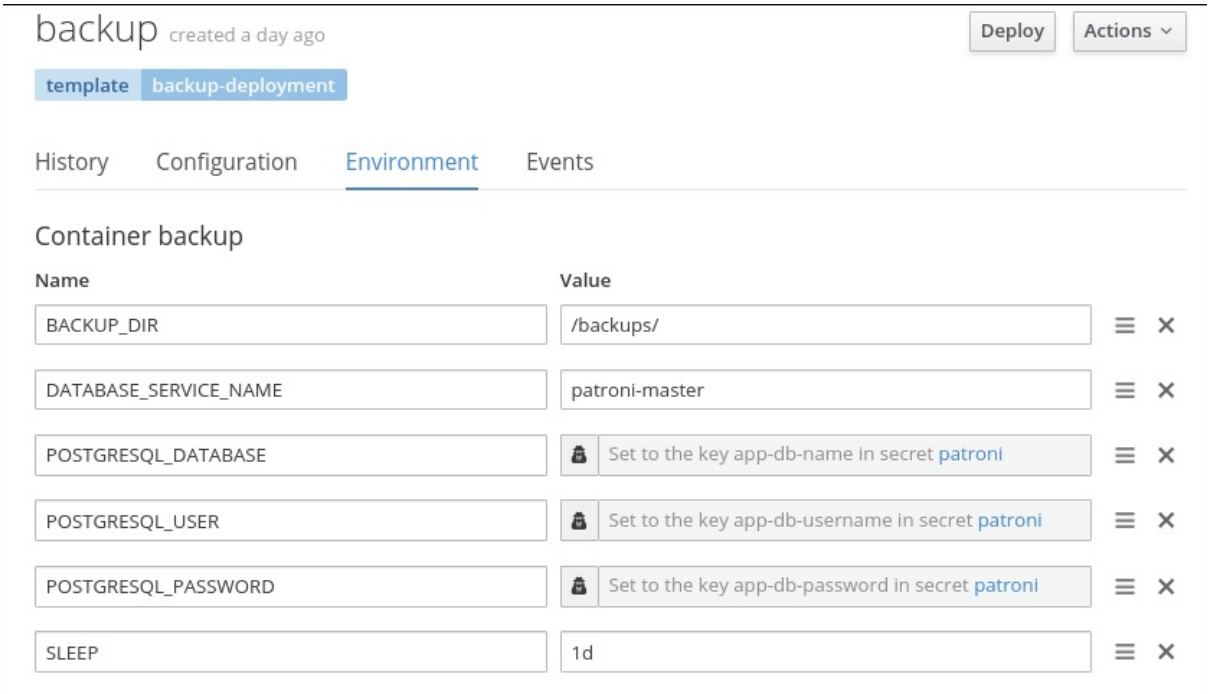
In this lab we will create and test a backup system for our postgres DB. If time is short, you can use the provided json templates in your forked repository.

- Build a backup image. You can find the backup-simple.sh script in the forked repository if you would like to have the deployed pod stay up without running from a debug pod. If you choose to ignore that file, you will need to run as a debug pod in order to have a shell to run the backup commands.

Dockerfile:

```
# This image provides a postgres installation from which to run backups
FROM registry.access.redhat.com/rhscl/postgresql-10-rhel7

# Set the workdir to be root
WORKDIR /

# Load the backup script into the container (must be executable).
COPY backup-simple.sh /

# Set the default CMD.
CMD sh /backup-simple.sh
```

- Create a deploymentConfig using that image
- Create 2 PVC's and attach to your deploymentConfig
  - backup-pvc - attach to /backups
  - validation-pvc - attach to /var/lib/pgsql/data
- Add the following environment variables to your deploymentConfig



- rsh to your backup container for the rest of these steps.

If you did not add the `backup-simple.sh` to your image, your deployment will not start. Simply run a debug pod and proceed with the lab using the debug pod instead.

- Have a quick sanity check on the environment variables, and then run your backup.

```
env | grep -e "BACK" -e "POSTGRESQL" -e "DATABASE"

PGPASSWORD=${POSTGRESQL_PASSWORD} pg_dump -Fp \
  -h "${DATABASE_SERVICE_NAME}" -p "${PATRONI_MASTER_SERVICE_PORT}" \
  -U $POSTGRESQL_USER $POSTGRESQL_DATABASE \
  | gzip > "${BACKUP_DIR}/${DATABASE_SERVICE_NAME}.dmp-`date +\%Y-\%m-\%d_%H-%M-%S`
.gz"
```

Restore Process

- Start a DB in the backup pod

```
run-postgresql &
```

- Restore the DB

```
gunzip -c "${BACKUP_DIR}/[backup file]" \
  | psql -v ON_ERROR_STOP=1 -x -h localhost -d "${POSTGRESQL_DATABASE}"
```

- Connect and verify the data

```
psql -d "${POSTGRESQL_DATABASE}" -c "\d"
```

- Stop the DB and clean up files.

```
pg_ctl stop -D /var/lib/pgsql/data/userdata
rm -rf /var/lib/pgsql/data/userdata
```

Next Steps

You've just completed a backup/validation cycle

- created a generic postgres:v10 container (possibly with a simple "pause" entry script)
- backed up a live database
- recovered that backup into a temporary database for validation
- validated the temporary data
- cleaned up the temporary database and data

That is how quick and easy it is to setup a rudimentary backup and backup verification environment. For advanced scheduling and a more complex automation utilities, have a look at https://github.com/BCDevOps/backup-container

Application failover testing and monitoring

Time blocked for testing different failure scenarios.

Front End failures

Scale up your Grafana deployment, then try to create a visible user event.

DB failures

Simulate different database faults and try to create a visible user event.

Creating a custom health check

In this lab we will create a custom health check and insert it into our patroni containers.

Create a custom script

In order to execute a custom check script, it will need to be accessible from within the containers. We will mount a configMap that contains a custom script that we can execute in parallel with our other health checks.

- create a custom health check script

```
cat << EOT > lab-check.sh
# test to see if /tmp/badfile does not exist
test ! -e /tmp/badfile
EOT
```

- create the temporary configmap

```
oc create configmap checks --from-file lab-check.sh
```

Add the following to the appropriate locations in the deployment.yaml to mount the configmap in the /opt directory.

Add with GUI and can export the template

- update the configuration of the stateful set
- once the pods have re-deployed, you can rsh to a pod and run the script to see it work. Simply touch a file called `/tmp/badfile` to force a check failure.

Add the liveness test

Add the following section right after the readinessProbe:

```
          livenessProbe:
            initialDelaySeconds: 5
            failureThreshold: 4
            exec:
              command:
                - . /opt/lab-check.sh
```